OpenText Cloud Developer Tutorial

Building a Contract Approval application

This tutorial has been created for software version OpenText™ Cloud Developer Tools for VS Code 23.4.1.

It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Note that if you are using this tutorial with a later version of the OpenText™ Cloud Developer Tools for VS Code, the screenshots and usage might not always correspond.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111

Toll Free Canada/USA: 1-800-499-6544 | International: +800-4996-5440

Fax: +1-519-888-0677

Support: https://support.opentext.com

For more information, visit http://www.opentext.com

Last updated: 04/02/2024

Contents

Introduction

This tutorial is intended as a comprehensive end-to-end course for pro-code developers. Throughout the different sections you will learn how to:

- Build an application that utilizes the OpenText Cloud Platform Services APIs.
- Develop a React based application in Microsoft Visual Studio Code (VS Code).

The application you will be building is a simple Contract Approval application utilizing the OpenText Cloud Platform Services. This application will allow you to do the following:

- Upload documents
- Store document related metadata for two types of contracts
- View documents
- Perform document analysis to detect PII (Personally Identifiable Information)
- Use a decision table to determine the required approvals
- Use a workflow to automate the different contract approval steps

The following OpenText Cloud Platform Services will be used to build the Contract Approval application:

Service	Purpose
Content Storage Service (CSS)	Uploading and storing of documents
Content Metadata Service (CMS)	Defining and storing of document metadata
Viewer Service	Viewing documents
Magellan Risk Guard Service	Document analysis
Decision Service	Defining and executing business rules (decision tables)
Workflow Service	Defining, executing, and auditing business processes (workflows)

In this tutorial, the following exercise modules detail the different steps for building the Contract Approval application:

1. Get started
   5 min

2. Set up the OpenText Cloud Developer IDE
   20 min

3. Set up a connection to the developer organization
   10 min

4. Create an OpenText project
   10 min

5. Create a namespace
   5 min

6. Create a trait
   10 min

7. Create types
   25 min

8. Create (user) groups
   10 min

9. Create ACLs
   15 min

10. Create a decision table
    25 min

11. Create workflows
    140 min

12. Deploy an application to the OpenText Cloud Platform Services
    25 min

13. Work with the OpenText Cloud Platform Services APIs
    25 min

14. Build the Contract Approval application
    30 min

15. Test the Contract Approval application
    60 min

16. Bonus exercise: Use the ocp command line interface
    15 min

You can directly open the finished application in VS Code without going through all the exercise modules and familiarize yourself with the OpenText Cloud Platform Services API example code, different models, and the Contract Approval sample application. The project sources for the completed Contract Approval application are available for download. To directly run the completed project from VS Code, see Test the application.

1 [5’] Get started

The following software environment is used:

Software	Version
Operating System	Windows 11 Enterprise 22H2, 64-bit
VS Code	1.86.2
Node.js	20.11.0 LTS
Java	21.0.1
OpenText™ Cloud Developer Tools for VS Code	23.4.1

Note

The procedure steps, images, and the other references used in this tutorial are based on the above environment. If you are using any other environment or software versions, refer to the software documentation for that version.

1.1 Audience

This tutorial is intended for pro-code developers who want to learn how to build applications that utilize the OpenText Cloud Platform Services APIs.

1.2 Prerequisites

Make sure you can sign in to the developer.opentext.com website and have an active trial or paid developer plan.

If you do not have an active plan, navigate to developer.opentext.com/plans to sign up for one.

For a detailed description on how to create a developer account and sign up for a developer trial, refer to: developer.opentext.com/imservices/trial.

2 [20’] Set up the OpenText Cloud Developer IDE

Learn how to:

- Download and install VS Code
- Add the OpenText Cloud Developer Tools VS Code extension pack to your VS Code IDE
- Install the LTS version of Node.js

2.1 Download and install VS Code

1. Go to https://code.visualstudio.com/download and download the Microsoft VS Code distribution that matches your system.

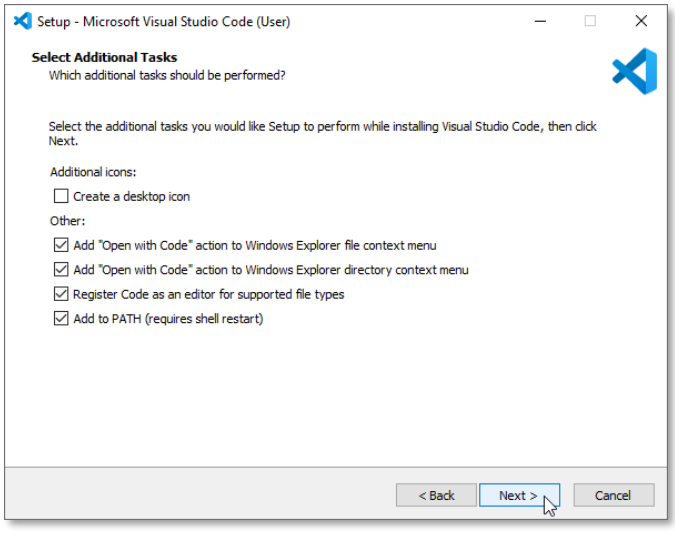
Note

To install VS Code on a 64-bit Windows 11 system, choose to download the x64 User Installer for Windows and install the latest version of VS Code.

Important

OpenText Cloud Developer Tools for VS Code is tested with Windows and Mac Operating Systems. You can use Linux Operating System but if you run into problems, OpenText might not be able to provide a solution.

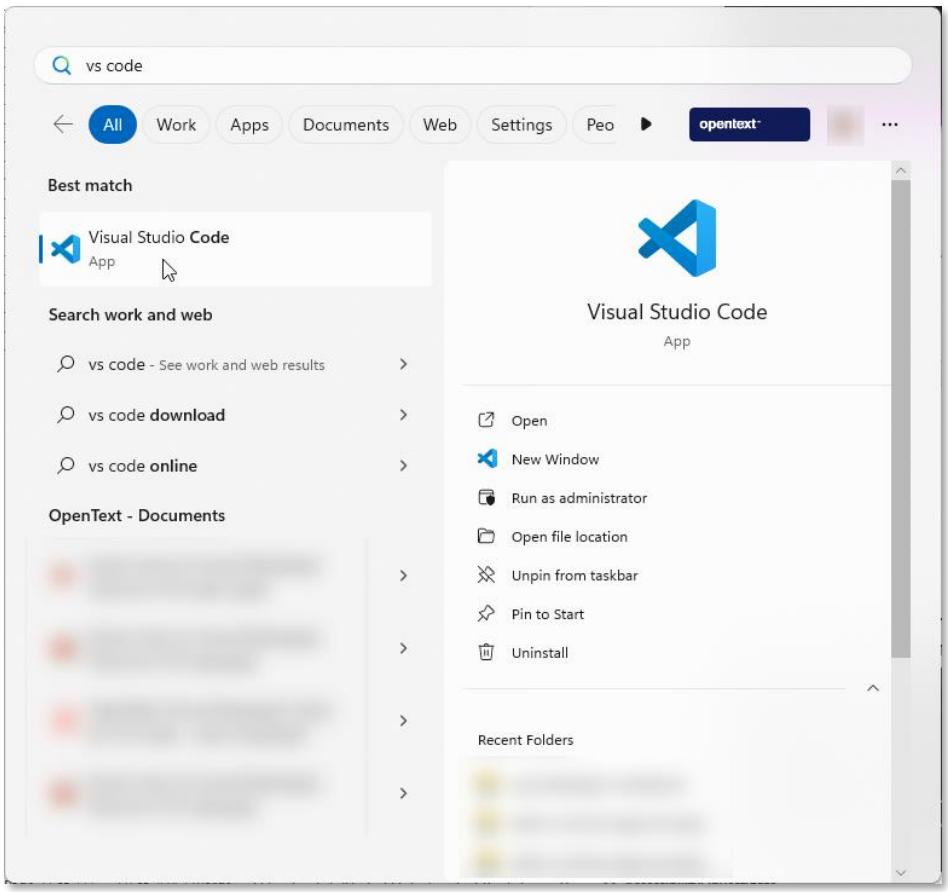
2. Save and run the installer.
3. Select **I accept the agreement** and click **Next** to continue.
4. Select the installation destination location and click **Next**. You can use the suggested default location.
5. Keep the default setting for the selecting of the start menu folder and click **Next**.
6. Select all the additional tasks under **Other** and optionally select **Create a desktop icon**. Click **Next** to continue.



7. Verify your choices and click **Install** to start the VS Code installation.
8. After the installation is complete, you can click **Finish** to close the VS Code Setup Wizard.
9. After VS Code is installed, in the Windows **Start** menu, type `vs code` in the search box and select the **Visual Studio Code** application.

Tip

You can Pin to taskbar for easy access from the task bar.



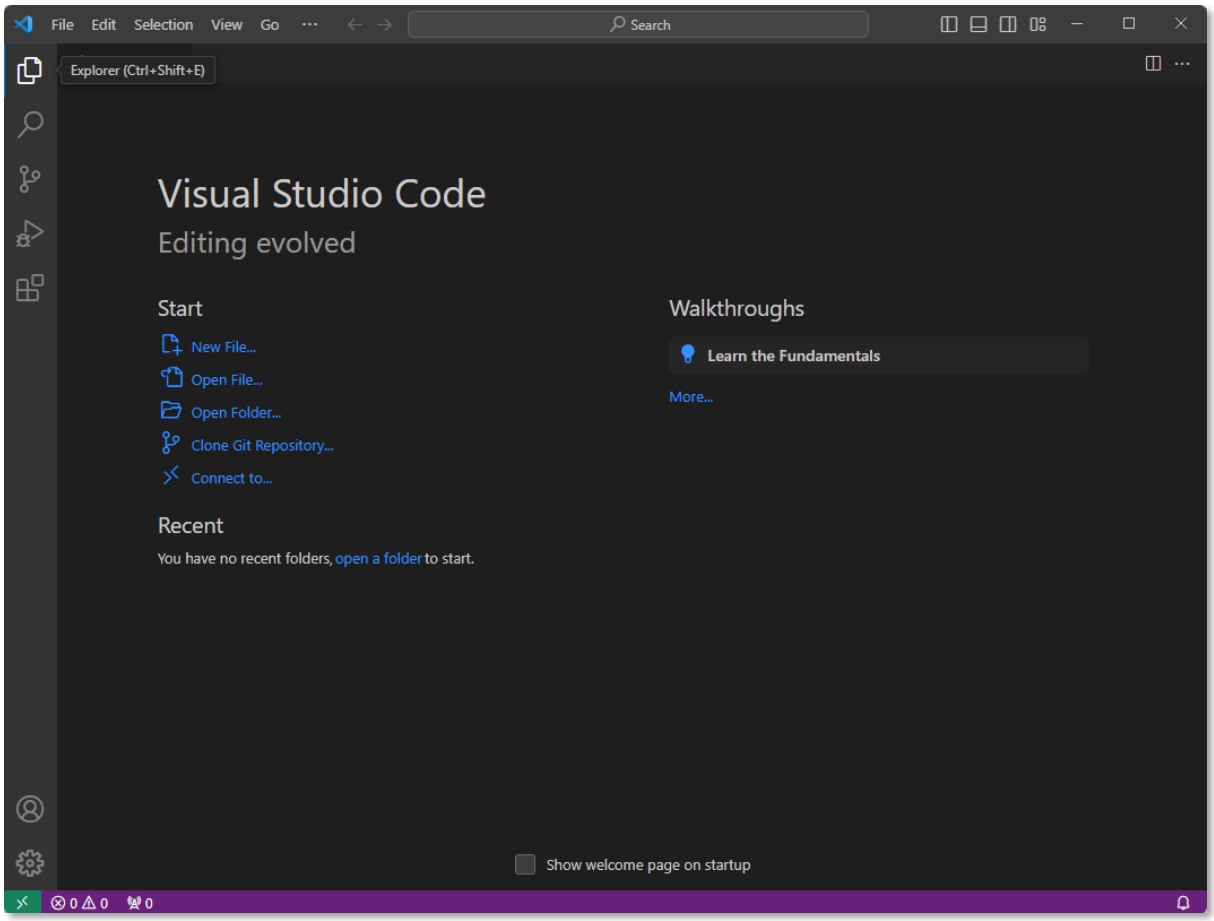
10. The **Get Started with VS Code** wizard is displayed when VS Code is launched for the first time.
11. Select a theme and click **Mark Done** to confirm your choice.

Note

For this tutorial the **Dark Modern** theme is selected. Take this into account when comparing your VS Code appearance with the VS Code screen shots throughout this document.

12. The standard VS Code welcome page is now displayed.

13. Make sure **Show welcome page on** startup is not selected so that the page does not appear when VS Code is launched every time.



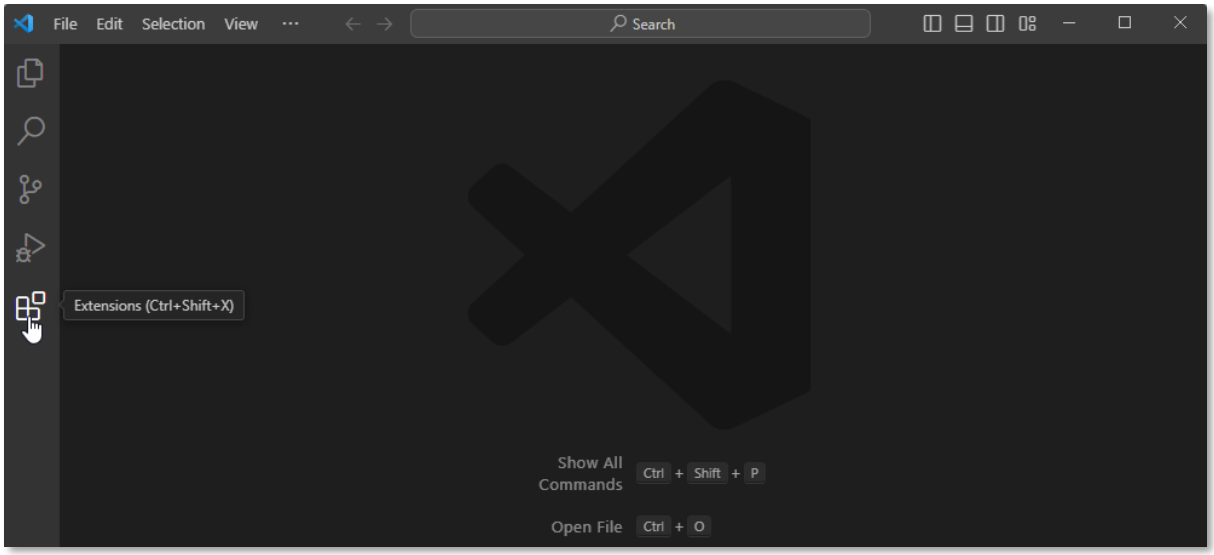
14. Close the welcome page.

Next step:

Add the **OpenText Cloud Developer Tools VS Code extension pack** to your VS Code IDE.

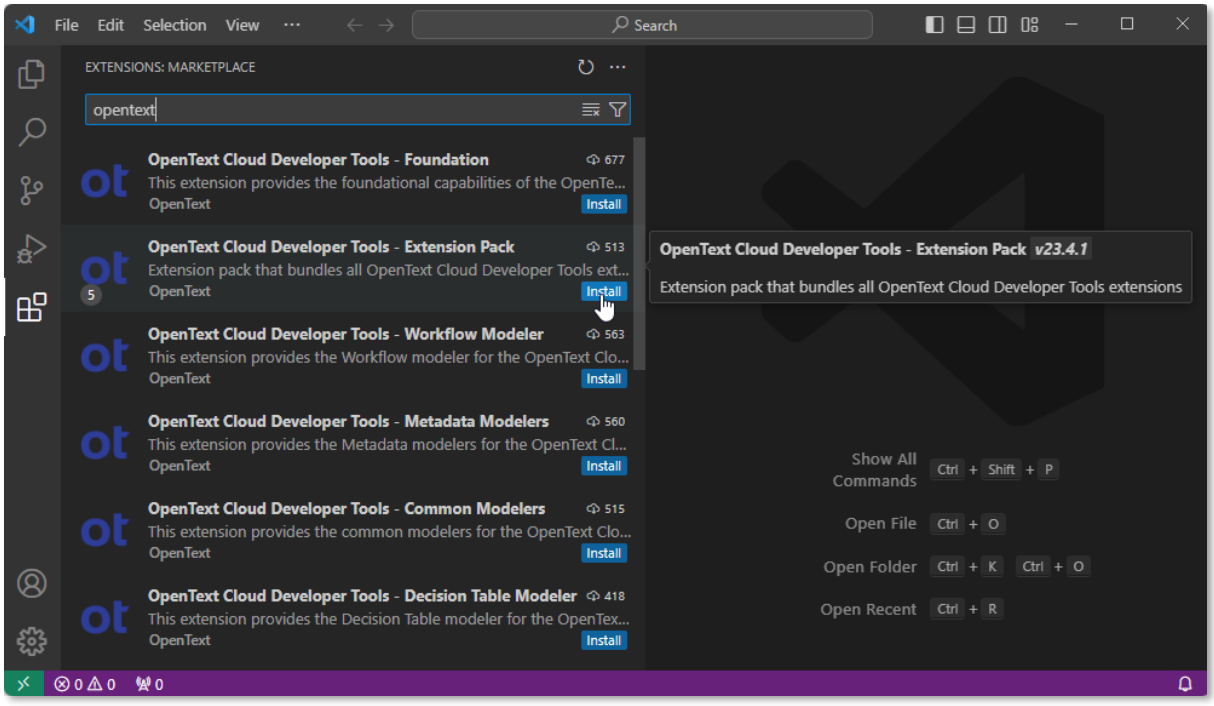
2.2 Add the OpenText Cloud Developer Tools extension pack to VS Code

1. On the VS Code Activity Bar, click **Extensions**.

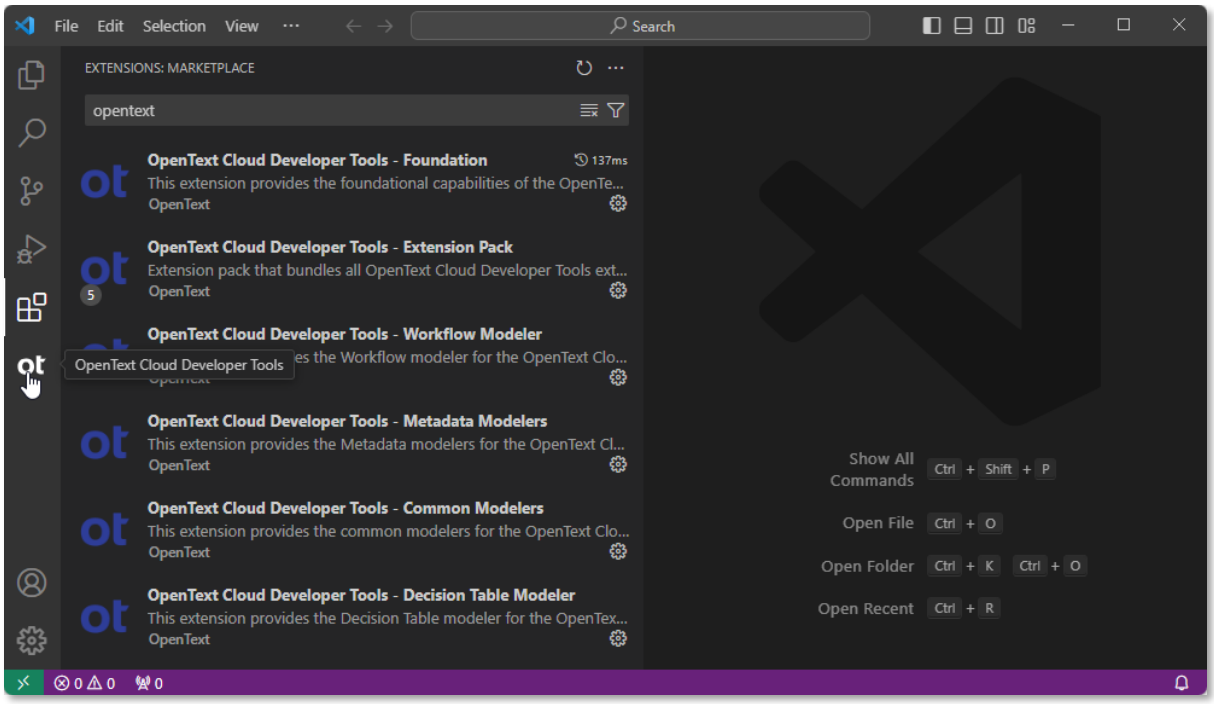


2. On the **Search** Extensions in Marketplace search bar, type `opentext` and choose to install the **OpenText Cloud Developer Tools - Extension Pack**.

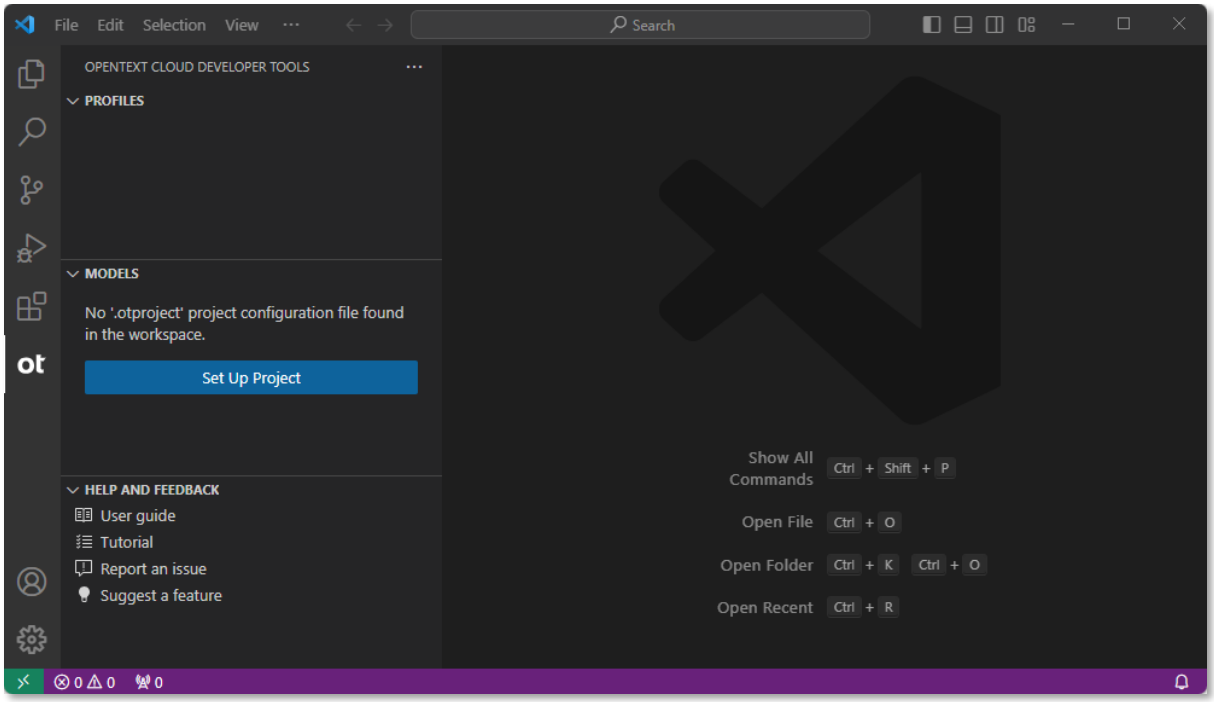
> **Note**
>
> If prompted, click Reload to ensure that the installed VS Code extension pack is enabled.



3. On the **Activity Bar**, click the **OpenText Cloud Developer Tools** button ot.



4. This opens the **OpenText Cloud Developer Tools** view.

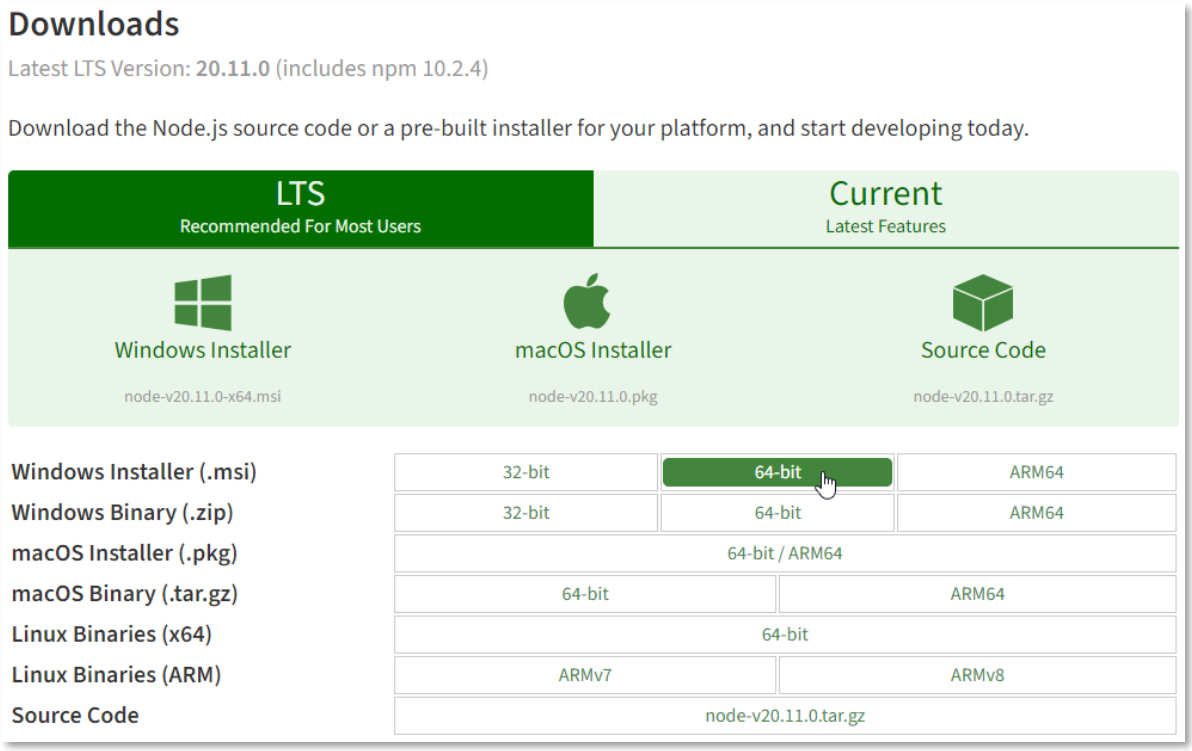


Next step:

Install the latest Long-Term Support (LTS) version of Node.js to support building and run the Contract Approval application.

2.3 Install the LTS version of Node.js

1. Go to https://nodejs.org/en/download/.
2. Locate the Node.js LTS version that corresponds with your Operating System and download the installer.

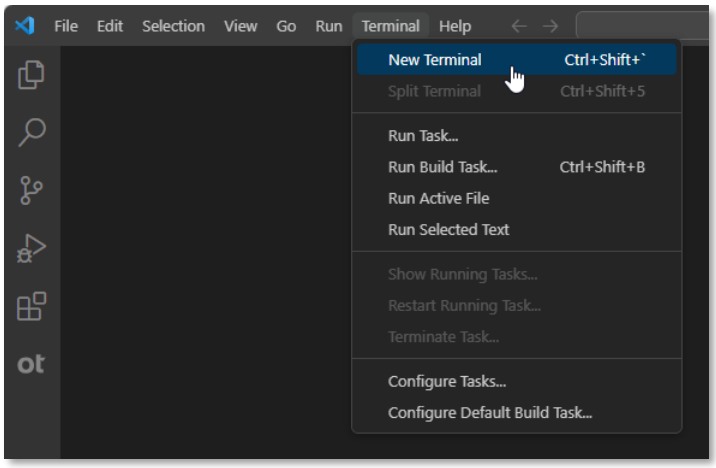


3. Run the installer. The **Node.js Setup Wizard** is displayed.
4. Click **Next**.
5. Select **I accept the terms in the License Agreement** and click **Next** to continue.
6. Select the installation destination location and click **Next**. You can use the suggested default location.
7. From the **Custom Setup** screen, you can accept the default and click **Next**.
8. From the Tools for Native Modules screen, you can accept the default and click **Next**.
9. Click **Install** to start the installation.
10. After the setup is complete, click **Finish** to close the **Node.js Setup Wizard**.

2.4 Verify the Node.js installation from VS Code

After Node.js is installed, verify that it is ready to be used when running Node.js code from VS Code.

1. In VS Code, select **Terminal > New Terminal**.



2. In the TERMINAL, type `node --version`. The installed version of Node.js is displayed (for example, 20.11.0).

Next exercise module:

Add an organization, a tenant, and test the connection.

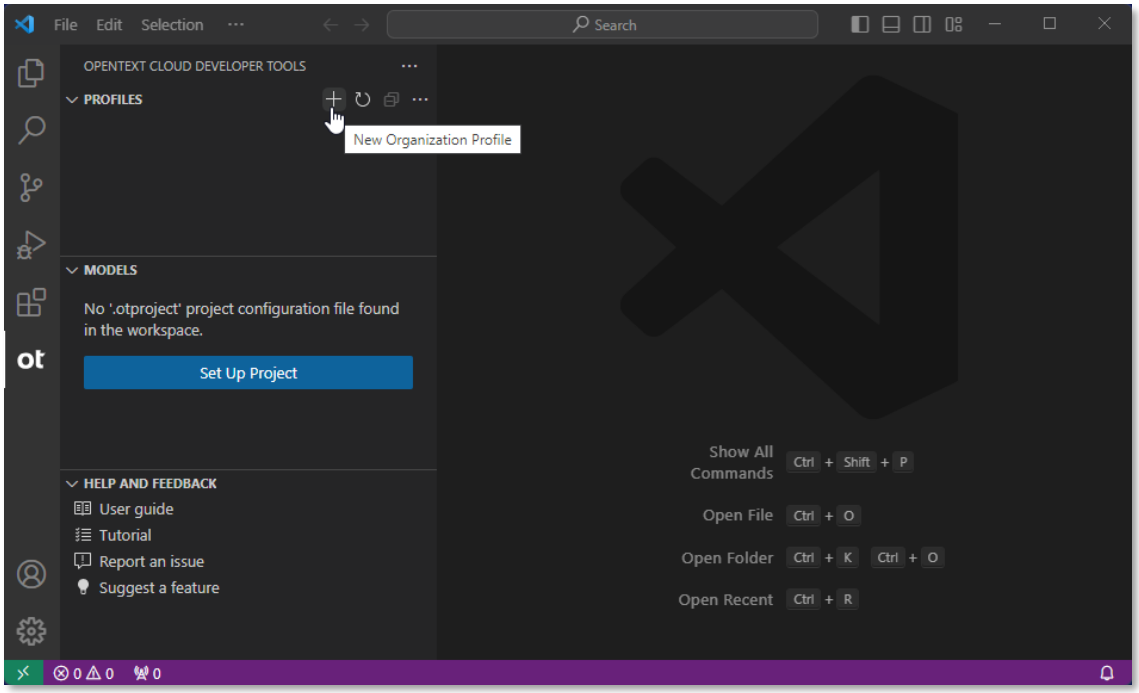
3 [10’] Set up a connection to the developer organization

Learn how to:

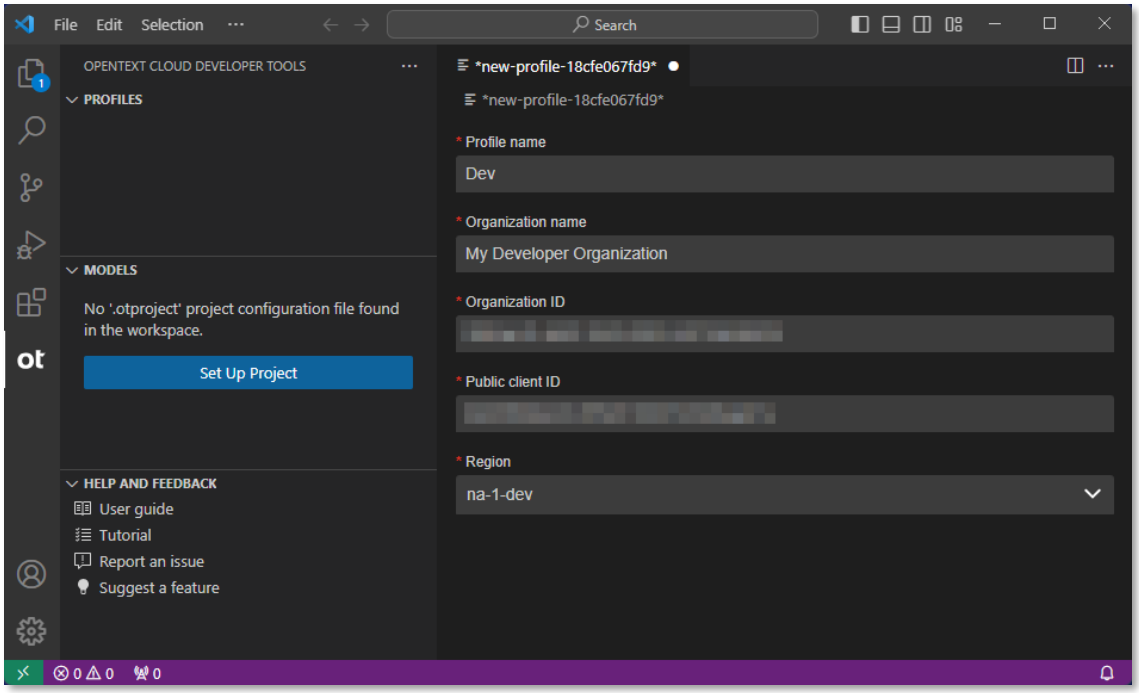
- Add an organization profile in VS Code to allow connecting to your developer organization
- Test the connection
- Add a tenant to the organization profile to allow deploying applications to that tenant

3.1 Add an organization profile

1. Open VS Code and on the Activity Bar, select the **OpenText Cloud Developer Tools** view.
2. In the **PROFILES** section, click the **New Organization Profile** button to create a new organization profile.



3. In the organization profile screen, fill the property fields:



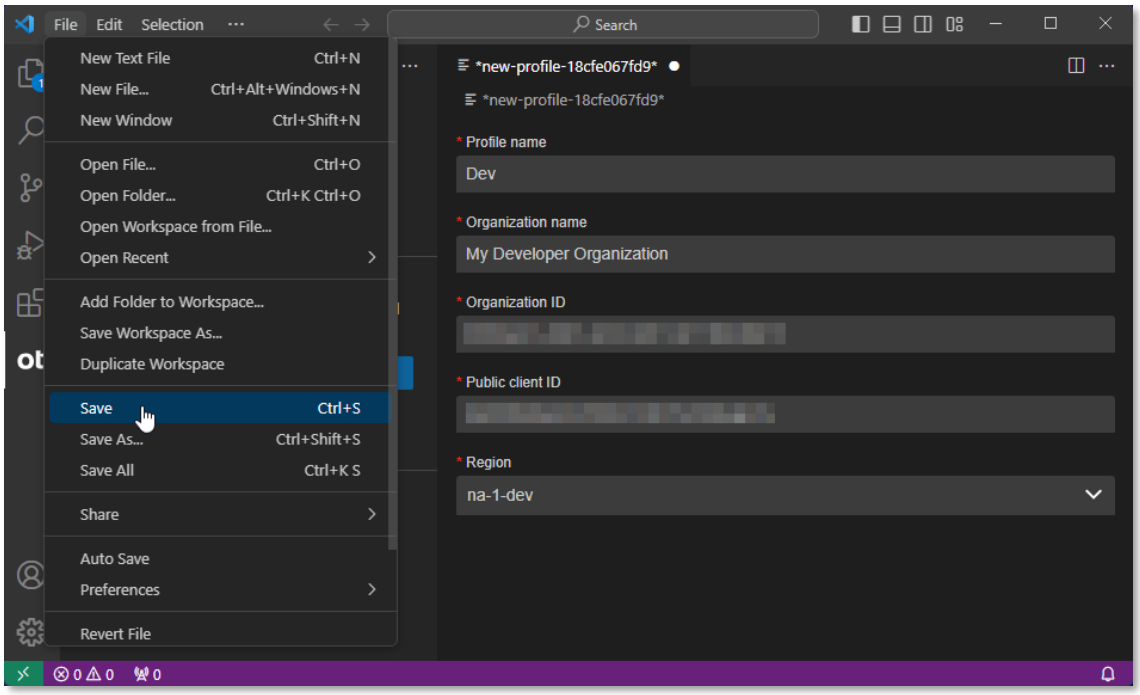
The following table describes the organization profile property fields:

Property	Description
Profile name	The profile name is used as a display name in the PROFILES section.
Organization name	The name of the organization. It is recommended to use the value from the organization overview page in Admin Center. For more information on accessing the organization overview page in Admin Center, see Admin Center User Interface.
Organization ID	The unique identifier for the organization. For more information about how to view your organization ID, see View your organization ID using Admin Center.
Public client ID	The unique identifier of the public OAuth service client for the organization. For more information about how to view your public client ID, see View and manage OAuth service clients.
Region	The region where the organization is made available. By default, the value is na-1-dev as this is the region for trial and developer plans. If you want to check the region of your organization, you can do this by opening Admin Center for the organization and using the URL (region is part of the domain name).

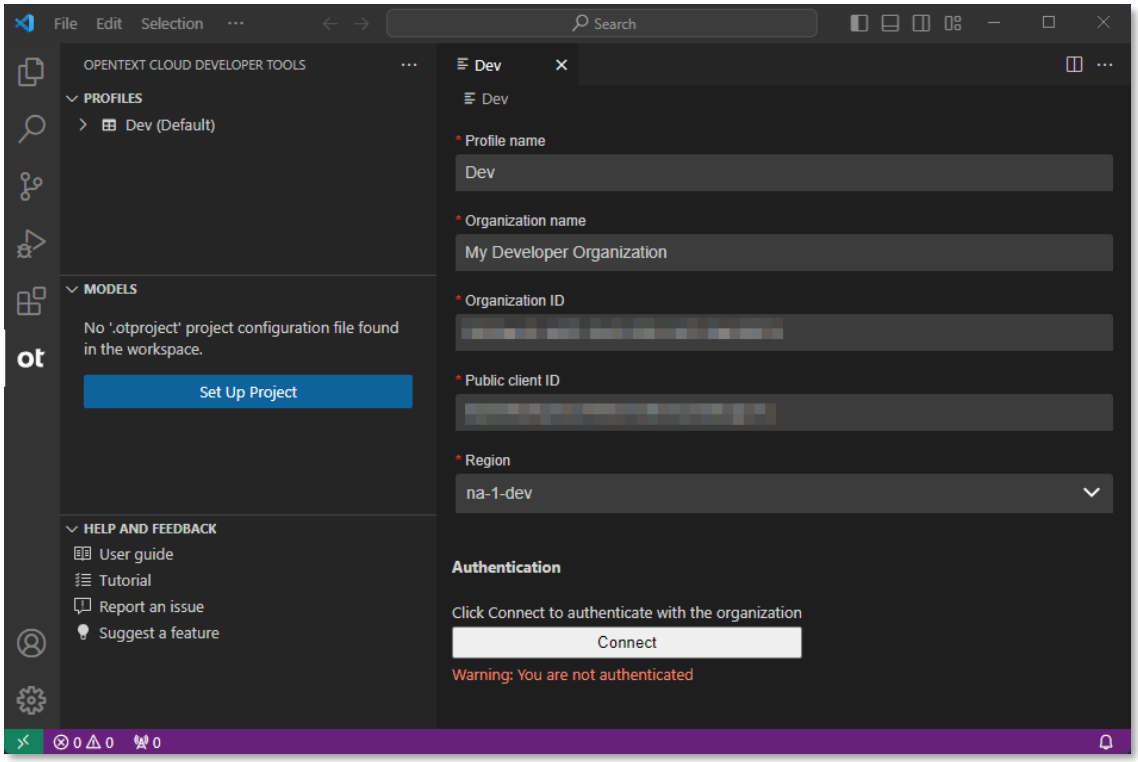
4. Select **File > Save** to save the organization profile.

Note

OpenText Cloud Developer Tools configuration artifacts such as setting up organization connection, project set up, and different model configurations use the standard VS Code file saving functionality. To save your changes to any configuration artifact, press **Ctrl+S** (for Windows systems) or select **File > Save**.



5. After saving, the organization profile is listed as the default profile in the **PROFILES** section and the tab above the organization profile is renamed to the profile name. On the organization profile screen an **Authentication** section with the **Connect** button appears.

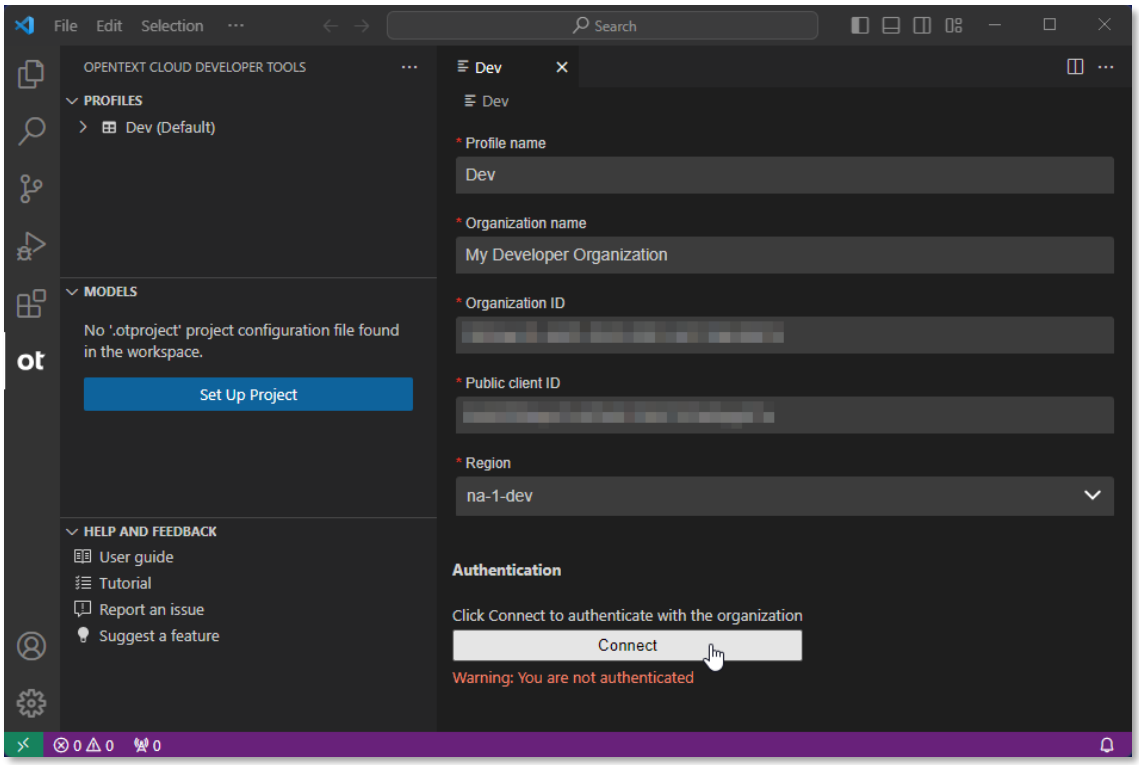


Next step:

Test the connection.

3.2 Test the connection

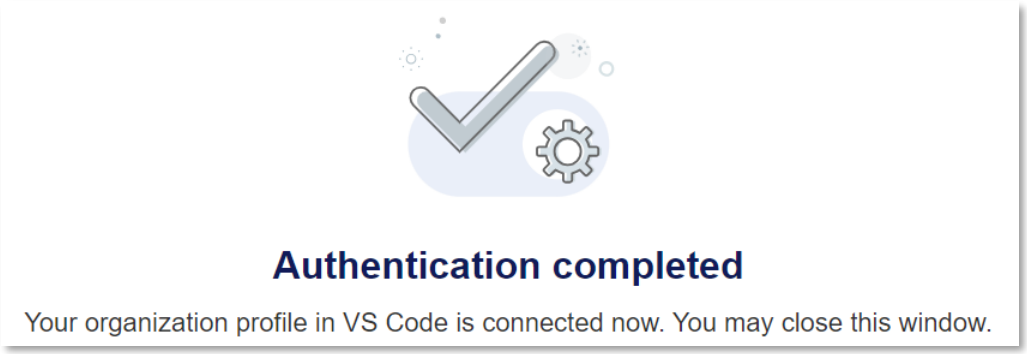
1. Click **Connect** to test the newly configured connection to your developer organization.



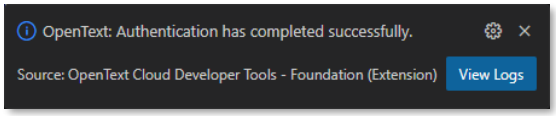
2. On the sign in page, authenticate with your developer.opentext.com username and password.

Note

If you are already signed in, the browser will immediately show the **Authentication completed** page.



3. After successful authentication, a confirmation message is displayed in VS Code.



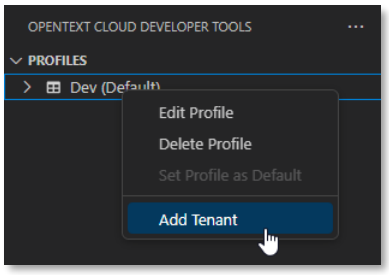
4. Close the organization profile screen.

Next step:

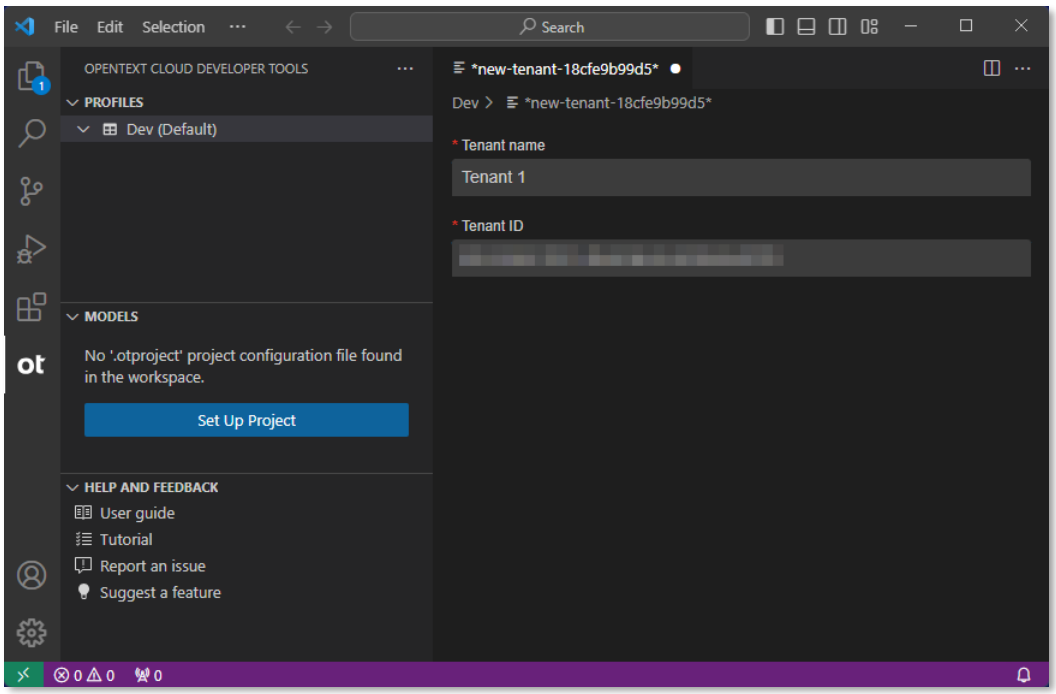
Add a tenant to the organization profile.

3.3 Add a tenant to the organization profile

1. In the **PROFILES** section, right-click the profile and select **Add Tenant**.



2. In the tenant screen, fill the property fields.



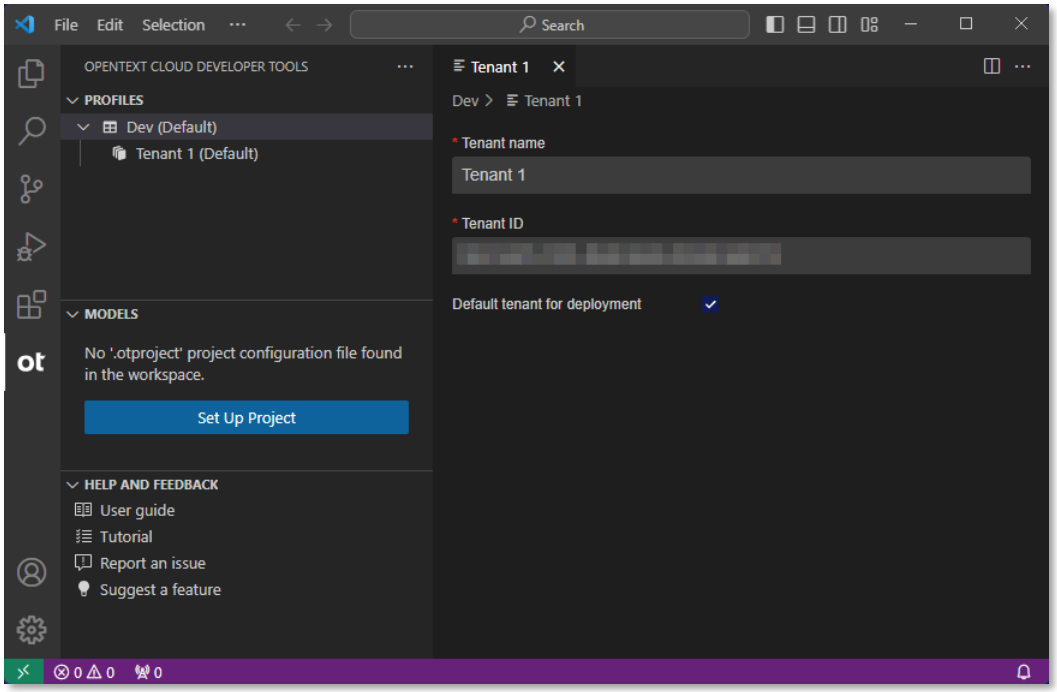
The following table describes the tenant property fields:

Property	Description
Tenant name	The name of the tenant. It is recommended to use the value from the tenant page in Admin Center. For more information on accessing the tenant page in Admin Center, see Retrieve your tenant ID or reset tenant password using Admin Center.
Tenant ID	The unique identifier for the tenant. For more information about how to view your tenant ID, see Retrieve your tenant ID or reset tenant password using Admin Center.

3. Save the tenant.

Note

The first tenant added to the organization profile is made the default tenant for deployment.



Next exercise module:

Create an OpenText project.

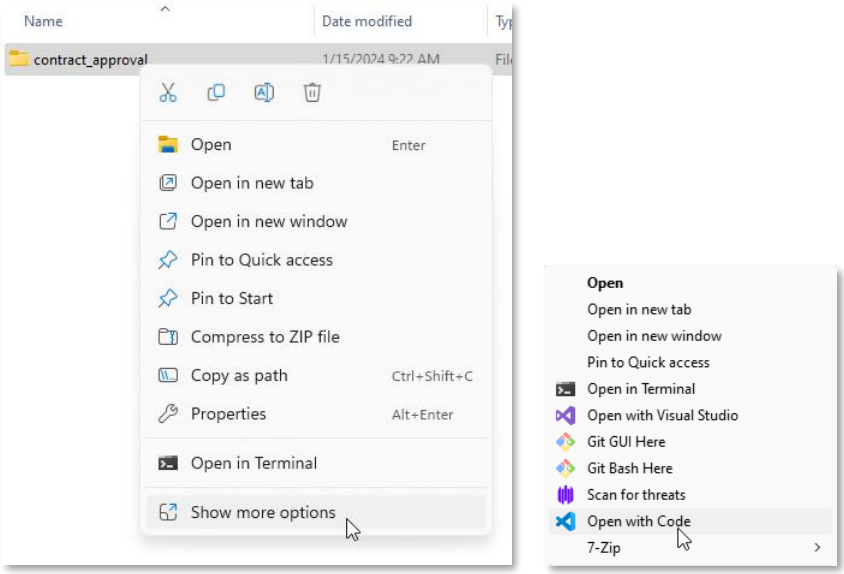
4 [10’] Create an OpenText project

Learn how to:

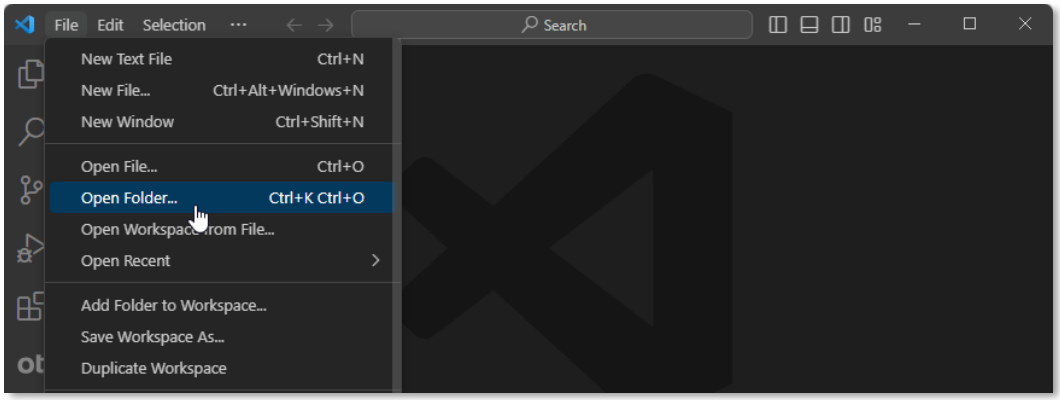
- Create a folder in your file system for building the Contract Approval application
- Set up an OpenText project for the Contract Approval application models

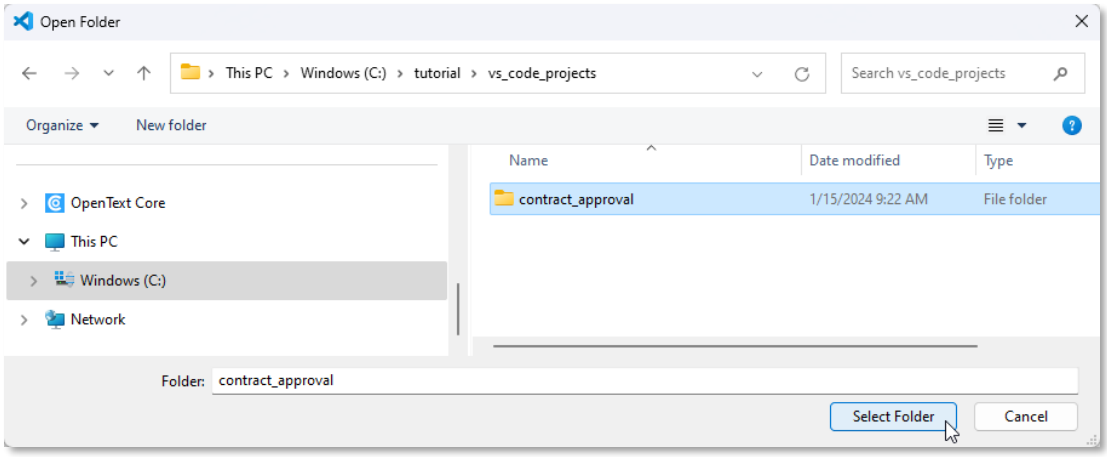
4.1 Create a file system folder for the Contract Approval application project

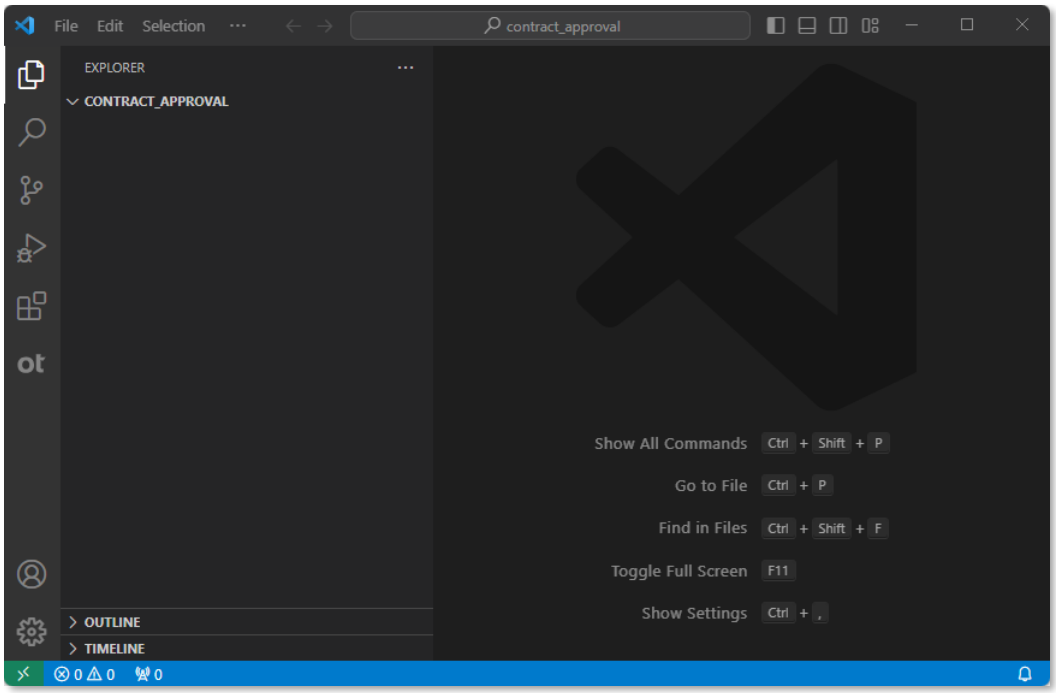
1. In your system’s file explorer (for example, Windows File Explorer) create a new folder. Name the folder **contract_approval**.
2. Do any one of the following to open the Contract Approval application project folder in VS Code:
   - In the Windows File Explorer, right-click the newly created **contract_approval** folder and select **Show more options** and then click **Open with Code**.



   - In VS Code, click **File > Open folder** and select the folder.





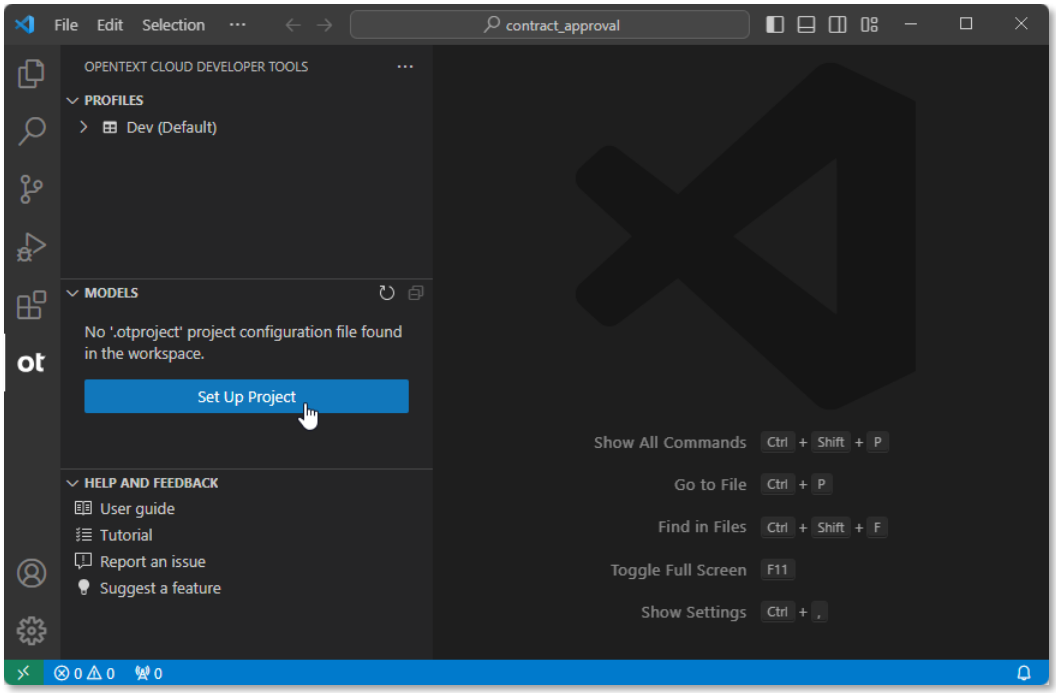


Next step:

Set up the OpenText project to allow building the Contract Approval application models.

4.2 Set up an OpenText project for the Contract Approval application

1. Go to the OpenText Cloud Developer Tools view and click **Set Up Project**.

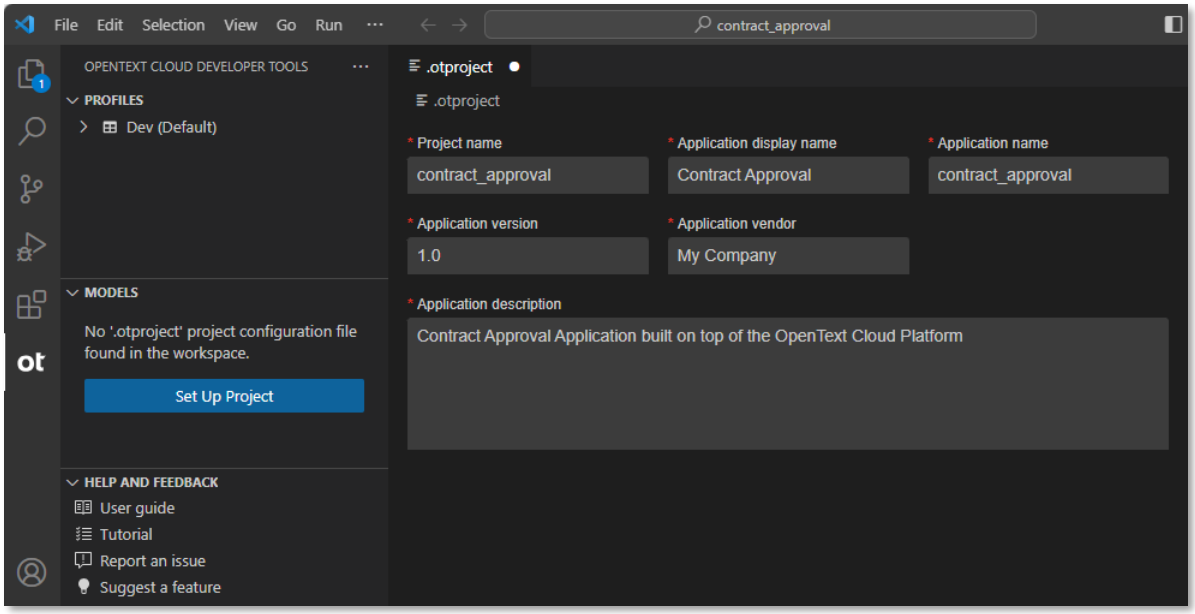


2. Fill the OpenText project properties:

Field	Value
Project name	The project name has been automatically populated from the project folder name and does not need to be changed.
Application display name	`Contract Approval`
Application name	The system automatically populates the application name from the display name. Leave the application name to the generated `contract_approval` value.
Application version	`1.0`
Application vendor	Name of the organization that owns the application. In this example, `My Company` is used.
Application description	`Contract Approval Application built on top of the OpenText Cloud Platform`

Note

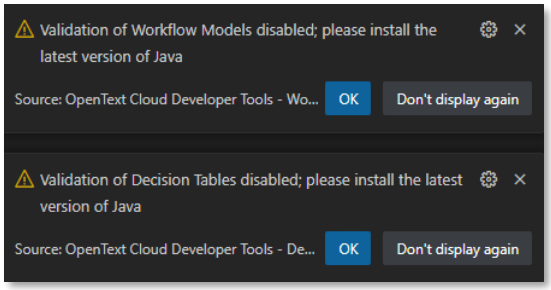
You can always view and modify the OpenText project properties from the VS Code Explorer view by choosing **OpenText: Project Properties** from the contextual menu of your project (root) folder or any of its subfolders, or by clicking/opening the **.otproject** file.



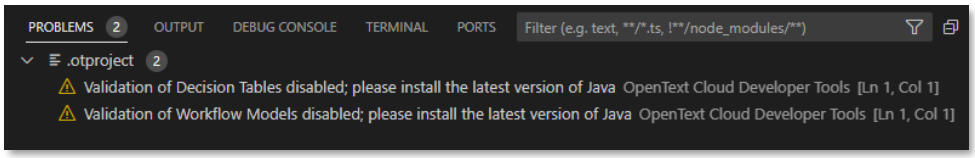
3. Save and close the OpenText project properties form.

Note

If Java is not installed on your system, VS Code will display the following two warning messages:



You can click **Don’t display again** to stop these warning messages from popping up. However, in addition to the pop ups, the error message will show under the **PROBLEMS** tab until Java is installed. To open the **PROBLEMS** tab, select **View > Problems**.

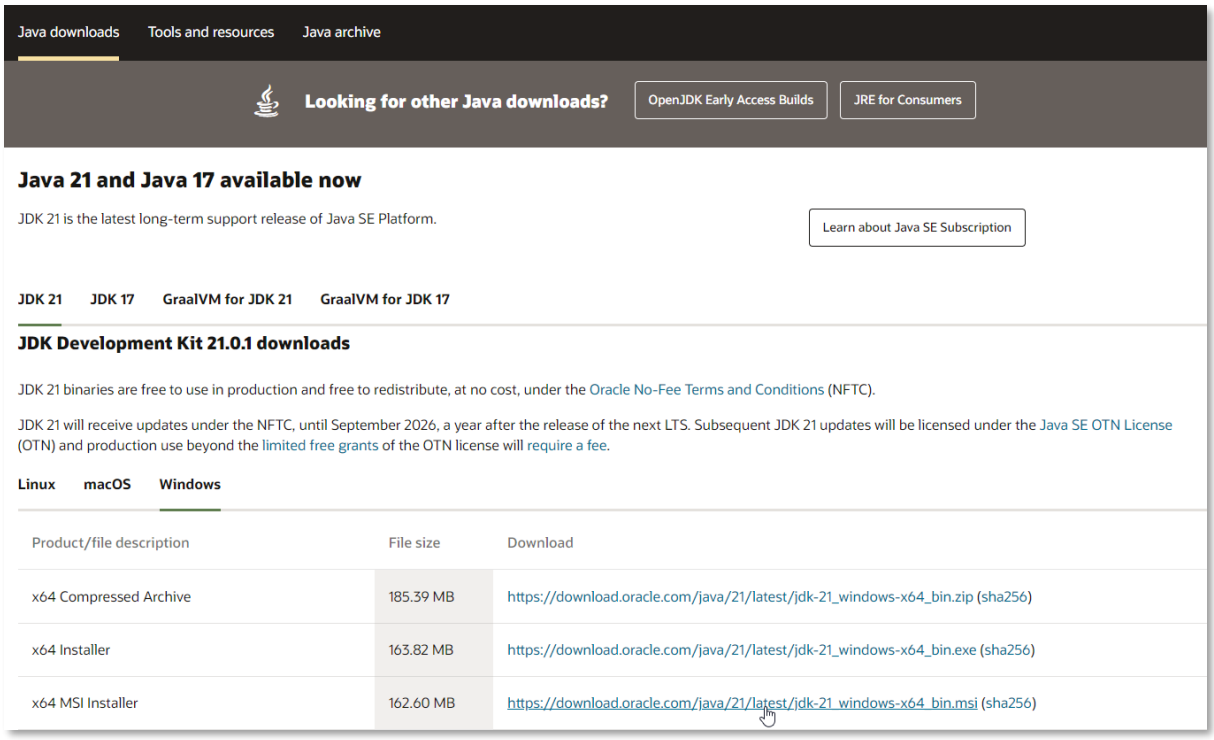


Tip

Install the latest LTS Java version. To install LTS Java version, see Install Java.

4.3 Install Java

1. Go to https://www.oracle.com/java/technologies/downloads.
2. Download and install the latest Java LTS version for your operating system.



Note

For this tutorial, the latest LTS JDK version is **21.0.1**, and the **x64 MSI Installer** for Windows is used.

Next exercise module:

Create a namespace.

5 [5’] Create a namespace

Learn how to:

- Create a namespace model

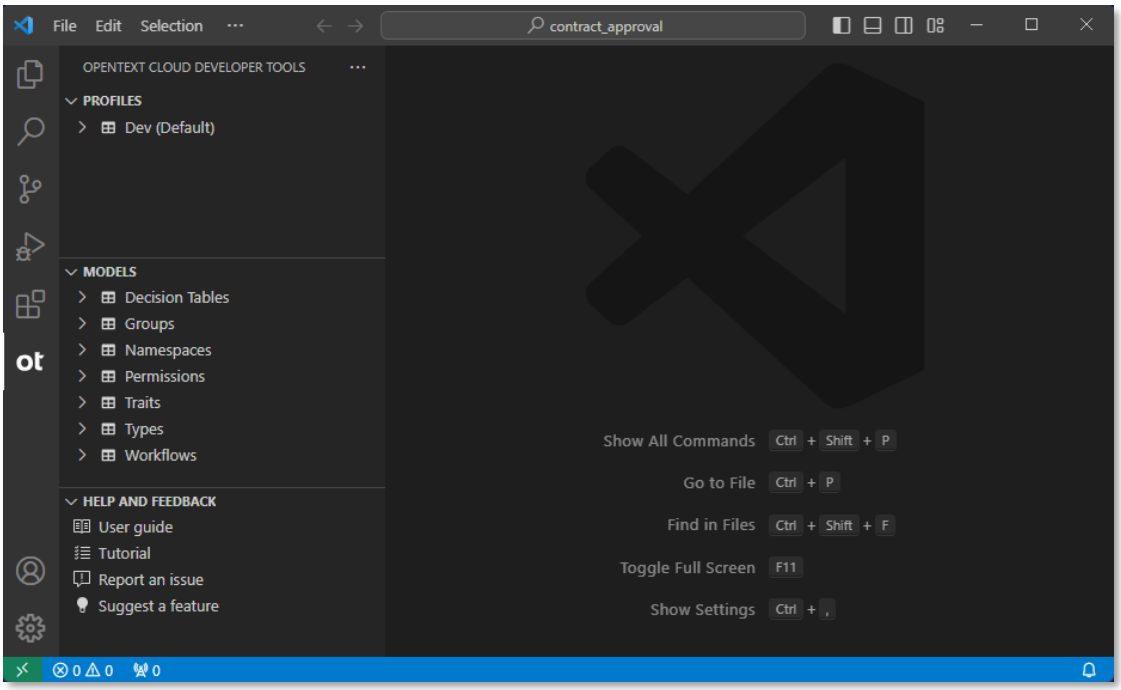
A namespace allows grouping different type, trait, workflow, and decision table definitions together. For example, within the context of an application.

For more information on namespaces, see Define a namespace, trait and "FILE" document type section in the Content Metadata Service product documentation or the **Namespace** resource documentation in the Content Metadata Service API reference.

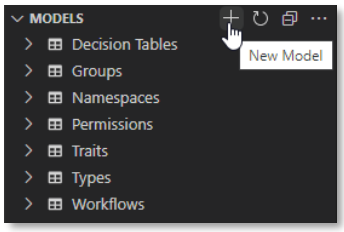
5.1 Create the Contract Approval namespace

1. In VS Code, switch to the **OpenText Cloud Developer Tools** view.

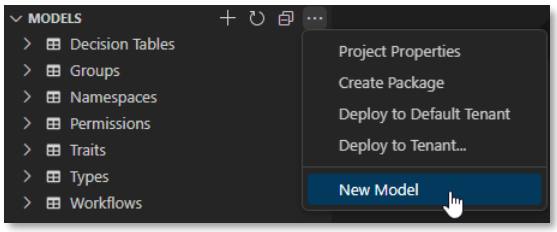
> **Note**
>
> After setting up the OpenText project, the **MODELS** section displays a tree view and allows you to explore the different models in your application project.



2. To create a new namespace, in the **MODELS** section, do any one of the following:
   - Click the **New Model** button .



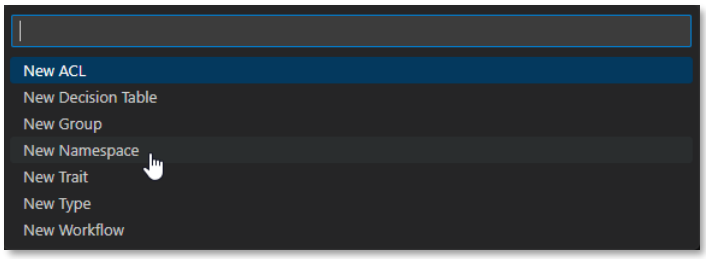
   - Click the **More Actions** button and select **New Model**.



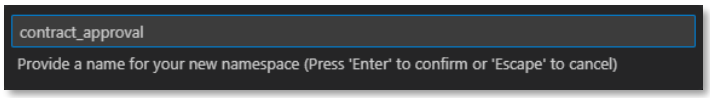
Note

There are different methods to create models. This is the first method. To understand the other methods for creating models, see Create the Approval trait and Create the Contract type.

3. From the list, select **New Namespace**.



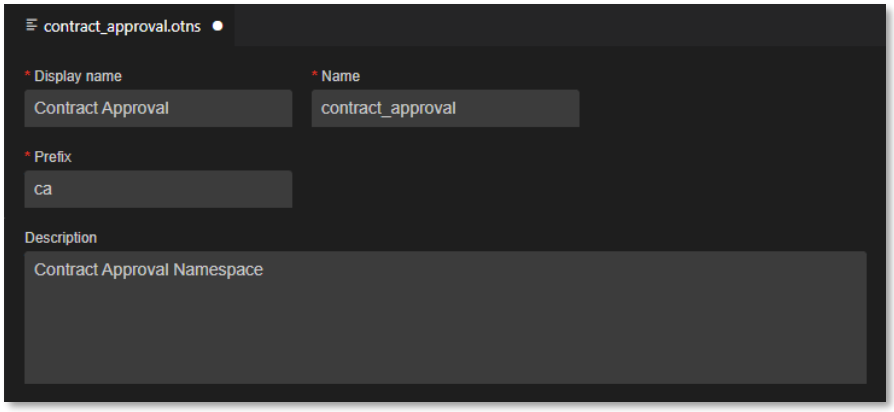
4. In the input box, type `contract_approval` for the (file) name of the namespace and press **Enter**.



5. Fill the Approval namespace properties using the following details:

Field	Value
Display name	The display name for the namespace. This is automatically populated based on the previously chosen namespace name (the model file name). Leave the value to be `Contract Approval`.
Name	The technical name for the namespace. This name must be unique within the tenant, and it is automatically populated based on the previously chosen namespace name (the model file name). Leave the value to be `contract_approval`.

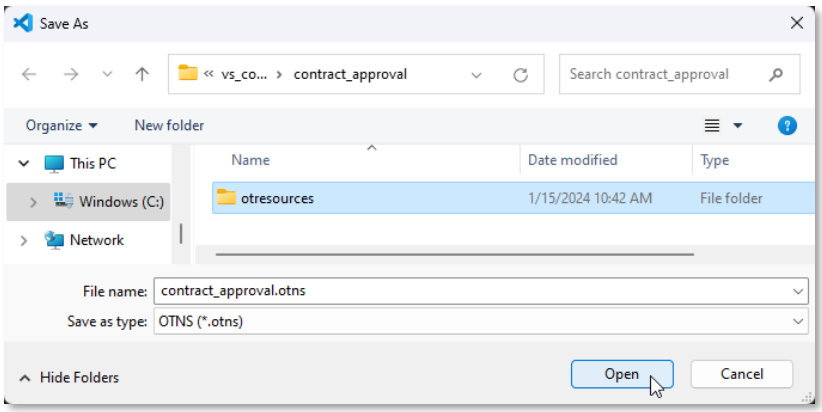
Field	Value
Prefix	The prefix representing the namespace. It is used for the (internal) system names of traits and types that belong to the namespace. The prefix must be unique within the tenant. Enter `ca` as value.
Description	`Contract Approval Namespace`

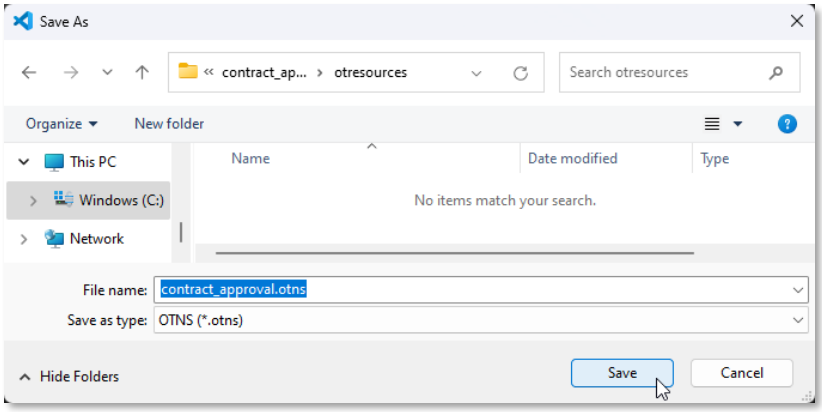


6. Save the Contract Approval namespace model.
7. In the **Save As** dialog box that opens when saving the model, select the **otresources** folder as target folder and make sure the file name is `contract_approval.otns`.

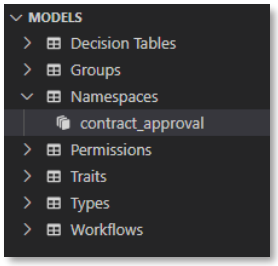
Note

The **otresources** folder is the model folder and it gets generated automatically during the project setup. You must save all the models inside this folder or one of its subfolders (as you can create sub folders to organize your work).





8. Click **Save** and close the namespace model.
9. The model explorer displays the new **contract_approval** namespace under **Namespaces**. The model explorer shows the different models according to their unique key, which in context of a namespace is the name property.



Next exercise module:

Create a trait model.

6 [10’] Create a trait

Learn how to:

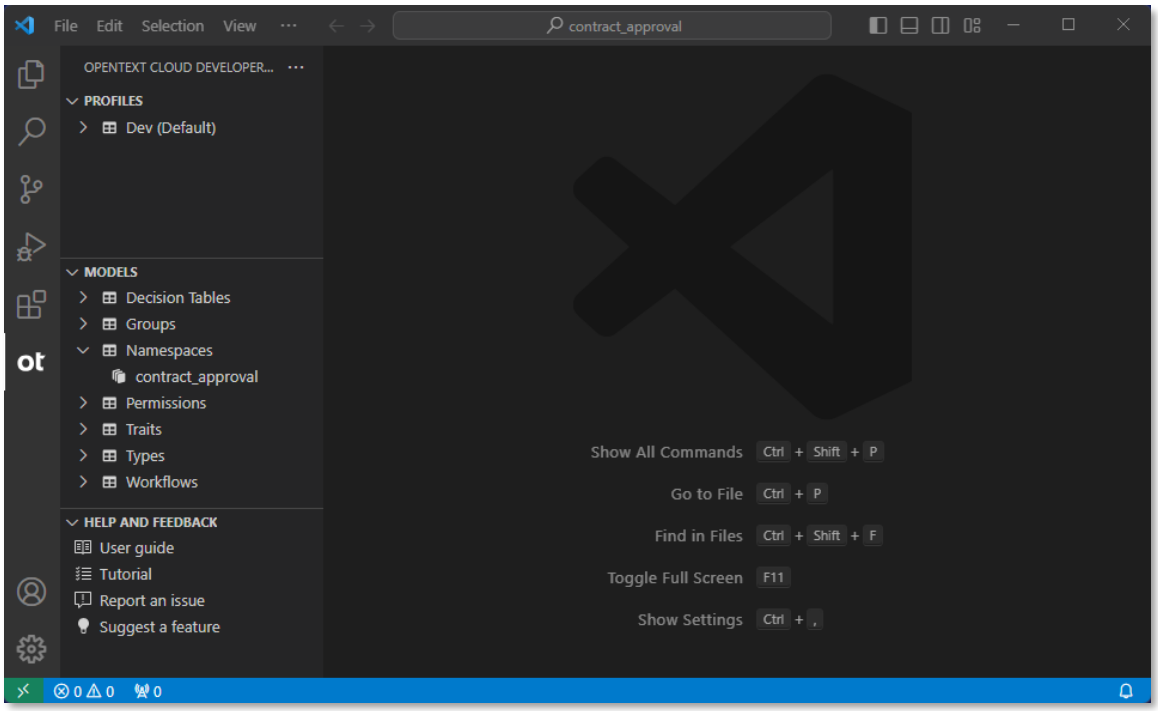
- Create a trait model

A trait allows grouping several attributes into one complex multi-attribute property. Trait instances can be dynamically added to a type instance as part of the business process when using the application. Traits can also be made mandatory in a type so that they must always be added when creating a new type instance. In this tutorial, the concept of mandatory traits is used to represent the different approval steps on a contract.

For more information on traits, see Define a namespace, trait and "FILE" document type and Create instances using custom type with trait sections in the Content Metadata Service product documentation or the **Trait** resource documentation in the Content Metadata Service API reference.

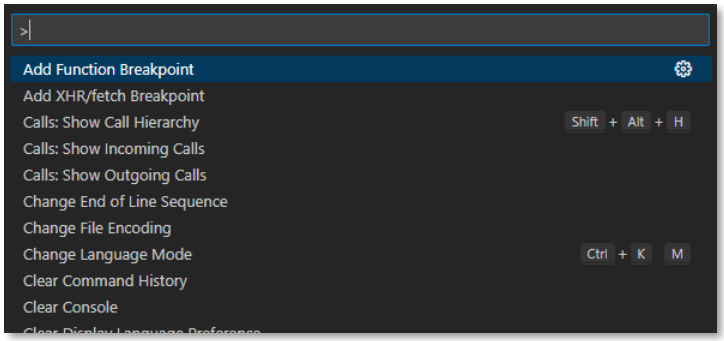
6.1 Create the Approval trait

1. In VS Code, switch to the **OpenText Cloud Developer Tools** view.



2. To create a new trait, press **F1** or **Ctrl+Shift+P**.

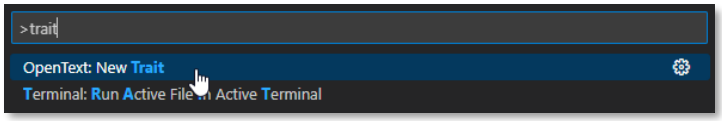
   The command palette is displayed.



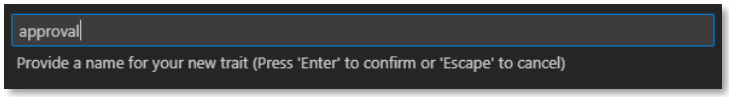
Note

There are different methods to create models. This is the second method. To understand the other methods to create models, see Create the Contract Approval namespace and Create the Contract type.

3. In the **command palette**, type `trait`. The command list is filtered and the entries containing trait are displayed.
4. From the list, select **OpenText: New Trait**.



5. In the **Provide a name for your new trait** input box, type `approval` for the (file) name of the trait and press **Enter**.



Note

The chosen name of “approval” reflects the purpose of this specific trait.

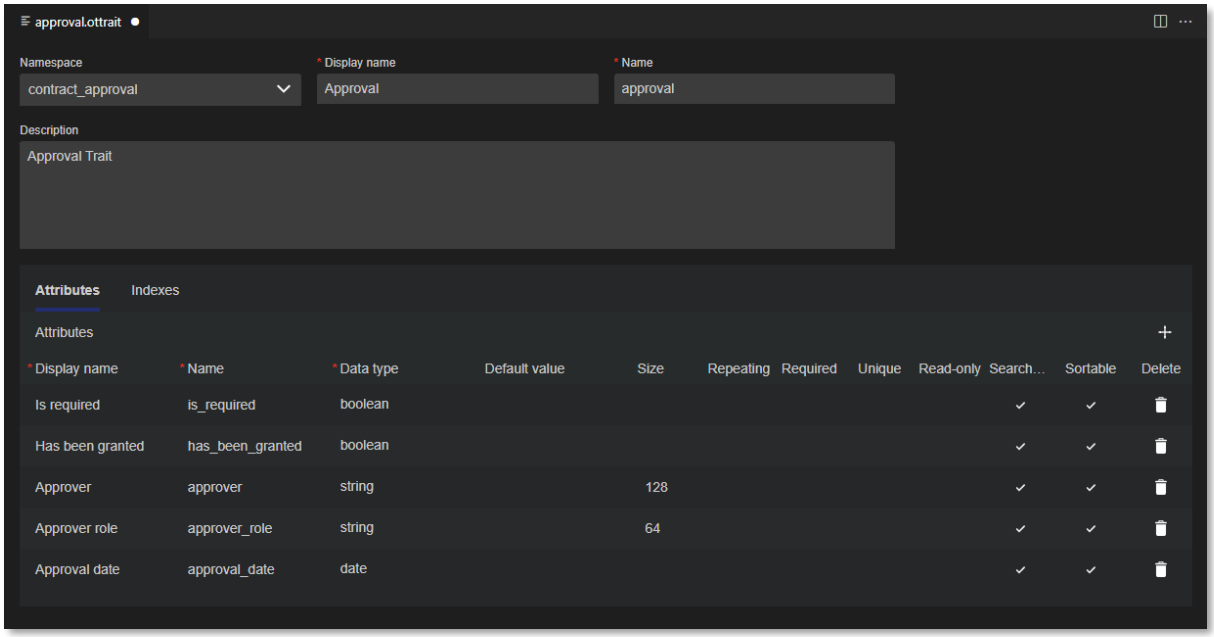
6. Fill the Approval trait properties using the following details:

Field	Value
Namespace	The namespace to which the trait belongs. Select the **contract_approval** namespace as the namespace.  **Note** The OpenText Cloud Developer Tools dynamically update the different model reference lists. For a trait, the list of available namespaces is dynamically updated based on the namespaces that exist in the project.
Display name	The display name is the user-friendly name for the trait. This does not have to be unique, and it is automatically populated based on the previously chosen trait name (the model file name). Leave the value to be `Approval`.
Name	The technical name for the trait. This name must be unique within the context of the tenant and selected namespace. It is automatically populated based on the previously chosen trait name (the model file name). Leave the value to be `approval`.
Description	`Approval Trait`
Attributes	The attributes list defines the different attributes of the trait. The following are the attribute properties: • Display name: The attribute display name. OpenText recommends that the display name is unique. • Name: The attribute technical name. The value must be unique within the trait, and it is automatically populated based on the specified display name. • Data type: The data type of the attribute. The data type can be bigint, boolean, date, datetime, double, id, integer, string, or user. • Default value: The default value for the attribute, which is automatically assigned when creating a new instance of the trait. Whether it is possible to assign a default value and how to assign it depends on the chosen data type. • Size: Indicates the maximum length constraint for the string attribute. Applicable only to the string data type. • Repeating: Indicates if the attribute can have multiple values. • Unique: Indicates if the attribute needs to be unique across all instances of the trait. • Required: Indicates if the attribute must have a value when creating an instance of the trait. • Read-only: Indicates if the attribute can be modified after creation. • Searchable: Indicates if the attribute can be filtered on when performing a search. • Sortable: Indicates if the attribute can be used to sort a search result. You will add the attributes for the Approval trait in the next step.

7. In the **Attributes** list, click the **Add** button ⊞ to add the different attributes.

   The Approval trait represents an approval step, and the following table describes each attribute and the property values to assign:

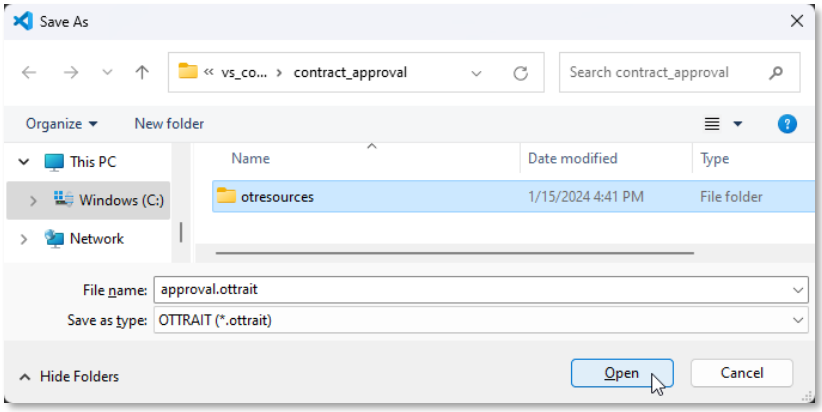
Attribute description	Display name	Name	Data type	Default value	Size	Boolean properties
Whether or not the approval is required	`Is required`	`is_required`	**boolean**			**searchable, sortable**
Whether or not the approval has been granted	`Has been granted`	`has_been_granted`	**boolean**			**searchable, sortable**
The email address of the approver	`Approver`	`approver`	**string**		`128`	**searchable, sortable**
The role of the approver	`Approver role`	`approver_role`	**string**		`64`	**searchable, sortable**
The exact date and time at which the approval request has been approved or at which it has been rejected	`Approval date`	`approval_date`	**date**			**searchable, sortable**

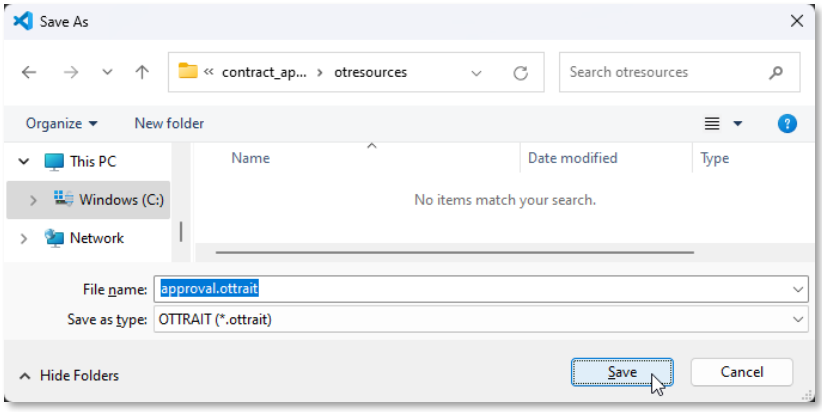


8. Save the Approval trait model.
9. In the **Save As** dialog box that opens when saving the model, select the **otresources** folder as target folder and make sure the file name is `approval.ottrait`.

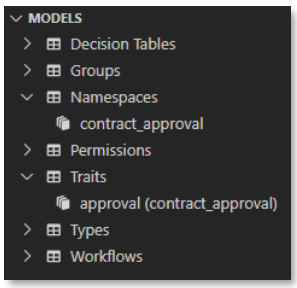
Note

The **otresources** folder is the model folder and it gets generated automatically during the project setup. You must save all the models inside this folder or one of its subfolders (as you can create sub folders to organize your work).





10. Click **Save** and close the trait model.
11. The model explorer displays the new **approval (contract_approval)** trait under **Traits**. The model explorer shows the different models according to their unique key, which in context of a trait is the combination of the namespace and name properties.



Next exercise module:

Create types.

7 [25’] Create types

Learn how to:

- Create a file type model
- Create a file type model that is a subtype
- Create a folder type model

A type is the main component for building an application’s (custom) data model. A type has its own attributes and required traits that are added to the type instances when they are created.

A type can be of the following categories:

- file
- folder
- object
- relation

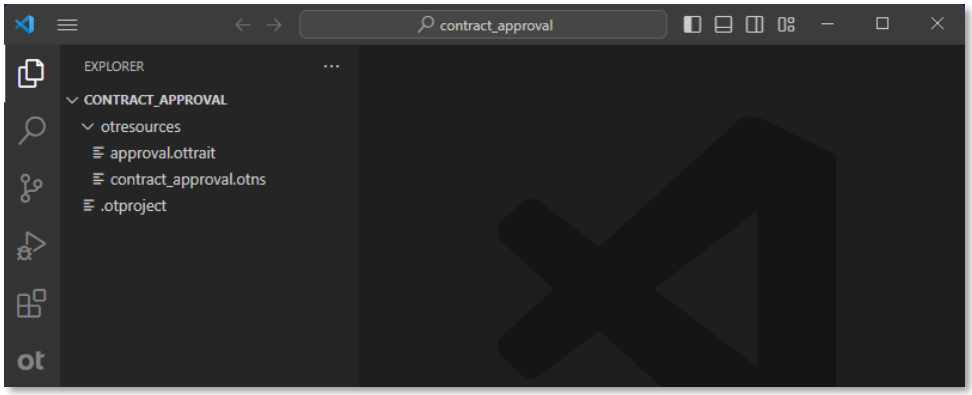
The following types will be created in this exercise module:

Type	Description
Contract	The Contract type represents a standard/base contract. It is the first out of two file types.
Loan Contract	The Loan Contract type is a specialization/subtype of the standard contract. It is the second file type.
Customer	The Customer type is the folder type. It is intended to contain all contracts (standard contracts and loan contracts) related to a specific customer.

For more information about types, see Define a namespace, trait and "FILE" document type and Create instances using custom type with trait sections in the Content Metadata Service product documentation or the **Type** resource documentation in the Content Metadata Service API reference.

7.1 Create the Contract type

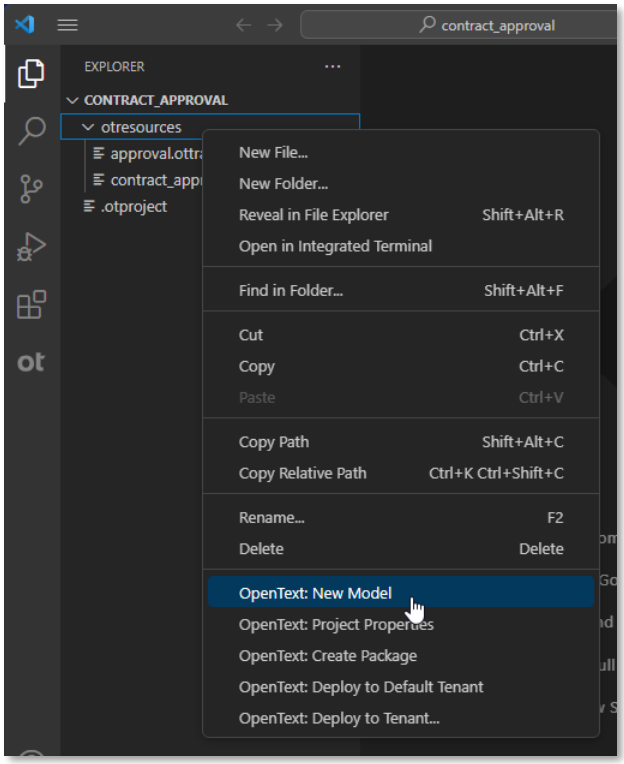
1. In VS Code, switch to the Explorer view.
2. From the **contract_approval** application root folder, expand the **otresources** (model) folder. You can see the previously created **approval** trait and **contract_approval** namespace models.



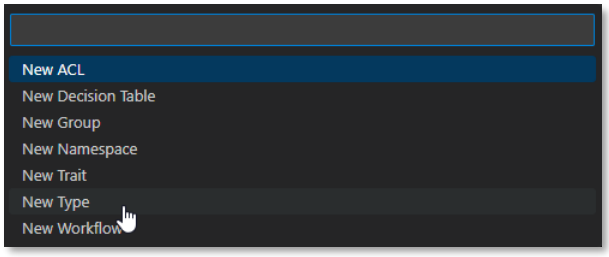
3. Right-click the **otresources** (model) folder and select **OpenText: New Model** to create a new type.

Note

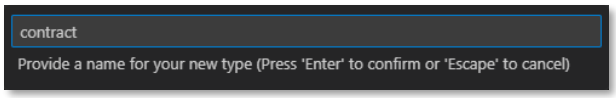
There are different methods to create models. This is the third method. To understand the other methods to create models, see Create the Contract Approval namespace and Create the Approval trait.



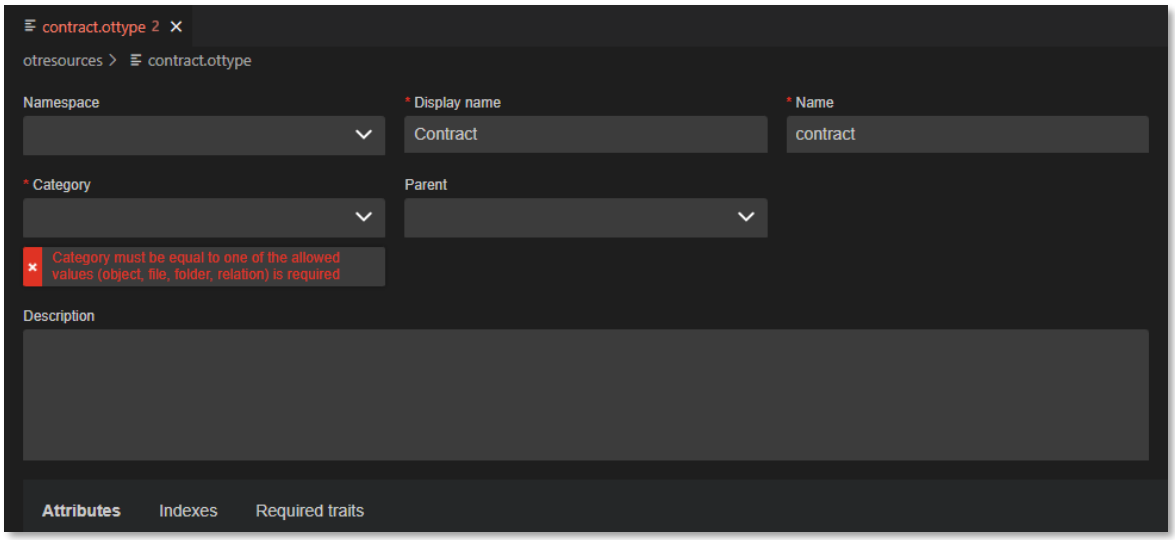
4. From the list, select **New Type**.



5. In the input box, type `contract` for the (file) name of the type and press **Enter**.

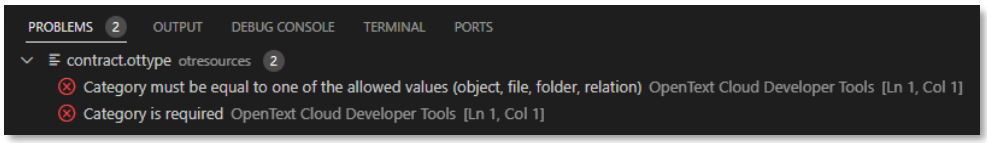


The value for the **Category** property is required and because the model is created directly in the file system (that is, it has already been saved), an error message is displayed for not selecting one of the allowed values.



Note

Validation errors for saved models are displayed under the **PROBLEMS** tab until the issues are resolved. To open the **PROBLEMS** tab, select **View > Problems**.



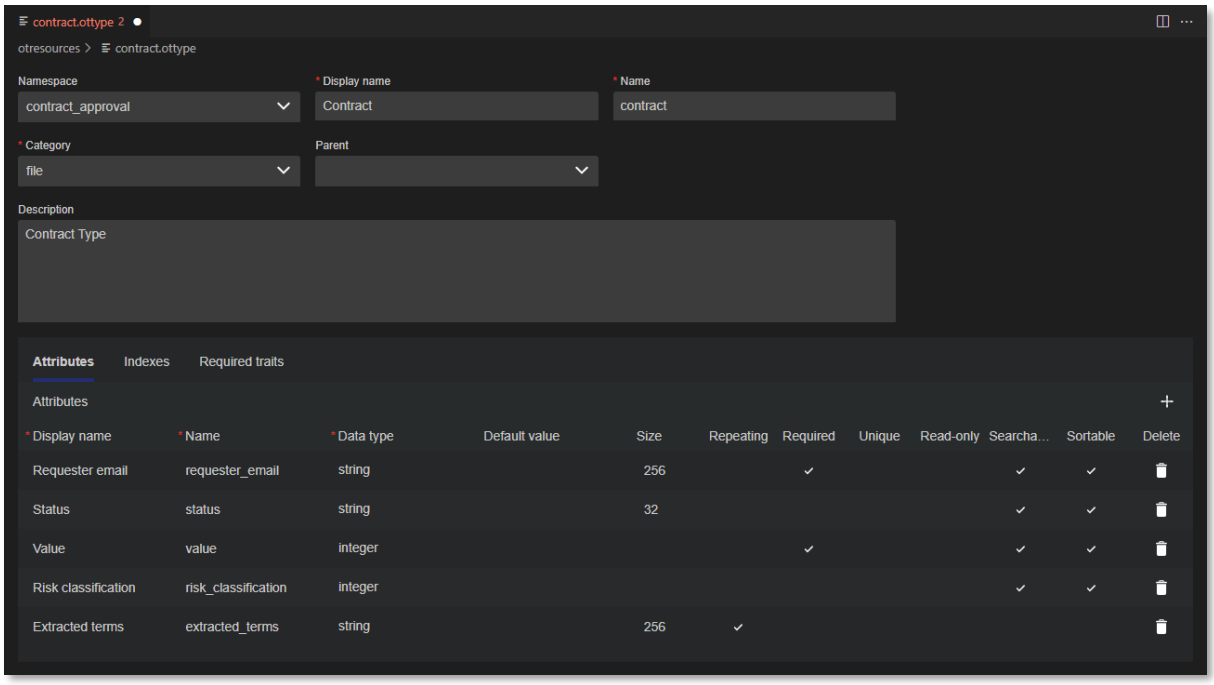
6. Fill the Contract type properties using the following details:

Field	Value
Namespace	The namespace to which the type belongs. Select the **contract_approval** namespace. **Note** The OpenText Cloud Developer Tools dynamically update the different model reference lists. For a type, the lists of available namespaces, traits and parent types are dynamically updated based on the namespaces, traits and types that exist in the project.
Display name	The display name is the user-friendly name for the type. This does not have to be unique, and it is automatically populated based on the previously chosen type name (the model file name). Leave the value to be `Contract`.
Name	The technical name for the type. This name must be unique within the context of the tenant and selected namespace. It is automatically populated based on the previously chosen type name (the model file name). Leave the value to be `contract`.
Category	The category to which the type belongs. It can be a file, folder, object or relation. Select the **file** category.
Parent	The parent type for the type you are creating. The Contract type does not have a parent type as it is the base type for contracts.
Description	`Contract Type`
Attributes	The attributes list defines the different attributes of the type. For more information about attribute properties, see Create the Approval trait. You will add the attributes for the Contract type in step 7.
Required Traits	The required traits list defines the different mandatory traits for the type. The following are the required trait properties: • Instance name: The name of the required trait. This must be unique across the type's required traits. In this tutorial, the required traits are all Approval traits and the instance name will represent the type of approval. • Trait name: The unique key (trait name + namespace) representing the selected trait. In this tutorial, all required traits will be **approval (contract_approval)** traits. You will add the required traits for the Contract type in step 9.

7. In the **Attributes** list, click the **Add** button ➕ to add the different attributes.

   The following table describes each attribute and the property values to assign:

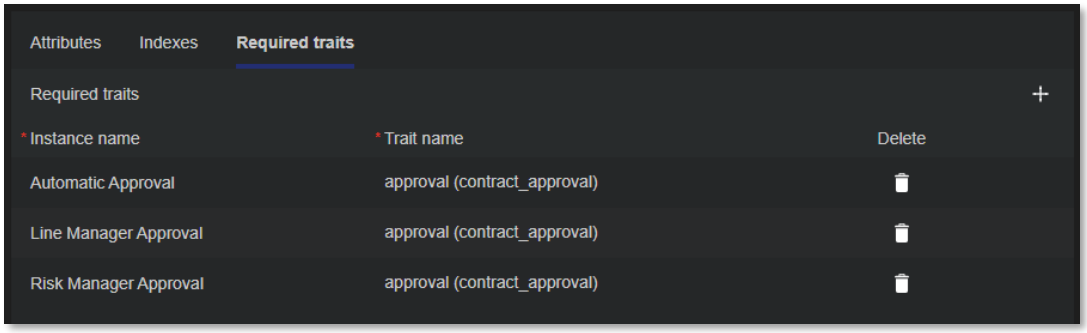
Attribute description	Display name	Name	Data type	Default value	Size	Boolean properties
The email address of the person requesting the approval of the contract	`Requester email`	`requester_email`	**string**		`256`	**required, searchable, sortable**
The current (approval) status of the contract	`Status`	`status`	**string**		`32`	**searchable, sortable**
The (monetary) value of the contract	`Value`	`value`	**integer**			**required, searchable, sortable**
The risk classification of the contract in context of the personal data it contains	`Risk classification`	`risk_classification`	**integer**			**searchable, sortable**
The personal data related terms that were found in the contract	`Extracted terms`	`extracted_terms`	**string**		`256`	**repeating**



8. Select the **Required traits** tab and click the **Add** button ➕ to add the required traits.

   The following table describes each required trait and the property values to assign:

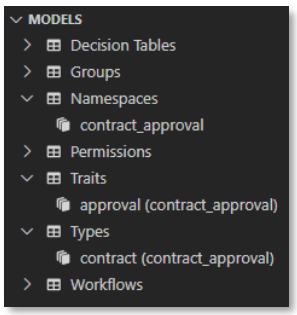
Required trait	Instance name	Trait Name
The automatic (by the system) approval, which is always required	`Automatic Approval`	**approval (contract_approval)**
The approval by the Line Manager, which is only required when the contract value is above 1000	`Line Manager Approval`	**approval (contract_approval)**
The approval by the Risk Manager, which is only required when the risk classification is above 3. For example, 4: HIGH or 5: VERY HIGH.	`Risk Manager Approval`	**approval (contract_approval)**



9. Save and close the Contract type model.

> **Note**
>
> Since you correctly filled the model properties, there no longer are validation errors under the **PROBLEMS** tab.

10. The model explorer displays the new **contract (contract_approval)** type under **Types**. The model explorer shows the different models according to their unique key, which in context of a type is the combination of the namespace and name properties.



Next step:

Create the Loan Contract type.

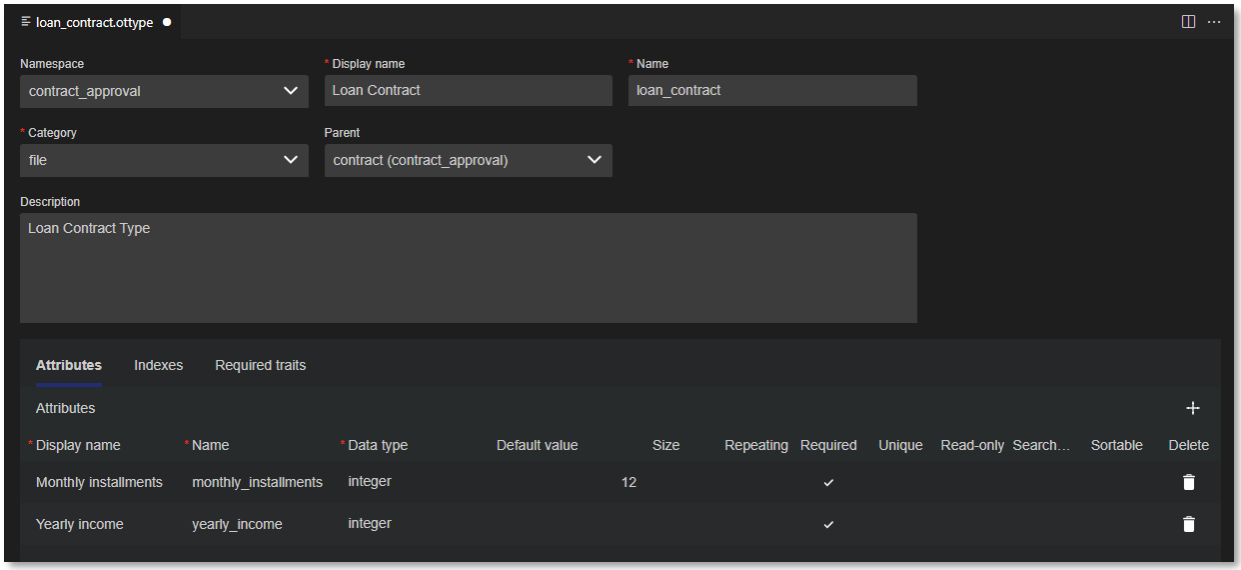
7.2 Create the Loan Contract type

1. In VS Code, create a new type model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `loan_contract` for the (file) name of the type.
3. Fill the Loan Contract type properties using the following details:

Field	Value
Namespace	Select the **contract_approval** namespace.
Display name	`Loan Contract`
Name	`loan_contract`
Category	To allow subtyping the Contract type, and thus inheriting all its attributes and required traits, the Loan Contract type must be of the **file** category. Select the **file** category.
Parent	The parent type for the type you are creating. Select the **contract (contract_approval)** type. **Note** The **contract (contract_approval)** parent type is only available to be selected from the Parent list when **file** is selected as category.
Description	`Loan Contract Type`
Attributes	See Attributes.
Required Traits	See Required Traits.

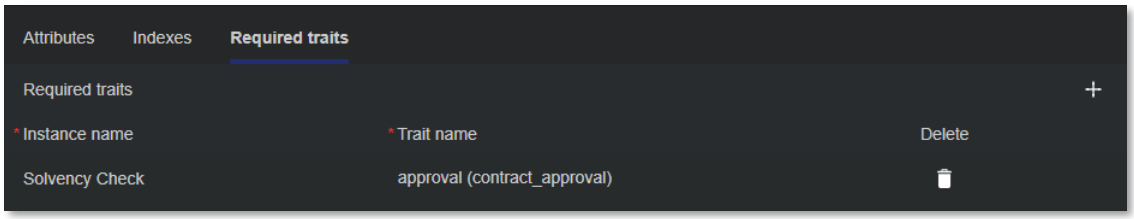
Attributes:

Attribute description	Display name	Name	Data type	Default value	Boolean properties
The total count of monthly payments required/chosen to reimburse the loan contract value	`Monthly installments`	`monthly_installments`	**integer**	`12`	**required**
The yearly income, which will be used together with the monthly payments and the loan contract value to determine solvency of the customer	`Yearly income`	`yearly_income`	**integer**		**required**

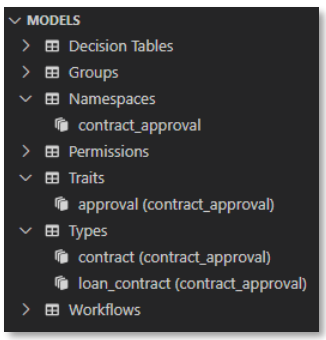


Required traits:

Required trait description	Instance name	Trait name
The automated approval step checks if the customer is solvent by checking the monthly cost. The monthly cost must not exceed 25% of the monthly income (calculated from the loan contract cost, the total count of monthly payments and the yearly income).	`Solvency Check`	**approval (contract_approval)**



4. Save and close the Loan Contract type model. The model explorer displays the new **loan_contract (contract_approval)** type under **Types**.



Next step:

Create the Customer type.

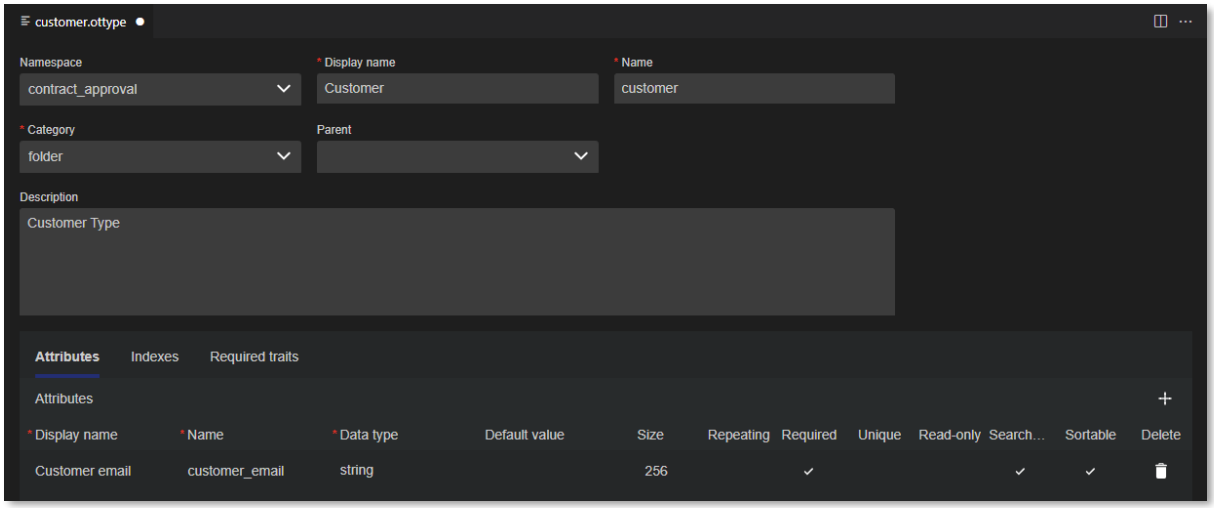
7.3 Create the Customer type

1. In VS Code, create a new type model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `customer` for the (file) name of the type.
3. Fill the Customer type properties using the following details:

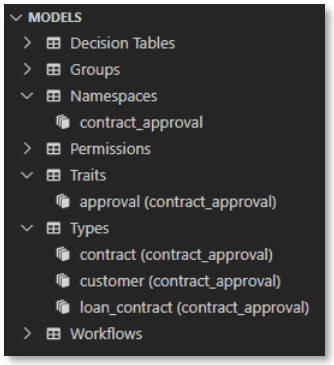
Field	Value
Namespace	Select the **contract_approval** namespace.
Display name	`Customer`
Name	`customer`
Category	Select the **folder** category.
Parent	The Customer type does not have a parent type.
Description	`Customer Type`
Attributes	See Attributes.

Attributes:

Attribute description	Display name	Name	Data type	Default value	Size	Boolean properties
The email address of the customer	`Customer email`	`customer_email`	**string**		`256`	**required, searchable, sortable**



4. Save and close the Customer type model. The model explorer displays the new **customer (contract_approval)** type under **Types**.



Next exercise module:

Create (user) groups.

8 [10’] Create (user) groups

Learn how to:

- Create group models
- Add subgroups

After deploying the application into a tenant, the deployed groups can be populated with users to allow managing role-based behavior and security (through ACLs).

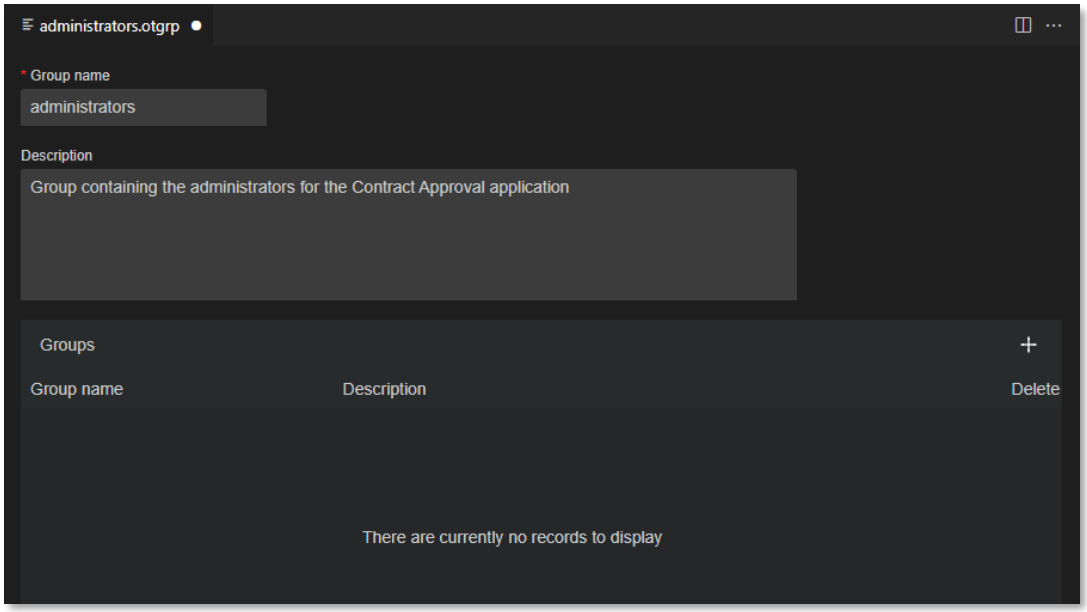
The following user groups will be created in this exercise module:

User group	User privilege
administrators	Has full access on all contracts created within the application to allow administering them.
line_managers	Can approve contracts as line manager.
risk_managers	Can approve contracts as risk manager.
contract_approval_users	Regular user of the Contract Approval application. Will not have administrator access or be allowed to approve contracts.

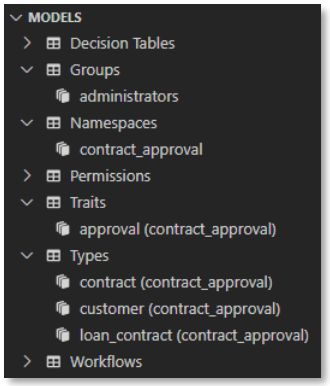
8.1 Create the administrators group

1. In VS Code, create a new group model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `administrators` for the (file) name of the group.
3. Fill the administrator group properties using the following details:

Field	Value
Group name	The technical name for the group. This name must be unique within the context of the tenant. It is automatically populated based on the previously chosen group name (the model file name). Leave the value to be `administrators`.
Description	`Group containing the administrators for the Contract Approval application`
Groups	The administrators group has no (sub) groups.



4. Save and close the administrators group model. The model explorer displays the new **administrators** group under **Groups**.



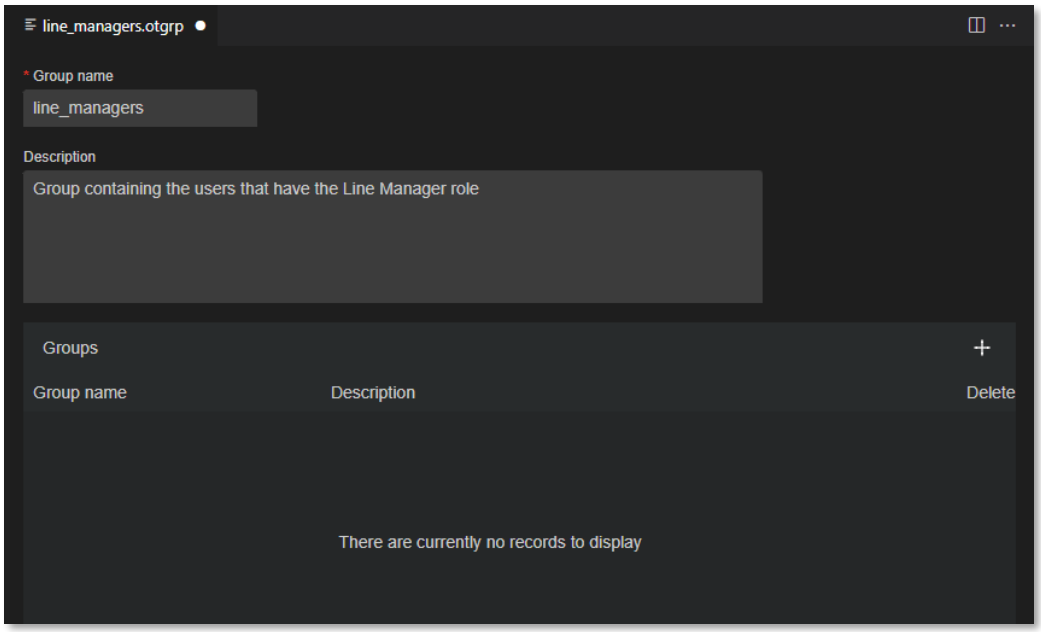
Next step:

Create the line_managers group.

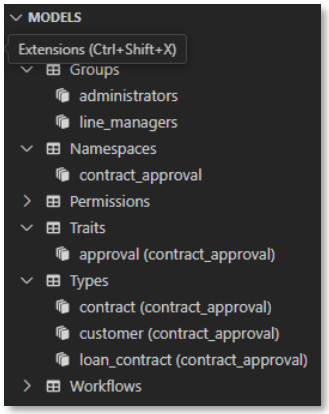
8.2 Create the line_managers group

1. In VS Code, create a new group model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `line_managers` for the (file) name of the group.
3. Fill the line_managers group properties using the following details:

Field	Value
Group name	`line_managers`
Description	`Group containing the users that have the Line Manager role`
Groups	The line_managers group has no (sub) groups.



4. Save and close the line_managers group model. The model explorer displays the new **line_managers** group under **Groups**.



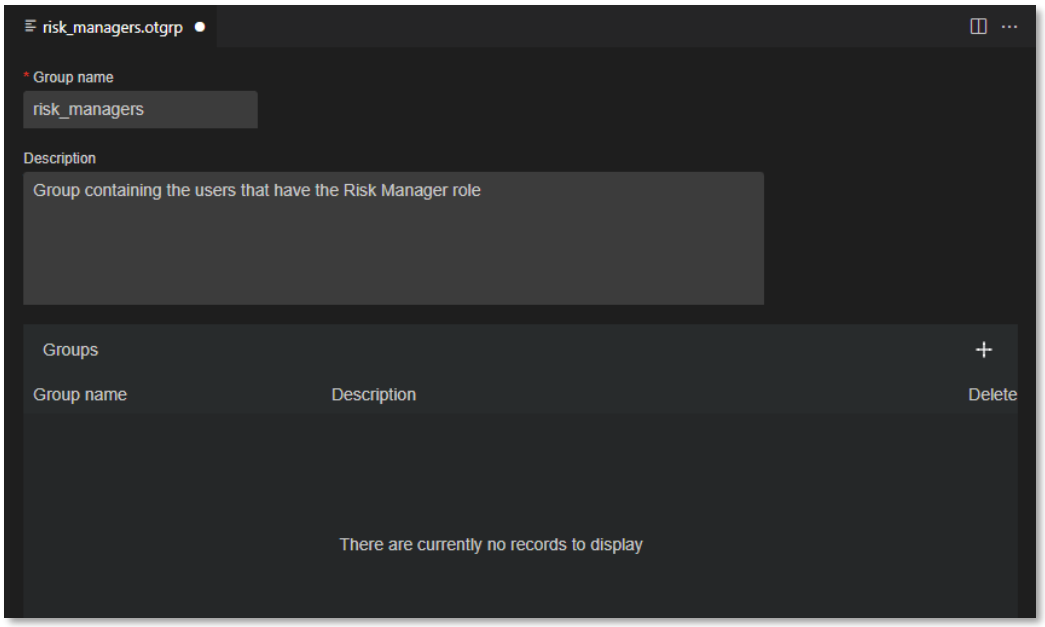
Next step:

Create the risk_managers group.

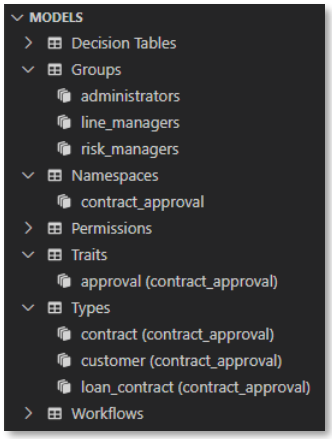
8.3 Create the risk_managers group

1. In VS Code, create a new group model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `risk_managers` for the (file) name of the group.
3. Fill the risk_managers group properties using the following details:

Field	Value
Group name	`risk_managers`
Description	`Group containing the users that have the Risk Manager role`
Groups	The risk_managers group has no (sub) groups.



4. Save and close the risk_managers group model. The model explorer displays the new **risk_managers** group under **Groups**.



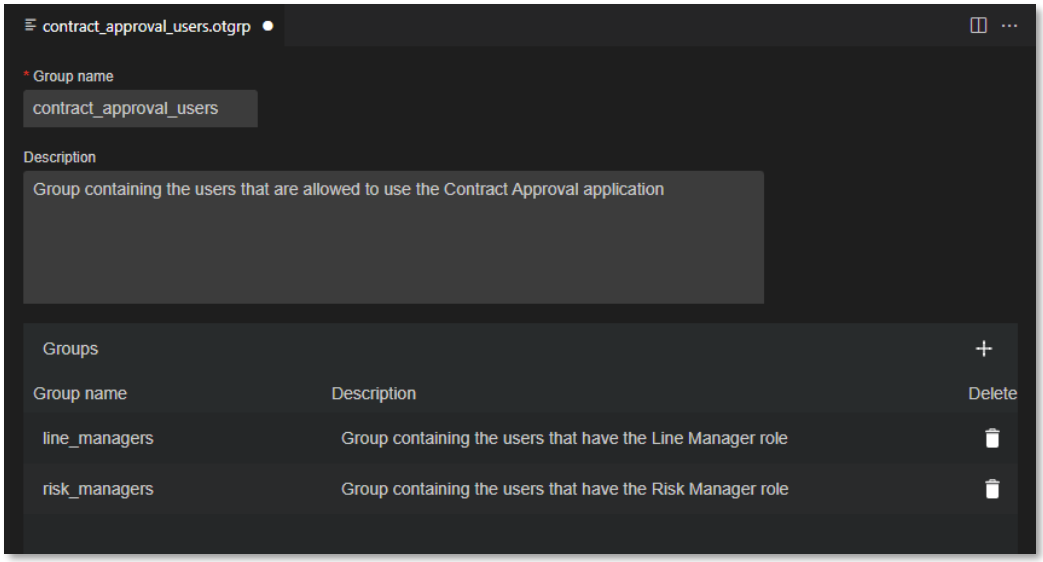
Next step:

Create the contract_approval_users group.

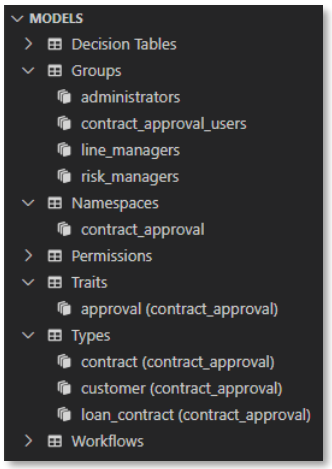
8.4 Create the contract_approval_users group

1. In VS Code, create a new group model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `contract_approval_users` for the (file) name of the group.
3. Fill the contract_approval_users group properties using the following details:

Field	Value
Group name	`contract_approval_users`
Description	`Group containing the users that are allowed to use the Contract Approval application`
Groups	Select both the **line_managers** and **risk_managers** groups using the ➕ in the **Groups** list.



4. Save and close the contract_approval_users group model. The model explorer displays the new **contract_approval_users** group under **Groups**.



Next exercise module:

Create ACLs.

9 [15’] Create ACLs

Learn how to:

- Create Access Control List (ACL) models that correspond with the different permissions for your Contract Approval application
- Assign custom permissions

The following ACLs will be created in this exercise module:

ACL	Description
created	Applied to newly created contracts, granting the following permissions: • Full control for the owner or creator of the new contract • Read access for the regular application users • Full control for the administrators
pending_approval	Applied to contracts that are waiting to be approved by a line manager, granting the following permissions: • Custom read access allowing to change the status and security of the contract (that is, approve it) for the owner of the contract, removing full control • Read access for the regular application users • Custom read access allowing to change the status and security of the contract (that is, approve it) for the line managers • Custom read access allowing to change the status and security of the contract (that is, approve it) for the risk managers • Full control for the administrators
completed	Applied to completed (approved, rejected, or expired) contracts, granting the following permissions: • Read access for the owner, removing custom approval permissions • Read access for the regular application users • Full control for the administrators

9.1 Create the created ACL

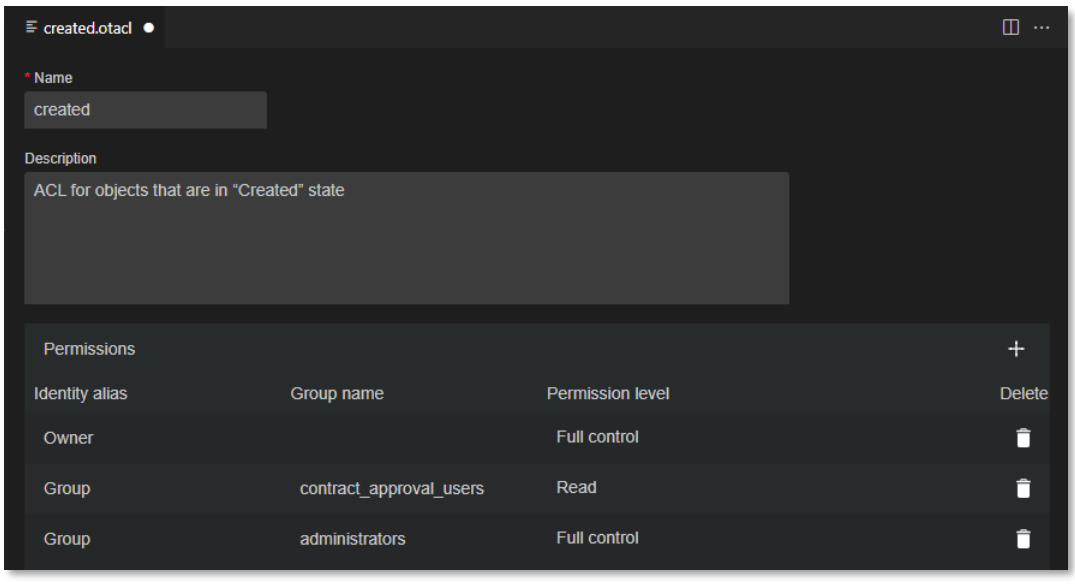
1. In VS Code, create a new ACL model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `created` for the (file) name of the ACL.
3. Fill the created ACL properties using the following details:

Field	Value
Name	`created`
Description	`ACL for objects that are in "Created" state`
Permissions	The permissions list defines the different permissions for each of the accessors. When creating an ACL, this list already contains one entry granting the owner full control. You will add the permissions for the created ACL in the next step.

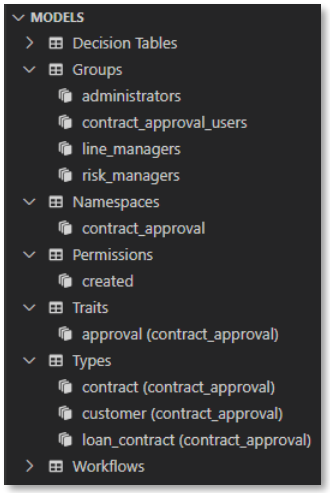
4. In the **Permissions** list, click ➕ to add the different permissions.

   The following table describes each permission and the property values to assign:

Identity alias	Group name	Permission level
Owner		**Full control**
Group	**contract_approval_users**	**Read**
Group	**administrators**	**Full control**



5. Save and close the created ACL model.
6. The model explorer displays the new **created** ACL under **Permissions**.



Next step:

Create the pending_approval ACL.

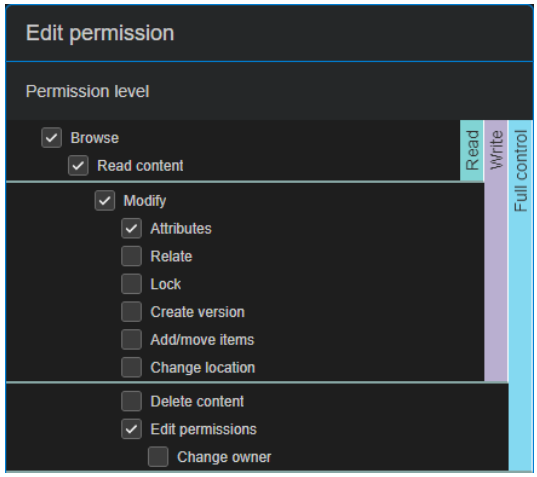
9.2 Create the pending_approval ACL

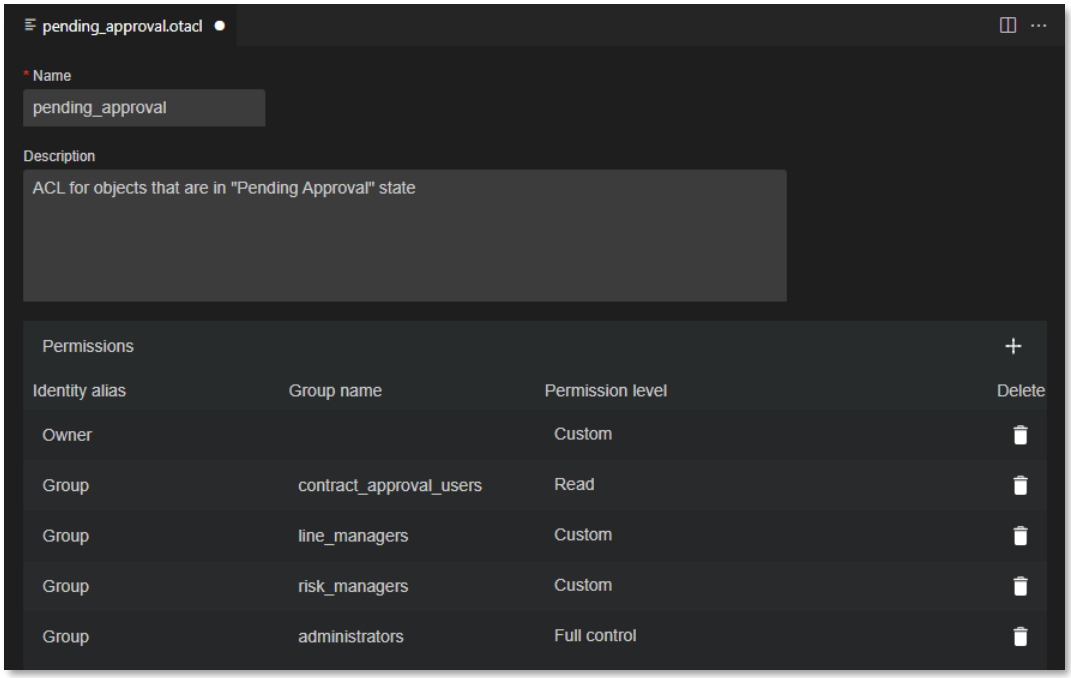
1. In VS Code, create a new ACL model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `pending_approval` for the (file) name of the ACL.
3. Fill the pending_approval ACL properties using the following details:

Field	Value
Name	`pending_approval`
Description	`ACL for objects that are in "Pending Approval" state`
Permissions	You will add the permissions for the pending_approval ACL in the next step.

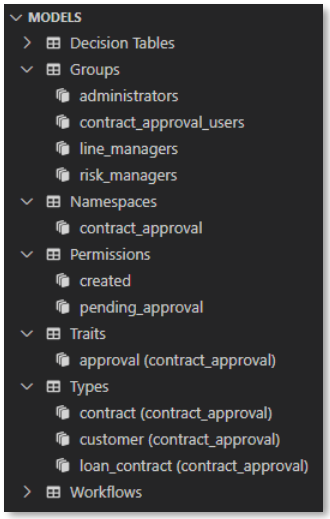
4. In the **Permissions** list, click ✚ to add the different permissions.

   The following table describes each permission and the property values to assign:

Identity alias	Group name	Permission level
Owner		**Custom: Browse, Read content, Modify, Attributes, Edit permissions** 
Group	**contract_approval_users**	**Read**
Group	**line_managers**	**Custom: Browse, Read content, Modify, Attributes, Edit permissions**
Group	**risk_managers**	**Custom: Browse, Read content, Modify, Attributes, Edit permissions**
Group	**administrators**	**Full control**



5. Save and close the pending_approval ACL model. The model explorer displays the new **pending_approval** ACL under **Permissions**.



Next step:

Create the completed ACL.

9.3 Create the completed ACL

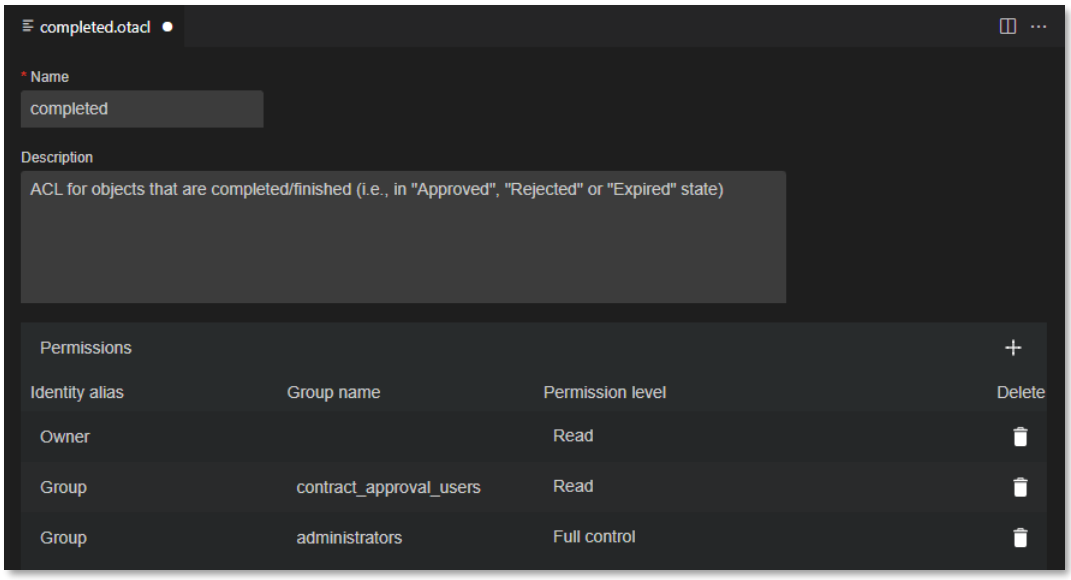
1. In VS Code, create a new ACL model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `completed` for the (file) name of the ACL.
3. Fill the completed ACL properties using the following details:

Field	Value
Name	`completed`
Description	`ACL for objects that are completed/finished (i.e., in "Approved", "Rejected" or "Expired" state)`
Permissions	You will add the permissions for the completed ACL in the next step.

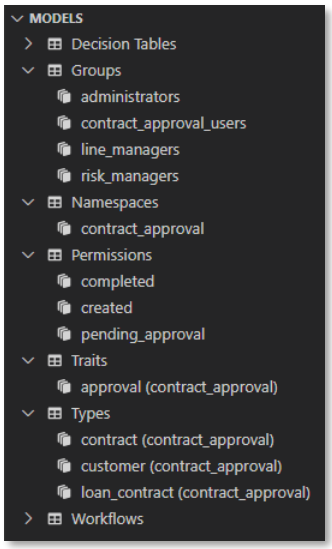
4. In the **Permissions** list, click the **Add** button ⊞ to add the different permissions.

   The following table describes each permission and the property values to assign:

Identity alias	Group name	Permission level
Owner		**Read**
Group	**contract_approval_users**	**Read**
Group	**administrators**	**Full control**



5. Save and close the completed ACL model. The model explorer displays the new **completed** ACL under **Permissions**.



Next exercise module:

Create a decision table.

10 [25’] Create a decision table

Learn how to:

- Create a decision table model
- Define input and output variables
- Determine the right hit policy for the output variables

A decision table allows configuring business rules to be used when building the application logic. These configured business rules can be consumed directly from the application code or from a workflow.

The following decision table will be created in this exercise module:

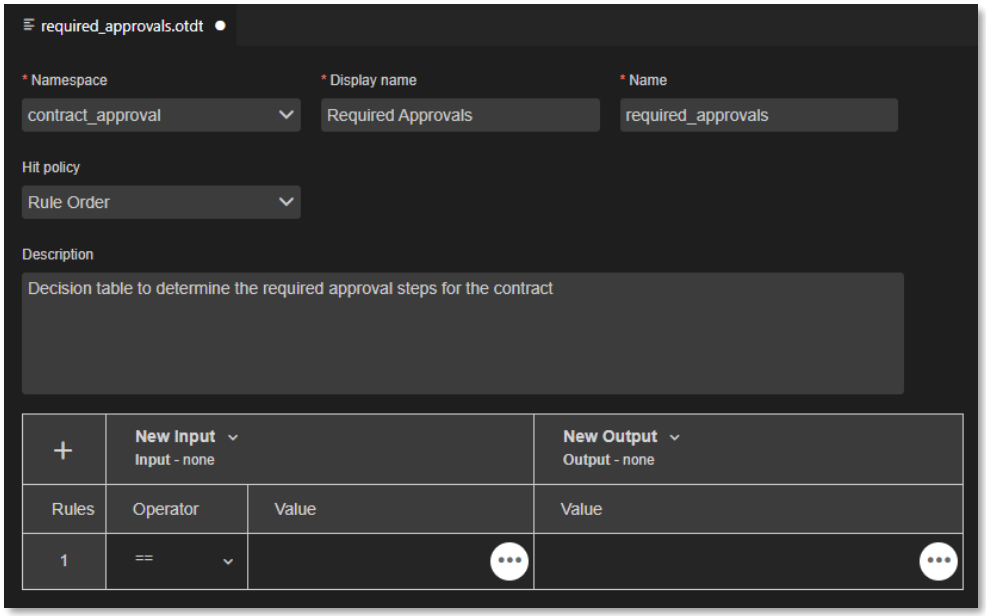
Decision table	Description
Contract Approvals	Defines the business logic to determine the necessary approvals for a given contract. Depending on the following criteria, the decision table decides the required approvals for the contract: • Loan contract: An automated solvency checks to ensure the loan applicant can pay back the loan according to the agreed conditions. • Standard contract (that is, not a loan contract) and its value is over 1000: A manual approval from line manager. • Loan contract and its value is over 5000: A manual approval from the line manager. The loan contract value requiring line manager approval is higher because there is already an automated solvency check for loan contracts. • Contract risk level is higher than 3 (medium risk): A manual approval from risk manager.

For more information on Decision Service decision tables, see the Decision Service Reference Guide, the Decision Table Modeler Reference Guide, or the Decision Service API Reference.

10.1Create the Required Approvals decision table

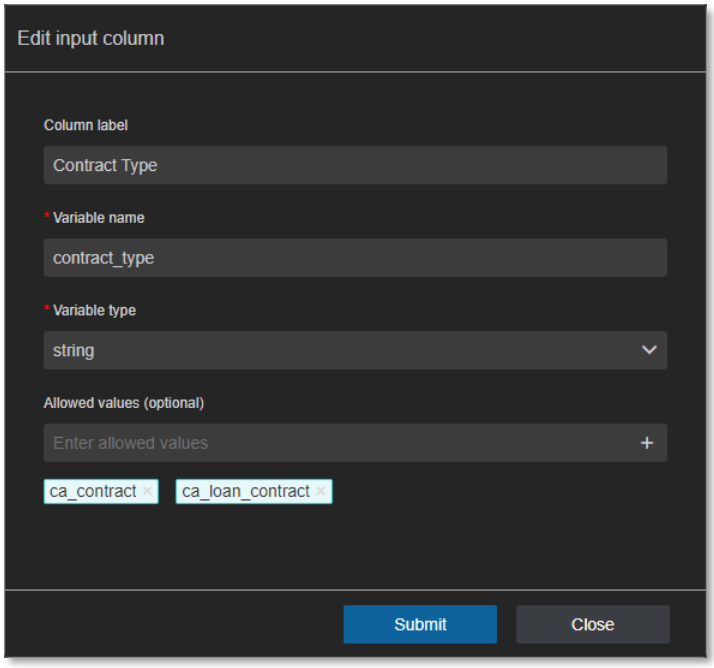
1. In VS Code, create a new decision table model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `required_approvals` for the (file) name of the decision table.
3. Fill the created decision table properties using the following details:

Field	Value
Namespace	Select the **contract_approval** namespace.
Display name	`Required Approvals`
Name	`required_approvals`
Hit policy	The hit policy provides instructions on how the business rules in the decision table must be executed, evaluated, and matched. Following are the options in the **Hit policy** list: Single hit policies • First: Multiple (overlapping) rules can match, with different output entries. The first hit by rule order is returned (and evaluation can halt). • Any: There can be an overlap, but all the matching rules show equal output entries for each output, so any match can be used. If the output entries are not equal, the hit policy is incorrect, and the result is undefined. • Unique: No overlap is possible, and all rules are disjoint. Only a single rule can be matched. • Priority: Multiple rules can match, with different output entries. This policy returns the matching rule with the highest output priority. Output priorities are specified in the ordered list of allowed output values, in decreasing order of priority. Priorities are independent from the rule order. Multiple hit policies • Rule Order: Returns all hits in rule order. • Output Order: Returns all hits in decreasing output priority order. Output priorities are specified in the ordered list of allowed output values in decreasing order of priority. Priorities are independent from the rule order. • Collect: Returns all hits in arbitrary order. • Collect (Sum): Returns the sum of the values of all hits. • Collect (Min): Returns the smallest value of all hits. • Collect (Max): Returns the largest value of all hits. • Collect (Count): Returns the number of hits. By default, the “First” hit policy is selected. Select the **Rule Order** hit policy to instruct the Decision Service to collect and return the output (required approval) of all business rules it hits in the same order as defined in the decision table.
Description	`Decision table to determine the required approval steps for the contract`

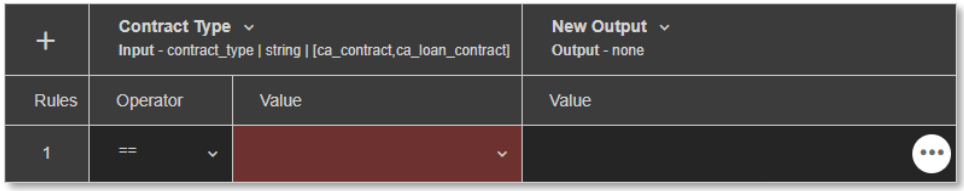


4. To configure the New input column, in the decision table, from the **New Input** column, click the **Edit Action** button and select **Edit**.
5. Fill the **Edit input column** dialog box using the following details:

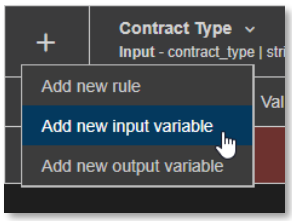
Field	Value
Column label	The column label is the label that is shown at the top of the input column in the decision table. Enter `Contract Type` for the column label.
Variable name	The variable name is the name of the input variable. Enter `contract_type` for the variable name.
Variable type	The variable type is the data type of the input variable. The data type can be string, number, boolean, date, dateTime, json, duration, or collection. Select the **string** variable type.
Allowed values (optional)	The allowed values is the optional ordered list (from highest to lowest priority) of the different possible/allowed values for the input variable. Add the following two allowed values by typing them in and clicking : • `ca_contract` • `ca_loan_contract`



6. Click **Submit** to confirm the input column properties.

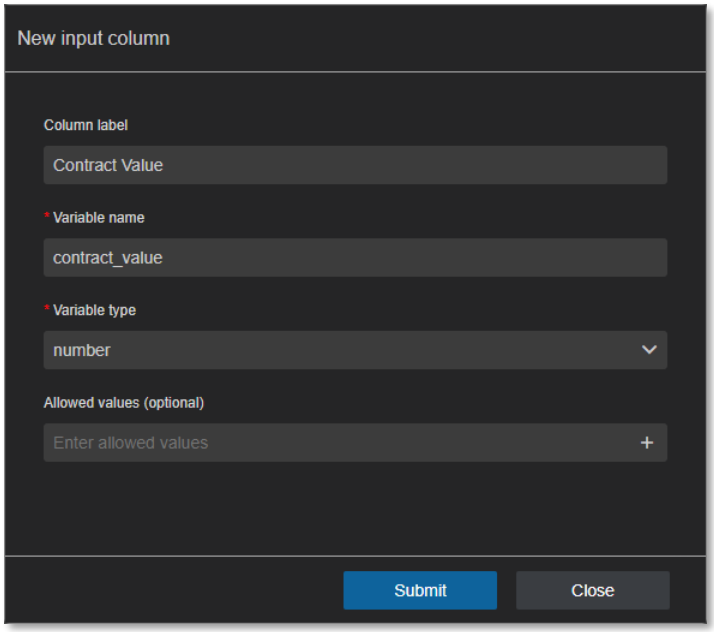


7. To create the second input variable, in the decision table, from the first column, click the **Add Action** button and select **Add new input variable**.

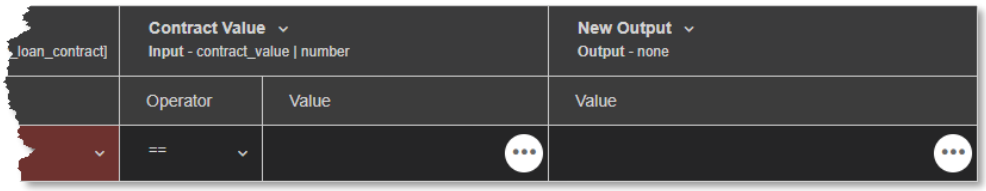


8. Fill the New input column dialog box using the following details:

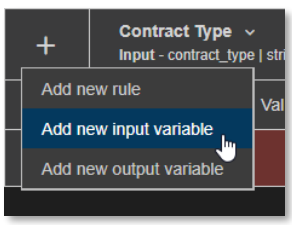
Field	Value
Column label	`Contract Value`
Variable name	`contract_value`
Variable type	Select the **number** variable type.
Allowed values (optional)	Allowed values list has no entries.



9. Click **Submit** to confirm the input column properties.

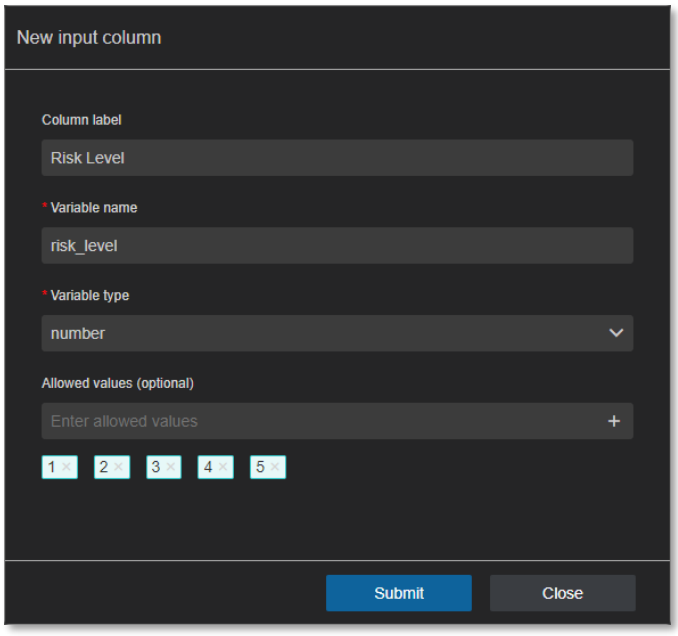


10. To create the third input variable, from the decision table first column, click the **Add Action** button and select **Add new input variable**.

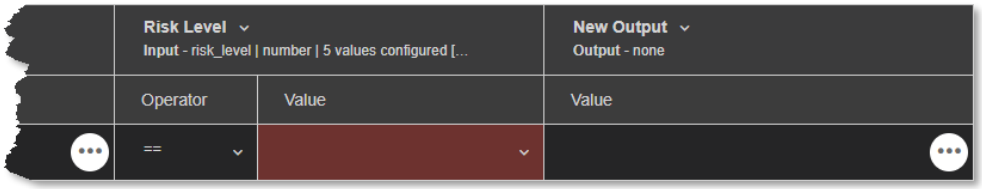


11. In the **New input column** dialog box, fill the input column properties for the new input variable using the following details:

Field	Value
Column label	`Risk Level`
Variable name	`risk_level`
Variable type	Select the **number** variable type.
Allowed values (optional)	• `1` • `2` • `3` • `4` • `5`

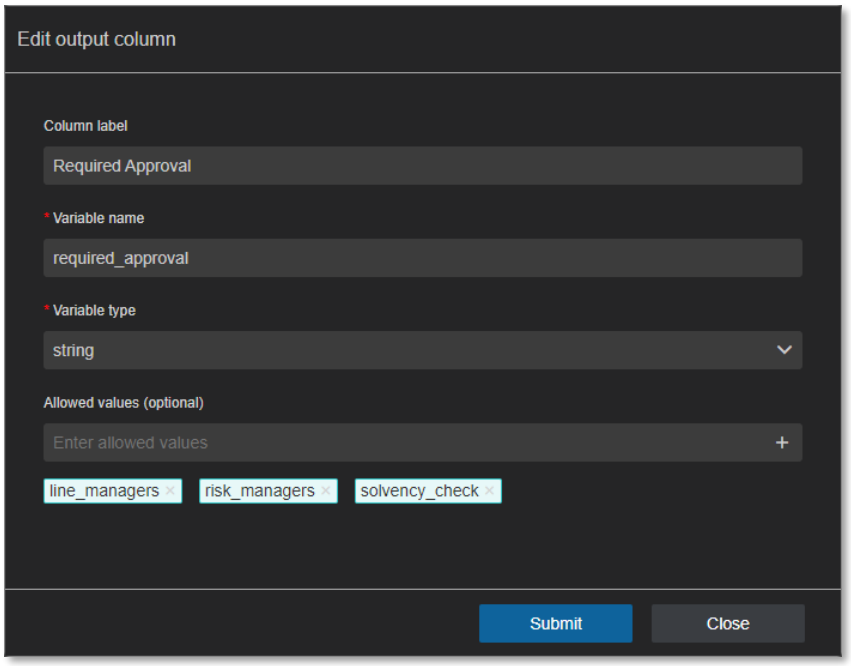


12. Click **Submit** to confirm the input column properties.

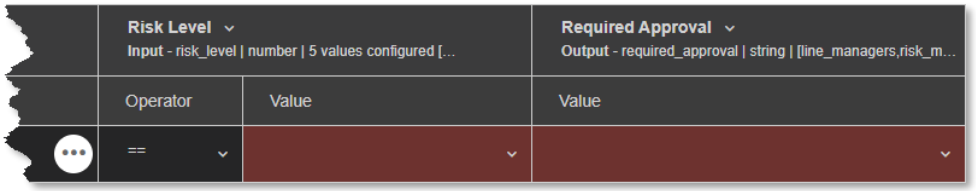


13. In the decision table, scroll and locate the **New Output** column and click the **Edit Action** button and select **Edit** to configure the output column.
14. Fill the **Edit output column** dialog box properties using the following details:

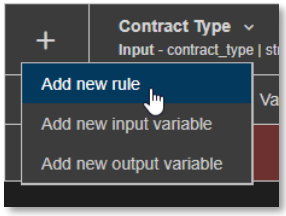
Field	Value
Column label	`Required Approval`
Variable name	`required_approval`
Variable type	Select the **string** variable type.
Allowed values (optional)	• `line_managers` • `risk_managers` • `solvency_check`



15. Click **Submit** to confirm the output column properties.



16. To add three more rules to the decision table, from the first column, click the **Add Action** button and select **Add new rule**.

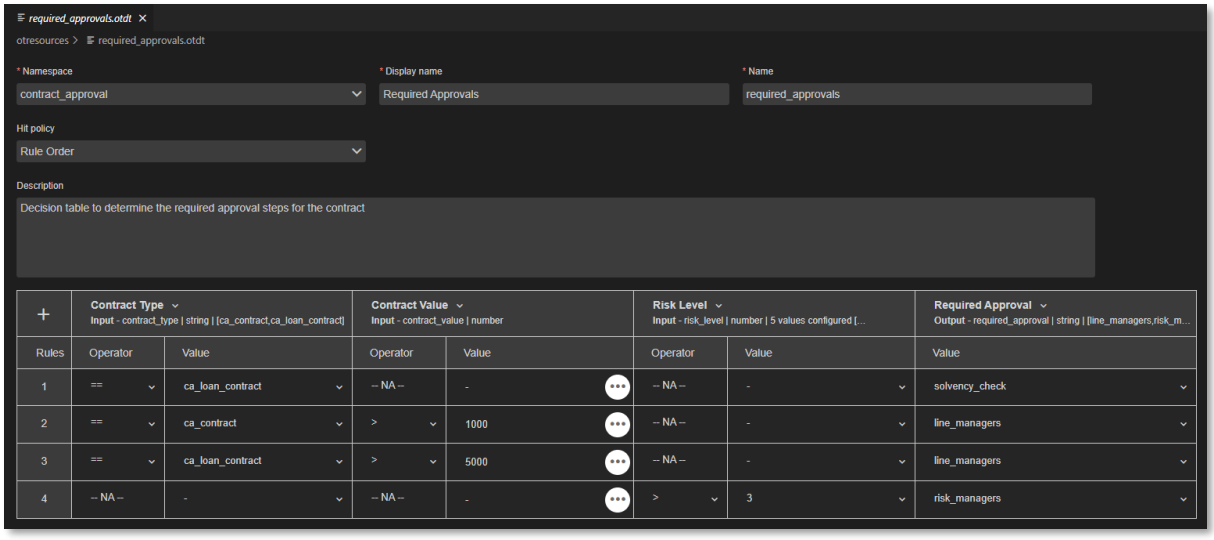


17. Fill all four rules according to the following table:

Note

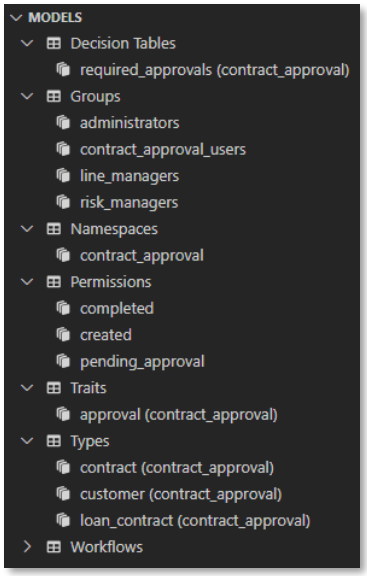
Selecting a **value** of "-" will set the corresponding **operator** to "--NA--" as there is no value to perform a comparison operation on.

	Contract Type		Contract value		Risk Level		Required Approval
Rules	Operator	Value	Operator	Value	Operator	Value	Value
1	==	**ca_loan_contract**	-- NA --	-	-- NA --	-	**solvency_check**
2	==	**ca_contract**	>	1000	-- NA --	-	**line_managers**
3	==	**ca_loan_contract**	>	5000	-- NA --	-	**line_managers**
4	-- NA --	-	-- NA --	-	>	3	**risk_managers**



18. Save and close the Required Approvals decision table model.

19. The model explorer displays the new **required_approvals (contract_approval)** decision table under **Decision Tables**. The model explorer shows the different models according to their unique key, which in context of a decision table is the combination of the namespace and name properties.



Next exercise module:

Create workflows.

11 [140’] Create workflows

Learn how to:

- Create a workflow model
- Perform different types of REST API calls in a workflow
- Use a decision table in a workflow
- Use another workflow in a workflow
- Use JavaScript code in a workflow
- Define a user/manual task in a workflow
- Send an email from a workflow
- Define workflow execution paths (decide which paths to activate/follow, parallel paths, joining execution paths)

A workflow model represents an executable (business) process model from which process instances can be created. The executable process model is stored as BPMN 2.0 encoded JSON.

The following workflows will be created in this exercise module:

Workflow	Description
Solvency Check	The Solvency Check workflow performs the automated solvency check for loan contracts. It determines whether the requester has enough cash flow to pay back the loan contract based on the yearly income, monthly payments, and contract value.
Manager Approval	The Manager Approval workflow orchestrates the manual approval of a contract by a user that belongs to a specific manager group. In the context of the Contract Approval application this is either the line managers group or the risk managers group.
Contract Approval	The Contract Approval workflow represents the overall business process of approving a contract and is the main workflow for the Contract Approval application. As such, it calls the decision table and the other workflows. It consists of automated and manual approval tasks and not all approval tasks are always required. They are conditional, based on the rules defined in the decision table. At the end an email is sent to inform the requester about the outcome of the approval process.

For more information on Workflow Service process models and process instances, see the Workflow Service product documentation, the Workflow Modeler product documentation, or the Workflow Service API reference.

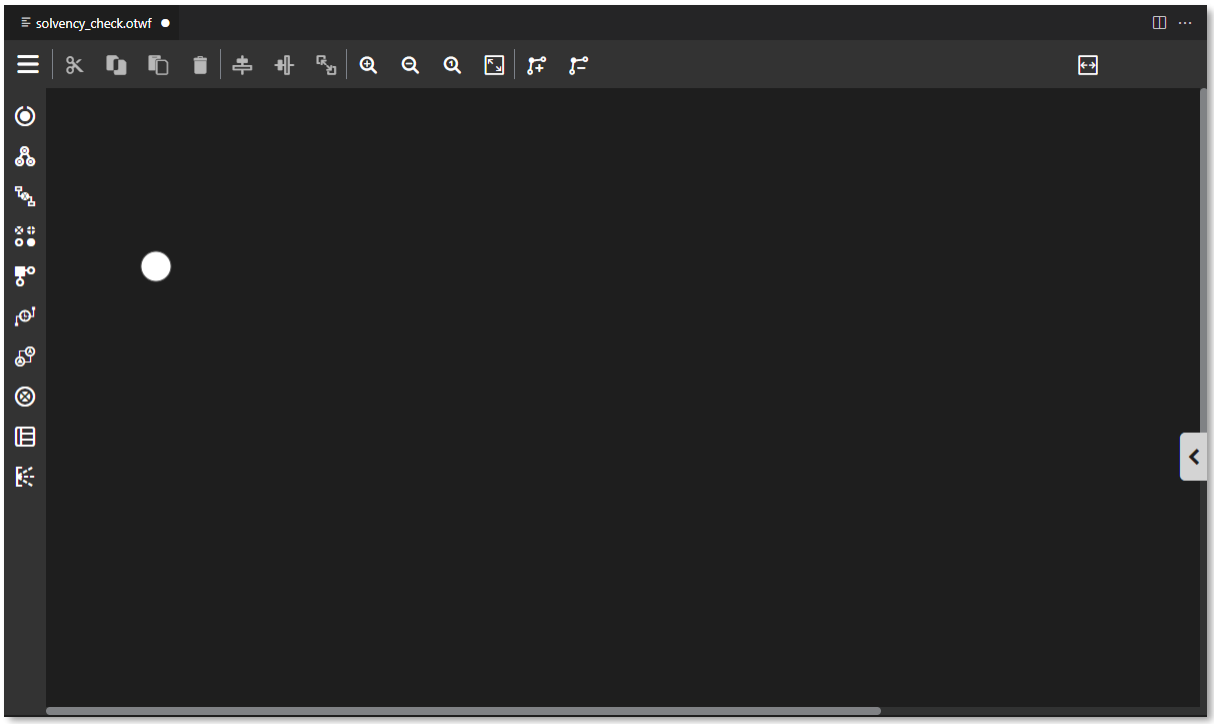
11.1Create the Solvency Check workflow

For convenience, the creation of the Solvency Check workflow is split into several sub chapters:

- Create the workflow model
- Understand the workflow editor user interface
- Set the workflow attributes
- Build the process definition

11.1.1 Create the workflow model

1. In VS Code, create a new workflow model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `solvency_check` for the (file) name of the workflow.

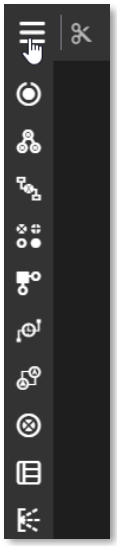


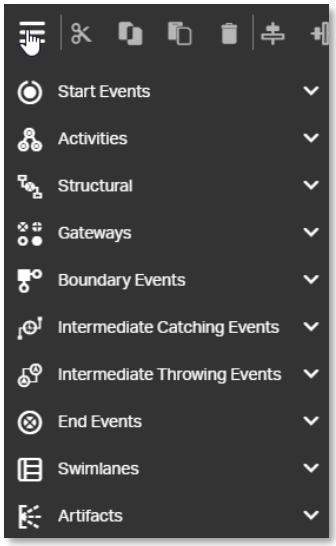
11.1.2 Understand the workflow editor user interface

1. In the workflow model editor, click the **Menu** button to expand the palette.

Note

The workflow model editor is the most elaborate editor of the Cloud Developer Tools extension pack. It is intended to build entire business process definitions in all its complexity.

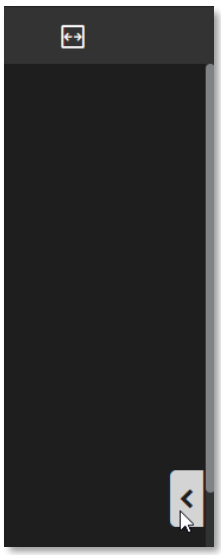


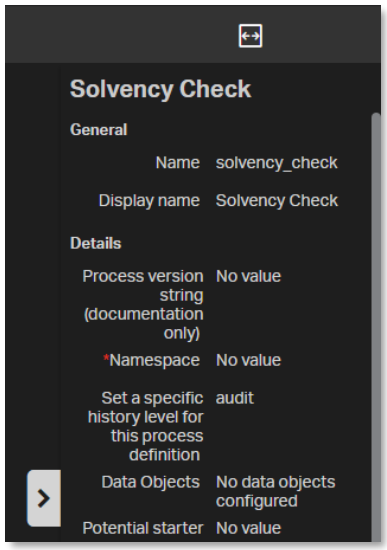


2. Click the in the workflow model editor to expand the attribute bar.

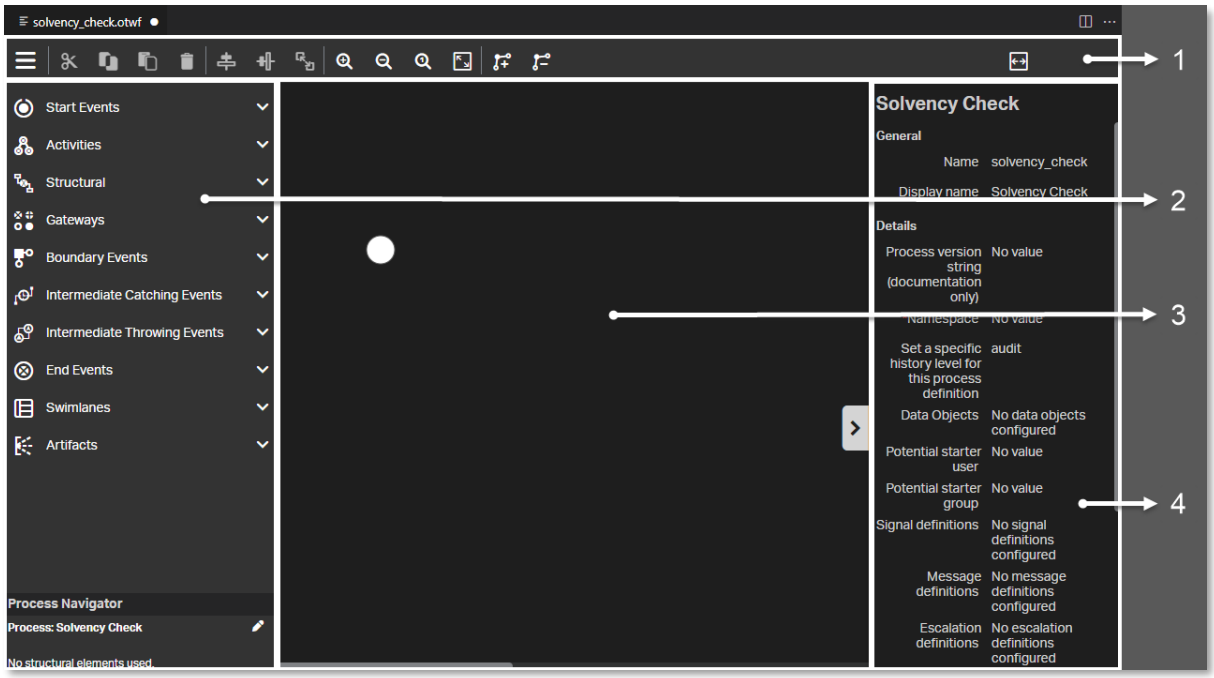
Tip

Use the **Toggle side panels** button to expand or collapse both side panes.





The following image illustrates the different user interface components of the workflow editor:



The following table explains the workflow editor user interface components from the preceding illustration:

Number	Component	Description
1	Menu bar	Contains buttons for the generic capabilities like copy, paste, delete, align, zoom, help, and other buttons.
2	Palette	Contains the different workflow elements which you can drag and drop on the canvas to build the workflow model.
3	Canvas	Area to build or draw the workflow model.
4	Attribute bar	Displays the attributes of the currently selected element. For example, in the preceding illustration, the canvas is selected and the attributes for the canvas are displayed.

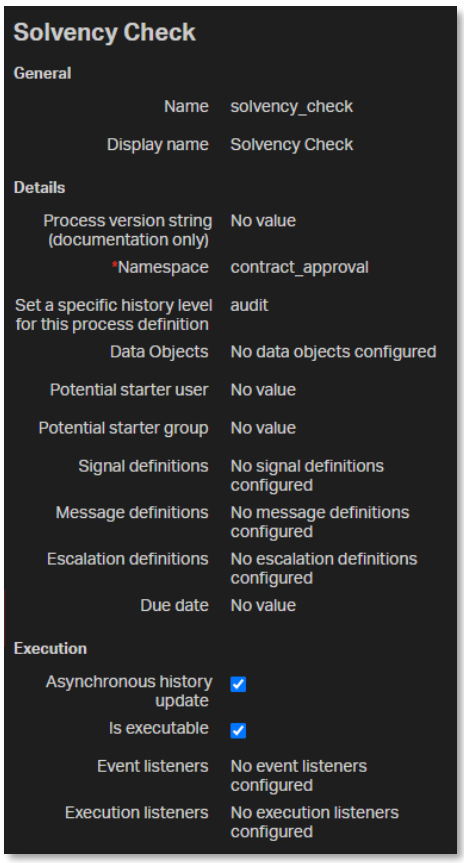
11.1.3 Set the workflow attributes

1. Click the empty space in the **canvas** (that is, do not select the start event represented by ).
2. Expand the attribute bar and fill the workflow attributes using the following details:

Attribute	Value
Name	The technical name for the workflow. This name must be unique within the context of the tenant and selected namespace. It is automatically populated based on the previously chosen workflow name (the model file name). Leave the value to be `solvency_check`.
Display name	The display name is the user-friendly name for the workflow. It is automatically populated based on the previously chosen workflow name (the model file name). Leave the value to be `Solvency Check`.
Namespace	The namespace to which this workflow belongs. Select the **contract_approval** namespace as the namespace.

Note

When building workflows, only those attributes that are relevant in context of the tutorial are explained. For more information on the different attributes of the different workflow elements, see Workflow Modeler product documentation.



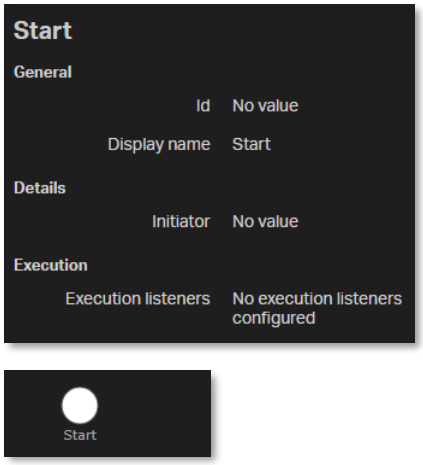
11.1.4 Build the process definition

1. Select the **Start Event**  that is already on the **canvas**.

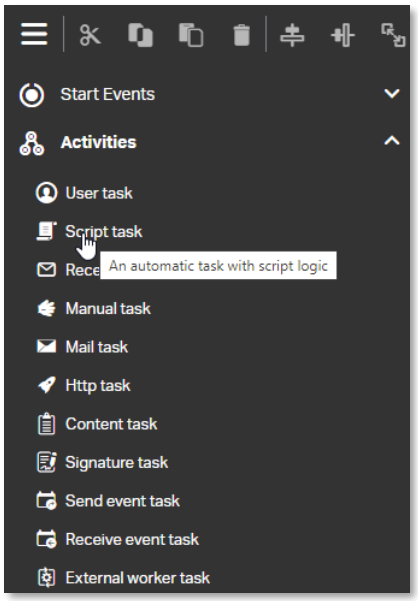
> **Note**
>
> A start event indicates where a process begins. The type of start event defines how the process starts. The (standard) **start event** is used when a process instance is started through an API call (that is, there is no specific trigger).

2. Fill the start event attributes using the following details:

Attribute	Value
Display name	Enter `Start`



3. In the **palette**, find **Activities > Script task**.



Note

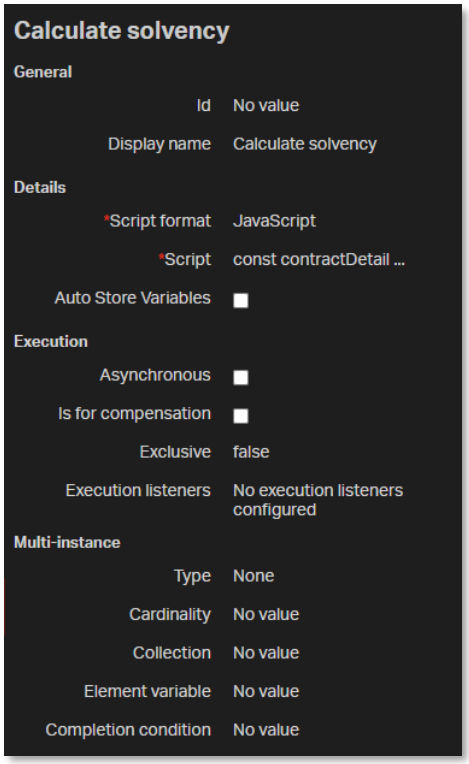
The script task is used to execute JavaScript scripts. Script tasks are mainly used to perform simple calculations or operations.

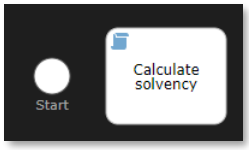
4. Drag and drop the **Script task** to the **canvas** next to the **Start event**.

5. Fill the Script task attributes using the following details:

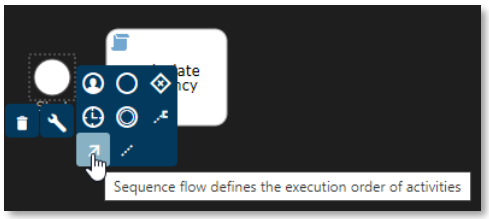
Attribute	Value
Display name	`Calculate solvency`
Script format	The format in which the script must be provided. Enter the following script format: `JavaScript`

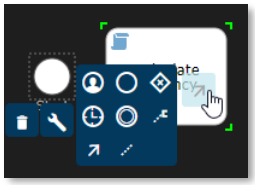
Script	Script that executes when the task executes.
	This task sets the **solvent** boolean (process) execution variable based on whether the **monthlyBudget** is larger or equal to the **monthlyPayments**.
	Enter the following script:
	`const contractDetails = JSON.parse(execution.getVariable("contract"));`
	`const monthlyPayments = contractDetails.properties.value / contractDetails.properties.monthly_installments;`
	`const monthlyBudget = contractDetails.properties.yearly_income / 12 / 4;`
	`execution.setVariable("solvent", monthlyBudget >= monthlyPayments);`

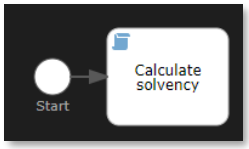




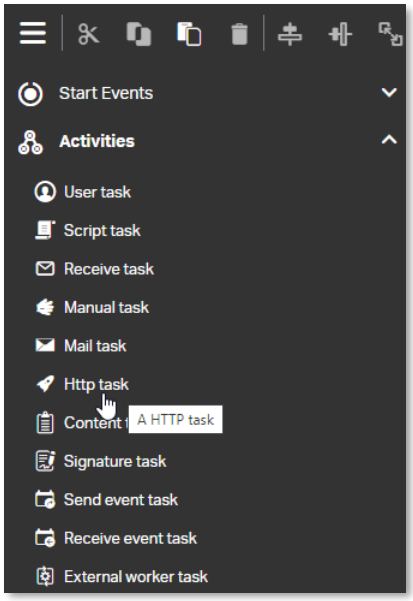
6. Select the **Start event** and drag the **sequence flow** (arrow connector) to the **Calculate solvency script task**.







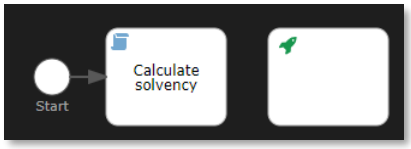
7. In the palette, find Activities > Http task.



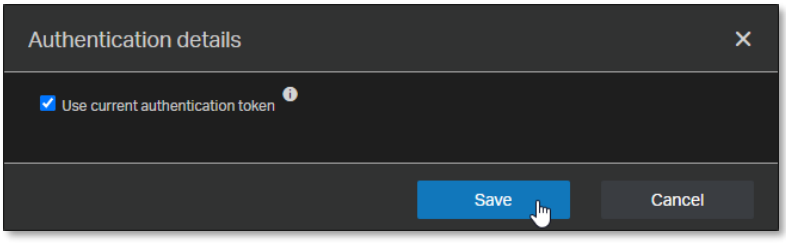
Note

The HTTP task allows to submit and store the result of an HTTP call.

8. Drag and drop the **Http task** from the **palette** to the **canvas** next to the **Calculate solvency script task**.



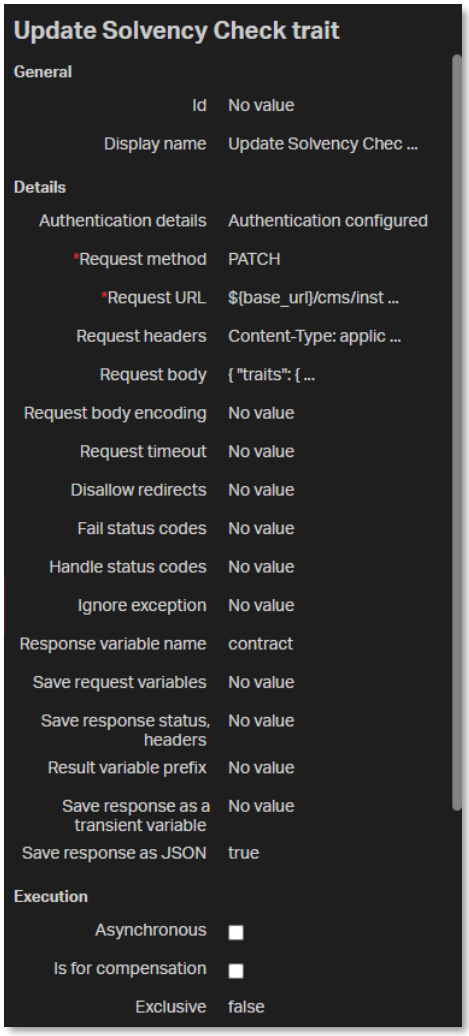
9. Fill the Http task attributes using the following details:

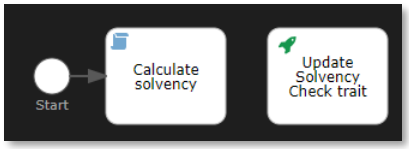
Attribute	Value
Display name	`Update Solvency Check trait`
Authentication details	How to authenticate with the REST API. For the Http tasks in this workflow, you will use the authentication credentials of the user performing the task. Select **Use current authentication token** and click **Save** to confirm. 

Attribute	Value
Request method	The request method to use in the HTTP call: GET, POST, PUT, DELETE or PATCH. This task updates the solvency check trait of the given contract, so this implies using the PATCH method. Select the **PATCH** request method.
Request URL	The request URL of the HTTP call. The request URL can contain expressions, such as ${some_variable}. To perform the request against the correct API base URL and to ensure updating the contract that was passed to the workflow through the **contract_id** variable, the request URL includes **${base_url}** and **${contract_id}**. Enter the following request URL: `${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	The line-separated HTTP request headers. Enter the following request headers: `Content-Type: application/json`

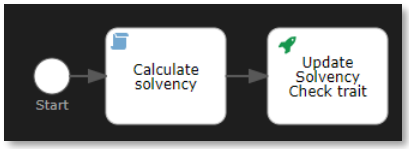
Request body	The request body to send. For example, a JSON file. Like the request URL, the request body can also contain expressions.
	This task sets the **is_required**, **has_been_granted**, **approver**, **approver_role**, and **approval_date** properties for the **Solvency Check ca_approval** trait of the contract.
	Enter the following request body:
	`{`
	`  "traits": {`
	`    "ca_approval": {`
	`      "Solvency Check": {`
	`        "is_required": true,`
	`        "has_been_granted": ${solvent},`
	`        "approver": "SYSTEM",`
	`        "approver_role": "Solvency Check",`
	`        "approval_date": "${contract.update_time}"`
	`      }`
	`    }`
	`  }`
	`}`

Response variable name	The variable name in which the HTTP response is stored. Enter the following response variable name: `contract`
Save response as JSON	Whether the response variable is stored as a JSON variable instead of a string. Enter the following value for the save response as JSON attribute: `true`

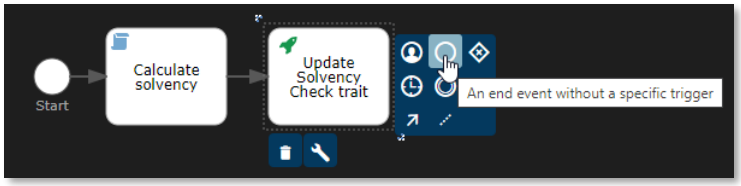


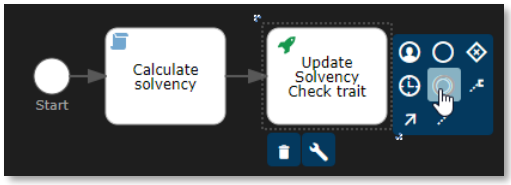


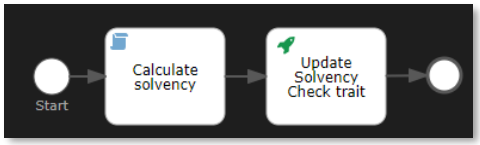
10. Select the **Calculate solvency script task** and drag a **sequence flow** to the **Update Solvency Check trait http task**.



11. Select the **Update Solvency Check trait http task** and drag and drop an **End event** next to it.





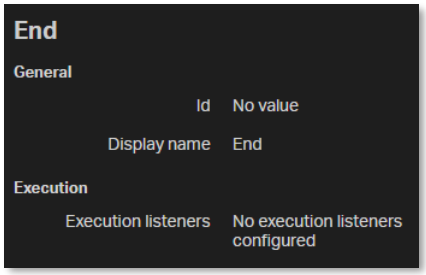


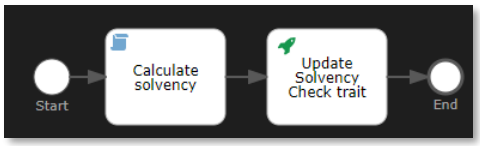
12. Select the **End event**.

> **Note**
>
> An end event signifies the end of a path in a process or sub-process. When the process execution arrives at an end event, a result is always thrown. The type of end event defines the type of result that is thrown. The (standard) **end event** means that an unspecified result is thrown upon reaching it. As such, the business process engine will not perform anything besides ending the current path of execution.

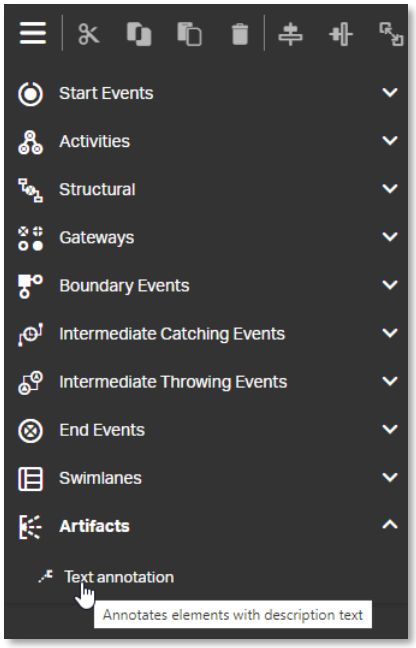
13. Fill the End event attributes using the following details:

Attribute	Value
Display name	`End`





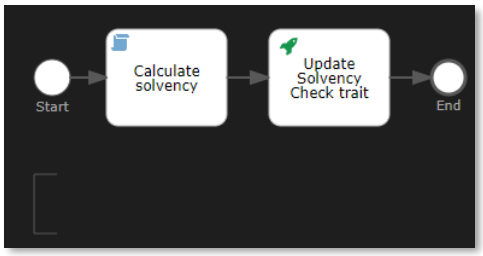
14. In the **palette**, find **Artifacts > Text annotation**.



Note

The text annotation allows you to describe the business process and flow objects in more detail. Add annotations to make your BPMN process more readable and further increase understanding of your process.

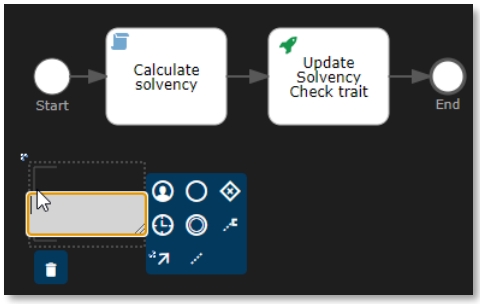
15. Drag and drop the **Text annotation** to the **canvas** under the **Start event**.

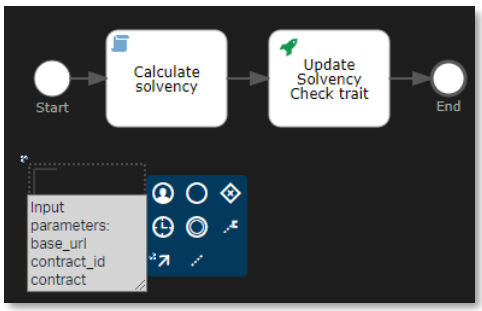


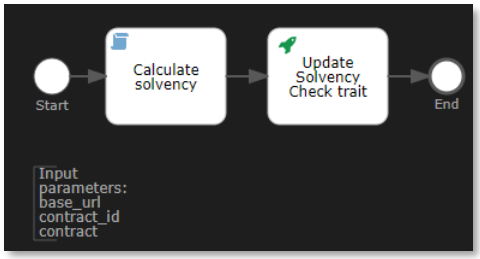
16. Double-click the **Text annotation** on the **canvas** to edit it inline.
17. Set the following text for the text annotation:

Text

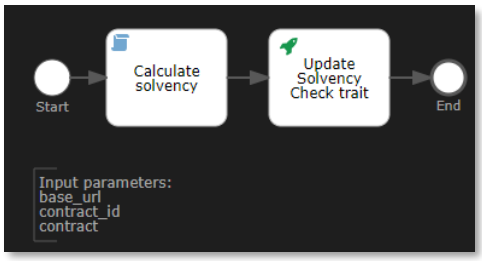
`Input parameters:`
`base_url`
`contract_id`
`contract`



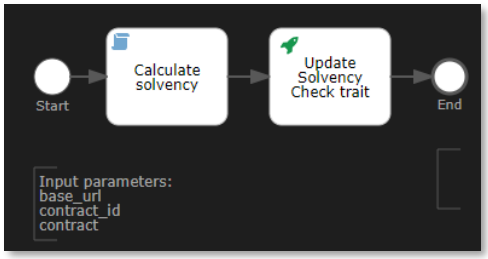




18. Resize the **text annotation** so that the text displays correctly and fits the square brackets.



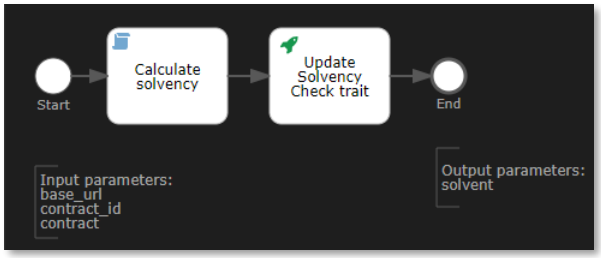
19. Drag and drop another **Text annotation** from the **palette** to the **canvas** under the **End event**.



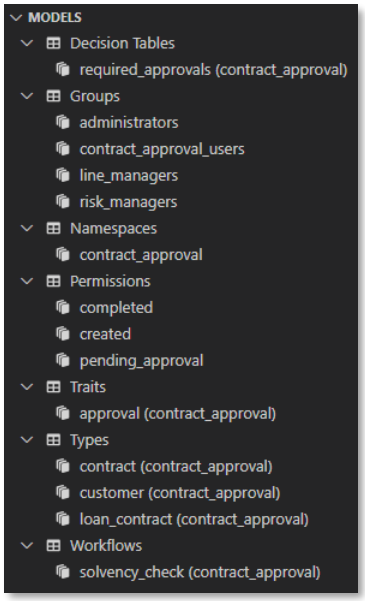
20. Double-click the **Text annotation** on the **canvas** to edit it inline and set the following text for the text annotation:

Text
`Output parameters:` `solvent`

21. Resize the **Text annotation** so that the text displays correctly and fits the square brackets.



22. Save and close the Solvency Check workflow model.
23. The model explorer displays the new **solvency_check (contract_approval)** workflow under **Workflows**. The model explorer shows the different models according to their unique key, which in context of a workflow is the combination of the namespace and name attributes.



Next step:

Create the Manager Approval workflow.

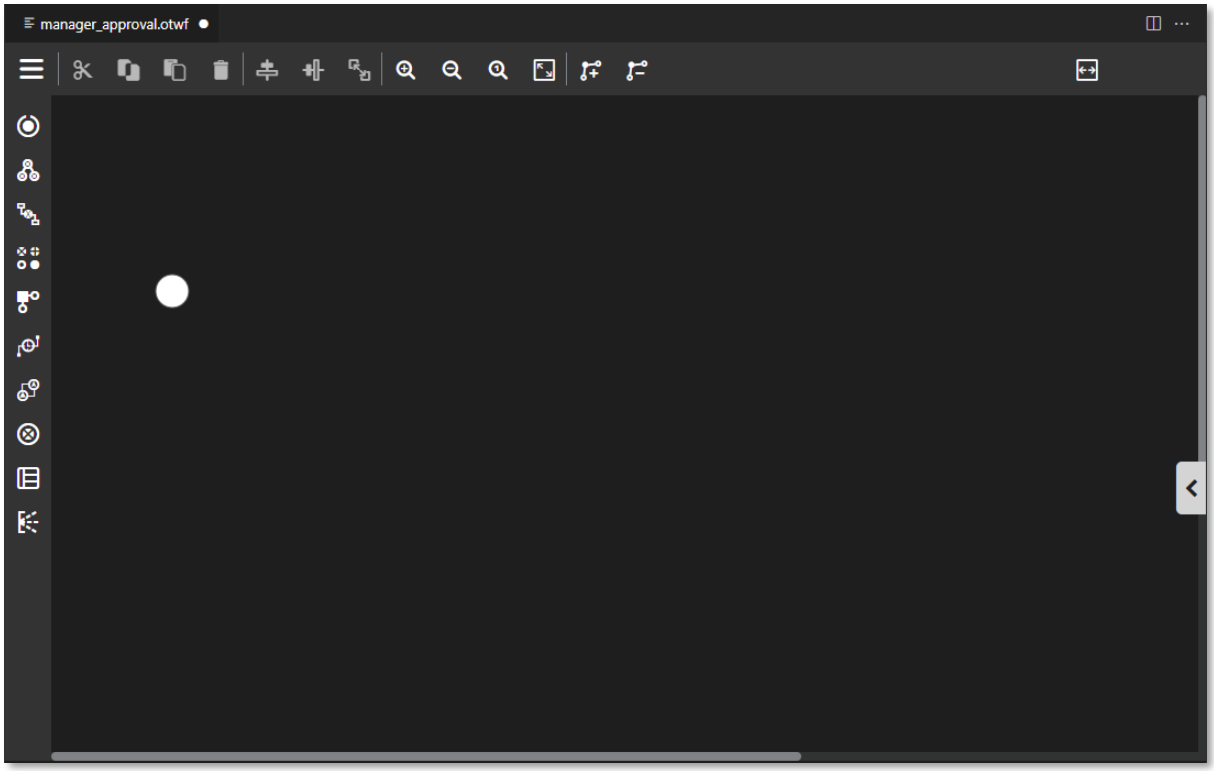
11.2Create the Manager Approval workflow

For convenience, the creation of the Manager Approval workflow is split into sub chapters:

- Create the workflow model
- Set the workflow attributes
- Build the process definition

11.2.1 Create the workflow model

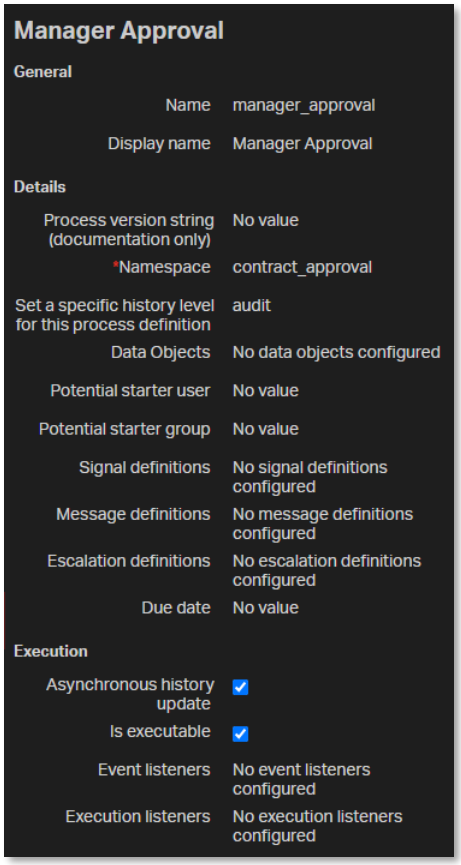
1. In VS Code, create a new workflow model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `manager_approval` for the (file) name of the workflow.



11.2.2 Set the workflow attributes

1. Click the empty space in the **canvas** (that is, do not select the start event).
2. Expand the attribute bar and fill the workflow attributes using the following details:

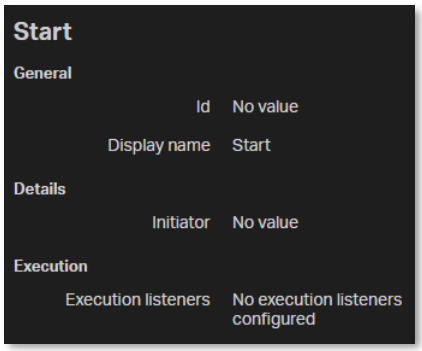
Attribute	Value
Name	`manager_approval`
Display name	`Manager Approval`
Namespace	Select the **contract_approval** namespace.

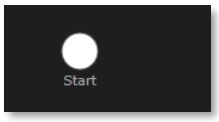


11.2.3 Build the process definition

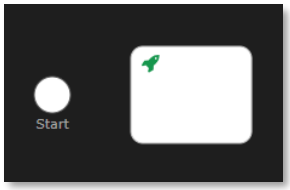
1. Select the **Start event** that is already on the **canvas**.
2. Fill the start event attributes using the following details:

Attribute	Value
Display name	`Start`





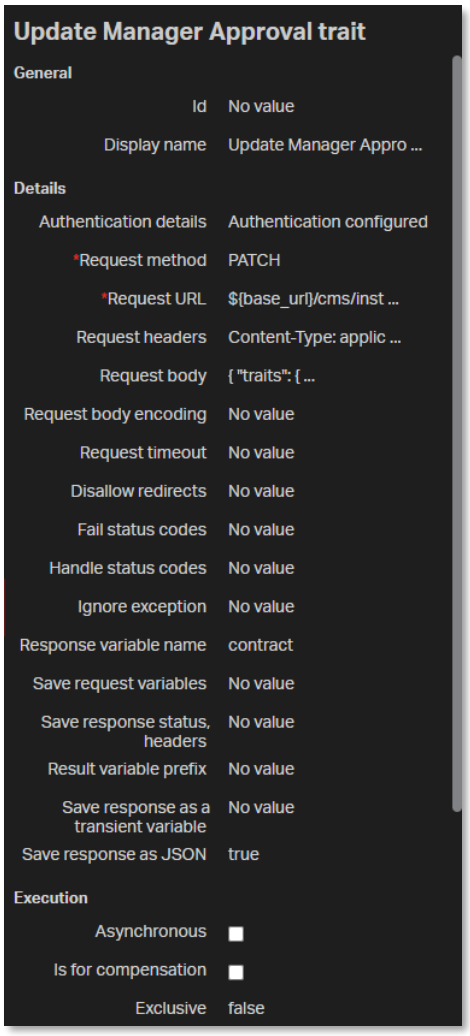
3. Drag and drop an **Http task** to the **canvas** next to the **Start event**.

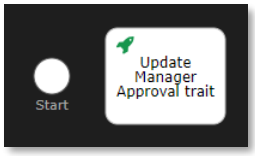


4. Fill the Http task attributes using the following details:

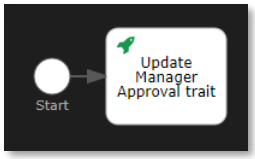
Attribute	Value
Display name	`Update Manager Approval trait`
Authentication details	Select **Use current authentication token**.
Request method	Select the **PATCH** request method.
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	`Content-Type: application/json`

Attribute	Value
Request body	This task sets the **is_required** property to **true** for the contract approval trait specified by the **approval_trait_name** process input parameter. Enter the following request body: `{ "traits": { "ca_approval": { "${approval_trait_name}": { "is_required": true } } } }`
Response variable name	`contract`
Save response as JSON	`true`

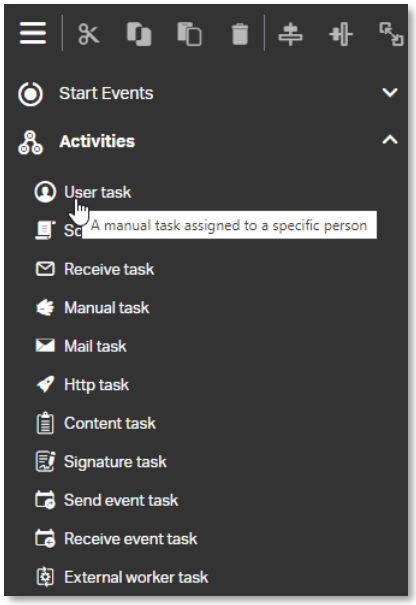




5. Select the **Start event** and drag a **sequence flow** to the **Update Manager Approval trait http task**.



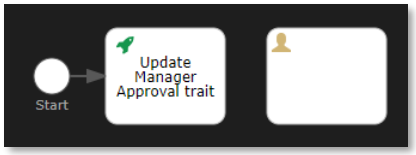
6. In the **palette**, find **Activities > User task**.



Note

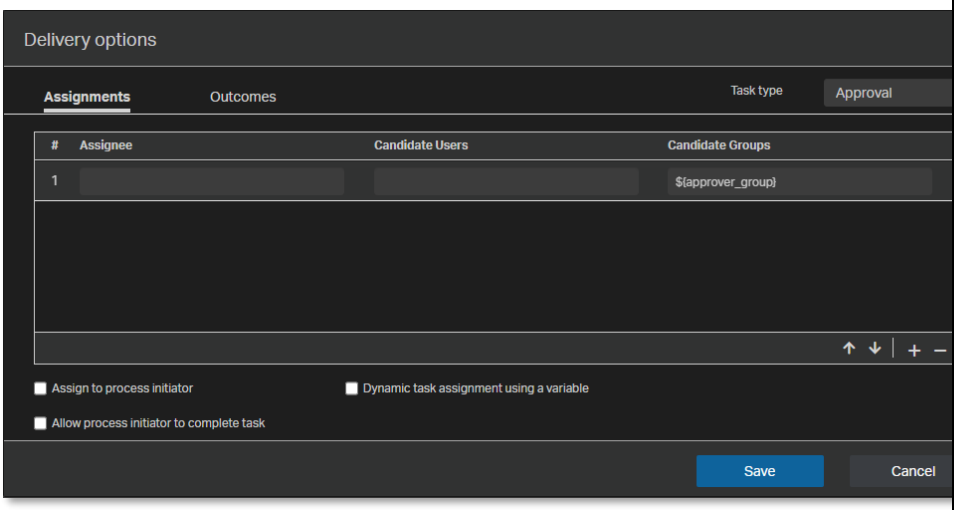
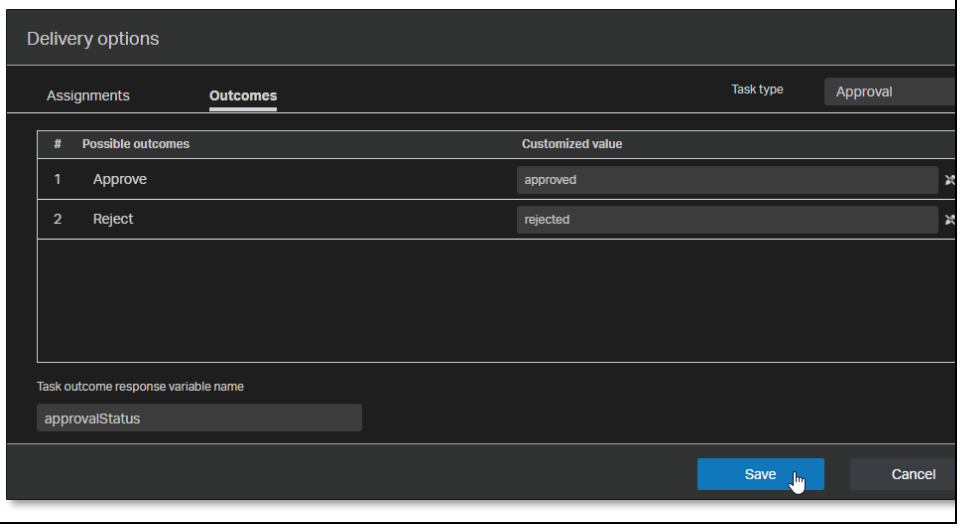
The user task is a typical workflow task where a user performs the task with the assistance of a software application. It is scheduled through a task list manager. In a workflow, user tasks are the primary way to interact with humans within a process. After the execution reaches such a task, a user is required to perform an action. As an outcome of the action, it is possible to create and update variables to use in other tasks or to control the flow of the process. Each task can be assigned to one or more users and shared with any number of groups. A task can optionally have a due date.

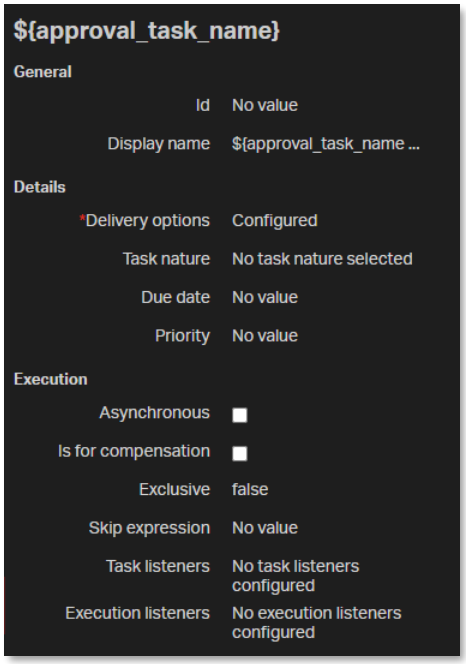
7. Drag and drop the **User task** to the **canvas** next to the **Update Manager Approval trait http task**.

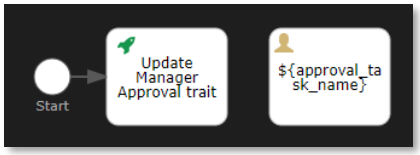


8. Fill the User task attributes using the following details:

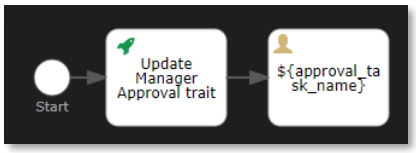
Attribute	Value
Display name	In the **manager_approval** workflow, the display name of the manual approval task which signals the type of approval (that is, who is doing the approval, the line manager or the risk manager) that is passed as the **approval_task_name** process input parameter. In this case we take the name from a variable. Enter the following display name: `${approval_task_name}`

Attribute	Value
Delivery options	Complete the user task behavior configuration using the following `options`: • Task type: The type of user task • Assignments: How the user task gets assigned • Outcomes: What to pass on (in a process variable) as result of the user action This user task is an approval task to assign to one of the members of the approver group (that is, line managers or risk managers) which is passed as the **approver_group** process input parameter. The two possible user task outcomes are approval or rejection of the contract. Set the **Task type** to **Approval**. Under the **Assignments** tab, click the **Add** button and enter the following for the **Candidate Groups**: `${approver_group}`  Select the **Outcomes** tab and change the **Customized value** entries to `approved` and `rejected`, set the **Task outcome response variable name** to `approvalStatus`, and click **Save** to confirm the delivery options. 

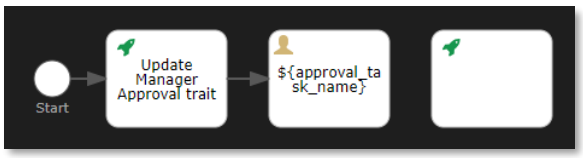




9. Select the **Update Manager Approval trait http task** and drag a **sequence flow** to the **User task**.



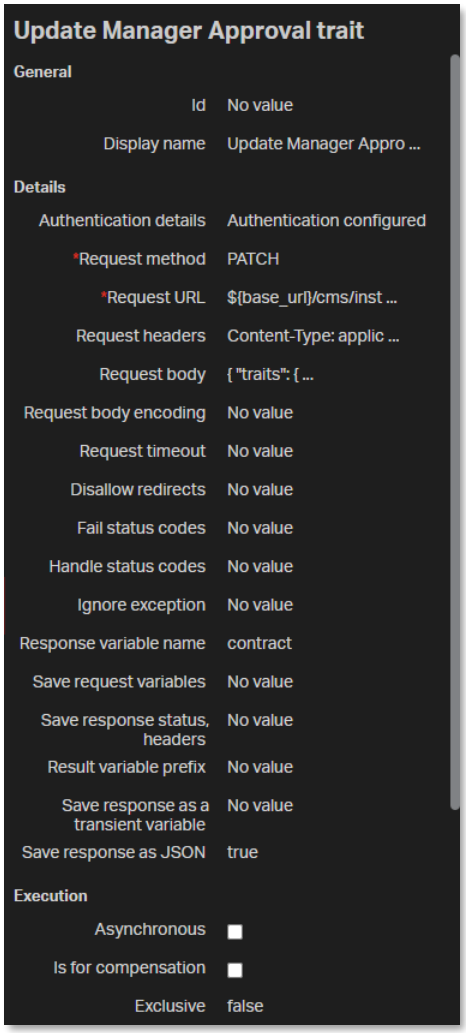
10. Drag and drop another **Http task** from the **palette** to the **canvas** next to the **User task**.

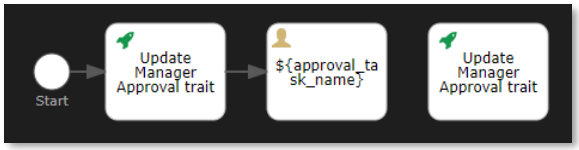


11. Fill the Http task attributes using the following details:

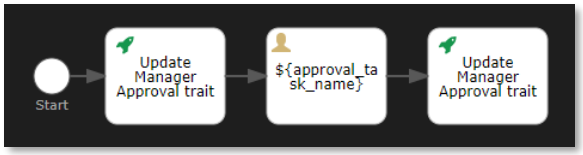
Attribute	Value
Display name	`Update Manager Approval trait`
Authentication details	Select **Use current authentication token**.
Request method	Select the **PATCH** request method.

Attribute	Value
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	`Content-Type: application/json`
Request body	This task sets the **is_required**, **has_been_granted**, **approver**, **approver_role**, and **approval_date** properties for the contract approval trait specified by the **approval_trait_name** process input parameter. Enter the following request body: `{ "traits": { "ca_approval": { "${approval_trait_name}": { "is_required": true, "has_been_granted": ${approvalStatus == "approved"}, "approver": "${approver}", "approver_role": "Line Manager", "approval_date": "${contract.update_time}" } } } }`
Response variable name	`contract`
Save response as JSON	`true`

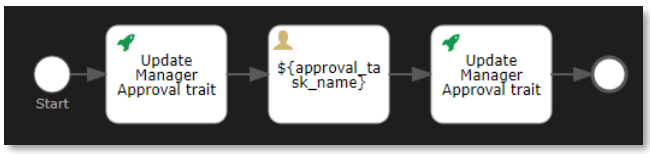




12. Select the **User task** and drag a **sequence flow** to the second **Update Manager Approval trait http task**.

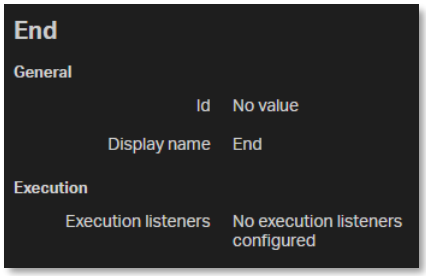


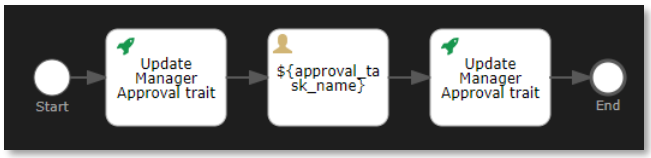
13. Select the second **Update Manager Approval trait http task** and drag and drop an **End event** next to it.



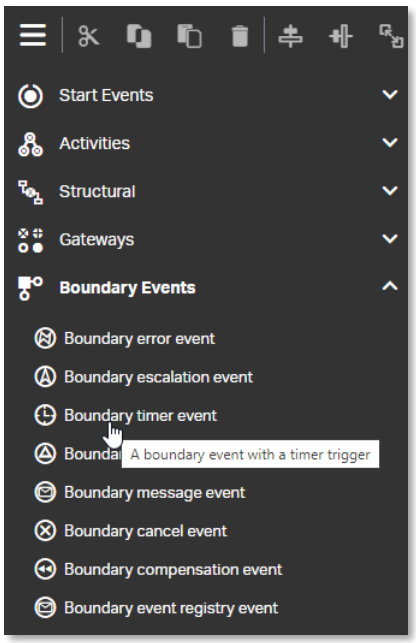
14. Fill the end event attributes using the following details:

Attribute	Value
Display name	`End`





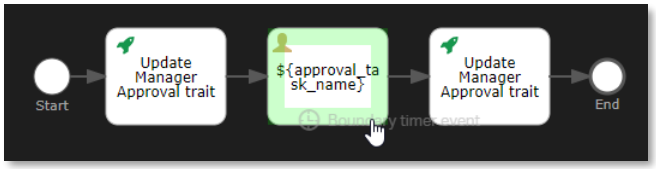
15. In the **palette**, find **Boundary Events > Boundary timer event**.



Note

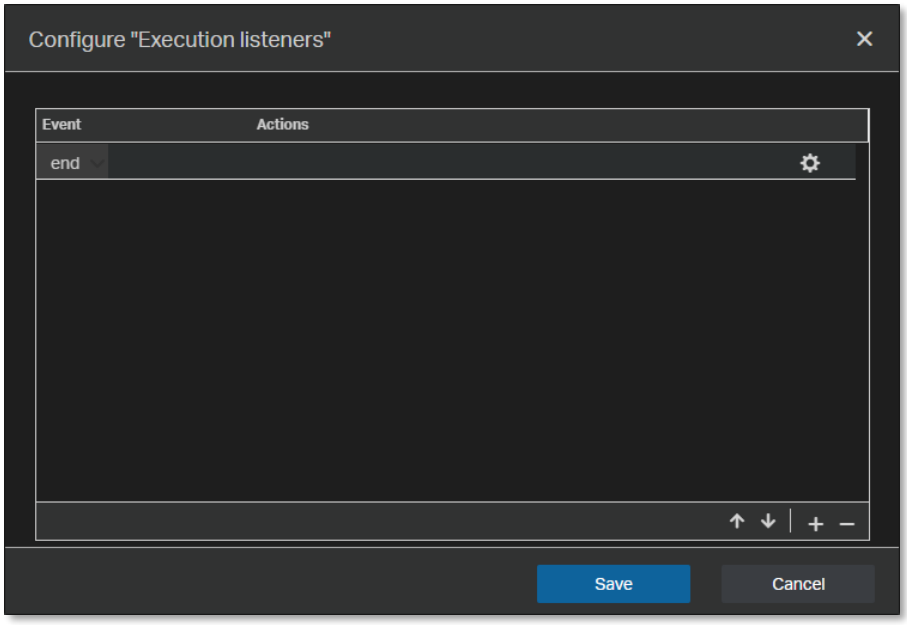
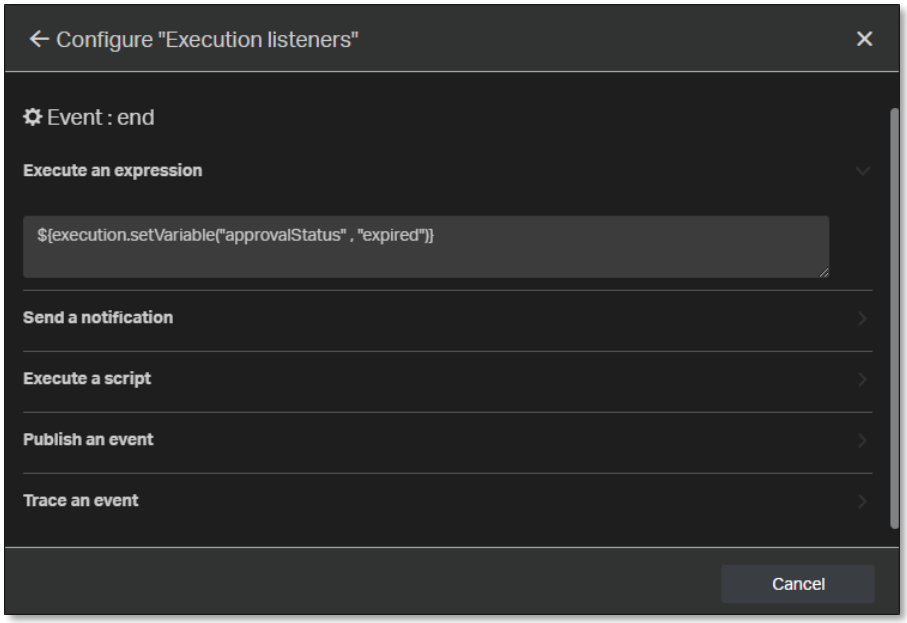
The boundary timer event acts as a stopwatch and alarm clock. When an execution arrives at the activity where the boundary event is attached, a timer starts. When the timer fires (for example, after a specified interval), the activity is interrupted, and the outgoing sequence flow of the boundary event is followed.

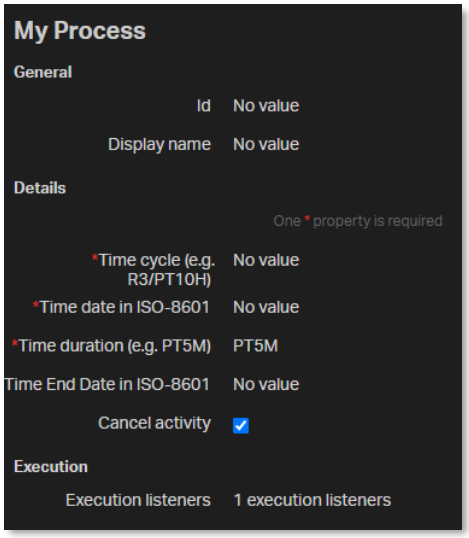
16. Drag and drop the **Boundary timer event** on top of the **User task**'s bottom corner that is nearest to the **End event**. The **User task** changes to green to indicate when the timer boundary event can be released/dropped to be correctly linked.



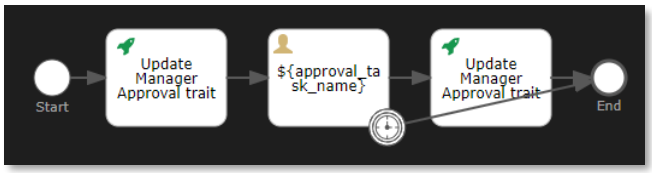
17. Select the **Boundary timer event** and fill its attributes using the following details:

Attribute	Value
Time duration (e.g. PT5M)	Specifies how long the timer must run before it is triggered. The ISO 8601 format is used as required by the BPMN 2.0 specification. **Note** There are three different ways to configure the timer: • Time cycle: Specifies (ISO 8601 format) a repeating interval for starting the process periodically or for sending multiple reminders. • Time date: Specifies a fixed date (ISO 8601 format) when the trigger will fire. • Time duration: Specifies (ISO 8601 format) how long the timer must run before it is fired. For the boundary timer event, it is required to fill one of the three timer configuration properties. The manager approval task must expire (that is, the timer event must fire) after **5 minutes**. Enter the following time duration: `PT5M`

Execution listeners	Active execution listeners of the activity. Allows configuring an action when the **start**, **take**, or **end** events occur for the boundary time event. For the manager approval task, the **approvalStatus** process variable must be set to **expired** when the timer fires. Configure the Execution listeners as follows: Click the **Add** button and select the **end** event.  Click the **Configure listener definition** button to configure the action and select **Execute an expression**. In **Execute an expression** box, enter the following expression: `${execution.setVariable("approvalStatus" , "expired")}`  Click the **Switch to previous view** button to go back and click **Save** to confirm the execution listeners.



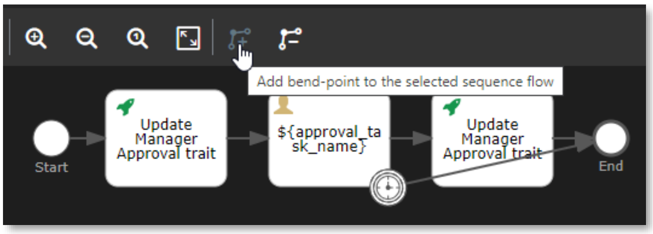
18. Select the **Boundary timer event** and drag a **sequence flow** to the **End event**.

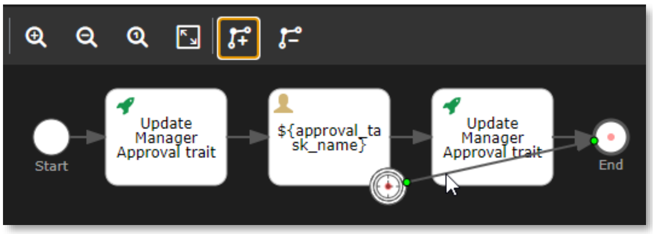


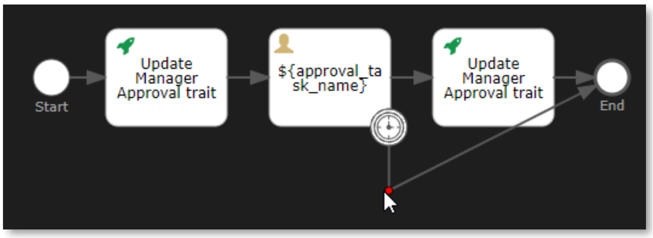
19. Click the **Sequence flow bend-point** button  to add two **bend-points** to the **sequence flow** between the **Boundary timer event** and the **End event**.

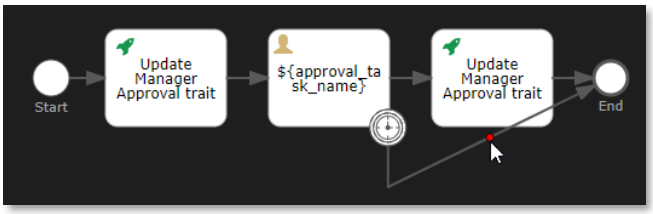
Note

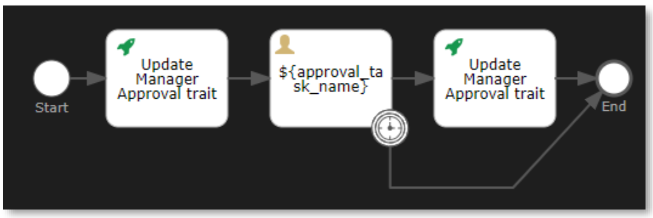
Bend-points allow you to change the visual path of a sequence flow to display the BPMN process more clearly. Add bend-points where needed to make your BPMN process more readable.



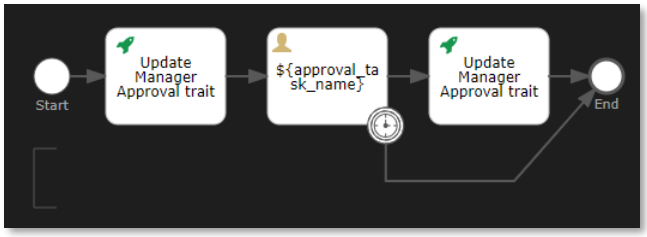








20. Drag and drop a **Text annotation** from the **palette** to the **canvas** under the **start event**.

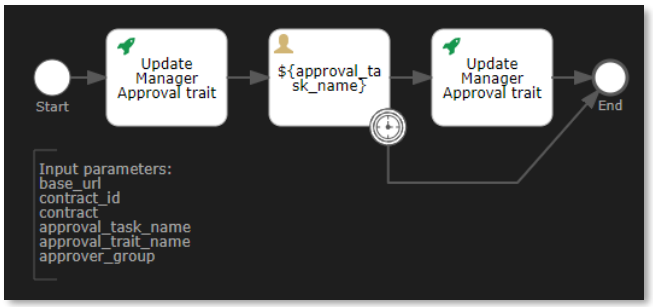


21. Double-click the **Text annotation** on the **canvas** to edit it inline.
22. Set the following text for the text annotation:

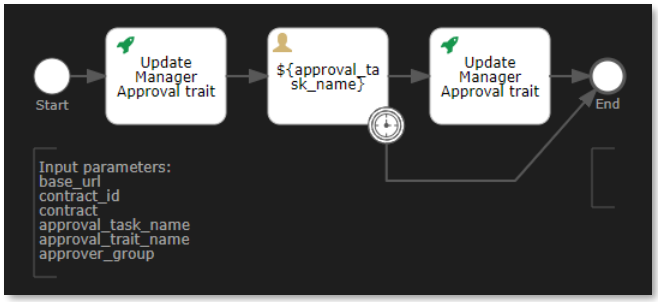
Text

```
Input parameters:
base_url
contract_id
contract
approval_task_name
approval_trait_name
approver_group
```

23. Resize the **text annotation** so that the text displays correctly and fits the square brackets.



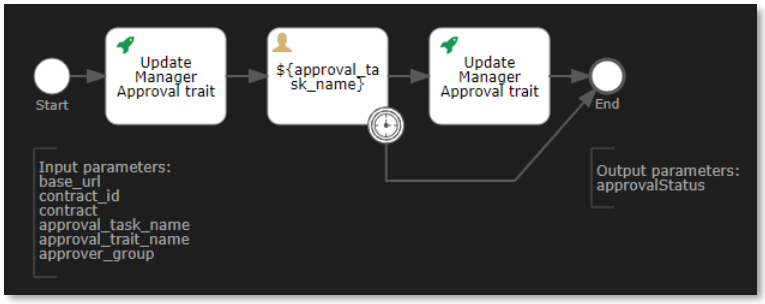
24. Drag and drop another **Text annotation** from the **palette** to the **canvas** under the **End event**.



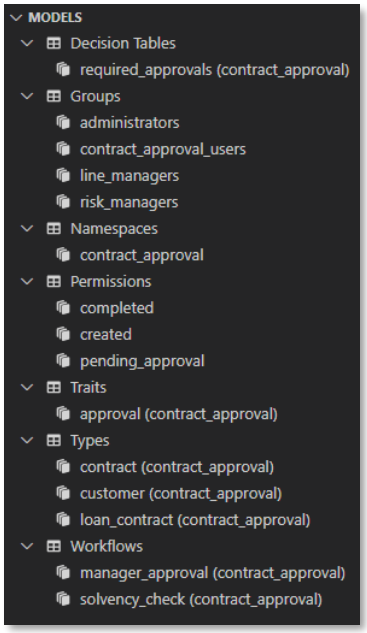
25. Double-click the **Text annotation** on the **canvas** to edit it inline and set the following text for the text annotation:

Text
`Output parameters:` `approvalStatus`

26. Resize the **Text annotation** so that the text displays correctly and fits the square brackets.



27. Save and close the Manager Approval workflow model. The model explorer displays the new **manager_approval (contract_approval)** workflow under **Workflows**.



Next step:

Create the Contract Approval workflow.

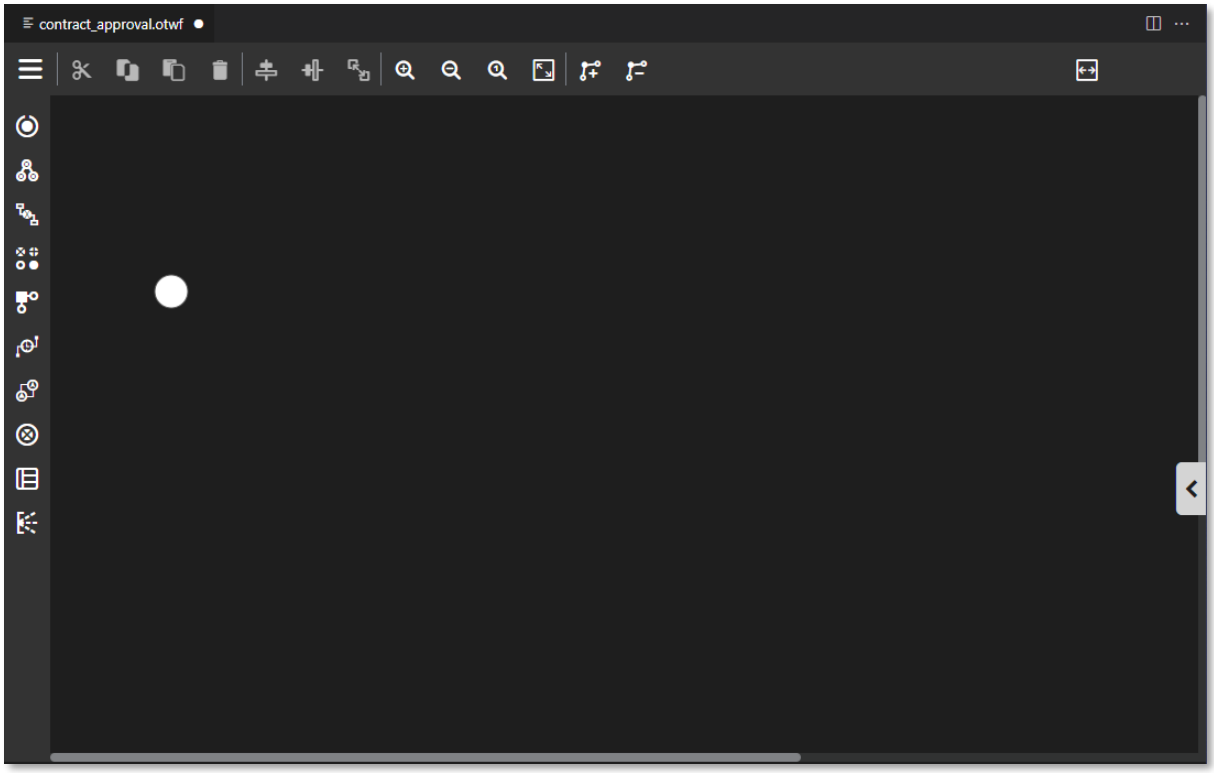
11.3Create the Contract Approval workflow

For convenience, the creation of the Contract Approval workflow is split into several sub chapters:

- Create the workflow model
- Set the workflow attributes
- Build the process definition

11.3.1 Create the workflow model

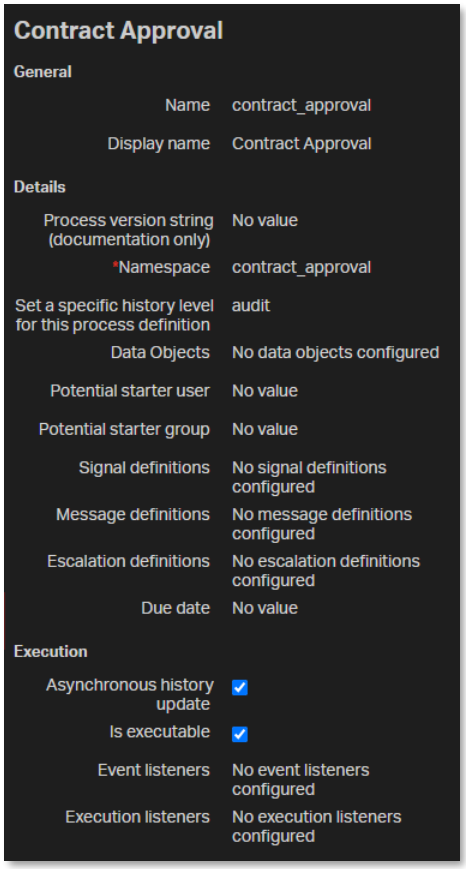
1. In VS Code, create a new workflow model using any of the three model creation methods explained in the following exercises:
   - Create the Contract Approval namespace
   - Create the Approval trait
   - Create the Contract type
2. Type `contract_approval` for the (file) name of the workflow.



11.3.2 Set the workflow attributes

1. Click the empty space in the **canvas** (that is, do not select the start event).
2. Expand the attribute bar and fill the workflow attributes using the following details:

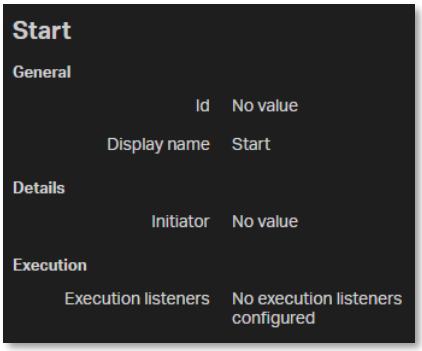
Attribute	Value
Name	`contract_approval`
Display name	`Contract Approval`
Namespace	Select the **contract_approval** namespace.

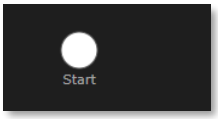


11.3.3 Build the process definition

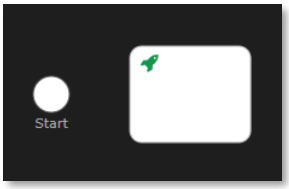
1. Select the **Start event** that is already on the **canvas**.
2. Fill the Start event attributes using the following details:

Attribute	Value
Display name	`Start`



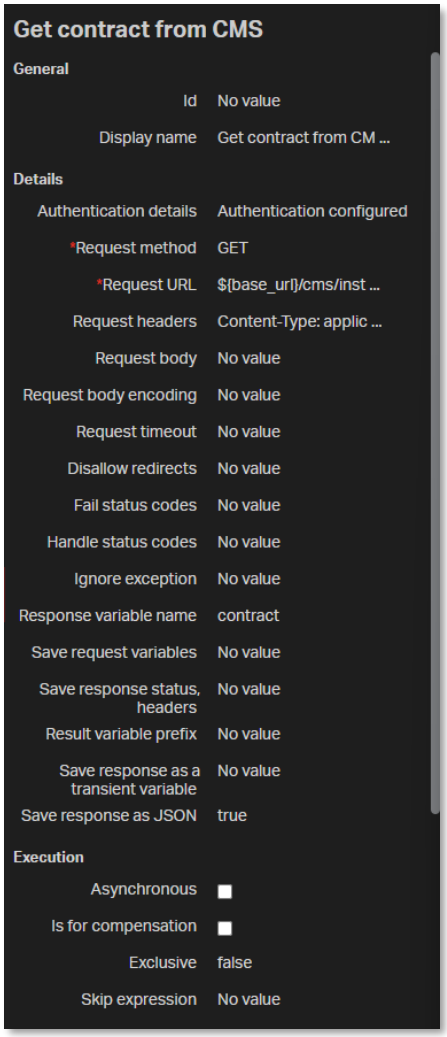


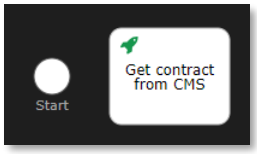
3. Drag and drop an **Http task** to the **canvas** next to the **Start event**.



4. Fill the Http task attributes using the following details:

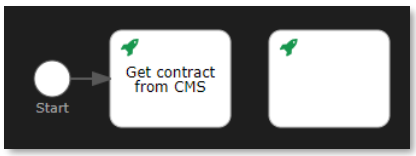
Attribute	Value
Display name	`Get contract from CMS`
Authentication details	Select **Use current authentication token**.
Request method	This task fetches the contract metadata from CMS, so this implies using the GET method. Leave the selected **GET** request method as the request method.
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	`Content-Type: application/json`
Response variable name	`contract`
Save response as JSON	`true`





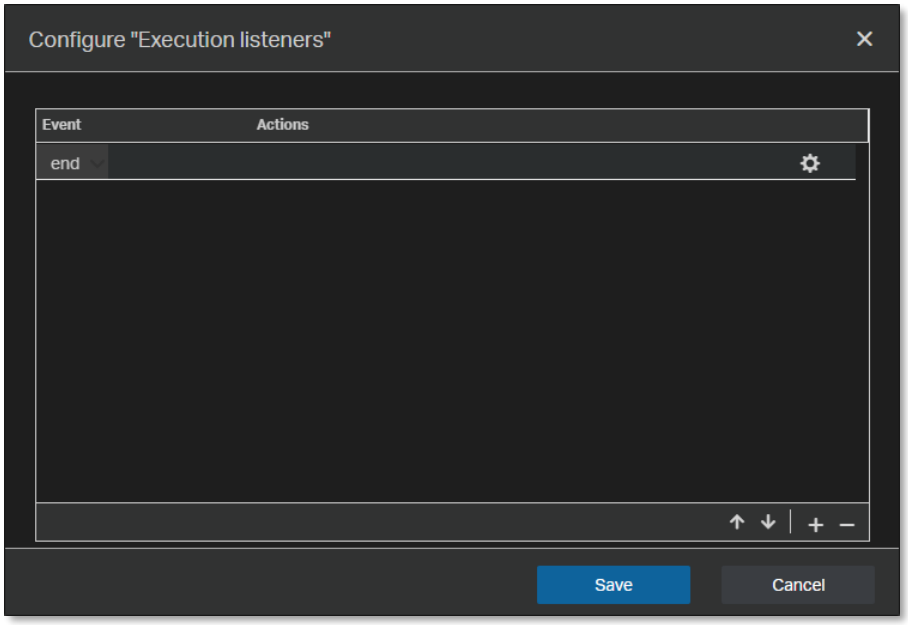
5. Select the **Start event** and drag a **sequence flow** to the **Get contract from CMS http task**.

6. Drag and drop a second **Http task** from the **palette** to the **canvas** next to the **Get contract from CMS http task**.

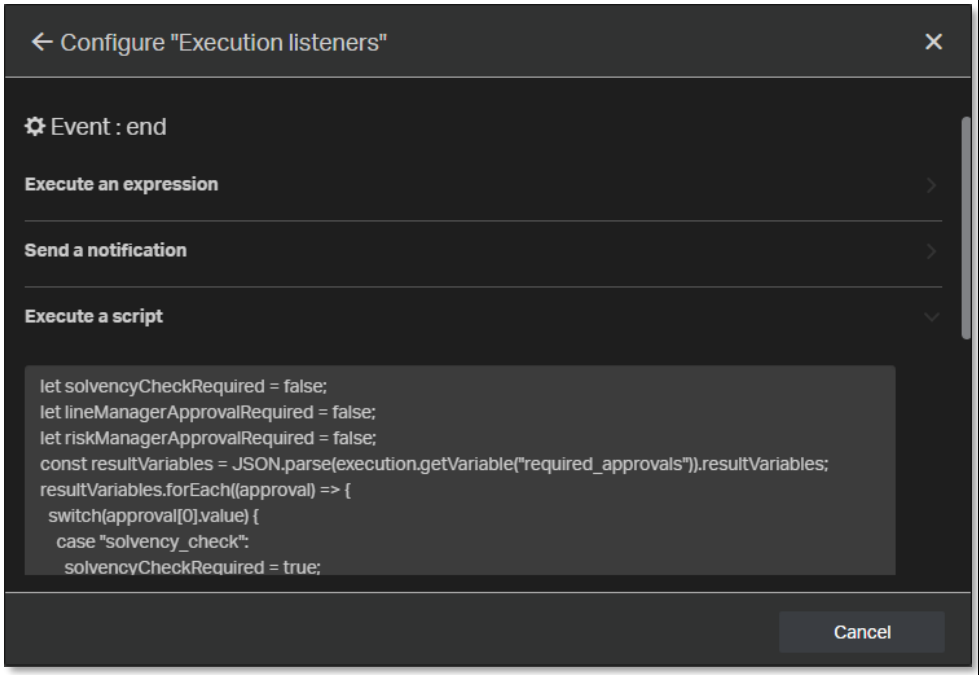
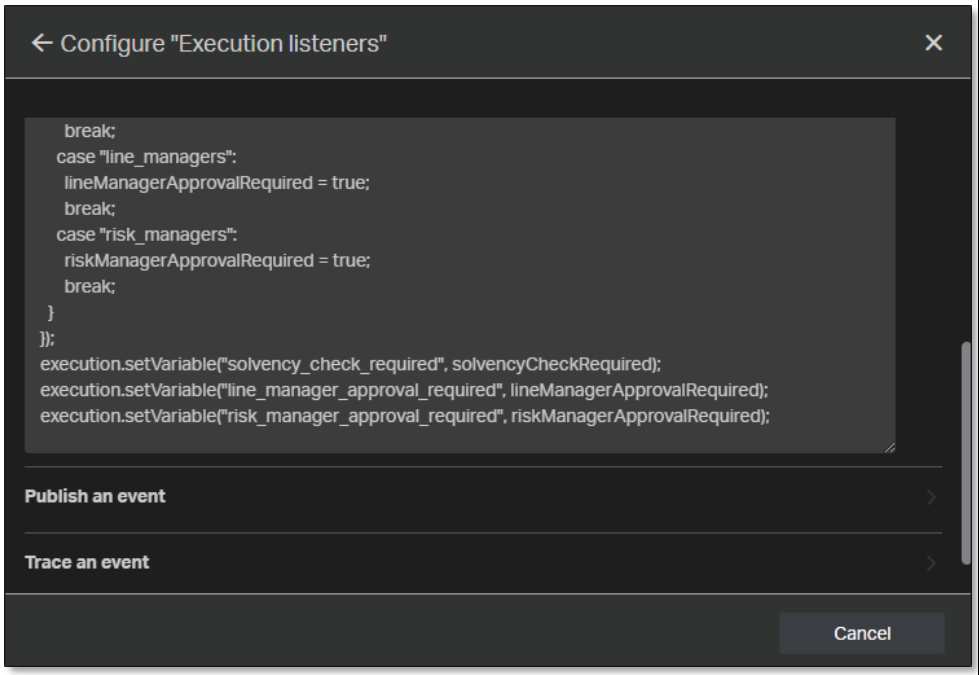


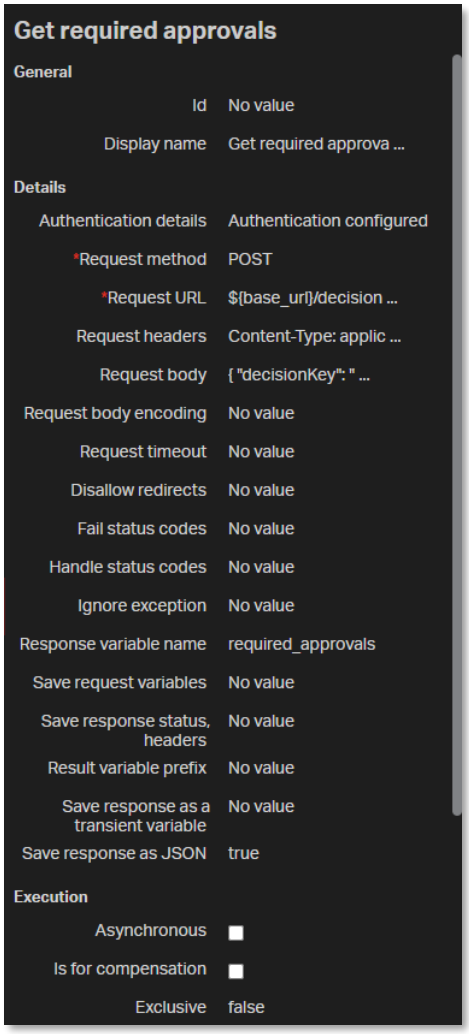
7. Fill the Http task attributes using the following details:

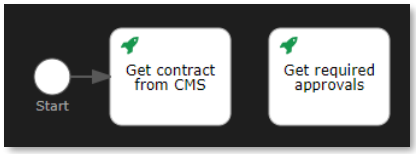
Attribute	Value
Display name	`Get required approvals`
Authentication details	Select **Use current authentication token**.
Request method	This task calls the **Decision Service** to execute the decision table that calculates the required approvals. This implies using the POST method. Select the **POST** request method.
Request URL	Enter the following request URL to call the **Decision Service**: `${base_url}/decision/v1/execute`
Request headers	`Content-Type: application/json`
Request body	This task executes the **required_approvals** decision table with the **contract_type**, **contract_value**, and **risk_level** contract property values as input variables. Enter the following request body: `{ "decisionKey": "required_approvals", "inputVariables": [{ "name": "contract_type", "value": "${contract.type}", "type": "string" }, { "name": "contract_value", "value": ${contract.properties.value}, "type": "integer" }, { "name": "risk_level", "value": ${contract.properties.risk_classification}, "type": "integer" }] }`
Response variable name	`required_approvals`
Save response as JSON	`true`

Execution listeners	Active execution listeners of the activity. Allows configuring an action when the **start**, **take**, or **end** events occur for the boundary time event.
	
	For the Get required approvals http task the **solvency_check_required**, **line_manager_approval_required**, and **risk_manager_approval_required** process variables must be set according to the **required_approvals** result from the Decision Service call.
	
	Click the **Add** button and select the **End** event.
	
	 
	
	Click the **Configure listener definition** button to configure the action and select **Execute a script**.
	
	Expand the script input box (for readability) and enter the following script:
	
	`let solvencyCheckRequired = false;`
	`let lineManagerApprovalRequired = false;`
	`let riskManagerApprovalRequired = false;`
	
	`const resultVariables = JSON.parse(`
	`  execution.getVariable("required_approvals")`
	`).resultVariables;`
	`resultVariables.forEach((approval) => {`
	`  switch(approval[0].value) {`
	`  case "solvency_check":`
	`  solvencyCheckRequired = true;`
	`  break;`
	`  case "line_managers":`
	`  lineManagerApprovalRequired = true;`
	`  break;`
	`  case "risk_managers":`
	`  riskManagerApprovalRequired = true;`
	`  break;`
	`  }`
	`});`
	
	`execution.setVariable("solvency_check_required",`
	`solvencyCheckRequired);`

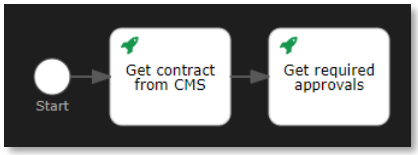
Attribute	Value

	`execution.setVariable("line_manager_approval_required", lineManagerApprovalRequired);`
	`execution.setVariable("risk_manager_approval_required", riskManagerApprovalRequired);`
	 
	
	 
	
	Click the **Switch to previous view** button to go back and click **Save** to confirm the execution listeners.

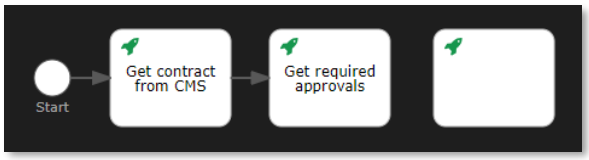




8. Select the Get contract from CMS http task and drag a sequence flow to the Get required approvals http task.



9. Drag and drop a third **Http task** from the **palette** to the **canvas** next to the **Get required approvals http task**.

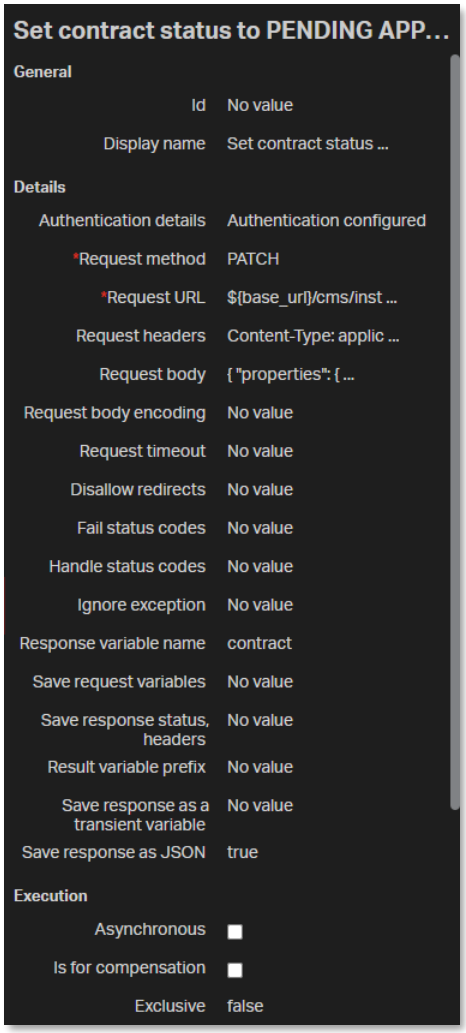


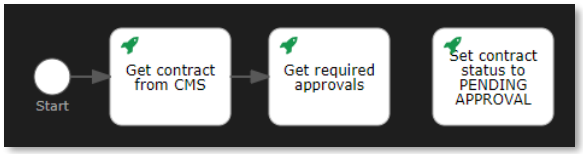
10. Fill the Http task attributes using the following details:

Attribute	Value
Display name	`Set contract status to PENDING APPROVAL`
Authentication details	Select **Use current authentication token**.
Request method	This task sets the value for the status attribute of the given contract, so this implies using the PATCH method. Select the **PATCH** request method.
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	`Content-Type: application/json`

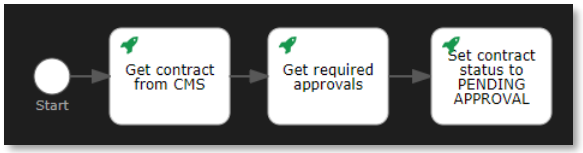
Request body	This task sets the contract's **status** property to **PENDING APPROVAL**.
	Enter the following request body:
	`{`
	`  "properties": {`
	`  "status": "PENDING APPROVAL"`
	`  }`
	`}`

Response variable name	`contract`
Save response as JSON	`true`

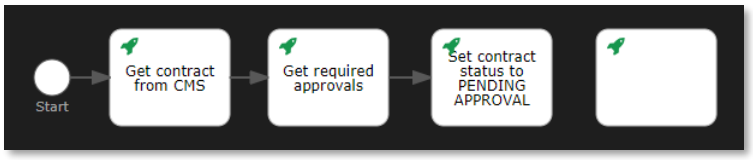




11. Select the **Get required approvals http task** and drag a **sequence flow** to the **Set contract status to PENDING APPROVAL http task**.



12. Drag and drop a fourth **Http task** from the **palette** to the **canvas** next to the **Set contract status to PENDING APPROVAL http task**.

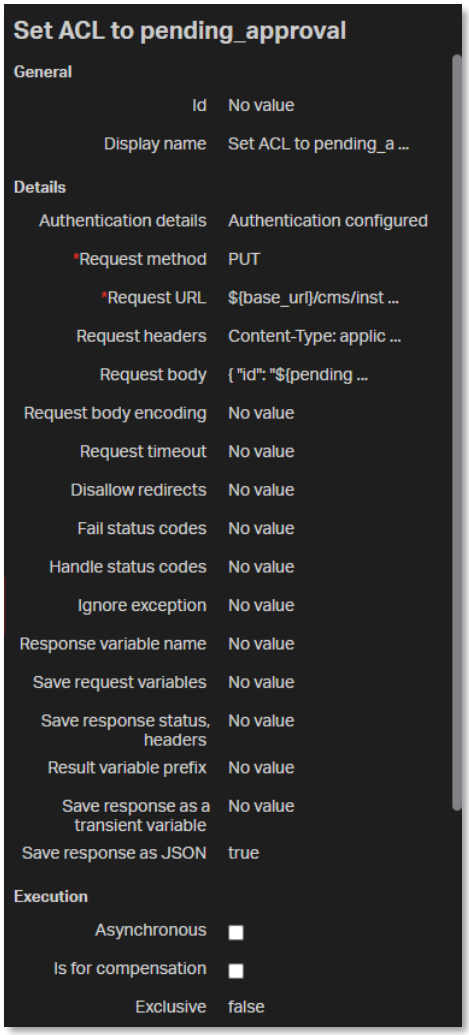


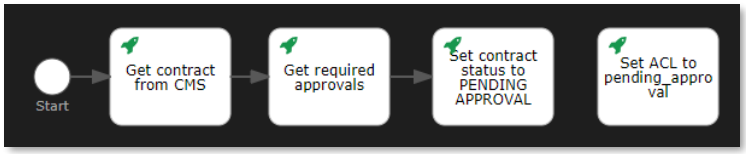
13. Fill the Http task attributes using the following details:

Attribute	Value
Display name	`Set ACL to pending_approval`
Authentication details	Select **Use current authentication token**.
Request method	This task sets the ACL of the given contract. Setting the ACL for a file instance requires calling a separate (acl) file instance endpoint to replace the current ACL resource, which implies using the PUT method. Select the **PUT** request method.
Request URL	Enter the following request URL to call the **acl** endpoint for the given contract: `${base_url}/cms/instances/file/ca_contract/${contract_id}/acl`
Request headers	`Content-Type: application/json`

Request body	This task sets the ACL of the contract by using the **pending_approval_acl_id** process input parameter as the **id** of the ACL to apply.
	Enter the following request body:
	`{`
	`  "id": "${pending_approval_acl_id}"`
	`}`

Save response as JSON	`true`

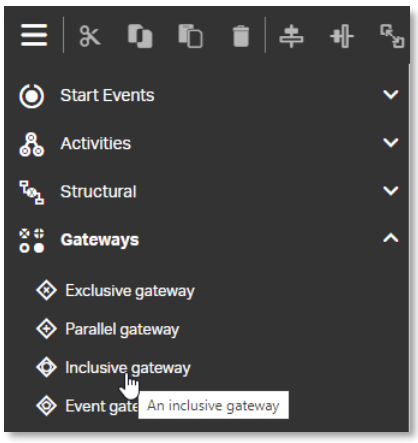




14. Resize the **Set ACL to pending_approval http task** on the **canvas** to ensure the text displays correctly and “pending_approval” does not get split across lines.

15. Select the **Set contract status to PENDING APPROVAL http task** and drag a **sequence flow** to the **Set ACL to pending_approval http task**.

16. In the **palette**, find **Gateways > Inclusive gateway**.



Note

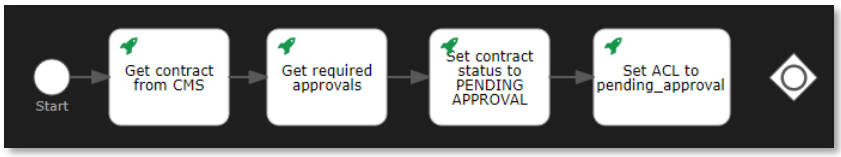
A gateway controls the flow of execution. A gateway is capable of consuming or generating tokens.

There are four types of gateways:

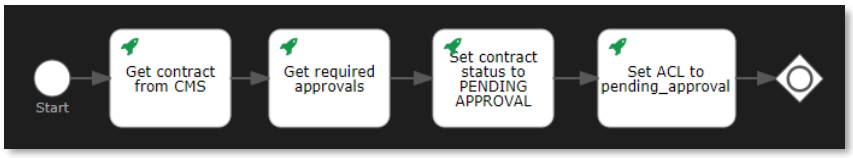
- Exclusive gateway: Is used to model a decision in the process. When the execution arrives at this gateway, all outgoing sequence flows are evaluated in the order in which they are defined. The first sequence flow whose condition evaluates to true is selected for continuing the process.
- Parallel gateway: Is the most straightforward gateway to introduce concurrency in a process model. It allows you to fork into multiple paths of execution or join multiple incoming paths of execution.
- Inclusive gateway: Can be seen as a combination of an exclusive and a parallel gateway. Like an exclusive gateway, you can define conditions on outgoing sequence flows and the inclusive gateway will evaluate them. However, the main difference is that like the parallel gateway, the inclusive gateway can take more than one sequence flow.
- Event gateway: Provides a way to take a decision based on events. Each outgoing sequence flow of the gateway must be connected to an intermediate catching event. When process execution reaches an event-based gateway, the gateway acts like a wait state: execution is suspended. In addition, for each outgoing sequence flow, an event subscription is created.

In this tutorial you will be using inclusive and exclusive gateways to control the contract approval flow execution.

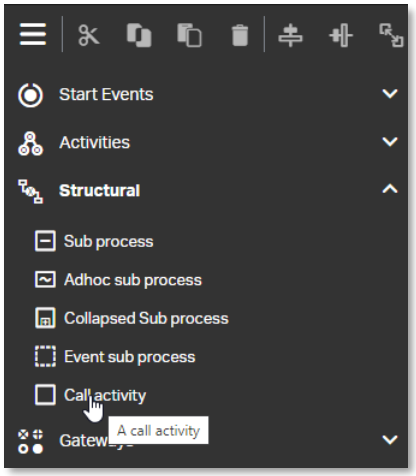
17. Drag and drop the **Inclusive gateway** to the **canvas** next to the **Set ACL to pending_approval http task**.



18. Select the **Set ACL to pending_approval http task** and drag a **sequence flow** to the **inclusive gateway**.



19. In the **palette**, find **Structural > Call activity**.



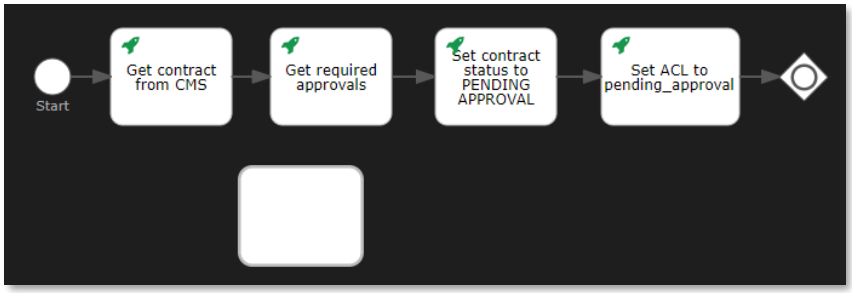
Note

BPMN 2.0 makes a distinction between a regular sub-process, often also called embedded sub-process, and the call activity. Both call a sub-process when the process execution arrives at the activity. The difference is that the call activity references a process that is external to the process definition, whereas the sub-process is embedded within the original process definition. The main use case for the call activity is to have a reusable process definition that can be called from multiple other process definitions.

When process execution arrives at the call activity, a new execution is created that is a sub-execution of the execution that arrived at the call activity. This sub-execution is then used to execute the sub-process, potentially creating parallel child executions within a regular process. The super-execution waits until the sub-process ends and continues with the original process afterward.

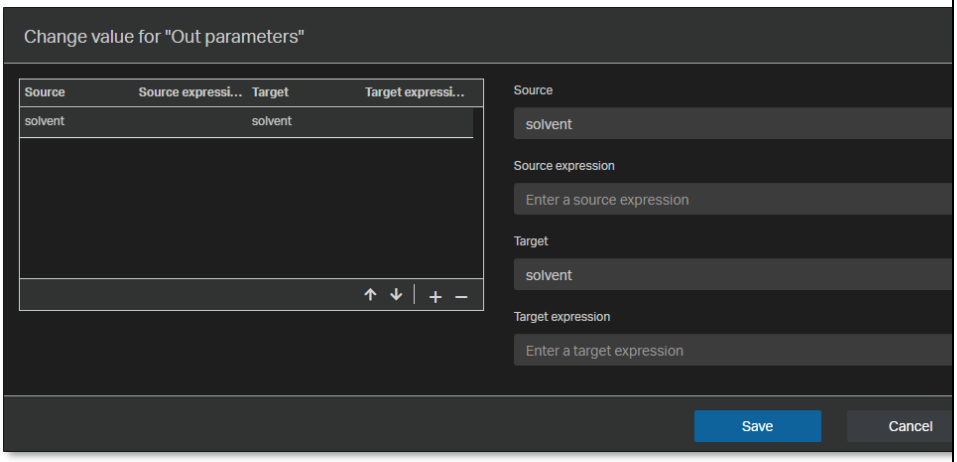
For the contract approval workflow, the call activity will be used to call both the previously created Solvency Check and Manager Approval workflows as sub-processes.

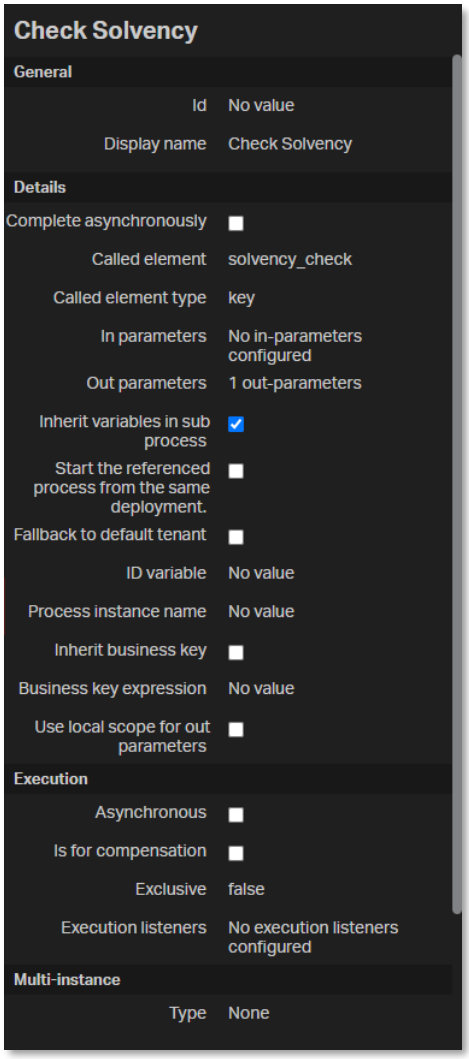
20. Drag and drop the **Call activity** to the **canvas** below the previously created **Http tasks** and **Inclusive gateway.** Leave enough vertical space to draw sequence flow connectors from the inclusive gateway and leave enough horizontal space to add a text annotation under the **Start event** and **Get contract from CMS http task**.

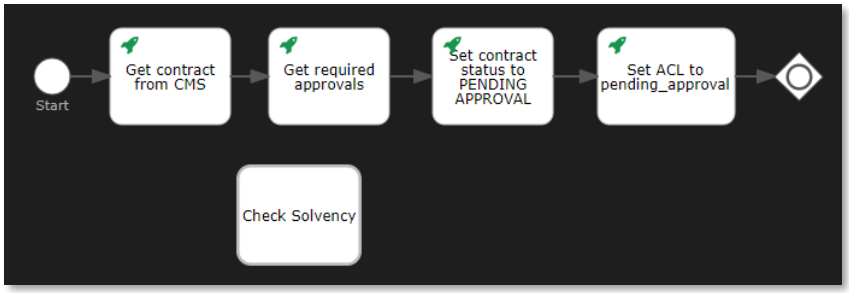


21. Fill the call activity attributes using the following details:

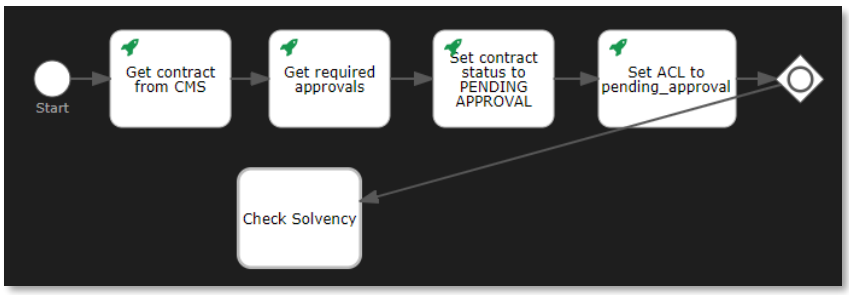
Attribute	Value
Display name	This call activity calls the **Solvency Check** workflow to perform the solvency check. Enter the following display name: `Check Solvency`
Called element	The name (also known as the key) or ID of the workflow to call. This call activity calls the **Solvency Check** workflow by name to perform the solvency check. Enter the following for the called element: `solvency_check`
Called element type	Whether to use the key (that is, the name) or the ID of the deployed process definition to start the process referenced in the **Called element** property. Select the **key** value for the called element type.

Attribute	Value
Out parameters	Optional output parameter map. Allows the mapping of parameters and variables from the called sub-process to the calling process (that is, the workflow that contains the call activity). For the Check Solvency call activity, the **solvent** process variable of the called (that is, **source**) solvency_check workflow must be mapped to a new **solvent** process variable of the calling (that is, **target**) workflow. Click the **Add** button and enter `solvent` for both the **Source** and **Target** process variables.  Click **Save** to confirm the out parameters.
Inherit variables in sub process	Whether to inherit the parent process variables in the sub-process. **Select** the checkbox.

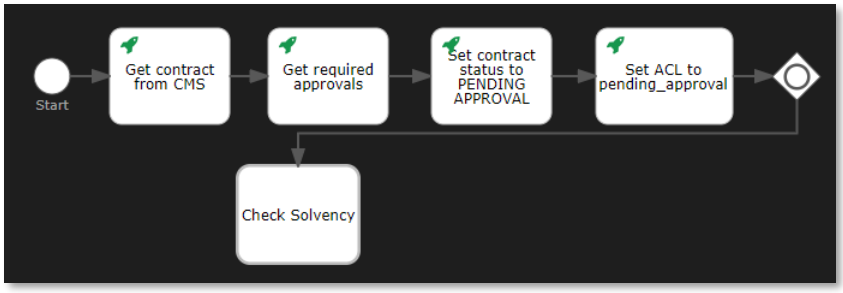




22. Select the **inclusive gateway** and drag a sequence flow to the Check Solvency call activity.

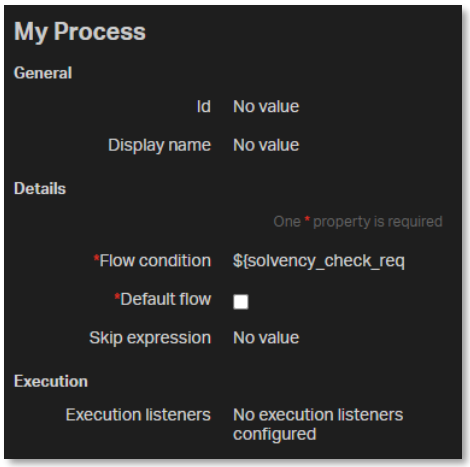


23. Add two bend-points to the new sequence flow.

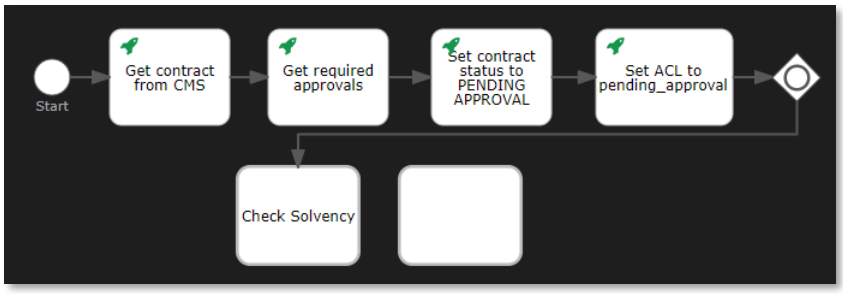


24. Select the new sequence flow and fill the sequence flow attributes using the following details:

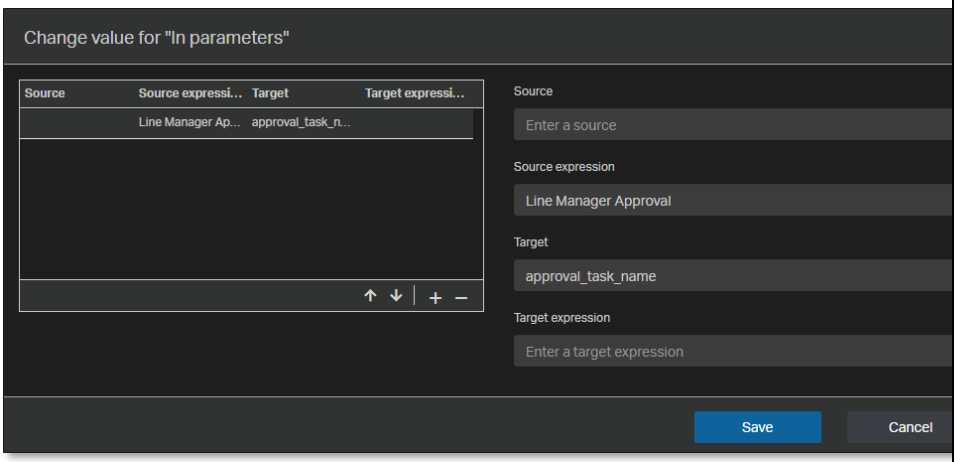
Attribute	Value
Flow condition	The condition that defines whether the sequence flow is selected from the connected gateway. This flow must be selected by the inclusive gateway if an automated solvency check is required. This information is available from the **solvency_check_required** process variable that was set after calling the Decision Service in the Get required approvals http task. Enter the following flow condition: `${solvency_check_required}`

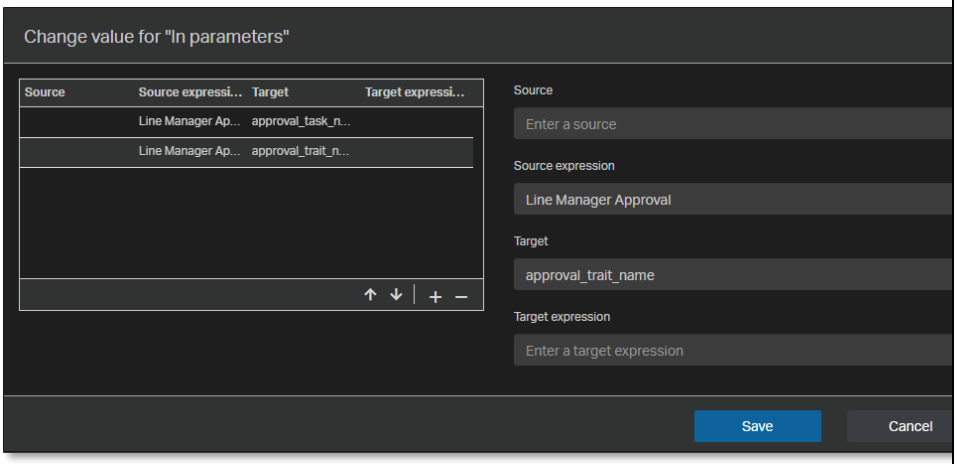
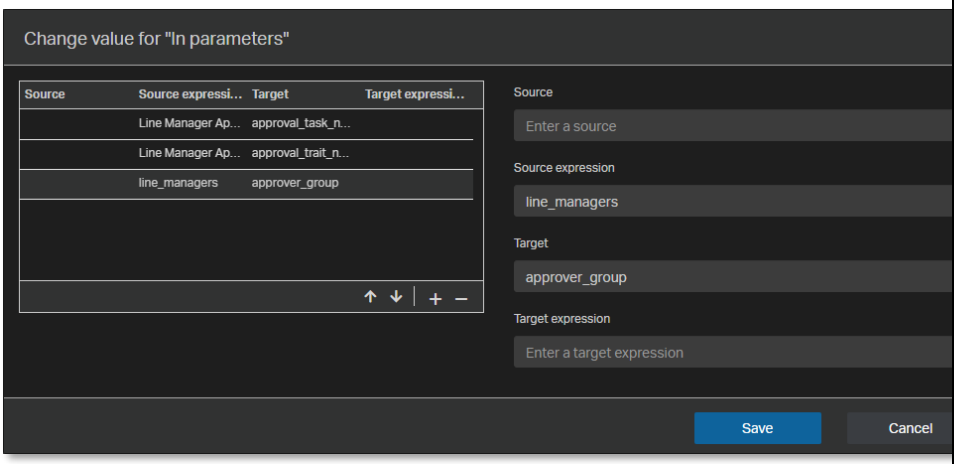


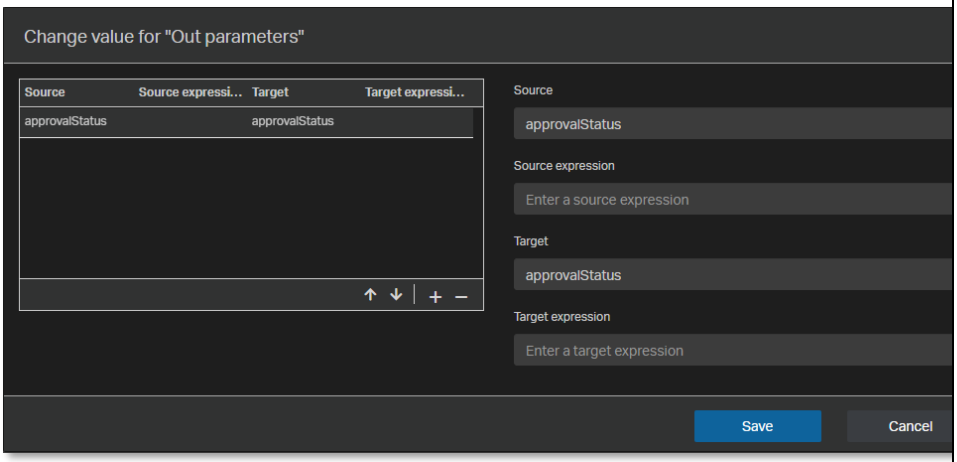
25. Drag and drop a second **Call activity** from the **palette** to the **canvas** next to the **Check Solvency call activity**.

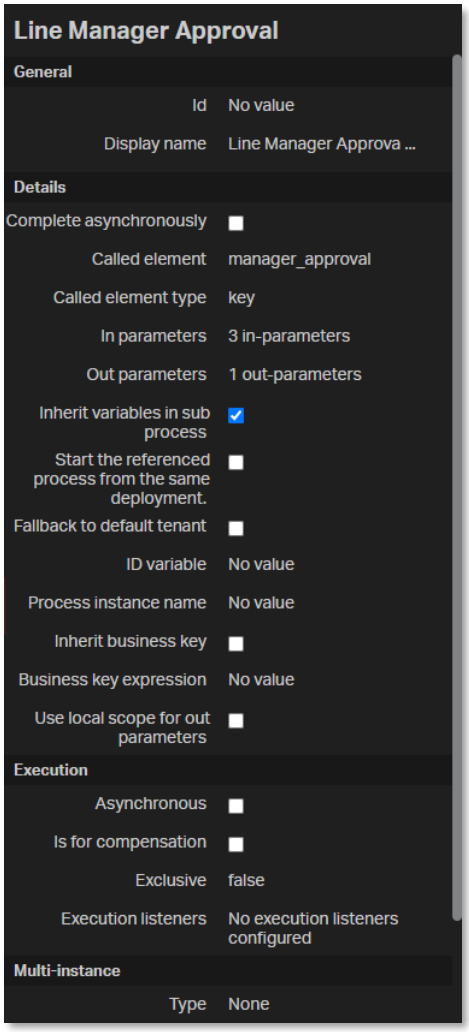


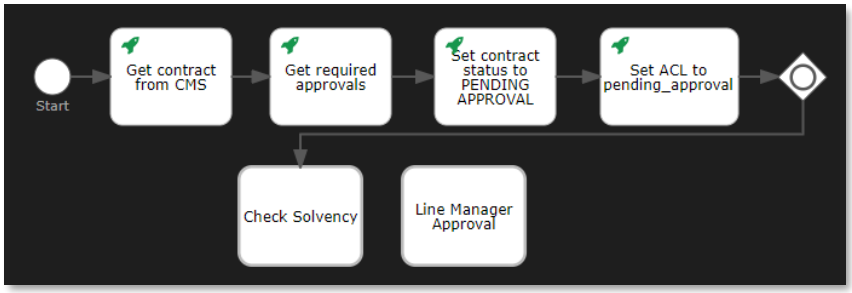
26. Fill the call activity attributes using the following details:

Attribute	Value
Display name	This call activity calls the **Manager Approval** workflow to perform the line manager approval. Enter the following display name: `Line Manager Approval`
Called element	`manager_approval`
Called element type	Select the **key** value for the called element type.
In parameters	Optional input parameter map. Allows the mapping of parameters and variables from the calling process (that is, the workflow that contains the call activity) to the called sub-process. For the Line Manager Approval call activity, the values of **t**he **approval_task_name**, **approval_trait_name**, and **approver_group** process variables of the called manager_approval sub-process need to be set. Click the **Add** button and enter `Line Manager Approval` as the **source expression** (in this case a literal text value) and `approval_task_name` as the **target** process variable to map it to in the called sub-process. 

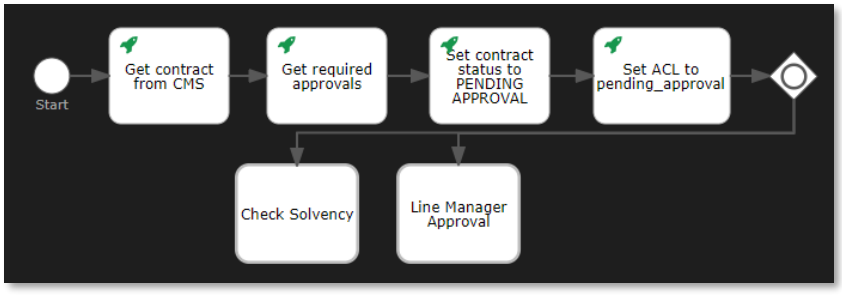
Attribute	Value
	Click the **Add** button to add a second input parameter and enter `Line Manager Approval` (literal text value) as the **source expression** and `approval_trait_name` as the **target** process variable to map it to in the called sub-process.  Click the **Add** button to add a third input parameter and enter `line_managers` (literal text value) as the **source expression** and `approver_group` as the **target** process variable to map it to in the called sub-process.  Click **Save** to confirm the in parameters.

Attribute	Value
Out parameters	For the Line Manager Approval call activity, the **approvalStatus** process variable of the called (that is, **source**) manager_approval workflow must be mapped to a new **approvalStatus** process variable of the calling (that is, **target**) workflow. Click the **Add** button and enter `approvalStatus` for both the **Source** and **Target** process variables.  Click **Save** to confirm the out parameters.
Inherit variables in sub process	Select the checkbox.



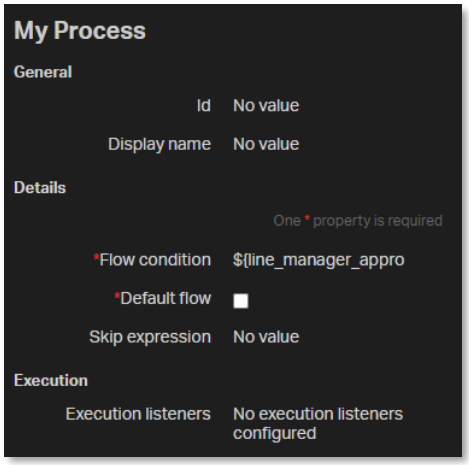


27. Select the **inclusive gateway**, drag a **sequence flow** to the **Line Manager Approval call activity**, and add two bend-points to the new sequence flow.

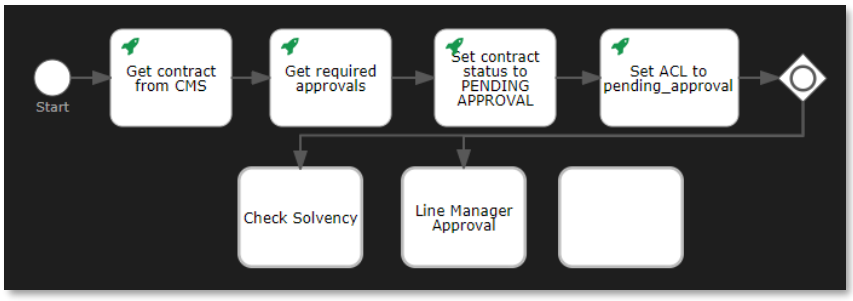


28. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Flow condition	This flow must be selected by the inclusive gateway if a line manager approval is required. This information is available from the **line_manager_approval_required** process variable that was set after calling the Decision Service in the Get required approvals http task. Enter the following flow condition: `${line_manager_approval_required}`

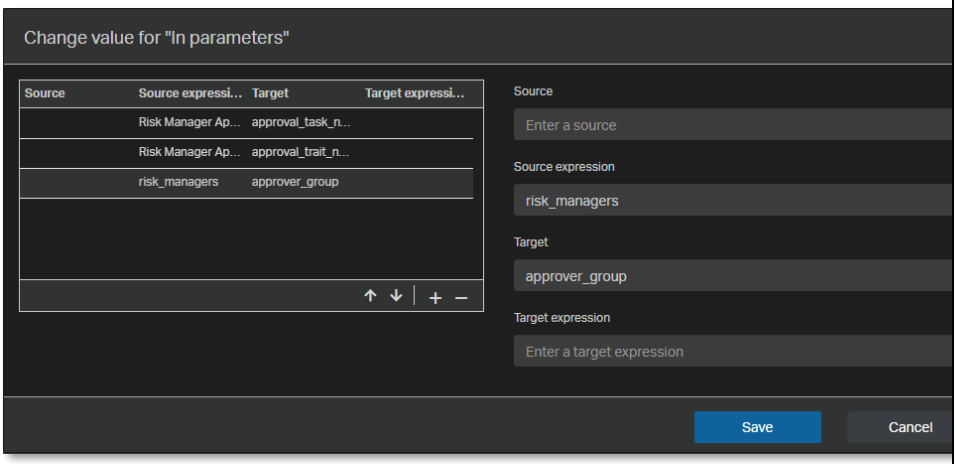
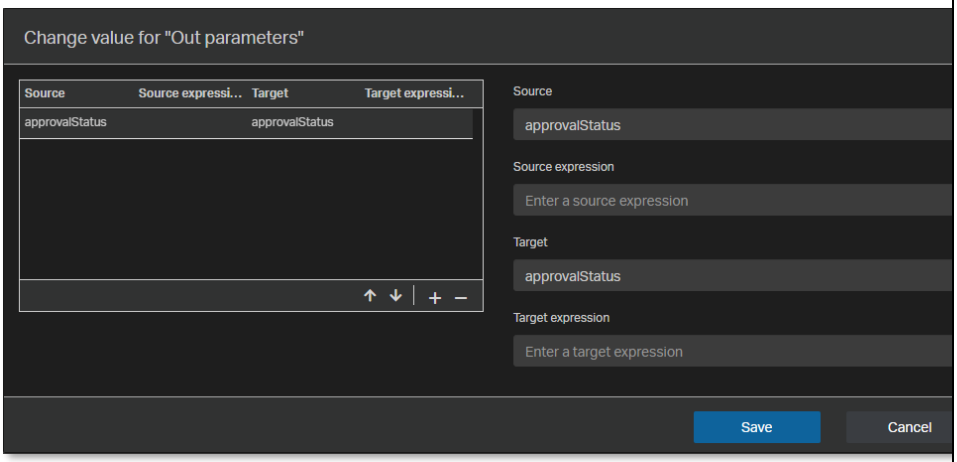


29. Drag and drop a third **Call activity** from the **palette** to the **canvas** next to the **Line Manager Approval call activity**.

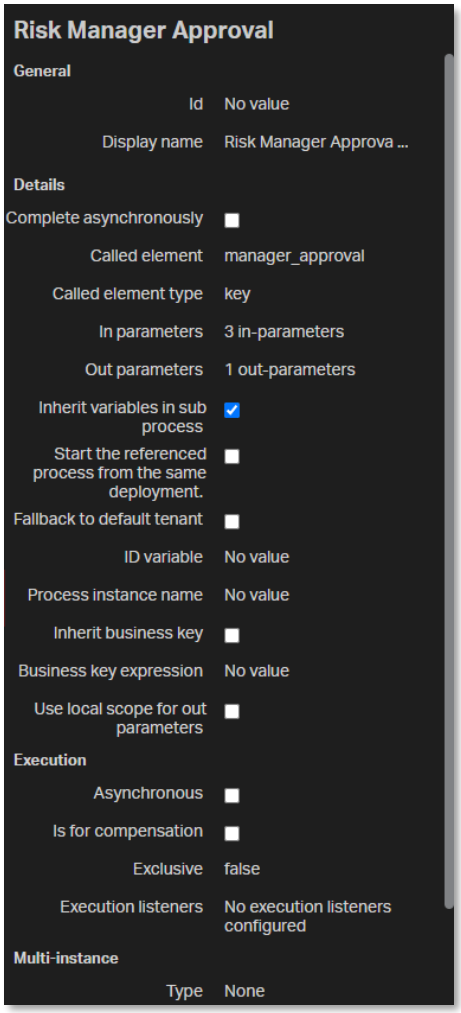


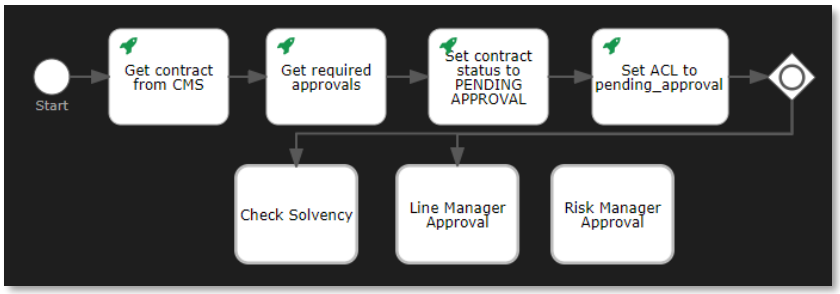
30. Fill the call activity attributes using the following details:

Attribute	Value
Display name	This call activity calls the **Manager Approval** workflow to perform the risk manager approval. Enter the following display name: `Risk Manager Approval`
Called element	`manager_approval`
Called element type	Select the **key** value for the called element type.

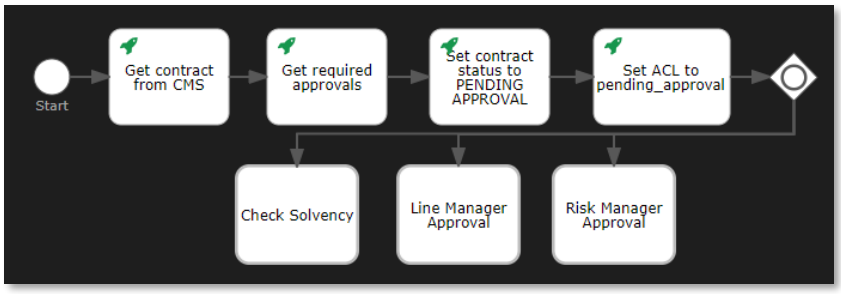
Attribute	Value
In parameters	For the Risk Manager Approval call activity, the values of the **approval_task_name**, **approval_trait_name**, and **approver_group** process variables of the called manager_approval sub-process need to be set. Click the **Add** button and enter `Risk Manager Approval` (literal text value) as the **Source expression** and `approval_task_name` as the **Target** process variable to map it to in the called sub-process. Click the **Add** button to add a second input parameter and enter `Risk Manager Approval` (literal text value) as the **Source expression** and `approval_trait_name` as the **Target** process variable to map it to in the called sub-process. Click the **Add** button to add a third input parameter and enter `risk_managers` (literal text value) as the **Source expression** and `approver_group` as the **Target** process variable to map it to in the called sub-process.  Click **Save** to confirm the in parameters.
Out parameters	For the Risk Manager Approval call activity, the **approvalStatus** process variable of the called (that is, the **Source**) manager_approval workflow must be mapped to a new **approvalStatus** process variable of the calling (that is the **Target**) workflow. Click the **Add** button and enter `approvalStatus` for both the **Source** and **Target** process variables.  Click **Save** to confirm the out parameters.

Attribute	Value
Inherit variables in sub process	Select the checkbox.





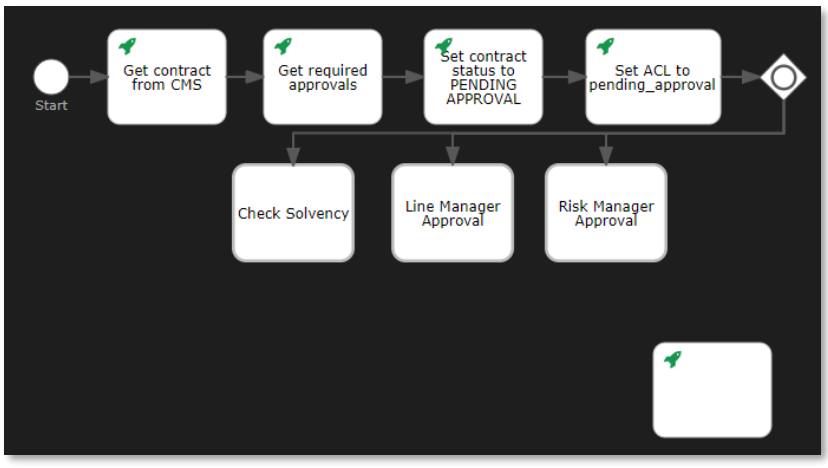
31. Select the **inclusive gateway**, drag a **sequence flow** to the **Risk Manager Approval call activity**, and add two bend-points to the new sequence flow.



32. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Flow condition	This flow must be selected by the inclusive gateway if a risk manager approval is required. This information is available from the **risk_manager_approval_required** process variable that was set after calling the Decision Service in the Get required approvals http task. Enter the following flow condition: `${risk_manager_approval_required}`

33. Drag and drop a fifth **Http task** from the **palette** to the **canvas** under the bottom corner of the **Risk Manager Approval call activity**.

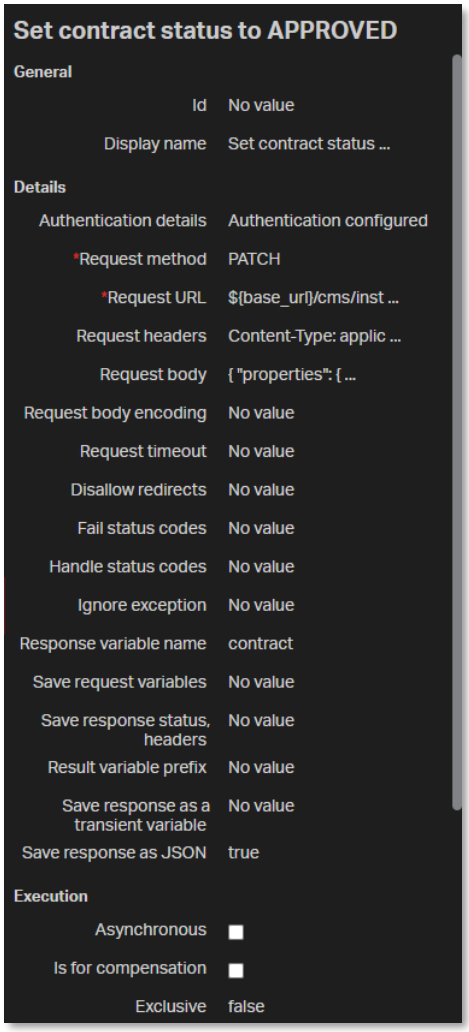


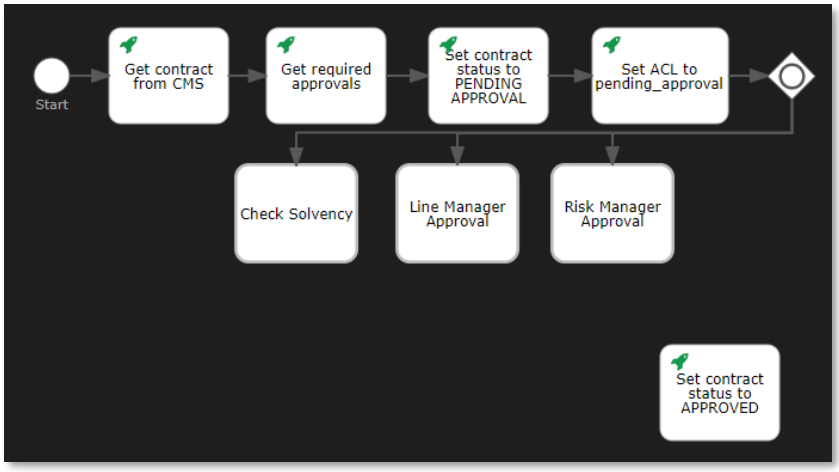
34. Fill the Http task attributes using the following details:

Attribute	Value
Display name	`Set contract status to APPROVED`
Authentication details	Select **Use current authentication token**.
Request method	Select the **PATCH** request method.
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	`Content-Type: application/json`

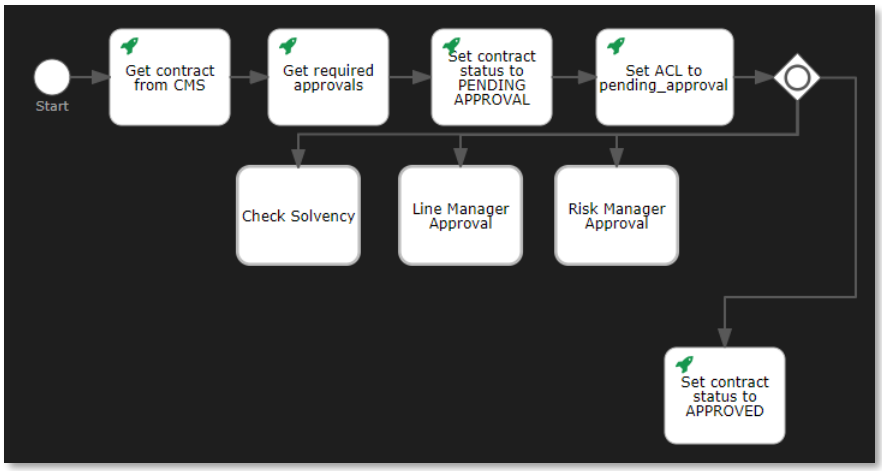
Request body	This task sets the **status** property of the contract to **APPROVED**.
	Enter the following request body:
	`{`
	`  "properties": {`
	`    "status": "APPROVED"`
	`  }`
	`}`

Response variable name	`contract`
Save response as JSON	`true`



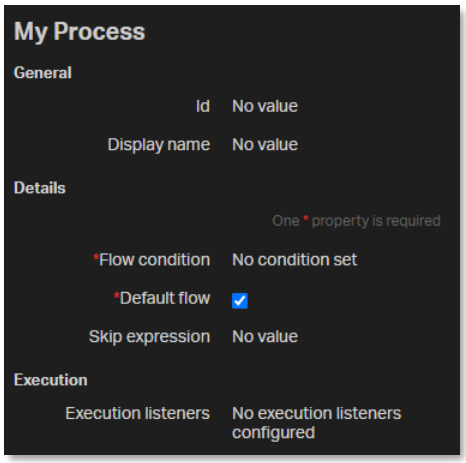


35. Select the **inclusive gateway**, drag a **sequence flow** to the **Set contract status to APPROVED http task**, and add three bend-points to the new sequence flow.



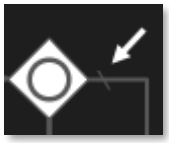
36. Select the new sequence flow and fill the sequence flow attributes using the following details:

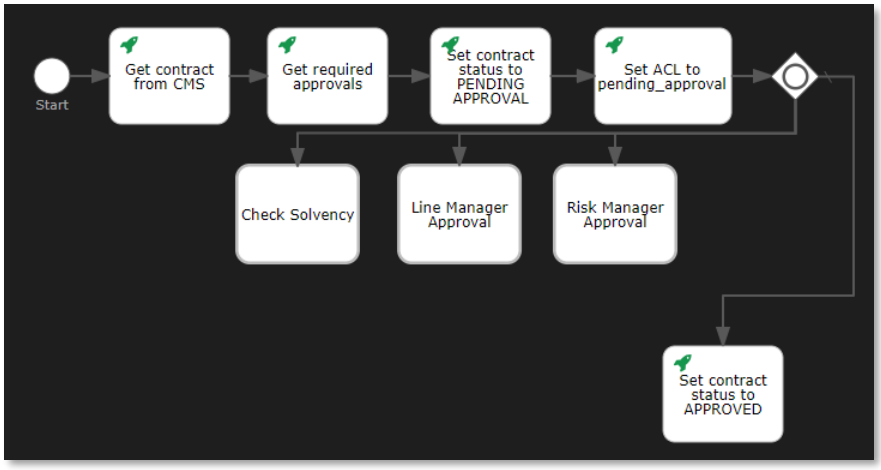
Attribute	Value
Default flow	Whether the sequence flow is the default flow to be selected from the connected gateway. The default flow gets selected when none of the other connected flows are (that is, none of the flow conditions of the other flows have been fulfilled). This flow is the default flow, and it results in setting the contract status directly to APPROVED when there are no required approvals. Select the checkbox.



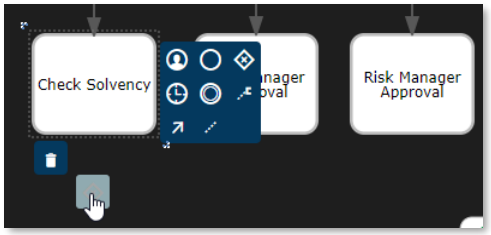
Note

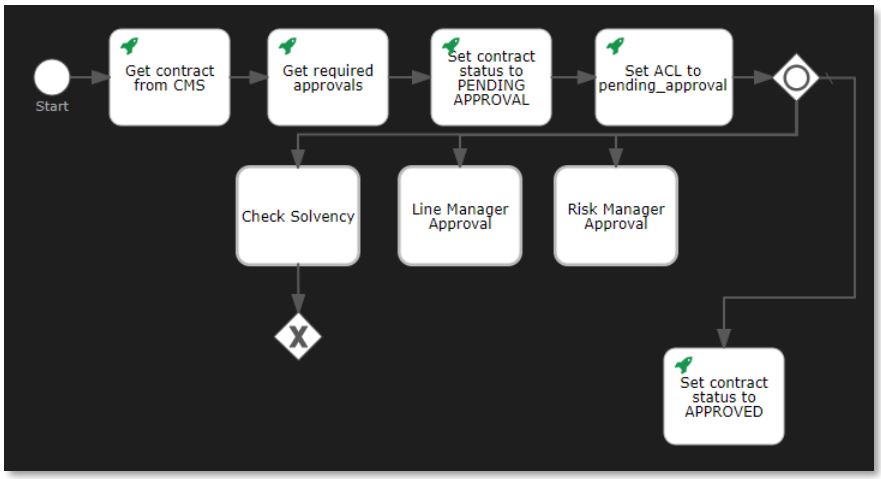
The default flow displays with an additional mark near the connected gateway:



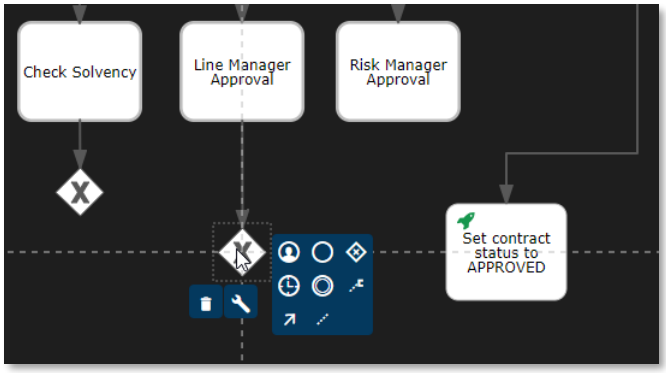


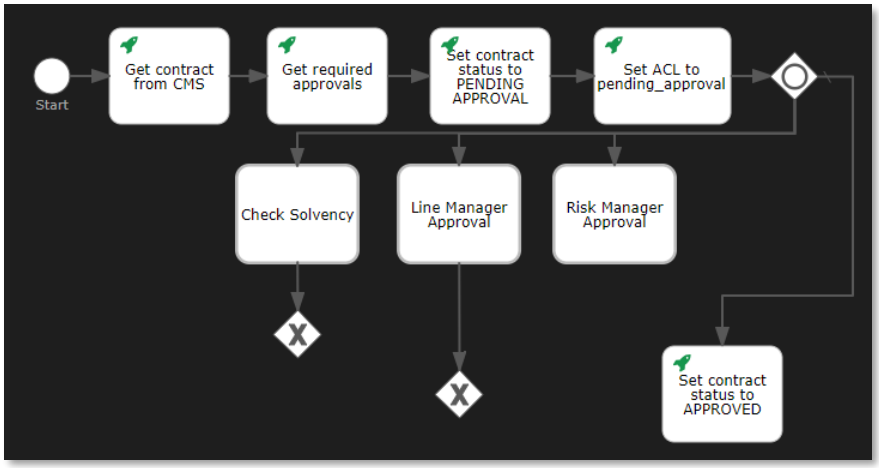
37. Select the **Check Solvency call activity** and drag and drop an **Exclusive gateway** right below it.



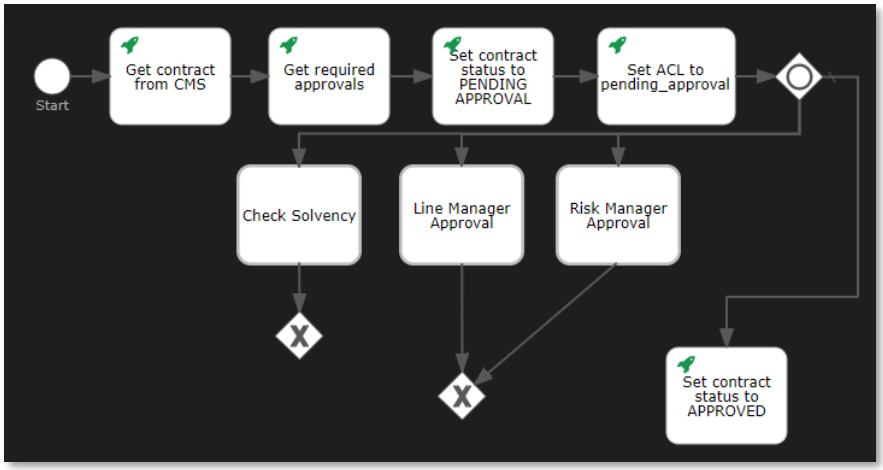


38. Select the **Line Manager Approval call activity** and drag and drop an **Exclusive gateway** below it, making sure it is aligned with the **Set contract status to APPROVED http task** to ensure readability.

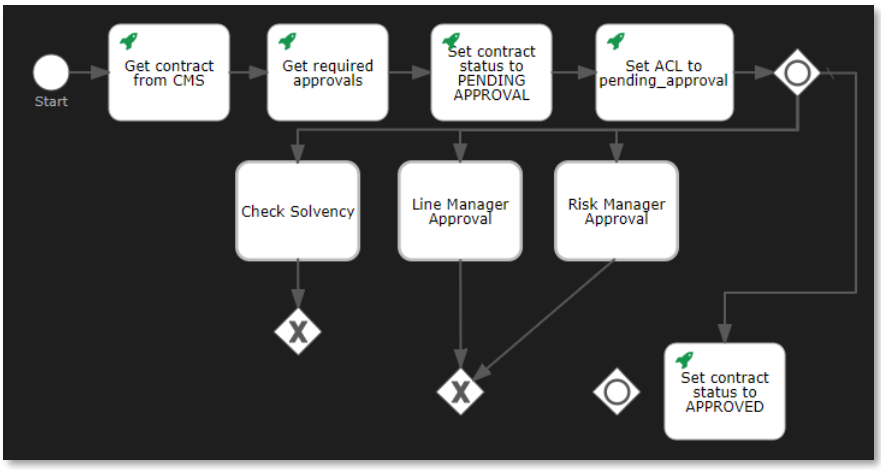




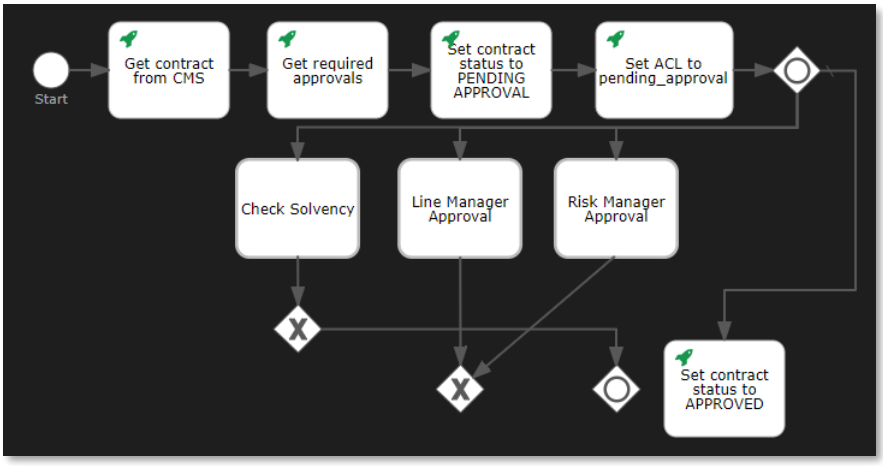
39. Select the **Risk Manager Approval call activity** and drag a **sequence flow** to the second **exclusive gateway** (that is, the one below the Line Manager Approval call activity).



40. Drag and drop a second **Inclusive gateway** to the **canvas** between the second **exclusive gateway** and the **Set contract status to APPROVED http task**. Make sure that the new inclusive gateway is close to the Set contract status to APPROVED http task, so that there is a maximum amount of space between the exclusive gateway and the inclusive gateway. This allows adding text to the connecting sequence flow.

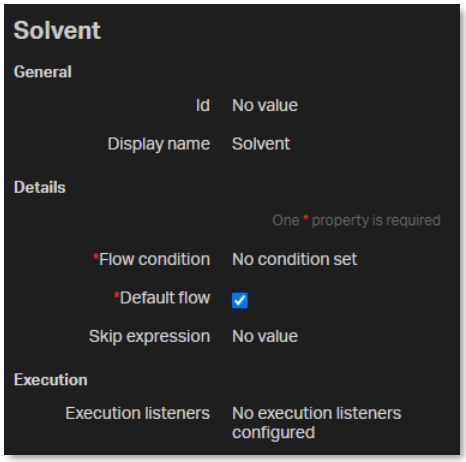


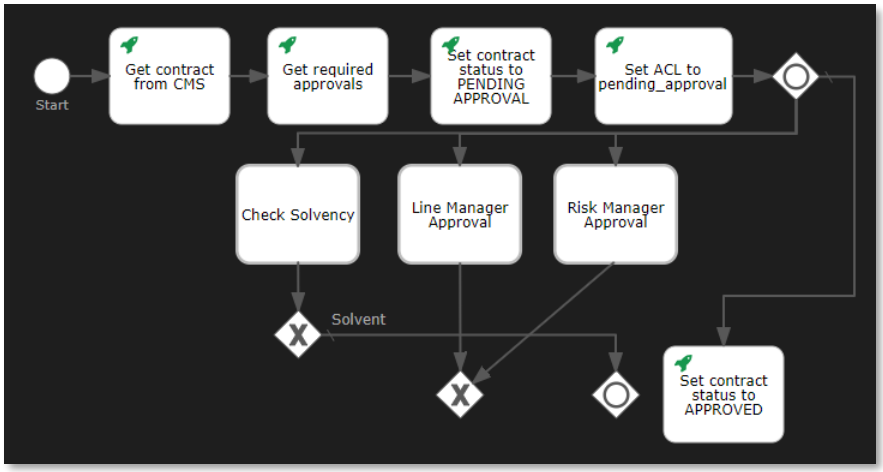
41. Select the first **exclusive gateway** (that is, the one below the Check Solvency call task), drag a **sequence flow** to the **Inclusive gateway**, and add one bend-point to the new sequence flow.



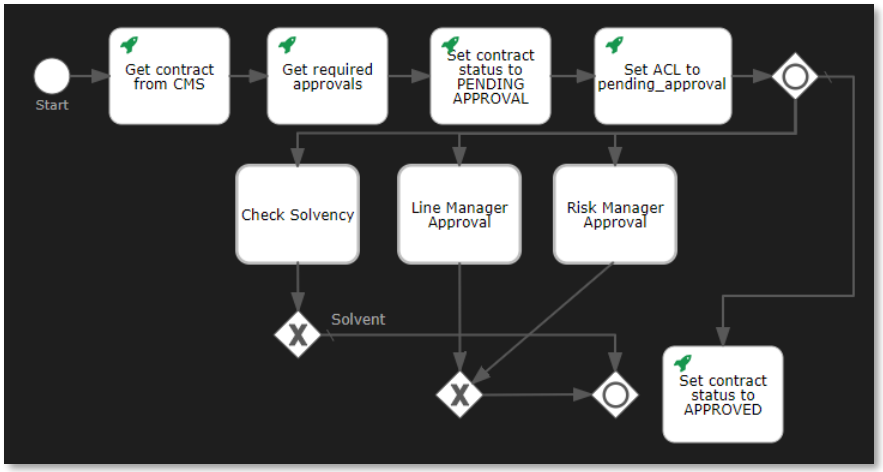
42. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	The display name of a sequence flow displays on the canvas and can be filled to improve readability (to show what the sequence flow stands for). This sequence flow is selected when the outcome of the Check Solvency call activity is that the contract approval requester is solvent, and more specifically that the **Solvent** process variable is equal to **True**. Enter the following display name: `Solvent`
Default flow	This flow is the default flow, and it moves the flow forward to the inclusive gateway that waits for all required approvals (that is, selected/active flows from the first inclusive gateway) to arrive. Select the checkbox.



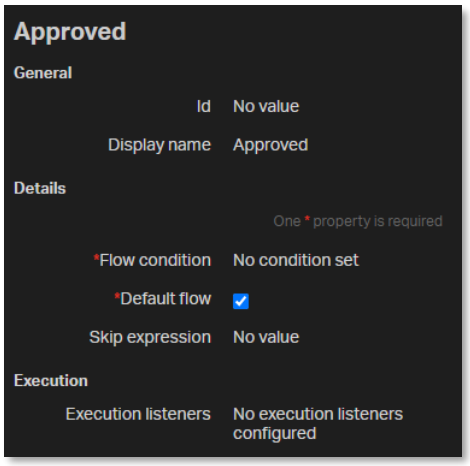


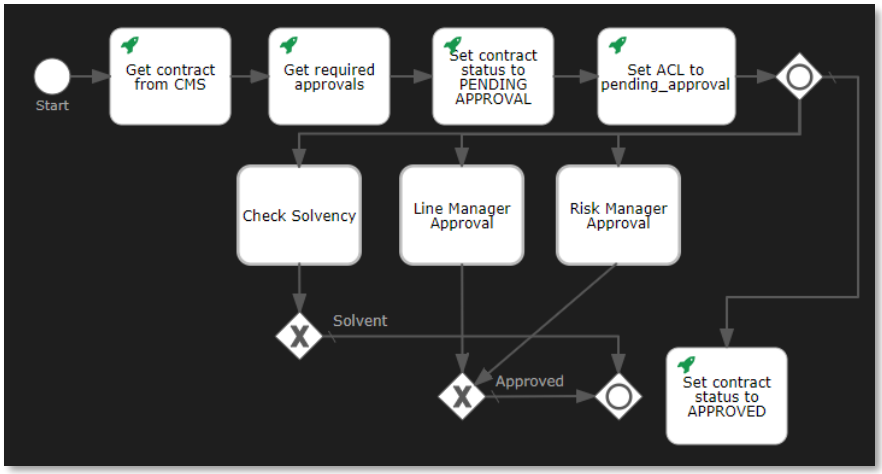
43. Select the second **exclusive gateway** (that is, the one below the Line Manager Approval call task) and drag a **sequence flow** to the **inclusive gateway**.



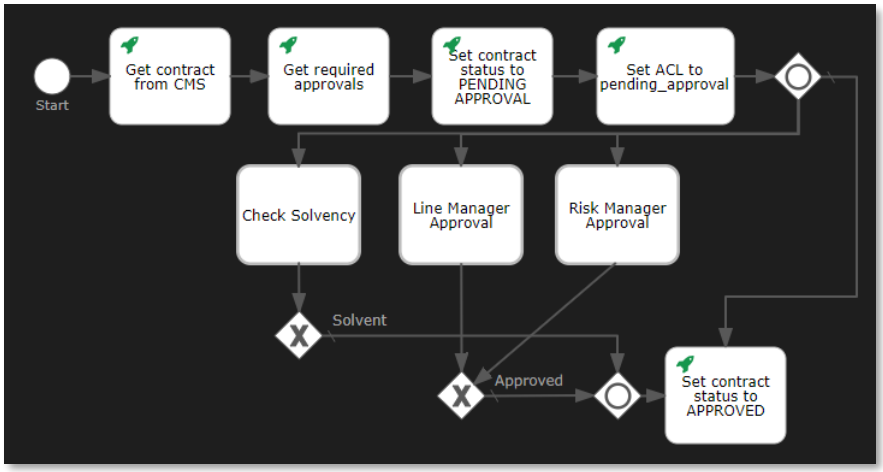
44. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	This sequence flow is selected when the outcome of either the Line Manager Approval call activity or the Risk Manager Approval call activity is that the approver has approved the contract, and more specifically that the **approvalStatus** process variable is not equal to **rejected** or **expired**. Enter the following display name: `Approved`
Default flow	This flow is the default flow, and it moves the flow forward to the inclusive gateway that waits for all required approvals (that is, selected/active flows from the first inclusive gateway) to arrive. If both manager approver call activities are selected, the inclusive gateway requires this flow to be selected for each of them. Select the checkbox.

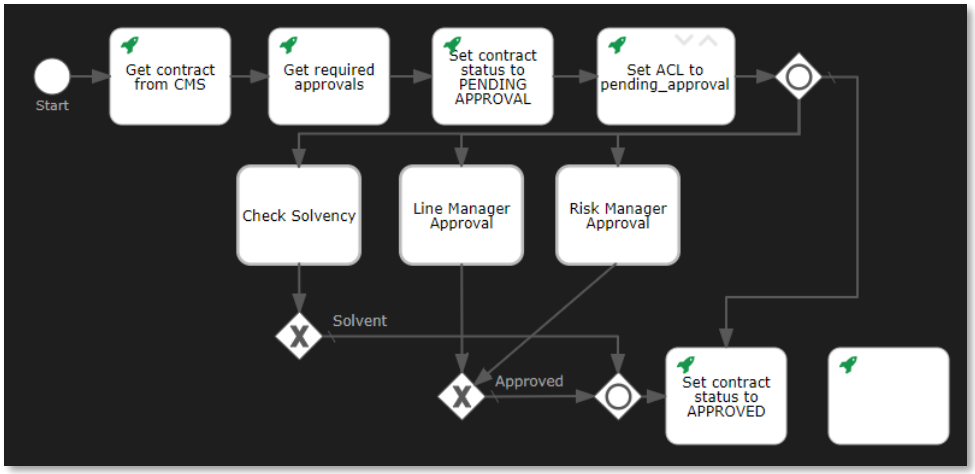




45. Select the second **inclusive gateway** (that is, the one next to the Set contract status to APPROVED http task) and drag a **sequence flow** to the **Set contract status to APPROVED http task**.



46. Drag and drop a sixth **Http task** from the **palette** to the **canvas** next to the **Set contract status to APPROVED http task**.

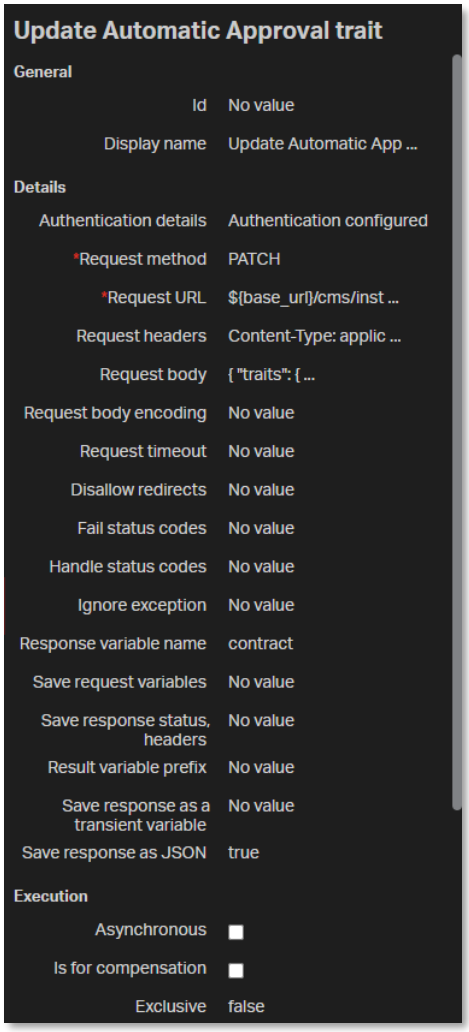


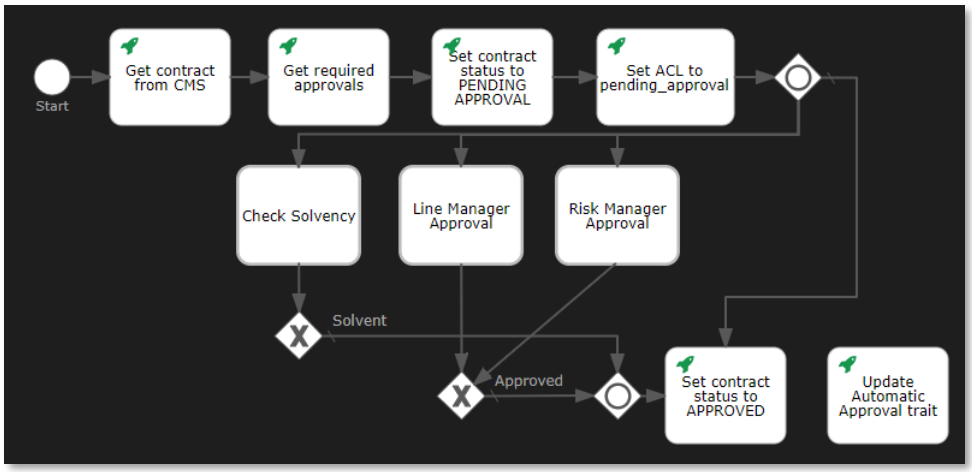
47. Fill the Http task attributes using the following details:

Attribute	Value
Display name	`Update Automatic Approval trait`
Authentication details	Select **Use current authentication token**.
Request method	Select the **PATCH** request method.
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	`Content-Type: application/json`

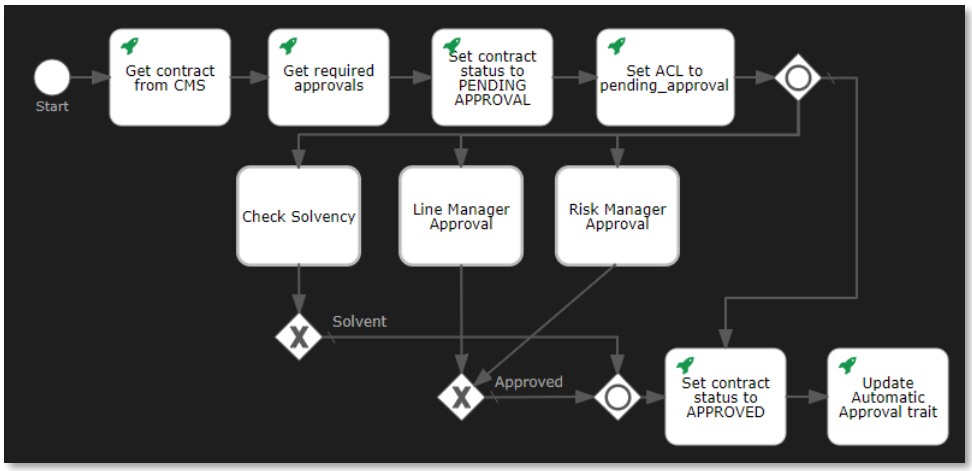
Request body	This task sets the **has_been_granted**, **approver**, **approver_role**, and **approval_date** properties for the **Automatic Approval** contract approval trait.
	Enter the following request body:
	`{`
	`  "traits": {`
	`    "ca_approval": {`
	`      "Automatic Approval": {`
	`        "has_been_granted": true,`
	`        "approver": "SYSTEM",`
	`        "approver_role": "Automatic Approval",`
	`        "approval_date": "${contract.update_time}"`
	`      }`
	`    }`
	`  }`
	`}`

Response variable name	`contract`
Save response as JSON	`true`

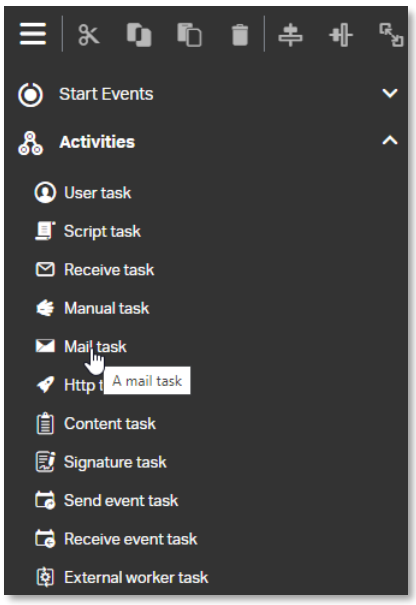




48. Select the **Set contract status to APPROVED http task** and drag a **sequence flow** to the **Update Automatic Approval trait http task**.



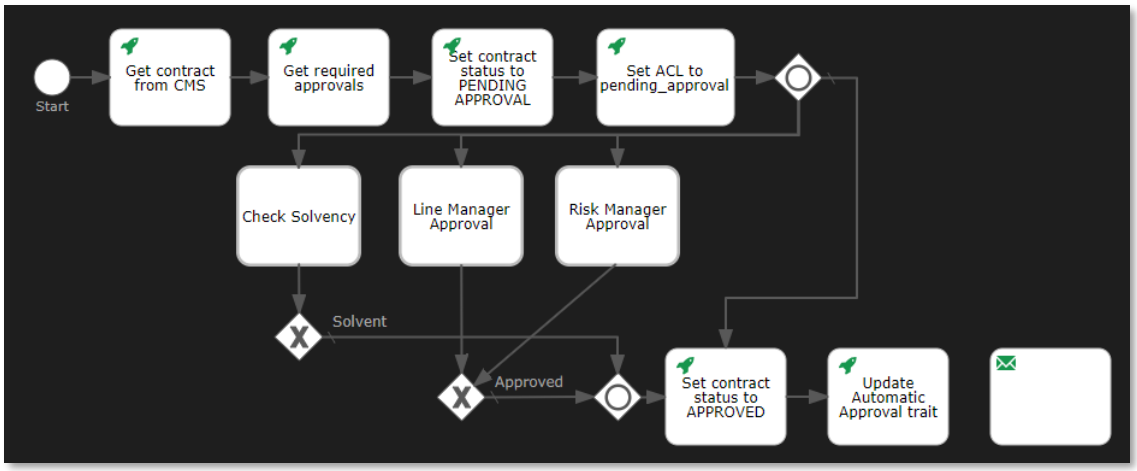
49. In the **palette**, find **Activities > Mail task**.



Note

The mail task sends emails to one or more recipients. It supports normal email features, such as cc lists, bcc lists, and HTML content.

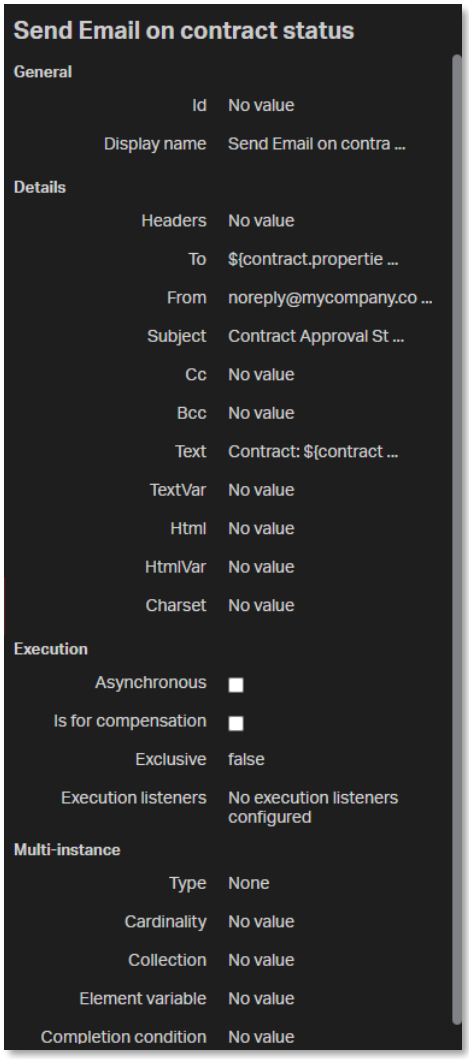
50. Drag and drop the **Mail task** to the **canvas** next to the **Update Automatic Approval trait http task**.

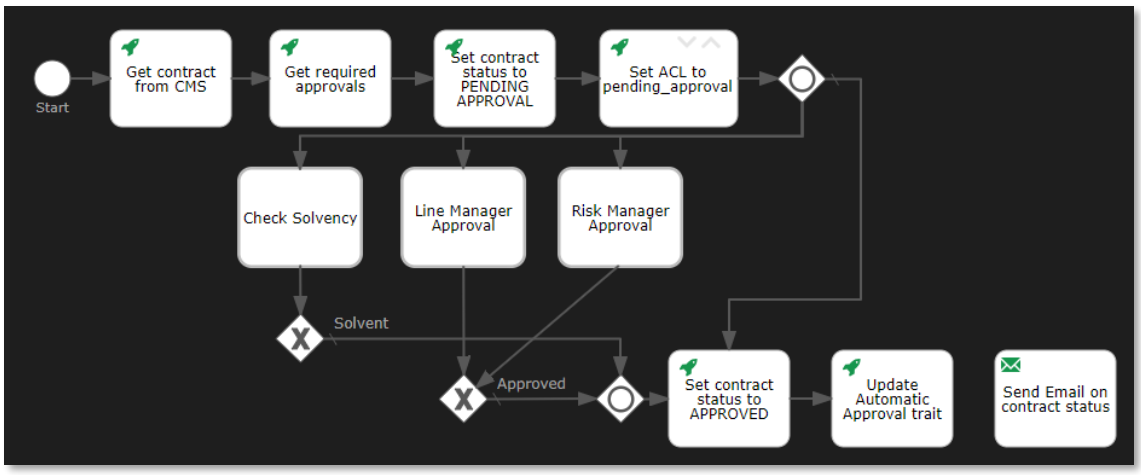


51. Fill the mail task attributes using the following details:

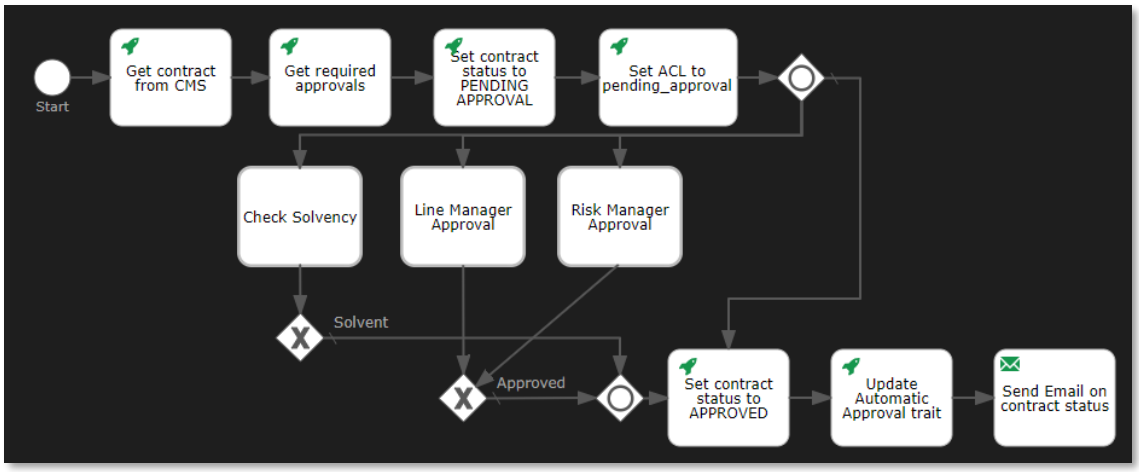
Attribute	Value
Display name	`Send Email on contract status`
To	The recipient of the email. Specify multiple recipients in a comma-separated list. This can be an expression. This task sends the email to the requester of the contract approval. Enter the following recipient email: `${contract.properties.requester_email}`
From	The sender's email address. This can be an expression. Enter the following sender email: `noreply@mycompany.com`
Subject	The subject of the email. This can be an expression. Enter the following email subject: `Contract Approval Status`

Text	The text content of the email. This can be an expression.
	The email text details the name of the contract and its final status (that is, the result of the approvals).
	Enter the following email text:
	`Contract: ${contract.name}`
	`Status: ${contract.properties.status}`

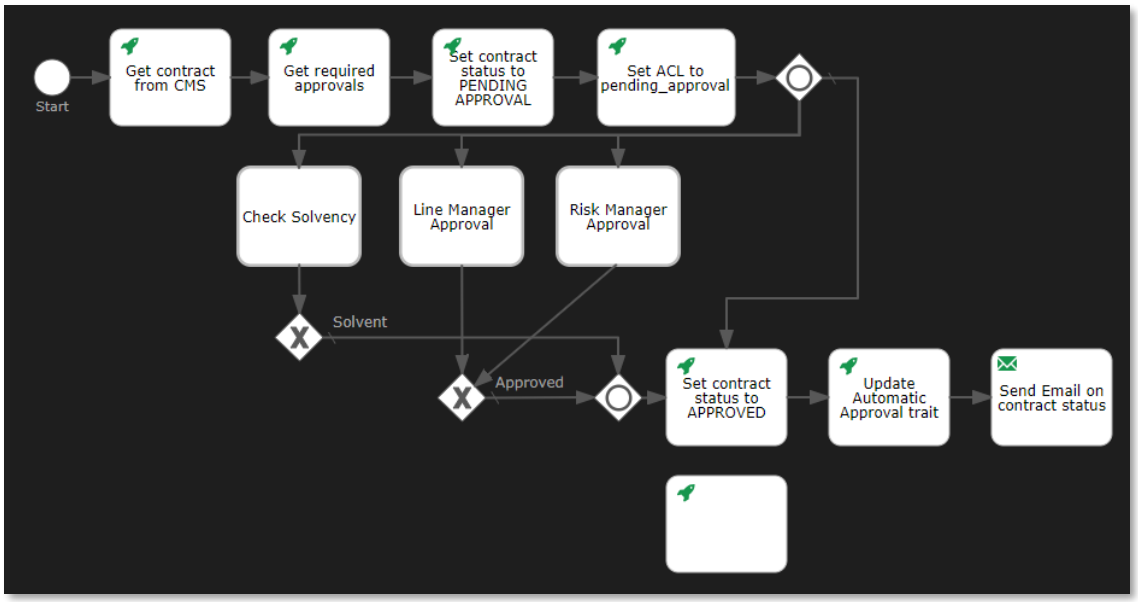




52. Select the **Update Automatic Approval trait http task** and drag a **sequence flow** to the **Send Email on contract status email task**.



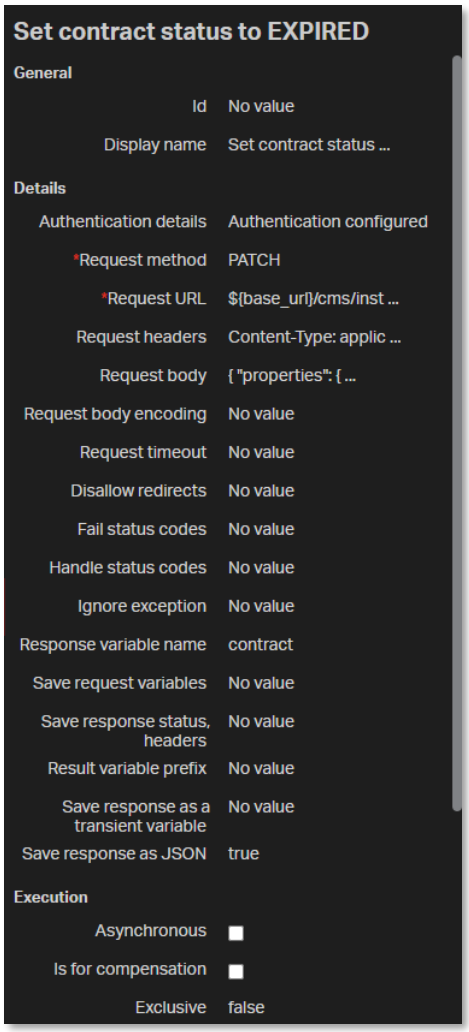
53. Drag and drop a seventh **Http task** from the **palette** to the **canvas** under the **Set contract status to APPROVED http task**.

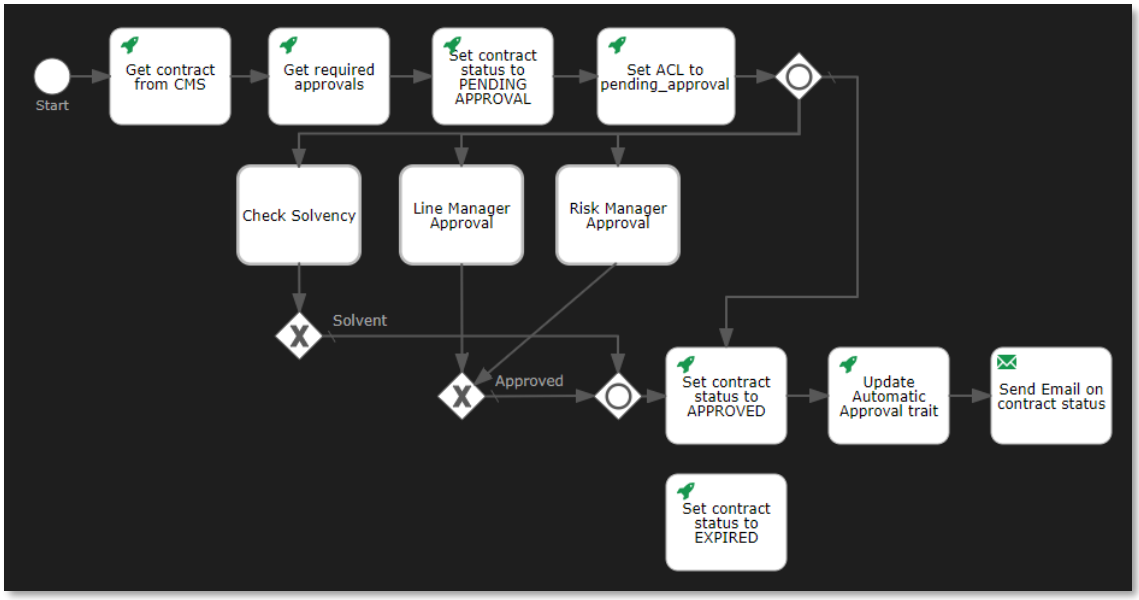


54. Fill the Http task attributes using the following details:

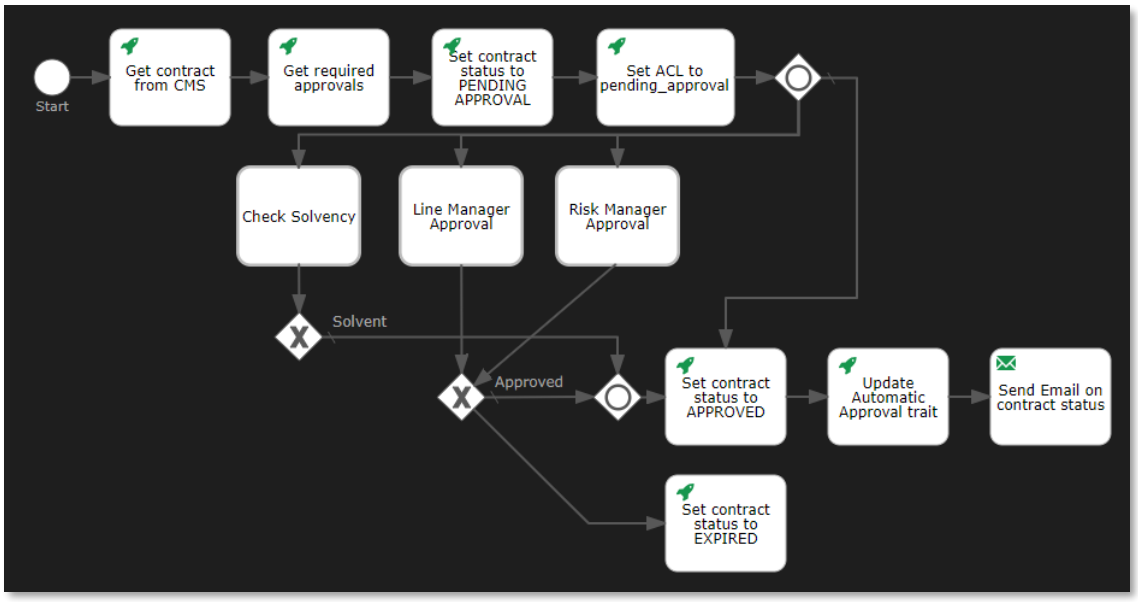
Attribute	Value
Display name	`Set contract status to EXPIRED`
Authentication details	Select **Use current authentication token**.
Request method	Select the **PATCH** request method.
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	`Content-Type: application/json`

Attribute	Value
Request body	This task sets the **status** property of the contract to **EXPIRED**. Enter the following request body: `{ "properties": { "status": "EXPIRED" } }`
Response variable name	`contract`
Save response as JSON	`true`



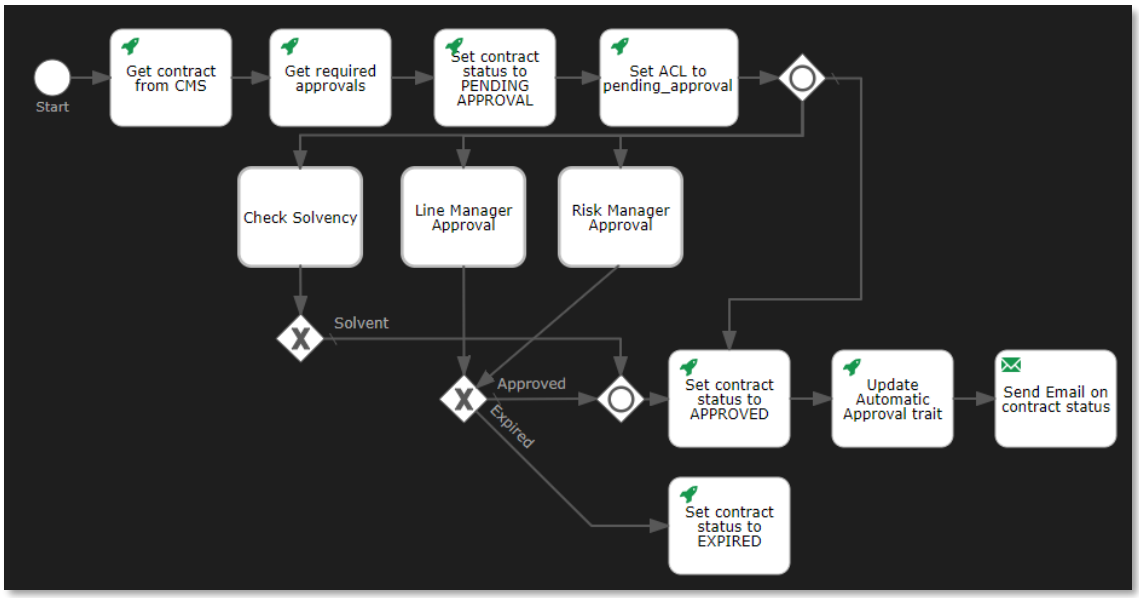


55. Select the second **exclusive gateway** (that is, the one below the Line Manager Approval call task), drag a **sequence flow** to the **Set contract status to EXPIRED http task**, and add 1 bend-point to the new sequence flow.

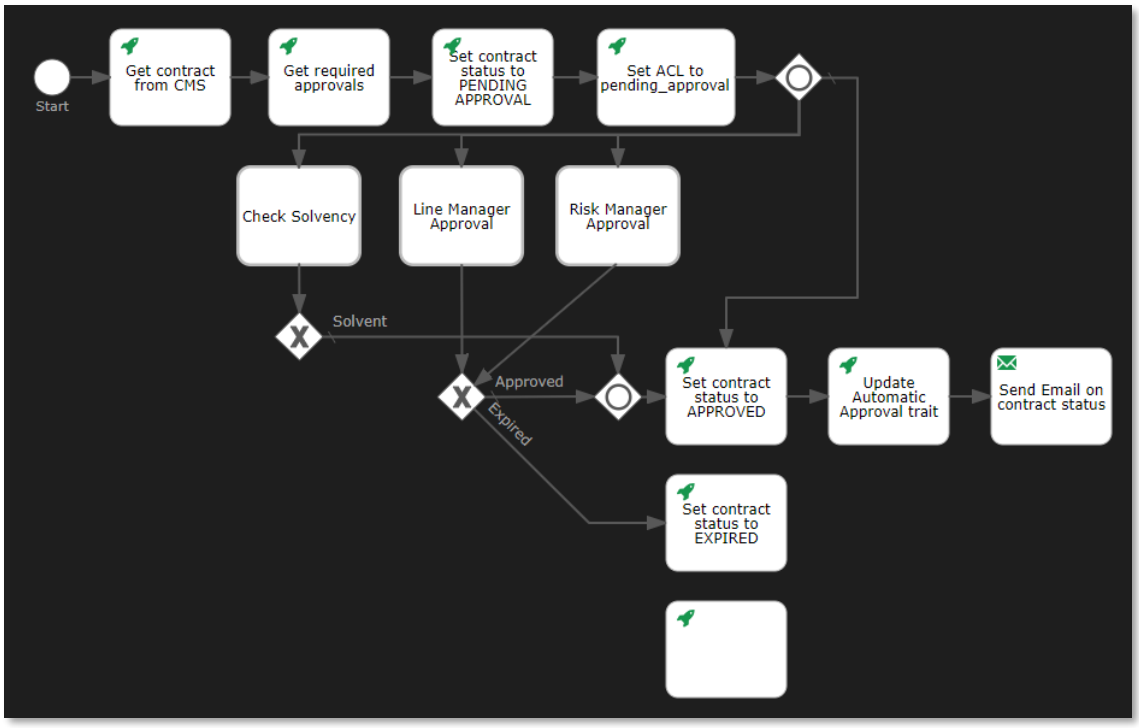


56. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	`Expired`
Flow condition	This sequence flow is selected when the outcome of either the Line Manager Approval call activity or the Risk Manager Approval call activity is that the approval task has expired, and more specifically that the **approvalStatus** process variable is equal to **expired**. Enter the following flow condition: `${approvalStatus == "expired"}`



57. Drag and drop an eighth **Http task** from the **palette** to the **canvas** under the **Set contract status to EXPIRED http task**.



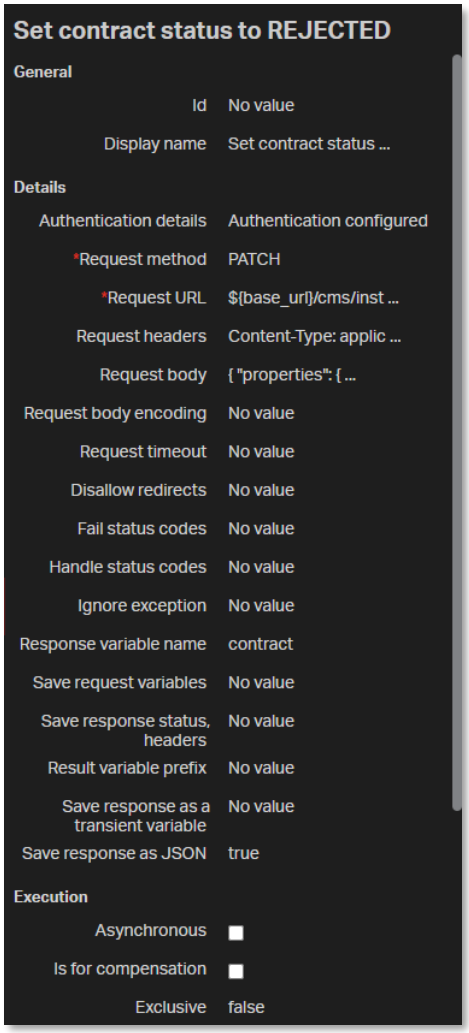
58. Fill the Http task attributes using the following details:

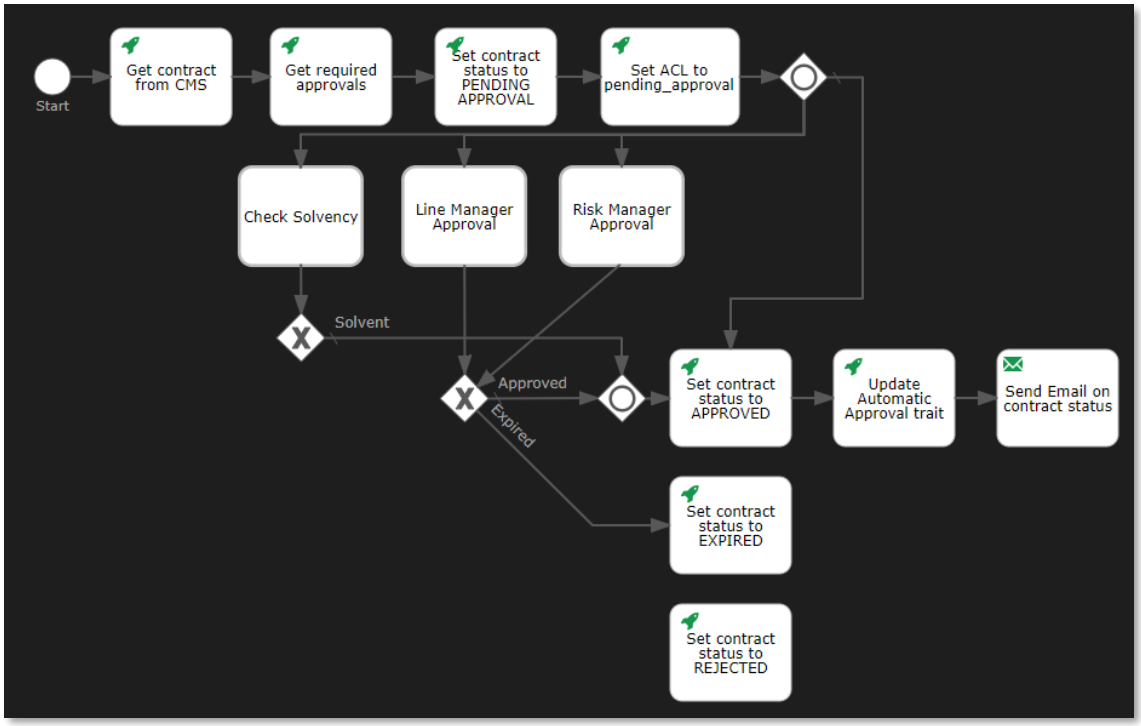
Attribute	Value
Display name	`Set contract status to REJECTED`
Authentication details	Select **Use current authentication token**.
Request method	Select the **PATCH** request method.
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}`
Request headers	`Content-Type: application/json`

Request body	This task sets the **status** property of the contract to **REJECTED**.
	Enter the following request body:
	`{`
	`  "properties": {`
	`  "status": "REJECTED"`
	`  }`
	`}`

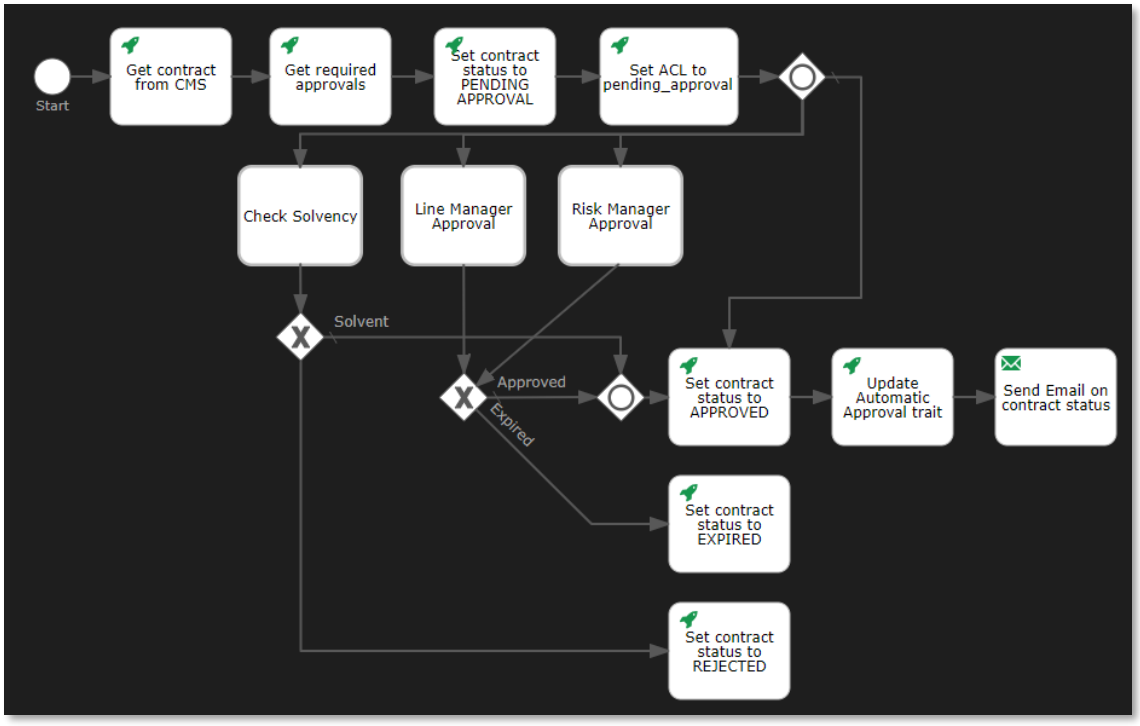
Response variable name	`contract`

Attribute	Value
Save response as JSON	`true`



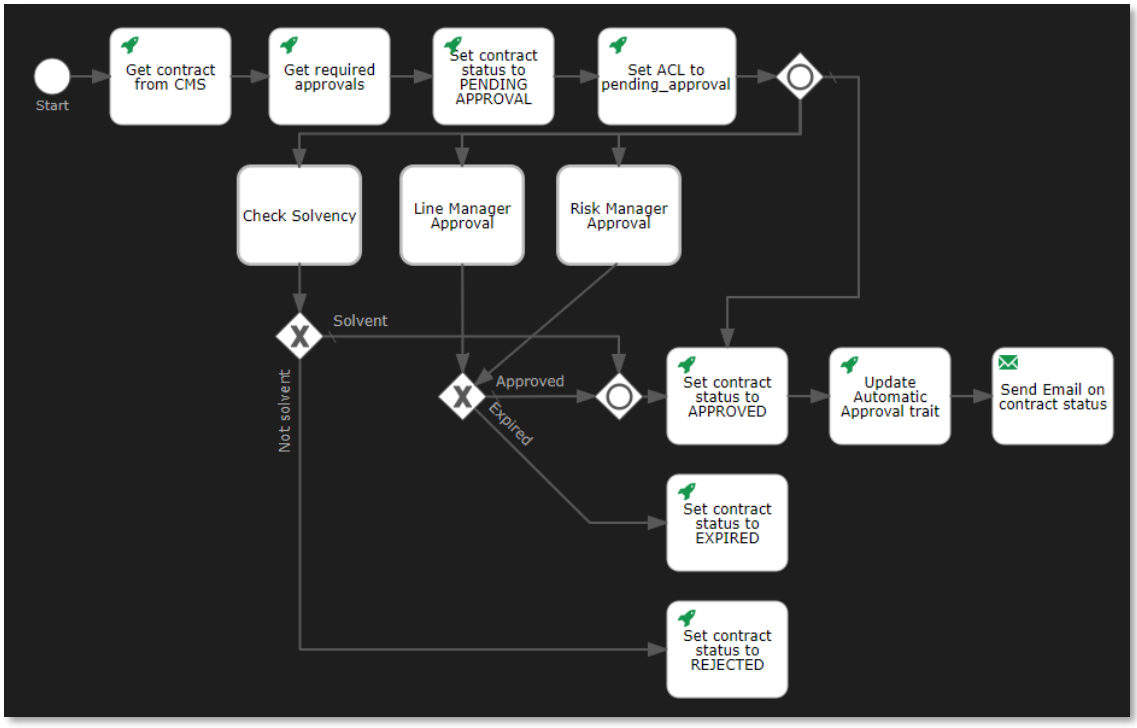


59. Select the first **exclusive gateway** (that is, the one below the Solvency Check call task), drag a **sequence flow** to the **Set contract status to REJECTED http task**, and add 1 bend-point to the new sequence flow.

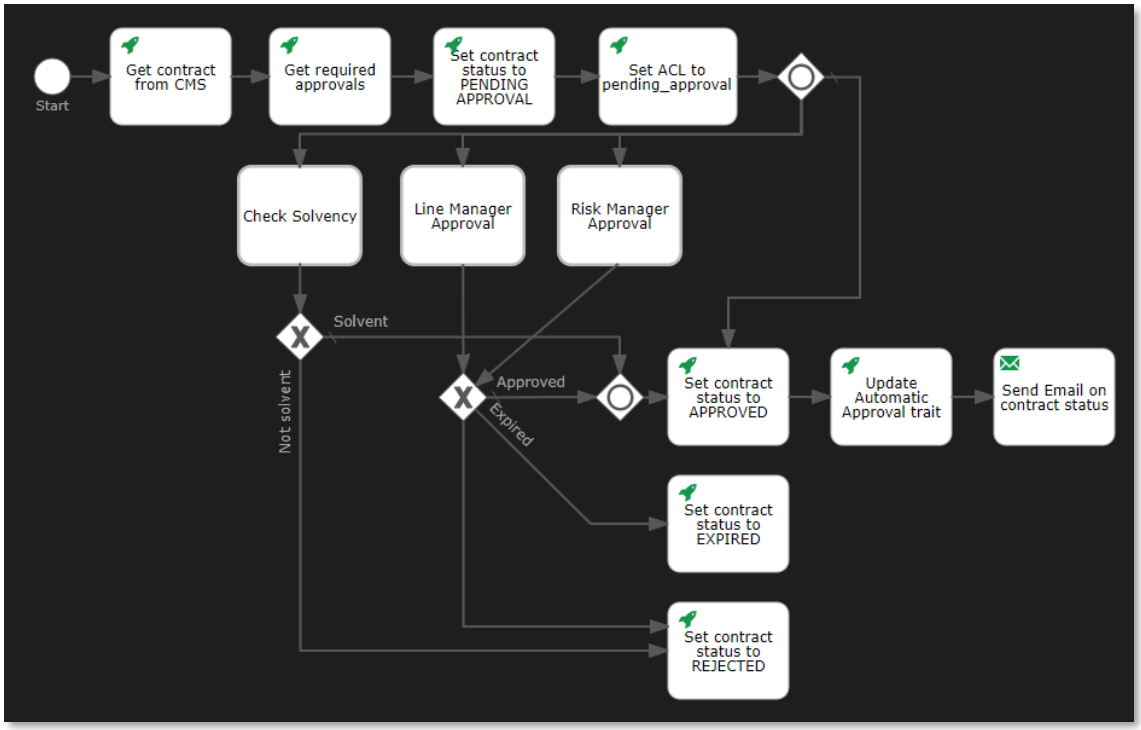


60. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	`Not solvent`
Flow condition	This sequence flow is selected when the outcome of the Check Solvency call activity is that the contract approval requester is not solvent, and more specifically that the **solvent** process variable is not equal to **true**. Enter the following flow condition: `${!solvent}`

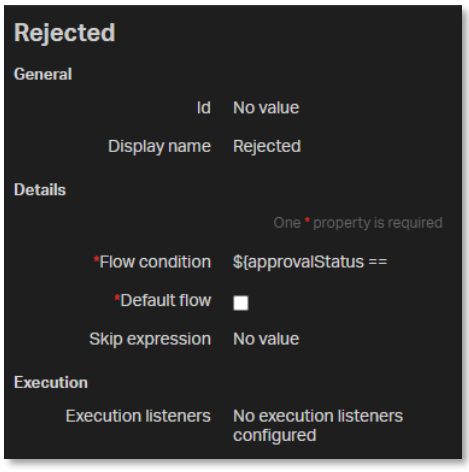


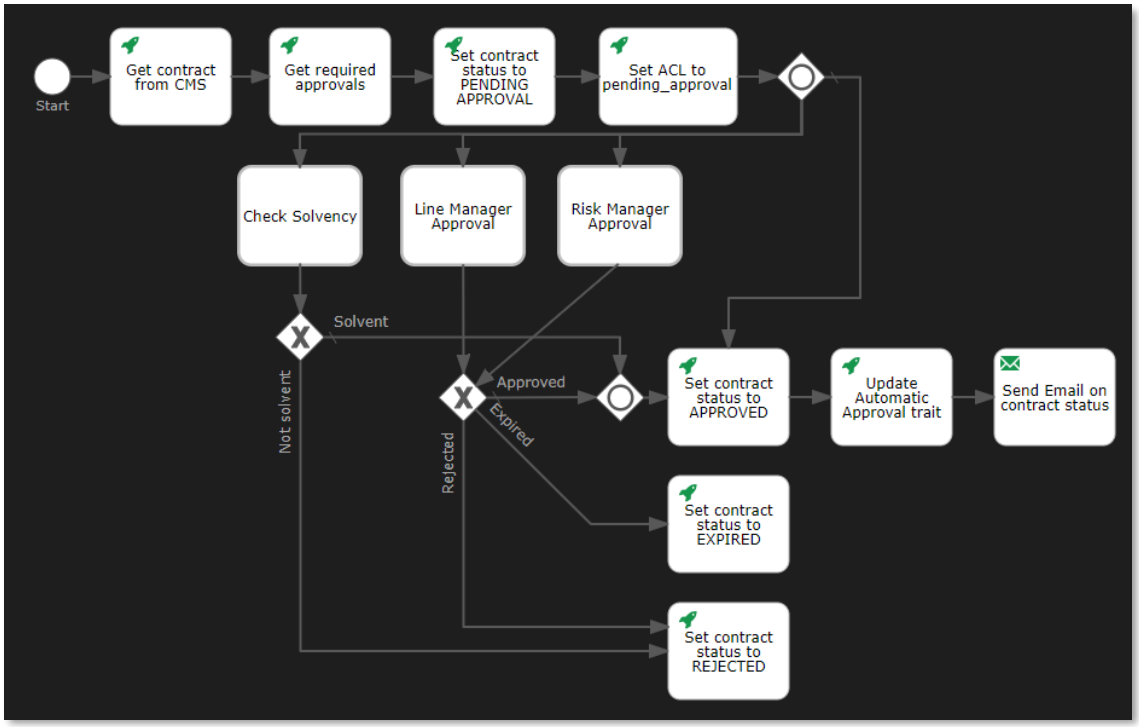
61. Select the second **exclusive gateway** (that is, the one below the Line Manager Approval call task), drag a **sequence flow** to the **Set contract status to REJECTED http task**, and add 1 bend-point to the new sequence flow.



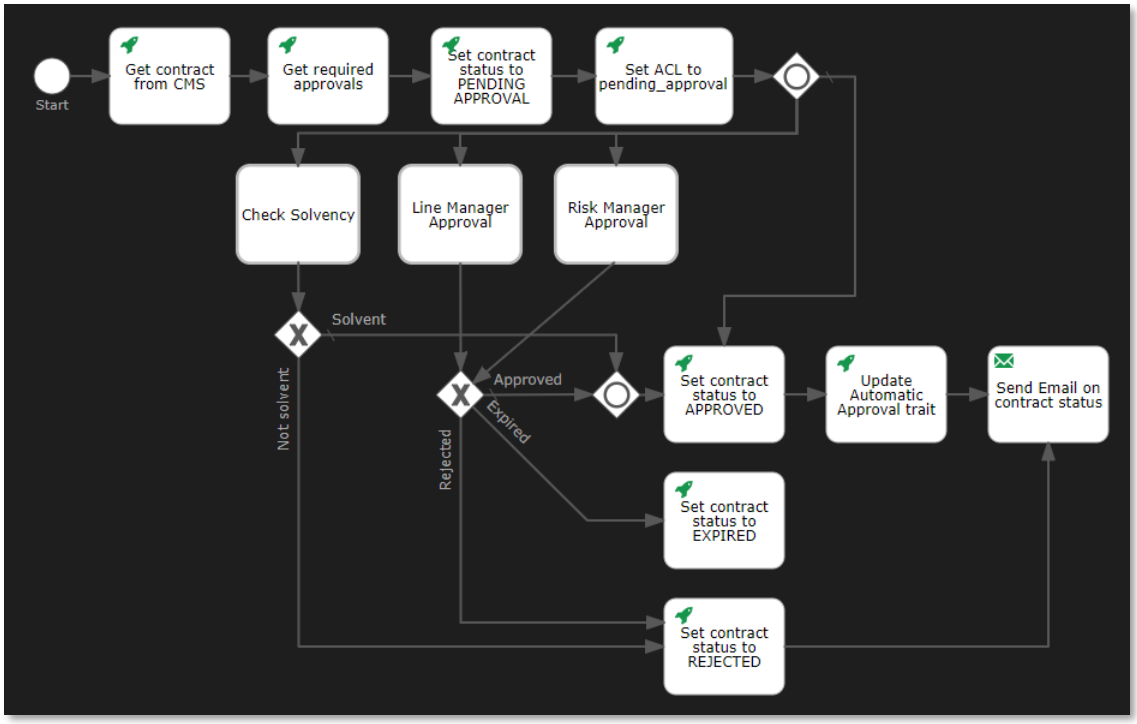
62. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	`Rejected`
Flow condition	This sequence flow is selected when the outcome of either the Line Manager Approval call activity or the Risk Manager Approval call activity is that the approval is rejected, and more specifically that the **approvalStatus** process variable is equal to **rejected**. Enter the following flow condition: `${approvalStatus == "rejected"}`

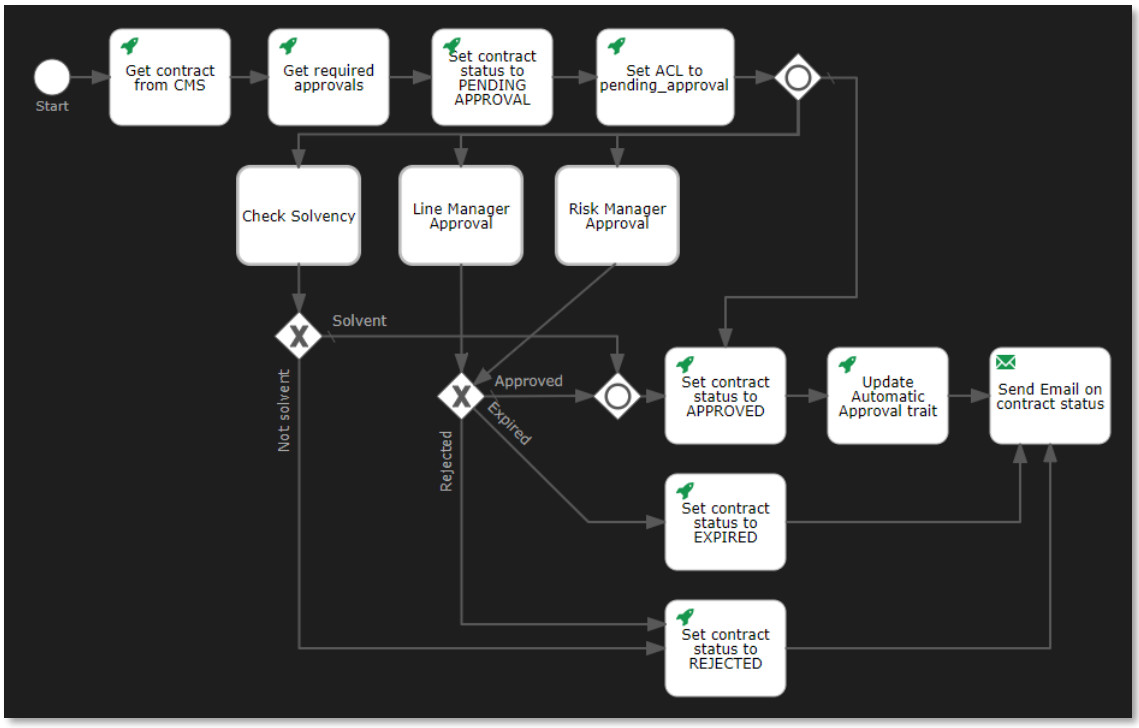




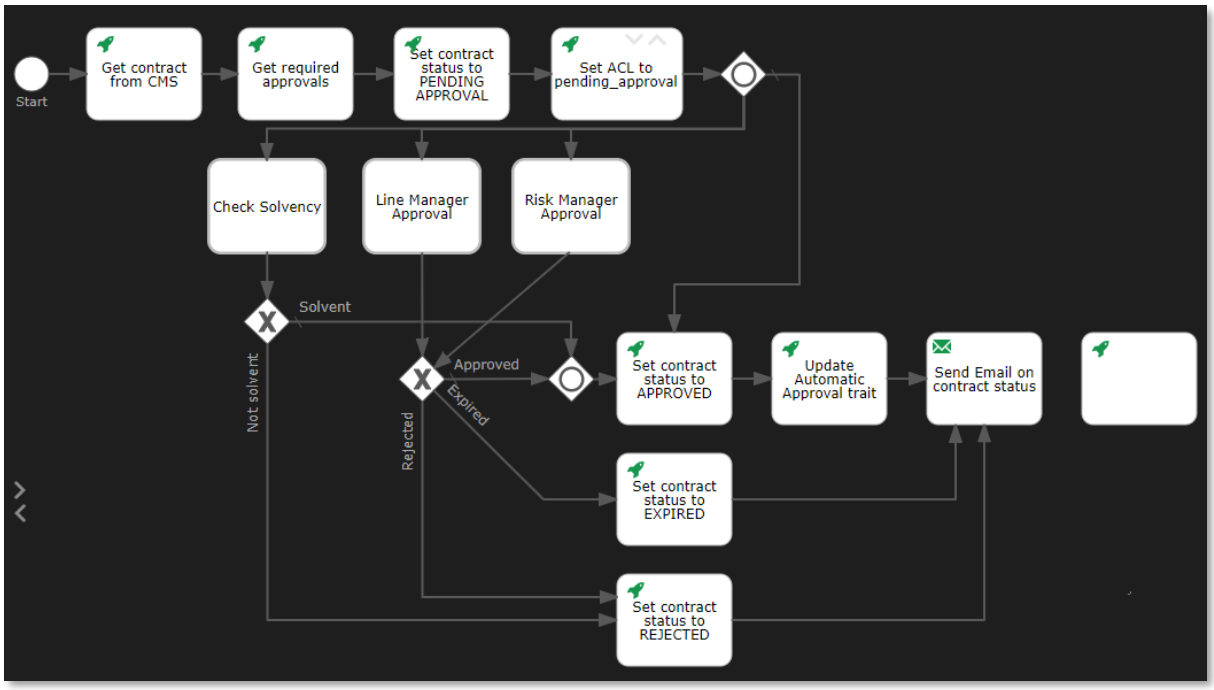
63. Select the **Set contract status to REJECTED http task**, drag a **sequence flow** to the **Send Email on contract status email task**, and add 1 bend-point to the new sequence flow.



64. Select the **Set contract status to EXPIRED http task**, drag a **sequence flow** to the **Send Email on contract status email task**, and add 1 bend-point to the new sequence flow.



65. Drag and drop a ninth **Http task** from the **palette** to the **canvas** next to the **Send Email on contract status mail task**.

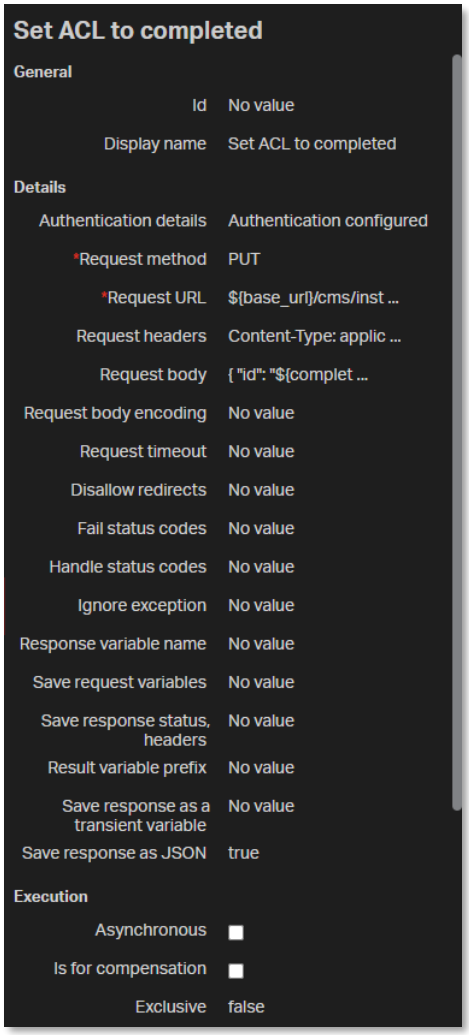


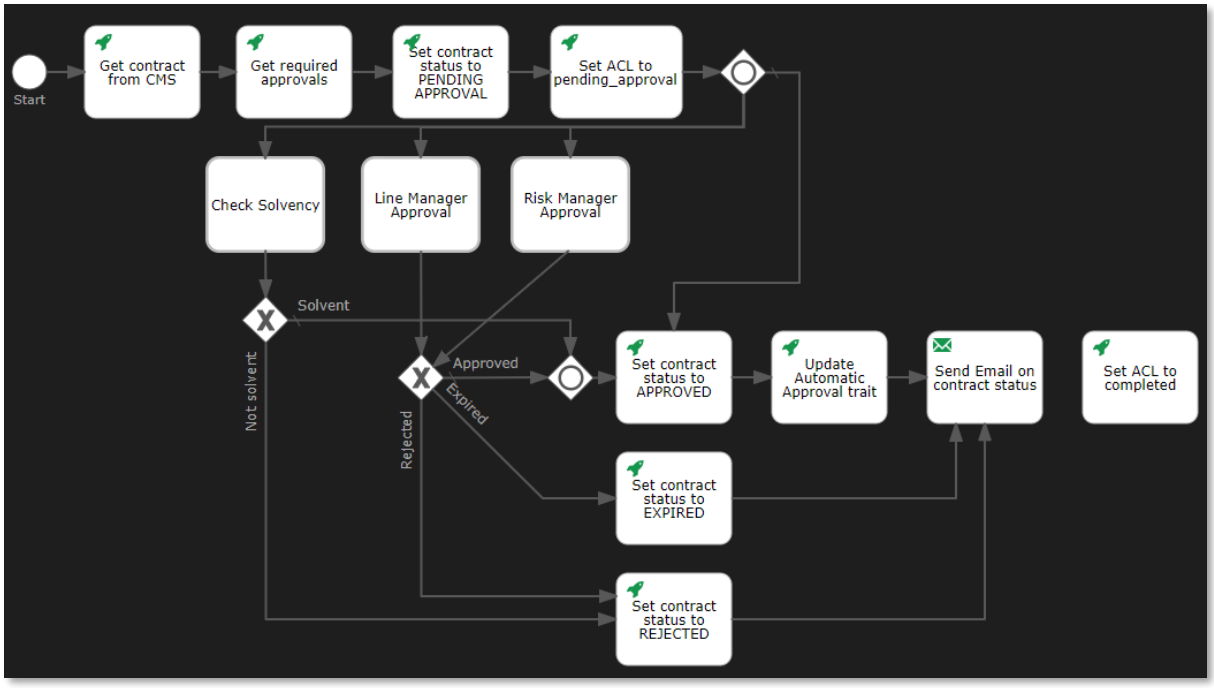
66. Fill the Http task attributes using the following details:

Attribute	Value
Display name	`Set ACL to completed`
Authentication details	Select **Use current authentication token**.
Request method	Select the **PUT** request method.
Request URL	`${base_url}/cms/instances/file/ca_contract/${contract_id}/acl`
Request headers	`Content-Type: application/json`

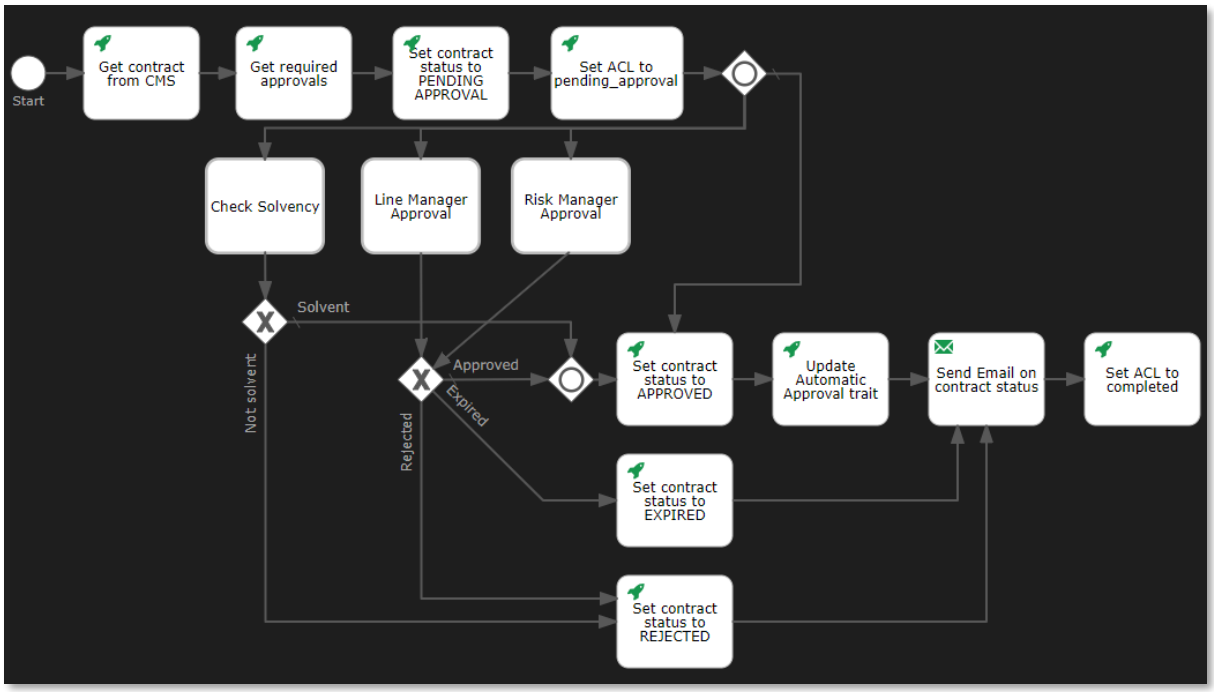
Request body	This task sets the ACL of the contract by using the **completed_acl_id** process input parameter as the **id** of the ACL to apply.
	Enter the following request body:
	`{`
	`  "id": "${completed_acl_id}"`
	`}`

Save response as JSON	`true`

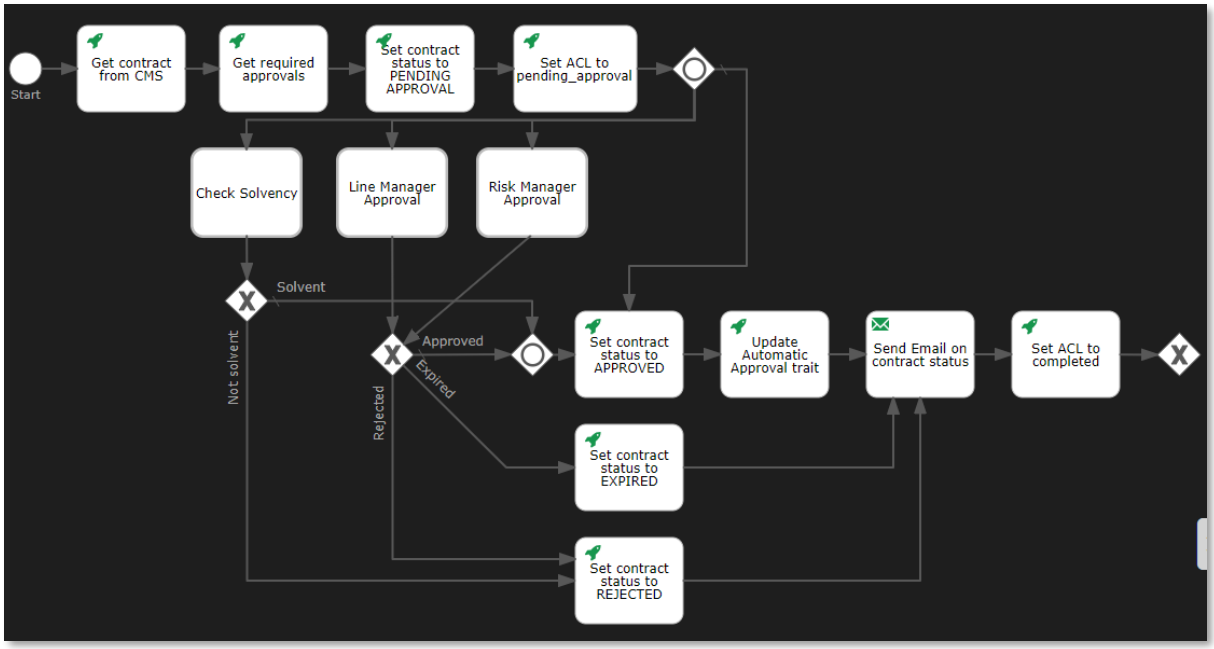




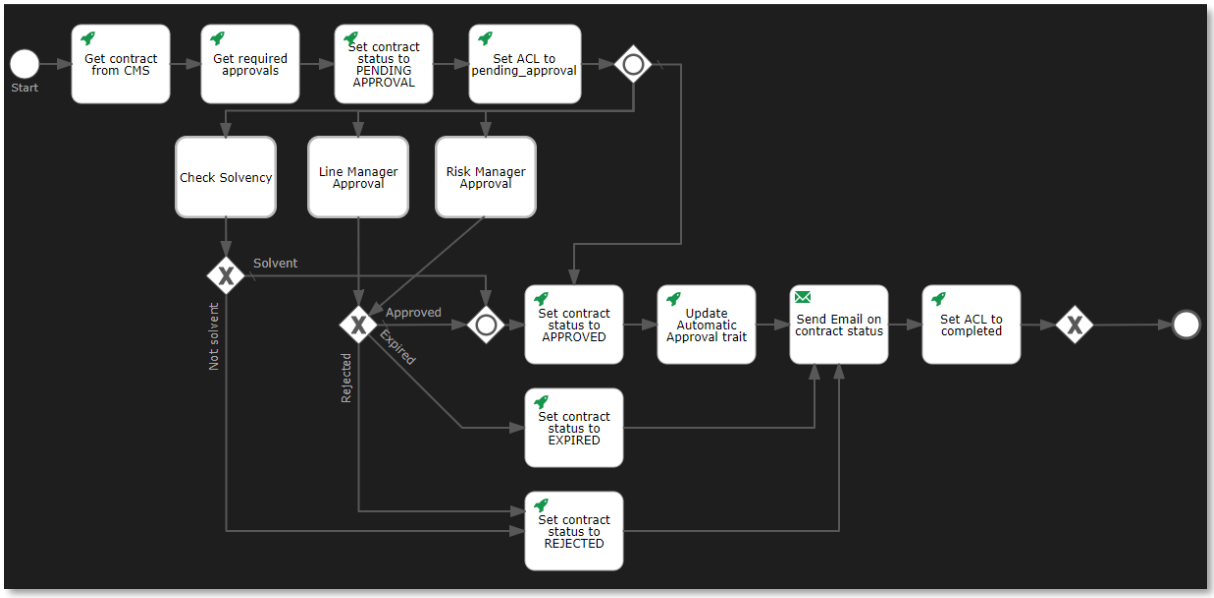
67. Select the **Send Email on contract status mail task** and drag a **sequence flow** to the **Set ACL to completed http task**.



68. Select the **Set ACL to completed http task** and drag and drop an **Exclusive gateway** next to it.

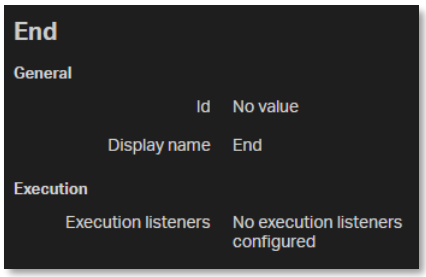


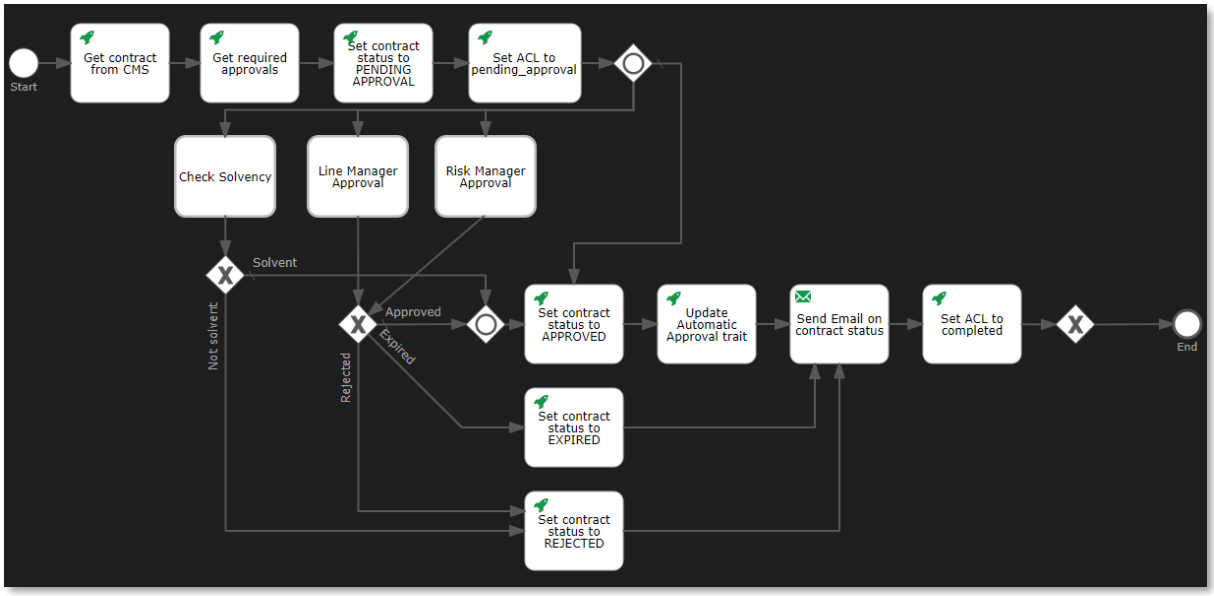
69. Select the third (that is, newly created) **exclusive gateway** and drag and drop an **End event** next to it. Leave enough space between the exclusive gateway and the end event to allow adding text for the connecting sequence flow.



70. Fill the End event attributes using the following details:

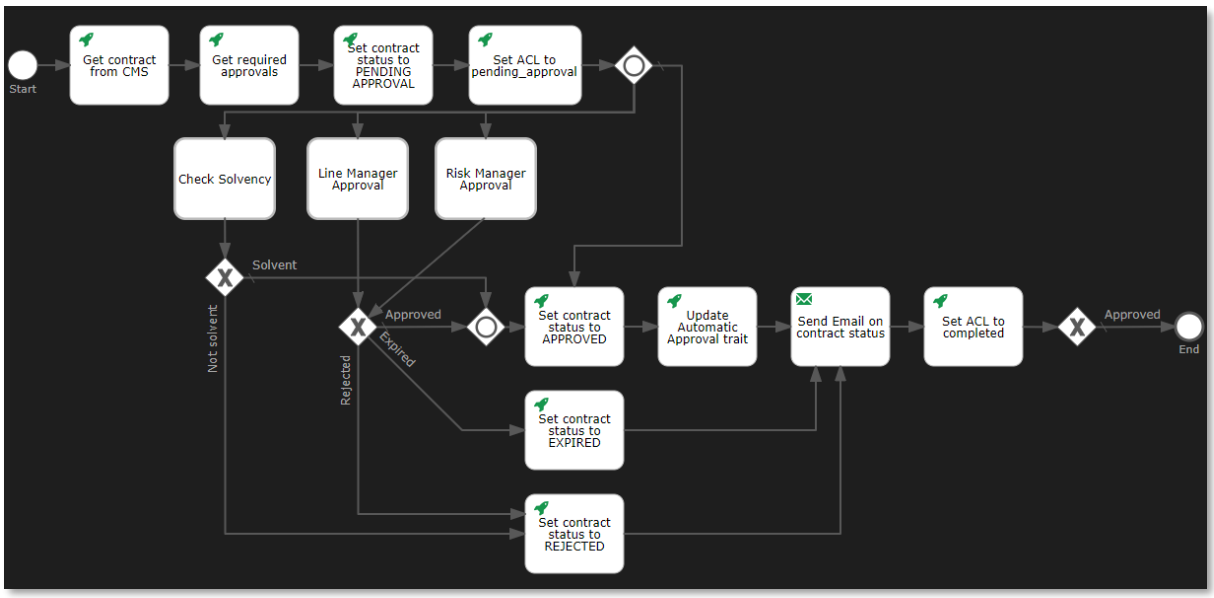
Attribute	Value
Display name	`End`



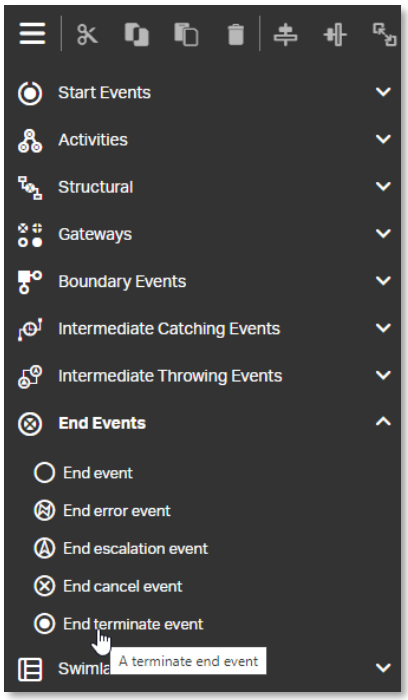


71. Select the new sequence flow between the third **exclusive gateway** and the **End event** and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	This sequence flow is selected when the **status** property of the contract is equal to **APPROVED**. Enter the following display name: `Approved`
Default flow	Select the checkbox.



72. In the **palette**, find **End events > End terminate event**.

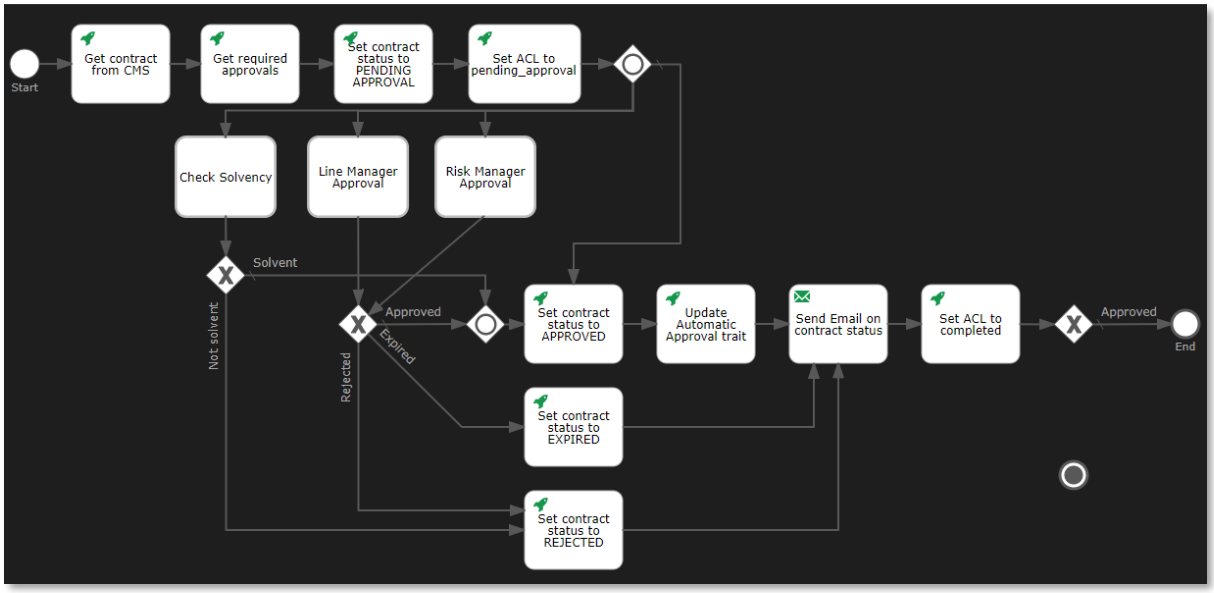


Note

The end terminate event is mostly used with parallel or inclusive gateways. While a normal (untyped) end event indicates that a single process sequence ends, the terminate end event ends the whole process and thereby, ends every activity that may be running at that time.

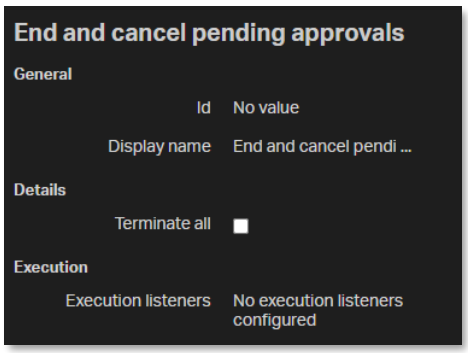
For example, in the contract approval workflow, if any of the manager approval tasks is still pending, the end terminate event results in terminating the sub processes and corresponding activities. This is required to ensure cleaning up in case of one of the selected required approvals marking the contract as EXPIRED or REJECTED.

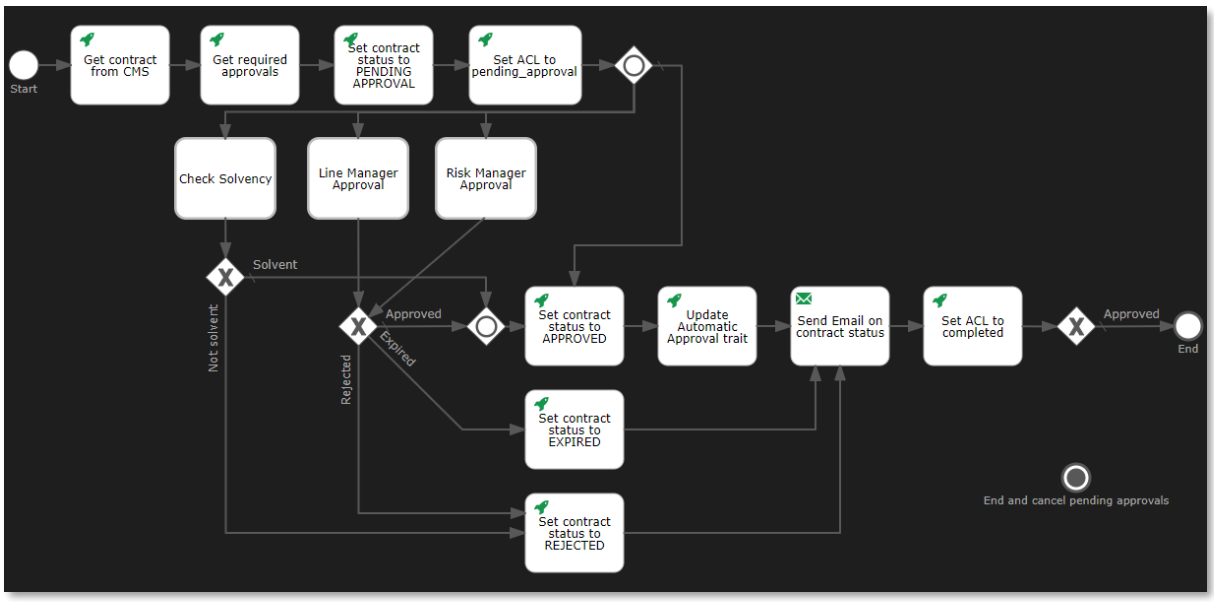
73. Drag and drop the **End terminate event** to the **canvas** under the third **exclusive gateway**. Leave enough space between the exclusive gateway and the end terminate event for adding text to the connecting sequence flow.



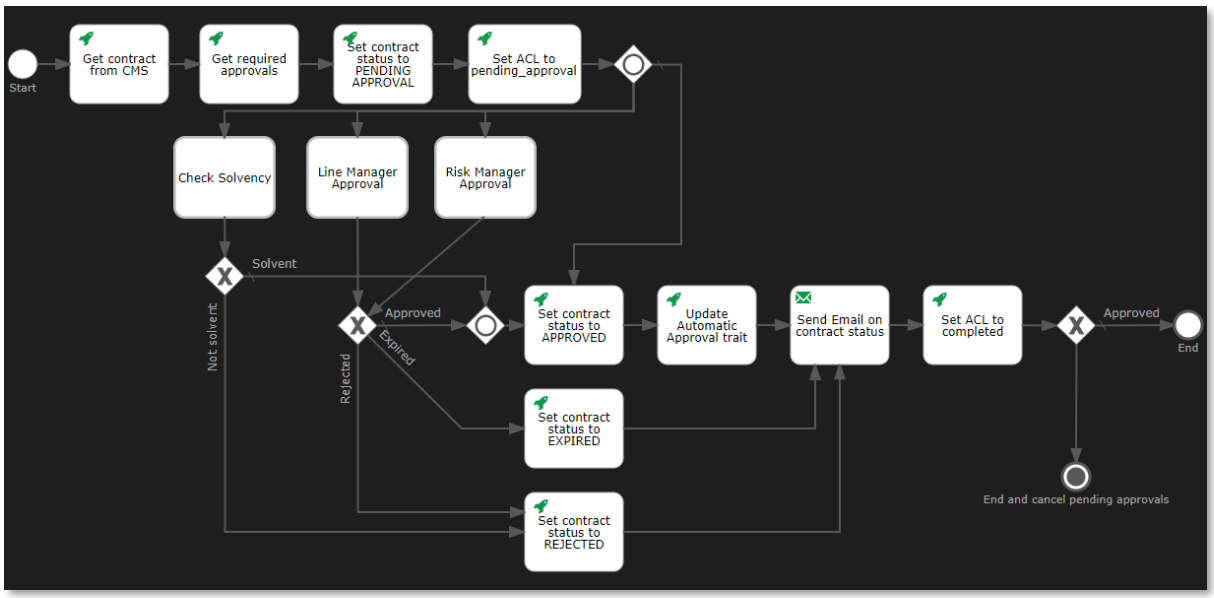
74. Fill the End terminate event attributes using the following details:

Attribute	Value
Display name	`End and cancel pending approvals`



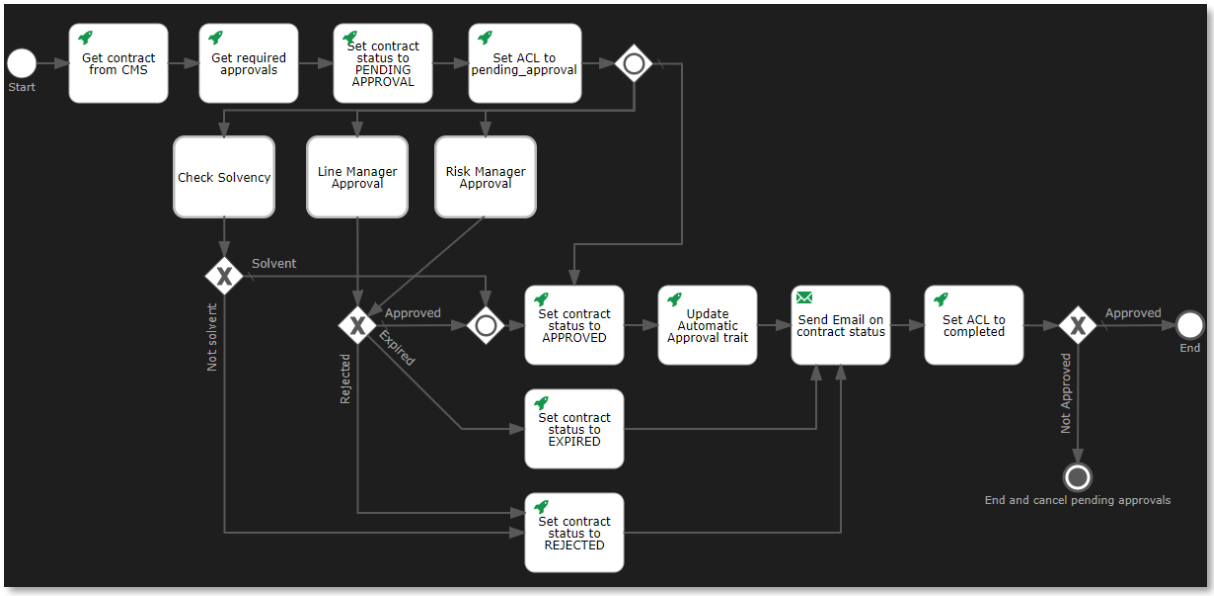


75. Select the third **exclusive gateway** and drag a **sequence flow** to the **End terminate event**.

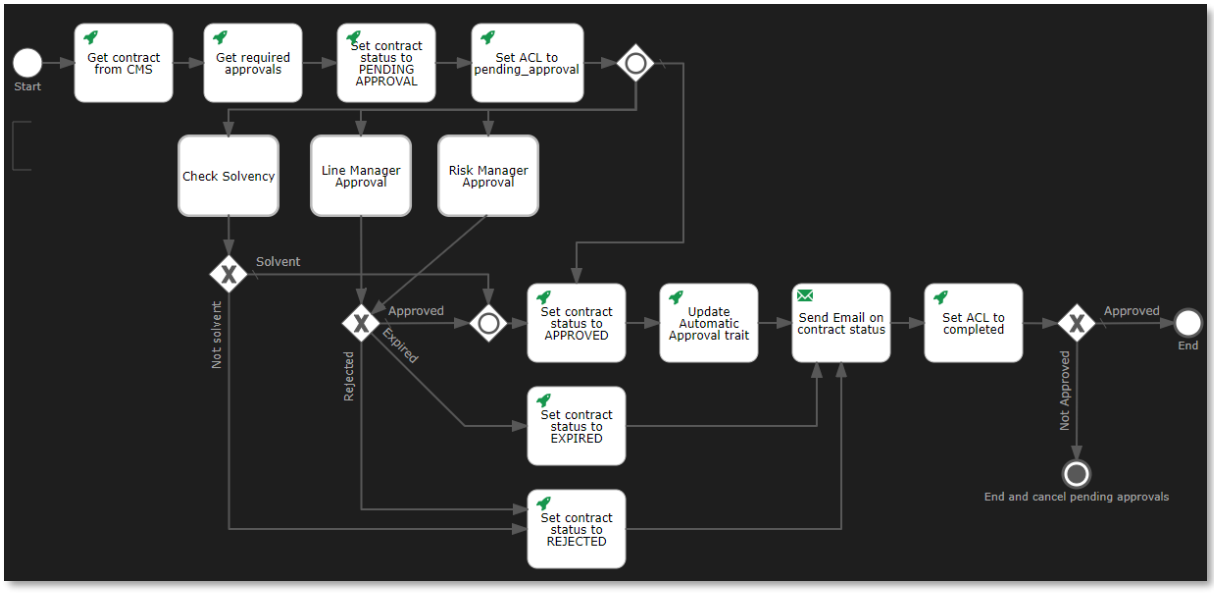


76. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	`Not Approved`
Flow condition	This sequence flow is selected when the **status** property of the contract is not equal to **APPROVED**. Enter the following flow condition: `${contract.properties.status != "APPROVED"}`



77. Drag and drop a **Text annotation** from the **palette** to the **canvas** under the **start event**.

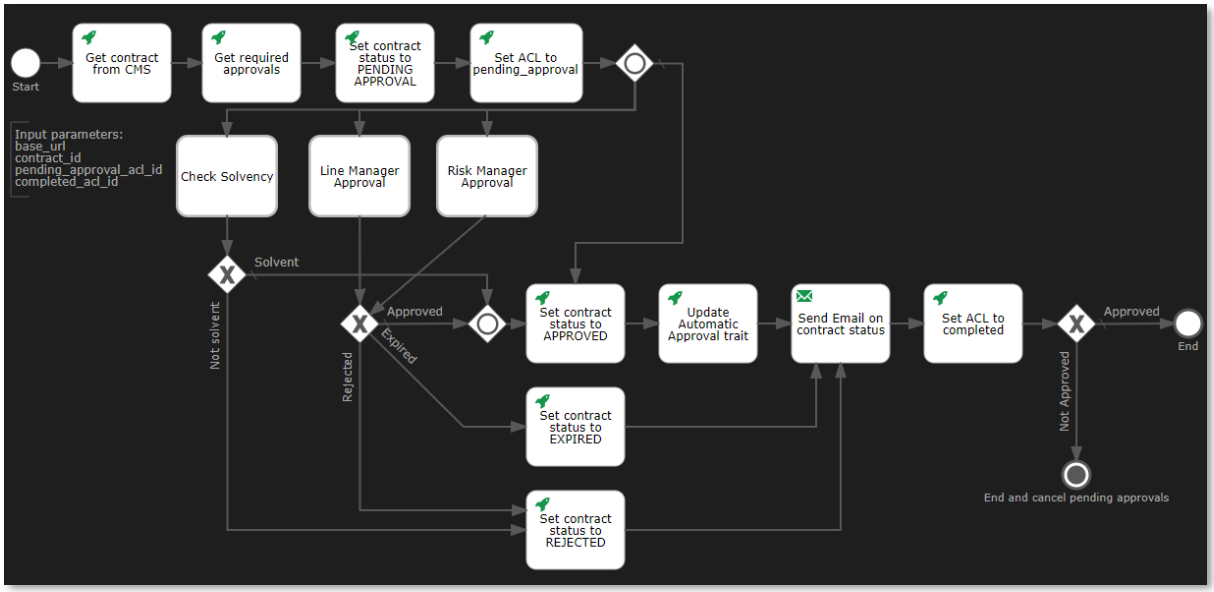


78. Double-click the **Text annotation** on the **canvas** to edit it inline.
79. Set the following text for the text annotation:

Text

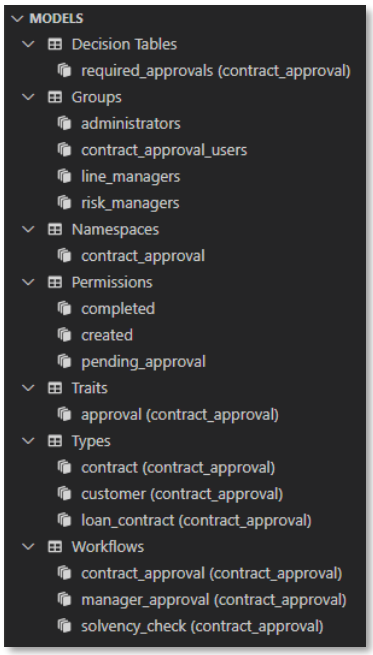
`Input parameters:`
`base_url`
`contract_id`
`pending_approval_acl_id`
`completed_acl_id`

80. Resize the **Text annotation** so that the text displays correctly and fits the square brackets.



81. Save and close the Contract Approval workflow model.

82. The model explorer displays the new **contract_approval (contract_approval)** workflow under **Workflows**.



Next exercise module:

Deploy an application to the OpenText Cloud Platform Services.

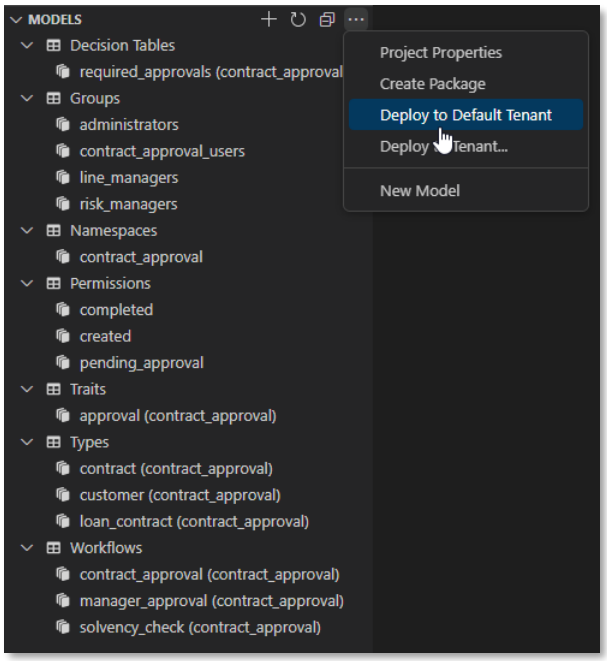
12 [25’] Deploy an application to the OpenText Cloud Platform Services

Learn how to:

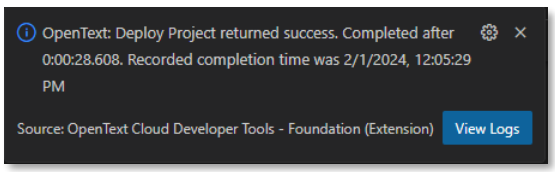
- Deploy an application project (with its models) to the OpenText Cloud Platform Services and get the application credentials
- Verify that the application is deployed using Admin Center
- Add the redirect URLs for the deployed application using Admin Center
- Add the application users to the deployed groups using Admin Center

12.1 Deploy the Contract Approval application and get the application credentials

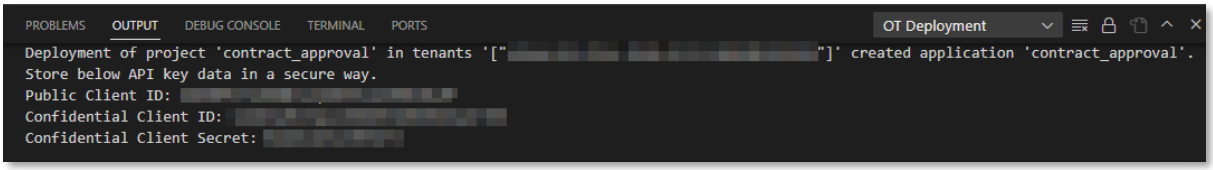
1. Open VS Code and, from the Activity Bar, select the **OpenText Cloud Developer Tools** view.
2. In the **MODELS** section, click the **More Actions** button ··· and select **Deploy to Default Tenant** to deploy your application project.



After deployment, the success message is displayed.



The **OUTPUT** view automatically opens and displays the **OT Deployment** output.



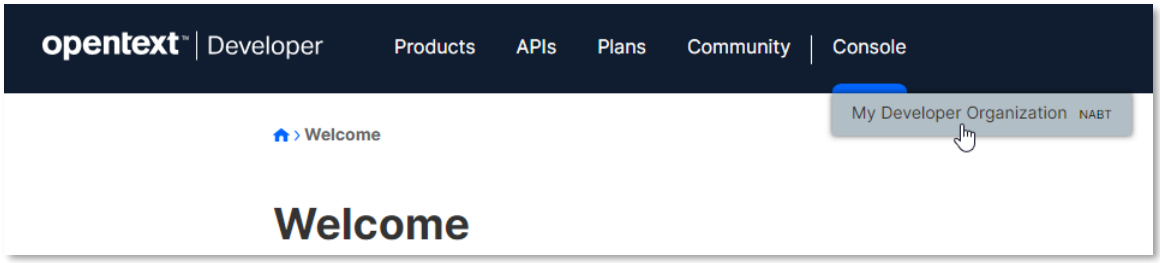
3. Make note of the **Tenant ID** (value between **'["** and **"]'**) on the first line), the **Public Client ID**, the **Confidential Client ID**, and the **Confidential Client secret**. You require these values to test the deployment and run the application.

Note

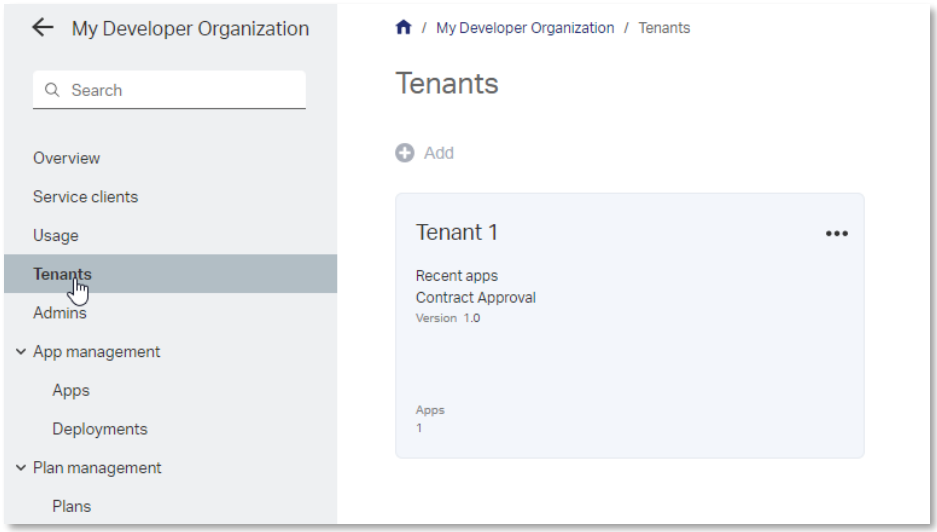
You can copy the above client credentials into a text file and save it. OpenText recommends that you store this type of sensitive information in a secure way.

12.2Verify that the application is deployed

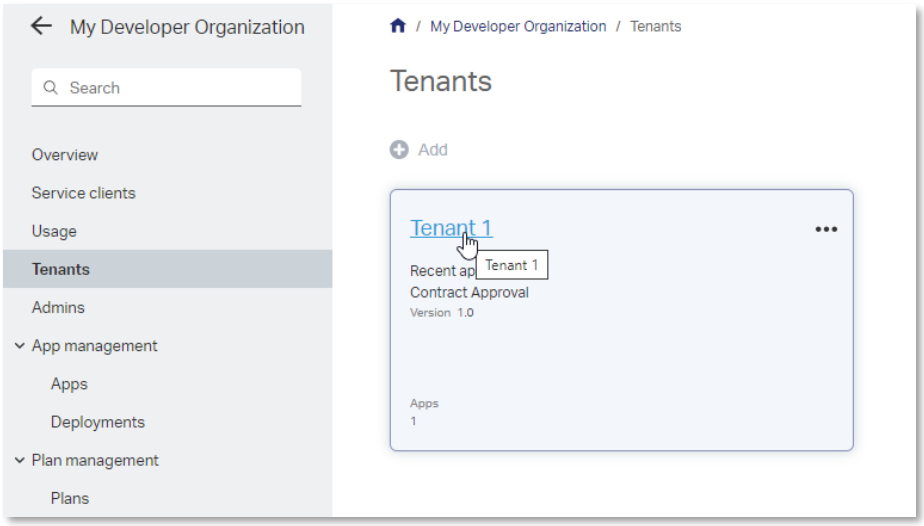
1. Sign in to the developer.opentext.com website.
2. Select the **<organization name> <region>** combination for your developer organization from the **Console** menu.



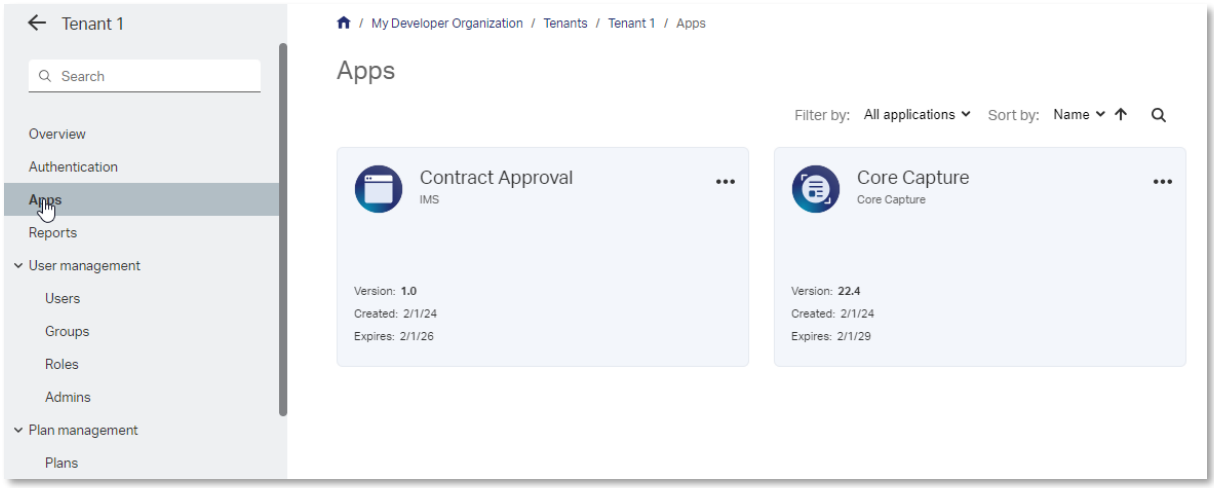
3. In the **Admin Center** navigation menu, click **Tenants**.



4. Select the tenant to which you deployed the application.



5. In the (tenant) navigation menu, click **Apps** and confirm the existence of the deployed **Contract Approval** app.



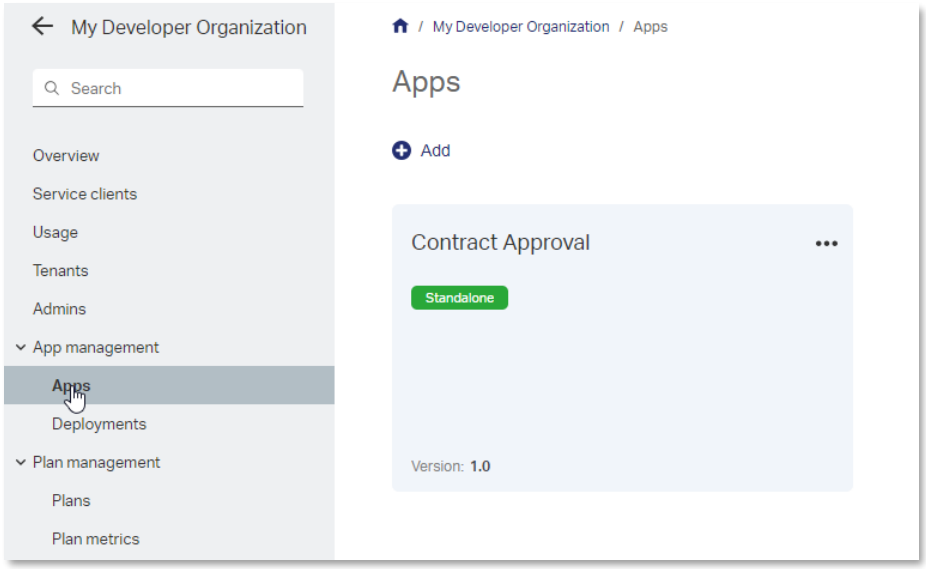
Next step:

Add the redirect URL for the application authentication flow.

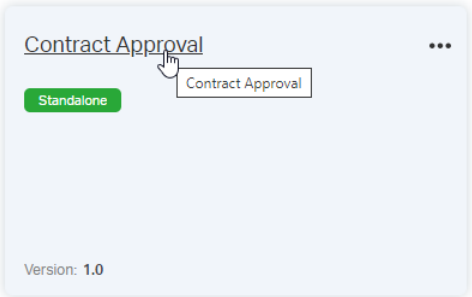
12.3 Add the redirect URL for the application authentication flow

To authenticate, users of your Contract Approval application need to sign in using an external (to the application) login screen as part of the authentication (OAuth) flow. This means that the web browser in which you open the Contract Approval application will first be redirected to a login screen, and after it is authenticated, it will be redirected back to your Contract Approval application (which you will configure to run on https://localhost:4000). This redirect URL needs to be added at the organization level to the deployed application's public service client configuration (represented by the Public Client ID) for the authentication to work.

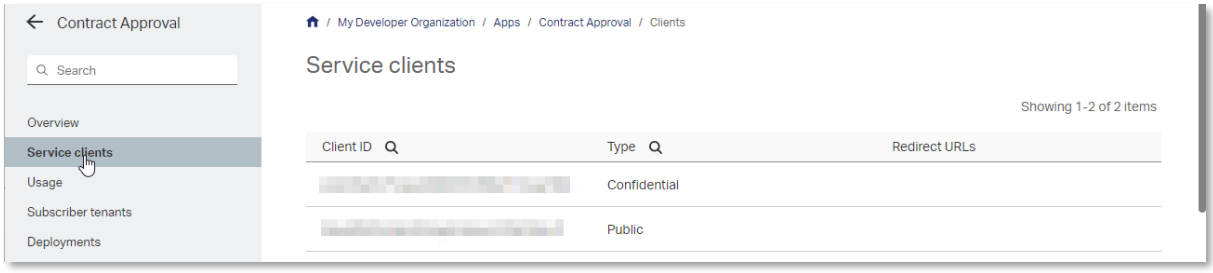
1. Sign in to developer.opentext.com.
2. Select the **<organization name> <region>** combination for your developer organization from the **Console** menu.
3. In the **Admin Center** navigation menu, click **Apps**.



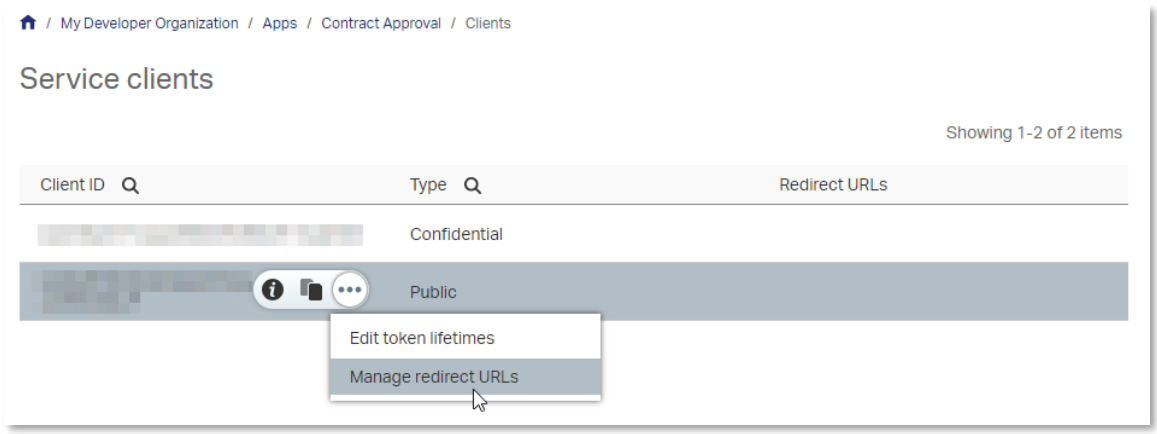
4. Click on the **Contract Approval** title link of the Contract Approval app tile.



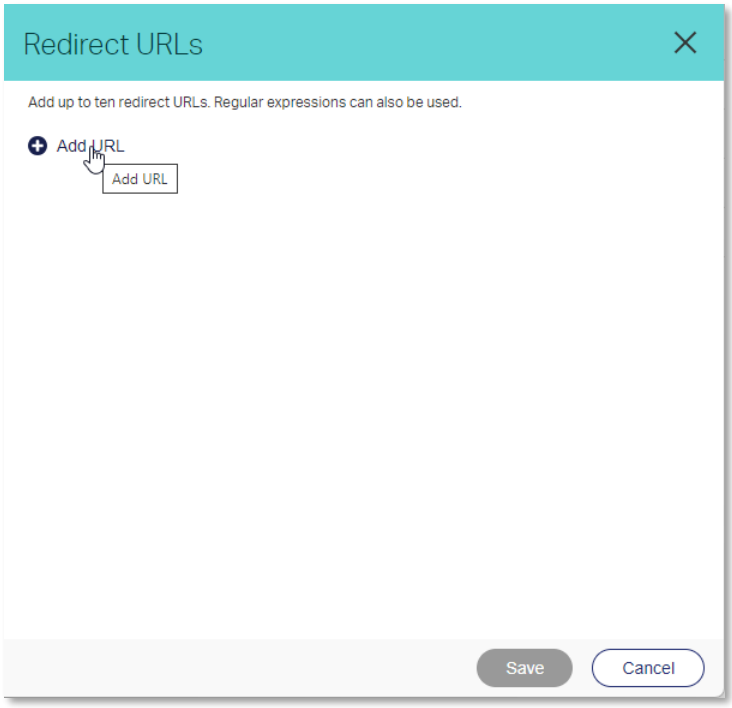
5. In the **Contract Approval app** navigation menu, click **Service clients**.



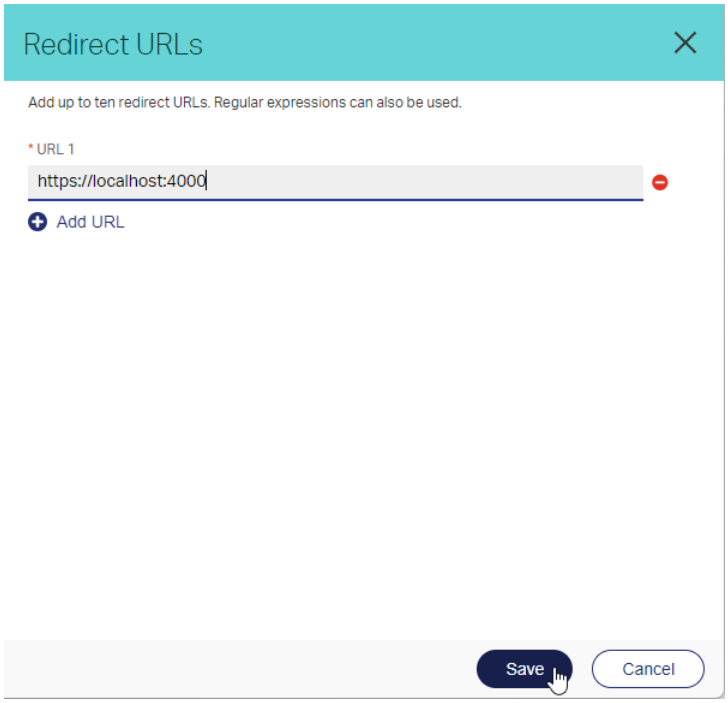
6. Hover over the **Public** client entry in the client list and click the **More** button ••• select **Manage redirect URLs**.

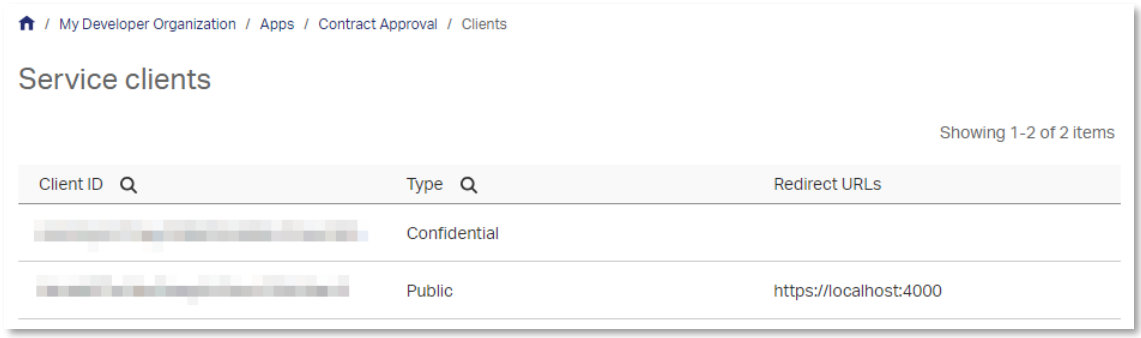


7. In the **Redirect URLs** screen, click **Add URL**.



8. In the URL 1 box, enter `https://localhost:4000` and click **Save**.



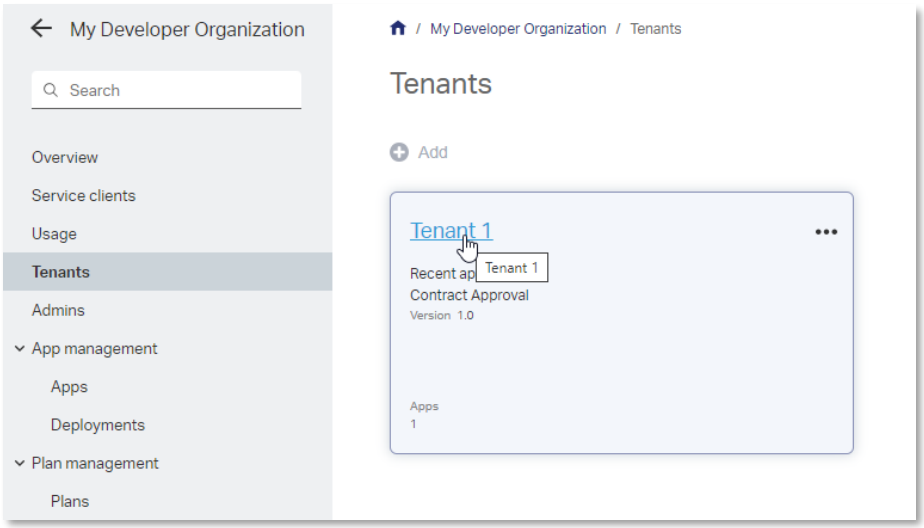


Next step:

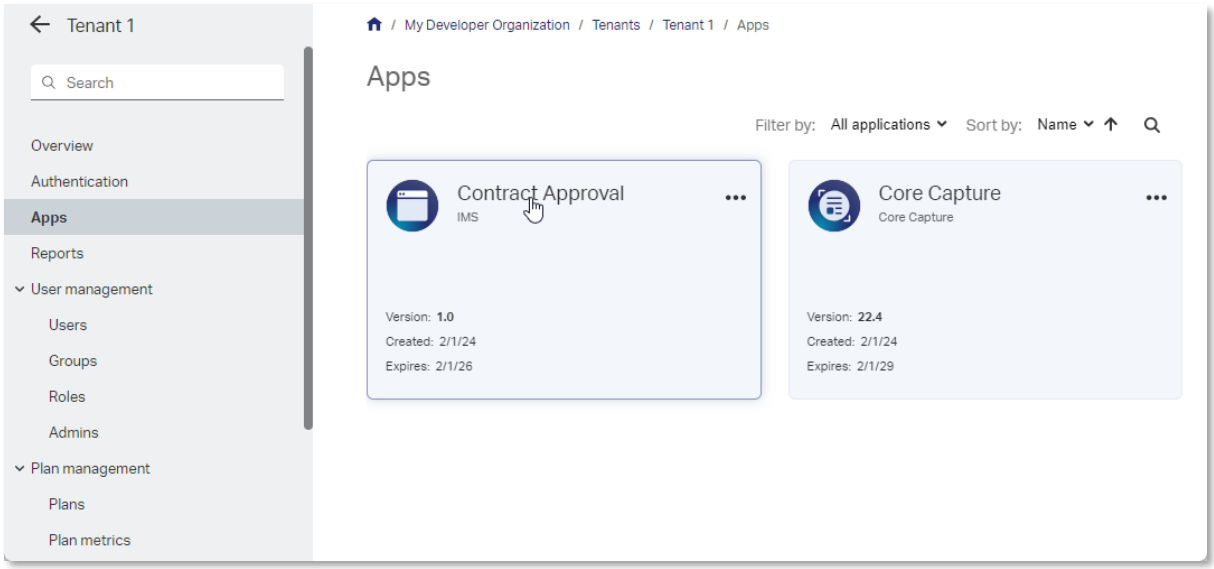
Add the application users to the application groups.

12.4 Add the application users to the application groups

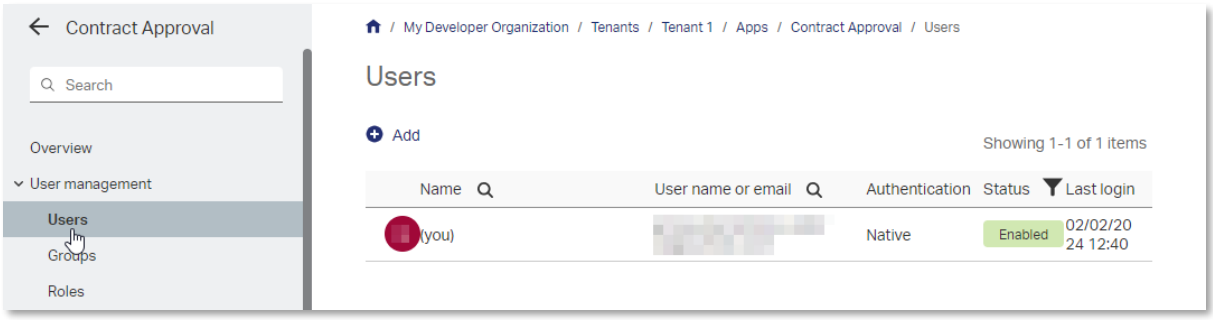
1. Sign in to developer.opentext.com.
2. Select the **<organization name> <region>** combination for your developer organization from the **Console** menu.
3. In the **Admin Center** navigation menu, navigate to **Tenants** and select the tenant to which you deployed the application.



4. In the (tenant) navigation menu, go to **Apps** and click on the **Contract Approval** title link of the Contract Approval app tile.



5. In the (tenant app) navigation menu, go to **User management > Users**.



The user used for subscribing to the trial or developer plan is the only user listed in the **Users** list.

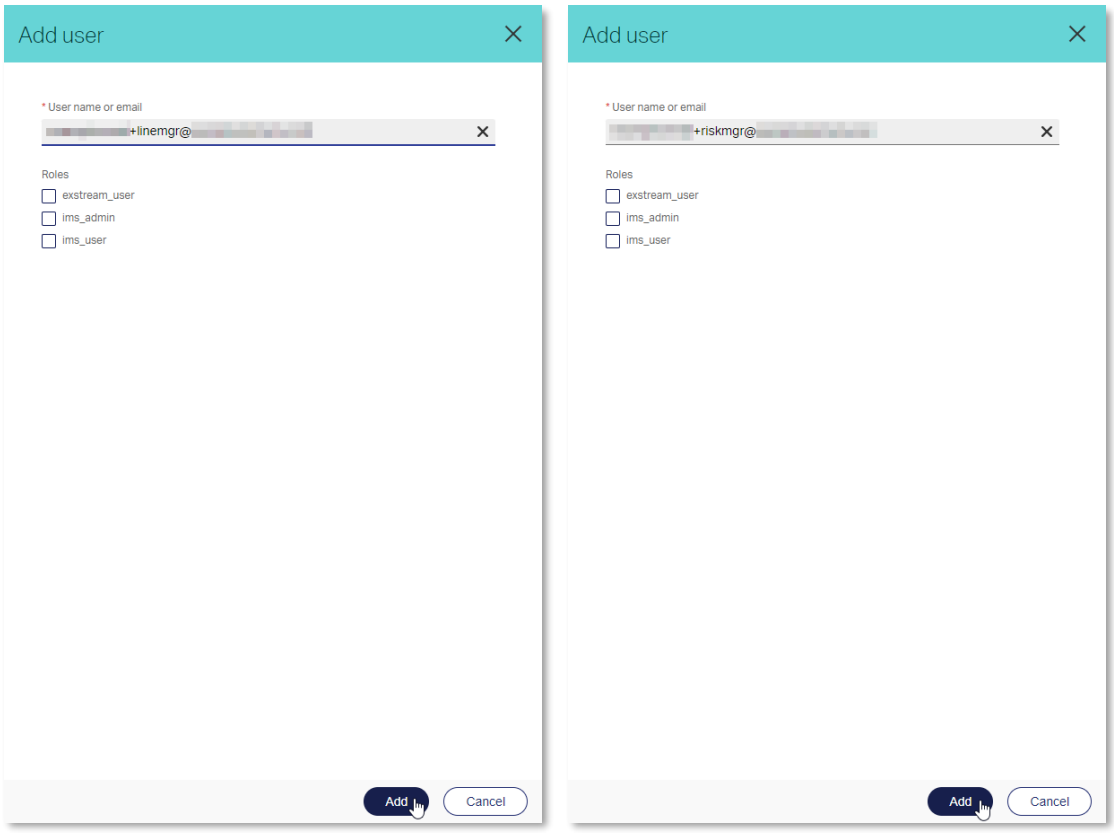
6. Click **Add** and select **Add user** to add two additional users (email accounts) to represent the line manager and risk manager as approvers.

7. From the Add user dialog box, in the **User name or email** box, to represent a role specific email address, enter the email address used for subscribing to the trial or developer plan, but add a “+” and the role (“linemgr” or “riskmgr”) right before the “@”.

   Using a “+” in an email address is very useful as it allows differentiating recipients/email addresses while delivering them all to the same email address (the original address without the “+” and text before “@”).

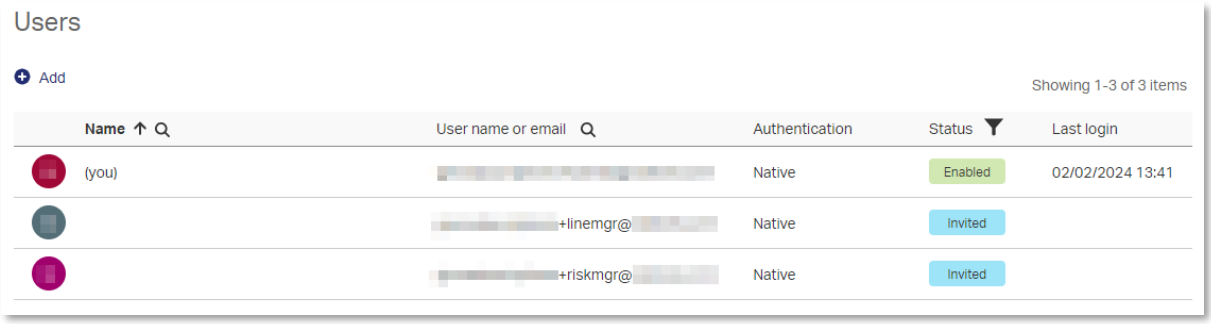
   So, for example, if the email address used to subscribe to the trial or developer plan is **myname@somedomain.com**, applying the approach of adding a “+” with the role results in the following two emails to represent the **line manager** and **risk manager**:

   - Line manager: **myname+linemgr@somedomain.com**
   - Risk manager: **myname+riskmgr@somedomain.com**

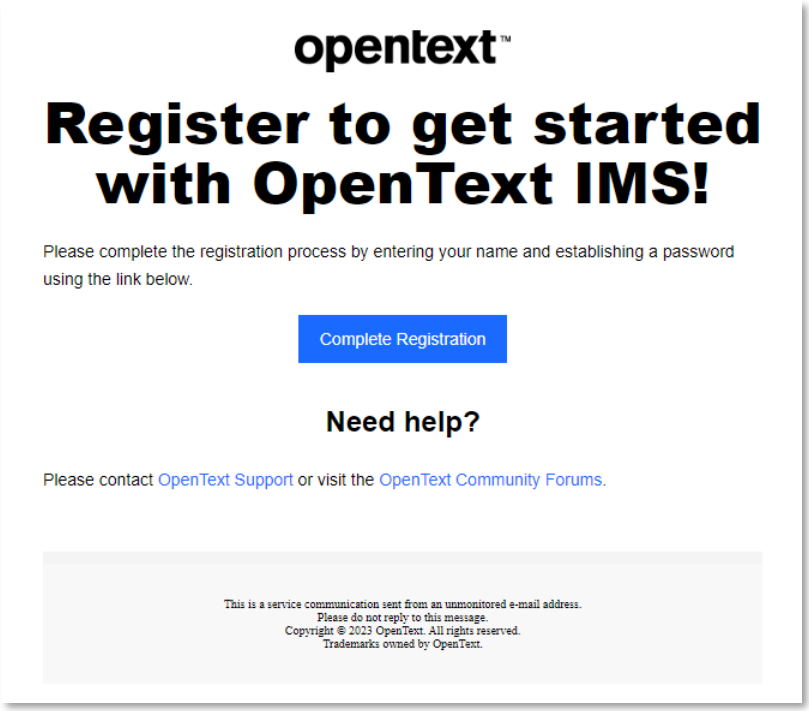


Note

Only two new users need to be added. For the **regular user** (that is, the approval requester), the email address used for subscribing to the trial or developer plan will be used.

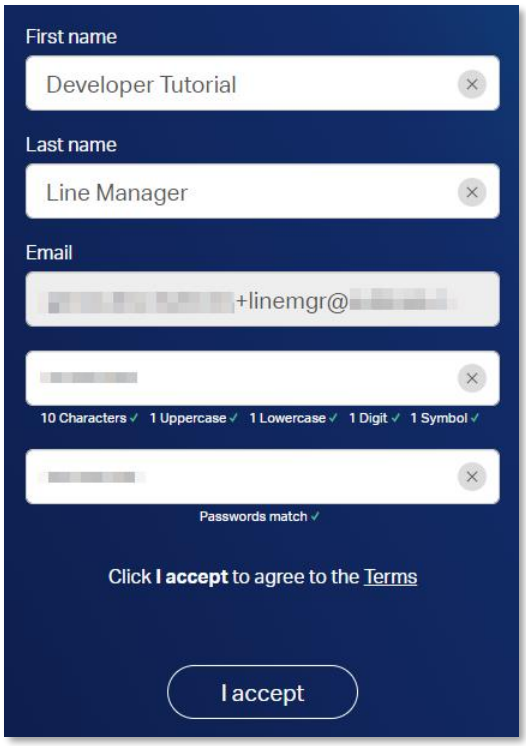


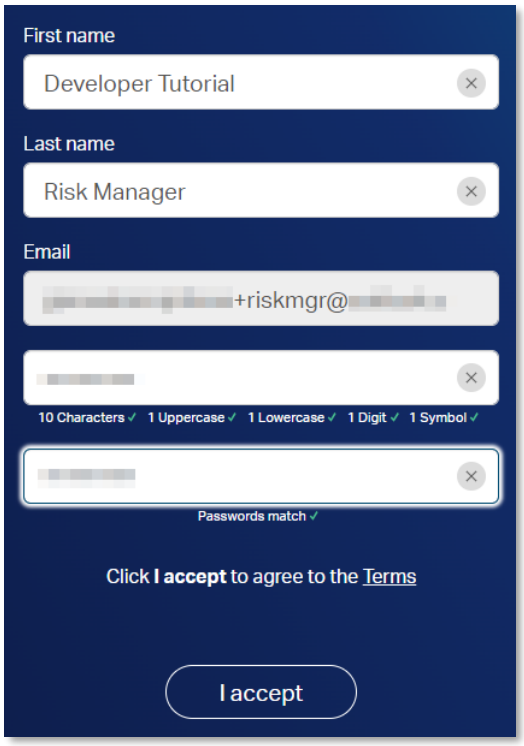
8. Go to your email client for the developer trial subscription email account and locate the two emails with the Subject: **Register your new OpenText IMS account**.



9. For each email, click **Complete Registration** and fill the **OpenText Cloud Platform** registration form using the following details:

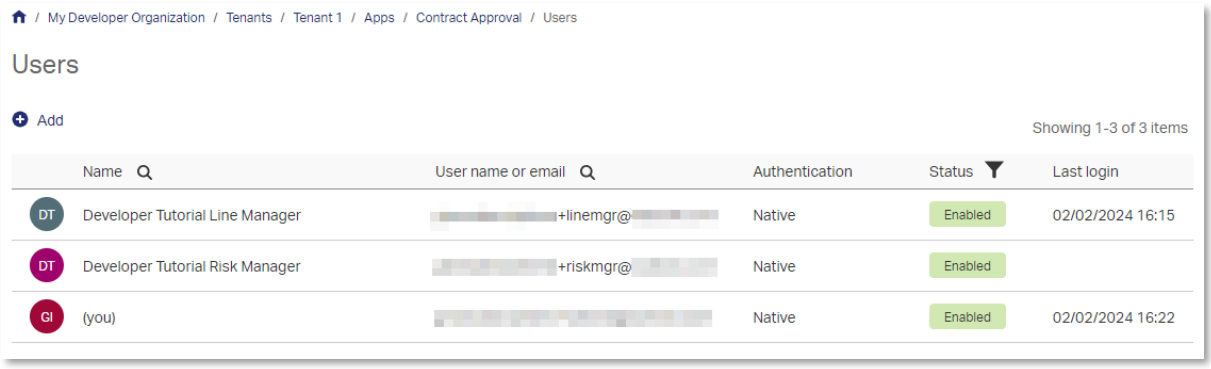
Field	Value
First name	`Developer Tutorial`
Last name	Use the application role as value for the last name field. Depending on the email address in the **Email** field of the form, fill the correct application role, that is: `Line Manager` or `Risk Manager`
Email	The email address for the specific approver role and it appears automatically.
Password	Specify the password.
Confirm password	Confirm the password.



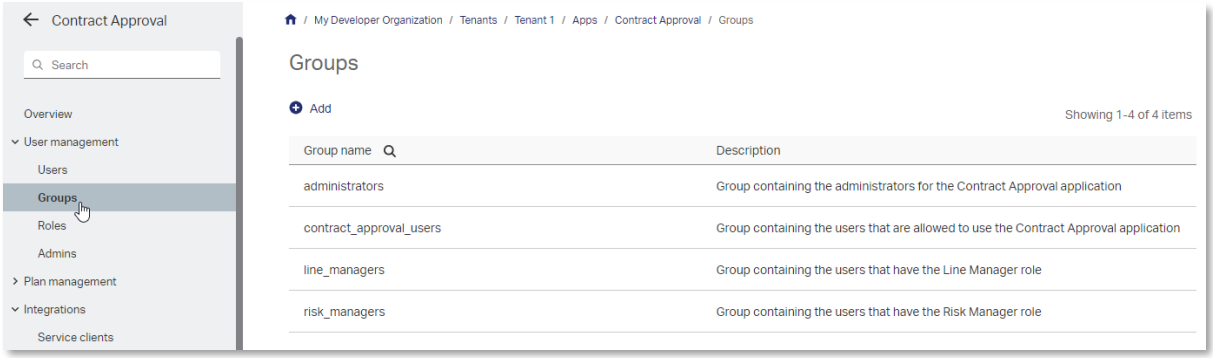


10. Click **I accept** to accept the terms and proceed with the registration of the two new users.

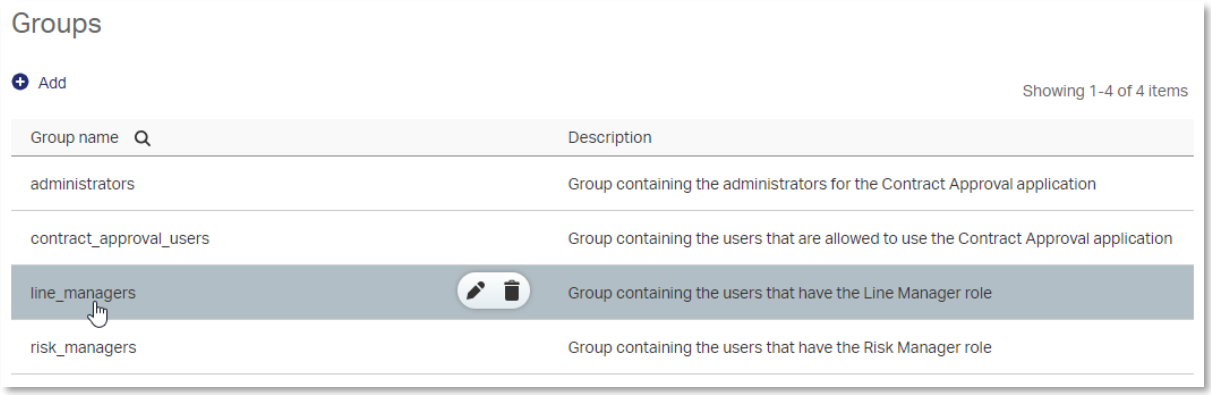
11. Go back to **Admin Center** and refresh the **User management > Users** page to confirm that the status of the two newly added users is **Enabled**.



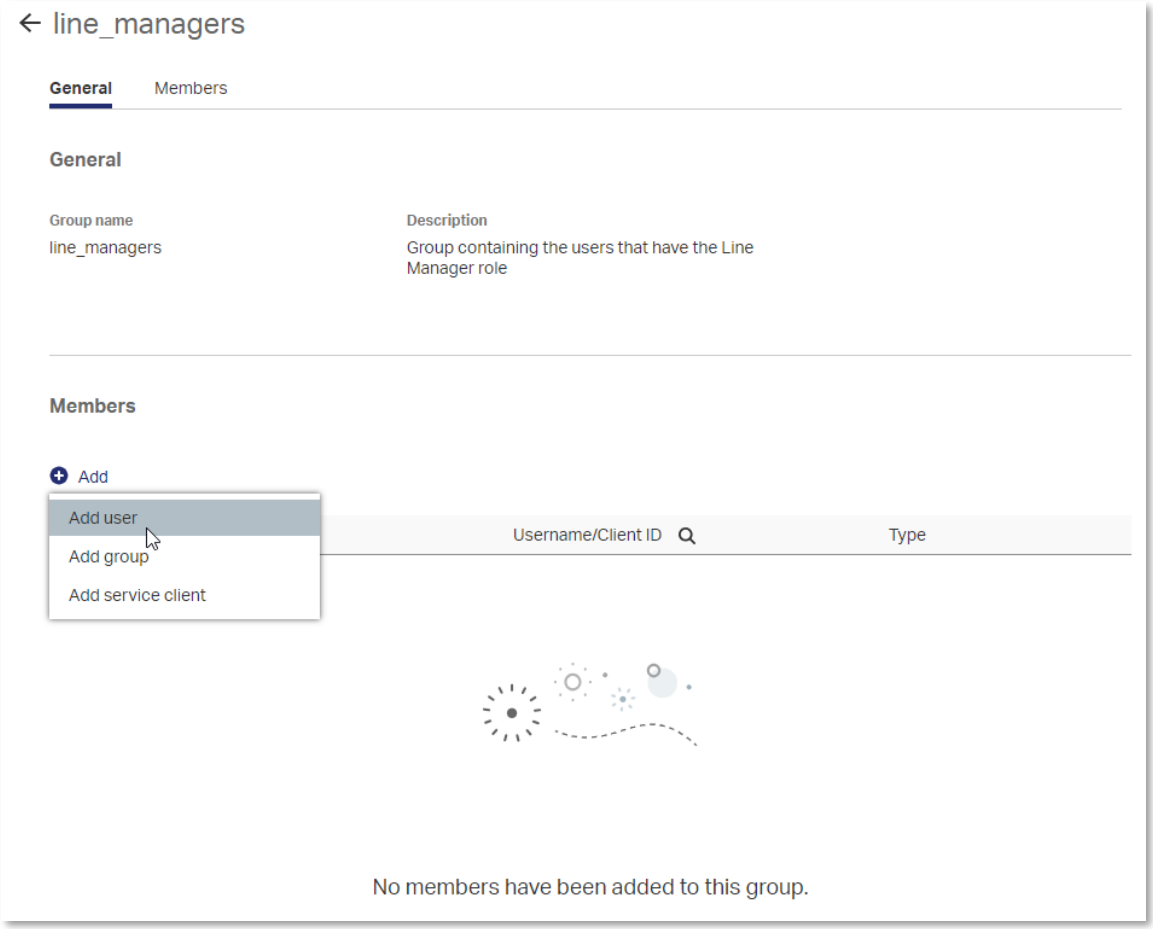
12. In the (tenant app) navigation menu, select **User management > Groups**.



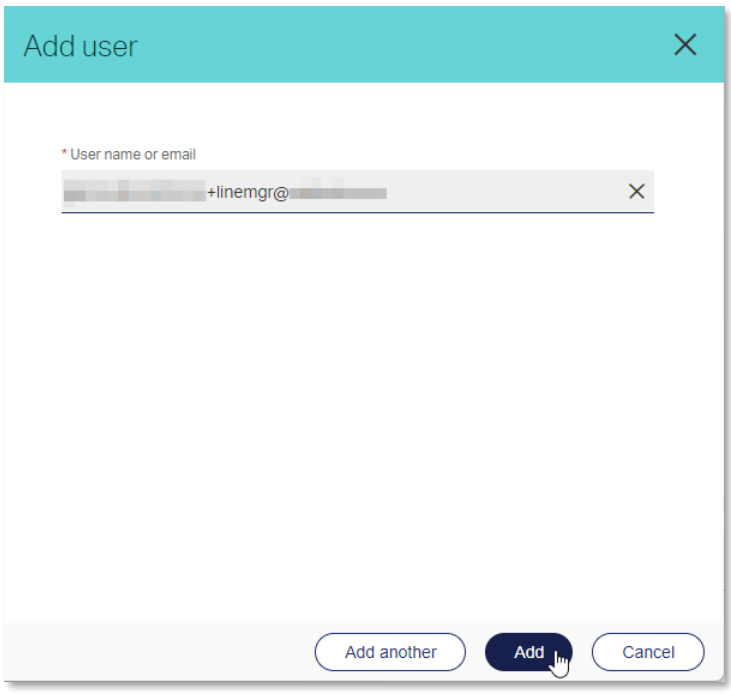
13. From Groups, select **line_managers**.

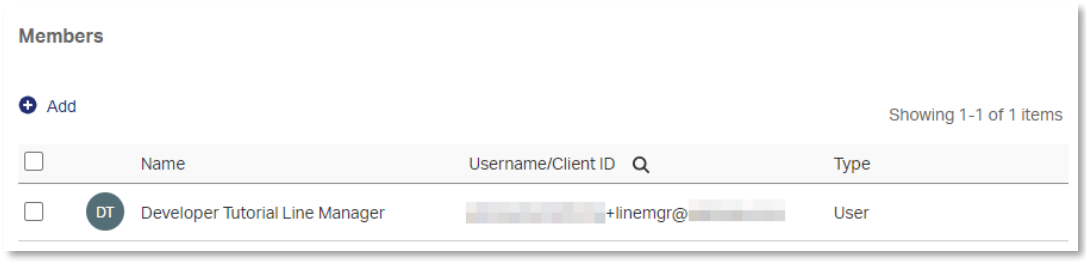


14. In the **Members** section, click **Add > Add user** to add the line manager approver as a group member.

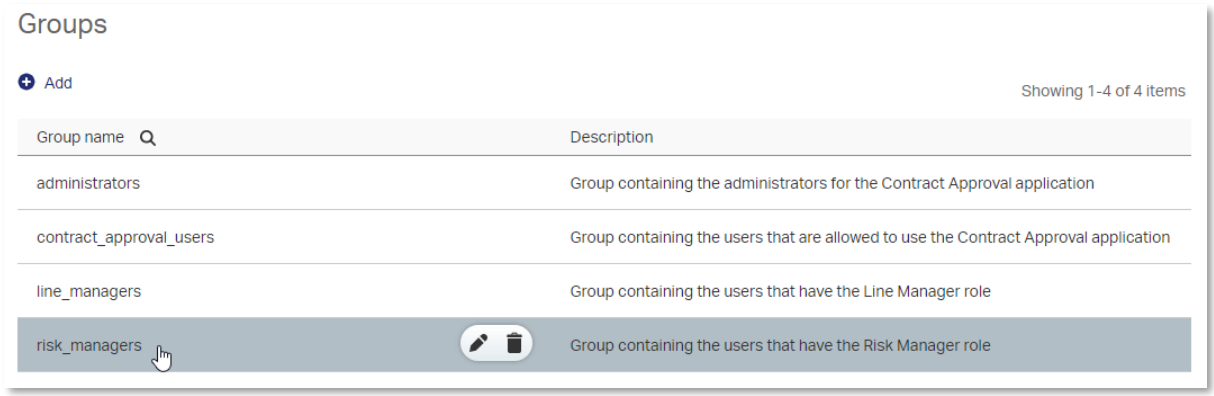


15. In the **User name or mail** box, enter the **line manager approver email** and click **Add**.

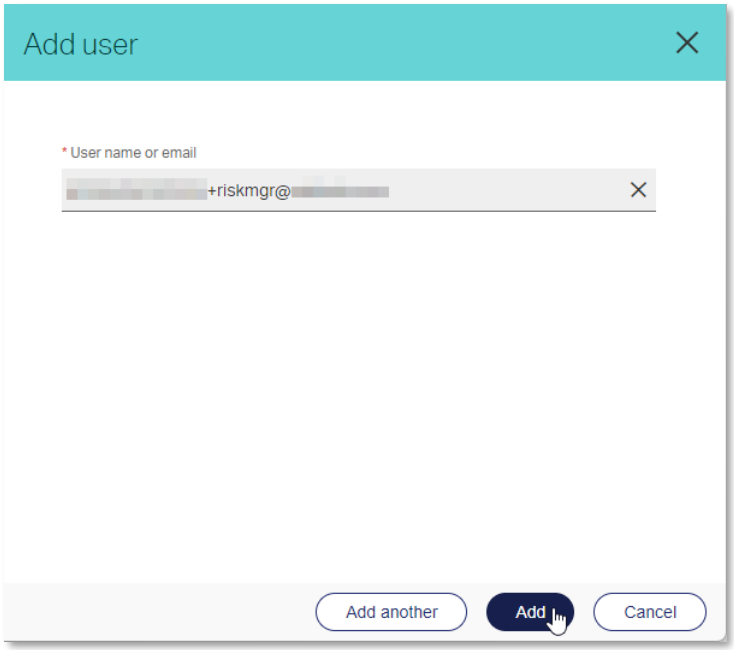


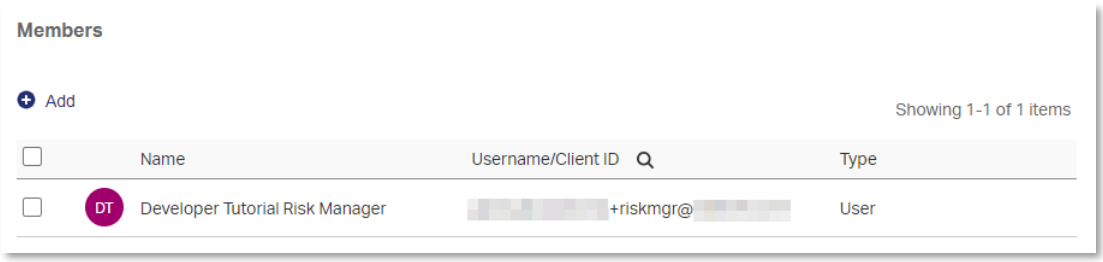


16. Click ← (back arrow icon) to go back to the **User management > Groups** list.
17. Click the **risk_managers** group.

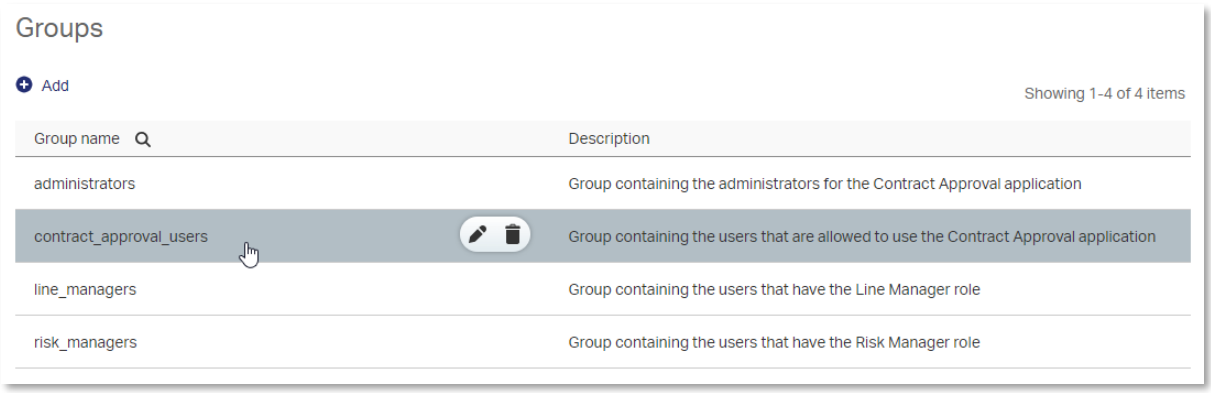


18. In the **Members** section, click **Add > Add user** to add the risk manager approver as a group member.
19. Enter the **risk manager approver email** and click **Add**.

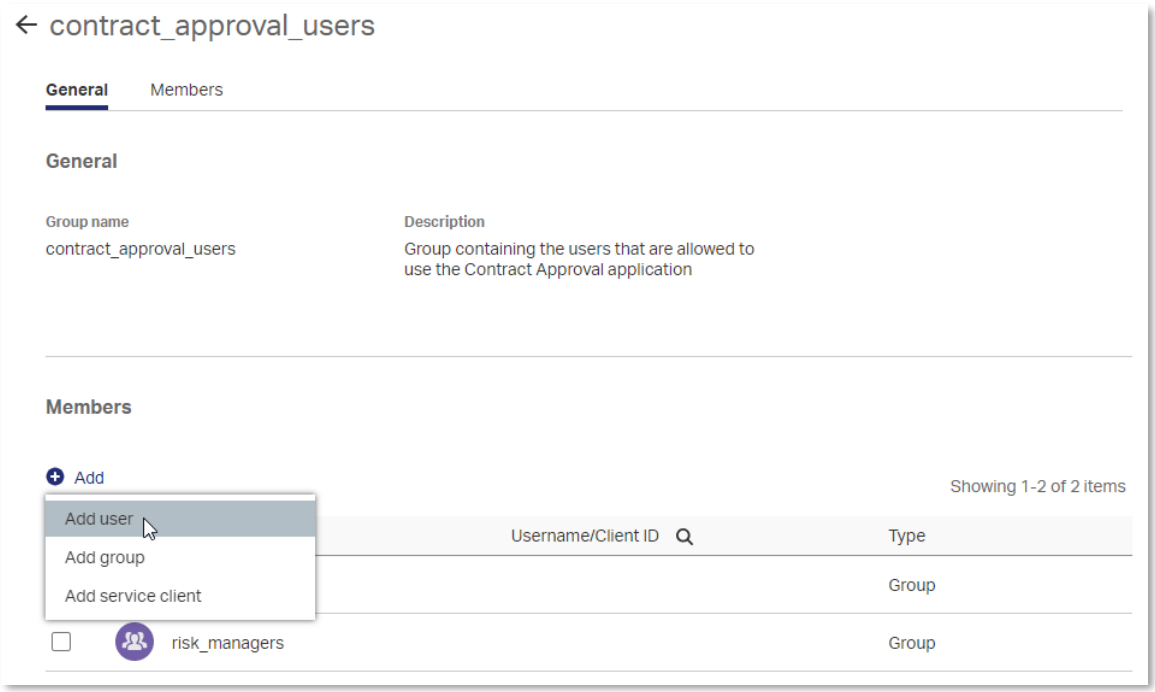




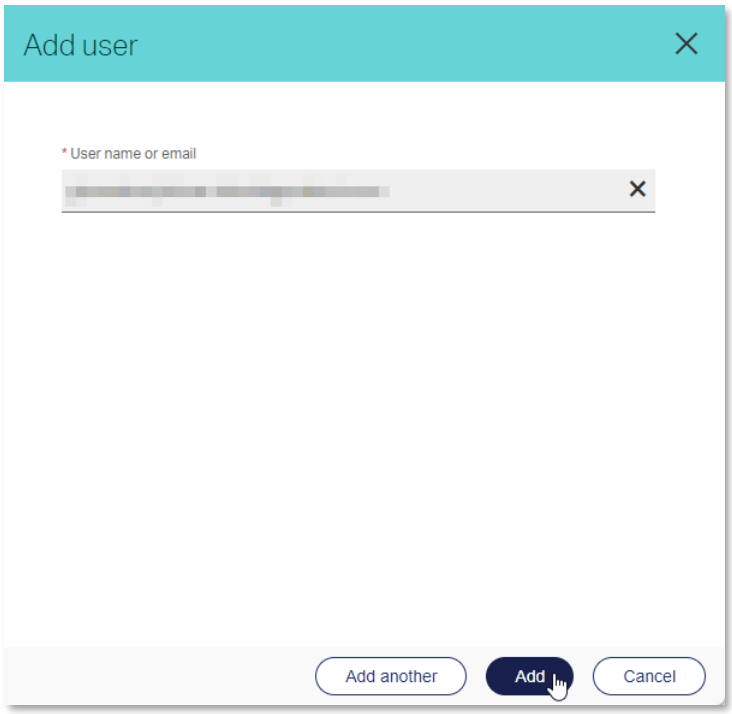
20. Click ← (back arrow icon) to go back to the **User management > Groups** list.
21. Click the **contract_approval_users** group.

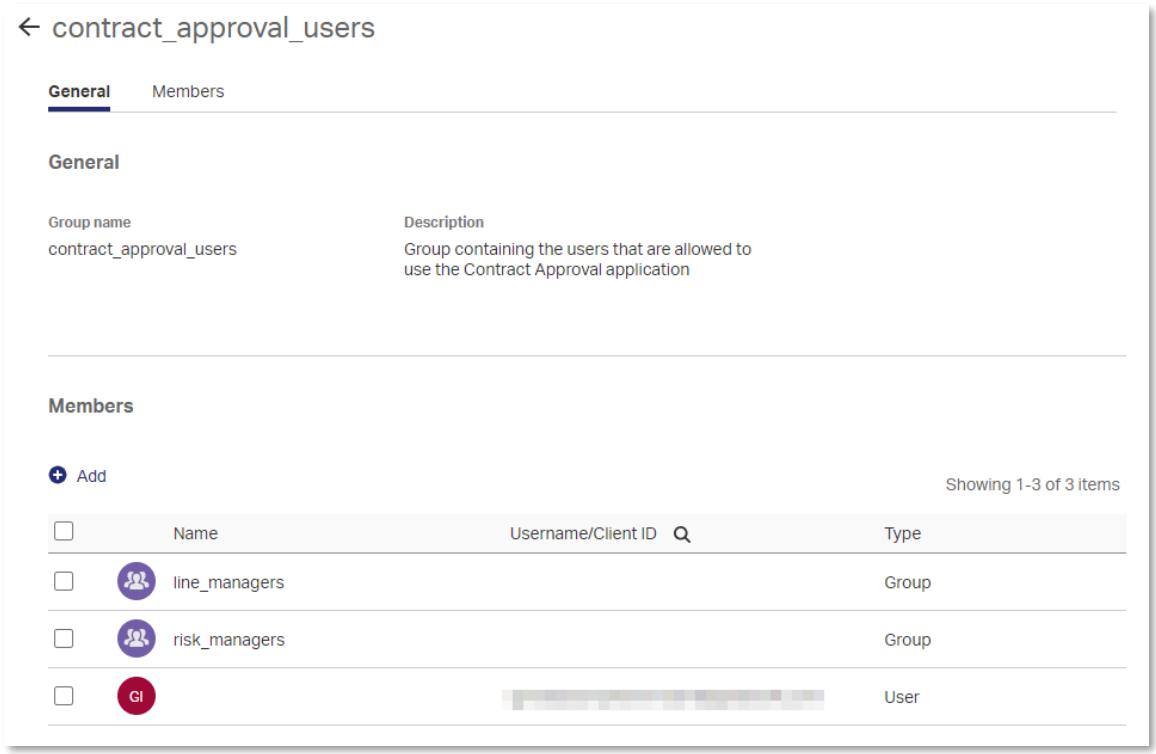


22. In the **Members** section, click **Add > Add user** to add the regular user (that is, the user requesting the contract approval and not having line manager or risk manager approval privileges).



23. Enter the **regular user email** (the email address used for subscribing to the trial or developer plan) and click **Add**.





Next exercise module:

Work with the OpenText Cloud Platform Services APIs.

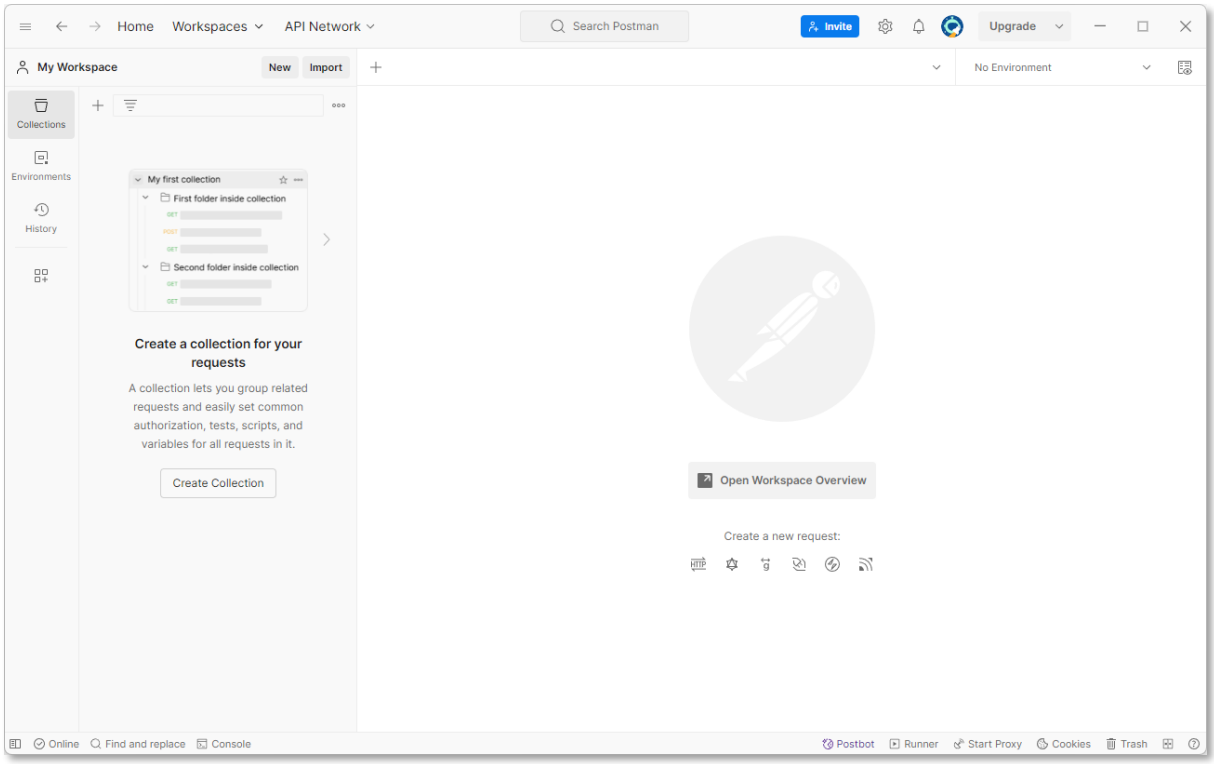
13 [25’] Work with the OpenText Cloud Platform Services APIs

Learn how to:

- Download and install Postman
- Download the finished Contract Approval App from GitHub
- Import the Postman collection and environment
- Verify the deployment of the different models using the OpenText Cloud Platform Services APIs using Postman

13.1Download and install Postman

1. Download and install postman from https://www.postman.com/downloads.
2. Sign in or create an account to allow using collections.



Next step:

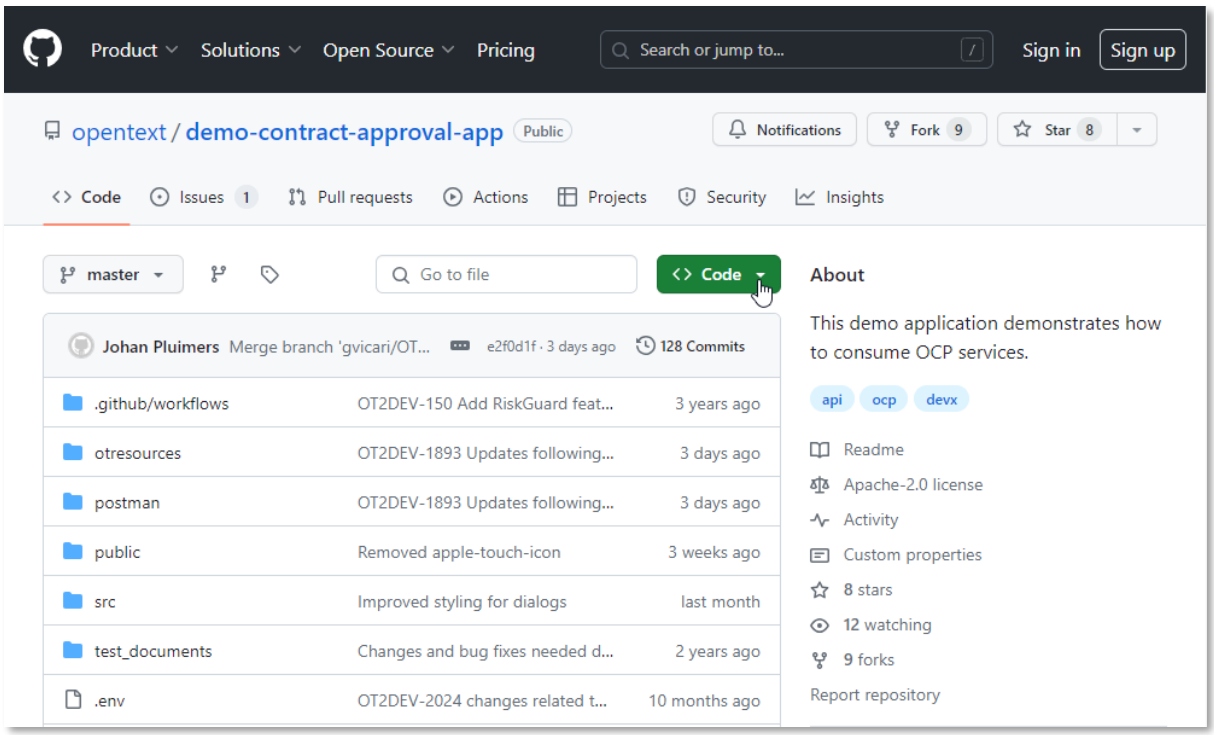
Download the Contract Approval application.

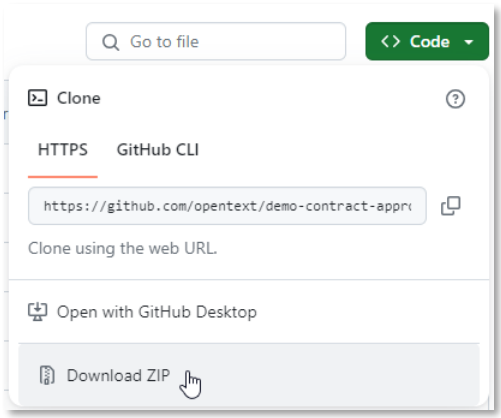
13.2Download the Contract Approval application

1. Go to web address https://github.com/opentext/demo-contract-approval-app and download the Contract Approval App (demo-contract-approval-app) from GitHub.

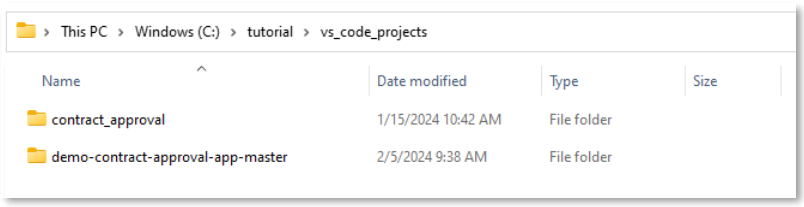
Note

The latest version of the Contract Approval App is available from GitHub. For this tutorial, you will use “Download ZIP” to download the application. To ensure having the latest version of the Contract Approval App, you can download the ZIP at any time, but you can also choose to clone the repository and regularly perform a git pull.





2. Extract the **demo-contract-approval-app-master** folder from the downloaded ZIP file to your file system (for example, in the same folder as the tutorial project).

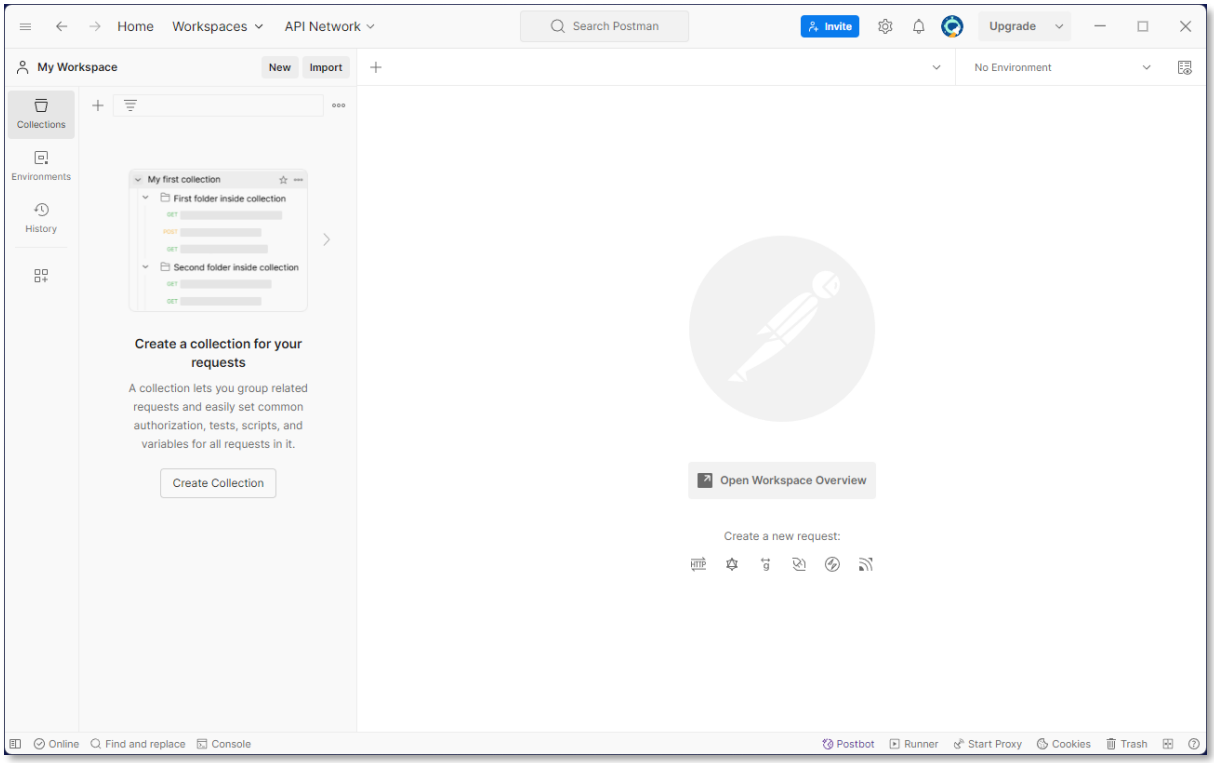


Next step:

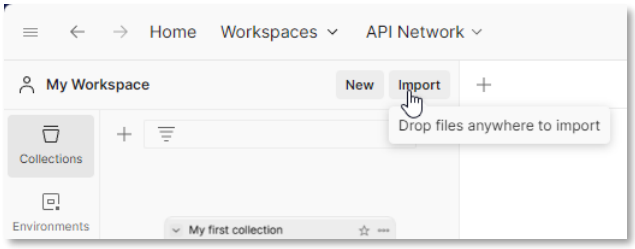
Import the Postman collection and environment into Postman.

13.3 Import the Postman collection and environment into Postman

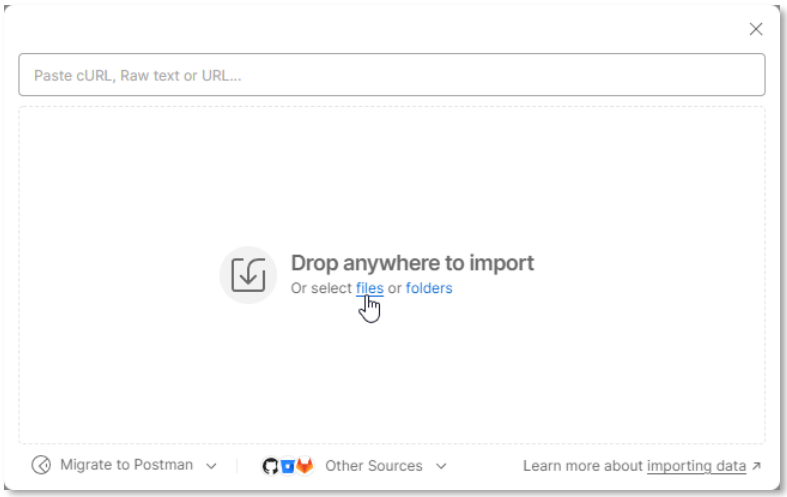
1. Open **Postman**.

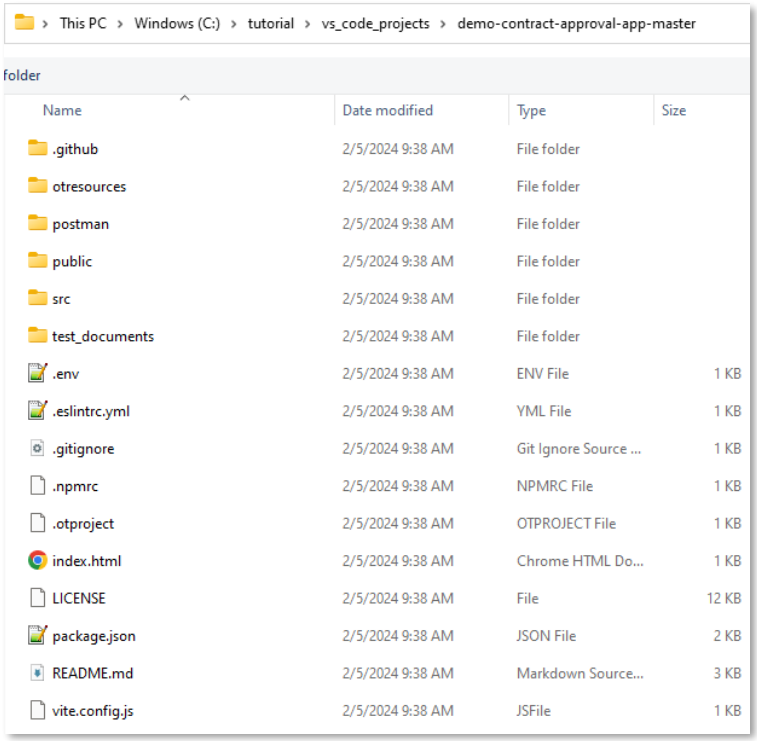


2. Select **Collections** from the sidebar and click **Import**.

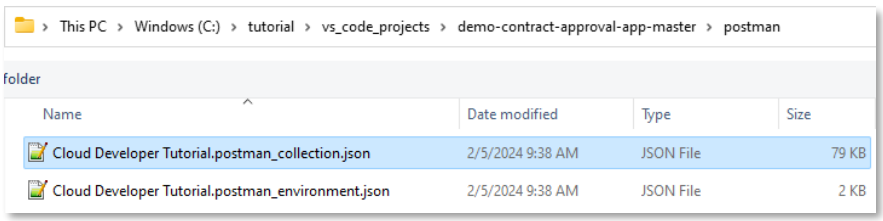


3. From the import dialog box, click to select **files** and navigate to the previously extracted **demo-contract-approval-app-master** folder.

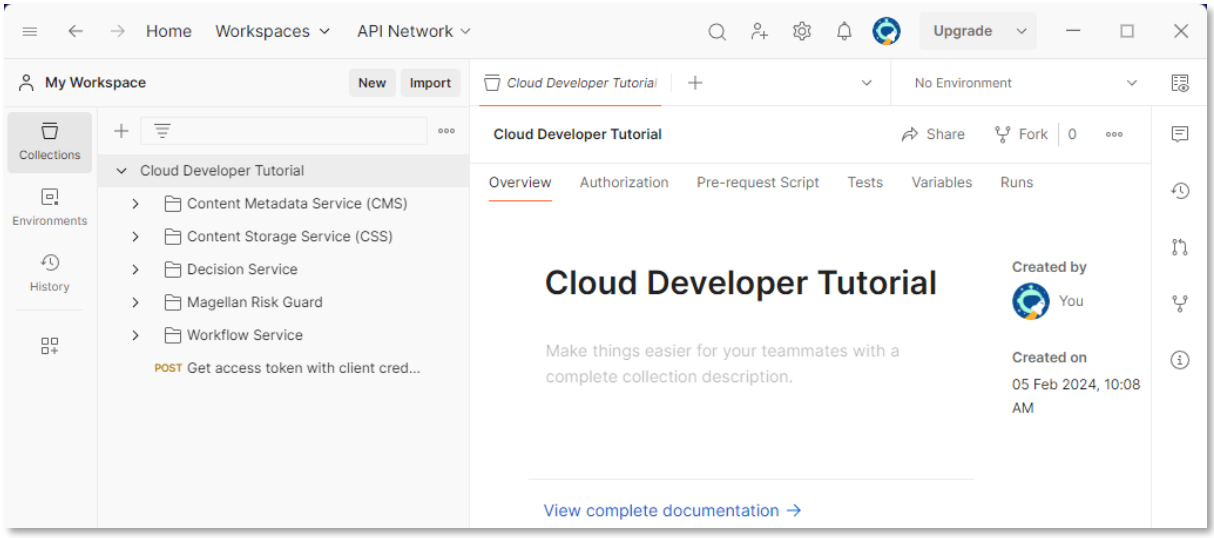




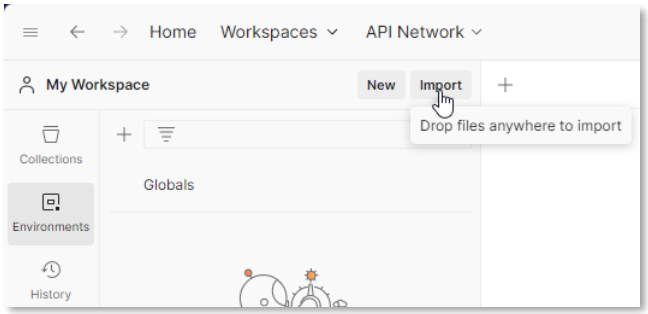
4. In the **demo-contract-approval-app-master** folder, navigate to the **/postman** folder and select the **Cloud Developer Tutorial.postman_collection.json** collection file.



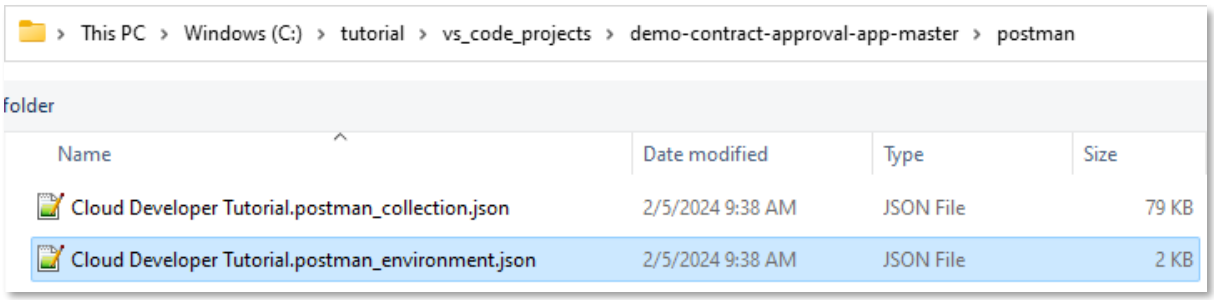
5. Click **Open** to import the **Cloud Developer Tutorial** collection into Postman.



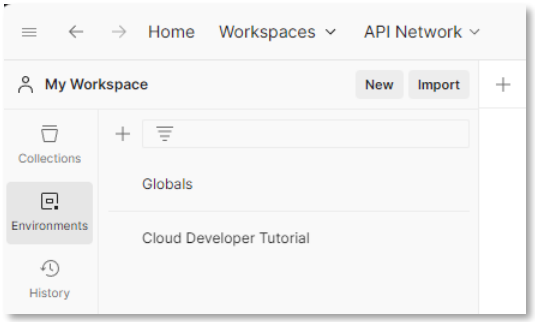
6. Select **Environments** from the Postman sidebar and click **Import**.



7. From the import dialog box, click to select **files** and navigate to the previously extracted **demo-contract-approval-app-master** folder.
8. In the **demo-contract-approval-app-master** folder, navigate to the **/postman** folder and select the **Cloud Developer Tutorial.postman_environment.json** environment file.



9. Click **Open** to import the **Cloud Developer Tutorial** collection into Postman.

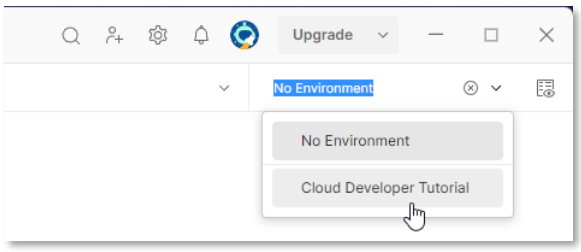


Next step:

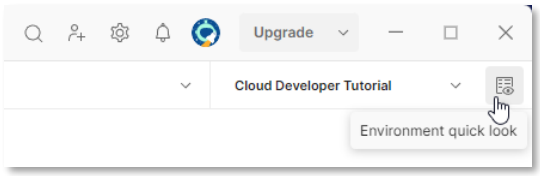
Verify the deployment of the application models using the OpenText Cloud Platform Services APIs.

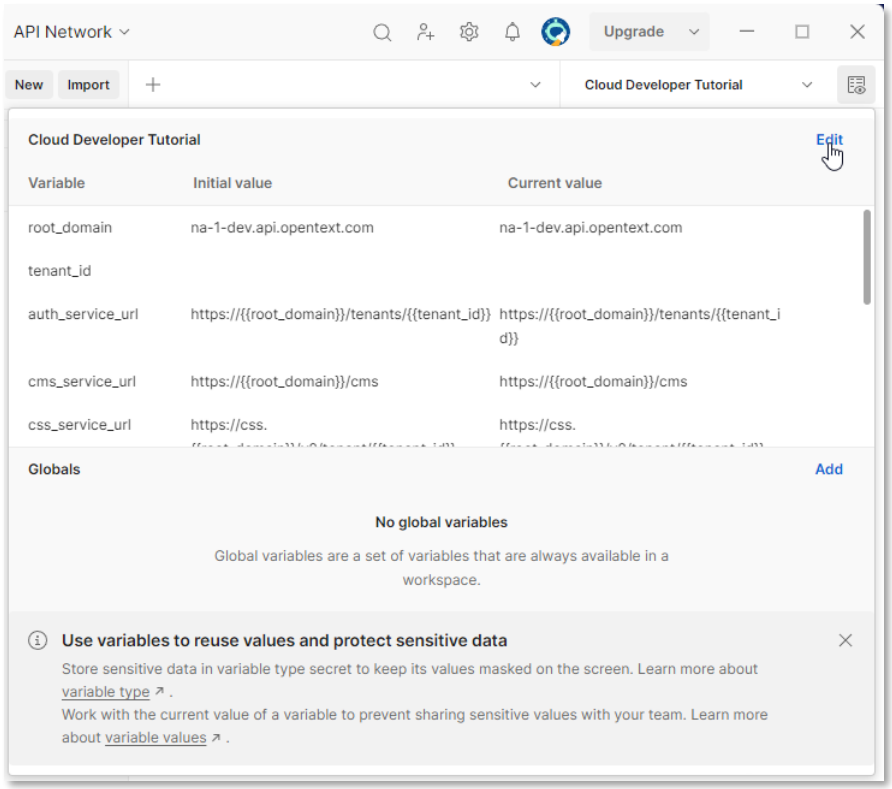
13.4 Verify the deployment of the application models using the OpenText Cloud Platform Services APIs

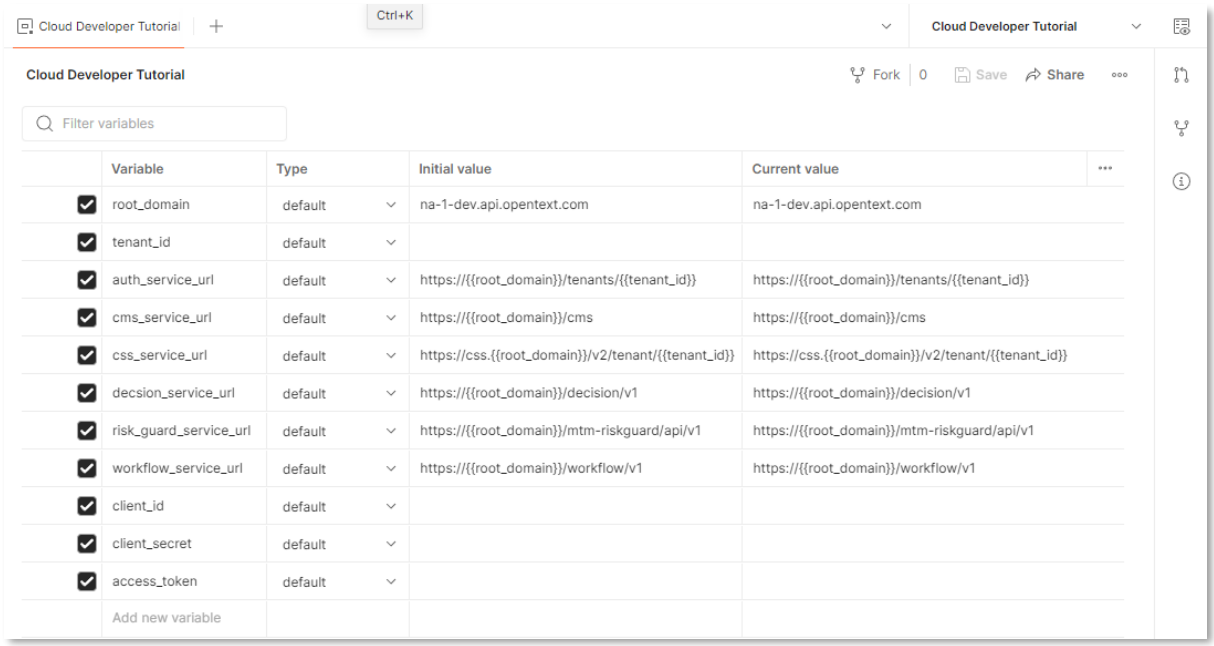
1. In Postman, from the environment selector, select the **Cloud Developer Tutorial** environment.



2. Select to open the **environment quick look** and click **Edit** to fill the environment variables.





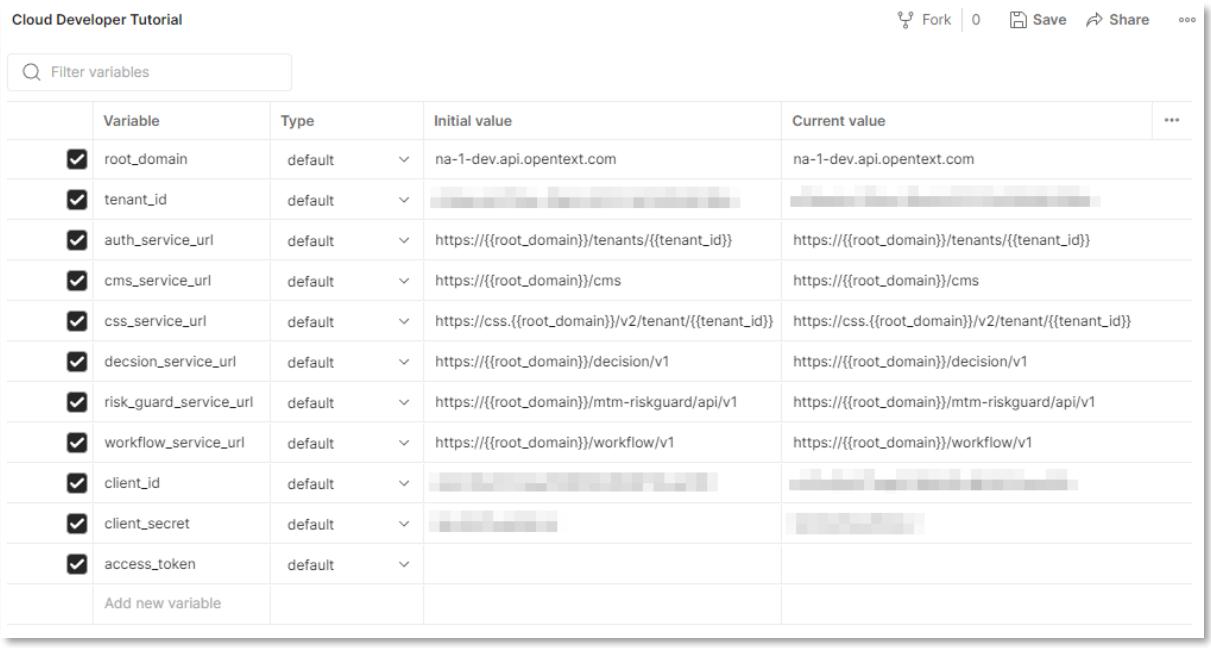


3. Use the following details to fill the Cloud Developer Tutorial environment's environment variables with the values that correspond with the deployed application:

Field	Value
tenant_id	Use the tenant id you saved after deploying the application project (that is, the text value between **'["** and **"]'**) on the first line).
client_id	Use the **Confidential Client ID** you saved after deploying the application project.
client_secret	Use the **Confidential Client Secret** you saved after deploying the application project.

Note

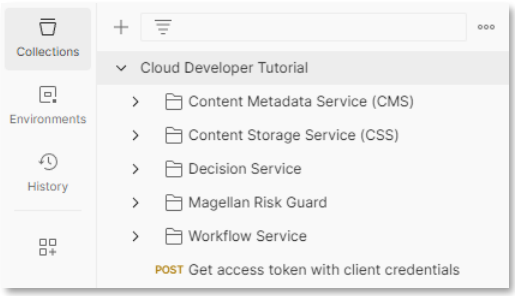
Make sure to fill both the **Initial value** and **Current value** columns for each of the environment variables (the ones that are not mentioned, you must leave as is). The initial value can be used if you want to reset the environment variables to their initial value, and the current value is the environment variable value that will be used when performing API calls.



4. Click S**ave** and close the **Cloud Developer Tutorial** environment configuration screen.

> **Note**
>
> In this context you are using the confidential client ID and confidential client secret. This is typically the type of credentials you would use to authenticate from a back-end service (that is, running on a server, and not an end user device).
>
> Later in the tutorial (when looking at the application code), you will be using a different authentication mechanism that uses the public client ID to authenticate through a trusted (OpenText Cloud Platform) login web page so that the application, which is front-end code (that is, running on the end user device, in the browser), can authenticate against the OpenText Cloud Platform APIs in a secure manner.

5. Select **Collections** from the Postman sidebar and expand the **Cloud Developer Tutorial** Postman collection.



You can see five folders representing the different OpenText Cloud Platform Services (**CMS**, **CSS**, **Decision Service**, **Magellan Risk Guard**, and **Workflow Service**) for which the collection has example requests.

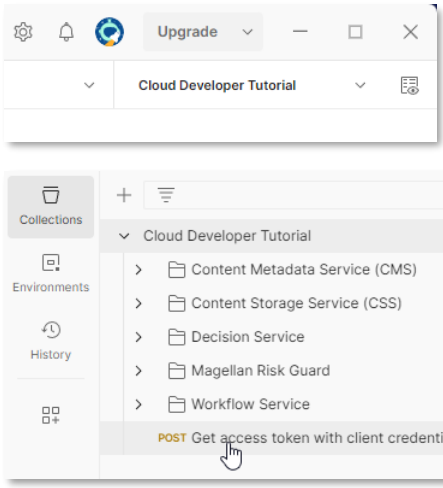
There is also a **Get access token with client credentials** POST request in the root of the collection, as this is the single request you will be using to get an access token to use for all other requests. After you run this request successfully, the **access_token** environment variable gets populated and can be used in every other request.

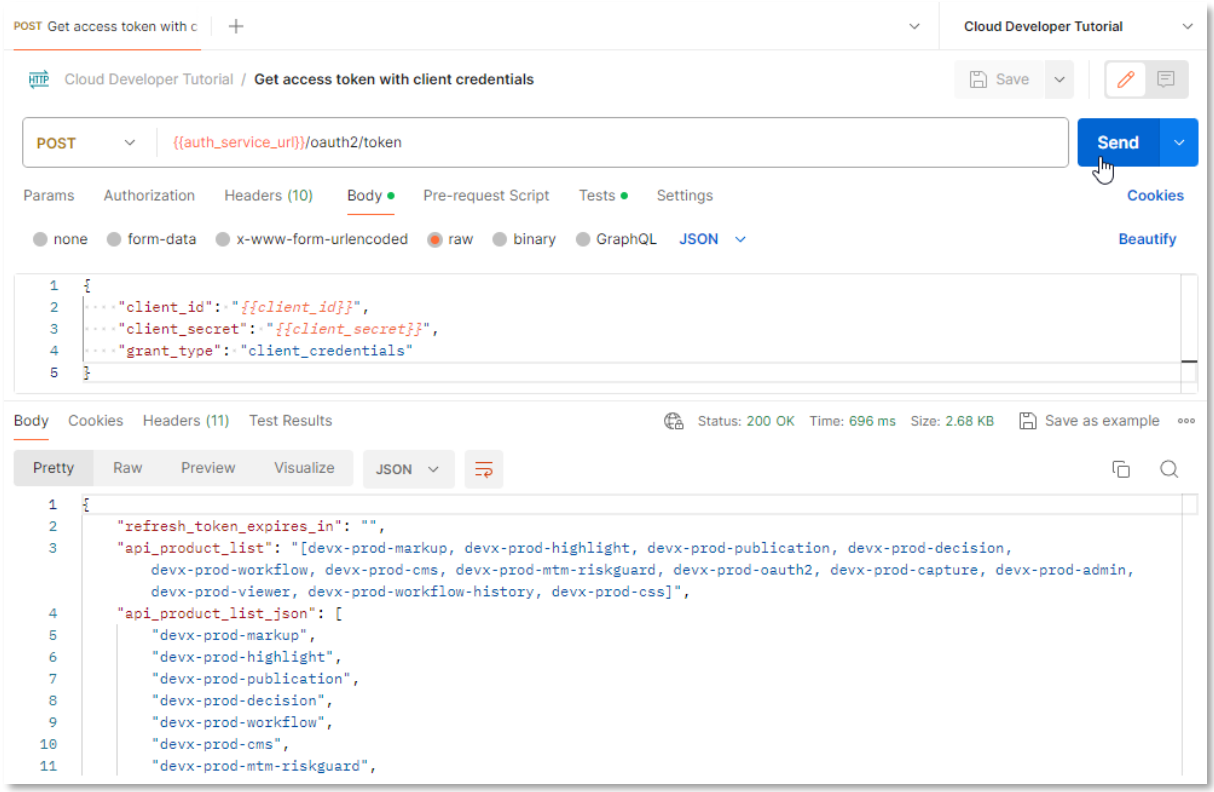
Note

If the access_token expires, the API requests will start failing with the token not valid error. To resolve this error, re-run the **Get access token with client credentials** request.

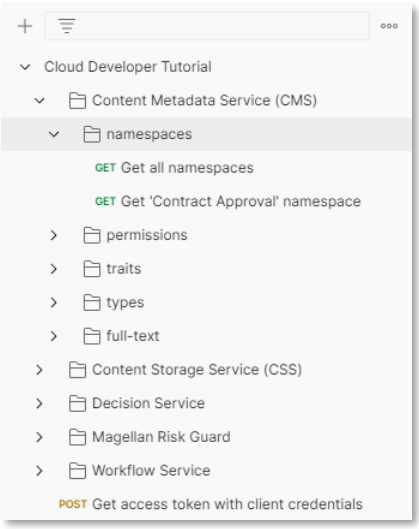
6. Run the **Get access token with client credentials** POST request to get the token.

   Make sure you have selected the (previously configured) **Cloud Developer Tutorial** environment, open the **Get access token with client credentials** request and click **Send**.



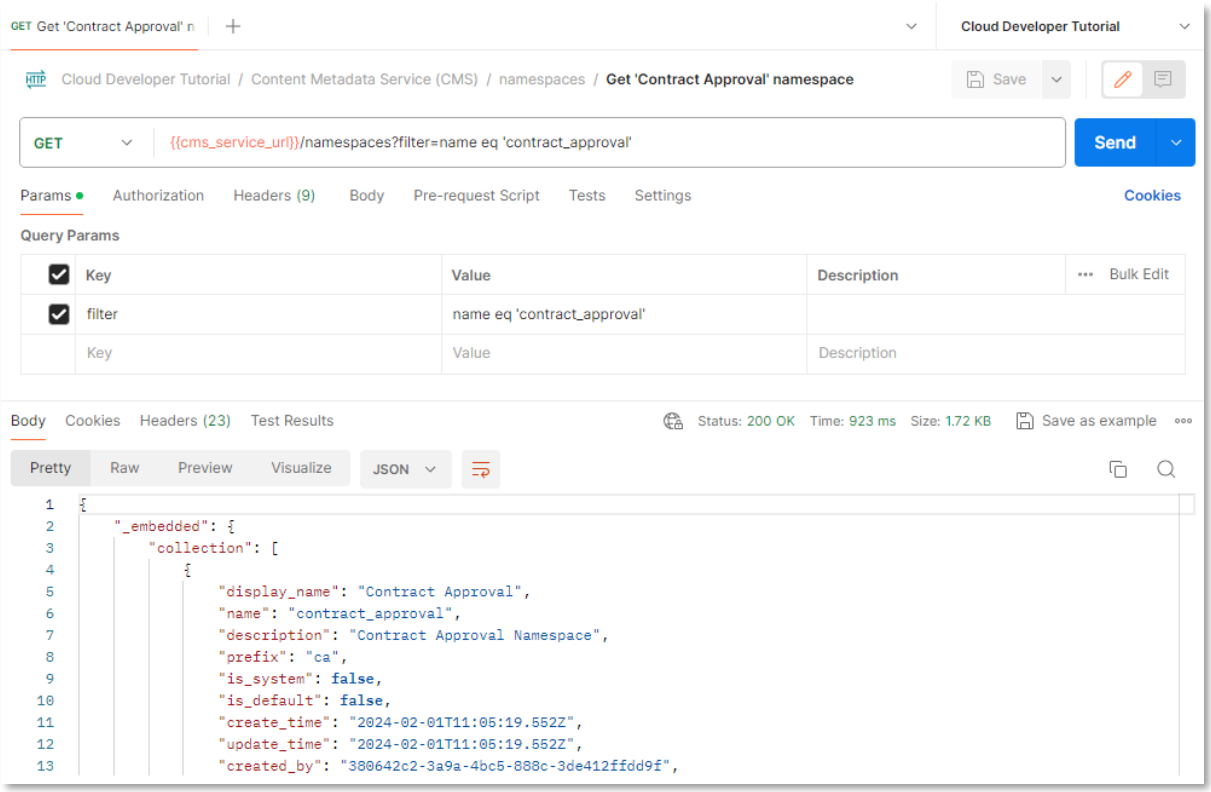


7. To verify that the **contract_approval** namespace is correctly deployed or created in CMS, select **Cloud Developer Tutorial > Content Metadata Service (CMS) > namespaces**. The examples related to the namespace requests are listed.



You will have a look at the one that allows to retrieve the namespace you created.

8. Open the **Get 'Contract Approval' namespace** request and click **Send**.



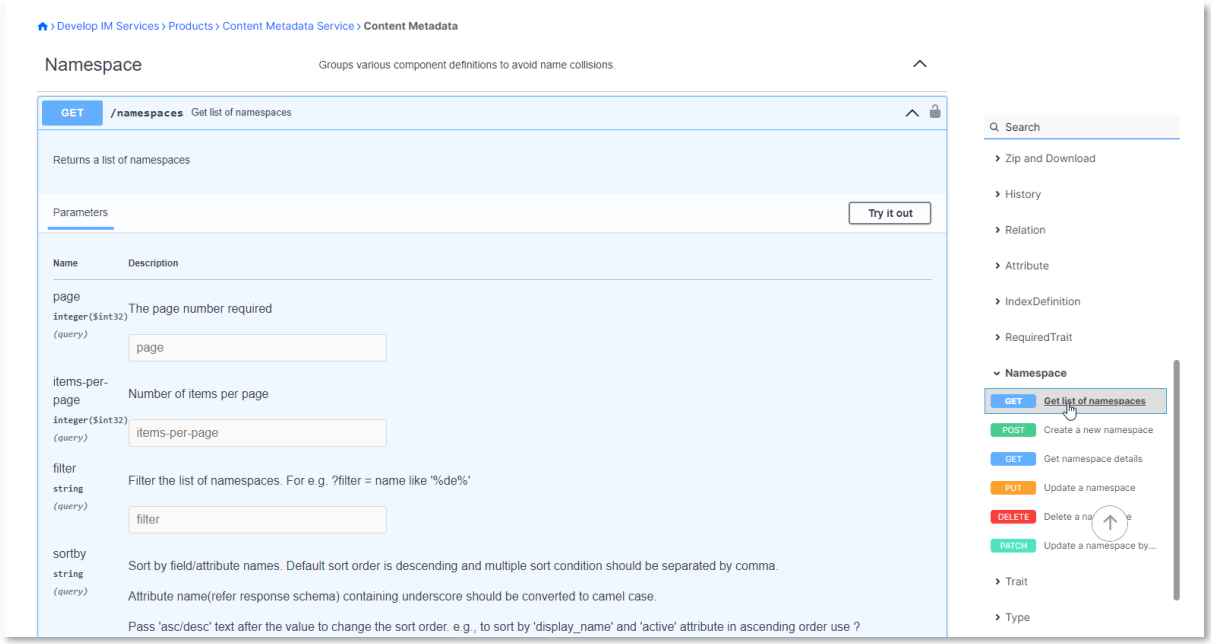
As you can see, the Contract Approval namespace is displayed, which means the collection is deployed. The **display_name**, **name**, **description**, and **prefix** attributes are shown.

Note that the request that is used, **{{base_url}}/cms/namespaces?filter=name eq 'contract_approval'** has a filter to retrieve the namespaces with a name equal to 'contract_approval'.

All the requests are based on the API reference documentation. Go to developer.opentext.com, select the appropriate request explanation. This applies to all requests in the collection. For related documentation, see:

- Content Metadata Service API reference
- Content Storage Service API reference
- Decision Service API reference
- Magellan Risk Guard API reference
- Workflow Service API reference

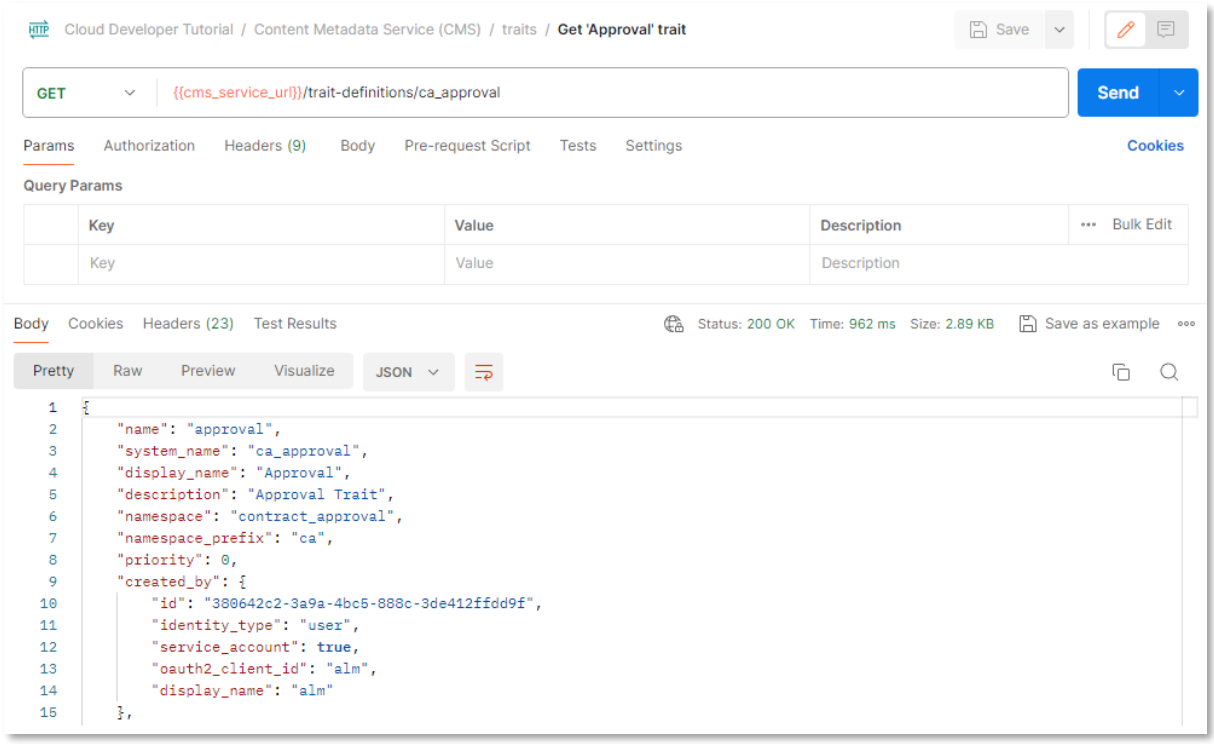
In the case of this specific request, you will find the explanation under **[GET] Get list of Namespaces** for the **Namespace** resource.



In theory, the request to use to get a specific namespace is the **Get Namespace Details** GET request, but this requires the unique ID (in UUID string format) of the namespace to be passed.

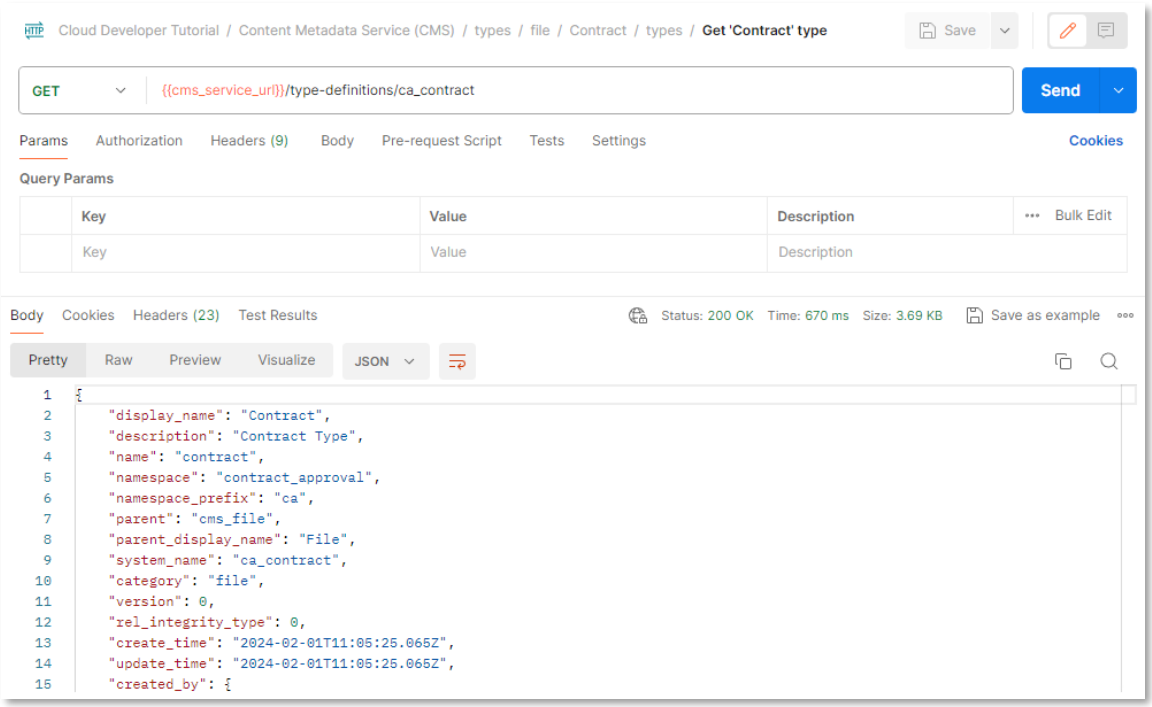
That's why the filtering mechanism on the **Get list of Namespaces** request is used (so that it always works, no matter the namespace's ID value).

9. To verify the **approval** trait, select **Collections > Content Metadata Service (CMS) > traits** and perform the **Get 'Approval' trait** GET request.

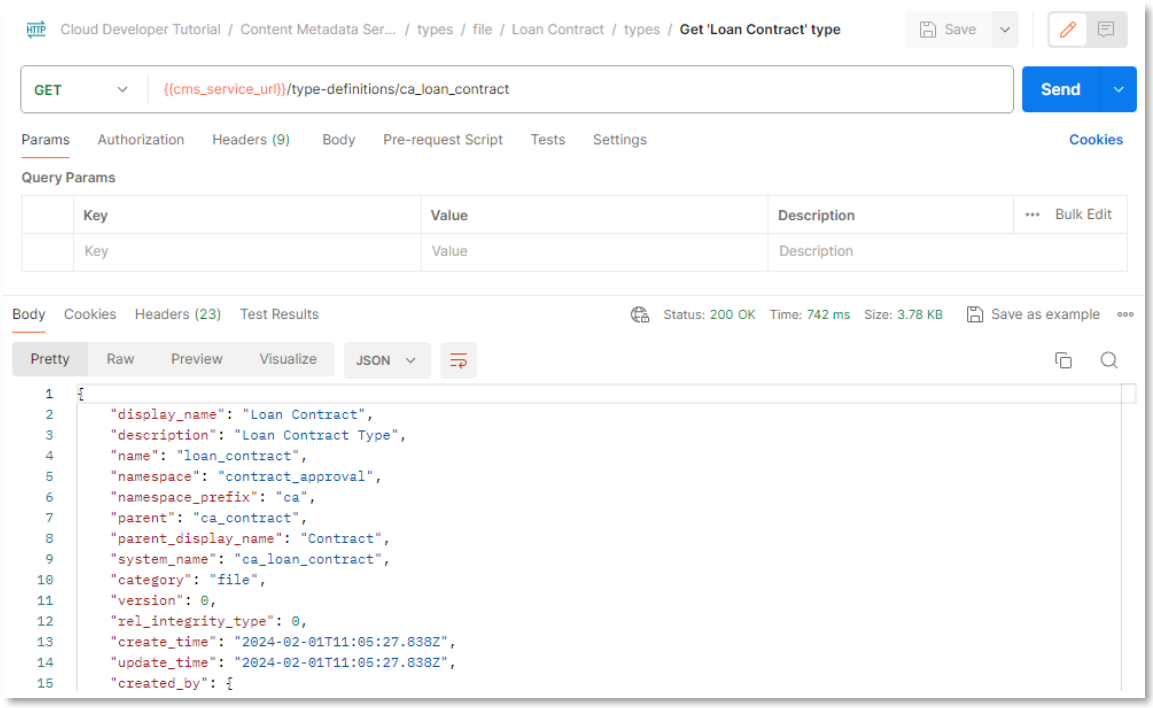


10. To verify the deployment of the **contract**, **loan_contract** and **customer** types, perform the following requests:

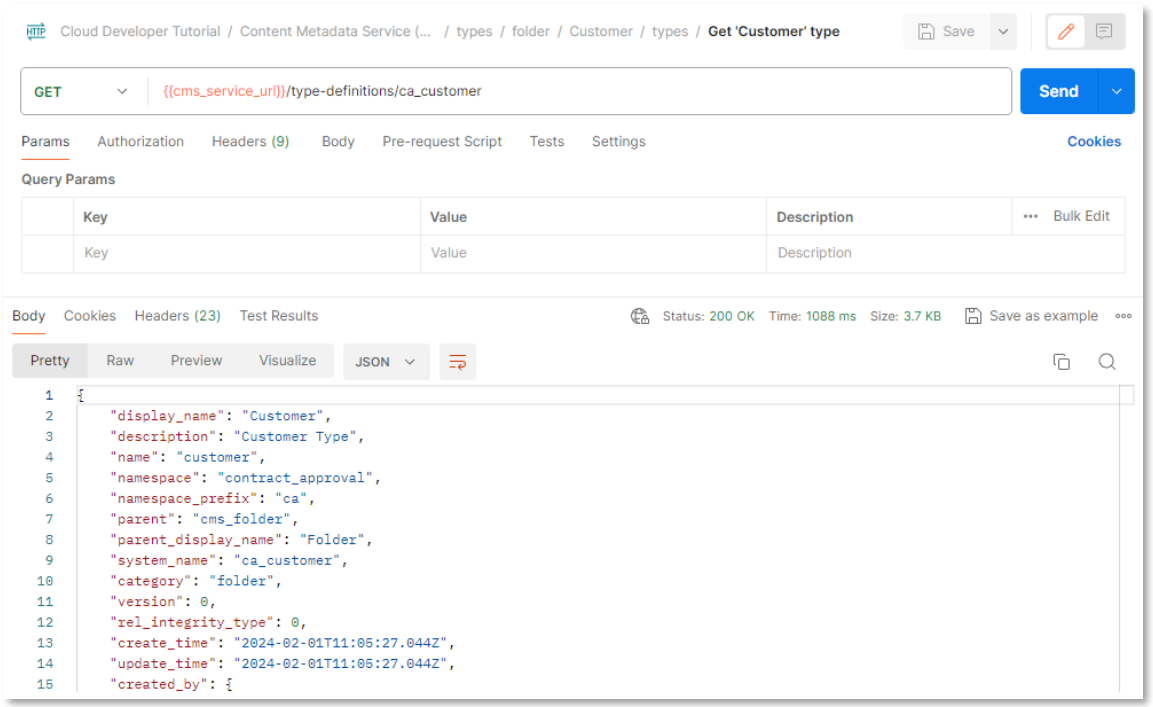
- **/Content Metadata Service (CMS)/types/file/Contract/types/Get 'Contract' type**



- **/Content Metadata Service (CMS)/types/file/Loan Contract/types/Get 'Loan Contract' type**

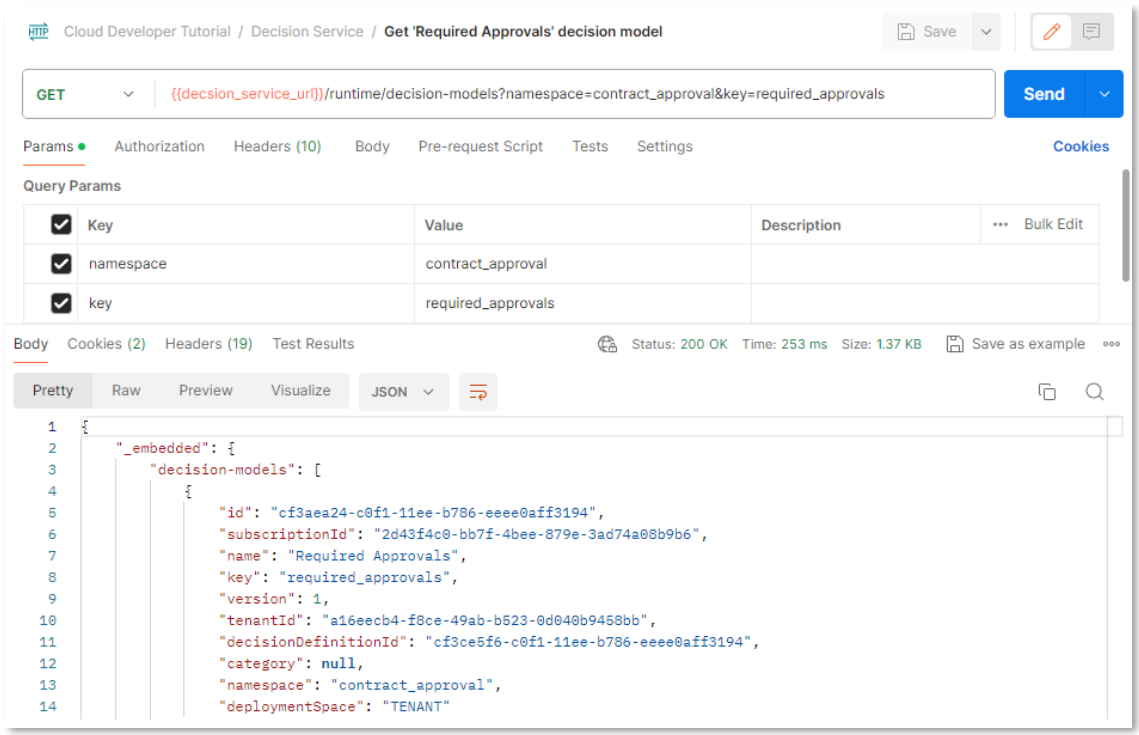


- **/Content Metadata Service (CMS)/types/folder/Customer/types/Get 'Customer' type**

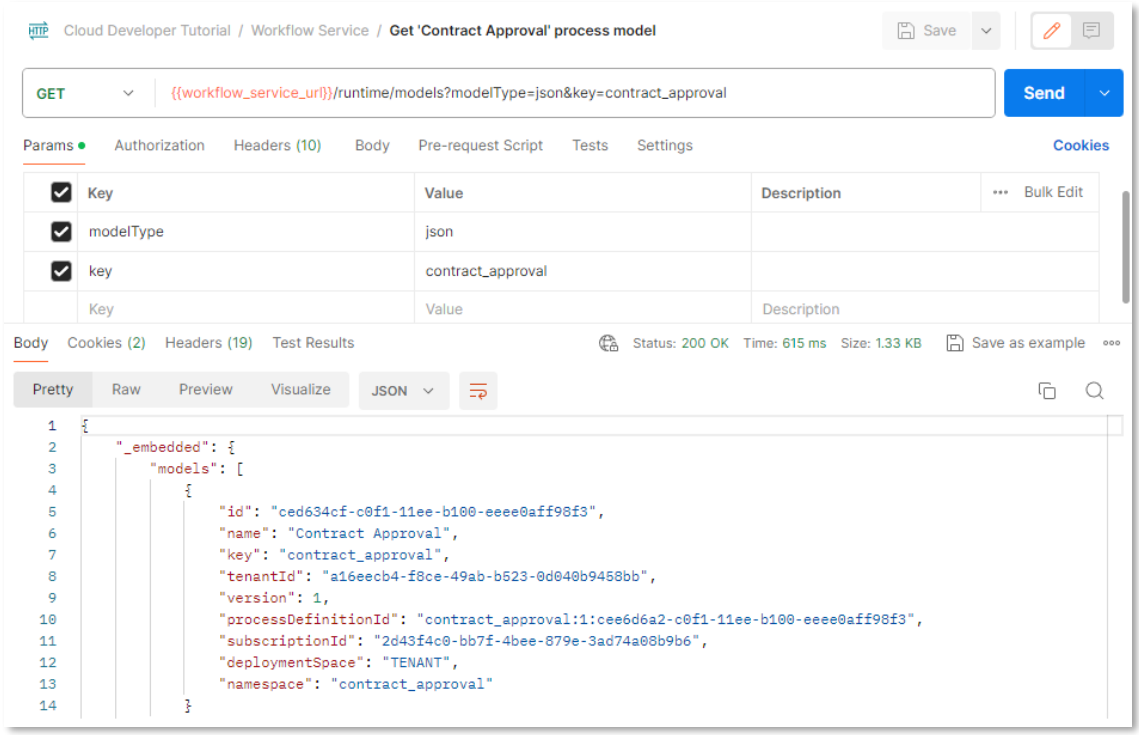


For types, the collection structure has additional levels to distinguish between the **file** and **folder** type category, and the individual types of the Contract Approval application. Requests to retrieve and manipulate **type instances** (that are created when using the application) are provided as well.

11. To verify the **required_approvals** decision model, execute the **/Decision Service/ Get 'Required Approvals' decision model** request.



12. The last deployed model to verify is the **contract_approval** workflow model. To do this, perform the **/Workflow Service/Get 'Contract Approval' process model** request.



Next exercise module:

Build the Contract Approval application.

14 [30’] Build the Contract Approval application

Learn how to:

- Import the finished Contract Approval App code into your project
- Understand the main logic of the Contract Approval application from the App.jsx React code
- Authenticate and get authorized with the OpenText Cloud Platform Services APIs
- Set the environment variables for the Contract Approval application to run from your computer
- Use the content (CSS and CMS) APIs
- Use the Workflow Service API
- Use the Viewer Service API
- Use the Magellan Risk Guard API

During this exercise module you will be going through the code of the Contract Approval application. Although you will import and not actually write the (JavaScript and React) code, except for setting the environment variables in the **.env** file, you will go over its structure, logic and how it calls the different OpenText Cloud Platform services (CSS, CMS, Decision Service, Workflow Service, Viewer Service and Magellan Risk Guard).

It is not the intent to go over every single detail of how the code was written but you will touch upon the key aspects of how the Contract Approval application is developed, so that you have a good starting knowledge to build any application of your own.

For more information on the CSS, CMS, Decision Service, Workflow Service, Viewer Service and Magellan Risk Guard APIs, you can refer to their API reference documentation, respectively CSS API Reference, CMS API reference, Decision Service API reference, Workflow Service API reference, Viewer Service API reference and Magellan Risk Guard API reference. Each service also has a more functional product documentation (that is, how to do things): CSS product documentation, CMS product documentation, Decision Service product documentation, Workflow Service product documentation, Viewing & Transformation Services product documentation, and Magellan Risk Guard Service product documentation.

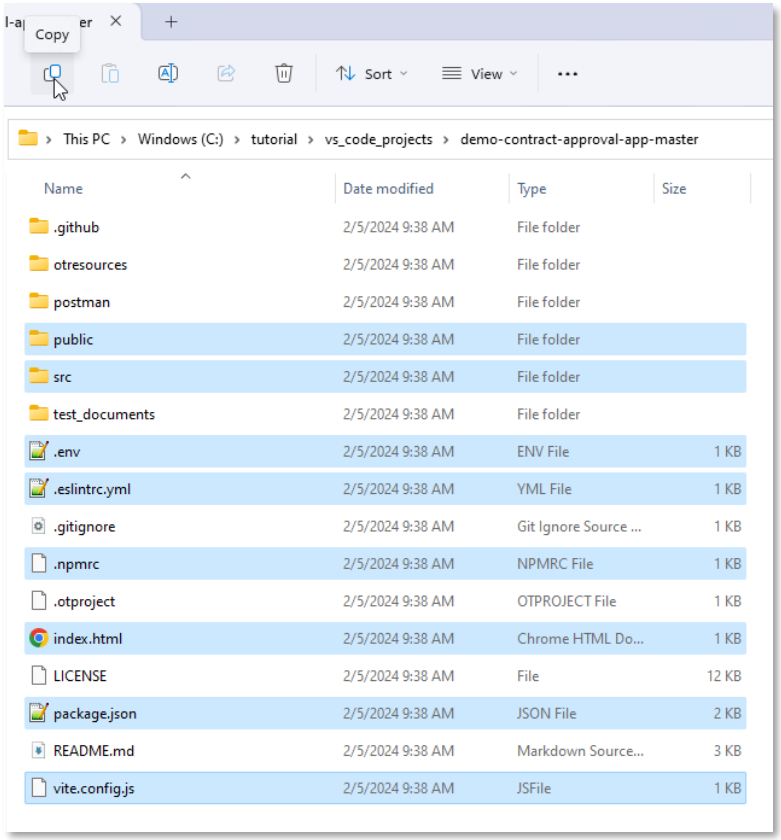
After completing this session, you will understand how the code of the Contract Approval application is written, how it consumes the deployed models, and how it calls the different OpenText Cloud Platform Services APIs.

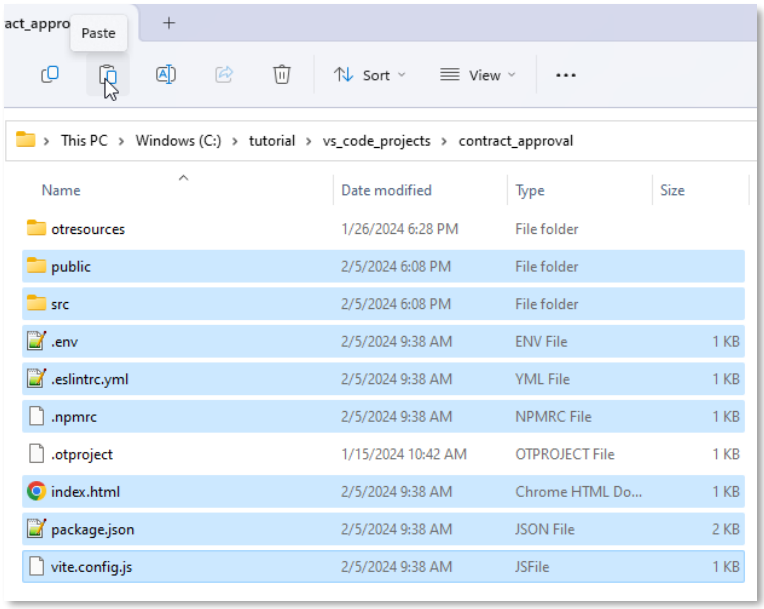
14.1 Import the Contract Approval App code into your project

The first step in this exercise is to import the code into your VS Code project. Make sure you have saved a copy of the finished Contract Approval application on your file system as described in Download the Contract Approval application.

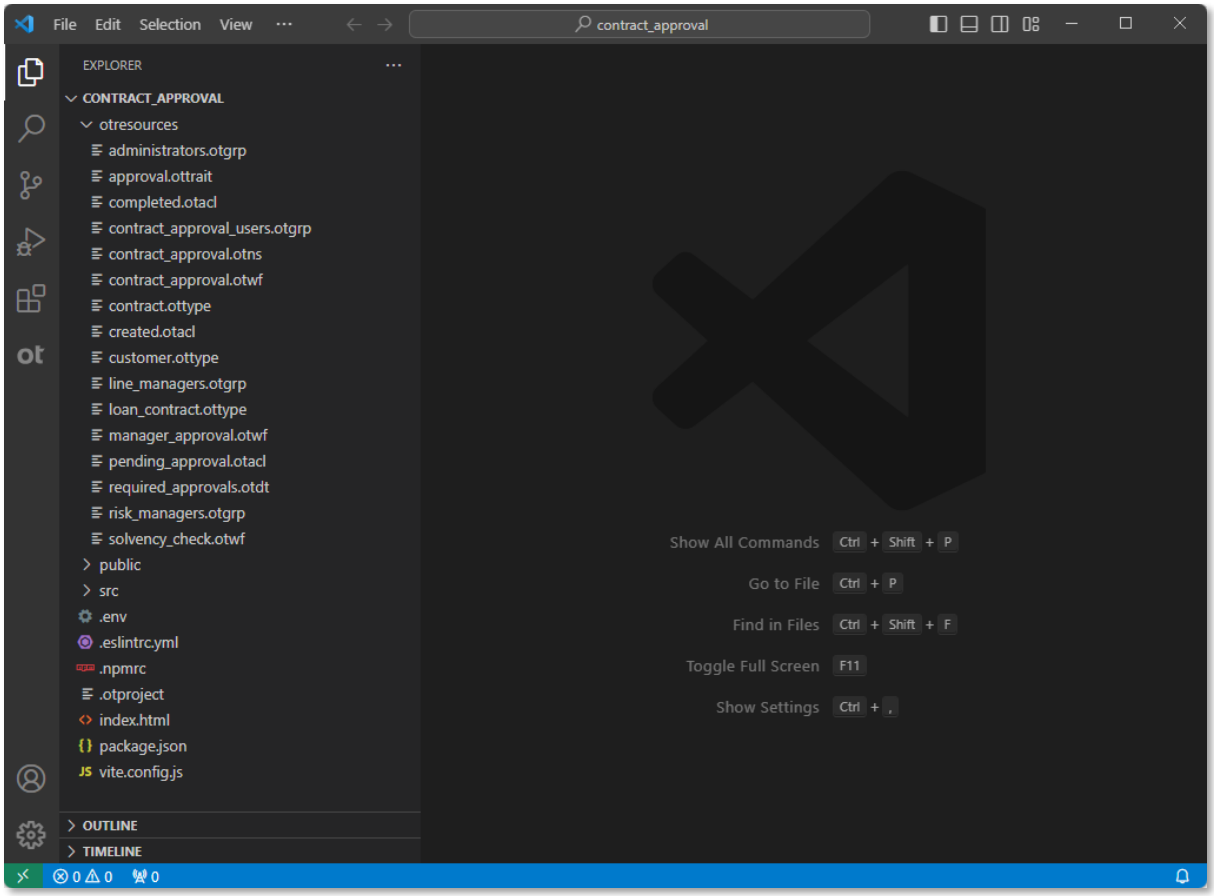
1. Navigate into the project folder of the finished Contract Approval application (that is, **demo-contract-approval-app-master** if you did not modify the project folder name) and copy the following folders and files into the root of your Contract Approval project:

   - **public** folder
   - **src** folder
   - **.env** file
   - **.eslintrc.yml** file
   - **.npmrc** file
   - **index.html** file
   - **package.json** file
   - **vite.config.js** file





2. Open your project in VS Code. The copied folders and files are visible in the **Explorer** view.



Next step:

Understand the main logic of the Contract Approval application.

14.2 Understand the main logic of the Contract Approval application

The **App.jsx** file contains the main logic for the application (React application). This file is available from the **/src** folder.

1. Open the **App.jsx** file and scroll to the return statement of the **App()** functional component that returns the JSX (XML-like syntax that allows to write HTML in React) that represents the main application user interface (UI).

```
return (
  <div className="App">
    <Header />
    {
      groups.includes('contract_approval_users')
        ? (
          <div className="page-content">
            <Tabs
              orientation="horizontal"
              value={tabValue}
              onChange={handleTabChange}
            >
              <Tab className="tab-caption" label="Created Contracts" />
              {
                groups.includes('line_managers')
                && <Tab className="tab-caption" label="Line Manager Tasks" />
              }
              {
                groups.includes('risk_managers')
                && <Tab className="tab-caption" label="Risk Manager Tasks" />
              }
              <Tab className="tab-caption" label="All Contracts" />
            </Tabs>
            <ApplicationProvider>
              <TabPanel value={tabValue} index={tabIndex}>
                <CreatedContractList />
              </TabPanel>
              {
                groups.includes('line_managers')
                && (
                  // eslint-disable-next-line no-plusplus
                  <TabPanel value={tabValue} index={++tabIndex}>
                    <TasksList taskName="Line Manager Approval" />
                  </TabPanel>
                )
              }
              {
                groups.includes('risk_managers')
                && (
                  // eslint-disable-next-line no-plusplus
                  <TabPanel value={tabValue} index={++tabIndex}>
                    <TasksList taskName="Risk Manager Approval" />
                  </TabPanel>
                )
              }
              <TabPanel value={tabValue} index={tabIndex + 1}>
                <ContractList />
              </TabPanel>
            </ApplicationProvider>
          </div>
        )
        : (
          <div className="page-content">
            <p>You are not authorized to use this application</p>
            <button type="button" style={{ margin: '0.50rem' }} onClick={signoutRedirect}>
              Logout
            </button>
          </div>
        )
    }
  </div>
);
}
```

- The **Header** (cf. **Header.jsx** for more details) component is the header bar of the Contract Approval application. It displays the OpenText logo, the application name, and the username of the connected user. The username is also a menu that allows the connected user to log out from the application.
- The **groups.includes('contract_approval_users')** statement is used to verify that the user is part of the **contract_approval_users** group (or one of its sub groups) to determine whether they are allowed to use the application.
- If the user is authorized to use the application, a four (horizontal) tabs UI layout is displayed, with the **Tabs** component and its **Tab** child components representing the tabs, and the **ApplicationProvider** component with its **TabPanel** child components representing the views to show when clicking the tabs.
  The four horizontally stacked tabs generated by this part of the code are:
  - The **Created Contracts** tab with the **CreatedContractList** component providing the “created contracts list” view to show all newly created contracts, that is, where status = ‘CREATED’.
  - The **Line Manager Tasks** tab with the **TasksList** component providing the “Line Manager Approval” tasks view to show all approval tasks for the Line Manager. This tab is only shown if the list of groups, the authenticated user is a member of, includes the **line_managers** group.
  - The **Risk Manager Tasks** tab with the **TasksList** component providing the “Risk Manager Approval” tasks view to show all approval tasks for the Risk Manager. This tab is only shown if the list of groups, the authenticated user is a member of, includes the **risk_managers** group.
  - The **All Contracts** tab with the **ContractList** component providing the “all contracts” view to show all contracts in the application, independently of their status.

  If the user is not authorized to use the application, they are presented with a “not authorized” message and a button to log out.

Next step:

Understand the main logic of the Contract Approval application.

14.3 Authenticate and get authorized with the OpenText Cloud Platform Services APIs

Beyond the main application UI, the **App.jsx** file also contains code that manages the authentication and authorization logic in combination with the code in the following files:

- **WrappedSecuredApp.jsx** (under **/src**)
- **OidcConfig.js** (under **/src/authorization**)

1. Open the **WrappedSecuredApp.jsx** file. This file is the component that “wraps” the application with security.

```
import { AuthProvider } from 'react-oidc-context';
import App from './App';
import OidcConfig from './authorization/OidcConfig';

function WrappedSecuredApp() {
  return (
    /* eslint-disable-next-line react/jsx-props-no-spreading */
    <AuthProvider {...OidcConfig}>
      <App />
    </AuthProvider>
  );
}
```

Note

To authenticate with the OpenText Cloud Platform Services APIs, the Contract Approval App uses **OpenID Connect (OIDC)**, an authentication protocol that sits on top of OAuth 2.0. More specifically, using the **Public Client ID** for the deployed application, an authentication flow that uses an external (to the Contract Approval application) login screen is used to authenticate with the OpenText Cloud Platform, allowing to subsequently call the different Services APIs.

- The **AuthProvider** component wraps the security around the **App** component (from the **App.jsx** file). It is imported from the **react-oidc-context** library to deliver the OIDC authentication mechanism. The authentication and authorization parameters for the security to apply to the contained **App** are defined through the **OidcConfig** configuration object, which the **AuthProvider** component takes as a parameter.
- The **OidcConfig** configuration object is imported from **OidcConfig.js**

Note

The **react-oidc-context** library is available to be imported/referenced, because it is defined as a dependency in the **package.json** file. To check the different project dependencies, you can open the package.json file.

The package.json file is the main configuration file for a Node.js project, and the Contract Approval application is a React application set up to run in Node.js.

2. Open the **OidcConfig.js** file to examine the different OIDC (that is, authentication and authorization) configuration parameters.

```
const OidcConfig = {
  authority: `${process.env.REACT_APP_BASE_SERVICE_URL}/tenants/${process.env.REACT_APP_TENANT_ID}`,
  client_id: process.env.REACT_APP_CLIENT_ID,
  redirect_uri: process.env.REACT_APP_REDIRECT_URI,
  response_type: 'code',
  scope: 'openid otds:groups',
  post_logout_redirect_uri: process.env.REACT_APP_REDIRECT_URI,
  onSigninCallback: () => {
    window.history.replaceState({}, document.title, window.location.pathname);
  },
};
```

- The **authority** is the URL of the authority (or identity provider) and it is used as the main URL for the authentication flow.
- The **client_id** is the **Public Client ID** for the deployed application.
- The **redirect_url** is passed as part of the authentication flow to allow redirecting back into the (Contract Approval) application once authenticated.
- The **response_type** tells the authorization server which grant to execute.
- The **scope** is a space-delimited list of permissions that the application requires.
- The **post_logout_redirect_uri** defines the application page to redirect to after the logout completes.
- The **onSigninCallback** is the callback function that defines the logic to execute when the sign in occurs. Here the authorization code is removed from the url for security reasons.

As you can see, to construct the parameter values, **process.env.<ENVIRONMENT_VARIABLE>** is used, which corresponds with the environment variables filled in the **.env** file (available in the project root).

Note

The **.env** file needs to be correctly filled for the application to work and how to do this is described in the Set the environment variables section.

3. Go back to the **App.jsx** file to examine the use of the security mechanism and configuration provided by the wrapping **WrappedSecuredApp** component and configured by the **OidcConfig** configuration object.

```
const {
  activeNavigator,
  error,
  isAuthenticated,
  isLoading,
  user,
  signinRedirect,
  signoutRedirect,
} = useAuth();
const [isAppLoaded, setIsAppLoaded] = useState(false);
const [tabValue, setTabValue] = useState(0);
const [groups, setGroups] = useState([]);

const handleTabChange = (event, newTabValue) => {
  setTabValue(newTabValue);
};

useEffect(() => {
  if (!isLoading) {
    if (!isAuthenticated) {
      signinRedirect();
    } else if (!isAppLoaded && isAuthenticated) {
      setGroups(
        jwtDecode(user.id_token)
          .grp
          .map((group) => group.substring(0, group.indexOf('@'))),
      );
      setIsAppLoaded(true);
    }
  }
}, [isAuthenticated, isLoading, isAppLoaded]);

switch (activeNavigator) {
  case 'signoutRedirect':
    return (
      <div className="loading">
        <CircularProgress color="inherit" />
      </div>
    );
  default:
}

if (error) {
  return (
    <div className="error">
      <Alert severity="error">
        Authentication error:
        {' '}
        {error.message}
      </Alert>
    </div>
  );
}

if (!isAppLoaded) {
  return (
    <div className="loading">
      <CircularProgress color="inherit" />
    </div>
  );
}
```

- The **useAuth** method is imported from the **react-oidc-context** library to provide the **activeNavigator**, **error**, **isAuthenticated**, **isLoading**, **user**, **signinRedirect**, and **signoutRedirect**. These are used for signing in (if not authenticated), signing out, returning user information (including user groups), handling errors, and managing whether the application is still loading and authenticating.
- The **jwtDecode** method is imported from the **jwt-decode** library to allow decoding the JWT id token available from the (authenticated) user object. This is important because it allows retrieving the groups the user belongs to in order to display the correct application tabs.

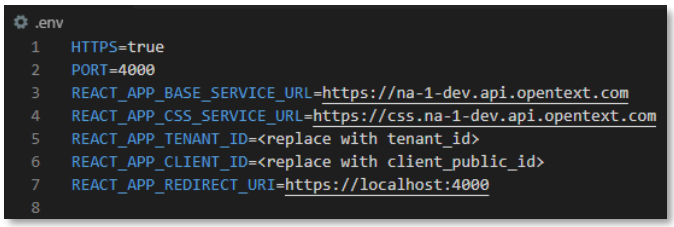
Next step:

Set the environment variables.

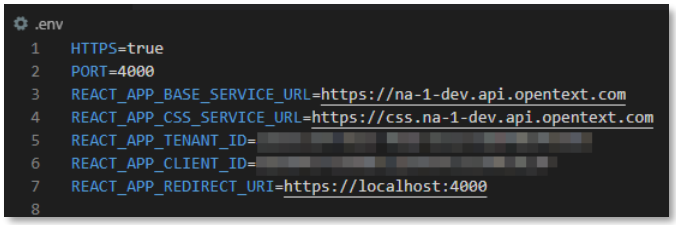
14.4Set the environment variables

In the previous section on authentication and authorization, it is explained that to access the different OIDC configuration parameters, **process.env.<ENVIRONMENT_VARIABLE)** is being used.
The corresponding environment variables are configured in the **.env** file.

1. Open the **.env** file (available from the project root folder) to configure the environment variables to be able to run the Contract Approval application.



2. Update the **.env** file by replacing **<replace with tenant_id>** and **<replace with client_public_id>** with the corresponding values you saved after deploying the application project. More specifically, use the tenant id (from the text **in tenants '["<tenant id>"]'**) and **Public Client ID**.



3. **Save** and close the **.env** file.

Next step:

Use the content (CSS and CMS) APIs.

14.5Use the content (CSS and CMS) APIs

To store and manage content in the OpenText Cloud Platform, two services need to be used. The Content Storage Service (CSS) API allows to store and retrieve the actual (binary) content files (that is, the different file renditions), whereas the Content Metadata Service (CMS) API can be used to manage the associated file object metadata. It is through combining these two APIs that you can manage file objects.

Beyond file objects (that is, objects with content), the CMS API also provides the capability to manage folders (containers for other folders, files, and objects), objects (contentless objects) and relations.

To understand how to use the CSS and CMS APIs, let's have a closer look at the first of the four application views (that is, the view for the first tab). The "created contracts list" view provided through the **CreatedContractList** component does not only show the newly created contracts (status = 'CREATED'), but it also provides the button to add new contracts to the system.

1. Open the **CreatedContractList.jsx** file from the **/src/components** folder.

```
/**
 * This view displays the list of created contracts.
 * From here the user can request approval for any of them.
 */
function CreatedContractList() {
  const { user } = useAuth();
  const [state, setState] = useState(
    {
      contracts: [],
      pendingApprovalAclId: '',
      completedAclId: '',
      openContractDetails: false,
      selectedContract: { properties: {} },
      openAddContract: false,
      addNumberOfContracts: 0,
      pageNumber: 0,
      count: -1,
      openDocumentDialogView: false,
      fileId: '',
      fileName: '',
      showBackdrop: false,
      showSnackBar: false,
      snackBarMessage: '',
      snackBarSeverity: 'success',
    },
  );
  const didMountRef = useRef(false);
  const addNumberOfContractsRef = useRef(state.addNumberOfContracts);
  const pageNumberRef = useRef(state.pageNumber);

  const raiseError = useCallback((errorMessage) => {
    setState((prevState) => ({
      ...prevState,
      snackBarSeverity: 'error',
      snackBarMessage: errorMessage,
      showSnackBar: true,
    }));
  }, []);

  const getAcls = useCallback(() => {
    if (
      state.pendingApprovalAclId.length === 0
      || state.completedAclId.length === 0) {
      let pendingApprovalAclFound = false;
      let completedAclFound = false;
      axios({
        method: 'get',
        url: `${baseUrl}/cms/permissions?filter=name eq "pending_approval" or name eq "completed"`,
        headers: {
          Authorization: `Bearer ${user.access_token}`,
        },
      }).then((res) => {
```

2. Scroll to the return statement of the **CreatedContractList()** functional component as it represents the “created contract list” UI.

```
return (
  <div>
    <Button variant="contained" color="primary" disabled={!user.profile.preferred_username} startIcon={<Add />}
    <div className="content-header">All created contracts</div>
    <TableContainer component={Paper}>
      <Table size="small" aria-label="a dense table">
        <TableHead>
          <TableRow>
            <TableCell>Contract name</TableCell>
            <TableCell align="left">Creation date</TableCell>
            <TableCell align="left">Value</TableCell>
            <TableCell align="left">Risk classification</TableCell>
            <TableCell align="left">View document</TableCell>
            <TableCell align="left">Action</TableCell>
            <TableCell align="left" />
          </TableRow>
        </TableHead>
        <TableBody>
          {state.contracts.map((row) => (
            <TableRow key={row.id}>
              <TableCell component="th" scope="row">
                {row.name}
              </TableCell>
              <TableCell align="left">{getDateValue(row.create_time)}</TableCell>
              <TableCell align="left">{row.properties.value}</TableCell>
              <TableCell align="left"><RiskClassification row={row} /></TableCell>
              <TableCell align="left">
                <Button size="small" variant="outlined" color="primary" onClick={() => openDocumentDialogView(r
              </TableCell>
              <TableCell align="left">
                <Button size="small" variant="outlined" color="primary" onClick={() => startContractForApproval
              </TableCell>
              <TableCell align="left">
                <IconButton size="small" variant="outlined" color="primary" title="Show details" onClick={() =>
                  <ArrowForwardIos />
                </IconButton>
              </TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>
    <Pagination
      pageNumber={state.pageNumber}
      count={state.count}
      handlePageNumber={handlePageNumber}
    />
    <ContractDetails
      open={state.openContractDetails}
      selectedContract={state.selectedContract}
      parentRaiseError={raiseError}
      onClose={handleCloseContractDetails}
    />
    <AddContract
      open={state.openAddContract}
      onAddContract={handleContractAdded}
      onClose={handleCloseAddContract}
    />
    <DocumentDialogView
      open={state.openDocumentDialogView}
      fileId={state.fileId}
      onClose={handleCloseDocumentDialogView}
    />
    <Backdrop style={{ zIndex: 9999 }} open={state.showBackdrop}>
      <CircularProgress color="inherit" />
    </Backdrop>
    <Snackbar
      anchorOrigin={{
```

- The “Add” **Button** at the top of created contract list allows adding new contracts via the **AddContract** component.
- The **TableContainer** component provides the table layout to display the list of the different contracts that have the ‘CREATED’ status.
- The **TableHead** and **TableBody** components represent the column headers and the rows with the values, each table row in the table body will show the following information for the displayed contract:
  - **Contract name**
  - **Creation date**
  - **Value**
  - **Risk classification**
  - **Original** (this is not a property value but a button to open the actual document in the viewer from the Viewer Service through the **DocumentDialogView** component)
  - **Request Approval** (this is not a property value but a button to start the contract approval workflow)
  - An **arrow icon button** allowing to open the contract details (the contract attributes screen) through the **ContractDetails** component

  The table rows are generated by iterating over the **state.contracts** array, and in its turn **state.contracts** is populated by the **getContracts** method (called when the created contracts list needs updating).

3. If you look at the **getContracts** method, you easily recognize the (axios) GET request to the **/cms/instances** endpoint of the **CMS** API with the **user.access_token** (bearer) token passed as the **Authorization** header.

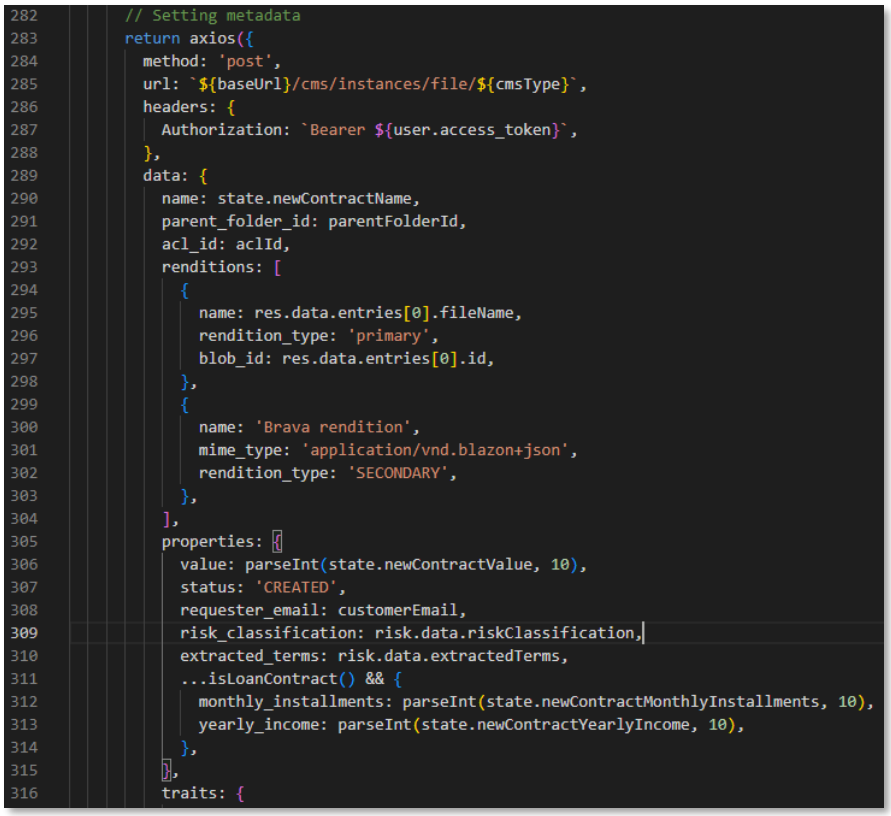
```
117   const getContracts = useCallback(() => {
118     if (user.profile.preferred_username) {
119       setState((prevState) => ({ ...prevState, showBackdrop: true }));
120       axios({
121         method: 'get',
122         url: `${baseUrl}/cms/instances/file/ca_contract/?include-total=true&sortby=create_time desc&filter=status eq "CREATED"
123         headers: {
124           Authorization: `Bearer ${user.access_token}`,
125         },
126       }).then((res) => {
127         setState((prevState) => ({
128           ...prevState,
129           contracts: res.data && res.data._embedded ? res.data._embedded.collection : [],
130           count: res.data.total,
131         }));
132       }).catch((error) => {
133         let errorMessage = 'Could not get contracts: ';
134         if (error.response != null && error.response.data != null) {
135           errorMessage += error.response.data.exception;
136         } else {
137           errorMessage += error.message;
138         }
139         raiseError(this, errorMessage);
140       }).finally(() => {
141         setState((prevState) => ({ ...prevState, showBackdrop: false }));
142       });
143     } else {
144       setState((prevState) => ({ ...prevState, contracts: [], count: -1 }));
145     }
146   }, [
147     raiseError,
148     user.access_token,
149     user.profile.preferred_username,
150     state.pageNumber,
151   ]);
```

4. To understand how to use the CSS and CMS APIs to create a new contract, open the **AddContract.jsx** file (again from the **/src** folder) and first look for the **// Adding Contract** comment in the code.

```
// Adding Contract
const formData = new FormData();
formData.append(
  'file',
  state.selectedFile,
  state.selectedFile.name,
);
axios.post(
  `${process.env.REACT_APP_CSS_SERVICE_URL}/v2/tenant/${process.env.REACT_APP_TENANT_ID}/content`,
  formData,
  {
    headers: {
      'Content-Type': 'multipart/form-data',
      Authorization: `Bearer ${user.access_token}`,
    },
  },
).then((res) => {
  let cmsType;
  if (isLoanContract()) {
    cmsType = 'ca_loan_contract';
  } else {
    cmsType = 'ca_contract';
  }
```

- The (axios) POST call uploads the selected (contract) file to CSS.
- Once the file is uploaded to CSS, the type of contract to create is determined using the system names, **ca_loan_contract** or **ca_contract**, where the ca prefix represents the namespace.

5. After uploading the file to CSS, the second step in the contract creation code is to create the metadata object. Look for **// Setting metadata** comment.



```
// Setting metadata
return axios({
  method: 'post',
  url: `${baseUrl}/cms/instances/file/${cmsType}`,
  headers: {
    Authorization: `Bearer ${user.access_token}`,
  },
  data: {
    name: state.newContractName,
    parent_folder_id: parentFolderId,
    acl_id: aclId,
    renditions: [
      {
        name: res.data.entries[0].fileName,
        rendition_type: 'primary',
        blob_id: res.data.entries[0].id,
      },
      {
        name: 'Brava rendition',
        mime_type: 'application/vnd.blazon+json',
        rendition_type: 'SECONDARY',
      },
    ],
    properties: {
      value: parseInt(state.newContractValue, 10),
      status: 'CREATED',
      requester_email: customerEmail,
      risk_classification: risk.data.riskClassification,
      extracted_terms: risk.data.extractedTerms,
      ...isLoanContract() && {
        monthly_installments: parseInt(state.newContractMonthlyInstallments, 10),
        yearly_income: parseInt(state.newContractYearlyIncome, 10),
      },
    },
    traits: {
```

- Another (axios) POST call creates the file metadata object in CMS. It passes the contract's properties, rendition (linked to the previously uploaded CSS file via the **blob_id**), and traits as payload.

Next step:

Use the Workflow Service API.

14.6Use the Workflow Service API

To illustrate the use of the Workflow Service API, you will look at the piece of code where the approval workflow is launched.

1. Open the **CreatedContractList.jsx** file again and locate the **startContractForApproval** method.

```
202   const startContractForApproval = (contractId) => {
203     setState((prevState) => ({ ...prevState, showBackdrop: true }));
204
205     const data = {
```

2. First, the **data** object to pass as payload to the Workflow Service API gets constructed. It contains the **processDefinitionKey** matching the name of the workflow model and allowing to identify the workflow to create an workflow instance for, a user-friendly **name**, and the process **variables** to pass to the new workflow instance (cf. Create the Contract Approval workflow).

```
205     const data = {
206       processDefinitionKey: 'contract_approval',
207       name: 'Approve contract',
208       outcome: 'none',
209       variables: [
210         {
211           name: 'base_url',
212           value: baseUrl,
213         },
214         {
215           name: 'contract_id',
216           value: contractId,
217         },
218         {
219           name: 'pending_approval_acl_id',
220           value: state.pendingApprovalAclId,
221         },
222         {
223           name: 'completed_acl_id',
224           value: state.completedAclId,
225         },
226       ],
227       returnVariables: true,
228     };
```

3. Secondly, the Workflow Service API that launches the workflow instance (with the **data** payload) gets called via a (axios) POST call.

```
230     axios({
231       method: 'post',
232       url: `${baseUrl}/workflow/v1/process-instances`,
233       headers: {
234         Authorization: `Bearer ${user.access_token}`,
235       },
236       data,
237     }).then(() => {
238       setState((prevState) => ({ ...prevState, snackBarMessage: 'Approval requested successfully.' }));
239       setState((prevState) => ({ ...prevState, showSnackBar: true }));
240       getContracts();
241     }).catch((error) => {
242       const statusCode = error.response.status;
243       let errorMessage = 'Error requesting approval: ';
244       if (statusCode === 400) {
245         errorMessage += error.response.data.exception;
246       } else {
247         errorMessage += error.message;
248       }
249       raiseError(errorMessage);
250     }).finally(() => {
251       setState((prevState) => ({ ...prevState, showBackdrop: false }));
252     });
253   };
```

4. To have an example of how to call the task related service endpoints, feel free to explore the **Tasks.js** file under **/src/services/workflow**.

   The tasks API is used in the Contract Approval application to interact with the contract approval workflow's manual user tasks (that is, the Line Manager approval tasks and the Risk Manager approval tasks). More specifically, it provides the different REST API endpoints that are called from the **getTasks()**, **claimTask()**, and **completeTask()** methods from the **Tasks** utility class.

Next step:

Use the Viewer Service API.

14.7Use the Viewer Service API

The Viewer Service exposes a powerful file viewer that can be configured in a very granular way.

To understand how to use and configure the viewer from the Viewer Service, it is recommended that you have an in-depth look at the **FileViewer.jsx** file available from the **/src** folder.

For this tutorial, you will focus on how to interact with the viewer API (**bravaApi**).

1. Open the **FileViewer.jsx** file and explore the **toolbarWithMarkupStuff**, **tabContainerWithMarkups** and **markupTools** constants. They represent the detailed configuration of the different viewer components/features.

```
const toolbarWithMarkupStuff = {
  left: [
    { component: 'ToggleSidebarButton', side: 'tabContainerWithMarkups' },
    { component: 'DownloadButton' },
    { component: 'PanButton' },
    { component: 'ZoomToRectangleButton' },
    { component: 'SaveButton' },
    { component: 'ZoomInButton' },
    { component: 'ZoomOutButton' },
    { component: 'ZoomExtentsButton' },
    { component: 'ZoomWidthButton' },
    { component: 'RotateButton' },
  ],
  center: [{ component: 'TitleText' }],
  right: [
    { component: 'PageSelector', style: { marginLeft: '0.5em' } },
    { component: 'CloseButton', size: 18 },
  ],
};

const tabContainerWithMarkups = {
  sidebarName: 'tabContainerWithMarkups',
  primary: 'primary',
  tabs: [
    { component: 'ThumbnailPane', title: 'tab.thumbnails' },
    { component: 'MarkupPane', title: 'tab.markups', layoutKey: 'markupTools' },
  ],
};

const markupTools = [
  {
    title: 'Tools',
    tools: [
      {
        label: 'select markup',
        tool: 'select',
        icon: 'Select',
      },
    ],
  },
  {
    title: 'toolPalette.annotations',
    tools: [
      {
        label: 'text', tool: 'text', icon: 'Text', props: {},
      },
      {
```

2. Explore the **useEffect()** hook method to understand how the configuration gets applied to the **bravaApi** and ultimately returned to render the viewer.

```
useEffect(() => {
  if (didMountRef.current) {
    if (bravaApi) {
      bravaApi.setHttpHeaders({
        Authorization: `Bearer ${user.access_token}`,
      });
      bravaApi.setScreenBanner('Viewer Service by OpenText | Document Viewed at %Time');
      bravaApi.enableMarkup(true);
      bravaApi.editableMarkupPredicate = () => true;
      bravaApi.commentableMarkupPredicate = () => true;
      bravaApi.visibleToolPropertyPredicate = () => true;
      bravaApi.deletableMarkupPredicate = () => true;
      bravaApi.deletableStampPredicate = () => true;
      bravaApi.editableStampPredicate = () => true;
      bravaApi.addableStampPredicate = () => true;
      bravaApi.setMarkupHost(window.ViewerAuthority);
      bravaApi.setUserName(user.profile.preferred_username);
      bravaApi.setScreenWatermark('ORIGINAL');
      bravaApi.setLayout({
        topToolbar: 'toolbarWithMarkupStuff',
        toolbarWithMarkupStuff,
        mainContainer: [
          { component: 'TabContainer', layoutKey: 'tabContainerWithMarkups' },
          { component: 'PageContainer' },
        ],
        tabContainerWithMarkups,
        markupTools,
      });
      bravaApi.addPublication(publicationData, true);
      bravaApi.render(VIEWER_ID);
    }
  } else {
    window.addEventListener('bravaReady', bravaReadyEventListener);
    loadBravaViewer();
    didMountRef.current = true;
  }
}, [
  bravaApi,
  publicationData,
  user.access_token,
  user.profile.preferred_username,
  loadBravaViewer,
]);
```

3. The absolute minimum for the viewer to render correctly is the following code snippet (showing a drastically reduced **useEffect()** method):

```
useEffect(() => {
  if (didMountRef.current) {
    if (bravaApi) {
      bravaApi.setHttpHeaders({
        Authorization: `Bearer ${user.access_token}`,
      });
      bravaApi.addPublication(publicationData, true);
      bravaApi.render(VIEWER_ID);
    }
  } else {
    window.addEventListener('bravaReady', bravaReadyEventListener);
    loadBravaViewer();
    didMountRef.current = true;
  }
}, [
  bravaApi,
  publicationData,
  user.access_token,
  loadBravaViewer,
]);
```

It is recommended you try out different configurations by modifying the **useEffect()** method's content.

4. To help with understanding the Viewer Service configuration, two additional files in the **/src/components** folder are included: **FileViewer.jsx-minimal** and **FileViewer.jsx-full**. They respectively provide an example of the minimal set of configuration settings (as shown in the above code snippet) and an example of the viewer with all configuration settings enabled and configured. Feel free to try adding or removing configuration settings using these two example files as reference.

   The **FileViewer.jsx** file that is used for the Contract Approval application shows you an example of how the configuration can be applied to display/overlay a screen banner (footer) and a watermark on the viewed document pages, and to provide markup tools from the left panel under a markups tab.

Next step:

Use the Magellan Risk Guard API.

14.8Use the Magellan Risk Guard API

The last API to look into is the Magellan Risk Guard API. Magellan Risk Guard offers the ability to submit the content of a file/document to analyze it and identify potentially risky or sensitive content.

The Contract Approval application calls the Magellan Risk Guard API to determine the risk level of a newly created contract (that is, uploaded file).

1. First, (re)open the **AddContract.jsx** file as it holds the code that creates a new contract. More specifically, the Magellan Risk Guard Service API gets called at the beginning of the **submitContract** method.

   The **processDoc()** method of the **riskGuardService** (instance of **RiskGuard** class) is called to return the **riskClassification** and **extractedTerms** for the contract that is being submitted.

```
130   const submitContract = async () => {
131     setState((prevState) => ({ ...prevState, showBackdrop: true }));
132     const risk = await riskGuardService.processDoc(
133       state.selectedFile,
134       state.selectedFile.name,
135     );
136     setState((prevState) => ({
137       ...prevState,
138       contractRisk: risk.data.riskClassification,
139       extractedTerms: risk.data.extractedTerms,
140     }));
```

2. The code that interacts with the Magellan Risk Guard API sits within the **RiskGuard** utility class, so the second bit of code to look at is in the **RiskGuard.js** file, located under **/src/services/riskguard**.

   The **processDoc()** method calls the Magellan Risk Guard API through a (axios) POST request, passing the given (contract) file as the **FormData data** payload of the call (which requires to set the **Content-Type** header to **multipart/form-data**).

```
107   async processDoc(fileData, fileName) {
108     const form = new FormData();
109     form.append('File', fileData, fileName);
110     const postRequest = {
111       method: 'post',
112       url: this.url,
113       headers: {
114         Authorization: `Bearer ${this.user.access_token}`,
115         'Content-Type': 'multipart/form-data',
116       },
117       data: form,
118     };
119
120     return new Promise((resolve, reject) => {
121       axios(postRequest).then((postResponse) => {
122         resolve({ data: RiskGuard.calculateRisk(postResponse) });
123       }).catch((response) => {
124         reject(response.response);
125       });
126     });
127   }
128 }
```

3. Feel free to also have a look at how the risk classification is computed in the **calculateRisk()** method.

CONGRATULATIONS!

You have now finished building the Contract Approval application. In the next exercise module, you will test it and run through the different contract approval scenarios.

Next exercise module:

Test the Contract Approval application.

15 [60’] Test the Contract Approval application

Learn how to:

- Run a React application that consumes the OpenText Cloud Platform Services APIs
- Sign in to the application using an external login screen and redirect into the application
- Explore the Contract Approval application UI and features
- Test uploading a (contract) file and filling its metadata (consuming the Content Storage Service and Content Metadata Service)
- Test the applying of ACLs (consuming Content Metadata Service)
- Test the updating of document metadata and traits (consuming the Content Metadata Service)
- Test the detection of sensitive content (consuming the Magellan Risk Guard Service)
- Test the Viewer (consuming the Viewer Service)
- Test the different contract approval flows, including rejection and expiry of manual approval tasks (consuming the Workflow Service and Decision Service)

During this exercise module you will be testing the Contract Approval application. You will run through different scenarios to demonstrate the different behaviors that depend on the automated and manual choices that can be made within the application.

Note

If you have not gone through all the exercise modules to build the Contract Approval application yourself, you must first set up your testing environment according to the following steps:

- Be certain that you have checked and fulfilled the Prerequisites.
- Set up your IDE as described in Set up the OpenText Cloud Developer IDE.
- Connect to your developer organization as described in Set up a connection to the developer organization.
- Download the finished version of the Contract Approval application as described in Download the Contract Approval application.
- Once downloaded and extracted, make sure to open the **demo-contract-approval-app-master** folder in VS Code, as this is the root of your project.
- Deploy the application into your developer organization as described in Deploy an application to the OpenText Cloud Platform Services.
- Update the **.env** file as described in Set the environment variables.

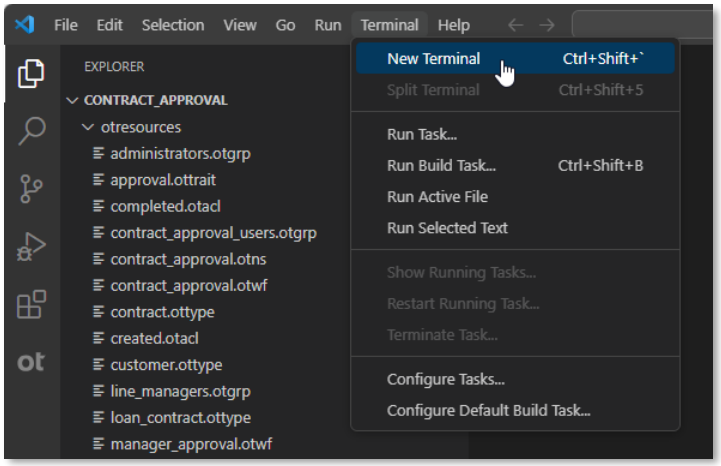
To help with understanding the application that you are going to test, this is a very short explanation of its main project folders:

- **src**: contains the JavaScript and React code that communicates with the OpenText Cloud Platform Services APIs and provides the User Interface (UI) of the application
- **otresources**: contains the different models (built with the OpenText Cloud Developer Tools for VS Code) that you deployed to the OpenText Cloud Platform Services

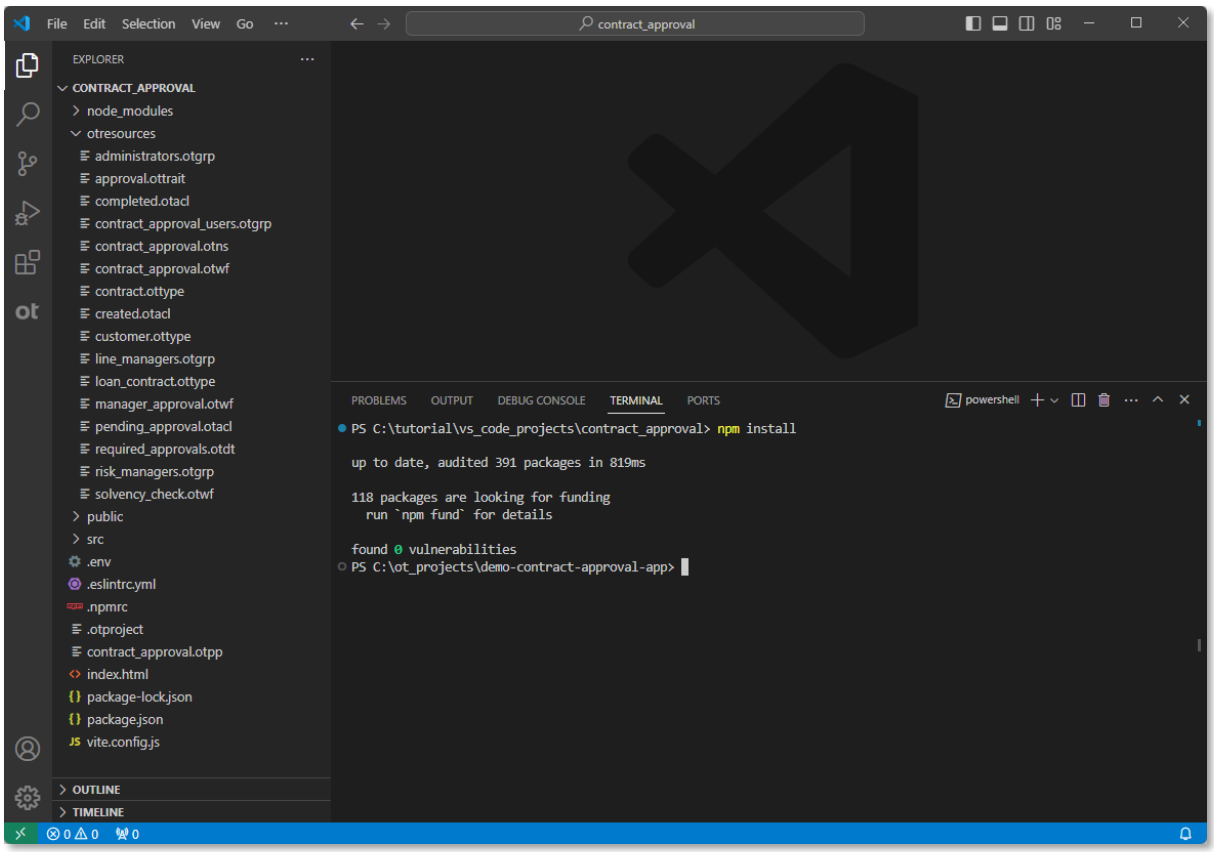
You are now ready to proceed with this exercise and test the Contract Approval application.

15.1 Start the Contract Approval application and sign in

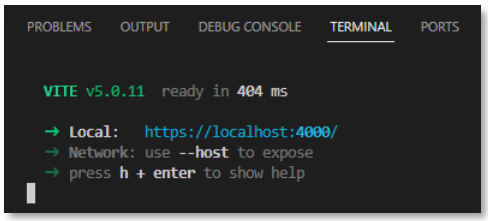
1. Open the Contract Approval application project in VS Code and open a new **Terminal** window.

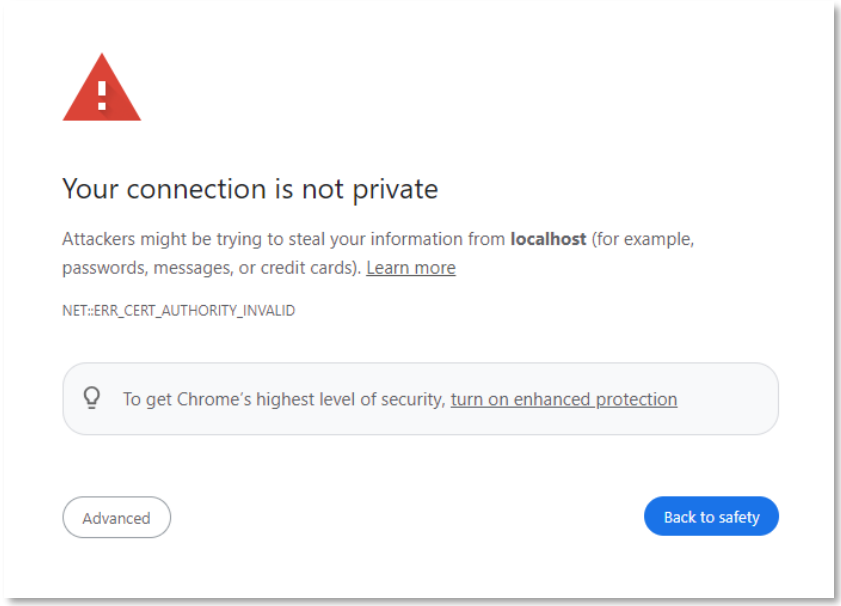


2. In the **Terminal** window run the `npm install` command. This will install all dependencies needed by your application to run.

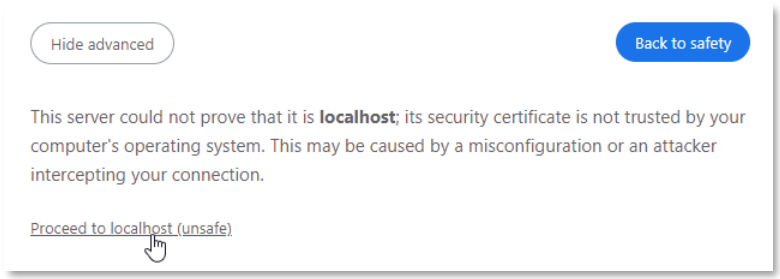


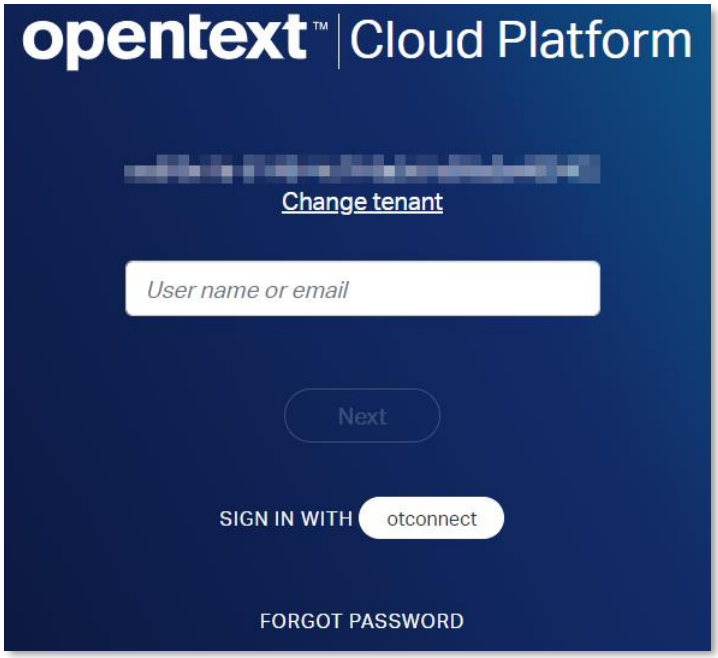
3. After the npm install process has completed, launch the application using `npm start`. This will result in a new browser window opening on **https://localhost:4000**.



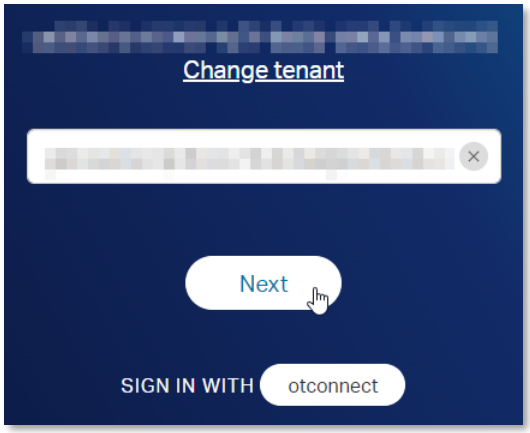


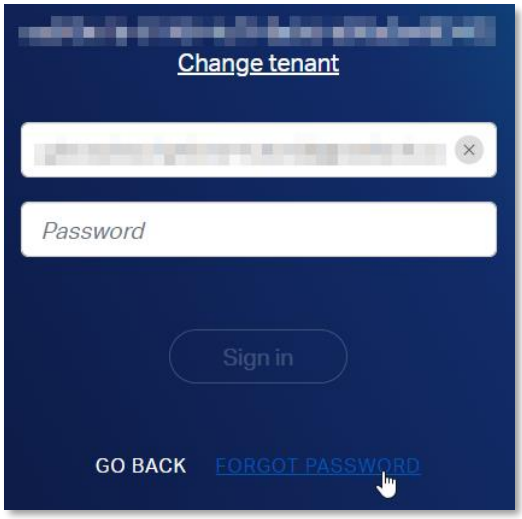
Click the **Advanced** button and select **Proceed to localhost (unsafe)**. Note that the screen shots are of Google Chrome and that the equivalent action on your own web browser can be different.

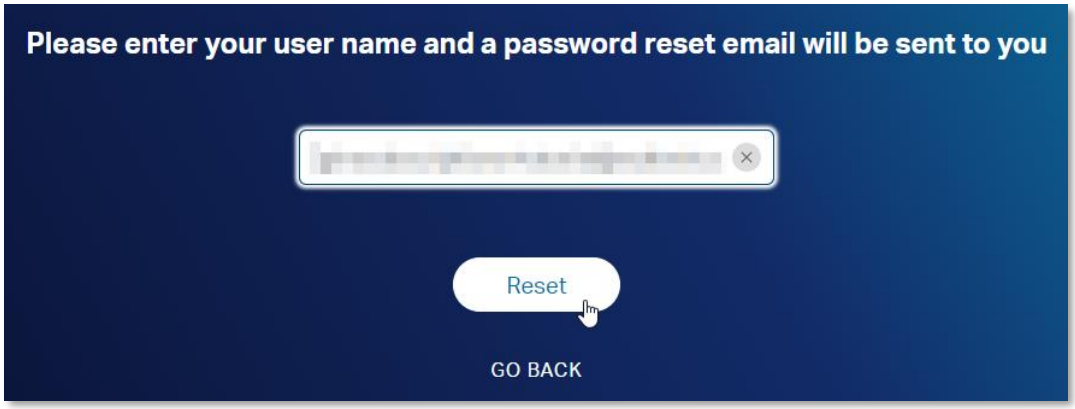


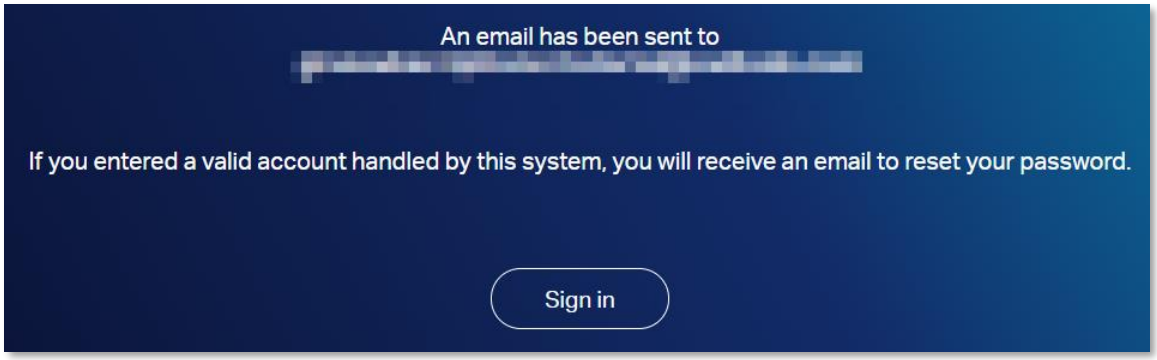


4. Fill your developer organization account email (that is, the **regular user** email) and select **FORGOT PASSWORD** to set the password (as you only set passwords for the **line manager** and **risk manager** users).

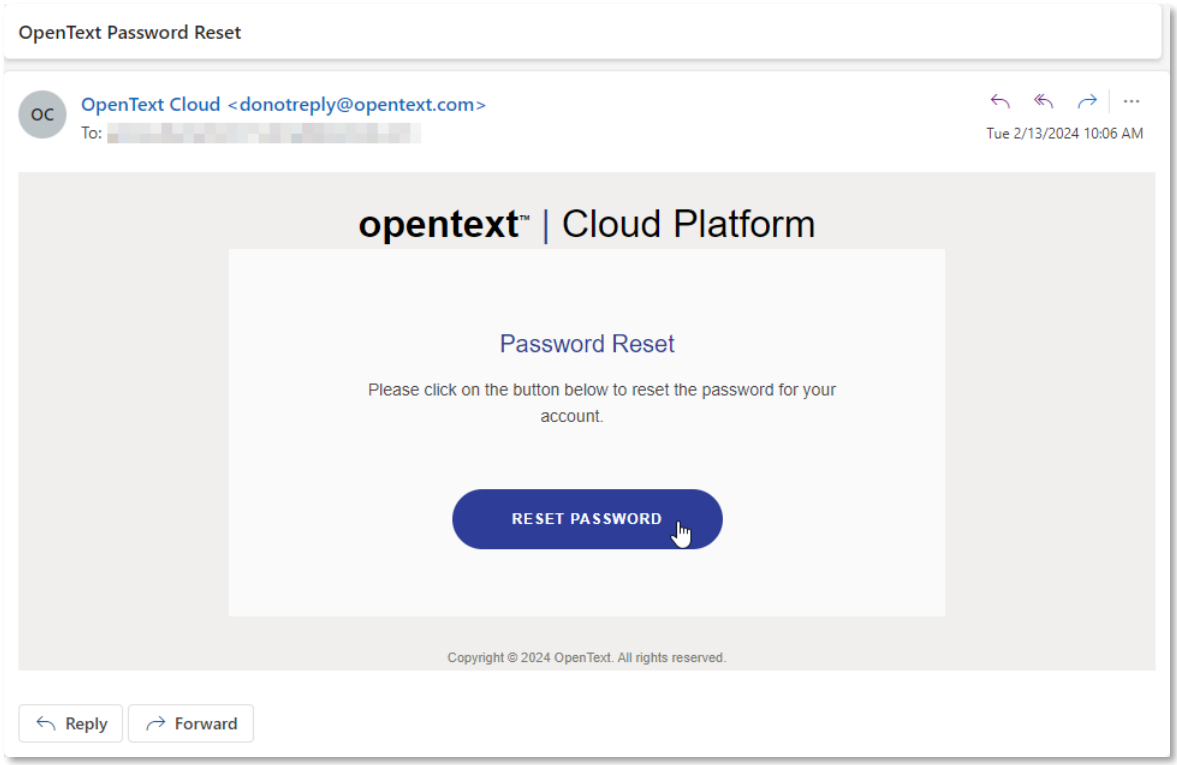


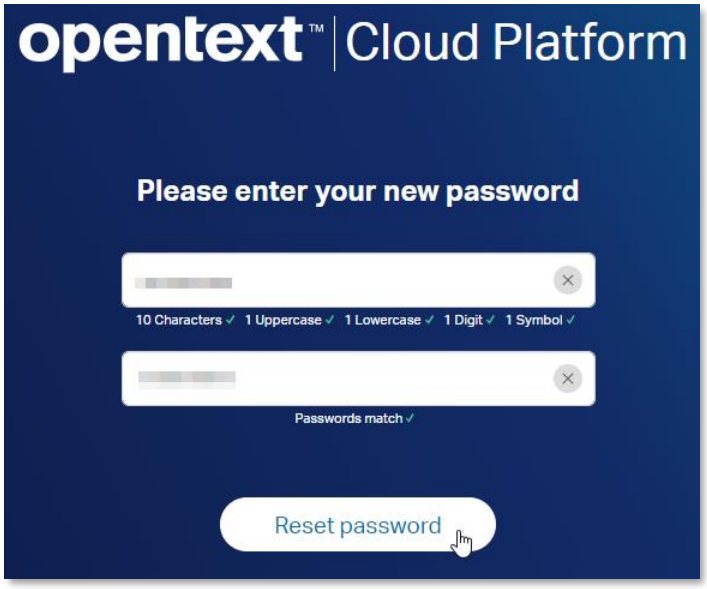




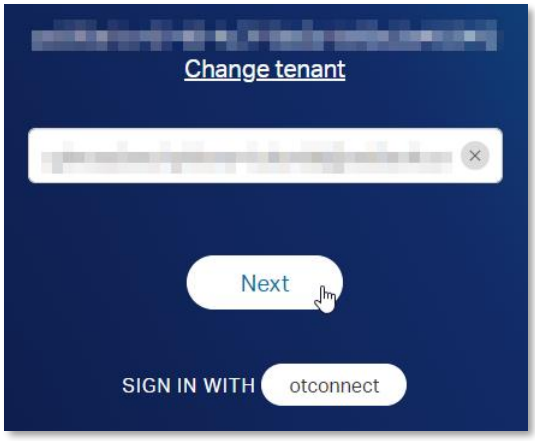


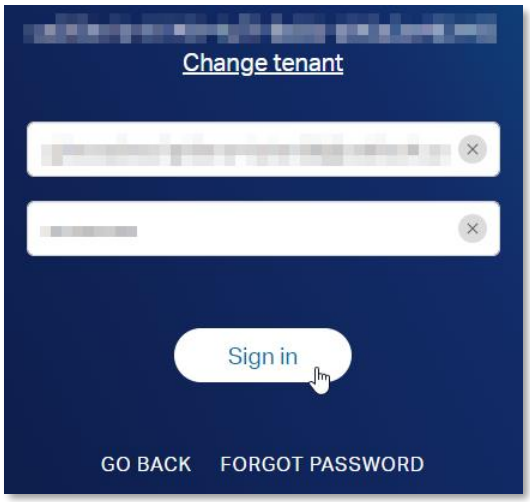
5. Go to your email inbox and from the new **OpenText Password Reset** email, click the **RESET PASSWORD** button, and reset the password.

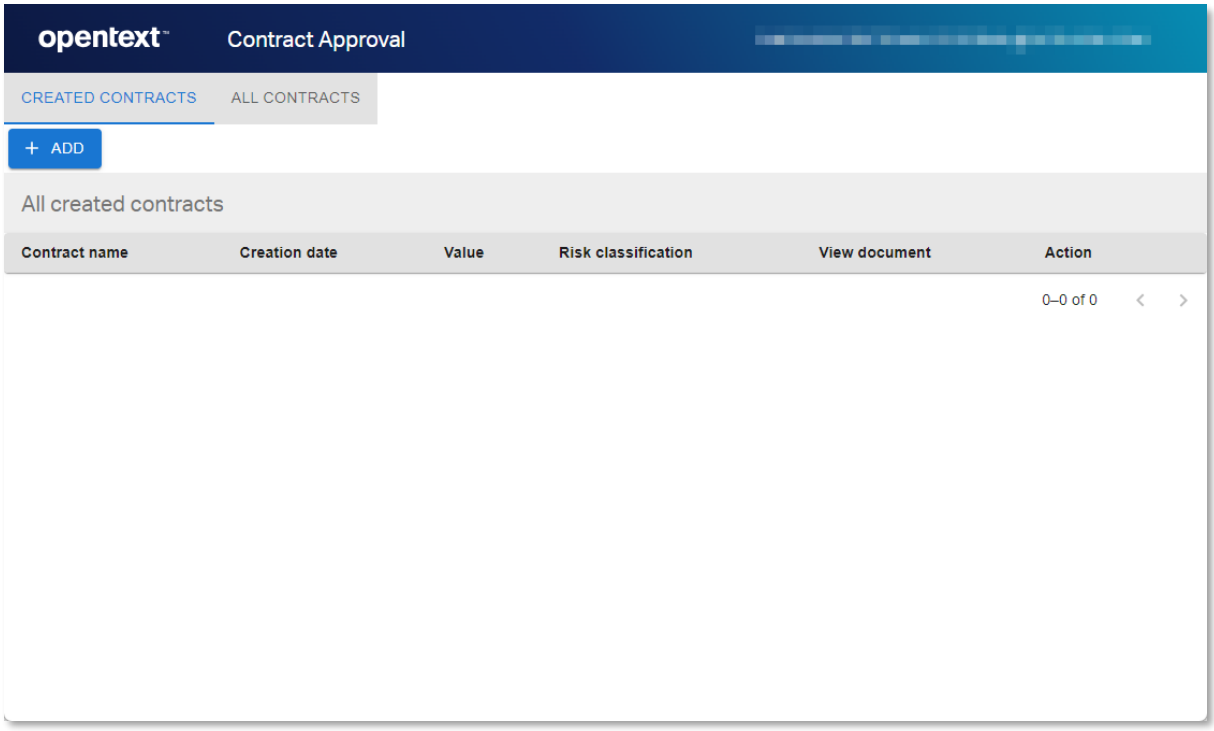




6. Close and reopen the browser, navigate to **localhost:4000** and sign in.







You are now logged in to the Contract Approval application.

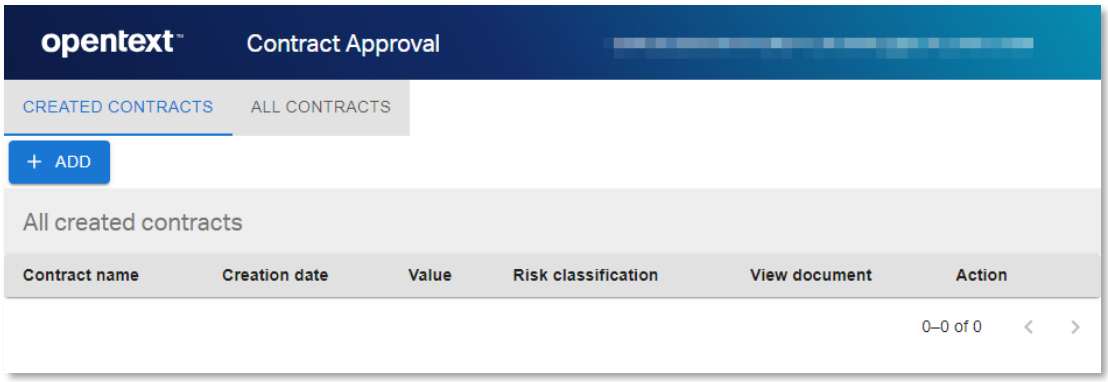
Next step:

Approve a standard contract that only requires automatic approval (no additional required approvals).

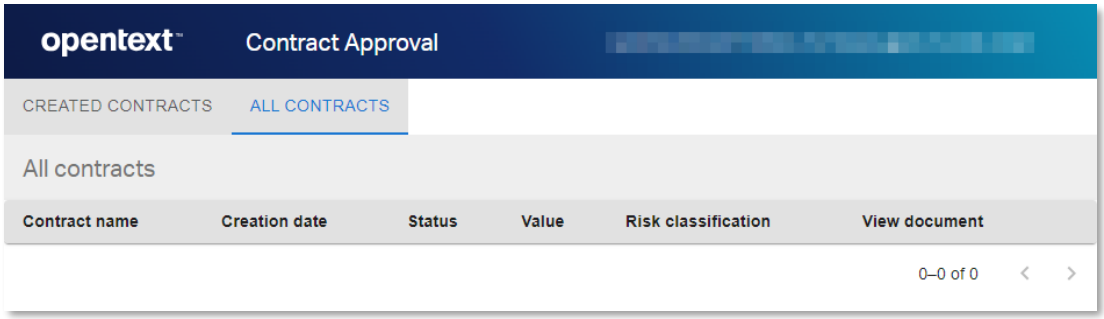
15.2 Approve a standard contract that only requires automatic approval (no additional required approvals)

1. Have a look at the different tabs. As previously described when discussing the application code, as a regular user (that is, not being a member of the **line_managers** or **risk_managers** groups), the currently connected user can only see two tabs from the four tabs in total:

   - CREATED CONTRACTS: this tab shows all newly created contracts that have not yet been submitted for approval (that is, contracts with the status attribute equal to 'CREATED')



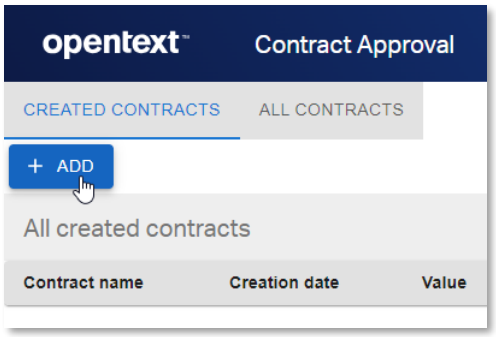
   - ALL CONTRACTS: this tab shows all contracts, independently of their status (that is, newly created, under approval, approved, rejected, expired)



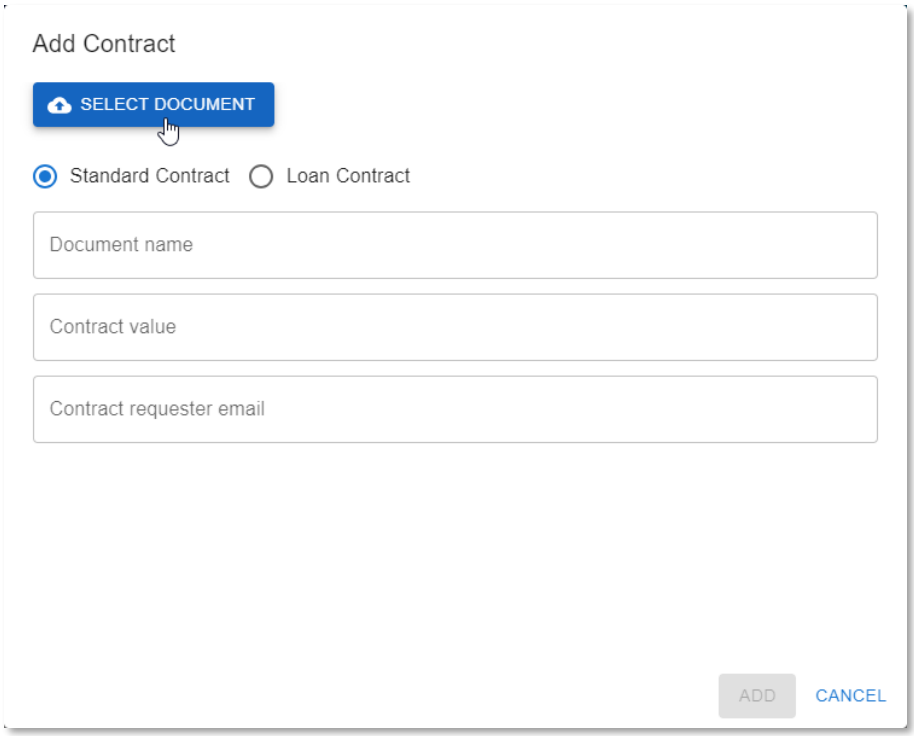
2. After checking the different tabs that the regular user can see, create a first standard contract.

   To trigger the simplest approval process, you will create a contract with the following characteristics:
   - **Type: standard contract** (doesn't require solvency check)
   - **Value: below 1000** (doesn't require Line Manager approval)
   - **Risk classification: below 4, that is, NONE, LOW or MEDIUM** (doesn't require Risk Manager approval)

3. Select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.

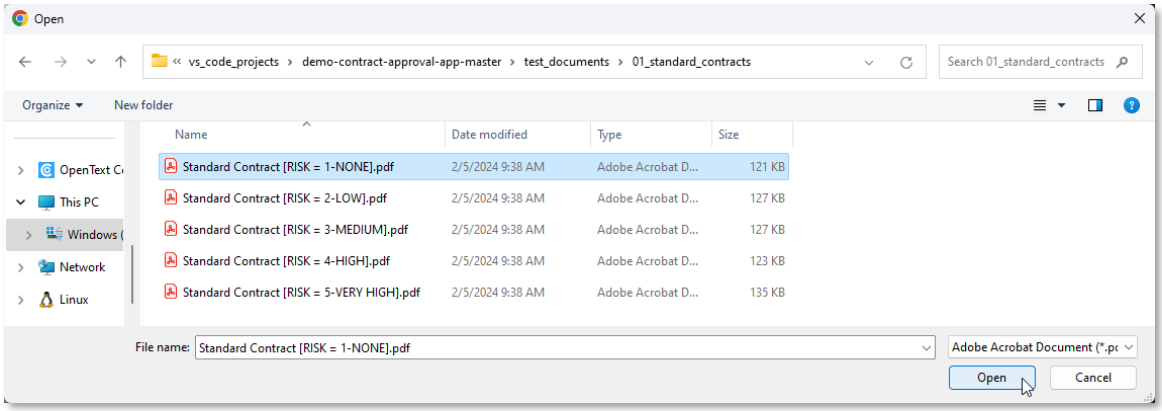


4. From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.



5. To help with selecting a file that matches the intended contract properties (certainly the risk classification, as this gets determined by what the Magellan Risk Guard Service discovers in the document), a **test_documents** folder is provided under the finished version of the Contract Approval application project.

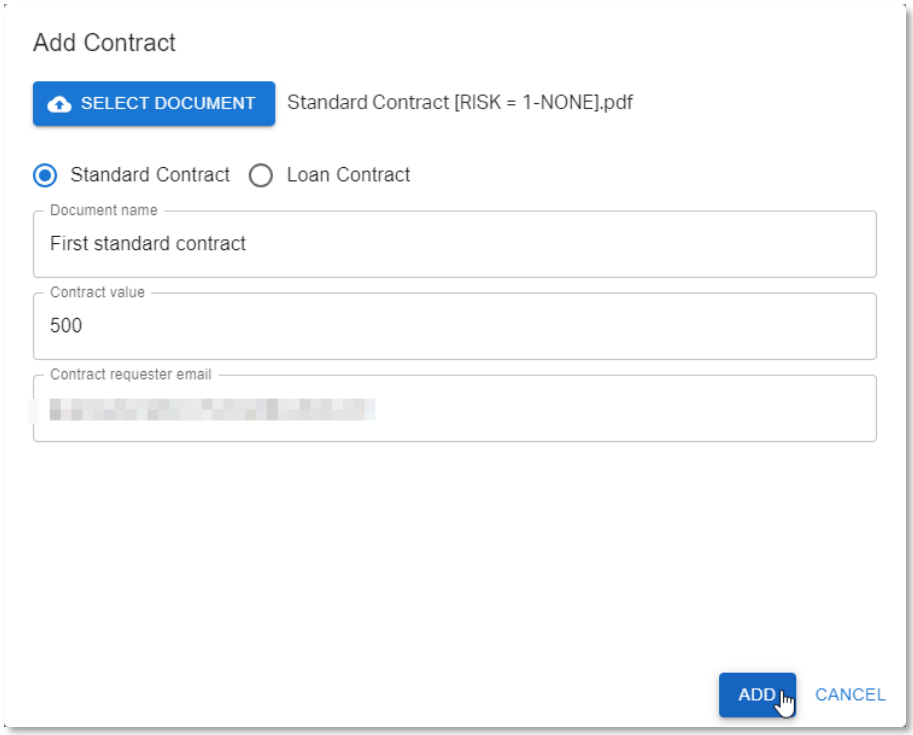
   From this **test_documents** folder, open the **01_standard_contracts** subfolder and select the **Standard Contract [RISK = 1-NONE].pdf** file.



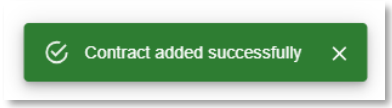
6. Make sure the **Standard Contract** option is selected and fill the contract properties as follows:

Property	Value
Document name	`First standard contract`
Contract value	`500`
Contract requester email	**<your email>**

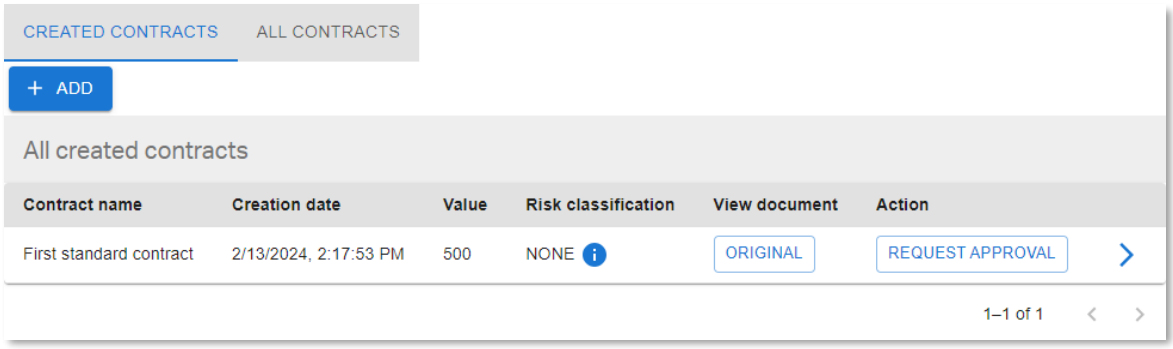
7. Click **Add** to create the contract.



At the bottom of the screen, the creation of the new contract is confirmed by a “Contract added successfully” message.

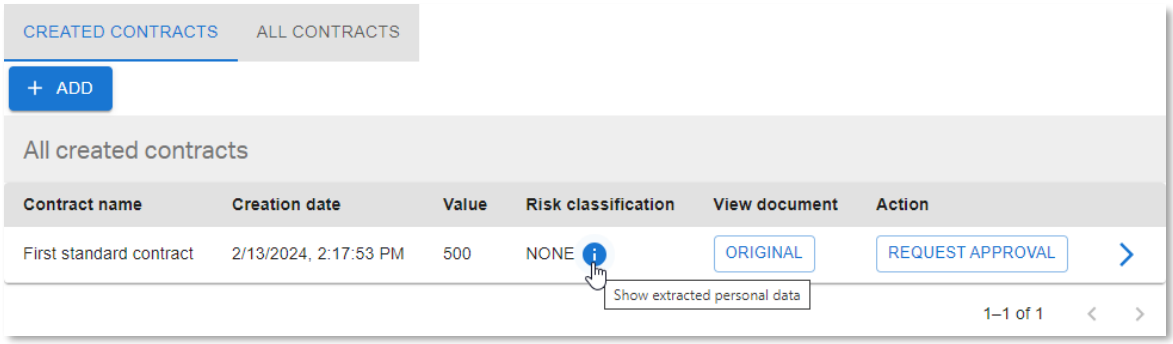


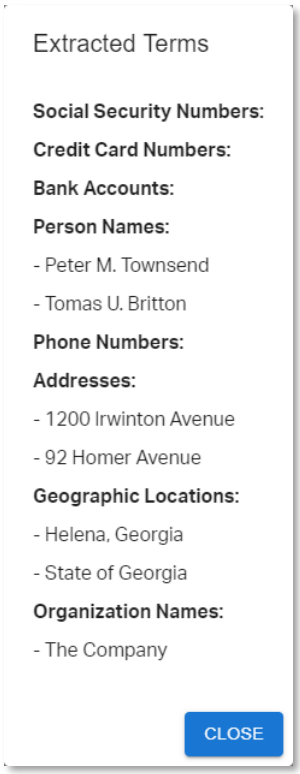
The first standard contract is now created.



8. Explore the contract list capabilities from the **CREATED CONTRACTS** view.

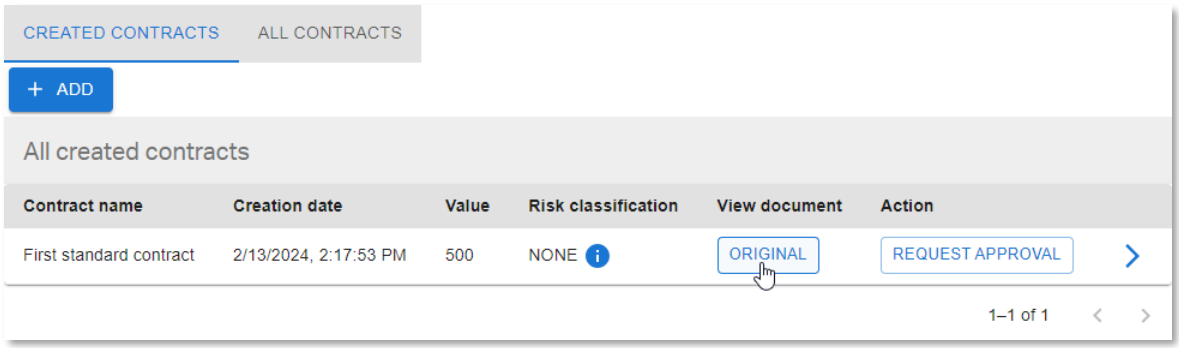
   First, note that the **Risk classification** property indeed shows **NONE** as risk level. Click on the (information icon) next to the **NONE** risk classification value to see which terms the call to the Magellan Risk Guard API has identified and extracted.



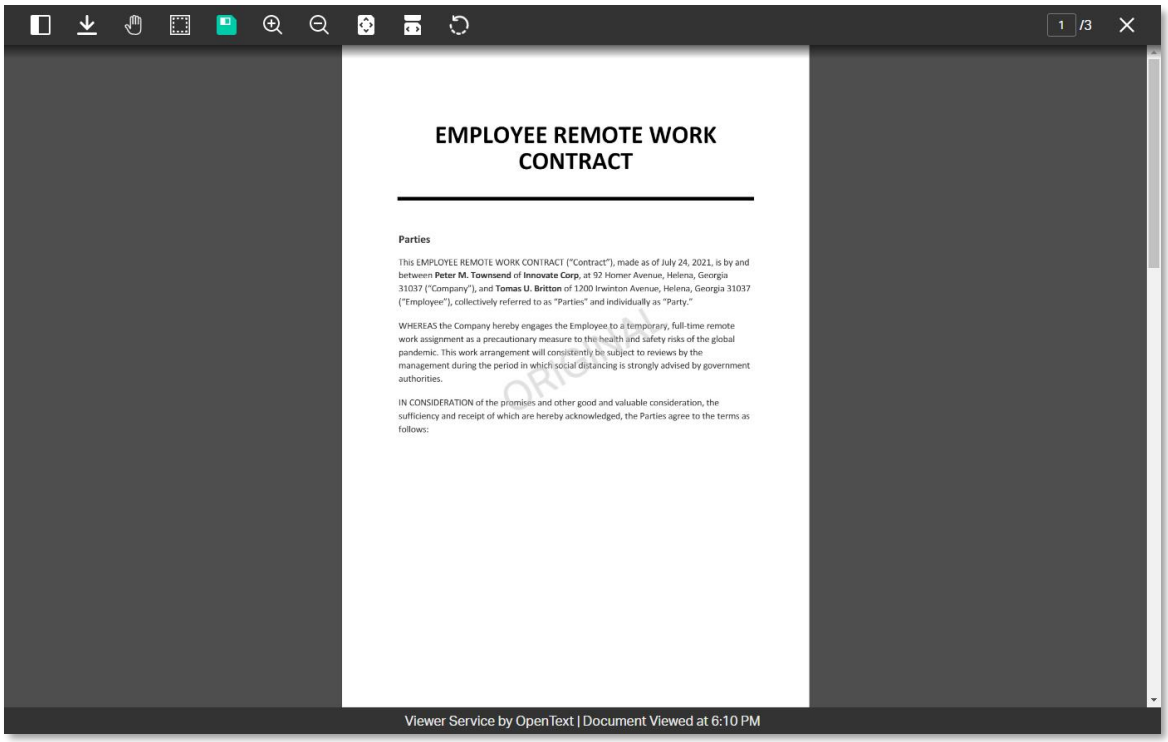


A few names, addresses, geographic locations, and organization names were found, but nothing that warrants increasing the risk level (hence risk classification = NONE).

9. Click **CLOSE** to close the **Extracted Terms** information screen.
10. Click on the **ORIGINAL** button (in the **View document** column) to view the uploaded document content.



The file content of the contract displays to be viewed (in the viewer from the Viewer Service).

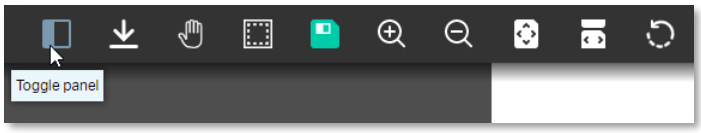


The toolbar at the top of the viewer allows multiple actions (as defined in the code of the FileViewer.jsx file):

- Toggle panel
- Download document
- Toggle pan mode (allowing to use the cursor to navigate the document)
- Zoom window (allowing to draw a window on the document to zoom into)
- Save markups
- Zoom in
- Zoom out
- Fit document to screen
- Fit width to screen
- Rotate left
- 1 /3 Page selector

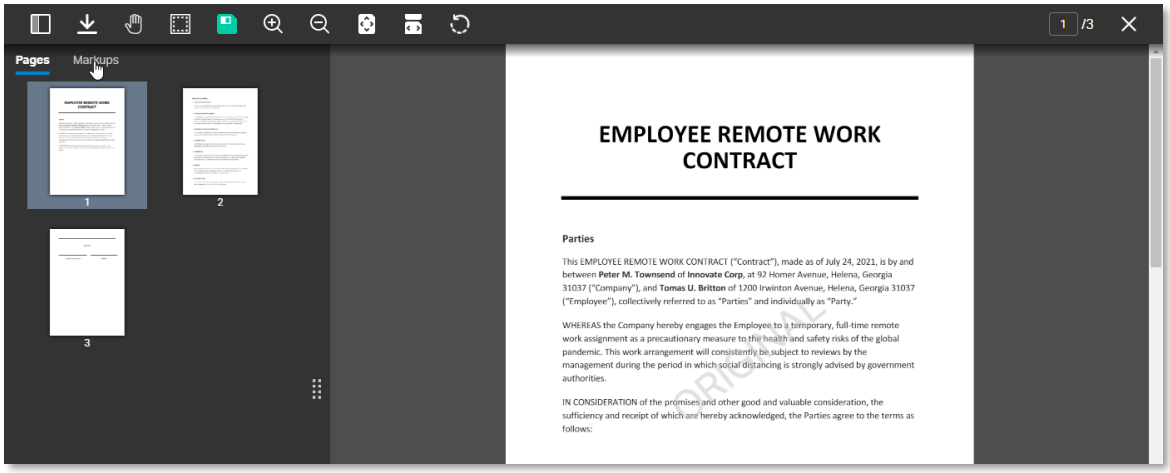
11. Although most of the actions in the toolbar are self-explanatory, have a look at the panel toggle.

    Click the **Toggle panel** button to open the (left) side panel.

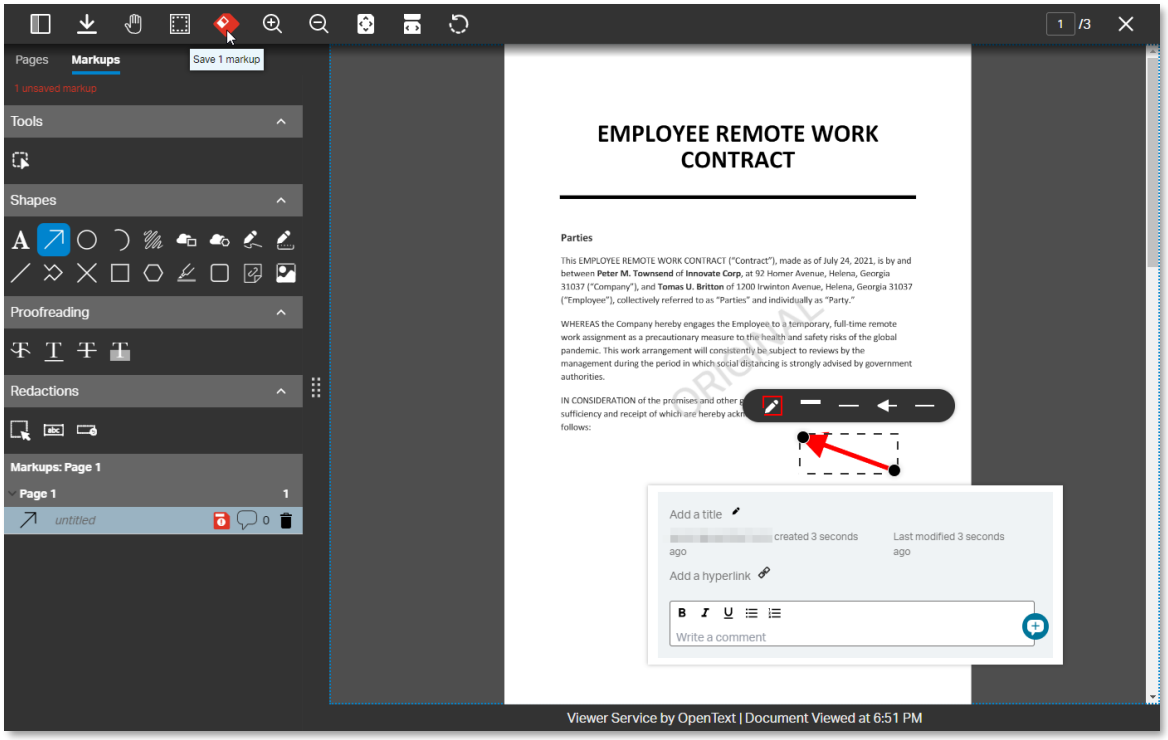


    The side panel shows two tabs (again as per the code in FileViewer.jsx). The **Pages** tab shows the page thumbnails, and the **Markups** tab provides the markup tools.

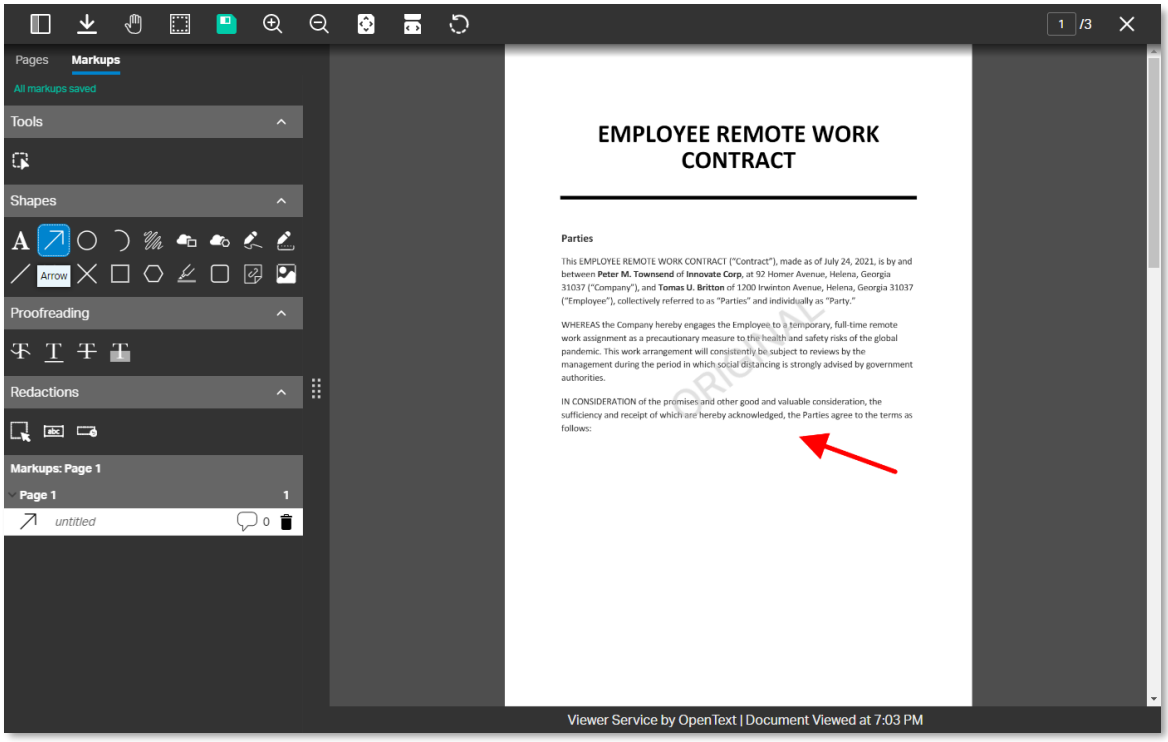
12. Click the **Markups** tab to switch to the markup tools view.



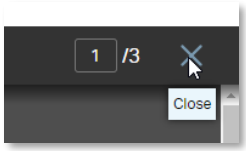
13. Select a markup shape, draw it on the document (for example, an **arrow**), and click the **Save markups** button to save it.



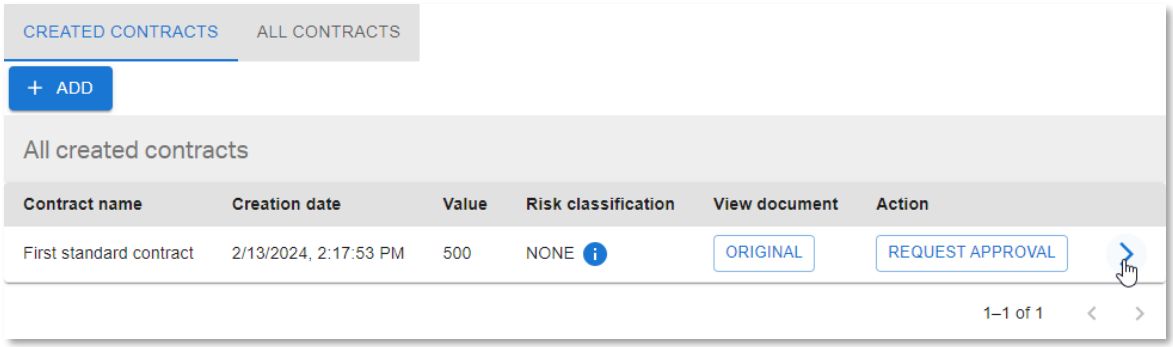
14. You can explore creating/modifying/deleting/commenting other markups, but for the purpose of this tutorial just press the **esc** key to unselect the markup and proceed with further examining the viewer.



15. The two last things defined in the FileViewer.jsx code to draw your attention to are the **ORIGINAL** watermark (across the document pages) and the **Viewer Service by OpenText | Document Viewed at <computed viewing time>** footer.
16. To go back to the main application screen, close the viewer with the ☒ on the top right.



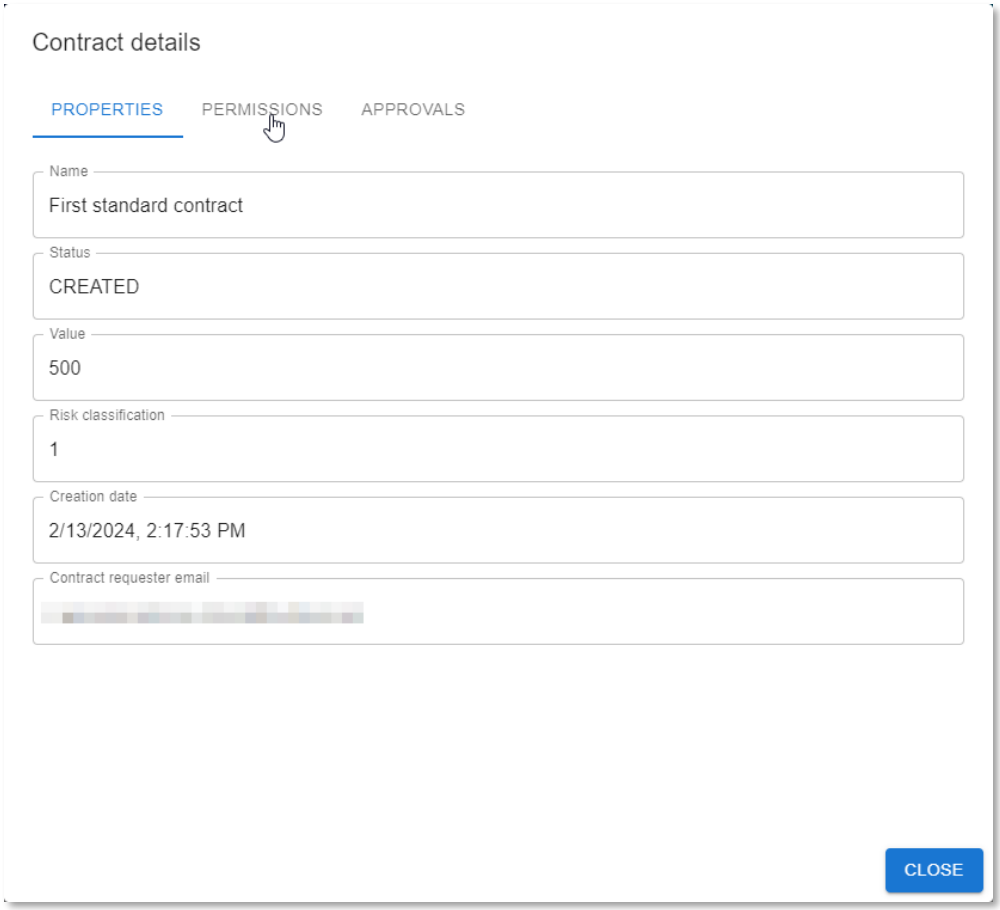
17. Click on › (caret arrow icon) to view the contract details.



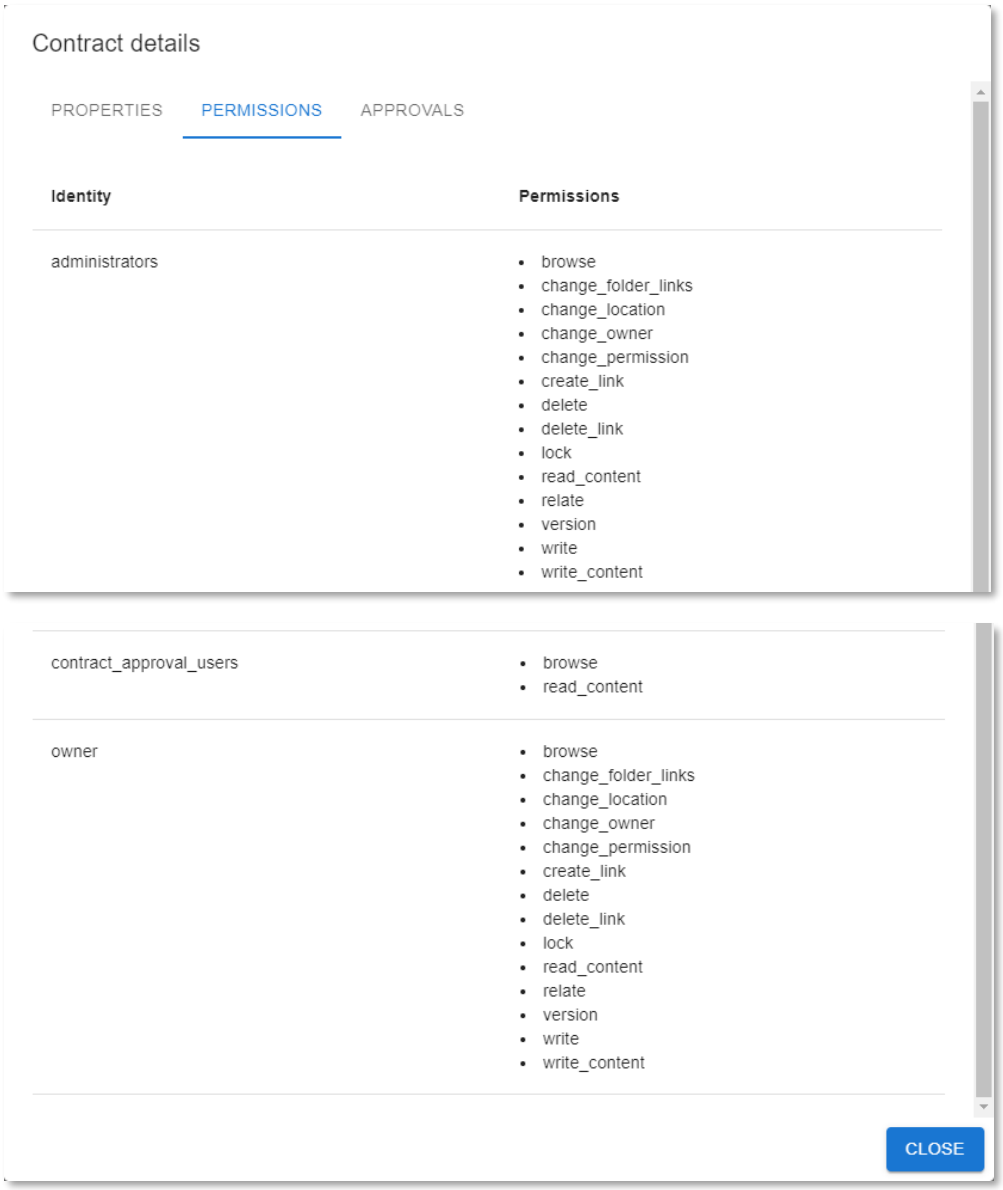
18. In the **Contract details** screen, there are three tabs. The first one shows the contract properties.

    Note that the **PROPERTIES** tab indeed displays a status of ‘CREATED’ and that the risk classification is 1 (which is the corresponding integer value for the NONE risk level).

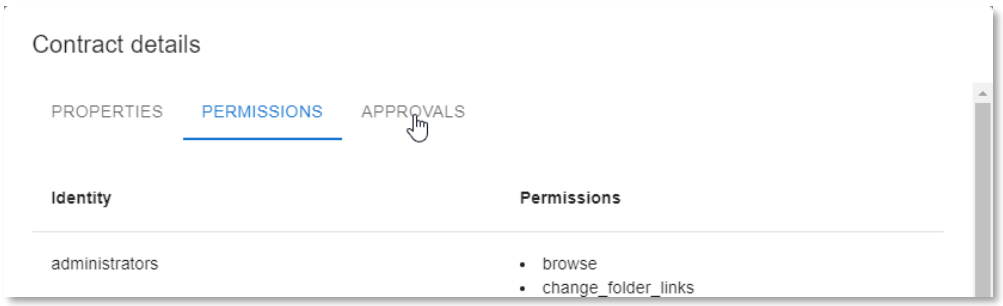
19. Click the **PERMISSIONS** tab to have a look at the different permissions for this contract.



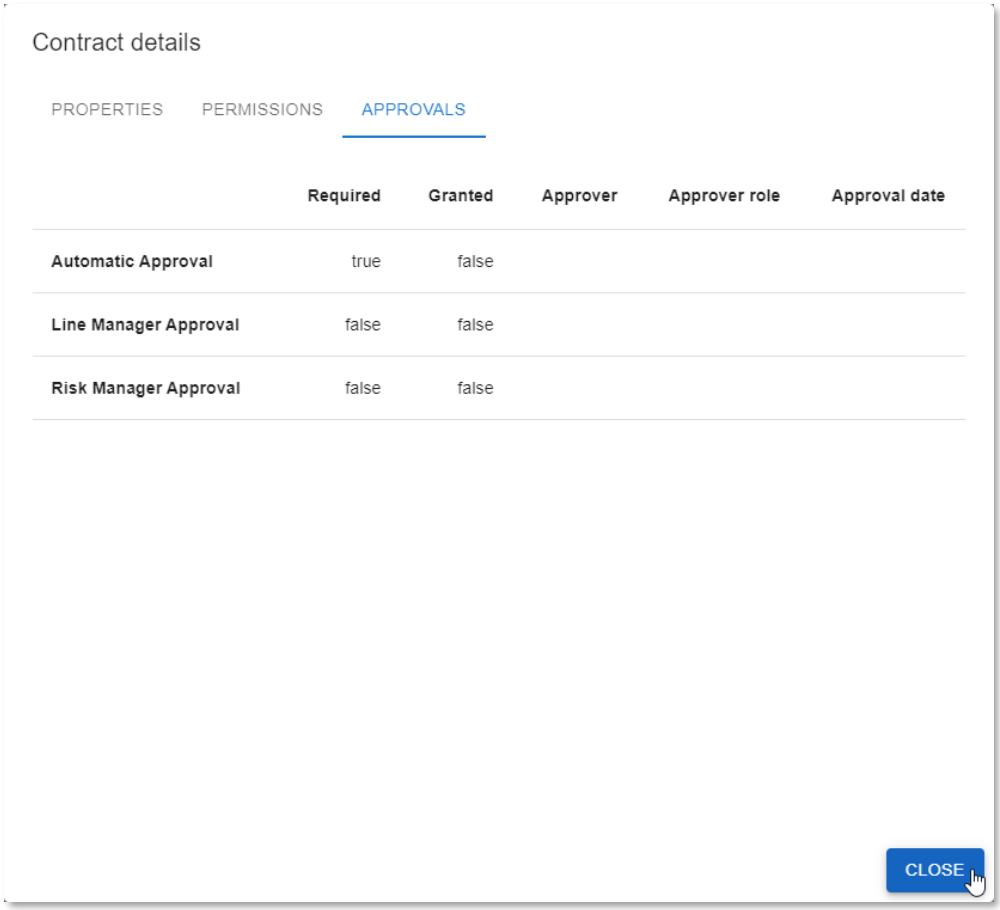
20. The **PERMISSIONS** tab displays the permissions for the contract (full control access for the **administrators**, read access for the **contract_approval_users**, and full control for the **owner**).



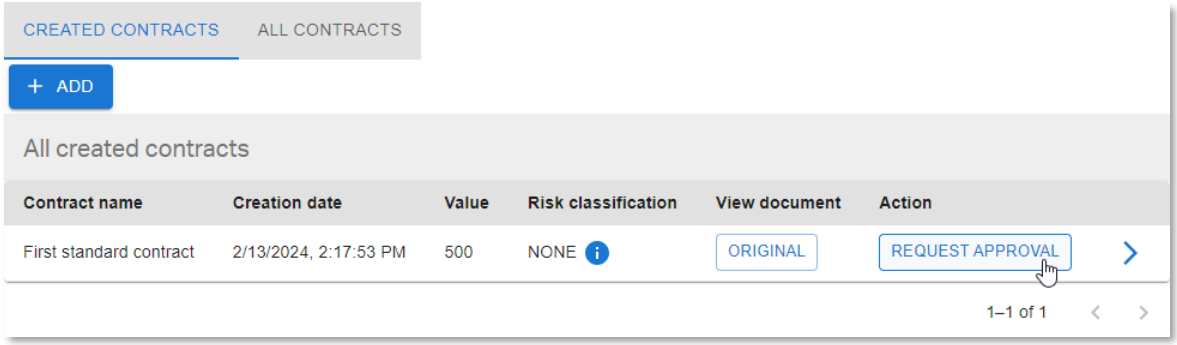
21. Click the **APPROVALS** tab to view the different approvals for this contract.



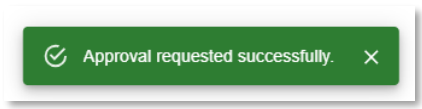
22. The **APPROVALS** tab displays the different approval steps (traits) for the standard contract type (**Automatic Approval**, **Line Manager Approval** and **Risk Manager Approval**). As you can see, the only approval step marked as required is the **Automatic Approval**.
23. Click **CLOSE** to close the **Contract details** screen.



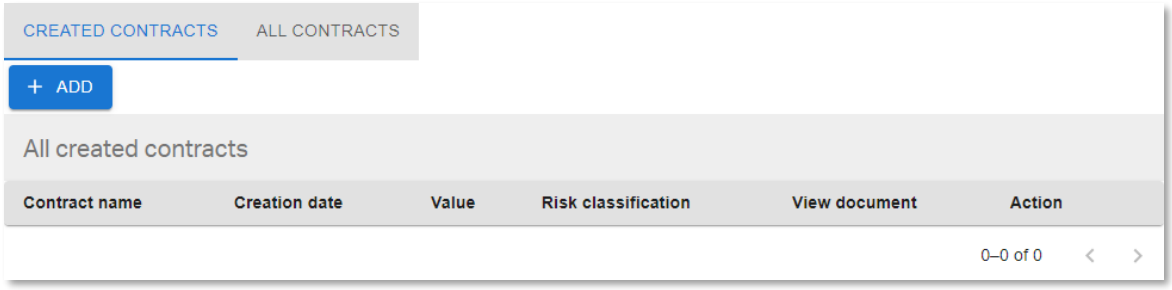
24. Back in the **CREATED CONTRACTS** view, you can now launch the approval workflow. Click the **REQUEST APPROVAL** button in the **Action** column to do that.



At the bottom of the screen the “Approval requested successfully” message confirms that the approval process has started.



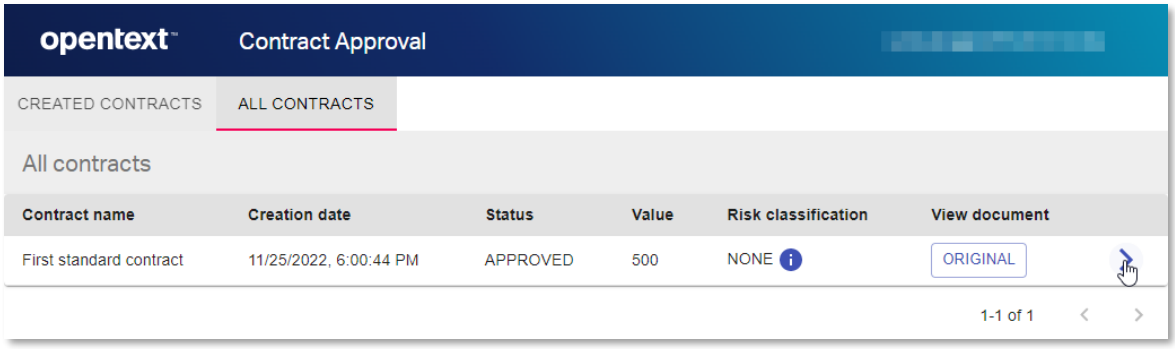
25. The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in ‘CREATED’ status.



26. Since the value of the contract is below 1000, no line manager approval is required. There is also no need for the risk manager to approve, as the risk classification is NONE (1) which is below HIGH (4).

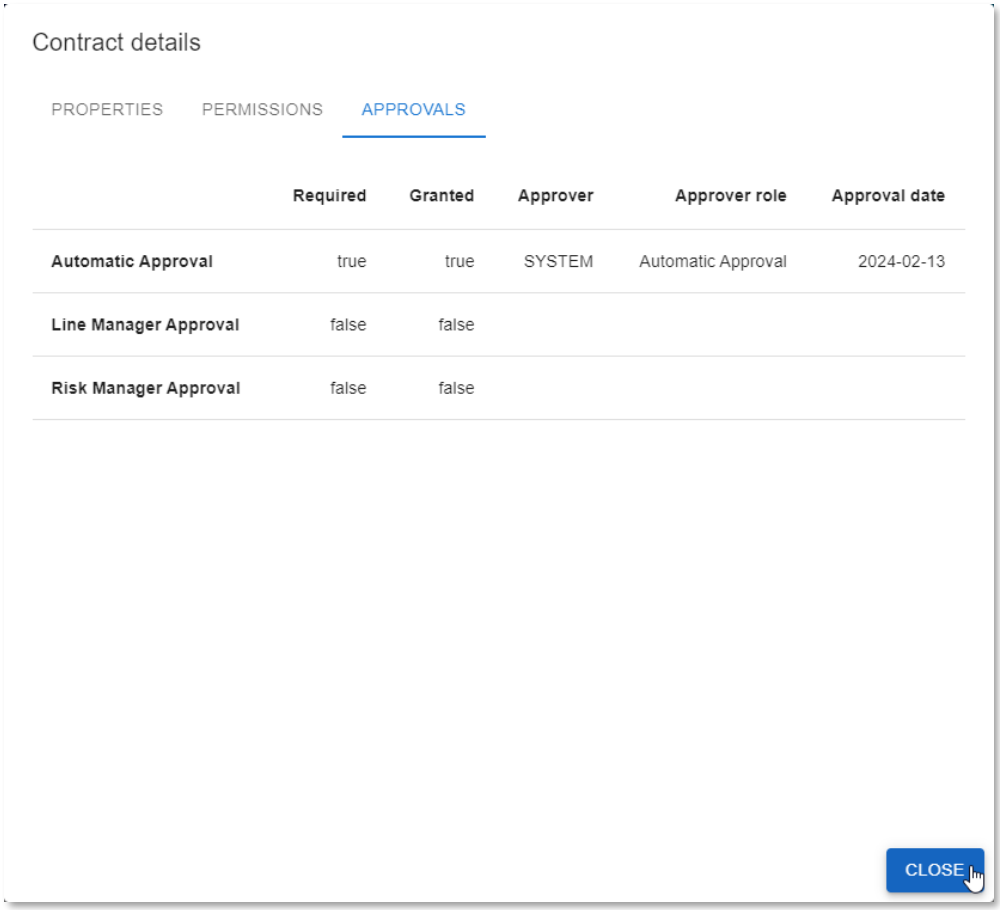
    Open the **ALL CONTRACTS** view to confirm the contract has been automatically approved (**Status** column shows **APPROVED** status), without needing any manual approval by the line manager or risk manager.

27. Open the contract details as well, and more specifically the approval steps/traits by clicking › (caret arrow icon) and selecting the **APPROVALS** tab.



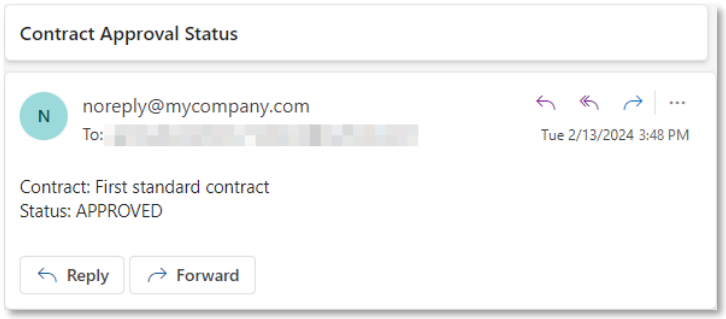
28. The **APPROVALS** tab still shows that only the **Automatic Approval** was required, but with the difference that it has now been granted by **Approver SYSTEM** with the **Approver role** of **Automatic Approval** at a specific **Approval date** and time.

    Click **CLOSE** to return to the **ALL CONTRACTS** view.



29. The first standard contract is now approved.

    Go to your email client and check the inbox and confirm recieving (due to the email task in the workflow) a **Contract Approval Status** email from **noreply@mycompany.com**.

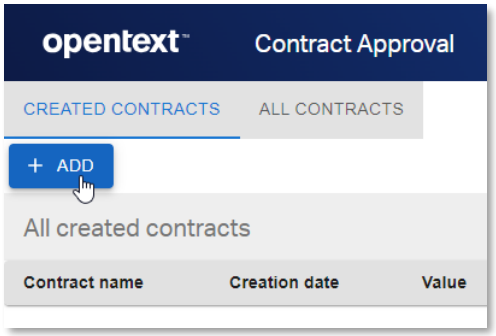


Next step:

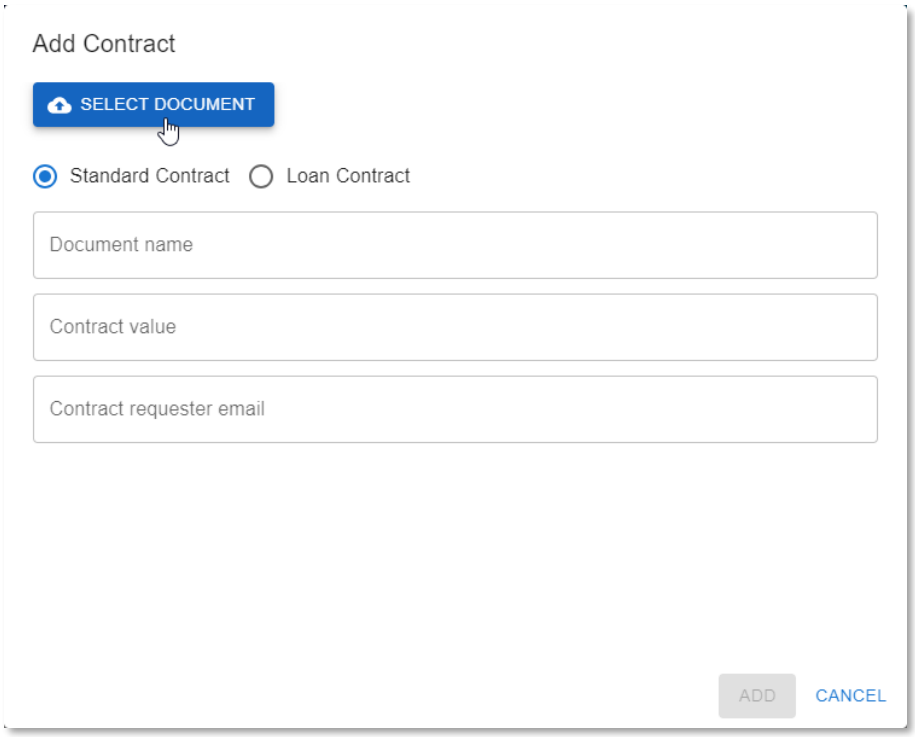
Approve a loan contract that requires all additional approvals.

15.3 Approve a loan contract that requires all additional approvals

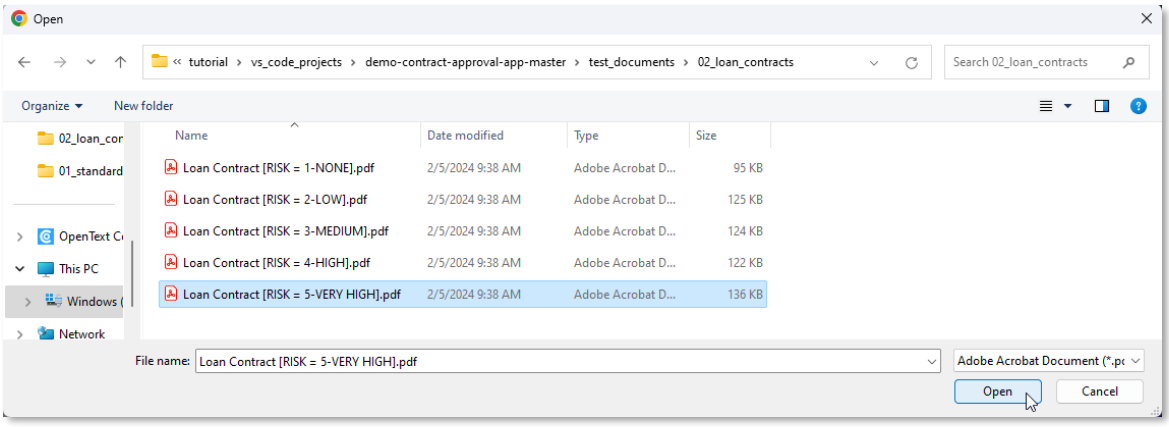
1. The second contract to create will follow the most extensive process flow (requiring all automated and manual approvals). To that end, you will create a contract with the following characteristics:
   - **Type: loan contract** (requires solvency check)
   - **Monthly loan cost is below or equal to 25% of monthly income** (requester is solvent, so automatic solvency check will not reject the contract approval request)
   - **Value: above 5000** (requires Line Manager approval)
   - **Risk classification: above 3, that is, HIGH or VERIFY HIGH** (requires Risk Manager approval)
2. Select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



3. From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.



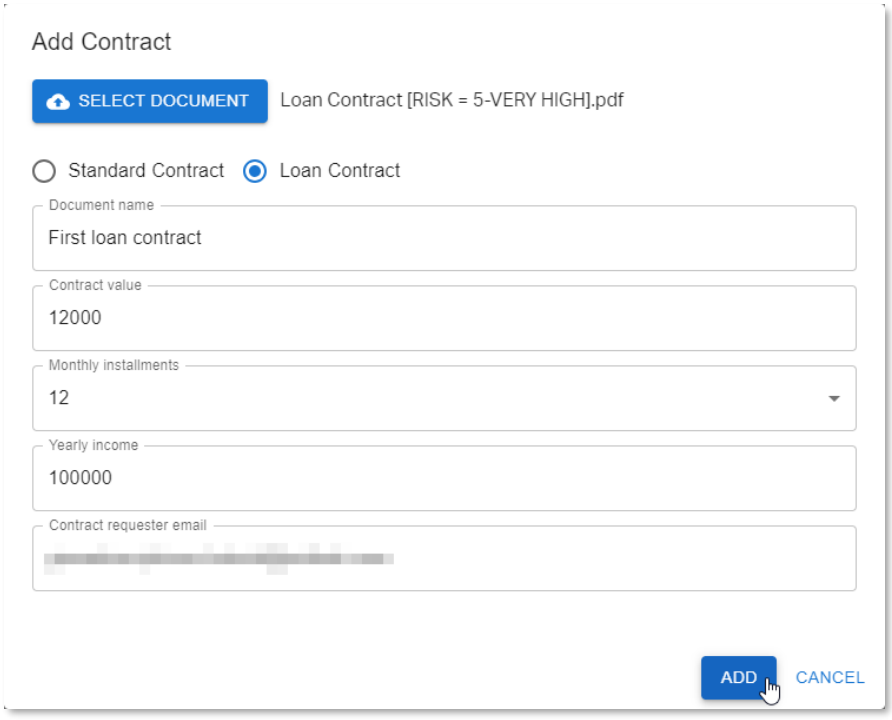
4. From the **test_documents** folder, open the **02_loan_contracts** subfolder and select the **Loan Contract [RISK = 5-VERY HIGH].pdf** file.

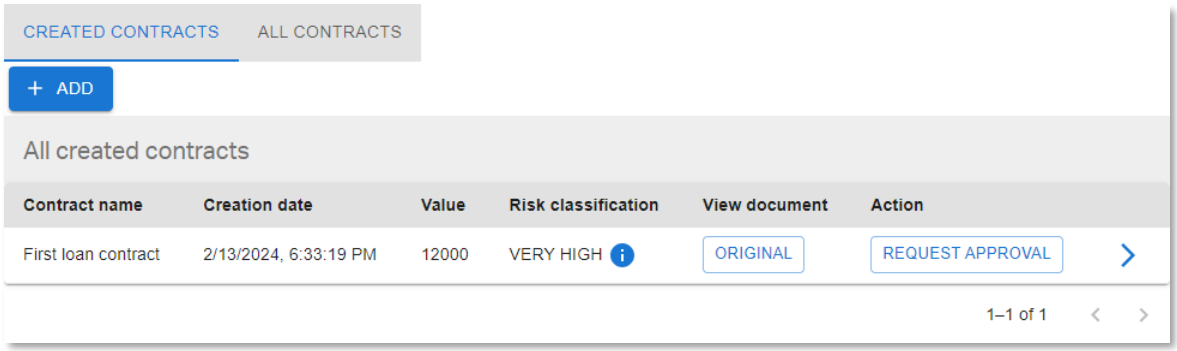


5. Select the **Loan Contract** option and fill the contract properties as follows:

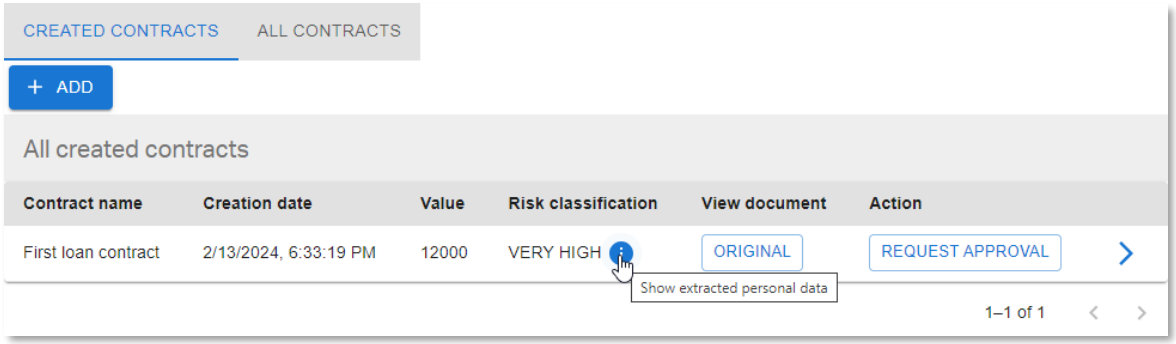
Property	Value
Document name	`First loan contract`
Contract value	`12000`
Monthly installments	`12`
Yearly income	`100000`
Contract requester email	**<your email>**

6. Click **Add** to create the contract.





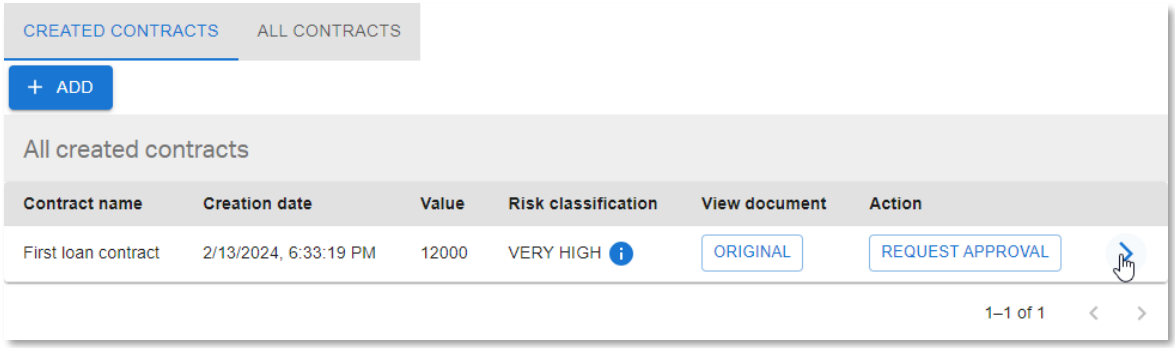
7. Click on (information icon) next to the **VERY HIGH** risk classification value to see which terms the call to the Magellan Risk Guard API has identified and extracted.



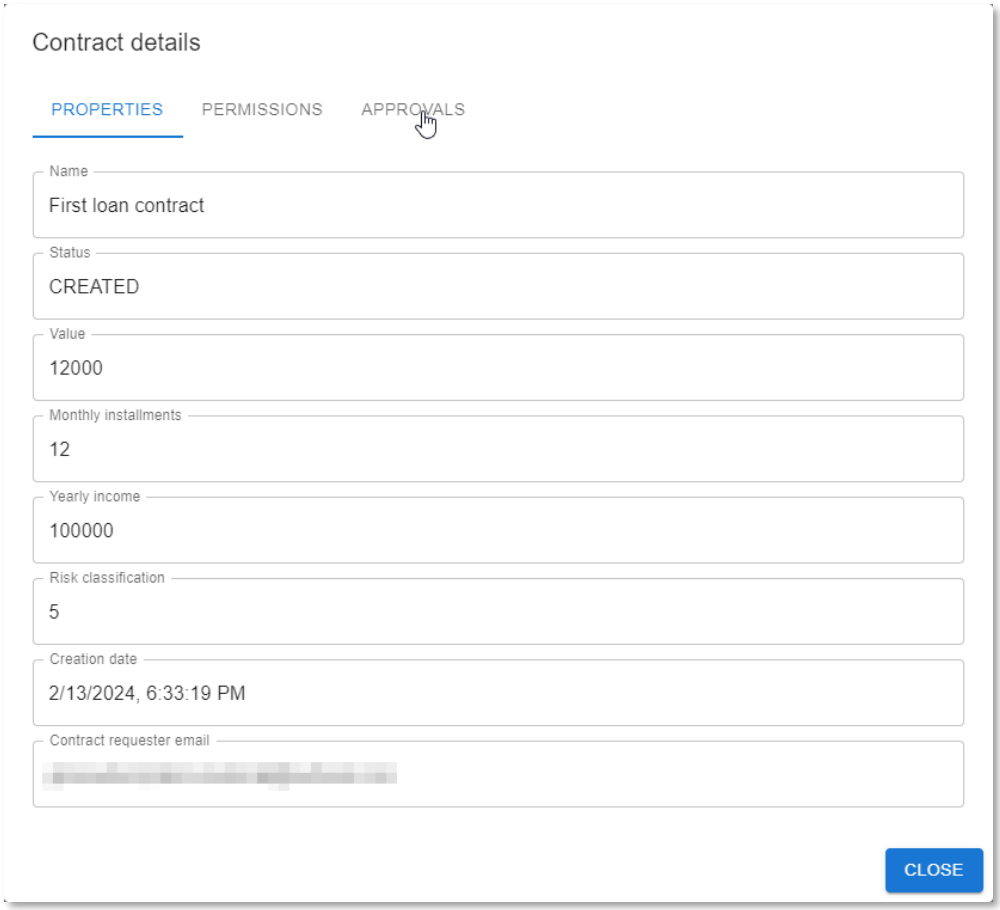
Contrary to the previous contract you created, this contract contains high risk personal information, such as a social security number (considered very high risk), a credit card number, a bank account, and many person names (hence risk classification = VERY HIGH). Some addresses, geographic locations, and organization names were also found.

8. Click **CLOSE** to close the **Extracted Terms** information screen.

9. Click on > (caret arrow icon) to view the contract details.

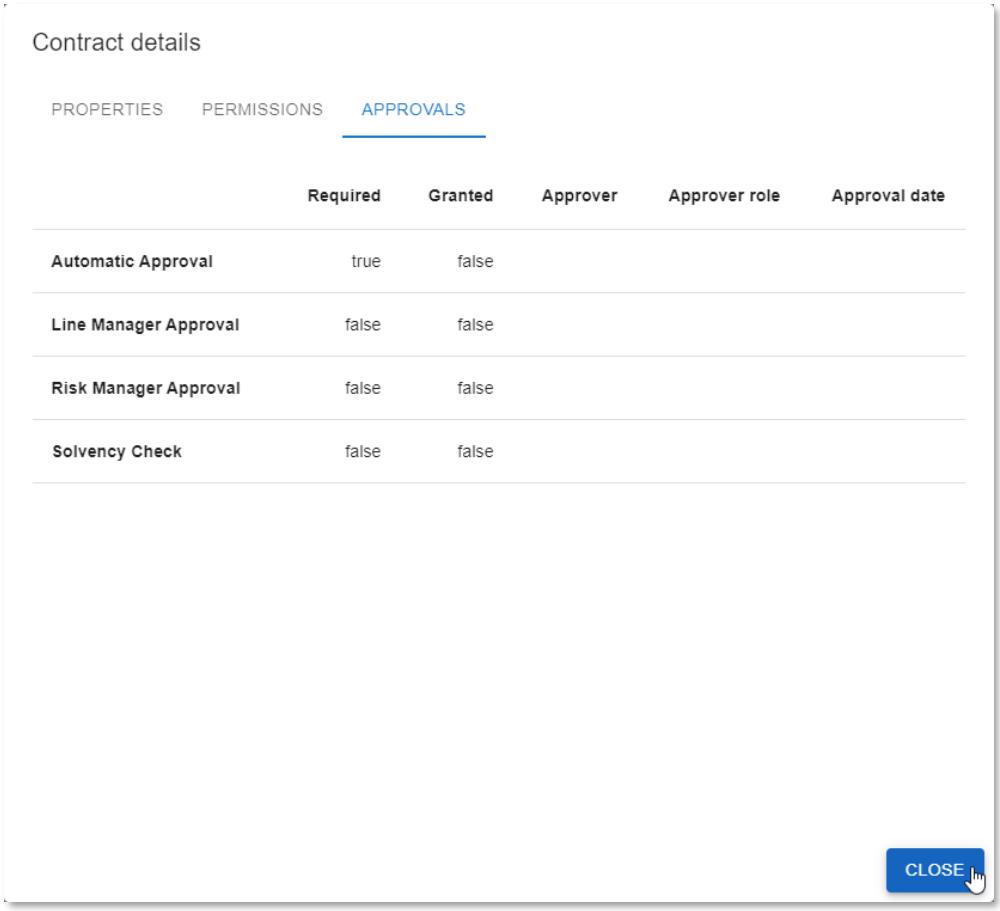


10. In the **PROPERTIES** tab of the **Contract details** screen note that, since this is a loan contract, the monthly installments and yearly income are also displayed. The risk classification is now equal to 5 (the corresponding integer value for the VERY HIGH risk level).
11. Click the **APPROVALS** tab to have a look at the different approvals for this contract.

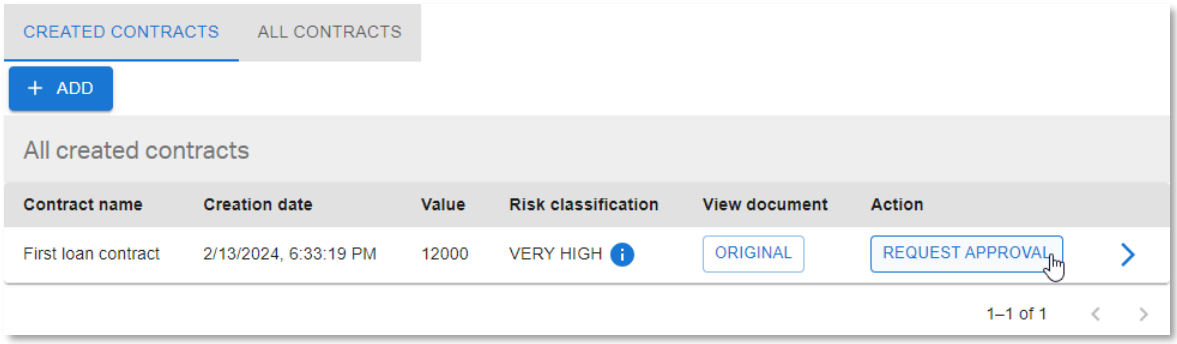


12. The **APPROVALS** tab displays the different approval steps (traits) for the loan contract type (**Automatic Approval**, **Line Manager Approval**, **Risk Manager Approval**, and the additional **Solvency Check**).

13. Click **CLOSE** to close the **Contract details** screen.



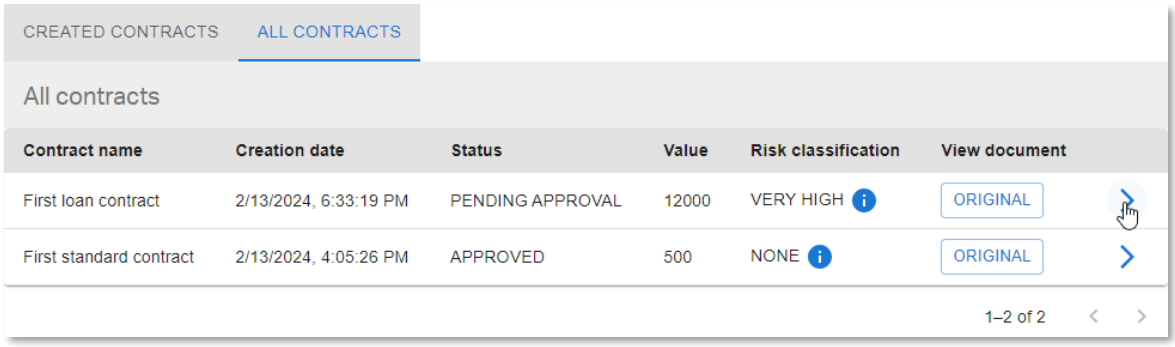
14. Back in the **CREATED CONTRACTS** view, you can now launch the approval workflow. Click the **REQUEST APPROVAL** button in the **Action** column to do that.



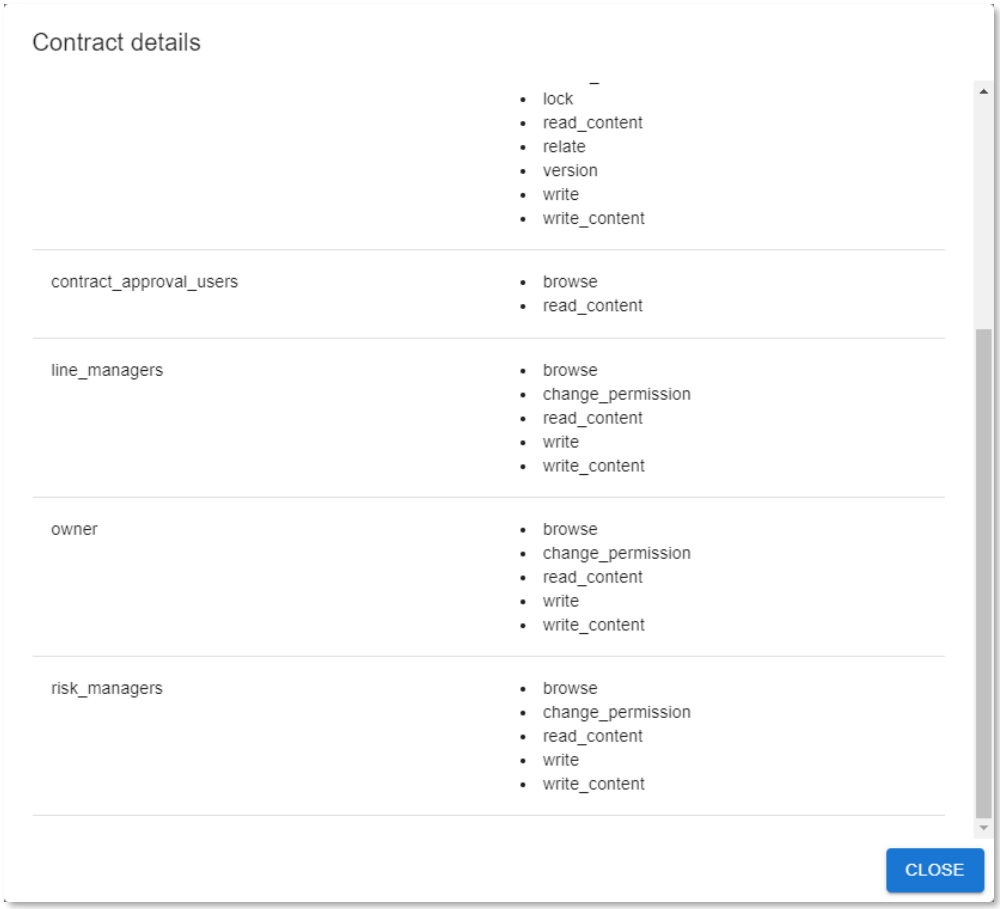
15. The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in ‘CREATED’ status. Open the **ALL CONTRACTS** tab to see its current status.

    The status is now ‘PENDING APPROVAL’.

16. Click > (caret arrow icon) to view the contract details again.

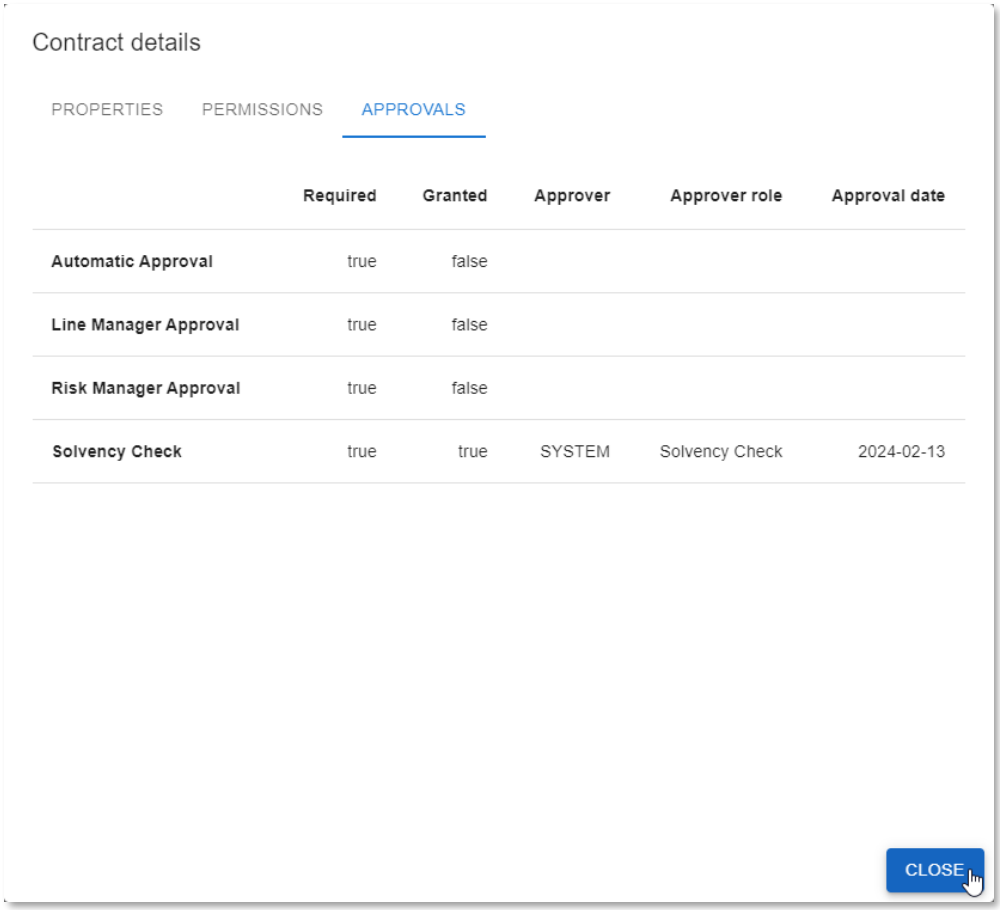


17. From the **PERMISSIONS** tab you can see that the ACL is changed to grant custom permissions to allow the approval by the line manager and the risk manager.

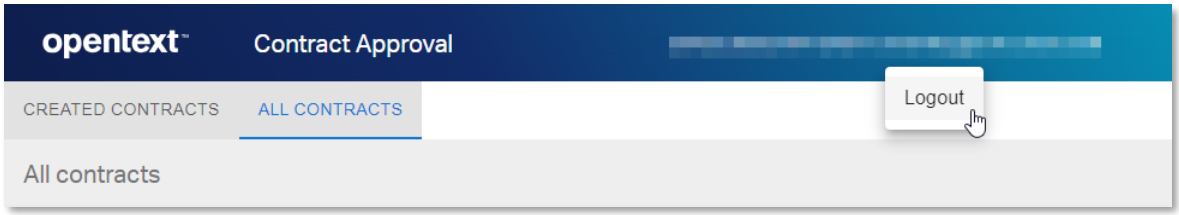


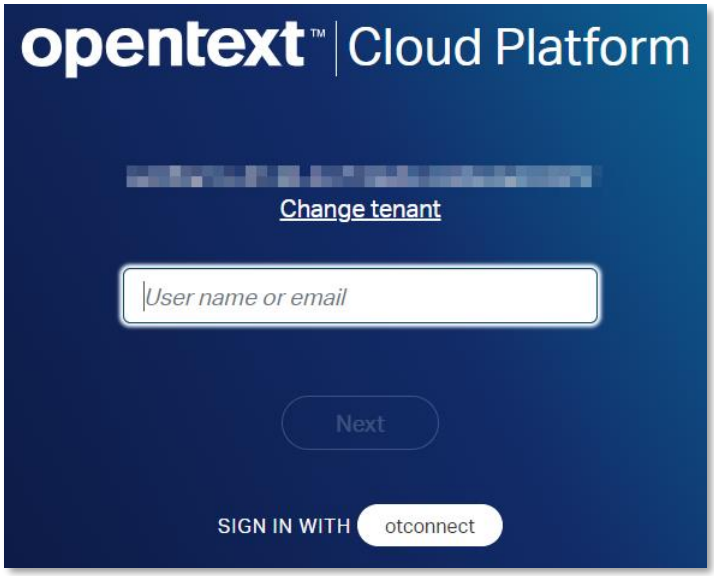
18. Additionally, from the **APPROVALS** tab, you can see that the **Solvency Check** approval is granted (because the requester is solvent) and that the **Line Manager Approval** and **Risk Manager Approval** are required.

    Click **CLOSE** to close the **Contract details** screen.

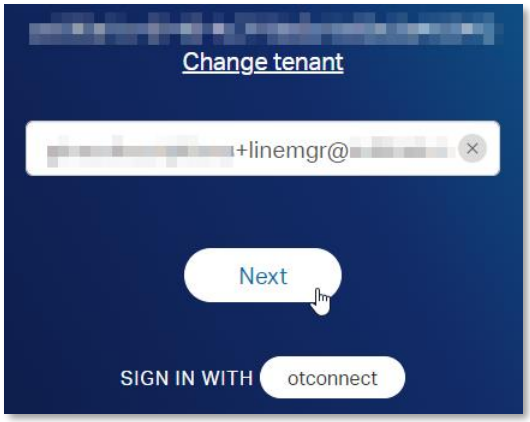


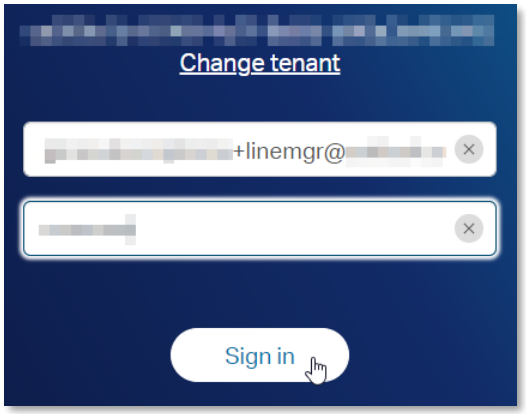
19. To perform the manual approval by the line manager, you must first sign in as the line manager. To do this, click the user name (email) at the top of the application screen and choose **Logout**.



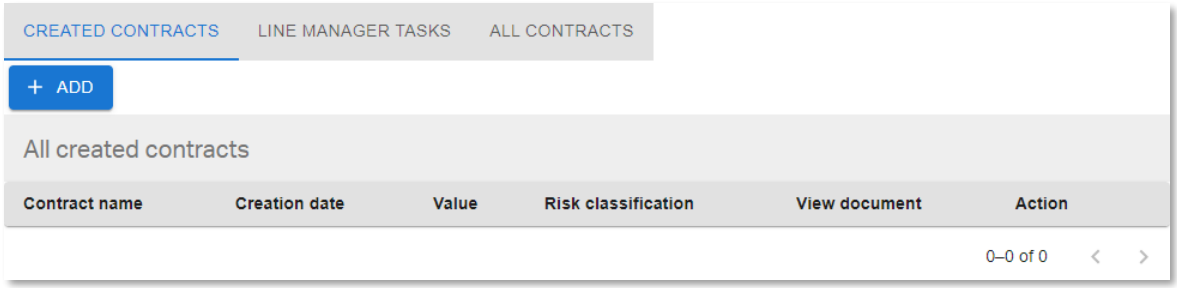


20. Sign in with the **line manager** email (the one with **+linemgr** before the @).

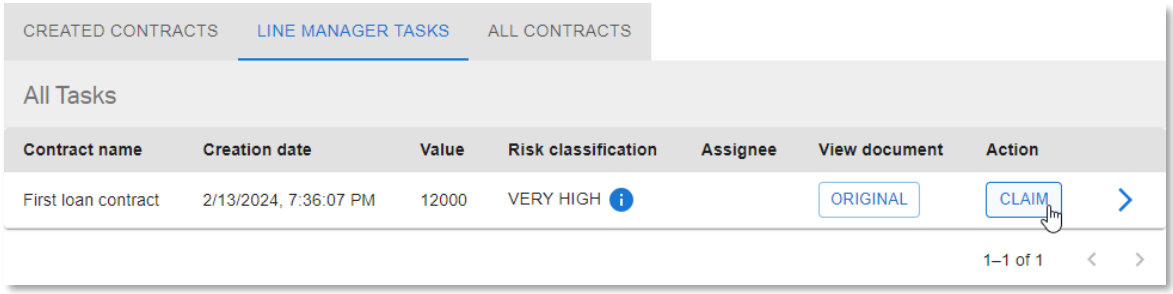




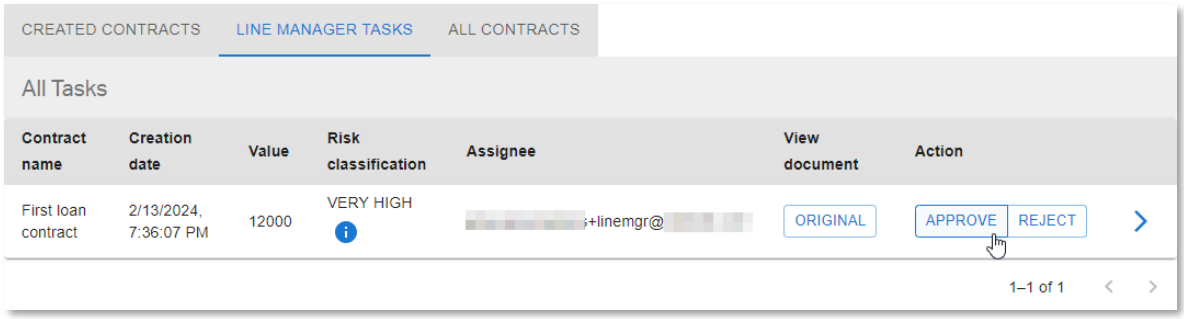
21. Since you are now logged in as line manager, in addition to the regular user's **CREATED CONTRACTS** and **ALL CONTRACTS** tabs, you can also see the **LINE MANAGER TASKS** tab.



22. Click the **LINE MANAGER TASKS** tab. An approval task is waiting for the Line Manager to approve.
23. First a line manager has to claim the approval task. So, click **CLAIM** to assign the approval to your user.



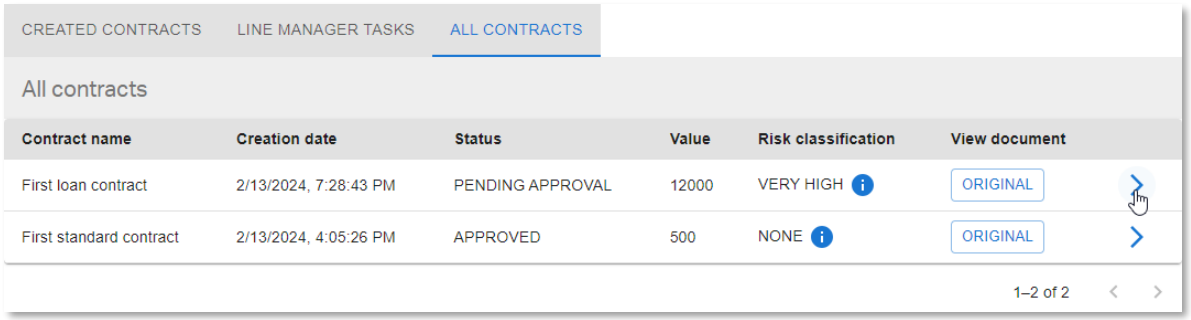
24. Now you can click **APPROVE** to approve as the Line Manager.



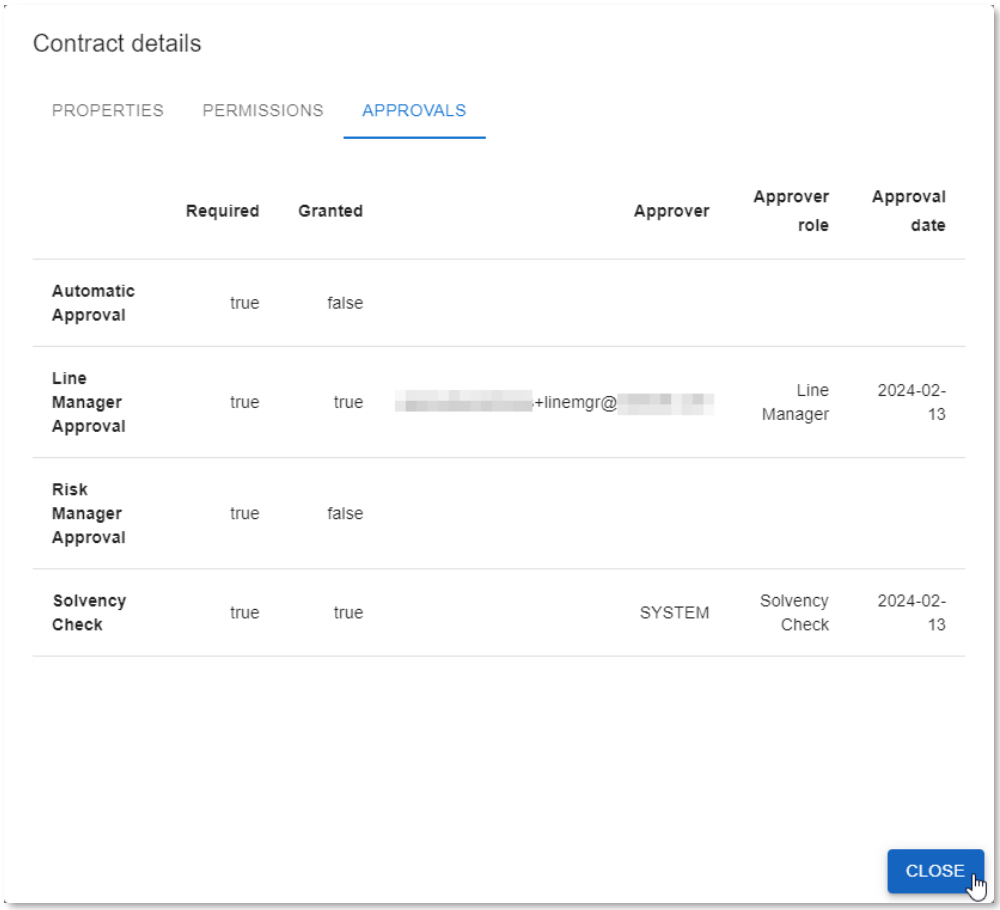
25. The contract has now disappeared from the **LINE MANAGER TASKS** view. Open the **ALL CONTRACTS** tab to see its current status.

    The status is still 'PENDING APPROVAL' because the risk manager approval is also pending.

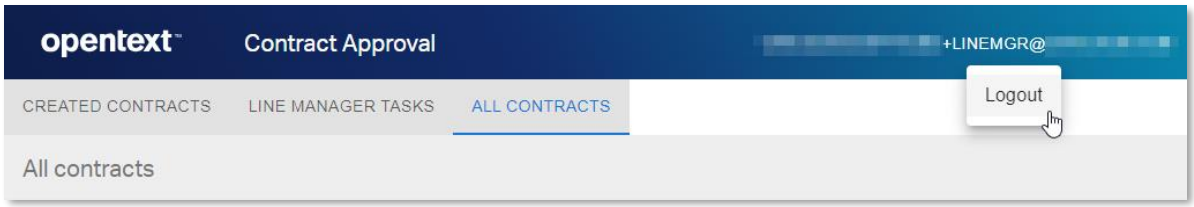
26. Click on > (caret arrow icon) to view the contract details again.

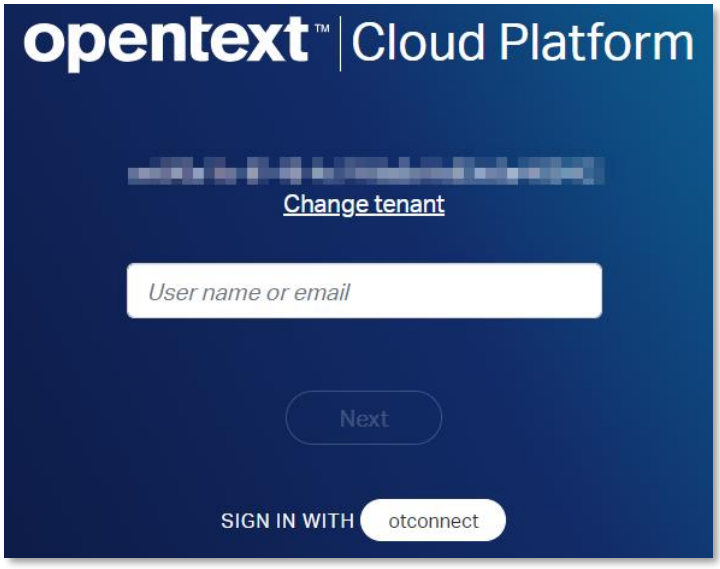


27. The **APPROVALS** tab shows that the **Line Manager Approval** is granted.
    Click **CLOSE** to return to the **ALL CONTRACTS** view.

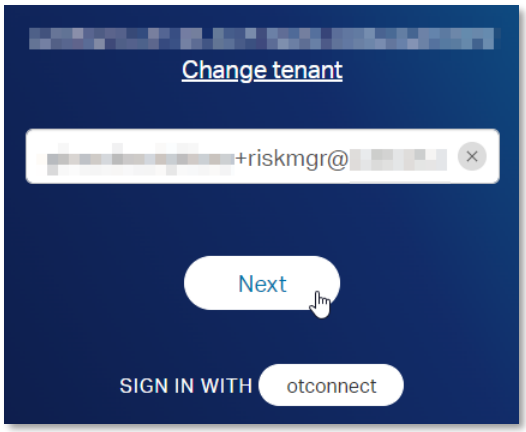


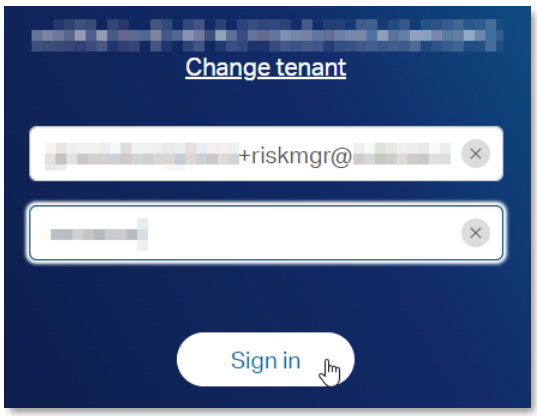
28. In the same way as with the line manager, to perform the manual approval by the risk manager, choose to **Logout** and sign in as the risk manager.



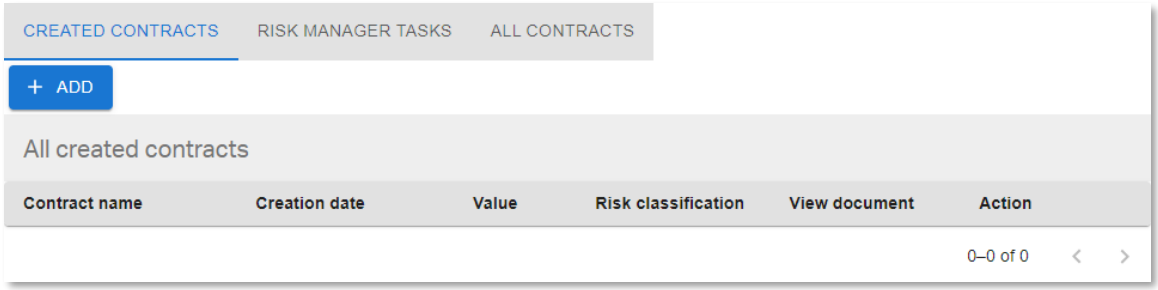


29. Sign in with the **risk manager** email (the one with **+riskmgr** before the @).

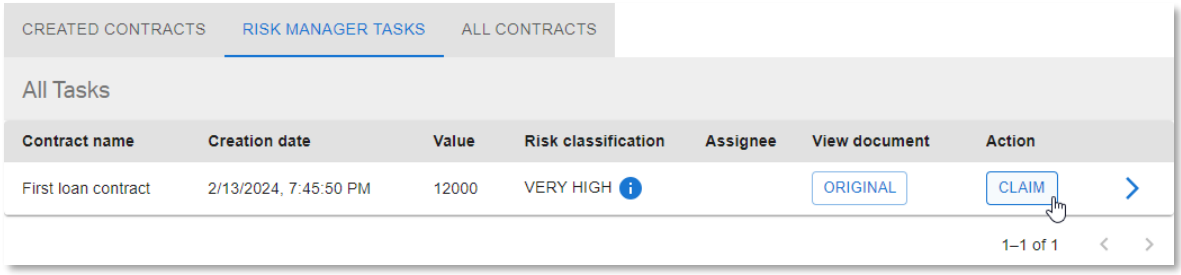




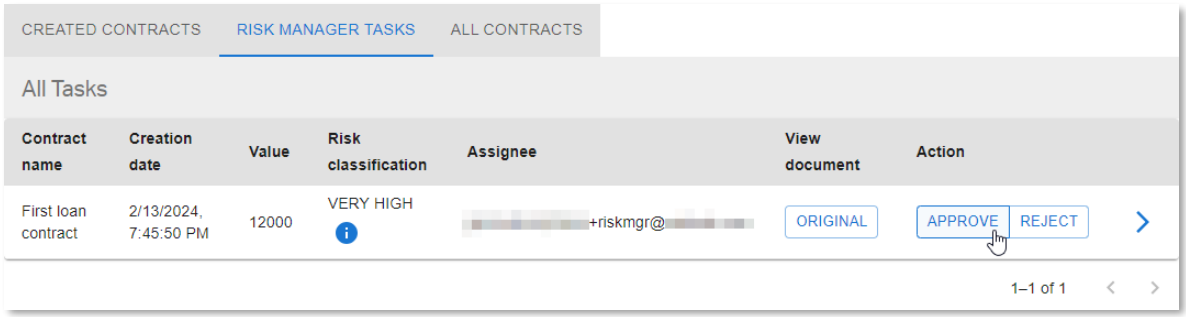
30. Since you are now logged in as risk manager, you can see the **RISK MANAGER TASKS** tab (and no **LINE MANAGER TASKS** tab).



31. Click the **RISK MANAGER TASKS** tab. An approval task is now waiting for the Risk Manager to approve.
32. First a risk manager has to claim the approval task. So, click **CLAIM** to assign the approval to your user.

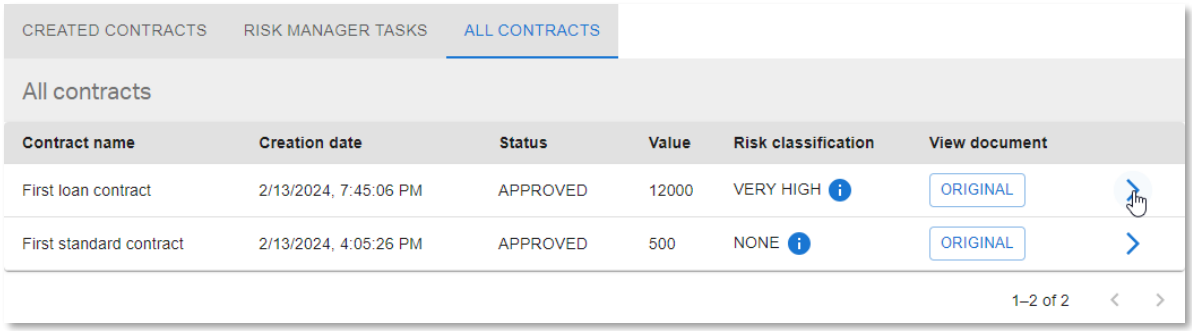


33. Now you can click **APPROVE** to approve as the Risk Manager.

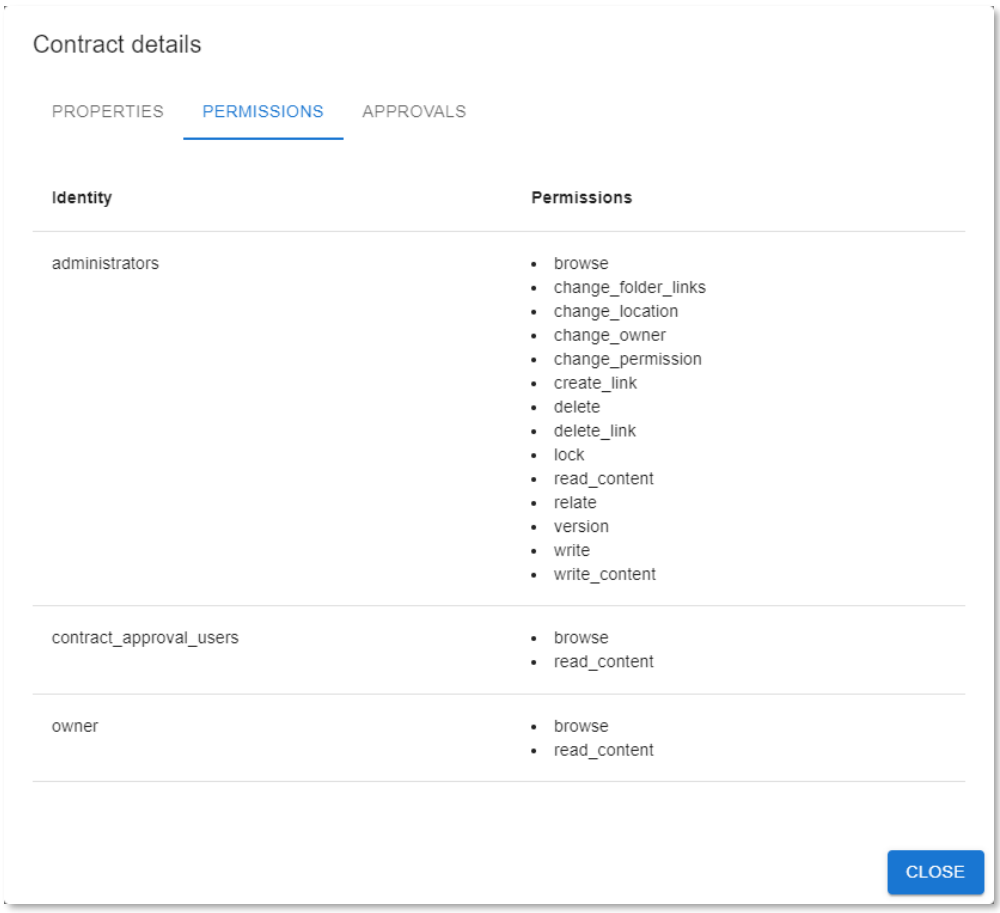


34. The contract has now disappeared from the **RISK MANAGER TASKS** view. Open the **ALL CONTRACTS** view to confirm that the contract has been automatically approved (**Status** column shows **APPROVED** status).

35. Click on > (caret arrow icon) to view the contract details.

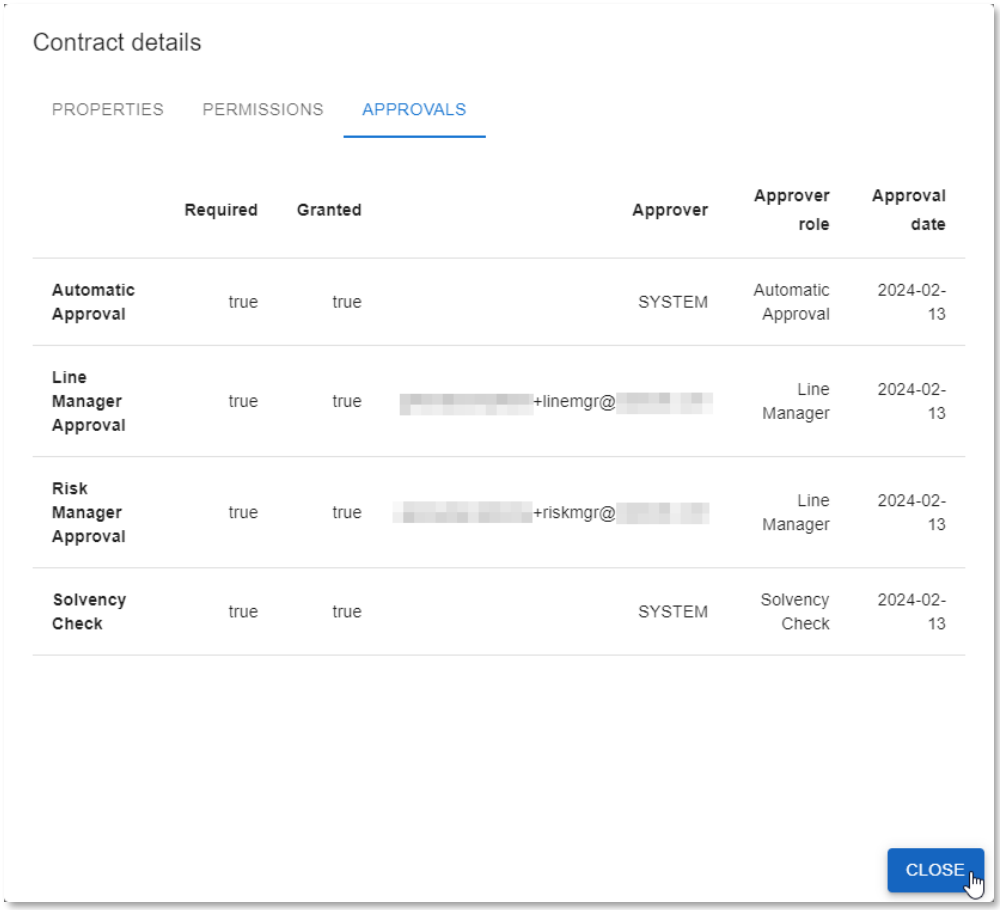


36. From the **PERMISSIONS** tab you can see that the ACL is changed so that everyone except the administrators has read permissions.



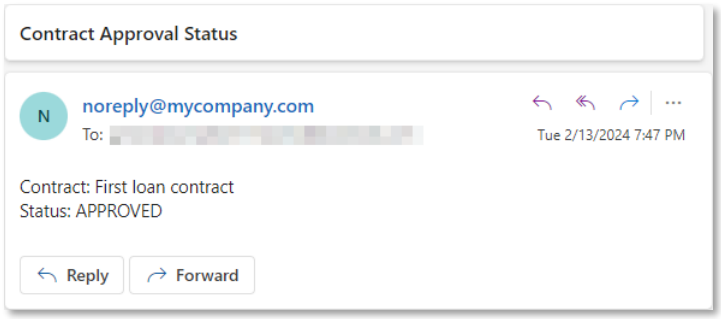
37. The **APPROVALS** tab now shows that both the **Risk Manager Approval** and the **Automatic Approval** are granted. That is, all four approvals were required for this loan contract, and all four approvals are granted.

    Click **CLOSE** to return to the **ALL CONTRACTS** view.



38. The first loan contract has now been approved.

    Go to your email client and check the inbox to confirm recieving the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.



Next step:

Reject a manual contract approval task.

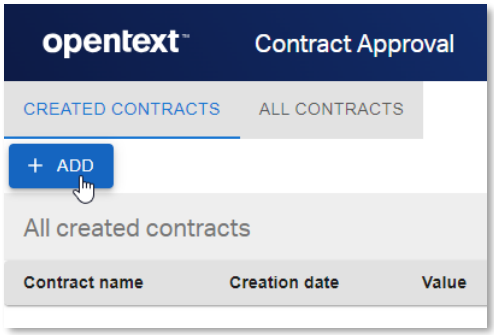
15.4Reject a manual contract approval task

You have now successfully approved two contracts with two completely different approval flows. In this section, you will test the scenario where an approver does not approve (that is, rejects) the contract.

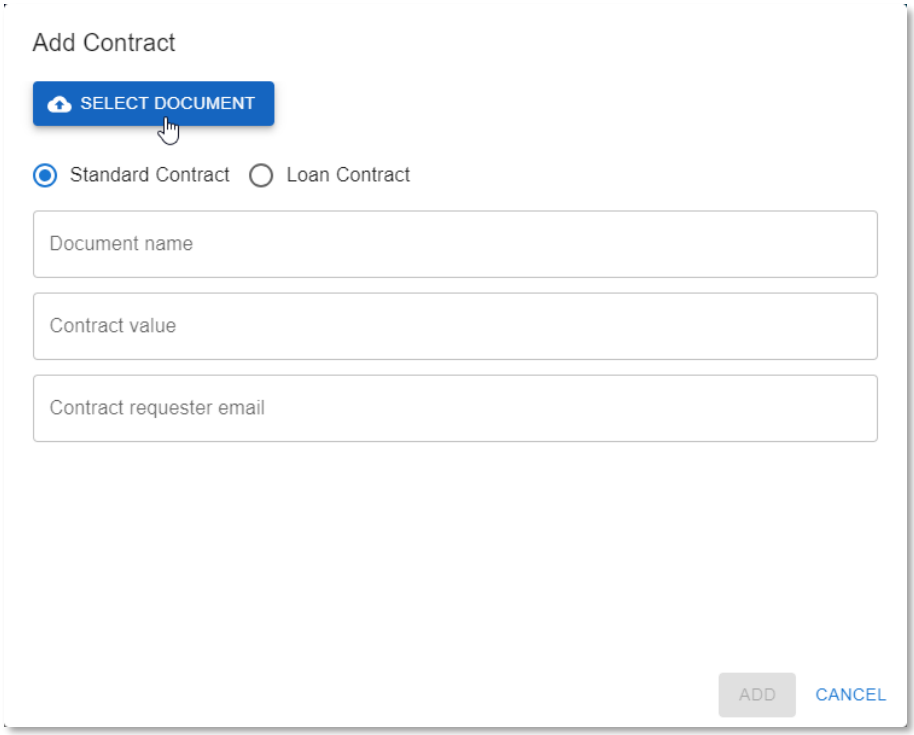
You will create a contract with the following characteristics:

- **Type: standard contract**
- **Value: above 1000** (requires Line Manager approval)
- **Risk classification: above 3, that is, HIGH or VERIFY HIGH** (requires Risk Manager approval)

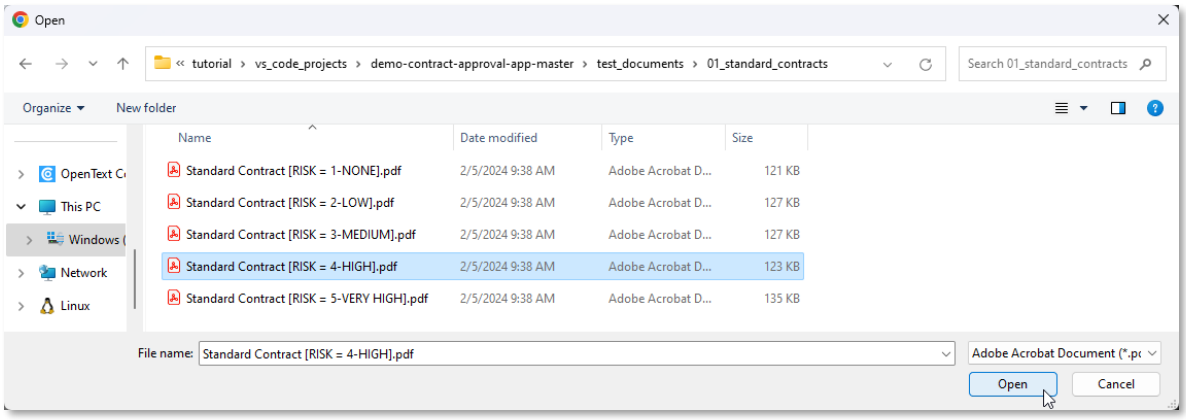
1. Log back in as the regular user, select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



2. From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.



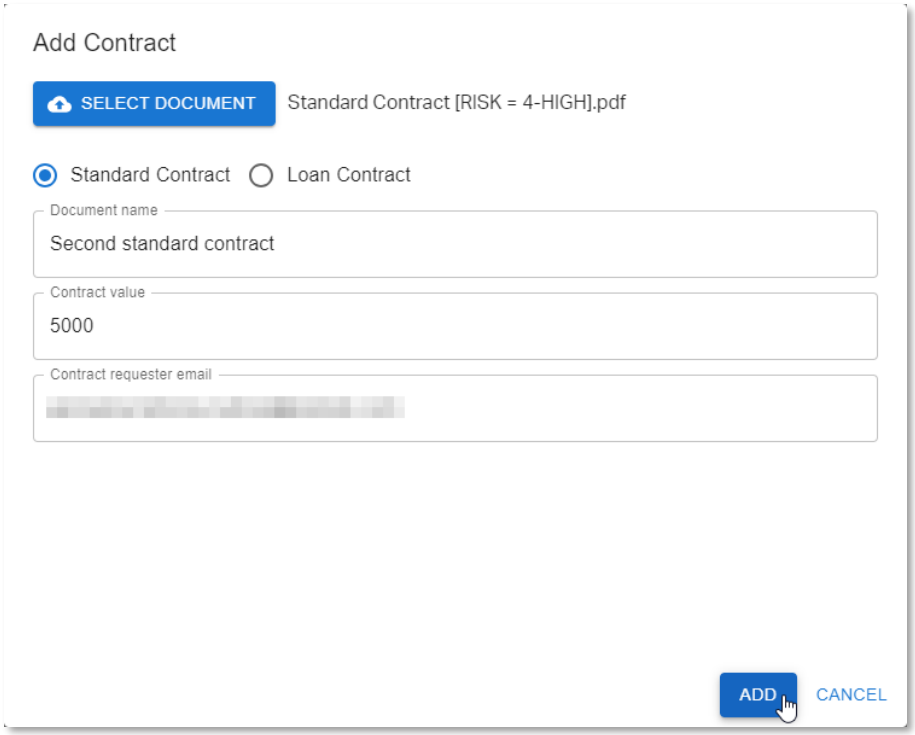
3. From the **test_documents** folder, open the **01_standard_contracts** subfolder and select the **Standard Contract [RISK = 4-HIGH].pdf** file.



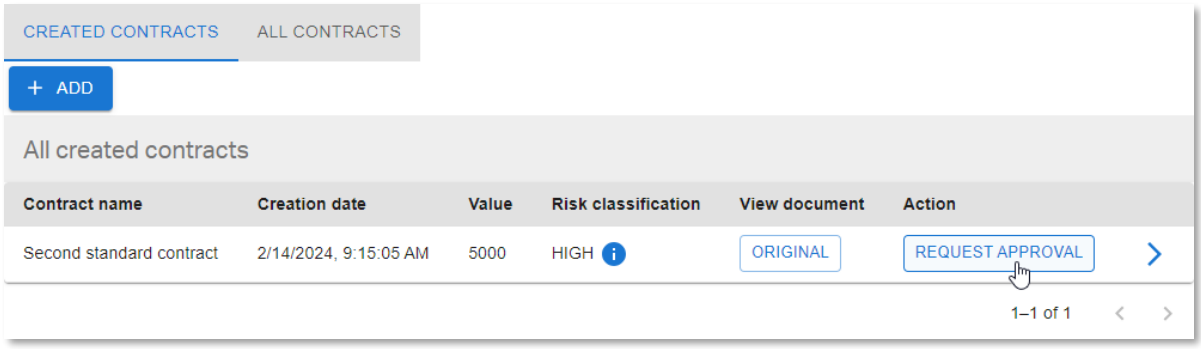
4. Make sure the **Standard Contract** option is selected and fill the contract properties as follows:

Property	Value
Document name	`Second standard contract`
Contract value	`5000`
Contract requester email	**<your email>**

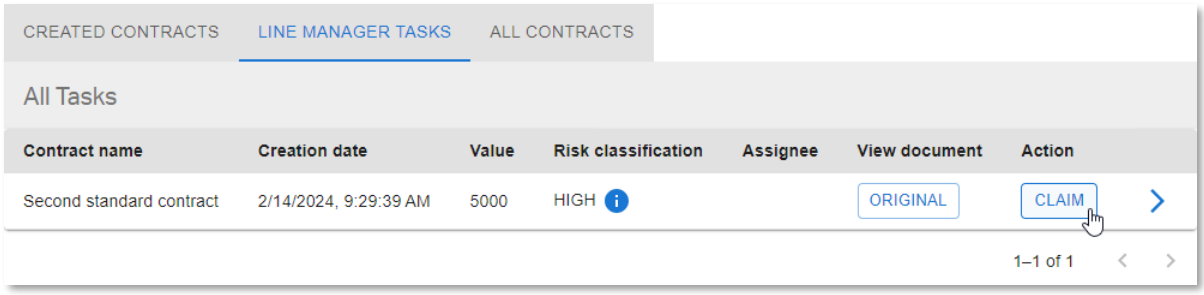
5. Click **Add** to create the contract.



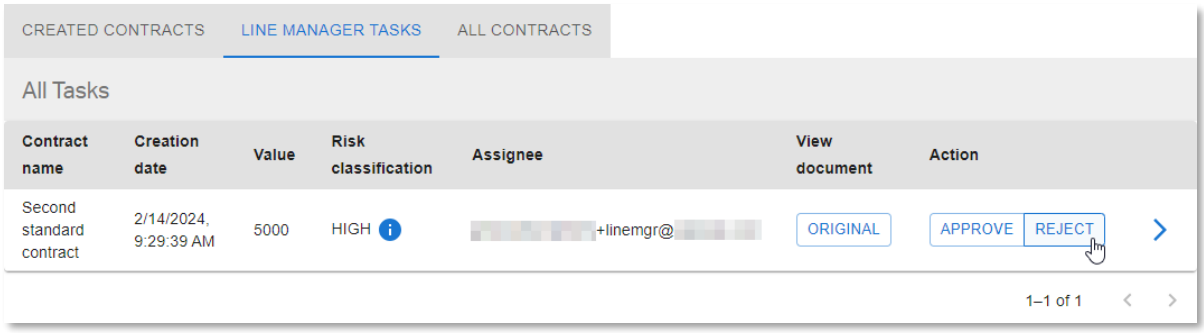
6. Click the **REQUEST APPROVAL** button in the **Action** column to launch the approval workflow.



7. The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status.
8. Sign in as the line manager and go to the **LINE MANAGER TASKS** tab. You can see the new Line Manager approval task.
9. Click **CLAIM** to assign the approval to your user.

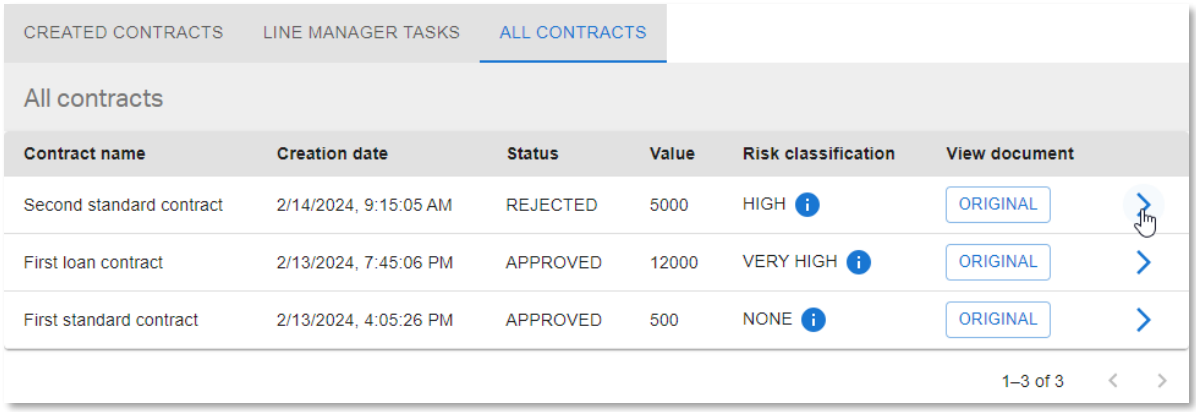


10. Click **REJECT** to reject the contract approval.

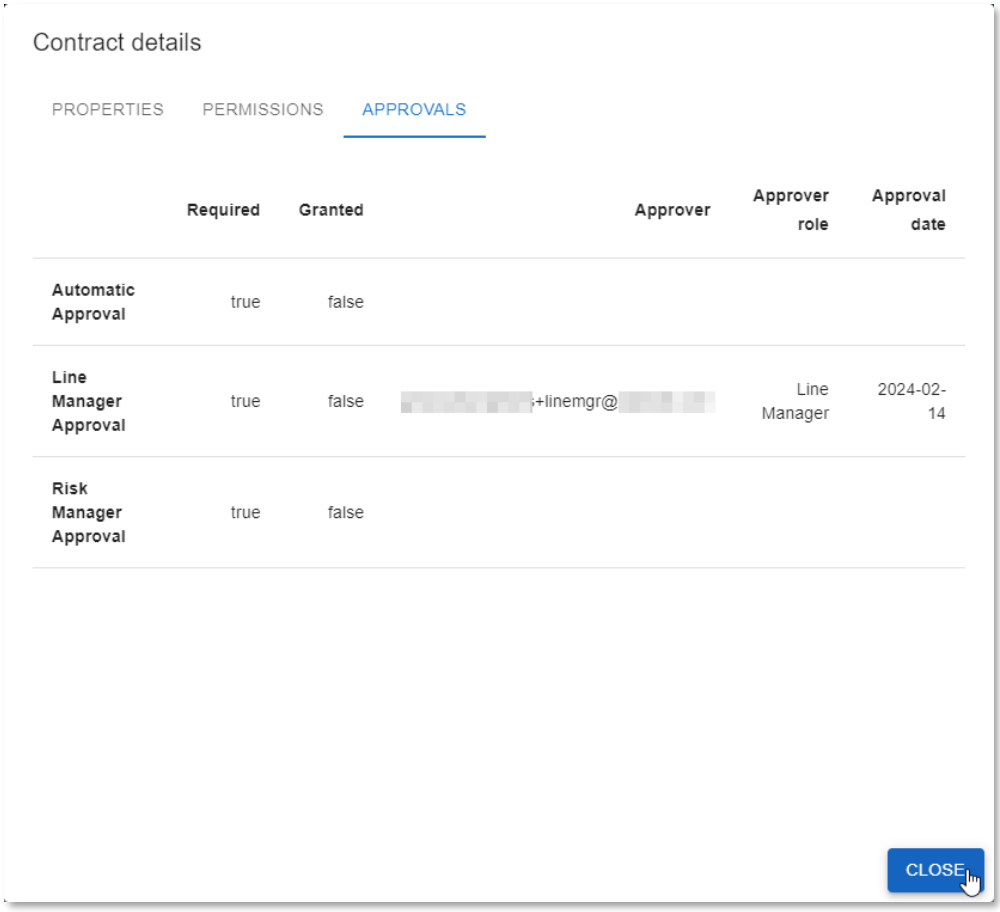


11. The contract has now disappeared from the **LINE MANAGER TASKS** view. Open the **ALL CONTRACTS** view to see that the contract has indeed been rejected (**Status** column shows **REJECTED** status).

12. Click > (caret arrow icon) to view the contract details and check the different approvals (traits).

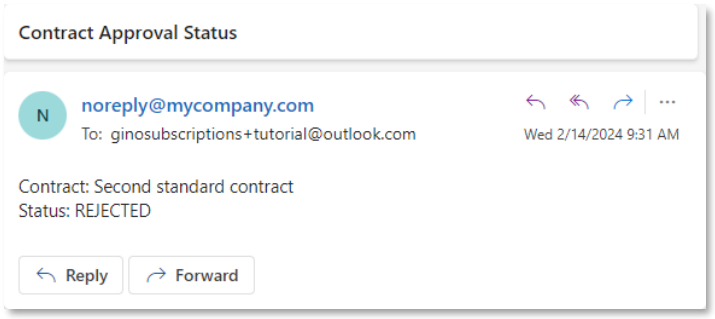


13. From the **Line Manager Approval** entry, you can see that the contract has been rejected because, although the **Approval date** has been set, **Granted** is false.



14. The second standard contract has not been approved (that is, it has been rejected).

    Go to your email inbox and confirm receiving the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.



Next step:

Expire a manual contract approval task.

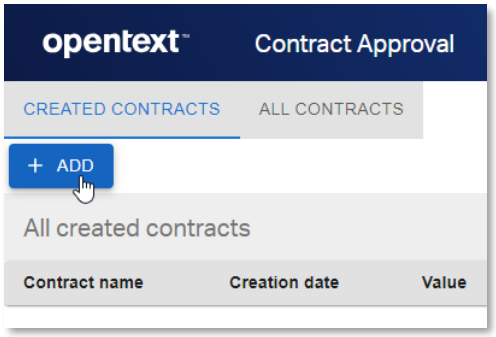
15.5Expire a manual contract approval task

1. The last scenario to run through is to let an approval task expire (happens after 5 minutes).

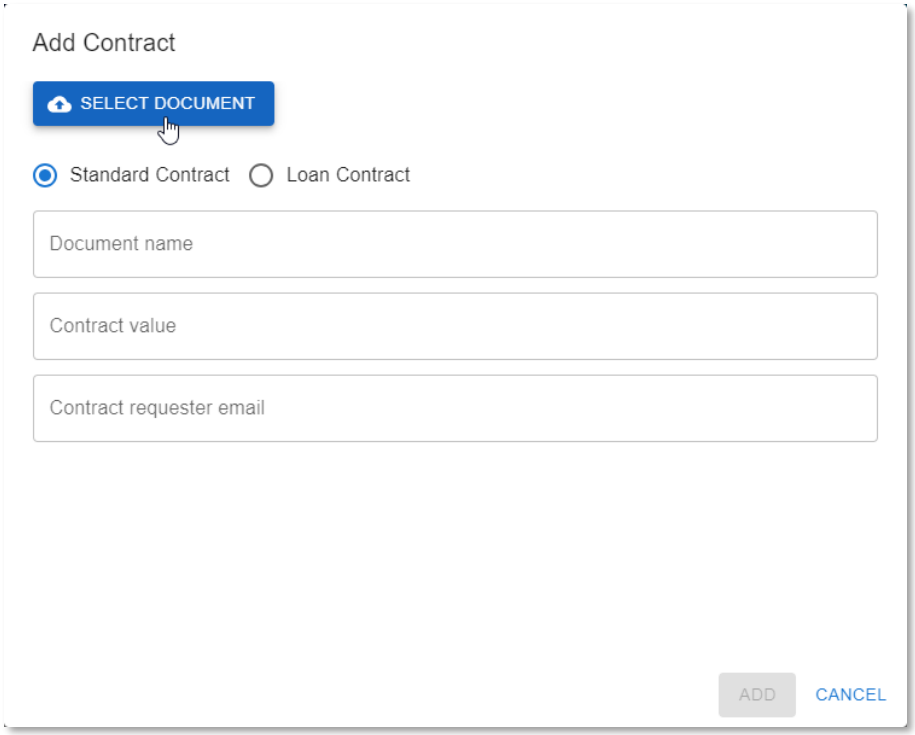
   You will create a contract with the following characteristics:

   - **Type: loan contract**
   - **Value: below 5000** (doesn't require Line Manager approval)
   - **Risk classification: above 3, that is, HIGH or VERIFY HIGH** (requires Risk Manager approval)

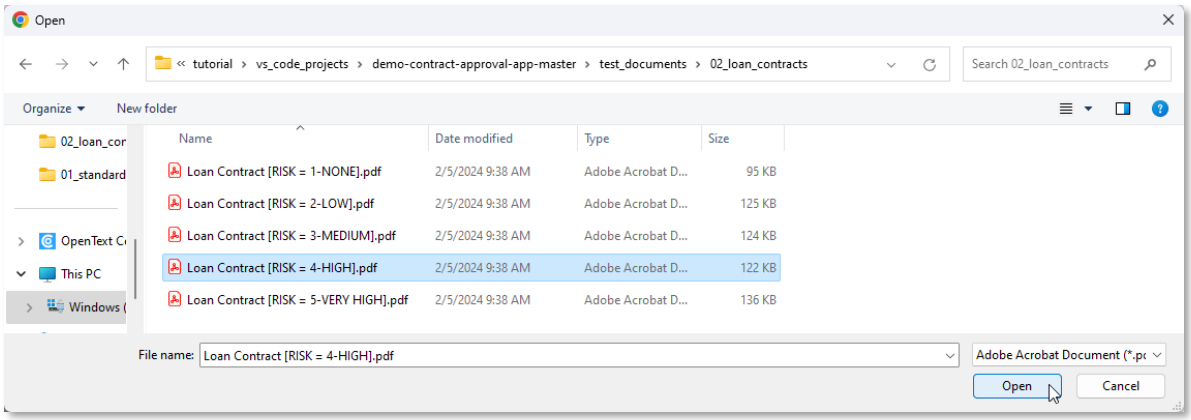
2. One more time, login with the regular user, select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



3. From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.



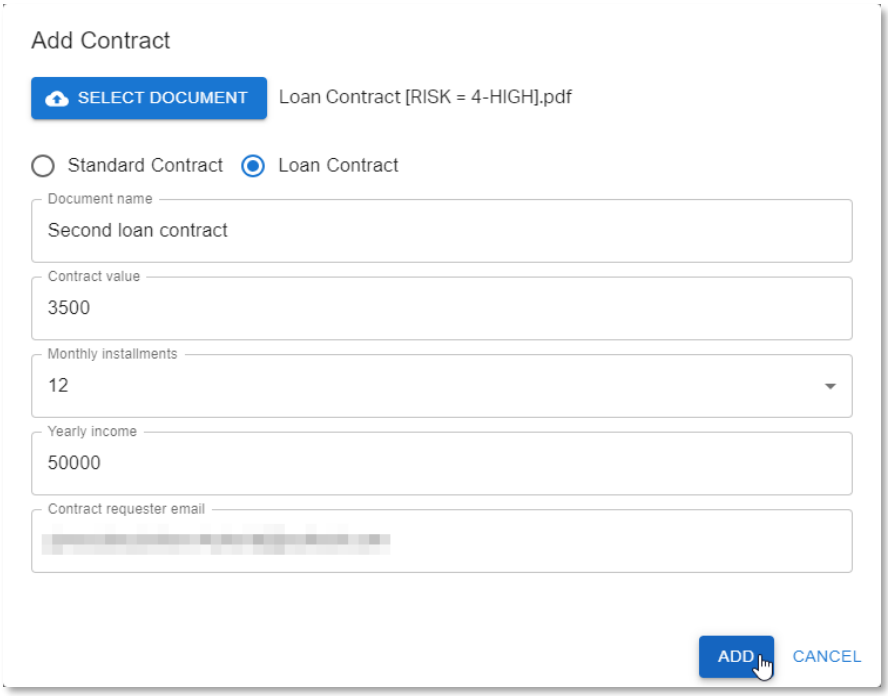
4. From the **test_documents** folder, open the **02_loan_contracts** subfolder and select the **Loan Contract [RISK = 4-HIGH].pdf** file.



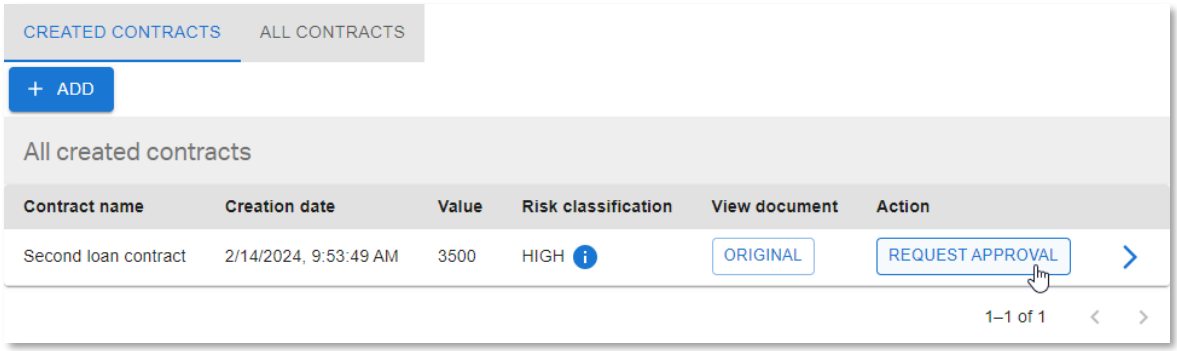
5. Select the **Loan Contract** option and fill the contract properties as follows:

Property	Value
Document name	`Second loan contract`
Contract value	`3500`
Monthly installments	`12`
Yearly income	`50000`
Contract requester email	**<your email>**

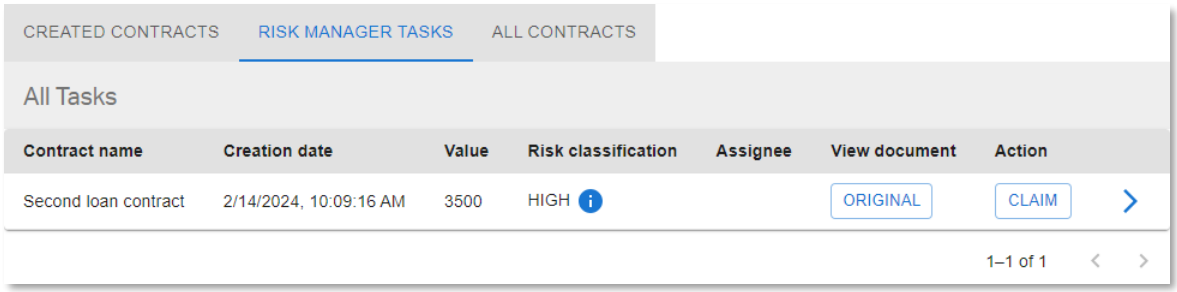
6. Click **Add** to create the contract.



7. Click the **REQUEST APPROVAL** button in the **Action** column to launch the approval workflow.

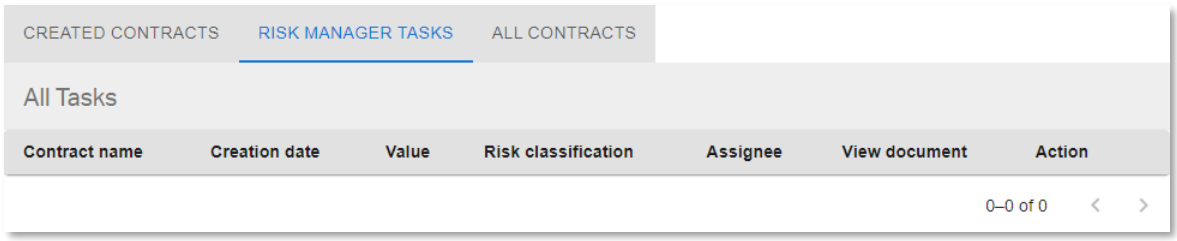


8. The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status.
9. Sign in as the risk manager and go to the **RISK MANAGER TASKS** tab. You can see the new Risk Manager approval task.



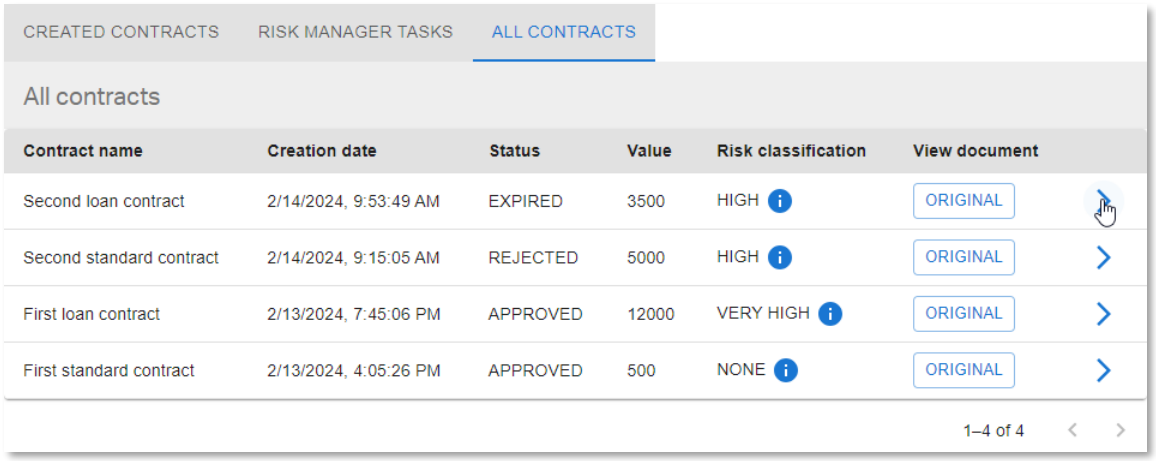
10. To test whether or not the approval task will expire, wait for more than 5 minutes and refresh the application page from the web browser.
11. After refreshing the application screen, go back to the **RISK MANAGER TASKS** view.

    The contract has now disappeared from the **RISK MANAGER TASKS** view.

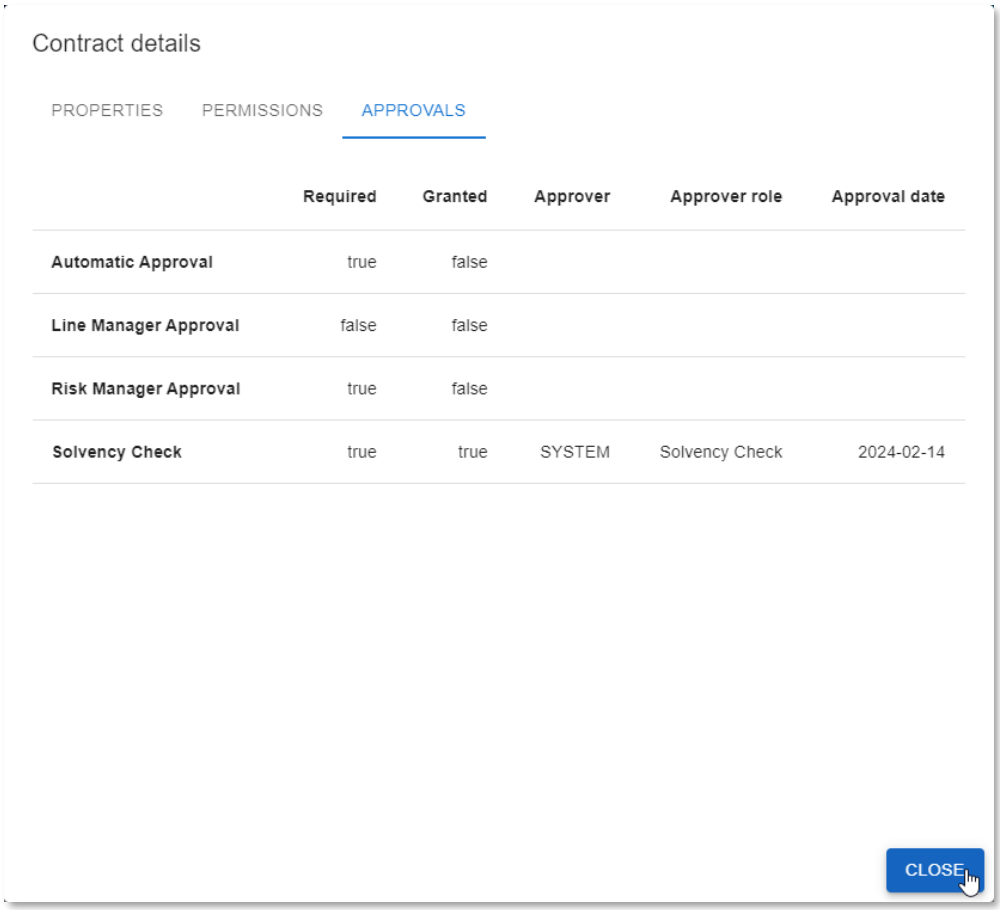


12. Open the **ALL CONTRACTS** view to see that the contract approval has indeed expired (**Status** column shows **EXPIRED** status).

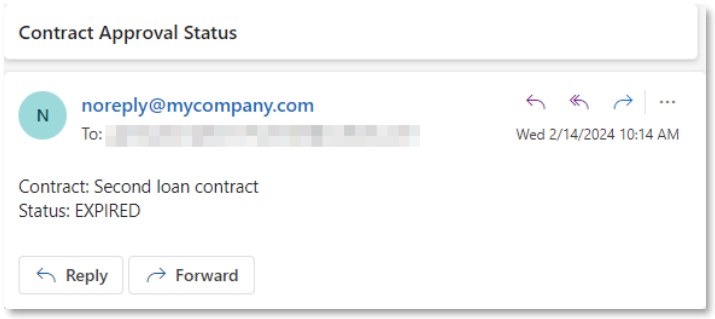
13. Click > (caret arrow icon) to view the contract details and check the different approvals (traits).



14. You can see that the contract is not approved by the Risk Manager since the **Risk Manager Approval** has NOT been granted. However, you can also see that there has not been a rejection action as the approval date is not filled. The approval activity simply timed out.



15. The second loan contract has not been approved since the Risk Manager Approval step expired. Go to your email inbox and confirm receiving the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.



CONGRATULATIONS!

You have now finished building and testing the Contract Approval application. You are at the end of the main part of the tutorial.
There is one more bonus exercise where you will learn about the ocp cli. If you are interested in build automation and CI/CD for your applications, it is recommended that you do that exercise module as well.

Next exercise module:

Bonus exercise: Use the ocp command line interface.

16 [15’] Bonus exercise: Use the ocp command line interface

Learn how to:

- Install the stand-alone ocp command line interface (cli)
- Use organization profiles with the ocp cli
- Deploy projects with the ocp cli

In this exercise module you will use the OpenText Cloud Platform command line interface, or in short **ocp cli**. This cli allows you to manage organization profiles, deploy projects, and to create local packages.

Note

One of the main purposes of this cli is to be used in a CI/CD scenario. It allows to automate the deployment of the application models needed to run automated tests. Please refer to the OpenText Cloud Developer Tools for VS Code User Guide for more details on CI/CD and how to use the ocp cli in that context.

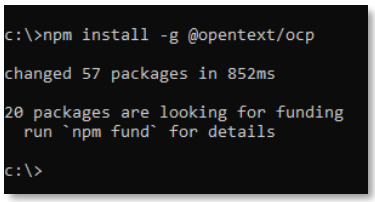
16.1Install the ocp cli

The ocp cli is available from npm, which is a public software registry. Installing the ocp cli is done with the npm command, which is already available as part of the Node.js installation.

1. Open a command prompt or terminal and execute the command:

   ```
   npm install -g @opentext/ocp
   ```

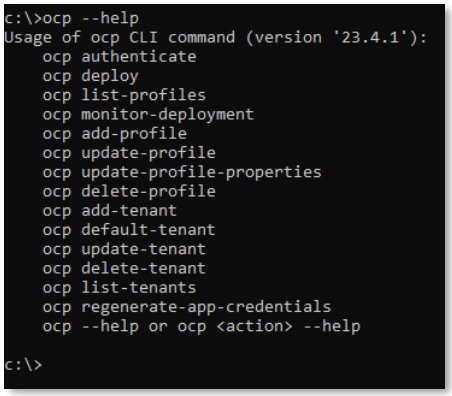
   This installs the ocp cli on your system and makes it available globally.

2. In the command prompt or terminal execute the following command to see the information on how to use the ocp cli:

   ```
   ocp --help
   ```

   This shows the actions that can be used with the cli.

3. To see the detailed information on how to use a specific command use:

   ```
   ocp <action> --help
   ```

Next step:

Use the developer profile.

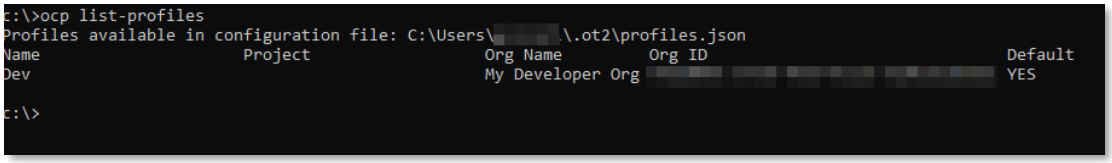
16.2Use the developer profile

To access the OpenText Developer Cloud APIs you already set up a developer profile in VS Code (see Add an organization profile). This profile's data is shared between the ocp cli and VS Code. This means that the cli can be used to see and manage the already added profile.

1. In the command prompt or terminal execute the following command to display the already configured profile:

   `ocp list-profiles`

   This shows the available profile as added earlier:

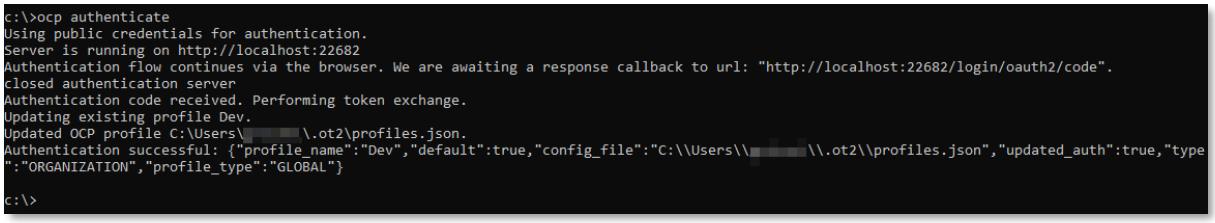
   

   If you have set up your organization profile and tenant according to the tutorial and have not added any other organization, there is only one organization profile being shown and it is also the Default organization profile. Otherwise, multiple organization profiles will be listed with one being the Default organization profile.

2. Use the **authenticate** action to authenticate using the default profile. In the command prompt or terminal execute the command:

   `ocp authenticate`

   This starts the authentication flow and after successful authentication a message is displayed that an authentication code was received:

Next step:

Deploy the application project from command line.

16.3 Deploy the application project from command line

One of the main purposes of the ocp cli is to deploy application projects. When deploying an application project, a temporary package is first created, after which it gets uploaded to the ALM deployment service. The package is validated and the contained models are deployed. When an application project is deployed for the first time the API key data is returned and displayed.

Deploy an application project with the ocp cli using the **deploy** action.

1. In the command prompt or terminal, change the directory to be the folder that contains the **.otproject** file, as this is the main project folder of the application project.

```
c:\>cd c:\tutorial\vs_code_projects\contract_approval

c:\tutorial\vs_code_projects\contract_approval>dir
 Volume in drive C is Windows
 Volume Serial Number is [redacted]

 Directory of c:\tutorial\vs_code_projects\contract_approval

02/14/2024  12:57 PM    <DIR>          .
02/05/2024  09:38 AM    <DIR>          ..
02/06/2024  03:57 PM               302 .env
02/05/2024  09:38 AM               302 .eslintrc.yml
02/05/2024  09:38 AM                15 .npmrc
01/15/2024  10:42 AM               378 .otproject
02/05/2024  09:38 AM               598 index.html
02/13/2024  09:48 AM    <DIR>          node_modules
01/26/2024  06:28 PM    <DIR>          otresources
02/13/2024  09:44 AM           194,578 package-lock.json
02/05/2024  09:38 AM             1,200 package.json
02/05/2024  06:08 PM    <DIR>          public
02/06/2024  10:18 AM    <DIR>          src
02/05/2024  09:38 AM               705 vite.config.js
               8 File(s)        198,078 bytes
               6 Dir(s)  368,244,916,224 bytes free

c:\tutorial\vs_code_projects\contract_approval>
```

2. In the command prompt or terminal execute the command to deploy the application project:

```
ocp deploy
```

```
c:\tutorial\vs_code_projects\contract_approval>ocp deploy
The project directory is [c:\tutorial\vs_code_projects\contract_approval]
Adding https://www.opentext.com/ocp/devx/security/1.0.0/Group artifact: c:\tutorial\vs_code_projects\
.otgrp
Adding https://www.opentext.com/ocp/devx/metadata/1.0.0/Trait artifact: c:\tutorial\vs_code_projects\
it
Adding https://www.opentext.com/ocp/devx/security/1.0.0/ACL artifact: c:\tutorial\vs_code_projects\cc
```

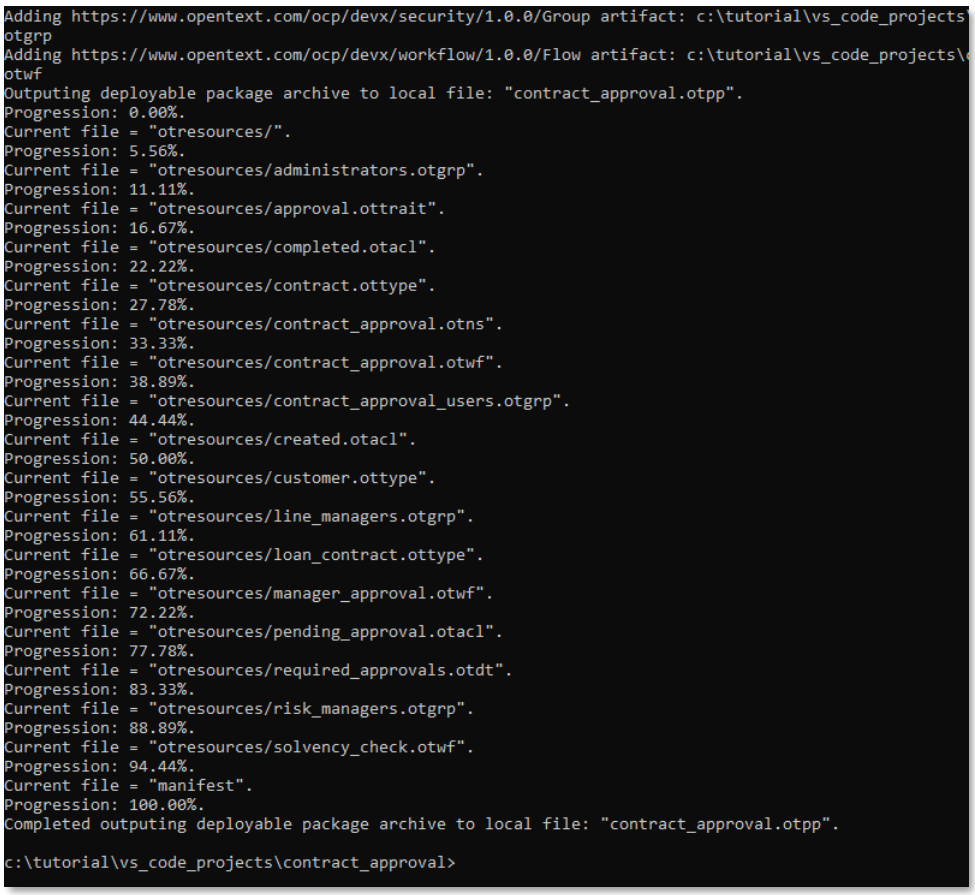
```
        ID: c315497e-3763-4d2b-8a8f-b2556806c6df. Model ID: 2fb7e90d-5af4-48e7-803e-8de1e0849b9b
            Status: success
        Name: otresources/pending_approval.otacl
        ID: a44a380f-684a-419e-840a-dabae4e9f13b. Model ID: 8d4098b7-a668-4141-8097-b6536f0e62ea
            Status: success
        Name: otresources/contract_approval_users.otgrp
        ID: 45bb1bb9-a4c3-4ef6-aff1-27bef3d1ffb0. Model ID: 615d83ce-12ac-4e85-acd7-d34cc33c8ed9
            Status: success
        Name: otresources/manager_approval.otwf
        ID: 17f3e0b6-7209-4e74-94f6-6e13b8668592. Model ID: 69cb85de-cbaf-4d5c-af5e-a31e75ea3203
            Status: success
        Name: otresources/approval.ottrait
        ID: 58f10413-283e-40b2-a8f3-cb0a7e6808fa. Model ID: a5023d26-5b25-4ceb-9a10-9ec70df9ff99
            Status: success
        Name: otresources/risk_managers.otgrp
        ID: 9f128d16-d961-4d86-8e24-d287b3a58dde. Model ID: e40a3729-afcd-4f2a-977f-a3082c03d7f3
            Status: success
        Name: otresources/solvency_check.otwf
        ID: cb9faae8-1d48-4a1e-8c07-7c98cb560b16. Model ID: 51bd0f86-f261-48a1-a19a-3b28146c4c0d
            Status: success
--------------------------------------------------------------------------------------------------
In tenants ["[redacted]"] application "contract_approval" was updated.

c:\tutorial\vs_code_projects\contract_approval>
```

3. Normally the application package is deleted after it is uploaded to the ALM deployment service. By using the deploy flag **--target** the package can be saved in a local file. This allows the file to be stored for deploying at a later moment or to a different environment.

   In the command prompt or terminal execute the following command to save the package to a local file:

   ```
   ocp deploy --target contract_approval
   ```

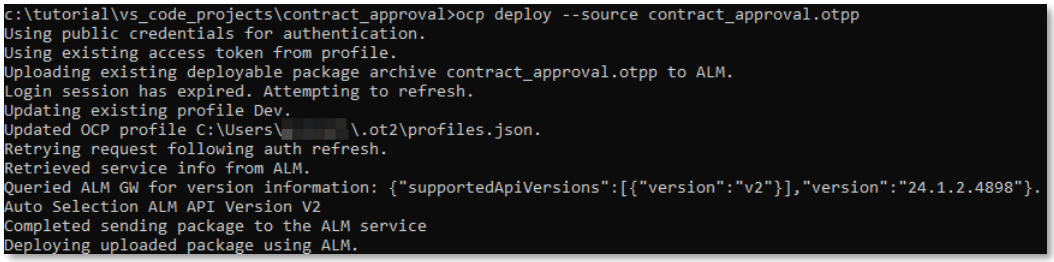
   This saves the application package to the file called **contract_approval.otpp** in the current folder.

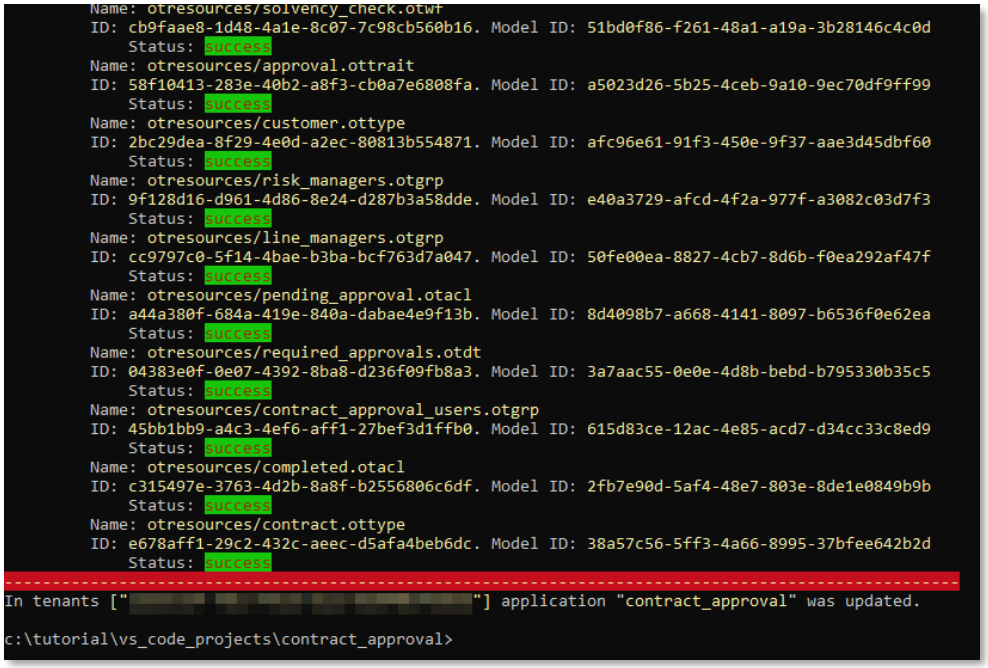
4. The created package can be deployed in a different tenant in the organization, or in another tenant in another organization. The package doesn't contain any tenant or organization specific data.

   In the command prompt or terminal execute the following command to deploy the saved package:

   ```
   ocp deploy --source contract_approval.otpp
   ```

   This deploys the application package in the same way as a deploy of the application project for which the package was created.



You have now completed this bonus exercise about using the stand-alone ocp command line interface.

About OpenText

OpenText enables the digital world, creating a better way for organizations to work with information, on-premises or in the cloud. For more information about OpenText (NASDAQ/TSX: OTEX), visit opentext.com.

Connect with us:

OpenText CEO Mark Barrenechea's blog

Twitter | LinkedIn