OpenText Cloud Developer Tutorial

Building a Contract Approval application

This tutorial has been created for software version OpenText™ Cloud Developer Tools for VS Code 22.4.1.

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

Disclaimer

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Open Text Corporation and its affiliates accept no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication. This includes the application code that is being provided with this tutorial, as this code is intended for educational purposes only and should not be used in a production setting.

Last updated: 09/26/2022

Contents

1 Introduction

This document is intended as a comprehensive course for pro-code developers who want to learn how to build applications that utilize the OpenText IM (Information Management) APIs, and how to do this in the most efficient way. More specifically, it shows how to develop a React based application in VS Code (Microsoft Visual Studio Code). The OpenText Cloud Developer functionality is added to the standard VS Code IDE (Integrated Development Environment) when you install the OpenText Cloud Developer Tools VS Code extension pack.

The application you will be building is a simple Contract Approval application which will allow to upload documents, store document related metadata for two types of contracts, do document analysis to detect PII (Personally Identifiable Information), and use a workflow to automate the different contract approval steps.

In this context, the following IM services will be used:

- The Content Storage Service (CSS) for uploading and storing of documents
- The Content Metadata Service (CMS) for storing document metadata
- The OpenText Magellan Risk Guard Service for document analysis
- The Workflow Service for executing the contract approval process

In different step-by-step exercises throughout the document, we will cover the creation of models (configuration artifacts) and how to deploy them, how to write code that talks to the IM APIs, and how to run and test the finished Contract Approval application. Each exercise builds on the previous one, so you need to perform the steps exactly as written in the order in which they are presented, and without skipping any. It should take about 6 to 8 hours to complete the tutorial. You can complete all the exercises in a single session or break them up into multiple sessions.

This is what you will learn in the different exercises, and how much time it requires:

- [25’] Setting up the Cloud Developer IDE
- [10’] Adding an organization and testing the connection
- [15’] Creating an OpenText project
- [10’] Creating a namespace
- [15’] Creating a trait definition
- [20’] Creating a file type definition
- [10’] Creating a file type definition that is a subtype
- [05’] Creating a folder type definition
- [120’] Creating a workflow model
- [10’] Deploying the application to the IM services
- [25’] Working with the IM APIs
- [20’] Building the application
- [50’] Testing your application
- [00’] Bonus exercise: Using the otcloud Command Line Interface

This tutorial is a complete “from scratch” application building guide. However, if you don't want to build the application but still want to familiarize yourself with the IM API consuming example code, the different models, and the Contract Approval sample application, you can directly open the finished application in VS Code without the need to completely go through all the exercises. The project sources for the completed Contract Approval application are available to be downloaded to your computer. To directly run the completed project from VS Code, you can skip ahead to the Testing your application section.

2 Prerequisites

As this is a complete end-to-end application building guide, we will be going through the downloading and installation of the required software and source code as part of the different tutorial steps. The only prerequisite to be able to start with the first exercise is that you have signed up for the OpenText Developer trial account, and that this trial has not yet expired. If you do not yet have your own trial account set up, please follow the instructions in this chapter.

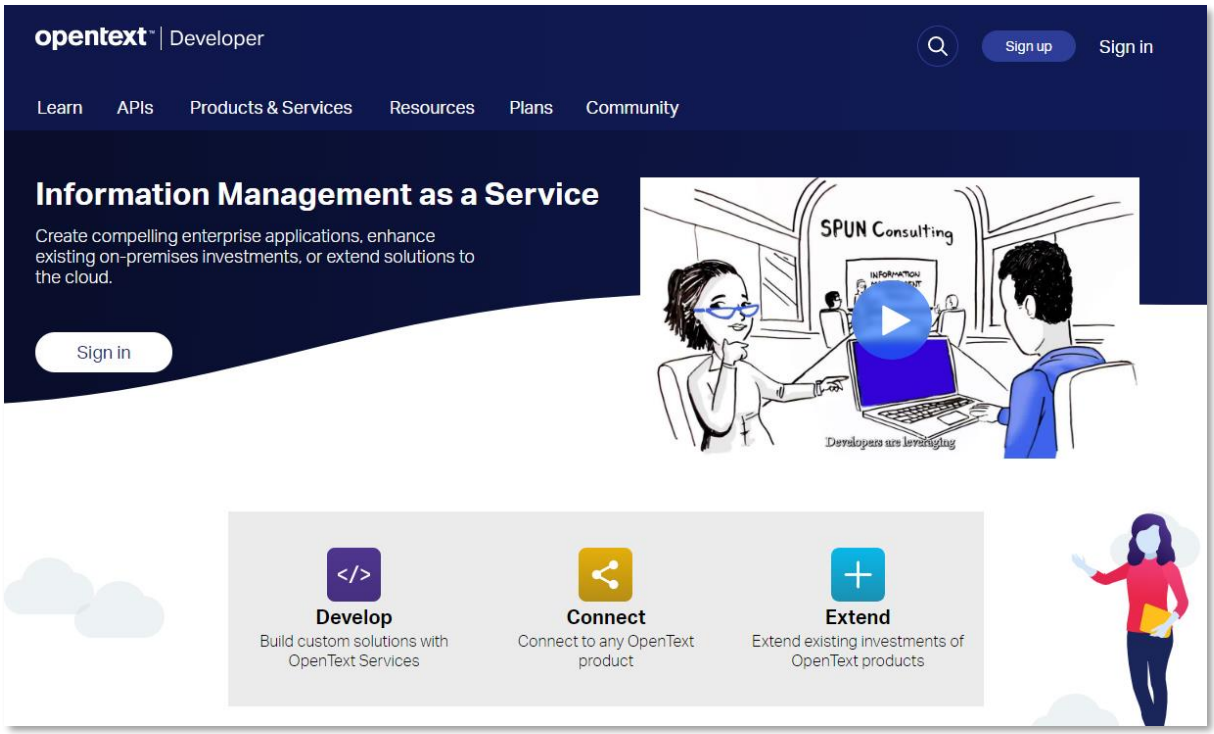
Also note that although you could theoretically use another OpenText Developer subscription than the aforementioned trial account, we recommend you still set up and use a trial account, as this tutorial has been explicitly constructed in that context. I.e.: differences between the trial account and another subscription might result in you getting stuck, or at least needing to deviate from the exact steps laid out in the tutorial.

2.1 [20'] Setting up an OpenText Developer Trial Account

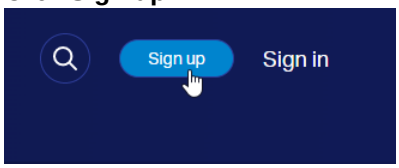
As previously stated, this tutorial requires using a trial account to avoid running into issues related to differences between subscriptions.

To set up an OpenText Developer free trial Account, proceed as follows:

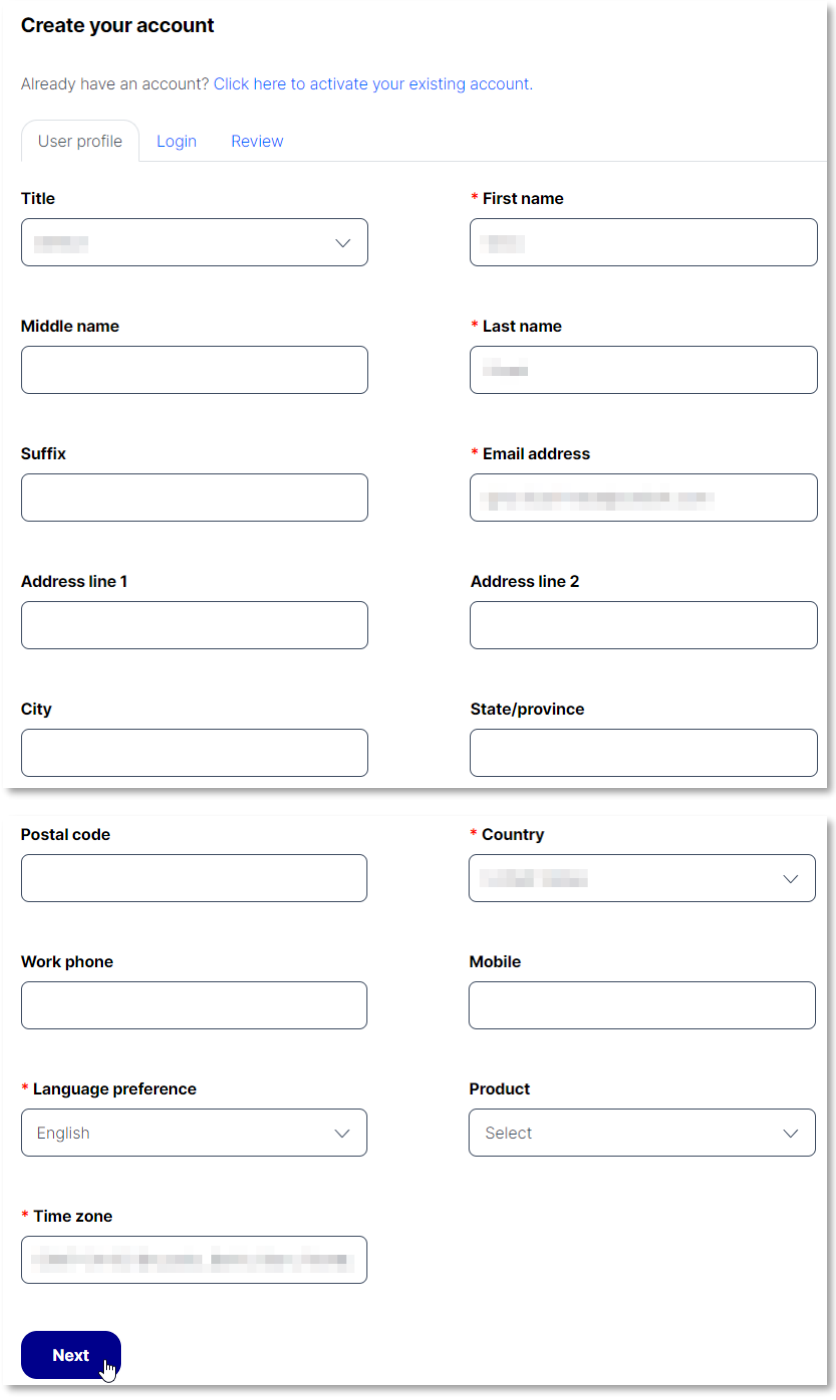
- Navigate to https://developer.opentext.com.



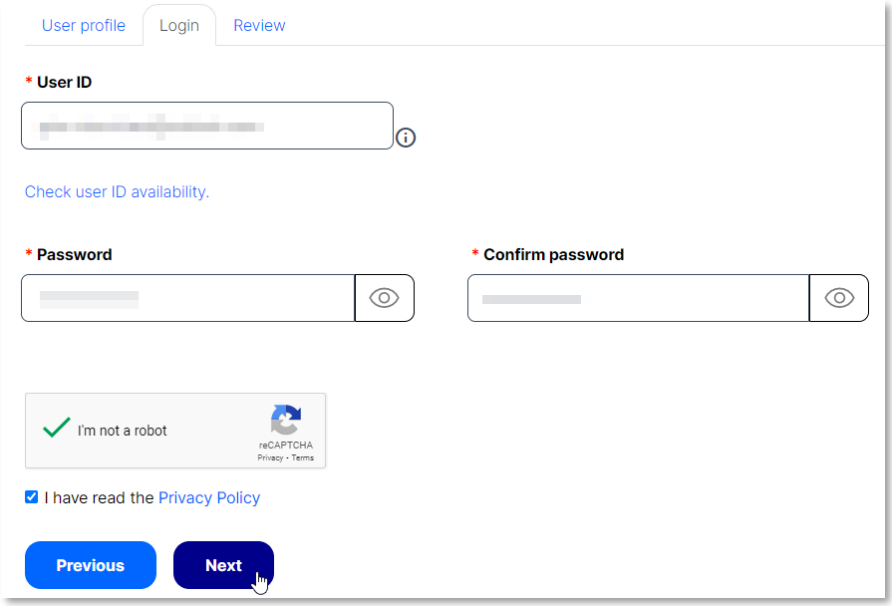
- If you already have an OpenText Connect account, you can immediately skip to the next step to Sign in with your OpenText Connect account.
  However, if you do not yet have an OpenText connect account, follow the following steps to sign up for one.
  Click **Sign up**.



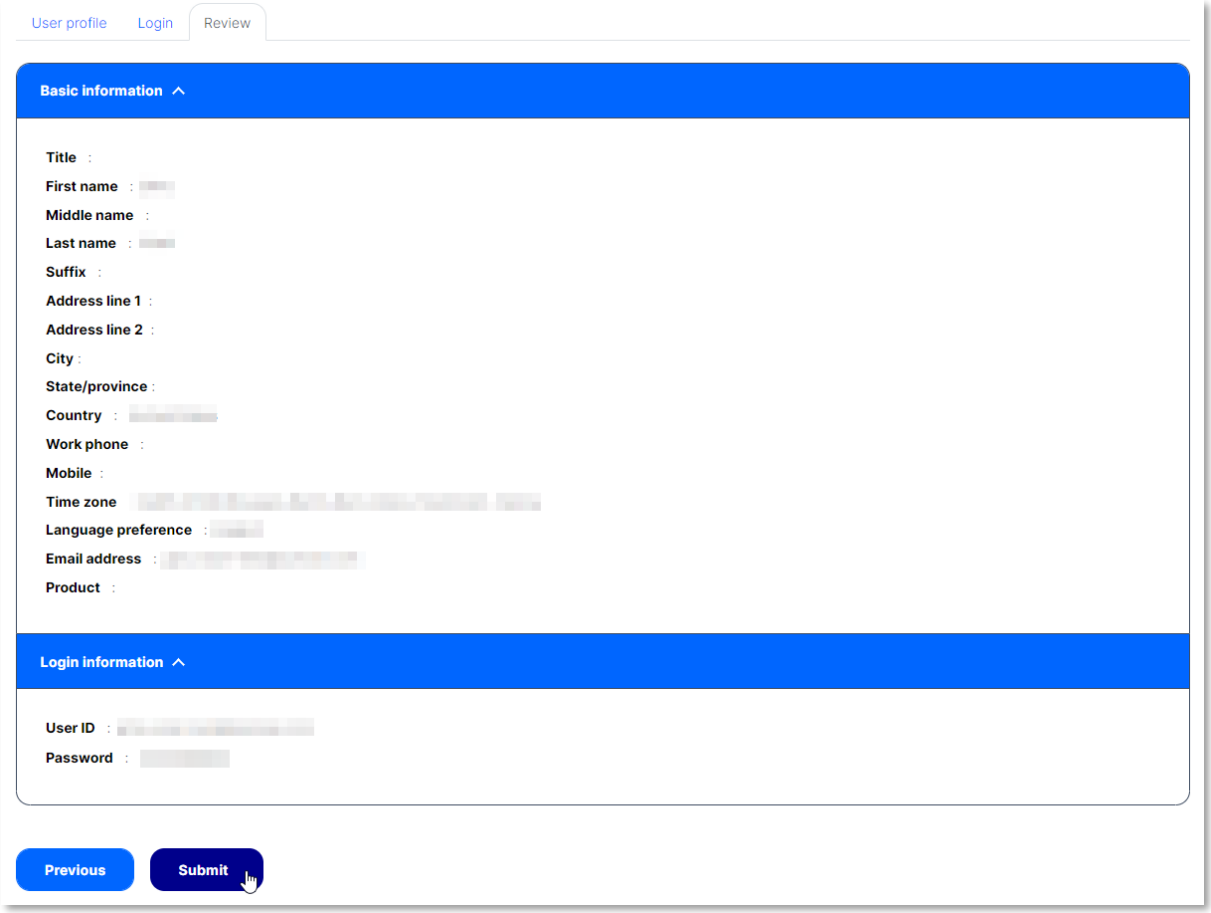
  Fill your personal information and click **Next** to continue.

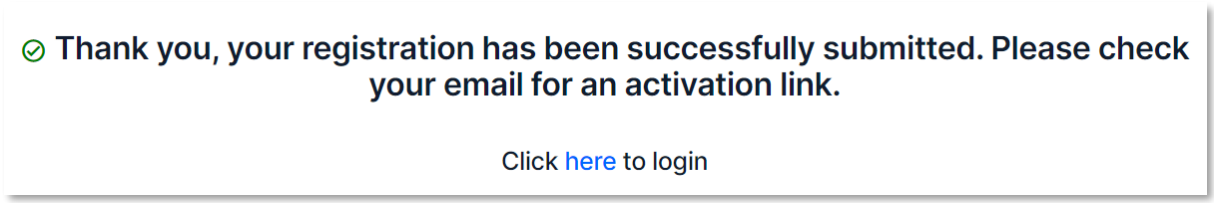


Use your email address for the User ID (**IMPORTANT: this has to be you email address**), choose a password, confirm you are “not a robot”, and confirm you have read the Privacy Policy. Click **Next** to continue with the OpenText Connect account creation.

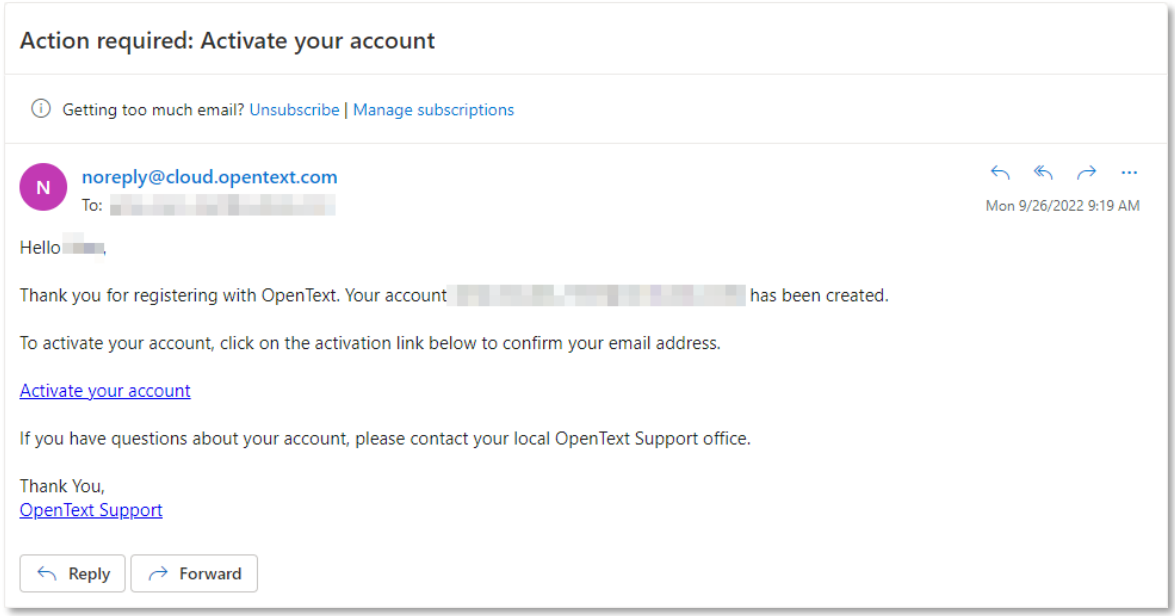


Review your account creation information and click **Submit** to create your new OpenText Connect account.

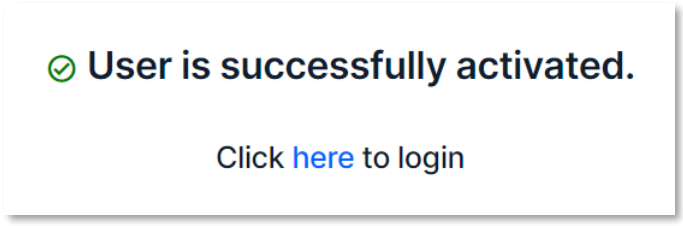




Open your email client and click **Activate your account** from the “Action required: Activate your account” email you have received (make sure it did not get blocked or moved to your spam folder).

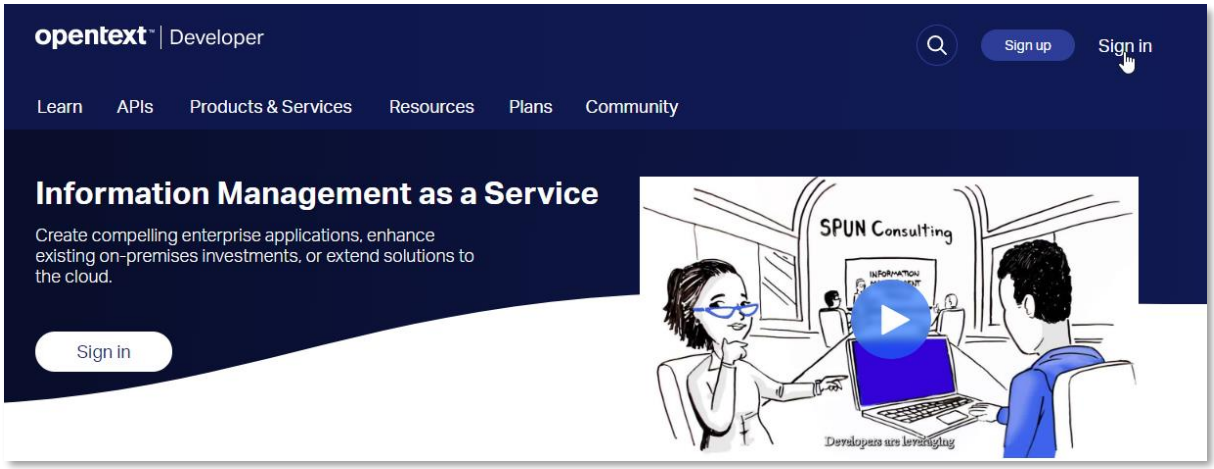


The account activation link should take you to a confirmation page from which you can log in to the newly created OpenText Connect account.

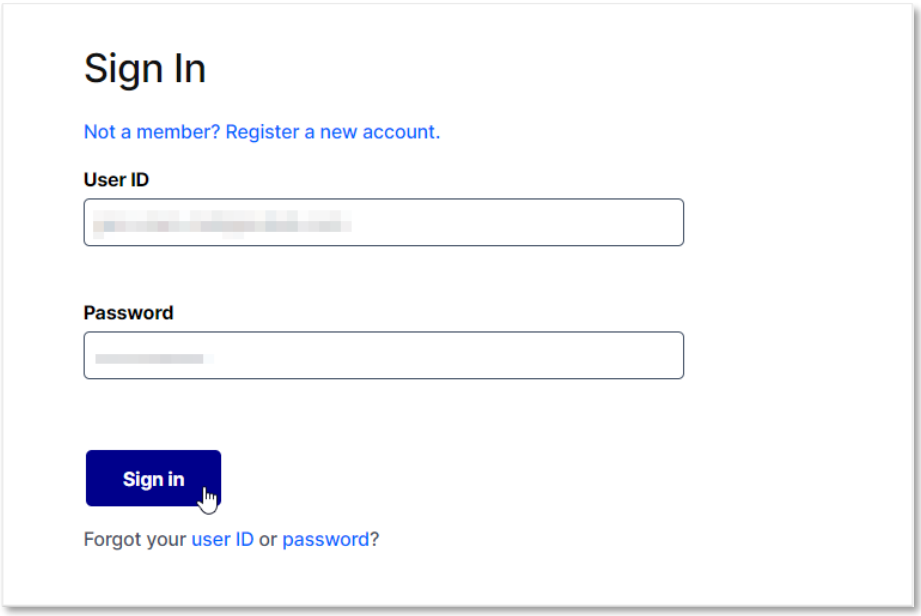


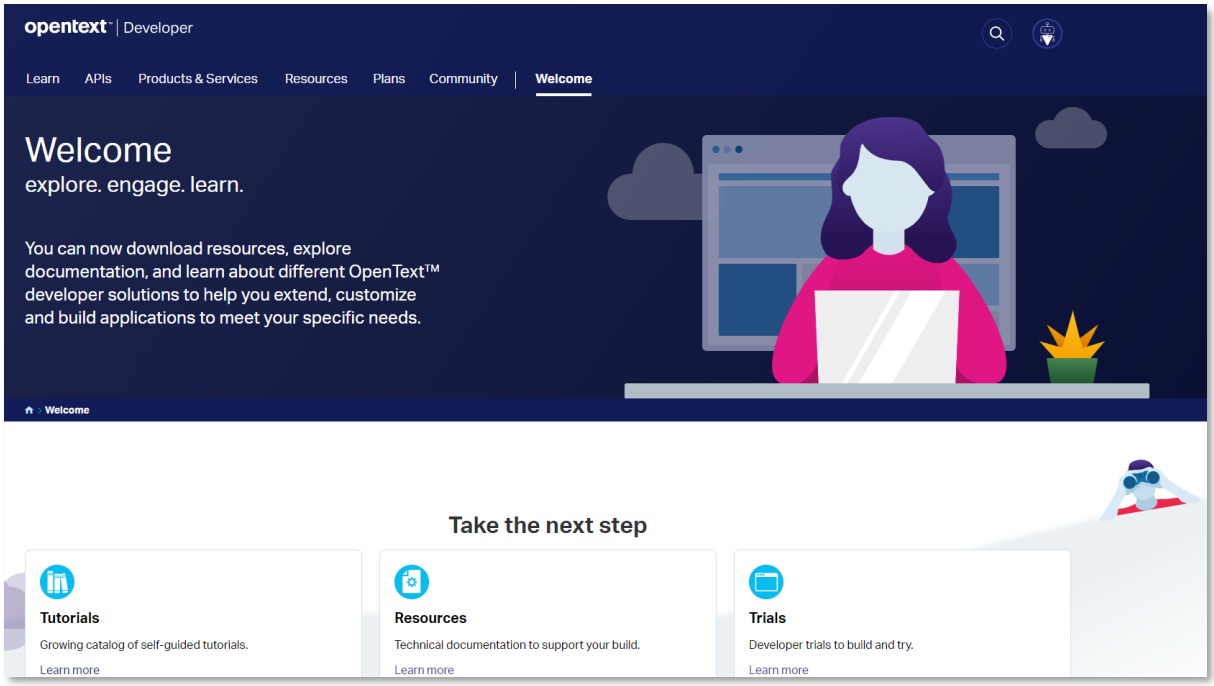
No need to log in from here, just close (all open instances of) your web browser to ensure a “fresh browser context” and navigate to https://developer.opentext.com to sign in with your newly created OpenText Connect account.

- To sign up for a trial account you need to sign in to your OpenText Connect account first. Click **Sign in**.

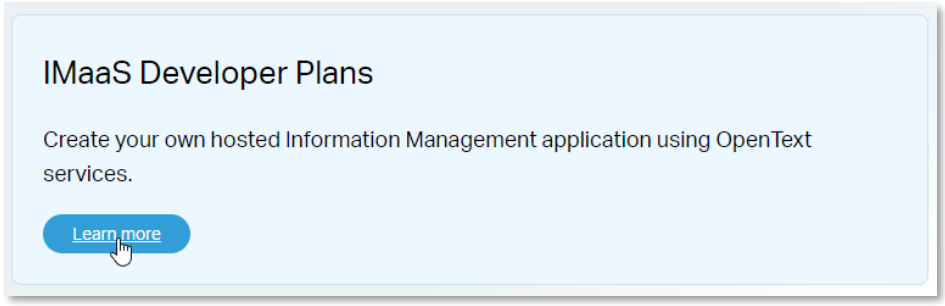


Fill your email and password and click **Sign in**.

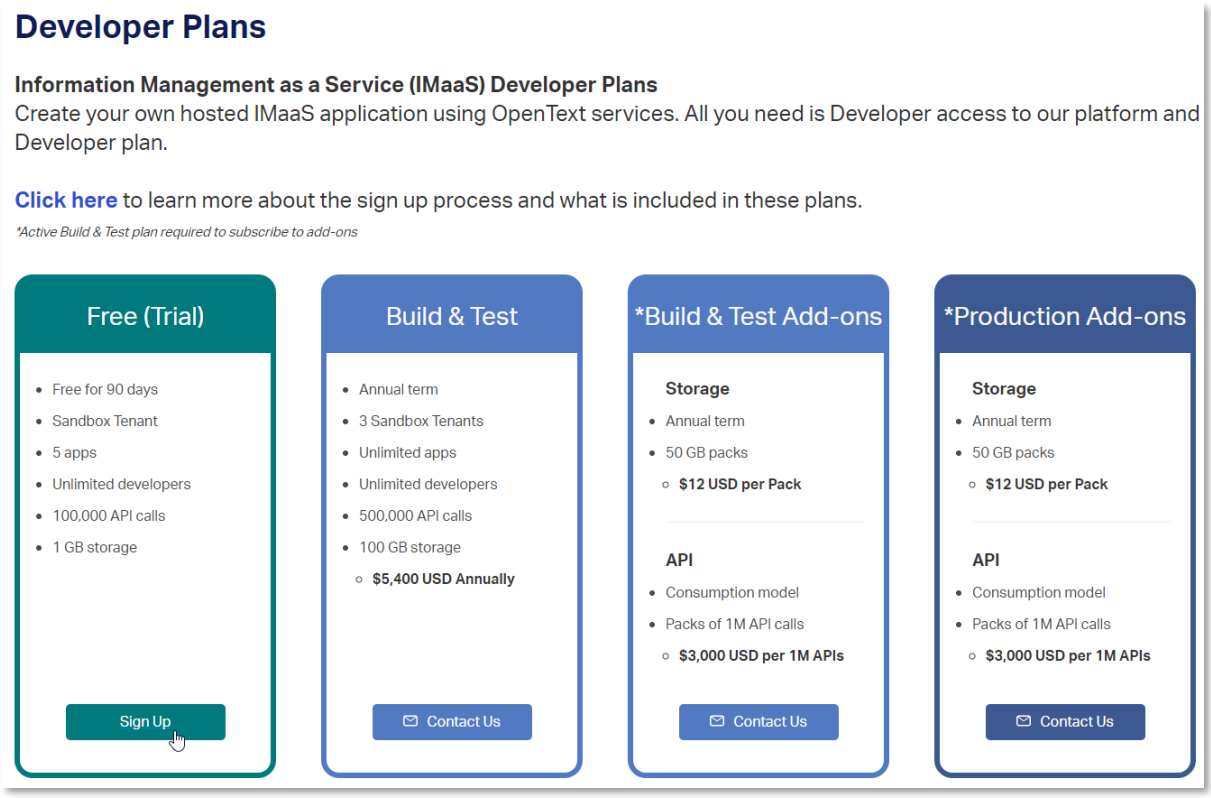




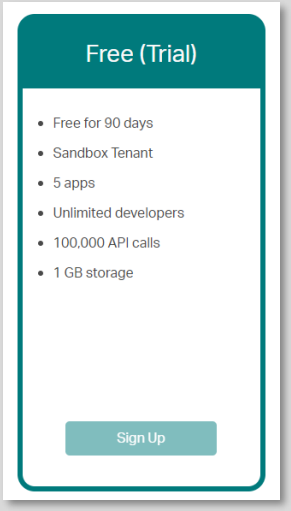
On the developer welcome page, scroll down to **Cloud Developer Plans** and click on **Learn more**.



Select to **Sign Up** for the **Free (Trial)** Cloud Developer Plan.



REMARK:
If you already had an account (did not just create one) and did previously sign up for the Trial plan with that account, the **Sign Up** button of the **Free (Trial)** will be disabled (you cannot click it).

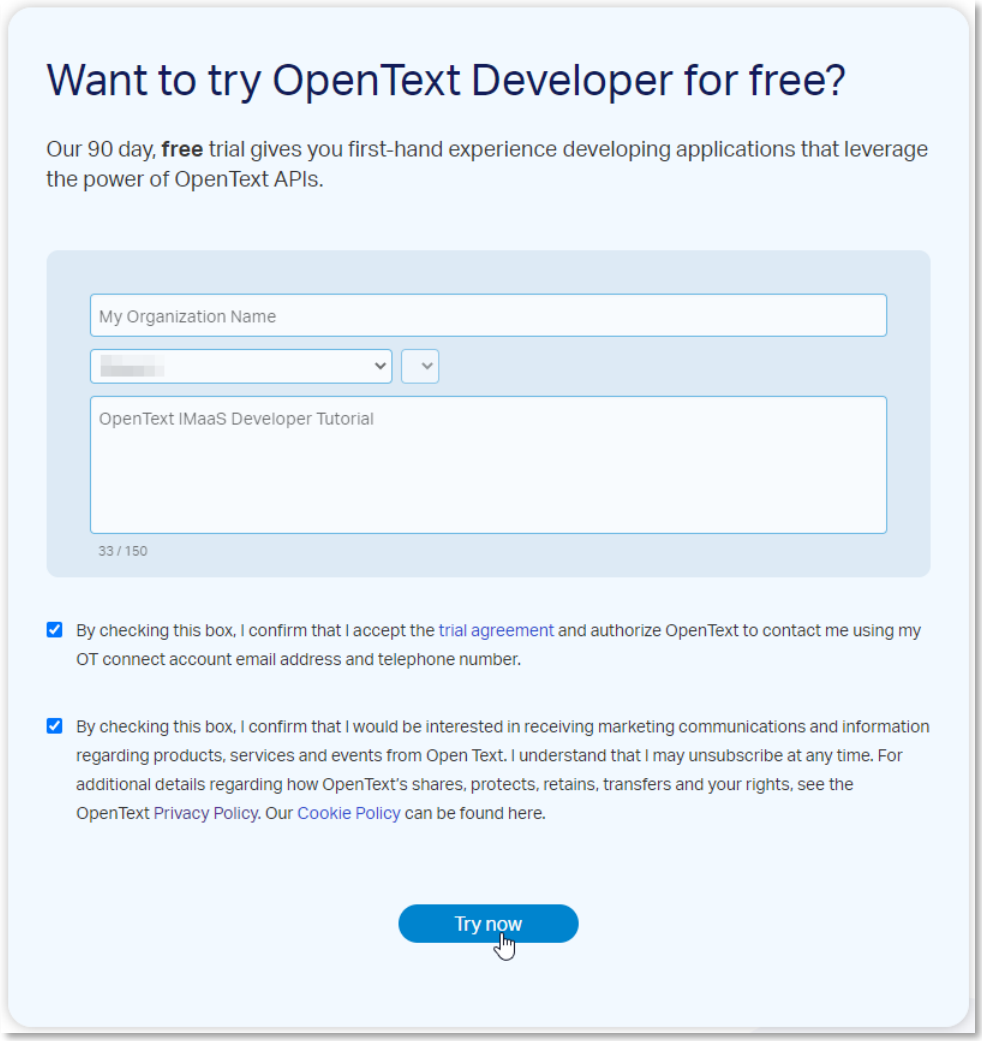


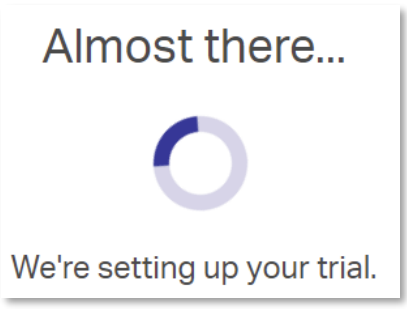
In this case, if your trial expired, you need to request an extension. If it did not expire (or if you got your extension), you are good to go and skip ahead to Building the Contract Approval application. Beware though, that during the tutorial you will require the organization connection details from the **organization config file** that was provided during the Trial sign up process. If you don't have this anymore, you can get the required organization connection information through the developer console's Organization information screen (**ID** value and **Manage** button provide the organization id and organization client information).

On the OpenText Developer trial registration form, fill the following information:

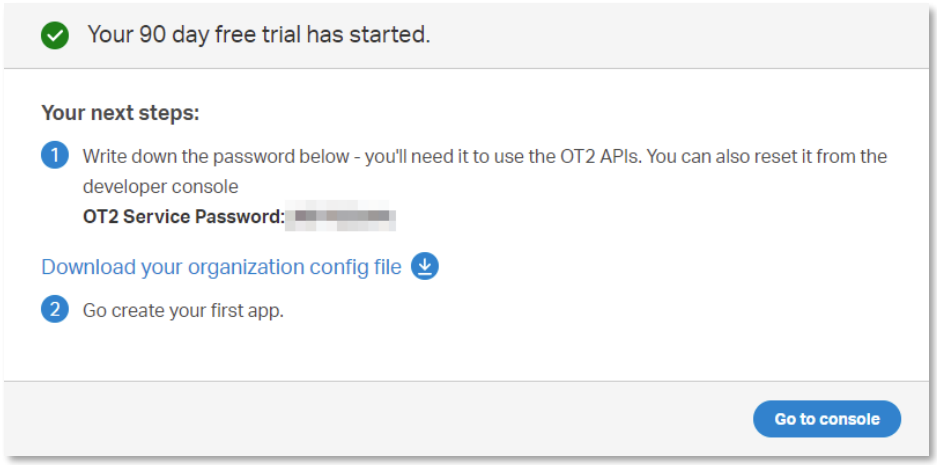
- **Organization:** the name of your organization; this will be used in the developer environment as the actual name of your developer organization, so make sure to choose it correctly
- **Country**: your country
- **State/province**: depending on the selected country, you might have to fill your state or province as well
- **Purpose**: the purpose for which you are registering for the OpenText Developer free trial account
- You'll need to check both the **trial agreement** and **marketing communications and information regarding products, services and events** check boxes

Once the registration form is correctly filled, click **Try now** to start the 90-day trial.



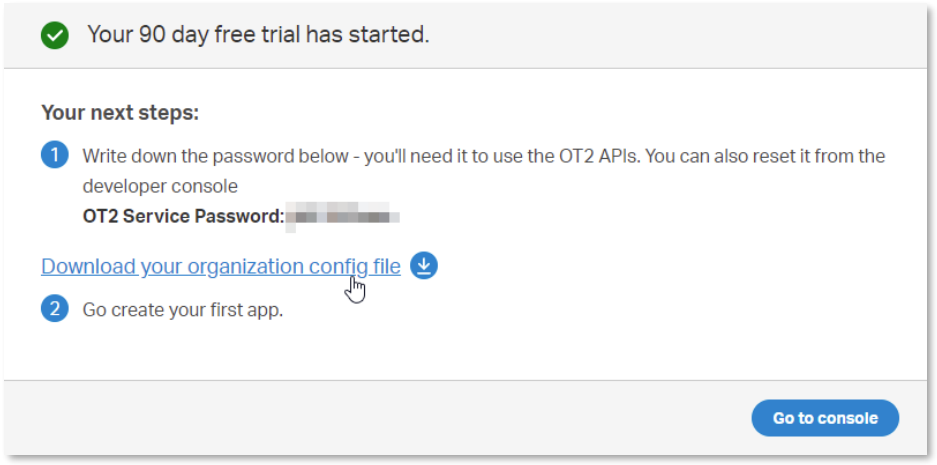


You are now presented with the confirmation that your 90-day free trial has started.

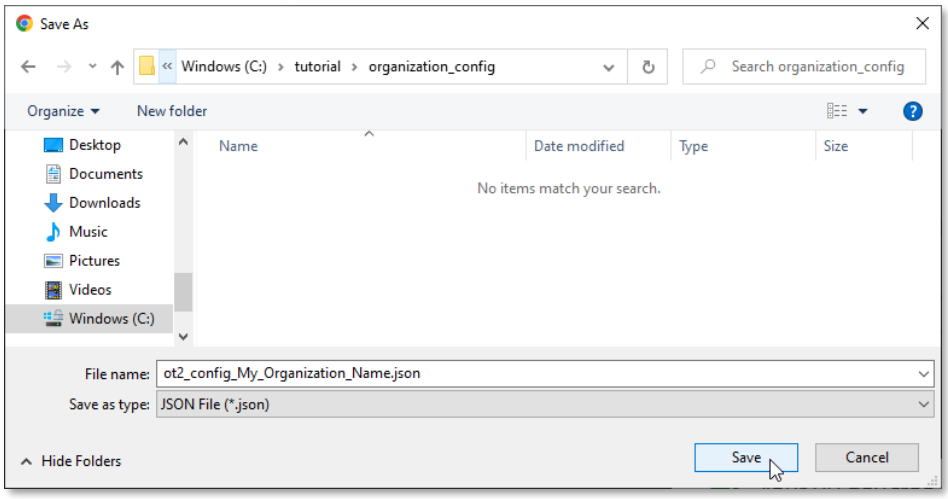


IMPORTANT:
The 90-day trial confirmation screen contains important information, so make sure not to skip the next steps (under **Your next steps**) of copying the **OT2 Service Password** and downloading the **organization config file**.
You will need the organization configuration details later on in this tutorial (and won't be able to retrieve them beyond this point). The OT2 Service Password is not required in context of this tutorial, but if you ever need it when interacting with IM APIs at the organization level, you will also not be able to retrieve it beyond this point (reset pwd is possible though).

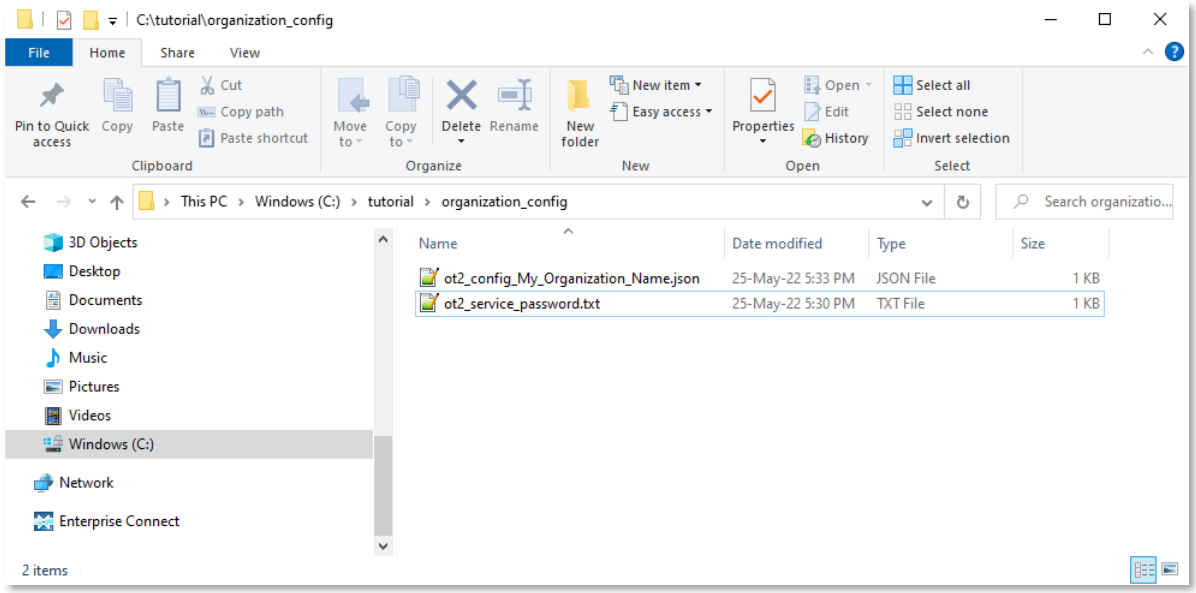
As per the above important remark, make sure you click the **Download your organization config file** link.



Create a folder (e.g.: **organization_config**) to store all organization related configuration for the Cloud Developer tutorial and save the **ot2_config_<organization name, we are using My Organization Name>.json** organization configuration file in that newly created folder.

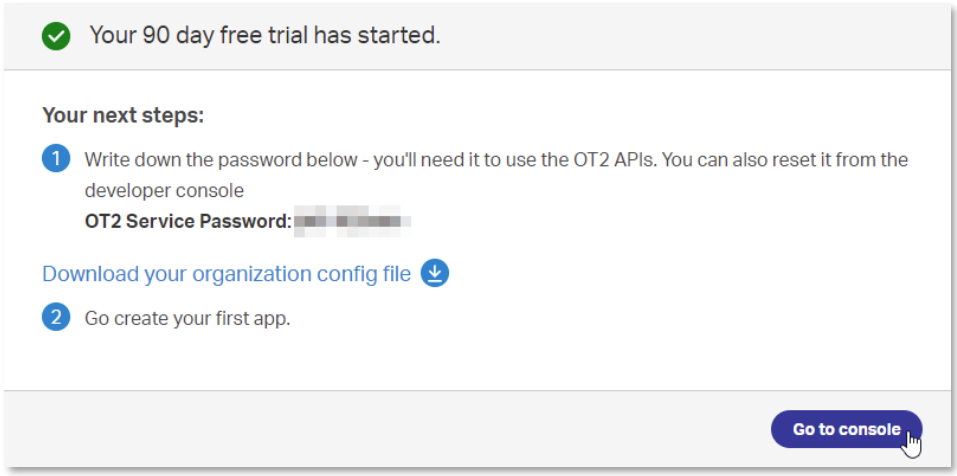


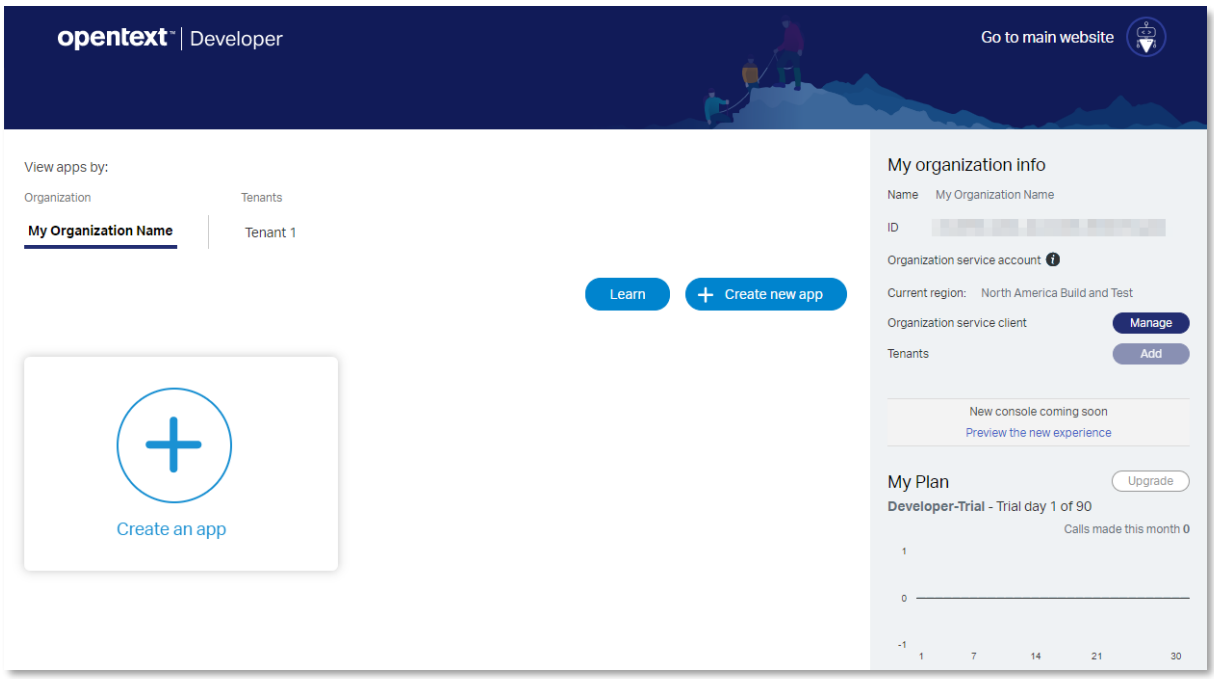
It is also very important you copy the **OT2 Service Password** from the trial confirmation screen and keep it for later use (even if we don't use it in this tutorial). We suggest you create an **ot2_service_password.txt** file next to the previously saved organization configuration file (inside of the new organization configuration folder) that contains the copied service password.



IMPORTANT:
Although we are using unencrypted/insecure text files to store the different key and password information for the purpose of this tutorial, it is of course recommended for real life scenarios to store any API key or password information in a secure way.

Once you have saved the OT2 Service Password and organization configuration file, click **Go to console** to go to the Cloud Developer console.





3 Building the Contract Approval application

This is the chapter throughout which you will be setting up your OpenText Cloud development environment, and build, deploy and test the Contract Approval application.

It consists of 14 subsequent exercises that build on top of each other, so you cannot skip ahead, and it is very important you perform the exercises exactly as described.

That being said, if you are only interested in running the application and having a look at the finished project (incl. its models and code), you can choose to skip to Testing your application.

Of course, we recommend you go through all 14 exercises. In that case, you should start with the first exercise in this chapter (section 3.1) which details setting up your Cloud Developer IDE.

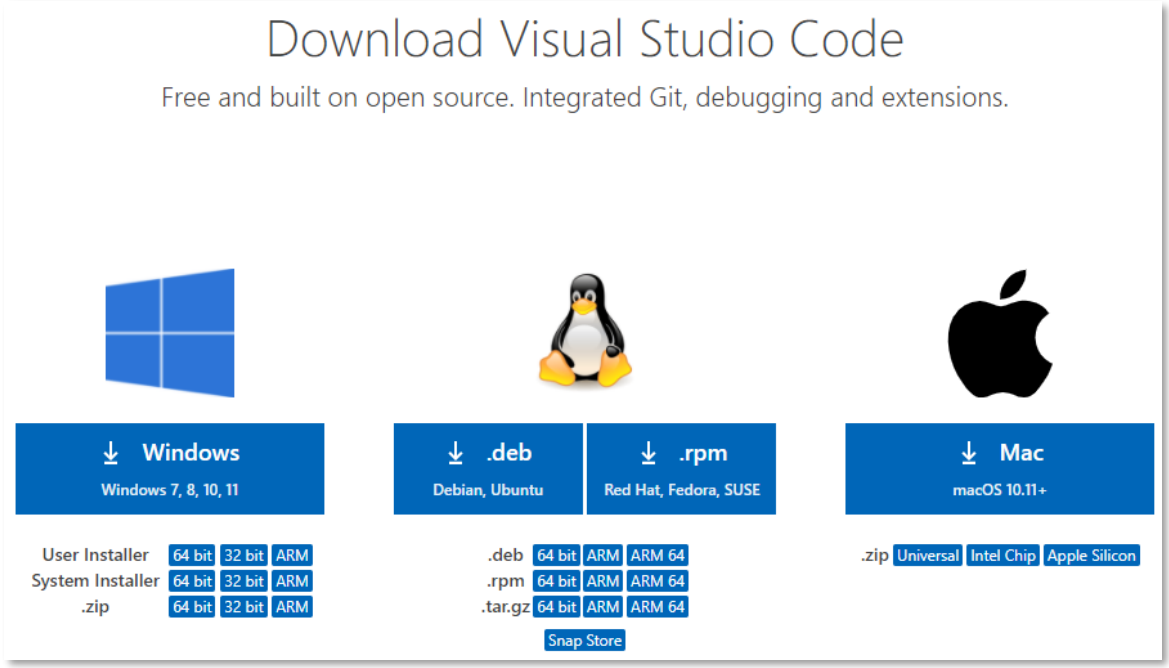
3.1 [25'] Setting up the Cloud Developer IDE

This exercise will guide you through the downloading and installing of Microsoft Visual Studio Code (VS Code) and the adding of the Cloud Developer Tools VS Code extension pack to your VS Code installation.

Once you are done with this section, you will have set up your Cloud Developer IDE, and you are ready to set up a connection to your Cloud Developer organization.

To set up your Cloud Developer IDE, proceed as follows:

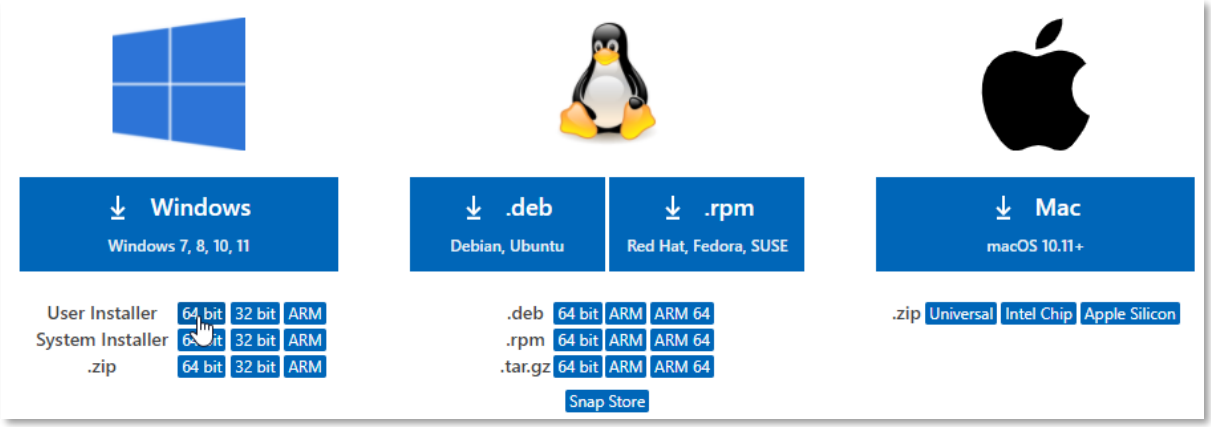
- Navigate to https://code.visualstudio.com/download to download the Microsoft Visual Studio Code distribution that matches your system.



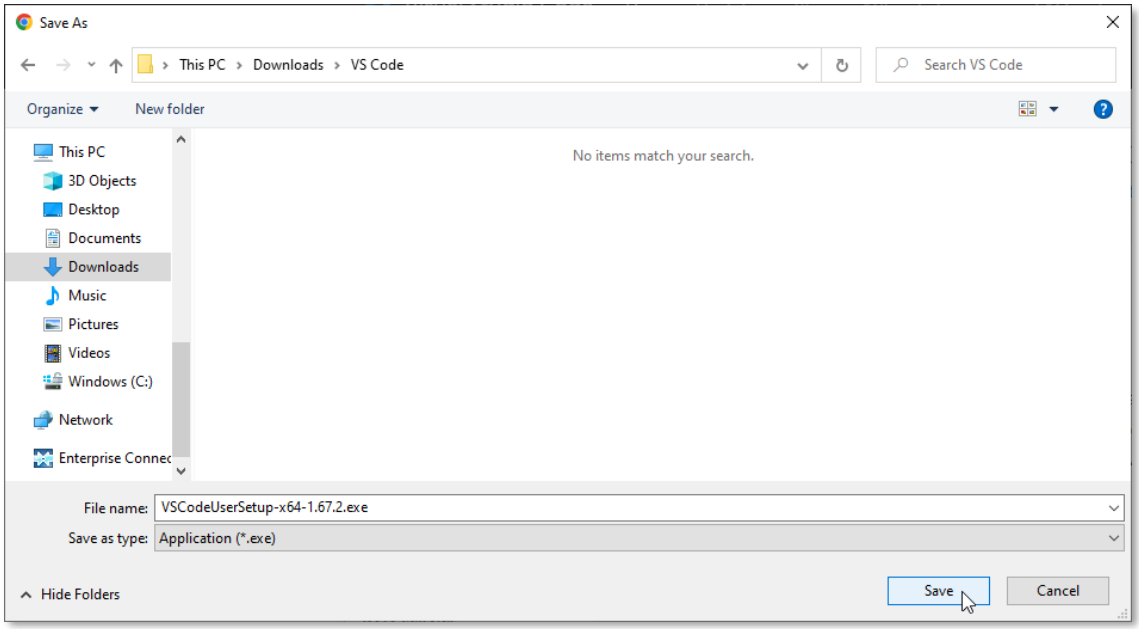
IMPORTANT:
Note that we have currently tested the OpenText Cloud Developer Tools for VS Code with Windows and macOS systems.
You can decide to use Linux, but if you run into problems on a Linux system, OpenText may not be able to provide you with a solution. Simply put, if you cannot perform the steps laid out in this tutorial when on a Linux system, you will have to switch to Windows or macOS.

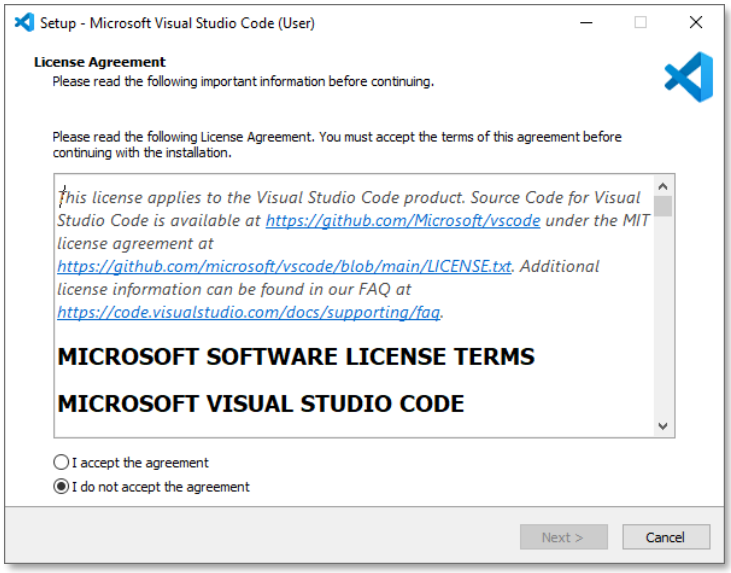
In this tutorial, we will only provide you with the steps to perform the setting up of the (User Installer based) VS Code version for Windows 10.
For other OS type systems or other versions of VS Code, you can follow the installation steps as outlined below as a guideline, but if the installation process is different, please refer to the VS Code documentation for further help.
Also note that we recommend you install the latest version of VS Code, but that throughout the tutorial we will be using the current latest version for Windows 10 (1.67.2 64-bit), so if you have a different version and/or system, the screen shots might not always exactly match.

- If you are installing VS Code on a 64-bit Windows 10 system, choose to download the **64 bit User Installer for Windows**.

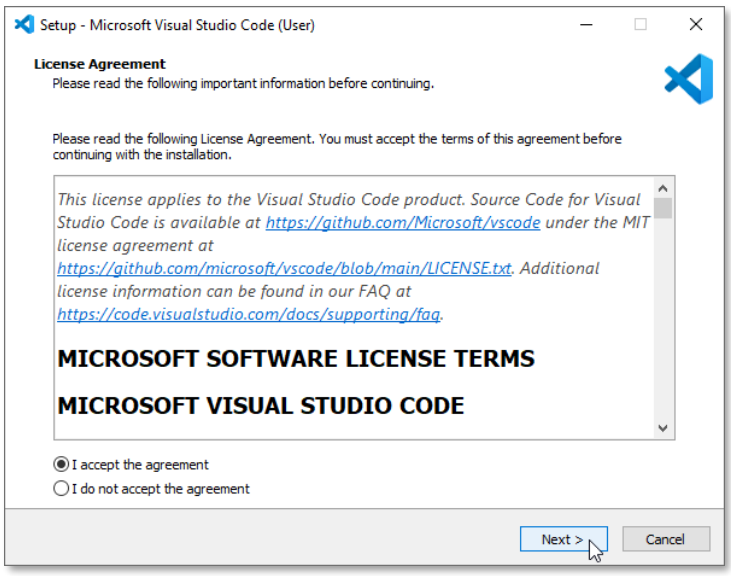


Save and run the installer.

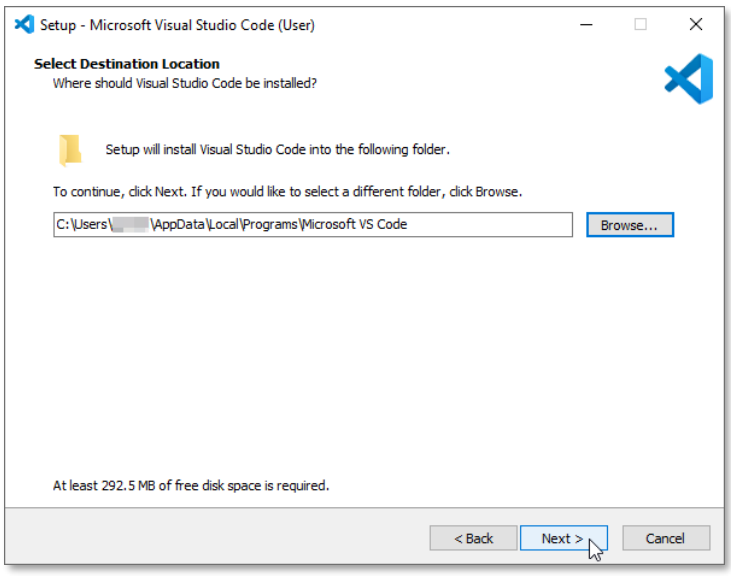




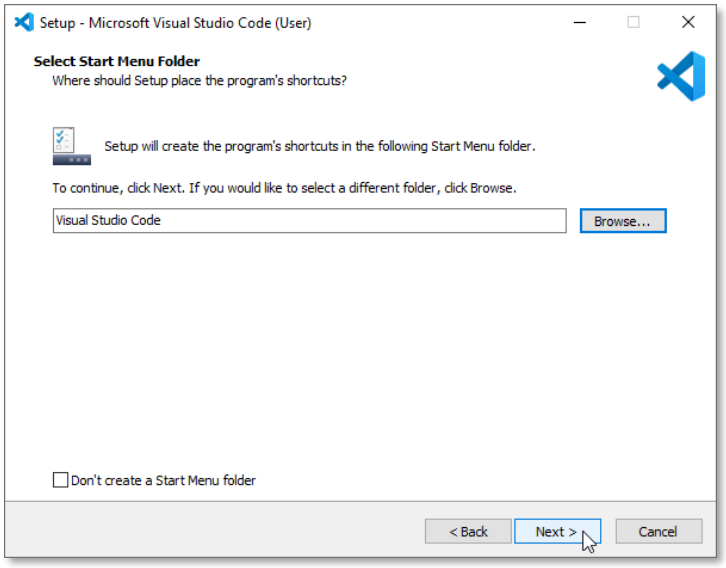
Select **I accept the agreement** and click **Next >** to continue.



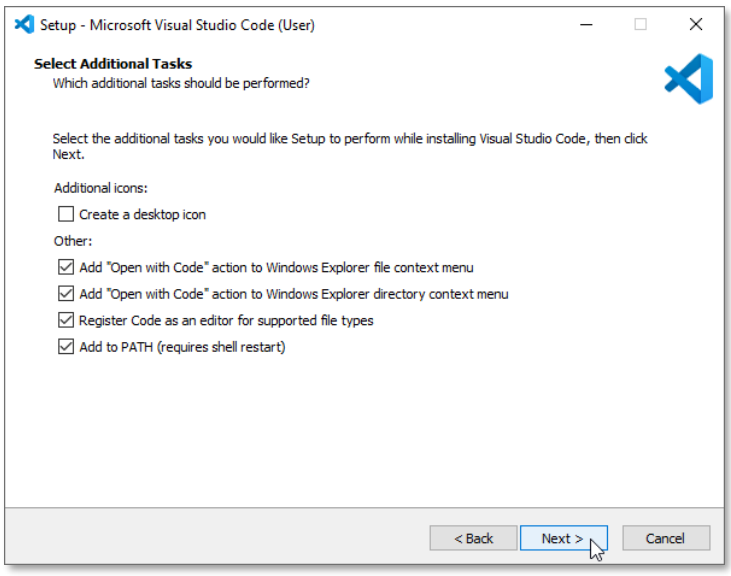
Select the installation destination location and click **Next >**. You can use the suggested default location.



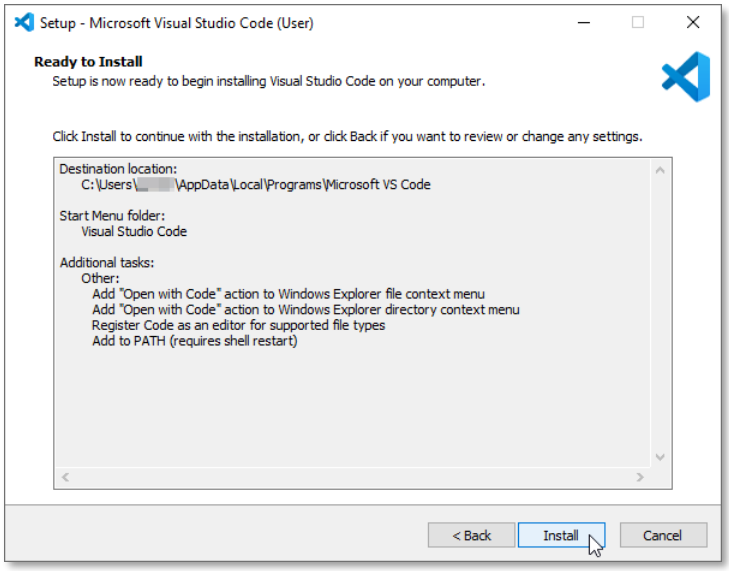
Keep the default setting for the selecting of the start menu folder and click **Next >**.



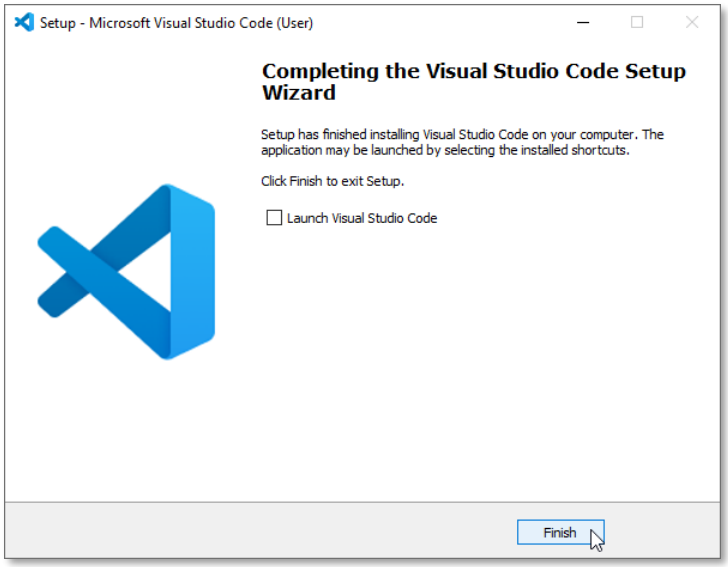
Select all additional tasks under **Other** and optionally select **Create a desktop icon**. Click **Next >** to continue.



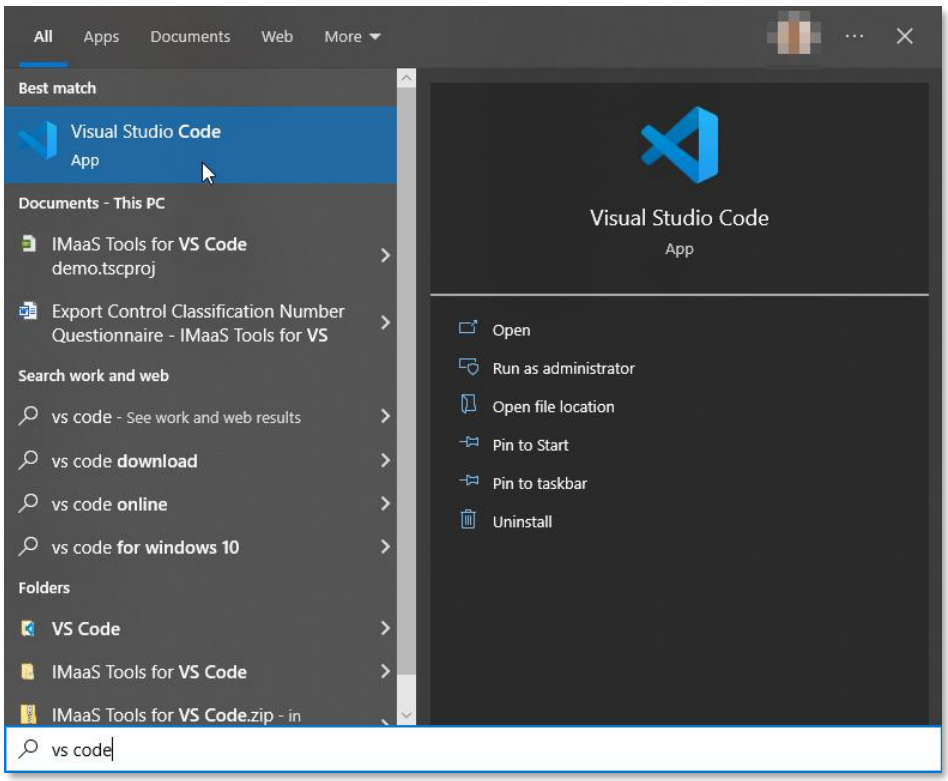
Verify all your choices and click **Install** to start the VS Code installation.



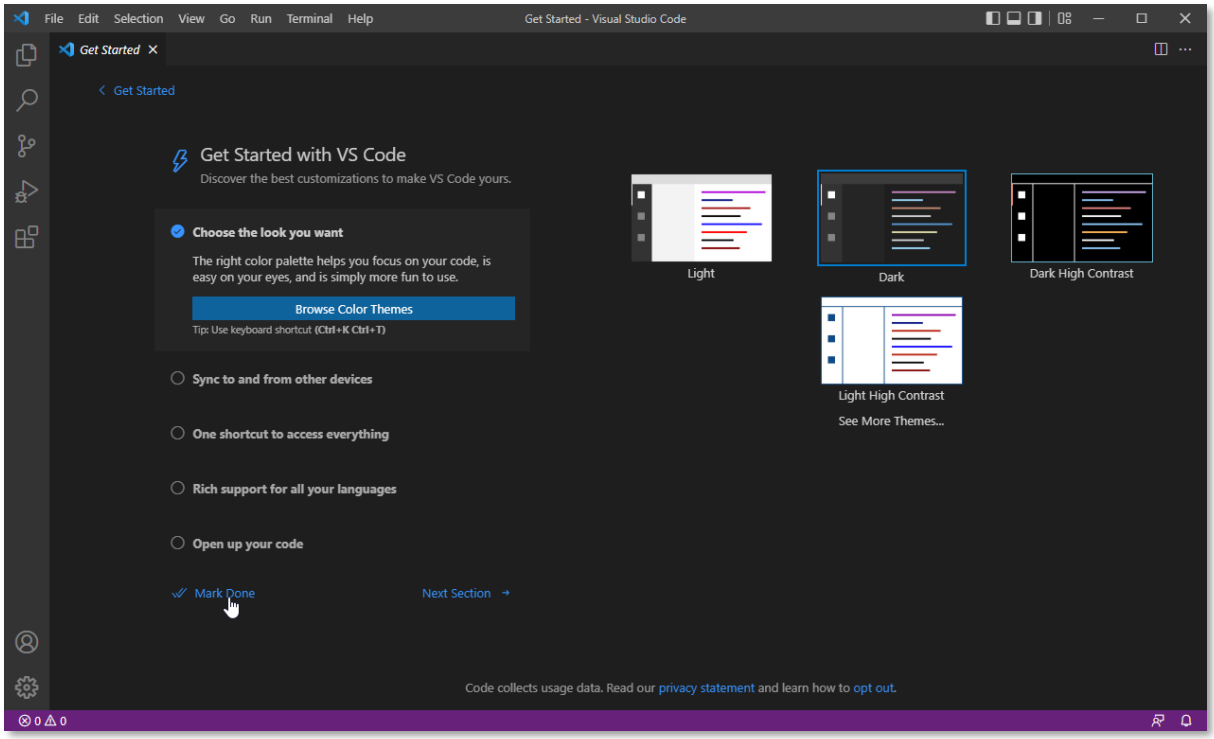
Once the installation is complete, you can click **Finish** to close the VS Code Setup Wizard (optionally leaving the **Launch Visual Studio Code** checkbox checked).



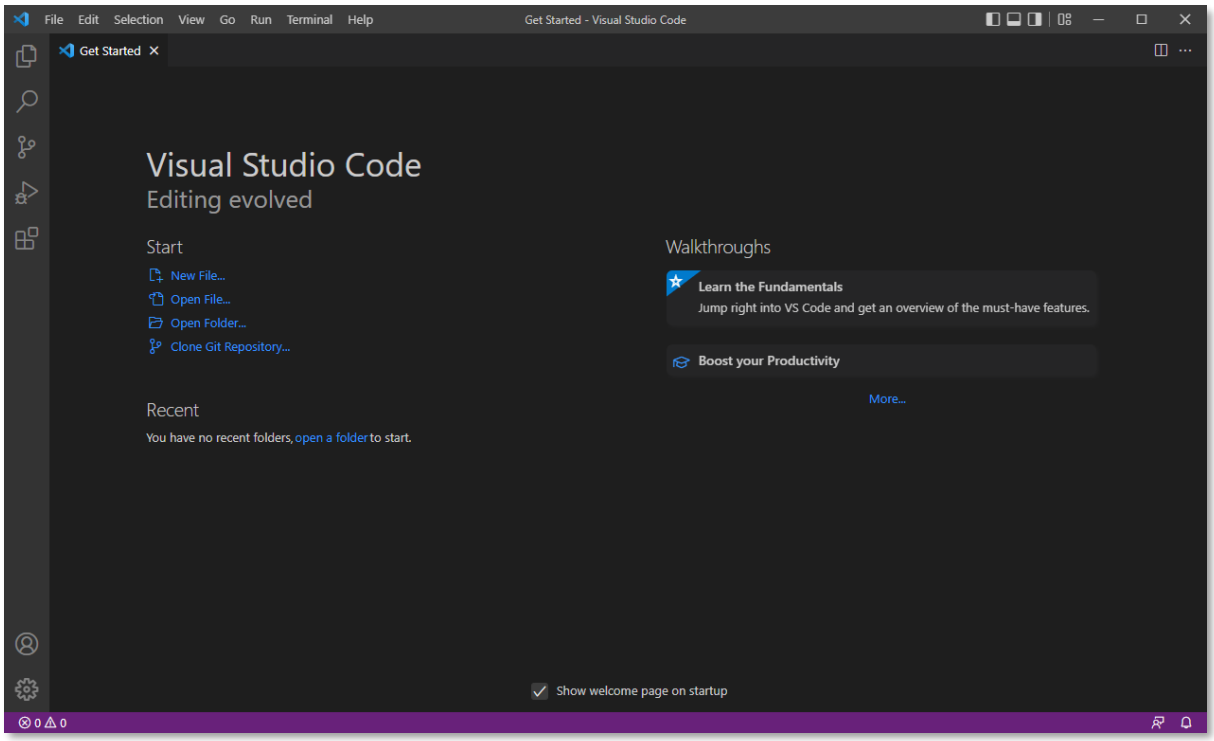
Once VS Code is installed, if not already open, you can open it by typing “vs code” in the Windows Start Menu search box and selecting the **Visual Studio Code** App.
Feel free to **Pin to taskbar** for later easy access from the taskbar.



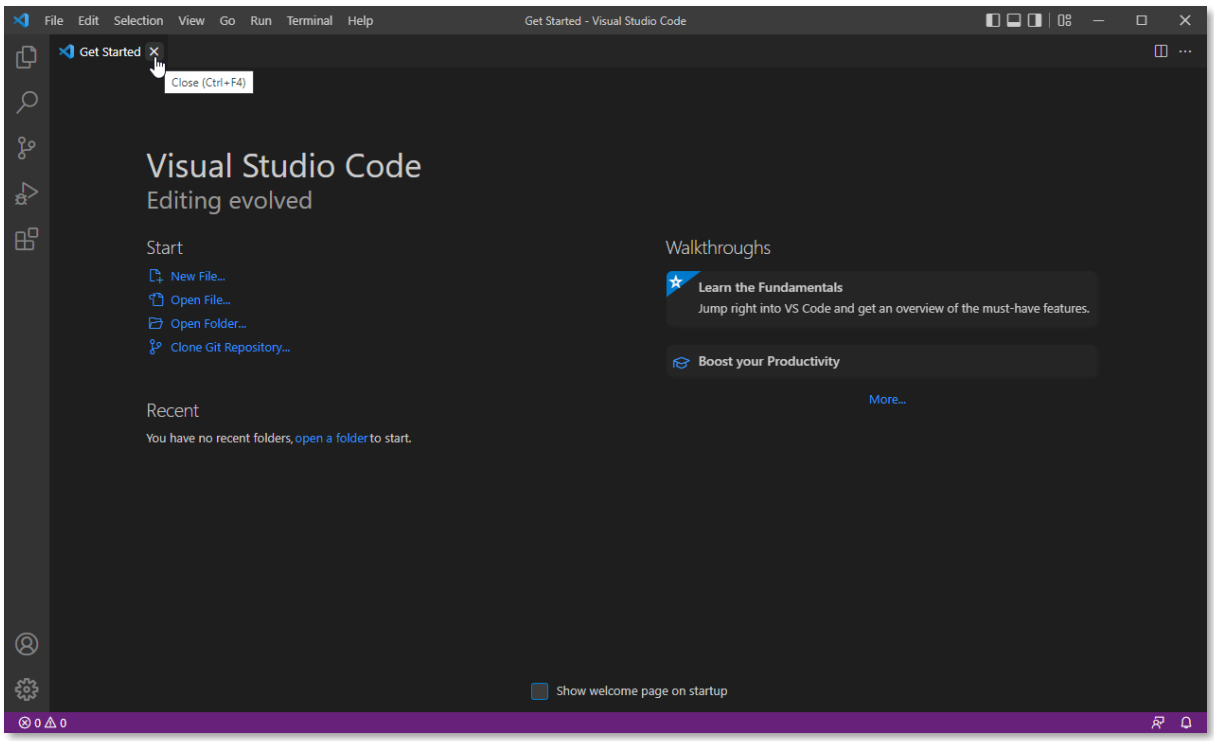
When opening VS Code for the first time, you are presented with the **Get Started with VS Code** wizard. Just perform the first activity of **Choose the look you want** by selecting the theme of your choosing. We are selecting the **Dark** theme.
Click **Mark Done** to confirm your choice.



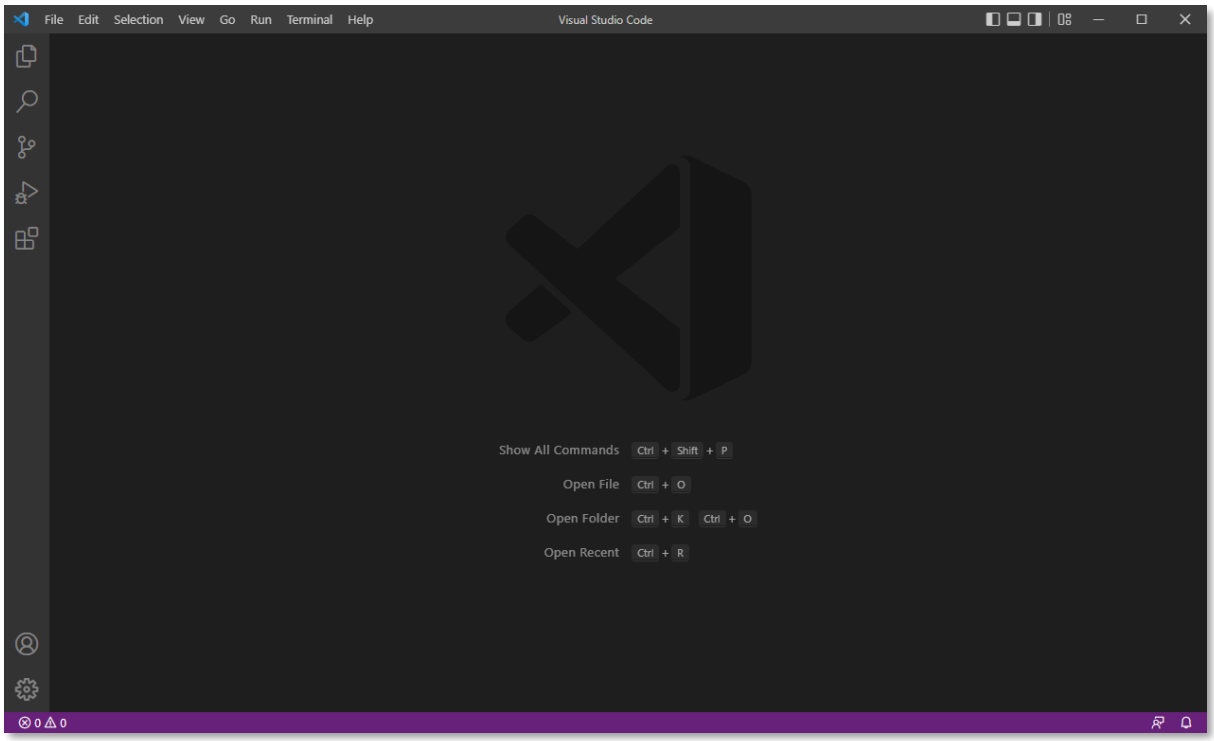
The standard VS Code **Get Started** welcome page now displays.



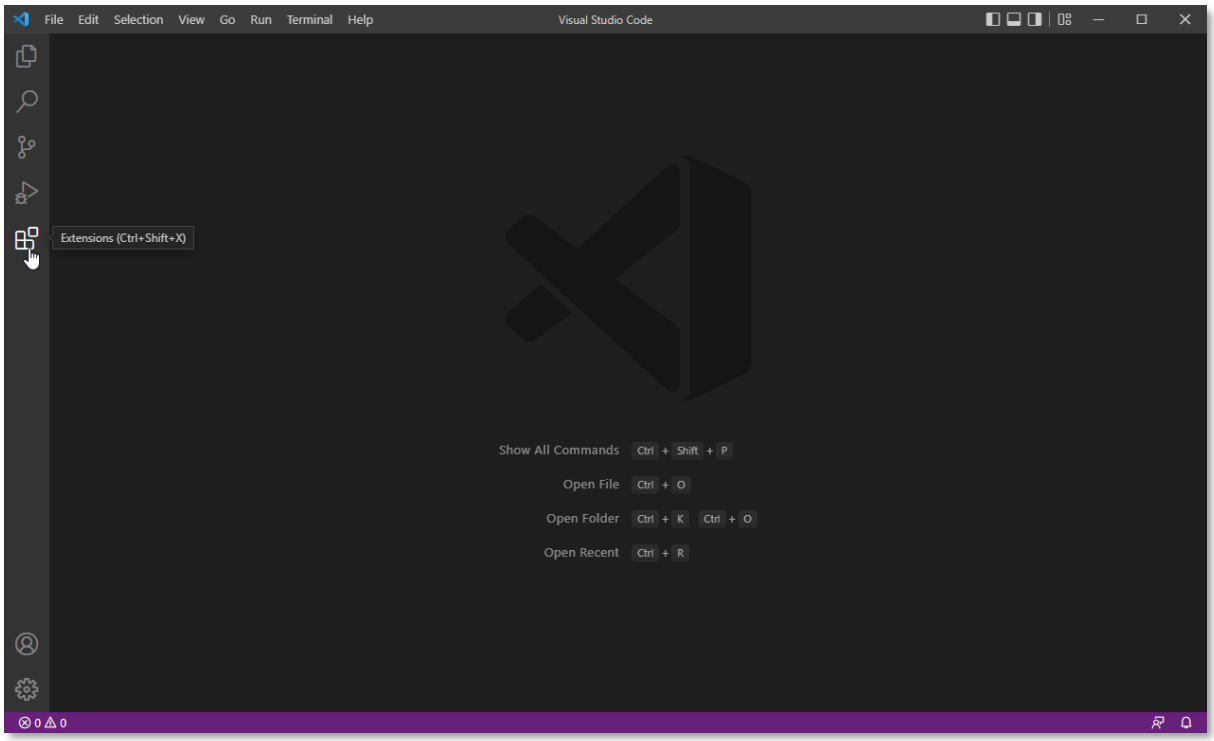
You can uncheck the **Show welcome page on startup** and close the **Get Started** page so that it will no longer display when you open VS Code.



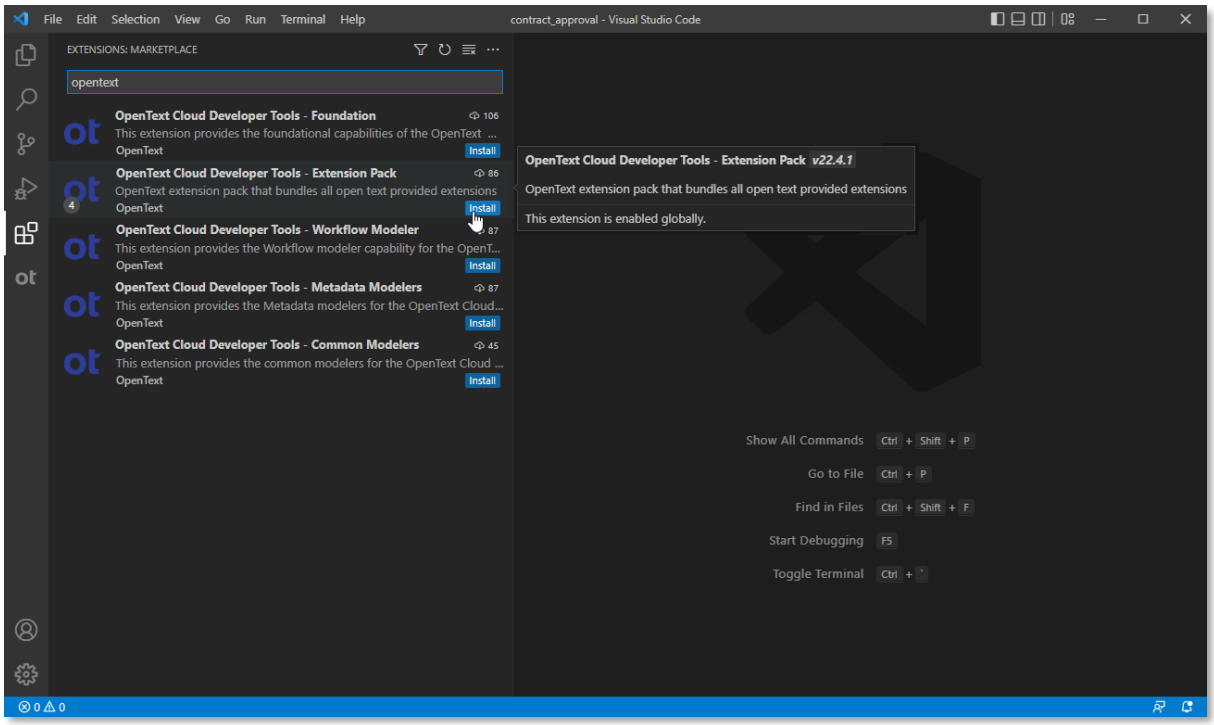
You are now ready to add the **OpenText Cloud Developer Tools for VS Code** to your VS Code IDE to enable the Cloud Developer capabilities directly in VS Code.

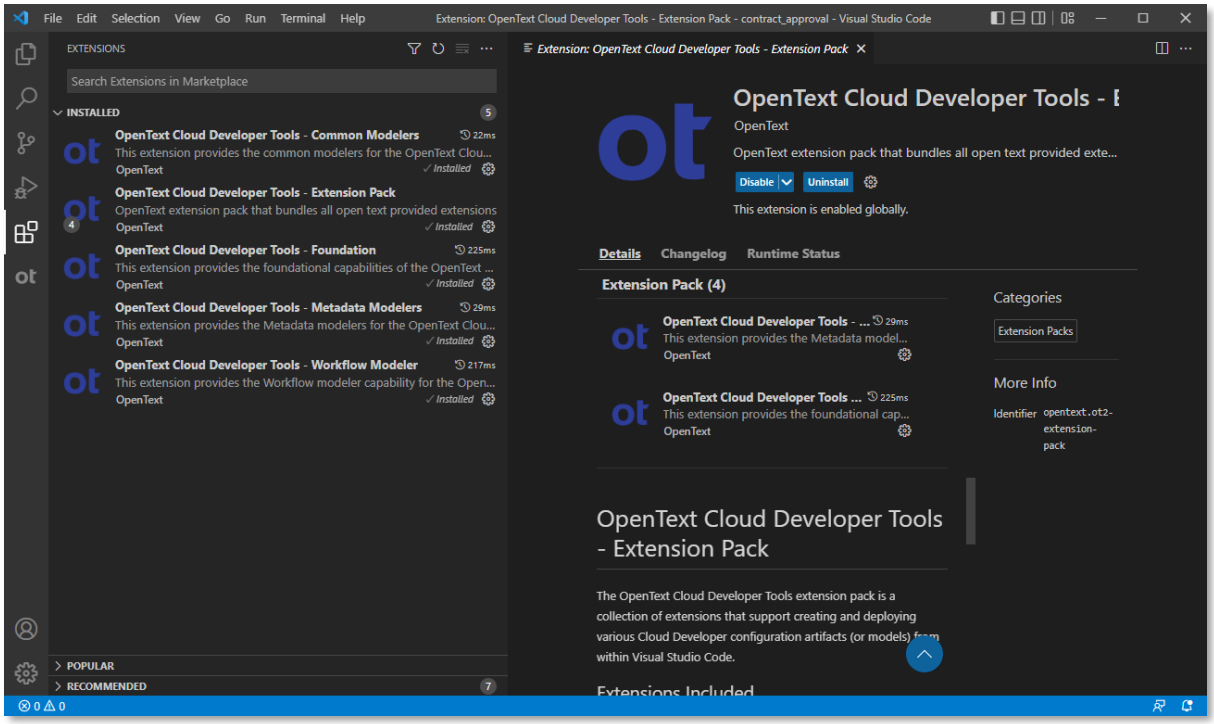


- The **OpenText Cloud Developer Tools** VS Code extension pack is available in the VS Code Marketplace, and you can install it directly from VS Code.
  The VS Code Activity Bar on the left side lets you quickly switch between different views. To access the VS Code Marketplace, you need to click  to switch to the **Extensions** view.



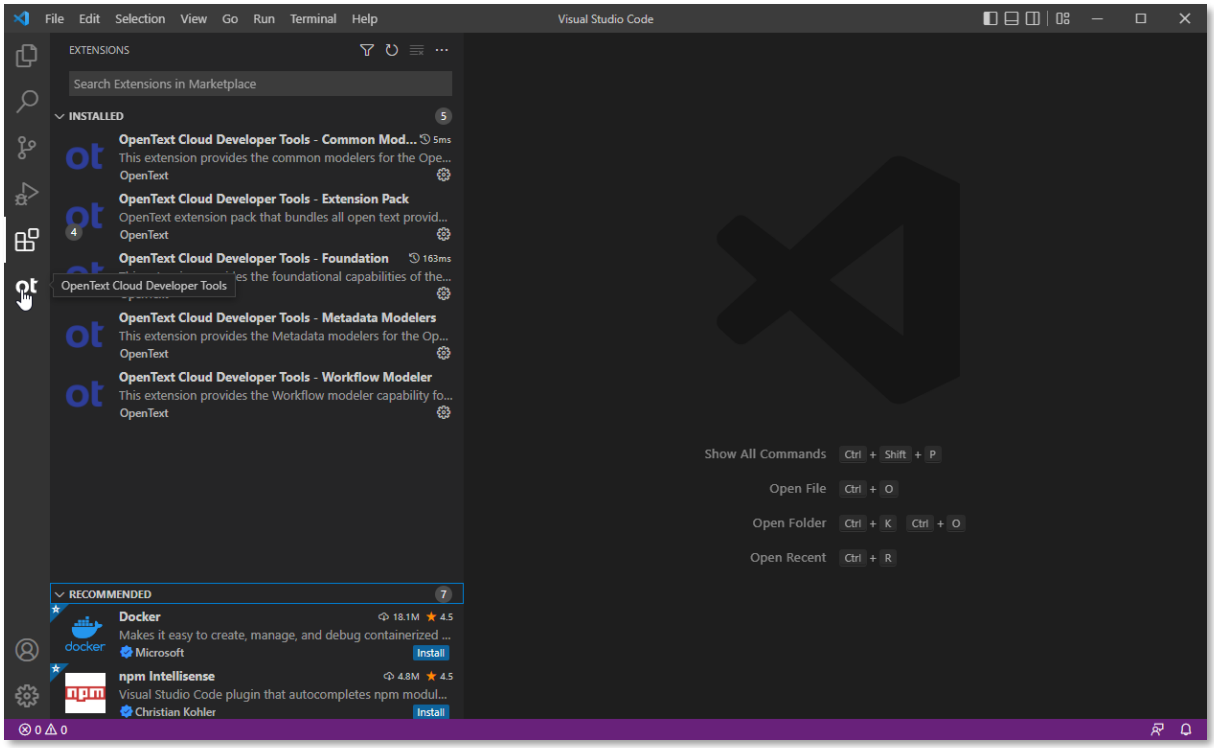
In the **Search Extensions in Marketplace** search bar, type **opentext** and choose to install the **OpenText Cloud Developer Tools - Extension Pack**.

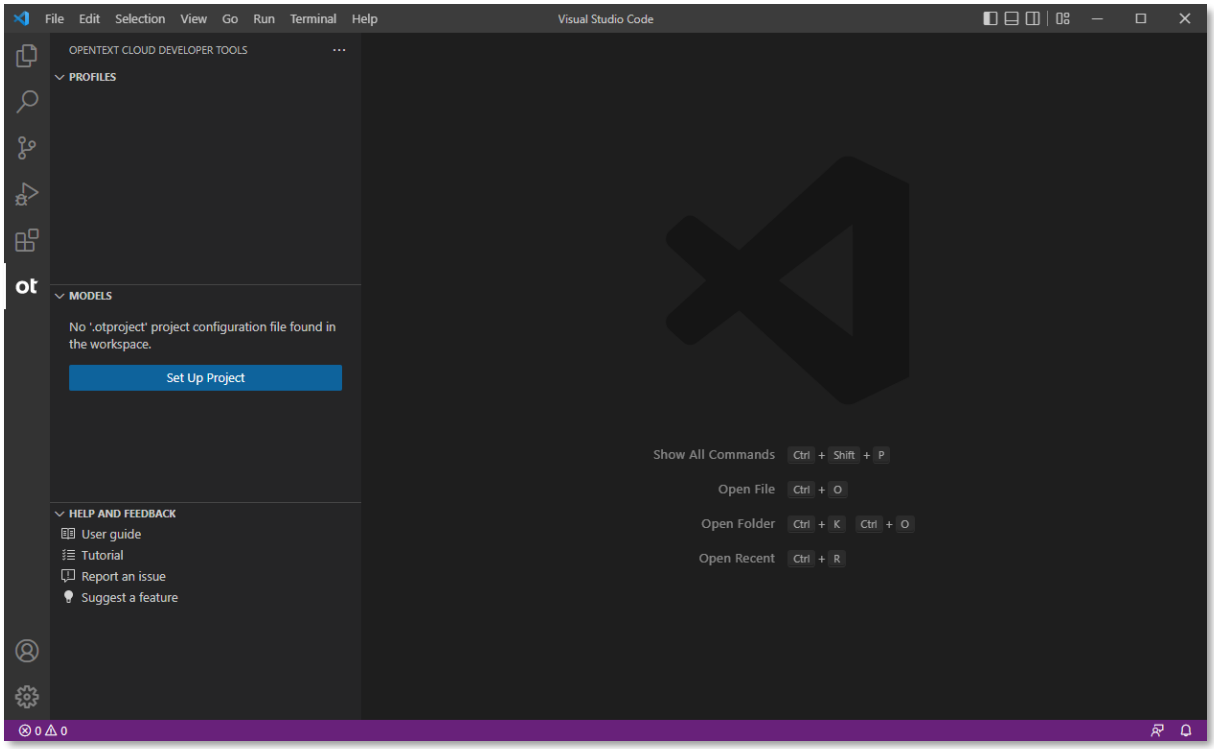




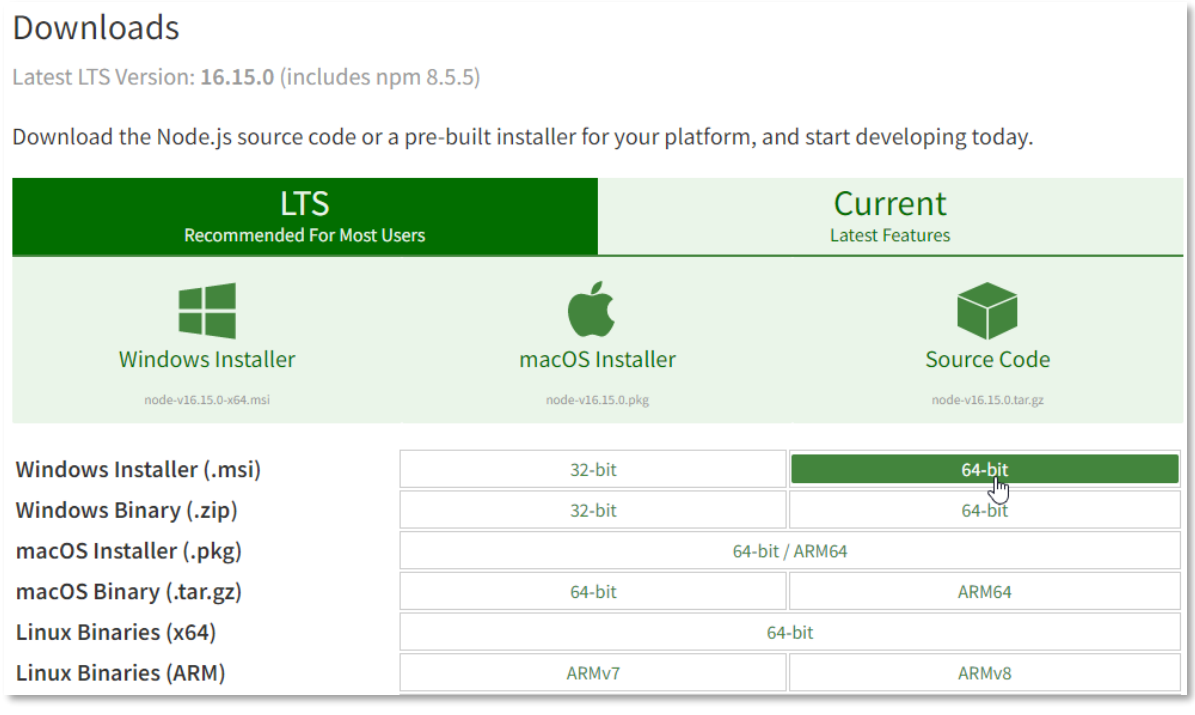
If asked, you can click **Reload** to ensure the VS Code extension pack you just installed gets properly enabled.

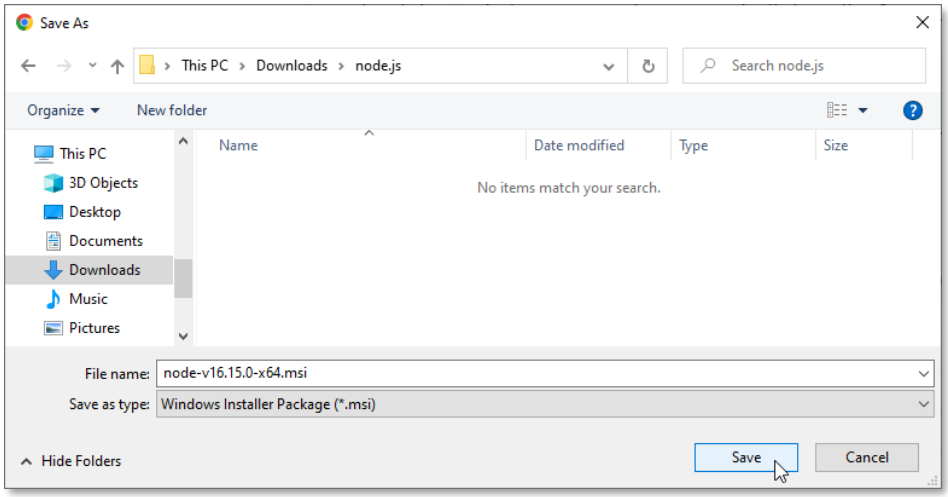
To confirm that the **OpenText Cloud Developer Tools for VS Code** are installed and working correctly, you can now switch to the **OpenText Cloud Developer Tools** view from the Activity Bar on the left side by clicking ot.



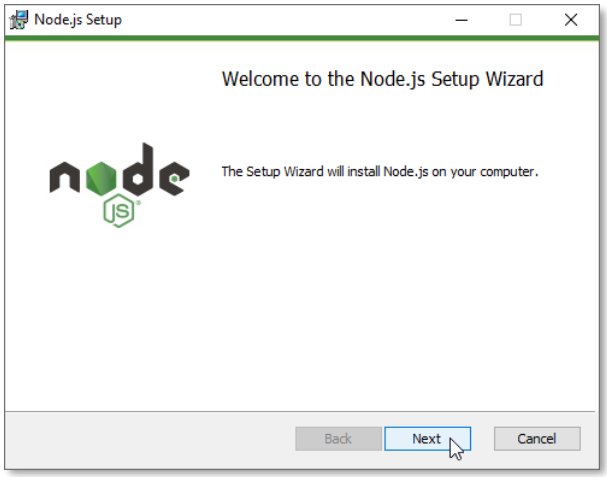


- Although you have now installed your VS Code IDE with the OpenText Cloud Developer Tools extension pack, you also need to install the latest Long-Term Support (LTS) version of Node.js (at the time of writing of this tutorial it is **16.15.0**), as you require this to support building and running the Contract Approval application specifically.
  To install **Node.js 16.15.0**, navigate to https://nodejs.org/en/download/, and download and run the correct installer for your OS.
  Like with the VS Code installation earlier, we will only go through the steps of the Node.js set up process for Windows 10. If you are installing another version of Node.js or are installing on another OS or version of Windows, screenshots and installation steps might differ.



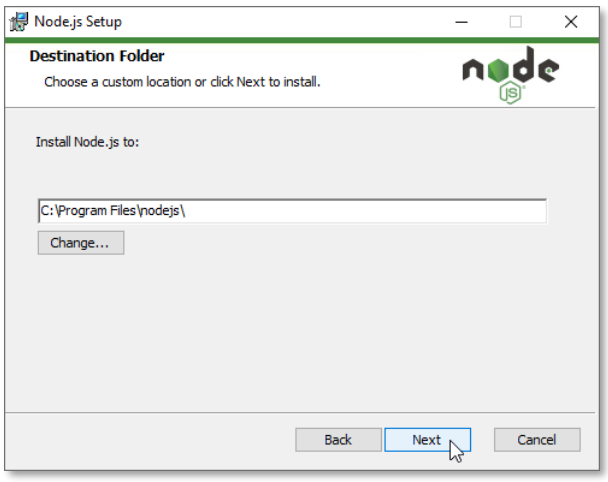


Click **Next** to continue.

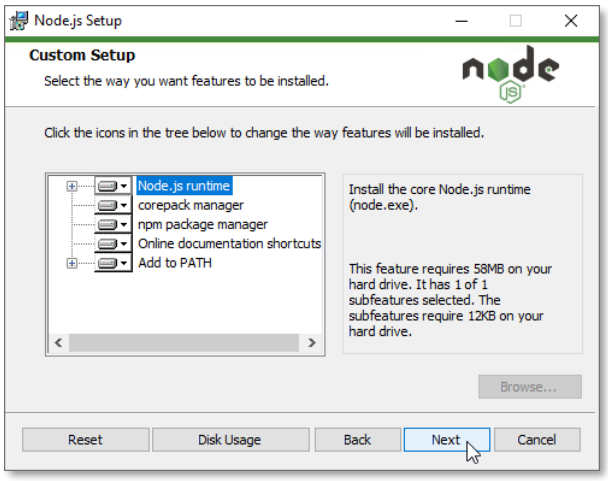


Select **I accept the terms in the License Agreement** and click **Next** to continue.

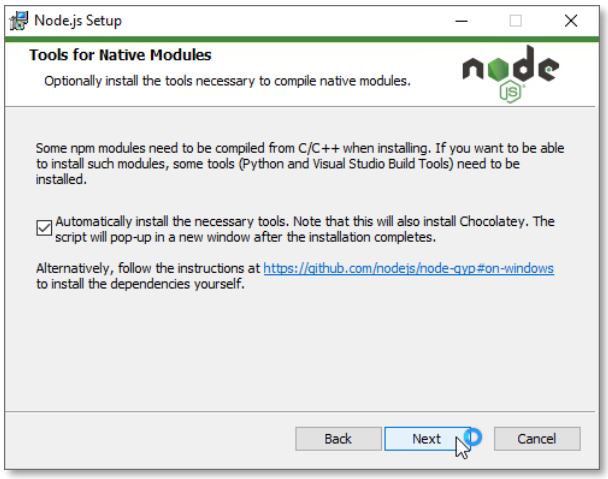
Select the installation destination location and click **Next**. You can use the suggested default location.



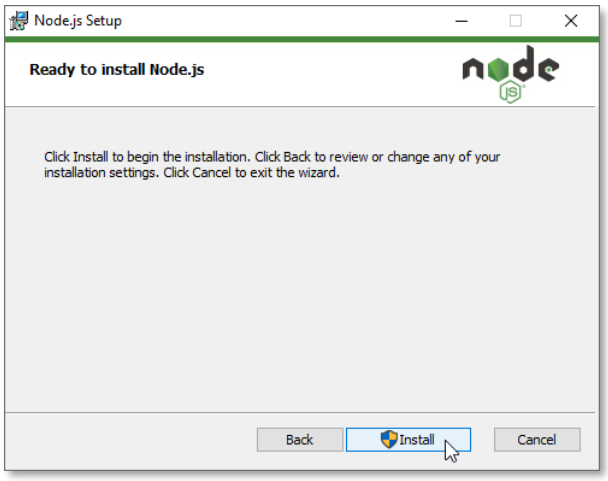
From the **Custom Setup** screen, we recommend you accept the default and just click **Next**.



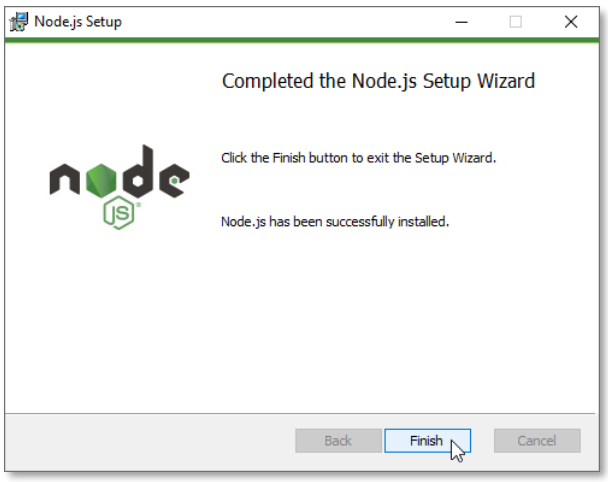
Select to **Automatically install the necessary tools** and click **Next**.



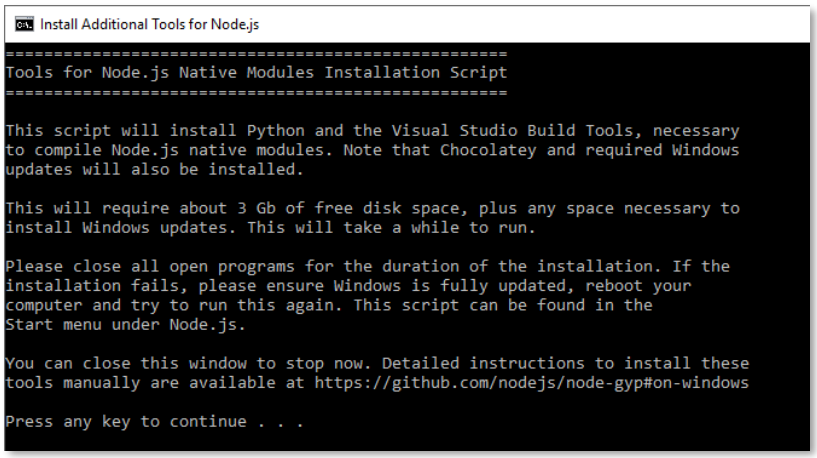
Click **Install** to start the installation.

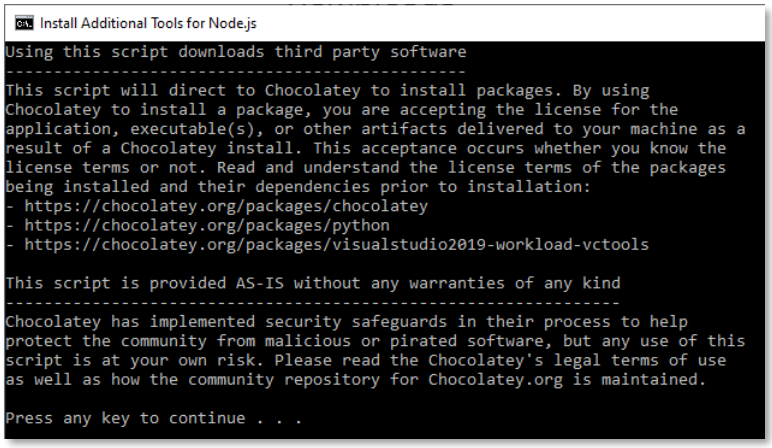


Once the setup has completed, you can click **Finish** to close the Node.js Setup Wizard.

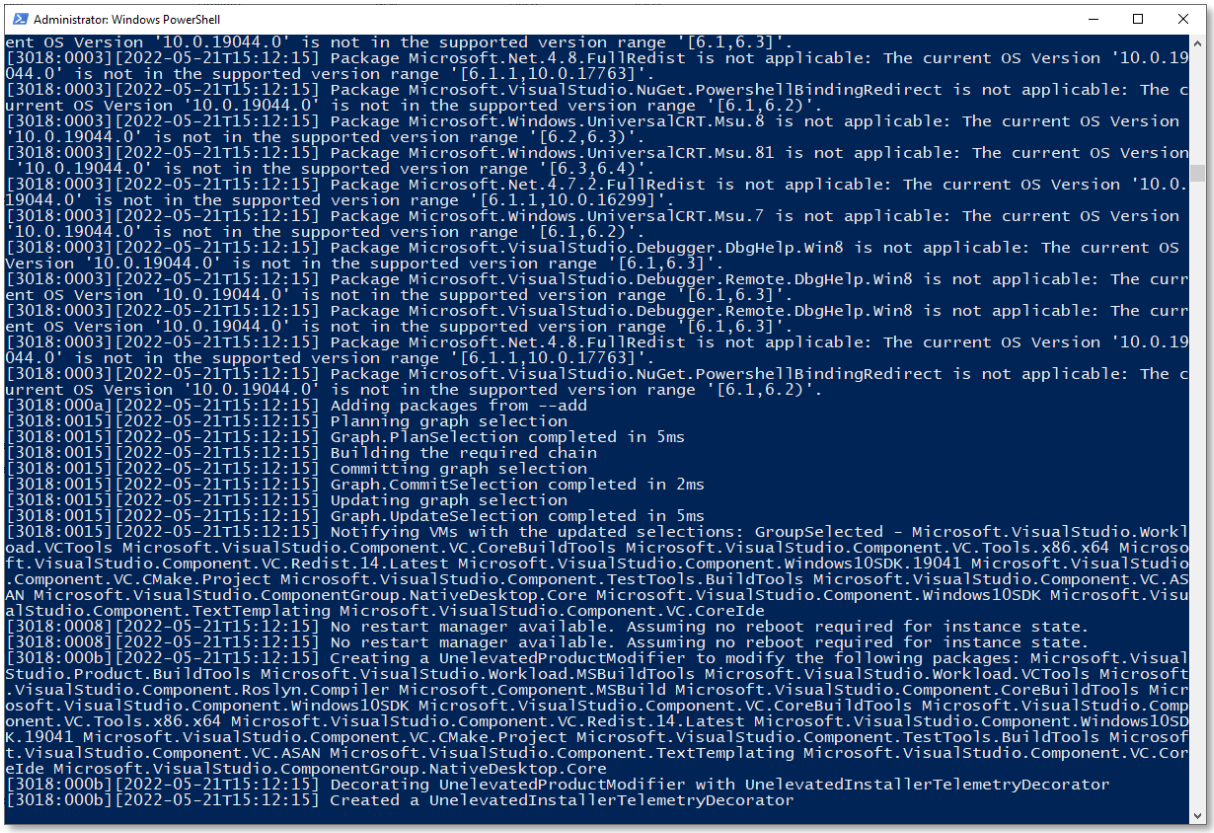


Press a key (e.g.: **space bar**) several times to run through the installation of the Additional Tools for Node.js.

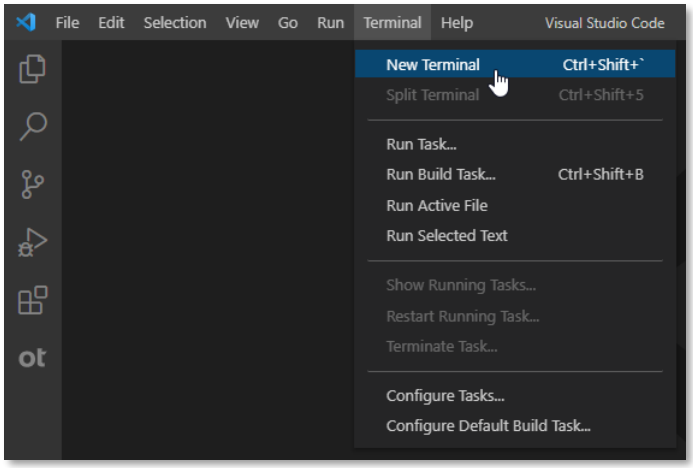




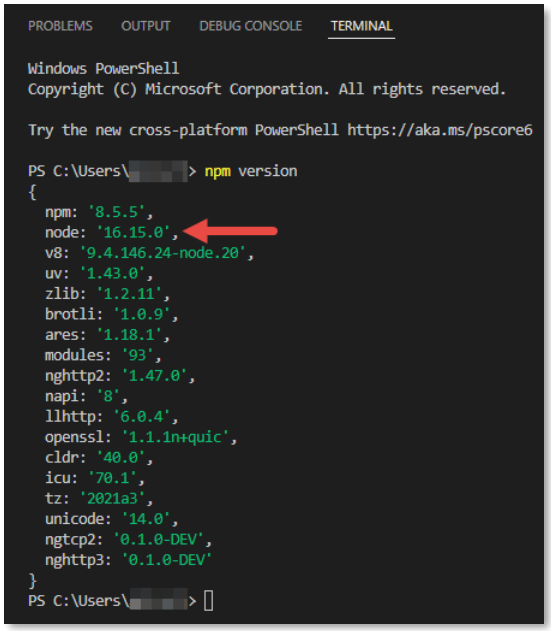
Depending on the components that are already existing on your system, PowerShell will install all the remaining necessary tools. Once done, press **Enter** to exit the PowerShell window that popped up.
If the PowerShell process seems to hang without a “Type Enter to exit” instruction (e.g.: in the below screenshot it hangs on “*Created a UnelevatedInstallerTelemetryDecorator*”), you can select the PowerShell window and try to push the installation forward by pressing **Enter**. This might not succeed immediately (i.e.: nothing happens), but when the installation process is ready to proceed, pressing **Enter** will result in continuing and when entirely done, the PowerShell window will close.



Node.js is now installed. Let's quickly verify that VS Code is ready to use it when running your Node.js code. If you left VS Code open while installing Node.js, just close and open it again to ensure the fresh install of Node.js is picked up.
In VS Code open a new **Terminal**.



In the newly opened **Terminal**, type **npm version** and confirm node is indeed of version **16.15.0**.



You are now ready to start building your Contract Approval Application project.

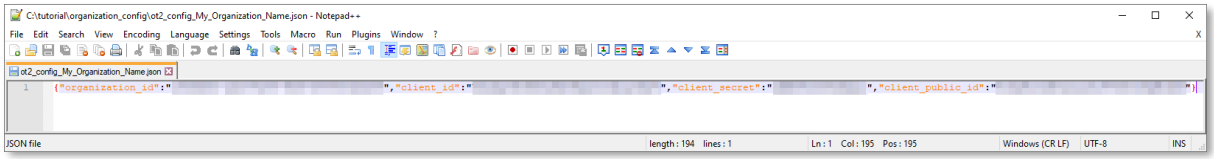
3.2 [10’] Adding an organization and testing the connection

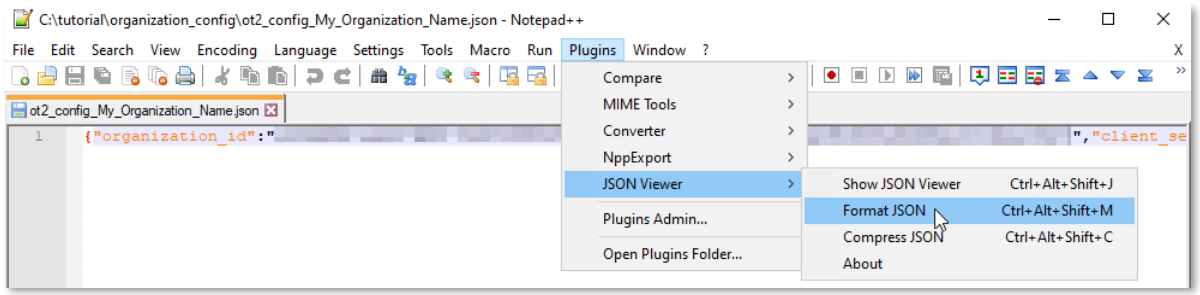
During this exercise you will add the organization you previously created (when registering for an OpenText Developer free trial Account) to your VS Code OpenText Cloud Developer user settings. You will also test the connection to your organization, to ensure all is indeed set up correctly. The purpose of adding an organization is that you will later be able to deploy the application you have built into the organization, and more specifically the corresponding single developer tenant.

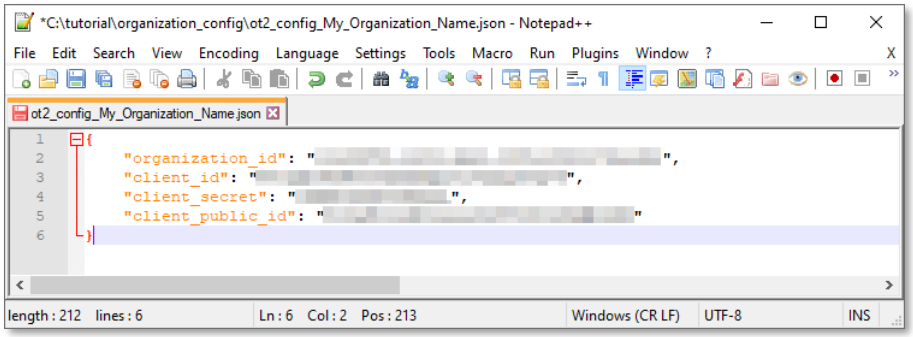
Once you are done with this section, you will have set up your Cloud Developer organization connection, and you are ready to start creating your Contract Approval application project.

To add your organization to your user settings and test the connection, proceed as follows:

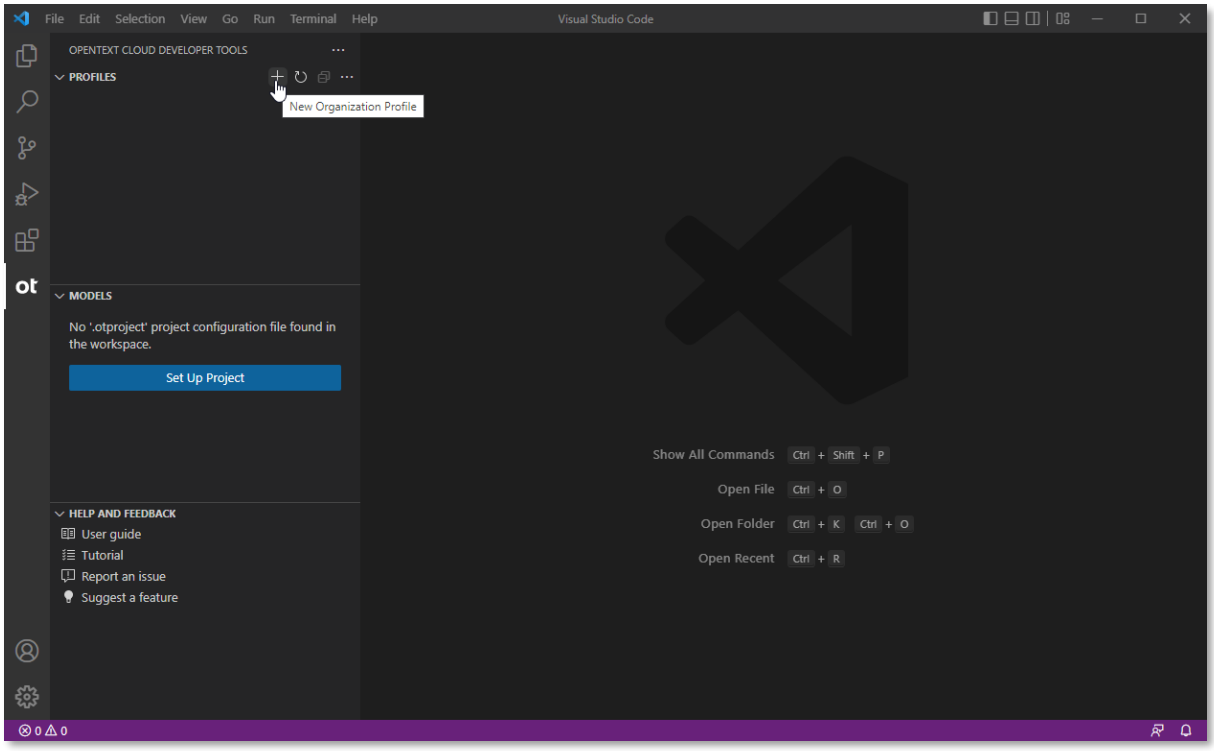
- Open the **ot2_config_<organization name, we are using My Organization Name>.json** organization configuration file you previously saved in a text editor. We recommend you use **Notepad++** with the **JSON Viewer** plugin installed, so that you can format the ot2_config_<organization name>.json file to be more easily consumable (as shown in below screen shots). This is not required though to perform the steps in this exercise, as any text editor will allow you to copy the different required values.





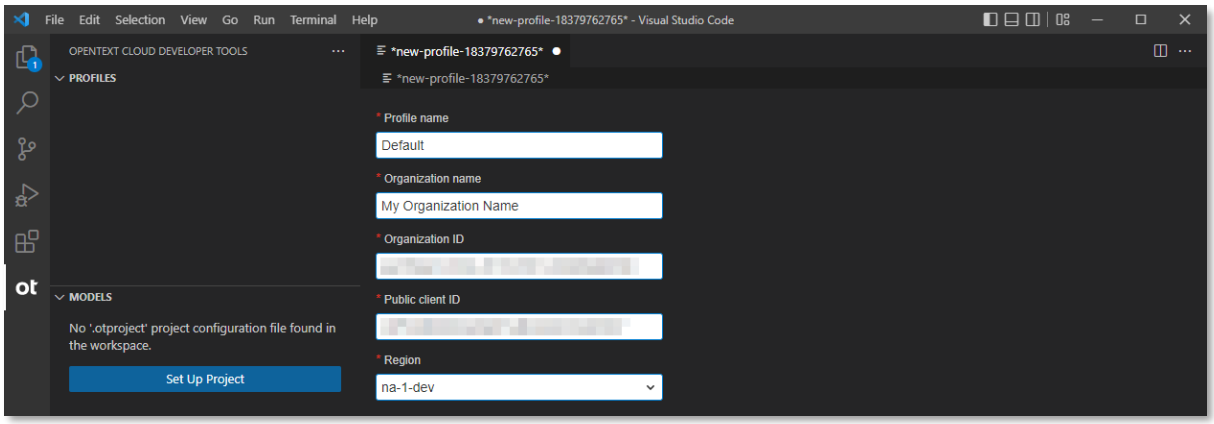


- You can add the organization connection configuration by adding a new Organization Profile. To do this, open VS Code, select the **OpenText Cloud Developer Tools** view from the Activity Bar on the left side, and click the ⊞ button in the **PROFILES** section.

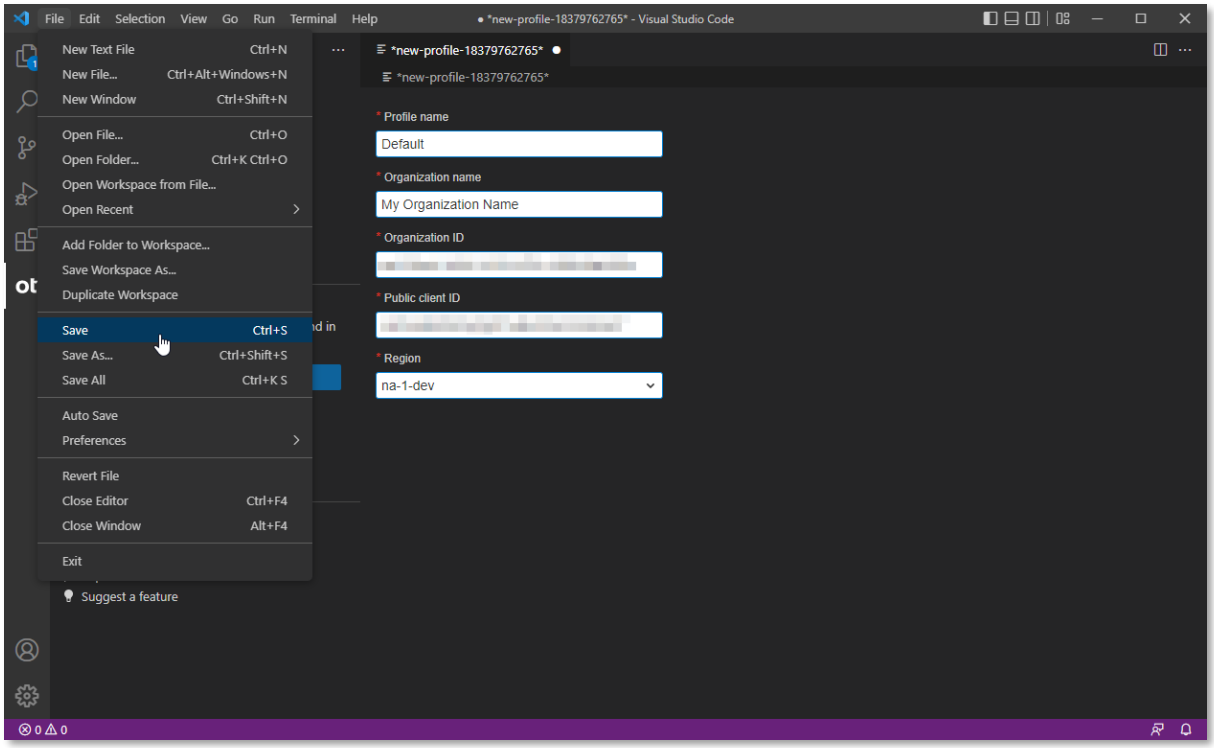


Fill the different authentication profile property values on the Organization Profile creation form as follows:

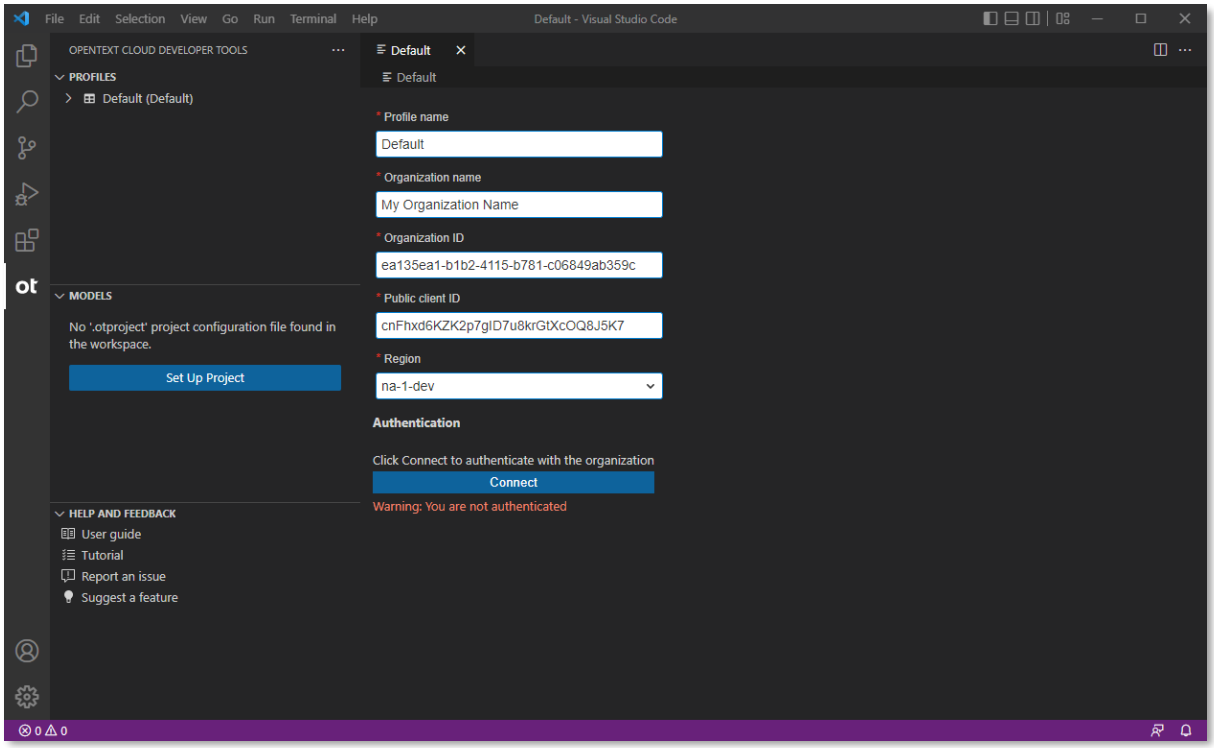
- **Profile name:** you can leave the **Default** value here
- **Organization name:** the name you chose for your organization
- **Organization ID:** copy/paste the value of the **organization_id** JSON property from the ot2_config_<organization name>.json organization config file, which you previously opened in a text editor
- **Public client ID:** copy/paste the value of the **client_public_id** JSON property from the ot2_config_<organization name>.json organization config file, which you previously opened in a text editor
- **Region:** you can leave this to **na-1-dev**



To save the authentication profile configuration, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

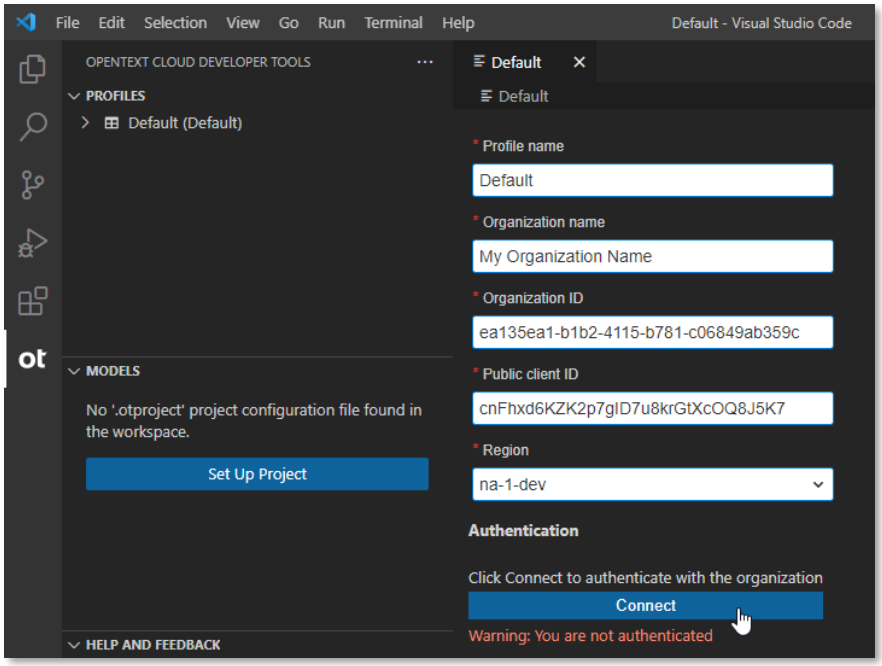


The “new profile” tab now indeed indicates that the authentication profile configuration has been saved, as it no longer displays as unsaved (and indeed has been renamed to “Default”).

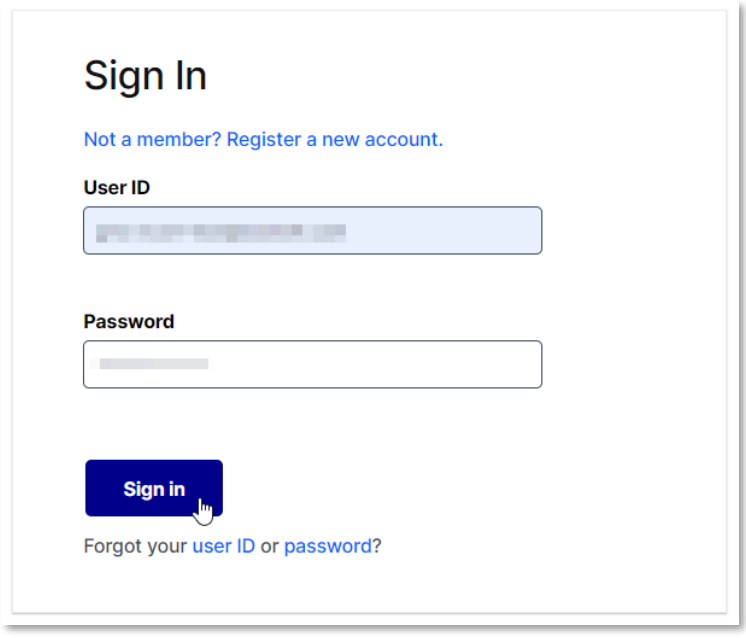


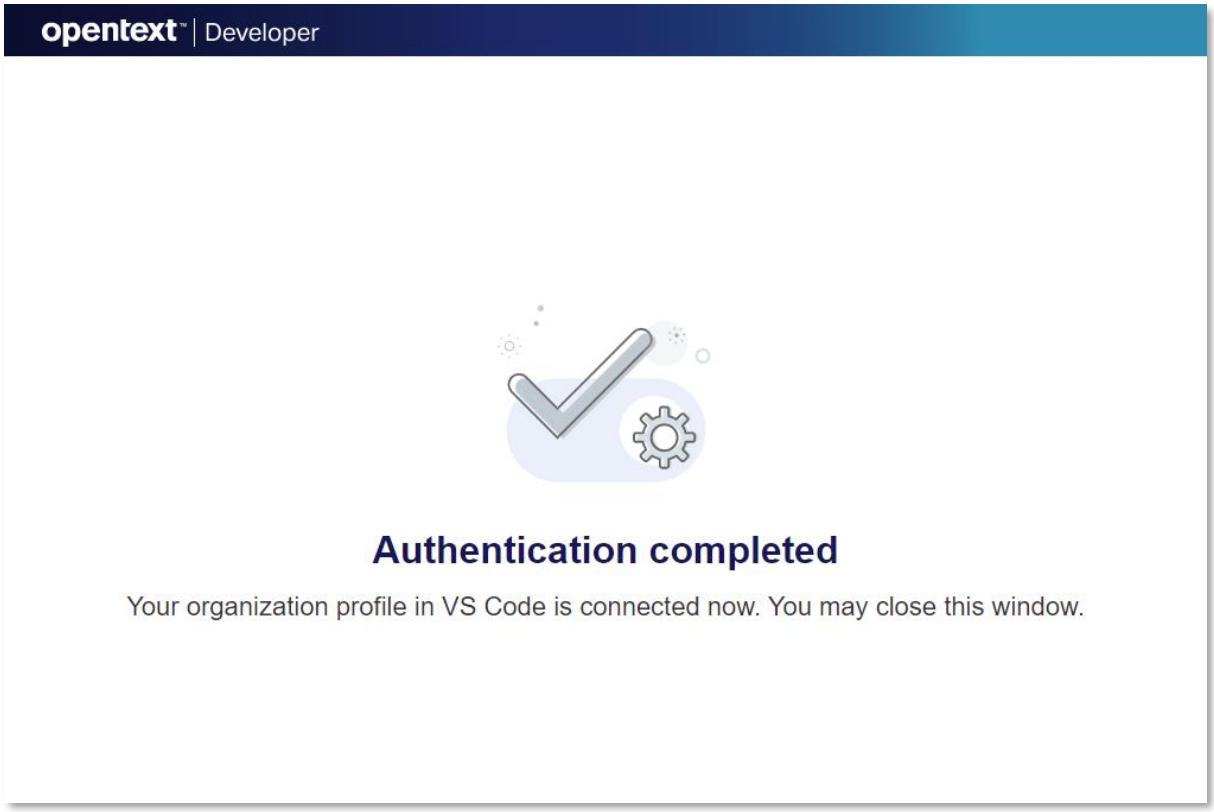
REMARK:
Note that all OpenText Cloud Developer Tools configuration artifacts, such as the previously set up organization connection, but also the project set up and different model configurations, use the standard VS Code file saving functionality. I.e.: to save your changes to any configuration artifact, you can always press **Ctrl+S** on your keyboard (for Windows systems) or use the **Save** menu entry from the **File** menu. You will also be presented with a "Do you want to save the changes you made to …?" dialog box when you try to close an unsaved configuration artifact.

- Directly from the Organization Profile form, click **Connect** to test the newly configured connection to your developer organization.

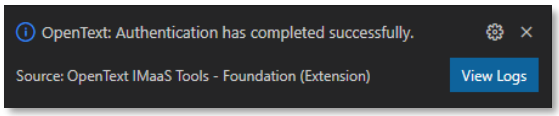


If not already connected (from an open session in your web browser), you will need to fill your OpenText Connect account **username (email address)** and **password** and click **Sign in**.

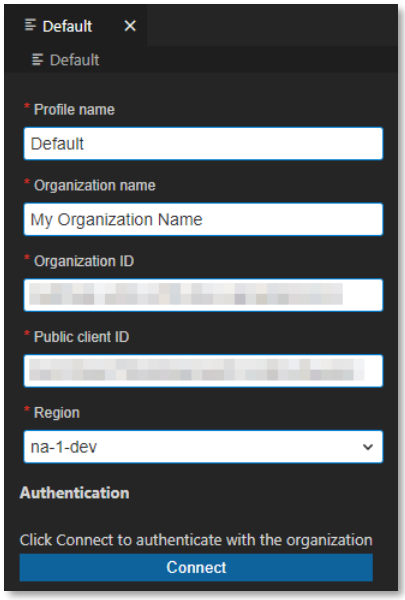




A pop-up message on the bottom right confirms that the authentication has completed successfully.



In addition to the authorization confirmation message, the “Warning: You are not authenticated” warning message should now no longer appear, indicating that you indeed have successfully authenticated with your configured organization.



You can now close the Organization Profile configuration screen.

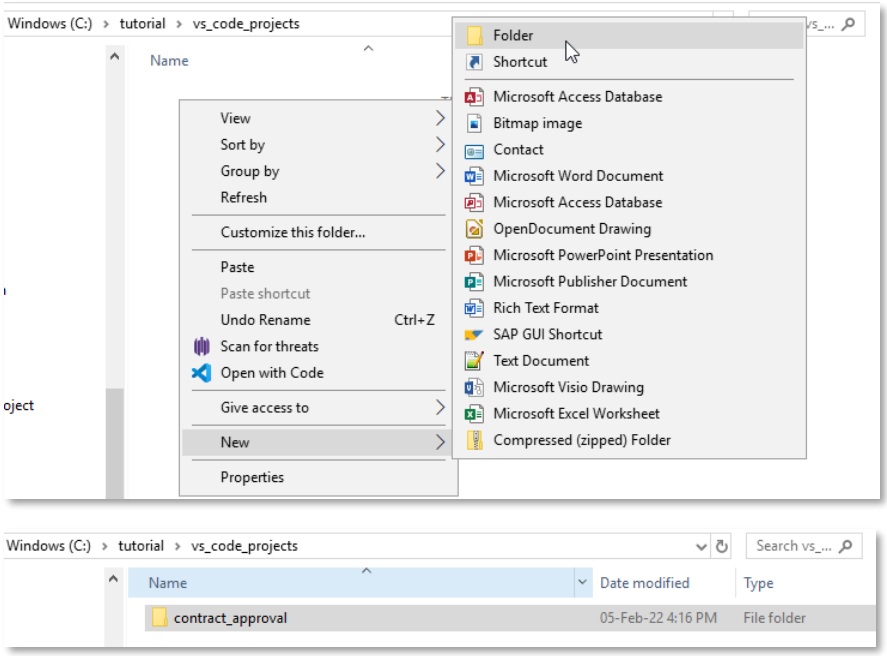
3.3 [15'] Creating an OpenText project

During this exercise you will create the folder on your file system in which you will be building your application. You will also set up your OpenText project. This includes filling the different application properties, which will be used to create the **Contract Approval** application in your developer organization when deploying the project for the first time. It is required to set up an OpenText project before you can start building your application components (or models).

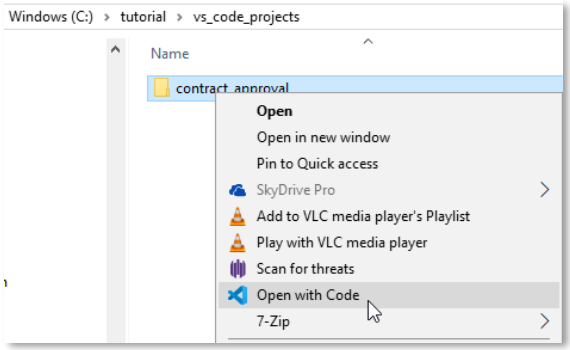
Once you are done with this section, you will have set up your OpenText project, and you are ready to start creating the Contract Approval namespace.

To create an OpenText project for your **Contract Approval** application, proceed as follows:

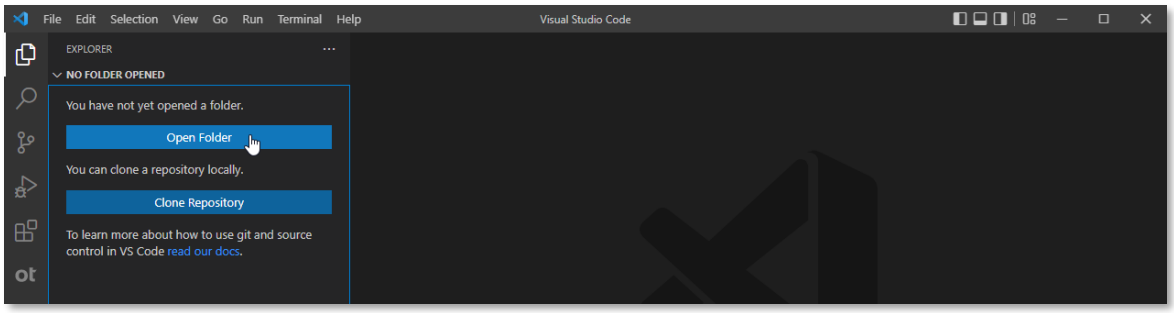
- Use your system's file system explorer (example shows Windows File Explorer) to create a new **contract_approval** folder.

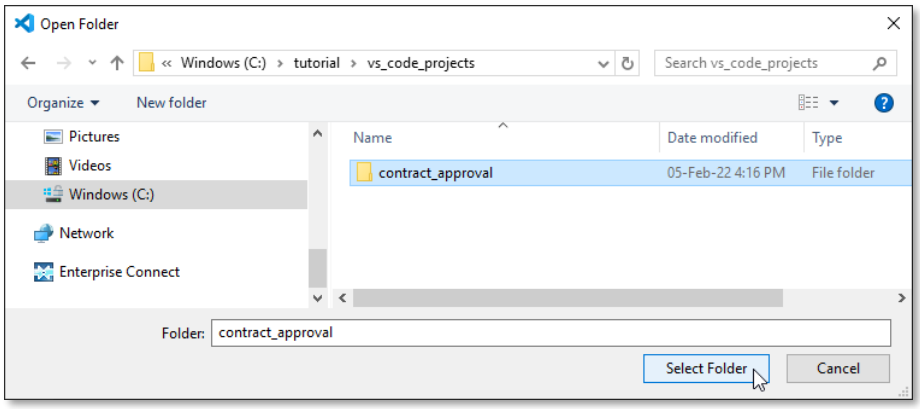


- Choose one of the following two options to open the Contract Approval project folder in VS Code:
  - OPTION 1:
    If you are using Windows (and you followed the VS Code installation instructions exactly as described in this tutorial), in the Windows File Explorer, right-click the newly created **contract_approval** folder and select the **Open with Code** contextual menu item to open the Contract Approval project folder in VS Code.

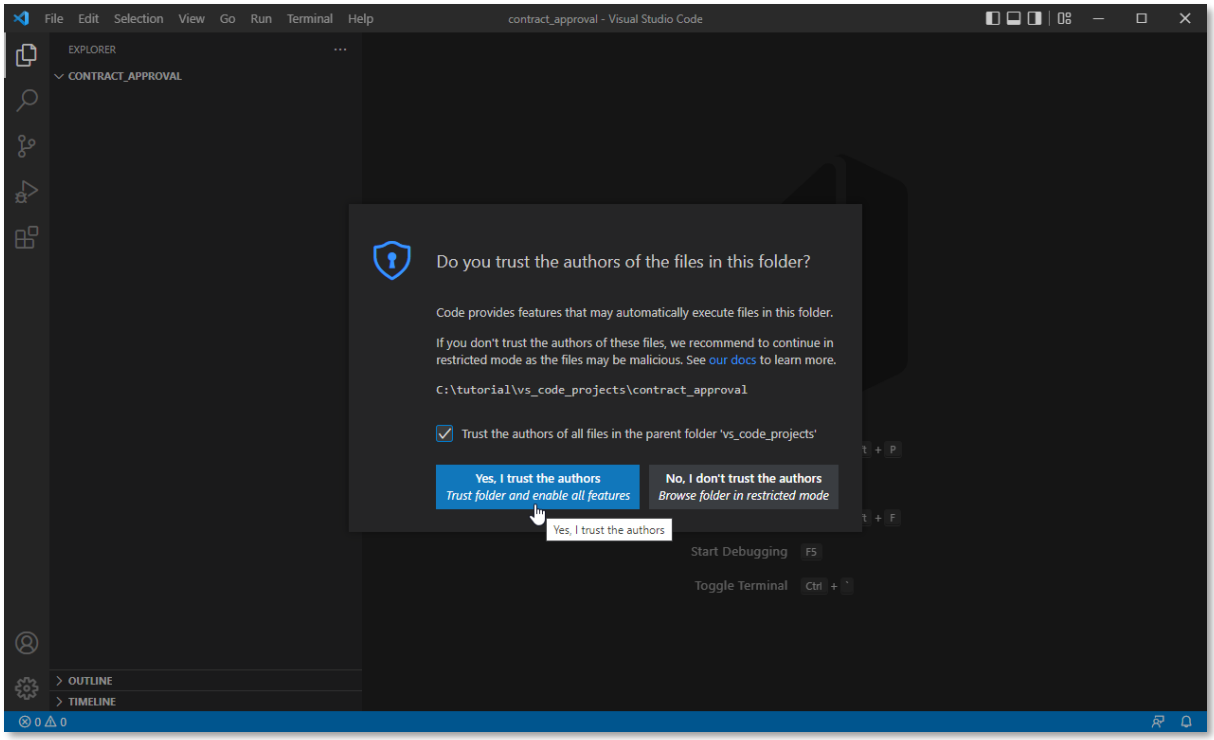


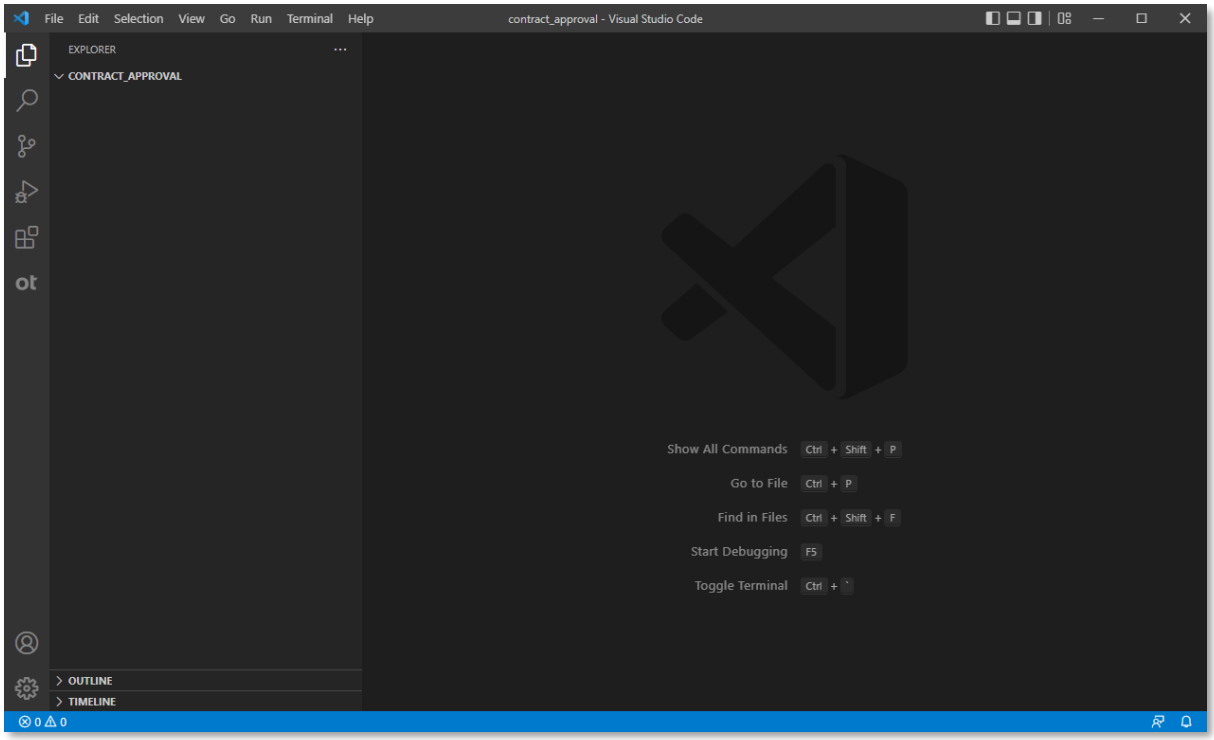
- OPTION 2 (should work with all Operating Systems):
  Alternatively (if the contextual menu option is not available), you can use VS Code's **Open Folder** button from the **Explorer** view in the Activity Bar.



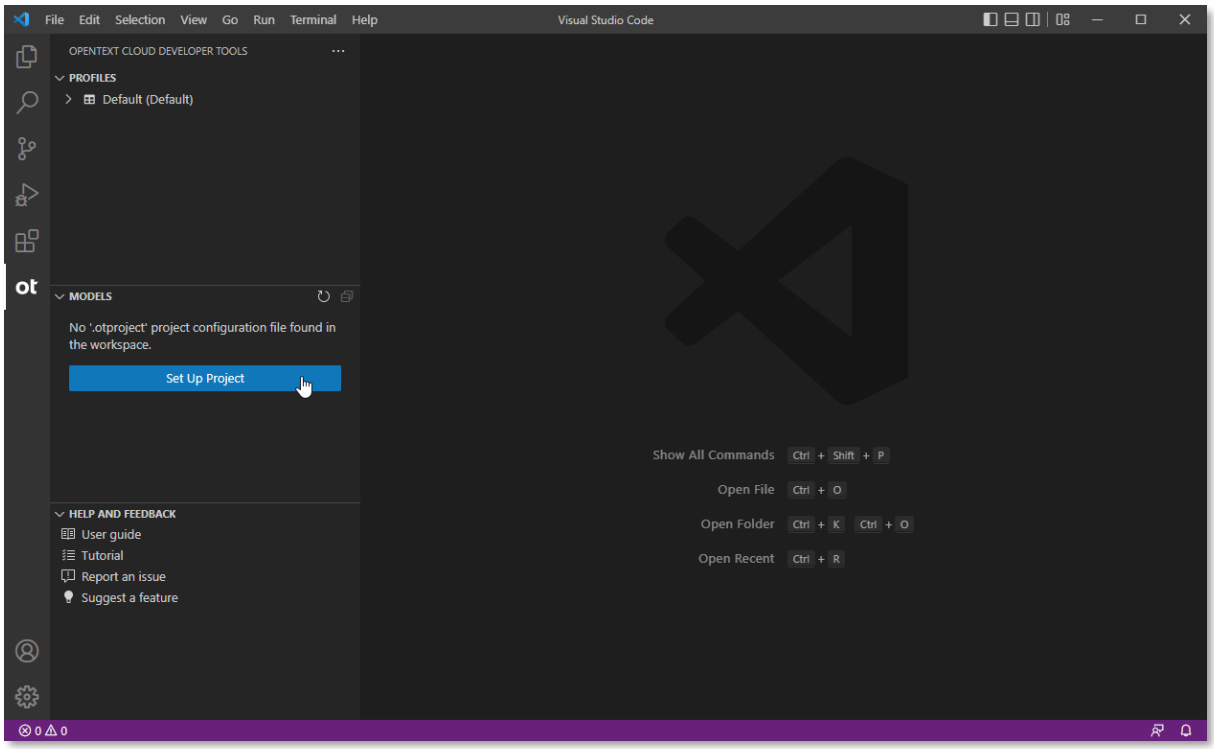


- Once the **contract_approval** folder is open in VS Code, choose to **Trust the authors of all files in the parent folder '…'** and click the **Yes, I trust the authors** button.



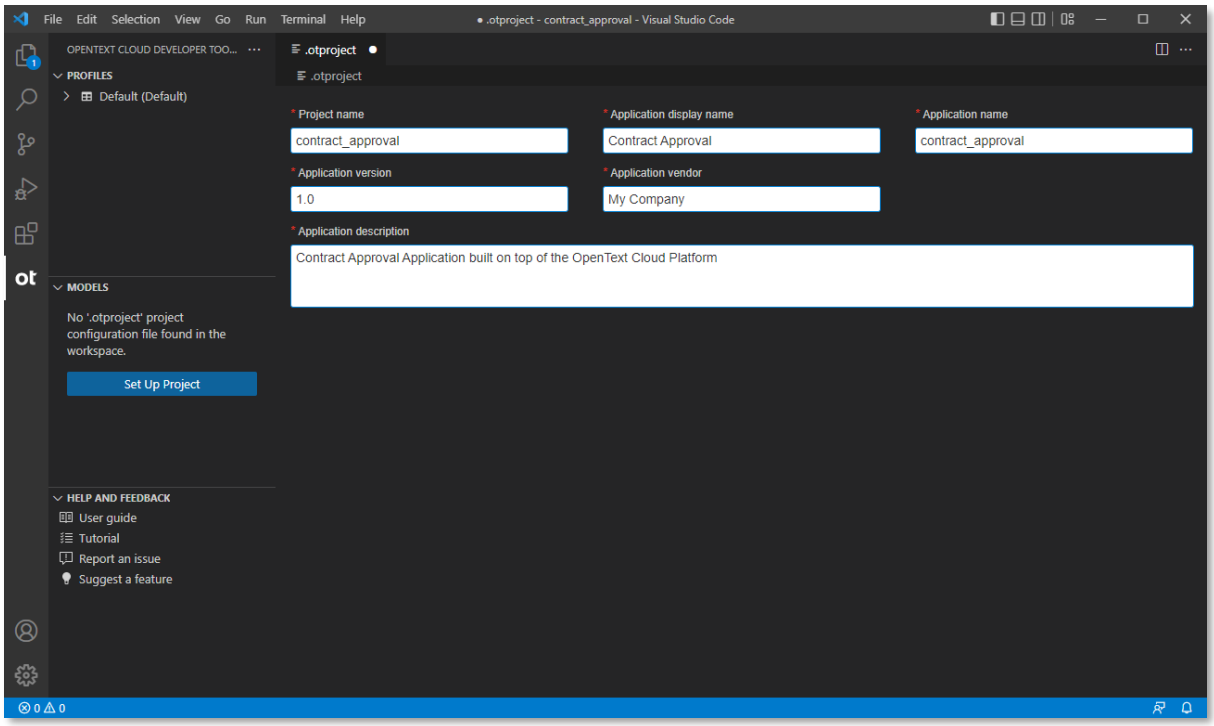


- Although you can now start creating files from VS Code in your Contract Approval application project folder, you still need to set up the OpenText project to be able to start building models. To set up the OpenText project, switch to the **OpenText Cloud Developer Tools** view and click the **Set Up Project** button.

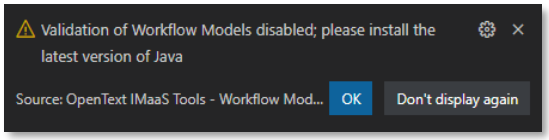


Fill the **OpenText Project Properties** as follows:

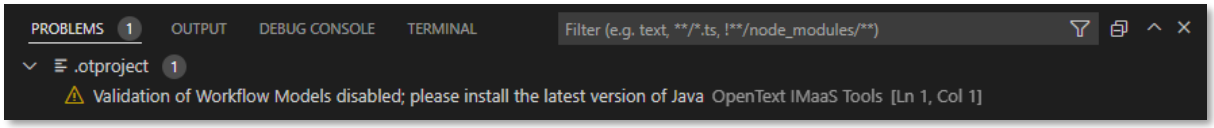
- **Project name:** the project name has been automatically populated from the project folder name (**contract_approval**) and does not require changing
- **Application display name: Contract Approval**
- **Application name:** when filling the application display name, the system will automatically populate the application name and you can leave the generated **contract_approval** value
- **Application version: 1.0**
- **Application vendor:** you can use any company name you like; we are using **My Company**
- **Application description: Contract Approval Application built on top of the OpenText Cloud Platform**



You can now save and close the **OpenText Project Properties** form.
Depending on whether or not you have Java installed on your system, you will get the following warning message popping up in the bottom right of VS Code (when Java is not installed).

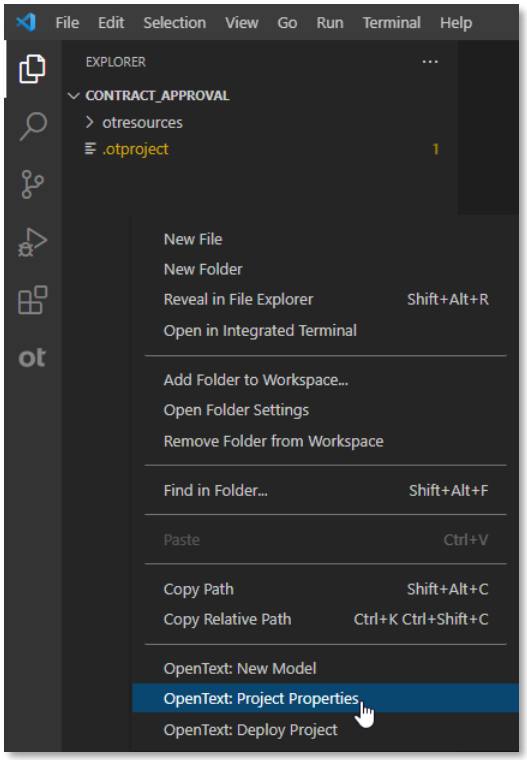


You can choose to simple close this message (clicking **OK**) or to not display it again (clicking **Don't display again**). Whichever option you chose, as long as Java is not installed you will always find the warning message in the **PROBLEMS** tab:

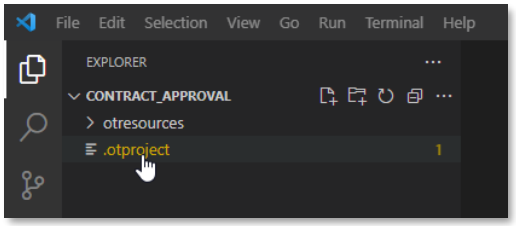


- If you previously got the **Validation of Workflow Models disabled; please install the latest version of Java** warning message, please continue with the next steps to install Java on your system.
  If Java is already installed (you did not get any warning message), you can directly start with the next exercise section (Creating a namespace) as your OpenText project has been set up and you are ready to start creating models.

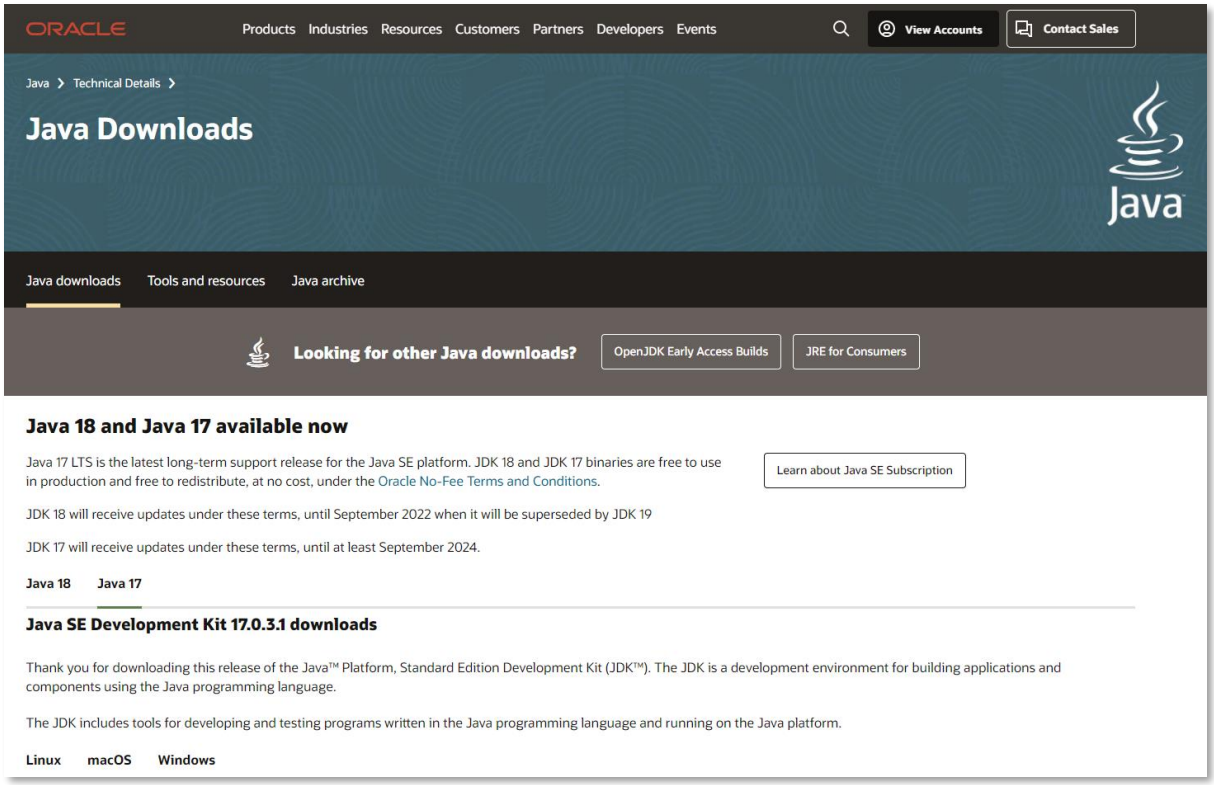
  Before you do though, note that you can always view and modify the OpenText project properties from the VS Code **Explorer** view by choosing **OpenText: Project Properties** from the contextual menu of your project (root) folder or any of its subfolders, or by clicking/opening the **.otproject** file:



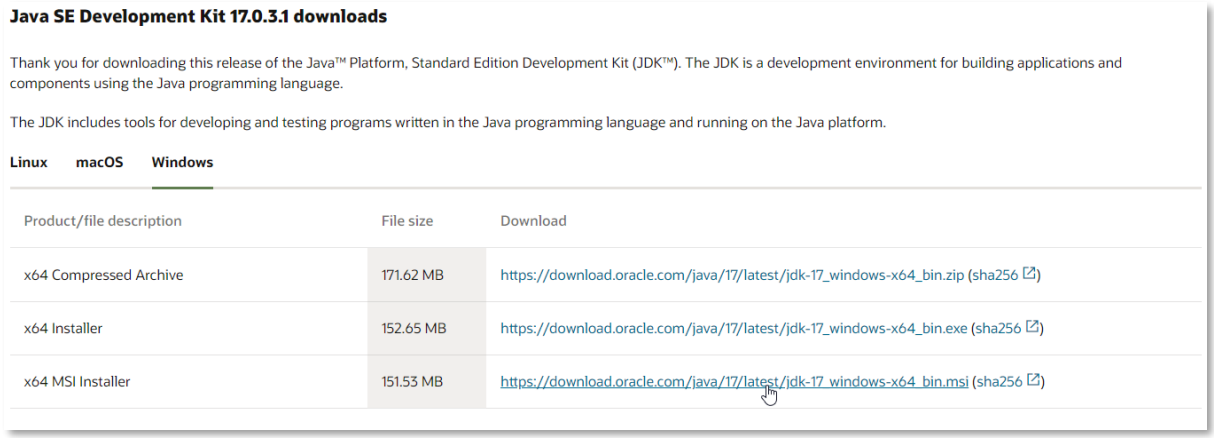
  Or



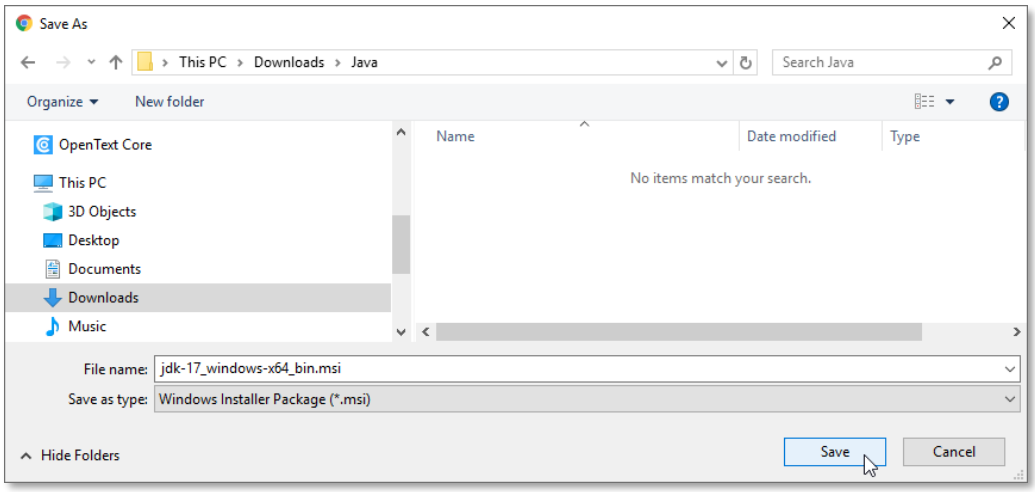
- Although you can choose to install any Java distribution (OpenJDK, Oracle or other) on your system, for your convenience we are providing you with the steps to install the Oracle Java version. To install Java on your system, first navigate to https://www.oracle.com/java/technologies/downloads.



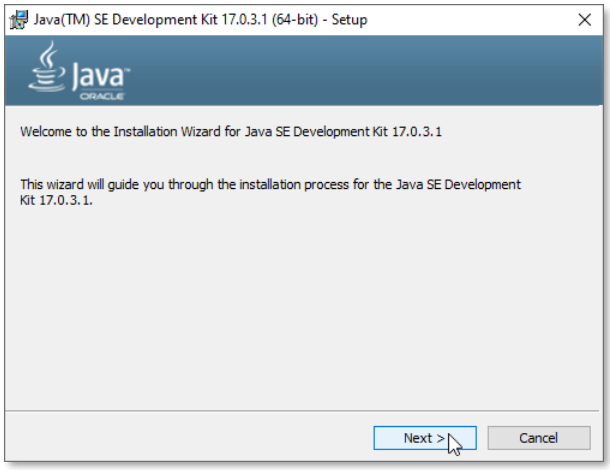
At the time of writing of this tutorial, Java 17 LTS (we recommend you use the latest LTS version) and the corresponding JDK 17 binaries are free to use in production and free to redistribute, at no cost, under Oracle No-Fee Terms and Conditions (feel free to read this for more information).
In that context we recommend you install the **Java SE Development Kit 17** version that corresponds with your OS on your system (currently 17.0.3.1).
For this tutorial we will be using the **x64 MSI Installer** for Windows. If you are also on a Windows machine, feel free to use the https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi link.



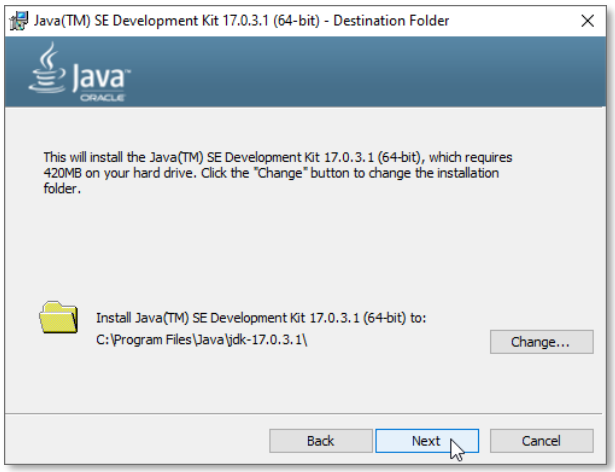
Save and run the **jdk-17_windows-x64_bin.msi** installer.



Choose **Next >** to start with the Java SE Development Kit installation process.

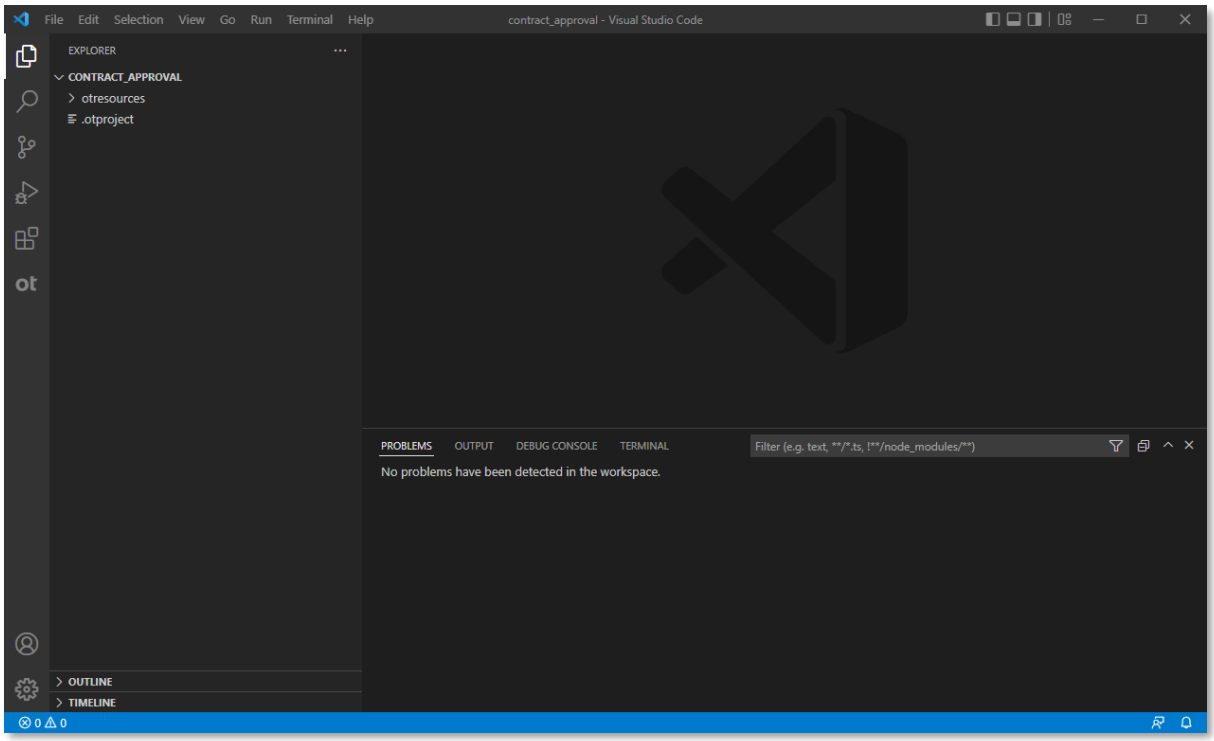


Click **Next** to accept the default installation folder (feel free to change it, if needed).



Once the installation is complete, click **Close** to exit the installation wizard.

Close and reopen VS Code, and you should now see that the **Validation of Workflow Models disabled; please install the latest version of Java** warning message does no longer appear (not as a pop up in the bottom right and not in the **PROBLEMS** tab).



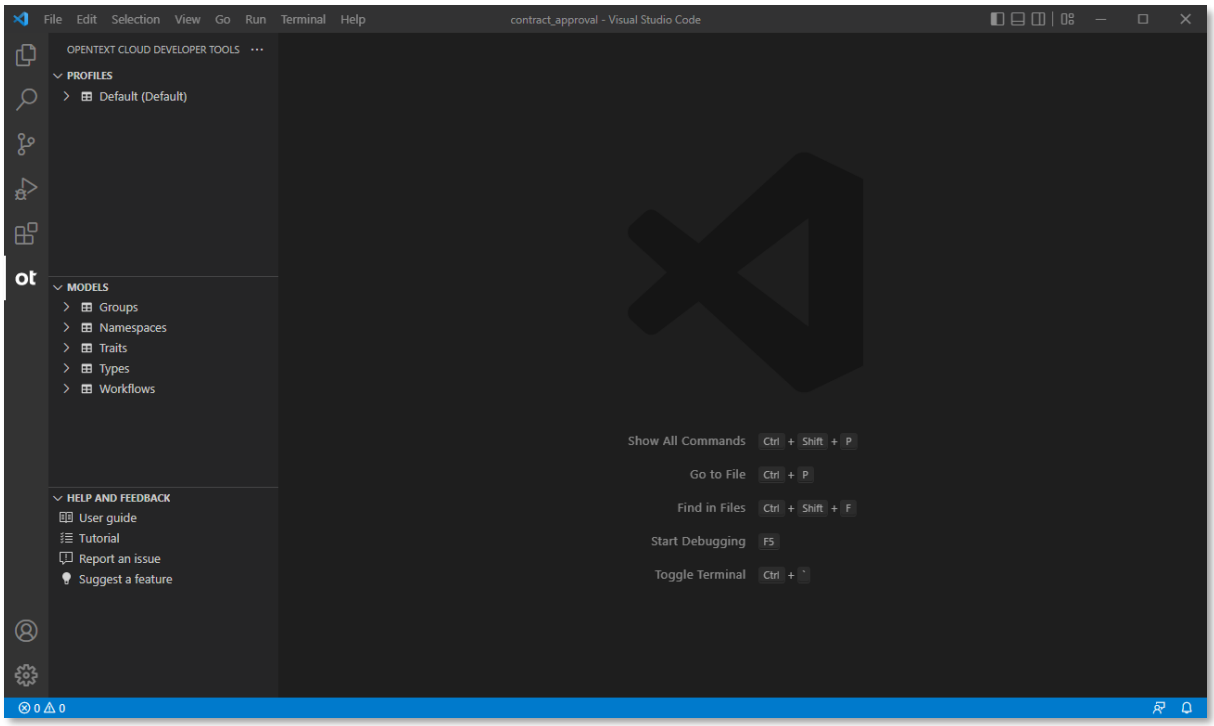
3.4 [10’] Creating a namespace

During this exercise you will create the **Contract Approval** namespace model. A namespace allows grouping the different type and trait definitions together (e.g.: within the context of an application). For more information on namespaces, you can refer to the 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.

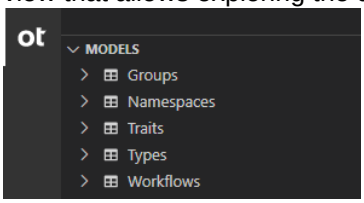
Once you are done with this section, you will have created the namespace that corresponds with your Contract Approval application, and you are ready to start creating the Approval trait definition.

To create the **Contract Approval** namespace, proceed as follows:

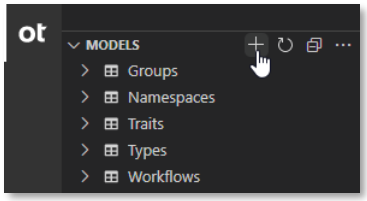
- In VS Code, switch to the **OpenText Cloud Developer Tools** view.

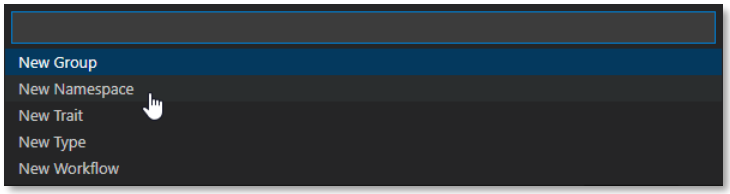


  As you can see, since your OpenText project has been set up, the **MODELS** section shows a tree view that allows exploring the different models in your application project.

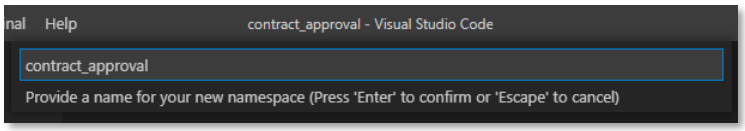


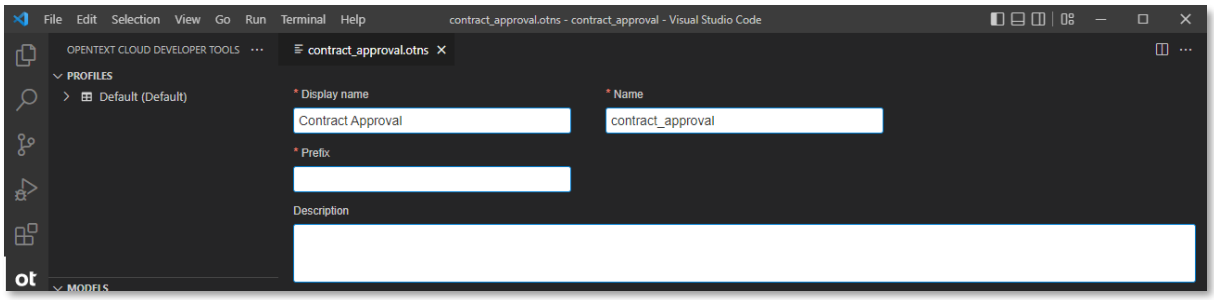
- To create a new namespace, click ⊞ at the top right of the **MODELS** section and from the menu that drops down from the top of VS Code, select **New Namespace**.





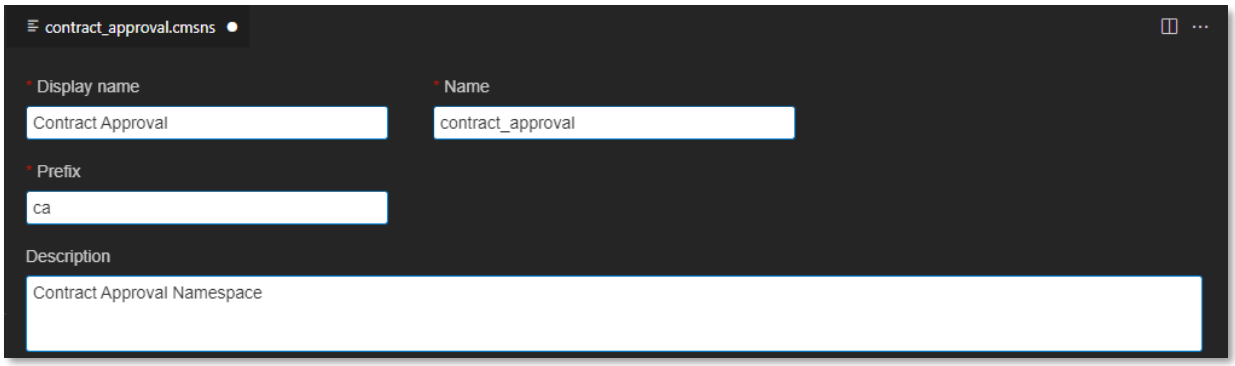
In the input box that appears at the top of VS Code, fill **contract_approval** as the name for the new namespace and press **Enter**.





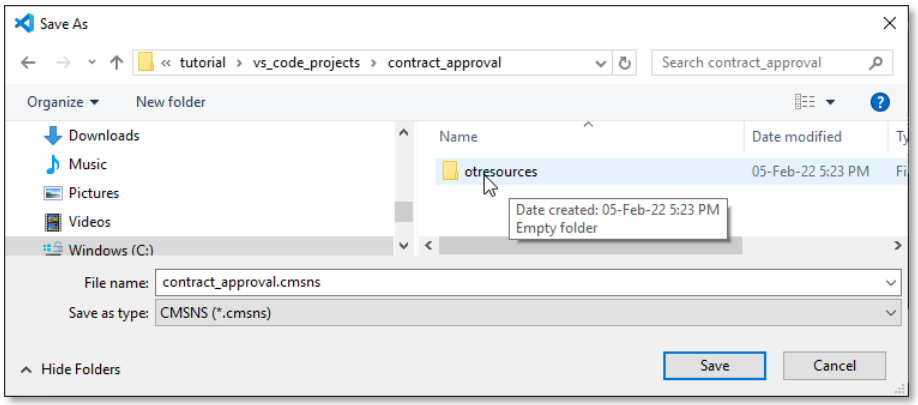
Fill the namespace properties as follows:

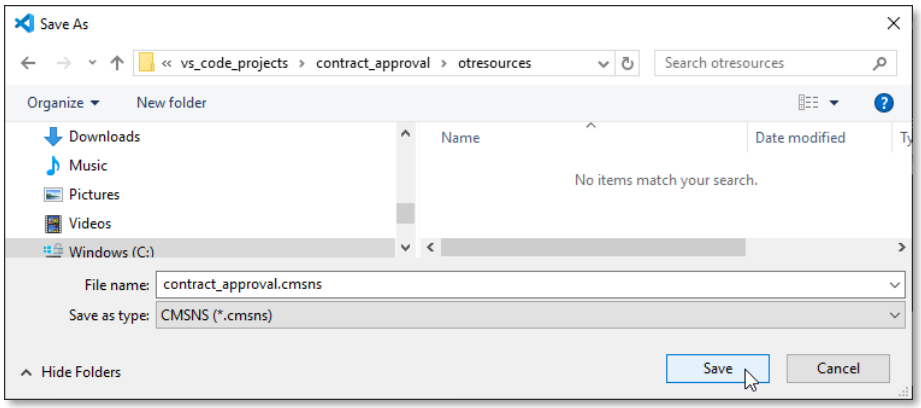
- **Display name:** the display name is the user-friendly name for the namespace; this does not have to be unique, and it has been automatically populated for your convenience based on the previously chosen namespace name (the model file name); you should best leave the value to be **Contract Approval** as it nicely aligns with the model file name and model name
- **Name:** the name is the technical name for the namespace; this has to be unique (within your developer tenant), and it has been automatically populated for your convenience based on the previously chosen namespace name (the model file name); you should best leave the value to be **contract_approval** as it nicely aligns with the model file name and model display name
- **Prefix:** the prefix is the prefix representing the namespace (used in system naming of traits and types that are within that namespace); this has to be unique (within your developer tenant); please fill **ca** as its value
- **Description: Contract Approval Namespace**



REMARK:
Note that for the current namespace model creation exercise we are using the ⊞ from the **MODELS** section in the **OpenText Cloud Developer Tools** view. From the **MODELS** section you could also have used the **[…]** menu and the **New Model** menu entry instead of the ⊞ button. There are two more ways to create models and those will be illustrated in the next tutorial exercises.

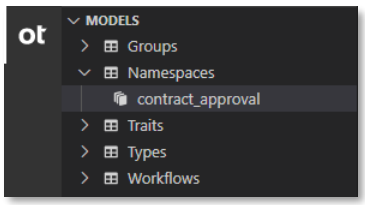
You can now save and close the **Contract Approval** namespace model. Saving will pop up a **Save As** file saving dialog box. Make sure to select the **otresources** folder as target folder, leave the file name as is (**contract_approval.cmns**) and click **Save**.





IMPORTANT:
The **otresources** folder is the model folder and it was automatically generated during the project setup. It is important you create all models inside this **otresources** folder, as the system will refuse to store models anywhere else.

The model explorer tree in the **MODELS** section should now show your 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).



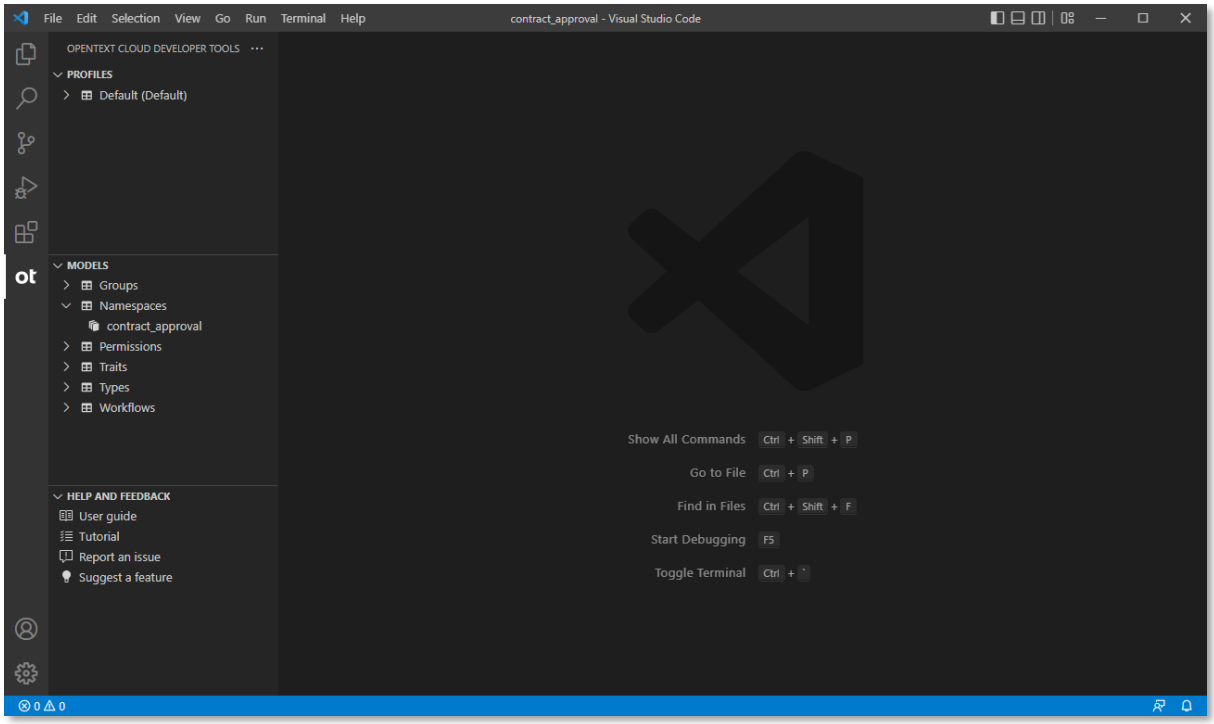
3.5 [15’] Creating a trait definition

During this exercise you will create the **Approval** trait definition model. A trait definition allows grouping several attributes into one more complex multi-attribute property. Trait instances can be dynamically added to a type instance as part of the business process when using the application, but they can also be made mandatory as a required trait in a type definition, so that they must always be added when creating a new type instance. For the purpose of this tutorial, we will be using the concept of mandatory traits to represent the different approval steps on a contract (hence the Approval name of the trait definition). For more information on traits (definitions and instances), you can refer to the 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.

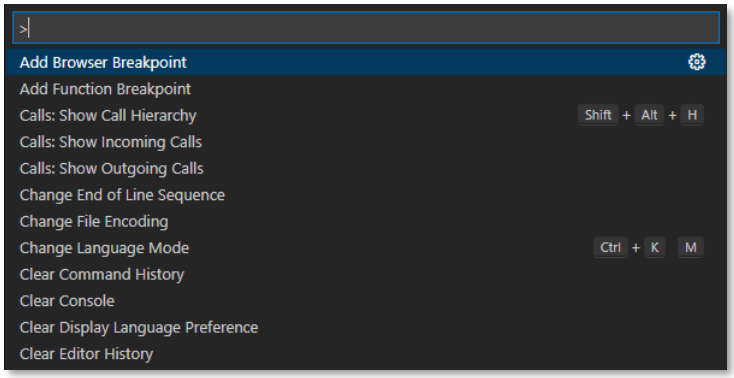
Once you are done with this section, you will have created the trait definition that corresponds with your Contract Approval application’s approval steps, and you are ready to start creating the Contract type definition.

To create the **Approval** trait definition, proceed as follows:

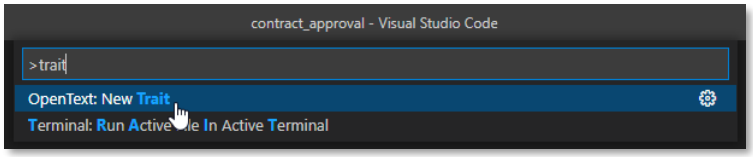
- In VS Code, switch to the **OpenText Cloud Developer Tools** view.



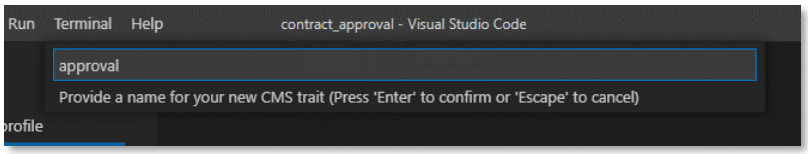
- To create a new trait definition, press **F1** on your keyboard (or **Ctrl+Shift+P** if F1 doesn't work). This opens the Command Palette at the top of VS Code.

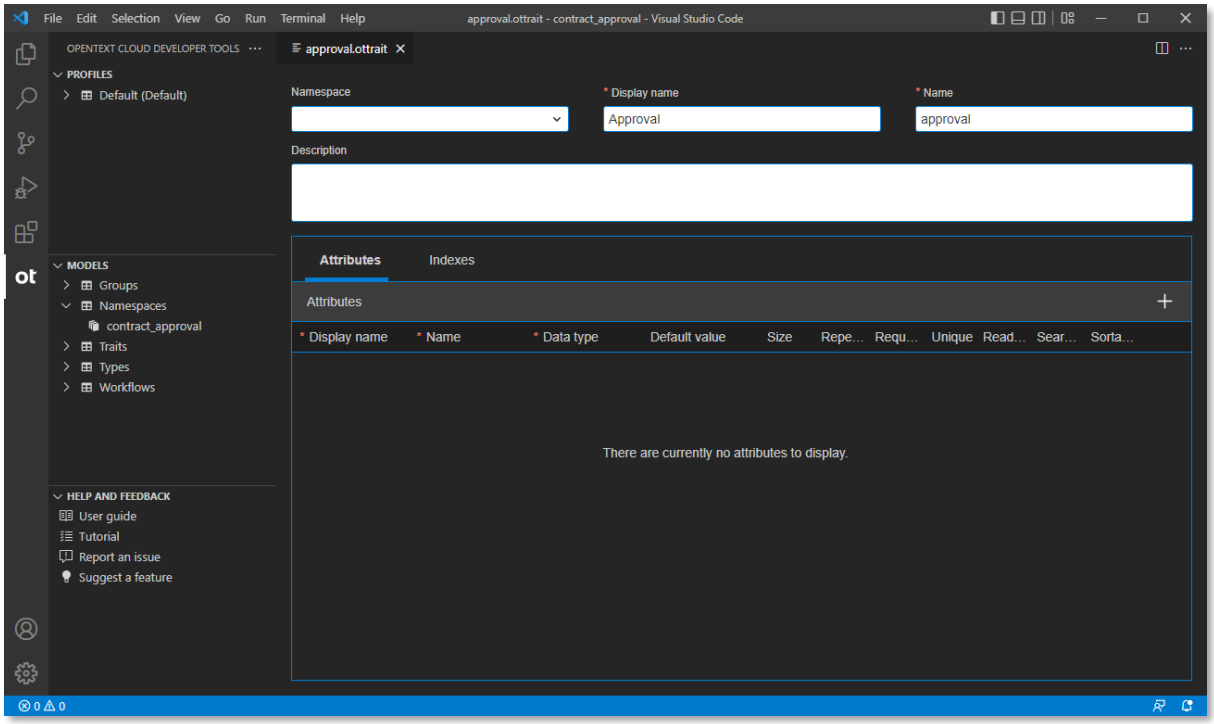


In the Command Palette, type **trait** and you should see the command list being filtered to show the **OpenText: New Trait** command near the top (in the case of this tutorial it's the first entry). Select the **OpenText: New Trait** command.



In the input box that appears, fill **approval** as the name for the new trait definitions and press **Enter**.





Fill the trait definition properties as follows:

- **Namespace:** the namespace to which this trait definition belongs; as you previously created the **contract_approval** namespace, you can now select it as the namespace for your trait definition

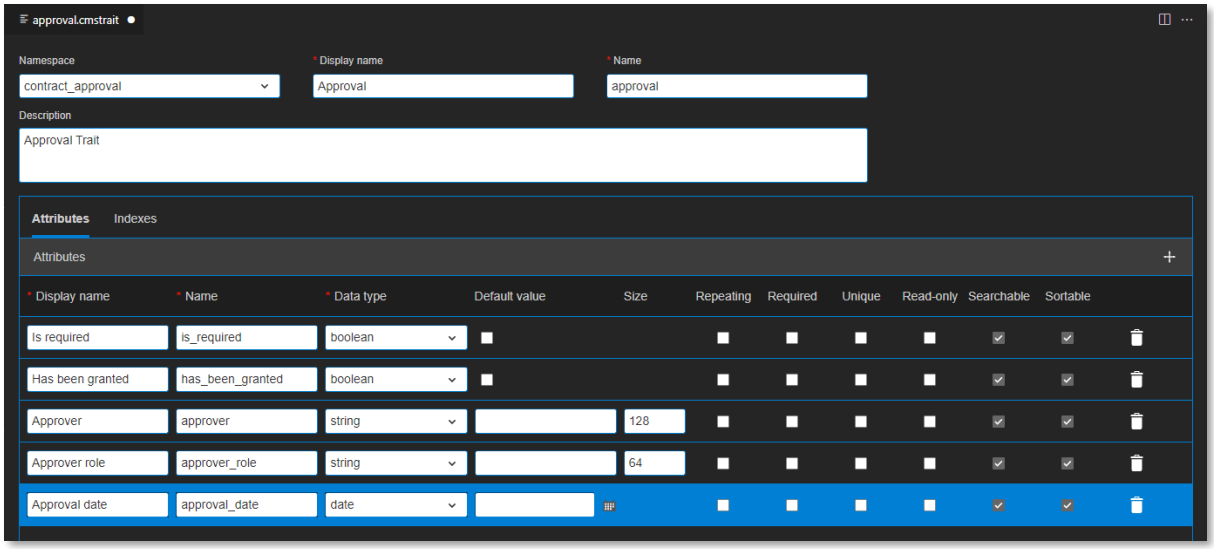
> **REMARK:**
> Note that the OpenText Cloud Developer Tools dynamically update the different model reference lists. In the context of this trait definition, the list of available namespaces is dynamically updated when you add namespaces, but e.g., in context of type definitions, the list of available traits and parent types will also update, based on the available trait and type definitions.

- **Display name:** the display name is the user-friendly name for the trait definition; this does not have to be unique, and it has been automatically populated for your convenience based on the previously chosen trait definition name (the model file name); you should best leave the value to be **Approval** as it nicely aligns with the model file name and model name
- **Name:** the name is the technical name for the trait definition; this has to be unique in context of the selected namespace (and the combination of namespace and trait definition name needs to be unique within your developer tenant), and it has been automatically populated for your convenience based on the previously chosen trait definition name (the model file name); you should best leave the value to be **approval** as it nicely aligns with the model file name and model display name
- **Description: Approval Trait**
- **Attributes:** the attributes list defines the different attribute definitions of the trait definition; each attribute definition has the following properties:
  - **Display name:** the display name is the user-friendly name for the attribute definition; this does not have to be unique, but this is recommended (to avoid confusion)
  - **Name:** the name is the technical name for the attribute definition; this has to be unique within the trait definition, and it gets automatically populated for your convenience based on the display name you fill
  - **Data type:** the data type of the attribute; this is a pick list (bigint, boolean, date, double, integer, string and uuid)
  - **Default value:** you can assign a default value for the attribute (i.e.: the value that gets automatically assigned to the attribute when creating a new instance of the trait definition); whether it is possible to assign a default value and how to assign it depends on the chosen data type (e.g.: a date data type gets a date picker, and a uuid data type doesn't allow for a default value)
  - **Size:** the size property only applies to the string data type and can thus only be chosen when picking the string data type; it represents the maximum length constraint for the string attribute
  - **Repeating:** whether or not the attribute is multi-valued (can have multiple values)
  - **Unique:** whether or not the attribute needs to be unique across all instances of the trait definition
  - **Required:** whether or not the attribute must be filled upon creation
  - **Read-only:** whether or not the attribute can be modified after creation
  - **Searchable:** whether or not the attribute can be filtered against when performing a search
  - **Sortable:** whether or not the attribute can be used to sort a search result

In the context of this **Approval** trait definition, the different attribute definitions represent an approval step, and the below table describes each attribute definition and the property values to assign:

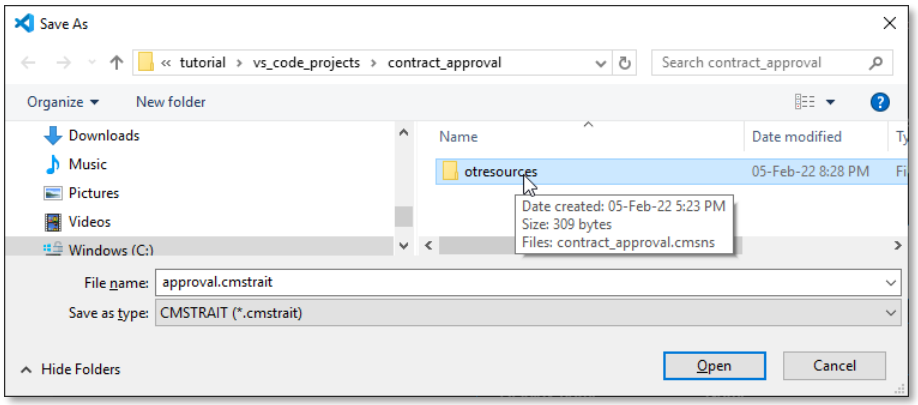
Attribute description	Display name	Name	Data type	Default value	Size	Boolean properties
Whether or not the approval is required	Is required	is_required	boolean			searchable, sortable
Whether or not the approval has been granted	Has been granted	has_been_granted	boolean			searchable, sortable
The email 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

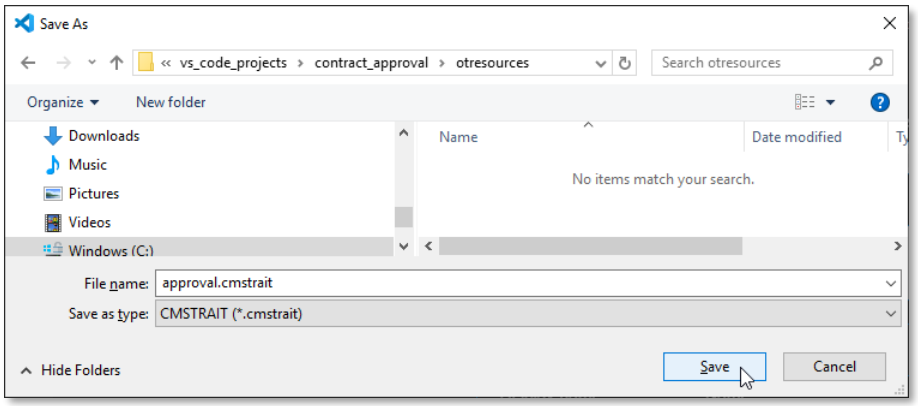
Note that to add an attribute definition to a trait definition, you need to use the ➕ on the top right of the **Attributes** list.



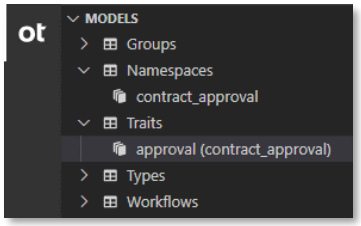
REMARK:
Note that for the current trait definition creation exercise we are using the VS Code Command Palette and the corresponding (**OpenText: New Trait**) model creation command. This is the second way to create models. There is one more way that remains, and this will be illustrated in the next tutorial exercise.

You can now save and close the **Approval** trait definition model. Saving will pop up a **Save As** file saving dialog box. Make sure to select the **otresources** folder as target folder, leave the file name as is (**approval.cmstrait**) and click **Save**.





The model explorer tree in the **MODELS** section should now show your new **approval (contract_approval)** trait definition under **/Traits**. The **contract_approval** value between brackets represents the namespace to which the trait definition belongs, as the model explorer shows the different models according to their unique key (which in context of a trait definition is the combination of the namespace and name properties).



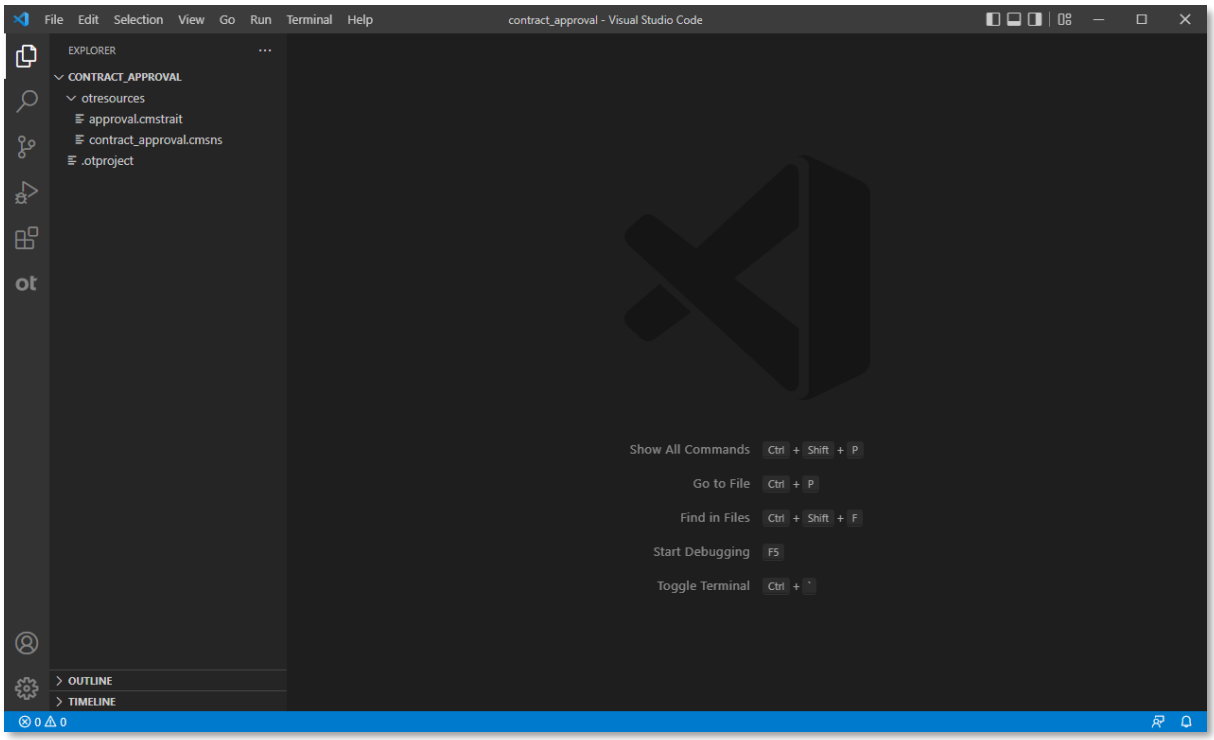
3.6 [20’] Creating a file type definition

During this exercise you will create the **Contract** type definition model. A type definition is the main component for building your application’s (custom) data model. A type definition has its own attributes and required traits (i.e.: traits that are always added to the type instance upon creation). A type definition can also be of several categories: object, file or folder. For the purpose of this tutorial, we will be creating two file type definitions and one folder type definition. This **Contract** type definition is the first **file** type definition, and it will be the parent type of the Loan Contract file type definition that we will create during the next exercise. For more information on types (definitions and instances), you can refer to the 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.

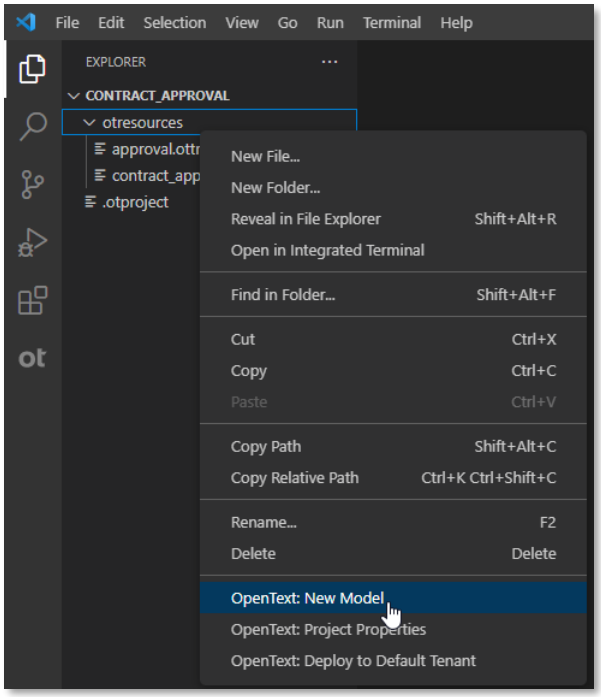
Once you are done with this section, you will have created the **Contract** type definition that corresponds with your Contract Approval application’s (parent) contract file type, and you are ready to start creating the (child) Loan Contract type definition.

To create the **Contract** type definition, proceed as follows:

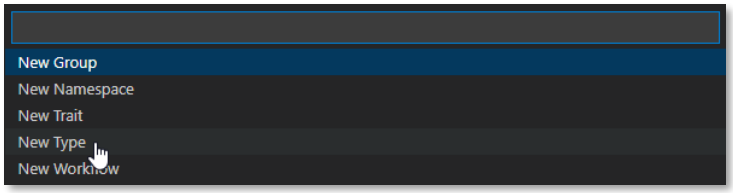
- In VS Code, switch to the **Explorer** view.
  If you expand the **otresources** (model) folder under your **contract_approval** application root folder, you can indeed see the previously created **contract_approval** namespace and **approval** trait.



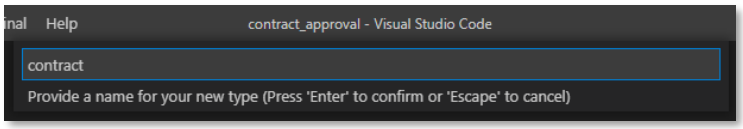
- To create a new type definition, right-click the **otresources** (model) folder and select **OpenText: New Model**.

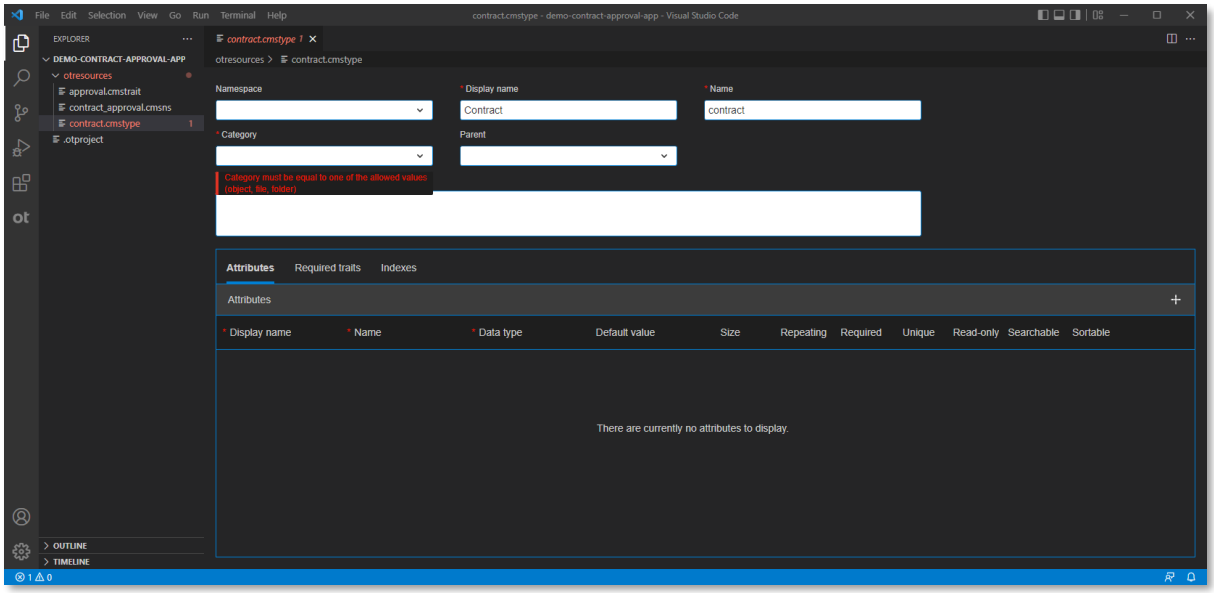


From the menu that drops down from the top of VS Code, select **New Type**.

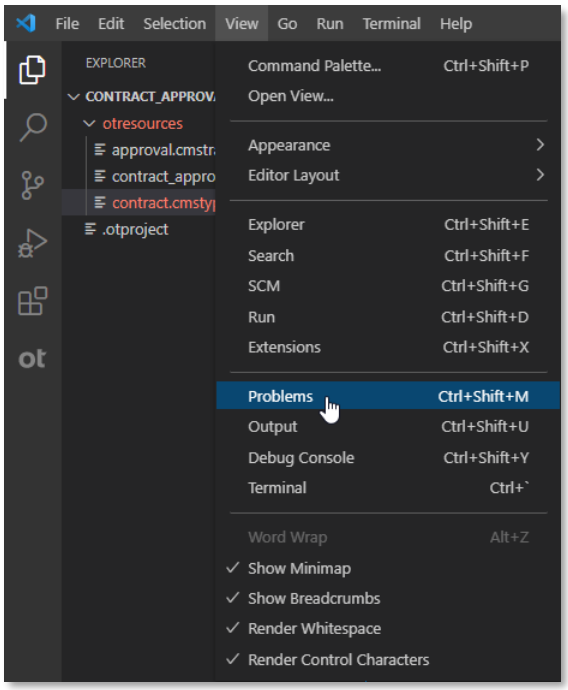


In the input box that now appears at the top of VS Code, type the new type model name for your type definition, **contract**, and press **Enter** to confirm.

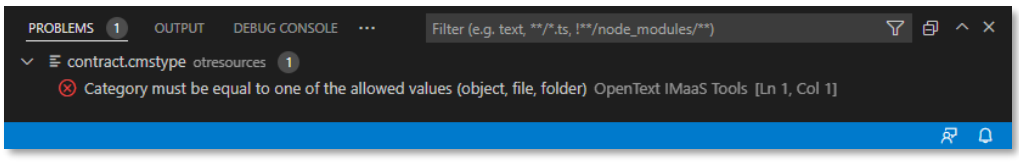




As you can see from the red color of both the file explorer tree and the type definition model's tab, and from the error message displayed right under the **Category** field, there seems to be a problem.
Let's have a look at the **PROBLEMS** tab as this will list and describe the problem/error as well.
Select **Problems** from the **View** menu.



The error message that displays in both the (type) model editor and the **PROBLEMS** tab is "Category must be equal to one of the allowed values (object, file, folder)", and this is indeed correct. You created a new model file under the otresources folder for which all mandatory properties have not yet been properly filled.



Let's now fill the different type definition model properties and after saving the model, the error should disappear.
Fill the type definition properties as follows:

- **Namespace:** the namespace to which this type definition belongs; select the **contract_approval** namespace
- **Display name:** the display name is the user-friendly name for the type definition; this does not have to be unique, and it has been automatically populated for your convenience based on the previously chosen type model file name; you should best leave the value to be **Contract** as it nicely aligns with the model file name and model name
- **Name:** the name is the technical name for the type definition; this has to be unique in context of the selected namespace (and the combination of namespace and type definition name needs to be unique within your developer tenant), and it has been automatically populated for your convenience based on the previously chosen type definition file name; you should best leave the value to be **contract** as it nicely aligns with the model file name and model display name
- **Category:** the type category to which the type definition belongs; this can be object, file or folder; the **Contract** type definition is of the **file** category
- **Parent:** the parent type definition for the type definition you are creating; the **Contract** type has no parent
- **Description: Contract Type**
- **Attributes:** the attributes list defines the different attribute definitions of the type definition; for a description of the attribute definition properties, please refer to the previous exercise
  The below table describes each attribute definition and the property values to assign:

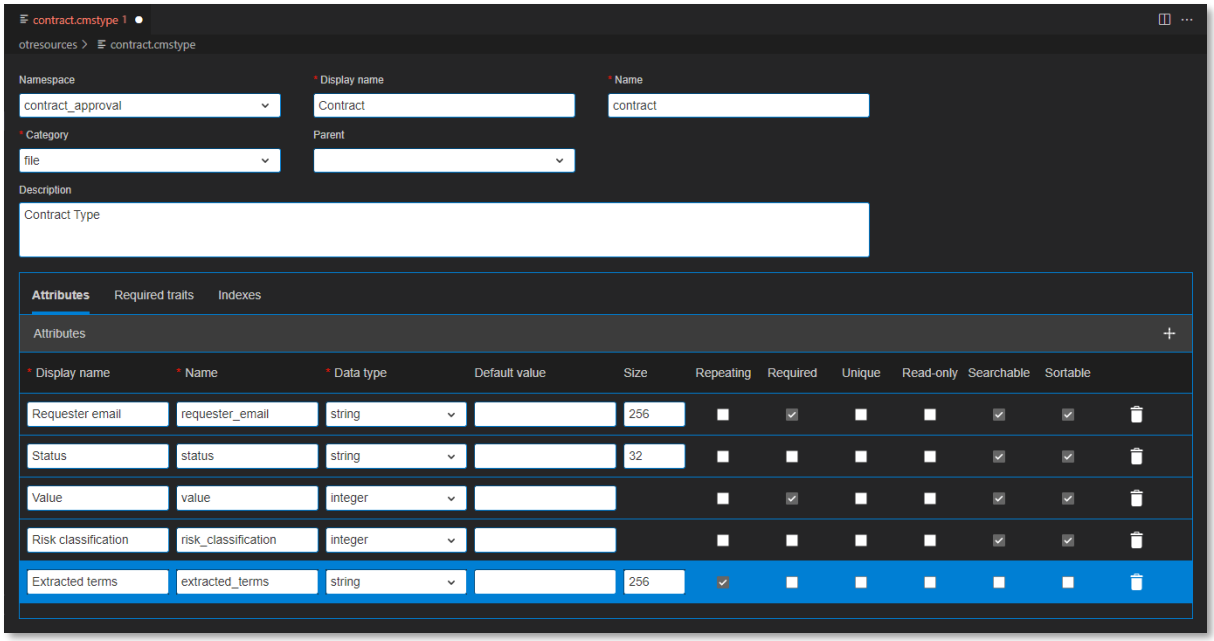
Attribute description	Display name	Name	Data type	Default value	Size	Boolean properties
The email address of the person requesting the approval of the contract	Requester email	requester_email	string		256	required, searchable, sortable
The current (approval) status of the contract	Status	status	string		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

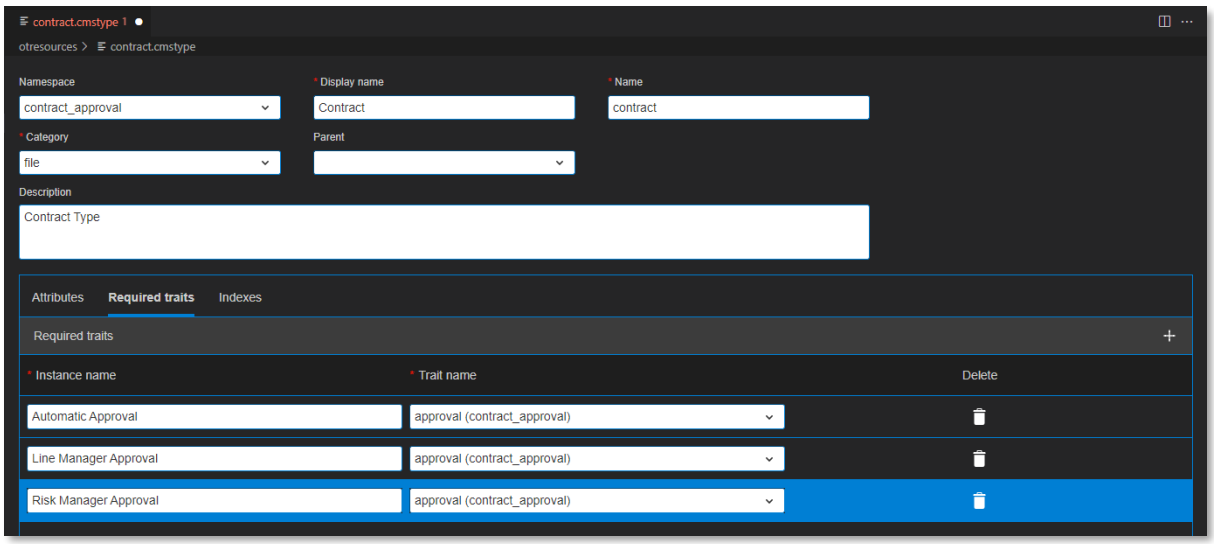
- **Required traits:** the required traits list defines the different mandatory traits for the type definition; each required trait definition has the following properties:
  - **Instance name:** the instance name is the name of the required trait; this must be unique across the type definition's required traits; in context of this tutorial where the required traits are all **Approval** traits, the instance name will represent the type of approval
  - **Trait name:** the trait name is the unique key representing the selected trait definition (i.e.: combination of name and namespace); in the context of this tutorial, all required traits will be **approval (contract_approval)** traits

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

Required trait description	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 (i.e.: 4: HIGH or 5: VERY HIGH).	Risk Manager Approval	approval (contract_approval)

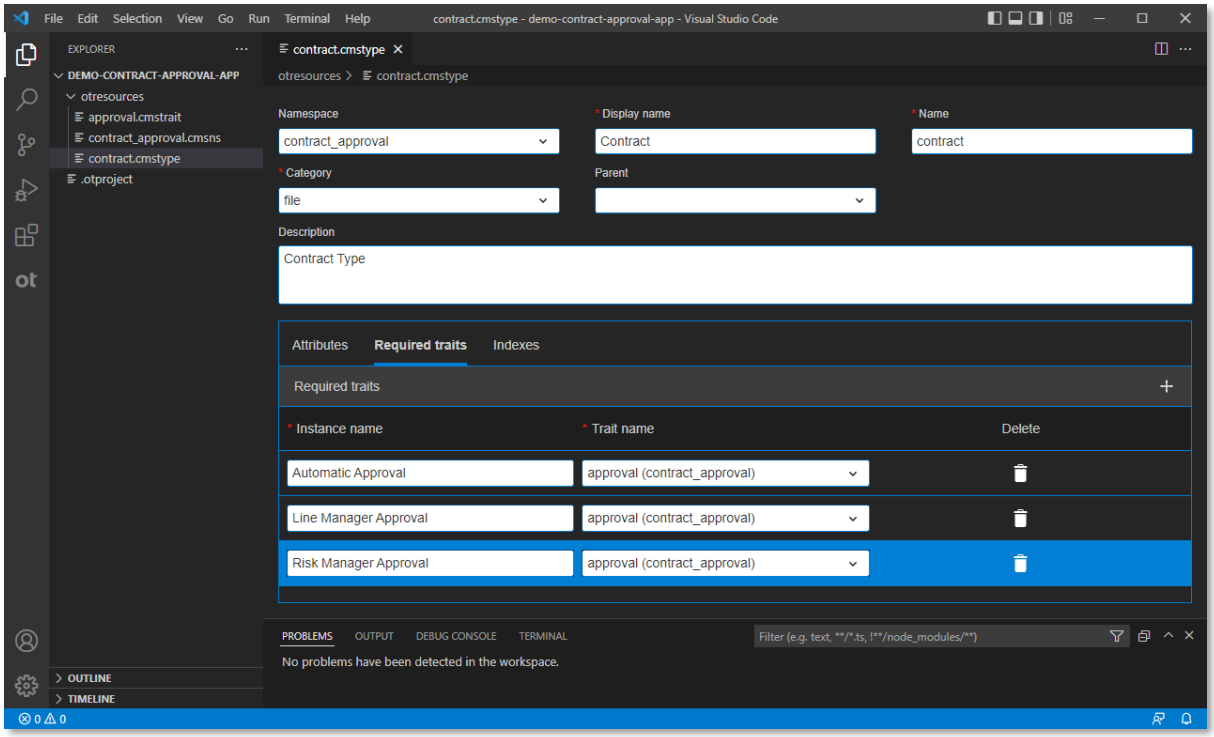
Note that to add a required trait definition to a type definition, you need to use the ➕ on the top right of the **Required traits** list.





REMARK:
Note that for the current type definition creation exercise we are using the **OpenText: New Model** menu item from the contextual menu on the **otresources** folder in the VS Code file explorer tree. This is the third and last way to create models we wanted to illustrate in this tutorial. In the next exercises, feel free to choose whichever of the three methods you prefer.

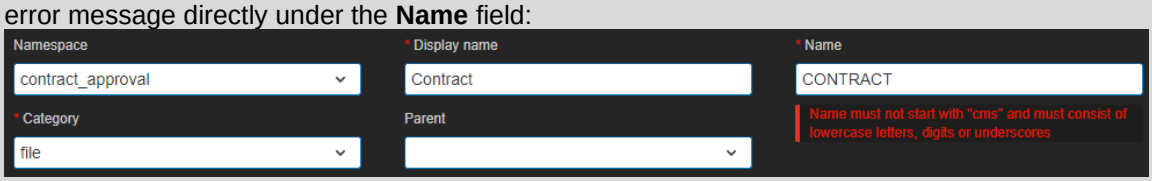
Save the **Contract** type definition model. This time, saving will NOT pop up a **Save As** file saving dialog box because the file already exists in the **otresources** folder. Also, having filled the mandatory properties, the “Category must be equal to one of the allowed values (object, file, folder)” error has now disappeared from the **PROBLEMS** tab and the model editor.



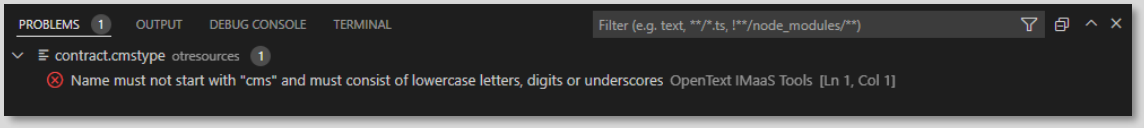
You can now close the **Contract** type definition model.

REMARK:
This is a good time to mention that all models get validated to enforce constraints/rules. So, very much like with the previous example of a type model file having a mandatory Category field, the system enforces any constraints that have been defined for the type of model you are creating. The validation for your model happens at two different moments:

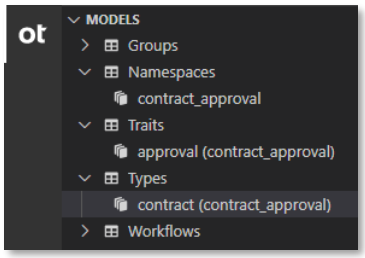
- **When you change a value of a model property.** Any constraint violation resulting from modifying a model property is immediately shown inline in your model editor. Try for example to put some upper-case letters in the **Name** property of your **contract** type model. This will result in showing a “must not start with ‘cms’ and must consist of lowercase letters, digits or underscores” error message directly under the **Name** field:



- **When you save the model file.** This additional validation looks at what is actually saved on disk, and it traps all constraint violations (including broken references) in your model folder (**/otresources**). The errors that reside inside your saved model files are shown in the **PROBLEMS** tab of VS Code. As an additional example, feel free to save your model with the upper-case letters in the **Name** property. You will now also see this error appear in the **PROBLEMS** tab:



If you switch to the **OpenText Cloud Developer Tools** view, the model explorer tree in the **MODELS** section should now show your new **contract (contract_approval)** type definition under **/Types**. The **contract_approval** value between brackets represents the namespace to which the type definition belongs, as the model explorer shows the different models according to their unique key (which in context of a type definition is the combination of the namespace and name properties).



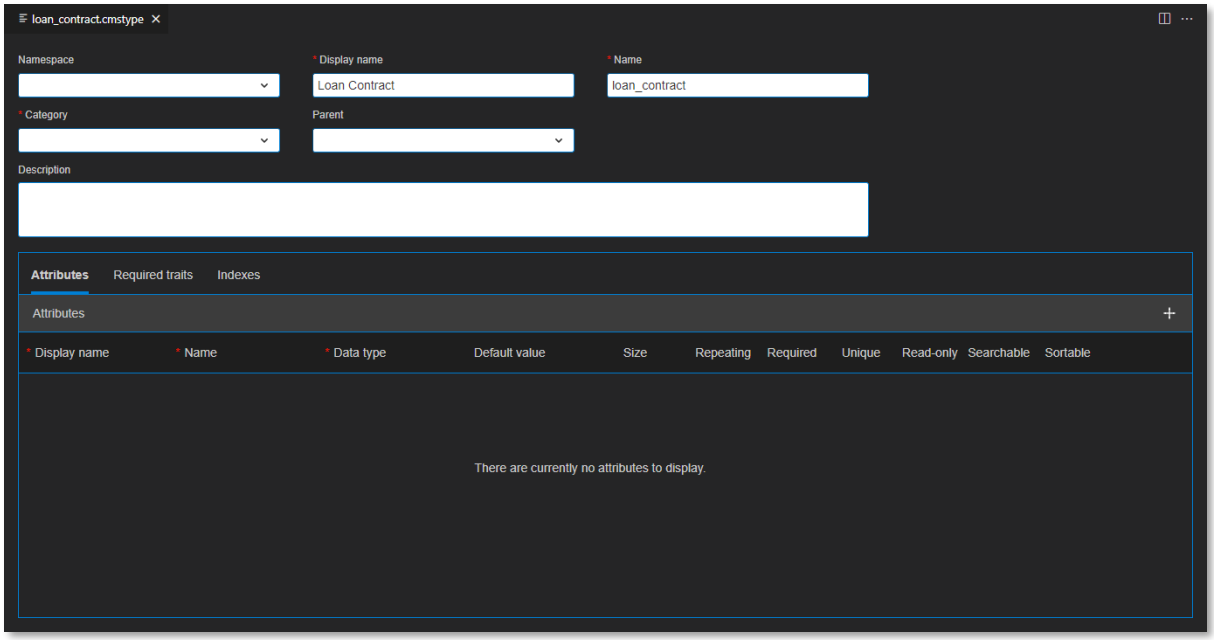
3.7 [10’] Creating a file type definition that is a subtype

During this exercise you will create the **Loan Contract** type definition model. This will be a subtype of the previously created **Contract** type definition (i.e.: it will have **contract** for its **Parent** property). To allow inheriting (i.e.: subtyping) the **Contract** type definition, the **Loan Contract** type definition also has to be of the **file** category.

Once you are done with this section, you will have created the **Loan Contract** type definition that corresponds with your Contract Approval application’s (child) contract file type, and you are ready to start creating the Customer type definition.

To create the **Loan Contract** type definition, proceed as follows:

- In VS Code, using one of the three previously explained ways of creating a model, create a new type definition and name it **loan_contract** when asked to enter the model (file) name.



  Fill the type definition properties as follows:
  - **Namespace: contract_approval**
  - **Display name: Loan Contract**
  - **Name: loan_contract**
  - **Category: file**
  - **Parent:** the parent type definition for the type definition you are creating; choose **contract (contract_approval)** as parent for the **Loan Contract** type

    REMARK:
    Note that the **contract (contract_approval)** parent type is only available to be selected from the **Parent** drop-down list when the **file Category** is selected. If you empty the **Category** value or select **folder**, you will not be able to select **contract (contract_approval)** as it is indeed a file type definition.

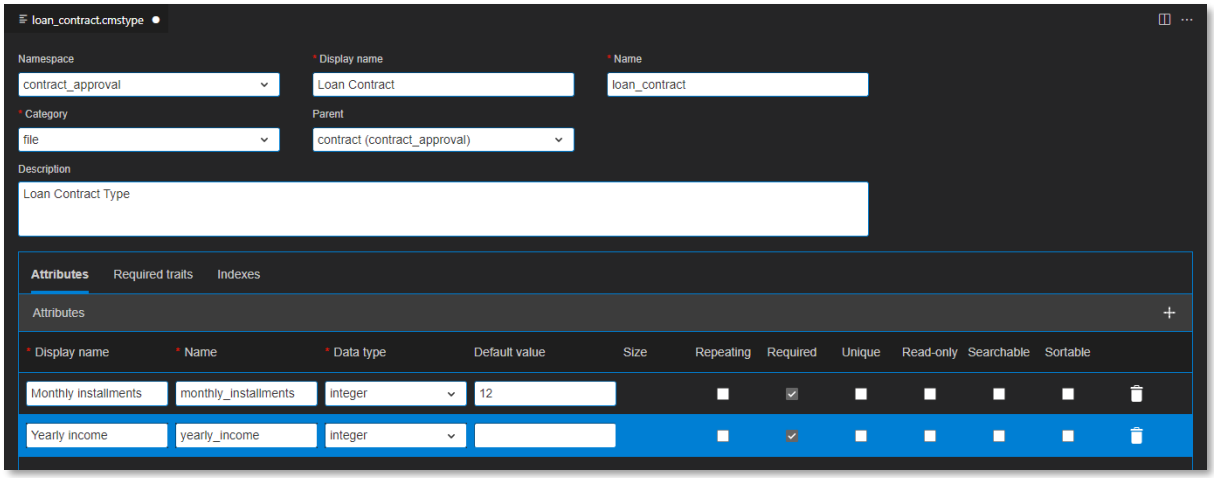
  - **Description: Loan Contract Type**

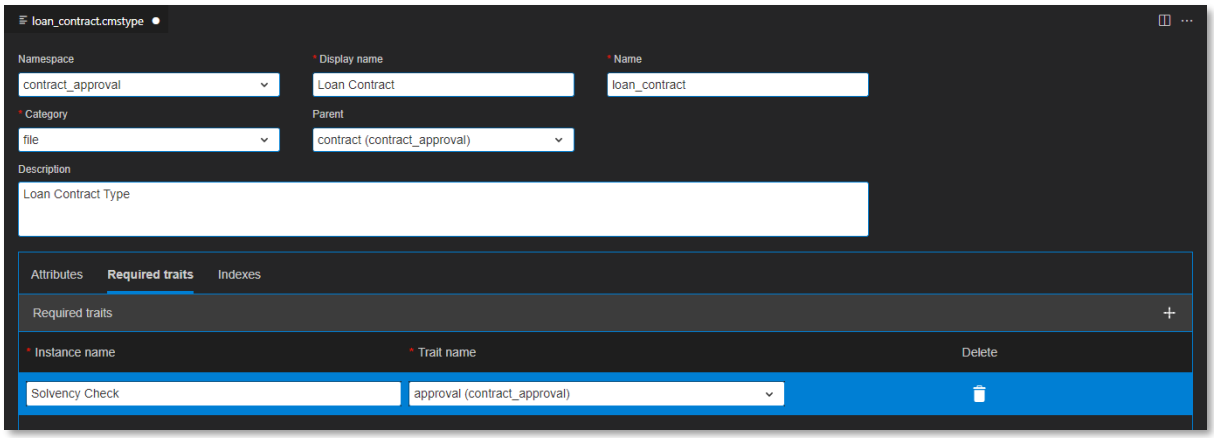
- **Attributes:** each attribute definition and the property values to assign is described in the table below

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:** each required trait definition and the property values to assign is described in the table below

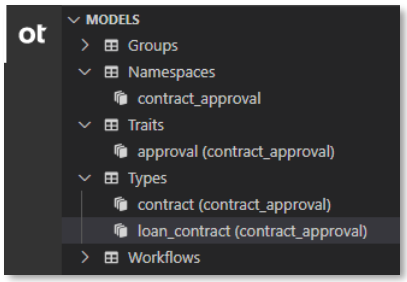
Required trait description	Instance name	Trait name
The automated approval step that checks whether or not the customer is solvent by checking that the monthly cost doesn't exceed 25% of the monthly income (calculated from the loan contract cost, the total count of monthly payments and the yearly income)	Solvency Check	approval (contract_approval)





REMARK:
Although we only assigned two attributes and one required trait to the **Loan Contract** type definition, being a subtype of the **Contract** type definition implies that all the attributes and required traits of that type definition are also present on the **Loan Contract** type definition.

You can now save and close the **Loan Contract** type definition model.
If you switch to the **OpenText Cloud Developer Tools** view, the model explorer tree in the **MODELS** section should now show your new **loan_contract (contract_approval)** type definition under **/Types**.



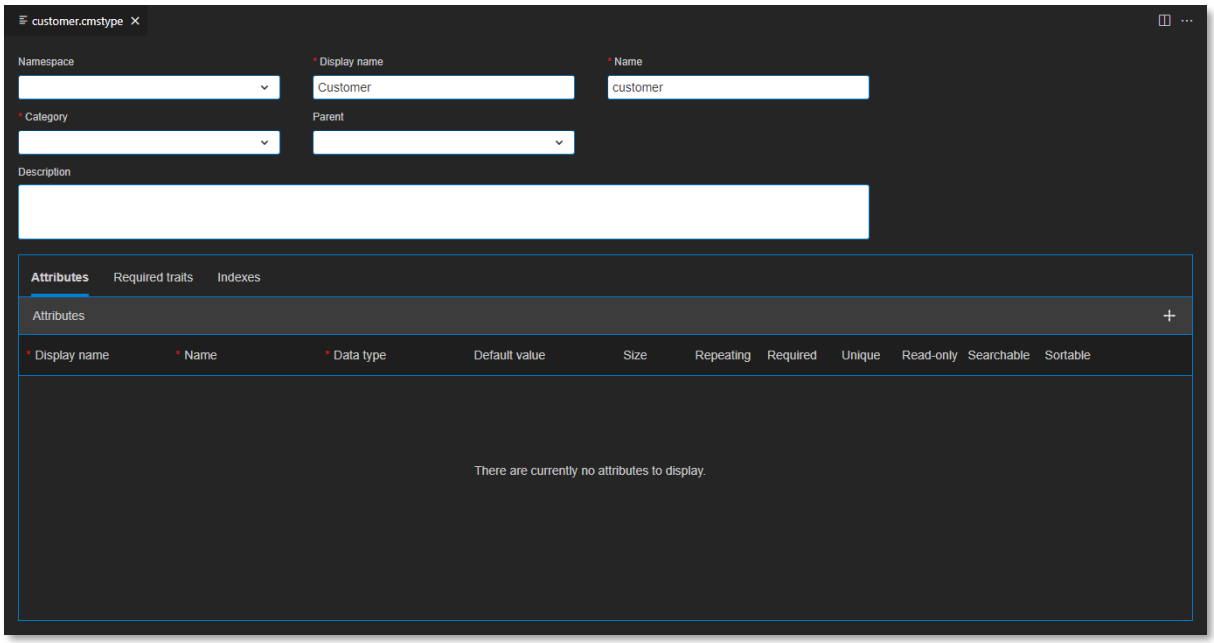
3.8 [05’] Creating a folder type definition

During this exercise you will create the **Customer** type definition model. This will be a type definition of the **folder** category as it represents a customer, and it will contain (as a folder) the different contracts of file type **Contract** or **Loan Contract** related to that customer.

Once you are done with this section, you will have created the **Customer** type definition that corresponds with your Contract Approval application’s customer folder, and you are ready to start creating the Contract Approval workflow model.

To create the **Customer** type definition, proceed as follows:

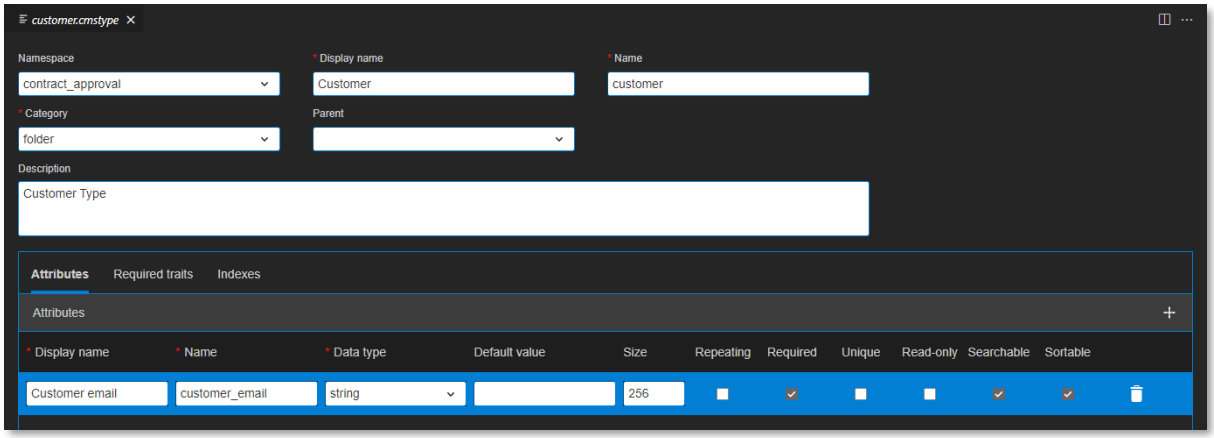
- In VS Code, using one of the three previously explained ways of creating a model, create a new type definition and name it **customer** when asked to enter the model (file) name.



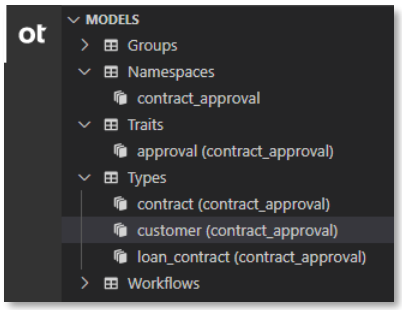
  Fill the type definition properties as follows:
  - **Namespace: contract_approval**
  - **Display name: Customer**
  - **Name: customer**
  - **Category: folder**
  - **Parent:** the **Customer** type has no parent
  - **Description: Customer Type**
  - **Attributes:** each attribute definition and the property values to assign is described in the table below

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

  - **Required traits:** the **Customer** type has no required traits



You can now save and close the **Customer** type definition model.
If you switch to the **OpenText Cloud Developer Tools** view, the model explorer tree in the **MODELS** section should now show your new **customer (contract_approval)** type definition under **/Types**.



3.9 [120'] Creating a workflow model

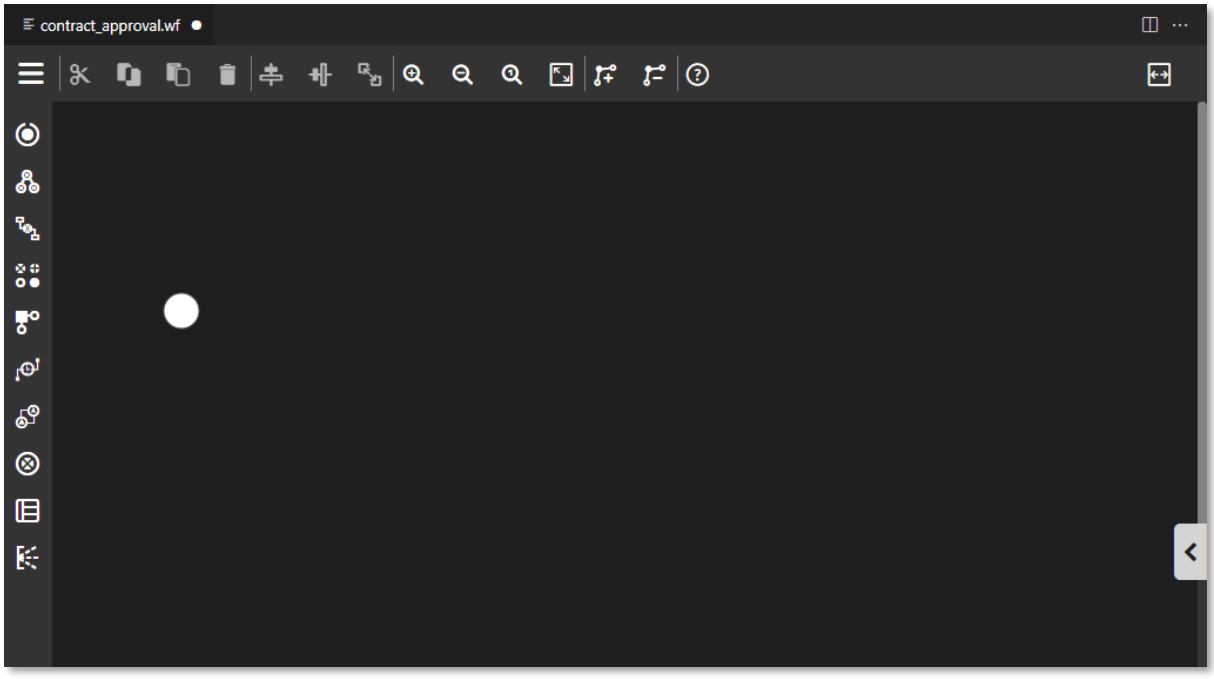
During this exercise you will create the **Contract Approval** workflow model. A workflow model represents an executable process model from which process instances will be created. The executable process model is stored as BPMN 2.0 encoded JSON. This **Contract Approval** workflow model will automate the contract approval process of your Contract Approval application. It consists of a number of automated and manual approval tasks. Not all approval tasks are required. They are conditional, based on the type, value, and risk of the contract. Throughout this exercise we will go into detail on how to configure the different events, activities, and conditional gateways (choices) of the **Contract Approval** workflow model. For more information on Workflow Service process models and process instances, you can refer to the Workflow Service product documentation , the Workflow Modeler product documentation or the Workflow Service API reference.

Once you are done with this section, you will have created the **Contract Approval** workflow model that corresponds with your Contract Approval application's approval process, and as this is the last model for your application, you are ready to deploy your **Contract Approval** application project (i.e.: its models) to the different IM services.

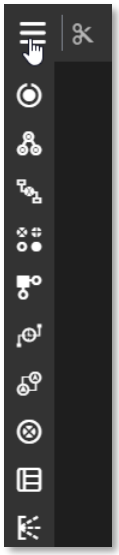
REMARK:
We certainly recommend you go through this entire exercise and build the workflow yourself. However, inherent to the complexity of the workflow model you are going to build is that there could be issues that require troubleshooting, and which you might have trouble fixing. To help with that, whether it is to compare your workflow model with a working version or just to use an existing working version instead of the one you built, you can download the finished Contract Approval application here. You will find the **contract_approval.wf** workflow under the **otresources** folder.

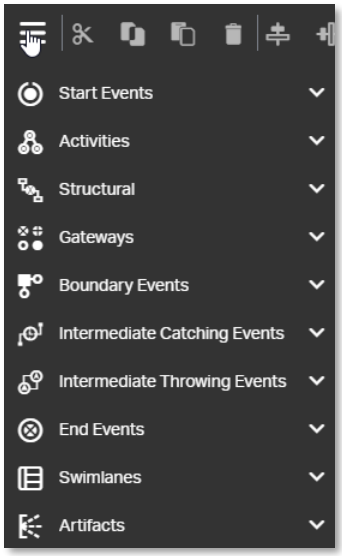
To create the **Contract Approval** workflow model, proceed as follows:

- In VS Code, using one of the three previously explained ways of creating a model, create a new workflow and name it **contract_approval** when asked to enter the (file) name. If you are using **New File** to create the new workflow, note that you should assign the "**.wf**" extension for the system to open the workflow model editor.

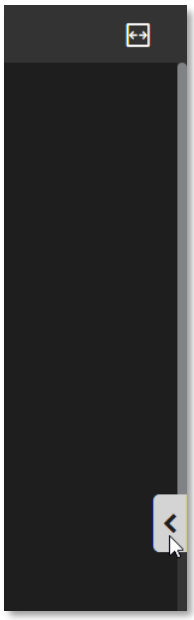


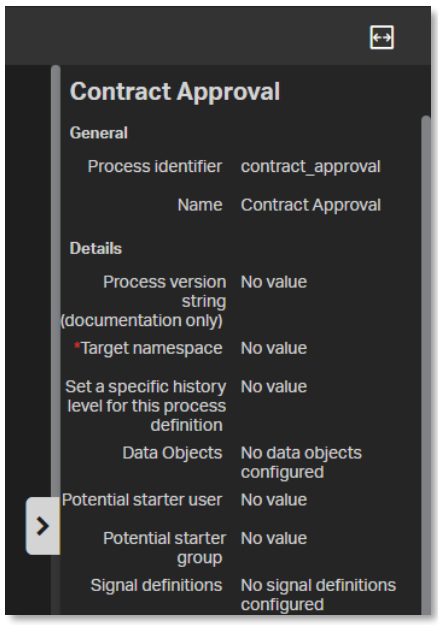
- Compared to the previously used metadata model editors, the workflow model editor is much more elaborate. It is intended to build entire business process definitions with all the complexity this brings. Throughout this exercise we will step by step build the **Contract Approval** workflow, and this will indeed illustrate this much higher degree of complexity.
  However, before getting started with building the actual workflow, we should first have a look at the model editor itself to understand its different features.
  Let's first expand the two collapsed side panes. On the top left of the editor, click the button to expand the **palette** (i.e.: the left pane).





  Now, click the button on the middle right of the editor to expand the **attribute bar** (i.e.: the right pane).

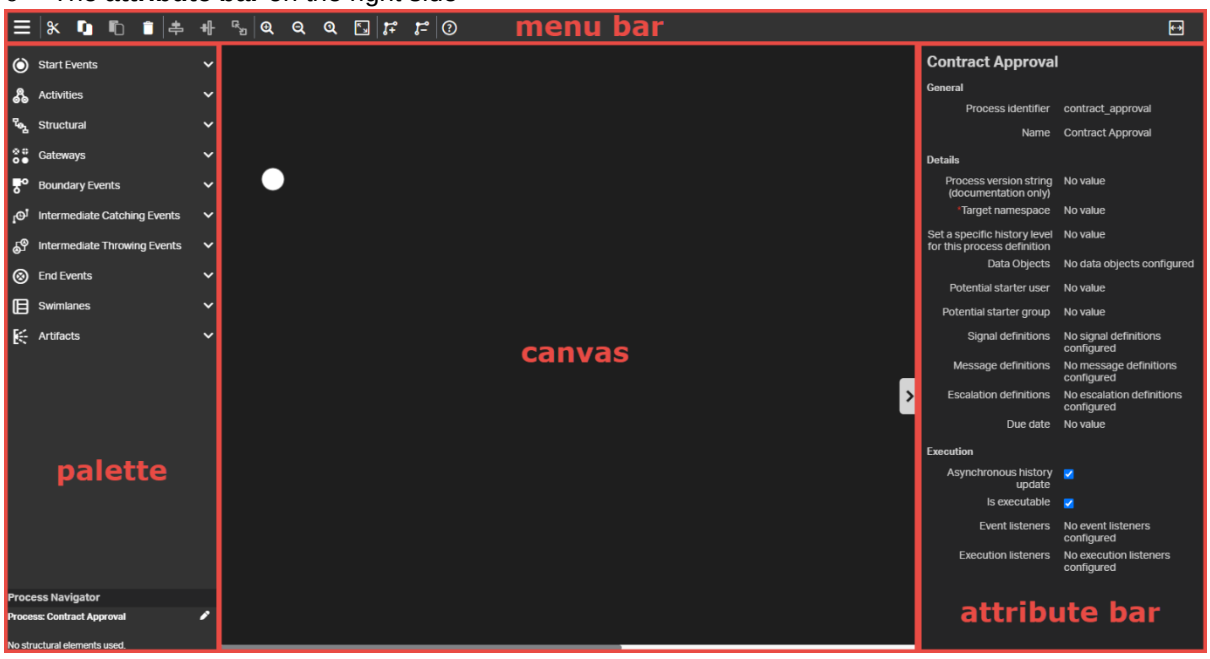




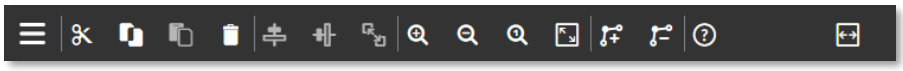
  Note that the button on the top right allows to expand or collapse both side panes at once.

As you can now see, the workflow editor has four areas:

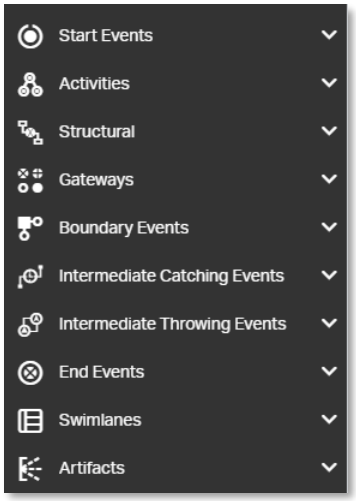
- The **menu bar** on top
- The **palette** on the left side
- The **canvas** in the middle
- The **attribute bar** on the right side



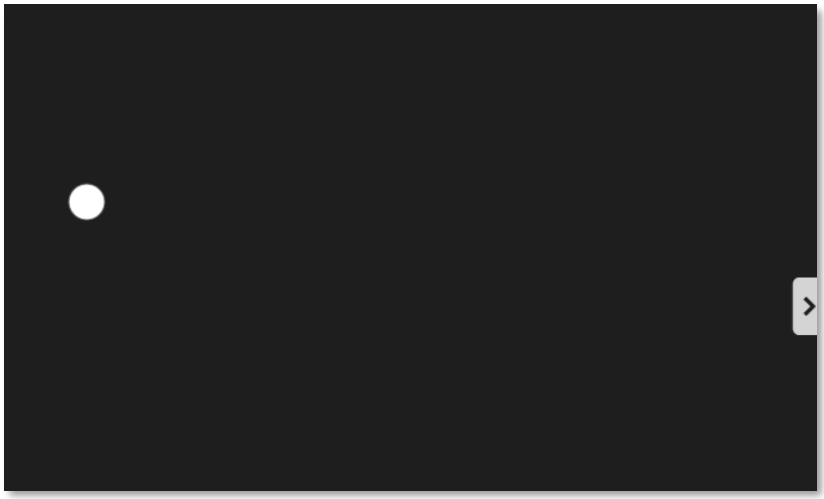
The **menu bar** contains the generic capabilities/buttons (e.g.: copy/paste, delete, align, zoom, help, etc.).



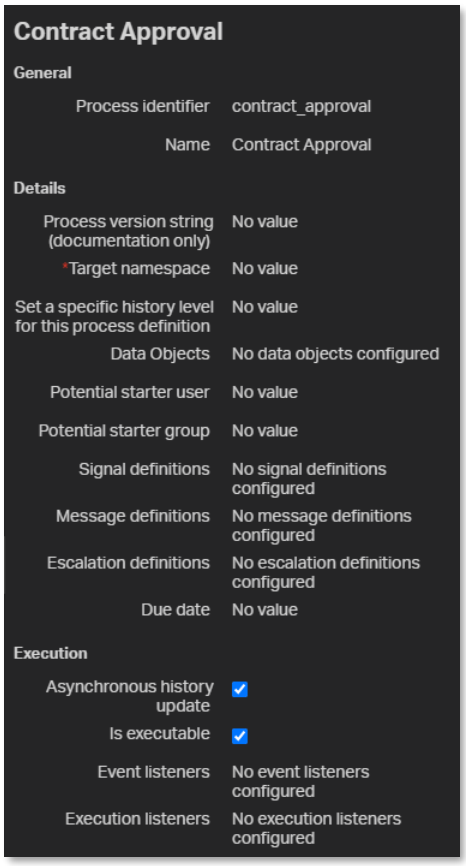
The **palette** contains the different workflow elements, which you can drag and drop on the canvas to build your workflow model.



The **canvas** is the area where you will be building/drawing your workflow model.



The **attribute bar** displays the attributes of the currently selected element. At the moment, no element is selected (which is the same as clicking on an empty area on the canvas), so the workflow model's attributes are displayed.

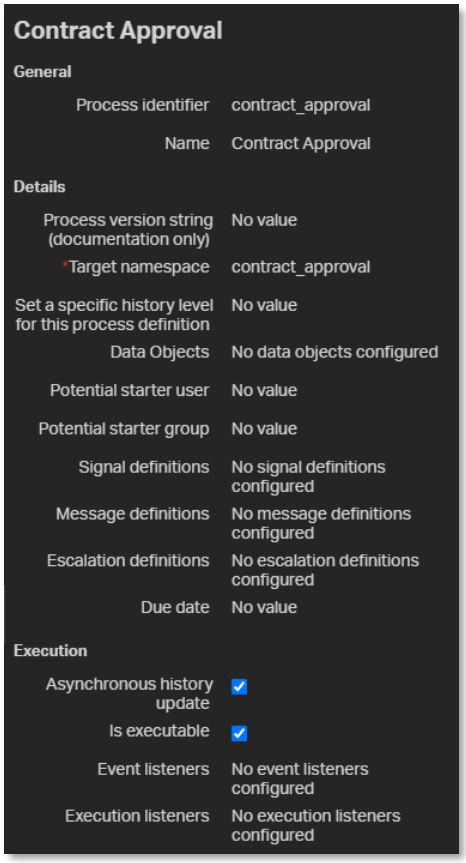


- Now that you have an understanding of the different areas of the workflow model editor, let's start building our workflow model.

REMARK:
Although we will cover many features of the workflow model editor in this tutorial, providing you with a good and practical understanding of how to build business process definitions, we will not cover every single feature and capability. This means we will only use a subset of the buttons on the **menu bar** and the workflow elements in the **palette**. We will also not describe all attributes in the **attribute bar**.
Once you understand how to build workflow models, the Workflow Modeler product documentation (also available from the button on the menu bar) provides you a more exhaustive and in depth explanation of the functionality and usage of the workflow model editor (or workflow modeler).

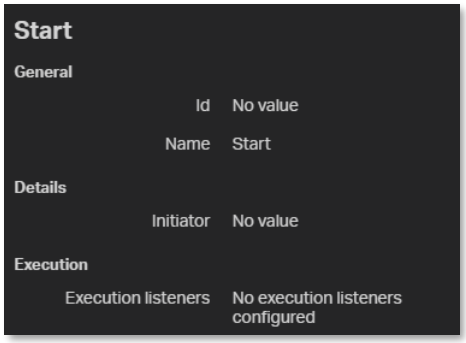
The first thing we will look at is filling the workflow model's own attributes, visible from the attribute bar when you have no element selected (i.e.: you click an empty area on the canvas to ensure the entire workflow model is selected). As per the above remark, we will only explain those attributes we consider relevant in context of this tutorial. Feel free to open the *Processes* section of the Workflow Modeler product documentation for more details on the different process model attributes.
Fill the workflow model attributes as follows:

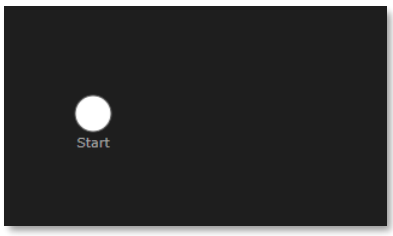
- **Process identifier:** the process identifier is the technical name for the workflow model; this has to be unique (within your developer tenant), and it has been automatically populated for your convenience based on the previously chosen workflow name (the model file name); you should best leave the value to be **contract_approval** as it nicely aligns with the model file name and model name
- **Name:** the name is the user-friendly name for the workflow model; this does not have to be unique, and it has been automatically populated for your convenience based on the previously chosen workflow name (the model file name); you should best leave the value to be **Contract Approval** as it nicely aligns with the model file name and model process identifier
- **Target namespace:** The target namespace allows grouping the different workflow models together; you can fill **contract_approval** as value
- All other workflow model attributes can remain unchanged



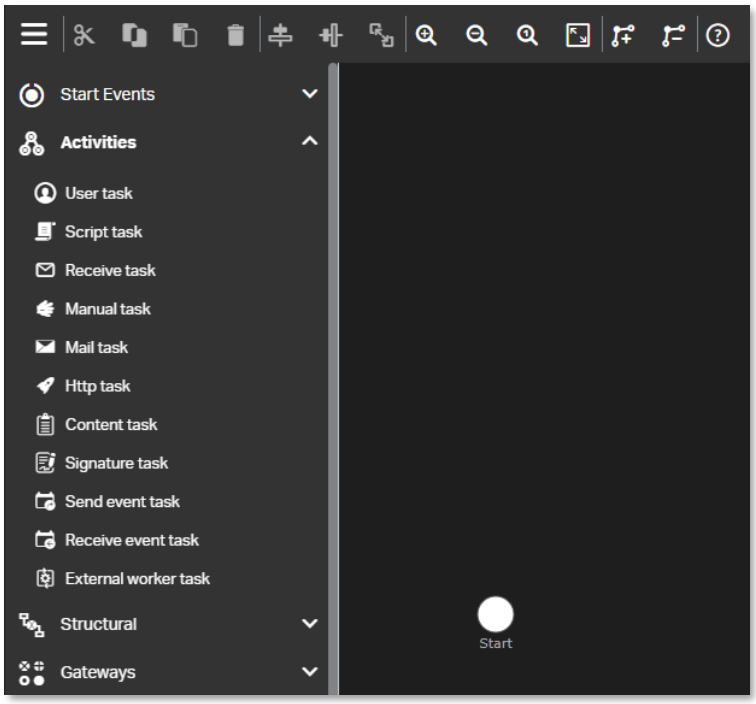
Since building a workflow model contains many steps, it is recommended that you regularly save your workflow model to avoid losing work. You can now save the workflow model.

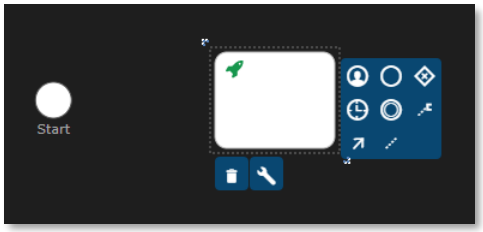
- Now that the workflow model attributes have been set, we can start building the actual business process definition by drawing it on the canvas.
  The first element to set is the start event. As this element is already on the canvas, we just need to set the attributes.
  Select the **start event** element (on the canvas) and set the attributes as follows:
  - **Name: Start**
  - All other element attributes can remain unchanged





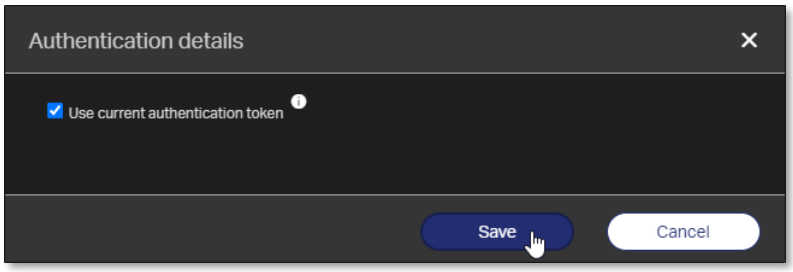
- The next element we are going to add is an http task (i.e.: REST API call). This task will perform a GET request to fetch the JSON object that holds the metadata of the contract to approve. This contract JSON object can then be used throughout the subsequent steps of the business process.
  To add the new http task, expand the **Activities** section from the palette and drag and drop an **Http task** to the right of the **Start event**.





Select the **http task** element and set the element attributes as follows:

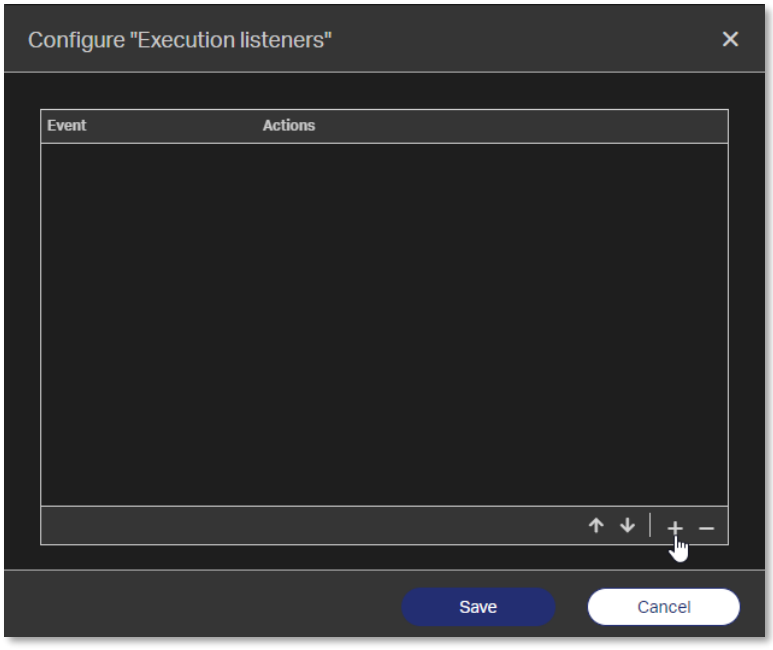
- **Name: Get contract from CMS**
- **Authentication details: Use current authentication token**

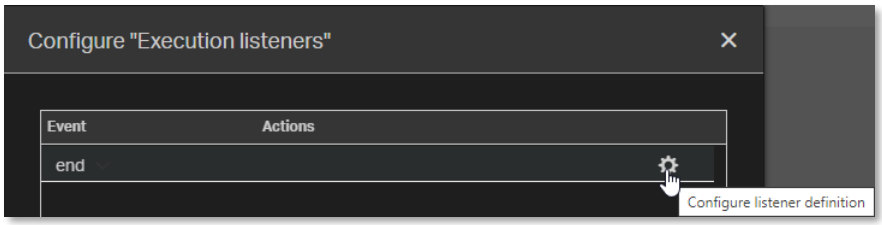


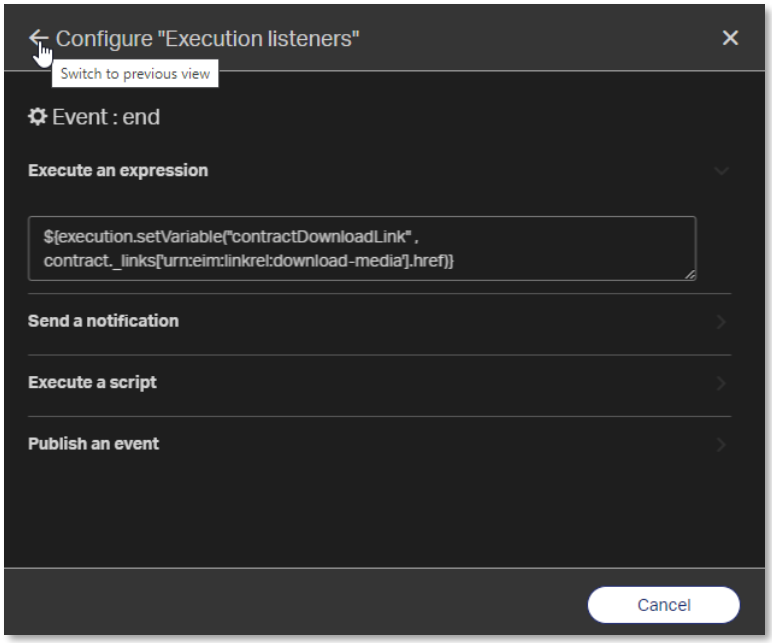
- **Request method: GET**
- **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  => The 'base_url' parameter is referring to the base URL of the IM services for your trial account
  => The 'contract_id' parameter is referring to the unique identifier of the contract to approve (so that you can use it to fetch contract information)
  Both the 'base-url' and 'contract_id' are being passed to the workflow when initializing it from your application's code (this will be explained/shown later, under exercise Building the application).

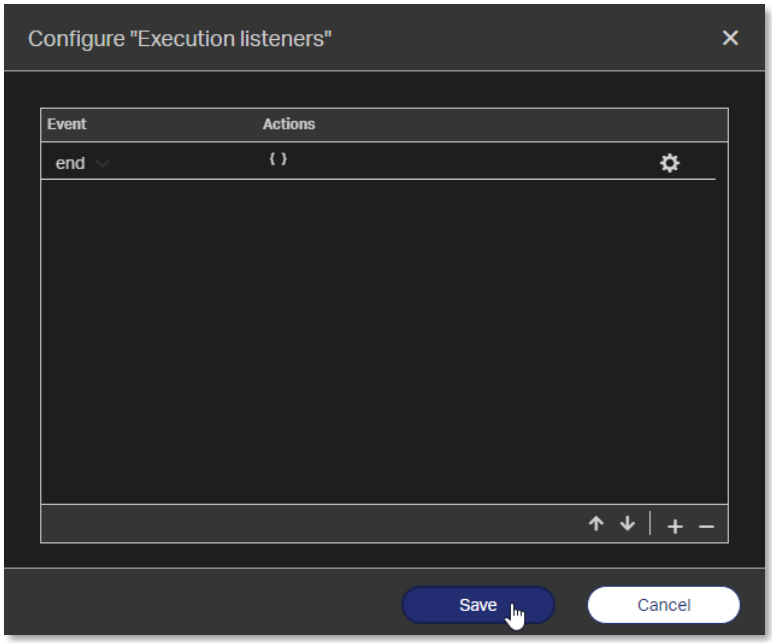
REMARK:
In the request URL you can see we are using **${}** to pass parameters. This is the standard mechanism to pass process variables as (string) parameters to expressions.

- **Response variable name: contract**
- **Save response as JSON: true**
- **Exclusive: true**
- **Execution listeners:**
  - **Event: end**
  - **Execute an expression: ${execution.setVariable("contractDownloadLink" , contract._links['urn:eim:linkrel:download-media'].href)}**

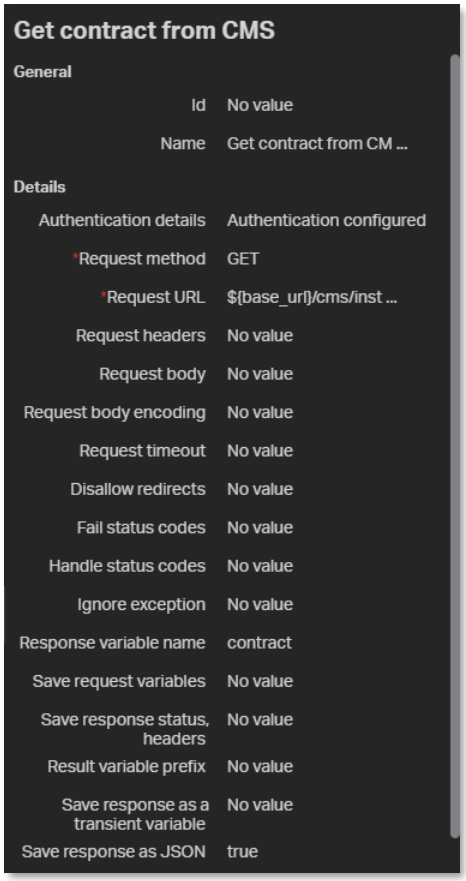


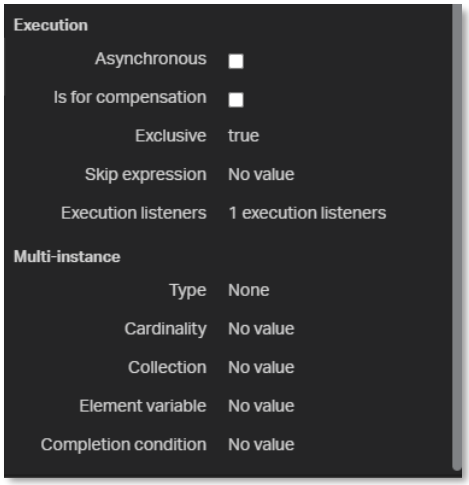


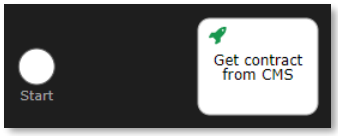




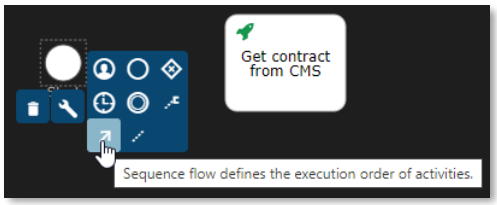
- All other element attributes can remain unchanged

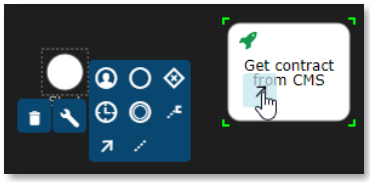






Drag a **sequence flow** (arrow) connector from the **Start event** element to the **Get contract from CMS http task** element.

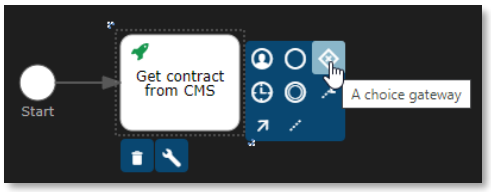


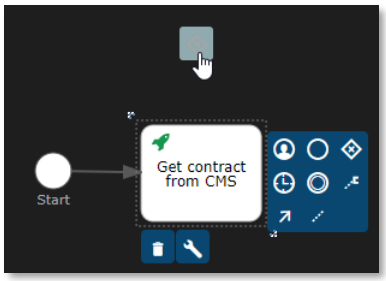


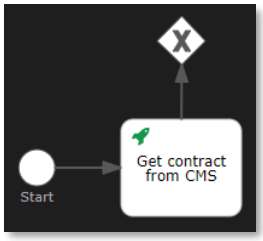
There is no need to set any attributes for the sequence flow connector you just added.

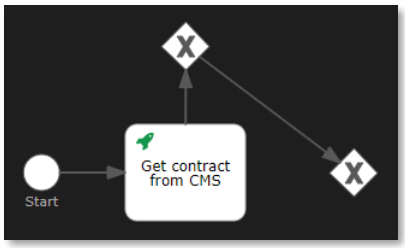
- Let's now add three exclusive (or choice) gateways to, based upon certain conditions, route the business process execution in one of two directions. The three exclusive gateways we are going to add are going to have the following behavior:
  - Take a different route based on whether the contract is a **Standard contract** (of type **contract**) or a **Loan Contract** (of type **loan_contract**)
  - Take a different route based on whether the **value** (attribute) of the contract is **below or equal to 1000** or **above 1000**
  - Take a different route based on whether the **risk_classification** (attribute) is **MEDIUM (3) or lower** or **HIGH (4) or higher**

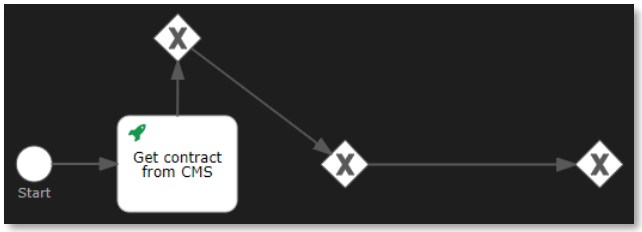
  To create the three exclusive gateways, drag the **Exclusive gateway** element from the previous element (i.e.: for the first exclusive gateway, this is the **Get contract from CMS http task** and for the two subsequent exclusive gateways, it is the **Exclusive gateway** element you created before).



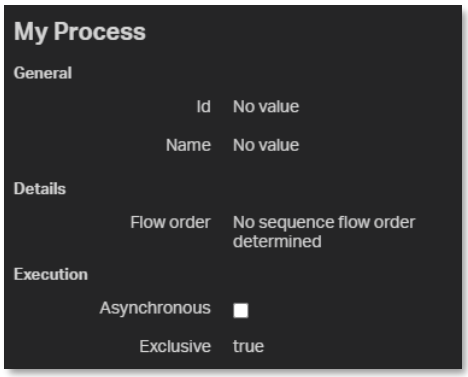






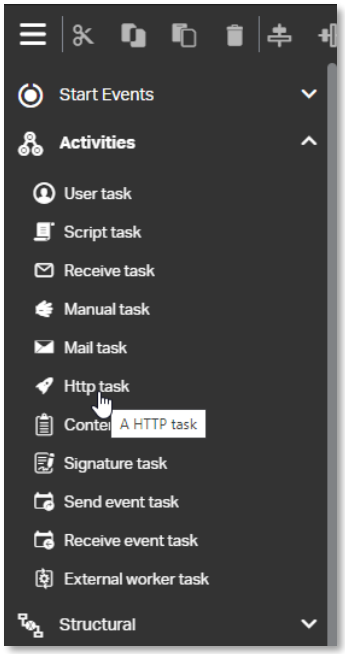


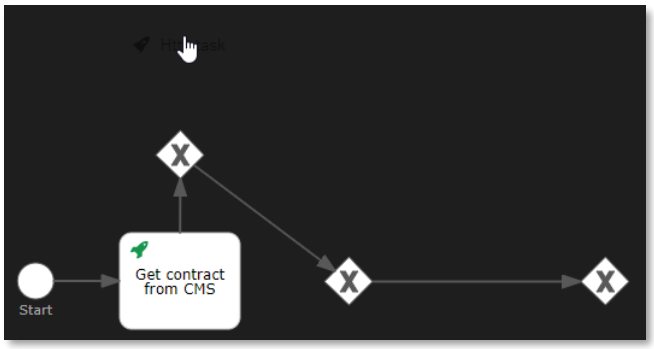
For all three exclusive gateway elements you just created, set the **Exclusive** attribute to **true**:

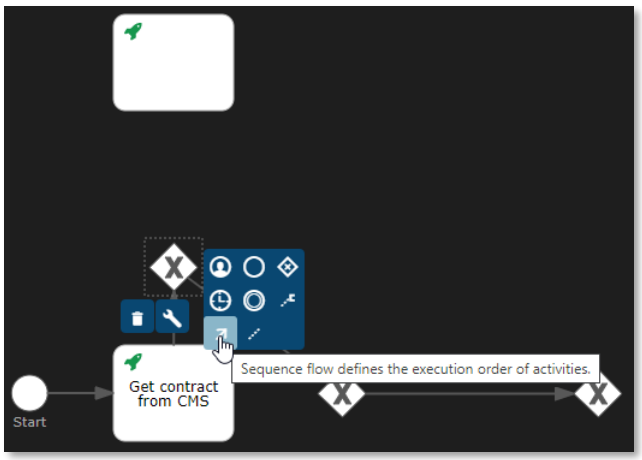


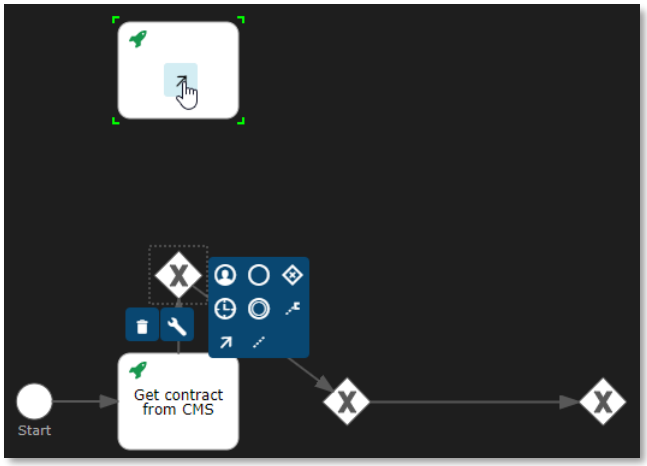
- The next step is to add all activities that correspond with the different choices of the exclusive gateways. More specifically, for the first and the second exclusive gateway, we already have one sequence flow connector representing the (first) choice that moves the business process to the next exclusive gateway. Let's now add the second choice for the first and second exclusive gateways, and both choices for the last gateway. To explain what those choices are, let's revisit the logic of the exclusive gateways and add the activity/behavior for each choice:
  - First exclusive gateway:
    - In case the contract is a **Standard contract** (of type **contract**), the process can advance to the second gateway
    - In case the contract is a **Loan Contract** (of type **loan_contract**), the solvency (i.e.: the ability to pay back the loan) of the customer needs to be checked, and the process has to advance to a new **Http task** that updates (i.e.: **PATCH** request) the status of the contract to **SOLVENCY CHECK**
  - Second exclusive gateway:
    - In case the **value** (attribute) of the contract is **below or equal to 1000**, the process can advance to the third gateway
    - In case the **value** (attribute) of the contract is **above 1000**, the contract needs to be (manually) approved by a line manager, and the process has to advance to a new **Http task** that updates (i.e.: **PATCH** request) the status of the contract to **LINE MANAGER APPROVAL**
  - Third exclusive gateway:
    - In case the **risk_classification** (attribute) is **MEDIUM (3) or lower**, the process can advance to the (always required) automatic approval, and the process can advance to a new **Http task** that updates (i.e.: **PATCH** request) the status of the contract to **APPROVED**
    - In case the **risk_classification** (attribute) is **HIGH (4) or higher**, the contract needs to be (manually) approved by a risk manager, and the process has to advance to a new **Http task** that updates (i.e.: **PATCH** request) the status of the contract to **RISK MANAGER APPROVAL**

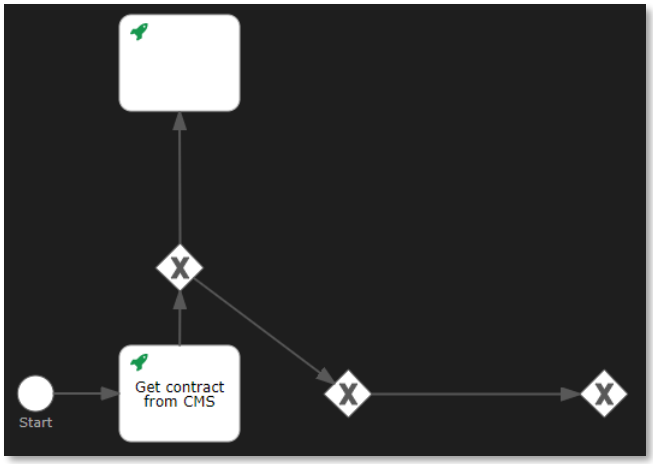
To add the four new (PATCH request) http tasks that update the status of the contract, drag and drop the **Http task** elements from the palette (under the **Activities** section) to the canvas, and connect the three exclusive gateways with these new http tasks using **sequence flow** (arrow) connectors.

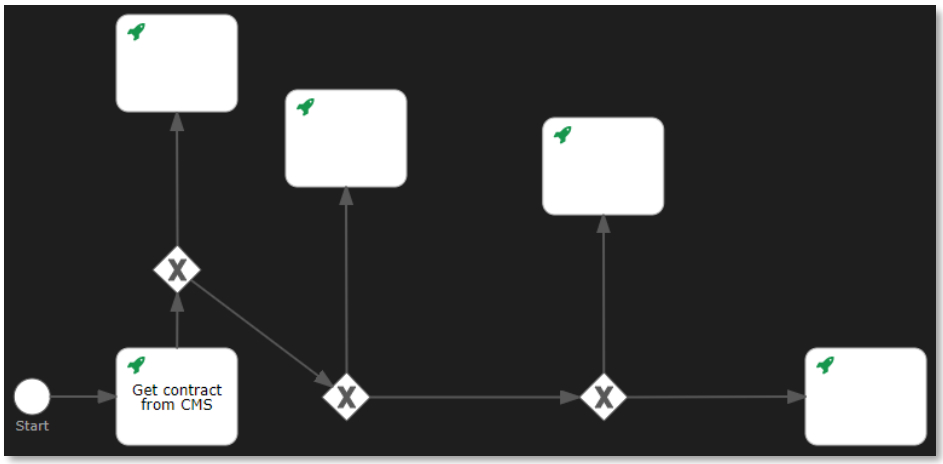








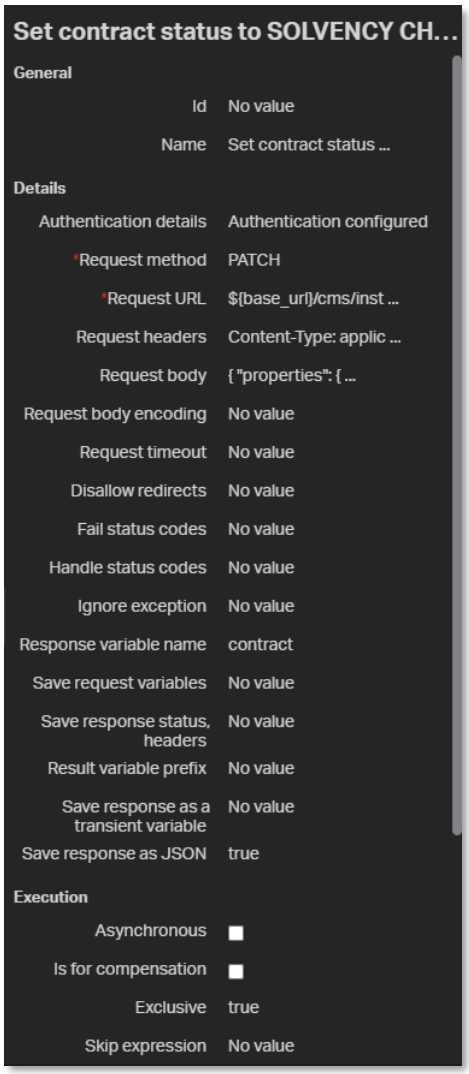




Now that you have created the four different http tasks that update the status of the contract, you can set their respective attributes. From left to right (on the previous picture), please fill the http task attributes as follows:

- Http task to set contract status to SOLVENCY CHECK:
  - **Name: Set contract status to SOLVENCY CHECK**
  - **Authentication details: Use current authentication token**
  - **Request method: PATCH**
  - **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  - **Request headers: Content-Type: application/json**
  - **Request body:**

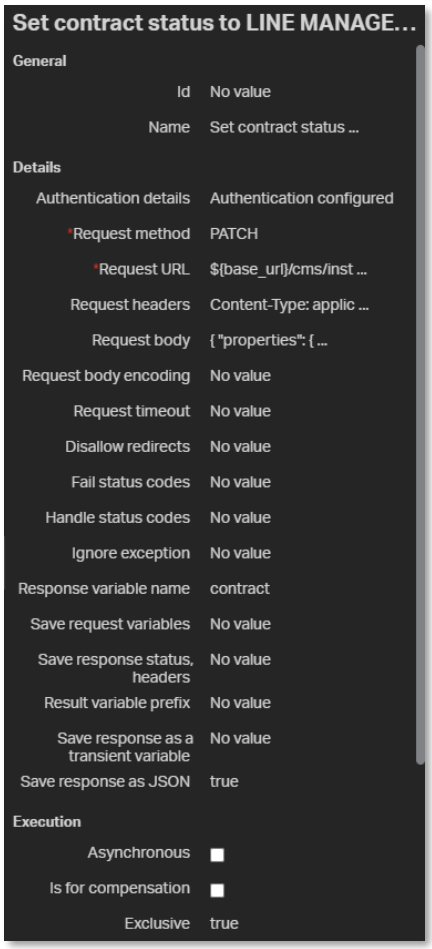
    {
    "properties": {
    "status": "SOLVENCY CHECK"
    }
    }
  - **Response variable name: contract**
  - **Save response as JSON: true**
  - **Exclusive: true**



- Http task to set contract status to LINE MANAGER APPROVAL:
  - **Name: Set contract status to LINE MANAGER APPROVAL**
  - **Authentication details: Use current authentication token**
  - **Request method: PATCH**
  - **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  - **Request headers: Content-Type: application/json**
  - **Request body:**

```
{
 "properties": {
  "status": "LINE MANAGER APPROVAL"
 },
 "traits": {
  "ca_approval": {
   "Line Manager Approval": {
    "is_required": true,
    "has_been_granted": false,
    "approver": "${contract.updated_by.email}",
    "approver_role": "Line Manager"
   }
  }
 }
}
```

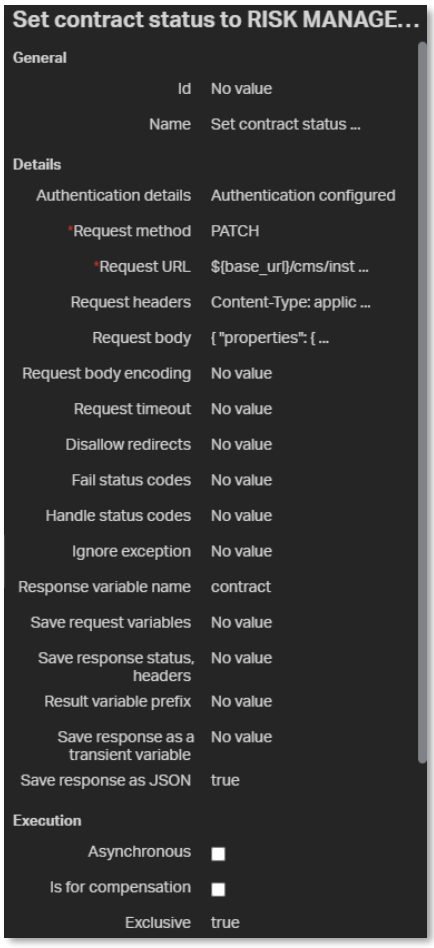
  - **Response variable name: contract**
  - **Save response as JSON: true**
  - **Exclusive: true**



- Http task to set contract status to RISK MANAGER APPROVAL:
  - **Name: Set contract status to RISK MANAGER APPROVAL**
  - **Authentication details: Use current authentication token**
  - **Request method: PATCH**
  - **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  - **Request headers: Content-Type: application/json**
  - **Request body:**

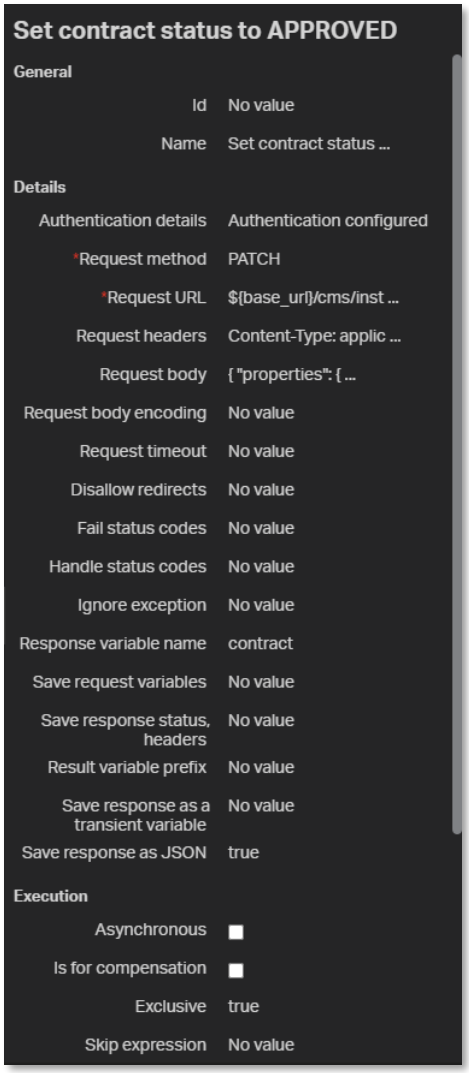
```
{
 "properties": {
  "status": "RISK MANAGER APPROVAL"
 },
 "traits": {
  "ca_approval": {
   "Risk Manager Approval": {
    "is_required": true,
    "has_been_granted": false,
    "approver": "${contract.updated_by.email}",
    "approver_role": "Risk Manager"
   }
  }
 }
}
```

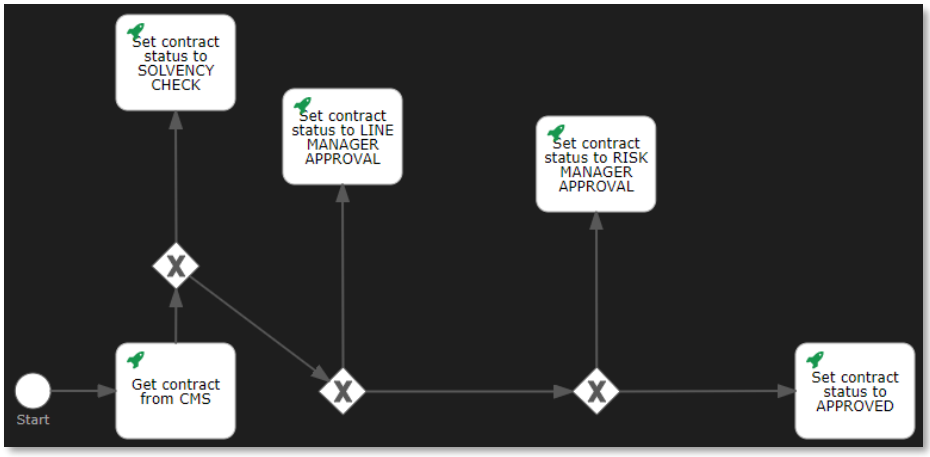
  - **Response variable name: contract**
  - **Save response as JSON: true**
  - **Exclusive: true**



- Http task to set contract status to APPROVED:
  - **Name: Set contract status to APPROVED**
  - **Authentication details: Use current authentication token**
  - **Request method: PATCH**
  - **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  - **Request headers: Content-Type: application/json**
  - **Request body:**

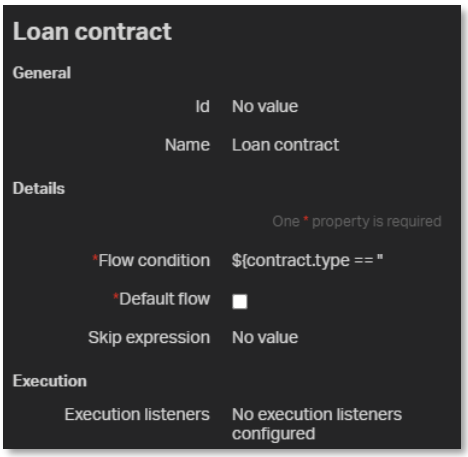
    {
    "properties": {
    "status": "APPROVED"
    }
    }
  - **Response variable name: contract**
  - **Save response as JSON: true**
  - **Exclusive: true**



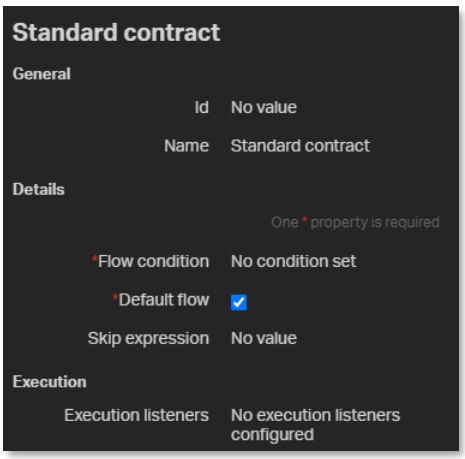


The last step to completing the exclusive gateway (choices) logic is to configure the different sequence flows (i.e.: set the attributes of the arrow connectors). Per exclusive gateway, for the outgoing sequence flow arrow connectors, set the attributes as follows:

- First exclusive gateway:
  - Sequence flow going to the “Set contract status to SOLVENCY CHECK” http task:
    - **Name: Loan contract**
    - **Flow condition: ${contract.type == "ca_loan_contract"}**
    - **Default flow: <not checked>**



  - Sequence flow going to the second exclusive gateway:
    - **Name: Standard contract**
    - **Default flow: <checked>**

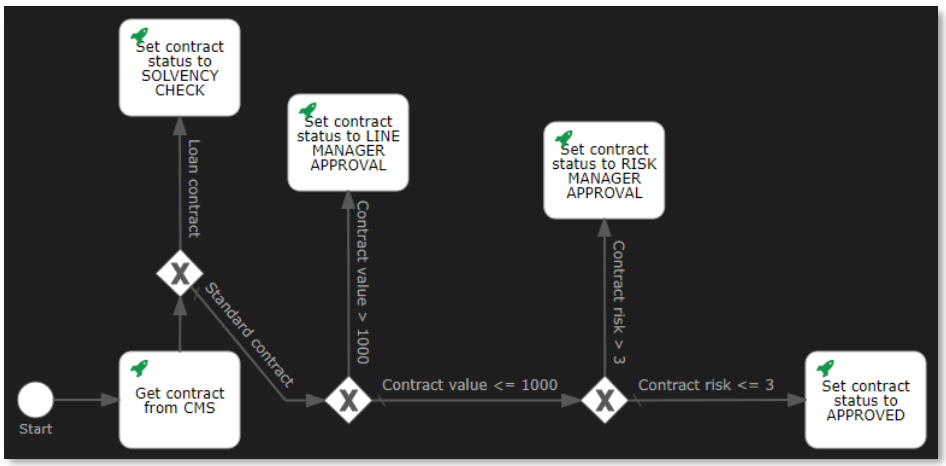


- Second exclusive gateway:
  - Sequence flow going to the “Set contract status to LINE MANAGER APPROVAL” http task:
    - **Name: Contract value > 1000**
    - **Flow condition: ${contract.properties.value > 1000}**
    - **Default flow: <not checked>**

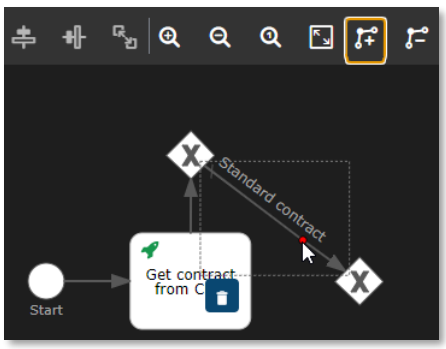
  - Sequence flow going to the third exclusive gateway:
    - **Name: Contract value <= 1000**
    - **Default flow: <checked>**

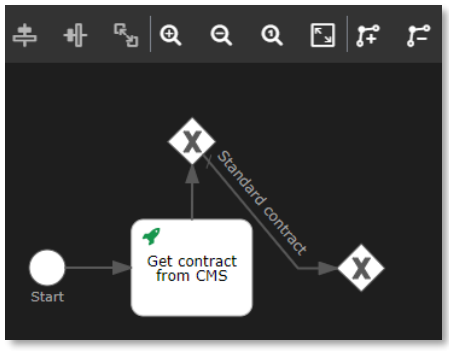
- Third exclusive gateway:
  - Sequence flow going to the “Set contract status to RISK MANAGER APPROVAL” http task:
    - **Name: Contract risk > 3**
    - **Flow condition: ${contract.properties.risk_classification > 3}**
    - **Default flow: <not checked>**

  - Sequence flow going to the “Set contract status to APPROVED” http task:
    - **Name: Contract risk <= 3**
    - **Default flow: <checked>**



You might have noticed that the “Standard contract” sequence flow arrow connector now has a bend point (angle). This can be useful for clarity/layout purposes. To add bend points to the arrow connectors, you should use the button from the button bar.



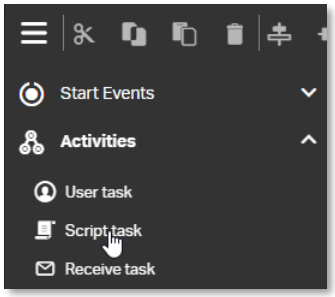


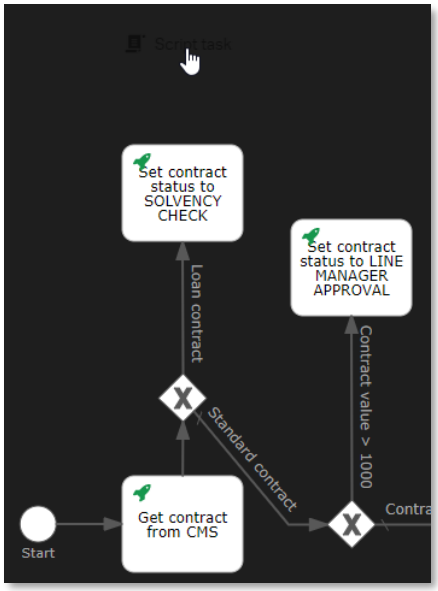
If you did not recently, this is a good time to save your workflow model again.

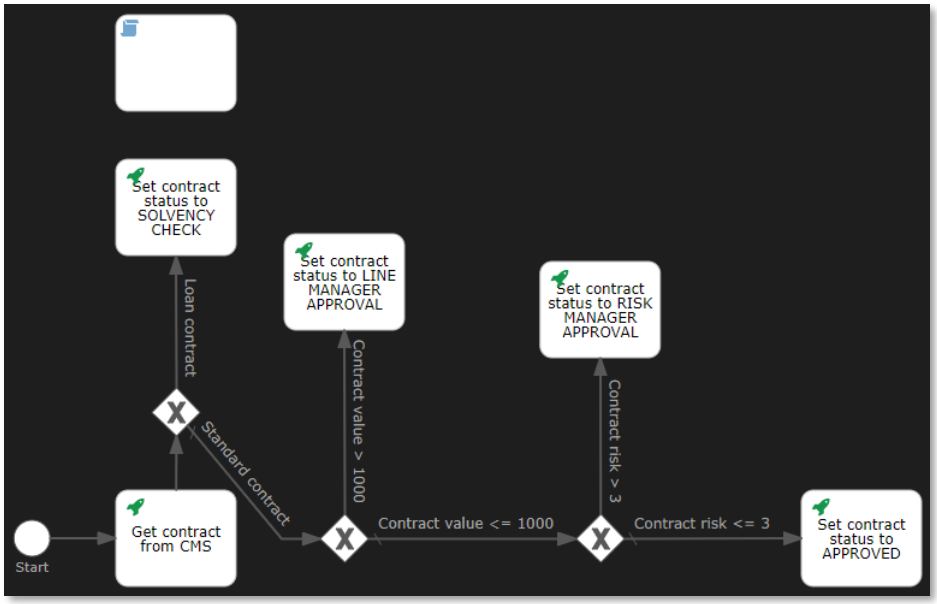
REMARK:
As is the case with all models, if your workflow model contains validation errors these will be shown under the **PROBLEMS** tab from VS Code. Note that, unlike with the other models, a workflow model doesn’t show inline errors right away. That would be too distracting (annoying even) for the user building the workflow. To see any validation issue while you are building the workflow, you indeed need to save the model to trigger the validation.

- Now that we have created all activities that set the status of the contract to indicate that the different approval activities are in progress, we can add the actual approval steps themself. Except for the automatic approval that just sets the contract to be approved, there are indeed three approval steps requiring an approval activity:
  - Calculate solvency:
    This is an automated check that calculates whether or not the person requesting the approval of the loan contract has enough monthly cash flow to pay back the loan. The logic it follows is that it compares the monthly available cash (based on dividing the yearly income by 12) with the monthly payments (based on the total monthly installments count and the value of the contract). If the monthly payment/cost exceeds 25% of the monthly available cash, the customer is considered not to be solvent (enough) and the loan contract approval will be automatically rejected.
  - Line Manager Approval:
    This is a manual approval task by a Line Manager. The Line Manager can choose to approve or reject the contract.
  - Risk Manager Approval:
    This is a manual approval task by a Risk Manager. The Risk Manager can choose to approve or reject the contract.

  Let’s start with the “Calculate solvency” approval task. From the palette (under the **Activities** section), drag and drop a **Script task** onto the canvas above the “Set contract status to SOLVENCY CHECK” http task.



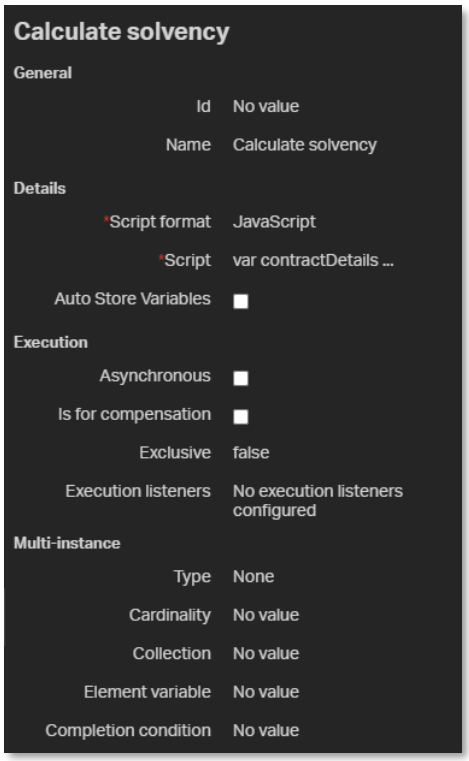




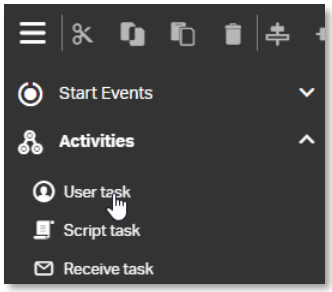
Select the **script task** element you just added and set the element attributes:

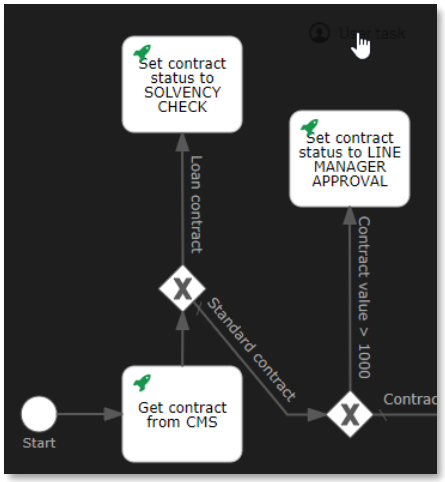
- **Name: Calculate solvency**
- **Script format: JavaScript**
- **Script:**

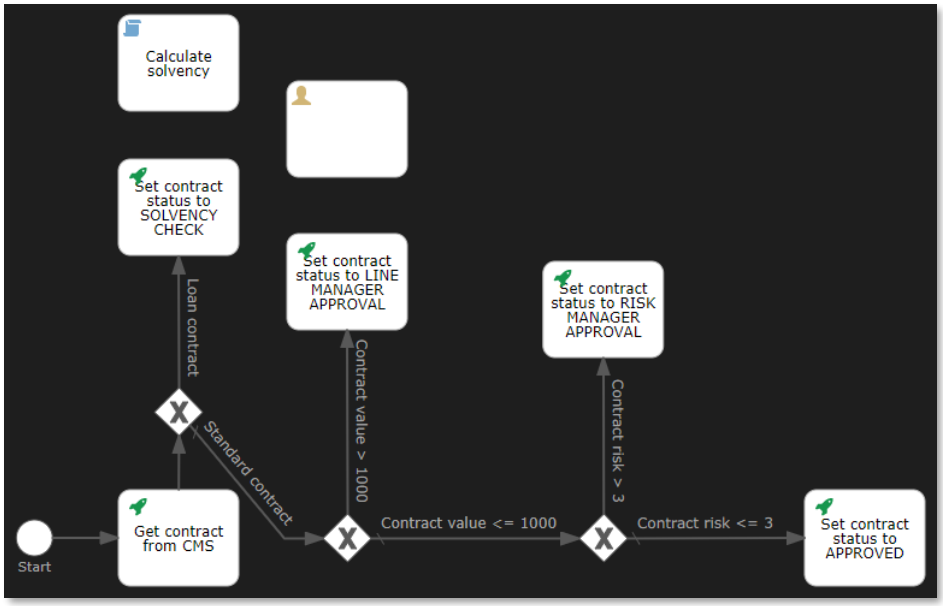
```
var contractDetails = JSON.parse(execution.getVariable("contract"));
var monthlyPayments = contractDetails.properties.value /
contractDetails.properties.monthly_installments;
var monthlyBudget = contractDetails.properties.yearly_income / 12 / 4;

execution.setVariable("solvent", monthlyBudget >= monthlyPayments);
```

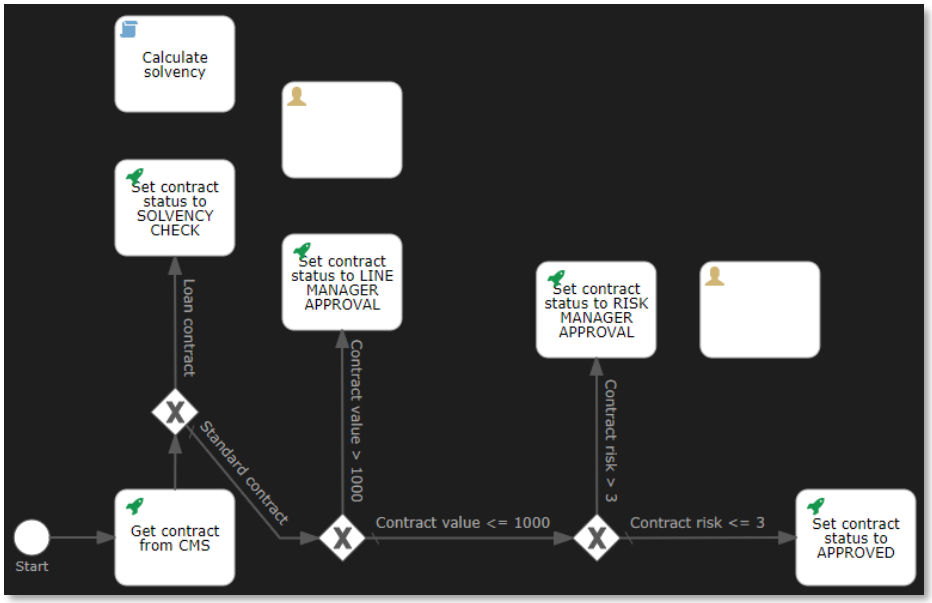


Let's now add both the "Line Manager Approval" and "Risk Manager Approval" manual approval tasks. From the palette (under the **Activities** section), drag and drop a **User task** onto the canvas twice. One above the "Set contract status to LINE MANAGER APPROVAL" http task, and one to the right of the "Set contract status to RISK MANAGER APPROVAL" http task.



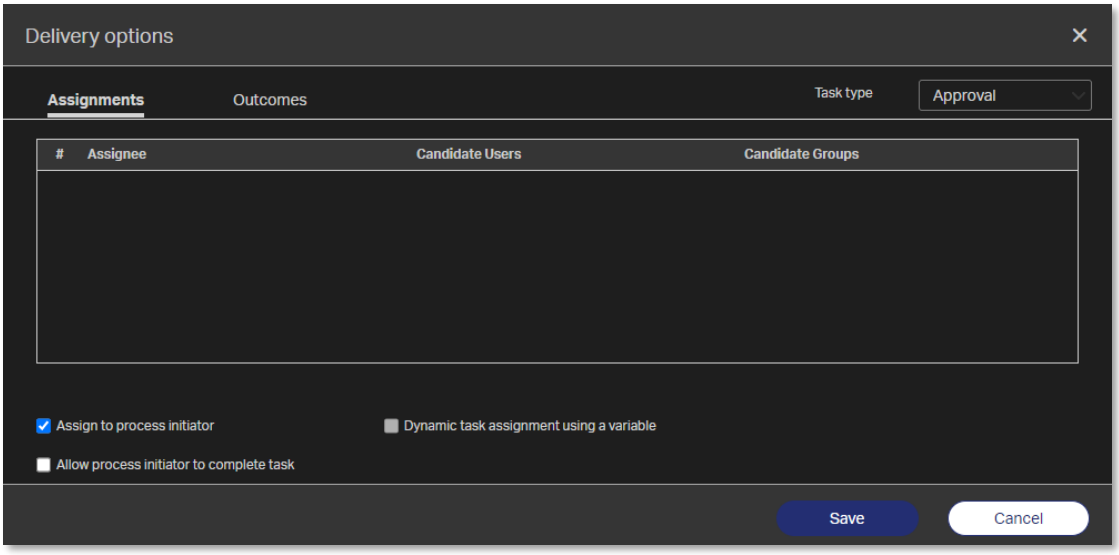






From left to right, set the element attributes for both manual approval tasks:

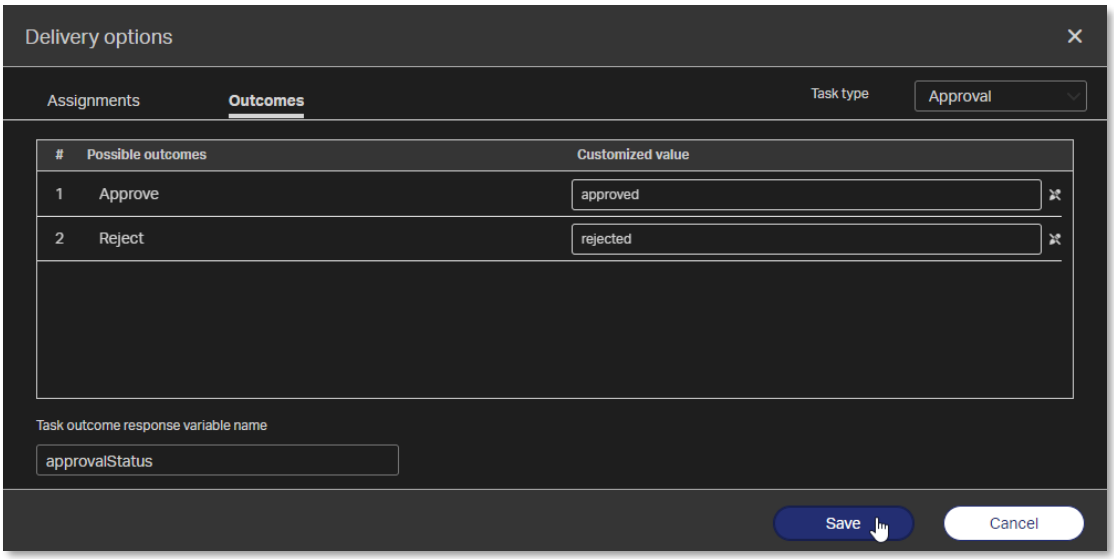
- User task for the Line Manager Approval step:
  - **Name: Line Manager Approval**
  - **Delivery options**
    Assignments tab:
    - **Task type: Approval**
    - **Assign to process initiator: <checked>**



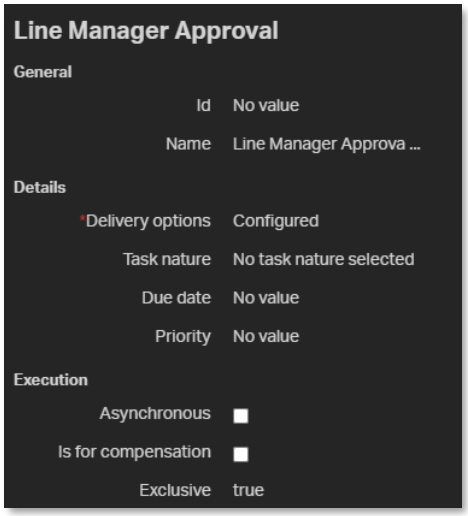
Outcomes tab:

- Modify the customized value for both possible outcomes:
  1. **Possible outcomes: Approve**
     Customized value: approved
  2. **Possible outcomes: Reject**
     Customized value: rejected
- **Task outcome response variable name: approvalStatus**

Once you have filled the **Outcomes** tab, you can click **Save** to save the delivery options and continue filling the manual task properties.



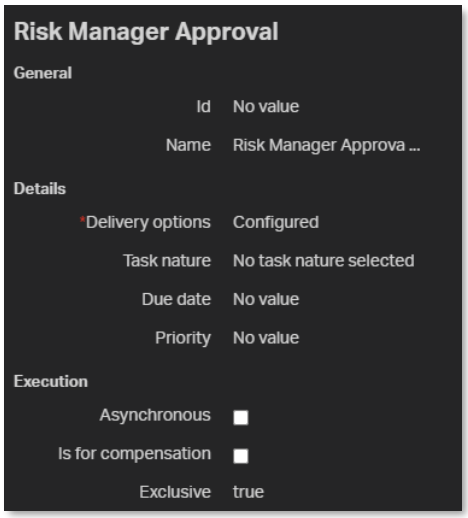
- **Exclusive: true**

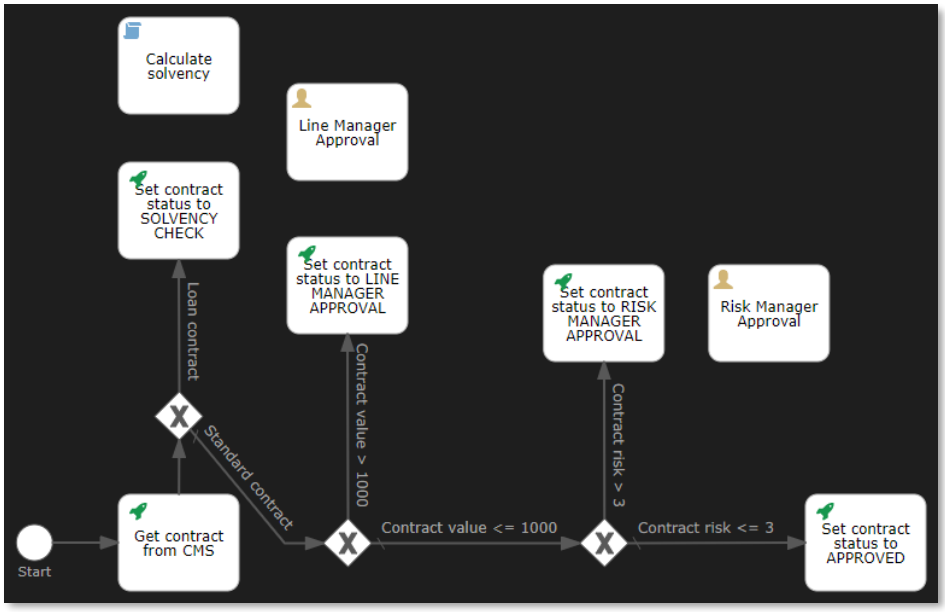


- User task for the Risk Manager Approval step:
  - **Name: Risk Manager Approval**
  - **Delivery options:**
    Assignments tab:
    - **Task type: Approval**
    - **Assign to process initiator: <checked>**

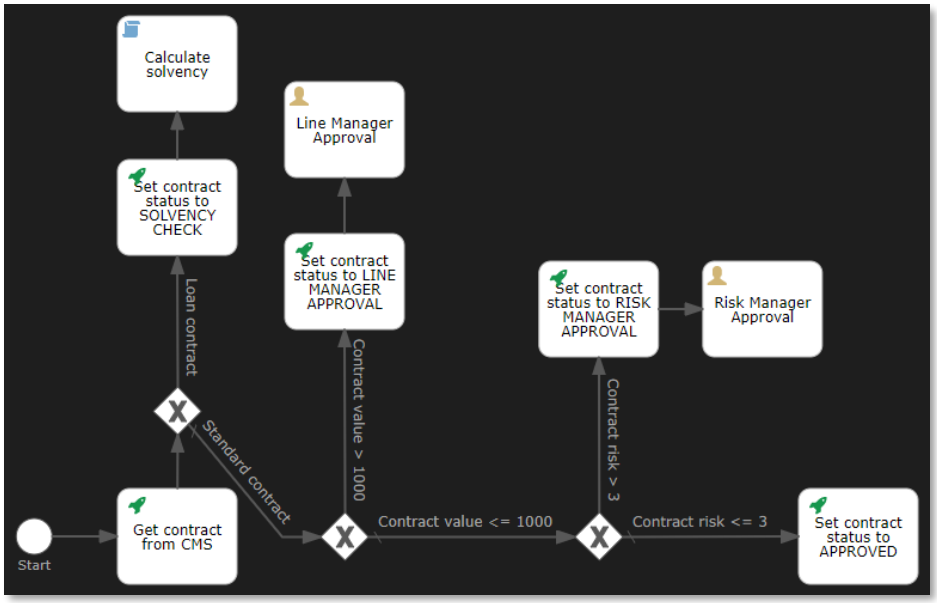
    Outcomes tab:
    - Modify the customized value for both possible outcomes:
      1. **Possible outcomes: Approve**
         Customized value: approved
      2. **Possible outcomes: Reject**
         Customized value: rejected
    - **Task outcome response variable name: approvalStatus**

    Once you have filled the **Outcomes** tab, you can click **Save** to save the delivery options and continue filling the manual task properties.
  - **Exclusive: true**



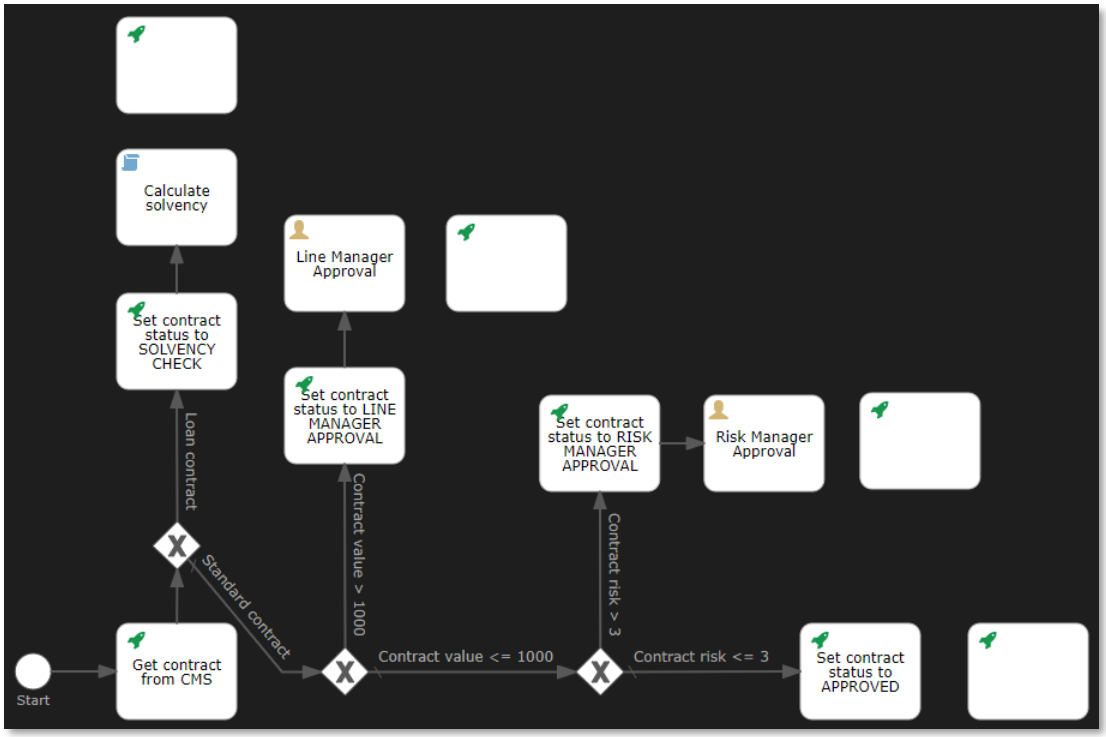


To finish creating the three approval tasks (Calculate solvency, Line Manager Approval, and Risk Manager Approval), you can now connect them from their respective preceding tasks by adding the three corresponding sequence flow arrow connectors:



There is no need to set any attributes for the sequence flow connectors you just added. Please save your workflow model to make sure not to lose your work.

- Now that we have added all approval steps, we can add the http tasks that update the contract with the results of the approval. More specifically, each approval step has its own corresponding (required) trait on the contract instance that is being approved, and we will create an “Update trait” http task for each approval activity.
  Note that we have already done this before. You might have previously noticed that for the manual approval steps, the corresponding traits have been already partially updated (during the “Set contract status to …” http tasks) to signal that a manual approval is required.
  Let’s now add all four “Update trait” (PATCH request) http tasks. Drag and drop four **Http task** elements onto the canvas. One above the “Calculate solvency” script task, one to the right of the “Line Manager Approval” user task, one to the right of the “Risk Manager Approval” user task, and one to the right of the “Set contract status to APPROVED” http task.

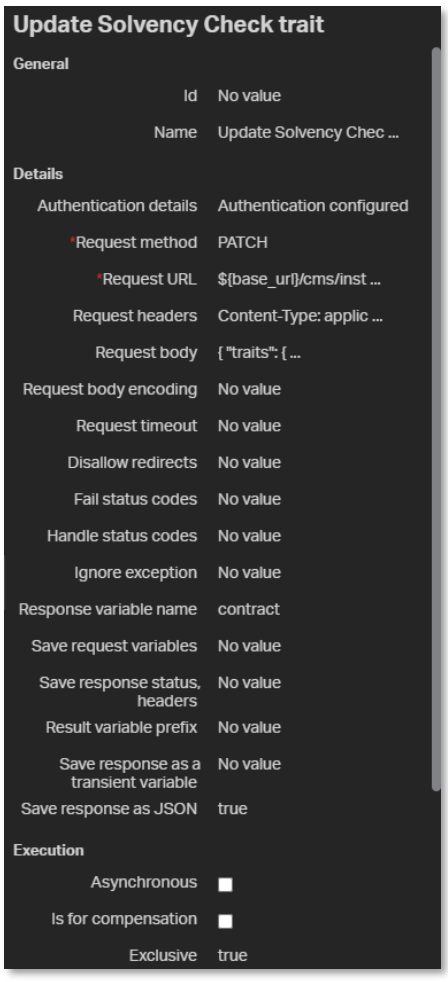


Set the attributes of the four new “Update trait” (PATCH request) http tasks as follows:

- Http task to update the Solvency Check trait:
  - **Name: Update Solvency Check trait**
  - **Authentication details: Use current authentication token**
  - **Request method: PATCH**
  - **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  - **Request headers: Content-Type: application/json**
  - **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}"
      }
    }
  }
}
```

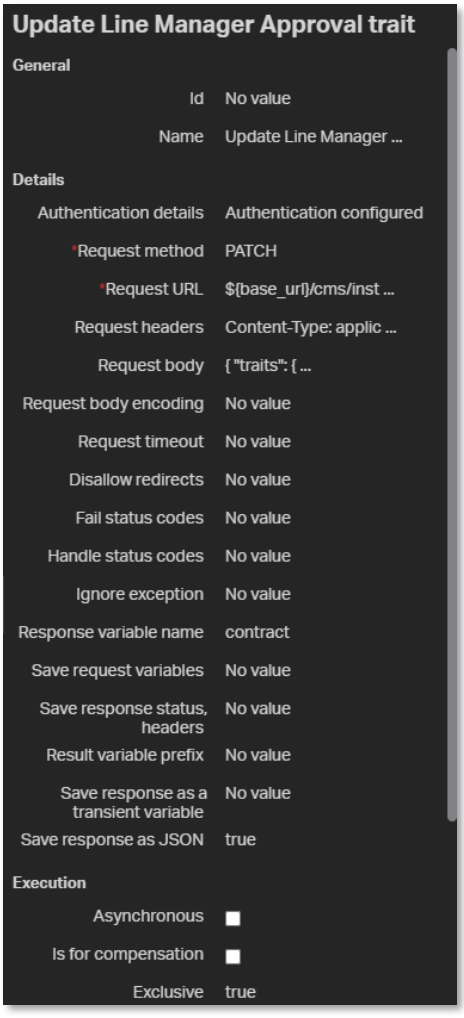
  - **Response variable name: contract**
  - **Save response as JSON: true**
  - **Exclusive: true**



- Http task to update the Line Manager Approval trait:
  - **Name: Update Line Manager Approval trait**
  - **Authentication details: Use current authentication token**
  - **Request method: PATCH**
  - **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  - **Request headers: Content-Type: application/json**
  - **Request body:**

```
{
  "traits": {
    "ca_approval": {
      "Line Manager Approval": {
        "is_required": true,
        "has_been_granted": ${approvalStatus == "approved"},
        "approver": "${contract.updated_by.email}",
        "approver_role": "Line Manager",
        "approval_date": "${contract.update_time}"
      }
    }
  }
}
```

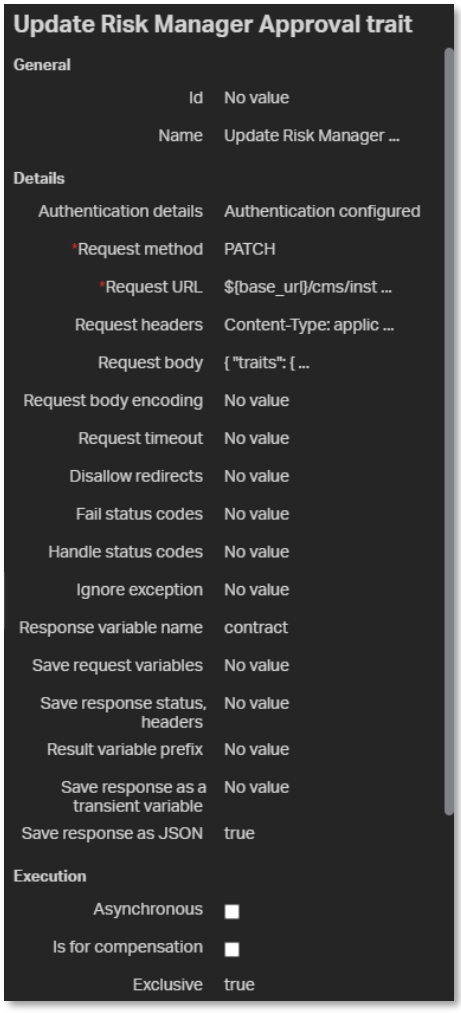
  - **Response variable name: contract**
  - **Save response as JSON: true**
  - **Exclusive: true**



- Http task to update the Risk Manager Approval trait:
  - **Name: Update Risk Manager Approval trait**
  - **Authentication details: Use current authentication token**
  - **Request method: PATCH**
  - **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  - **Request headers: Content-Type: application/json**
  - **Request body:**

```
{
  "traits": {
    "ca_approval": {
      "Risk Manager Approval": {
        "is_required": true,
        "has_been_granted": ${approvalStatus == "approved"},
        "approver": "${contract.updated_by.email}",
        "approver_role": "Risk Manager",
        "approval_date": "${contract.update_time}"
      }
    }
  }
}
```

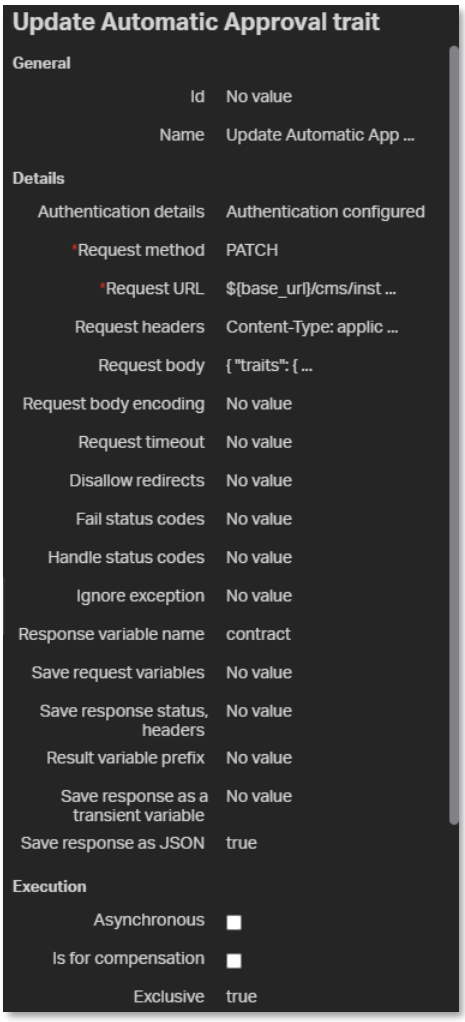
  - **Response variable name: contract**
  - **Save response as JSON: true**
  - **Exclusive: true**

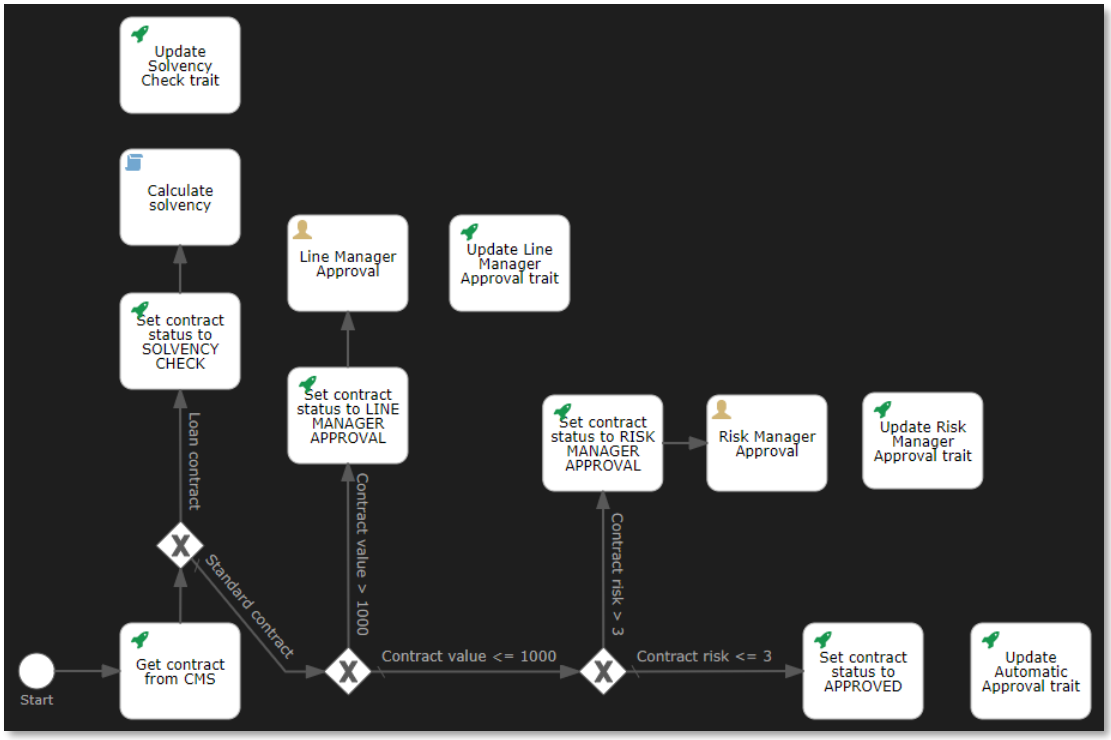


- Http task to update the Automatic Approval trait:
  - **Name: Update Automatic Approval trait**
  - **Authentication details: Use current authentication token**
  - **Request method: PATCH**
  - **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
  - **Request headers: Content-Type: application/json**
  - **Request body:**

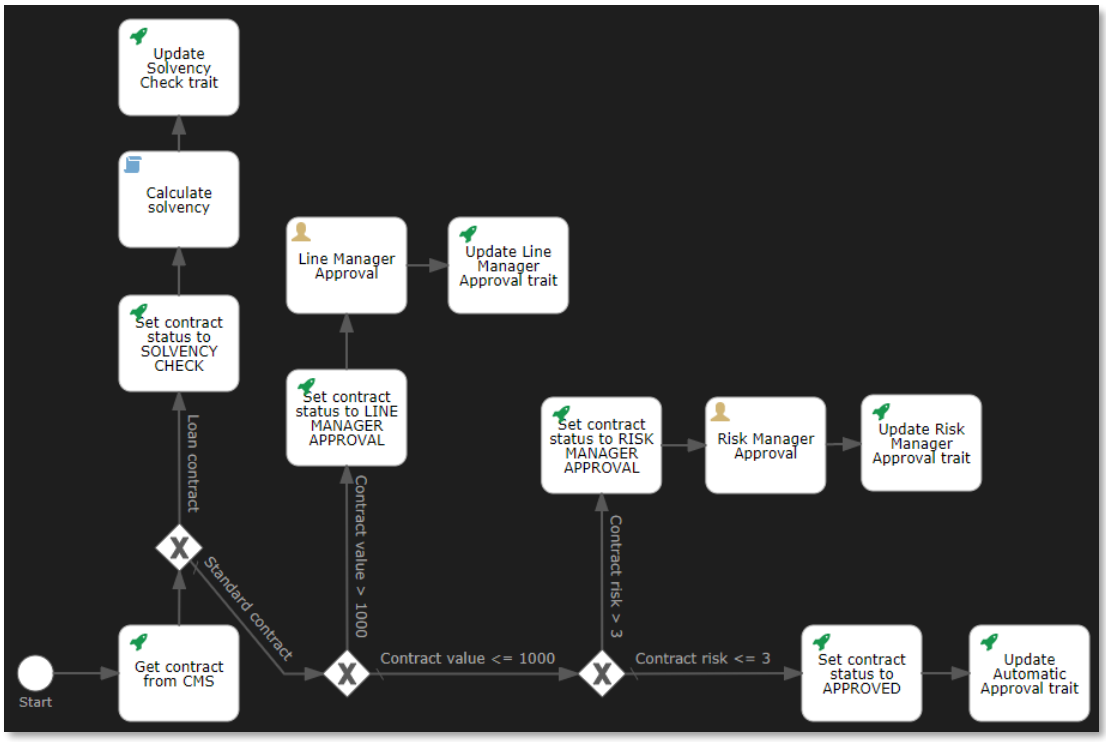
```
{
  "traits": {
    "ca_approval": {
      "Automatic Approval": {
        "has_been_granted": true,
        "approver": "SYSTEM",
        "approver_role": "Automatic Approval",
        "approval_date": "${contract.update_time}"
      }
    }
  }
}
```

  - **Response variable name: contract**
  - **Save response as JSON: true**
  - **Exclusive: true**



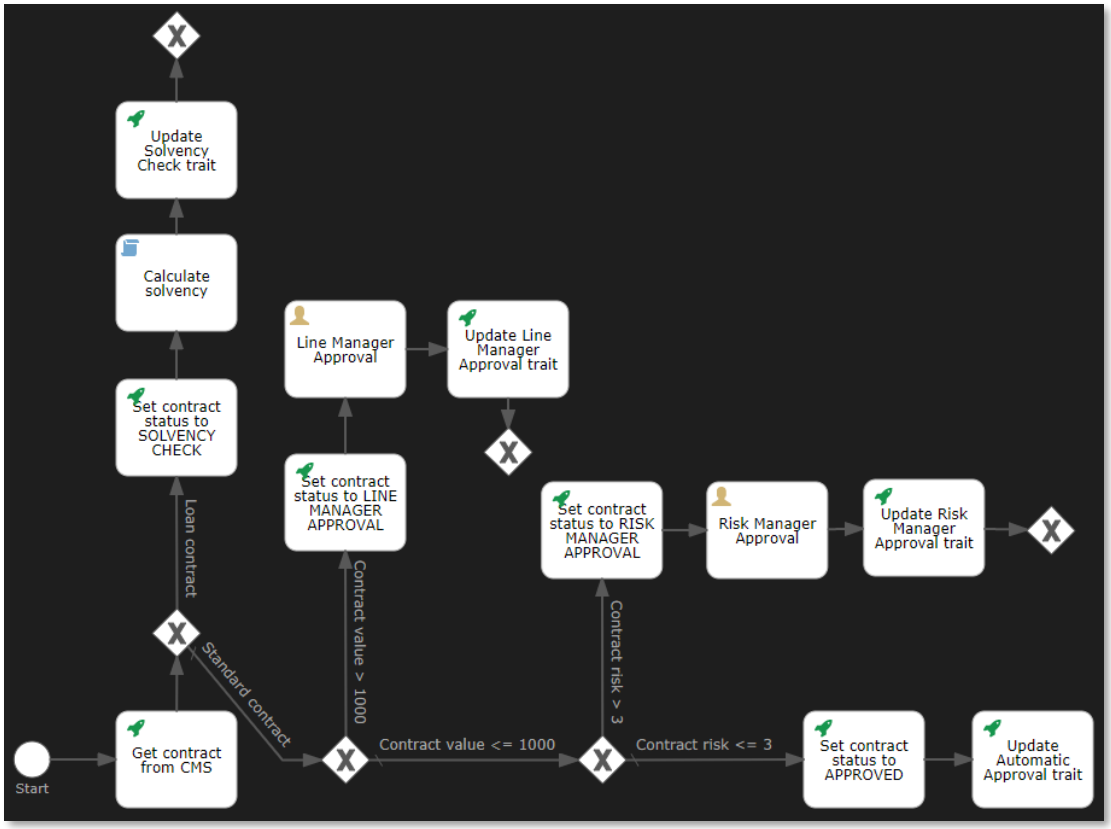


You can now add the four sequence flow arrow connectors that link the previous tasks to the “Update trait” http tasks:

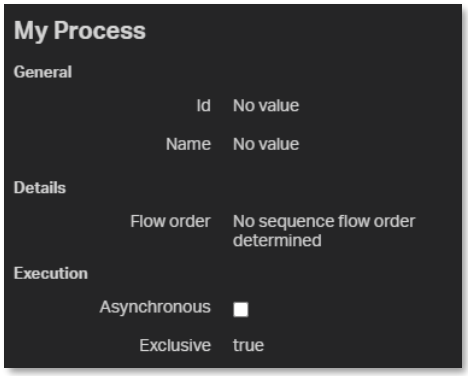


There is no need to set any attributes for the sequence flow connectors you just added. Please save your workflow model to make sure not to lose your work.

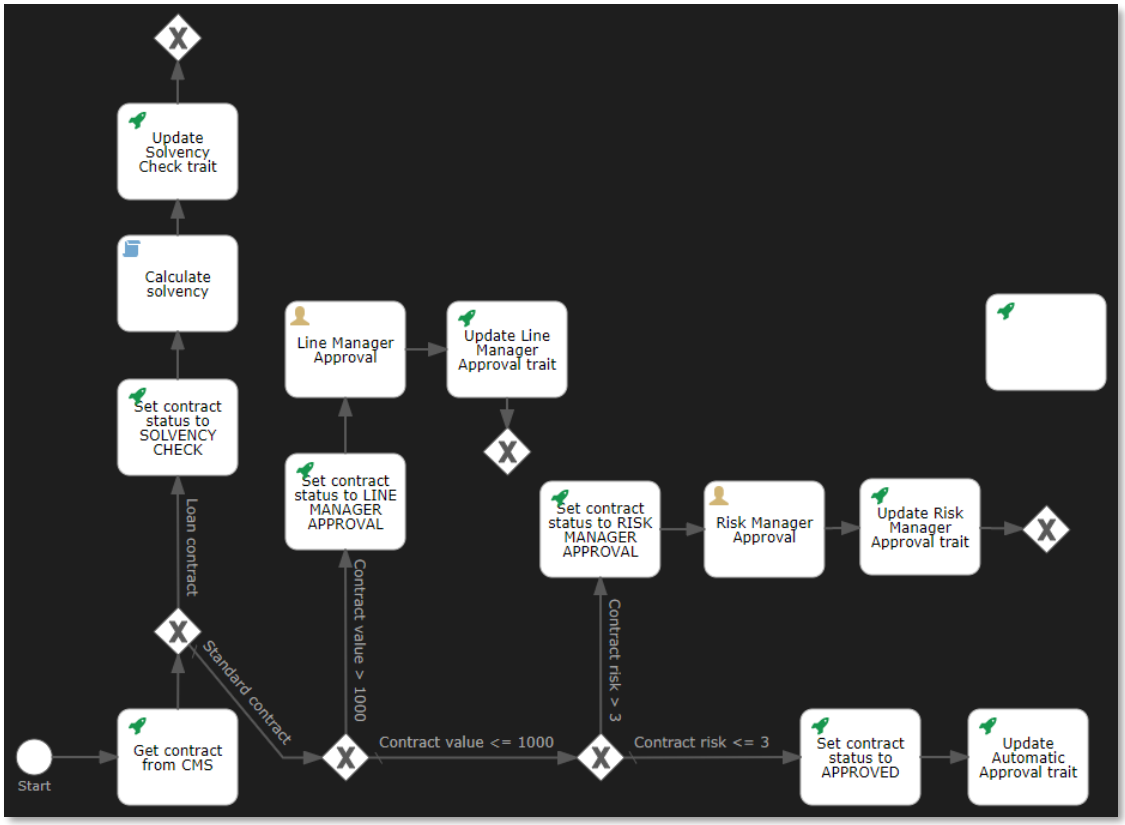
- The result of the three approval tasks that have the option to approve or reject the contract (Solvency check, Line Manager Approval, and Risk Manager Approval) is now known. So, it is now possible, depending on whether the approval step resulted in approval being granted or not, to advance to the next step in the approval process when approved, or terminate to the process via the rejection flow when rejected.
  To implement this logic of choosing the correct process route, based on the results of the approval steps, we will add (you might have guessed it already) three new **Exclusive gateway** elements. In the same way as with the previously created exclusive gateways, you can also already add the sequence flow arrow connectors coming from the previous (approval) task elements.



There's no need to set any attributes for the sequence flow arrow connectors.
For all three new exclusive gateway elements, set the **Exclusive** attribute to **true**:



- Before we can connect the exclusive gateways to the two possible next process steps (depending on being approved or rejected), we need to create the activity that represents what needs to happen when the approval task resulted in a rejection, as this missing activity is one of the two outcomes/options. More specifically, we need to create the http task that sets the contract's status to REJECTED.
  Proceed by dragging and dropping a new **Http task** element onto the canvas right above the exclusive gateway element that is most to the right, out of the three new exclusive gateway elements.

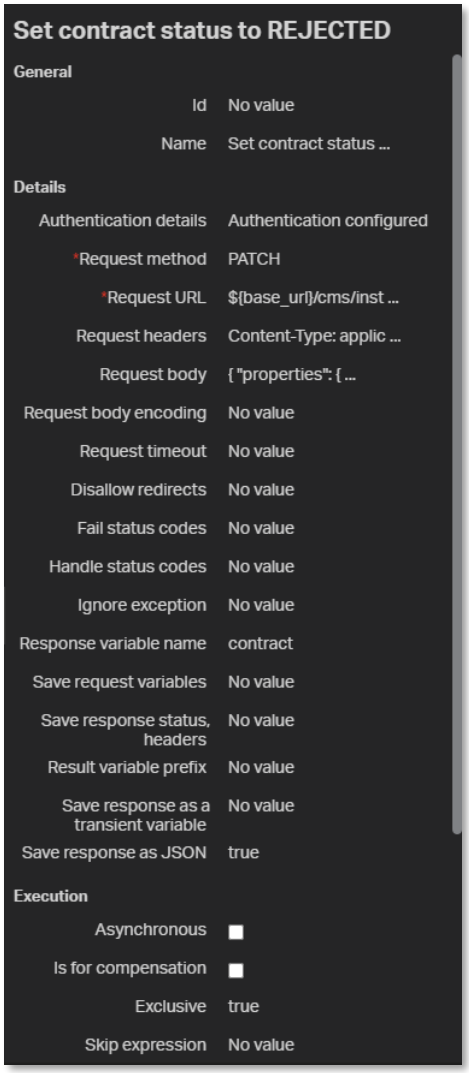


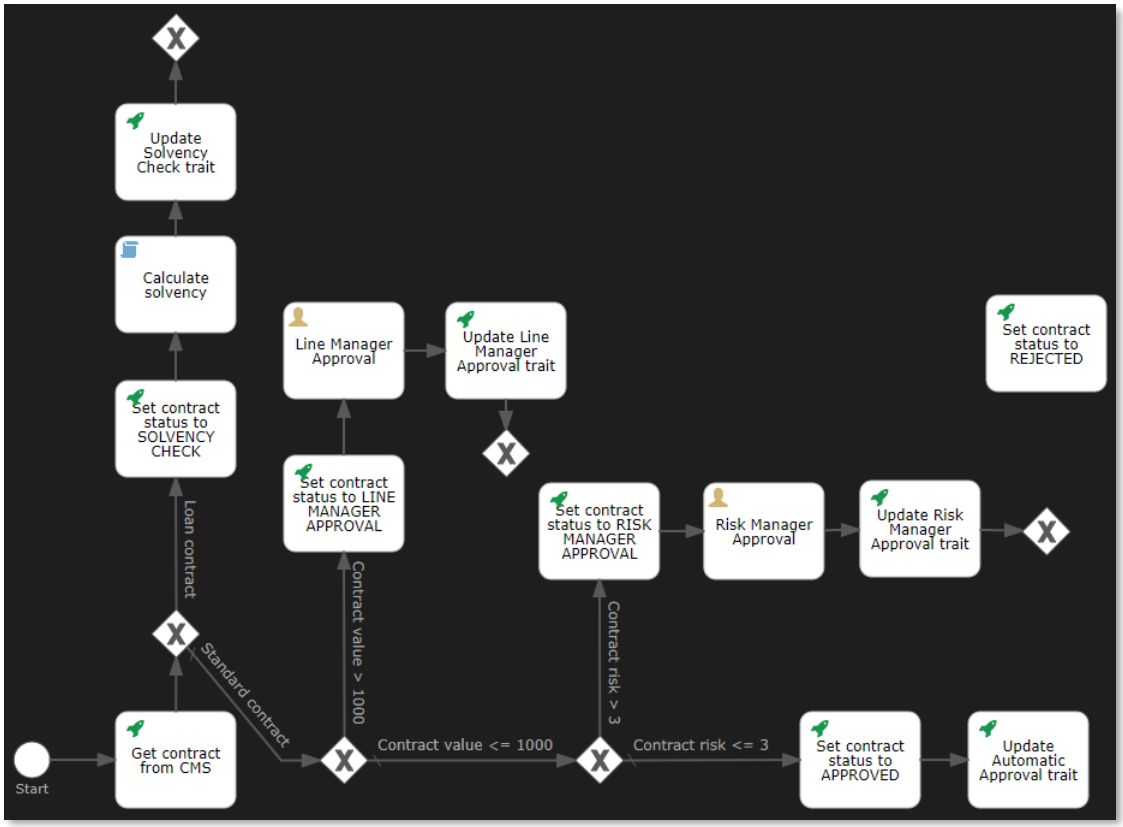
Very much like with the other “Set contract status to …” tasks, set the attributes of this new “Set contract status to REJECTED” http task as follows:

- **Name: Set contract status to REJECTED**
- **Authentication details: Use current authentication token**
- **Request method: PATCH**
- **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
- **Request headers: Content-Type: application/json**
- **Request body:**

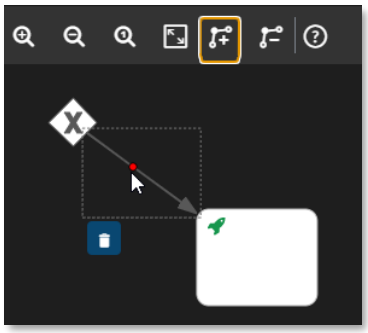
```
{
  "properties": {
    "status": "REJECTED"
  }
}
```

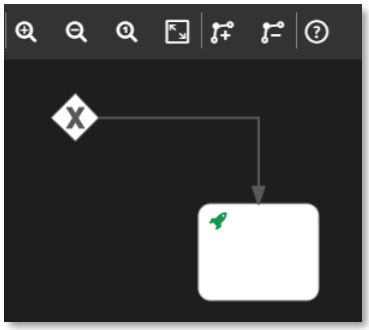
- **Response variable name: contract**
- **Save response as JSON: true**
- **Exclusive: true**



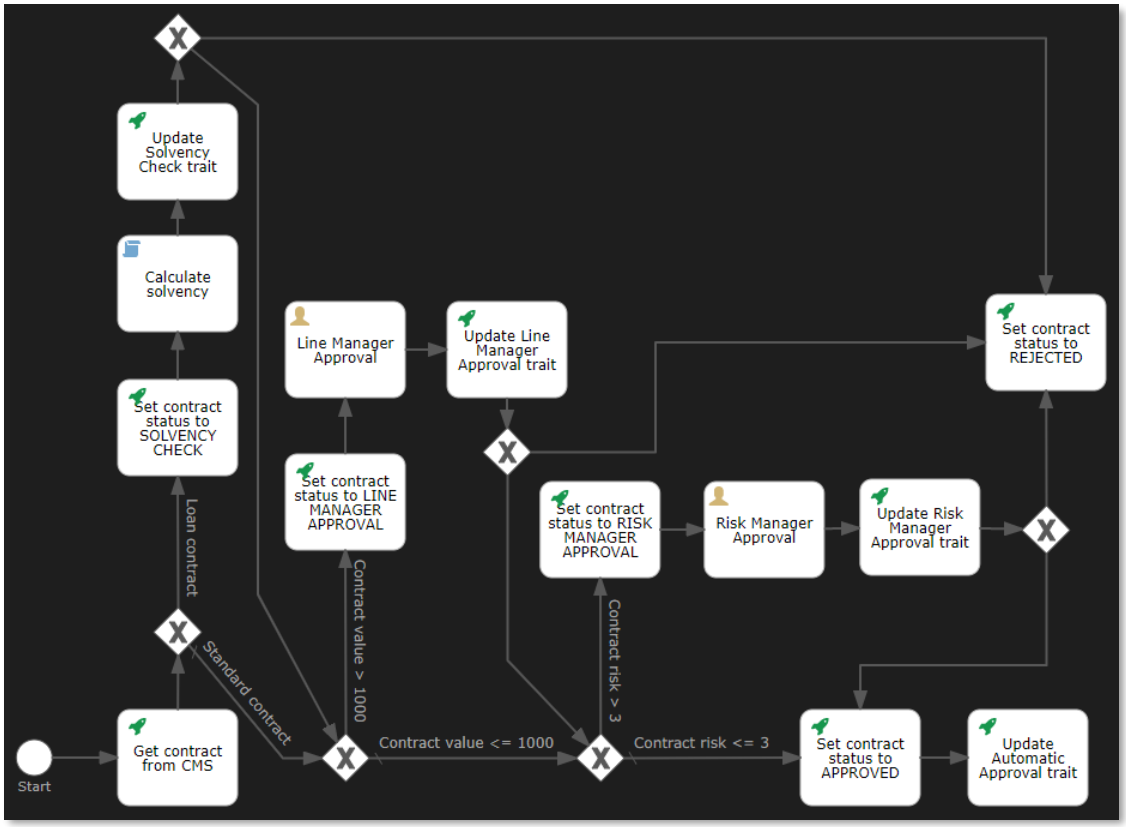


- We can now connect the three approved/rejected exclusive gateways to their two possible next process steps. For each of the three new exclusive gateway elements, use two sequence flow arrow connectors to connect them with their respective subsequent process step elements. Remember that to add bend points (angles) to the arrow connectors, you should use the button from the button bar.





Once done, you should have created 6 new sequence flow arrow connectors, as shown in the next screen shot.

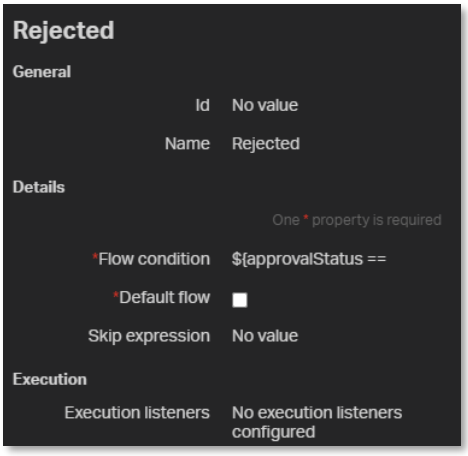


What's left for the three approved/rejected exclusive gateways is to set the attributes of the different outgoing sequence flows. From left to right, set the sequence flow attributes as follows:

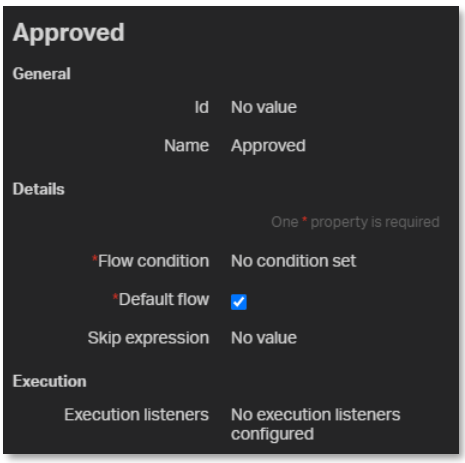
- First exclusive gateway:
  - Sequence flow going to the "Set contract status to REJECTED" http task:
    - **Name: Not solvent**
    - **Flow condition: ${!solvent}**
    - **Default flow: <not checked>**

- Sequence flow going to the contract value checking exclusive gateway:
  - **Name: Solvent**
  - **Default flow: <checked>**

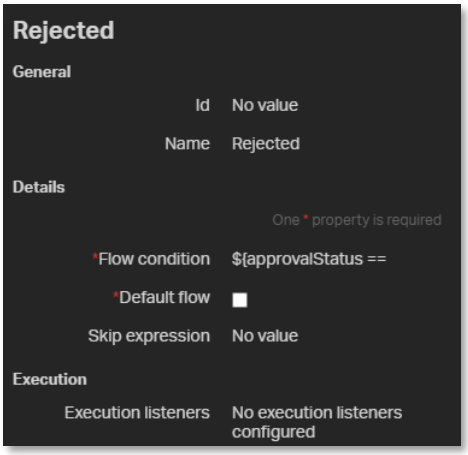
- Second exclusive gateway:
  - Sequence flow going to the “Set contract status to REJECTED” http task:
    - **Name: Rejected**
    - **Flow condition: ${approvalStatus == "rejected"}**
    - **Default flow: <not checked>**



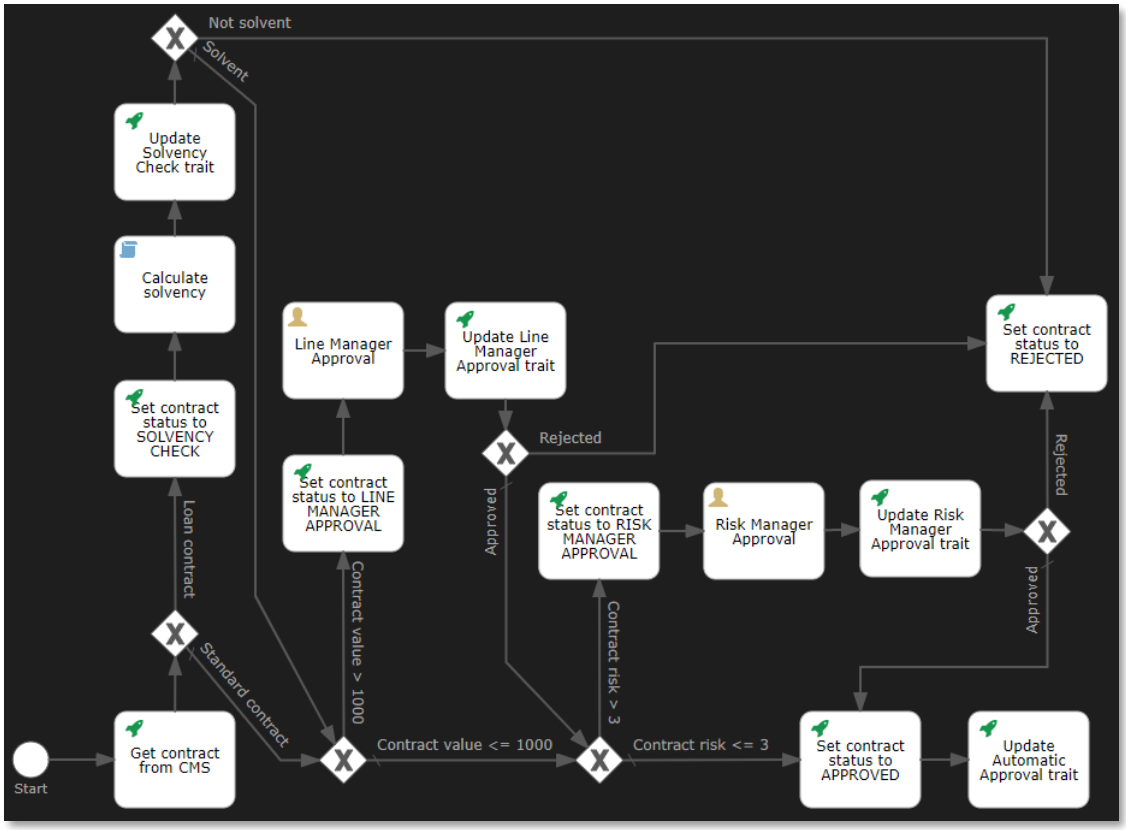
  - Sequence flow going to the contract risk checking exclusive gateway:
    - **Name: Approved**
    - **Default flow: <checked>**



- Third exclusive gateway:
  - Sequence flow going to the “Set contract status to REJECTED” http task:
    - **Name: Rejected**
    - **Flow condition: ${approvalStatus == "rejected"}**
    - **Default flow: <not checked>**

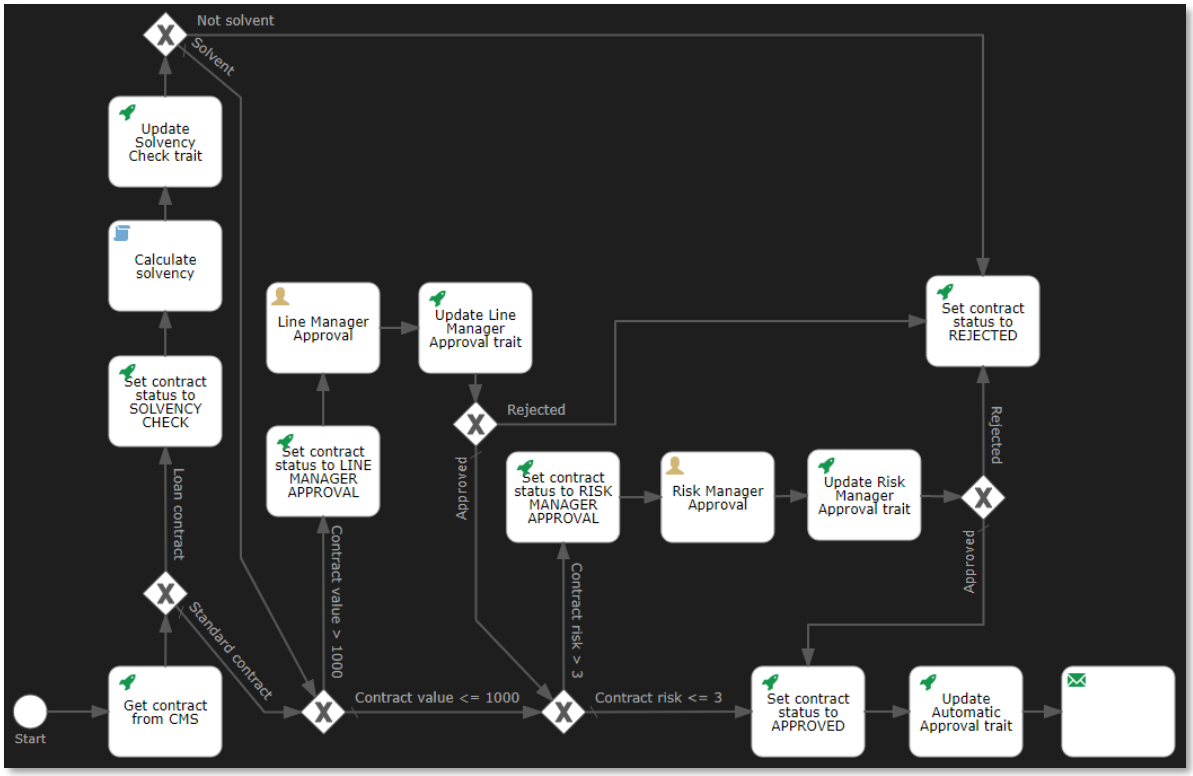


  - Sequence flow going to the “Set contract status to APPROVED” http task:
    - **Name: Approved**
    - **Default flow: <checked>**

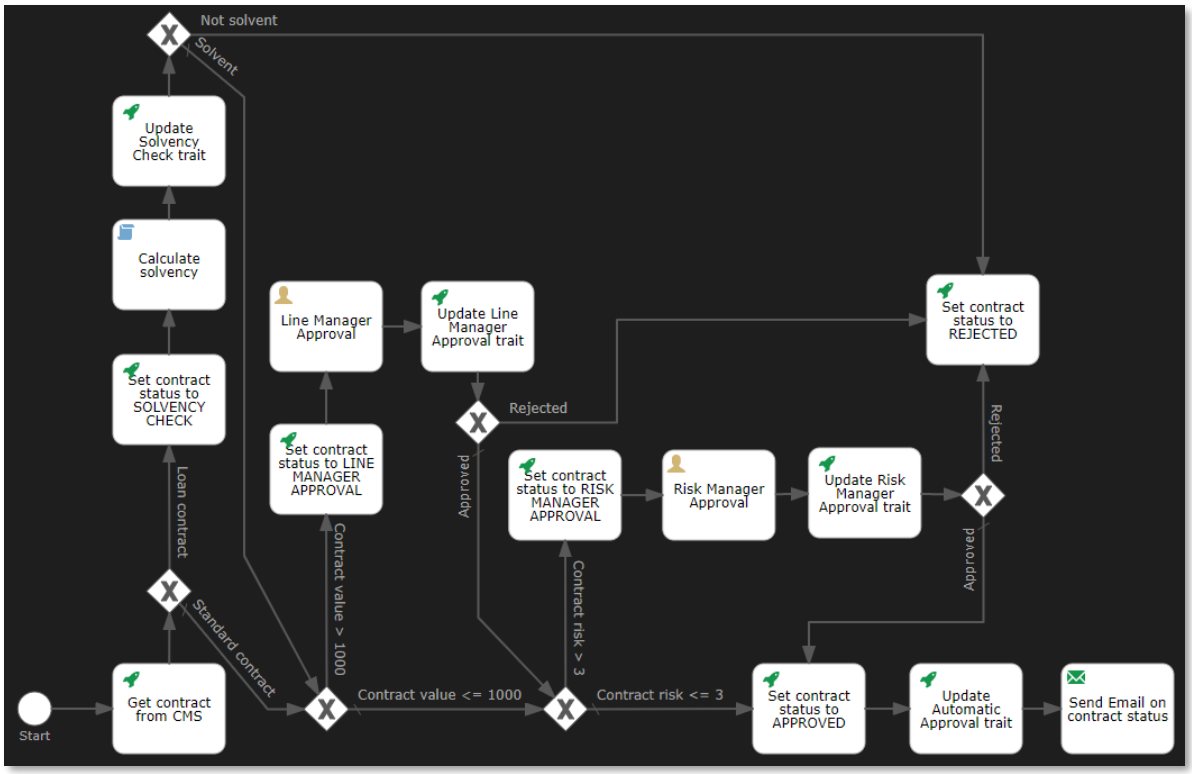


This is again a good time to save your work.

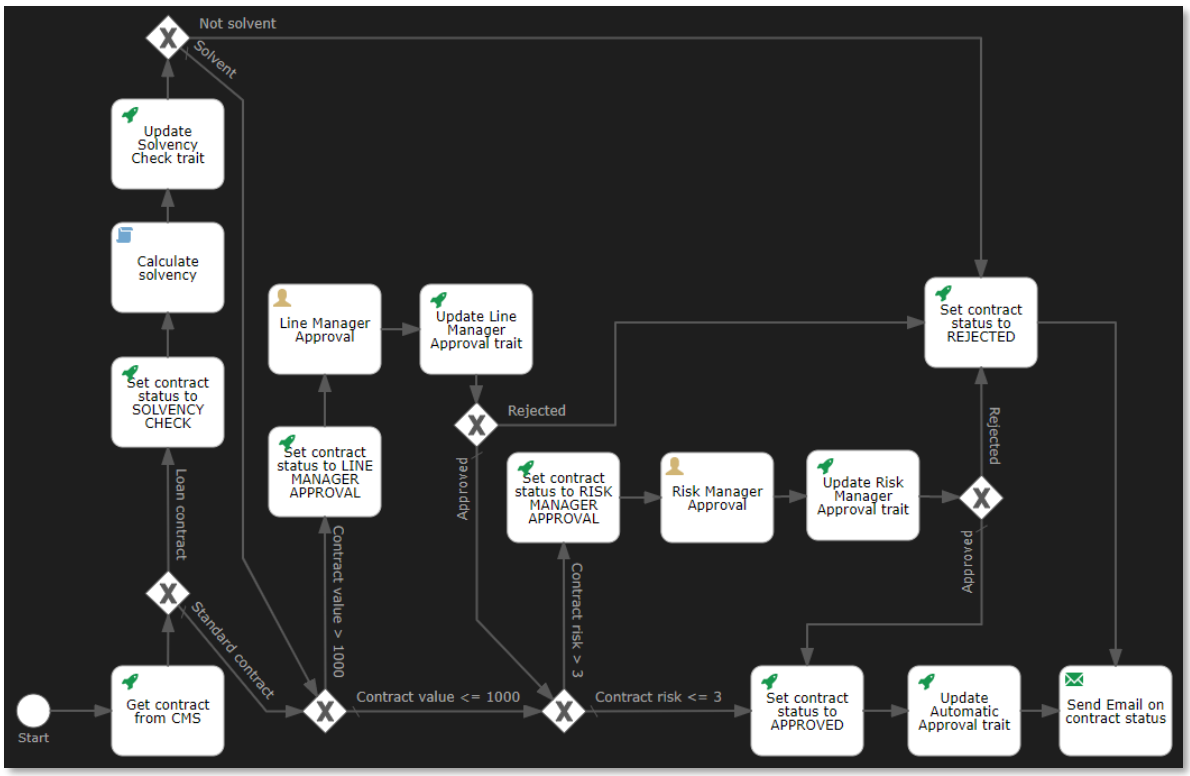
- We have now built the main flow of the contract approval business process. The only thing left, before adding an end event to the workflow model, is to add an email task that sends an email to the person having requested the approval of the contract (i.e.: the customer) to inform them about the approval status (APPROVED or REJECTED) of the contract.
  To do this, drag and drop a **Mail task** from the palette (under the **Activities** section) onto the canvas to the right of the "Update Automatic Approval trait" http task. You can also immediately add the sequence flow arrow connector, as there are no specific sequence flow attributes to set.



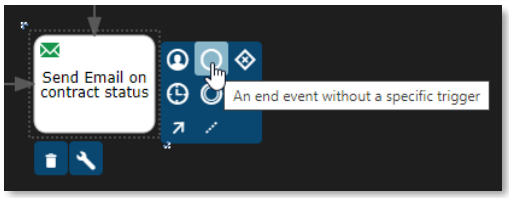
  Fill the mail task attributes as follows:
  - **Name: Send Email on contract status**
  - **To: ${contract.properties.requester_email}**
  - **From: noreply@mycompany.com**
  - **Subject: Contract Approval Status**
  - **Text:**
    Contract: ${contract.name}
    Status: ${contract.properties.status}
  - **Exclusive: true**

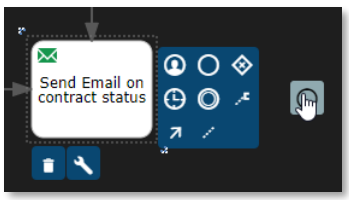


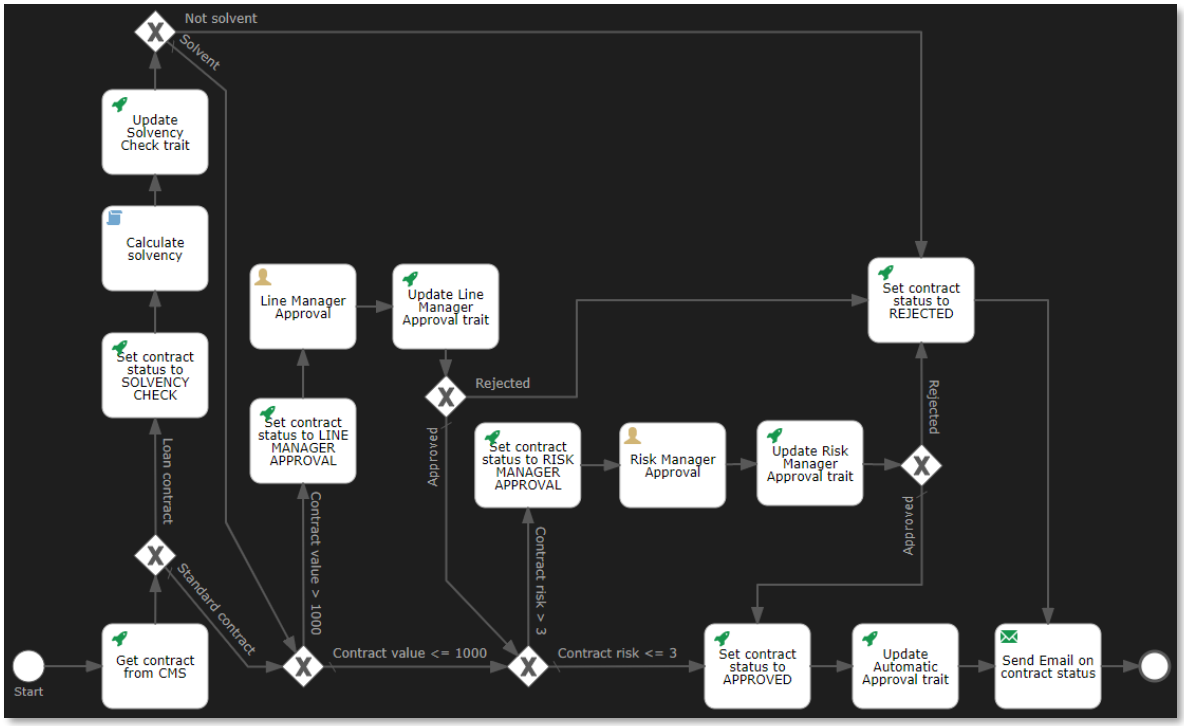
To ensure the REJECTED contract approval status email gets sent as well, connect the “Set contract status to REJECTED” http task to the new “Send Email on contract status” task.



- You can now add the end event as termination event of your business process. Do this by simply selecting the “Send Email on contract status” **email task** and drag and dropping the **end event** (middle top icon) to the right of the email task.

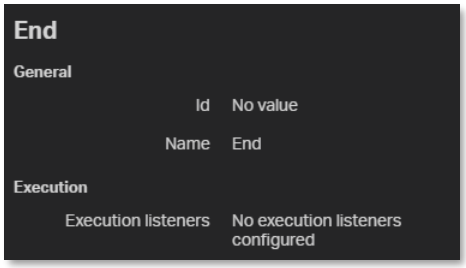


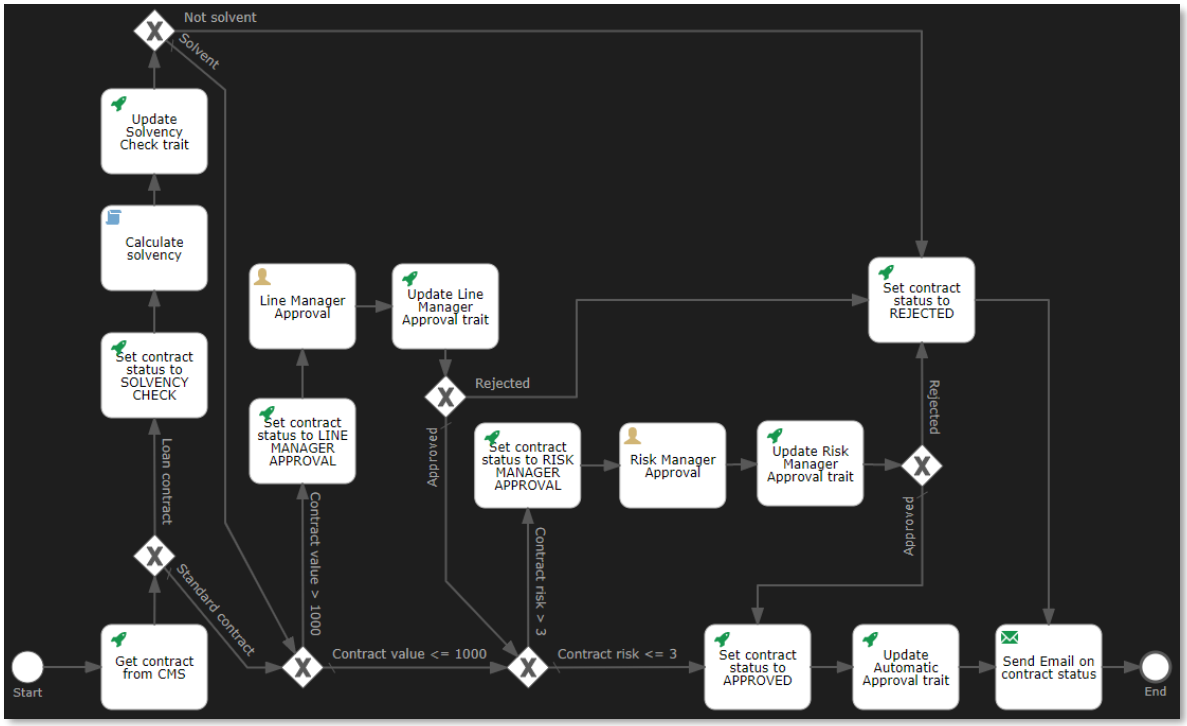




Set the attributes for the end event as follows:

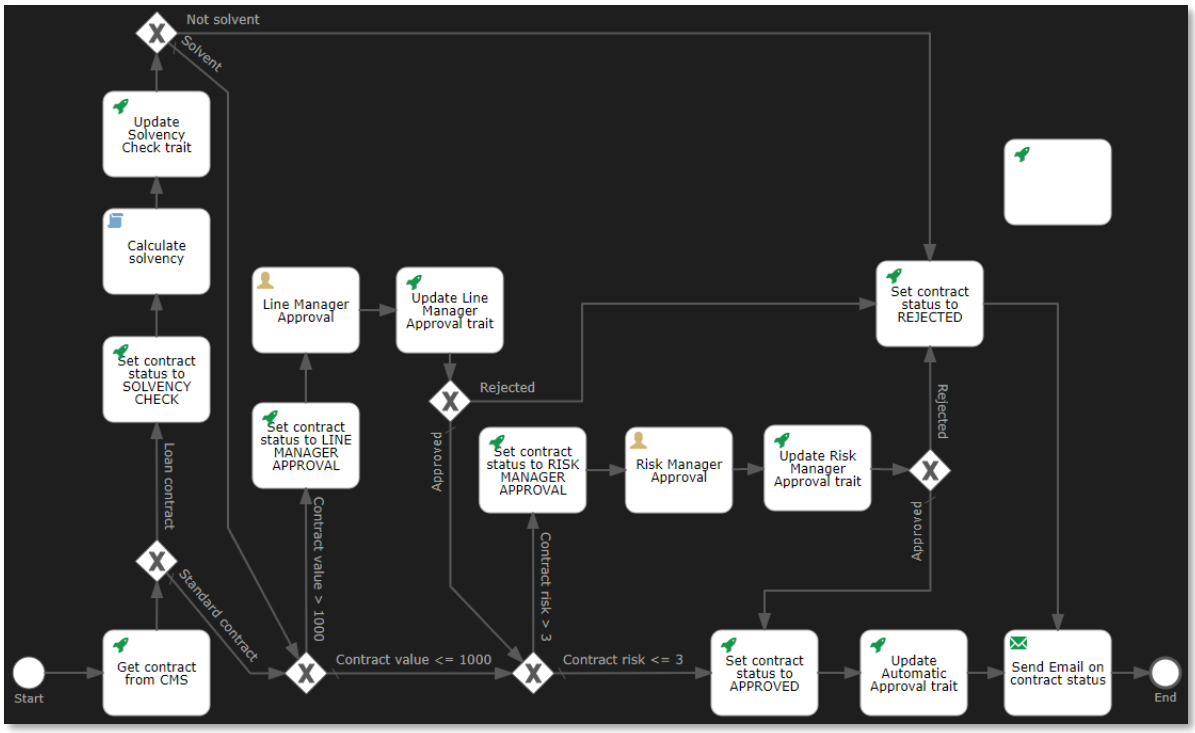
- **Name: End**





Again, please save the workflow model to ensure you don't lose any work.

- You have now implemented the main logic of the contract approval business process. Let's add one more activity to address a special case. More specifically, we want the manual approval tasks (by Line Manager and Risk Manager) to expire if a certain wait period has been exceeded, so that approval tasks don't get stuck forever in the approver's inbox. For that, we will add one more http task that marks the contract as EXPIRED (as special exception status).
  So, please drag and drop a new **Http task** element onto the canvas above the "Send Email on contract status" email task at the top end of the canvas (so that it is higher than the "Set contract status to REJECTED" http task).

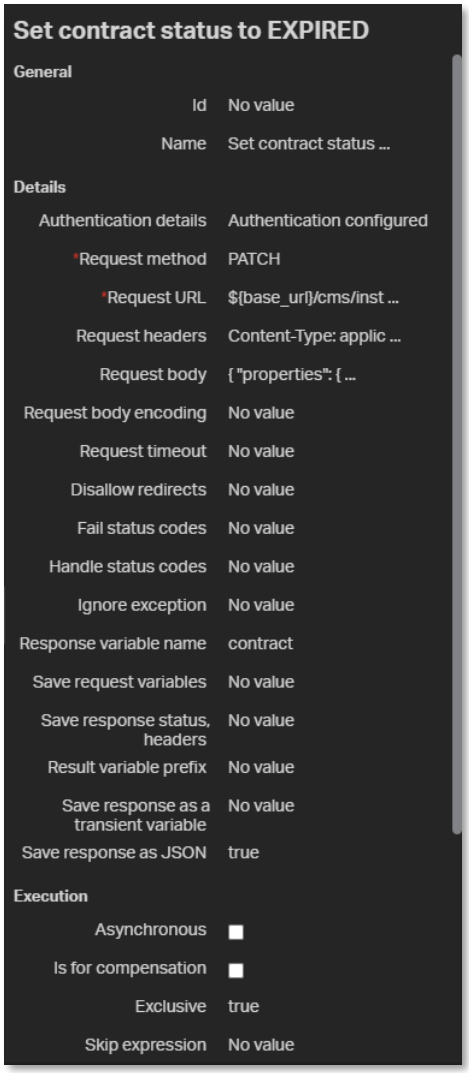


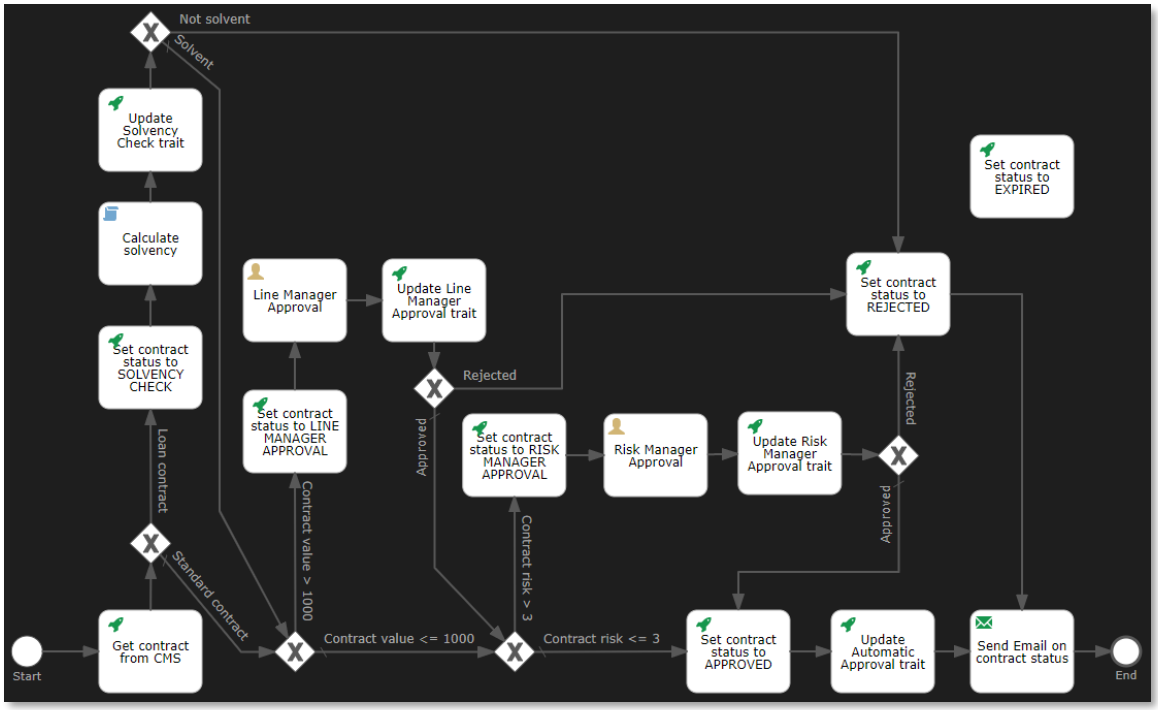
Fill the http task attributes as follows:

- **Name: Set contract status to EXPIRED**
- **Authentication details: Use current authentication token**
- **Request method: PATCH**
- **Request URL: ${base_url}/cms/instances/file/ca_contract/${contract_id}**
- **Request headers: Content-Type: application/json**
- **Request body:**

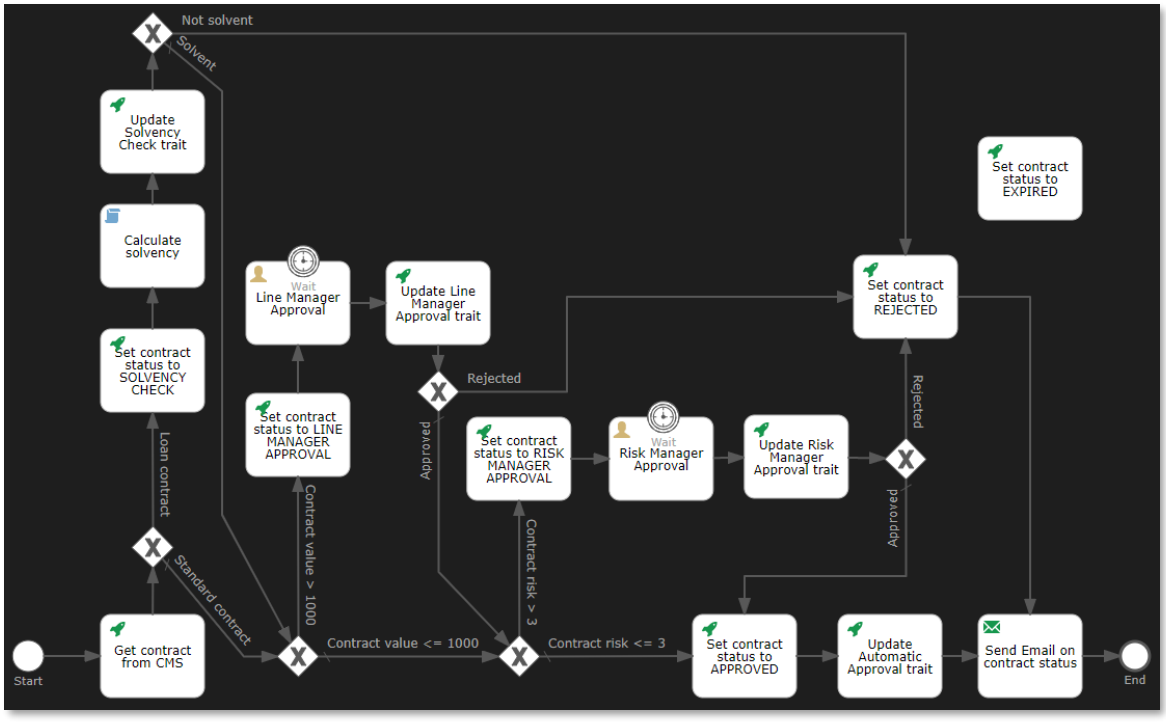
```
{
  "properties": {
    "status": "EXPIRED"
  }
}
```

- **Response variable name: contract**
- **Save response as JSON: true**
- **Exclusive: true**



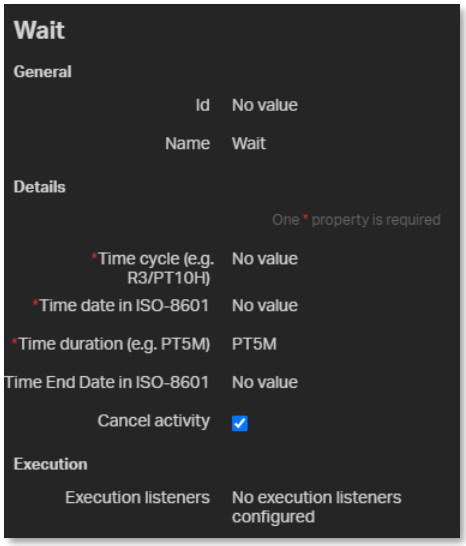


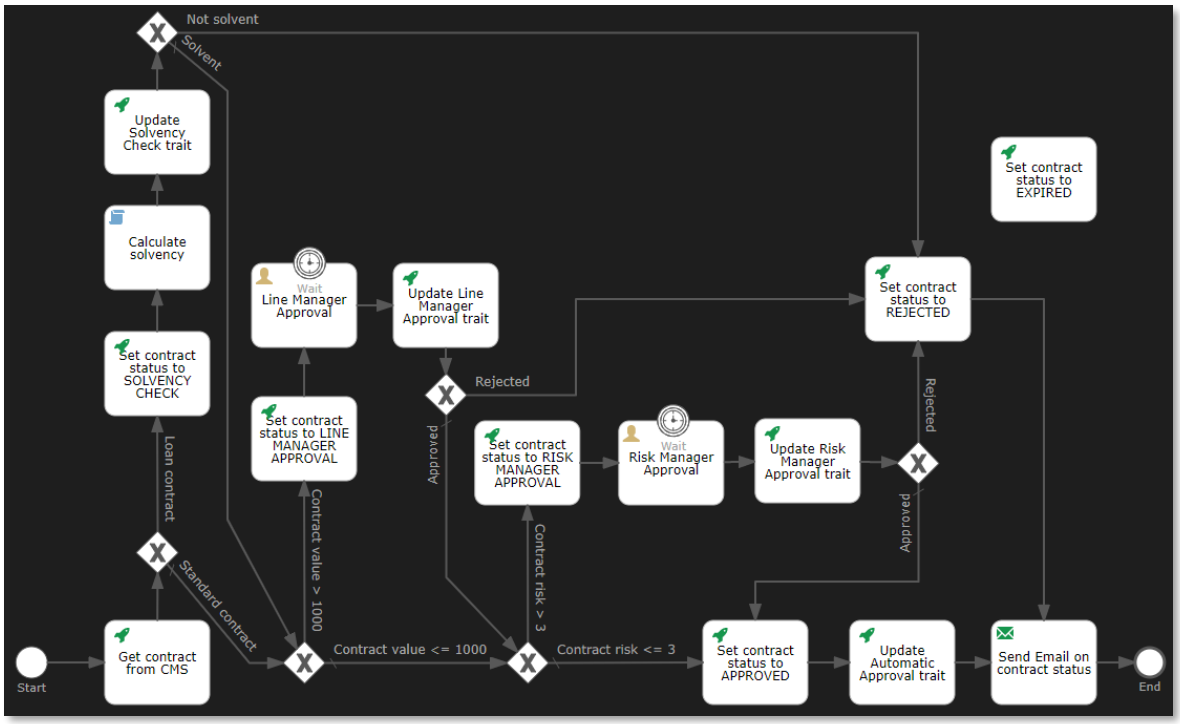
Before you can connect the sequence flow connector arrows from the two manual approval tasks to the “Set contract status to EXPIRED” http task, there is one more thing you need to consider. This is not a normal sequence flow, as it only gets triggered when the manual approval task times out. More specifically, you need to add a boundary timer event to both approval tasks.
To add the “Wait for timeout” timer event to the “Line Manager Approval” and “Risk Manager Approval” user tasks, drag and drop (from the **Boundary Events** section on the palette) a **Boundary timer event** on top of both the **Line Manager Approval** and **Risk Manager Approval** http tasks (which will show green to indicate when the boundary event can be released/dropped to be linked to them). Both start timer events should be aligned to the top of the http tasks, so that they sit half inside and half outside of the http task element.



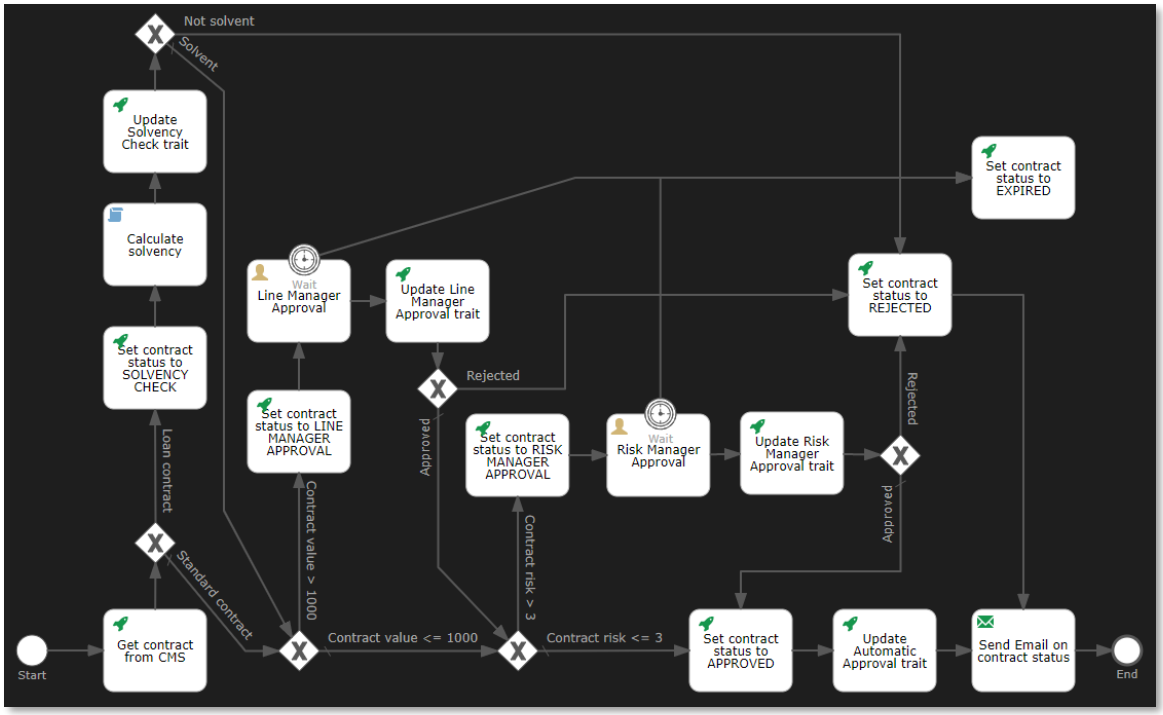
Set the attributes for both start timer events as follows:

- **Name: Wait**
- **Time duration (e.g. PT5M): PT5M**
- **Cancel activity: <checked>**





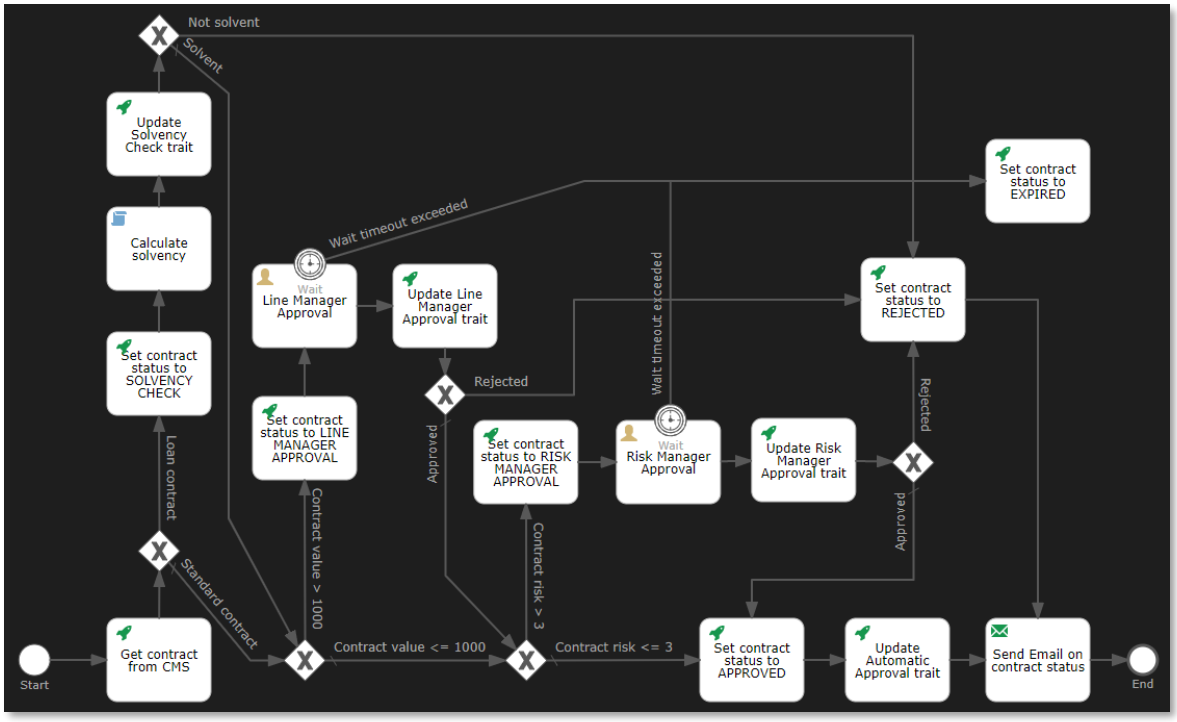
You can now connect both “Wait” start timer events to the “Set contract status to EXPIRED” http task with sequence flow arrow connectors.



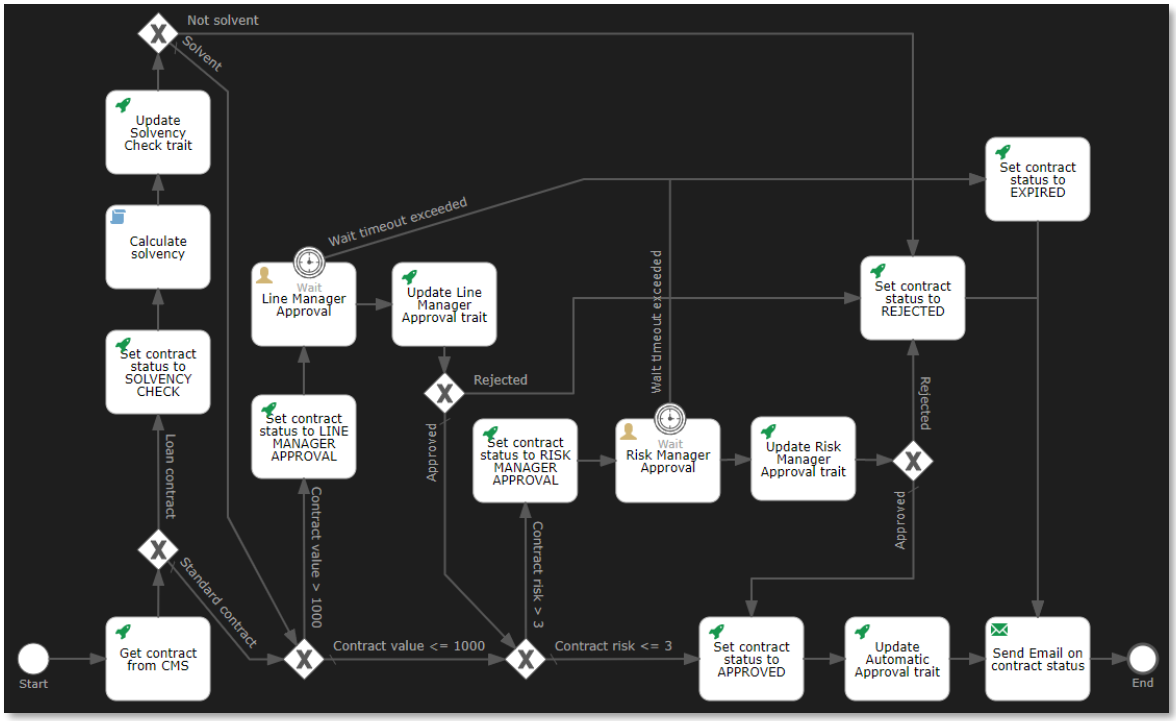
Set the attributes for the two new sequence flow arrow connectors to the following:

- For the sequence flow originating from the “Line Manager Approval” http task:
  - **Name: Wait timeout exceeded**
  - **Flow condition: ${contract.properties.status == "LINE MANAGER APPROVAL"}**

- For the sequence flow originating from the “Risk Manager Approval” http task:
  - **Name: Wait timeout exceeded**
  - **Flow condition: ${contract.properties.status == "RISK MANAGER APPROVAL"}**

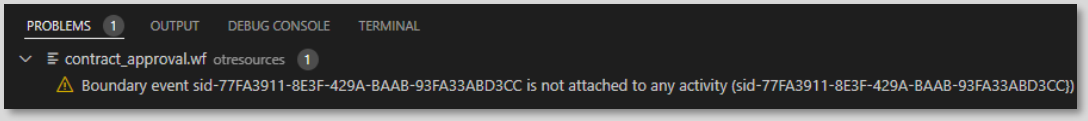


There's one last sequence flow arrow connector to add. To ensure the EXPIRED contract approval status email gets sent as well, connect the "Set contract status to EXPIRED" http task to the "Send Email on contract status" task.



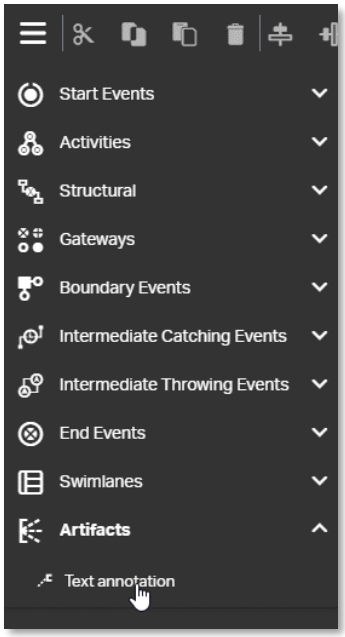
Please save the workflow and verify that there are no errors in the **PROBLEMS** tab.

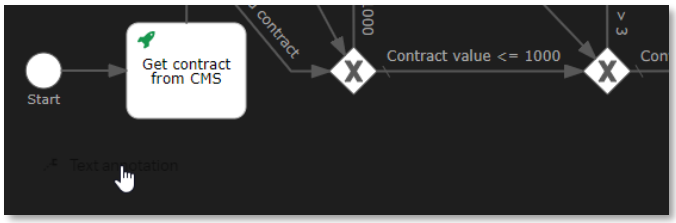
WARNING:
You might get the following warning message in the **PROBLEMS** tab (and corresponding highlighted boundary timer event in the workflow designer canvas):

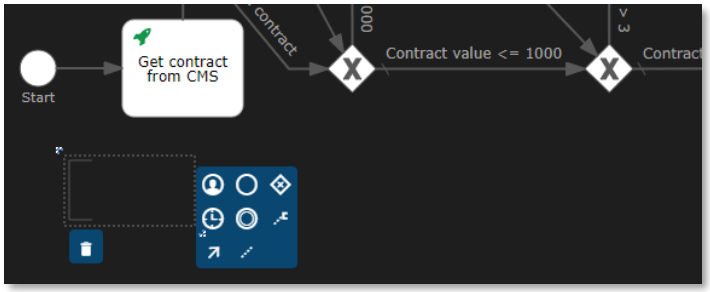


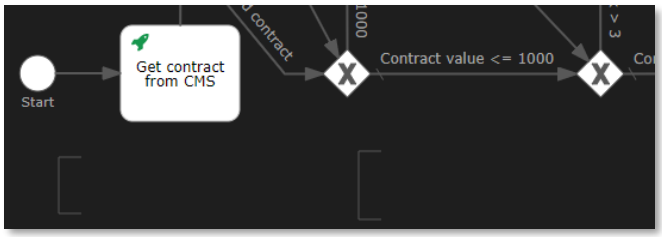
This is most likely caused by an error in the Contract Approval workflow model, where the **Boundary timer event** is not properly connected to the related **User task** (i.e.: the **Line Manager Approval** and/or **Risk Manager Approval** task). To solve this, open the workflow model and move the **Boundary timer event** on top of the **User task** and make certain the **User task** highlights green before you release it. You can even wiggle the **Boundary timer event** around a little after you dropped it to make sure the **User task** indeed lights up as green again (it has to still show a green border, if it doesn't show green the **Boundary timer event** is not properly connected). Save the changes to the workflow and confirm the **PROBLEMS** tab does no longer contain the error.

You don't have to do this last step, but as the proverbial cherry on the workflow model cake, we'd like to add some additional information to the workflow that describes the workflow input and, specifically in case of this contract approval workflow model, the different contract states.
Drag and drop two **Text annotation** elements from the **Artifacts** section of the palette onto the bottom left of the canvas.







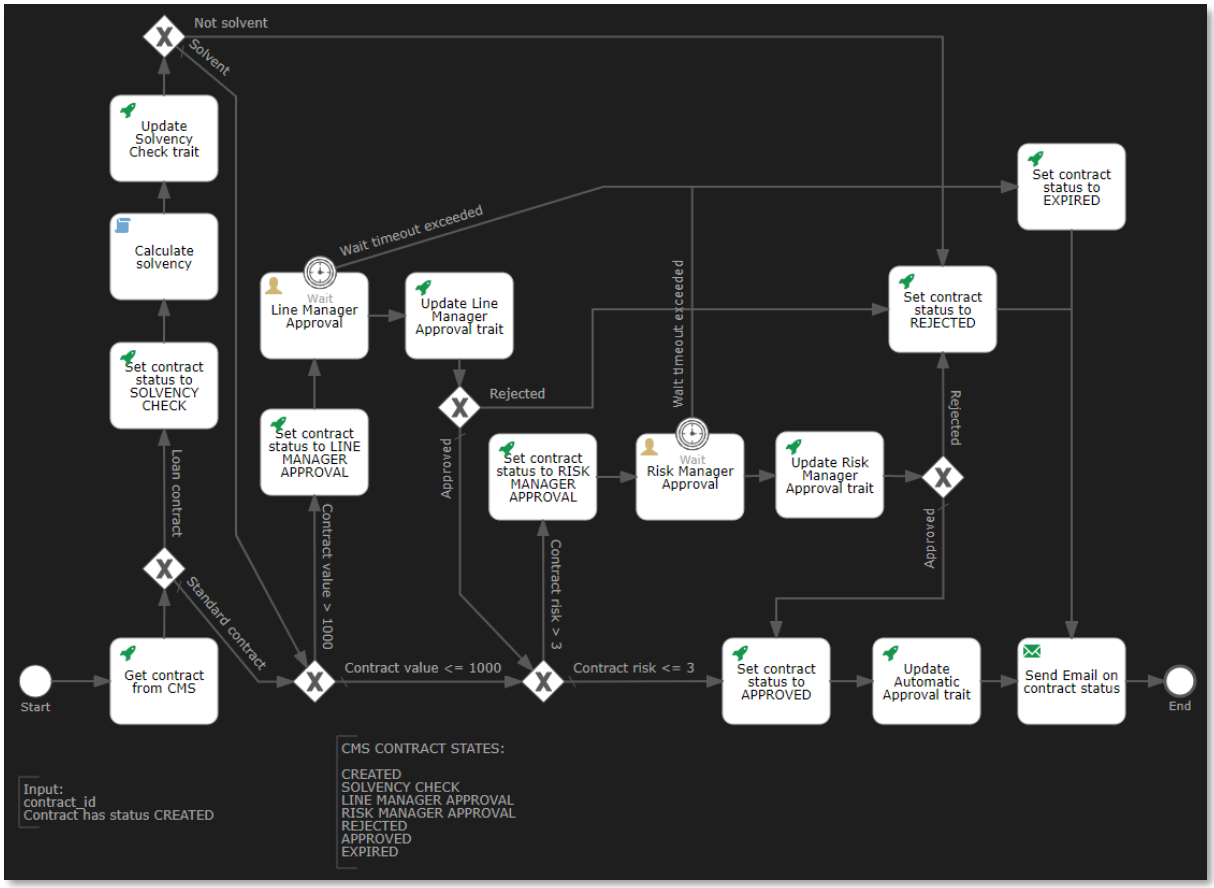


From left to right, double-click and fill the text annotations with the following text (i.e.: information for the developer looking at the workflow model):

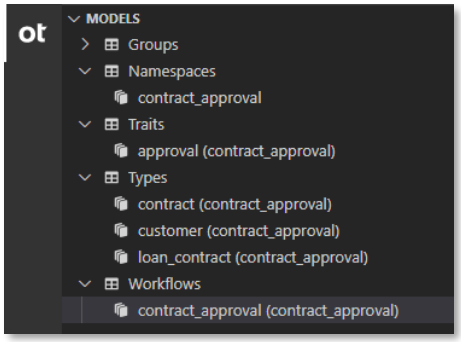
- First text annotation:
  Input:
  contract_id
  Contract has status CREATED
- Second text annotation:
  CONTRACT STATES:

  CREATED
  SOLVENCY CHECK
  LINE MANAGER APPROVAL
  RISK MANAGER APPROVAL
  REJECTED
  APPROVED
  EXPIRED

If the text doesn't fit, you can drag the **text annotation** element's bottom right or top left corner to expand it.



Your **Contract Approval** workflow model is now complete. Please, save it one last time, and close the workflow editor.
If you switch to the **OpenText Cloud Developer Tools** view, the model explorer tree in the **MODELS** section should now show your new **contract_approval (contract_approval)** workflow model under **/Workflows**.



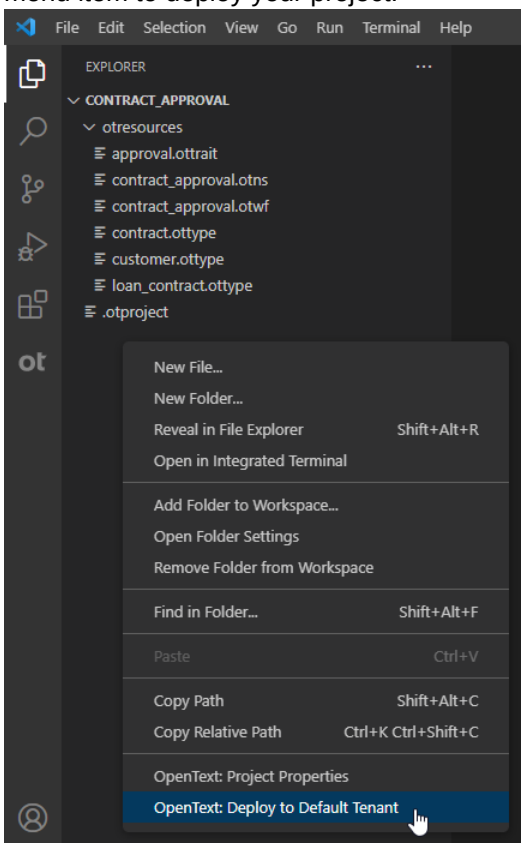
3.10[10’] Deploying the application to the IM services

During this exercise you will deploy your **Contract Approval** application project (i.e.: its models) to the different IM services. Thanks to the organization connection you previously created, this is as simple as selecting the “Deploy Project” contextual menu from your project root folder.

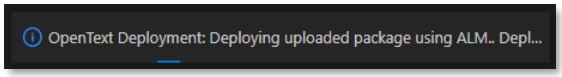
Once you are done with this section, you will have deployed the **Contract Approval** application into your developer organization’s single tenant, and you are ready to verify the deployment of the different models, using the IM APIs via Postman.

To deploy the **Contract Approval** application project, proceed as follows:

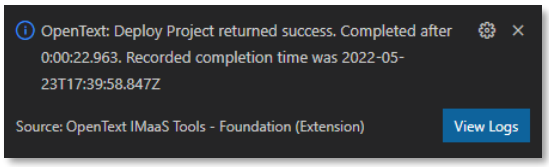
- In the **Explorer** view of VS Code, right-click inside your **contract_approval** project root folder (in the empty area to avoid selecting a sub folder or file) and click the **OpenText: Deploy Project** menu item to deploy your project.

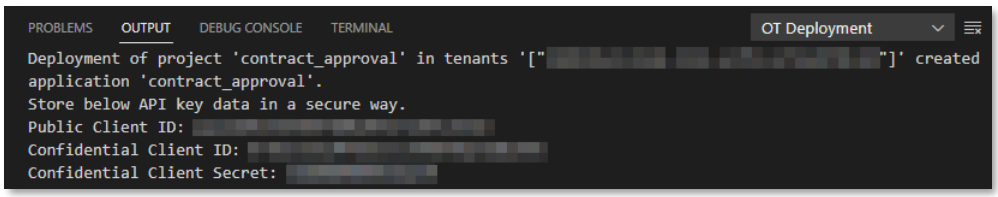


REMARK:
Since you have tested your connection at the beginning of this tutorial, it can be that the connection (i.e.: authentication token) has expired. In that case you will see the system will automatically reconnect first (which launches a web browser window) when clicking **OpenText: Deploy Project**.

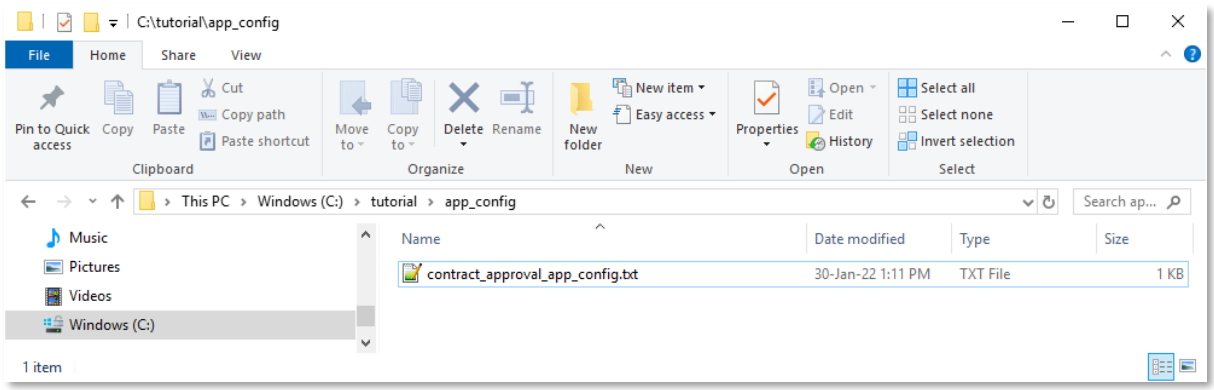


After successfully deploying, the system pops up a “Deploy Project returned success” message, and the **Output** view automatically opens to display the **OT Deployment** output. It contains the tenant ID and (application) API key data.



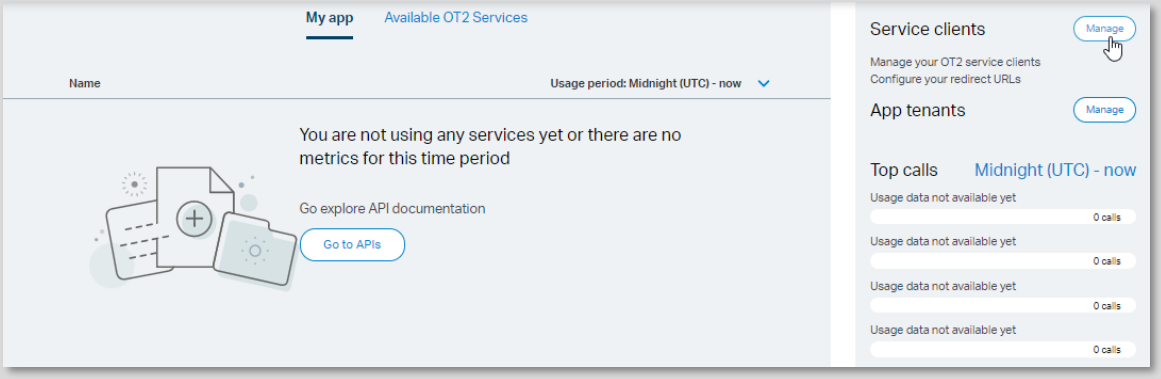


- You need to store the **OT Deployment** information, as you will need it when verifying that your application models are correctly deployed. You will also need this when developing the actual application (i.e.: the Contract Approval application code).
  Copy the text from the **Output** view window into a **contract_approval_app_config.txt** text file and save it (e.g.: under a new **app_config** folder).

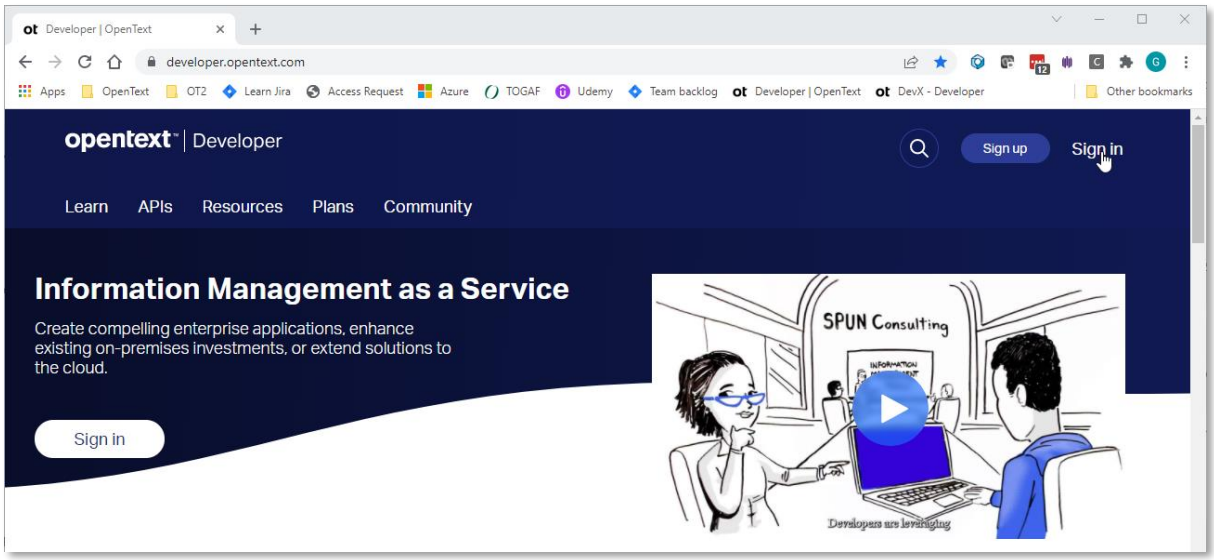


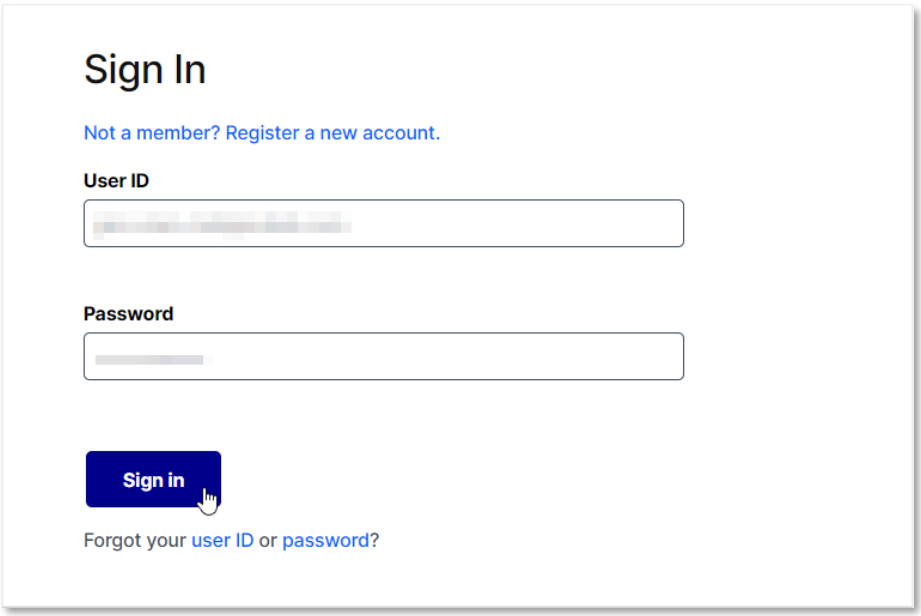
IMPORTANT:
Although we are using unencrypted/insecure text files to store the different key and password information for the purpose of this tutorial, it is of course recommended for real life scenarios to store any API key or password information in a secure way.

REMARK:
If you somehow missed the opportunity to copy/save the **OT Deployment** information (Tenant ID and client credentials), you can always retrieve the Tenant ID from your developer organization's console (see here how to access the console). The Tenant ID can be copied from the Tenant Info screen, and the client credentials can be regenerated from the App Details:

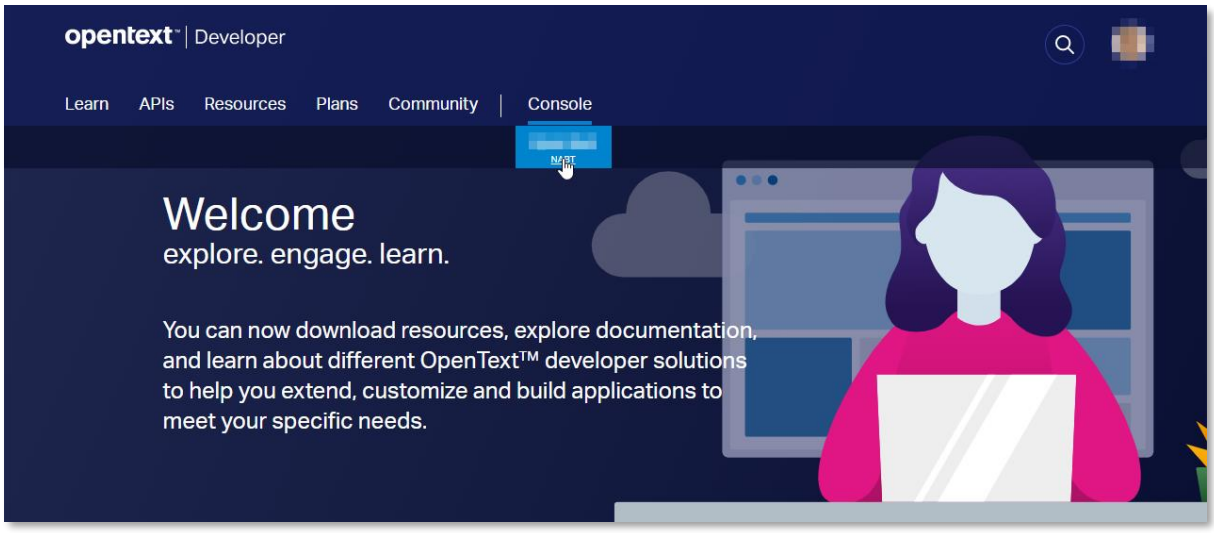


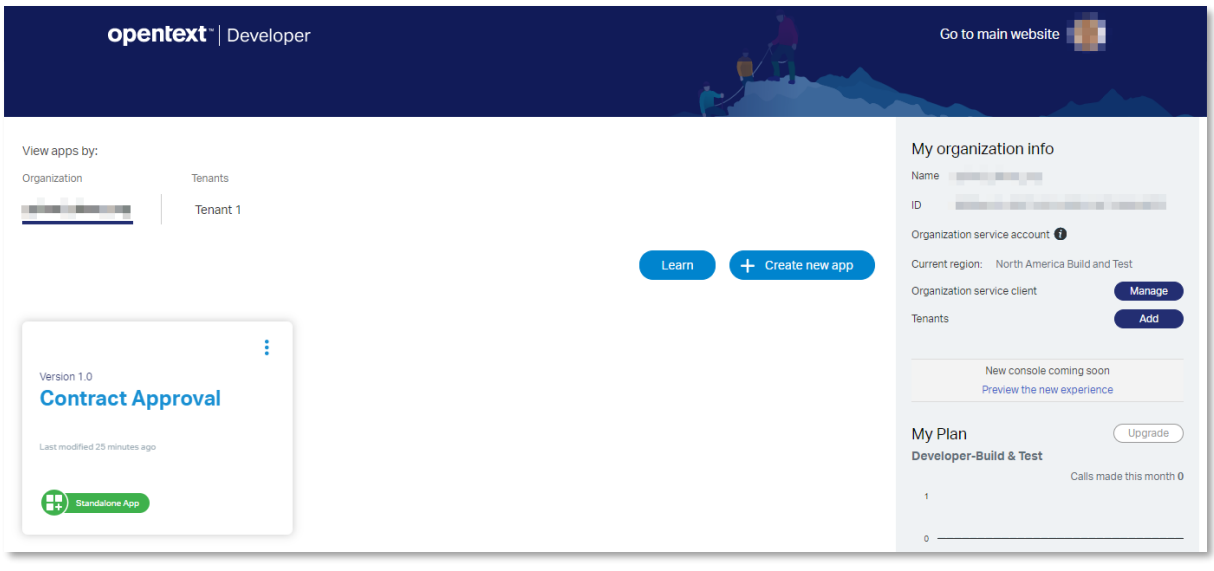
- To validate that deploying your application has successfully added an application entry in your developer organization, use your web browser to navigate to **developer.opentext.com** and log in with your developer (trial) account user.



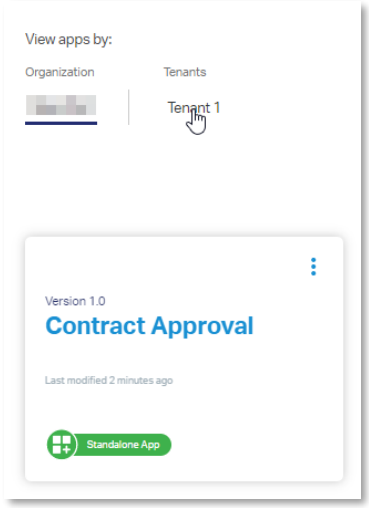


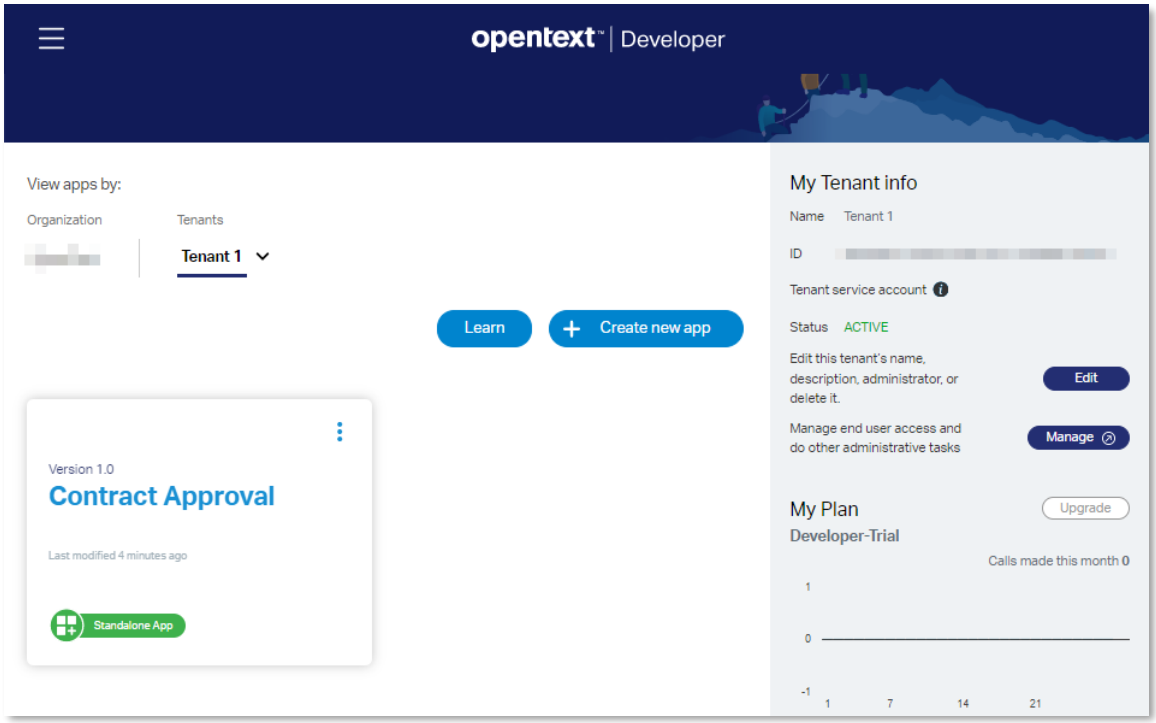
Open the **Console** for your developer organization.

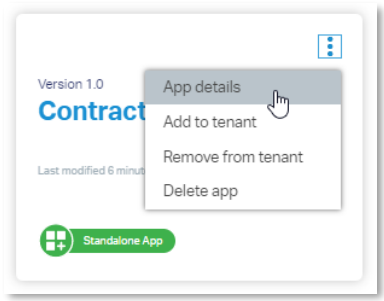


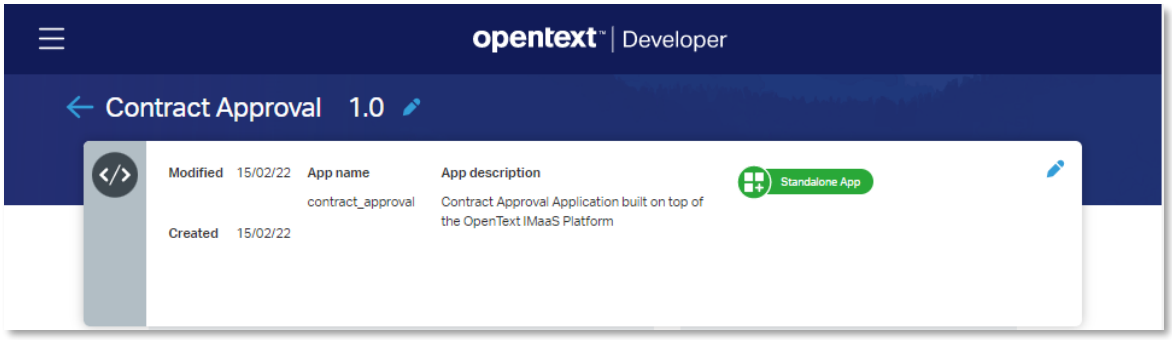


Confirm you can find your application deployed under your organization's single tenant **Tenant 1** and have a look at the **App details**.









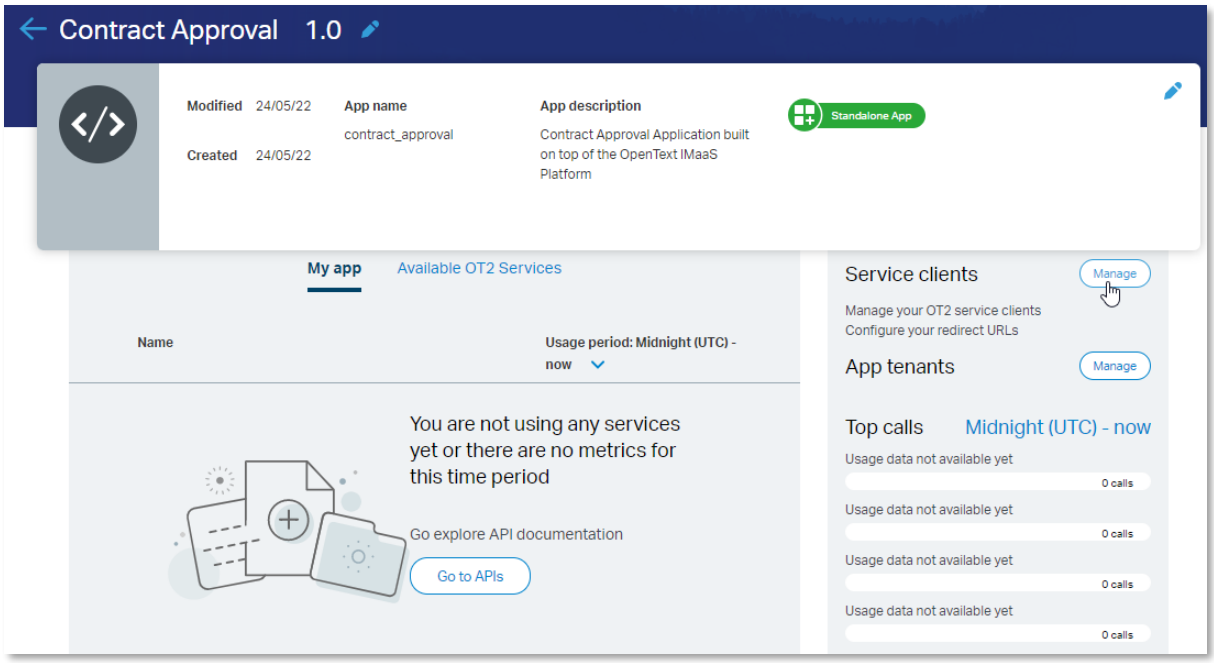
This confirms your application has been deployed and is now available in your **Tenant 1** developer tenant.

Adding the redirect URL for your application authentication flow

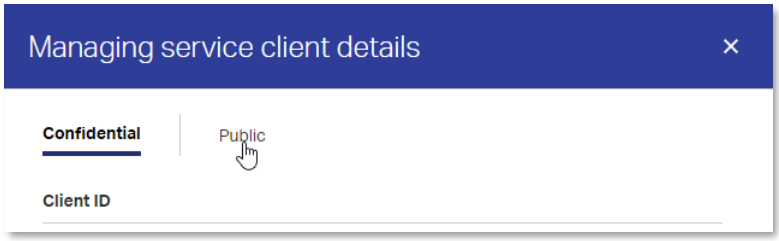
To authenticate, users of your Contract Approval application will need to sign in using an external (to the application) login screen as part of the authentication (OAuth) flow. This means the web browser in which you open the Contract Approval application will first be redirected to a login screen,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

Proceed as follows:

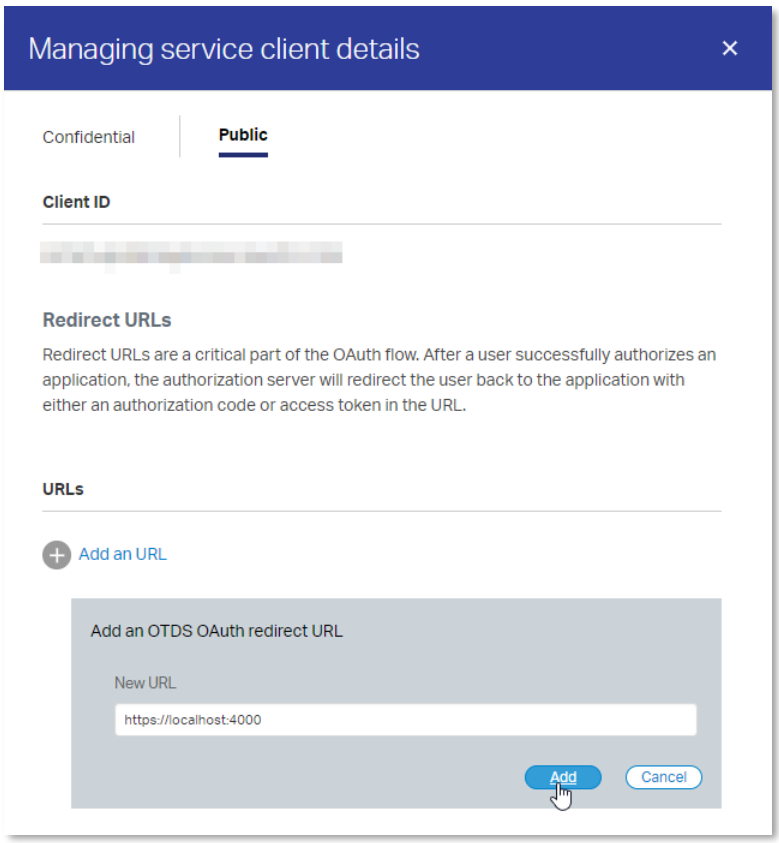
- From the (currently open) Contract Approval application details page, click the **Service clients Manage** button.



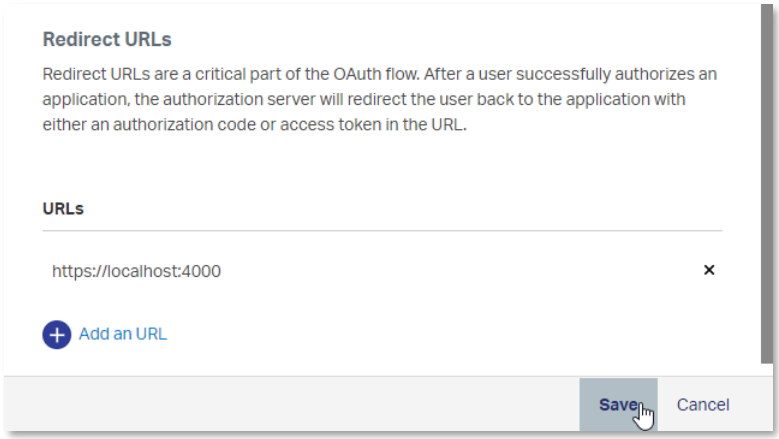
Select the **Public** tab.



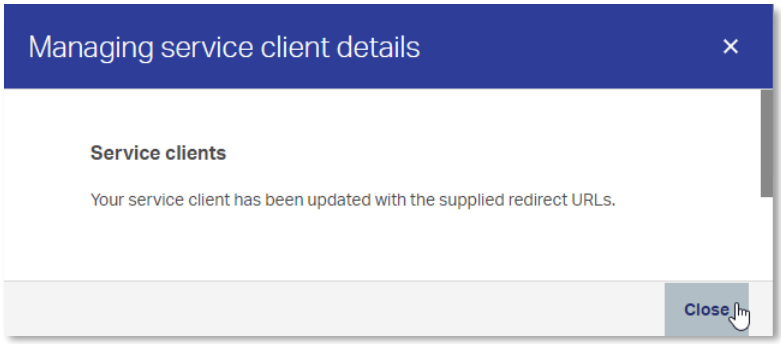
Select **Add an URL** (under the **URLs** section), fill **https://localhost:4000**.and click **Add**.



Choose **Save** to confirm your changes.



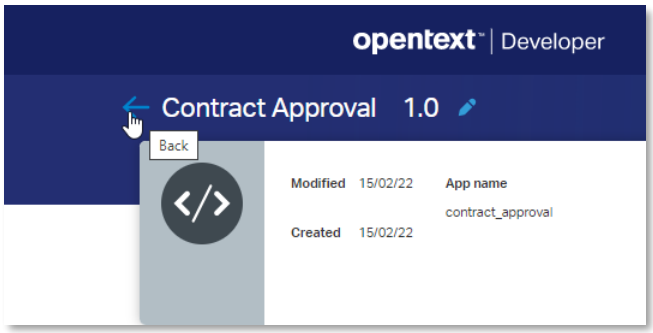
You can click **Close** to return to the App details screen.

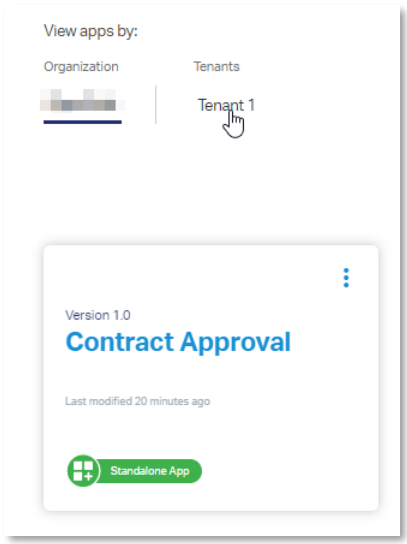


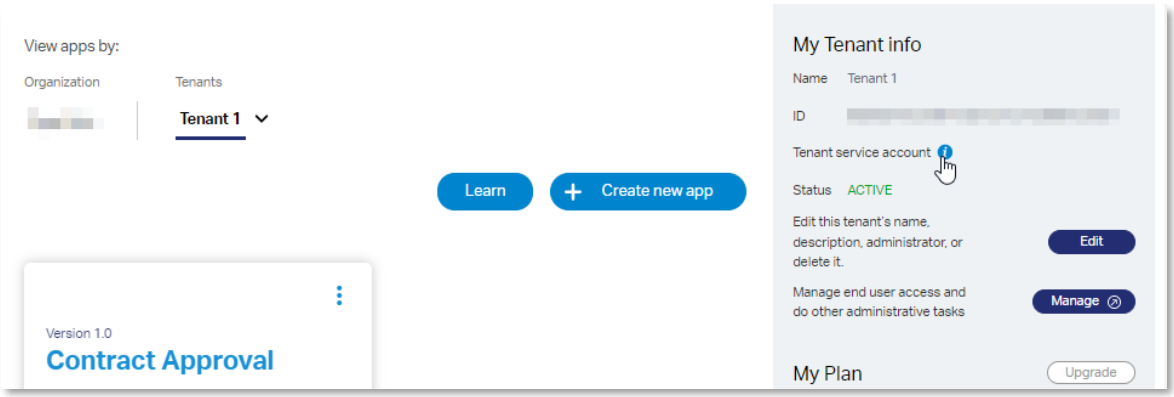
Resetting the Tenant password

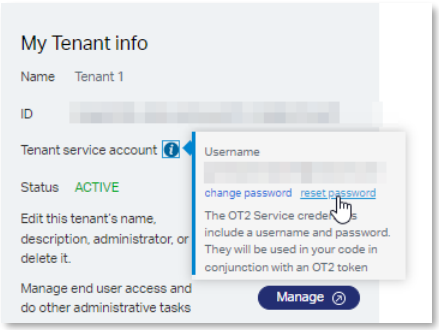
- This is also the right time to (re)set your **Tenant 1** password. You need to do this, as you have never actually set this password. Although the developer tenant has been generated alongside your developer organization, for security reasons, you need to set a password specifically at the tenant level as well.

  Click the  to go back to your organization view, make sure to select the tenant view, and click  next to **Tenant service account** from the **My Tenant info** pane to request a password reset.

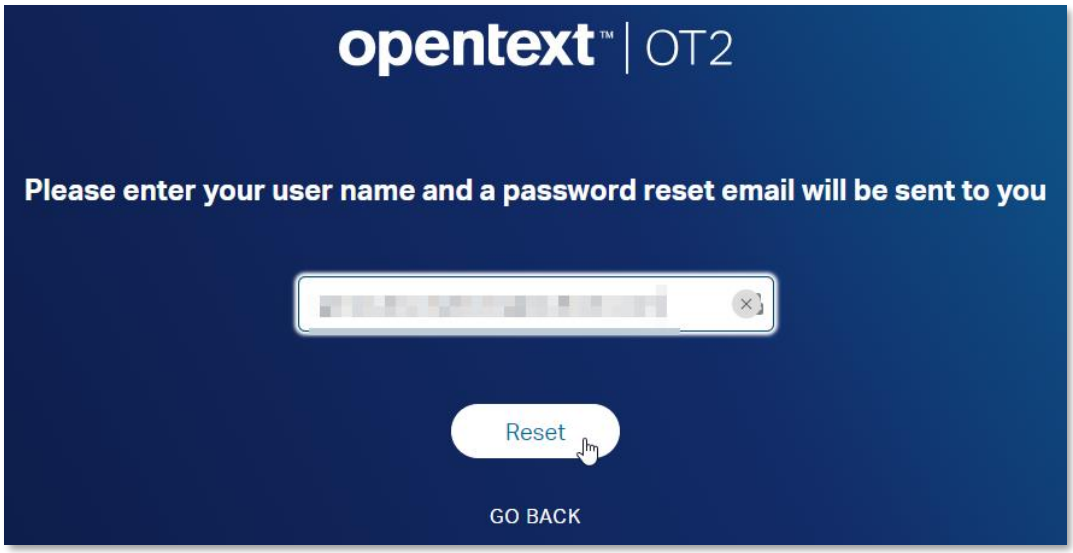


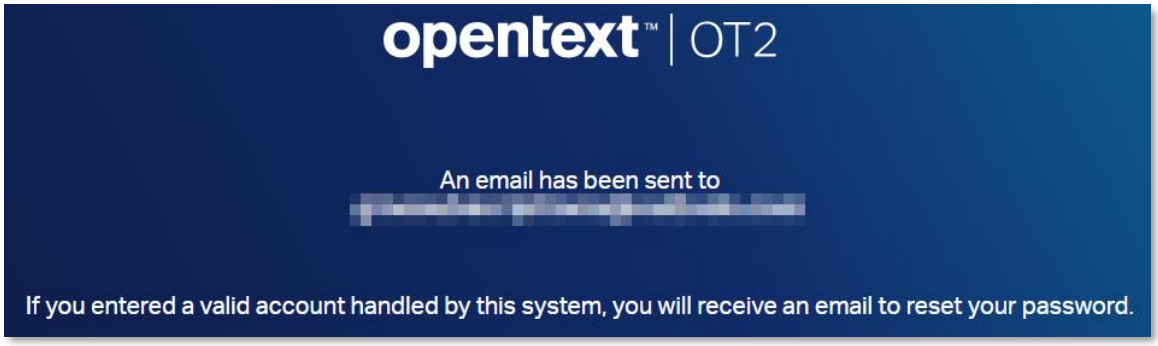




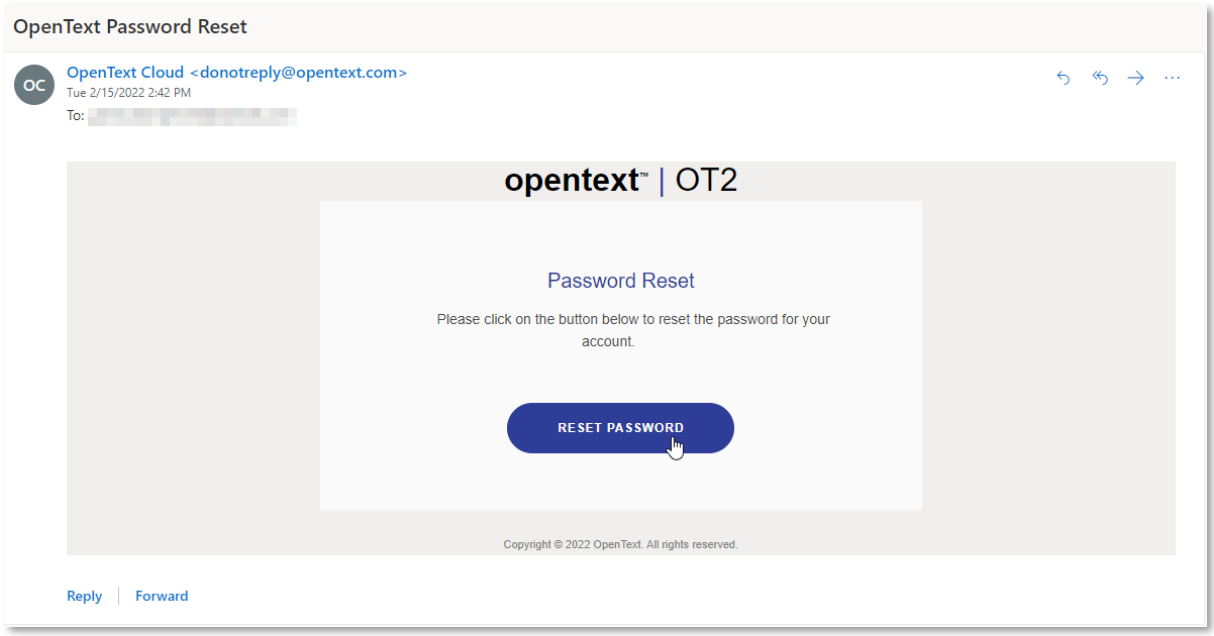


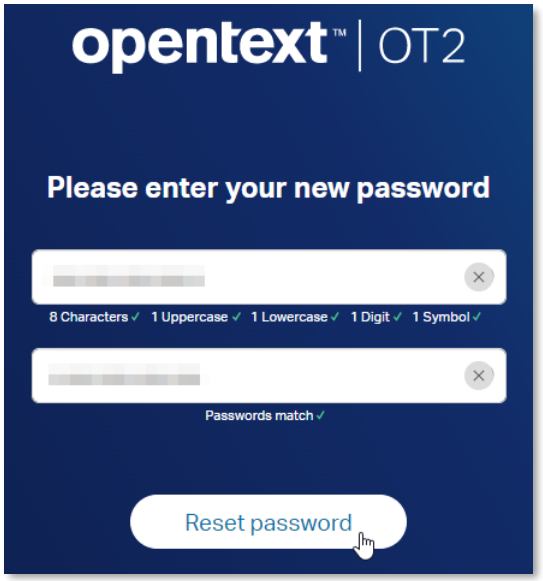
Fill the email address corresponding with the (tenant service account) **Username** and click **Reset**.





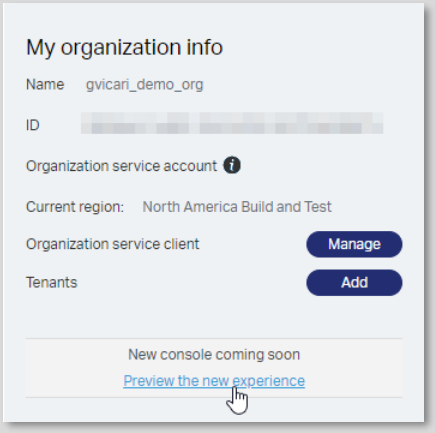
Go to your email client and open the **Password Reset** email you just received from **OpenText Cloud** and choose your new password.

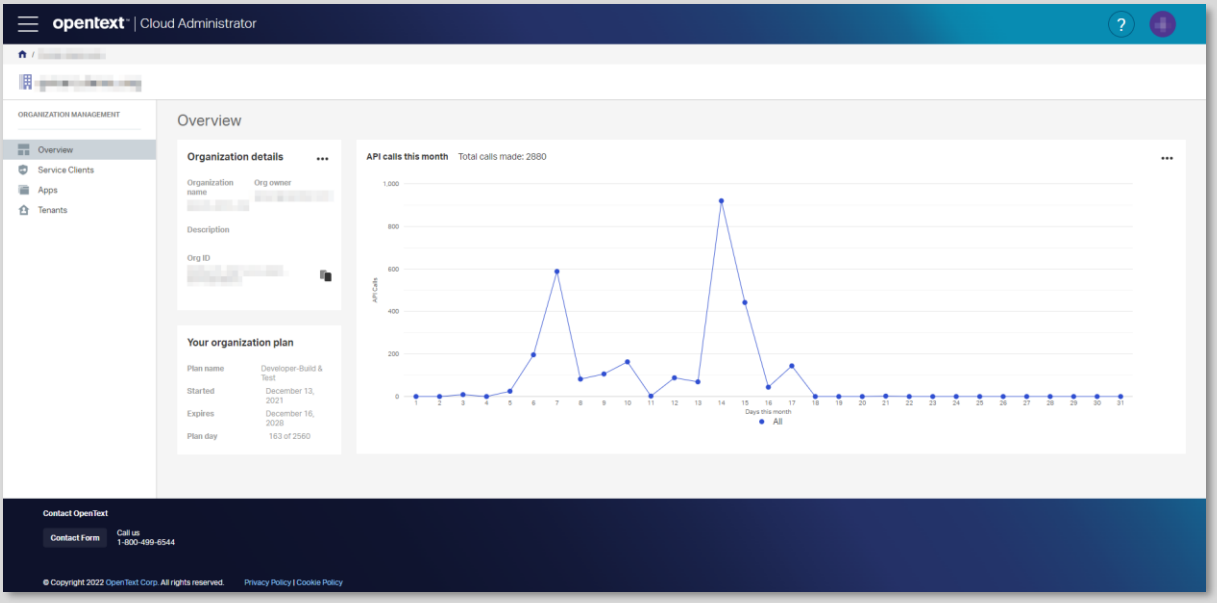




- Once you have chosen your new password, please save it for later use. You can simply add this new tenant password to the previously created **contract_approval_app_config.txt** text file.

REMARK:
The **Console** offers the administration view and capabilities for your OpenText Cloud Developer experience. Throughout this chapter you have been using it for verifying the successful deployment of the Contract Approval application and resetting your tenant password. Note that a new and improved version of the console exists and is currently available for preview (i.e.: you can try it out). To access this preview version of the new **Console**, click the **Preview the new experience** link from the organization information screen (under **New console coming soon**).





The new console provides a more hierarchical view and a more logical way of navigating the different administration artifacts and capabilities. We recommend you try it out.

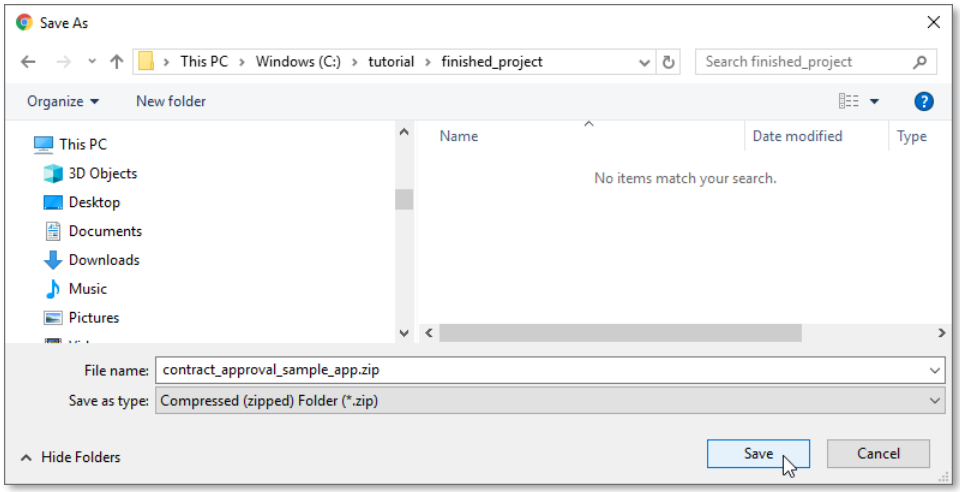
3.11[25'] Working with the IM APIs

During this exercise you will use the CMS and Workflow Service IM APIs (via Postman) to verify that your application models have been correctly deployed. For more information on the CMS and Workflow Service APIs, you can refer to their API reference documentation, respectively CMS API reference and Workflow Service API reference.

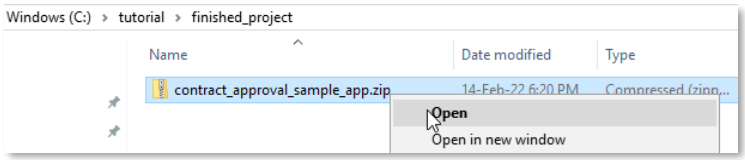
Once you are done with this section, you will have confirmed that your application models have indeed been correctly deployed and you will have a good understanding of how to use the IM APIs, so that you are ready to start building (i.e.: writing the code of) your Contract Approval application.

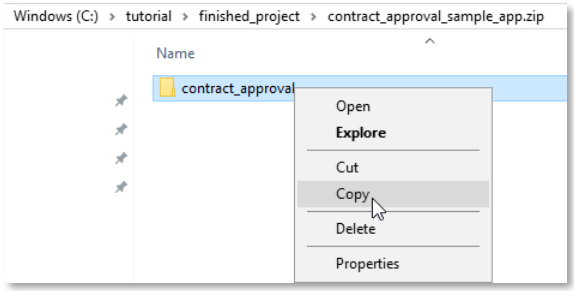
To verify the deployment of your application models, proceed as follows:

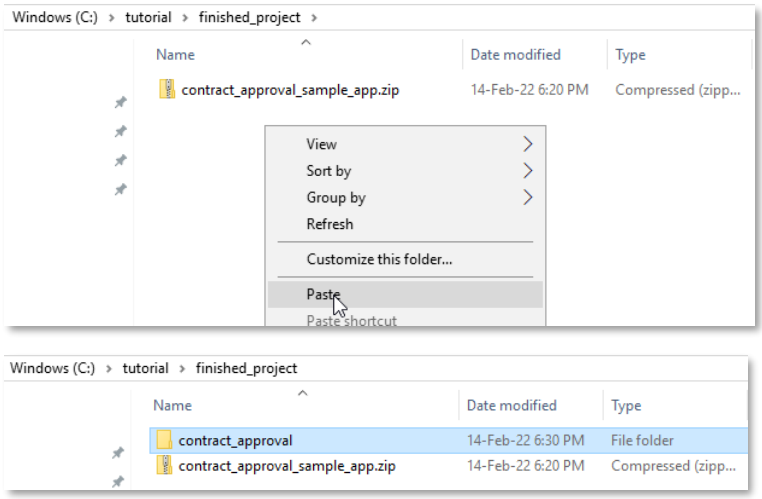
- Postman is a very popular API testing tool, and it is available for download from the following link: https://www.postman.com/downloads. If you don't already have it installed on your computer, please download and install the version that fits your system.
- You will also need to download the finished version of the application you are building, as it contains the Postman collection and Postman environment that you will be using to test that your models have been correctly deployed. You can download the full Contract Approval application (ZIP file) through this link (e.g. into a **finished_project** folder).



Extract (copy from within the ZIP file) the **contract_approval** application project folder.

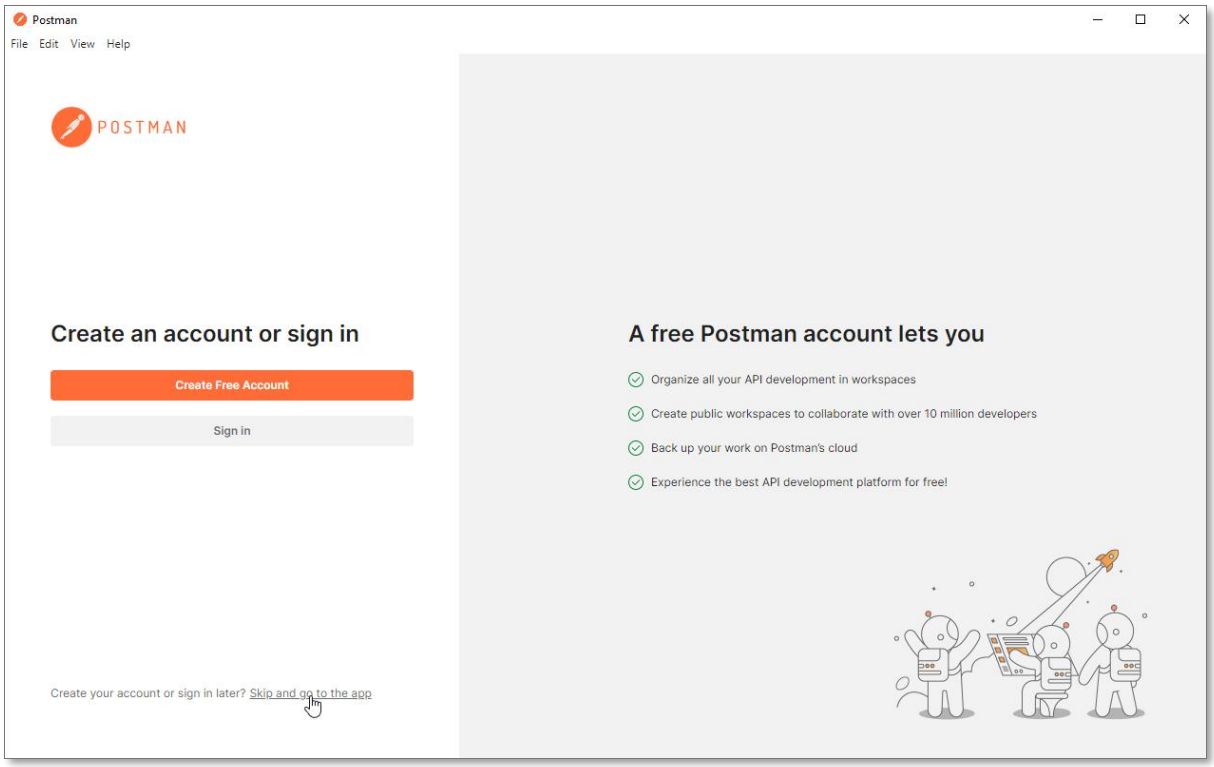


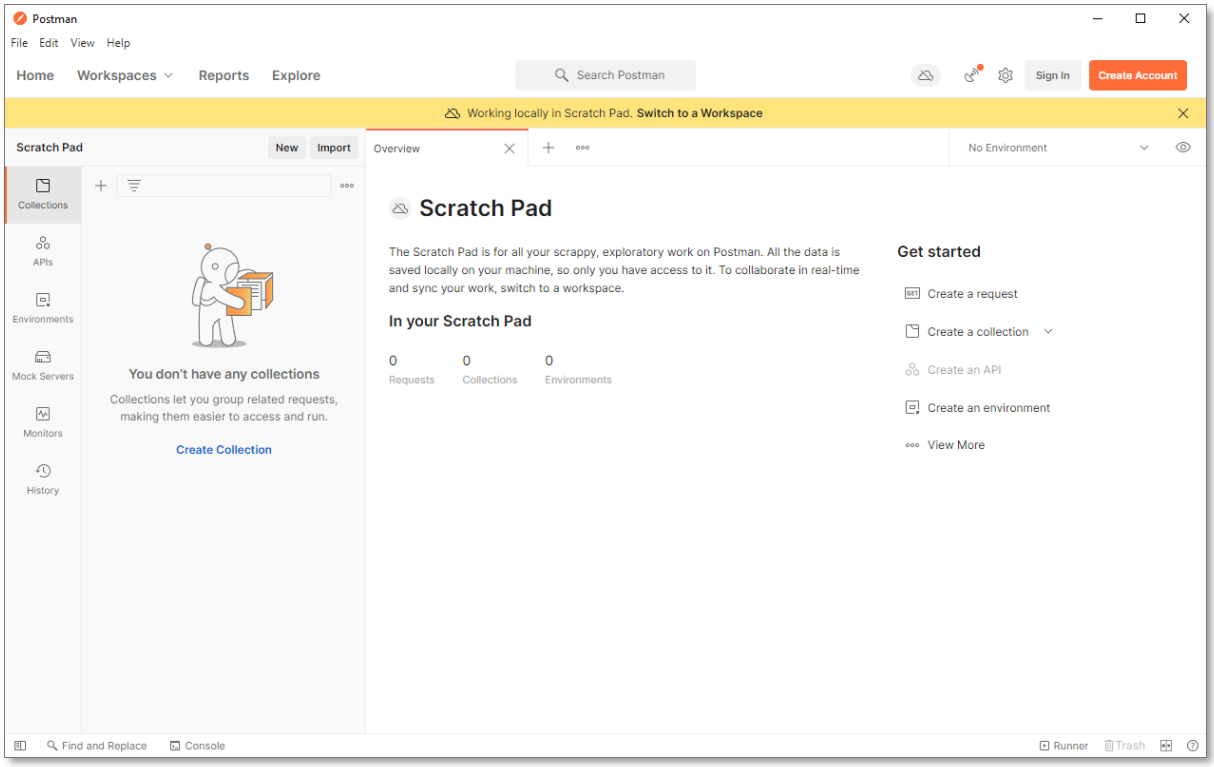




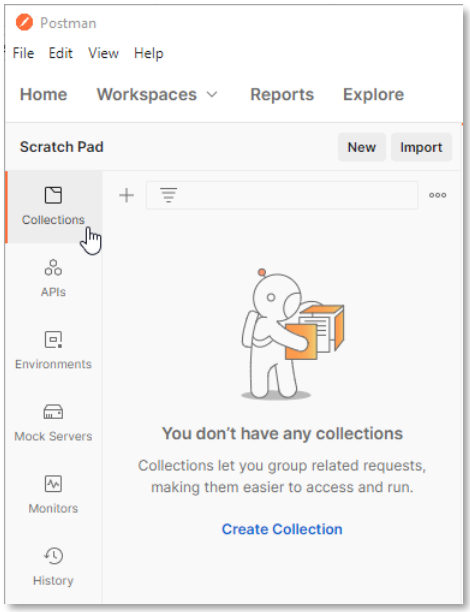
Feel free to delete the ZIP file after extracting the contract_approval folder.

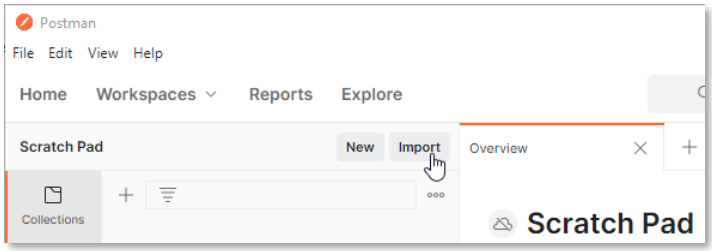
- After downloading and extracting the finished Contract Approval application project, we can now configure Postman to work with the OpenText CMS & Workflow Service APIs. This is done by importing the Postman collection and Postman environment into your Postman application. First, open Postman (the screen shots are for a freshly downloaded and installed Windows 64-bit version).



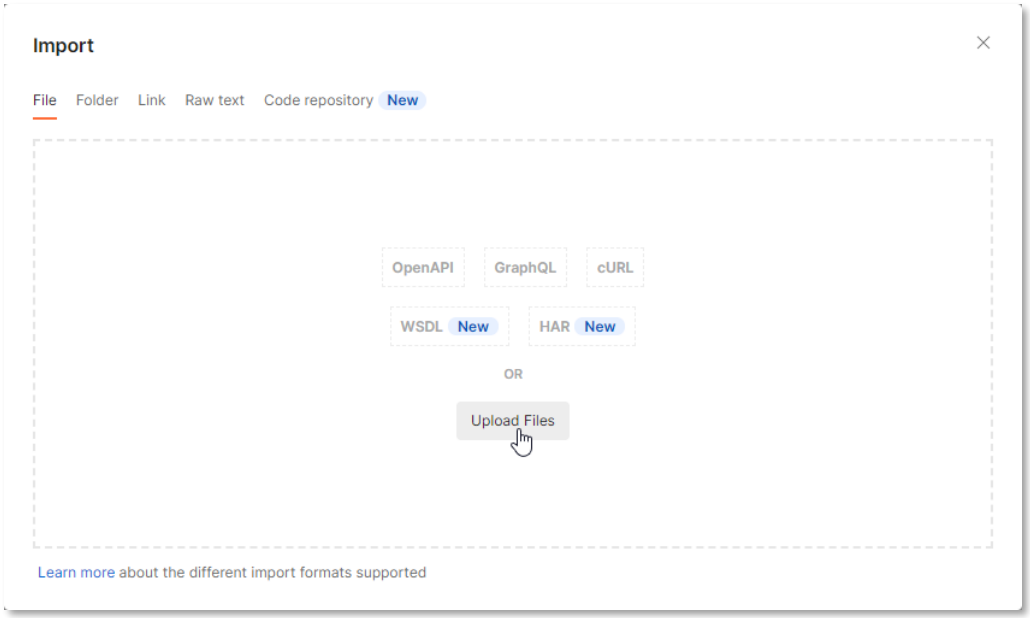


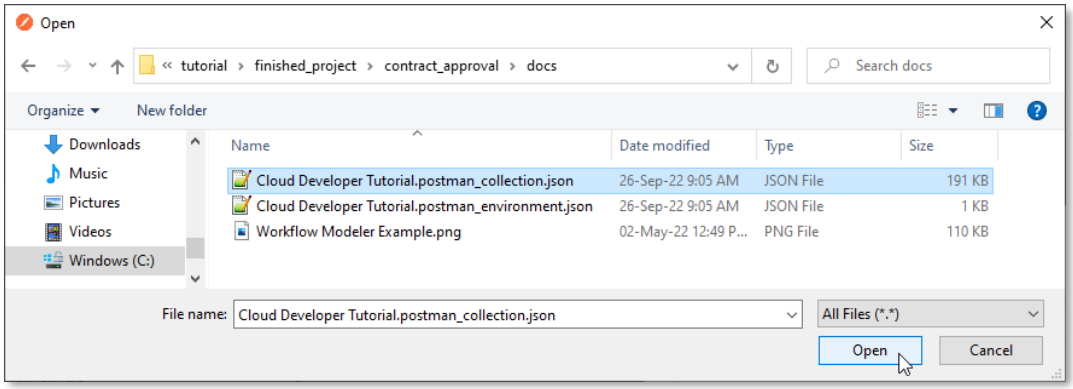
To import the "Cloud Developer Tutorial" collection, make sure to select **Collections** from the left sidebar and click **Import**.

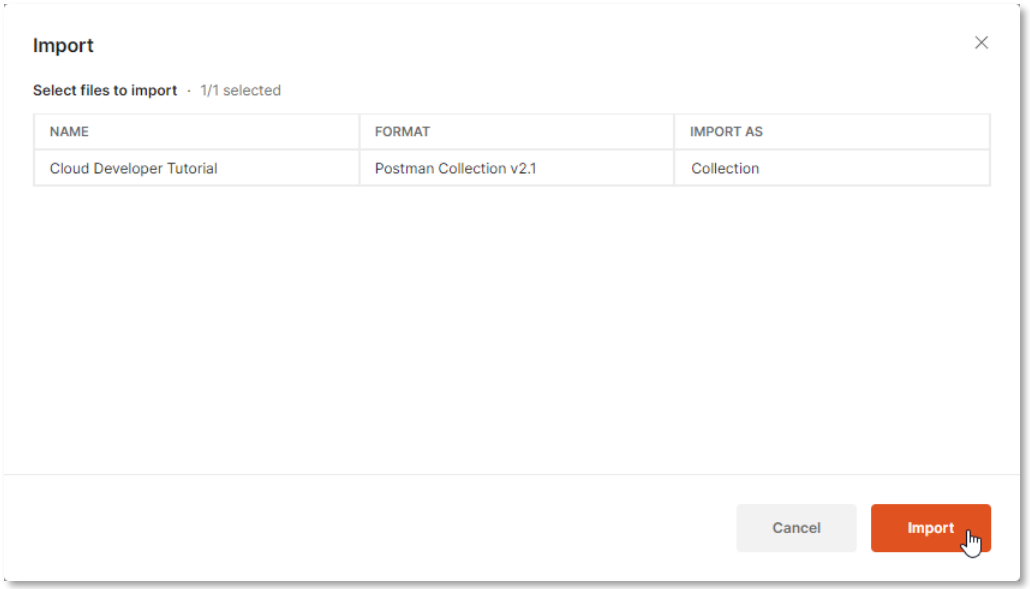


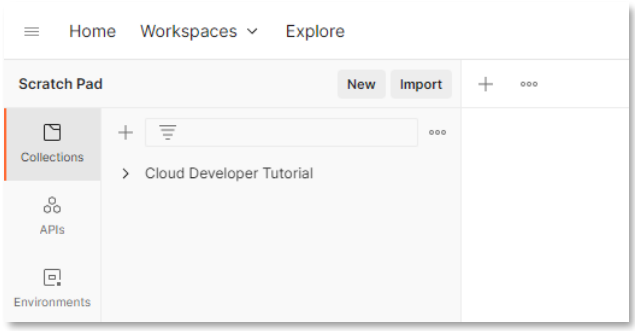


From the **Upload Files** dialog, navigate to the **/docs** folder in the **contract_approval** folder you just extracted and choose to import the **Cloud Developer Tutorial.postman_collection.json** collection file.

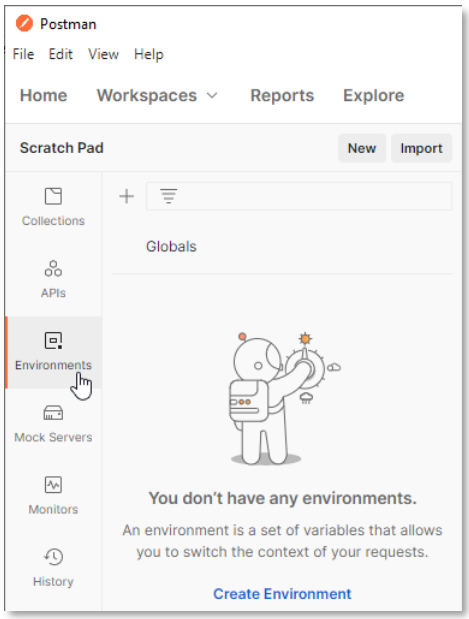


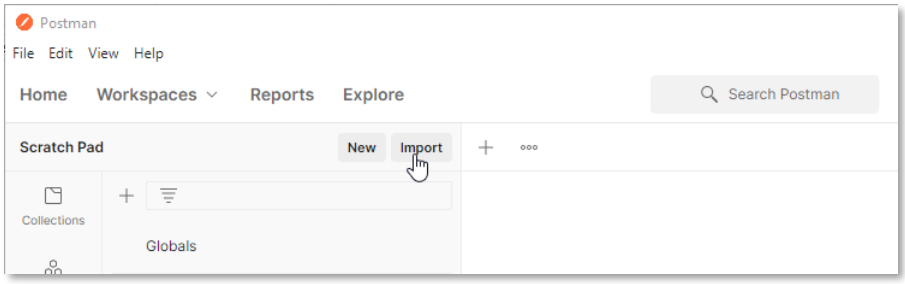


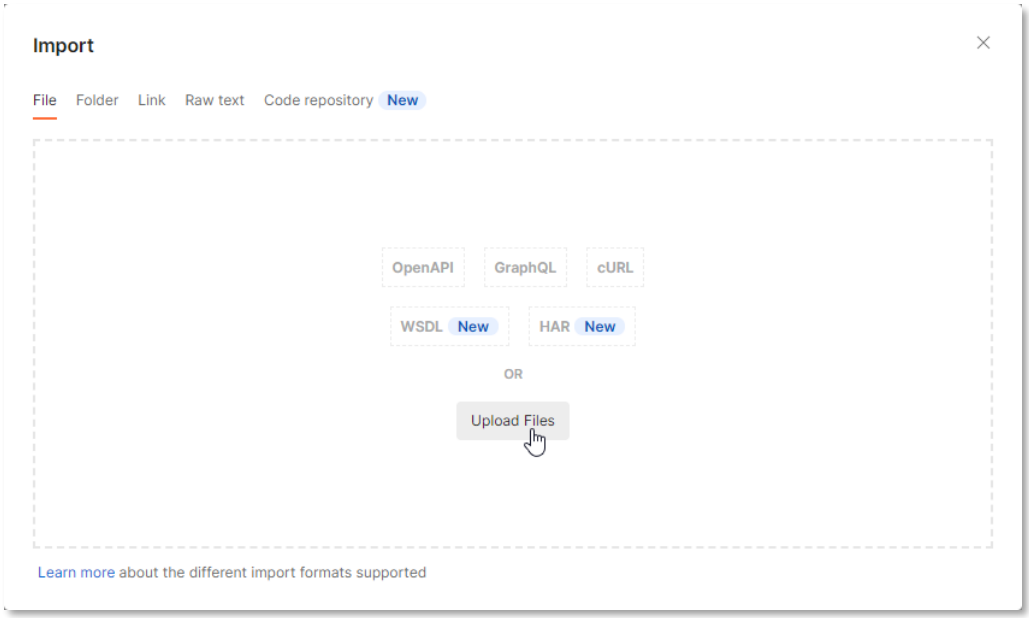


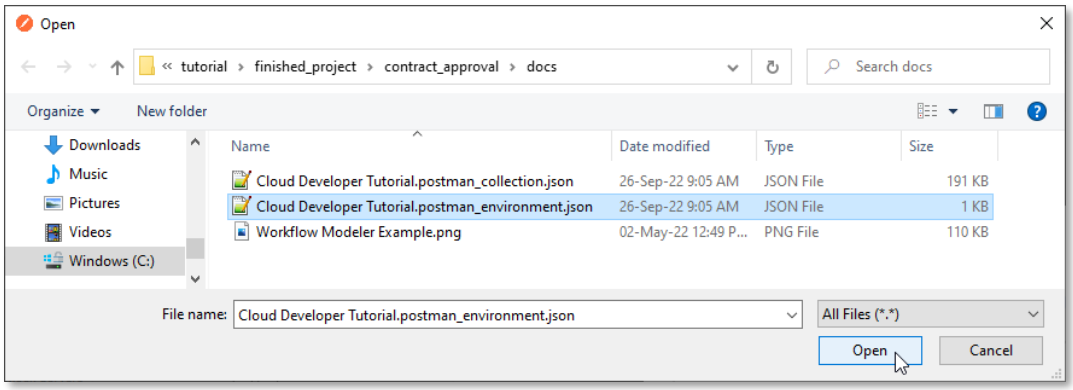


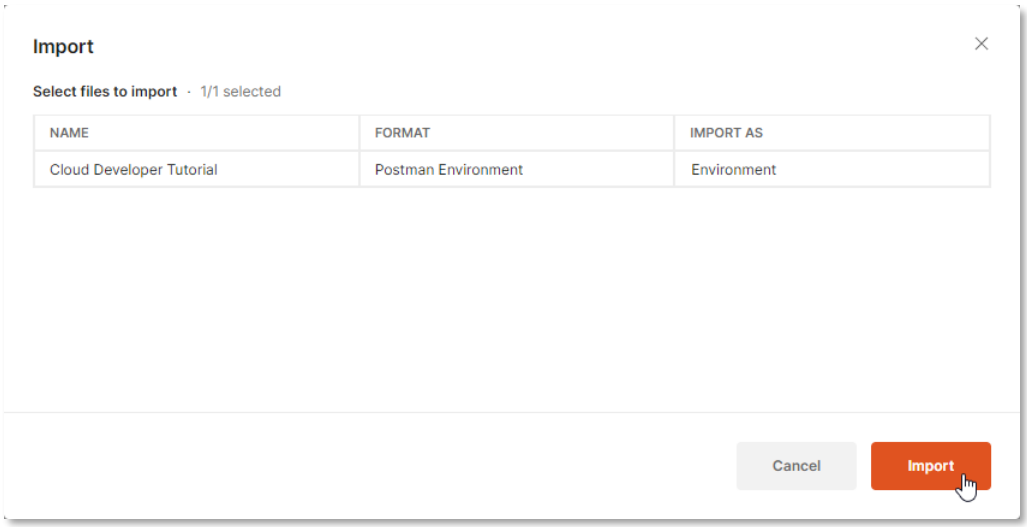
Similar to importing the “Cloud Developer Tutorial” collection, import the “Cloud Developer Tutorial” environment by clicking **Import** after having selected **Environments** from the left sidebar.

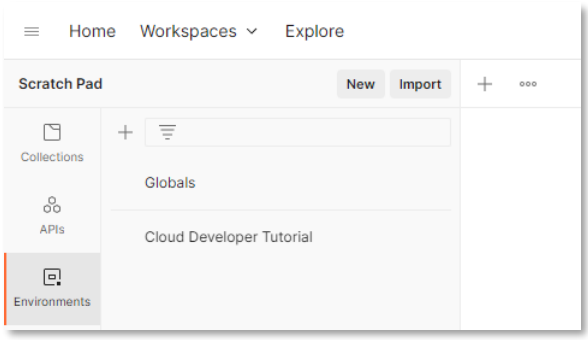




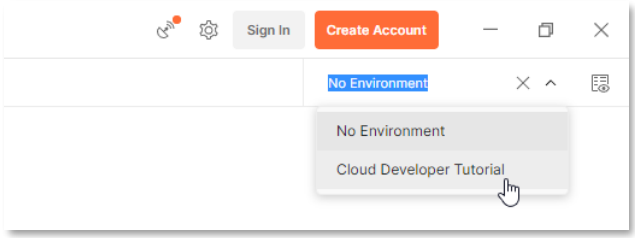


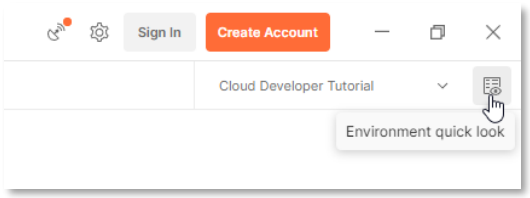


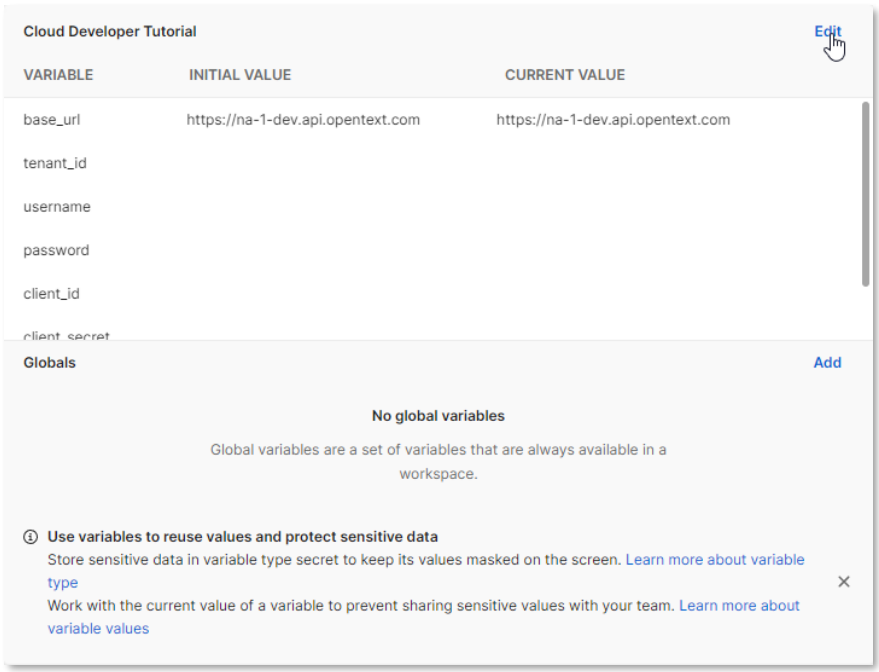


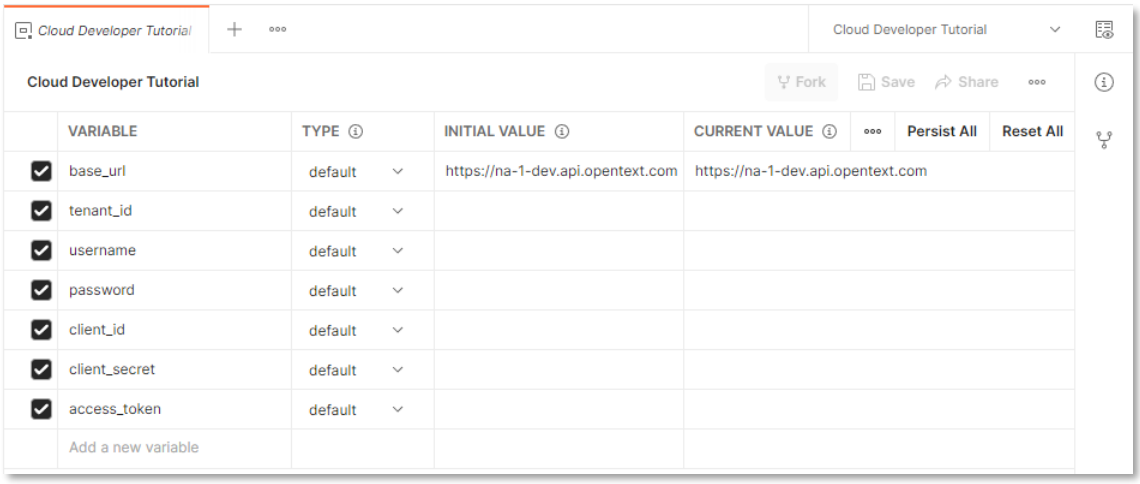


- Before you can call the IM APIs to verify your deployment, you must fill the Cloud Developer Tutorial environment's environment variables with the values that correspond with your developer organization/tenant and deployed application.





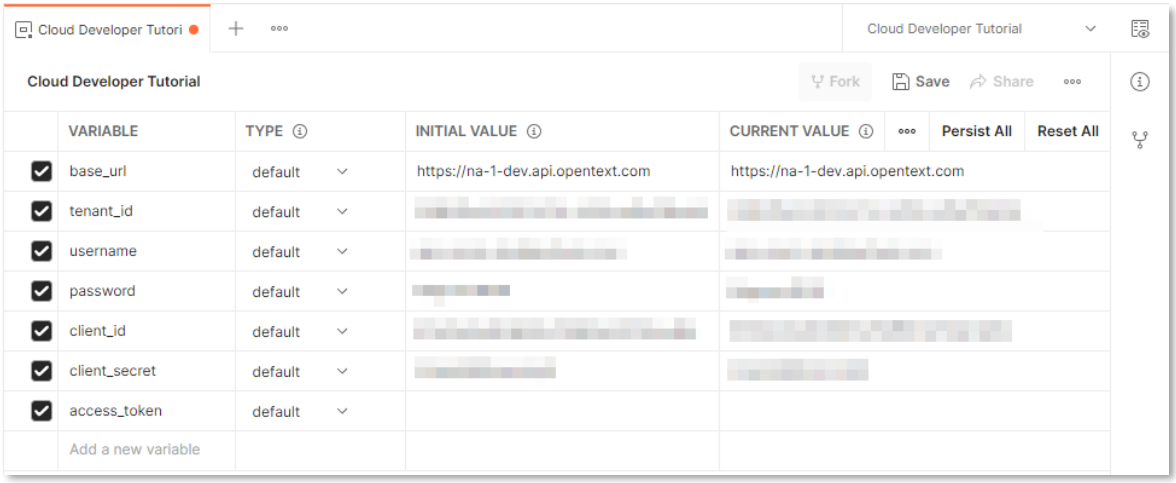




Fill both the INITIAL VALUE and CURRENT VALUE columns for each of the following environment variables (the ones that are not mentioned, you must leave as is):

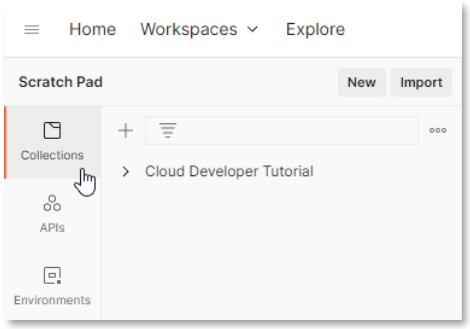
- **tenant_id:** use the tenant id (from text **in tenant '<tenant id>'**)you saved after deploying the application project
- **username:** use the email address you used for your tenant service account
- **password:** use the password you saved for the tenant service account (after having reset it)
- **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

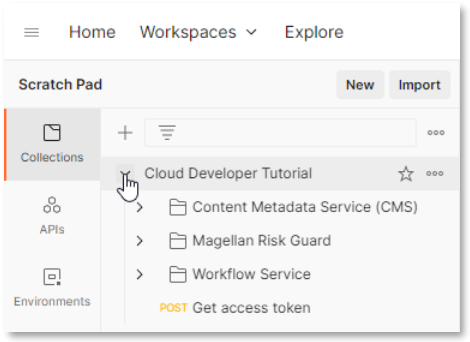
Note that if you followed the tutorial exactly and used the **contract_approval_app_config.txt** file to save the application configuration information (see: Deploying the application to the IM services), all above values should be available from that file (except for the email address, which is the standard email address you used to create your organization).



You can now save and close the **Cloud Developer Tutorial** environment configuration screen.

- Let's now switch to the **Collections** view from the left side bar and have a first look at the **Cloud Developer Tutorial** Postman collection.

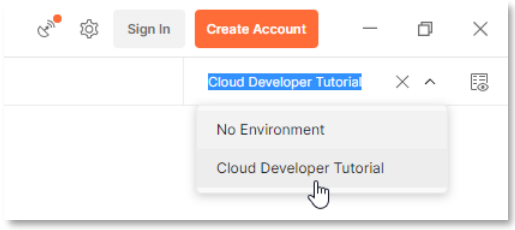


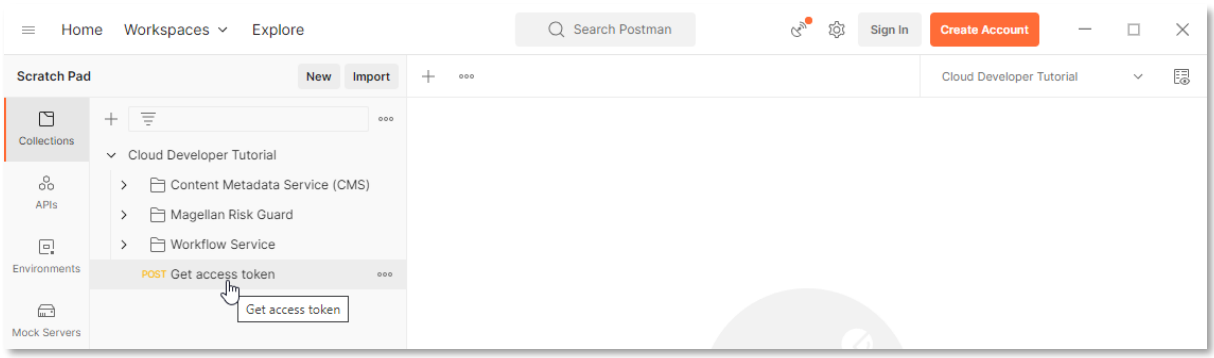


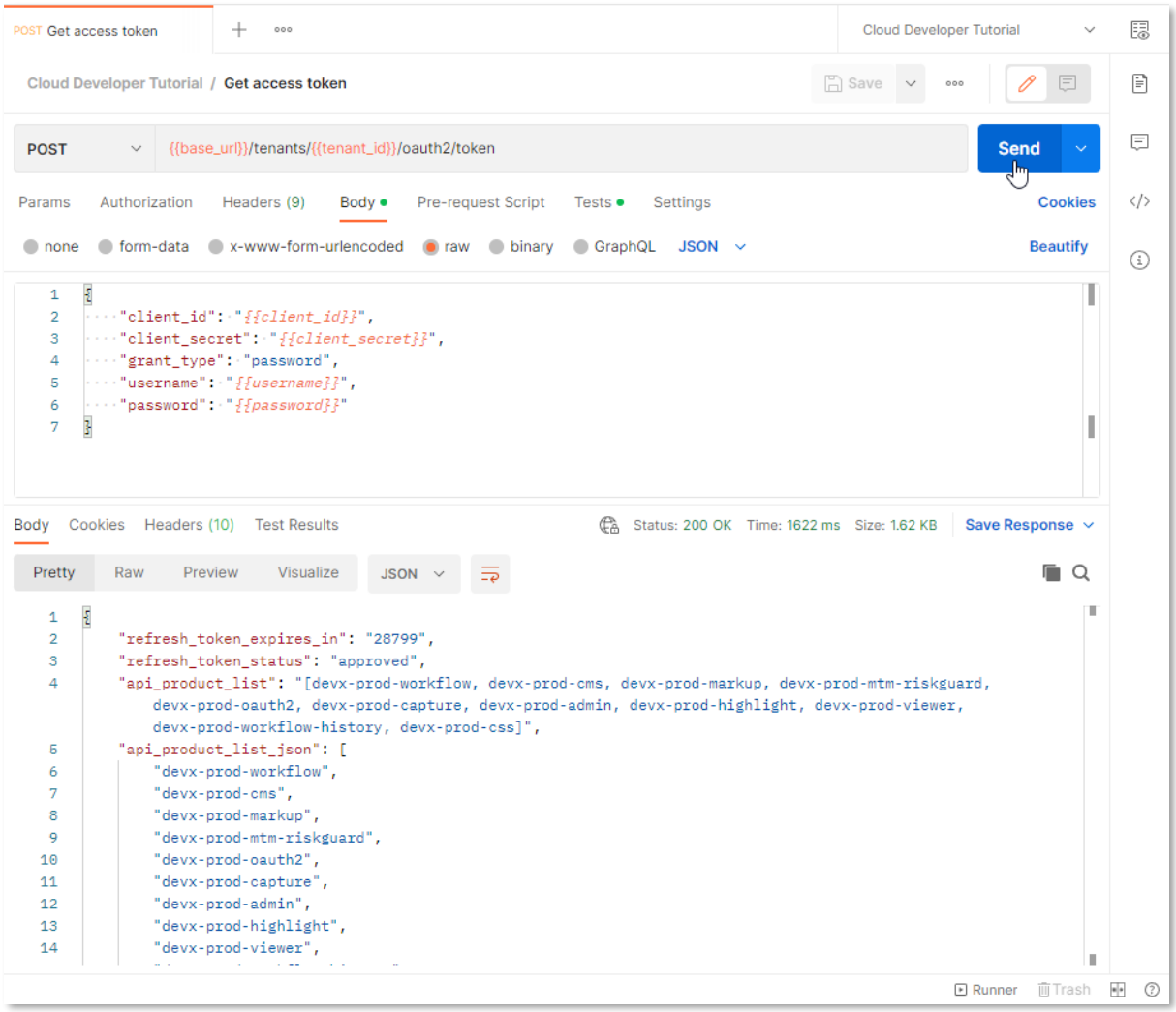
When you expand the **Cloud Developer Tutorial** collection, you can see the three folders representing the services (CMS, Magellan Risk Guard and Workflow Service) for which the collection has example requests.
There's also a **Get access token** POST request in the root of the collection, as this is the single request you will be using to get an access token to use for all other requests. Once you run this request successfully the **access_token** environment variable gets populated and can be used in every other request.

REMARK:
The access_token can expire. If this happens your API requests will start failing, pointing to the token not being valid. When that happens, just re-run the **Get access token** request and you should be able to continue (perform the other requests again).

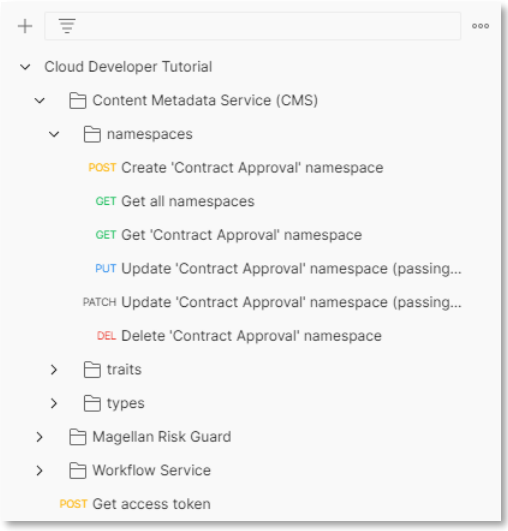
- To get started, let's now indeed get a token by running the **Get access token** POST request. Make sure you have selected the **Cloud Developer Tutorial** environment, open the **Get access token** request and click **Send**.



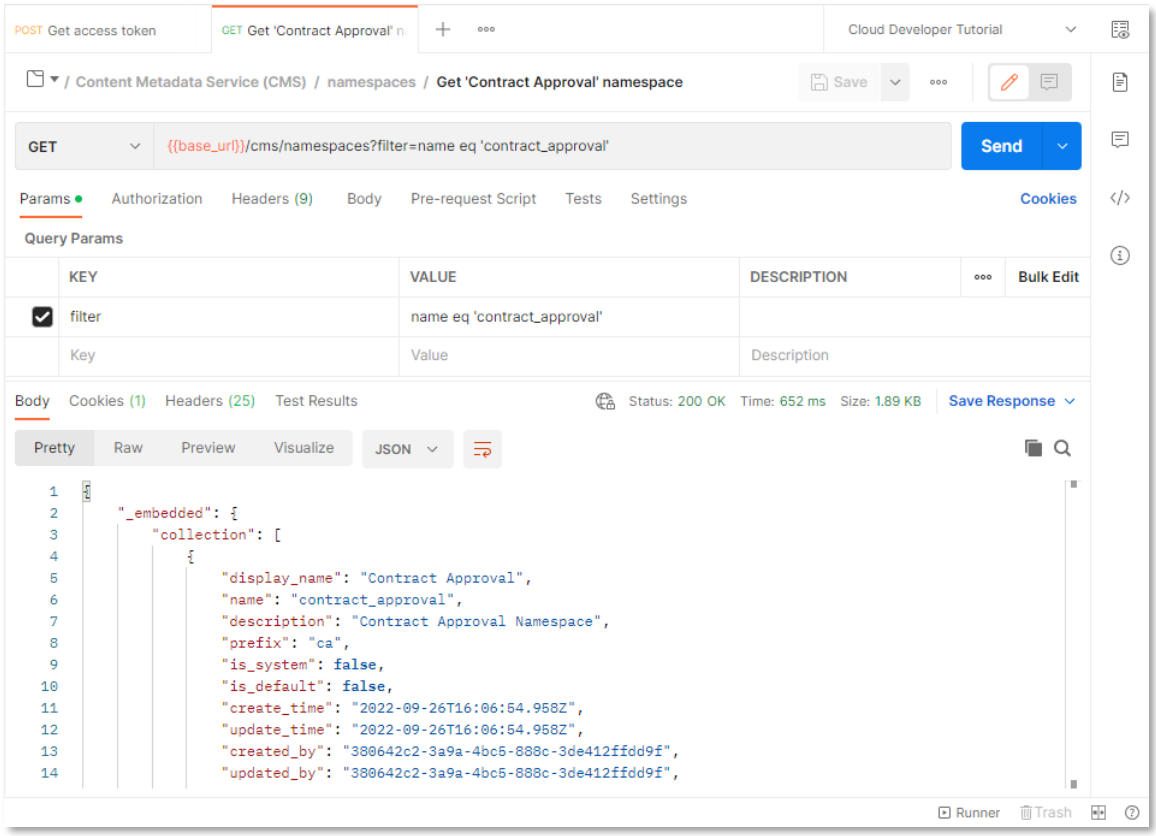




- Let's first verify that the **contract_approval** namespace has been correctly deployed/created in CMS.
  Expand the **Content Metadata Service (CMS)** folder from the Cloud Developer Tutorial collection, followed by expanding the **namespaces** folder.

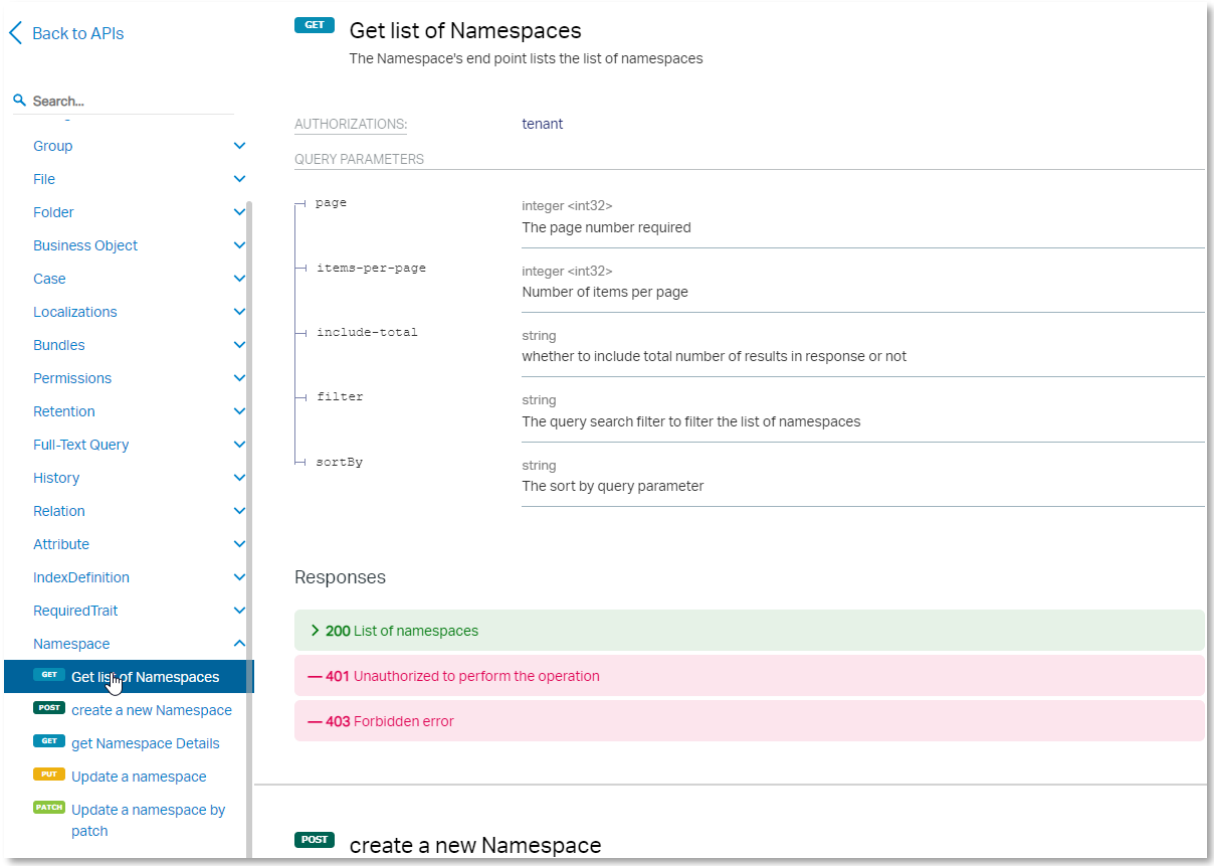


  What you see here are the different example requests relating to namespaces. We'll have a look at the one that allows to retrieve the namespace we created.
  Open the **Get 'Contract Approval' namespace** request and press **Send**.

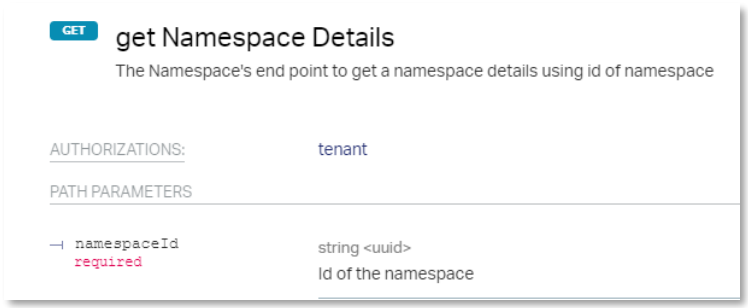


  As you can see, the Contract Approval namespace is being returned (i.e.: it has been correctly deployed) with its **display_name**, **name**, **description**, and **prefix** attributes.
  Note that the request we are using, **{{base_url}}/cms/namespaces?filter=name eq 'contract_approval'** is using a filter to retrieve the namespaces with a name equal to 'contract_approval'.

All the requests are based on the API reference documentation, so if you want to look at the related documentation, just go to the CMS API reference on developer.opentext.com and select the appropriate request explanation. This applies to all requests in the collection (note that for Workflow Service you need to refer to the Workflow Service API reference and for Magellan Risk Guard to the Magellan Risk Guard API reference).
In the case of this specific request, you'll find the explanation under **[GET] Get list of Namespaces** for the **Namespace** resource.



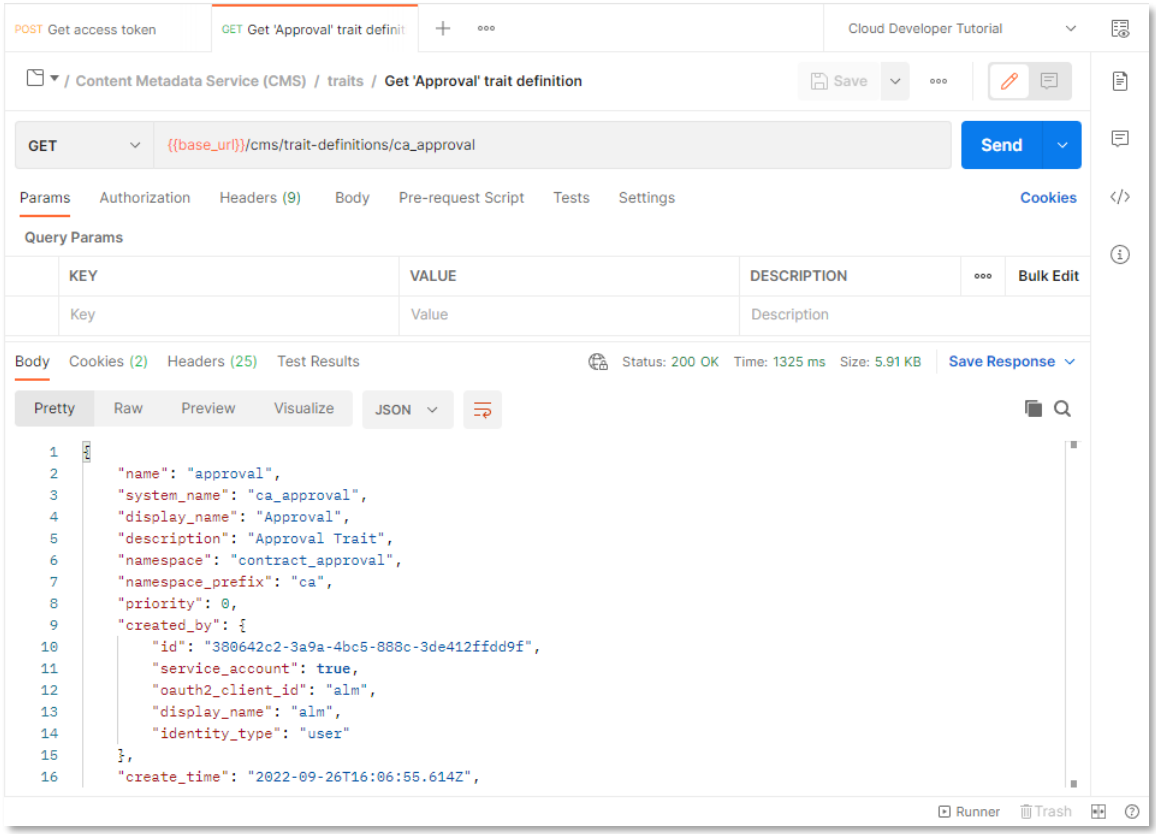
In theory, the request to use to get a specific namespace is the **get Namespace Details** GET request, but this requires the unique ID (of UUID string format) of the namespace to be passed.



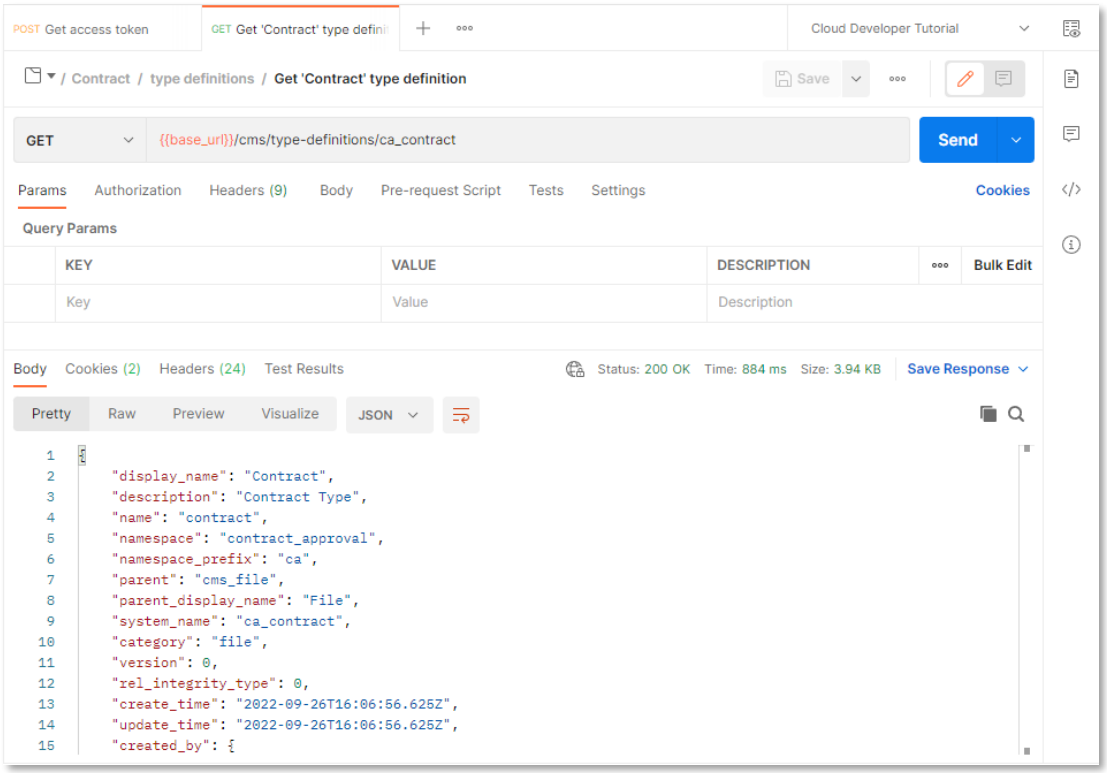
That's why we used the filtering mechanism on the **get list of Namespaces** request (so that it always works, no matter the namespace's ID value).

REMARK:
Throughout the Postman collections you'll find UUIDs representing unique IDs of resources as parameters for GET, PUT, PATCH and DELETE requests. In case you want to use those requests, you of course need to replace the UUID values with the value of the resource ID you want to work with.

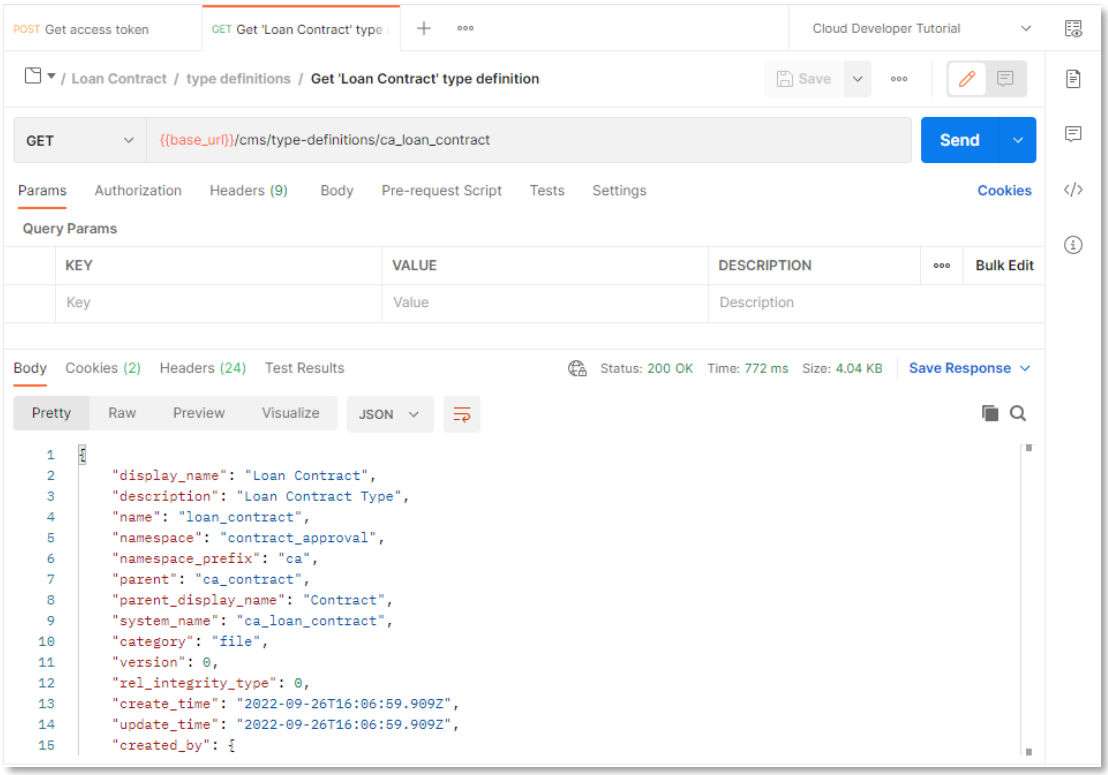
- The next deployed model to verify is the **approval** trait definition.
  Similar to how we verified the namespace, expand the **traits** folder under the **Content Metadata Service (CMS)** folder in the collection, and perform the **Get 'Approval' trait definition** GET request.



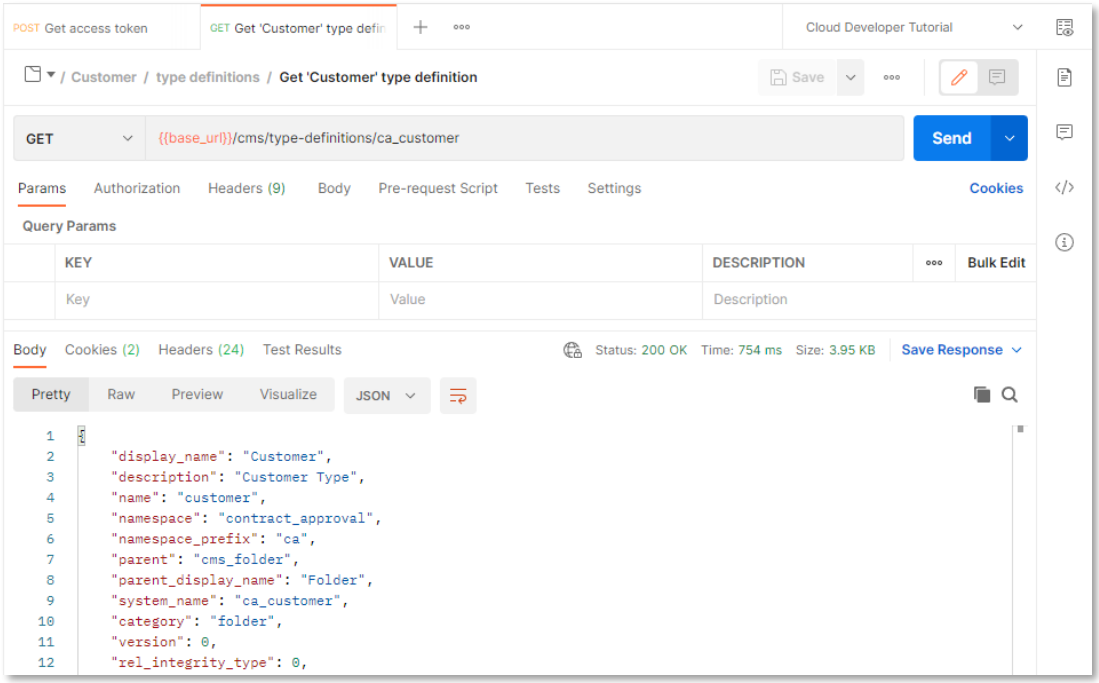
- To verify the deployment of the **contract**, **loan_contract** and **customer** type definitions, respectively perform the following requests:
  - **/Content Metadata Service (CMS)/types/file/Contract/type definitions/Get 'Contract' type definition**



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

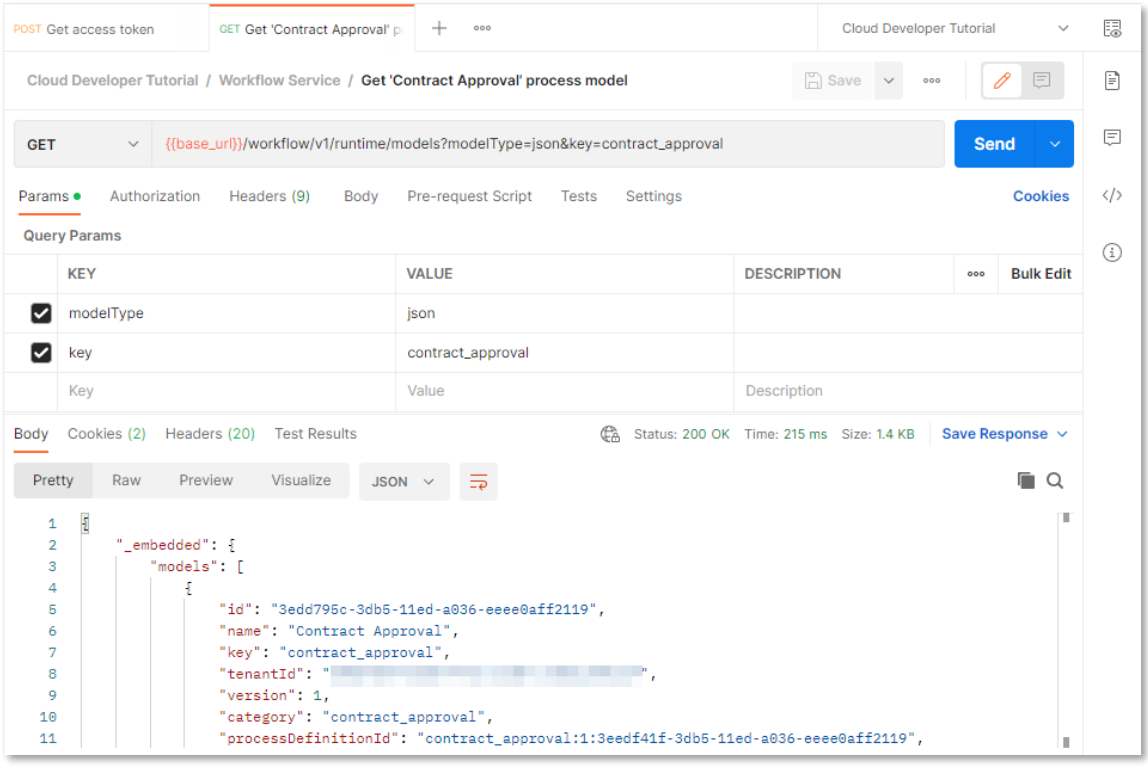


- **/Content Metadata Service (CMS)/types/folder/Customer/type definitions/Get 'Customer' type definition**



As you will probably have noticed, for type definitions, we added some levels to the collection folder structure to make the distinction between the **file** and **folder** type category, and the actual type definitions. We also provided **type instances** related requests (on top of the **type definitions** request) to allow retrieving and manipulating (including a "delete all instances" request) the type instances that you create when testing the application.

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



You have now verified the deployment of all your models, and you have seen how to use the API reference documentation and the **Cloud Developer Tutorial** Postman collection in the process.

It is now time to start with the actual code writing part of the tutorial. Don’t worry, we’ll provide you with the sample code, so that you can import it into your VS Code project. The next chapter will take you through this sample code.

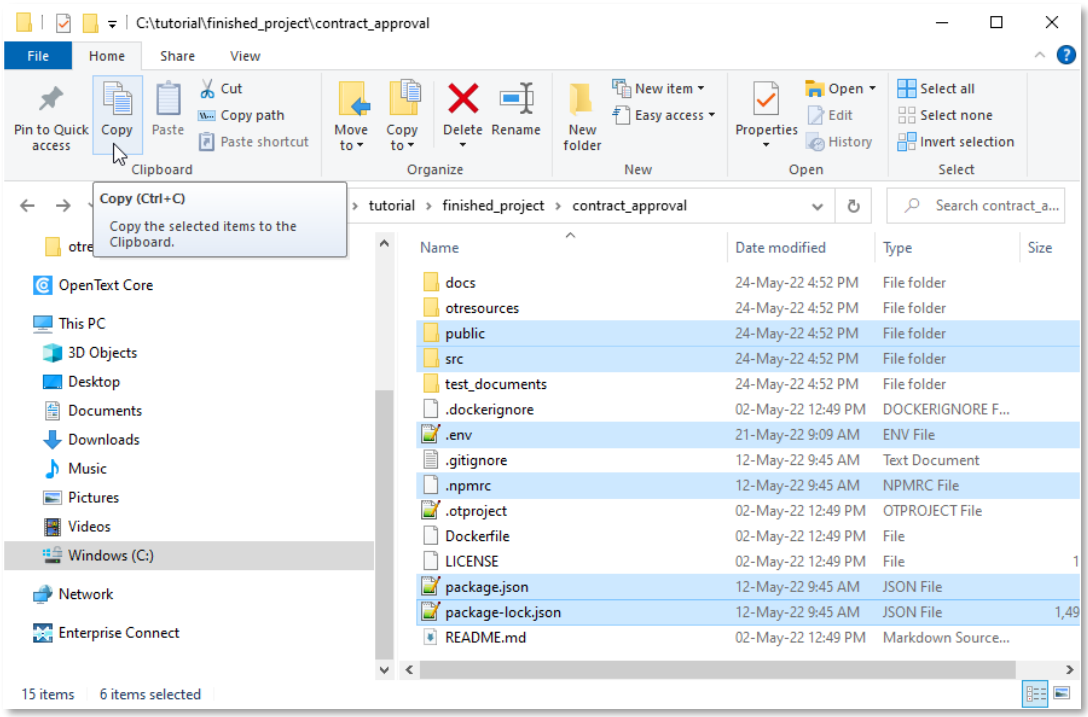
3.12[20'] Building the application

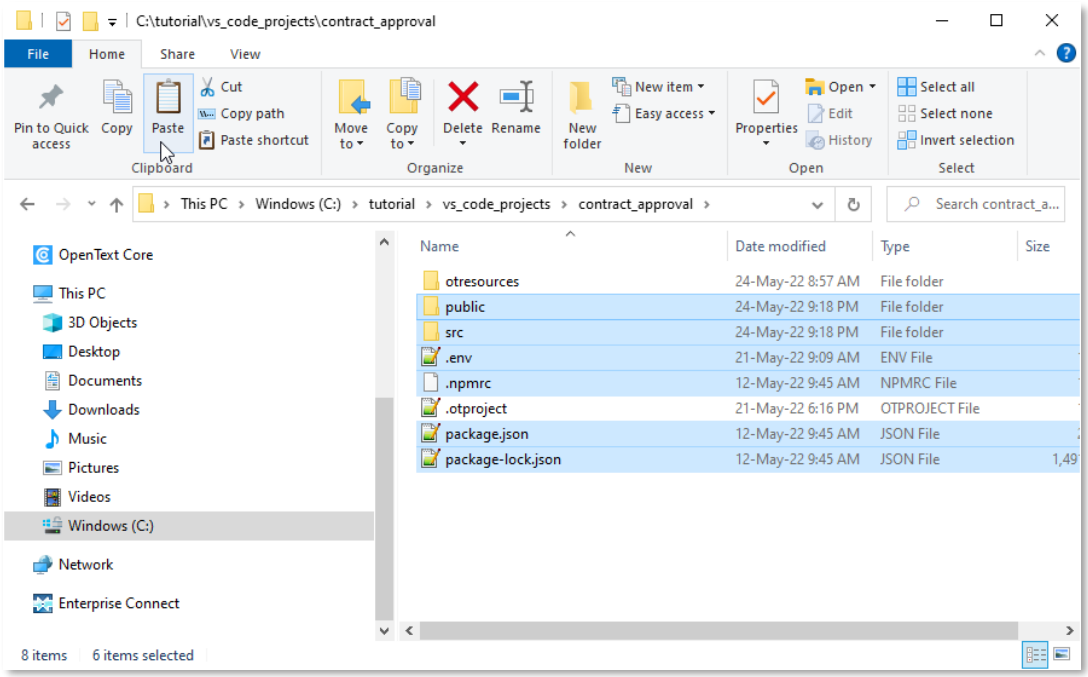
During this exercise we will be going through the code of the Contract Approval application. Although we will import and not actually write the (JavaScript and React) code, we will go over its structure, logic and how it calls the different IM services (CMS, Workflow Service and Magellan Risk Guard). It is not the intent to go over every single detail of how the code was written, and which file does what, but we will touch upon some key aspects of how the application has been developed. For more information on the CMS, Workflow Service and Magellan Risk Guard APIs, you can refer to their API reference documentation, respectively CMS API reference, Workflow Service API reference and Magellan Risk Guard API reference.

Once you are done with this section, you will have understood how the code of the Contract Approval application has been written, how it consumes the deployed models, and how it calls the different IM APIs. In the next exercise you will be running the application and testing the functionality you developed.

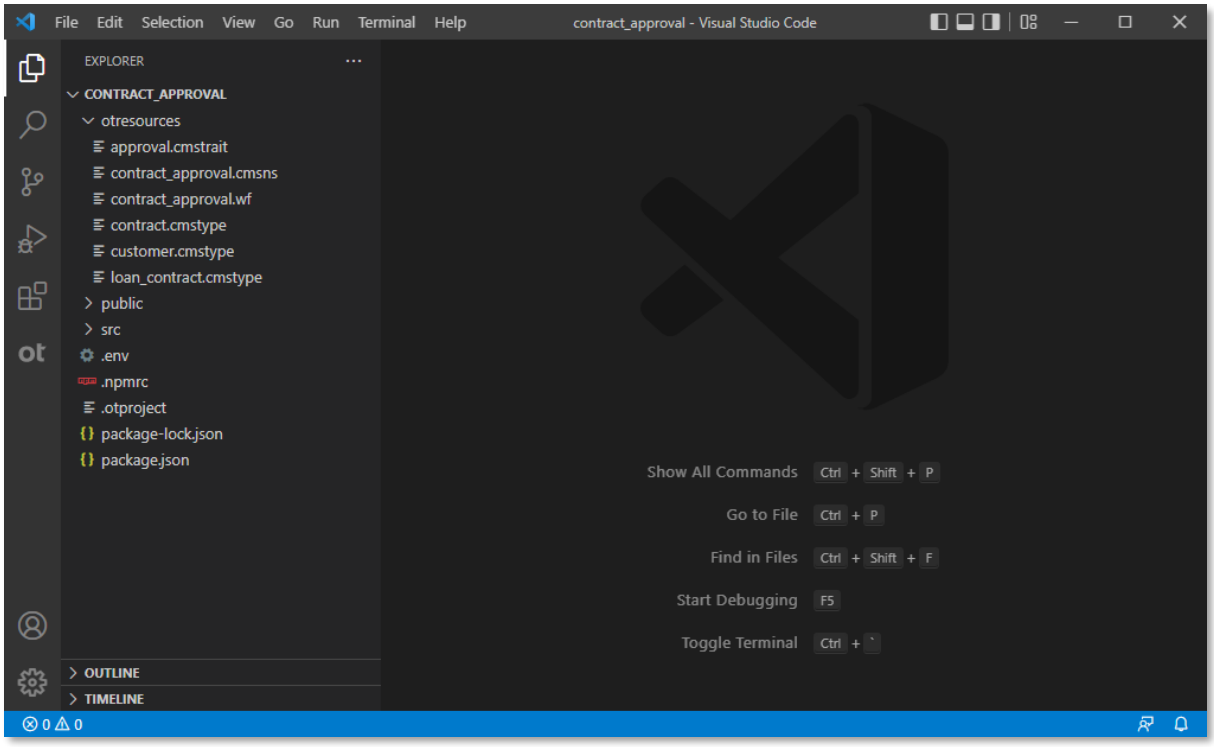
To import and go over the code of the Contract Approval application, proceed as follows:

- The first step in this exercise is indeed to import the code into your VS Code project. If you performed the step in the previous exercise of extracting the **contract_approval** project folder (containing the finished application), navigate into that **contract_approval** folder and copy the following folders and files into the root of your Contract Approval project:
    - **public** folder
    - **src** folder
    - **.env** file
    - **.npmrc** file
    - **package.json** file
    - **package-lock.json** file



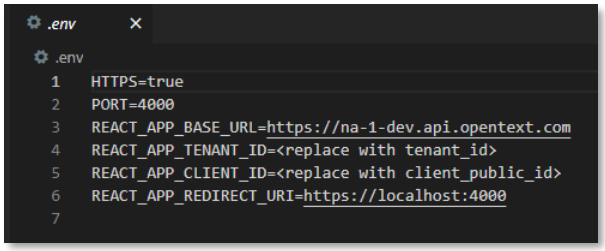


You can now open VS Code. The copied folders and files should be visible in the **Explorer** view.

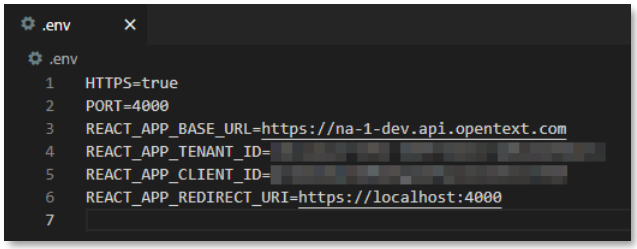


- The main entry point for the application (being a React application) is the **App.jsx** file, available from the **/src** folder.
  From line **20** to **27** of **App.jsx** you see how an **authService** is being initialized with the public (organization) **clientId**, the different authorization related endpoints, the **redirectUri** and the **scopes**.

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



  Please 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 (in the **contract_approval_app_config.txt**). More specifically, use the tenant id (from text **in tenant '<tenant id>'**) and the client public id (**Public Client ID** value).



Returning to **App.jsx**, take a further look at the authentication code (line **32** to **72**). More specifically, the different authentication functions provided by the **authService** and how they are being used to provide the authorization (reset, login, and logout) capabilities for the contract approval application.

```
32  const { authService } = useAuth();
33
34  const login = async () => authService.authorize();
35  const logout = async (shouldEndSession) => authService.logout(shouldEndSession);
36
37  if (authService.isPending()) {
38    return (
39      <div className="page-content">
40        <p>Loading...</p>
41        <button style={{ margin: "0.50rem" }} onClick={() => { logout(true); login(); }}>Reset</button>
42      </div>
43    );
44  }
45
46  if (!authService.isAuthenticated()) {
47    login();
48    return (
49      <div className="page-content">
50        <p>Logging in ...</p>
51      </div>
52    );
53  }
54
55  const authContext = {
56    userName: authService.getUser().preferred_username,
57    idToken: authService.getAuthTokens().id_token,
58    headers: {
59      Authorization: `Bearer ${authService.getAuthTokens().access_token}`
60    }
61  }
62
63  const handleChange = (event, newValue) => {
64    setValue(newValue);
65  }
66
67  // This custom logout is required, as the standard PKCE React library (react-oauth2-pkce) doesn't support the id_tok
68  // A request with the owners of the react-oauth2-pkce library has been logged to add the id_token_hint support. Feel
69  const logoutWithIdTokenHint = (shouldEndSession, idToken) => {
70    logout(shouldEndSession);
71    window.location.replace(process.env.REACT_APP_BASE_URL + '/tenants/' + process.env.REACT_APP_TENANT_ID + '/oauth2/
72  }
```

Also note how a new **authContext** is being built to hold the authorization context (**userName**, **idToken** and **headers** with bearer token) coming from the a**uthService**. This **authContext** can now be passed to the child react components of the **App.jsx** component to support the different IM REST API calls.

To finish looking at the **App.jsx** code let's examine the main React code, which is located under the **return** statement (starting at line **74**). Note that the returned React code is wrapped as a "secured app" in the **AuthProvider** (using the authentication from the **authService**).

App.jsx

src > App.jsx > App

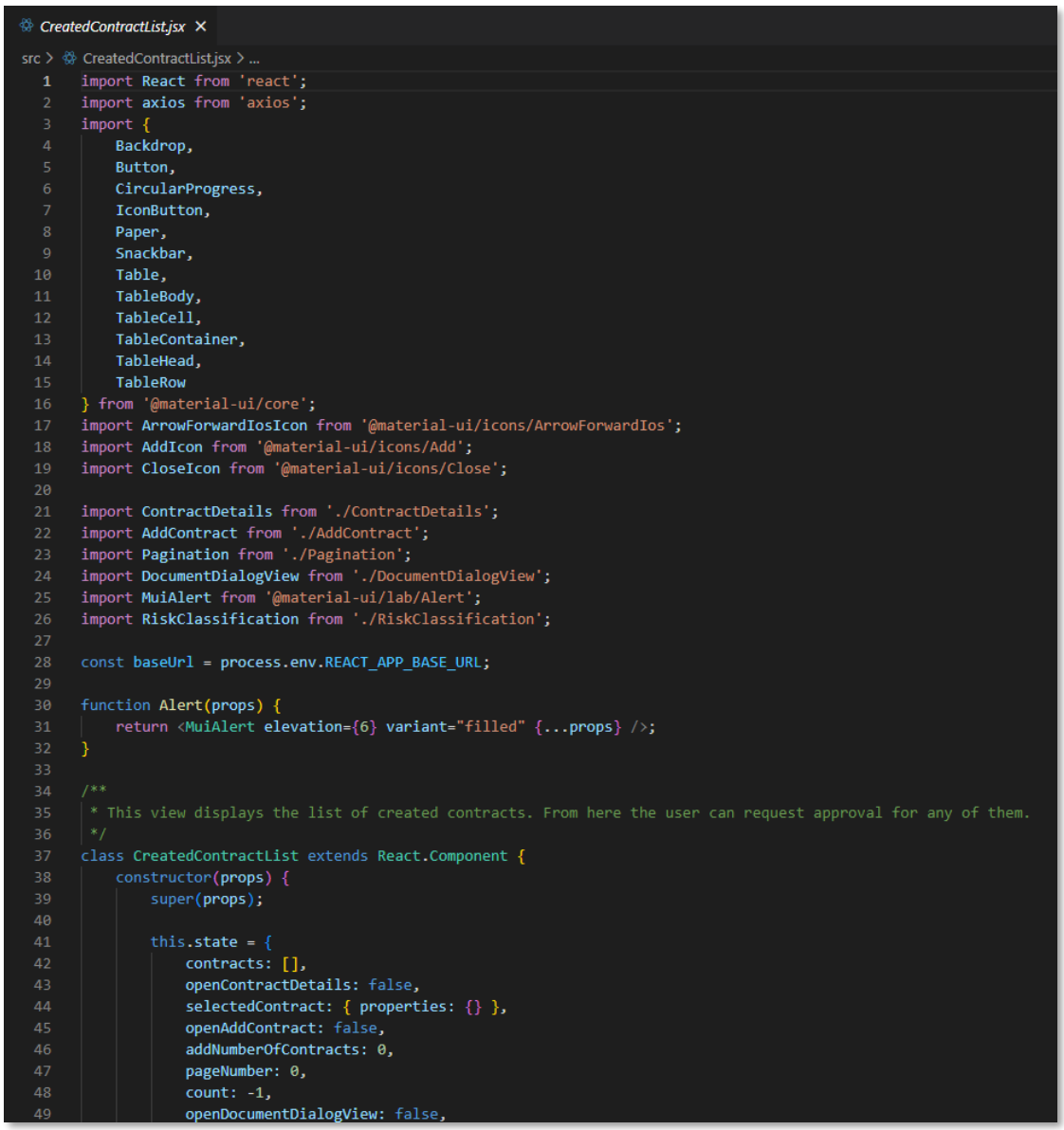
```
73
74    return (
75      <div className="App">
76        <Header
77          authContext={authContext}
78          logout={logoutWithIdTokenHint}
79        />
80        <div className="page-content">
81          <Tabs orientation="horizontal"
82            value={value}
83            onChange={handleChange}>
84            <Tab className="tab-caption" label="Created Contracts" />
85            <Tab className="tab-caption" label="Line Manager Tasks" />
86            <Tab className="tab-caption" label="Risk Manager Tasks" />
87            <Tab className="tab-caption" label="All Contracts" />
88          </Tabs>
89          <ApplicationProvider>
90            <TabPanel value={value} index={0}>
91              <CreatedContractList authContext={authContext} />
92            </TabPanel>
93            <TabPanel value={value} index={1}>
94              <TasksList authContext={authContext} taskname="Line Manager Approval" />
95            </TabPanel>
96            <TabPanel value={value} index={2}>
97              <TasksList authContext={authContext} taskname="Risk Manager Approval" />
98            </TabPanel>
99            <TabPanel value={value} index={3}>
100             <ContractList authContext={authContext} />
101           </TabPanel>
102         </ApplicationProvider>
103       </div>
104     </div>
105   );
106 }
107
108 function WrappedSecuredApp() {
109   return (
110     <AuthProvider authService={authService} >
111       <App />
112     </AuthProvider>
113   );
114 }
115
116 export default WrappedSecuredApp;
117
```

If we focus on the contents of the **Tabs** and **ApplicationProvider** container components, we can distinguish a four (horizontal) tabs UI layout (with the **Tab** components representing the tabs and the **TabPanel** components representing the corresponding views when clicking the tab). In short, this code will generate a UI with four horizontally stacked tabs:

- The **Created Contracts** tab (with the **CreatedContractList** component providing the "created contracts list" view to show all newly created contracts, i.e.: 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)
- 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)
- The **All Contracts** tab (with the **ContractList** component providing the "all contracts" view to show all contracts in the application, independently of their status)

Note that each of the tab views are React (child) components and that they are, as previously mentioned, indeed passed the **authContext** to ensure proper authorization when calling the IM REST APIs.

- Let's have a closer look at the first of the four views. The "created contracts list" view (i.e.: the **CreatedContractList** React component) does not only show the newly created contracts (status = 'CREATED'), but it also provides the button to add new contracts to the system. You can open the corresponding **CreatedContractList.jsx** file from the same **/src** folder you already opened the **App.jsx** file from.

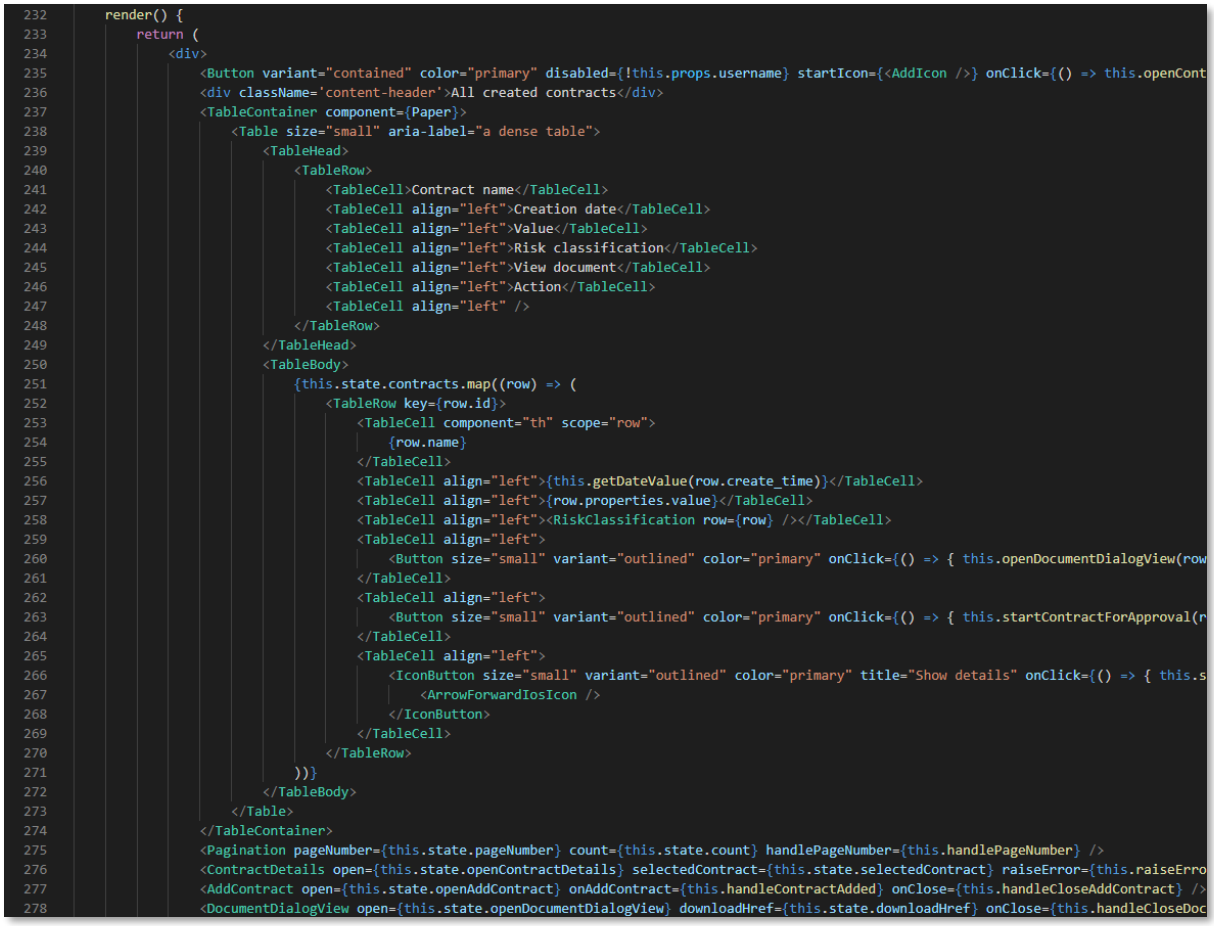


CreatedContractList.jsx

src > CreatedContractList.jsx > ...

```jsx
import React from 'react';
import axios from 'axios';
import {
    Backdrop,
    Button,
    CircularProgress,
    IconButton,
    Paper,
    Snackbar,
    Table,
    TableBody,
    TableCell,
    TableContainer,
    TableHead,
    TableRow
} from '@material-ui/core';
import ArrowForwardIosIcon from '@material-ui/icons/ArrowForwardIos';
import AddIcon from '@material-ui/icons/Add';
import CloseIcon from '@material-ui/icons/Close';

import ContractDetails from './ContractDetails';
import AddContract from './AddContract';
import Pagination from './Pagination';
import DocumentDialogView from './DocumentDialogView';
import MuiAlert from '@material-ui/lab/Alert';
import RiskClassification from './RiskClassification';

const baseUrl = process.env.REACT_APP_BASE_URL;

function Alert(props) {
    return <MuiAlert elevation={6} variant="filled" {...props} />;
}

/**
 * This view displays the list of created contracts. From here the user can request approval for any of them.
 */
class CreatedContractList extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            contracts: [],
            openContractDetails: false,
            selectedContract: { properties: {} },
            openAddContract: false,
            addNumberOfContracts: 0,
            pageNumber: 0,
            count: -1,
            openDocumentDialogView: false,
```

The main method of this React component is the **render** method (starting at line **232**) as it returns the actual “created contract list” UI.



```
232     render() {
233         return (
234             <div>
235                 <Button variant="contained" color="primary" disabled={!this.props.username} startIcon={<AddIcon />} onClick={() => this.openCont
236                 <div className='content-header'>All created contracts</div>
237                 <TableContainer component={Paper}>
238                     <Table size="small" aria-label="a dense table">
239                         <TableHead>
240                             <TableRow>
241                                 <TableCell>Contract name</TableCell>
242                                 <TableCell align="left">Creation date</TableCell>
243                                 <TableCell align="left">Value</TableCell>
244                                 <TableCell align="left">Risk classification</TableCell>
245                                 <TableCell align="left">View document</TableCell>
246                                 <TableCell align="left">Action</TableCell>
247                                 <TableCell align="left" />
248                             </TableRow>
249                         </TableHead>
250                         <TableBody>
251                             {this.state.contracts.map((row) => (
252                                 <TableRow key={row.id}>
253                                     <TableCell component="th" scope="row">
254                                         {row.name}
255                                     </TableCell>
256                                     <TableCell align="left">{this.getDateValue(row.create_time)}</TableCell>
257                                     <TableCell align="left">{row.properties.value}</TableCell>
258                                     <TableCell align="left"><RiskClassification row={row} /></TableCell>
259                                     <TableCell align="left">
260                                         <Button size="small" variant="outlined" color="primary" onClick={() => { this.openDocumentDialogView(row
261                                     </TableCell>
262                                     <TableCell align="left">
263                                         <Button size="small" variant="outlined" color="primary" onClick={() => { this.startContractForApproval(r
264                                     </TableCell>
265                                     <TableCell align="left">
266                                         <IconButton size="small" variant="outlined" color="primary" title="Show details" onClick={() => { this.s
267                                             <ArrowForwardIosIcon />
268                                         </IconButton>
269                                     </TableCell>
270                                 </TableRow>
271                             ))}
272                         </TableBody>
273                     </Table>
274                 </TableContainer>
275                 <Pagination pageNumber={this.state.pageNumber} count={this.state.count} handlePageNumber={this.handlePageNumber} />
276                 <ContractDetails open={this.state.openContractDetails} selectedContract={this.state.selectedContract} raiseError={this.raiseErro
277                 <AddContract open={this.state.openAddContract} onAddContract={this.handleContractAdded} onClose={this.handleCloseAddContract} />
278                 <DocumentDialogView open={this.state.openDocumentDialogView} downloadHref={this.state.downloadHref} onClose={this.handleCloseDoc
```

The **TableContainer** component will provide the table layout to display the list of the different contracts that have the ‘CREATED’ status. As you can see from the **TableHead** and **TableBody** components, each table row will show the following information for the displayed contract:

- **Contract name**
- **Creation date**
- **Value**
- **Risk classification**
- **View document** (this is not a property value but a button to open the actual document in a viewer)
- **Action** (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)

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

If you look at the **getContracts** method on line **112**, you can easily recognize the (axios) GET request to the **/cms/instances** endpoint of the **CMS** IM API with the authorization (bearer) token passed in the **headers**.

CreatedContractList.jsx

src > CreatedContractList.jsx > ...

```
112     getContracts() {
113         if (this.props.authContext.userName) {
114             this.setState({ showBackdrop: true });
115             axios({
116                 method: 'get',
117                 url: baseUrl + '/cms/instances/file/ca_contract/?include-total=true&sortby=create_time desc&filter=status eq "CREATED"&page='
118                     + (this.state.pageNumber + 1),
119                 headers: this.props.authContext.headers
120             }).then(res => {
121                 this.setState({
122                     contracts: res.data && res.data._embedded ? res.data._embedded.collection : [],
123                     count: res.data.total
124                 });
125             }).catch(error => {
126                 let errorMessage = 'Could not get contracts: ';
127                 if (error.response != null && error.response.data != null) {
128                     errorMessage += error.response.data.exception;
129                 } else {
130                     errorMessage += error.message;
131                 }
132                 this.raiseError(this, errorMessage);
133             }).finally(() => {
134                 this.setState({ showBackdrop: false });
135             })
136         } else {
137             this.setState({ contracts: [], count: -1 });
138         }
139     }
```

Going back to the **render** method, let's also take a look at the **AddContract** component (line **259**), as using this feature will trigger a call to the type instance creation endpoint from the **CMS** IM API.

```
<AddContract authContext={this.props.authContext} open={this.state.openAddContract} onAddContract={this.handleContractAdded} onClose={this.handleCloseAddContract} />
```

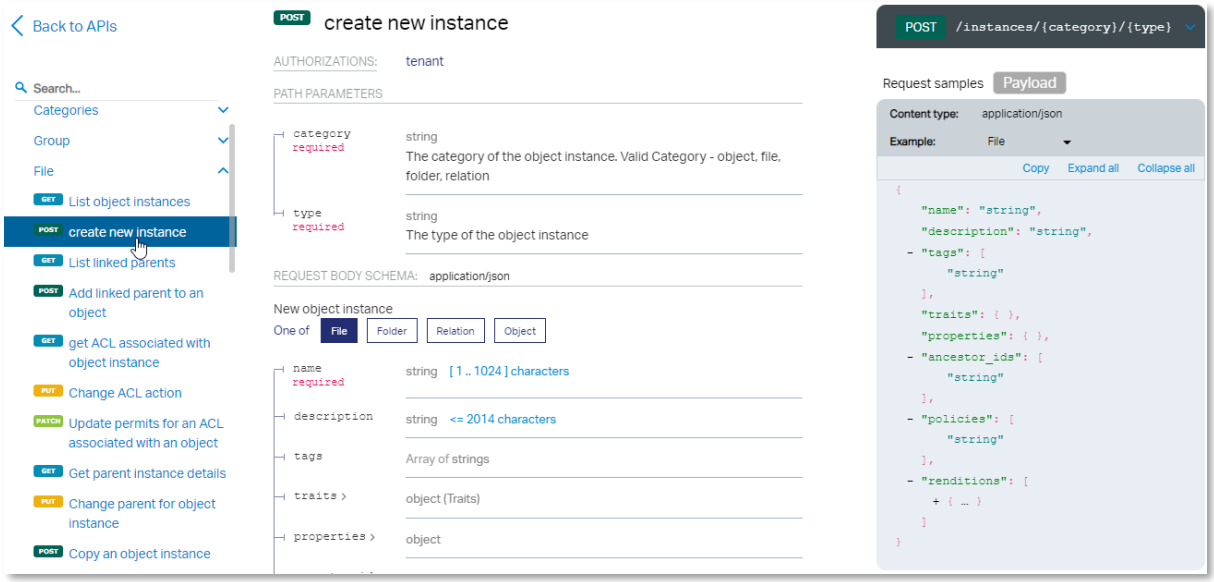
More specifically, it will open the contract creation dialog box to add a contract.

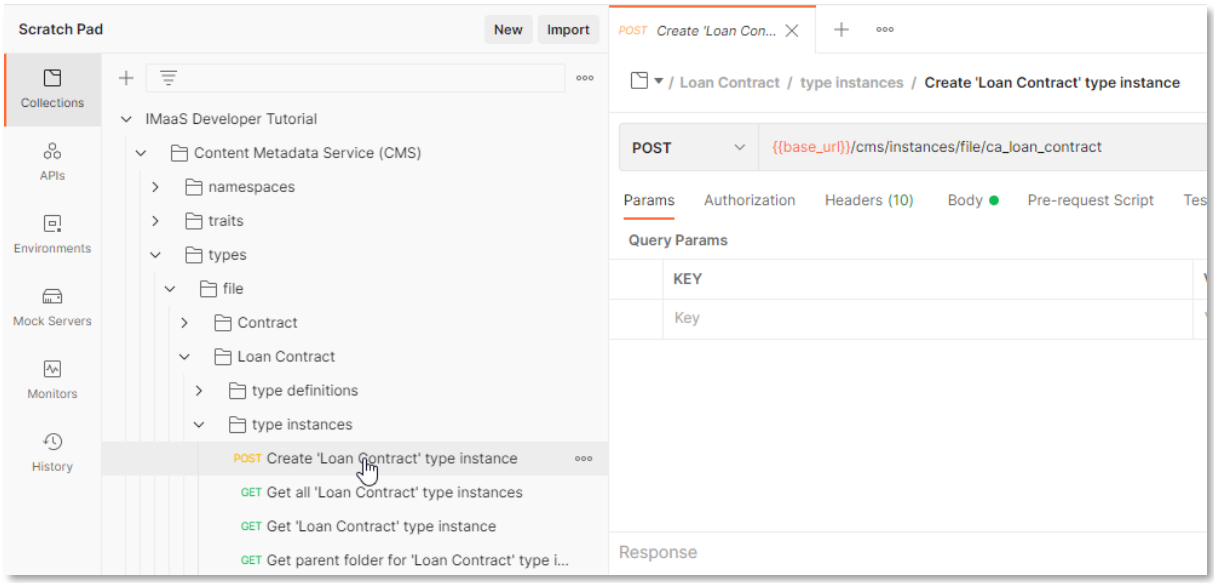
- How the contract creation works is part of the dialog box code, so as the last step in this exercise, let's open the **AddContract.jsx** React component (again from the **/src** folder) and look at line **222** where the actual action of calling the IM APIs to create the contract happens.

```
221         // Adding Contract
222         const formData = new FormData();
223         formData.append(
224             'file',
225             this.state.selectedFile,
226             this.state.selectedFile.name,
227         );
228         axios.post(
229             baseUrl +  '/css/v2/tenant/' + process.env.REACT_APP_TENANT_ID + '/content?avs-scan=false',
230             formData,
231             {
232                 headers: {
233                     'Content-Type': 'multipart/form-data',
234                     'Authorization': this.props.authContext.headers.Authorization
235                 },
236             }
237         ).then(res => {
238             let cmsType;
239             if (this.isLoanContract()) {
240                 cmsType = 'ca_loan_contract'
241             } else {
242                 cmsType = 'ca_contract'
243             }
244 
245             // Setting metadata
246             return axios({
247                 method: 'post',
248                 url: `${baseUrl}/cms/instances/file/${cmsType}`,
249                 headers: this.props.authContext.headers,
250                 data: {
251                     "name": this.state.newContractName,
252                     "parent_folder_id": parentFolderId,
253                     "renditions": [
254                         {
255                             "name": res.data.entries[0].fileName,
256                             "rendition_type": "primary",
257                             "blob_id": res.data.entries[0].id
258                         }
259                     ],
260                     "properties": {
261                         "value": parseInt(this.state.newContractValue, 10),
262                         "status": "CREATED",
263                         "requester_email": customerEmail,
264                         "risk_classification": this.state.contractRisk,
265                         "extracted_terms": this.state.extractedTerms,
266                         ...this.isLoanContract() && {
267                             "monthly_installments": parseInt(this.state.newContractMonthlyInstallments),
268                             "yearly_income": parseInt(this.state.newContractYearlyIncome)
```

Note that there are two subsequent (axios) POST requests. One at line **228** to upload the file to the **Content Storage Service (CSS)** and one at line **246** to create the contract metadata in the **Content Metadata Service (CMS)** with the contract's properties, rendition (linked to the previously uploaded CSS file), and traits as payload.

Note that the **CMS** IM API call (starting at line **246**) should look very familiar, as it corresponds to what we've already been covering in the Working with the IM APIs exercise. More specifically, if you look at the CMS API reference and Postman collection respectively, the code contained in the **axios** call represents exactly the same as the following two screen shots:





This actually takes us to the end of this exercise, as we have covered the examples of

- getting the list of newly created contracts,
- and creating a new type instance.

Feel free to further explore the Contract Approval code if you want more code examples on how to talk to the different IM API endpoints and how to expose their features in the Contract Approval application.

REMARK:
In the Contract Approval sample application, we are using the **PKCE** authorization flow, with the help of the **react-oauth2-pkce** React library as it provides an easy-to-use experience for the developer. PKCE is also the better choice for our sample application, as it allows authenticating with the IM APIs without the need to store confidential information (such as confidential client id and client secret).

If you want to further look into authentication and authorization related to the IM APIs, we recommend you try out the **ot2-sso-sample-application** available on Github at this location: https://github.com/opentext/ot2-sso-sample-application.

CONGRATULATIONS!

You have now completely finished building your Contract Approval application. In the next exercise we will test it and run through the different contract approval scenarios.

3.13[50’] Testing your application

During this exercise we will be testing the Contract Approval application you have just built. We 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.

Once you are done with this section, you will have tested the different application flows and will have effectively completed the main part of the tutorial. Still, there will be one more remaining exercise that we recommend you run through. During this bonus exercise, you will learn how to use the otcloud Command Line Interface (CLI) to perform OpenText Cloud Platform related operations (such as deploying a project). This is certainly valuable, for example, in context of build automation and CI/CD use cases.

READ THIS IF YOU SKIPPED THE PREVIOUS EXERCISES:
If you decided to skip ahead and test the application without actually going through the building exercises, you will need to do a few additional steps before you can start.

To ensure that you can correctly test the Contract Approval application, perform the following activities:

- Be certain that you have checked and fulfilled the Prerequisites.
- Set up your IDE as described in Setting up the Cloud Developer IDE.
- Connect to your developer organization as described in Adding an organization and testing the connection.
- Download the finished version of the Contract Approval application (ZIP file) through this link and extract it.
- Once downloaded and extracted, make sure to open the **contract_approval** folder in VS Code, as this is the root of your project.
- Finally, deploy the application into your developer organization as described in Deploying the application to the IM services.

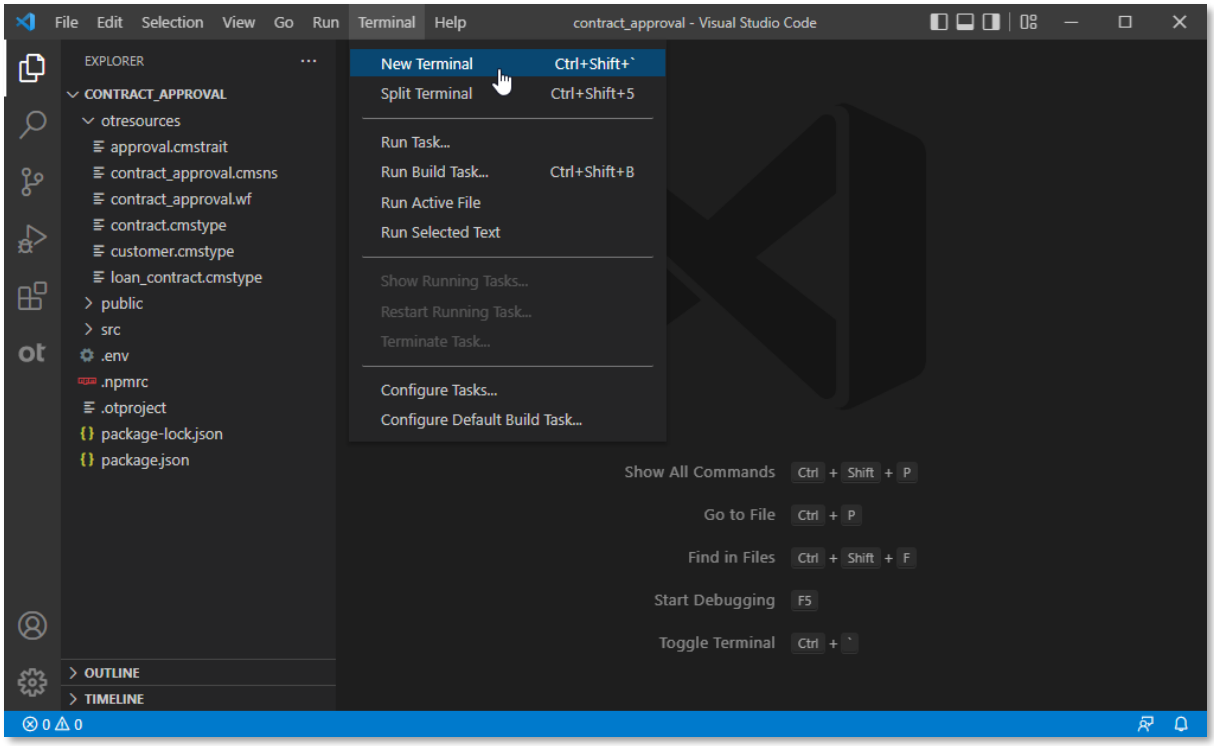
You are now ready to proceed with this exercise and test the Contract Approval application.

REMARK:
Although you did not go through the steps to build the application yourself, you might still want to understand how it was built. To that end, you can just browse the previous exercises, and/or look at the VS Code project you just downloaded.
If you want to dive into the VS Code project, this is a very short explanation of its main project folders:

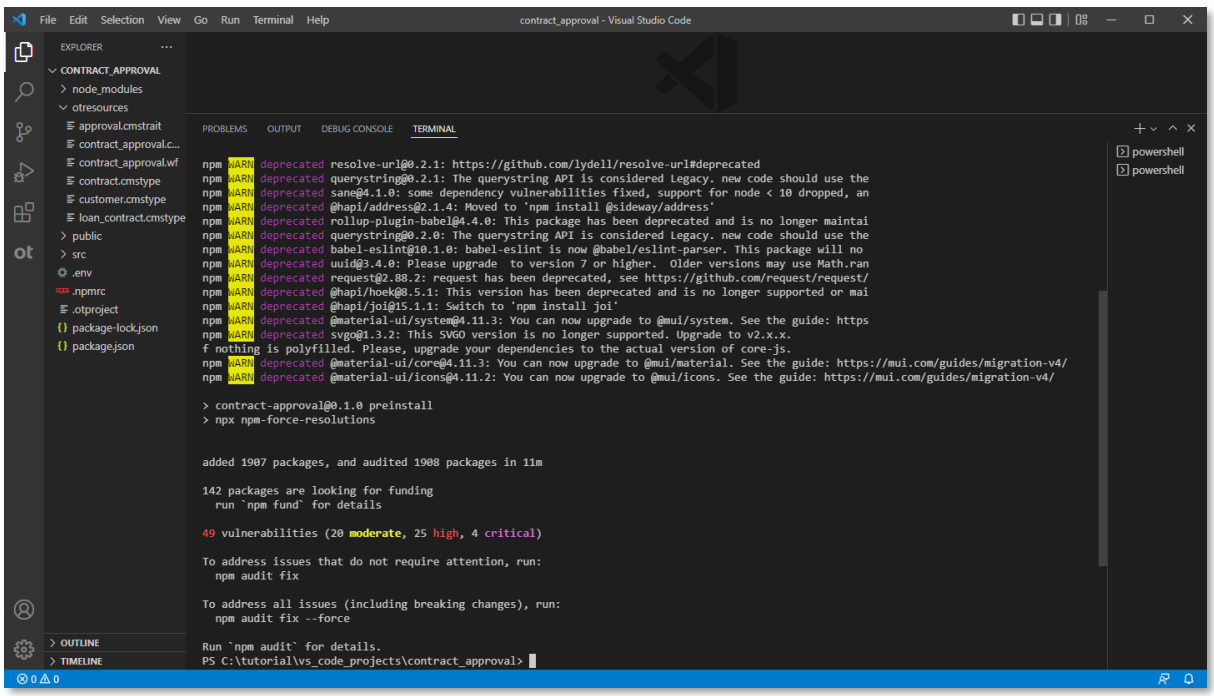
- **src**: contains the JavaScript and React code that communicates with the IM APIs and provides the User Interface (UI) of the application
- **otresources**: contains the different models (built with the OpenText Cloud Developer Tools for VS Code) that will be deployed to the IM services of the OT2 Platform

To test the Contract Approval application, proceed as follows:

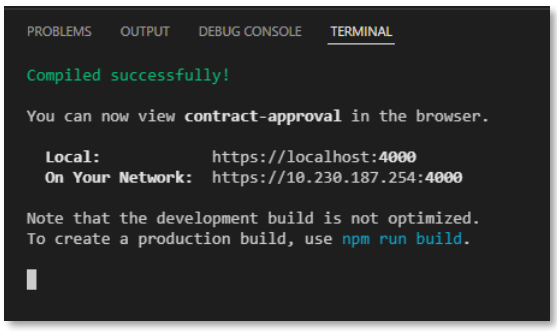
- Open your Contract Approval application project in VS Code and open a new **Terminal** window.

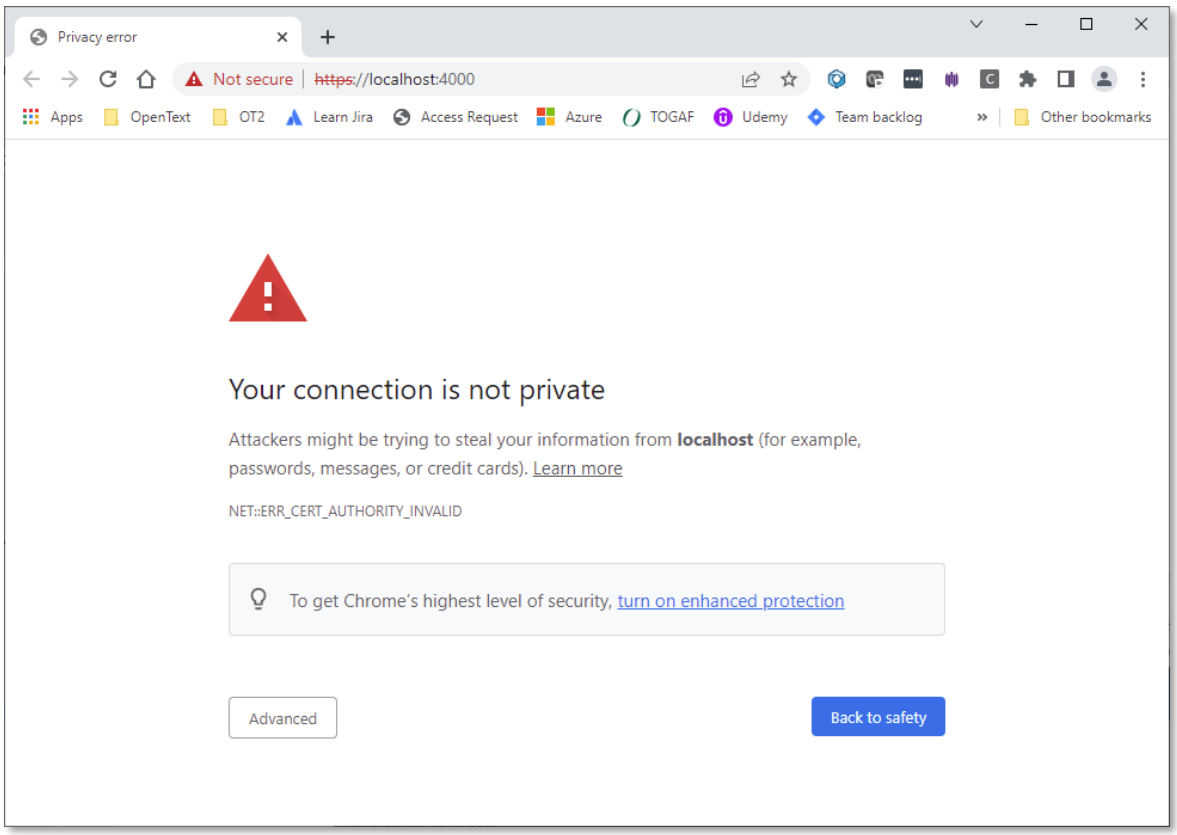


In the **Terminal** window run the **npm install** command. This will install all dependencies/libraries needed by your application to run.
Note that this process can certainly take a while (typically 10 to 15 minutes), so you need to wait until it completes (as long as the automatically generated **node_modules** folder size increases, the process is still ongoing).

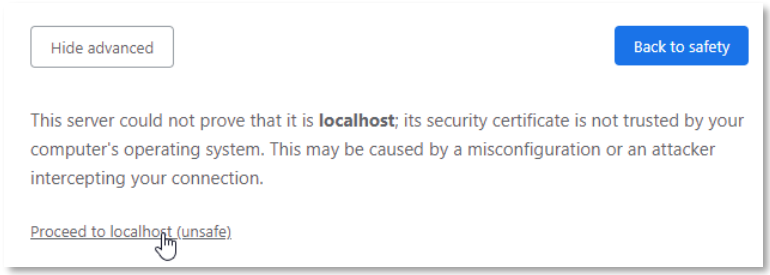


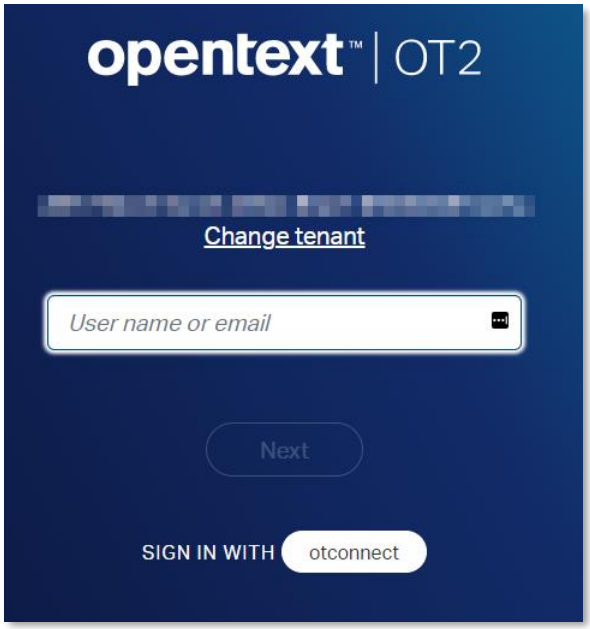
Once the npm install process has completed, you can launch the application using **npm start**. This will result in a new browser window opening on **https://localhost:4000**.



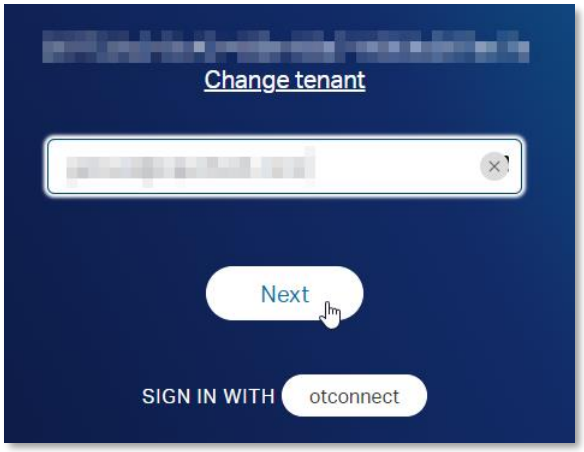


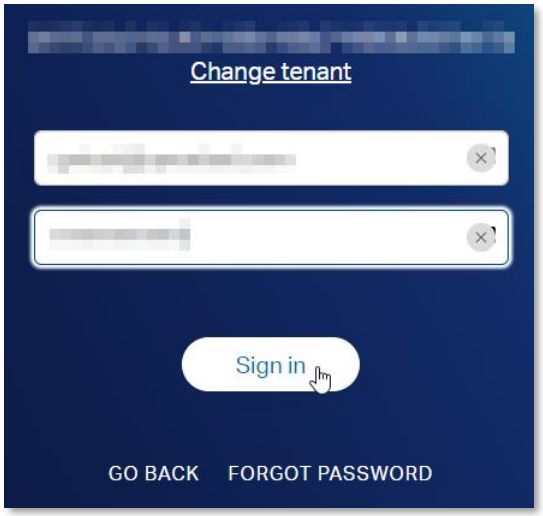
Click the **Advanced** button and choose to **Proceed to localhost (unsafe)**. Note that the screen shots are of Google Chrome and that the equivalent action on your own web browser can be different.



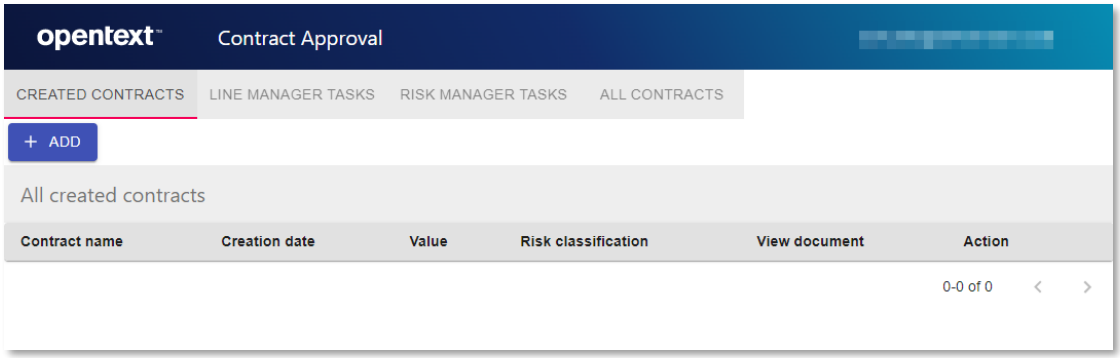


Log in using your tenant service account email (as username) and password. Again, if you have been following the exact tutorial steps, the password should be available from the **contract_approval_app_config.txt** file.

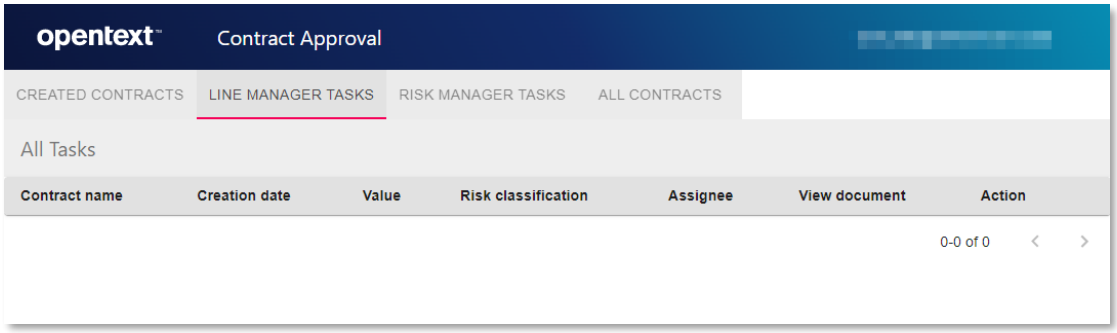




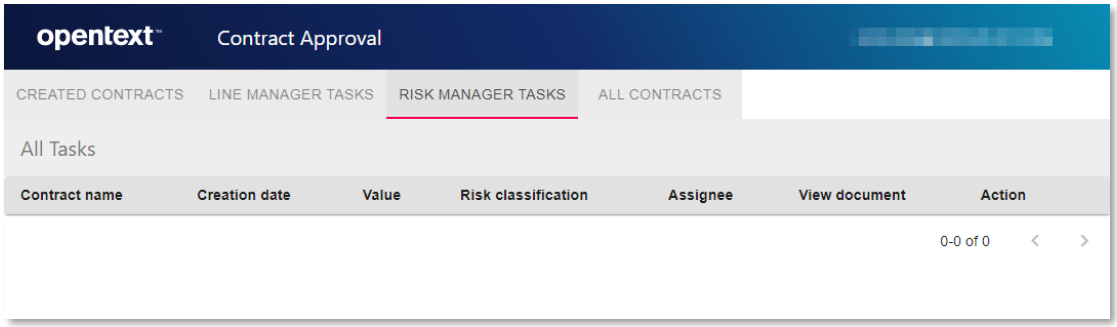
- You are now logged in to your Contract Approval application. Let's have a look at the different tabs (the four tabs we previously described when discussing the application code).
  - CREATED CONTRACTS: this tab shows all newly created contracts that have not yet been submitted for approval (i.e.: contracts with the status attribute equal to 'CREATED')



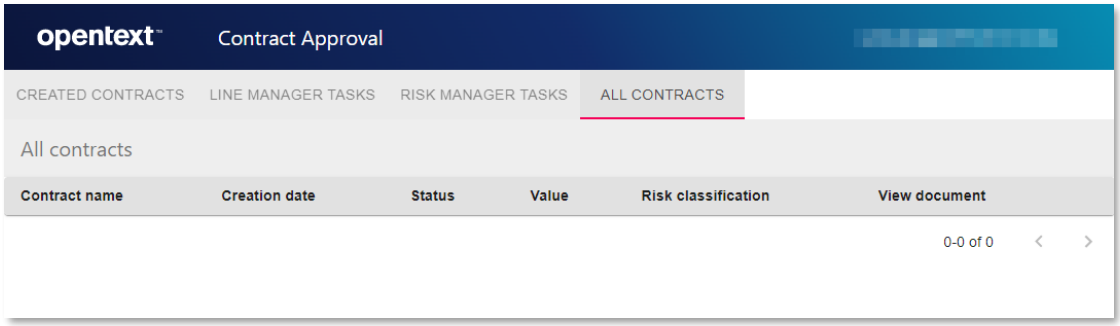
  - LINE MANAGER APPROVAL TASKS: this tab shows all approval tasks to be performed by the Line Manager



  - RISK MANAGER APPROVAL TASKS: this tab shows all approval tasks to be performed by the Risk Manager

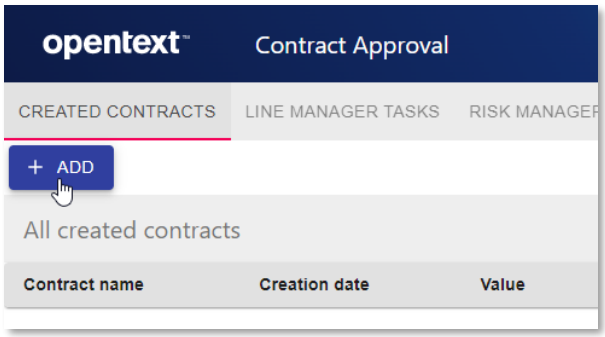


  - ALL CONTRACTS: this tab shows all contracts, independently of their status (i.e.: newly created, under approval, approved, rejected, expired)

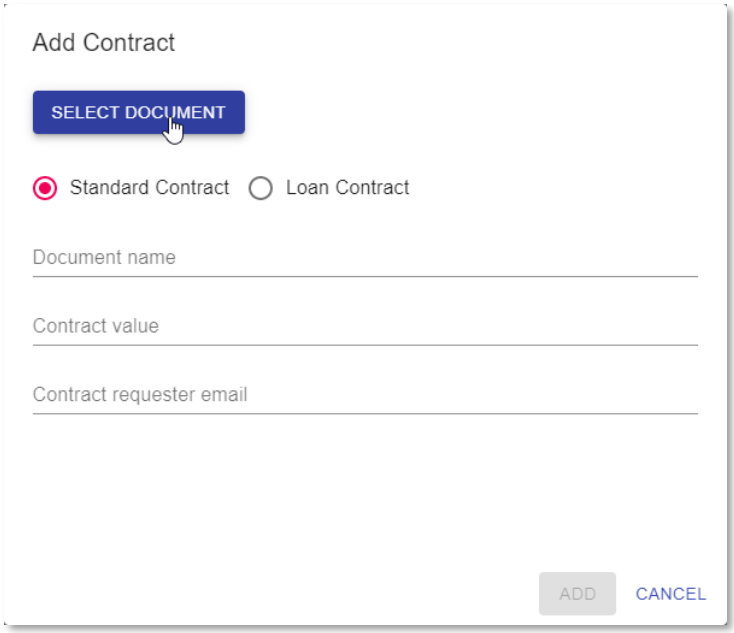


- Now that we have looked at the different tabs, let's create and approve our first contract.
  Let's start with the simplest approval process, i.e.: create a contract with the following characteristics:
  - **Type: standard contract** (doesn't require solvency check)
  - **Value: below 1000** (doesn't require Line Manager approval)
  - **Risk classification: below 4, i.e.: NONE, LOW or MEDIUM** (doesn't require Risk Manager approval)

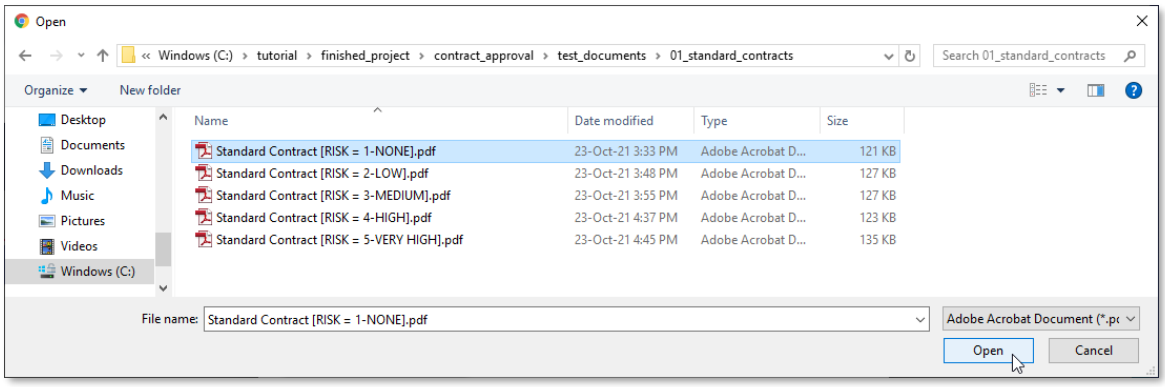
  Select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



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



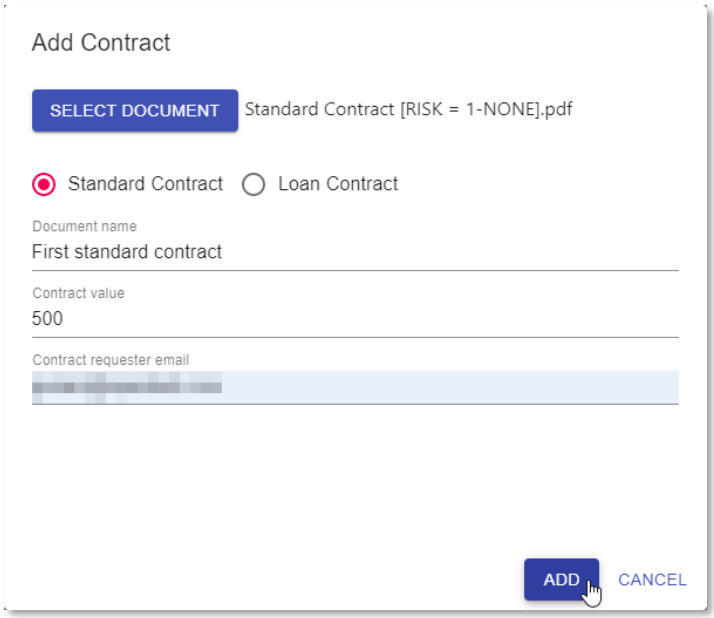
To help with selecting a file that matches the intended contract properties (certainly the risk classification, as this gets determined by what the Magellan Risk Guard text mining service discovers in the document), we have provided a **test_documents** folder under the finished version of the Contract Approval application project.
From this **test_documents** folder, open the **01_standard_contracts** subfolder and select the **Standard Contract [RISK = 1-NONE].pdf** file.



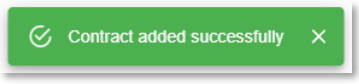
Make sure the **Standard Contract** option is selected and fill the contract properties as follows:

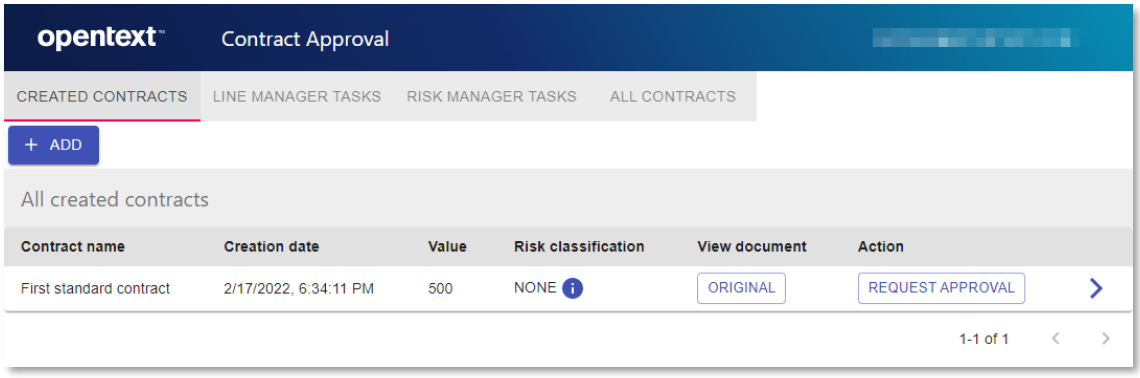
- **Document name: First standard contract**
- **Contract value: 500**
- **Contract requester email: <your email>**

Click **Add** to create the contract.

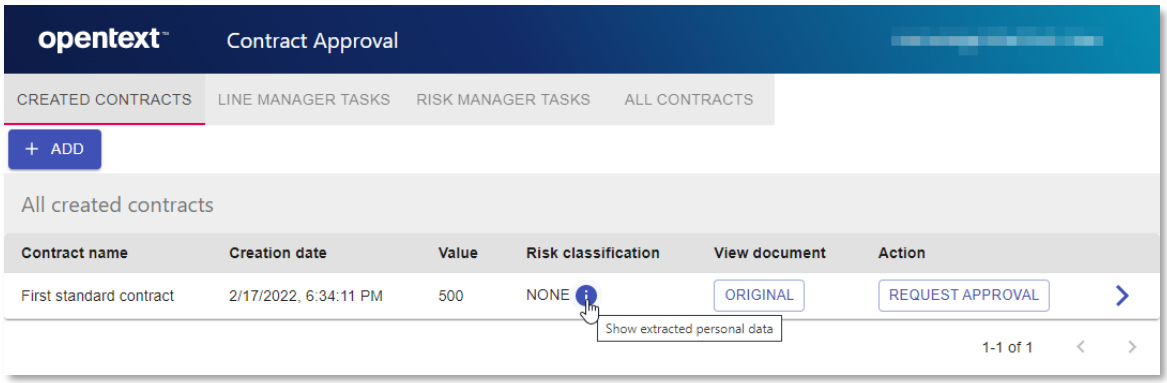


At the bottom of the screen, the creation of your new contract is confirmed by a “Contract added successfully” message.

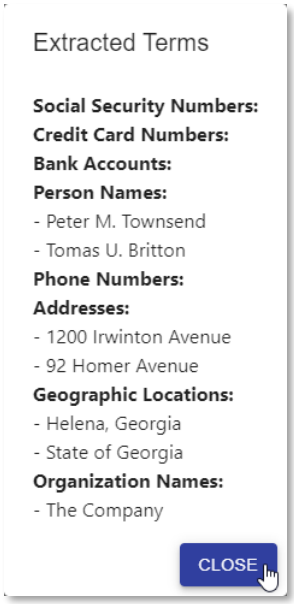




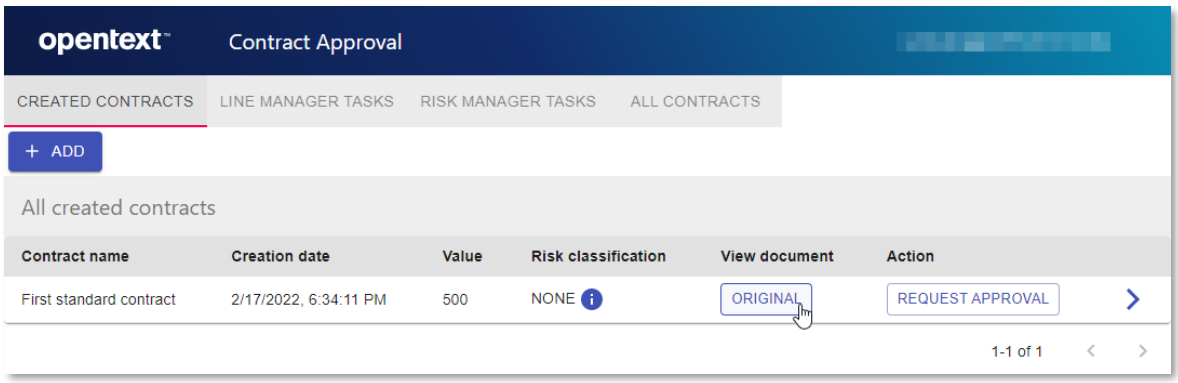
Your first standard contract is now created. Let's explore the contract list capabilities from this **CREATED CONTRACTS** view.
First, note that the **Risk classification** property indeed shows **NONE** as risk level. Click on the ⓘ icon right next to the **NONE** risk classification value to see which terms the call to the Magellan Risk Guard API has identified and extracted.



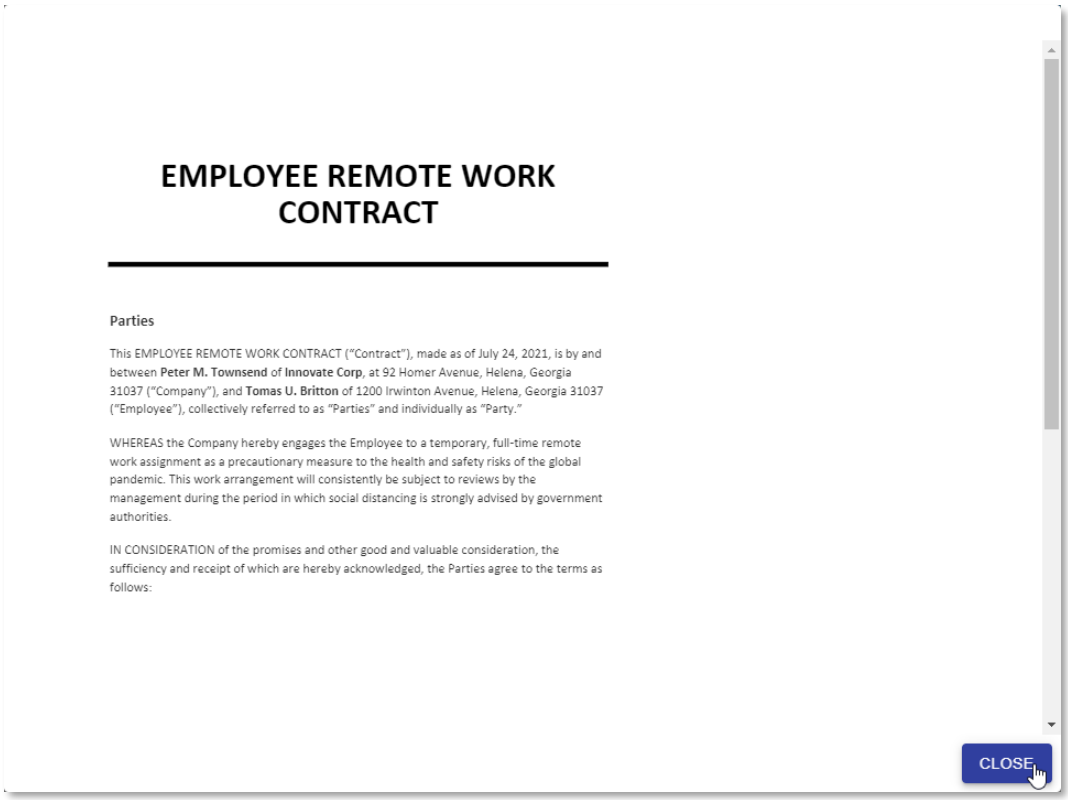
A few names, addresses, geographic locations, and organization names were found, but nothing that warrants increasing the risk level (hence risk classification = NONE). Click **CLOSE** to close the **Extracted Terms** information screen.



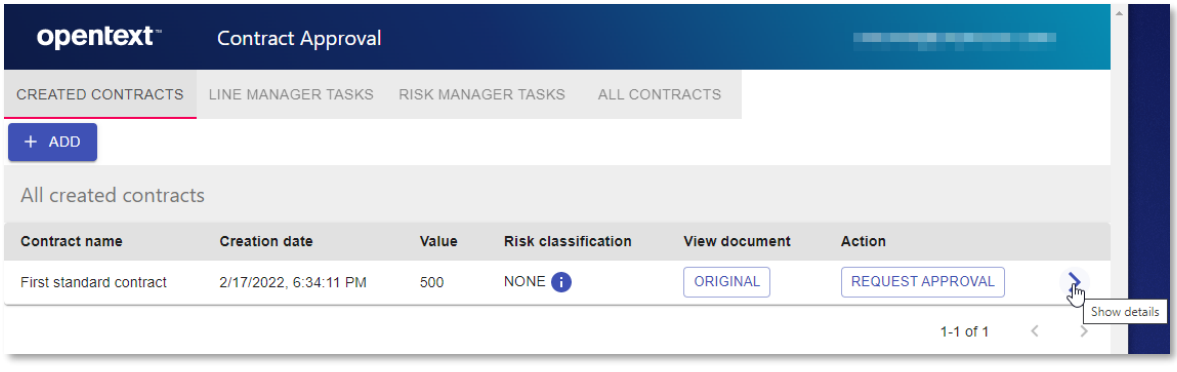
Click on the **ORIGINAL** button (in the **View document** column) to view the uploaded document content.



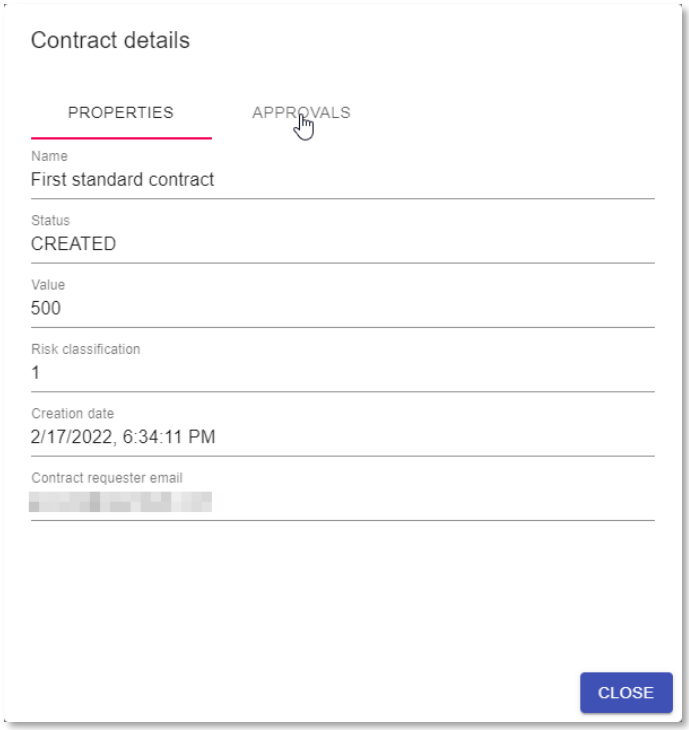
The file content of the contract indeed displays to be viewed.
Click **CLOSE** to close the document viewer screen.



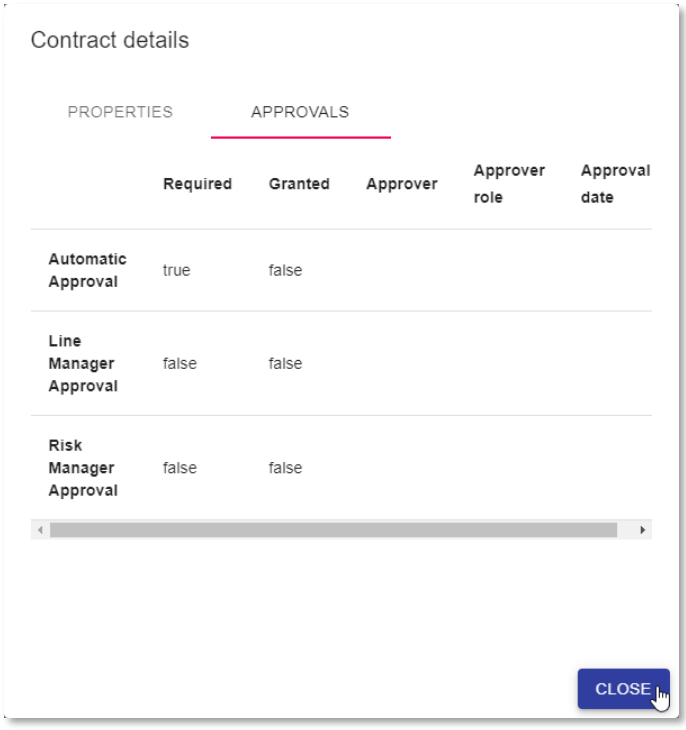
Before requesting to start the contract approval process (clicking the **REQUEST APPROVAL** button under the **Action** column), click on > to view the contract details.



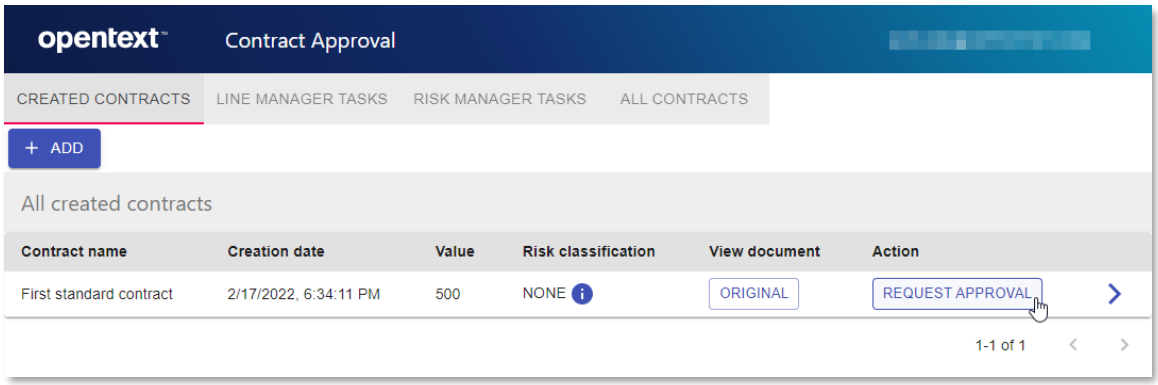
In the **Contract details** screen, you can see two tabs. The first one shows the contract properties.
Note that the **PROPERTIES** tab indeed displays a status of 'CREATED' and that the risk classification is 1 (which is the corresponding integer value for the NONE risk level).
Click the **APPROVALS** tab to have a look at the different approvals for this contract.



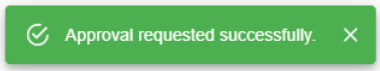
The **APPROVALS** tab displays the different approval steps (traits) for the standard contract type (**Automatic Approval**, **Line Manager Approval** and **Risk Manager Approval**). As you can see, the only approval step that has been marked as required is the **Automatic Approval**.
Click **CLOSE**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



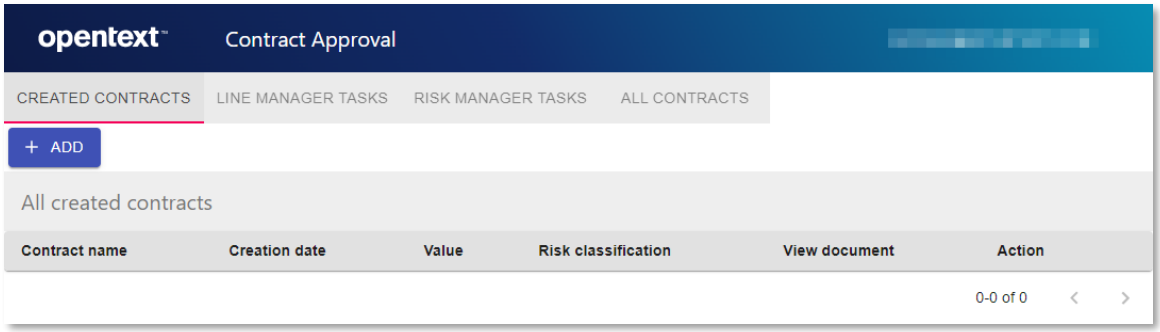
Back into the **CREATED CONTRACTS** view, we can now launch the approval workflow. Click the **REQUEST APPROVAL** button in the **Action** column to do that.



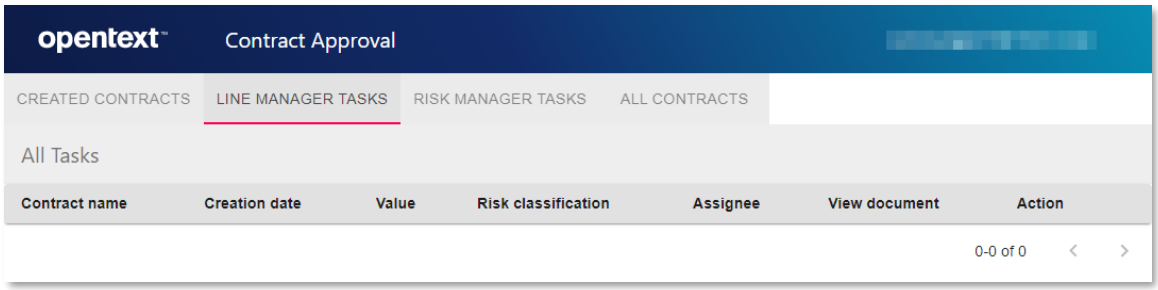
At the bottom of the screen the “Approval requested successfully” message confirms the approval process has been started.



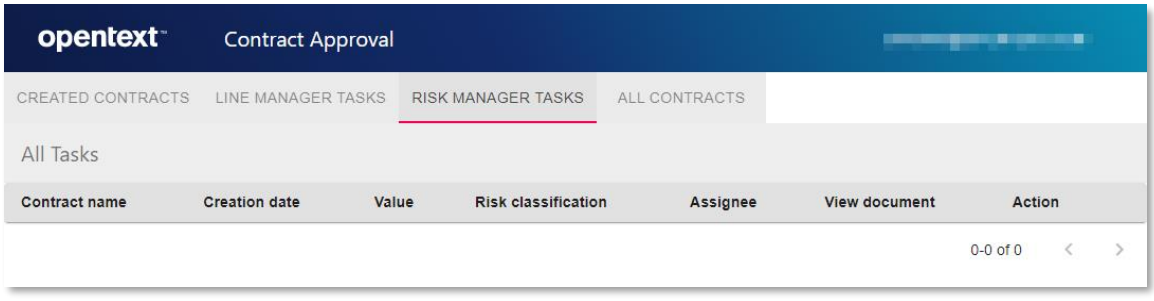
The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in ‘CREATED’ status.



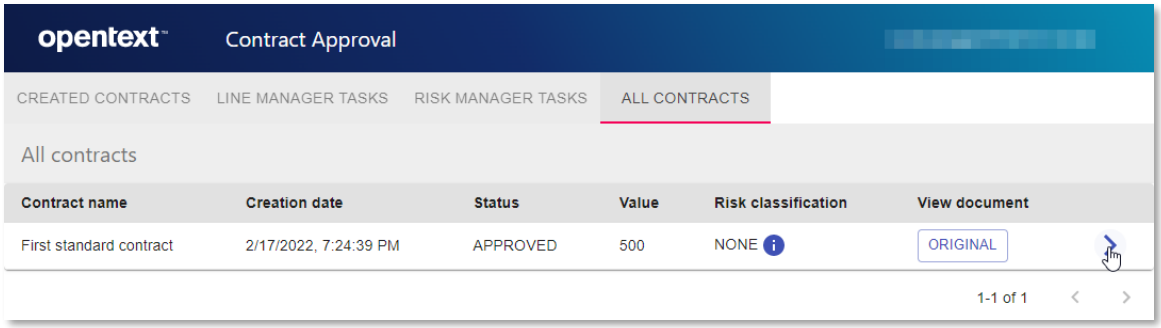
Since the value of the contract is below 1000, there is no approval task waiting in the Line Manager task inbox.



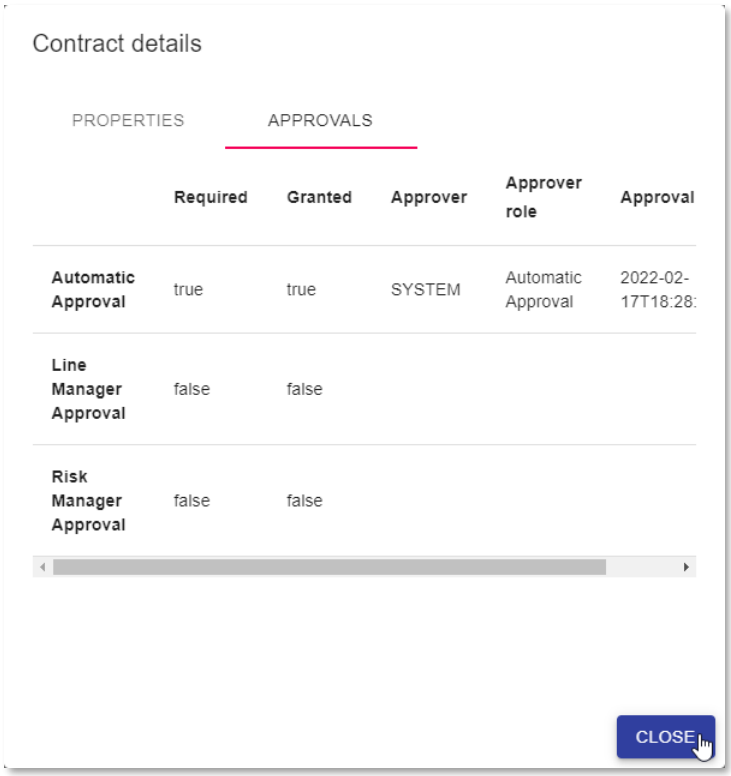
There is also no approval task waiting in the Risk Manager task inbox, as the risk classification is NONE (1) which is certainly below HIGH (4).



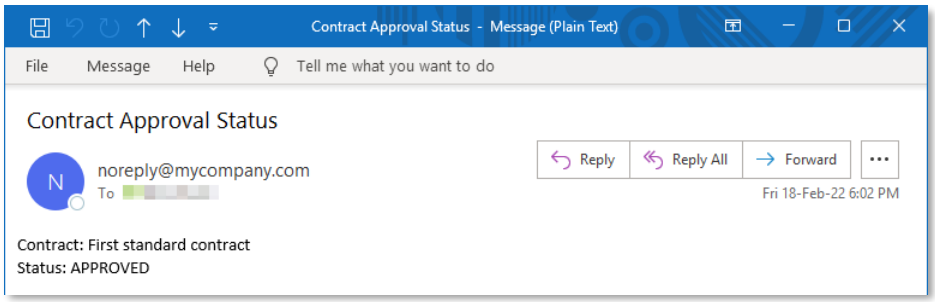
When you open the **ALL CONTRACTS** view, you will see that the contract has been automatically approved (**Status** column shows **APPROVED** status).
Let's now have another look at the contract details as well, and more specifically, the approval steps/traits (click › and select the **APPROVALS** tab).



As you can see, 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.

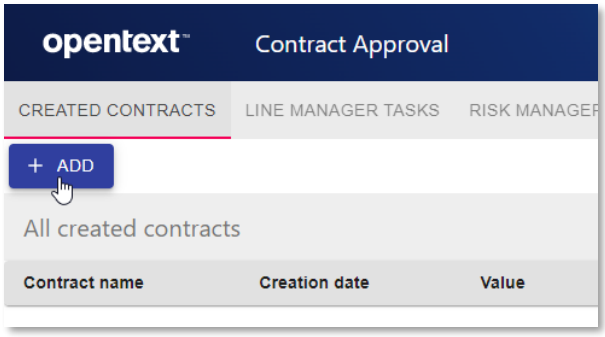


Your first standard contract has now been approved, and you should have received (due to the email task in the workflow) a **Contract Approval Status** email from **noreply@mycompany.com**.

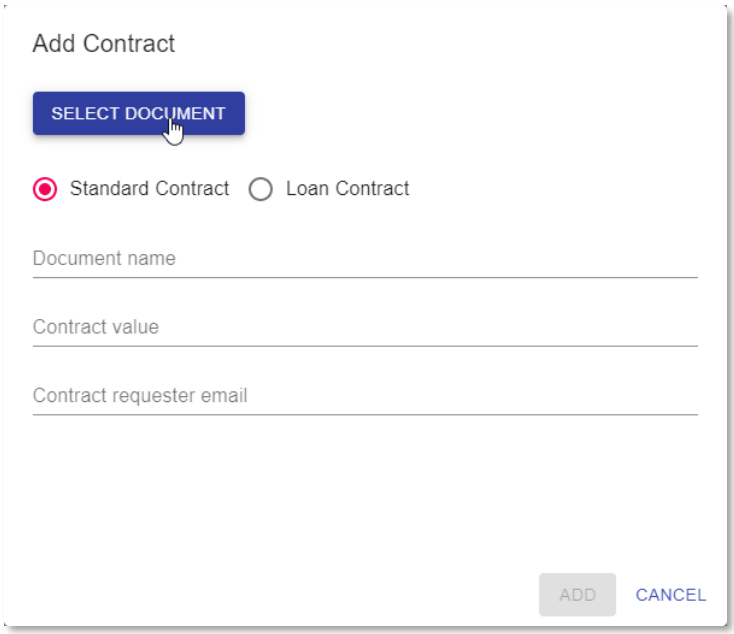


- The first contract we just created and approved has allowed us to go over the different application components and run through the contract approval process. As you know from building the workflow model, there are different possible scenarios that we cater for (standard contract vs. loan contract, solvency check, line manager approval, risk manager approval, automatic approval, reject, and expire). For the rest of this exercise, we will go through these different scenarios by creating and approving (or rejecting, and even expiring) additional contracts.
  The second contract we will be creating will follow the most extensive process flow, i.e.: we 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 should not reject the contract approval request)
  - **Value: above 1000** (requires Line Manager approval)
  - **Risk classification: above 3, i.e.: HIGH or VERIFY HIGH** (requires Risk Manager approval)

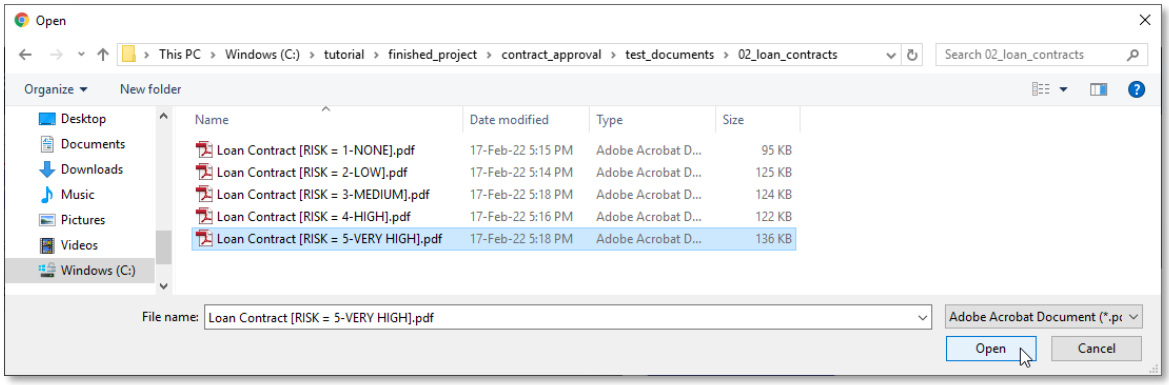
  Select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



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



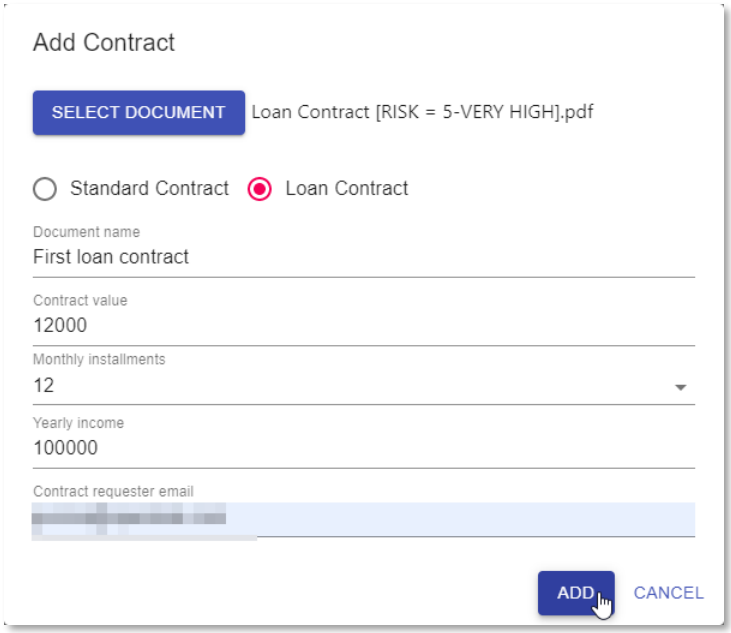
From the **test_documents** folder, open the **02_loan_contracts** subfolder and select the **Loan Contract [RISK = 5-VERY HIGH].pdf** file.

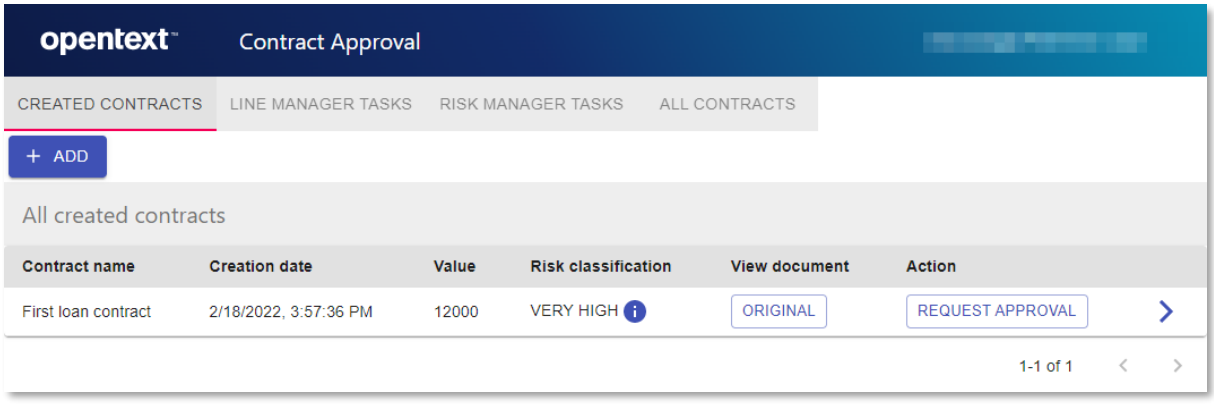


Select the **Loan Contract** option and fill the contract properties as follows:

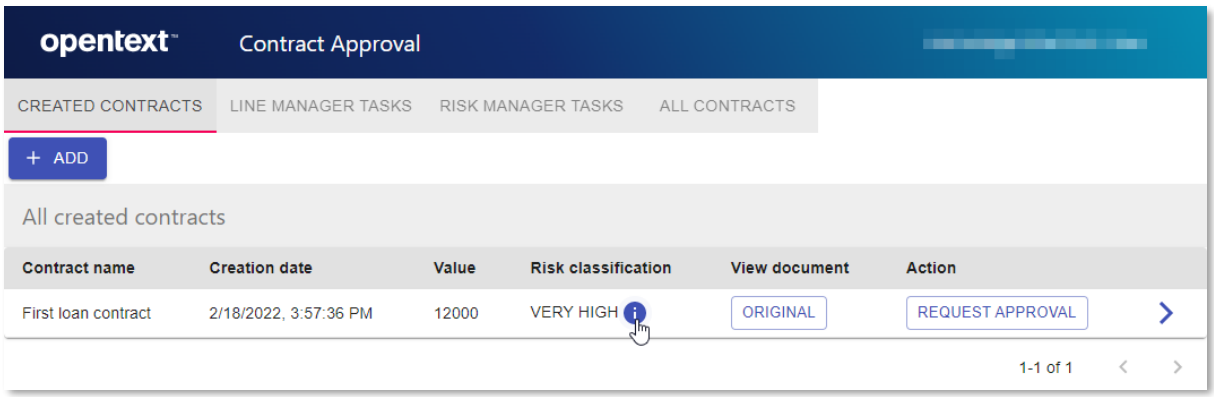
- **Document name: First loan contract**
- **Contract value: 12000**
- **Monthly installments: 12**
- **Yearly income: 100000**
- **Contract requester email: <your email>**

Click **Add** to create the contract.

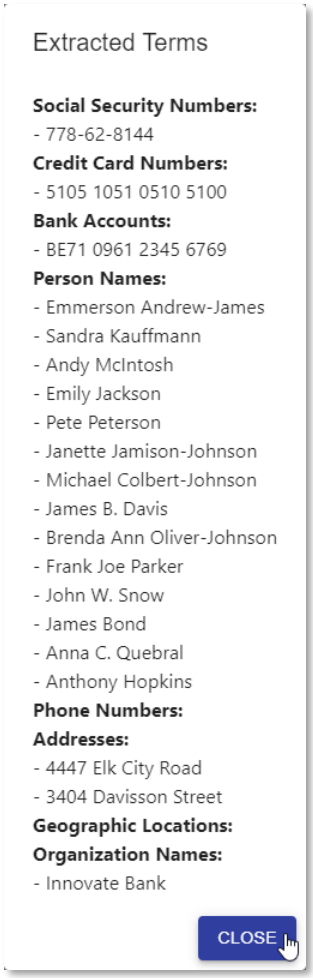




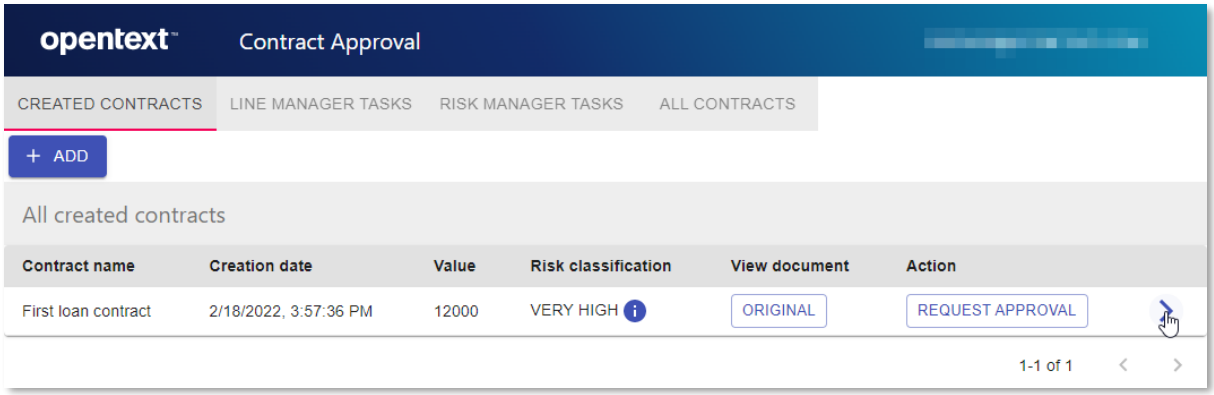
Click on the ⓘ icon right next to the **VERY HIGH** risk classification value to see which terms the call to the Magellan Risk Guard API has identified and extracted.



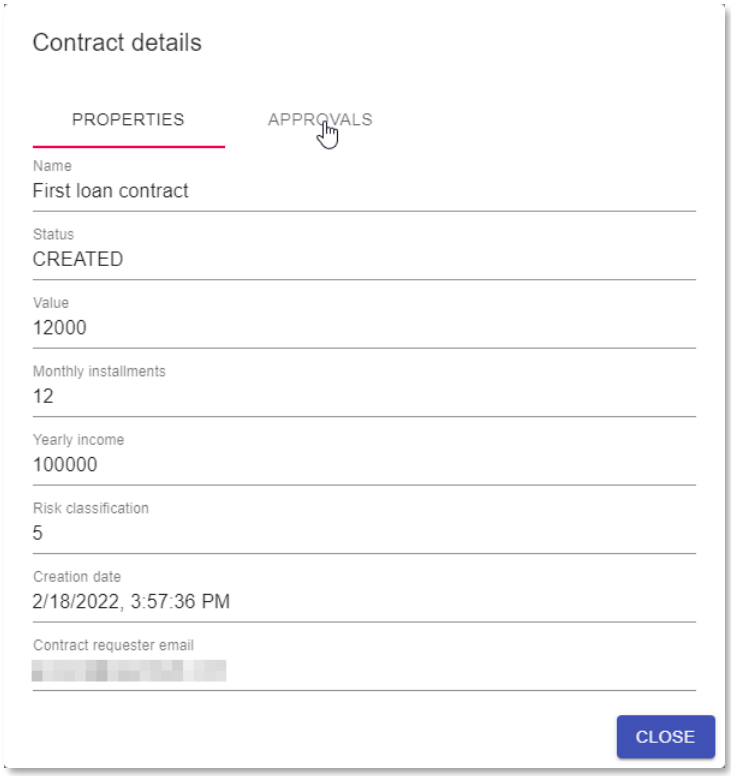
Contrary to the previous contract we created, this contract contains high risk personal information, such as a social security number (considered very high risk), a credit card number, a bank account, and many person names (hence risk classification = VERY HIGH). Some addresses, geographic locations,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



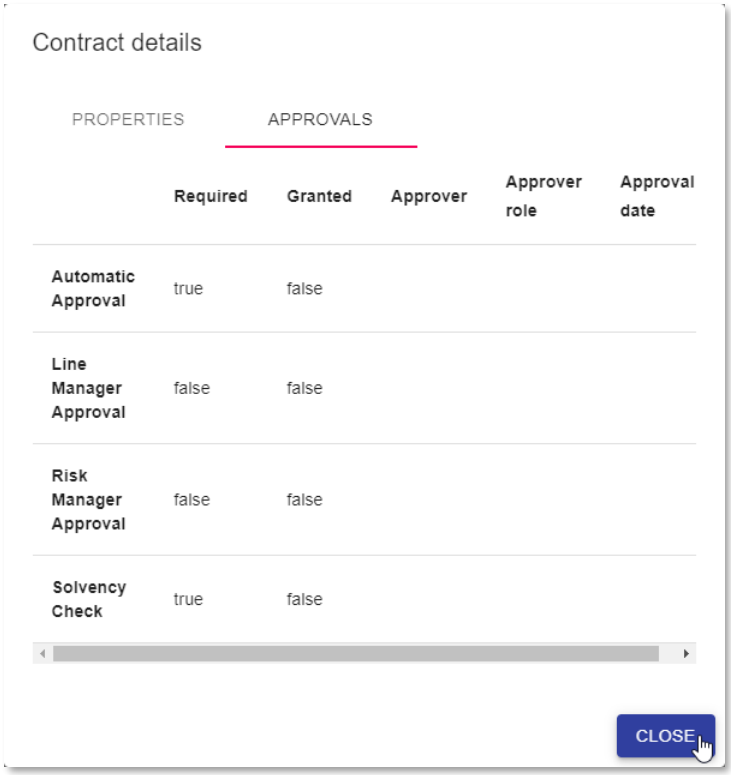
Before requesting to start the contract approval process, click on › to view the contract details for your first loan contract.



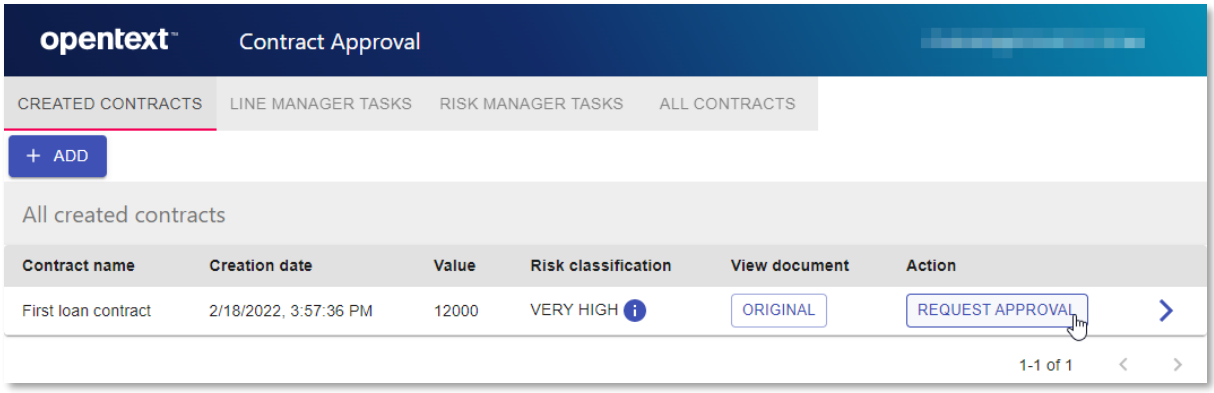
In the **PROPERTIES** tab of the **Contract details** screen note that, since this is a loan contract, the monthly installments and yearly income are also displayed. The risk classification is now equal to 5 (the corresponding integer value for the VERY HIGH risk level).
Click the **APPROVALS** tab to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



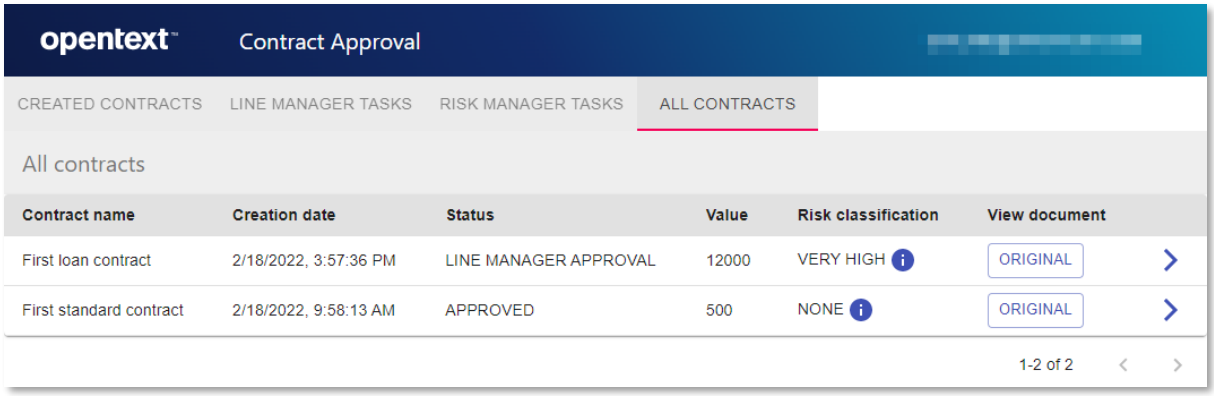
The **APPROVALS** tab displays the different approval steps (traits) for the loan contract type (**Automatic Approval**, **Line Manager Approval**, **Risk Manager Approval**, and the additional **Solvency Check**). As you can see, there are now more approval steps that have been marked as required: the **Automatic Approval** and the **Solvency Check**.
Click **CLOSE** to close the **Contract details** screen.



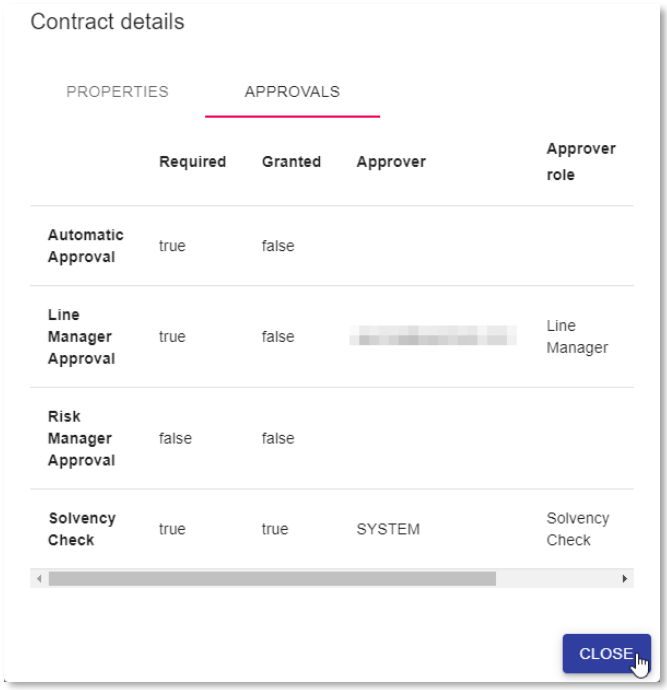
Back into the **CREATED CONTRACTS** view, we can now launch the approval workflow. Click the **REQUEST APPROVAL** button in the **Action** column to do that.



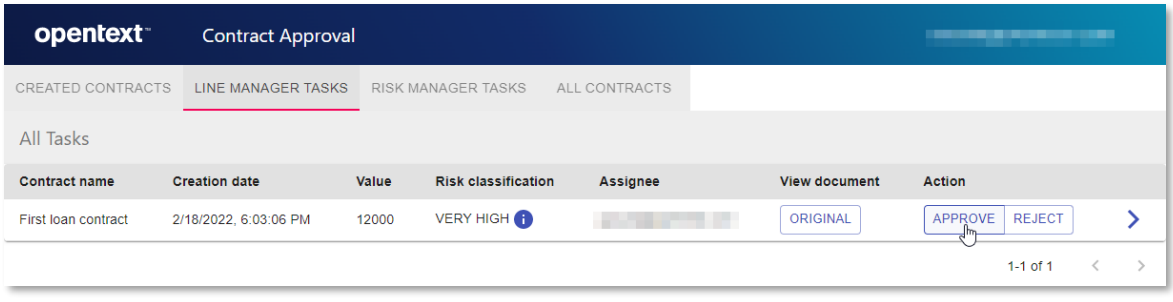
The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status. Let's have a look at the **ALL CONTRACTS** tab to see its current status.
The status is now 'LINE MANAGER APPROVAL' (since the value of the contract is above 1000 a Line Manager approval is required).
Click on > to view the contract details again.



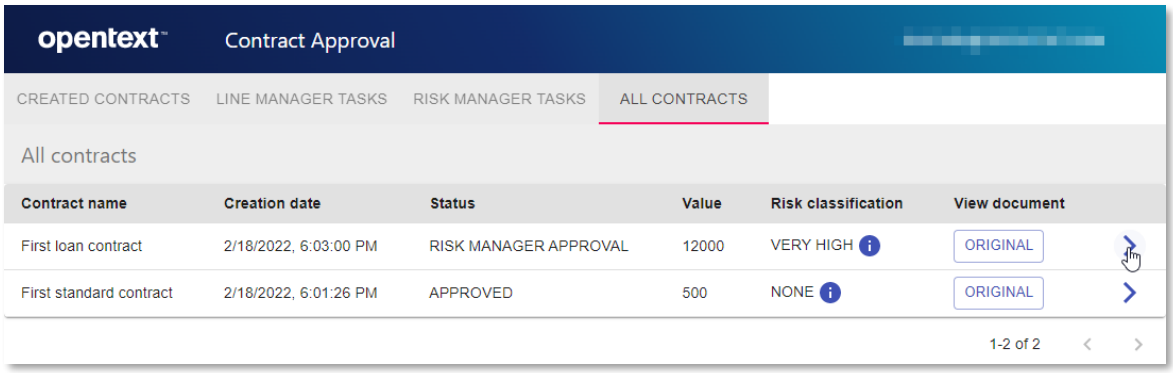
You can now see that the **Solvency Check** approval has been granted (since the requester is indeed solvent) and that the **Line Manager Approval** is required (and the approver has been assigned).



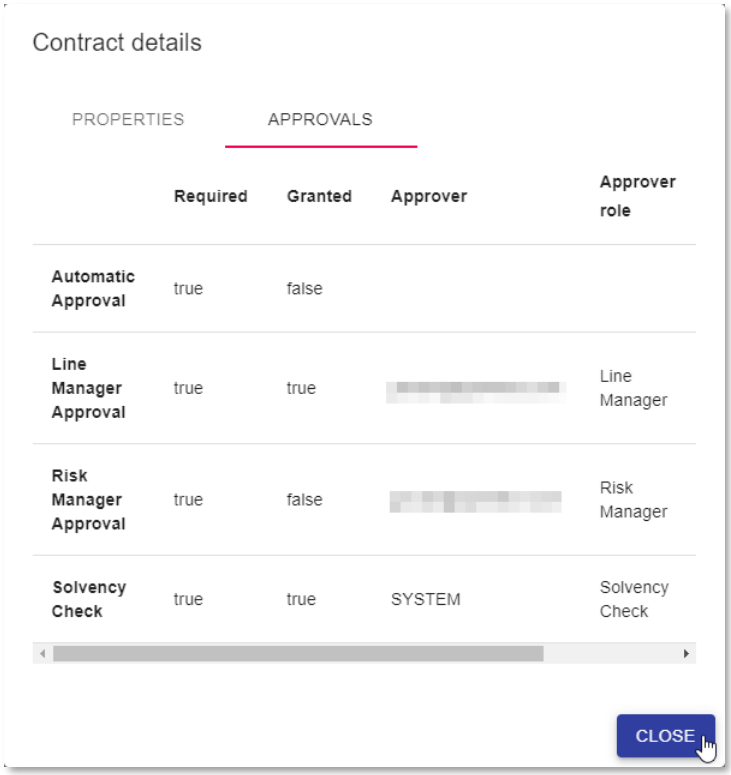
Go to the **LINE MANAGER TASKS** tab. As you can see, an approval task is waiting for the Line Manager to approve.
Click **APPROVE** to approve as the Line Manager.



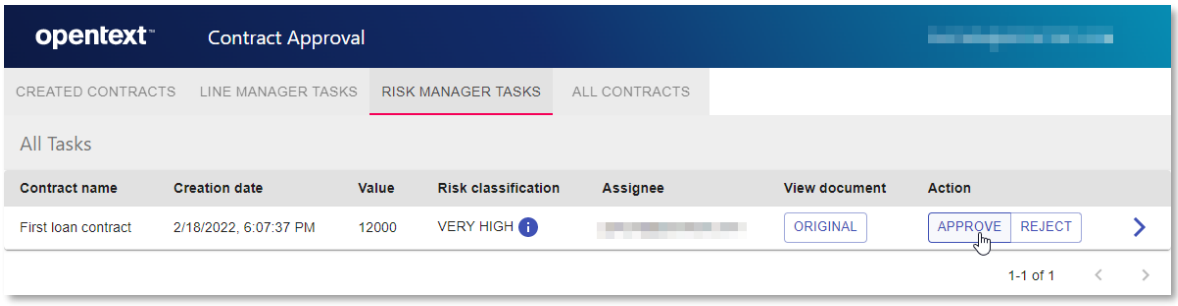
The contract has now disappeared from the **LINE MANAGER TASKS** view, as it is no longer in 'LINE MANAGER APPROVAL' status. Let's have a look at the **ALL CONTRACTS** tab to see its current status.
The status is now 'RISK MANAGER APPROVAL' (since the risk classification of the contract is VERY HIGH).
Click on > to view the contract details again.



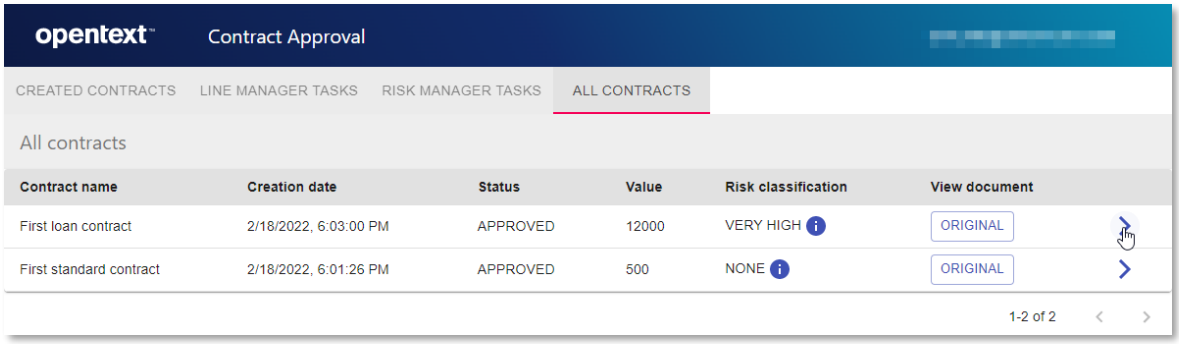
You can now see that the **Line Manager Approval** has been granted and that the **Risk Manager Approval** is required (and the approver has been assigned).



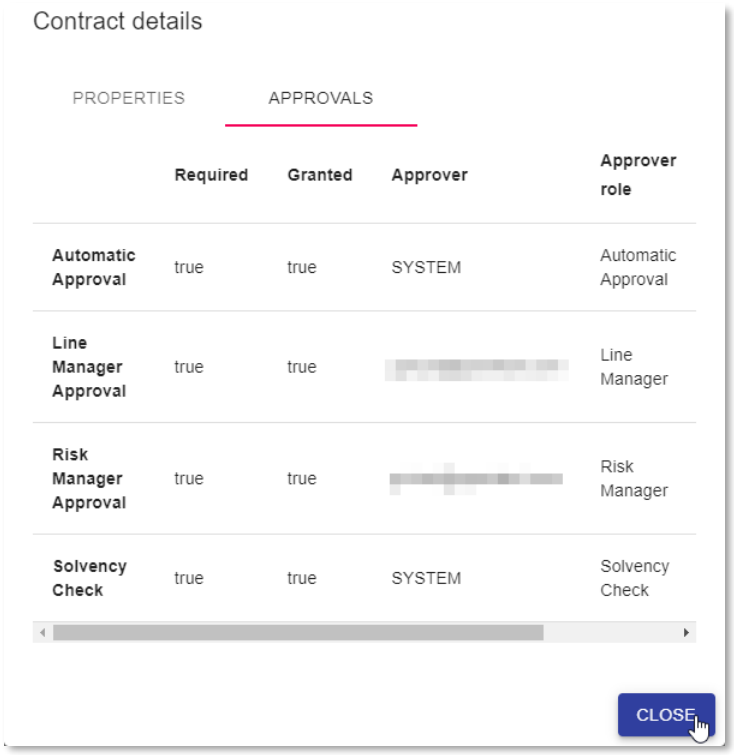
Go to the **RISK MANAGER TASKS** tab. As you can see, an approval task is waiting for the Risk Manager to approve.
Click **APPROVE** to approve as the Risk Manager.



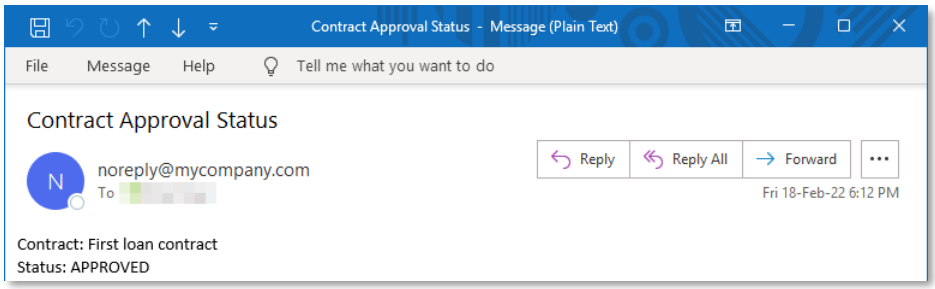
The contract has now disappeared from the **RISK MANAGER TASKS** view, as it is no longer in 'RISK MANAGER APPROVAL' status. When you open the **ALL CONTRACTS** view, you will see that the contract has been automatically approved (**Status** column shows **APPROVED** status).
Click on > to view the contract details to check the different approvals (traits).



You can now see that both the **Risk Manager Approval** and the **Automatic Approval** have been granted. I.e.: all four approvals were required for this loan contract, and all four approvals have been granted.

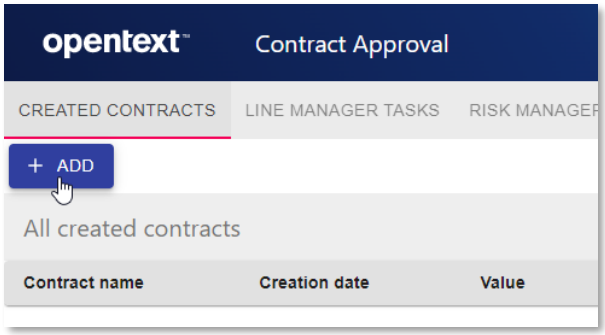


Your first loan contract has now been approved, and you should have received the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.

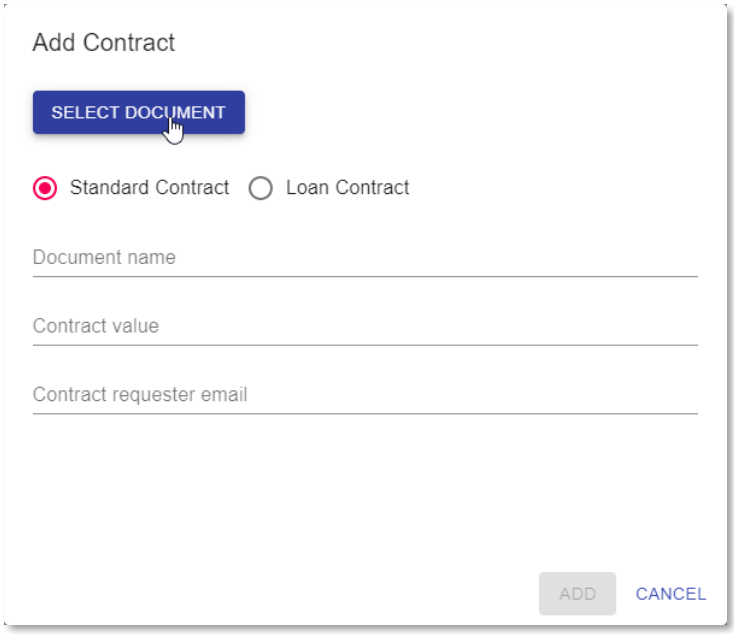


- We have now successfully approved two contracts with two completely different approval flows. Let's continue with the scenario where an approver does not approve (i.e.: rejects) the contract. We will create a contract with the following characteristics:
  - **Type: standard contract**
  - **Value: above 1000** (requires Line Manager approval)
  - **Risk classification: above 3, i.e.: HIGH or VERIFY HIGH** (requires Risk Manager approval)

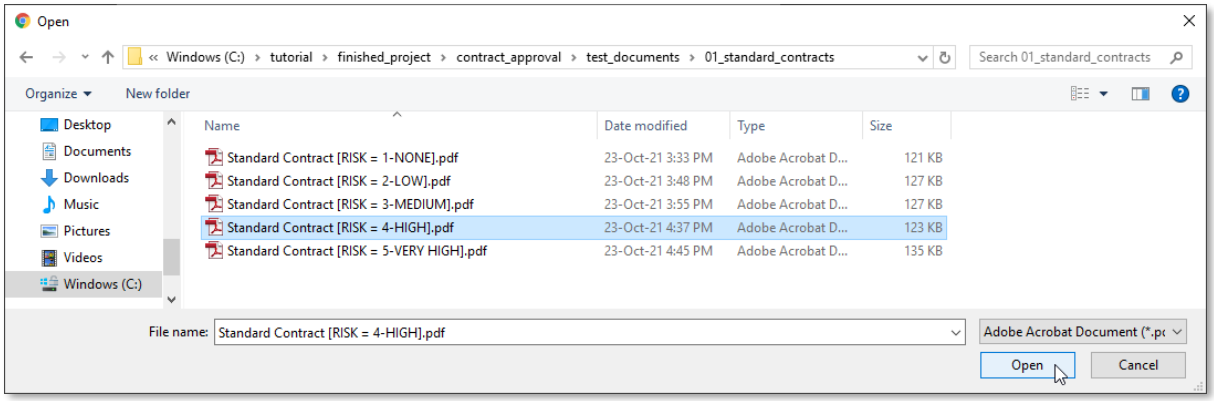
  Select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



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



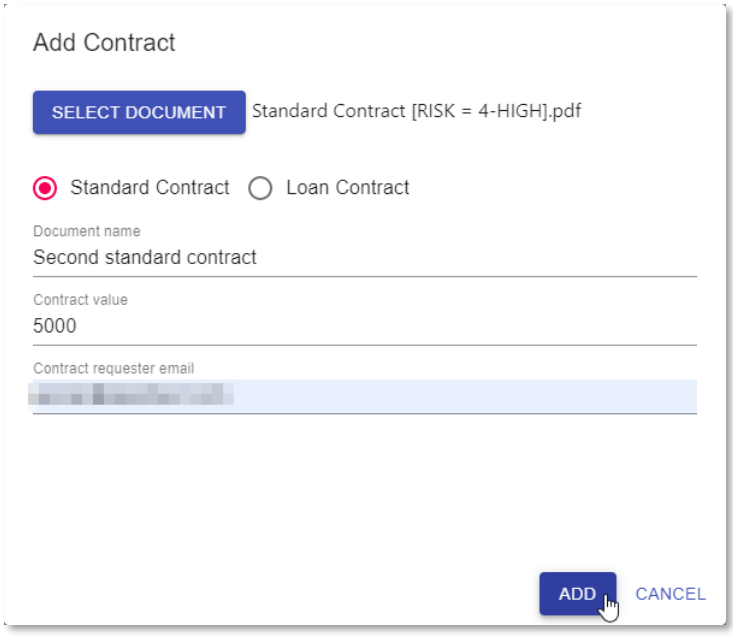
From the **test_documents** folder, open the **01_standard_contracts** subfolder and select the **Standard Contract [RISK = 4-HIGH].pdf** file.



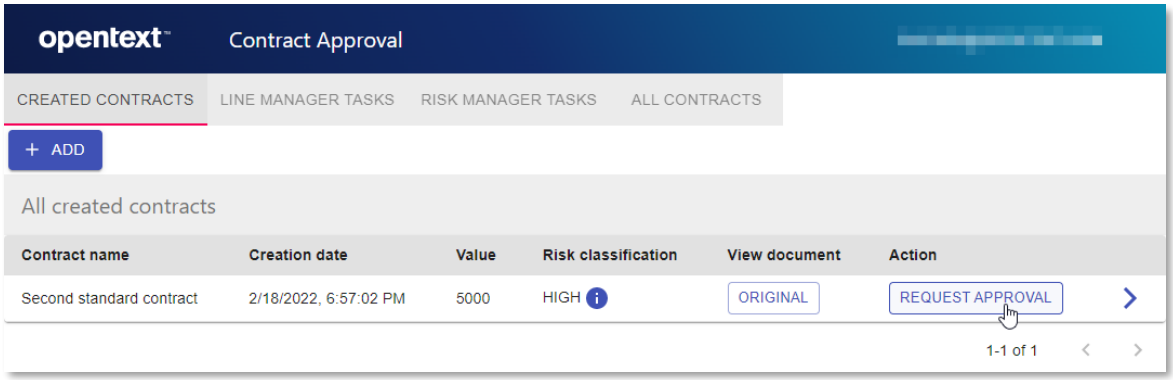
Make sure the **Standard Contract** option is selected and fill the contract properties as follows:

- **Document name: Second standard contract**
- **Contract value: 5000**
- **Contract requester email: <your email>**

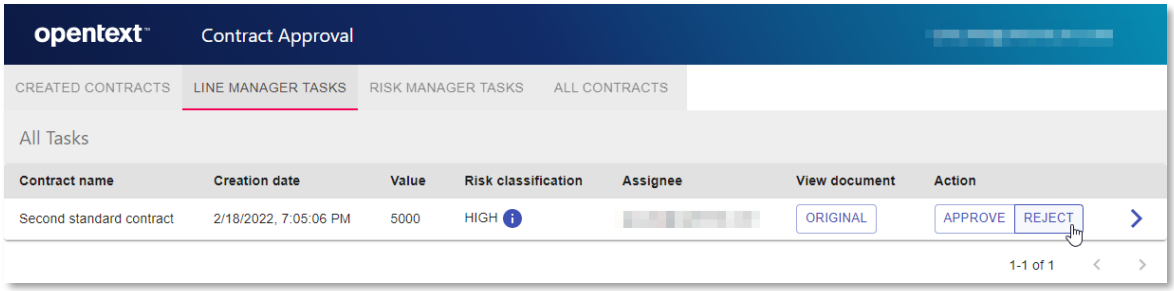
Click **Add** to create the contract.



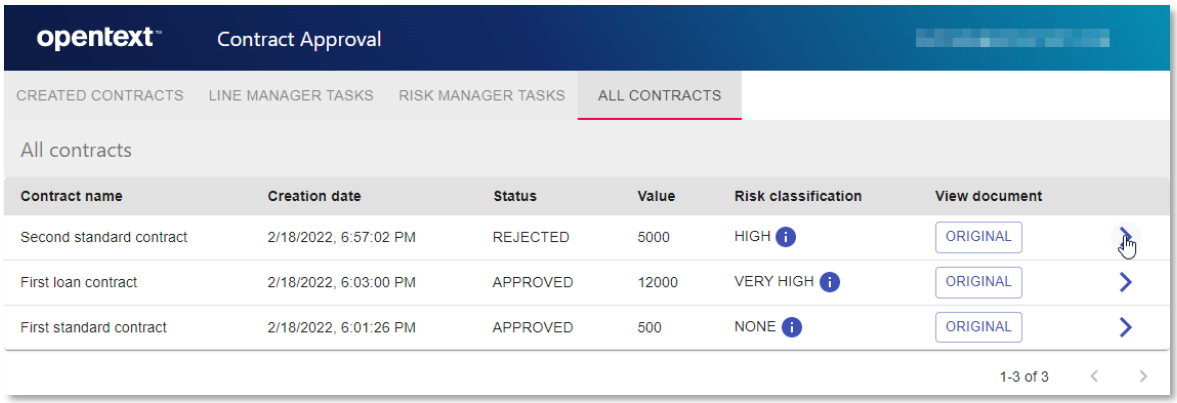
Click the **REQUEST APPROVAL** button in the **Action** column to launch the approval workflow.



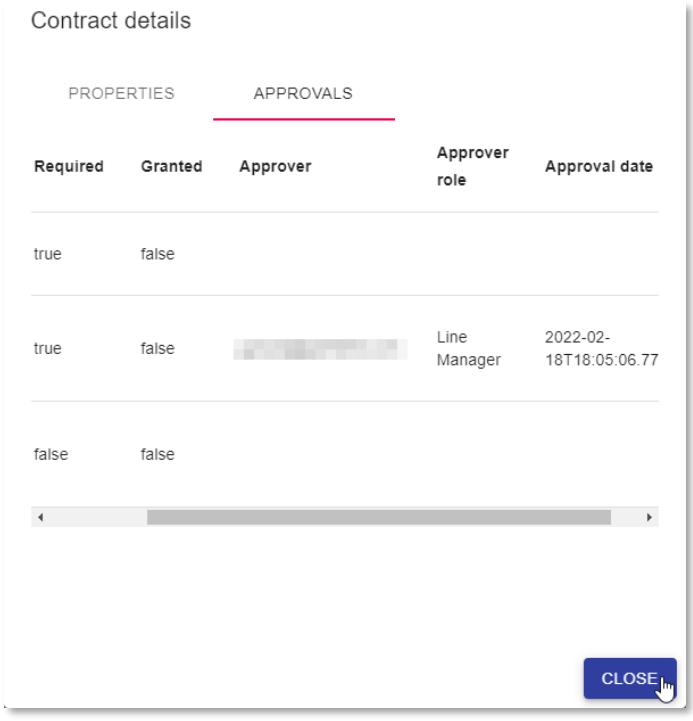
The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status. Go to the **LINE MANAGER TASKS** tab. You should see the new Line Manager approval task.
Click **REJECT** to reject the contract approval.



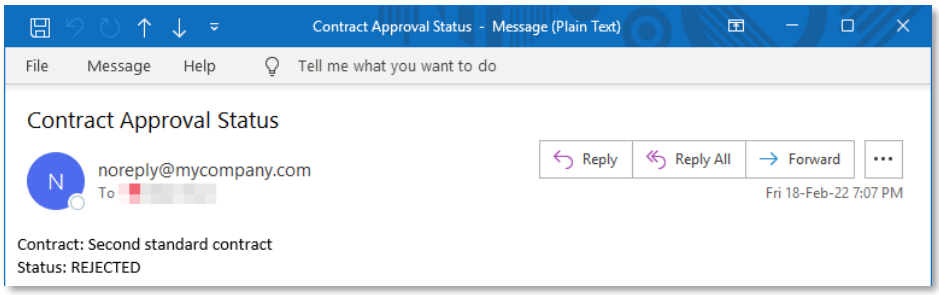
The contract has now disappeared from the **LINE MANAGER TASKS** view, as it is no longer in 'LINE MANAGER APPROVAL' status. When you open the **ALL CONTRACTS** view, you will see that the contract has indeed been rejected (**Status** column shows **REJECTED** status).
Click on > to view the contract details to check the different approvals (traits).



You can see the contract is rejected, since the **Line Manager Approval** has NOT been granted while it has a date at which the approval task was completed.

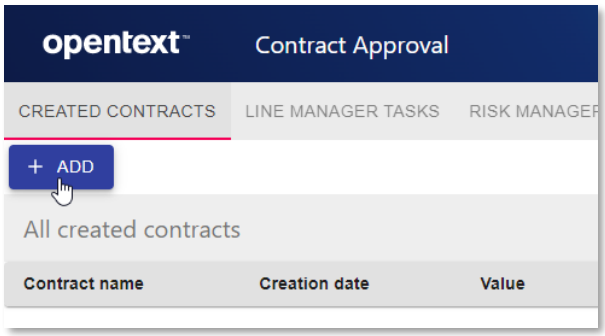


Your second standard contract has NOT been approved (i.e.: it has been rejected), and you should have received the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.

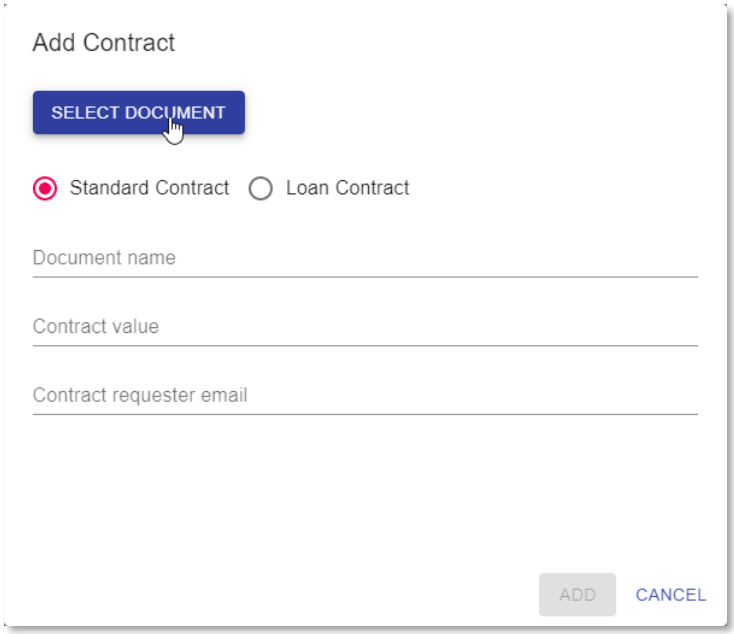


- The last scenario we want to run through is to let an approval task expire (happens after 5 minutes).
  We will create a contract with the following characteristics:
  - **Type: loan contract**
  - **Value: below 1000** (doesn't require Line Manager approval)
  - **Risk classification: above 3, i.e.: HIGH or VERIFY HIGH** (requires Risk Manager approval)

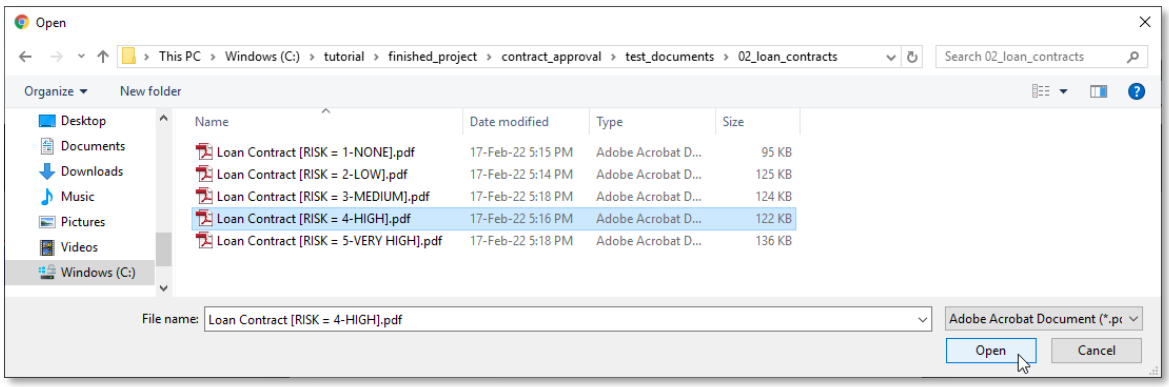
  Select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



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



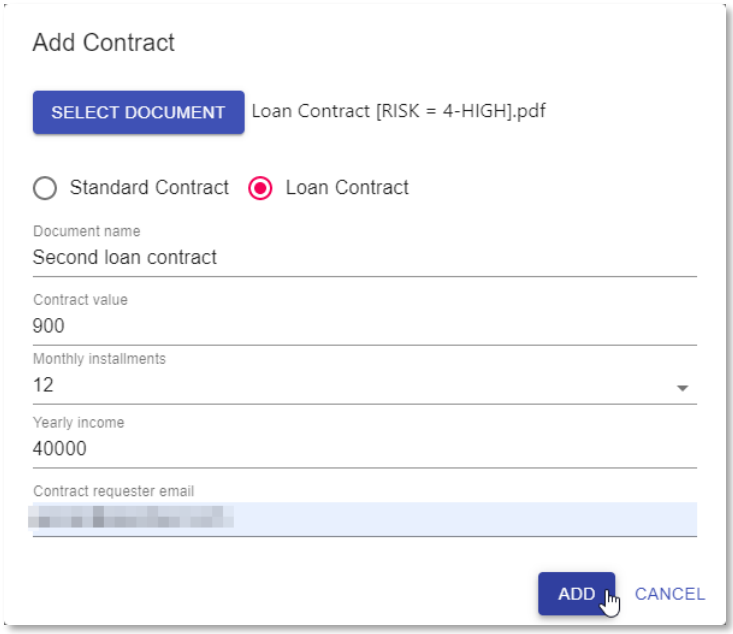
From the **test_documents** folder, open the **02_loan_contracts** subfolder and select the **Loan Contract [RISK = 4-HIGH].pdf** file.



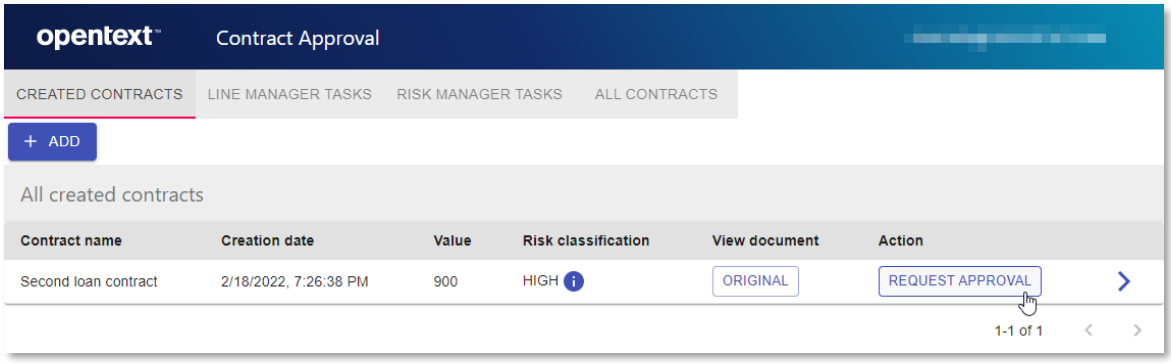
Select the **Loan Contract** option and fill the contract properties as follows:

- **Document name: Second loan contract**
- **Contract value: 900**
- **Monthly installments: 12**
- **Yearly income: 40000**
- **Contract requester email: <your email>**

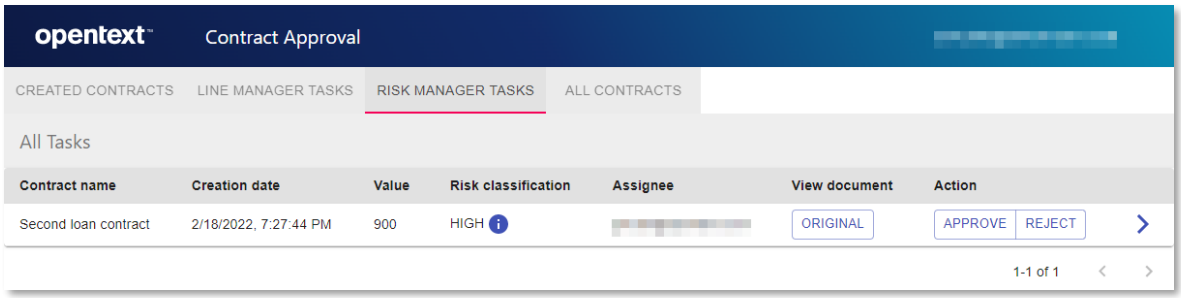
Click **Add** to create the contract.



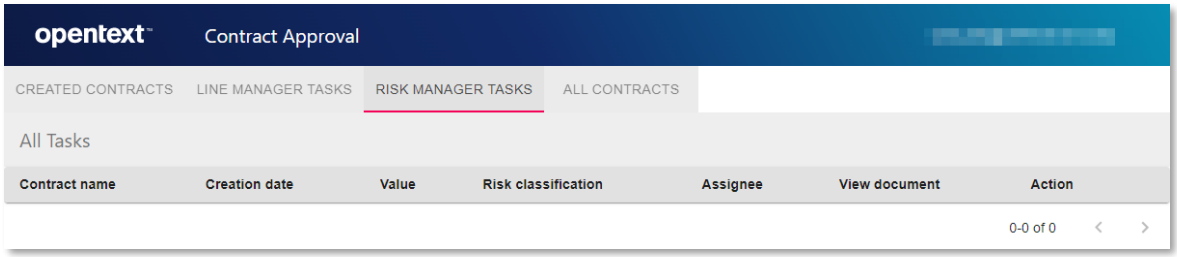
Click the **REQUEST APPROVAL** button in the **Action** column to launch the approval workflow.



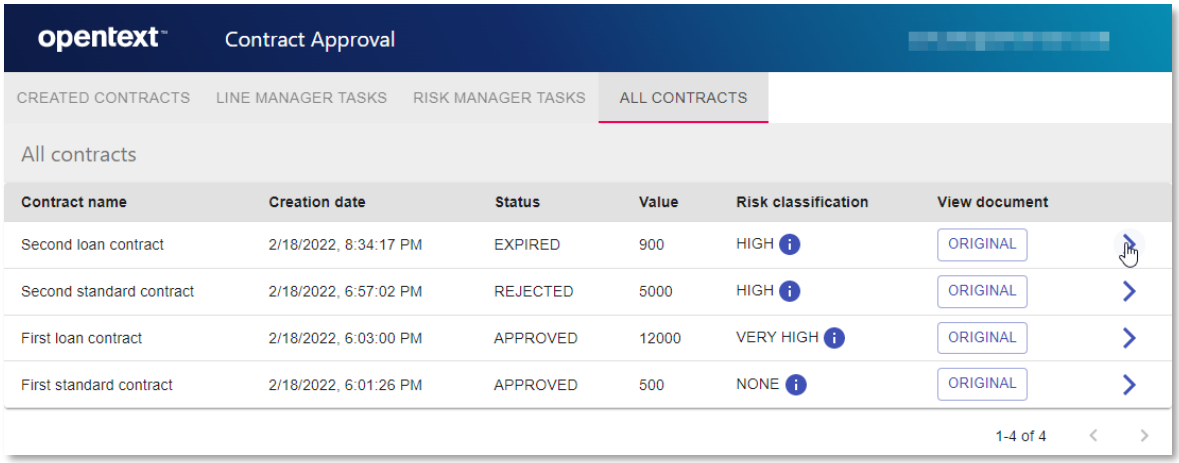
The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status. Go to the **RISK MANAGER TASKS** tab. You should see the new Risk Manager approval task.



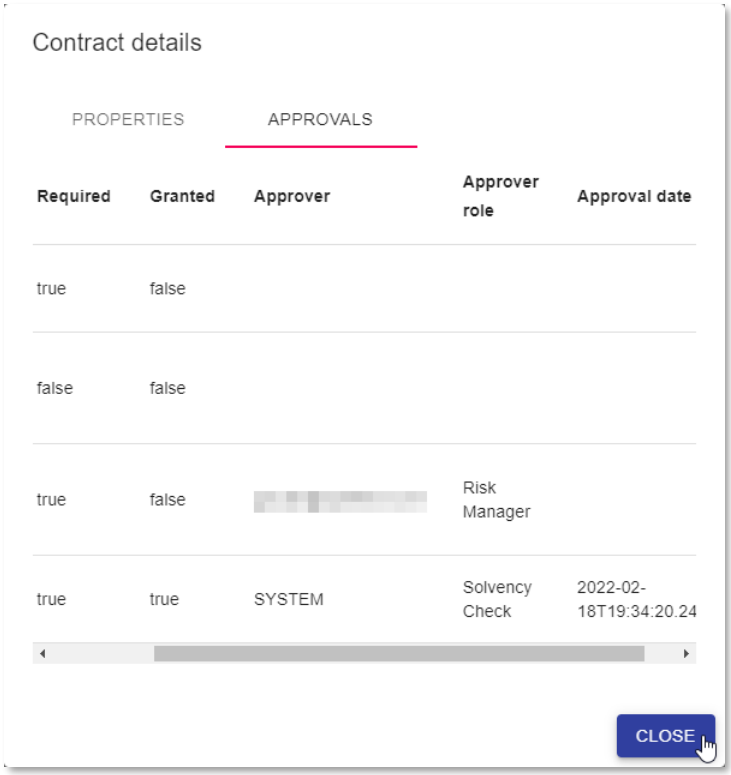
Since we want to test whether or not the approval task will expire, you need to wait. You have configured the timeout wait time to be 5 minutes, so wait a little longer than that, let's say 10 minutes, and refresh your browser (you might have to log in again as well).
After refreshing your application screen, go back to the **RISK MANAGER TASKS** view.
The contract has now disappeared from the **RISK MANAGER TASKS** view, as it is no longer in 'RISK MANAGER APPROVAL' status.



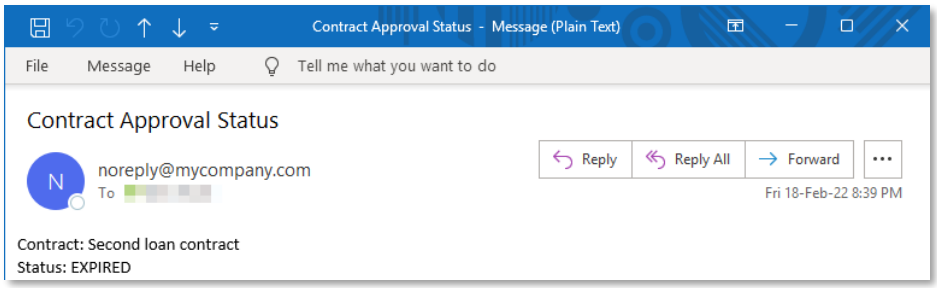
When you open the **ALL CONTRACTS** view, you will see that the contract approval has indeed expired (**Status** column shows **EXPIRED** status).
Click on > to view the contract details to check the different approvals (traits).



You can see 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.



Your second loan contract has not been approved since the Risk Manager Approval step expired, and you should have received the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.



This calls for a second **CONGRATULATIONS!**

You have now completely finished building and testing your 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 otcloud CLI. If you are interested in build automation and CI/CD for your applications, we recommend you certainly do that exercise as well.

3.14[00’] Bonus exercise: Using the otcloud Command Line Interface

COMING SOON

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