

The Basic Process

Overview

A new travel request case is created with three stages: defining the travel data, approving the travel, and organizing it. The approval process is straightforward with only one approver.

make the simple approval process more sophisticated using a DMN based decision table in front to define, if and how many approvers are needed. It also uses a multi-instance process to do the approval.

Create a Basic Case Model, App and Deploy

Create Travel Request App

Models in Flowable are app focused. An **app** is a container of all the items related to a specific process, and in this example, all the content related to the case model we are building.

Open your Workspace on Flowable Design and hit Create.

Set the name to *Travel Request*, the key is automatically filled to *travelRequest* and optionally add a meaningful description of what the app contains.

Create a new app

CreateImport

Name *

Travel Request

Key *

travelRequest

Palette type *

Work

Description

B *I* {}

This is an app for automating Business Travel Request approvals.

Tags

Select...

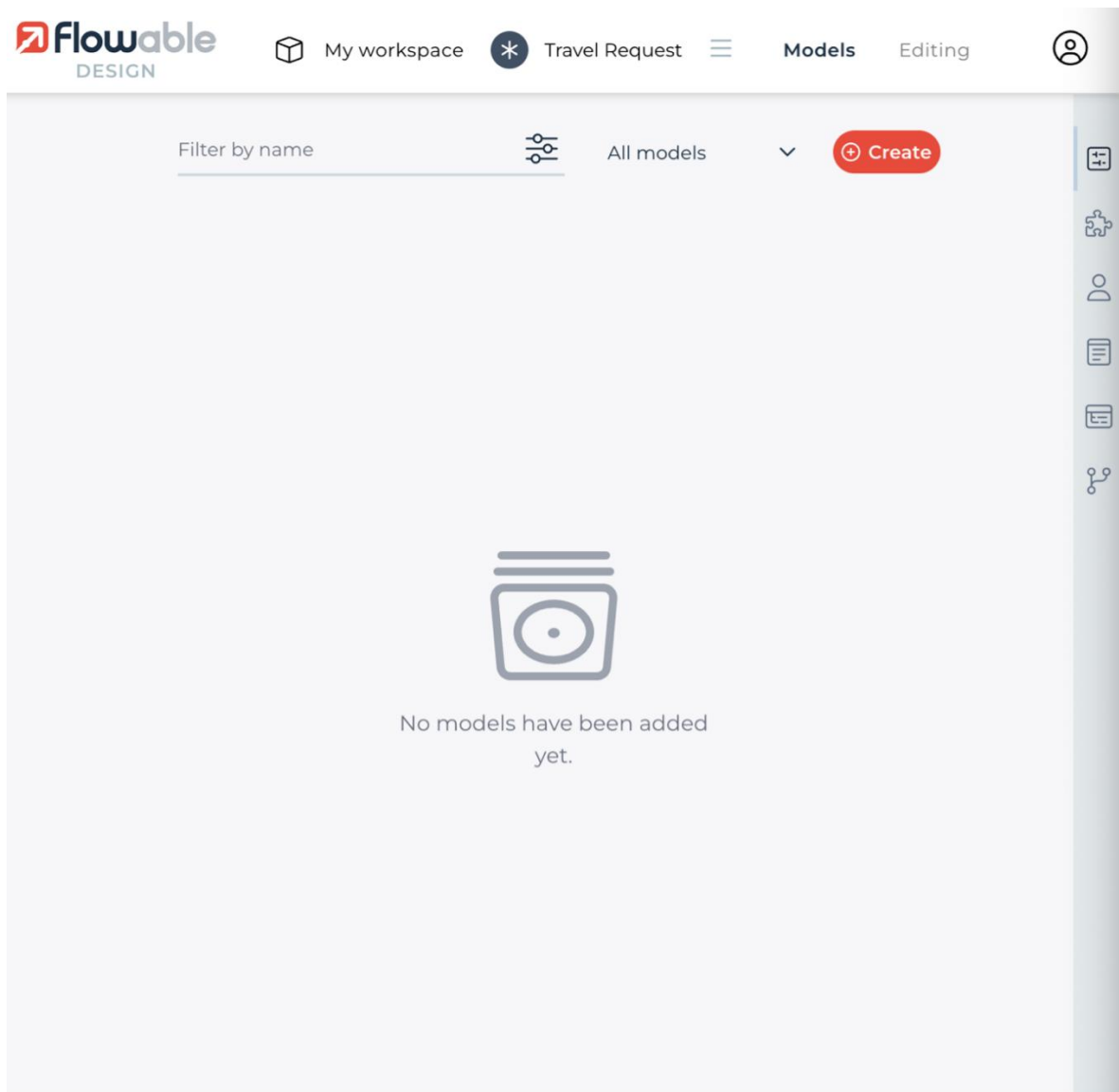
i

You can create a new app by giving it a name. Its key can be auto-generated from its name or you can specify one yourself. You also have the option of importing an app by uploading a file of the correct format

Cancel

Create

Click on `Create` and you are in the app view:



Create the Case

In the app view, click `Create`. In the dialog, name the case model *Travel Request*, the key is automatically set to *travelRequest* and optionally add a meaningful description of the case model. Then click `Create`.

×

Open or create a new model

CreateImportInclude

Model Type *

Case

Name *

Travel Request

Key *

travelRequest

Palette type

Flowable Work CMMN

Tags

Select...

Description

B *I* **{}**

Using a travel request, you can create, manage, approve and organize a business trip.

Template

Select...

You can create a new model by choosing the type of model and giving it a name. Its key can be auto-generated from its name or you can specify one yourself. You also have the option of importing a model by uploading a file of the correct format.

For models that are designed with a palette and canvas, the palette type can be chosen when multiple palettes are installed in the system. This palette determines which

Cancel

Create

Once created, the case editor opens, and we can start defining our travel request case.

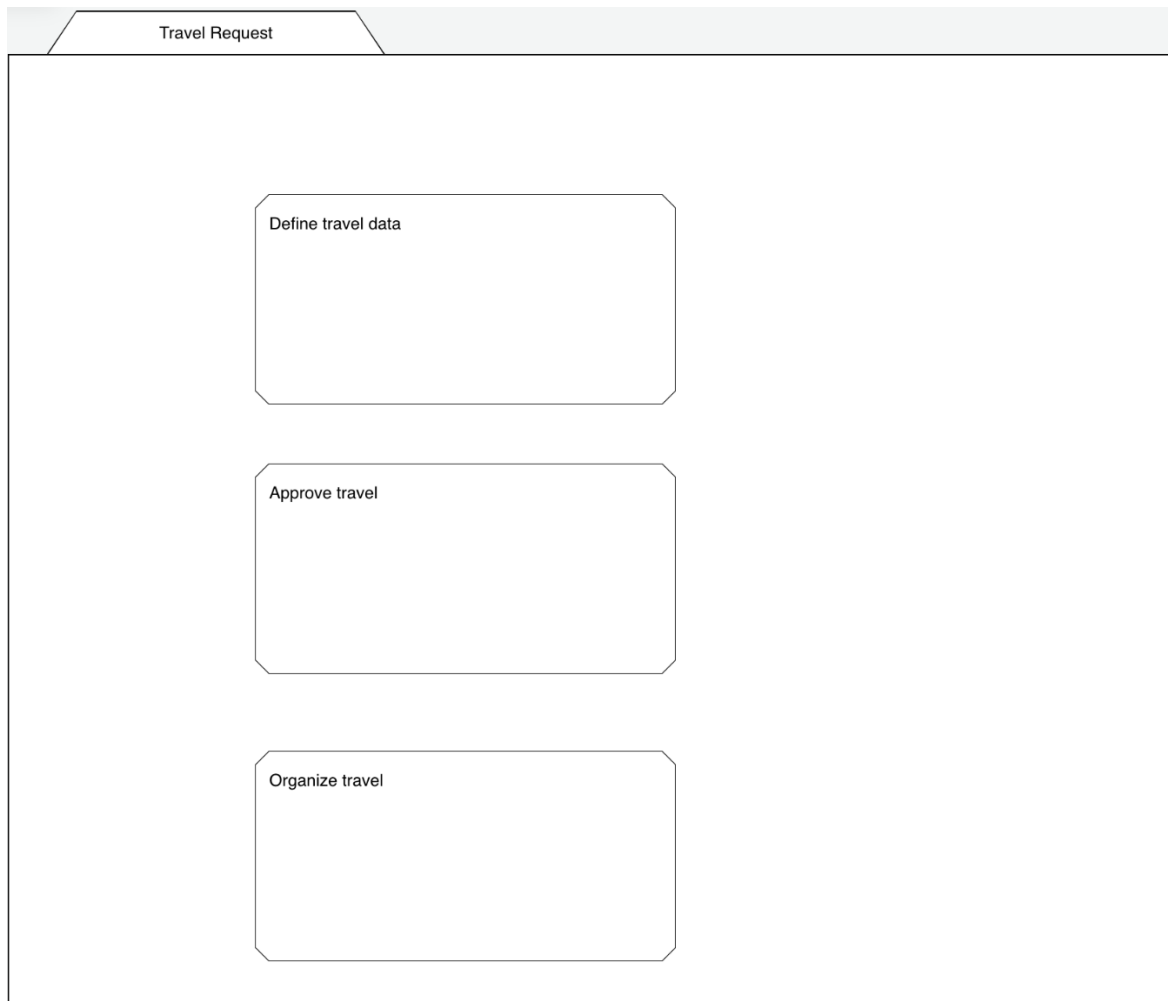
Double click the `Case plan model` title and rename it to *Travel Request*. The name has no execution relevance but makes more sense from a business model perspective.

Add Case Stages

Now let us add some stages. A stage is like a phase within the case lifecycle and can mean anything, from a technical to a business perspective. Think of your case at a high level and ask yourself: through what kind of phases (stages) is the case going? What needs to be accomplished in the case?

Let us add three stages (`Containers>Stage`) for our travel request model: *Define travel data*, *Approve travel* and *Organize travel*.

The resulting case looks like:



Define Starting User Group

First, we need to define who can start this case. Click outside the case model and open the panel on the right hand side and select the `Assignment>Potential starter group` attributes of the case model and set the group to *Flowable User*. Setting this attribute allows all users of the group `Flowable Users` the ability to start a travel request case.

Case Diagram
Travel Request (travelRequest)

Potential starter user ⓘ
Select...
↕

Potential starter group ⓘ
Flowable User x |
x ↕

i

The **Potential starter users or groups** determine which users or groups can start case instances from a published definition of this CMMN case model.

In Flowable Work, this is automatic. In Flowable Core, this information is available as metadata to build your own filters.

Cancel

Finish

Create Travel Request App and Publish it

To deploy our case model, we need to include it in an **app** that we can then use as a deployment unit. Since we created the case model inside the app context, it is automatically assigned to the app.

In the toolbar of your case model you can press the `Publish` button to deploy the app and make it available for use.

Create Token

Disclaimer: Creating a token is only required when using the Self-Managed(Hosted) version of Flowable. Please skip this step if you are using the Flowable cloud version.

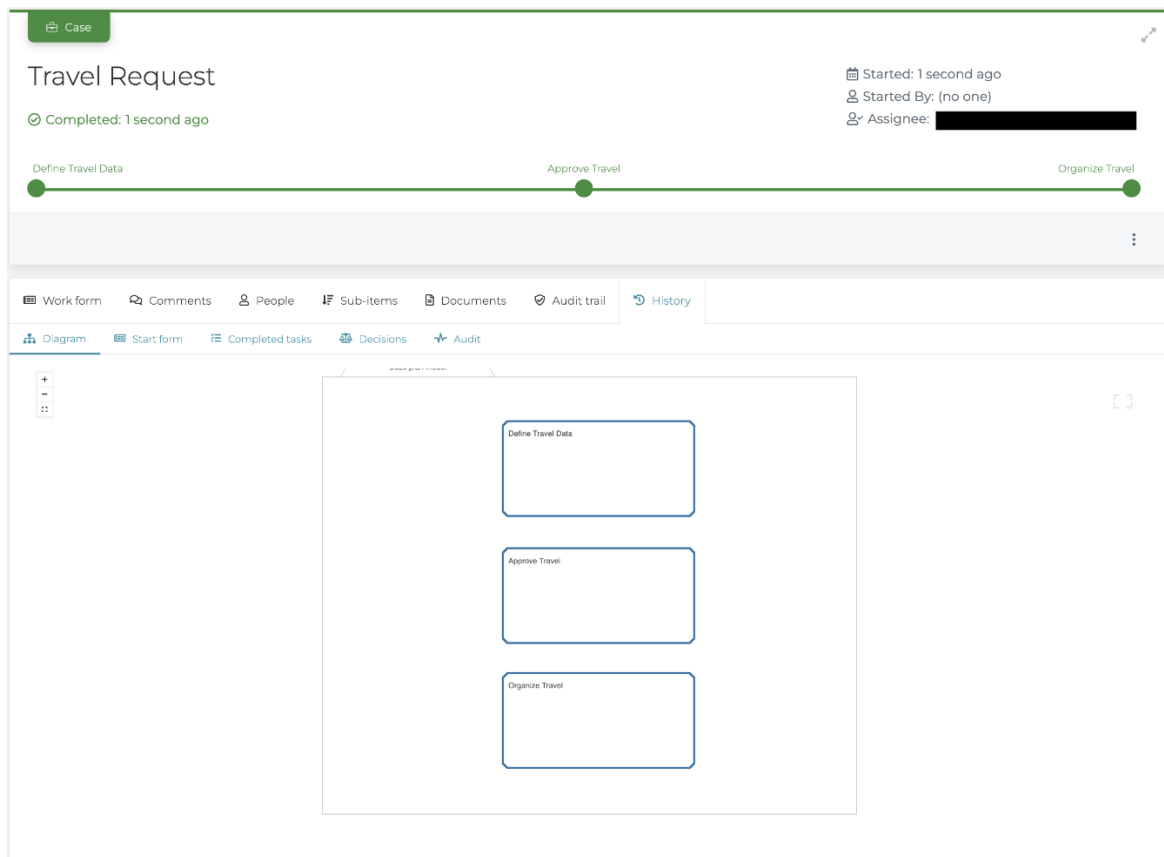
On Flowable Design, click on your profile and click on `Token management` and proceed to set up the access token. Copy the token key. Post this key on Flowable Work by clicking on the left - hand side panel and entering the token in the given space.

First Run of the Travel Request Case

Now that we have published the very first and basic version of our app, let us run it and create a first case instance.

Go to the Flowable Work runtime, log in with any user and select the `New>Create>Work` app in the left-hand side menu. Choose our `Travel Request` in the dialog and confirm the creation of a new case instance.

You should now see your first case instance in the inbox and the case details. If you go to the "History" tab, you can even see our case model, showing that we already completed all the stages and the case itself is already completed.



As we did not add any necessary work to any stage, and we did not create a start and work form there nothing to do. That is why the engine started the case and immediately completed it.

Add a Start and Work Form

Add and Define a Start Form

We are going to add a start and work form to our travel request case model, so we can add and display some travel relevant data of the case.

Go back to Flowable Design and select the case plan model element (outer container) named *Travel Request*. In the attribute section on the right hand side, click the `Start form` attribute and enter *Travel Request Start Form* for the "Name" section of that attribute, a default key name is automatically generated, then select `Create`. A new form definition is created and linked as the start form for that case model.

Drag and drop a `Data entry>Text` field unto the form canvas, name its label *Travel subject* and hit `Enter`. This action creates a value binding (variable name), named `travelSubject`. As we want the requester to enter a subject, make it mandatory by clicking the `Validation` attribute on the right hand side panel and click on edit button (pencil shaped icon). Check the `Required` attribute.

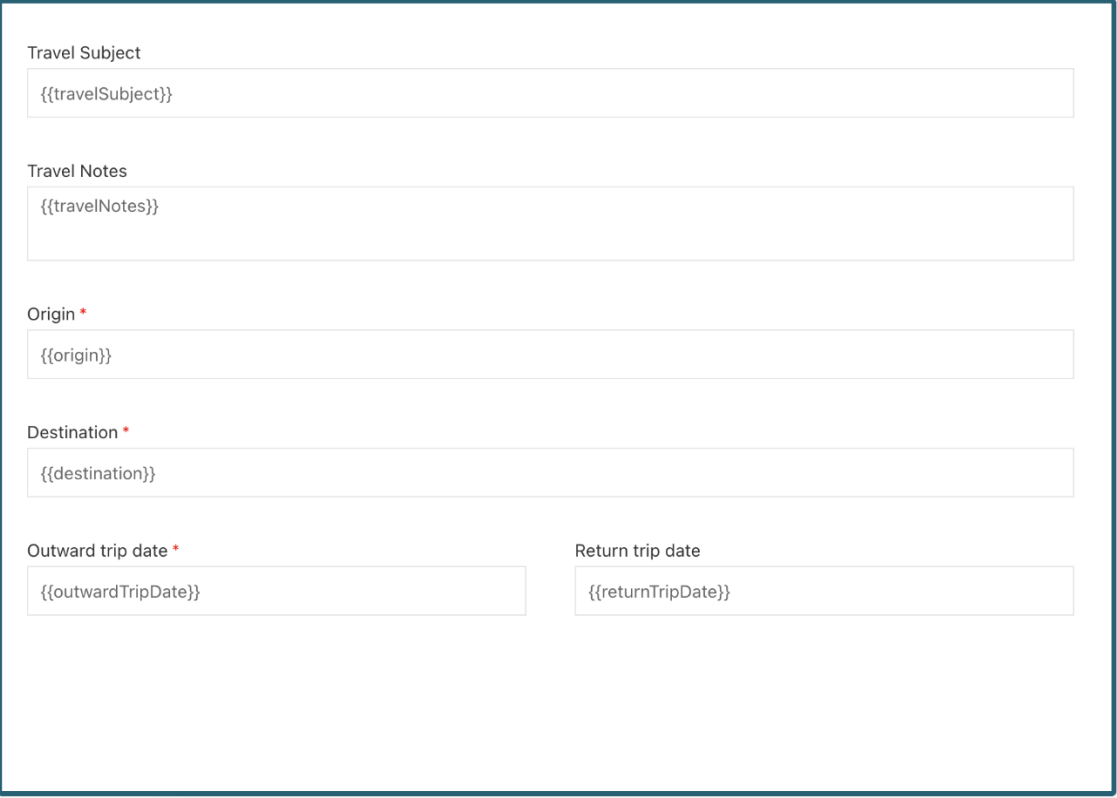
Next, we want to allow the user to enter any additional notes, requests, or whatever is needed to describe the need and details for the travel. Add a `Data entry\Multiline text` field to

the form and name it *Travel notes*. You can also add some description (General>Description) to further give hints on what the user should enter.

Let us add two more fields for origin and destination of the travel; both are Text fields and name them *Origin* and *Destination*. The fields are both required.

Next, we add two Date fields (Data entry>Date) to enter the beginning and return dates, name the fields *Outward trip date* and *Return trip date*. We make them mandatory, and we want to add a validation only to be able to enter a trip date in the future as we cannot organize a trip in the past. Select the *Outward trip date* and then the Validation>Minimum date attribute. In the attribute choose Relative to define a date relative to the current day. Let us choose 0, so we could even create a travel request for the same day. For the *Return trip date* field, we also choose Validation>Minimum date and this time, select Expression where we enter the binding (value expression) of the first date field as the return date cannot be before the outward date. Enter `{{outwardTripDate}}` into the expression field which means the minimal date entered in the return date picker cannot be before the date selected in the outward date picker, represented by that value expression.

The form should look something similar to this one:



Travel Subject

{{travelSubject}}

Travel Notes

{{travelNotes}}

Origin *

{{origin}}

Destination *

{{destination}}

Outward trip date *

{{outwardTripDate}}

Return trip date

{{returnTripDate}}

We also want to modify the Submit button, so let us define the outcomes section of the form by editing the Other>Outcome variable name attribute on the form (you may have to first click outside the form canvas to see its global attributes). Open the Outcomes dialog and enter one item to specify our submit button for the start form:

Form
(travelRequestStartForm)

Outcome variable name

Outcomes

- Create travel request - crea...

Label [?]

Value [?]

Visible [?]

Enabled [?]

Optional outcome form [?]

Navigation URL [?]

Style class [?]

☐ Ignore payload [?] ☐ Ignore validation [?] ☐ Primary [?]

By default a form is submitted with a standard button to complete it. In Flowable Work, this is the **Complete button**. One or more buttons, called **outcome buttons** can be configured to override this. Each button has a custom label and value.

This value is persisted and can be used in subsequent steps, for example in an exclusive gateway in a BPMN process model to switch to the correct branch depending on which outcome the user had selected.

Cancel Finish

Select `Finish` to save the outcomes.

Save the form and go back to the case model. We have now defined a form to enter the data to start the travel request case.

Add and Define a Work Form

The start form is used to collect data required to start a case. The work form is used to show information of a case once created. A work form can be read-only, but it might also contain fields allowing to change data associated with the case. In our example, we want to make it read-only and only allow case data changes through tasks.

Click the add new tab sign (+) and select `Create` and choose `Form` as the model type. Name the new form *Travel Request Work Form*, hit `Create` to create and open a new form.

We want to display the same data as we enter on the start form, so let us add a subform and re-use the start form directly, but make it read-only. Add a `Container\Subform` component and name it *Travel data*. We want it to display the case data directly, so no variable binding is needed. Remove the check from the `General\Store` subform data in single variable and leave it unchecked.

Select the attribute `Details\Form` reference and select `Travel Request Start Form` (the one we just created) in the "Reference" tab. Within the case work form, we do not allow the travel data to change, so uncheck the `General>Enabled` attribute which makes the whole subform non-editable (not enabled).

The work form should now look something like this:

The screenshot shows a form titled "Travel Request" within a blue-bordered editor window. In the top right corner, there is a button labeled "travelRequestStartForm" with a small icon. The form itself consists of three input fields: a single-line text field at the top, and two side-by-side single-line text fields below it. Each field has a light gray placeholder bar above it.

This is a good start so let us save the form for now and return to the case model.

As we added a start and work form to the case model, we also need to save the case, and then we can directly deploy again to test the additions.

Deploy and Run the Travel Request Again

We can deploy directly from within the editor, so click the `Cloud` icon to publish and select the `Travel Request` app.

In `Flowable Work`, select the `Work>Create new item` and choose our `Travel Request` model again. You should now see the start form, so let us enter some trip data and click on `Create travel request`.

The screenshot shows a "Create case or process" dialog box with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- Travel subject ***: A text input field containing "My first travel request".
- Travel notes**: A text area containing "Some travel notes or special requirements for the trip." Below it is a smaller line of text: "Optionally add any travel notes, special requests or whatever is helpful for approving and organizing the travel."
- Origin ***: A text input field containing "Zürich".
- Destination ***: A text input field containing "Singapore".
- Outward trip date ***: A text input field containing "2019-03-18".
- Return trip date ***: A text input field containing "2019-03-22".

At the bottom of the dialog, there are two buttons: "Create travel request" (a dark teal button) and "Cancel" (a light gray button).

We now have our first travel request containing some structured data:



Of course, the case is again directly completed as we only added a start and a work form and did not change the case model itself. That is our next move.

Add Work and Dependencies to the Stages

We are going to enhance the case model by adding dependencies between stages and add some work to the first stage.

Add Dependencies Between Stages

Although in CMMN, there might be more than one stage active at the same time, we want our stages to be sequential, one after the other. We use entry sentries for that, but first, let us define the stage display order.

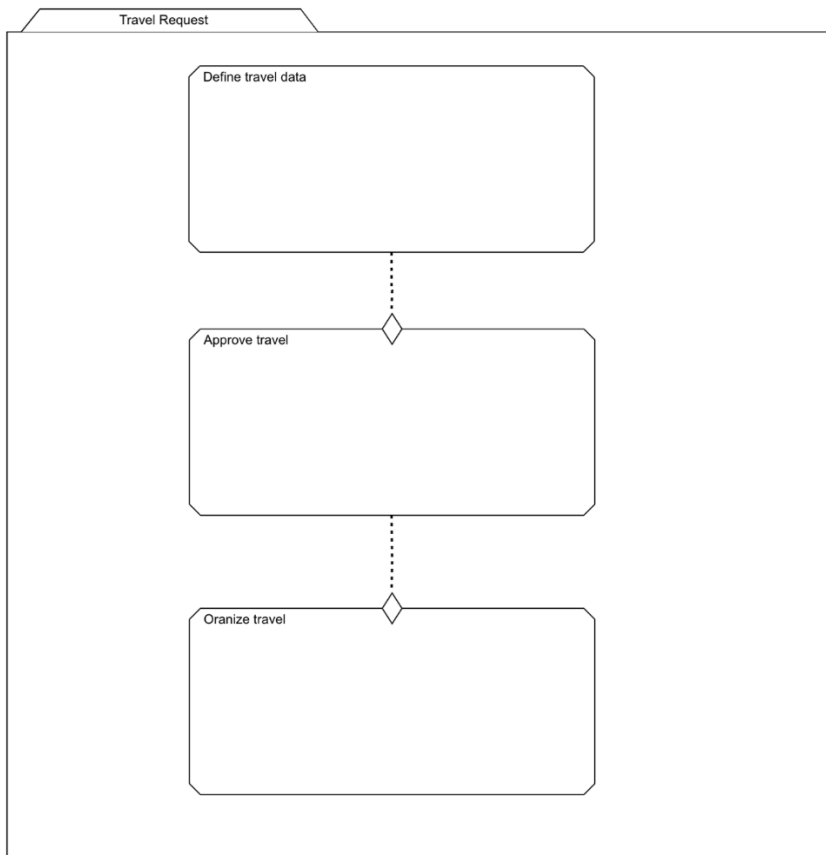
Select the first stage named *Define travel data* and set its `Details\Display order` attribute to 0. Then drag and drop the little diamond icon in the upper right corner of the stage onto the second *Approve travel* stage (drop it once the second stage becomes green).

As `complete` is the default event type, we do not need to display it, so select the connector between the stages (the dotted line) and uncheck the `Details>Display name in diagram` checkbox.

Now select the second stage named *Approve travel*, set its display order to 1 and also drag and drop the entry sentry from its upper right corner to the last stage and uncheck its display name attribute.

Lastly, select the third stage named *Organize travel* and set its display order to 2.

We have now specified the display order in which the stages are displayed at runtime and set the dependencies between them. Now only the first stage does not have an entry criterion (entry sentry), and it is immediately activated once the case is started. The second and third stages depend on completing its previous stage, so they are enabled once the case starts, but only activated if that complete event is triggered when the previous stage is completed.



Add an Optional Task to a Stage

In the first stage, we can add an optional task to modify the travel data at any time by the creator of the case (the requestor). As we have already specified the necessary data within the start form, we make it optional.

Drag and drop a `Controls\Human task` onto the first stage and name it *Modify travel data*. The assignee is specified to be the initiator of the case by default, which is fine for us in this case. Additionally, check the `Execution\Repetition` attribute, so we can start this task as many times as we want.

Also check the `Activation\Manual activation` attribute, as we offer this as an option to the user to start this task.

Now select the `Human Task - General\Form reference` attribute, click "New" and enter *Modify Travel Request Data Task Form*, hit *Create* and *Finish* and you have a new task form to model.

We can again use that start form as a subform, so drag and drop a `Container\Subform` to the form canvas and label it *Modify travel data*. Similar to the work form, remove the `General>Store subform data in a single variable` check and leave it unchecked as we directly want to modify the case data with the subform. Next select the *Travel Request Start Form* in the "Reference" tab of the `Details>Form reference` attribute.

We want to change the outcomes for this task form as we want the `Submit` (complete) button labeled `Modify travel data`. Select the `Details>Outcomes` section and enter into the outcome item *Modify travel data* for the "Label" and *modify* as the "Value".

Form
(modifyTravelData)

Outcome variable name

Outcomes

Modify travel data - modify

Label

Modify travel data

Value

modify

Visible

Enabled

Optional outcome form

Navigation URL

Style class

☐ Ignore payload

☐ Ignore validation

☐ Primary

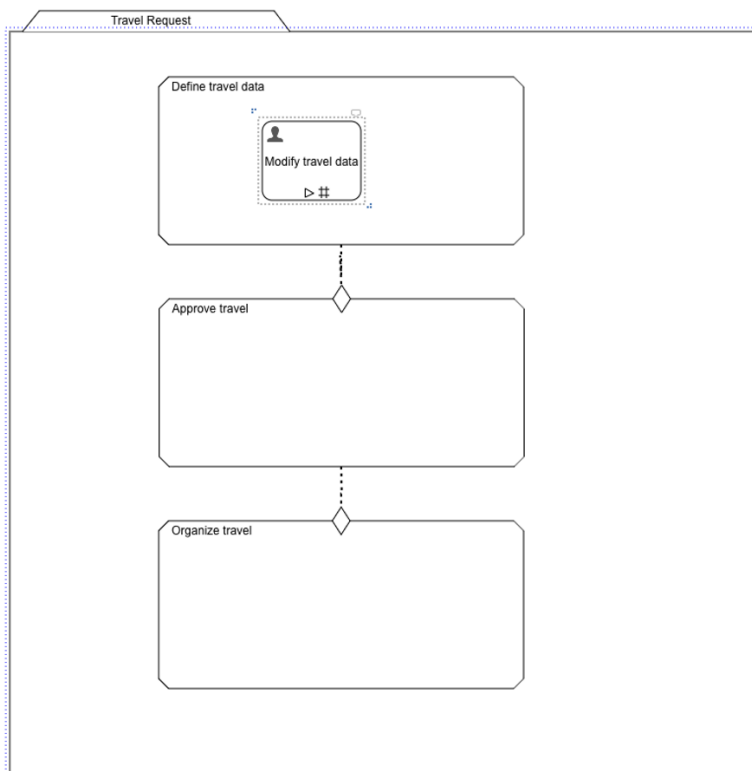
By default a form is submitted with a standard button to complete it. In Flowable Work, this is the **Complete button**. One or more buttons, called **outcome buttons** can be configured to override this. Each button has a custom label and value.

This value is persisted and can be used in subsequent steps, for example in an exclusive gateway in a BPMN process model to switch to the correct branch depending on which outcome the user had selected.

Cancel

Finish

Click `Finish` on the `Outcomes` dialog and save the form, return to the case model and save that again, it should look like this:



Deploy and Run the Travel Request Again

As before, we want to test that next step in our modeling, so publish again using the `Cloud` icon in the editor, then switching to the runtime tab in your browser and create a new case again for our `Travel Request` case.

After providing the travel details on the start form and clicking on the `Create travel request` button we see that this time the case is not completed immediately, and we see the stages rendered to reflect the current state of the case:

Case

Travel Request

Started: 1 minute ago
Started By: Shane Bowen
Assignee: Shane Bowen

Organize travel Define travel data Approve travel

Terminate Modify travel data

Open tasks Work form People Sub-items Documents Audit trail History

No tasks to show.

Also, pay attention to the buttons in the header bar: we now see `Terminate` (which is a default action to terminate the case, if needed) and the `Modify travel data` button.

Let us modify the travel data by hitting `Modify travel data`, and it creates our task to modify the case data. Select the task by clicking on it, and we have our travel form again where we can modify the basic travel data.

Task

Modify travel data

Created: less than a minute ago
Assignee: Shane Bowen
Due: No due date

Modify travel data Save

Task People Subtasks Documents History

Modify travel data

Travel subject *
New travel request

Travel notes
Some travel notes or special requirements for the trip.

Optionally add any travel notes, special requests or whatever is helpful for approving and organizing the travel.

Origin * Destination *
Zurch Singapore

Outward trip date * Return trip date *
2019-06-28 2019-07-06

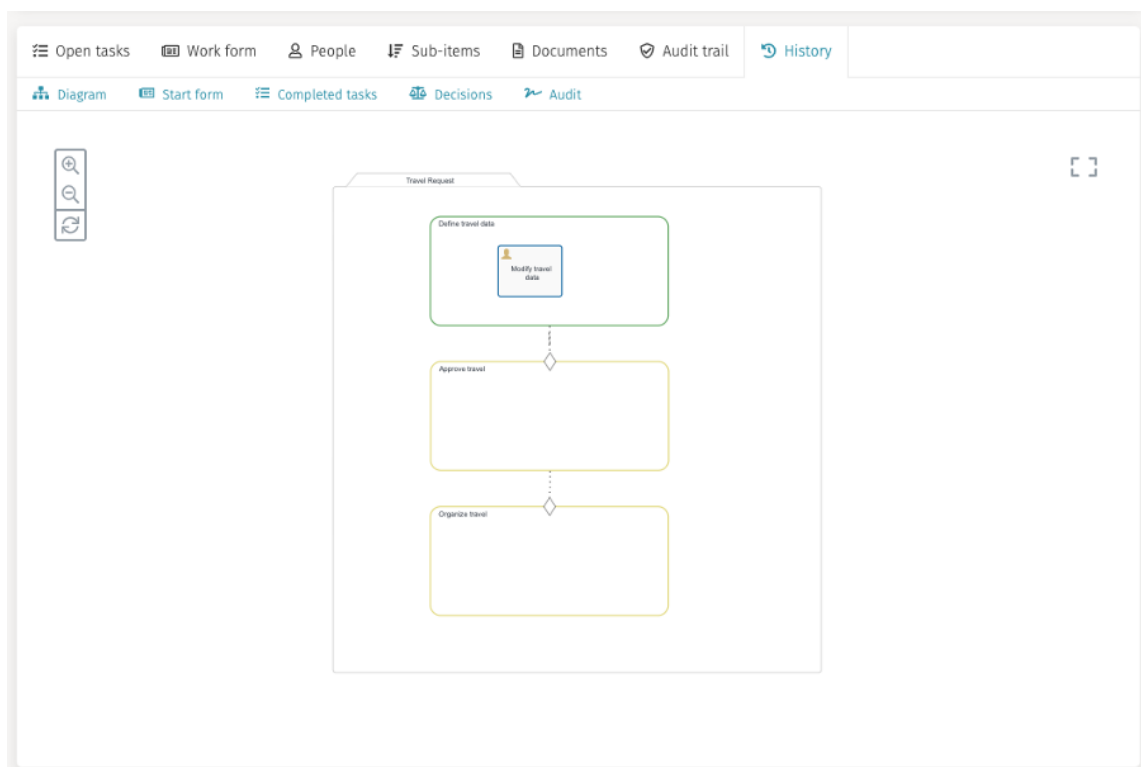
Modify some data, hit `Modify travel data` and then select "Work form" in the case where you see your changed data in a read-only mode.

If you go to the tab "Sub-items", you see your completed task and exactly what you entered in that form.

Explore the History of the Case

The "History" tab of a case shows information about the current state and the history of a case.

The first one is the diagram showing the current state of the case from its model point of view:



We see the first stage being green, meaning it is currently active, the other two are yellow, which represents an enabled stage, but not yet active as their entry sentries are not triggered (the first stage did not complete yet).

The task is blue which means the task was created and completed.

Selecting the "Start form" sub-tab shows the initially entered case data with the start form for audit purposes, to see the initial data.

In the "Completed tasks" section you find all tasks already being completed, we should see our completed modifying task there and also what data we entered if you select it.

We do not have any decisions yet, so the "Decisions" section does not show anything.

The "Audit" section has some interesting information too, especially for us as modelers; it shows all the elements of the case and in which state they are, like `AVAILABLE`, `ENABLED`, or `COMPLETED`. This tab is also a big help once the case is not behaving like we would expect as it might give us valuable hints.

Adding a User Listener to Complete the First Stage

Maybe you have already realized: how can we complete the first stage? The case does not move any further before we can complete that stage.

There are various options to complete a stage:

- The stage completes automatically if there is no more work (not even optional work) to be started or completed.

In the initial stage of this example, this is not working, as we have an optional task with manual activation and repetition, which means, the engine cannot know if we do not want to start that task again, so it never completes the stage automatically.

- Make use of auto complete.

We could mark the stage "Auto complete", but because we have a manually activated, optional task, the engine would again immediately complete the stage once we start the case. Why? Because there is no mandatory work to be done, only optional work and hence the engine would complete the stage immediately. In this case, we do not even time to start that manually activated task as the stage is already completed.

- Create a user listener to manually complete the stage once we are ready with whatever we needed to do in that stage.

This last alternative is the best choice for our example.

Create a User Listener and Exit Sentry in the First Stage

Add a `Listeners\User` event listener to the first stage and name it *File for approval*. A user listener is quite similar to a manually activated plan item in the user interface: it appears as an action button in the case header and once clicked, executes whatever you trigger with it. Optionally, it can also have a start form where you can enter additional data, once triggering that user event. We only want to finish the first stage, so we do not need a start form. However, we want to terminate the first stage, once selected.

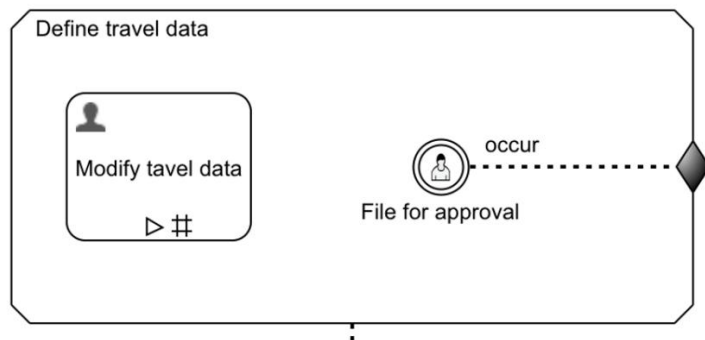
We only want to make that user event listener available, if the stage is completable and does not have unfinished work left. For this, we can specify a condition (expression) when the user listener should become available. Use `${cmmn:isStageCompletable() }` as the expression within the `Details\Available` condition attribute of the user event listener.

Add a `Sentries>Exit` criterion (exit sentry) to the stage by dropping it on its right-hand side boundary (once the boundary becomes green, you can drop it there). An exit sentry is

used to finish (terminate) a stage and optionally can have a condition associated with it, for example, an expression which needs to evaluate to `true`, to trigger that sentry.

Now we need to use the user event to trigger that exit sentry, so select the user listener again, drag and drop the connector icon (small arrow icon) from the user event listener to the exit sentry. For the connector, select `Occur` as the `Connector -General>Standard` event in the right hand side panel attribute to listen to whenever that user listener is triggered.

Now, this is how our first stage should appear:



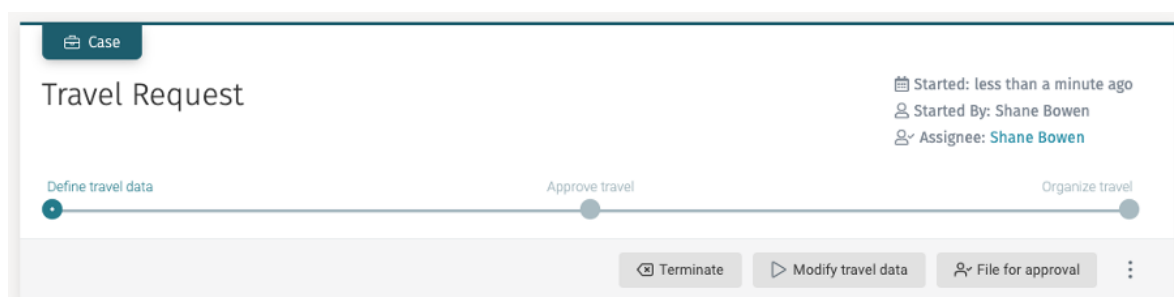
An exit sentry according to the CMMN specification terminates a stage and does not complete it. Stage termination is quite a limitation as in our case, as we want to complete the stage, not terminate it. Termination kills all existing, still active plan items if there are any. Termination is different than completing it. Given that we added the available condition on the user listener, we can only access that user listener, if the stage can be completed. We need to add that option in the future, so on an exit sentry, you can choose whether to terminate or complete the stage. For now, it terminates the stage, which means we need to change the connector connecting the first and second stage and listen to the exit event, not the complete event.

Select the connector between the first and second stage and change the `Connector-General>Standard` event from `Complete` to `Exit`.

Deploy and Run the Travel Request Again

As before, save the case model, hit the publish button again, switch to the Flowable Work runtime view, and start a new travel request case again.

Right after creation, as we did not start that optional modifying task yet, you now see the `File for approval` action button in the header of the case:



If you start the `Modify travel data` task, the `File for approval` action is hidden as the stage is then not completable. Once you complete the task, it becomes available again. Select the button to complete the first stage, and you see that the case is completed automatically as we do not have any work to perform in the other stages.

Add a Required Task to the Stage

Let us assume that we want the requester to start the travel case with not much information and let them add additional data later. That is what we did by not having much detail about the travel request in the start form. However, we do need additional information before we can approve the request, so let us add another user task to define that additional data. This new task is a required task.

Add Travel Details Task

Drag and drop another `Controls\Human` task to the first stage and label it *Add travel details*. Ensure that the task is required and non-repeating (these attributes are found in the `Execution` section). Also `Activation\Manual activation` must be unchecked so the task is started automatically by the case engine once the stage is activated.

Next, select `Form reference`, enter *Add Travel Details Task Form*, hit *Create* and *Finish* to create a new task form.

In the new form, drag and drop a `Selection>Radio buttons` component to the form and label it *Transportation*. By default, it has a static data source, which means we define the selection items directly within the form. Click on `Data source>Items` to define our options for travel transportation.

Add *Public transportation*, *Car*, *Flight* and *Flight and rental car* to the options list (remembering to add a corresponding value):

Radio Buttons
Transportation (radiobuttons1)

Data source ⓘ
Static

Items ⓘ

Text ⓘ	Value ⓘ
Public Transportation	publicTransportation
Car	car
Flight	flight

The datasource of the configures where the data options available to the user originate from. The source can be statically defined or dynamically populated from a third party service.

Cancel Finish

We can either arrange the radio buttons horizontally or vertically, let us choose `Horizontal` in the `Details>Orientation` attribute, to align our options horizontally.

Drag and drop a new `Data entry>Text` input field and name it *Travelcards* where we can enter the travelcards of the requestor to optimize the travel. However, we only want to show it, if we choose public transportation, so let us use the `General>Ignored` option and enter the following expression: `{{transportation != 'publicTransportation'}}`.

We also want to capture if we need a hotel, so let us drag and drop another `Selection>Radio buttons` component, give it the label, *Accommodation*, change its orientation to horizontal too and add options like *Hotel with breakfast*, *Hotel* and *No accommodation needed* to the list of options. Enter `{{transportation == 'publicTransportation'}}` in the `General>Ignored` attribute so the component is shown only if we do not choose to use public transportation.

Let us make both selections mandatory, either click on the red star or check the `Validation>Required` attribute for both radio groups.

If the outward and return date is on the same date, then we do not need any accommodations so let us use an expression for the default value of the hotel selection by setting the attribute `General>Default value` to the following expression:

```
{{((outwardTripDate == returnTripDate) && 'no') || ((outwardTripDate != returnTripDate) && 'hotel')}}
```

What does the expression do? The first part is true if both dates are identical and sets the default value to `no`; the second part is the opposite and sets the default value to `hotel`.

Let us also add a `Data entry\Decimal` field, name it *Estimated costs*, make it a required field and add a `Data entry\Multiline text` component and name it *Special requirements*.

Optionally add descriptions to the input fields to give some hints on what to enter for the user. The form should now look like this:

Transportation

☐ Public transportation
 ☐ Flight
 ☐ Flight and rental car

Please select the type of transportation for the travel.

Travelcards

if you have one or more travelcards for public transportation, please add them here.

Accommodation

☐ Hotel with breakfast
 ☐ Hotel
 ☐ No accommodation needed

Please select whether you need a hotel with or without breakfast or you don't need accommodation.

Estimated costs

Please add a rough estimation of the travel costs as good as you know it.

Special requirements

Add any special requirements to the travel here, if necessary.

Now save the form, go back to the case model, and save it as well.

Add New Details to the Case Work Form

As we added new detail data to the travel request, we want to include that too in the case overview (work) form.

Go back to the Travel Request Work Form tab in the editor.

We currently have one subform there, let us add another one by dragging and dropping a new `Container>Subform`, name it *Travel details* and uncheck the `General>Store` subform data in single variable checkbox. Then select `Details>Form` reference, choose "Reference" and select `Add Travel Details Task Form` as the subform reference.

We only want to show that detail data once we completed the details task and added that information, so add an expression (condition) on when we want that subform to render. Select the "lightning" icon on the `General\Ignored` attribute which shows an expression field where we simply enter `{{!transportation}}` as the expression for the ignored attribute. Setting this property means that the subform will only be visible once we enter the transportation option, which is a mandatory field in the details task. When we complete that task, the overview form shows the detail subform.

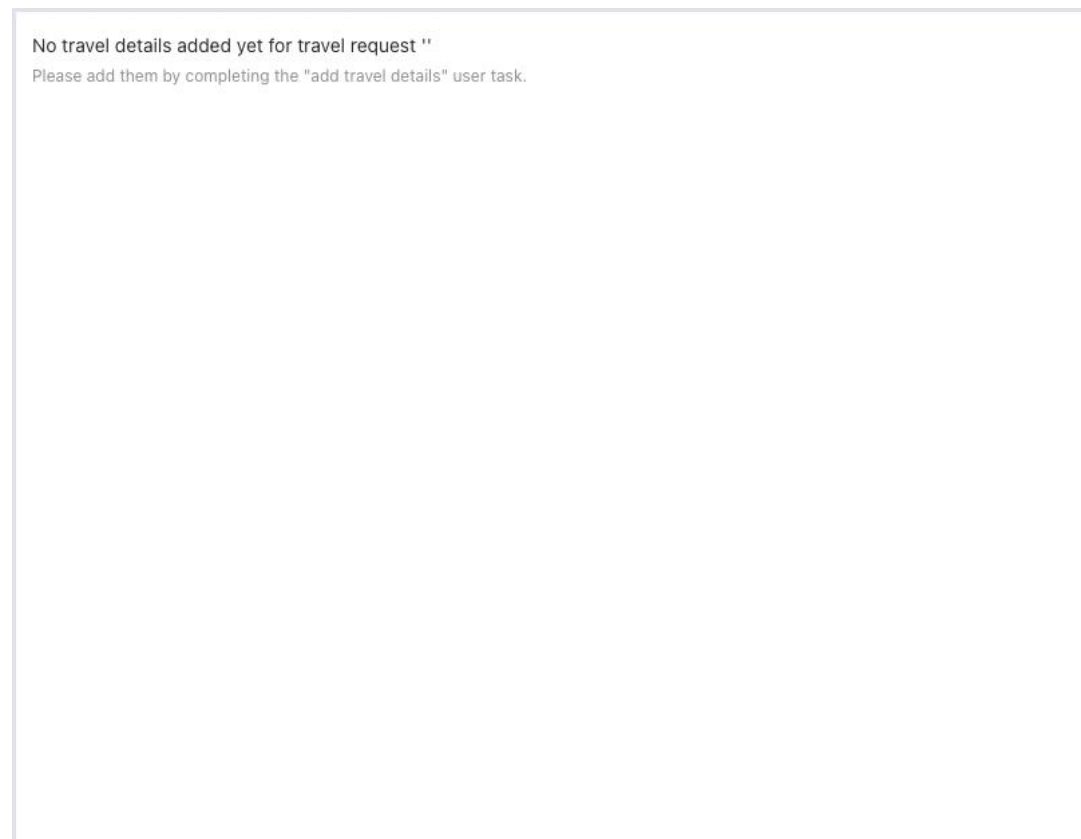
NOTE

We could have also used the `General\Visible` attribute to hide the subform. In that case the hidden subform would still be validated. If we tried to modify the travel request before adding details the required fields would fail validation and prevent us from completing it.

Also make the subform read-only like the first one by unchecking the `Enabled` checkbox on the subform component.

We do not show that subform until the travel details are provided. Add a `Display>Text display` component below the second subform and add a message about why the subform is empty.

Add something like this to the content:



No travel details added yet for travel request "

Please add them by completing the "add travel details" user task.

As you can see, we can also use expressions as part of such content. All case data variables are accessible.

Now finally, make that display component visible only if the details are not yet provided. Select the "lightning" icon in the `Visible` attribute and set its condition to `{{!transportation}}`. The effect of this expression is to show the component only if the transportation details are not filled out.

The case work form should now look like this:

Travel Data

travelRequestStartForm

Travel Details

addTravelDetailsTaskForm

No travel details added yet for travel request.

Add the Details Option to the Modifying Task

In addition to the case work form, we also want to make sure the user can change those details again once entered already using the optional modifying task.

Go back to the case model, select the `Modify travel data` task and either click the small link icon below the task or the `Form reference` attribute link named `Modify Travel Request Data Task Form`. Either option takes you to the form we want to modify.

We have one subform there at the moment, so let us add a second one with the travel details like we did in the `Travel Request Work Form`. This time, let us do it by copy and pasting the existing one from the case work form. So switch back to that work form, select the `Travel details` subform and either hit *Command-C* or the copy icon in the action bar of the editor. Then go back to the `Modify Travel Request Data Task Form` and either hit *Command-V* or the paste icon in the action bar. This inserts that subform with the same attributes as defined in the case work form.

We only have to make the details enabled (editable) for the task form, so check the `Enabled` attribute checkbox for the subform to allow editing at runtime.

We are ready for another test-drive, so let us save everything, and deploy the app again.

Deploy and Run the Travel Request Again

Now deploy the app again, switch to the Flowable work view and start yet another, new travel request.

Unlike the optional, manually activated task, the new task is created by the engine automatically, as it is a required one and shows up in the open task list of the case.

You can now play with the outwards and return date if they are the same, the "no accommodation" option is pre-selected, otherwise the "hotel" one. Also if you select the public transportation, see that the extra field shows up to enter the travelcards.

The screenshot shows a web interface for adding travel details to a task. At the top, there's a 'Task' tab and a title 'Add travel details'. On the right, it shows 'Created: less than a minute ago', 'Assignee: Shane Bowen', and 'Due: No due date'. Below this are 'Complete' and 'Save' buttons. A navigation bar includes 'Task', 'People', 'Subtasks', 'Documents', and 'History'. The form itself has several sections: 'Transportation' with radio buttons for 'Public transportation' (selected), 'Car', 'Flight', and 'Flight and rental car'; 'Travelcards' with a text input containing 'GA'; 'Accommodation' with radio buttons for 'Hotel with breakfast', 'Hotel', and 'No accommodation needed' (selected); 'Estimated costs' with a text input containing '0'; and 'Special requirements' with a large text area. Each section has a small instruction below it.

Task

Add travel details

Created: less than a minute ago
Assignee: Shane Bowen
Due: No due date

Complete Save

Task People Subtasks Documents History

Transportation *
☒ Public transportation ☐ Car ☐ Flight ☐ Flight and rental car
Please select the type of transportation for the travel.

Travelcards
GA
If you have one or more travelcards for public transportation please add them here.

Accommodation *
☐ Hotel with breakfast ☐ Hotel ☒ No accommodation needed
Please select whether you need a hotel with or without breakfast or you don't need accommodations.

Estimated costs *
0
Please add a rough estimation of the travel costs as good as you know it.

Special requirements

Add any special requirements to the travel here if necessary.

Also pay attention to our user listener to complete the stage: it does not show up until we have completed that mandatory task, and there are no more active tasks.

Moreover, when filling out that task form, you can also enter partial data and save the task without completing it. This saves the data, but you can come back to that task whenever you want and eventually complete it. Saving a task form does save that data in a temporary scope, so no data is yet saved to the case; this only happens once we complete the task. As there might be sentries or conditions waiting for some case data to be there or have certain values, we want to make sure they are only made available on the case level once we really complete the task and not when we temporarily save that task data.

If all the details are not provided, this is how the case overview (work form) should now look like:

Case

Travel Request

Started: less than a minute ago

Started By: Shane Bowen

Assignee: [Shane Bowen](#)

Define travel data

Approve travel

Organize travel

Save

Terminate

Modify travel data

Open tasks

Work form

People

Sub-items

Documents

Audit trail

History

Travel data

Travel subject *

Workshop in Singapore

Travel notes

Optionally add any travel notes, special requests or whatever is helpful for approving and organizing the travel.

Origin *

Zurich

Destination *

Singapore

Outward trip date *

2019-06-21

Return trip date *

2019-07-06

No travel details added yet for travel request 'Workshop in Singapore'.

Please add them by completing the "add travel details" user task.

Once you provide all the details, the overview looks like this:

Case

Travel Request

Started: 1 minute ago

Started By: Shane Bowen

Define travel data

Approve travel

Oranize travel

Save

Terminate

Modify tavel data

File for approval

Open tasks

Work form

Sub-items

Documents

History

Travel data

Travel subject *

Workshop in Singapore

Travel notes

Some travel notes or special requirements for the trip.

Optionally add any travel notes, special requests or whatever is helpful for approving and organizing the travel.

Origin *

Zürich

Destination *

Singapore

Outward trip date *

2019-03-18

Return trip date *

2019-03-22

Travel details

Transportation *

Public transportation

Car

Flight

Flight and rental car

Please select the type of transportation for the travel.

Accommodation *

Hotel with breakfast

Hotel

No accommodation needed

Please select whether you need a hotel with or without breakfast or you don't need accommodation.

Estimated costs *

3000

Please add a rough estimation of the travel costs as good as you know it.

Special requirements

So this is how the modify travel data task looks once all the travel details are entered:

Task

Modify travel data

Travel Request

Created: less than a minute ago

Assignee: Shane Bowen

Due: No due date

Modify case data

Save

Task

People

Subtasks

Documents

Modify travel data

Travel subject *

Workshop in Singapore

Travel notes

Some travel notes or special requirements for the trip.

Optionally add any travel notes, special requests or whatever is helpful for approving and organizing the travel.

Origin *

Zürich

Destination *

Singapore

Outward trip date *

2019-03-18

Return trip date *

2019-03-22

Travel details

Transportation *

☐ Public transportation
☐ Car
☒ Flight
☐ Flight and rental car

Please select the type of transportation for the travel.

Accommodation *

☒ Hotel with breakfast
☐ Hotel
☐ No accommodation needed

Please select whether you need a hotel with or without breakfast or you don't need accommodation.

Estimated costs *

3000

Please add a rough estimation of the travel costs as good as you know it.

Special requirements

Add and Define the Simple Approval Process

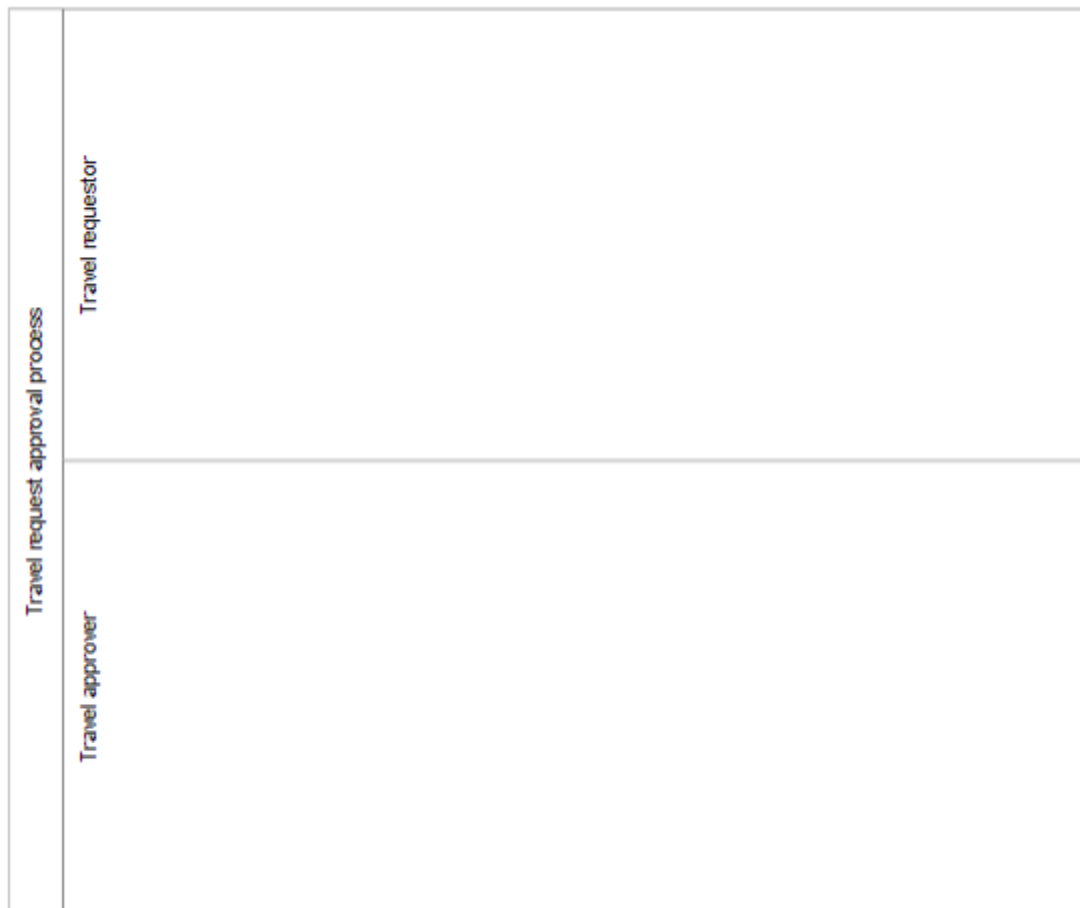
We are going to add a simple approval process to the travel request where the requestor has to choose a single approver with an approval task and feedback loop to eventually approve or decline the travel.

Later, we extend that approval process and make it a bit more sophisticated by using a DMN based decision table to automatically decide whether we need an approval and if yes, if we need one or two approvers.

Add the Approval Process Task to the Second Stage

Go back to the case model, drag and drop a `Controls\Process` task to the second stage named `Approve travel` and label it *Travel request approval process*. Select the attribute `Process reference` and enter *Travel Request Approval Process* in the name field, hit *Create* and then *Finish* to create a new process model for the approval process.

There is already a start event, so we could go ahead and directly start defining that process, however, it is a good practice to always use a pool and lanes to define the roles involved in a process. So delete that start event and then drag and drop a `Swimlanes>Pool/Lane` element onto the process canvas. We can name that pool the same as the process *Travel request approval process*. By default, one lane is defined, so double-click its label and name it *Travel requestor*. Add another lane by dragging and dropping a `Swimlanes>Pool/Lane` element again below the existing lane (drop it once you see that green box below the existing lane) and change its name to *Travel approver*.



Add a User Task to Select the Approver

Drag and drop a `Start events>Start event` to the requestor lane where we want to start the process. Next, click the small user task icon on the left of the start event to add a User task directly. Name the user task, *Select approver for travel request*. Now click the `link` icon below the user task or select the `Form reference` attribute of the user task, and enter *Select Approver Task Form* for the name, hit *Create* and then *Finish* to create a new task form.

Now drag a new `Display>Text display` component onto the form and add some text to describe the task:

Please select a person to review and approve or decline your travel request.
This might be your superior or project manager, depending on the reason for the travel.

Now add a selection for the person to approve our travel request by adding a `Selection>Person` component to the form and changes its name to *Select approver*. This time the default variable name is not a great fit, so on the right hand side panel, click on `Value` and set it `{{approverId}}` as it contains the ID of the selected person.

We are using Flowable's Identity Management (IDM) subsystem to find the appropriate employees. A default URL is already configured and we can adapt this to our needs:

```
{{endpoints.idm}}/users?displayNameLikeIgnoreCase=%25{{$searchText}}%25&size=100&memberOfGroup=employee
```

We use an expression for the IDM REST endpoint, `{{endpoints.idm}}`. The value of the expression is supplied automatically by the system. Next, we search for users using a search text and ignore the case of the string and use autocomplete matching. In addition, we only want members of the group `employee` and to limit the number of returned employees to 100. That is what this query URL actually means.

Finally, make that selection mandatory by either clicking the red star or checking the `Validation>Required` attribute checkbox.

We can optionally add outcomes to define the name of the complete task button. If desired, click the `Details>Outcomes` on the form canvas and label it *Set approver*.

The form now looks like the following:

Now save the form and let us go back to the process model.

Add Approval User Task

Select the user task again and click the small user task icon again to add another task. Name the new task, *Approve travel request* and move it into the *Travel approver* lane. By default, a user task is assigned to the initiator of the case, but for our approval task, we need it to be assigned to the selected approver from the previous task, so change the attribute `Assignment>Assignee` to `${approverId}` which resolves to the selected approving users ID at runtime and assigns that user task to them.

Next, click on the `Form` reference attribute on the right hand side panel and enter *Approve Travel Request Task Form*, hit *Create* and *Finish* to create a new task form.

To the newly created form, add a `Display>Text` display component to the form and add some description for this task:

Review and approve or decline the travel request

Please review the travel request data and make a decision whether to approve or decline the travel request. If you want to decline the travel, make sure to leave a comment why you declined it, otherwise you will not be able to use the "Decline travel" button.

Now add some review comments by adding a `Data entry>Multiline text` component to the form and name it *Review comments*.

We also want to include all travel request case data on that user task, so the approver does not have to switch back to the case overview to see it but directly has that data on the task form.

We can use a subform again, so drag and drop a `Container>Subform` component to the task form, but do not enter a name, leave the label blank. Select `Details>Form reference`, choose "Reference" and select the `Travel Request Work Form`. Now here is the tricky part; we are now designing the form for a task which is part of the approval process, not the case directly like the previous tasks. By default, the data of a task form is mapped to its parent, which for the first tasks in our example was the case directly, but this one now is part of a process, so its parent is the process and its root is the case. This means we need to scope that subform to "root" to get to the case data we want to display in that subform. So set `{{root}}` as the value within the field below the checkbox `General>Store subform data in single variable` and naturally leave it checked. By specifying this value scopes the data for the subform to the case directly and not the process (which would be the default).

Also, remove the `Enabled` check from the right hand side panel `Subform>Rendering` as we do not want to let the approver edit the travel data directly.

To make that subform based data stand out a little more, add a `Display>Horizontal line` just between the comment field and the subform.

To let the approver make a decision, we can do it in several ways:

- add a checkbox named *approve* which results in a Boolean variable having either value `true` or `false` for the approval.
- add a radio button group with two options: *approve* and *decline* which results in a text variable either having a value of `approved` or `declined`.
- use outcomes to let the approver decide and submit (complete) the task directly with a single action.

Let us go with option three for now. Click outside the form canvas and select `Form - General>Outcomes`. This time we add two options, *Approve travel* and *Decline travel*. As we want to make sure the approver adds a comment when declining a travel request, let us put an expression with that outcome and set it to `{{!!reviewComments}}`. Pay attention to the double negation before the variable name, which is necessary as you might enter some comments and then remove them again, which makes the variable available once again, but empty. Using the double negation, the expression is only `true` if the variable is not only there but also contains text.

Review and approve or decline the travel request

Please review the travel request data and make a decision whether to approve or decline the travel request. If you want to decline the travel, make sure to leave a comment why you declined it, otherwise you will not be able to use the "Decline travel" button.

The approval task form now looks like this:

Review and approve or decline the travel request

Review comments

{{reviewComments}}

Label

travelRequestWorkForm

Save the form, go back to the process, and save it as well.

Use Approval Outcome in the Case Model

Select the case model as we need to adapt it to the result of the approval process.

It is a very good practice to use scoped process data when using a process task (here, `Travel request approval process`) within a case to isolate its data from the case as long as the process is running. Of course, this depends on the type of the process and basically whether you want the data being entered or changed during the process made visible on the case level immediately or not. For this we can use the `Variable Mapping>In` and `Variable Mapping>Out` mappings of a process task to store data from the case within the process when it is started and the opposite; to get data from the process back to the case once finished.

In our simple case, we do not need data from the case to be copied to the process as we only show a read-only form of the case data which is bound directly to root, so no need to copy that data in. However, we want the result of the approval process taken back to the case level, so select the `Details>Out` attribute and define some output mappings for the process:

Process Task ✕
 Travel request approval process (cmmnTask_7)

Source Variable	Source Expression	Target Variable	✕	⋮
approveID		approverID	✕	⋮
reviewComments		reviewComment:	✕	⋮
form_approveTravel		approval	✕	⋮

⊕

i

Map variables from the child process instance created by the process task into the parent case instance, when the child process instances ends.

Remember, we used outcomes for the approval. They are stored in a variable named `form_{formKey}_outcome` by default, so we map that one back to a variable named `approval` on the case level.

As a next step, we need the organize travel stage. If the travel request was approved, add a condition to the entry sentry of that stage.

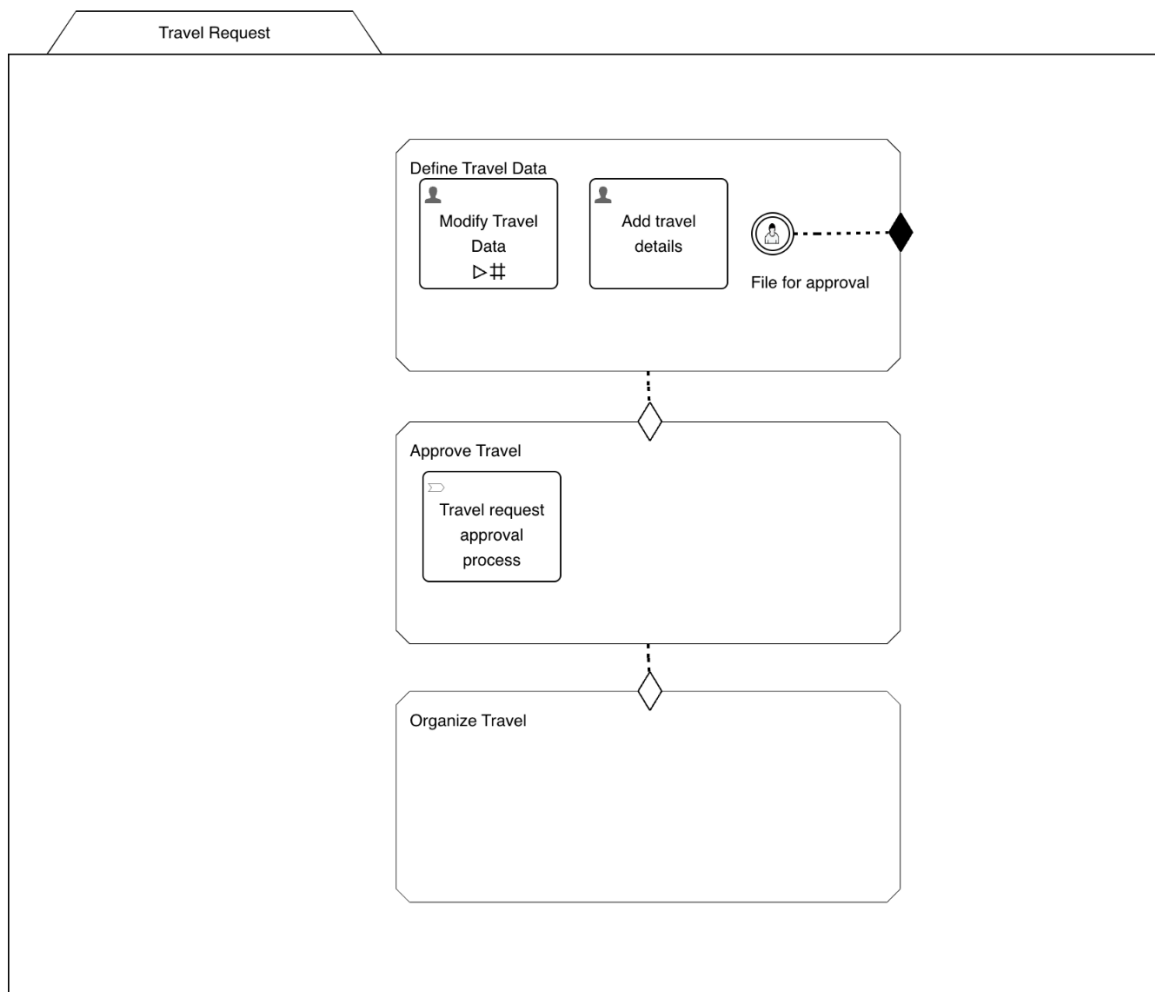
Select the entry criterion (entry sentry) of the last stage and click the attribute `Details>Condition` and set it to:

```
${vars:getOrDefault('approval', null) == 'approved'}
```

As this condition is evaluated at the very beginning when created, we need to use a special expression function if the variable `approval` is yet available. We can use the variable function, `vars` to either return the value of the variable named `approval` or if the variable does not exist, return `null` instead. Given the variable value then check its content to be `approved` and only in that case, start the third stage.

Also, let us set the `Case plan model` to `autocomplete` by checking the `Details>Auto complete` attribute on the main case plan model. This automatically ends the case if it was not approved. Otherwise, the case engine would wait for that entry condition of the last stage to eventually become `true` and in our example (at least for now), this is never the case.

Our case model should now look like this:



Add Approval Result to Case Work Form

As we have added a simple approval process, let us add the approval result to the case overview form.

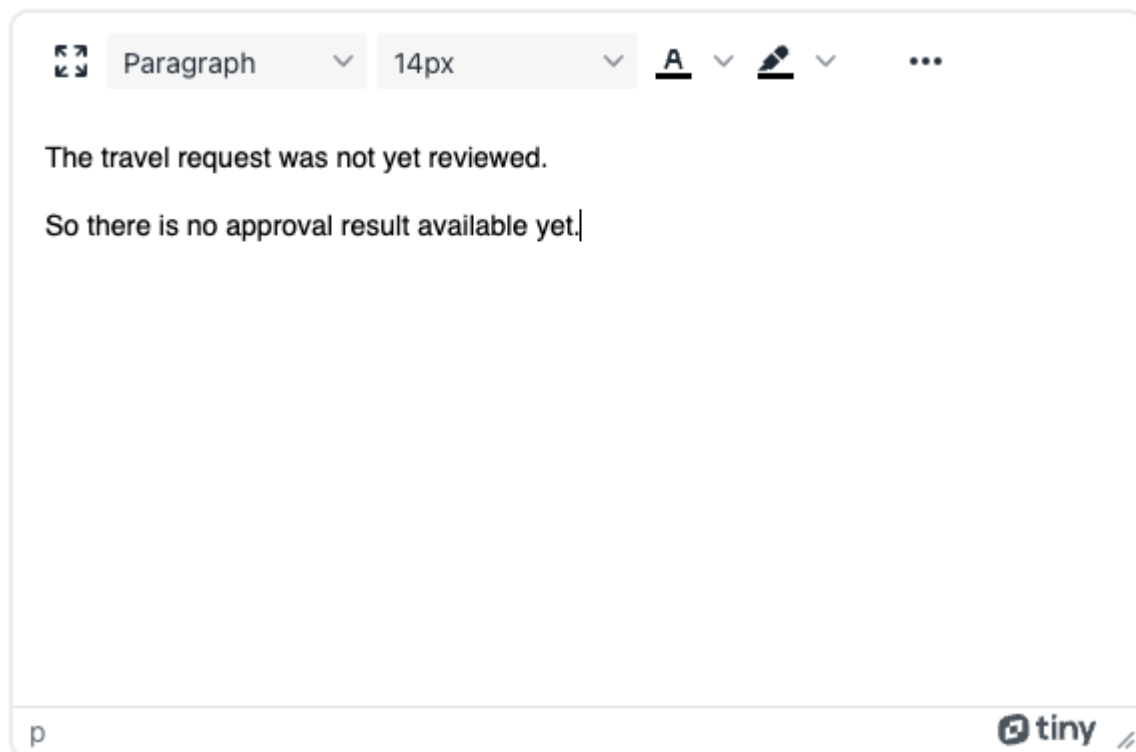
On the main case model click on the Work form named Travel Request Work Form to open the case overview form (or navigate there if it is already open).

As the form contains quite some data, let us make both subforms collapsible by checking the Subform - Form>Collapsible attribute. Let us even check the Subform - Form>Collapsed attribute on the second details subform, so by default, the first one is opened, and the details are collapsed.

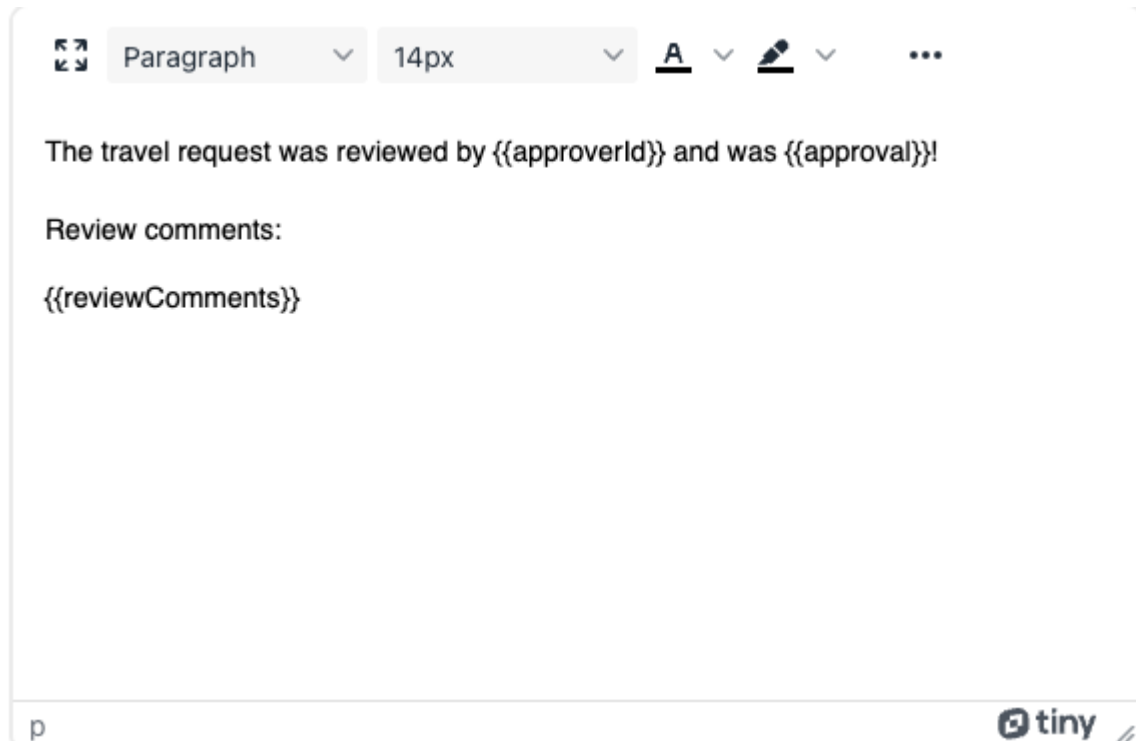
Next, add two Display>Text display components at the top of the form with the three options for the outcome of the travel case. Note that the second and third use the same text require modified by variable content.

- Option one: the travel case was not yet reviewed and approved or declined.
- Option two: the case was reviewed and approved.
- Option three: the case was reviewed and declined.

Here is the content for the first one:



And here for the second option:



As we use the approval variable in the text, we can use one text display to render the result of the review.

Now we want to make sure only one of them is rendered, so we set the `General\Ignored` attribute of the first text area to `{{approval}}` which ignores (does not render) the first text area if we have the variable `approval` already set. For the second text area, we do the opposite and set `General\Ignored` to `{{!approval}}`.

Now save everything and execute another test round.

Deploy and Run the Travel Request Again

You know the drill by now, so publish the travel request app again, switch to the runtime view and create a new case of our adapted travel request.

At least complete the details task then hit `File for approval` and the `Select approver for travel request` task in the cases open task list. Select the task and choose an approver (for simplicity, you can even select yourself so, you do not have to log out and log in again). Then play with the comment field on the approval task: you can only choose to decline if you entered a comment. As it is optional for the approval, that outcome action button is always enabled.

In all cases, the travel request case is completed after the decision as the last stage does not yet contain any work items; however, declining it does not activate that last stage at all.

Also, pay attention to the updated case overview (work) form. Initially, the form contains a comment that the travel request is not yet reviewed. Once the request is reviewed, the comments are the result of the approval step.

Add Travel Organization Tasks

We finally add some travel planning tasks for the travel agent to organize the travel once the request was approved.

Add a Travel Organization Task to Case

Go back to the case model and add a new `Controls>Human` task to the last stage and label it *Organize travel*. As in the past, select the attribute `Form` reference, enter the name *Organize Travel Task Form*. Now hit *Create* and *Finish* to create a new task form.

Let us add a new `Display>Text` display component to the form to describe the task in more details:

Please organize and book the travel according to the following information.

Also add any information for the travel requestor like the hotel and the flight information. If available, also upload any document (e.g., vouchers etc). You can always save your data with "Save" and once done, hit "Done with organization" which will then complete the task.

Add a new `Data entry>Multiline text` component to the form and name it *Travel details*. Make the new field required by selecting then `Validation>Required` checkbox. Here we can collect all the information and details about the travel as it gets organized.

As we also want to be able to upload any travel documents, add a new `Data entry\Attachment` component to the form and name it *Travel documents*. Choose `Yes` in the attribute `Attachment>Allow file deletions` and `Thumbnail` for the `Attachment>Preview type` property. Setting these two attributes allows us to attach any document to the task (and thus the case) and preview it using a thumbnail.

At the end of the task form, add the case overview form as a subform as we did within the approval task. You can copy and paste the `Display>Horizontal` line and `Container>Subform` from the Approve Travel Request Task Form or add and configure them from scratch.

The form should then look like this:

Please organize and book the travel according to the following information.

Travel details *

{{travelDetails}}

Travel documents

Drag and drop new files or [click](#) to select from file system...

Click here and paste from Clipboard

Label

Add Travel Agent Selection to the Start Form and Use it as the Assignee

As we need to know the person organizing the travel, we add that selection to the start form of the case and then use it as the assignee for organizing the travel task.

Go back to the case and select the start form by clicking the `General>Start form` link of the case plan model (top container).

The easiest way is to go back to the `Select Approver Task Form` and copy the `Select approver` user selection component and then paste it into the start form. Change the label from *Select approver* to *Select travel agent* and the `General\Variable` binding from `{{approverId}}` to `{{travelAgentId}}`.

Save the form and then go back to the case model and set the `Assignment>Assignee` attribute of the recently added `Organize travel` task to `${travelAgentId}` to assign that user task to the selected travel agent.

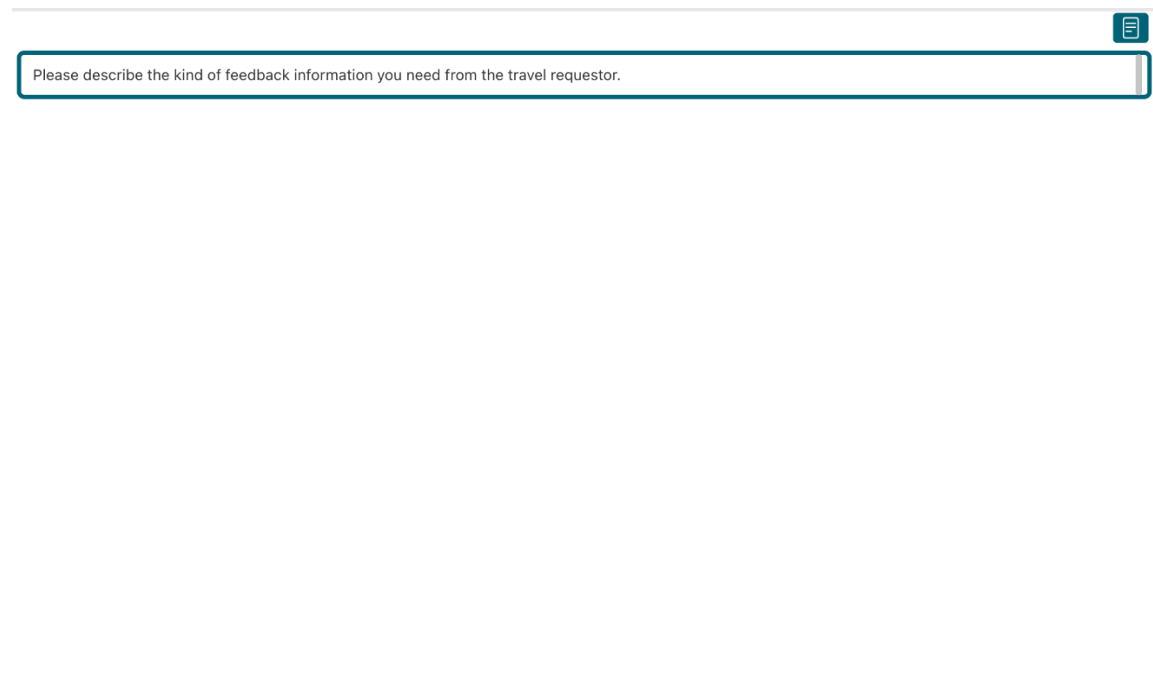
Add an Optional Feedback Task

Often one needs feedback from the requestor to book or organize parts of the travel like if there are several options for flights, you want the requestor to choose the preferred one and so on. For this use case, add a feedback task for the travel agent to get information from the requestor.

Add a new `Controls\Human` task to the last stage and name it *Request feedback*. As the task is optional and might be used more than once, we check the `Details>Repetition` attribute as well as the `Activation>Manual` activation checkbox.

As the travel agent needs to describe the kind of feedback needed, add a start form for that manually activated task. Select `Human Task - Control>Start form` and name it *Request Feedback Task Start Form*, then hit *Create* and *Finish* to create that start form.

As always it is a good practice to describe what you need to do in the form, so add a `Display>Text display` and set its content to:



The screenshot shows a form design interface. At the top right, there is a small icon of a document with a list. Below it, a text display component is shown with a blue border and a light blue background. The text inside the component reads: "Please describe the kind of feedback information you need from the travel requestor." The component is part of a larger form structure, with a vertical line on the right side indicating the form's edge.

Add a new `Data entry>Multiline text` component to the form and name it *Feedback request* and make it a required input by clicking the `Validation>Required` attribute.

Save the form and go back to the case model, select that *Request feedback* task again and go to the `Human Task - General>Form reference` property, and enter *Request Feedback Task Form*, hit *Create* and *Finish* to create the work form for that task.

As usual, first, we add a new `Display>Text display` component and add content like this:

The travel agent needs some information to continue organizing your travel.

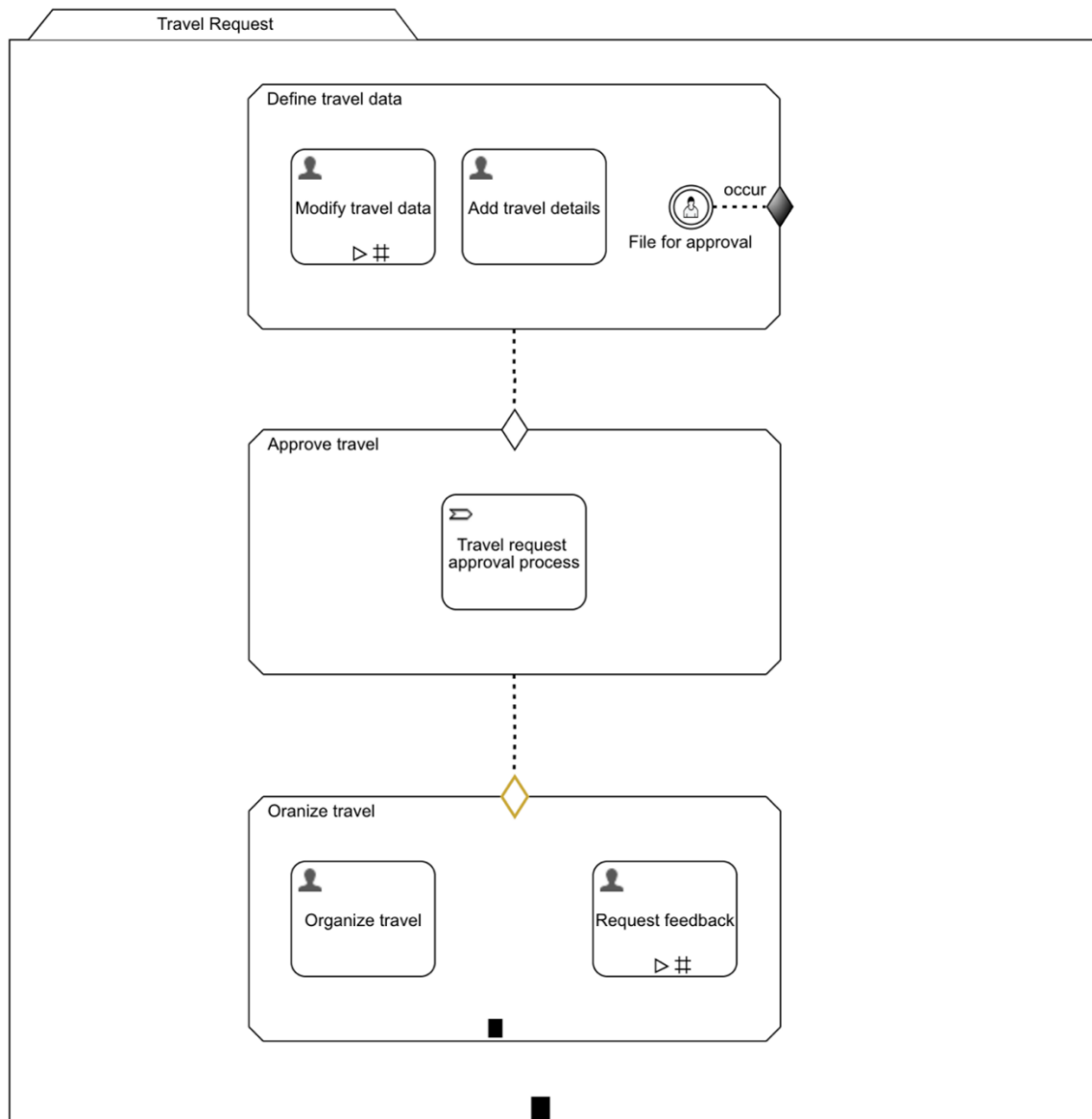
Please provide that information according to the request.

Below the text, add a new `Data entry\Multiline text` input field and name it *Feedback* and make the field required by clicking the `Validation\Required` box.

As we have now added some work to that last stage, let us think about how we want to complete it. We could use the same manual mechanism as we did with the first one, where we added a user event listener terminating the stage, which is then manually triggered. As we have optional work with the request feedback task, we cannot leave it as we did with the second stage where we only have the approval process and once finished, the case engine automatically completes the stage as there is nothing left to do. For the third one, let us use the autocomplete mode by checking the `Stage - Details>Auto complete` checkbox in the stage properties. Why does it make sense for this stage and not the first one?

Let us analyze it: we have the required `Organize travel` task, and as long as it is not completed yet, we have the option to ask for feedback, and if done, there is a second task active, waiting for the feedback to return. Now, once we complete the `Organize travel` task, there is nothing more to organize, and hence no feedback is required, and that is the perfect moment for the engine to complete the stage, and that is why we can use autocompletion here.

The case model now looks like this:



Save the form and go back to the case model and save it as well.

Deploy and Run the Travel Request Again

You can publish the changes again and run the travel request case up to the organizing stage and play with the request feedback task and its answer.

Make the Approval Process Multi-instance

Create DMN Based Multi-instance Approval Process

We are going to extend the approval process with a multi-instance activity based on the number of approvers we need. That number is calculated using a DMN decision table according to the specified travel data.

Add a New Main Approval Process

We currently have a straightforward, fixed approval process, and we are going to make it a bit more sophisticated. To get started, go back to the case model and select the `Travel request approval process task` in the `Approval travel` stage. Select the `Process Task - General`>`Process reference attribute` (but not the referenced process link). Create a new sub-process where we add the DMN based decision and multi-instance path.

Set the new name to *Travel Request Main Approval Process*, hit *Create* and then *Finish* to create the new main approval process.

As before, delete the predefined `Start event` and drag a new `Swimlanes`>`Pool` onto the process diagram and name it *Main Approval Process*. Then name the existing lane to *Travel Requestor* and add a new lane below (`Swimlanes`>`Lane`) named *System*.

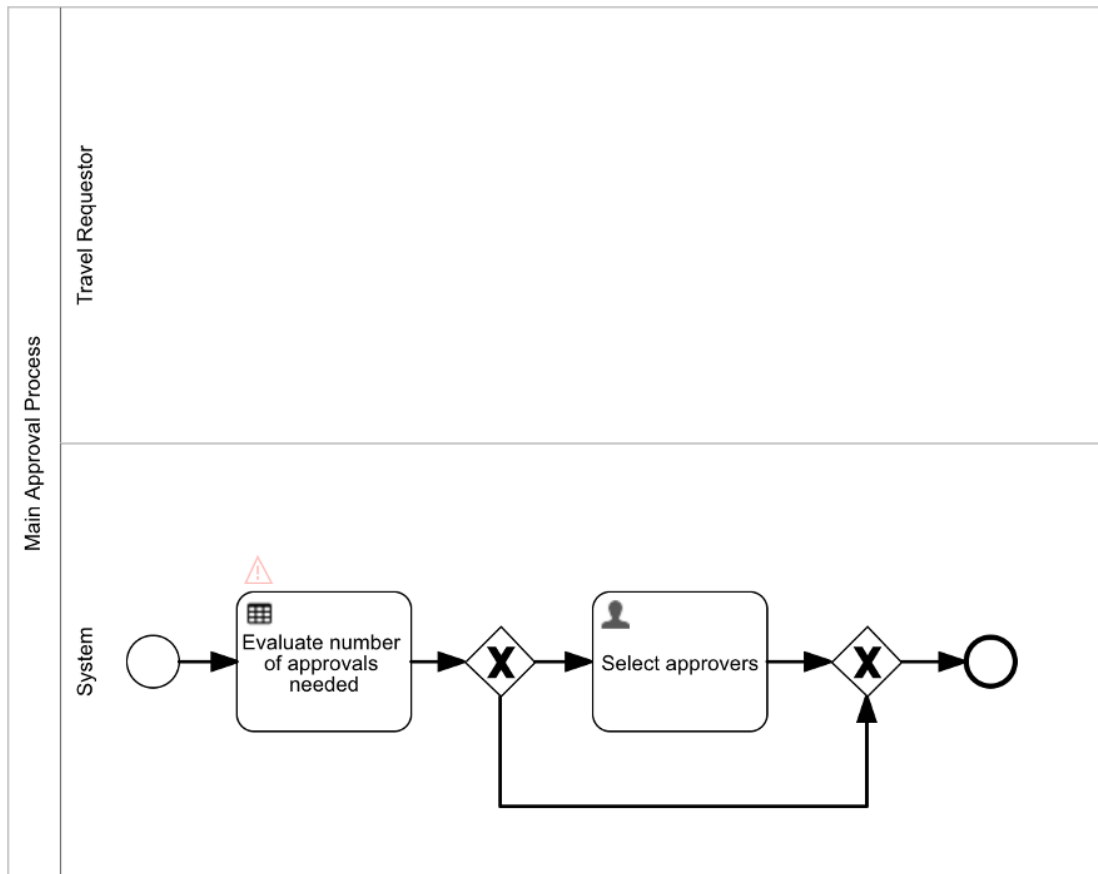
Add a `Start events`>`Start event` within the `System` lane and a new `Activities`>`Decision task`, name it *Evaluate number of approvals needed*. Next, connect `Start event` with the `Decision task` by dragging the small sequence flow icon (arrows) from the `Start event` onto the decision task.

Use the exclusive gateway icon after selecting the `Decision task` to add a gateway and then the `User task` icon to add a new `User task` after the gateway and name it *Select approvers*.

After the `User task`, add a gateway again by using that small icon, then in the same fashion an end event.

Go back to the first gateway, use the sequence flow icon again and drag it to the second gateway to add an alternative path, then use the centric handle of that sequence flow to drag it below the user task.

The process model now looks like this:



Define a DMN Decision Table for the Number of Approval Evaluation

In the next step, we want to define a DMN based decision table to evaluate if and how many approvals we need for the travel request.

Select the **Decision** task and click the **Decision table reference** property, enter *Travel Request Approval Decision Table*, hit *Create* and then *Finish* to create a new decision table.

A DMN based decision table is defined using some input values and one or even more output values. By default, the **First** hit policy is selected which means, the engine evaluates all the conditions on the input values defined and stops at the first row where all conditions are met and returns the output value(s) specified on that row.

Click the **New Input** blue header to define our first input parameter and modify it like this:

Edit input

×

Label

Estimated costs

Variable name *

estimatedCosts

Variable type *

Number

Allowed values (optional)

Add custom property (+)

Cancel

Save

We can map process variable values to our DMN input columns on which to base our decisions.

For the next column, click the small, blue plus icon to add another input column and define it like this:

Edit output

×

Label

Transportation

Variable name *

transportation

Variable type *

String

Allowed values (optional)

publicTransportation

car

flight

flightAndRentalCar

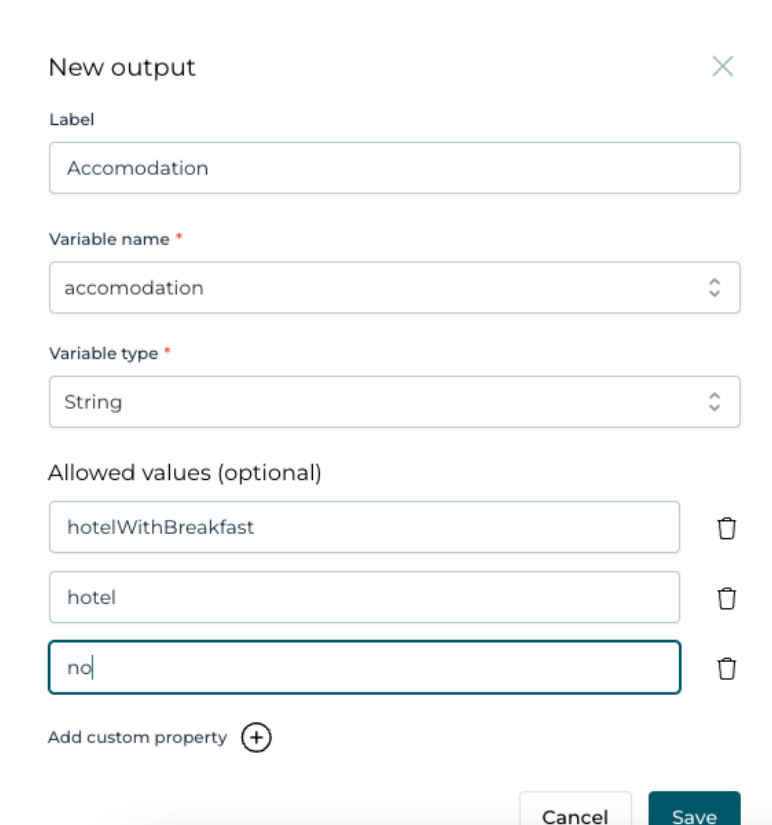
Add custom property (+)

Cancel

Save

As you can see, we can use our previously defined transportation options, so it is easier to use those values within our table later on.

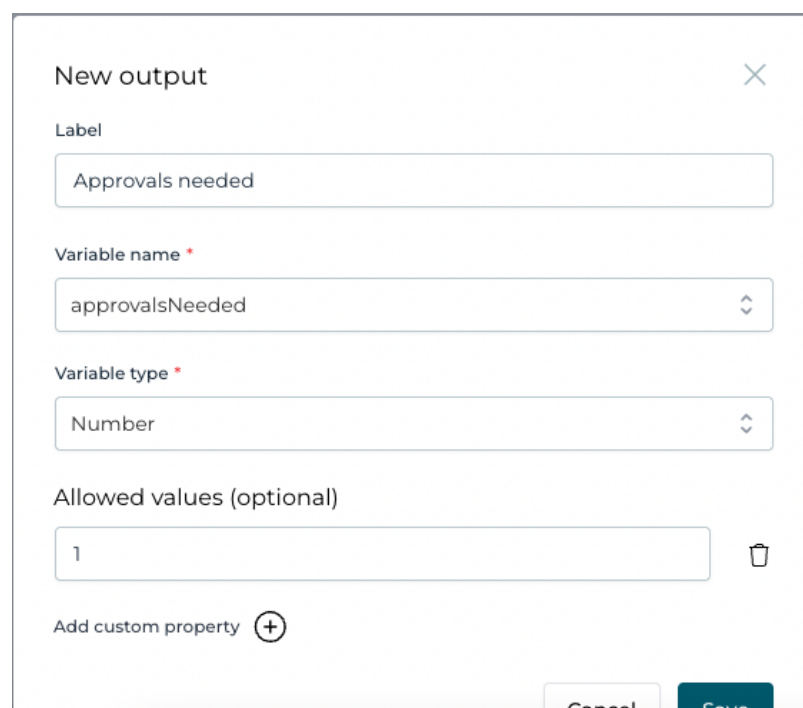
Let us add a third column with the following values:



The screenshot shows a 'New output' dialog box with the following fields and options:

- Label:** Accomodation
- Variable name ***: accomodation
- Variable type ***: String
- Allowed values (optional):** A list containing 'hotelWithBreakfast', 'hotel', and 'no'.
- Add custom property**: A button with a plus icon.
- Buttons:** Cancel and Save.

Finally, we want to define our outcome of the decision table, which is the number of approvals needed for the travel request according to the input values we have. For this click on the grey **New Output** header to define our output value:



The screenshot shows a 'New output' dialog box with the following fields and options:

- Label:** Approvals needed
- Variable name ***: approvalsNeeded
- Variable type ***: Number
- Allowed values (optional):** A list containing '1'.
- Add custom property**: A button with a plus icon.
- Buttons:** Cancel and Save.

Now we have defined our input and output values. The next step is to add some rules for the decision table, playing with the input values:

Hit Policy: F

Estimated costs <i>number</i>	Transportation <i>string</i>	Accomodation <i>string</i>	Approvals needed <i>number</i>
<= 200			0
<= 300	publicTransportation	-	0
<= 500		-	1
<= 1000	flight		1
<=			2

It is a good practice for a first-hit policy table to add the last row without any conditions on the input values to ensure the table always produces an outcome.

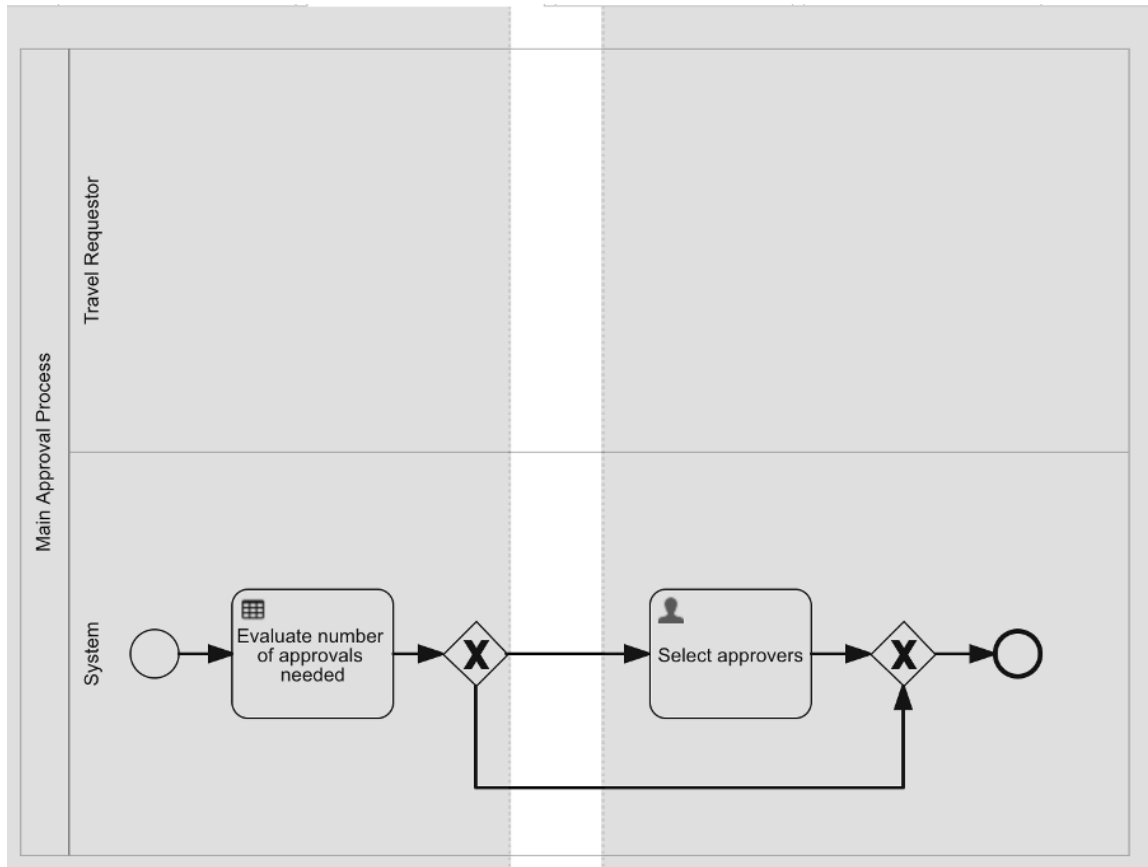
In our example, we defined some rules (rows) according to the estimated costs and transportation; we did not take the accommodation into account. Of course, you can adapt the rules to your needs.

Make sure you save the decision table and then return to our process.

Now we want to make use of our outcome, so select the first gateway and name it *Approvals needed?* and place the label on top of the gateway.

Making more space

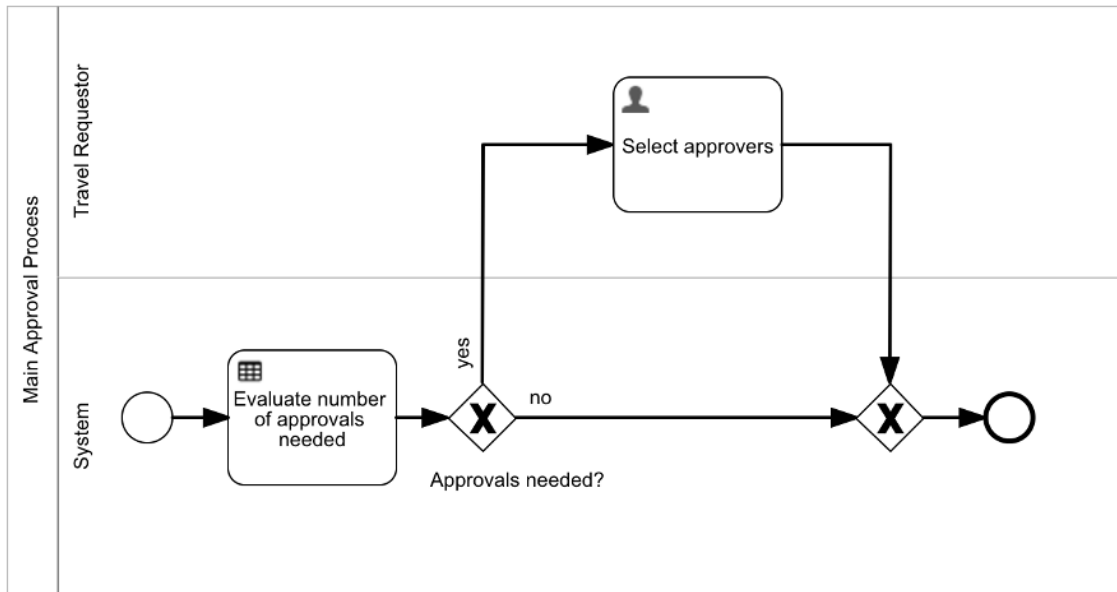
If you need more space between existing elements, select the **Make space** tool in the toolbar and drag the elements away from each other to make more space or towards each other to reduce the space between elements. Manipulating the amount of space works both horizontally and vertically (see the selected icon in the image below, it must be activated to make use of the spacing tool):



Now select the first outgoing flow leading to the `User` task, name it *yes*, set the `Details>Condition` type to `Conditional` flow and enter `${approvalsNeeded > 0}` as the `Details>Condition` expression. The engine evaluates that condition and if true (in our case, if the decision tables outcome was more than 0), it follows that path.

Now select the other outgoing flow, name it *no* and check the `Default` flow checkbox. It is a good practice to always have one outgoing flow set as the default. As an alternative, we also could make it a conditional flow and set its condition to `${approvalsNeeded == 0}`.

We also want to move the `Select approvers` user task to the `Travel Requestor` lane. You can drag and drop it from one lane to the other. You can then also rearrange the sequence flows and labels to get the process looking nice:



Modify the Task Form to Select Multiple Approvers

As there can be more than one approver, we need to modify the approver selection task form. For this first select the `Select approvers` user task in our process, then the `Form` reference property and choose "Reference", as we already have such a task form available, we need to modify it.

Select the `Select Approver Task Form` and then click on the form link to open it.

Double click the first text element and set its content to something similar to this:

Paragraph

14px

A

Please select {{approvalsNeeded}} person(s) to review and approve or decline your travel request.

This might include your superior or project manager, depending on the reason for the travel.

p

tiny

```
Please select {{approvalsNeeded}} person(s) to review and approve or
decline your travel request.
```

```
This might be your superior or project manager, depending on the reason for the travel.
```

We use the number of approvals needed from the output of our decision table as an expression.

In our old selection task form, we just had one selection element where we selected one person as the approver. Now we might need more than one. The multi-entry subform now comes in handy for this.

So let us add a **Container>Subform** at the bottom of the form and name it *Approvers*. Go to the right hand side panel and click on **Subform - General>Store subform data in a single variable** and set it to `{{root.approval}}`. Then you will be able to see **Minimum** and **Maximum** elements attribute under the **Subform - Validation>Validation**. Click on the pencil icon to edit the **Minimum** and **Maximum** elements with information as an output of our decision table, so use that information by clicking the small "lightning" icon on the right and then enter `{{approvalsNeeded}}` for both the elements.

As we want the user to enter exactly that amount of elements in the subform. We do not need to add or remove elements, so uncheck the **Subform - Form>Show remove** button and **Subform - Form>Show add** button.

The properties of the subform should look now like this:

Search for a property 🔍

Subform - Validation

Validation

Minimum elements:
{{approvalsNeeded}}

Maximum elements:
{{approvalsNeeded}}

Validation Panel Display:
Show

Custom validations:
<not set>

Validation Messages

Minimum elements:
<not set>

Maximum elements:
<not set>

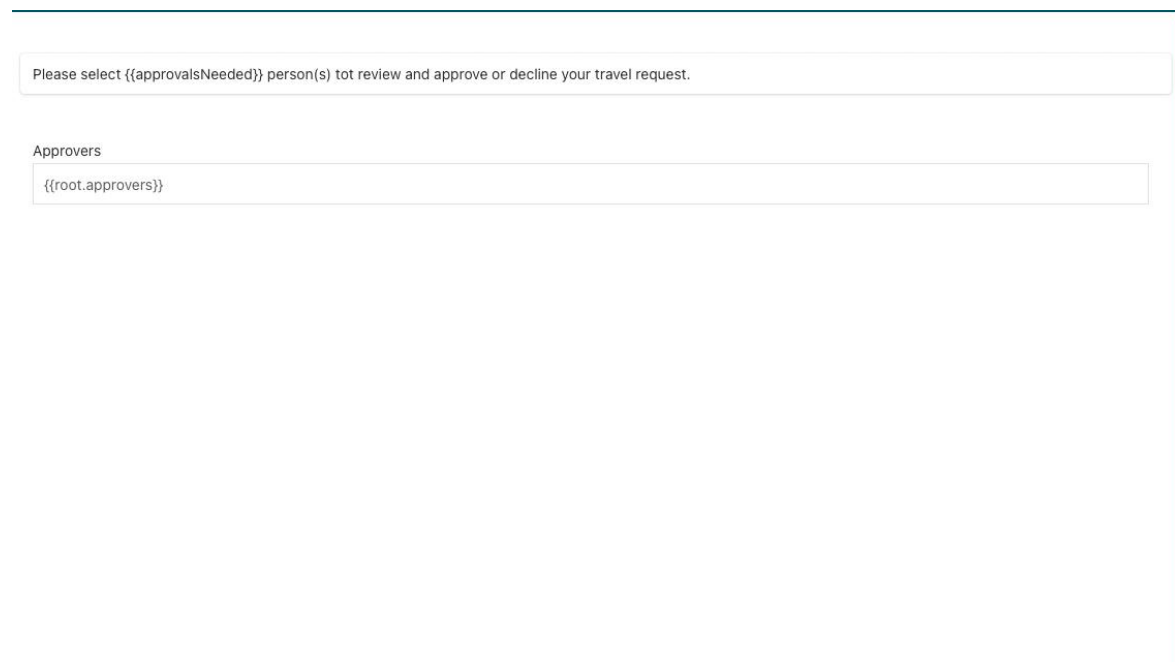
Next, we need an actual subform, so select the **Subform - General>Form** reference property, enter *Select Travel Request Approver Subform* as its name, hit *Create* and then *Finish* to create a new subform. This new subform is used within that list

of elements of our root subform component. Each element, depending on the number of approvers we need, renders that exact form.

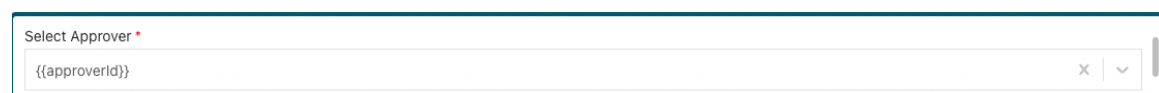
Go back to the `Select Approver Task Form` where we added the subform widget and select the `Select approver` component, then either hit *command-X* to delete it (or use the scissor icon in the toolbar) as we want it moved to the subform.

Go to the subform by clicking the `Details>Form` reference property link and paste the element from the clipboard onto the empty form.

The `Select Approver Task Form` now looks something like this:

A screenshot of a web form titled "Select Approver Task Form". At the top, there is a light gray instruction box that says "Please select {{approvalsNeeded}} person(s) tot review and approve or decline your travel request." Below this, the form is divided into two main sections. The first section is titled "Approvers" and contains a large, empty text input field with the placeholder text "{{root.approvers}}". The second section is a large, empty rectangular area, likely for a list of approvers or a table.

and the `Select Travel Request Approver Subform` like this:

A screenshot of a subform titled "Select Approver". It features a text input field with the placeholder text "{{approverId}}". To the right of the input field, there is a small "x" icon and a dropdown arrow. A vertical scrollbar is visible on the right side of the form.

The final step is to define the input variables needed for our process. Go back to the case model, select the `Travel request approval process`, and click the `Process Task - Variable Mapping>In` property to open the in variable mapping for the process, and enter this data there:

Source Variable	Source Expression	Target Variable		
<input type="text"/>	estimatedCosts	estimatedCosts		
<input type="text"/>	transportation	transportation		
<input type="text"/>	accommodation	accommodation		

i Map variables from the current case instance into the newly created process instance.

Now save everything (the best way is with the save-all toolbar icon), and we are ready for another test spin, at least to test the decision table output and the approver selection. Of course, we need to adapt the case again later, however, let us deploy and test in between nonetheless.

Deploy and Run the Travel Request Again

Publish the app again, switch to the runtime, and start a new travel request case. Fill out the details and choose something larger than 1000 to make sure we need two approvers to be selected. Then file the request, and you see a task named `Select approvers`. Select it, and there should be two selections required for approvers:

Task

Select approvers

[Travel Request](#) > [Travel Request Main Approval Process](#)

Created: 2 minutes ago

Assignee: [Shane Bowen](#)

Due: No due date

Set approver

Save

⋮

Task

People

Subtasks

Documents

Please select 2 person(s) to review and approve or decline your travel request.

This might include your superior or project manager, depending on the reason for the travel.

Approvers

Select approver *

▼

Select approver *

▼

We want to see what the decision table outcome was as there is a debug-view available at runtime. Select the `Travel Request Main Approval Process` in the header of the task to

navigate to the process. Next, select the "History" tab, and you can see the process diagram with the `Select approvers` as the current state. Now navigate to the sub-tab "Decisions" and the evaluated decision table is visible with the data we entered in the travel request:

Decision name		Rules	Result	Input	Initiator
Travel Request Approval Decision Table		5	approvalsNeeded: 2	initiator: shane.bowen, estimatedCosts: 3000, accom-	Shane Bowen
FIRST	Estimated costs {estimatedCosts} (number)	Transportation {transportation} (string)	Accommodation {accommodation} (string)	Approvals needed {approvalsNeeded} (number)	
1	<= 200	-	flight	0	2
2	<= 300	== 'publicTransportation'	-	0	
3	<= 500	-	-	1	
4	<= 1000	== 'flight'	-	1	
5	-	-	hotelWithBreakfast	2	

During modeling, this is very helpful, especially if the table does not produce the expected output. In this view, you can see how the rules are evaluated and which one was a hit or matched.

Add a Multi-instance Call Activity for Approvals

We want to use the collection of selected approvers to repeat the approval sub-process for each of the selected approvers.

Add a Multi-instance Call Activity

Go back to our `Travel Request Main Approval Process` and make sure there is some space between the `User` task and the gateway.

Add a `Structural>Call` activity by dragging and dropping it directly onto the sequence flow between the `User` task and the second (joining) gateway and name it *Approval subprocesses*. Select the `Details>Process reference` property and use the tab "Reference" and select the *Travel Request Approval Process* (the one from the first part we previously created).

Of course, we need to adapt that sub-process, but let us first set up the call activity to be multi-instance.

For this, go to the property `Multi instance>Multi instance type` and set it to `Parallel`, which means, we start a new sub-process instance in parallel for each of the selected approvers. We could also choose `Sequential`, which would start them one after the other.

The `Multi instance>Element index variable` is predefined as `loopCounter`, which is fine and allows us to use it as the index counting for each of the sub-process instances (starting with 0). We need to define the collection to loop for, so we set the property `Multi instance>Collection` to `${root.approvers}`. That variable was previously created through our multi-entry subform and contains a list (collection) of the selected approver user ids. More data is added for each element through that sub-process later on.

Let us also specify an element variable which we can then use within the sub-process to access our specific element. So set `Multi instance>Element` variable to *approver* for that. Next, add some values we need within the sub-process as in-variables. Select the `Process Task - Variable Mapping>In` property on the right hand side panel and set the following values:

Source Variable	Source Expression	Target Variable		
<input type="text"/>	approver	approver		
<input type="text"/>	loopCounter	loopCounter		

i Map variables from the current case instance into the newly created process instance.

Now let us adapt the approval sub-process to be ready for multi-instance approvals.

Select that process by clicking on the `Travel Request Approval Process` link on the `Details>Process reference` property of the call activity.

First, we remove (delete) the first user task (`Select approver for travel request`) where we previously selected the approver as this is now done in the main process. We also move the `Start` event into the second lane of the `Travel approver` and later remove the travel requestor lane entirely, as we do not need it anymore. If we add a feedback or adjusting loop between the approver and requestor at some point, then it would make sense again to have that lane.

We also change the expression used for the assignee of the approval user task which is just `${approverId}` in our previous version, but now needs the value of our local loop element variable.

Select the `Approve travel request` user task, go to the `Assignment>Assignee` property and set it to `${approver.approverId}`. This uses the approver id variable of the local element data we have on the sub-process scope.

Adapt Approver Task Form

Once done, let us adapt the approver task form, so select the user task, then click on the `Approve Travel Request Task Form` link to open that task form.

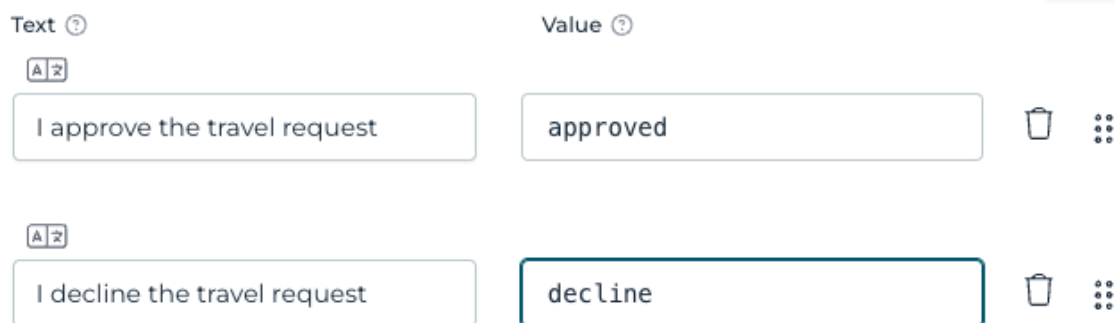
We need to change the binding of the review comments as it is added to each approver within our collection.

So select the `Review comments` input field and set its `General>Value` to `{{root.approvers[loopCounter].reviewComments}}`. Remember: we have our approver collection on the case level; hence, we need `root` to resolve it. Then we set that review comment at the index of our multi-instance loop counter. That is why we use the `loopCounter` as the index within the collection. Lastly, we use the dot-notation and add a new field named `reviewComments` to that collection. Now each comment is found at the same index as the approver user id.

In the first version, we used the outcomes to approve or decline. Let us change this also to see alternative outcomes. First, remove the outcomes by clicking the `Details>Outcomes` and then remove the two outcomes that are currently there.

Now drag and drop a `Selection>Radio` button component between the text display and the comment field and name it *Review decision*. Of course, we want that review decision stored in our approvers' collection as well, so set the `General>Value` of the radio buttons to `{{root.approvers[loopCounter].reviewDecision}}`. Now make the component required by selecting `Validation>Required` and set its `Details>Orientation` property to `Horizontal`.

Click the `Data source>Items` property and add the following values to it:



Text ?	Value ?
<input type="radio"/> I approve the travel request	<input type="radio"/> approved
<input type="radio"/> I decline the travel request	<input checked="" type="radio"/> decline

We want the required option of the review comment to depend on the decision, so click the lightning icon of the `Validation>Required` property and set its expression to `{{root.approvers[loopCounter].reviewDecision == 'declined'}}`. This expression means that whenever we select the declined option, the comment field is mandatory; otherwise, it is optional.

Save the form and go back to the sub-process as we need to deal with the decision of the task (approve or decline).

Collect the Approval Result from Each Approver

Each approval user task produces a decision according to the approver's selection whether to approve or decline the travel request, which is stored back at the same index within the approvers' collection. We want to add it to the parent process level and combining it with all the other approval results. To do that, add an `Flowable Work Activities>Initialize variables` service task after the `Approve travel request` user task. Label the new task, *Set approval result* and make sure `Details>Overwrite if existing` is checked and add one variable initialization to the `Details>Init variables` property:

```
${parent.approval && flwJsonUtils.getAtIndex(root.approvers, loopCounter).reviewDecision == 'approved'}
```

Initialize Variables

Set approval result (bpmnTask_6)

parent - approval - expression

Target

parent

Name

approval

Value type

Expression

Value

`${parent.approval && flwJsonUtils.getAtIndex}`

i

Specify the variables that will be initialized when this task is executed.

The **Target** column is optional. It can be used when defining a complex variable JSON structure or with values *root* or *parent*. Setting *root* or *parent* will set the variable not on the local instance, but on the parent or root instance in a nested hierarchy of process/case instances.

The **name** column is used to determine the name of the variable and is mandatory.

The **Value Type** column is used to determine the type of the variable, which is important as it switches the input field type depending on the chosen type.

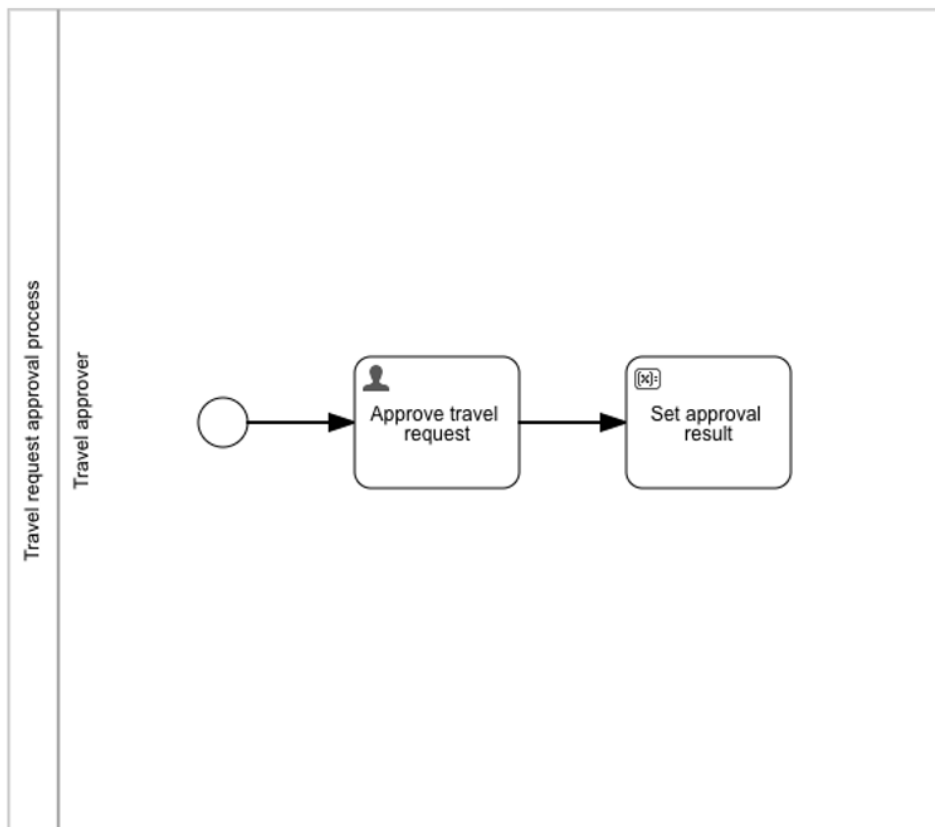
The **Value** field allows to set the value of the variable statically or

Cancel

Finish

We use `parent` as the target because we want to set that `approval` variable on our parent main process instance. As the variable value, we use an expression, where we use the currently set decision value and combine it with the outcome of the local selection.

The Travel Request Approval Process now looks something like this:



Now the first part of the expression, `parent.approval` is a variable that is not initialized the first time, so we need to make sure it exists and is set to true, otherwise, we would never be able to approve the case. So go back to the main travel request case model and select the **Travel request approval process** task and open the **Process Task - Variable Mapping**. In variable mapping and add initialization of our `approval` variable on the main process instance level:

Process Task

Travel request approval process (cmmnTask_7)

Source Variable	Source Expression	Target Variable		
estimatedCosts		estimatedCosts	🗑️	⋮
transportation		transportation	🗑️	⋮
accommodation	\$(true)	approval	🗑️	⋮

+

i Map variables from the current case instance into the newly created process instance.

With the expression `$(true)`, we create a Boolean true value to initialize the `approval` variable.

Now for the `Details>Out` mappings (what comes back from the process and is saved on case level), we remove all the previous mappings and only add *approval* to the list:

This means we store the final `approval` variable value from our main process back to the case, once all approval sub-processes are finished. We then have the final approval flag (true or false) back on the case and all the detailed results in our `approvers` collection.

As our result now is a Boolean value, we need to adapt the entry sentry expression (condition) on the entry criterion of our last stage. So select the entry criterion and set the Condition to `$(vars:getOrDefault('approval', false))` to create a Boolean variable value (from a string based in our first version of the approval process).

Improve the Case Overview Form

As we no longer have a single approver, let us modify the case overview (work) form by adding a multi-entry subform with an overview of the approvals.

Go back to the case model and open the work form (`Travel Request Work Form`). Delete the two existing text display components at the top of the form representing the approval review.

Add a `Container>Subform` component at the top of the form and name it *Approval overview*. Then set the value binding of `General>Store` subform data in single

variable to `{{approvers}}`. Also check the property `Details>Multiple elements` as we want to have more than one subform rendered, as the variable `approvers` is representing a collection, not a single value.

Turn off the `Details>Show add button` and the `Details>Show remove button` by unchecking each attribute. We also need the form to be read-only, so uncheck the `General>Enabled` property as well. Furthermore, we only want to show the form, if the approvers are initialized, so use the lightning icon on the `General>Ignored` property and set the expression to `{{!(approvers[0].approverId)}}`. The `approvers[0].approverId` part is only true, once we have at least one approver set in the collection of approvers.

Now select the `Details>Form reference` property, set the name for the new subform to *Travel Request Approval Overview Subform*, hit *Tab* and then *Enter* to create a new subform for our overview list.

We want to put the selected approver, the decision, and comments in the form. The easiest way is to go back to the approver selection form we did for the `Select approvers` task or just by opening the `Select Travel Request Approver Subform`. Select and copy that `Select approver` widget and go back to our newly created subform and paste it to the form. Remove its label and make it not required as we do not need it here.

Add a new `Data entry>Multiline text` component, and do not set a label as it is not needed, but set its `General>Value` property to `{{reviewComments}}`.

As the goal is to either display a text saying the review is not yet done or as an alternative, show the review result, add a `Container>Panel` component on the same row, in between the approver selection widget and the comment widget. Make the selection widget use 3 columns in width, the panel use 4 columns, and the comment use 5 columns.

Now drop a `Display>Text display` widget onto the panel and set the content to `Request was {{reviewDecision}}`. You can even mark the expression and make it bold for highlighting the decision result. The widget is shown once the decision is made, so set the `General>Ignored` property to `{{!reviewDecision}}`.

Now drop another `Display>Text display` in the panel below the existing one and set its content to *The request was not yet reviewed and decided..* This widget is visible if the review is not yet completed, so set the `General>Ignored` property to `{{reviewDecision}}`.

That subform now looks similar to this:

The screenshot shows a subform with the following fields:

- A dropdown menu with the label `{{approverId}}` and a value of `{{ApproverId}}`.
- A text field with the label "Request was `{{reviewDecision}}`".
- Another text field with the label "Request was `{{reviewDecision}}`".
- A text area with the label `{{reviewComments}}` and a value of `{{ReviewComments}}`.

Save the subform and go back to its parent form. Drag and drop a `Display>Text display` below the subform and set its content to something like:

The travel request review and approval was not yet started. So there is no approval result available yet.

Set the `General>Ignored` property to `{{approvers[0].approverId}}` so it is only shown if there is no approver set.

Now save everything, and we should be ready for another test.

Deploy and Run the Travel Request Again

Publish the app again, switch to the runtime view, and start a new travel request case as before. Make sure the estimated costs are greater than 1000, so the outcome requires two approvers and then file the case for approval.

After selecting the two approvers and completing that task, you should now see two approval tasks in the list of open tasks of our case:

Case

Travel Request

Started: 1 minute ago

Started By: Shane Bowen

Assignee: Shane Bowen

Define travel data

Approve travel

Organize travel

Terminate

Open tasks

Work form

People

Sub-items

Documents

Audit trail

History

Approve travel request

Anne Barton

Started less than a minute ago

No due date

Approve travel request

Ben Snow

Started less than a minute ago

No due date

They are assigned to the approvers you selected in the previous task and are created in parallel.

Of course, you cannot complete those tasks as the travel requestor, so log in with each of the selected approvers, go to their task inbox, select the approval task and make a decision (works like before). When both users approve the task, then the travel request is approved. If either one declines the request, then the case is declined.

If you approved with both approvers, you can go back to the case and see the `Organize travel` task like before.