Business Application Studio

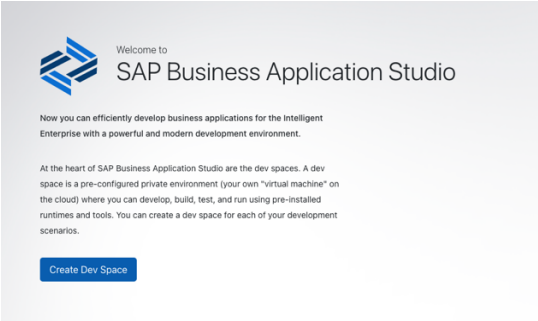
Login to your SAP HANA Cloud Trial Cockpit

- https://cockpit.hanatrial.ondemand.com/trial/#/home/trial

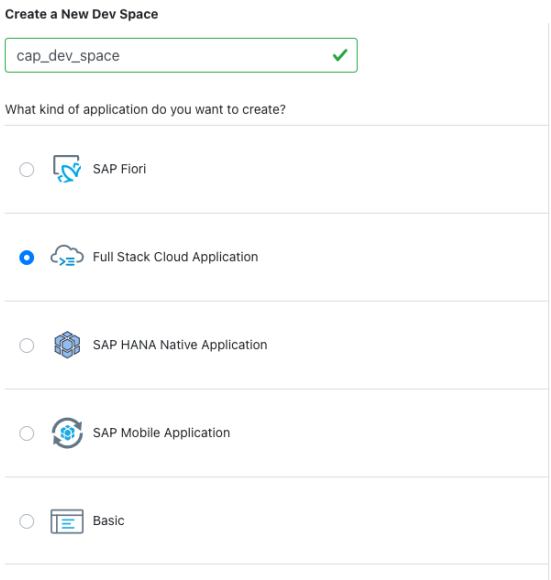
Open the Business Application Studio

- https://triallink.eu10.trial.applicationstudio.cloud.sap/

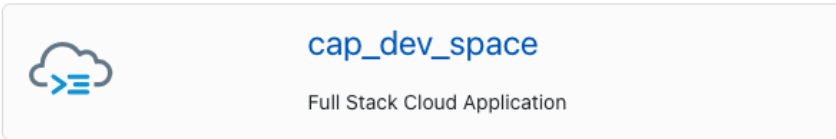
Create a new dev space



Select *Full Stack Cloud Application* and give it a nice name



Wait until the dev space is running, then start it by clicking the name

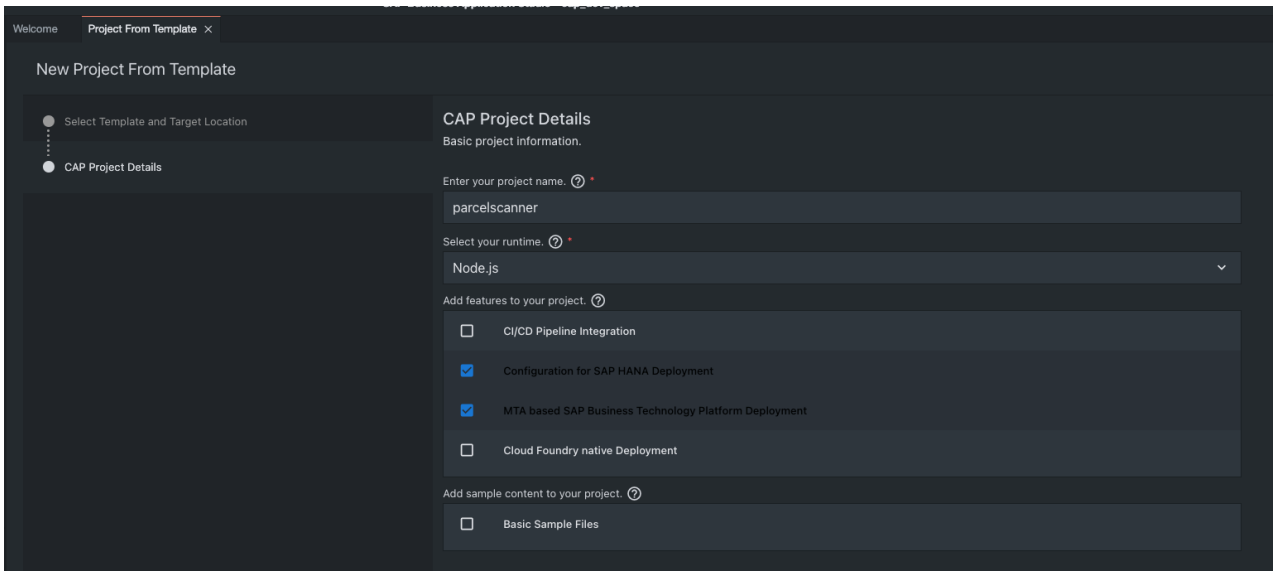


Project Setup

Open SAP Business Application Studio

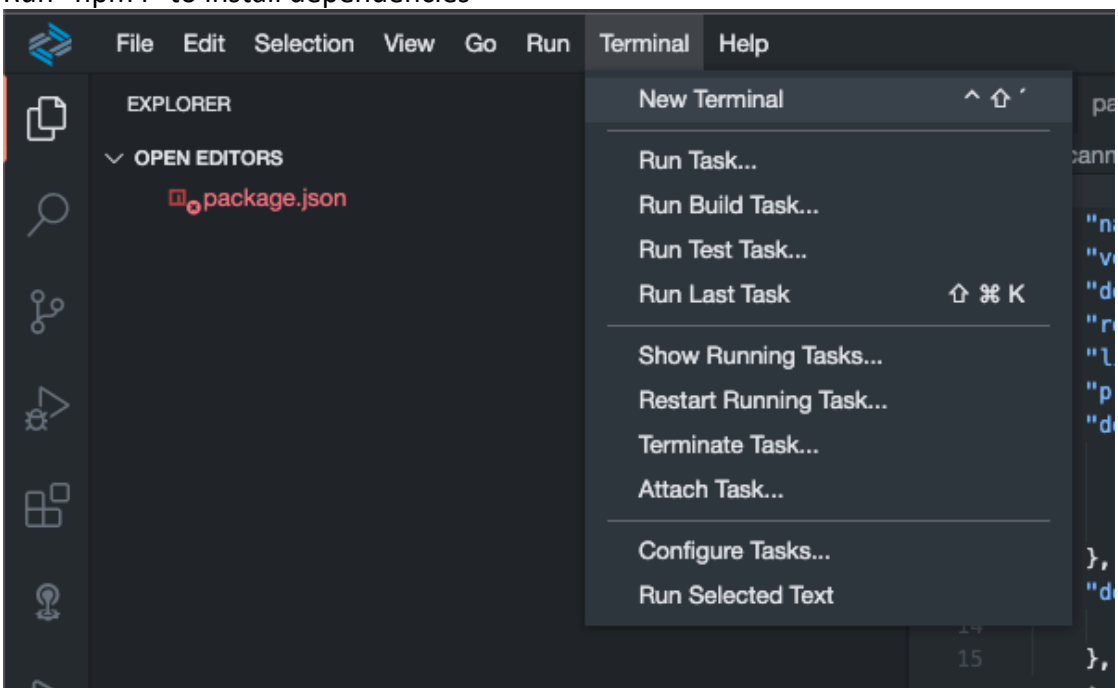
Create a new CAP Project

- Welcome > Start From Template > CAP Project



Open a new terminal

- Run “npm I” to install dependencies



Domain Modeling

Create Folder “db/data”

- This folder will contain our test data
- Download test data from https://github.com/danielhecker/parcelscanner/blob/main/db/data/my.parcelscanner-Parcels.csv
- Put the CSV file in the “db/data” folder

Create file “db/data-model.cds” and open it

Define namespace “my.parcelscanner”

```
namespace my.parcelscanner;
```

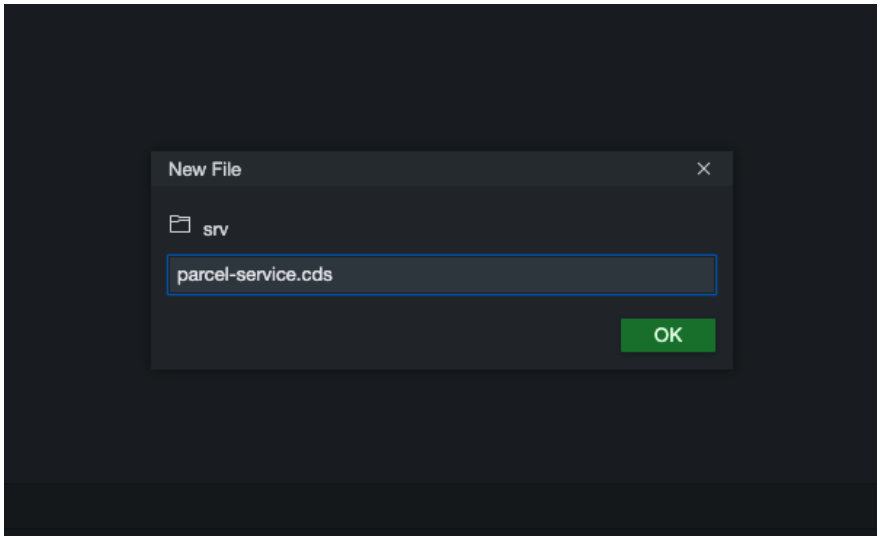
Define a new entity called “Parcels”

```
entity Parcels {}
```

TODO: Model the entity with the help of https://cap.cloud.sap/docs/guides/domain-models to match the provided test data

Core Data Service Definition

Create file “srv/parcel-service.cds” and open it



Import your data model

```
using my.parcelscanner as my from '../db/data-model';
```

Define a new service called “ParcelService”

```
service ParcelService {}
```

Add your imported entity Parcels to the service definition

```
service ParcelService {
  entity Parcels as projection on my.Parcels
}
```

Test your service

Open a new terminal

Run “cds watch” and open the provided link in a new browser tab

Click the link for your service endpoint and check if you can see the test data

Keep track on changes

Let the framework keeping track on changes by using the keyword “managed”. This will add fields for username and timestamp on create and modify operations.

Import the aspect “managed” from the CDS Common Library

```
using { managed } from '@sap/cds/common';
```

Modify the entity definition to use the aspect

```
entity Parcels: managed {
```

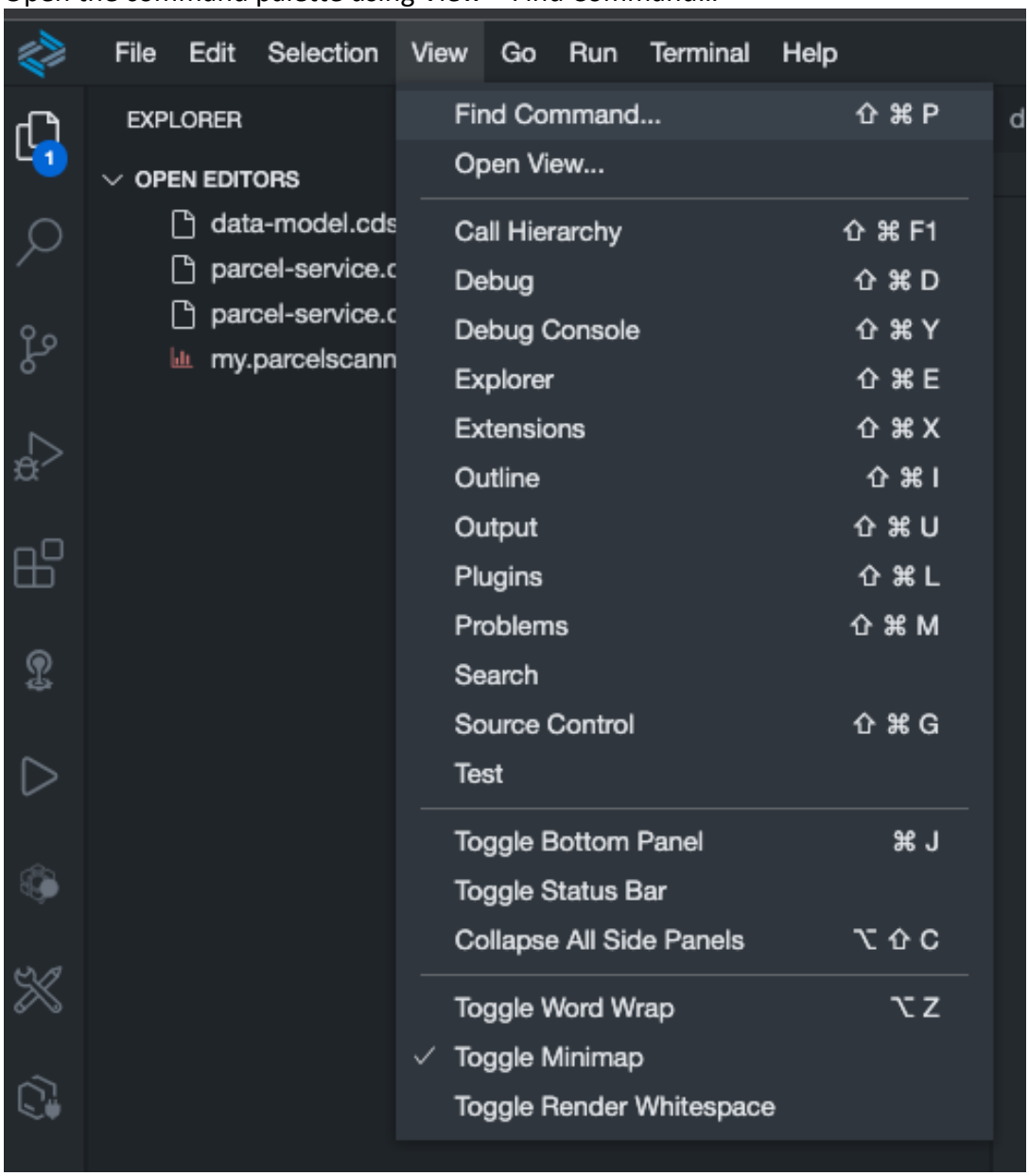
TODO Check the service endpoint in your browser and look for changes to the entity

Learn more about the aspect

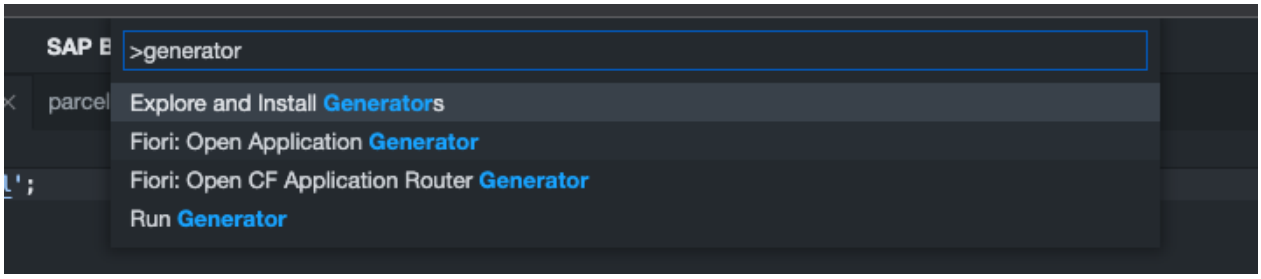
- https://cap.cloud.sap/docs/cds/common#aspect-managed
- https://cap.cloud.sap/docs/guides/providing-services#managed-data

Add a frontend to your application

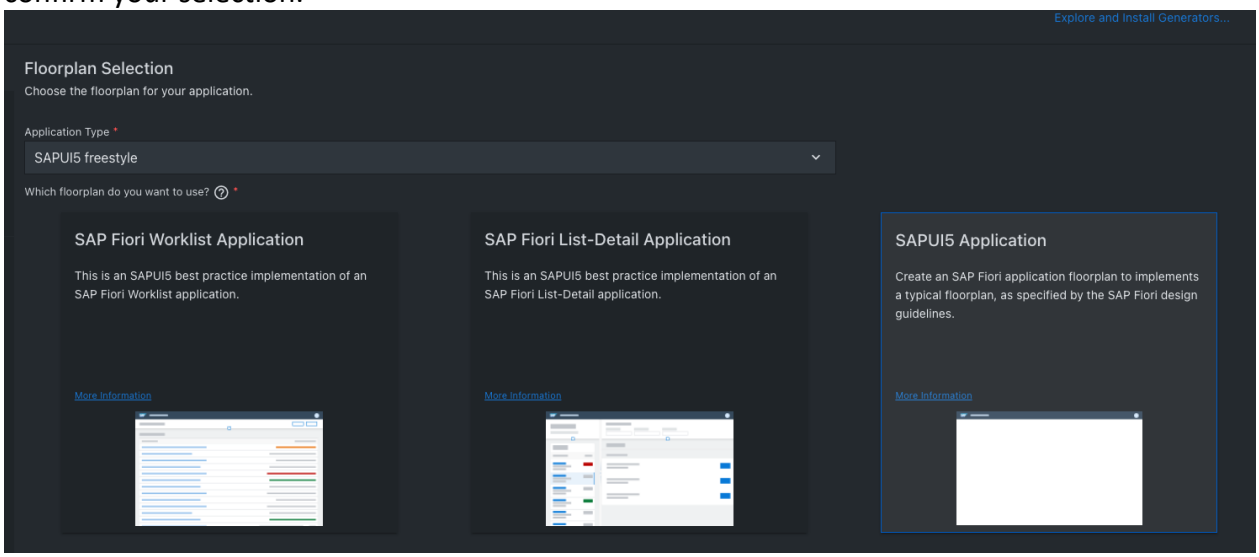
Open the command palette using View > Find Command…



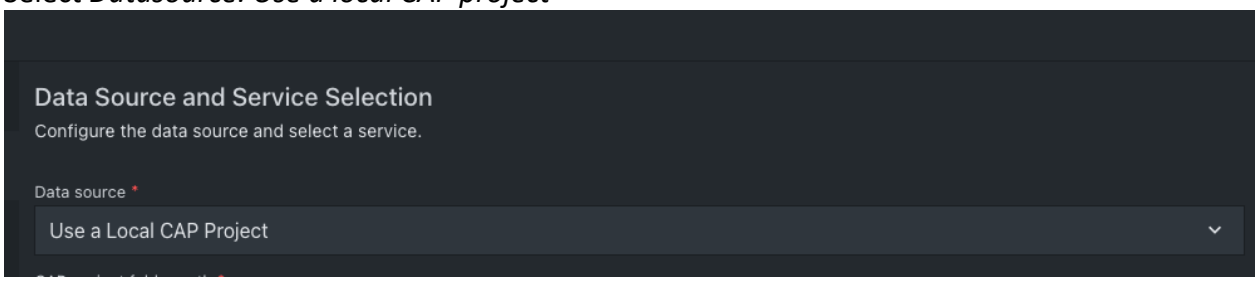
Search for “generator” and open the wizard by selecting *FIORI: Open Application Generator*



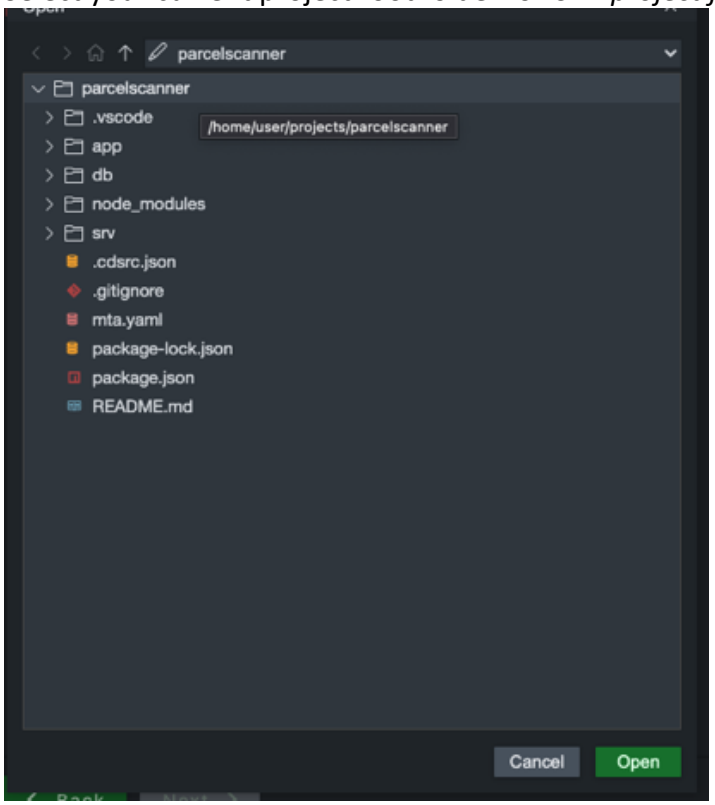
Select *Application Type: SAPUI5 Freestyle* and choose *SAPUI5 Application*. Click *next* to confirm your selection.



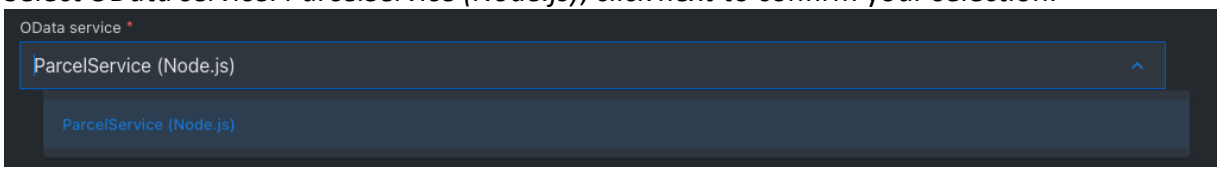
Select *Datasource: Use a local CAP project*



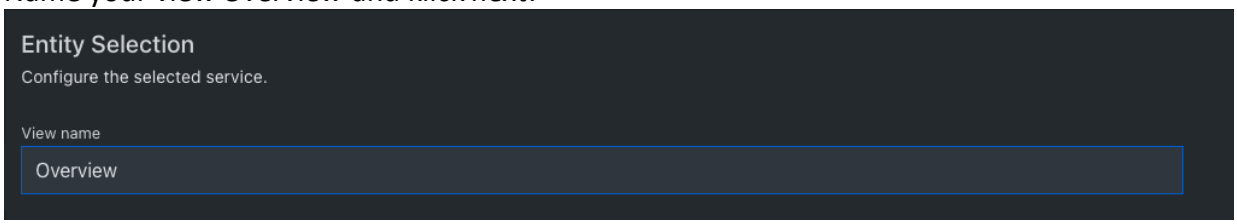
Select your current project root folder for *CAP project folder path*



Select *OData service: ParcelService (Node.js)*, click *next* to confirm your selection.



Name your view *Overview* and klick *next*.

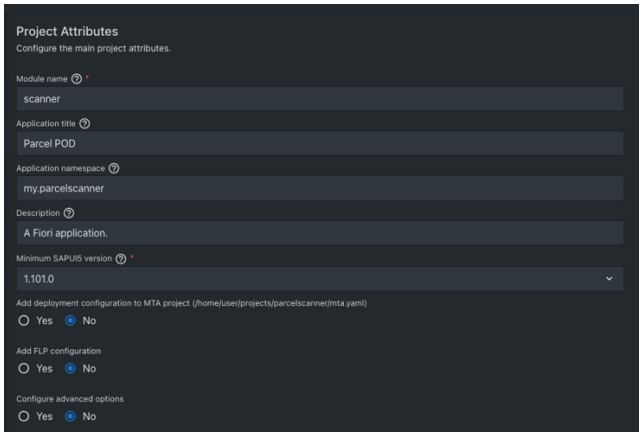


Fill out the project attributes

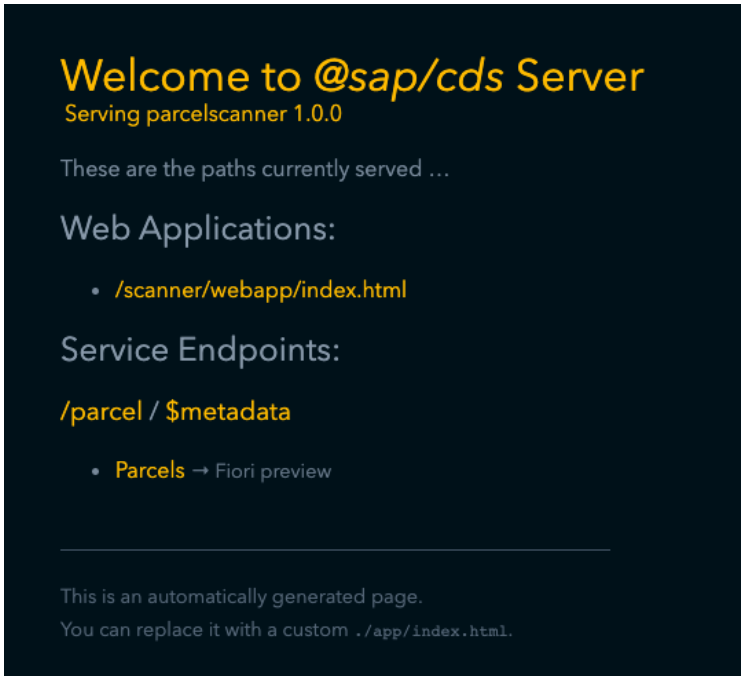
Module name: scanner
Application title: Parcel POD
Application namespace: my.parcelscanner
Description: A Fiori application
Minimum SAPUI5 version: Latest

Set all radio buttons to: No

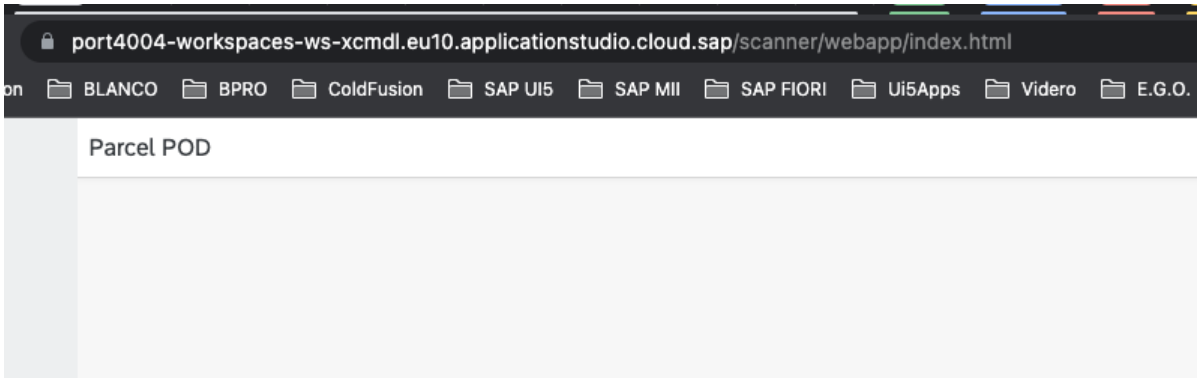
Confirm by clicking *Finish*



After the application has been generated, check your browser for the link to open it



Open the web application link, you should see an empty screen



Add UI elements to your frontend application

Navigate to your web app's source folder located at *app/scanner/webapp*

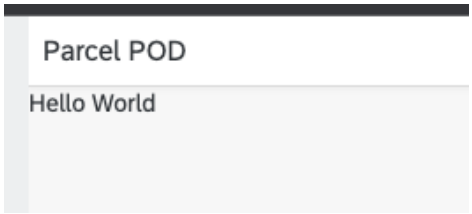
Open *view/Overview.view.xml* in a new editor tab. The xml file represents the app's UI elements.

Add a text control to the content of the page

```
<Page id="page" title="{i18n>title}">
  <Text text="Hello World" />
</Page>
```

We can remove the content tag as it's the default aggregation for a page control.

Check the web app in the browser, you should be greeted by *Hello World*



Learn more:

- https://sapui5.hana.ondemand.com/#/topic/1409791afe4747319a3b23a1e2fc7064

Add translations to your frontend application

The app supports internationalization (i18n) by default. Open the default i18n file at *i18n/i18n.properties*

Add a new translation variable to the file. It's a good practice to prefix your variable with the view name it's used in, something like *overview.hello=Hello World*

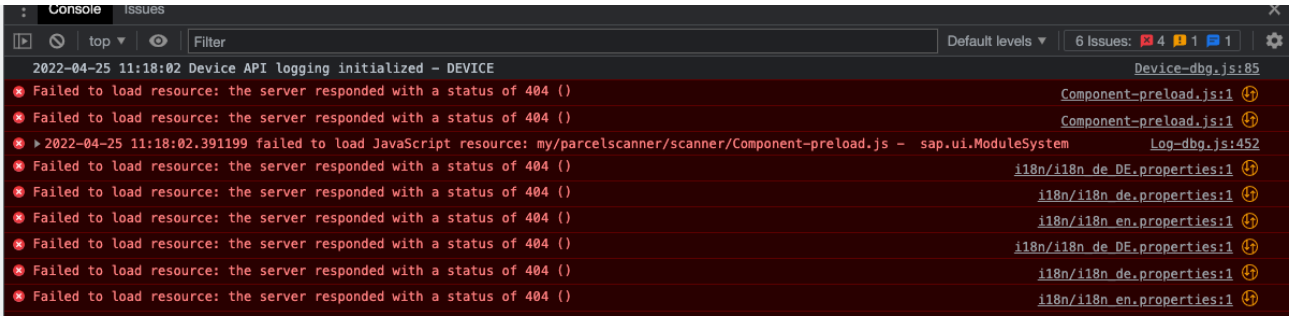
Navigate back to *Overview.view.xml* and replace your plain text with

```
<Page id="page" title="{i18n>title}">
   <Text text="{i18n>overview.hello}" />
</Page>
```

A couple of things are happening here:

- The brackets tell the framework to load data from a model
- "i18n" is the name of the model holding all translation texts (it's defined in manifest.json)
- "overview.hello" is the path inside the model from where the framework should read the content

Check the app in your browser and press *F12* to open your browser's developer tools. Check the console for error logs.



As you can see, the application is missing German translation texts and falls back to English.

TODO Create i18n/i18n_de.properties and add German translations for all texts defined in i18n/i18n.properties

Learn more:

- https://sapui5.hana.ondemand.com/#/topic/91f385926f4d1014b6dd926db0e91070

Add event handlers to your application

To handle events like a user clicking a button, we need to add an event handler to the controller.

First, we add a button control to our view, replacing the hello world text.

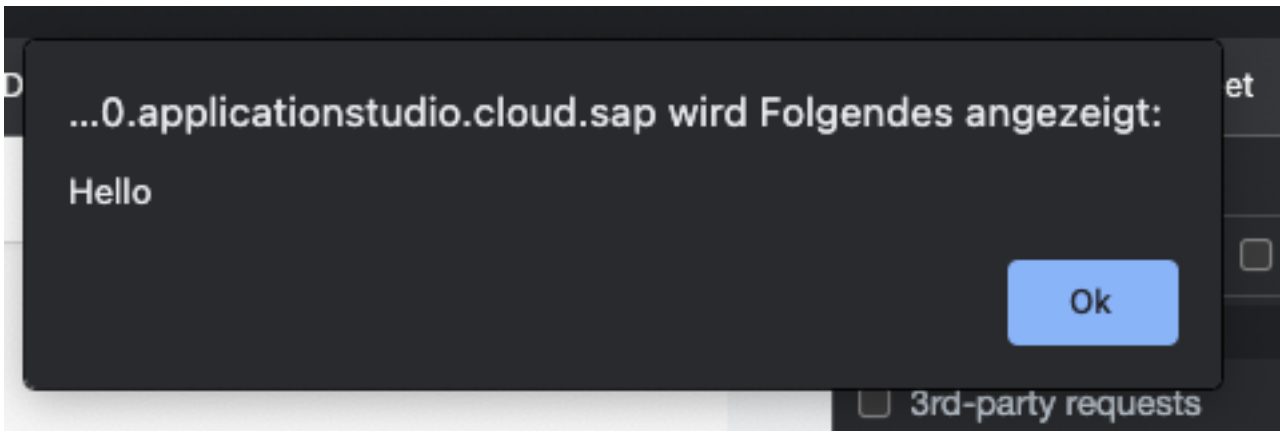
```
<Page id="page" title="{i18n>title}">
    <Button text="Say Hello" press=".onHelloWorldBtnPressed" />
</Page>
```

Next open *webapp/controller/Overview.controller.js* which controls our view.

Add a new function after the onInit function, which handles our button click event.

```
onHelloWorldBtnPressed(oEv) {
  alert("Hello");
},
```

Reload the app in your browser and click the button, check if an alert opens when clicking the button.



Learn more:

- https://sapui5.hana.ondemand.com/#/topic/50579ddf2c934ce789e056cfffe9efa9

Aggregation binding

By using the app generator our app already knows our ParcelService end point as backend. The definition can be found in the manifest.json in the web app's root folder.

```
"dataSources": {
    "mainService": {
      "uri": "/parcel/",
      "type": "OData",
      "settings": {
        "annotations": [],
        "localUri": "localService/metadata.xml",
        "odataVersion": "4.0"
      }
    }
  }
```

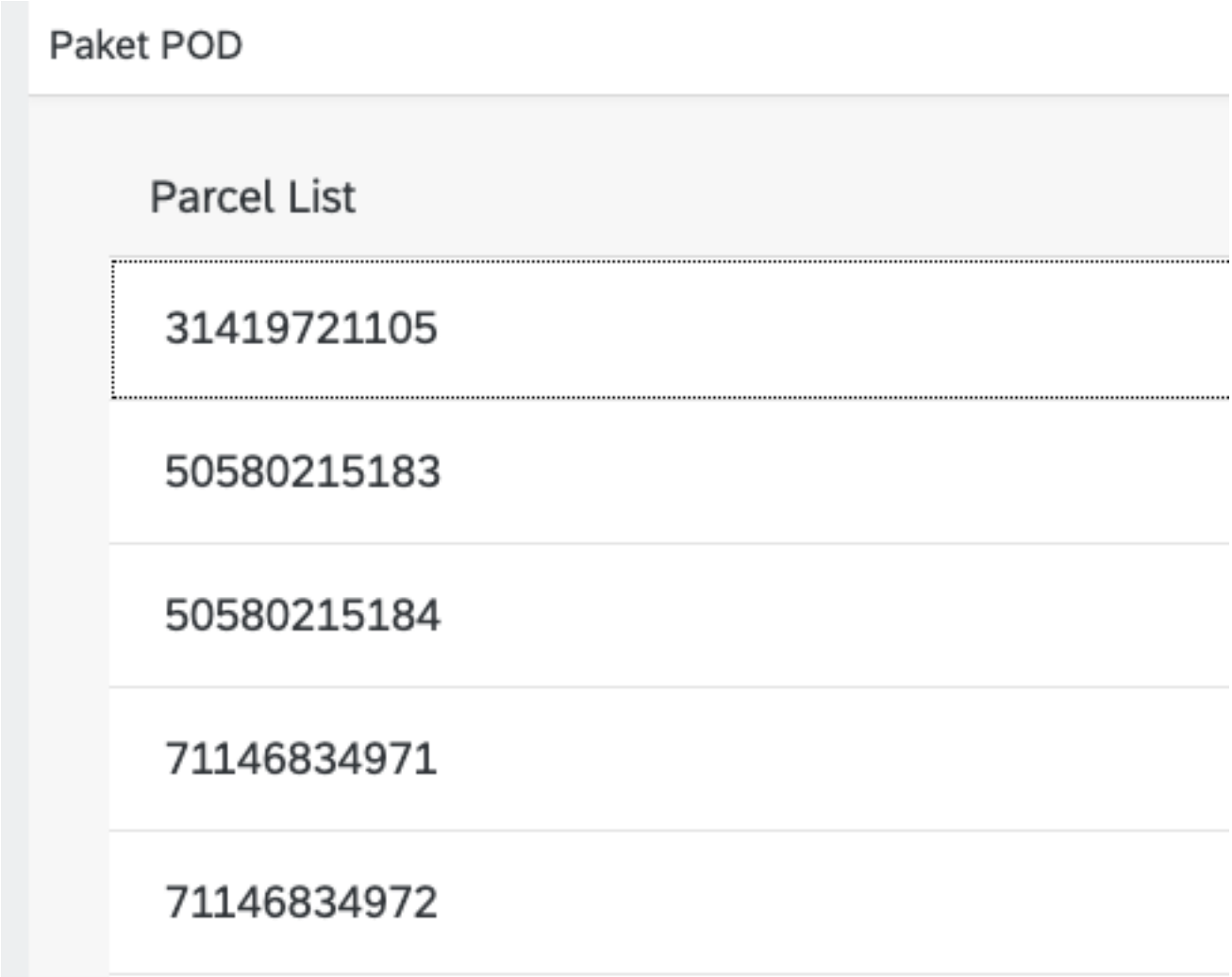
We add a list control to our view and bind its item aggregation to the /Parcels end point of our backend service to show all parcels currently in our database.

```
<List
   headerText="Parcel List"
   class="sapUiResponsiveMargin"
   width="auto"
   items="{/Parcels}" >
   <items>
       <ObjectListItem
               title="{ID}"/>
   </items>
</List>
```

In the items aggregation we define a template which will be automatically repeated for each entry returned by our end point. In our example we create a ObjectListItem control for every child of the item aggregation. The title of the control is the ID field of the Parcels entity.

We do not have to name the model as our backend service is the default (nameless) model for our app.

Save the view and refresh your web app, you should see all parcel IDs shown in a list.



TODO

- *Add a meaningful header text to the list and translate it*
- *Look up the List and ObjectListItem control in the official documentation*
- *Design the list in a way all important information is shown to the user*

Learn more:

- https://sapui5.hana.ondemand.com/#/topic/bf71375454654b44af01379a3c3a6273
- https://sapui5.hana.ondemand.com/#/api/sap.m.List
- https://sapui5.hana.ondemand.com/#/api/sap.m.ObjectListItem
- https://sapui5.hana.ondemand.com/#/entity/sap.m.List

Filtering

Our backend service supports filtering of data which means we can filter the data in the backend instead of the frontend and by doing so, sending only the relevant data over the network which results in an overall faster app and better UX.

We can apply static filters directly in the view without writing any code in the controller. We can add filter definitions to the data binding path of our list.

```
<List
    headerText="Parcel List"
    class="sapUiResponsiveMargin"
    width="auto"
    items="{
        path: '/Parcels',
        filters: [
                {
                  path: 'status',
                  operator: 'EQ',
                  value1: '1'
                }
              ]
         }" >
```

The filters array takes 1-n filters, every filter consists of a path (field of the entity which should be filtered), an operator (equals, not equals…) and a value.

In our use case, we're only interested in parcels which are handled by "Geek Logistics Systems".

TODO

- *Modify the sample code to filter the list for parcels handled by "Geek Logistics Systems"*

Learn more

- https://sapui5.hana.ondemand.com/#/api/sap.ui.model.Filter
- https://sapui5.hana.ondemand.com/#/api/sap.ui.model.FilterOperator

Sorting

Our backend is also able to sort the data before sending it to the client. Sorters are added in the same way as filters as part of the items binding.

```
items="{
    path: '/Parcels',
    sorter: [{
        path : 'ID',
        descending: false
    }]
    }
```

A sorter takes the path to the data and a descending flag which can be either true or false.

TODO

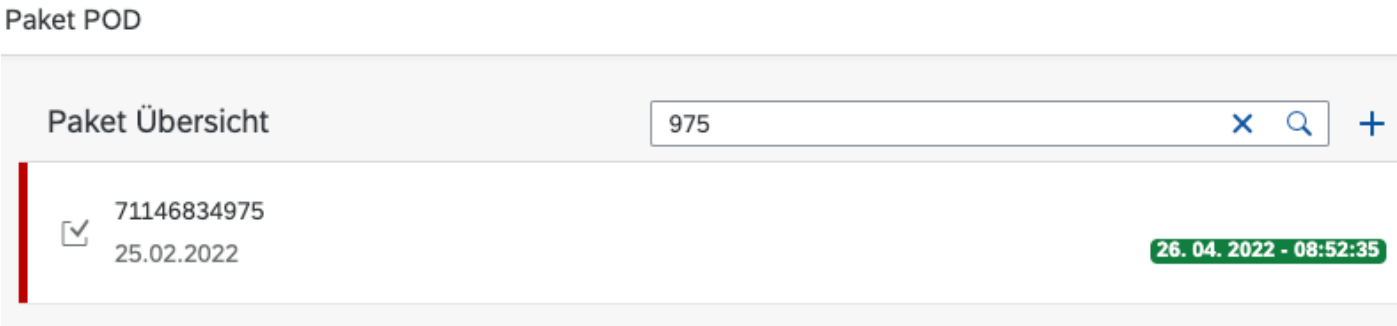
- *Discuss a meaningful sorting of the list*
- *The sorting should at least consist of deliveryDate, deliveryTime and ID*
- *Implement the sorters using the items binding of the list*

Learn more

- https://sapui5.hana.ondemand.com/#/topic/c4b2a32bb72f483faa173e890e48d812
- https://sapui5.hana.ondemand.com/#/api/sap.ui.model.Sorter

Add more features

Search the list



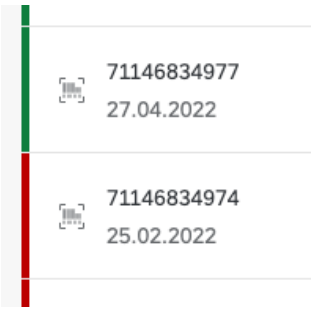
Add a SearchField control to the view to allow the user to filter the list for a given parcel ID. Such dynamic filters are applied on controller level. Follow SAP's official tutorial on how to filter a list for a search term.

- https://sapui5.hana.ondemand.com/#/topic/5295470d7eee46c1898ee46c1b9ad763

Learn more

- https://sapui5.hana.ondemand.com/#/api/sap.ui.model.Filter
- https://sapui5.hana.ondemand.com/#/api/sap.ui.model.FilterOperator
- https://sapui5.hana.ondemand.com/#/api/sap.m.SearchField

Custom formatters



For complex formatting of properties provided by the data model, you must write custom formatting functions, which are put in a separate formatter file. In our list we want to highlight parcels which were not delivered in time.

Write a custom formatter for the “highlight” property of your list item control which returns “Error” if the deliveryDate is later than today or “Success” if the deliveryDate equals or is before today.

Use the following code to create a JavaScript date object from the deliveryDate property of the data model

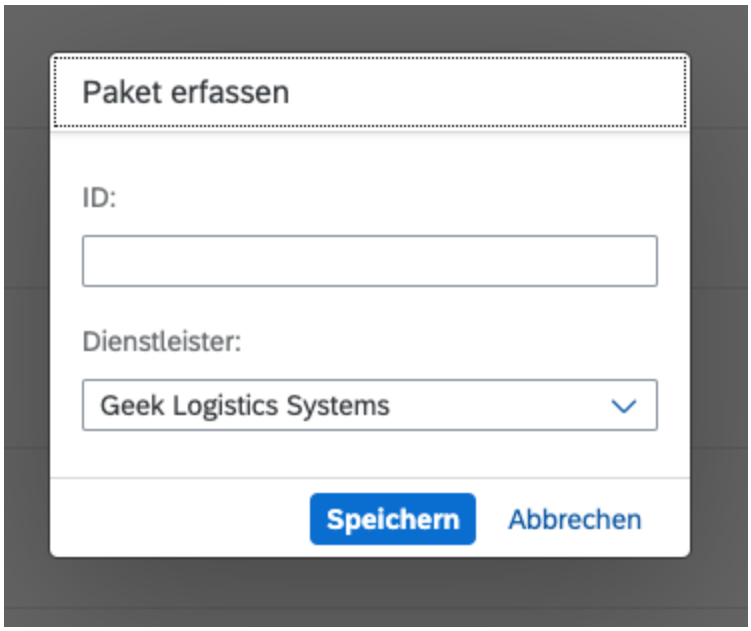
```
const oDateFormat = sap.ui.core.format.DateFormat.getDateInstance();
const oDeliveryDate = oDateFormat.parse(sDeliveryDate);
```

Learn more

- https://sapui5.hana.ondemand.com/sdk/#/topic/91f2eba36f4d1014b6dd926db0e91070.html
- https://sapui5.hana.ondemand.com/#/topic/0f8626ed7b7542ffaa44601828db20de

Create and update Entries

Use the “press” event of the ListItem control to update the entries status from 1 to 2. This is done in the controller by using the setProperty function of a list binding.



Create a form to allow the user to create a new parcel. This is done in the controller by using the create function of a list binding.

Sample Code

- https://github.com/danielhecker/parcelscanner/blob/main/app/scanner/webapp/controller/Overview.controller.js

Learn more

- https://sapui5.hana.ondemand.com/#/topic/c9723f8265f644af91c0ed941e114d46
- https://sapui5.hana.ondemand.com/sdk/#/topic/b4f12660538147f8839b05cb03f1d478