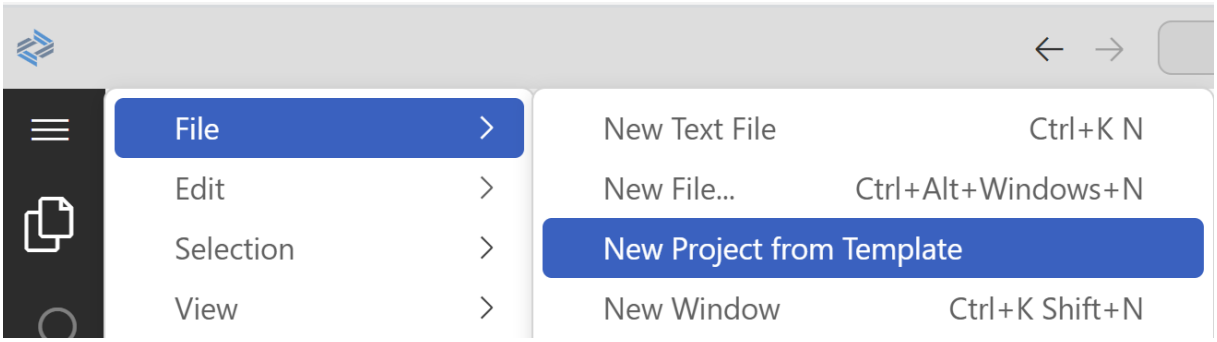
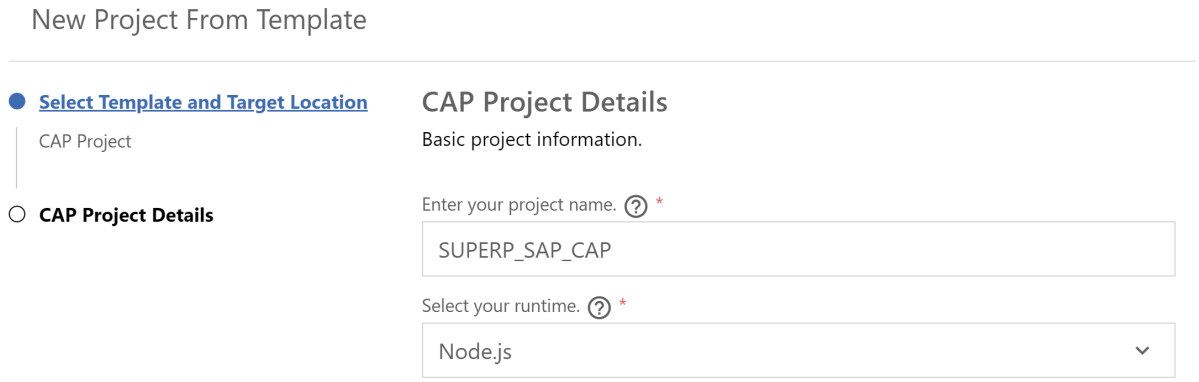
1. Creëer een nieuw project via een Template:



2. Voer een Projectnaam in en zorg dat de runtime op Node.js staat. Klik onderin op Finish.



3. Maak onder de DB folder een bestand aan ‘schema.cds’. Hierin staan de definities van je entiteiten. Maakt je eerste entiteit aan door deze code in het ‘schema.cds’ bestand te plakken:

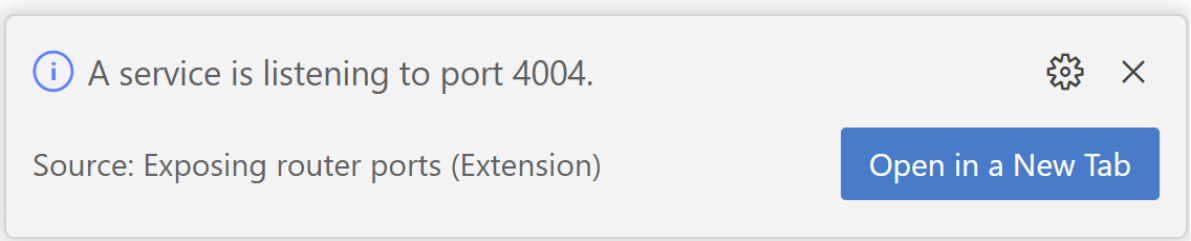
```
namespace nl.superp.cap;
entity Employees {
  key ID: Integer64;
  name : String;
  jobTitle : String;
  email : String;
 }
```

4. Maak onder de SRV folder een bestand aan ‘declaration-service.cds’. Hierin staat de definitie van de service bedoeld voor de medewerkers, voor het indienen en wijzigen van hun declaraties. Maak deze service aan door deze code in het ‘declaration-service.cds’ bestand te plakken:

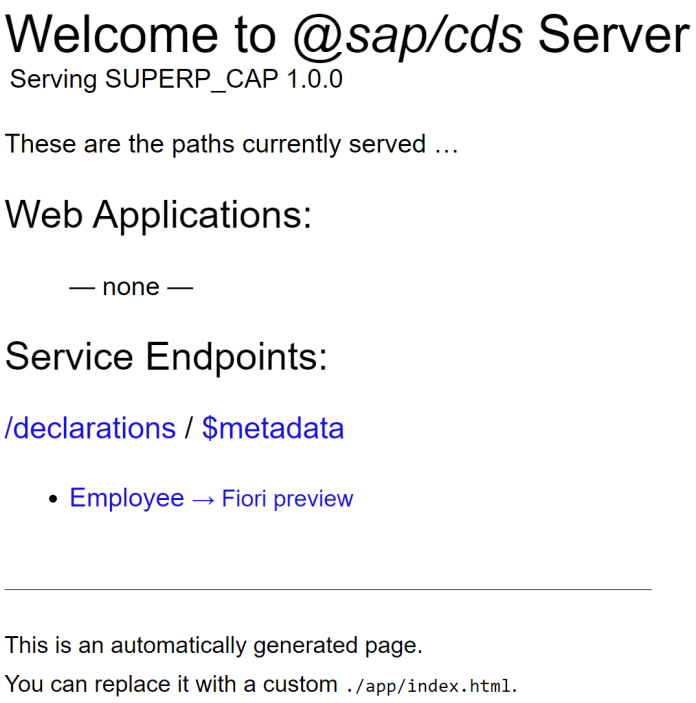
```
using nl.superp.cap from '../db/schema';

service Declarations {
      entity Employee as projection on cap.Employees;

}
```

5. Gebruik CTRL-J om het Panel te openen en ga naar het tabblad ‘Terminal’

6. Compileer, deploy en start het project door het command ‘cds watch’ in te voeren. Voor alle entiteiten worden nu tabellen in een in-memory database gemaakt. En de service definities worden services.
7. Zodra je deze popup ziet, druk je op de blauwe knop



8. In de webpage die wordt geopend klik je op de ‘Employees’ link om de GET van je gecreëerde service aan te roepen:



9. Je krijgt nu een lege resultset, omdat we nog geen data in de database hebben staan. Dit gaan we oplossen door een CSV file toe te voegen waarmee we elke keer dat de applicatie start de in-memory database vullen met data.

   Maak hiervoor een folder ‘csv’ aan onder de ‘db’ folder. Vervolgens maak je in de ‘csv’ folder een bestand aan met de naam ‘nl.superp.cap.Employees.csv’ (<namespace>.<entiteitnaam>).

   Vul dit bestand met data:

```
ID,name,jobTitle,email
2, Robin Laurenssen, UI5-er, r.laurenssen@superp.nl,
3, Frank Hammen, ABAP-er, f.hammen@superp.nl,
4, Ronald Groennou, Manager, r.groennou@superp.nl,
```

10. Het ‘cds watch’ command wat je eerder hebt gebruikt om de applicatie te starten zou het nieuwe bestand moeten herkennen en oppakken en de applicatie opnieuw moeten deployen. Klik weer op de blauwe ‘open in a new tab’ knop en start weer de ‘Employees’ link. Deze keer zie je de data uit je CSV bestand terug in de resultset.
11. Om bepaalde veel gebruikte velden niet bij iedere entiteit opnieuw te hoeven invoeren, heeft SAP enkele herbruikbare blokken (Aspects) beschikbaar gemaakt. We gaan dit nu gebruiken om created by/at en changed by/at toe te voegen. Ook gaan het ID vervangen door een ‘Canonic Key’.

    Hiervoor openen we weer de ‘schema.cds’ en voegen we deze regel toe onder de namespace declaratie:

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

    Vervolgens dienen we in de entiteit aan te geven dat we een CUID en de created/changed/ by/at velden willen toevoegen:

    entity Employees : cuid, managed {

    *vervang de em*ployees tabel data met het volgende:

    ID,name,jobTitle,email
    2dhg8f34-h34d-5321-5n58-fdshj89vvhfs, <Jouw naam>, <jouw rol>, <jouw email>,
    634a87c3-48d7-4401-84a3-e9e31baf603a, Robin Laurenssen, UI5-er, r.laurenssen@superp.nl,
    cc98bcb6-5edf-4730-91ff-ac586c4b9a96, Frank Hammen, ABAP-er, f.hammen@superp.nl,
    def9d635-8e28-4a7c-b4dc-fd93aa416441, Ronald Groennou, Manager, r.groennou@superp.nl,

    De ID waardes zijn veranderd naar de CUID format.

12. Bekijk je weer de resultset en je ziet de nieuwe ID waardes en created/changed by/at velden.

13. Nu gaan we een 2e entiteit toevoegen in de ‘schema.cds’

```
entity Declarations: cuid, managed {
description: String;
date: Date; }
```

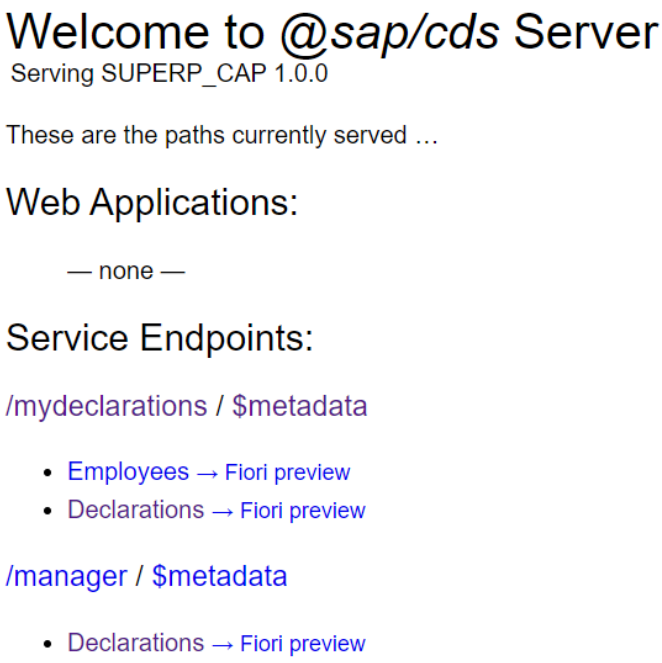
14. Deze entiteit voegen we ook toe aan onze Declarations service in ‘declaration-service.cds’

```
entity Declarations as projection on cap.Declarations;
```

15. Maak net zoals voor de employees een .csv file met mockup declaraties voor de Declaratie entiteit met de volgende data:

```
ID, description, amount, date,
41a233ee-0f62-48cc-a011-6714f26649b7, Tanken, 123.10, 16-06-2023
deed8520-fdee-4109-9b44-a80d5d329659, Lunch, 66.23, 16-06-2023
4fdf3fdf-f34e-6459-1fd4-fdskjl48973h, Diner, 239.23, 16-06-2023
```

16. Na het opslaan worden de database tabellen en service aangemaakt en is de Declarations endpoint zichtbaar binnen de service



Test de Declarations endpoint, die zou nu een lijst van declaraties moeten teruggeven.

17. In de volgende stap gaan we een relatie leggen tussen de Employees en Declarations entiteiten. Pas je ‘schema.cds’ aan naar:

```
namespace nl.superp.cap;

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

      entity Employees : cuid, managed {
        name : String;
        jobTitle : String;
        declaration: Association to many Declarations on
declaration.employee = $self;
```

```
        email: String;
    }

    entity Declarations: cuid, managed {
      description: String;
      amount: Decimal(10,2);
      date: Date;
      employee: Association to Employees;
    }
```

Deze associatie willen we ook terug zien in de data:
Voeg een ‘employee_ID’ kolom toe aan de Declarations CSV file door deze te vervangen met hetvolgende:

```
ID, description, amount, date, employee_ID
41a233ee-0f62-48cc-a011-6714f26649b7, Tanken, 123.10, 16-06-2023, 634a87c3-48d7-4401-84a3-e9e31baf603a
deed8520-fdee-4109-9b44-a80d5d329659, Lunch, 66.23, 16-06-2023, cc98bcb6-5edf-4730-91ff-ac586c4b9a96
4fdf3fdf-f34e-6459-1fd4-fdskjl48973h, Diner, 239.23, 16-06-2023, 2dhg8f34-h34d-5321-5n58-fdshj89vvhfs
```

De regels waarbij de UUID waarden in beide CSV’s gelijk zijn, zijn dan aan elkaar gekoppeld.

18. Open wederom de resultset van het Employees endpoint en breidt de URL uit met ‘?$expand=declaration’ (…declarations/Employee?$expand=declaration)

    Te zien is hoe het relationele kenmerk ‘declaration’ van de Employees entiteit niet voorkomt in die database tabel, maar wel een ‘navigatie kenmerk’ is geworden in onze oData service.

19. Vervolgens gaan we een actie op de Declaratie entiteit toevoegen om deze goed te keuren. Voeg het onderstaande toe aan de ‘manager-service.cds’:

```
service ManagerService {

      entity Declarations as  projection on cap.Declarations
      actions {
            action approveDeclaration ( ) returns Boolean;
    };

}
```

20. Voor de implementatie van deze Actie maak je een nieuwe bestand aan naast de manager-service.cds met de naam manager-service.js (CAP koppelt deze ook aan elkaar a.d.h.v. de gedeelde naam, net als bij de entiteiten en de CSV bestandsnamen) Hierin zet je de volgende code:

```
const cds = require("@sap/cds");


module.exports = function(srv){
const {DeclarationS} = srv.entities('ManagerService');
srv.on('approveDeclaration', Declarations, async(req) => {
  console.log(req.params);
        await UPDATE(Declarations).set({ isApproved: true }).where({ ID:
req.params });
} );
 }
```

In deze code haak je in op het 'approveDeclaration' event. Door middel van CQL (query language) statement (UPDATE(entity).set ……) wordt het isApproved veld op true gezet

21. Het starten van een Action gebeurt middels een POST operation. Dit is middels een browser erg lastig te testen. Gelukkig is er binnen SAP BAS een handige manier om dit te doen. Maak in je SRV (service) folder een 'request.http' file aan en zet de volgende inhoud hier in:

```
### GET all declarations as logged-in manager

GET http://localhost:4004/manager/Declarations
Content-Type: application/json


### POST a declaration approvement as logged-in manager
POST http://localhost:4004/manager/Declarations(deed8520-fdee-4109-9b44-
a80d5d329659)/approveDeclaration
Content-Type: application/json
```

Zorg dat je service draait en klik op 'Send request' van de GET Declarations endpoint:

In het resultaat zie je dat de isApproved van de declaraties nog op null staan.

Klik nu ook op 'Send Request' net boven de POST regel, hier zou je een HTTP 2xx code op terug moeten krijgen. Zodra je nu weer op de 'Send Request' bij de GET klikt, zou isApproved

van de declaratie, waarvan we de ID hebben meegeven in de request, op TRUE moeten staan.


```
{
  "ID": "deed8520-fdee-4109-9b44-a80d5d329659",
  "createdAt": "2023-07-04T13:09:05.470Z",
  "createdBy": "anonymous",
  "modifiedAt": "2023-07-04T13:09:16.817Z",
  "modifiedBy": "anonymous",
  "description": "Lunch",
  "amount": 66.23,
  "date": "16-06-2023",
  "employee_ID": "cc98bcb6-5edf-4730-91ff-ac586c4b9a96",
  "isApproved": true
}
```


22. Momenteel kan iedere gebruiker in principe deze manager service gebruiken, en dat mag natuurlijk niet, daarom voegen we nu een autorisatie check hieraan toe o.b.v. de gebruikers rol ‘manager’.

    In een productie deployed CAP app zouden deze rollen in de JWT (JSON web tokens) tokens zitten, deze worden bij ieder request mee verstuurd naar de CAP server. Ze bevatten de informatie over de gebruiker en dienen ook als authenticatie bewijs.

    Tijdens het opzetten van een CAP applicatie is het vaak, zeker in het begin van het development proces, niet nodig om gebruik te maken van een daadwerkelijke productie ready JWT token service, met echte accounts. Daarom komt CAP met de volgende standaard test gebruikers waarmee het JWT token mechanisme nagebootst kan worden:

```
"users": {
    "alice": { "roles": ["admin", "cds.Subscriber"] },
    "bob":   { "roles": ["cds.ExtensionDeveloper", "cds.UIFlexDeveloper"] },
    "carol": { "roles": ["admin", "cds.Subscriber", "cds.ExtensionDeveloper", "cds.UIFlexDeveloper"] },
    "dave":  { "roles": ["admin", "cds.Subscriber"] },
    "erin":  { "roles": ["admin", "cds.Subscriber", "cds.ExtensionDeveloper", "cds.UIFlexDeveloper"] },
    "fred":  { },
    "me":    { },
    "*": true //> all other logins are allowed as well
  }
}
```

Te zien is hoe alice de admin rol heeft, en fred niet. Dus als wij bijvoorbeeld een service level restrictie zouden aanbrengen o.b.v. de rol genaamd ‘admin’, dan zou fred deze service niet kunnen gebruiken.

We gaan nu onze eigen mockup gebruikers met rollen toevoegen. Dit doen we door het volgende stukje in de package.json te zetten.

```
  "cds": {
    "requires": {
      "auth": {
        "kind": "mocked",
        "users": {
          "<jouw naam>": {
            "roles": [],
            "userAttributes": {"email": "<jouw email>"}
          },
          "robin": {
            "roles": [],
            "userAttributes": {"email": "r.laurenssen@superp.nl"}
          },
          "frank": {
            "roles": [],
            "userAttributes": {"email": "f.hammen@superp.nl"}
          },
          "ronald": {
            "roles": [ "manager"],
            "userAttributes": {"email": "r.groennou@superp.nl"}
          }
        }
      }
    }
  }
```

Autorisaties kunnen zowel op service, entity als op property niveau worden vastgelegd. In ons geval hangen we hem aan de manager service. Door `@(requires:'manager')` toe te voegen aan de service definitie. Dit mag je doen in je eigen manager-service, zodat die er dan zo uit ziet:

```
using nl.superp.cap from '../db/schema';

service ManagerService @(requires:'manager') {
    entity Declarations as projection on cap.Declarations
    actions {
      action approveDeclaration () returns Boolean;
    }
}
```

Test of de autorisatie restrictie werkt door de GET request van de manager service aan te roepen vanuit de requests.http file. Deze zouden nu geen data terug mogen geven.

Om de requests af te vuren met een mockup gebruiker in het request object, moet ‘Authorization: Basic ronald:’ hieraan worden toegevoegd. Zorg dat de de volgende twee requests in je requests.http

komen te staan:

```
### GET all declarations as logged-in manager
GET http://localhost:4004/manager/Declarations
Content-Type: application/json
Authorization: Basic ronald:

### POST a declaration approvement as logged-in manager
POST http://localhost:4004/manager/Declarations(deed8520-fdee-4109-9b44-
a80d5d329659)/approveDeclaration
Content-Type: application/json
Authorization: Basic ronald:
```

Test of deze de service wél aan mogen roepen.

22. Nu moeten we er nog voor zorgen dat de niet-managers alleen hun eigen declaraties kunnen inzien, aanpassen en verwijderen.

Dit gaan we doen door een koppeling te leggen tussen de gebruikersdata uit de requests (uit de JWT tokens) en de employee tabel, via het gedeelde kenmerk ‘email’.

Als je het nog niet had gedaan mag je de <jouw naam> stukjes in de package.json en Employees.csv vervangen met je eigen naam en email.

Annoteer daarna de declaratie service met de volgende restrictie, door deze in de service definitie direct onder de entiteiten te plaatsen:

```
annotate Declarations with @(restrict:[{
  grant:[
    'READ',
    'CREATE',
    'UPDATE',
    'DELETE'],
  to: 'employee',
  where: 'exists employee[email = $user.email]'
}]);
```

In kenmerk ‘grant’ staan de acties die op de entiteit gedaan mogen worden door de geautoriseerde gebruiker. De rollen die deze gebruiker hiervoor moet hebben staat in ‘to’ (optioneel). De ‘where’ (optioneel) is een filter conditie die nog meer controle geeft over de restrictie.
Hierin ligt in ons geval ook de koppeling met de employee tabel.

We gebruiken ‘exists’ omdat de email uit de employee tabel die we vergelijken met de email van de gebruiker, geen deel uitmaakt van Declaratie entiteit waar we de restrictie aan hebben gehangen. Hiermee kunnen we kenmerken van gekoppelde entiteiten zoals employee bereiken.

Test deze nieuwe restrictie door je eigen gebruiker toe te voegen aan een test request, zoals hieronder:

```
### GET declarations
GET http://localhost:4004/mydeclarations/Declarations
Authorization: Basic <jouw naam>:
```

23. BONUS OPDRACHT: Voeg nu zelf een ‘declineDeclaration’ action toe dit de isApproved weer op ‘false’ zet. Neem de aanroep ook weer op