Exercise 3 - Implement new Business Service with CAP Model

Inhaltsverzeichnis

1. Objectives

- Create an simple service
- Add mock data
- Test the service
- Test and Deploy the service
- We will use the CDS and CF CLI

2. Change dev space

- Start the cds dev Space

3. Create a new Application `ratings-srv`

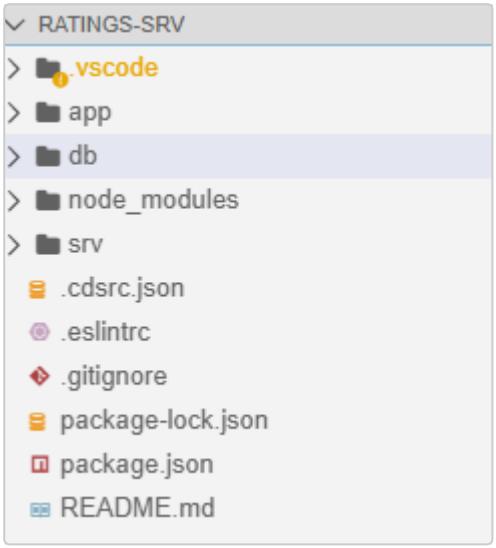
- Create New Folder `ratings-srv`
- Open New Terminal
- Type `cd ratings-srv`
- Type `cds init`

```
Problems 3        >_ user@workspaces-ws-bbk8w-deployment-79d85b4cdd-wd9mg: ~/projects/ratings-srv ×

user: projects $ cd ratings-srv/
user: ratings-srv $ cds init
[cds] - creating new project in current folder
done.

Find samples on https://github.com/SAP-samples/cloud-cap-samples
Learn about next steps at https://cap.cloud.sap/docs/get-started
user: ratings-srv $
```

- Run `npm install`

4. Folder structure

- app
- db
- srv
- node_modules



5. Step 1 - Create new schema

- We will create a 2 entities - Ratings and ProductRatings
- In DB Folder create a new file `schema.cds`
- Copy the code

```
namespace db;

entity Ratings {
    key RatingID  : Integer;
        ProductID : Integer;
        Name      : String;
        Date      : DateTime;
        Rating    : Integer;
}

entity ProductRatings {
    key ProductID : Integer;
        star1     : Integer;
        star2     : Integer;
        star3     : Integer;
        star4     : Integer;
        star5     : Integer;
        Ratings   : Composition of many Ratings
                        on ProductID = Ratings.ProductID;
}
```

6. Step 2 - Create a new service

- In srv folder create a new file `ratings-service.cds`
- We will use the previosly created entities from the `schema.cds` file - Ratings and Product Ratings
- We will run the service under the `/ratings` path with the command `@path : '/ratings'`
- We will add new properties that we will later calculate at runtime e.g. `star1Perc`
- Copy the code

```
using {db} from '../db/schema';

@path : '/ratings'

service RatingsService {

    entity Ratings        as projection on db.Ratings;

    entity ProductRatings as
        select from db.ProductRatings {
            *,
            null as count     : Integer,
            null as star1Perc : DecimalFloat,
            null as star2Perc : DecimalFloat,
            null as star3Perc : DecimalFloat,
            null as star4Perc : DecimalFloat,
            null as star5Perc : DecimalFloat
        };
}
```

7. Test the new service

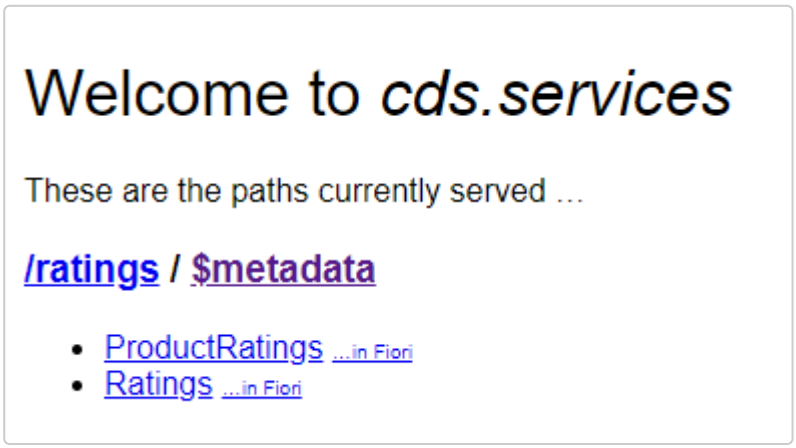
- In terminal run `cds watch`

```
user: ratings-srv $ cds watch

cds serve all --with-mocks --in-memory?
( watching: cds,csn,csv,ts,mjs,cjs,js,json,properties,edmx,xml,env... )

[cds] - model loaded from 2 file(s):

  db/schema.cds
  srv/ratings-service.cds

[cds] - using bindings from: { registry: '~/.cds-services.json' }
[cds] - connect to db > sqlite { database: ':memory:' }
/> successfully deployed to sqlite in-memory db

[cds] - connect to messaging > local-messaging {}
[cds] - serving RatingsService { at: '/ratings' }

[cds] - launched in: 1161.080ms
[cds] - server listening on { url: 'http://localhost:4004' }
[ terminate with ^C ]

>
```

A service is listening to port 4004.

Open in New Tab

- At popup windows choose "Open in New Tab" - out service is up and running



*Add `/ratings/$metadata` at the end of URL to show the metadata

8. Add some mock data

- Mock data can be added with `.csv` files
- Files have to be named according the Entities with the corresponding namespace
- In `db` create the `data` directory
- Create files `db.Ratings.csv` and `db.ProductRatings.csv`
- Content of `db.Ratings.csv`

```
RatingID;ProductID;Name;Date;Rating
1;1;John;2020-01-01T17:00:00Z;3
2;1;Mary;2020-01-03T17:00:00Z;5
3;1;Bart;2020-01-03T17:00:00Z;3
4;1;Eva;2020-01-05T17:00:00Z;4
5;1;Garry;2020-01-16T17:00:00Z;2
6;1;Michele;2020-01-18T17:00:00Z;5
7;2;John;2020-01-01T17:00:00Z;3
8;2;Mary;2020-01-03T17:00:00Z;4
9;2;Bart;2020-01-03T17:00:00Z;2
10;2;Eva;2020-01-05T17:00:00Z;1
11;2;Jane;2020-01-05T17:00:00Z;3
12;2;Michele;2020-01-18T17:00:00Z;3
```

- Content of `db.ProductRatings.csv`

```
ProductID;star1;star2;star3;star4;star5
1;0;1;2;1;2
2;1;1;3;1;0
```

- With `cds watch` still running you will see the service is filled from the 2 files

```
[cds] - using bindings from: { registry: '~/.cds-services.json' }
[cds] - connect to db > sqlite { database: ':memory:' }
 > filling db.ProductRatings from db/data/db.ProductRatings.csv
 > filling db.Ratings from db/data/db.Ratings.csv
/> successfully deployed to sqlite in-memory db
```

- In browser try the path `/ratings/ProductRatings(1)?$expand=Ratings`

```
{
    "@odata.context": "$metadata#ProductRatings(Ratings())/$entity",
    "ProductID": 1,
    "star1": 0,
    "star2": 1,
    "star3": 2,
    "star4": 1,
    "star5": 2,
    "count": null,
    "star1Perc": null,
    "star2Perc": null,
    "star3Perc": null,
    "star4Perc": null,
    "star5Perc": null,
    "Ratings": [
        {
            "RatingID": 1,
            "ProductID": 1,
            "Name": "John",
            "Date": "2020-01-01T17:00:00Z",
            "Rating": 3
        },
        {
            "RatingID": 2,
            "ProductID": 1,
            "Name": "Mary",
            "Date": "2020-01-03T17:00:00Z",
            "Rating": 5
        },
```

9. Calculate the `star*Perc` properties

- Create a new file `ratings-service.js`
- Copy the code

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

module.exports = cds.service.impl(async (service) => {
  const { Products } = service.entities;

  service.after("READ", "ProductRatings", (context, req) => {
    if (context.length === 0) {
      context.push({
        ProductID: req.data.ProductID,
        star1: 0,
        star2: 0,
        star3: 0,
        star4: 0,
        star5: 0,
        count: 0,
        star1Perc: 0,
        star2Perc: 0,
        star3Perc: 0,
        star4Perc: 0,
        star5Perc: 0,
      });
    } else {
      context.map((e) => {
        e.count = e.star1 + e.star2 + e.star3 + e.star4 + e.star5;
        e.star1Perc = (e.star1 / e.count) * 100;
        e.star2Perc = (e.star2 / e.count) * 100;
        e.star3Perc = (e.star3 / e.count) * 100;
        e.star4Perc = (e.star4 / e.count) * 100;
        e.star5Perc = (e.star5 / e.count) * 100;
      });
    }
  });
});
```

10. Test the service

- In browser try the path `/ratings/ProductRatings(1)?$expand=Ratings`


```
{
    "@odata.context": "$metadata#ProductRatings(Ratings())/$entity",
    "ProductID": 1,
    "star1": 0,
    "star2": 1,
    "star3": 2,
    "star4": 1,
    "star5": 2,
    "count": 6,
    "star1Perc": 0,
    "star2Perc": 16.666666666666664,
    "star3Perc": 33.33333333333333,
    "star4Perc": 16.666666666666664,
    "star5Perc": 33.33333333333333,
    "Ratings": [
        {
            "RatingID": 1,
            "ProductID": 1,
            "Name": "John",
            "Date": "2020-01-01T17:00:00Z",
            "Rating": 3
        },
        {
            "RatingID": 2,
            "ProductID": 1,
            "Name": "Mary",
            "Date": "2020-01-03T17:00:00Z",
            "Rating": 5
        },
```


- Out `star*Perc` entities are beying calculated

11. Deployment

- To deploy the service we have to modify our `package.json` file and run some commands in terminal
- Add to `package.json` file

```
"cds": {
  "requires": {
    "db": {
      "kind": "sql"
    }
  }
}
```

- In terminal run:
  - run `npm add @sap/hana-client --save`
  - run `cf login`
  - you should be now logged in your cloud account

```
Password: user: ratings-srv $ cf login
API endpoint: https://api.cf.eu10.hana.ondemand.com

Email: vladimir.forner@gmail.com

Password:
Authenticating...
OK

Targeted org 72e0e727trial

Select a space:
1. dev
2. fiori

Space (enter to skip): 1
Targeted space dev



API endpoint:   https://api.cf.eu10.hana.ondemand.com (API version: 3.86.0)
User:           vladimir.forner@gmail.com
Org:            72e0e727trial
Space:          dev
user: ratings-srv $
```

- Now we need to create the HANA service
  - run `cf create-service hanatrial hdi-shared ratings-srv-db`
  - run `cds build --production`
  - run `cf push -f gen/db`
  - run `cf push -f gen/srv --random-route`
- This will take some minutes
  - After succesfull deployement you will get the address of your service

```
Waiting for app to start...

name:                ratings-srv-srv
requested state:     started
isolation segment:   trial
routes:              ratings-srv-srv-busy-oribi-wz.cfapps.eu10.hana.ondemand.com
last uploaded:       Mon 19 Oct 13:06:33 UTC 2020
stack:               cflinuxfs3
buildpacks:          nodejs

type:            web
instances:       1/1
memory usage:    128M
start command:   npm start
     state     since                  cpu    memory          disk           details
#0   running   2020-10-19T13:07:03Z   0.0%   156K of 128M    255.8M of 1G
```

- And you can test the service in browser

ratings-srv-srv-busy-oribi-wz.cfapps.eu10.hana.ondemand.com/ratings/ProductRatings(1)?$expand=Ratings

Apps | asem Inkasso - C_In... | Inkasso - Prozessdi... | abap | Schweizerisches Idi... | conversational ai | cap | sap k


```
{
    "@odata.context": "$metadata#ProductRatings(Ratings())/$entity",
    "ProductID": 1,
    "star1": 0,
    "star2": 1,
    "star3": 2,
    "star4": 1,
    "star5": 2,
    "count": 6,
    "star1Perc": 0,
    "star2Perc": 16.666666666666664,
    "star3Perc": 33.33333333333333,
    "star4Perc": 16.666666666666664,
    "star5Perc": 33.33333333333333,
    "Ratings": [
        {
            "RatingID": 1,
            "ProductID": 1,
            "Name": "John",
            "Date": "2020-01-01T17:00:00Z",
            "Rating": 3
        },
        {
            "RatingID": 2,
            "ProductID": 1,
            "Name": "Mary",
            "Date": "2020-01-03T17:00:00Z",
            "Rating": 5
        },
```