openSAP SAP Cloud Platform API Management

Additional Download

PUBLIC

TABLE OF CONTENTS

SYSTEM PREREQUISITES 3
CREATE BINDING TO PERSISTENCY SERVICE..... 4
EXTEND GENERATED CODE WITH PERSISTENCY SERVICE..... 4
TESTING YOUR API WITH PERSISTENCY SERVICE..... 9
EXTEND GENERATED CODE WITH CLOUD FOUNDRY ENVIRONMENTS.....10
DEPLOY GENERATED TO YOUR CLOUD FOUNDRY ENVIRONMENT 12
FURTHER READS 12

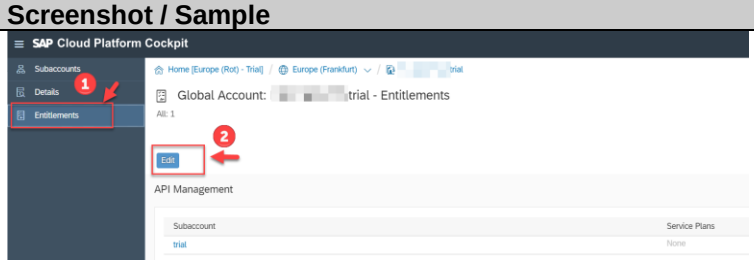
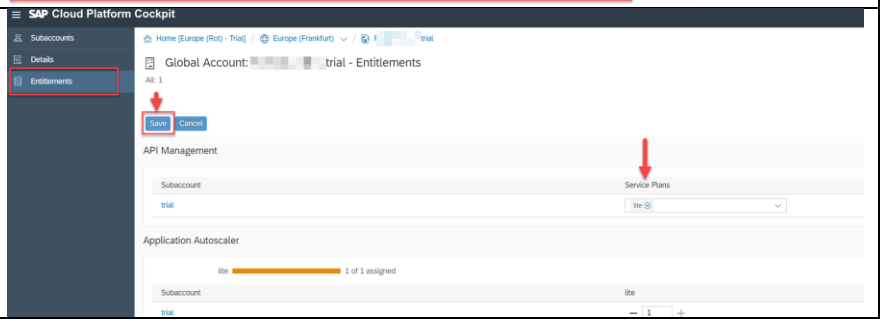
In this exercise, you'll explore the use of API first design and show how OpenAPI and the SAP API Designer can help develop your first micro service. The microservice you'll create would be for product catalog and product details like stock availability and price information.

SYSTEM PREREQUISITES

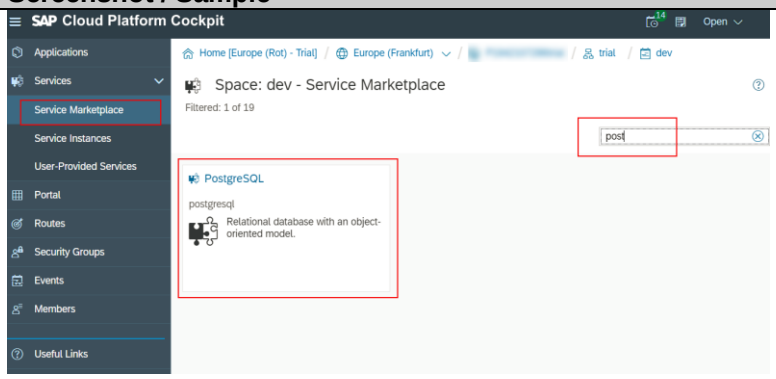
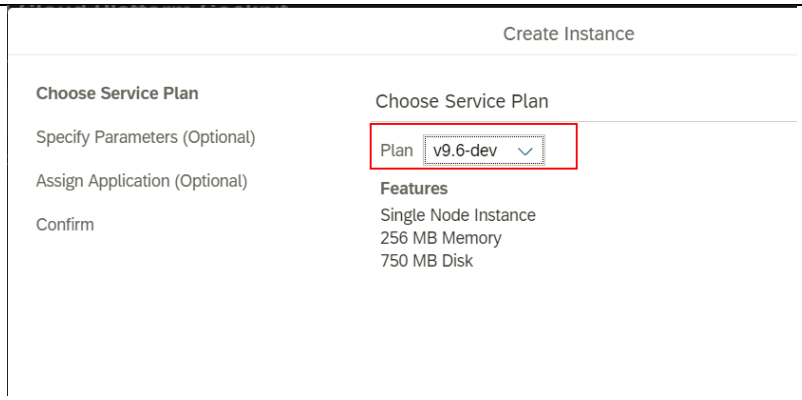
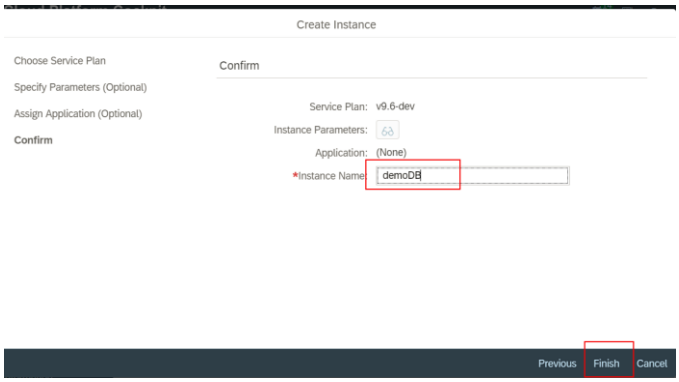
- Account on the SAP Cloud Platform
- Node JS version 6.12.3 or later installed (http://www.nodejs.org)
- Node Package Manager (NPM) 3.10 or later installed (should be automatically installed when you install Node JS)
- Visual Studio Code (https://code.visualstudio.com/) or another code editor
- PostgreSQL version 10.4 or later (https://www.postgresql.org/download/) for local testing
- Cloud Foundry Command Line Interface version 6.34 or later (https://docs.cloudfoundry.org/cf-cli/install-go-cli.html)

ENABLING POSTGRES PERSISTENCE SERVICE ENTITLEMENT

If in your SAP Cloud Platform trial tenant, PostgreSQL service or SAP Cloud Platform API Management is not available in the Service Market Place, enable these services from the Entitlement tab.
If the services are already available, then this section can be ignored.

Steps	Screenshot / Sample
Logon to SAP Cloud Foundry account, from Global Account select the **Entitlements** tab and click **Edit** button	
Search for **PostgreSQL** and then add 1 for the dev plan	
Search for API Management and then select the lite plan from the drop. Click on **Save** to persist the **Entitlement** changes	

CREATE BINDING TO PERSISTENCY SERVICE

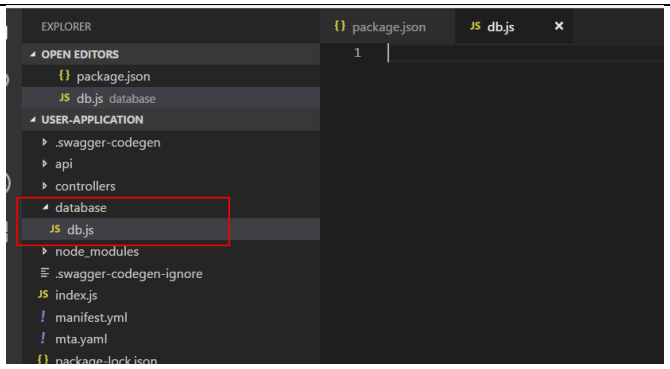
Steps	Screenshot / Sample
Logon to SAP Cloud Foundry account, from Services -> Service Marketplace search for **PostgreSQL** persistency service	
From the Instances tab, click on New Instance and create a service instance of plan **v9.6-dev**	
Enter the name of the MongoDB as **demoDB**	

EXTEND GENERATED CODE WITH PERSISTENCY SERVICE

Step	Screenshot/ Sample
Add dependency to pg library and cfenv library  package.json	

```
  "scripts": {
    "prestart": "npm install",
    "start": "node index.js"
  },
  "keywords": [
    "swagger"
  ],
  "license": "Unlicense",
  "private": true,
  "dependencies": {
    "connect": "^3.2.0",
    "js-yaml": "^3.3.0",
    "swagger-tools": "0.10.1",
    "sequelize": "~4.38.0",
    "pg": "~7.4.3",
    "pg-hstore": "~2.3.2",
    "cfenv": "~1.1.0"
  }
}
```

Create a folder named **database** and add a file named **db.js**



Open **db.js** and add in the following snippet to connect to the postgresql DB named **demoDB** and create the product schema as per the Open API specification file

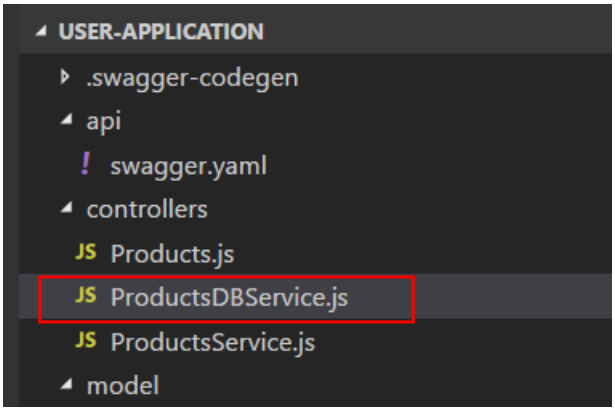
db.js

Note:- For the local testing, the table named demoDB would have to be created on the local Postgres DB. In the connection string user:pass would have to be set to DB user created by you for your local DB

```
'use strict';

const Sequelize = require('sequelize'), appEnv =
require("cfenv").getAppEnv();

let dbUrl = appEnv.getServiceURL('demoDB', {
  pathname: "dbname",
  auth: ["username", "password"]
});

if (!dbUrl) {
  dbUrl =
'postgres://user:pass@localhost:5432/demoDB';
}

const sequelize = new Sequelize(dbUrl);
```

```
sequelize.authenticate()
      .then(() => console.log('Connection has
been established successfully.'))
      .catch(err => console.error('Unable to
connect to the database:', err));

const product = sequelize.define('product', {
  id: { type: Sequelize.STRING, primaryKey: true
},
  name: Sequelize.STRING,
  description : Sequelize.STRING,
  price: Sequelize.JSON,
  availability: Sequelize.JSON
});

product.sync();

exports.product = product;
```

Create a controller file(named **ProductsDBService.js**)for handling DB calls like reading of product catalogs, creating products, reading product details



Add the snippet for DB handling in **ProductsDBService.js** file

ProductsDBService.js

```
'use strict';
var entityModel =
require("../database/db.js").product;

function validate(res, validateFunc) {
    const error = validateFunc();
    if (error != null) {
        setErrorResponse(res, 400, error.code,
error.message);
        return false;
    }
    return true;
  }
```

```
function validTopSkip(text, queryName) {
    let error = null;
    try {
        const value = parseInt(text);
        if (isNaN(value) || value < 0) {
            error = getError("INVALID_INPUT",
"Incorrect format for " + queryName + " argument
'" + text + "'. Provide a value positive integer
value");
        }

    } catch (e) {
        error = getError("INVALID_INPUT",
"Incorrect format for " + queryName + " argument
'" + text + "'. Provide a value positive integer
value");
    }
    return error
}


function getError(code, message) {
    return {
        code: code,
        message: message
    };
}

function setResponse(res, data, status) {
    res.setHeader('Content-Type',
'application/json');
    res.statusCode = status || 200;
    res.end(JSON.stringify(data));
}

function setErrorResponse(res, status, code,
message) {
setResponse(res, getError(code, message),
status);
}


exports.productsGET = (args, res, next) => {
    let skip = args.$skip;
    var query = {};
```

```
    if (skip && skip.value) {
        if (!validate(res,
validTopSkip.bind(null, skip.value, "$skip"))) {
            return;
        }
        query.offset = parseInt(skip.value);
    }

    let top = args.$top;
    if (top && top.value) {
        if (!validate(res,
validTopSkip.bind(null, top.value, "$top"))) {
            return;
        }
        query.limit = parseInt(top.value);
    }


    entityModel.findAll(query)
    .then(entity => setResponse(res,entity,200))
    .catch(error =>
setErrorResponse(res,500,"READ_FAILURE",
error.message));
}

exports.productsPOST = (args, res, next) => {
    entityModel.create(args.payload.value)
    .then(entity => setResponse(res,entity,201))
    .catch(error =>
setErrorResponse(res,500,"CREATE_FAILURE",
error.message));
}

exports.productsIdGET = (args, res, next) => {
    entityModel.findById(args.id.value)
    .then(entity => {
        if(!entity){
            setErrorResponse(res, 404,
"RESOURCE_NOT_FOUND", "Requested Product " + id +
" not found.");
        } else {
            setResponse(res,entity,200);
        }
    })
    .catch(error =>
```

<table>
<tr><td></td><td>

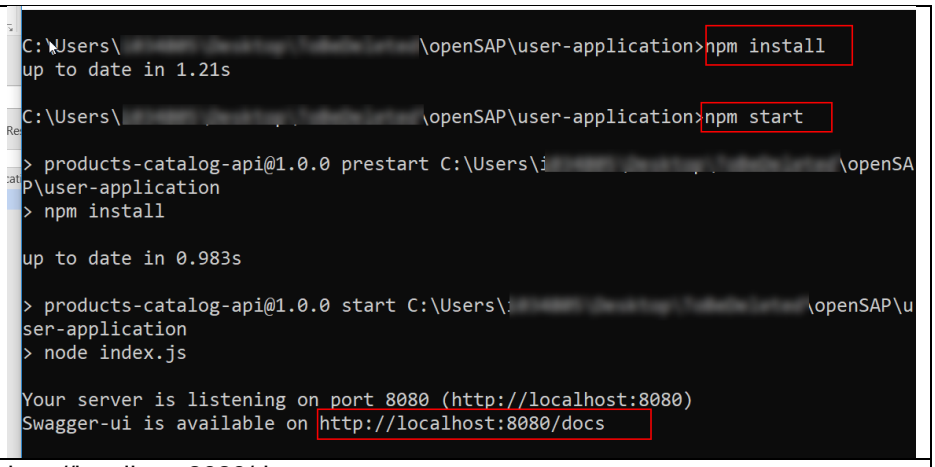
```
setErrorResponse(res,500,"READ_FAILURE",
error.message));
}

exports.productsIdDELETE = (args, res, next) => {
    entityModel.findById(args.id.value)
    .then(entity => {
        if(!entity){
            setErrorResponse(res, 404,
"RESOURCE_NOT_FOUND", "Requested Product " + id +
" not found.");
        } else {
            entity.destroy()
            .then(() => {
                res.statusCode = 204;
                res.end();
            })
            .catch(error =>
setResponse(res,500,"DELETE_FAILURE",
error.message));
        }
    })
    .catch(error =>
setErrorResponse(res,500,"DELETE_FAILURE",
error.message));
}
```

</td></tr>
<tr><td>Connect the generated productservice interface to the controller JS</td><td>

</td></tr>
<tr><td>Snippet for connect Product.js to the newly created ProductsDBService.js controller</td><td>var Products = require('./ProductsDBService');</td></tr>
</table>

TESTING YOUR API WITH PERSISTENCY SERVICE

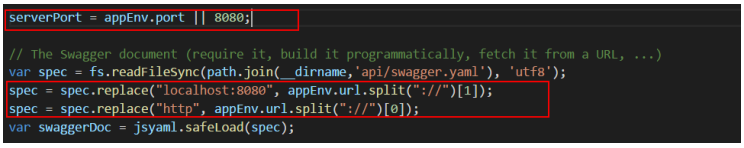
Step	Screenshot/ Sample

Open Command Line tool and then navigate to root folder of the downloaded project. Use command npm install to install all the dependent libraries Use command npm start to run the application	
Run the application	http://localhost:8080/docs

EXTEND GENERATED CODE WITH CLOUD FOUNDRY ENVIRONMENTS

Step	Screenshot/ Sample
Open index.js from the generated code and add in the following snippet to read the server port from CF application environment	`var appEnv = require("cfenv").getAppEnv();` `var serverPort = appEnv.port || 8080;`

Add the following snippet to dynamically updated the generated swagger.yaml file with CF application host name and port	`spec = spec.replace("localhost:8080",`
	`appEnv.url.split("://")[1]);`
	`spec = spec.replace("http", appEnv.url.split("://")[0]);`

Updated index.js file with changes highlighted	

Full index.js server for is added here for reference	`'use strict';`
	
	`var fs = require('fs'),`
	`path = require('path'),`
	`http = require('http');`

index.js

```
var app = require('connect')();
var swaggerTools = require('swagger-tools');
var jsyaml = require('js-yaml');
var appEnv = require("cfenv").getAppEnv();
var serverPort = appEnv.port || 8080;

// swaggerRouter configuration
var options = {
  swaggerUi: path.join(__dirname, '/swagger.json'),
  controllers: path.join(__dirname, './controllers'),
  useStubs: process.env.NODE_ENV === 'development' //
Conditionally turn on stubs (mock mode)
};

// The Swagger document (require it, build it
programmatically, fetch it from a URL, ...)
var spec =
fs.readFileSync(path.join(__dirname,'api/swagger.yaml'),
'utf8');
spec = spec.replace("localhost:8080",
appEnv.url.split("://")[1]);
spec = spec.replace("http", appEnv.url.split("://")[0]);
var swaggerDoc = jsyaml.safeLoad(spec);

// Initialize the Swagger middleware
swaggerTools.initializeMiddleware(swaggerDoc, function
(middleware) {

  // Interpret Swagger resources and attach metadata to
request - must be first in swagger-tools middleware
chain
  app.use(middleware.swaggerMetadata());

  // Validate Swagger requests
  app.use(middleware.swaggerValidator());

  // Route validated requests to appropriate controller
  app.use(middleware.swaggerRouter(options));

  // Serve the Swagger documents and Swagger UI
  app.use(middleware.swaggerUi());

  // Start the server
  http.createServer(app).listen(serverPort, function ()
{
```

	``` console.log('Your server is listening on port %d (http://localhost:%d)', serverPort, serverPort); console.log('Swagger-ui is available on http://localhost:%d/docs', serverPort); }); }); ```

DEPLOY GENERATED TO YOUR CLOUD FOUNDRY ENVIRONMENT

Step	Screenshot/ Sample
Open manifest.yml file from the generated code provide a unique name to your application say productcatalogs. Add in the services references to your persistency service	``` --- applications: - name: productcatalog memory: 512M buildpack: nodejs_buildpack services: - demoDB ```
Open Command Line tool and then navigate to root folder of the downloaded project.	
Set your Cloud Foundry Environment API endpoint	cf api https://api.cf.eu10.hana.ondemand.com
Login to your Cloud Foundry space and enter your Cloud Foundry user credentials	cf login
Deploy the application to Cloud Foundry tenant	cf push

FURTHER READS

- SAP Guidelines for REST API Harmonization
- OpenAPI Specification 2.0
- Accessing API Designer from SAP Cloud Platform Trial Account
- Documenting APIs with OpenAPI Specification in API Designer
- Video Tutorial for Creating and Documenting APIs in API Designer
- Blog on Develop and manage API first enterprise microservices with SAP Cloud Platform and API Management

Coding Samples

Any software coding or code lines/strings ("Code") provided in this documentation are only examples and are not intended for use in a production system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules for certain SAP coding. SAP does not warrant the correctness or completeness of the Code provided herein and SAP shall not be liable for errors or damages cause by use of the Code, except where such damages were caused by SAP with intent or with gross negligence.

www.sap.com/contactsap

THE BEST RUN

®