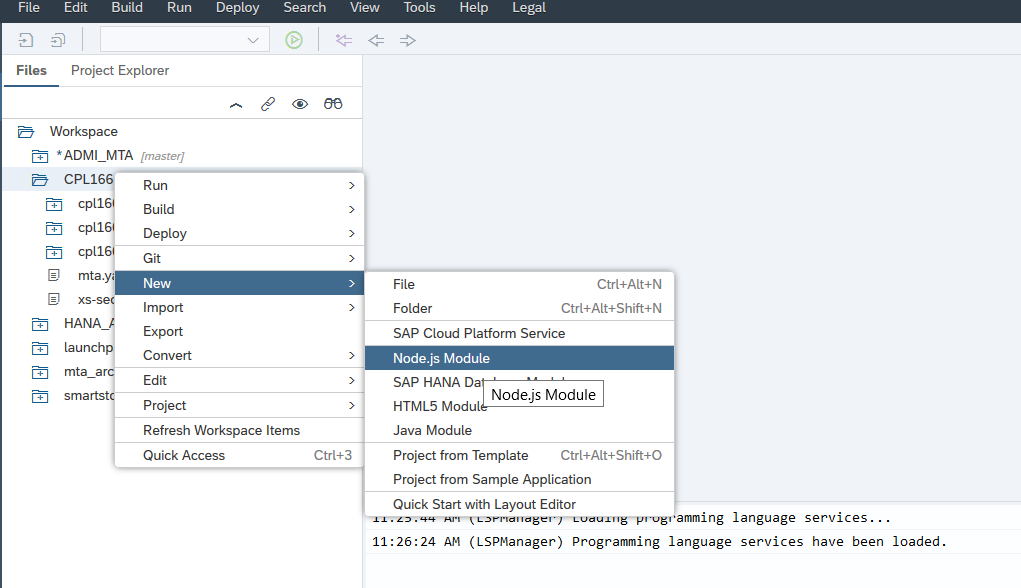
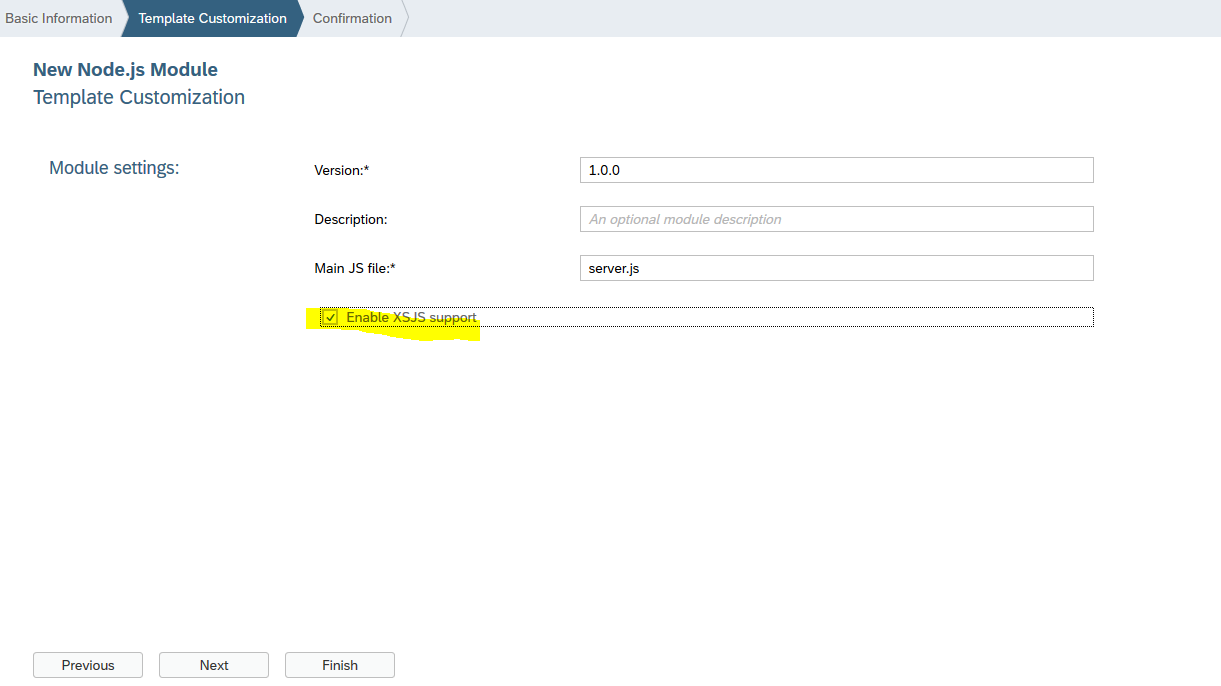
**Expose OData Service from MTA Application Node.JS Module**

**Purpose** – The main purpose of this module is to build the XSJS and XSODATA services which can be used to expose data model to the user interface.

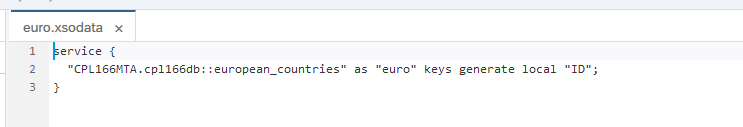
1. At first, on top of our earlier MTA project we will now add a Node.js module as shown below.



1. Right-click on the MTA project and select "New" -> "Node.js module" and name it as cpl166js and also Check the "Enable XSJS support" checkbox which is needed to make use of the OData provisioning capabilities via the XSJS library.



1. Inside workspace right-click on the lib folder inside the cpl166js module and create a new file called euro.xsodata. The .xsodata suffix enables the file to define OData services via declarative syntax.
2. Add the following code inside the .xsodata file created above and save it

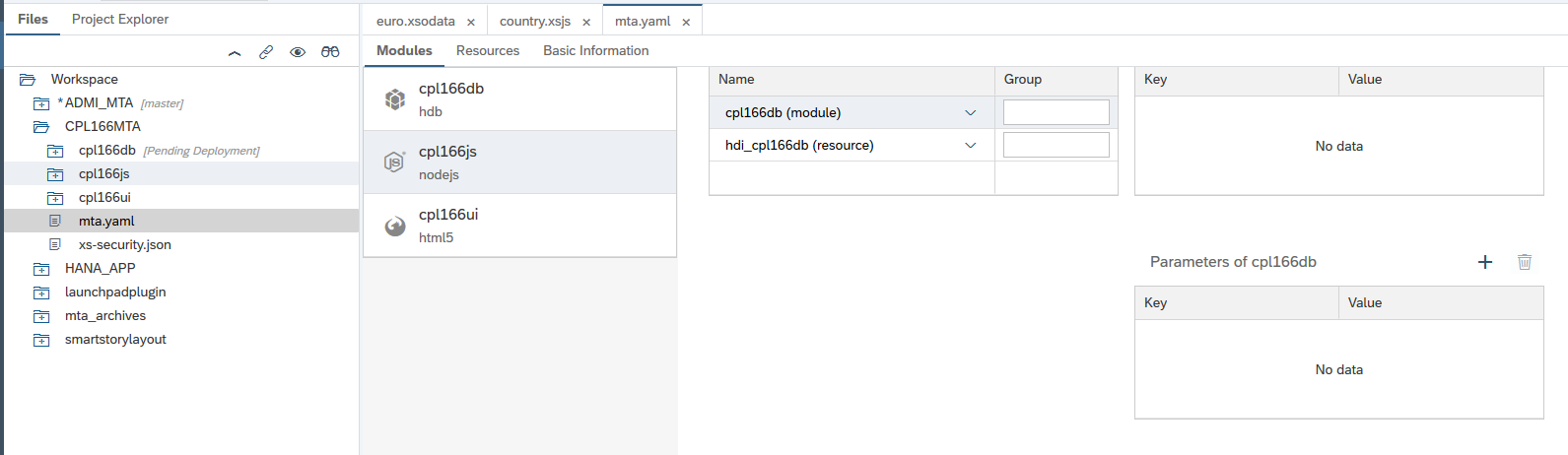


service {

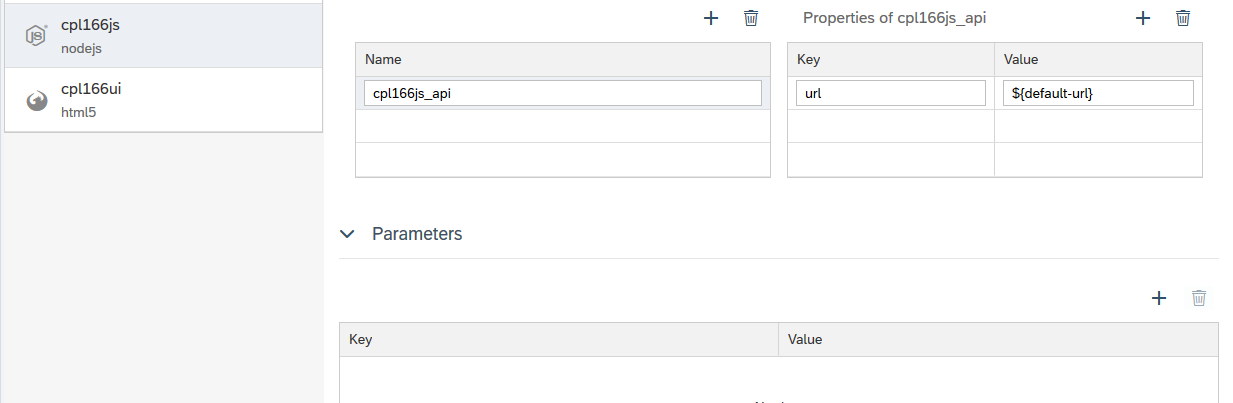
"CPL166MTA.cpl166db::european\_countries" as "euro" keys generate local "ID";

}

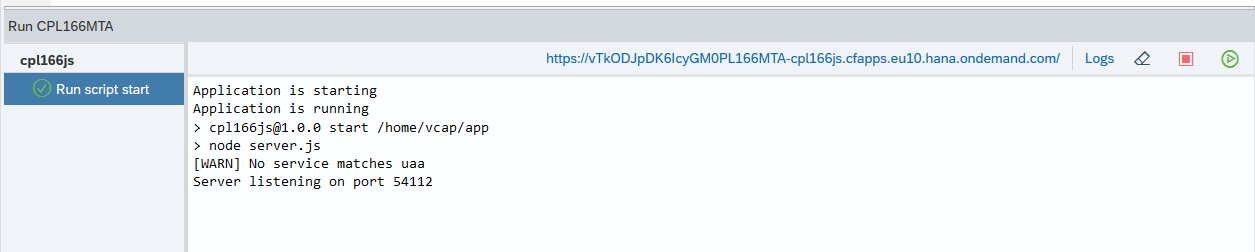
1. Using the above code we are basically defining a service named euro.xsodata with an entity called euro, based on the european\_countries Calculation View defined in earlier guide to create the MTA DB Module.
2. To test this service, we will have to define the dependency between the DB Module and Node.js module using MTA editor.
3. Open mta.yaml in the project root folder and select the cpl166js module. In the "Requires" section and add the dependencies to **cpl166db** and hdi\_cpl166db as shown below.



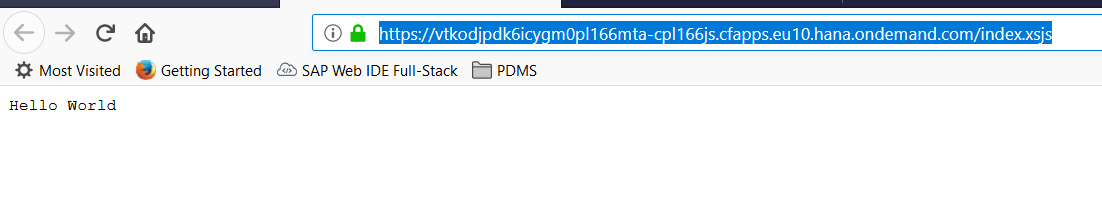
1. Also add the following configuration inside the "Provides" section. Define a name `cpl166js\_api` with a property called `url` and the value `${default-url}`, automatically gets generated in case XSJS support is enabled for the module. At the end save the changes.



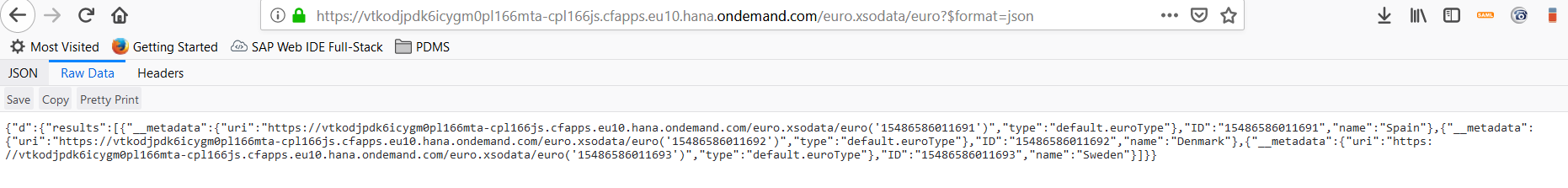
1. Right-click on the Node.js module cpl166js and choose "Run" -> "Run as Node.js Application" in the context menu. The command automatically builds the application, too and also shows the build/run progress in the run console at the bottom of the SAP Web IDE screen as shown below.



1. Once the application started successfully, a link will appear on top of the run console to navigate to it in the browser. Just click on it.



1. Now replace the URL segment /index.xsjs after the port number with /euro.xsodata/euro?$format=json, and you should see the result in an output as shown below



1. In the next step we will add custom logic inside our OData service. We will basically add logic by writing a JavaScript that receives inputs over a REST interface and writes the (validated) data into the database.
2. Right-click the cpl166js/lib/ folder, and create a new file with "New" -> "File". Enter country/country.xsjs. Add the following JavaScript code to the file, and save it.



Actual Code –

*function saveCountry(country) {*

*var conn = $.hdb.getConnection();*

*var output = JSON.stringify(country);*

*var fnCreateCountry = conn.loadProcedure("CPL166MTA.cpl166db::createCountry");*

*var result = fnCreateCountry({IM\_COUNTRY: country.name, IM\_CONTINENT: country.partof});*

*conn.commit();*

*conn.close();*

*if (result && result.EX\_ERROR != null) {*

*return {body : result,*

*status: $.net.http.BAD\_REQUEST};*

*}*

*else {*

*return {body : output,*

*status: $.net.http.CREATED};*

*}*

*}*

*var body = $.request.body.asString();*

*var country = JSON.parse(body);*

*// validate the inputs here!*

*var output = saveCountry(country);*

*$.response.contentType = "application/json";*

*$.response.setBody(output.body);*

*$.response.status = output.status;*

1. The above code snippet basically reads the values of two "GET" parameters, "name" and "continent" respectively, passes them to a function createCountry which ultimately write these values to the 'country' table, and also displays a confirmation message.
2. Right-click on the Node.js module cpl166js and choose "Run" -> "Run as Node.js Application" in the context menu. In your web browser, replace the URL element /index.xsjs with the string /country/country.xsjs?name=China&continent=Asia. Add some more countries like