**Assignment: First Application**

**Create your first JET application**

<http://www.oracle.com/webfolder/technetwork/jet/globalGetStarted.html>

<https://nodejs.org/en/>

**Requirements**:

* Node.js, make sure you have at least version 5.6 installed.
* JavaScript IDE. We recommend Visual Studio Code.

We are going to build a small application to evaluate all the courses in the Talent Launch. To set up an Oracle JET Application you can follow the instructions on the Oracle JET Get Started page:

* Install the JET CLI globally. Now you can start using all ‘ojet’ commands from the command line.
* Create your application. Choose the template ‘navdrawer’ for this assignment and give your application an appropriate name.
* Start your application. It opens in your browser and you are ready for development!
* Check out all the files that have been automatically created.

When you use one of the starting templates, Oracle JET provides you with a lot of boilerplate code and extended explanations of their use. Take some time to look at some of the options JET provides.

* Add some HTML to the index.html page. Check the result in the browser.
* Add some HTML to the dashboard page. Check the result in the browser.

The appcontroller.js contains data used in the template.

* Change the default name and e-mail of the application
* Rename the Incidents tab to Schedule. Make sure all references to Incidents in your application are replaced (Filenames, routing, viewModel and text in your browser)
* Add a new tab for Evaluation. Include a header “Evaluation form” in the html-file
* Delete the “about” tab

**Additional**

<http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html?component=iconfont&demo=iconfont>

* Change the icon in the tab for Evaluation.
* Change the size of your browser screen to the responsive features of your application. Maybe your browser provides built-in emulators to show your application in a mobile device. Take a look at these options.

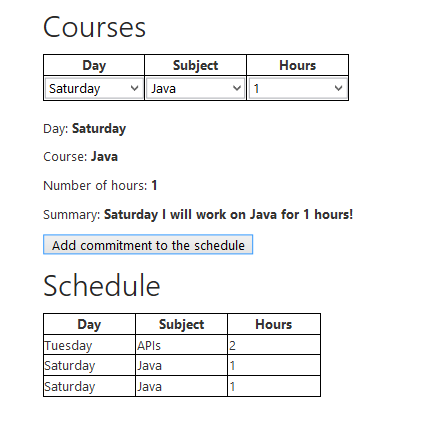
**Result: Project First Application on GitHub**

**Assignment: Knockout**

**Create a schedule**

<http://knockoutjs.com/>

After finishing the knockout tutorials, use your new knowledge to create a dynamic Talent Launch week schedule in your application in tab Schedule (use plain HTML and JavaScript). Create three drop-down selectors, fields which show your selection and a button to add courses to your schedule. Your result should look something like this:



**Additional**

* Show summary only when all options have been selected (use ko:if)

**Result: Project Knockout on GitHub**

**Assignment: JET Cookbook**

**Create a form**

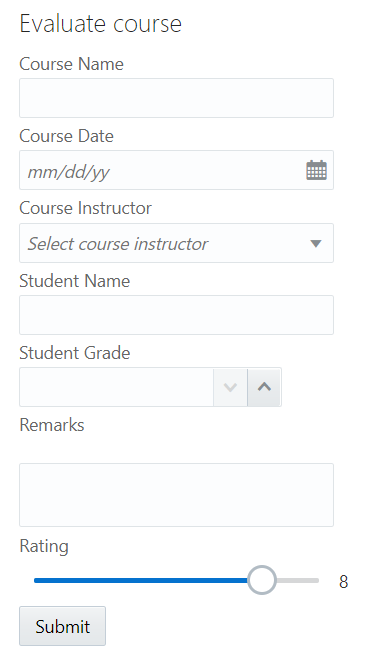
<http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html>

To familiarize yourself with the JET cookbook you are going to create a form with input fields and a button. For this you will use the “Forms” and “Controls” section of the Cookbook.

Checklist when using a component from the cookbook:

* Only use HTML from component (don’t copy all the code)
* Only use the JavaScript needed for the component
* Include component in define-block (watch out for doubling)

Make this form in the Evaluations tab of your application:

 The following components are used:

* HTML header
* Labels
* Input Text
* Input Date
* Select
* Input Number
* Text Area
* Slider
* Button

A few things to know when creating the form:

* Use a placeholder for the course instructor select.
* The rating is shown by using a read-only Input Text where an observable is reused

**Additional**

* Add other Cookbook Forms or Controls components to your page
* Add required fields and validation to your form

**Result: Project FormSetUp on GitHub**

**Assignment: JSON SERVER & POST**

**Submit information from the form**

**Install JSON-server**

<https://medium.com/codingthesmartway-com-blog/create-a-rest-api-with-json-server-36da8680136d>

The JSON-server will function as a mock server to store our data.

* Do an npm install of the JSON-server in your project
* Create a JSON-file in the top folder of your project: “evaluations.json”
* Add this JSON-object to your JSON-file: { “evaluations”: [] }
* In a second terminal run the JSON-server: json-server --watch evaluations.json
* Go to the URL where the JSON-server is running. You will see an empty array.

**Requests**

<https://www.getpostman.com/docs/v6/postman/sending_api_requests/requests>

* Copy the contents from submitForm.json (in the GitHub project) to your evaluations.json so you have some content.
* Do a GET-request to see the evaluations
* Create a POST-request in Postman
  + Add header Content-Type: application/json
  + Create body, you can copy an object from the GET-request, change the data and delete the id-row.
* When the POST-request works, you will see the newly added evaluation in your JSON-file and when you do a new GET-request.

This POST-request is what you want to happen when the submit-button is pressed in your form. The next step is to implement this in the application.

**Add POST-request to form**

* The button you have created in the previous exercise only opens an alert. Delete the alert and implement the POST-request.
  + Use the jQuery Ajax request: <http://api.jquery.com/jquery.ajax/>
  + Use: type, URL, data and contentType
  + For data: Recreate the body for the request with the observables
  + To test: Check in browser console -> Network if request comes back with status 201
* You will notice that the form doesn’t clear after pressing submit. You can use the .done() function to clear the observables. <https://api.jquery.com/deferred.done/>

**Additional**

* If you added any other components to your form add them to the POST-request body if needed
* Add a thank you message after the submit is done that disappears after a few seconds.

**Result: Project FormSubmit on GitHub**

**Assignment: Collection & Table**

**Create Table**

**Common Model**

<http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html?component=table&demo=ojCollectionTable>

<http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html?component=crud&demo=table>

The table-component needs to get data from the JSON-server in order to show it. JET has defined an easy way to do this with the Common Model Framework. In this framework you define a Model that represents a single record in the Collection. A table can use the collection to do CRUD-actions on the data.

* In the table we want to show all the data of all the evaluations. Use the “Customer” tab in your application for this table. Rename the tab.
* Define the Model for an evaluation.
* Parse data for easier use.
* Define the collection that uses the Model to fetch data.

**Table**

* Add the Table component from the cookbook (remember the Cookbook Checklist).
* Define the columns.
* Define the data source.
* Once you have your table up and running you might notice that a new entry through your form isn’t shown immediately in the table.

This is where the handleActivated function comes in. The handleActivated function is one of the functions that JET generates automatically in the navdrawer template (you will see it when opening a newly generated JS-file). It can be used to get data before the page is shown.

* Assign your data source inside the handleActivated. Your table will refresh.

**Additional:**

* Use the cookbook and the JS Doc of the table component to further configure your table.
  + Make the columns sortable / orderable / resizable
  + Add a Custom Cell Template for the rating (Visualizations – Rating Gauge)
* Keep the cell empty when remarks are shown as “undefined”.
* Use the same collection for a different representation of data in a different component (List View or Data Grid)

**Result: Project FormTable on GitHub**