Productive JavaScript Development: Oracle JET

Intro

Welcome

JavaScript-based, single-page applications (SPAs) are incredibly popular these days - and for good reason. This style of application provides developers with a great deal of flexibility and control, while end users enjoy the responsiveness and speed that comes with fewer full-page reloads.

To assist developers working on SPAs, Oracle developed the JavaScript Extension Toolkit (JET). Oracle JET is a collection of open-source JavaScript libraries, combined with a set of Oracle-contributed JavaScript libraries that make it as simple and efficient as possible to build applications that consume and interact with Oracle products and services.

HOL Overview

In this hands-on lab, you will be building an Oracle JET application from scratch. Here's an overview of what you'll be doing:

- Part 1: Creating a New Project
- Part 2: Adding Directories and Files
- Part 3: Adding Navigation and Routing
- Part 4: Adding a Model, Collection, and a Table
- Part 5: Adding a Model, Collection, and a Chart

Prerequisites

The instructions in this lab are based on the NetBeans IDE with the Oracle JET Support plugin. NetBeans is a free, cross-platform IDE that provides excellent support for building JET applications. For this lab, NetBeans will:

- download the JET source for the new project
- run the application via a built-in web server

If you don't want to use NetBeans, you'll need to adapt the instructions to meet your needs. The base distribution of Oracle JET can be downloaded here: http://www.oracle.com/technetwork/developer-tools/jet/downloads/index.html

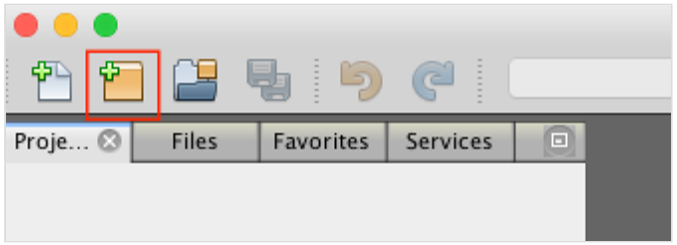
If you don't have a web server, you can run the app in FireFox as it allows JavaScript applications to run from the file system.

Part 1: Creating a New Project

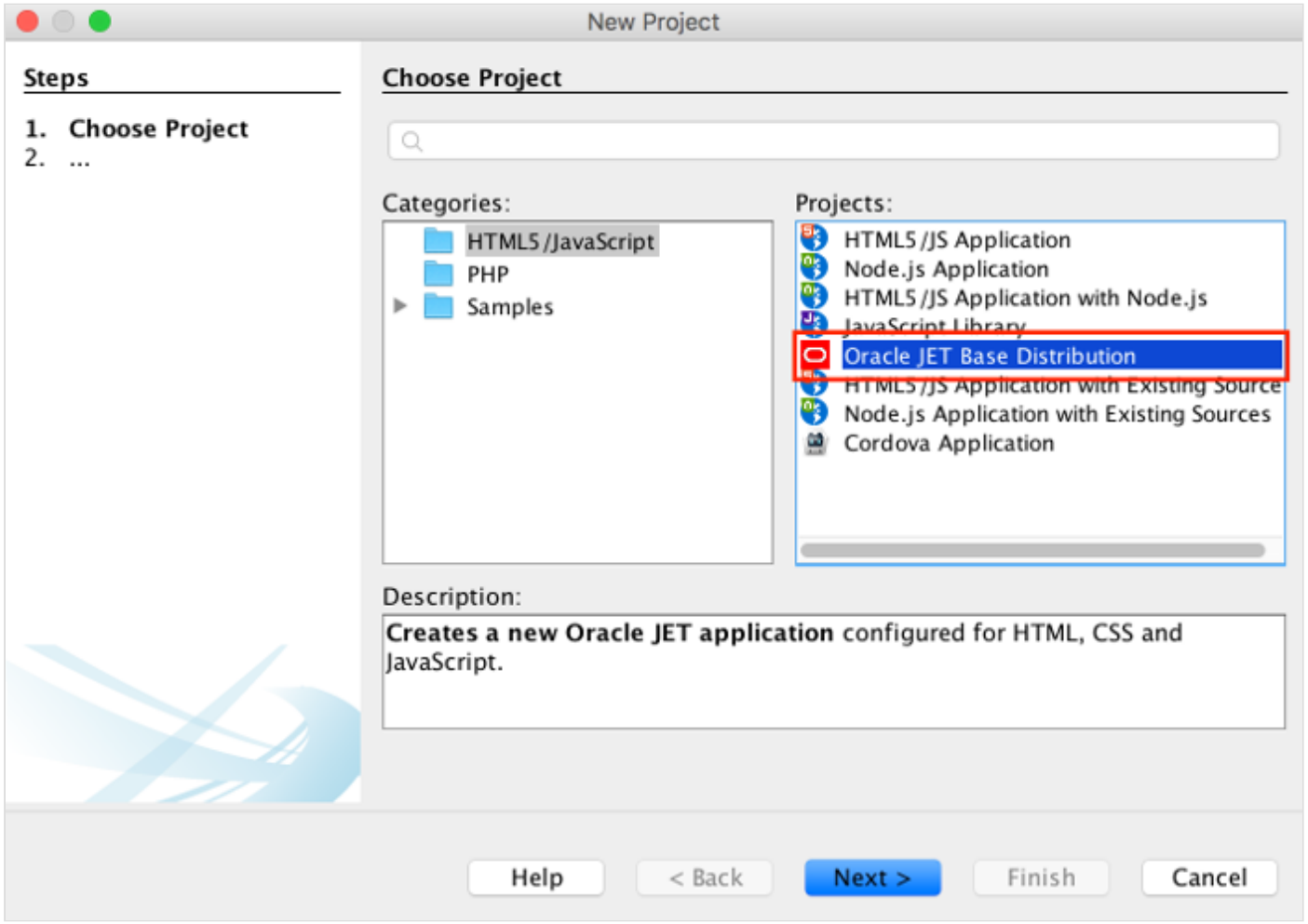
Let's kick things off by creating a new project in NetBeans using the base distribution of Oracle JET. The base distribution only includes JET and its dependencies which makes it a great place to start learning from the ground up.

Open the NetBeans IDE and follow these steps...

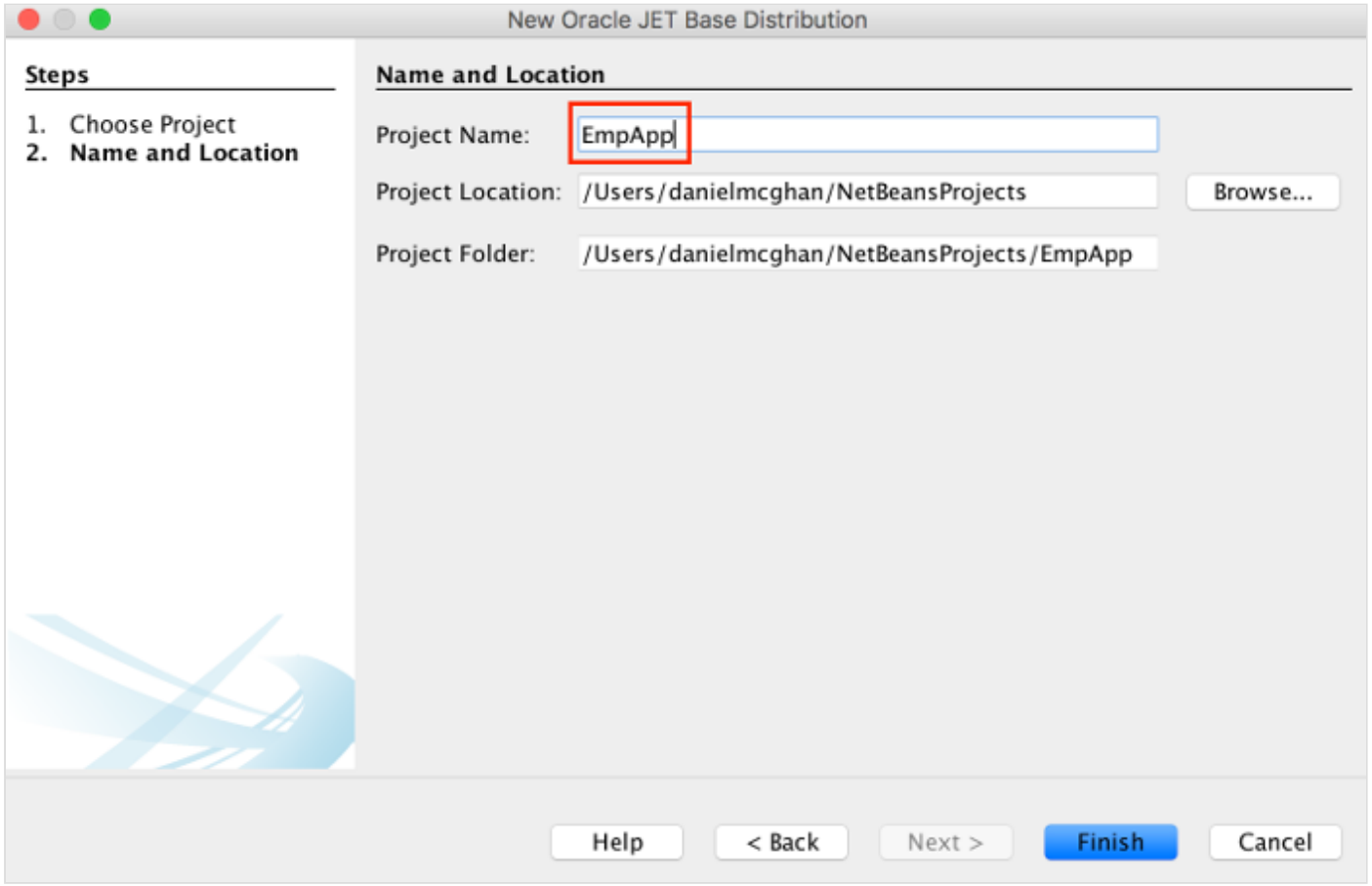
1. Click the **New Project** button.



2. Select **Oracle JET Base Distribution**. If you don't see the Oracle JET Base Distribution option, you may need to install the Oracle JET plugin in NetBeans. Go to Tools > Plugins. Click Available Plugins and search for "jet". Install the plugin named Oracle JET Support. You may need to restart Netbeans before the plugin works.
3. Click **Next >**.



4. Set **Project Name** to **EmpApp**.
5. Click **Finish**.



After clicking **Finish**, NetBeans will download the latest version of the Oracle JET base distribution and use it to create and open the new project. Be patient as this may take a minute or two.

Note that NetBeans provides additional options for starting Oracle JET projects under the "Samples > HTML5/JavaScript" category. The Oracle JET QuickStart Basic option is a great way to kick-start new projects as it includes responsive menus and a basic router configuration ready to go!

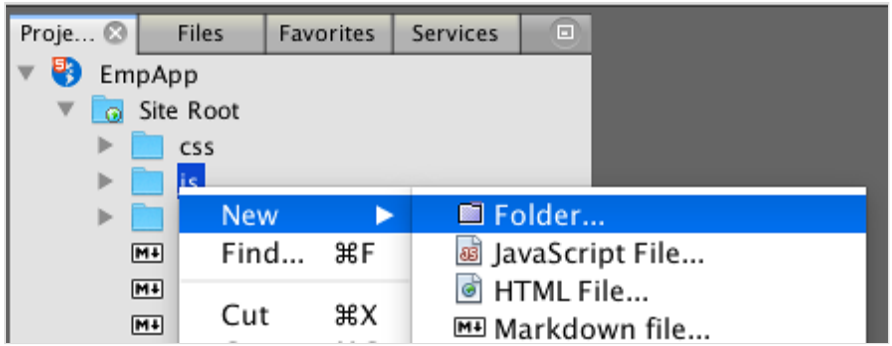
Part 2: Adding Directories and Files

The base distribution of JET is very bare-bones - we'll need to add some directories and files to get things rolling. We'll start by creating some directories that will be used in later sections. After that we'll create the `index.html` and the `main.js` files.

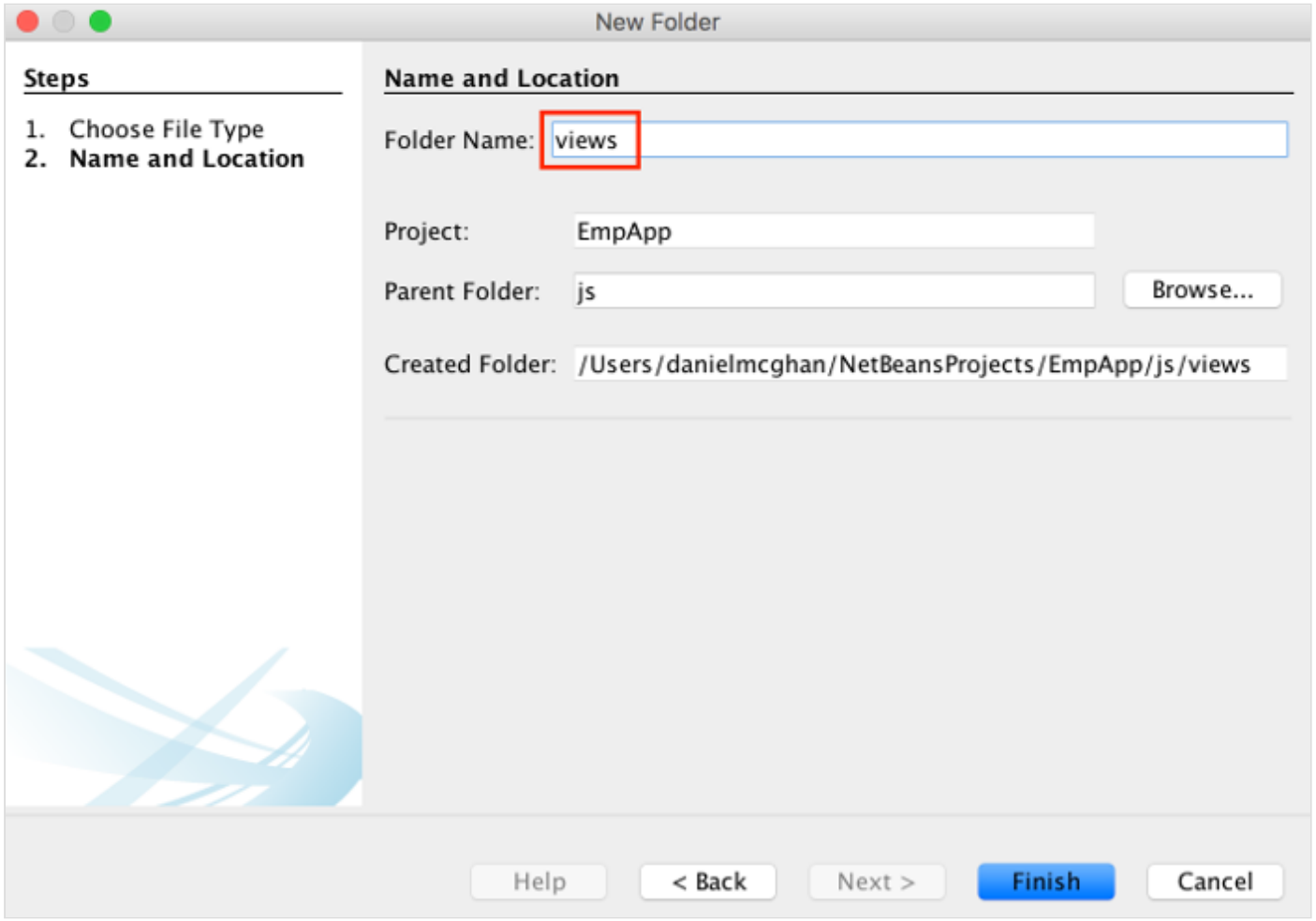
Unlike typical web applications that have many html "pages", SPAs typically have one `index.html` file that acts as the shell of the application. The `index.html` is loaded when a user first navigates to the application and it's responsible for loading initial CSS and JavaScript. In addition, the `index.html` provides some basic structure and entry points for Views and ViewModels, which will be covered in the next part.

The `main.js` file serves as a starting point for all the other JavaScript files (many of which will only be loaded if needed).

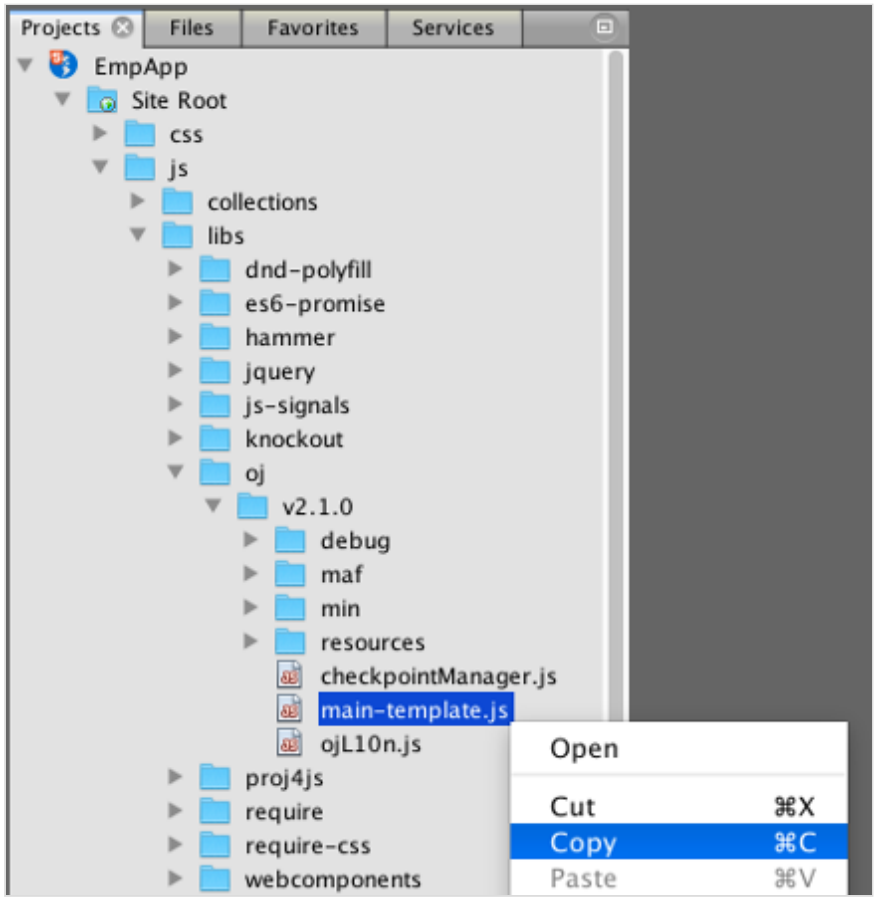
1. Right-click the **js** directory and select **New > Folder...**.



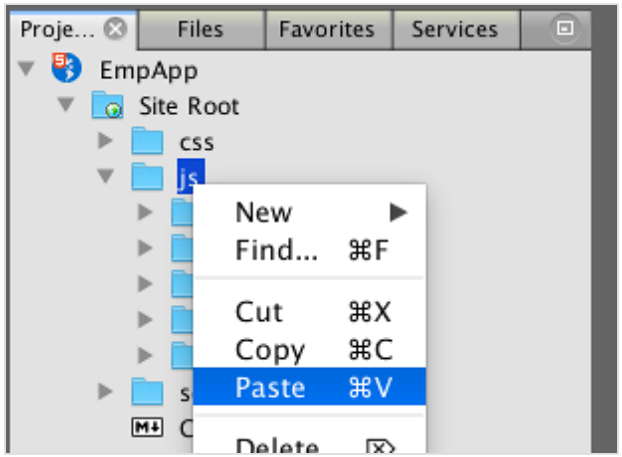
2. Set **Folder Name** to **views**.
3. Click **Finish**.



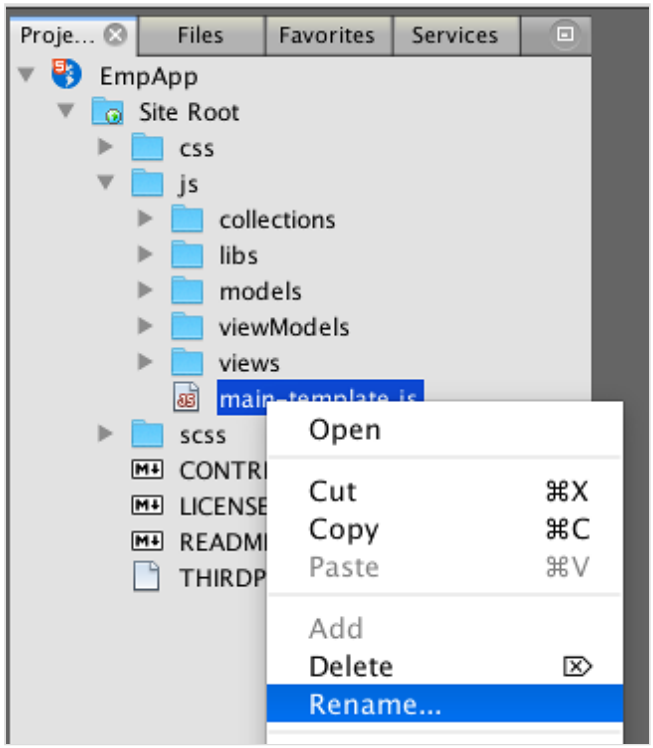
4. Repeat steps 1-3 to create a new folder named **viewModels** under the js directory.
5. Repeat steps 1-3 to create a new folder named **models** under the js directory.
6. Repeat steps 1-3 to create a new folder named **collections** under the js directory.
7. Locate and right-click **js/libs/oj/v2.1.0/main-template.js** and select **Copy**.



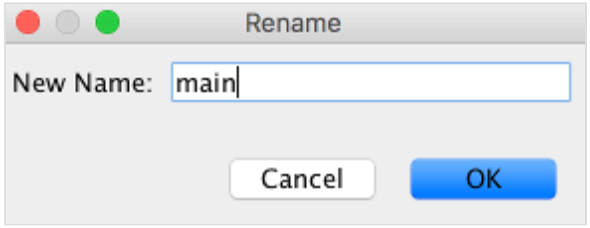
8. Right-click the **js** directory and select **Paste**.



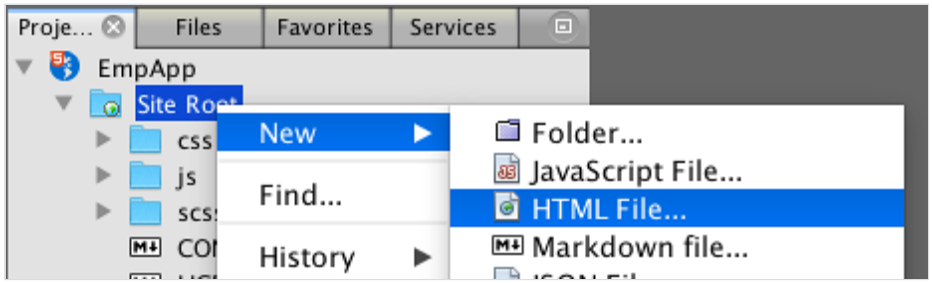
9. Right-click **/js/main-template.js** and select **Rename**.



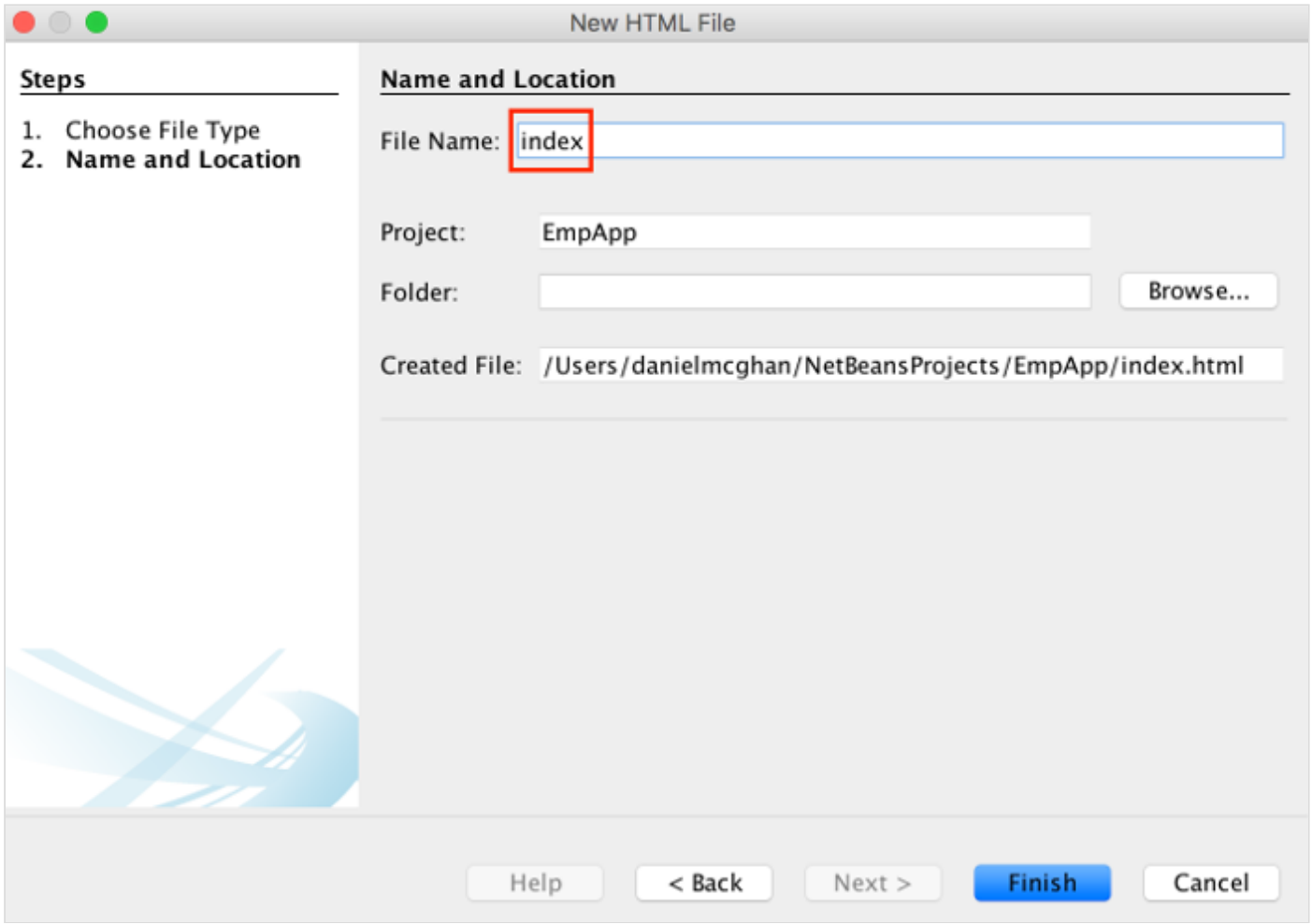
10. Set **New Name** to **main**.
11. Click **OK**.



12. Right-click **Site Root** and select **New > HTML File**.



13. Set **File Name** to **index**.
14. Click **Finish**. This should both create and open the file.



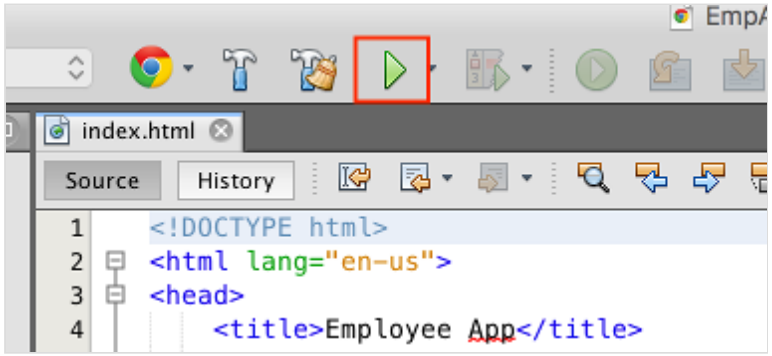
15. Replace the content in **index.html** with the code below and then save your changes.

```
<!DOCTYPE html>
<html lang="en-us">
<head>
    <title>Employee App</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- Main css file for the default Alta theme -->
    <link rel="stylesheet" href="css/libs/oj/v2.1.0/alta/oj-alta-min.css" type="text/css"/>

    <!-- RequireJS configuration file -->
    <script data-main="js/main" src="js/libs/require/require.js"></script>
</head>
<body>
    <div id="globalBody">
        Hello World!
    </div>
</body>
</html>
```

16. Click the green "run" button to run the application. You should see a browser open and display "Hello World!". You didn't think you'd make it through this lab without a "hello world" did you? :)



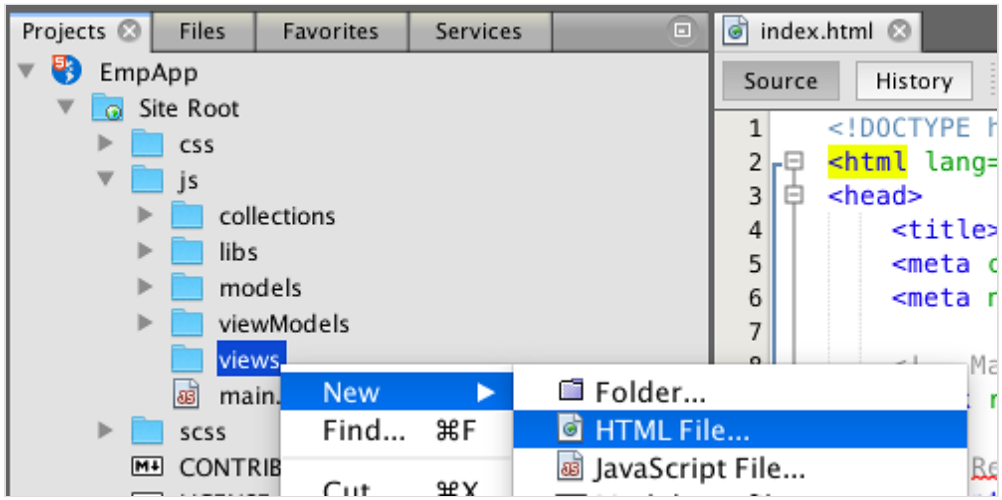
Part 3: Adding Navigation and Routing

Now that we have some basic structure in place, let's get to something more fun: navigation and routing! Because SPAs don't use full page reloads, we'll configure routing using Oracle JET's router along with Views and corresponding ViewModels. The term ViewModel comes from the MVVM development pattern which stands for Model-View-ViewModel. MVVM is an offshoot of the MVC (Model-View-Controller) pattern.

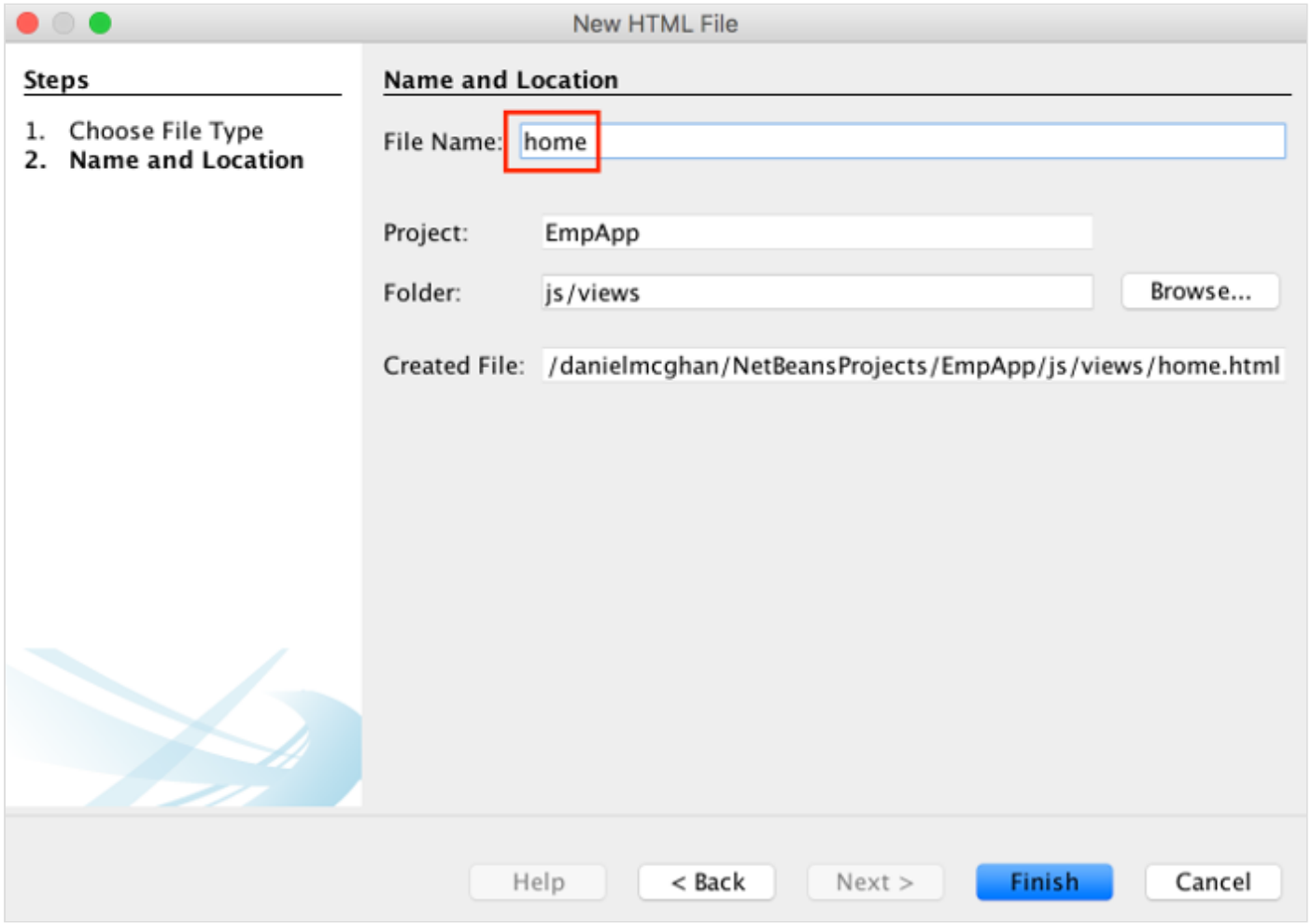
In MVVM, the ViewModel is a model that is specific to a View - meaning you'll find presentation related logic and data in the ViewModel. The View is a template (HTML based in our case) that should reflect the current state of the ViewModel. Oracle JET uses Knockout.js as the MVVM engine.

By default, Oracle JET looks in the `views` and `viewModels` directories to find these modules when needed (though that is configurable). As you create the views, notice that they are just parts (sometimes called partials) of an HTML page. These partials are injected into the page as needed.

1. Right-click the **views** directory and select **New > HTML File**.



2. Set **File Name** to **home**.
3. Click **Finish**. That should both create and open the file.



4. Replace the content of **home.html** with the following code and then save your changes. Notice the input which has a "data-bind" property that references an ojComponent named ojInputText. This is a common pattern you will see in Oracle JET applications.

```
<h1>Home</h1>

<div>
 Change the value of the input below. This will update the value of the "someValue"
 property exposed in the ViewModel. The view will be updated to reflect the change
 so the div will reflect the current value of the input.

 That is two-way data binding in a nutshell!
</div>
<br />
<label for="text-input">Change me</label>
```

```
<input id="text-input" type="text" data-bind="
   ojComponent: {
       component: 'ojInputText',
       value: someValue,
       rawValue: someValue
   }" />
<br />
<div data-bind="text: someValue"></div>
```

5. Repeat steps 1-4 to create a new HTML file named **employees** in the views directory. Replace the content with the following code and save your changes when done.

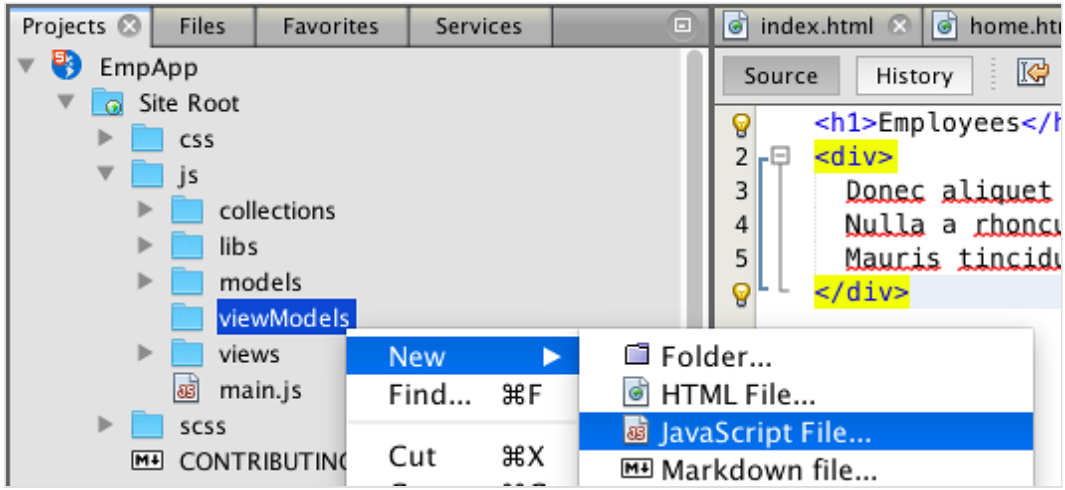
```
<h1>Employees</h1>

<div>
  Donec aliquet faucibus libero, ut tincidunt ipsum dictum vitae. Quisque lacinia aliquet tortor id consectetur.
  Nulla a rhoncus massa. In egestas tempus blandit. Duis gravida at erat vehicula condimentum.
  Mauris tincidunt dolor ut nulla convallis, eget tincidunt ex molestie.
</div>
```

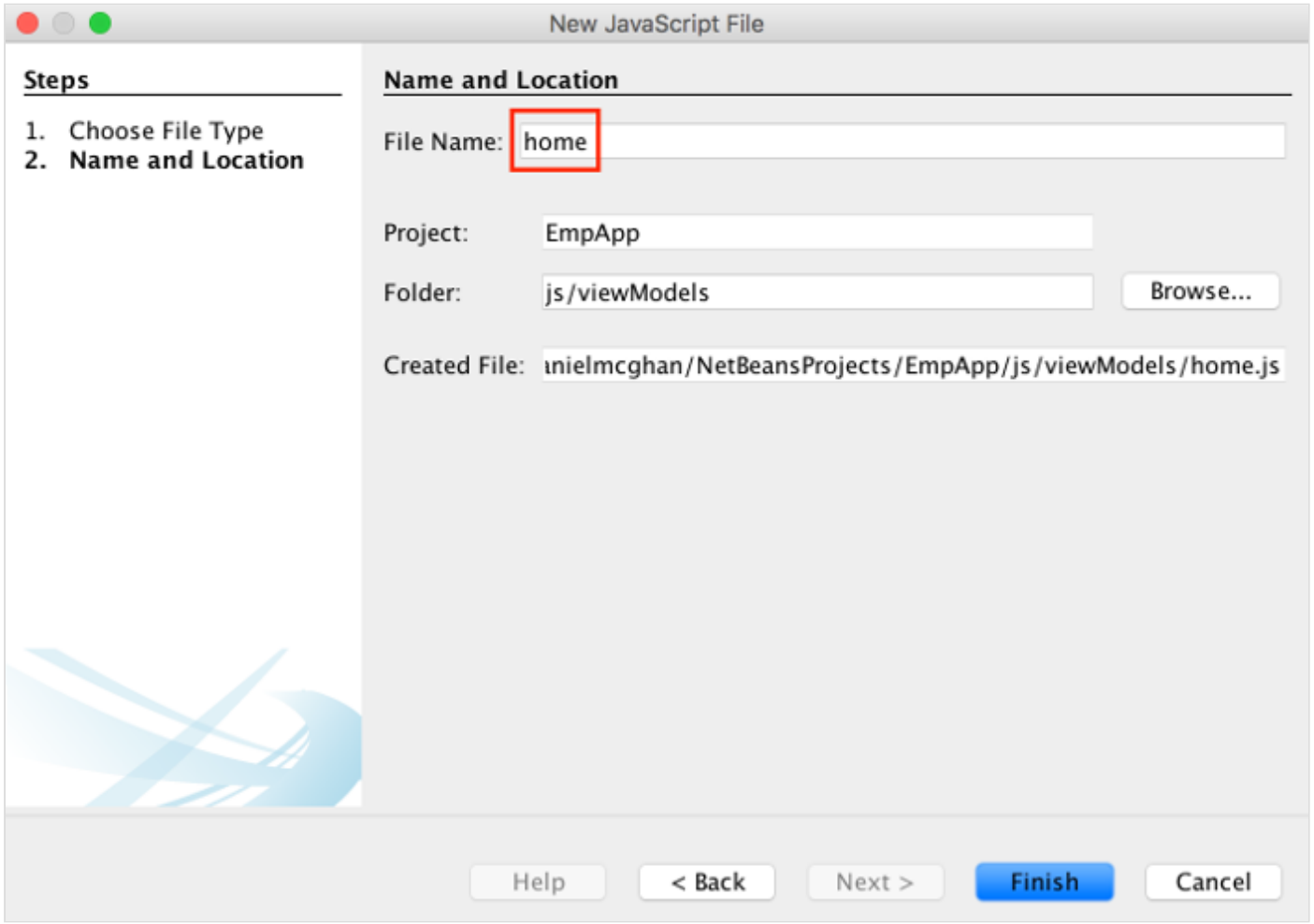
6. Repeat steps 1-4 to create a new HTML file named **header** in the views directory. Replace the content with the following code and save your changes when done.

```
<div class="oj-web-applayout-max-width oj-flex-bar oj-sm-align-items-center">
  <div class="oj-flex-bar-middle oj-sm-align-items-baseline">
    <span class="oj-web-applayout-header-title" title="Application Name" data-bind="text: appName"></span>
  </div>
</div>
<div role="navigation" data-bind="
  ojComponent: {
    component: 'ojNavigationList',
    optionChange: $parent.navChange,
    navigationLevel: 'application',
    item: {template: 'navTemplate'},
    data: $parent.navDataSource,
    selection: $parent.router.stateId(),
    edge: 'top'
  }"
  class="oj-web-applayout-navbar oj-web-applayout-max-width oj-navigationlist-item-dividers oj-md-condense
    j-md-justify-content-center oj-lg-justify-content-flex-end">
</div>
```

7. Right-click the **viewModels** directory and select **New > JavaScript File**.



8. Set **File Name** to **home**.
9. Click **Finish**. That should both create and open the file.



10. Replace the content of **home.js** with the following code and save your changes.

```
define(['ojs/ojcore', 'knockout', 'ojs/ojinputtext'],
   function(oj, ko) {
       /**
        * The view model for the home view
        */
       function HomeViewModel() {
           var self = this;

           self.someValue = ko.observable("Look at me, I'm something!");
       }

       return new HomeViewModel();
```

```
    }
);
```

11. Repeat steps 7-9 to create a new JavaScript file named **employees** in the viewModels directory. Replace the content with the following code and save your changes when done.

```
define(['ojs/ojcore', 'knockout'],
    function(oj, ko) {
        /**
         * The view model for the employees view
         */
        function EmployeesViewModel() {
            var self = this;
        }

        return new EmployeesViewModel();
    }
);
```

12. Repeat steps 7-9 to create a new JavaScript file named **header** in the viewModels directory. Replace the content with the following code and save your changes when done.

```
define(['ojs/ojcore', 'knockout'],
    function (oj, ko) {
        /**
         * The view model for the header view
         */
        function HeaderViewModel() {
            var self = this;

            // Application Name used in Branding Area
            self.appName = ko.observable("Employee App");
        }
        return new HeaderViewModel();
    }
);
```

13. Open the **index.html** file and replace **just the body portion** with the following code. Don't forget to save your changes.

```
<body>
    <!-- template for rendering navigation items -->
    <script type="text/html" id="navTemplate">
        <li><a href="#">
            <!-- ko text: $data['name'] --> <!--/ko-->
        </a></li>
    </script>

    <div id="globalBody">
        <div class="oj-web-applayout-page">

            <!-- This is where your header content will be loaded -->
            <header role="banner" class="oj-web-applayout-header"
                data-bind="ojModule: 'header'"></header>

            <!-- This is where your main page content will be loaded -->
            <div class="oj-web-applayout-max-width oj-web-applayout-content">
                <div class="oj-flex">
                    <div class="oj-flex-item oj-flex">
                        <div id="mainContent" role="main" class="oj-panel oj-margin oj-flex-item"
                            data-bind="ojModule: router.moduleConfig">
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>
```

14. Open the **main.js** file and replace **only the bottom portion** that starts with "require(['ojs/ojcore'..." using the code that follows. Save your changes when you are through.

```
require(['ojs/ojcore', 'knockout', 'jquery', 'ojs/ojknockout', 'ojs/ojrouter',
    'ojs/ojmodule', 'ojs/ojnavigationlist', 'ojs/ojarraytabledatasource'],
    function (oj, ko, $) { // this callback gets executed when all required modules are loaded
        var router = oj.Router.rootInstance;

        router.configure({
            'home': {label: 'Home', isDefault: true},
            'employees': {label: 'Employees'}
        });

        function RootViewModel() {
            var self = this;
            self.router = router;

            // Navigation data for nav bar
```

```
            var navData = [
                {name: 'Home', id: 'home'},
                {name: 'Employees', id: 'employees'}
            ];

            self.navDataSource = new oj.ArrayTableDataSource(navData, {idAttribute: 'id'});

            self.navChange = function(event, ui) {
                if (ui.option === 'selection' && ui.value !== self.router.stateId()) {
                    self.router.go(ui.value);
                }
            };
        }

        oj.Router.defaults['urlAdapter'] = new oj.Router.urlParamAdapter();
        oj.Router.sync().then(
            function () {
                // bind your ViewModel for the content of the whole page body.
                ko.applyBindings(new RootViewModel(), document.getElementById('globalBody'));
            },
            function (error) {
                oj.Logger.error('Error in root start: ' + error.message);
            }
        );
    }
);
```

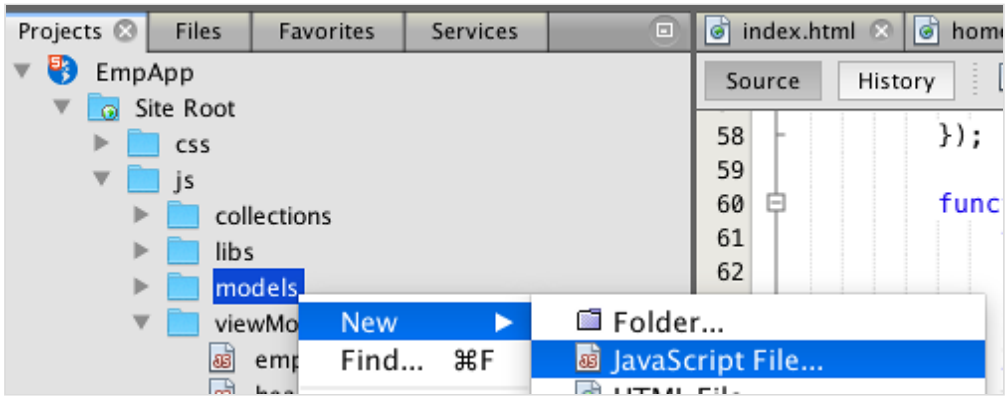
Now when you run your application you should see a navigation bar at the top that can be used to transition the state of your application. Note that your browser history will work despite the fact that you are not doing full page reloads.

Part 4: Adding a Model, Collection, and a Table

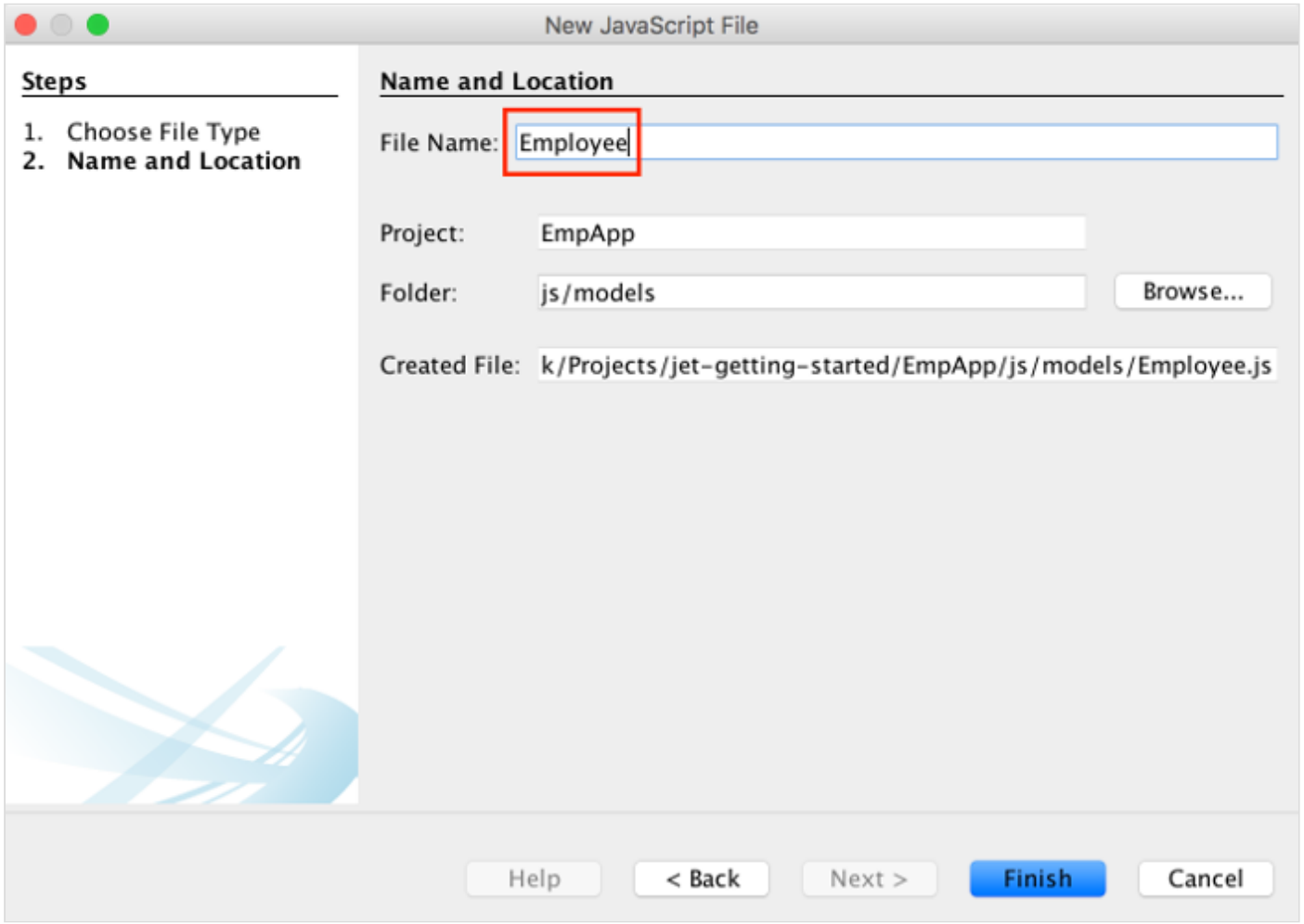
If ViewModels encapsulate presentation related logic, Models and Collections encapsulate the business logic of your application. They provide abilities such as linking to RESTful APIs, converting data, doing validations, specifying defaults, etc. If you come from a database background, it's probably easiest to think of models as a row in a table and collections as a table, or set of like models. Models and collections are part of the Common Model API in Oracle JET which was originally based on Backbone.js.

The models and collections you create in this app will be linked to RESTful endpoints that were created using ORDS for this lab. Once in place, they can be used in various Oracle JET components, such as the table and pagination controls we'll add in this part.

1. Right-click the **models** directory and select **New > JavaScript File**.



2. Set **File Name** to **Employee**.
3. Click **Finish**. That should both create and open the file.



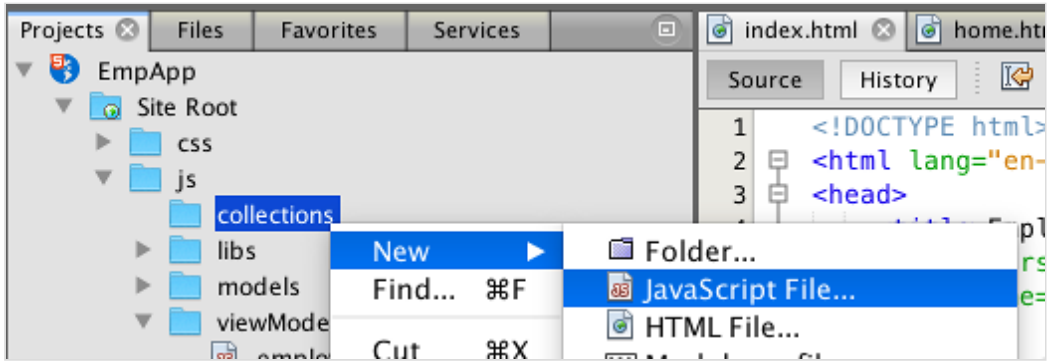
4. Replace the content of **Employee.js** with the following code and save your changes. Notice that new models are created by "extending" the base Model class.

```
define(['ojs/ojcore', 'ojs/ojmodel'],
    function (oj) {
        var Employee = oj.Model.extend({
            urlRoot: 'http://45.55.152.87:8080/ords/api/hr/employees',
            parse: function(emp) {
              return {
                id: emp.employee_id,
                firstName: emp.first_name,
                lastName: emp.last_name,
                salary: emp.salary
              };
```

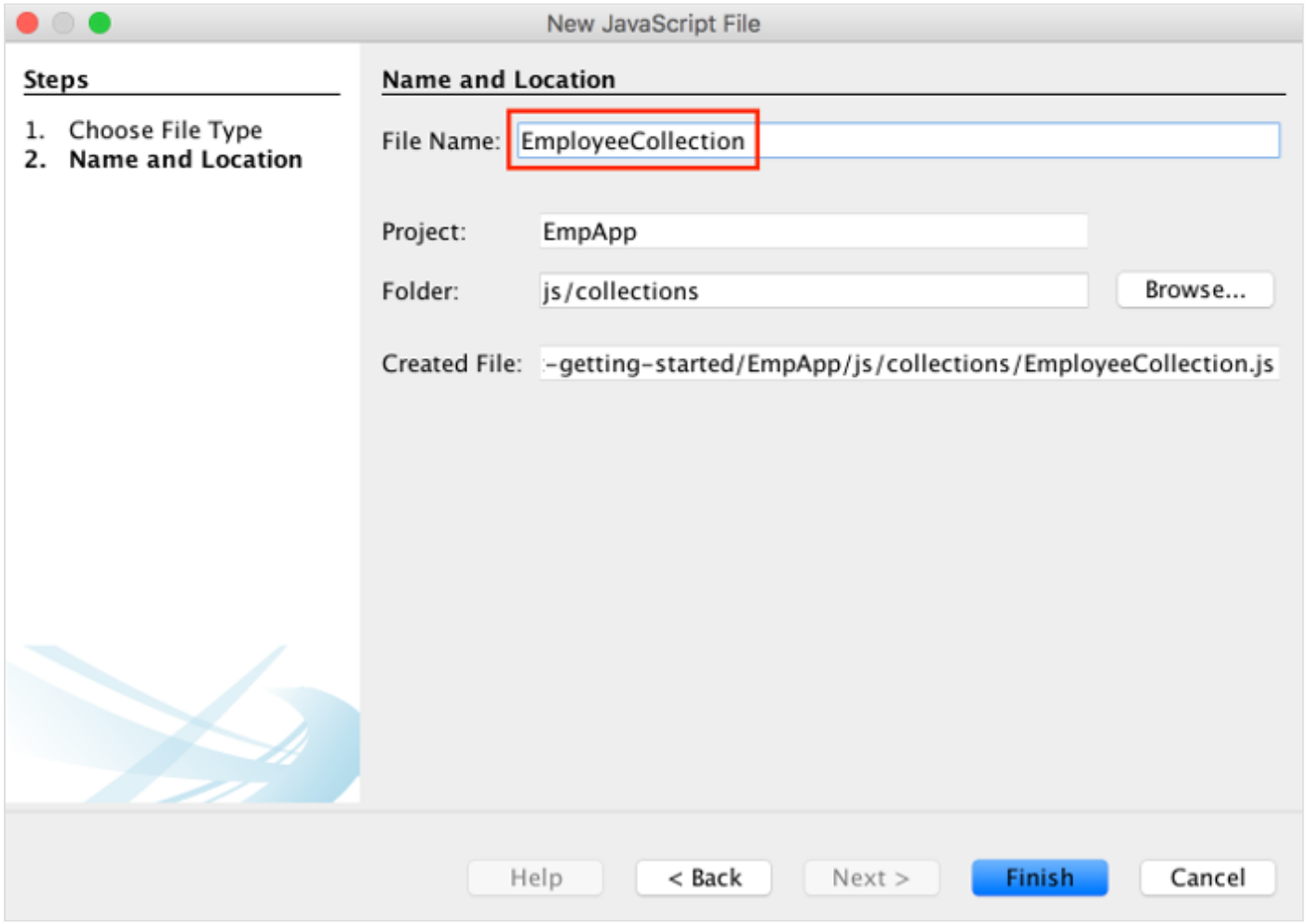
```
            },
            parseSave: function(emp) {
              return {
                employee_id: emp.id,
                first_name: emp.firstName,
                last_name: emp.lastName,
                salary: emp.salary
              };
            }
          });

        return Employee;
    }
);
```

5. Right-click the **collections** directory and select **New > JavaScript File**.



6. Set **File Name** to **EmployeeCollection**.
7. Click **Finish**. That should both create and open the file.



8. Replace the content of **EmployeeCollection.js** with the following code and save your changes. Notice that new collections are created by "extending" the base Collection class.

```
define(['ojs/ojcore', 'models/Employee', 'ojs/ojmodel'],
    function (oj, Employee) {
        var EmployeeCollection = oj.Collection.extend({
            url: 'http://45.55.152.87:8080/ords/api/hr/employees',
            fetchSize: 100,
            model: Employee
        });

        return EmployeeCollection;
    }
);
```

9. Open the **views/employees.html** file and replace the content with the following code, then save your changes.

```
<h1>These are employees!</h1>

<table id="table" summary="Employee List" aria-label="Employee Table"
  data-bind="ojComponent: {
    component: 'ojTable',
    data: employeesPagingTableDataSource,
    columnsDefault: {sortable: 'none'},
    columns: [
      {headerText: 'Employee Id', field: 'id'},
      {headerText: 'First Name', field: 'firstName'},
      {headerText: 'Last Name', field: 'lastName'},
      {headerText: 'Salary', field: 'salary'}
    ]
  }">
</table>

<div id="paging"
  data-bind="ojComponent: {
    component: 'ojPagingControl',
    data: employeesPagingTableDataSource,
    pageSize: 20
  }">
</div>
```

10. Open the **viewModels/employees.js** file and replace the content with the following code. Save your changes when finished.

```
define(
 ['ojs/ojcore', 'knockout', 'collections/EmployeeCollection', 'ojs/ojcollectiontabledatasource',
   'ojs/ojtable', 'ojs/ojpagingcontrol', 'ojs/ojpagingtabledatasource'],
 function(oj, ko, EmployeeCollection) {
   function employeesViewModel() {
       var self = this;
       var employeesColl;
       var employeesCollectionDataSource;

       employeesColl = new EmployeeCollection();

       employeesCollectionDataSource = new oj.CollectionTableDataSource(employeesColl);

       self.employeesPagingTableDataSource = new oj.PagingTableDataSource(employeesCollectionDataSource);
   }

   return new employeesViewModel();
```

```
  }
);
```

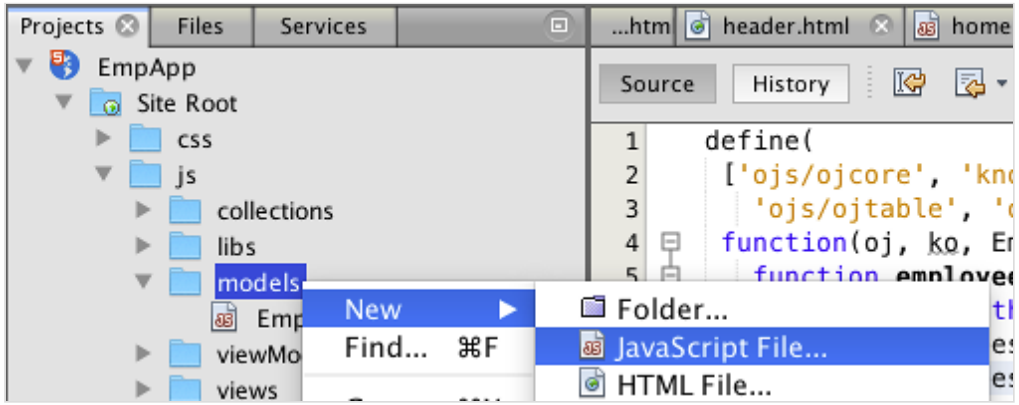
Now when you run the application and navigate to the employees page, you should see the employee data load and the pagination controls should allow you to navigate the data.

Part 5: Adding a Model, Collection, and a Chart

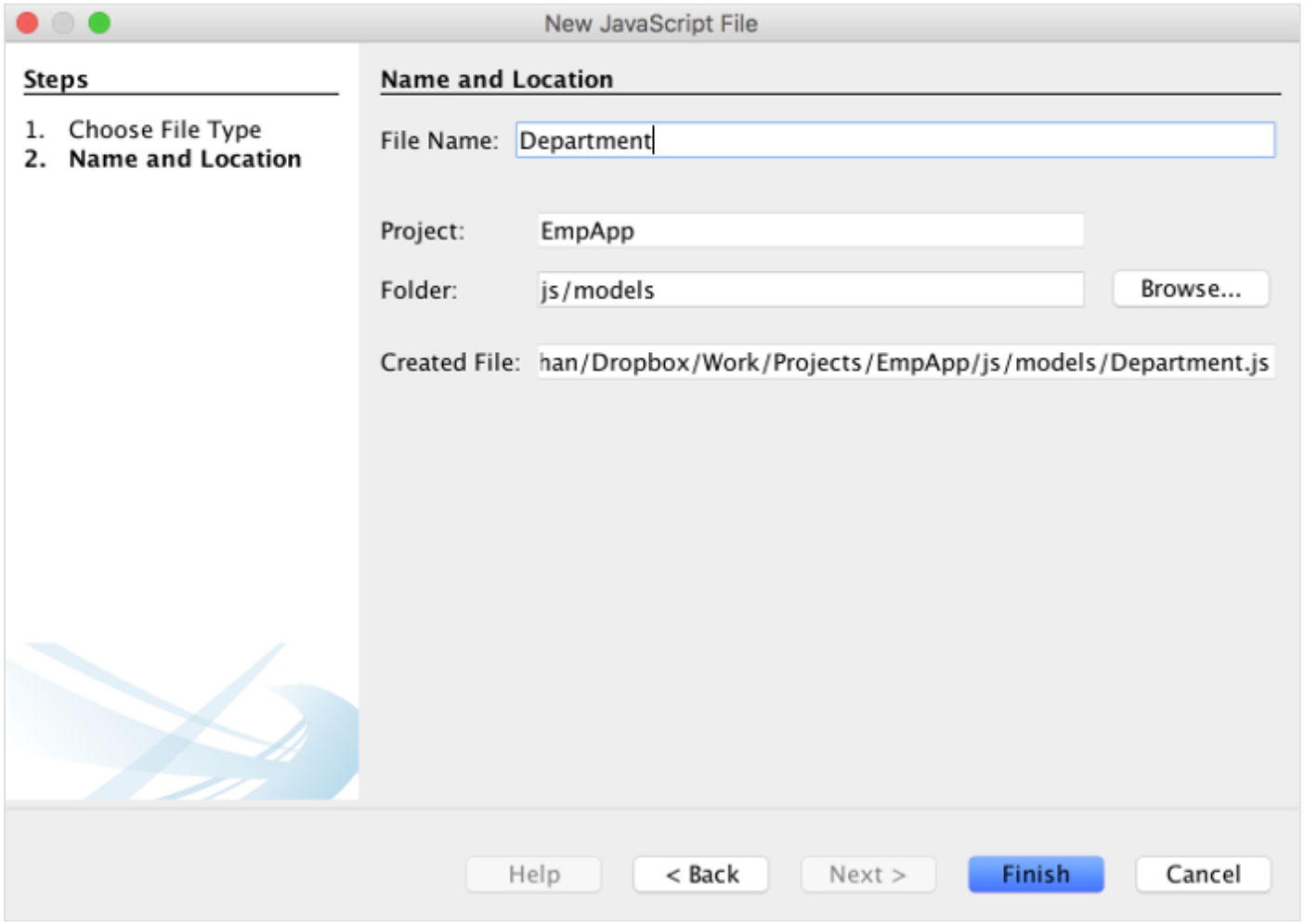
One of the things that differentiates Oracle JET from other JavaScript frameworks is its comprehensive set of data visualizations. You can have a look at the various offerings here: http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html?component=home&demo=rootVisualizations

In this part we'll add a pie chart to the home page which shows the total employee salary by department.

1. Right-click the **models** directory and select **New > JavaScript File**.



2. Set **File Name** to **Department**.
3. Click **Finish**. That should both create and open the file.



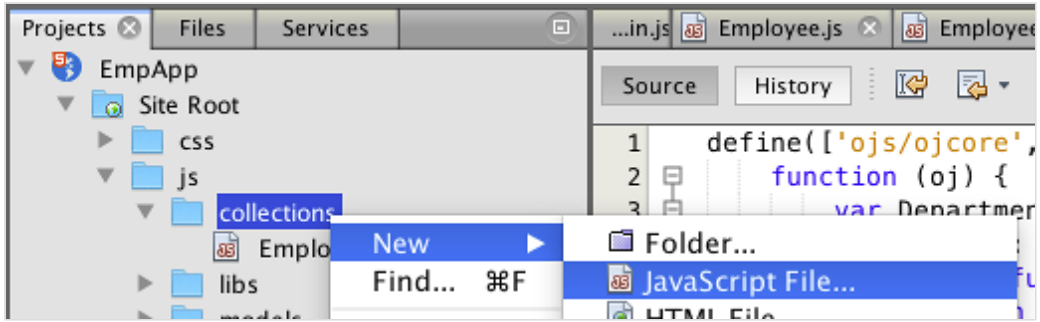
4. Replace the content of **Department.js** with the following code and save your changes.

```
define(['ojs/ojcore', 'ojs/ojmodel'],
    function (oj) {
        var Department = oj.Model.extend({
            urlRoot: 'http://45.55.152.87:8080/ords/api/hr/departments',
            parse: function(dept) {
              return {
                id: dept.department_id,
                name: dept.department_name,
                sumSalary: dept.sum_emp_salary
              };
            },
            parseSave: function(dept) {
```

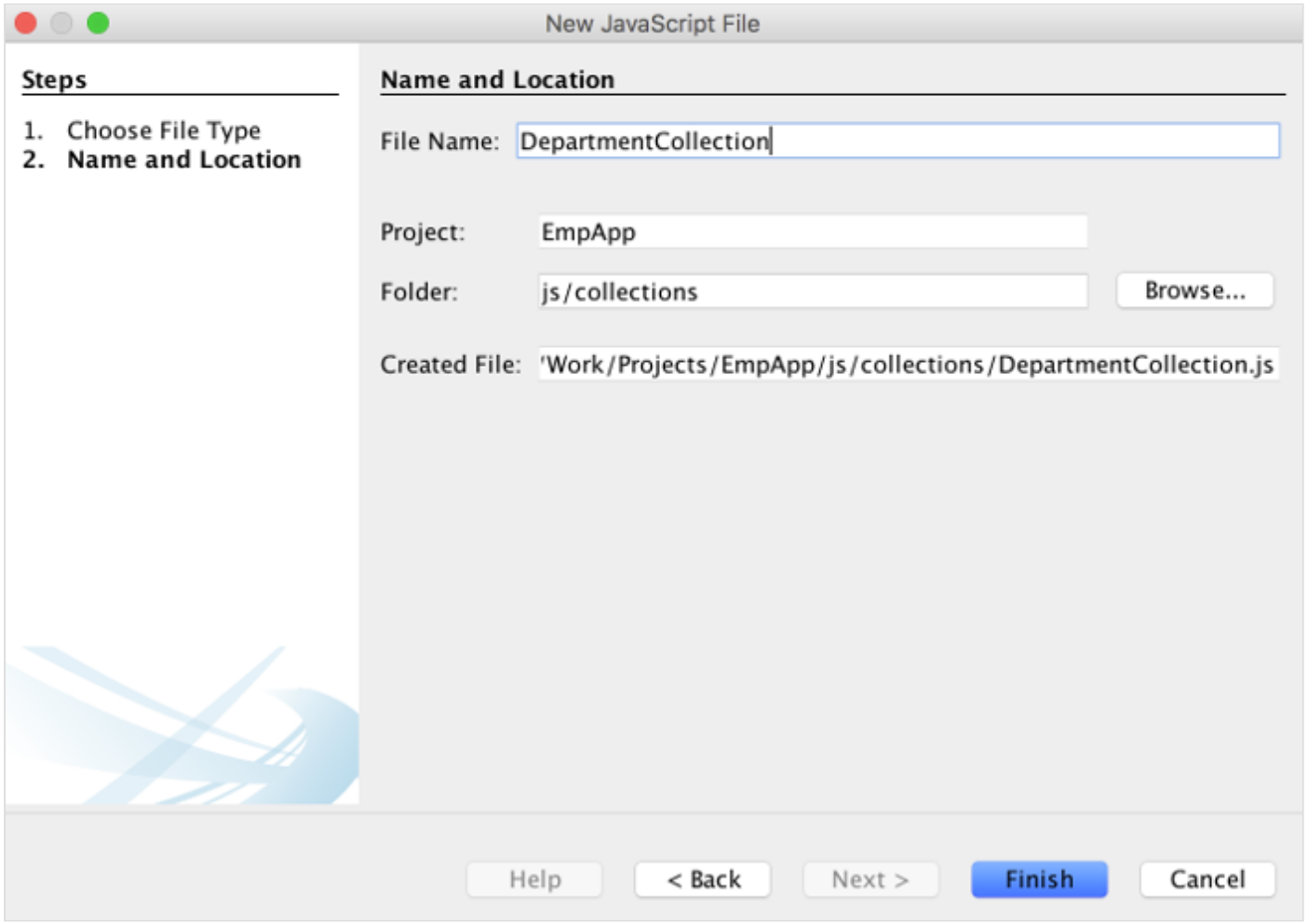
```
                return {
                  department_id: dept.id,
                  department_name: dept.name
                };
              }
          });

          return Department;
      }
  );
```

5. Right-click the **collections** directory and select **New > JavaScript File**.



6. Set **File Name** to **DepartmentCollection**.
7. Click **Finish**. That should both create and open the file.



8. Replace the content of **DepartmentCollection.js** with the following code and save your changes.

```
define(['ojs/ojcore', 'models/Department', 'ojs/ojmodel'],
    function (oj, Department) {
        var DepartmentCollection = oj.Collection.extend({
            url: 'http://45.55.152.87:8080/ords/api/hr/departments',
            fetchSize: -1, // Disables pagination/virtualization
            model: Department
        });

        return DepartmentCollection;
    }
);
```

9. Open the **views/home.html** file and replace the content with the following code, then save your changes.

```
<h1>Home</h1>

<div id="barChart"
    data-bind="ojComponent: {
        component: 'ojChart',
        type: 'pie',
        orientation: 'auto',
        series: deptChartData,
        hoverBehavior: 'dim',
        sorting: 'descending'
    }"
    style="max-width:500px;width:100%;height:350px;">
</div>
```

10. Open the **viewModels/home.js** file and replace the content with the following code. Save your changes when finished.

```
define(['ojs/ojcore', 'knockout', 'collections/DepartmentCollection', 'ojs/ojchart'],
  function(oj, ko, DepartmentCollection) {
      /**
       * The view model for the home view
       */
      function HomeViewModel() {
          var self = this;
          var departmentsColl;

          departmentsColl = new DepartmentCollection();

          // Populate the collection from the server
          departmentsColl.fetch()
              .then(function() {
                  // Convert the collection to a simple array for the chart
                  var data = departmentsColl.map(function(deptModel) {
                      return {
                          name: deptModel.get('name'),
                          items: [deptModel.get('sumSalary')]
                      };
                  });

                  // Remove departments with a sumSalary of 0
                  data = data.filter(function(dept) {
                      return dept.items[0] !== 0;
                  });
```

```
                    // Update the value of the observableArray
                    self.deptChartData(data);
                });

            self.deptChartData = ko.observableArray([]);
        }

        return new HomeViewModel();
    }
);
```

If you run your application now and navigate to the home page, you should see a pie chart that shows the sum of the employee salary by department.

Summary

We covered many different aspects of Oracle JET in this lab, including:

- Modules
- Views & ViewModels
- Routing
- Models & Collections
- Table and Chart Components

In reality, we've just scratched the surface with Oracle JET. If you're interested in learning more, be sure to check out the examples and cookbook sections of the Oracle JET website.

Also, don't forget to follow Oracle JET on Twitter for tips, announcements, and other content related to Oracle JET!