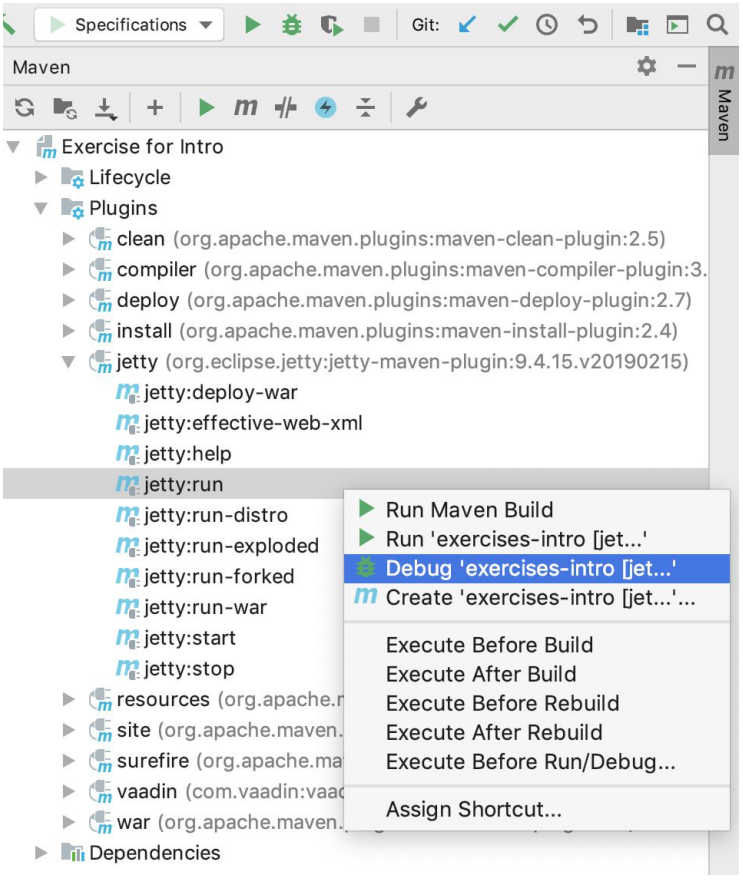
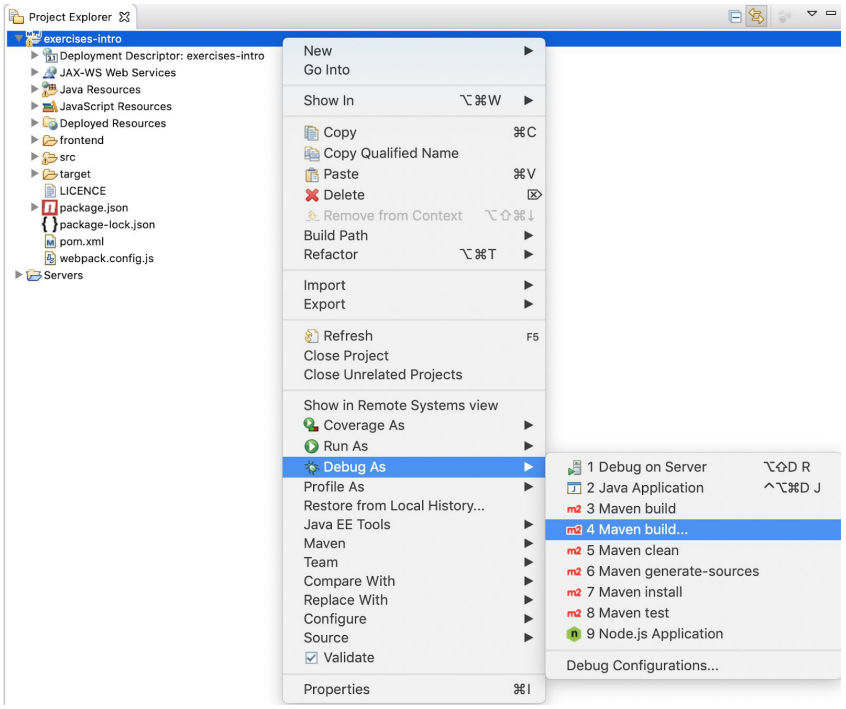
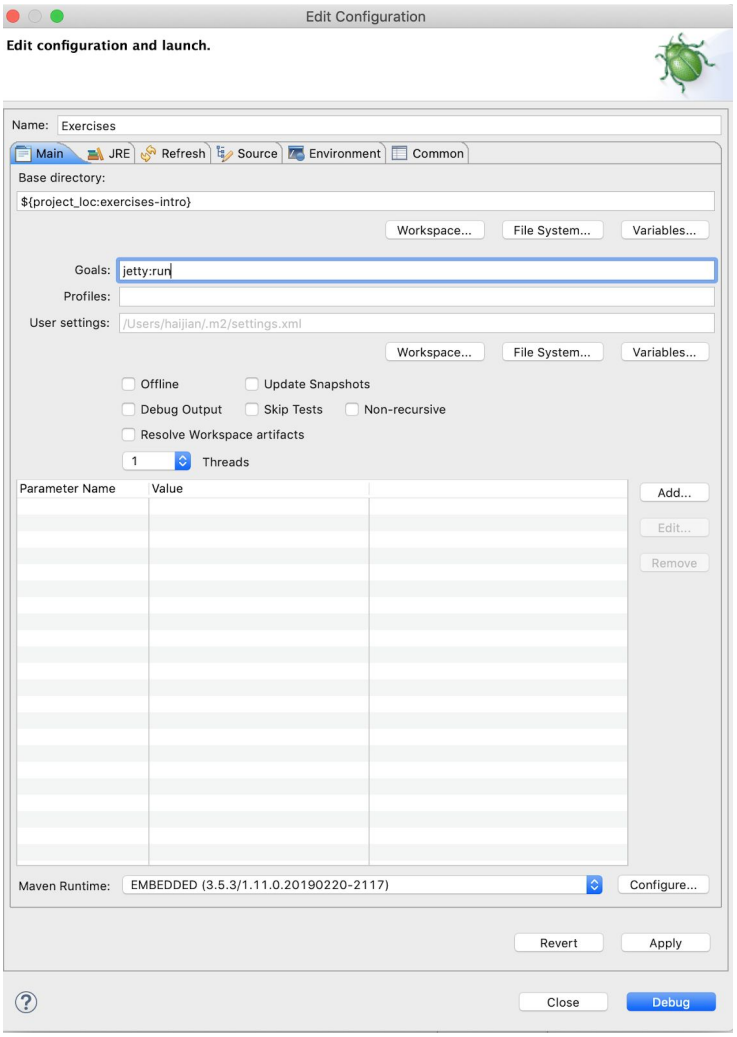
Instructions

1. Import the Maven project to your Favourite IDE.
2. Run the Maven goal jetty:run
    a. If you have command line Maven installed you can run mvn jetty:run in terminal OR
    b. Debug or run the application in IntelliJ



c. Debug or run the application in Eclipse





3. Go to localhost:8080

Exercise 1: Use @Id annotation

One typical use case for Templates is that you can use it as a RouterLayout (which we've covered in the Router API training). This means you can set up a layout component or container element and display some content inside. By default, Router will just append the content as a child, see RouterLayout::showRouterLayoutContent. If you need different behavior, basically you have two options:

1) Override the RouterLayout::showRouterLayoutContent method. Then you can put the content wherever you want it to be.
2) Put a <slot> where you want your content to be displayed. By default the content will append the content as a child, so the content would be added to light DOM, by adding a <slot>, the content will replace <slot> during rendering. For this exercise, we will practice the first approach, and in the next exercise, we will practice the second approach. We will also use the @Id annotation to connect the client-side element and server-side component.

- Step 1: Add an id attribute to the div element in the ex1-layout.js file.
- Step 2: Create a member field of type Div in Exercise1Layout class.
- Step 3: Add an @Id annotation on the Div. The value parameter of the annotation should be the same as the id attribute of the div element.
- Step 4: Override the showRouterLayoutContent method in Exercise1Layout, so that you can add the content to the Div.

Exercise 2: Use <slot>

With <slot>, creating a router layout is much easier. You can just put a <slot> as the placeholder for your content in the client-side template file. Then all the light DOM elements will be rendered in the place where <slot> is put. Of course, you can also use named slot to indicate which element in the light DOM should be placed in which <slot> in shadow DOM. There is just one single step for this exercise. Put a <slot> inside the div element in the ex2-layout.js file.

Exercise 3: Make a small ToDo application

In this exercise, we will practice the event handling and data binding features of Template by creating a small ToDo application. The application itself is quite simple, it has a paper input on the top, which takes user input for the new ToDo when the user hits the Enter key, then a new ToDo will be created and shown underneath. Under the paper input, there is a list, which shows all the ToDo items, each item has a checkbox for showing if the task is done and a text description next to the checkbox.

Step by step:

1. In the ToDoModel class, add abstract getter/setter methods for List<ToDo>, e.g. get/ setToDoList, which will be bind the ToDo list in the template. Note that you don't need to implement the methods.
2. In the ToDoModel class, add getter/setter methods for String, e.g. get/setNewText, which will be bound to paper input element in the template.
3. In the todo-view.js file, bind the paper-input element to the newText attribute, which is defined in the ToDoModel with getter and setter: value="{{newText}}"
4. In the todo-view.js file, add 'items' attribute to the <template is="dom-repeat"> element, bind it to toDoList property, which is defined in the ToDoModel with getter and setter.
5. When the "change" event occurs in the paper-input, invoke a server side event listener. Do it by adding an on-change attribute to the element and set its value to the name of a method, e.g. `onNewToDo`.
6. In the ToDoView class, create a method with a matching name used in the previous step. Annotate the method with @EventHandler. In the method

- Create a copy of the Model's ToDoList:
    - new ArrayList<ToDo>(getModel().getToDoList());
- Create a new ToDo with the text from your Model
- Place the new List in your Model
- Clear the new item text from your Model