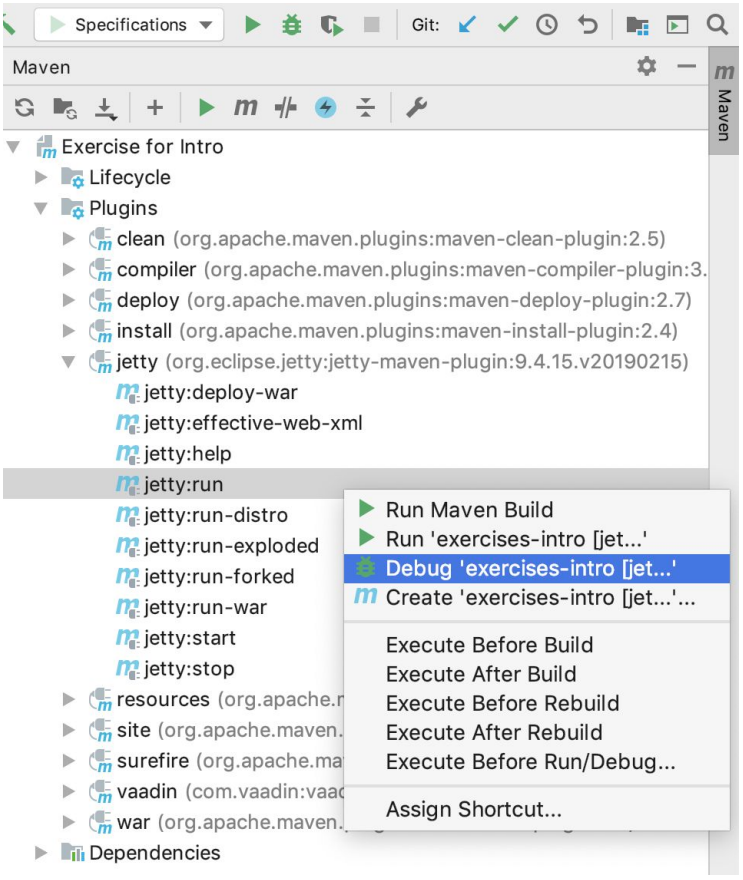
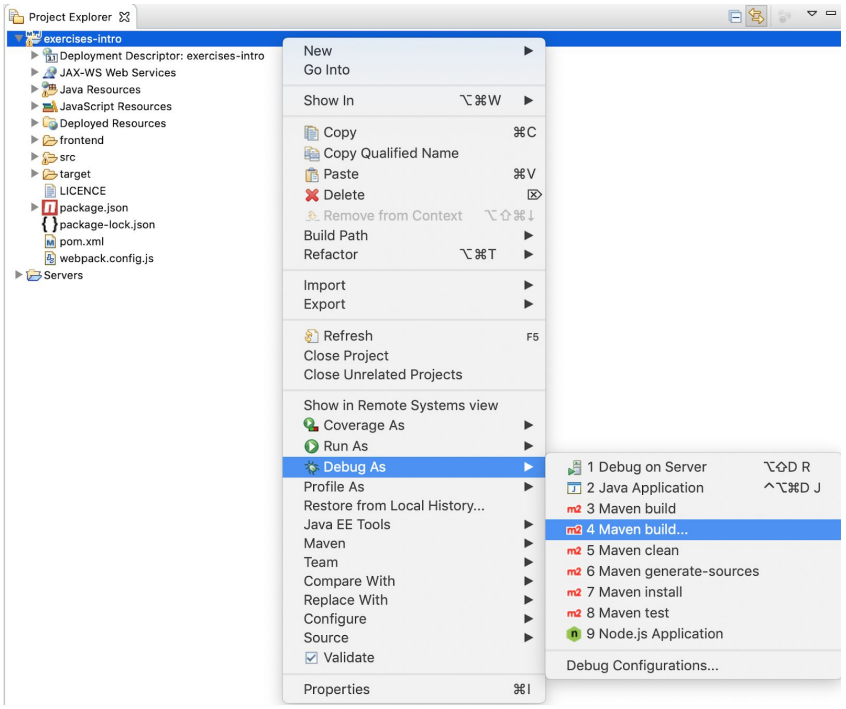
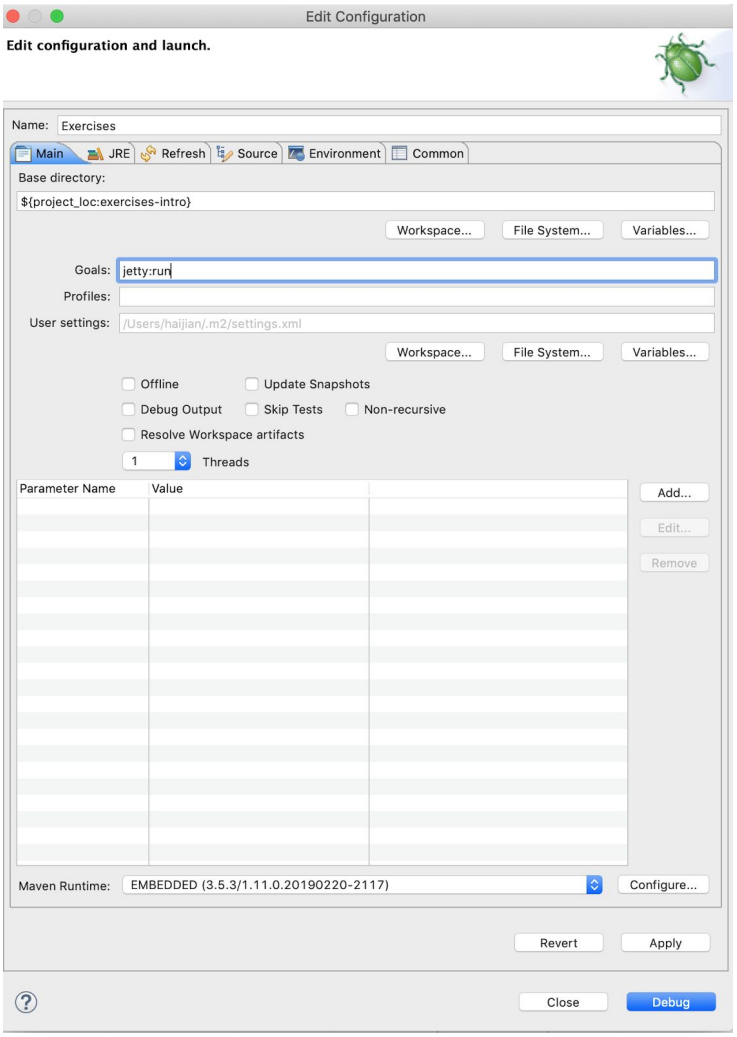
Instructions

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: Push

Sometimes we have operations in our backend that take a while to perform. If we perform heavy processes in the same thread as in which the HTTP request is processed, we will block any user interface interactions until the process is done. In most cases, we want the user to be able to continue using the user interface even though we are doing “heavy stuff” in the backend. Hence, the heavy process is typically performed in a separated thread.

Without Push, if we modify the user interface from a thread that is outside the HTTP request, we won’t see the changes in the browser until the user does something that triggers an HTTP request (so that we can send the changes of the UI in the response of that request). For the changes to be seen in the user interface immediately, we can use server push, which will allow us to “push” the changes to the browser, even if we don’t have an active HTTP request going on.

In this exercise, we will practice enabling push in a Vaadin application. The situation we are trying to simulate is that PersonService’s getEmployees method takes multiple seconds before it returns its data. We want to push the user information to the UI when the process is done. On a high level, we need to enable Push support in the Vaadin application, make the heavy process to be accessed in a separate thread and then handle possible concurrency issues that may occur if we modify the UI from a thread outside the HTTP request.

Things you’ll need to do in your application:

1. We want to have a grid filled with the results of the getEmployees service method. Loading the data will take some seconds and should be done on a dedicated thread. Therefore you need to create an ExecutorService that will provide you a safe way to get threads for executing long-running processes in the backend. Create the service as a class field like this:
   `ExecutorService executor = Executors.newCachedThreadPool();`
2. In the PushView.java, add @Push annotation to enable push support.
3. Override the onAttach() method, set the UI to the initial state. You can, for instance, add a ‘loading’-text.
4. Create a CompletableFuture and load (asynchronously) the Employee List using CompletableFuture.supplyAsync() and providing a lambda that loads the persons from the PersonService. For instance like this:
   `CompletableFuture.supplyAsync(() -> personService.getEmployees(), executor);`
5. The CompletableFuture object has a thenAccept(Consumer<? super T> action) method that takes a callback. In it, you should add the resulted list to PersonGrid with UI.access.
6. Remember to hide any loading text you might have added.