Exercise 1: Setup Vaadin 14

Goal: Make a test “hello world” page with Vaadin 14 to see if everything works

The project’s root pom.xml file has already been modified with the following steps so that you can build it without changes, but take a look at the following changes which will be needed in a real Vaadin 8 project:

1. The old <vaadin.version> has been renamed to <vaadin8.version>.
2. There’s a new <vaadin.version> property for Vaadin 14 and an added vaadin-bom platform dependency:

```
<properties>
    <vaadin.version>14.0.0</vaadin.version>
</properties>

<dependencyManagement>
 <dependencies>
  <dependency>
     <groupId>com.vaadin</groupId>
     <artifactId>vaadin-bom</artifactId>
     <version>${vaadin.version}</version>
     <type>pom</type>
     <scope>import</scope>
   </dependency>
 </dependencies>
</dependencyManagement>
```

Additionally, the existing Vaadin 8 dependencies like vaadin-server and vaadin-themes have been updated to use the new <vaadin8.version> property from the parent project. Check both the bookstore-starter-flow-ui-exercise and widgetset modules.

In a real life project, you might also need to update the version of the jetty-maven-plugin to 9.4+

Follow these steps for this exercise:

1. Update the POM file under bookstore-starter-flow-ui-exercise module.
   1.1. Add “vaadin-core” and “sl4j-simple” dependencies

```
<dependency>
  <groupId>com.vaadin</groupId>
  <artifactId>vaadin-core</artifactId>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
</dependency>
```

   1.2. Define a flow.plugin.version property in the exercise module’s pom.xml under the <project> root. It should match the Flow version of the project (you can find this from the vaadin-bom).

```
<properties>
  <flow.plugin.version>2.0.14</flow.plugin.version>
</properties>
```

   1.3. Add the flow-maven-plugin plugin (don’t remove the vaadin-maven-plugin):

```
<plugin>
  <groupId>com.vaadin</groupId>
  <artifactId>flow-maven-plugin</artifactId>
  <version>${flow.plugin.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>prepare-frontend</goal>
```

```
            </goals>
          </execution>
        </executions>
      </plugin>
```

2. Create Hello world test page
   2.1. Create a class HelloWorldPage in the bookstore-starter-flow-ui-exercise module, e.g. in the com.vaadin package.
   2.2. The content could be as simple as this

```
@Route("")
public class HelloWorldPage extends Div {
  public HelloWorldPage() {
    setText("Hello World!");
  }
}
```

3. Update the MyUIServlet in MyUI class to implement the createDeploymentConfiguration method as follows. Note the urlPatterns should be updated too:

```
@WebServlet(urlPatterns = {"/v8/*", "framework/*"}, name = "MyUIServlet", asyncSupported = true)
@VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
public static class MyUIServlet extends VaadinServlet {
  @Override
  protected DeploymentConfiguration createDeploymentConfiguration(
      Properties initParameters) {

    initParameters.setProperty(Constants.PARAMETER_VAADIN_RESOURCES,
        "/framework");

    return super.createDeploymentConfiguration(initParameters);
  }
}
```

Now the new “hello world page” with Vaadin 14 is available at localhost:8080. The legacy application is available at localhost:8080/v8

Step 2: Access Control

In this step, we will add access control to the HelloWorldPage so that the user will be redirected to a login view when trying to access the “hello world” page without logging in.

You can follow these steps for this exercise:

1. Create a new ‘flow’ package under com.vaadin.samples.authentication in the bookstore-starter-flow-ui module.
2. Create a simple LoginView class, which is quite similar to HelloWorldPage, but with @Route(“login”)

```
@Route("login")
public class LoginView extends Div {
  public LoginView(){
    setText("Login view");
  }
}
```

3. Authentication
   - 3.1. Copy CurrentUser and BasicAccessControl to the package com.vaadin.samples.authentication.flow.
   - 3.2. For CurrentUser, update the imports from com.vaadin.server.xxx to com.vaadin.flow.server.xxx
   - 3.3. For BasicAccessControl, add an import the for AccessControl interface, which is outside the flow package
4. Add access control to HelloWorldPage
   - 4.1. The old MyUI class holds an AccessControl instance, which does the authentication. We can put an AccessControl instance to a new UI instance by using ComponentUtil.
   - 4.2. Create an AccessControlFactory class in the com.vaadin.samples.authentication package, which is a utility class that helps to get the AccessControl instance from a UI (com.vaadin.flow.component.UI).

```
public class AccessControlFactory {
  public static AccessControl getAccessControl() {
    if(ComponentUtil.getData(UI.getCurrent(), AccessControl.class)==null){
      ComponentUtil.setData(UI.getCurrent(), AccessControl.class, new BasicAccessControl());
    }
    return ComponentUtil.getData(UI.getCurrent(), AccessControl.class);
  }
}
```

   - 4.3. We can finally add the authentication logic into HelloWoldPage
     - 4.3.1. Let the HelloWorldPage class implements BeforeEnterObserver.
     - 4.3.2. Implement the beforeEnter() method, if the user is not logged in, redirect to LoginView.

```
@Route("")
public class HelloWorldPage extends Div implements BeforeEnterObserver {
  public HelloWorldPage() {
    setText("Hello World!");
  }
  @Override
  public void beforeEnter(BeforeEnterEvent event) {
    if (!AccessControlFactory.getAccessControl().isUserSignedIn()) {
      event.rerouteTo(LoginView.class);
    }
  }
}
```

Now when trying to access localhost:8080, you will be redirected to localhost:8080/login (and you can't yet log in)

Step 3: LoginView

Now let's start migrating the UI code with LoginScreen.

You can follow these steps for this exercise:

1. Copy the code from LoginScreen to LoginView which was created in the last step.
2. Update the code to use components and APIs from Vaain 14. e.g.
   - 2.1. Remove all the imports from com.vaadin. Reimport the components from com.vaadin.flow
   - 2.2. Use Vaadin 14 Components and APIs. e.g.
     - 2.2.1. CssLayout -> FlexLayout or Div
     - 2.2.2. addStyleName() -> addClassName()
     - 2.2.3. setStyleName() -> setClassName()
     - 2.2.4. addComponents() -> add()
     - 2.2.5. formLayout.addComponent() -> formLayout.addFormItem()
     - 2.2.6. setWidth(15, Unit.EM) -> setWidth("15em");
     - 2.2.7. Change Label in HTML content mode to HTML component. Note that HTML component should only have one parent node, so you might want to wrap the content to a . e.g.
       new Label("<h1>Login Information<h1>Login as ...") ->
       new Html(" <h1>Login Information<h1>Login as ... ")
     - 2.2.8. addStyleName(ValoTheme.BUTTON_LINK) -> addThemeVariants(ButtonVariant.LUMO_TERTIARY)
     - 2.2.9. addStyleName(ValoTheme.BUTTON_FRIENDLY) -> addThemeVariants(ButtonVariant.LUMO_PRIMARY, ButtonVariant.LUMO_SUCCESS)
     - 2.2.10. There is currently no replacement for `password.setDescription()`, can be ignored for this exercise.
   - 2.3. Remove accessControl field, as AccessControl can be retrieved from AccessControlFactory.
   - 2.4. Remove LoginListener, as we only need to navigate to HelloWorldPage after login.
   - 2.5. Inside the login() method, call `getUI().ifPresent(ui->ui.navigate(HelloWorldPage.class));` instead of `loginListener.loginSuccessful();`
   - 2.6. Centering of the formlayout inside VerticalLayout can be done by either of these 2 approaches:
     - 2.6.1. setting margin: auto to the formlayout.
     - 2.6.2. wrapping formlayout into a div (because formlayout has align-self:stretch), and setting justify content mode and align items for vertical layout to CENTER.
   - 2.7. Set width of the FormLayout to 24em, otherwise, it will show 2 columns by default.
   - 2.8. The shortcut listener can be removed, or be replaced with a workaround `login.addClickShortcut(Key.ENTER).listenOn(password).allowBrowserDefault();`

Now the compile errors are all gone, you can go to localhost:8080 to check the result, it doesn't look as good as the old one, but no worries, it's just the styles are not applied yet, which we will do in the next step.

Step 4: Theming

Now it's time to make the LoginView look better.

You can follow these steps for this exercise:

1. Copy the img folder from src/main/webapp/VAADIN/themes/mytheme to the src/main/webapp folder
2. Create a frontend folder at the project's root directory
3. Create a styles folder under the frontend folder. Create a shared-styles.css file under the styles folder.
4. Copy login-screen related styles from styles.css file into the shared-styles.css file
5. Remove the .mytheme preceding, remove styles that rely on Valo theme, e.g.
   -webkit-animation: valo-animate-in-fade 1s backwards;
   -moz-animation: valo-animate-in-fade 1s backwards;
   animation: valo-animate-in-fade 1s backwards;
6. Add box-sizing: border-box to .login-information, otherwise, you will get annoying scrollbars
7. Apply the styles to LoginView by adding @CssImport(**"styles/shared-styles.css"**)

Styles are now applied, LogView now should look almost the same as the old one.

Step 5: Application Layout

Now let's move onto the MainScreen. It's a HorizontalLayout, with a menu bar on the left side and a view container on the right side, which fits as a RouterLayout perfectly.

You can follow these steps for this exercise:

1. Migrate the AboutView and the SampleCrudView. This should be very similar to the LoginView.
   - 1.1. Add @Route(value=AboutView.*VIEW_NAME*) for the new AboutView
   - 1.2. Add @Route(value=SampleCrudView.*VIEW_NAME*) for the new SampleCrudView
   - 1.3. Don't worry about the styles
   - 1.4. Now you should be able to access the about view in localhost:8080/About, and the crud view in localhost:8080/Inventory
2. Migrate Menu
   - 2.1. Create a flow package under com.vaadin.samples. Copy the Menu class there.
   - 2.2. The Menu class can be greatly simplified, as we don't need a navigator anymore.
   - 2.3. Menu contains an image and title on the top, some menu items in the middle and a logout button at the end. So we can let it extends from VerticalLayout instead of CssLayout.
   - 2.4. Remove all the fields, as we don't need a navigator nor the styling.
   - 2.5. Remove the navigator from the constructor, also the code using navigator or menuPart.
   - 2.6. Start by migrating the top HorizontalLayout. This should be rather easy at this point, just that we don't need a ThemeResource, can use an url of the image instead

     ```
     Image image = new Image("img/table-logo.png", "");
     ```

   - 2.7. We can use a Tabs component to accommodate the menu items.
     - 2.7.1. Set the orientation of the Tabs component to Vertical.
     - 2.7.2. The logout button should stay at the bottom, which means that we need to expand the Tabs, but expand the Tabs itself will make it look ugly, so instead, we can wrap the Tabs into a Div and expand the Div.
   - 2.8. For the logout, instead of a MenuBar, we can simply make a Button.
     - 2.8.1. To reload the page, call UI.*getCurrent*().getPage().reload();
     - 2.8.2. Button expects a Component as Icon, you can get Icon from VaadinIcon by calling the create() method. e.g. VaadinIcon.SIGN_OUT.create()
     - 2.8.3. Remove all other methods except addView(). We can add a RouterLink to a Tab to act as a menu item. The Routerlink should have an Icon and a text.

       ```
       Tab tab = new Tab();
       RouterLink routerLink = new RouterLink(null, viewClass);
       routerLink.setClassName("menu-link");
       routerLink.add(icon, new Span(caption));
       tab.add(routerLink);
       tabs.add(tab);
       ```

   - 2.9. Setup MainLayout
     - 2.9.1. Copy the MainScreen class to the com.vaadin.samples.flow package.

2.9.2. Change the viewContainer to be FlexLayout and extract it as a field.

2.9.3. remove navigator related code

2.9.4. Call setDefaultVerticalComponentAlignment(Alignment.***STRETCH***) so that both the Menu and the view container will have 100% height

2.9.5. Instead of setting expand ratio of view container to be 1, you can just call expand(**viewContainer**);

2.9.6. Let the MainLayout implement RouterLayout, override the showRouterLayoutContent() method so that a view will be displayed inside the view container

```
viewContainer.getElement().appendChild(content.getElement());
```

2.9.7. Copy the beforeEnter logic from HelloWorldPage to MainLayout

2.9.8. Update the @Route annotation is AboutView and SampleCrudView to set up the parent layout so that both views will be displayed inside the view container of Mainlayout

```
@Route(value=AboutView.VIEW_NAME, layout = MainLayout.class)
```

Step 6: Remove Vaadin 8

Now it's time to clean up. The Vaadin 14 application is up running, time to remove all the legacy Vaadin 8 stuff.

You can follow these steps for this exercise:

1. Update the project's POM file:
    1.1. Remove the <vaadin8.version> property.
    1.2. Follows the leads of error indicators in your IDE to remove all the dependencies that use <vaadin8.version> property
2. Remove the widgetset module.
3. Remove the code outside of a flow package in the ui module.
4. Optionally, rename the flow-maven-plugin to the Vaadin 14 vaadin-maven-plugin.

Now you have successfully migrated your legacy application to Vaadin platform.