

Migration from Vaadin 8 to Vaadin 14

Without Multiplatform Runtime

Agenda

- Setup V14
- Exercise 1
- Migrate UI class
- Exercise 2
- Component Comparison
- Layouting
- Exercise 3
- Theming
- Exercise 4
- Application Layout
- Exercise 5
- Remove Legacy Stuff
- Exercise 6

Migration Strategy

Keep both the legacy Vaadin 8 application and the new Vaadin 14 application running at the same time.

Migrate the the legacy application piece by piece.

When the migration is done, remove the legacy application completely.

Set up Vaadin 14 for your application

~~Ivy + Ant~~

Maven

Maven in 5 minutes

Maven setup

It's recommended to have 2 different version properties - one for Vaadin 8, the other for Vaadin 14, e.g. **vaadin8.version** property for Vaadin 8, vaadin.version for Vaadin 14

```
<vaadin.version>14.0.0</vaadin14.version>
```

```
<vaadin8.version>8.6.2</vaadin8.version>
```

Once migration is done, you can remove this



Maven setup

If you don't yet have a `vaadin-bom` dependency, add one like this. If you already have one, make sure the dependency uses your Vaadin 14 version.

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

Maven setup

Update the Vaadin 8 version explicitly for Vaadin 8 dependencies, since the versions cannot be derived from the vaadin-bom of version 14

```
<dependency>
  <groupId>com.vaadin</groupId>
  <artifactId>vaadin-server</artifactId>
  <version>${vaadin8.version}</version>
</dependency>
<dependency>
  <groupId>com.vaadin</groupId>
  <artifactId>vaadin-themes</artifactId>
  <version>${vaadin8.version}</version>
</dependency>
<!-- ... -->
```


Maven setup

Add the `vaadin-core` dependency - it contains all the free components and server side libraries needed from Vaadin 14.

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

Maven setup

Alternatively: If you want to use commercial components like Charts as well, use the `vaadin` dependency instead of `vaadin-core`.

```
<dependency>  
  <groupId>com.vaadin</groupId>  
  <artifactId>vaadin</artifactId>  
</dependency>
```

Maven setup

It's also recommended to add slf4j dependency to have better log information

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

Maven setup

Add `flow-maven-plugin` to handle the npm parts of the build. For a normal Vaadin 14 project, it should be `vaadin-maven-plugin`, but now since `vaadin-maven-plugin` is reserved for our Vaadin 8 app, we'll use `flow-maven-plugin`, which is a copy with a different name.

```
<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>
```

Maven setup

It's also recommended to add slf4j dependency to have better log information

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

npm

To fetch required frontend dependencies (like the client-side implementations of Vaadin components), Vaadin 14 uses the **npm** package manager from **Node.js**. Starting from Vaadin 14.2., the **vaadin-maven-plugin** will install the needed **node** and **npm** executables for you automatically if they're not found from the system. You can also install **node** and **npm** manually yourself - you will need **node** 10.0 or newer and **npm** 5.6 or newer.

Installing npm on Windows

1. Open a browser and go to the Node.js download page at nodejs.org/en/download.
2. Download and run the **Windows Installer** (.msi) for your system.
3. Follow the prompts in the wizard.

Installing npm on macOS

1. If you don't have Homebrew installed already, copy and paste the following into a terminal window:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. After that, to install Node.js, enter the following in a terminal window:

```
brew install node
```


Installing npm on Linux

Chek the instructions based on your distro from <https://vaadin.com/docs/v14/flow/installing/installing-linux.html> .

Note that your distro's package manager may point to an older version of npm by default, so you might need to add a custom repository for Node.js.

Updating the Jetty version

Vaadin 14 requires Jetty 9.4+. If you're using the `jetty-maven-plugin`, remember to update its version in `pom.xml`:

```
<plugin>  
  <groupId>org.eclipse.jetty</groupId>  
  <artifactId>jetty-maven-plugin</artifactId>  
  <version>9.4.19.v20190610</version>  
  <!-- ... -->
```

Or if you're using a property in your `pom.xml`:

```
<properties>  
  <jetty.plugin.version>9.4.19.v20190610</jetty.plugin.version>
```

Test Page

Use a test page to verify Vaadin 14 setup. With this class, navigating to <http://localhost:8080> should show "Hello World!"

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

Context path for the legacy app

- 1) Setup a new context path for the legacy app, like `/v8/*`. The legacy application is thus available under `http://localhost:8080/v8`

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

Context path for the legacy app

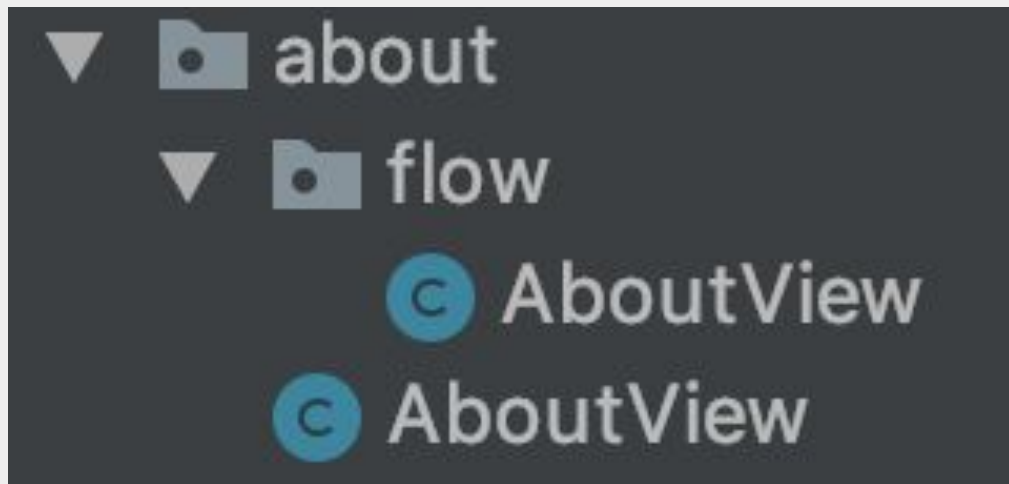
- 1) Setup a new context path for the legacy app, like `/v8/*`. The legacy application is thus available under `http://localhost:8080/v8`
- 2) Both Vaadin 8 and Vaadin 14 want to access resource requests through a specific route, `VAADIN/*` by default. Add URL pattern like `framework/*` and some deployment configuration so the Vaadin 8 app uses the custom path instead.

```
@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) {
        // Redirect the Vaadin 8 app's internal /VAADIN resource requests to /framework
        initParameters.setProperty(Constants.PARAMETER_VAADIN_RESOURCES, "/framework");
        return super.createDeploymentConfiguration(initParameters);
    }
}
```

Recommended: Create packages

To distinguish new classes from old ones, create a sub package like 'flow' for the to-be-converted classes.



Exercise

Step 1

Migrating UI class

Custom UI class is optional

The entry point of a Vaadin 14 application is not a class extended from `com.vaadin.ui.UI` any more.

No explicit UI class is normally needed.

The UI Component still represents the root element of the application, but any custom Vaadin code starts with a base Route.

Data Stored in a UI?

```
public class MyUI extends UI {  
    private EventBus eventBus = new EventBus();  
  
    public static EventBus getEventBus() {  
        return ((MyUI)UI.getCurrent()).eventBus;  
    }  
}
```

```
MyUI.getEventBus().register();
```

Data Stored in a UI

You can use the ComponentUtil helper to set/get data to/from any component, including the current UI

```
public class MyUI /* extends UI */ {  
  
    public static EventBus getEventBus() {  
        if(ComponentUtil.getData(UI.getCurrent(), EventBus.class)==null){  
            ComponentUtil.setData(UI.getCurrent(), EventBus.class, new EventBus());  
        }  
        return ComponentUtil.getData(UI.getCurrent(), EventBus.class);  
    }  
}
```

Logic in a UI class

It's also quite common to have access control in a UI class, like redirecting an unauthenticated user to the login view.

```
if (!isUserSignedIn()) {  
    showLoginView();  
} else {  
    showMainView();  
}
```

Logic in a UI class

BeforeEnterEvent can be used for such purpose

```
@Route("")
public class MainView extends Div implements BeforeEnterObserver {

    @Override
    public void beforeEnter(BeforeEnterEvent event) {
        if (!isUserSignedIn()) {
            event.rerouteTo("login");
        }
    }
}
```

Logic in a UI class

See the **Vaadin14: Router API** training for other navigation events and useful APIs

Exercise

Step 2

Components comparison

Components

GWT -> Web Components

The look and feel is different

Components in Vaadin platform

Some components have the same name, but now in a different package **com.vaadin.flow**

Button, Checkbox, CheckboxGroup, Combobox, FormLayout, Grid, HorizontalLayout, Image, Notification, PasswordField, ProgressBar, RadioButtonGroup, TextArea, TextField, TreeGrid, VerticalLayout, UI, Upload

Components in Vaadin platform

Some components have replacements

CssLayout

Vaadin 8

```
CssLayout cssLayout = new CssLayout();
```

Vaadin 14

```
Div div = new Div();  
  
// or  
FlexLayout flexLayout = new FlexLayout();
```

Div or FlexLayout?

Do you want to apply CSS flexbox properties in Java?

Yes -> **FlexLayout**

No, using CSS is fine OR I just want a simple wrapper layout -> **Div**

Date field

Vaadin 8

```
DateField dateField= new DateField();
```

Vaadin 14

```
DatePicker datePicker = new DatePicker();
```

```
// OR (since 14.2)
```

```
DateTimePicker dateTimePicker = new DateTimePicker();
```

Link

Vaadin 8

```
Link link= new Link();
```

Vaadin 14

```
Anchor anchor = new Anchor();
```

TabSheet

Vaadin 8

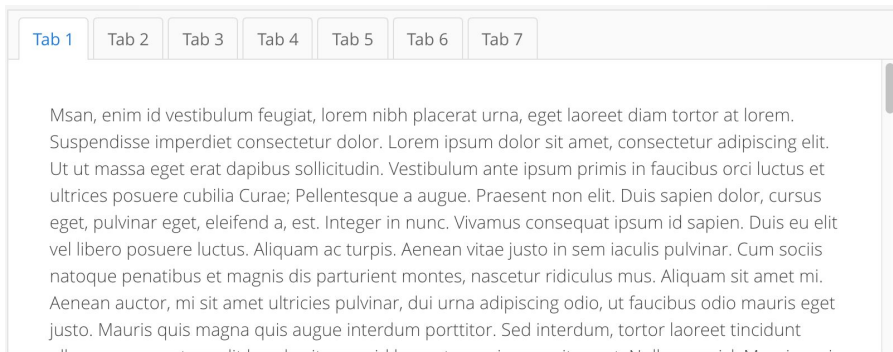
```
TabSheet tabSheet= new TabSheet();
```

Vaadin 14

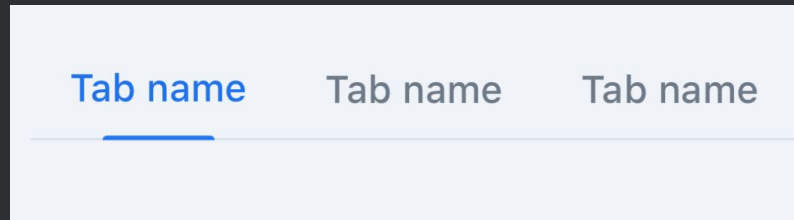
```
Tabs tabs = new Tabs();
```


TabSheet

Vaadin 8



Vaadin 14



Popup

Vaadin 8

```
Window window = new Window();
```

Vaadin 14

```
Dialog dialog = new Dialog();
```

Split layout

Vaadin 8

```
HorizontalSplitPanel split= new HorizontalSplitPanel();
```

```
VerticalSplitPanel split= new VerticalSplitPanel();
```

Vaadin 14

```
SplitLayout splitLayout = new SplitLayout();
```

```
splitLayout.setOrientation(Orientation.HORIZONTAL);
```

```
// or
```

```
splitLayout.setOrientation(Orientation.VERTICAL);
```

Scrollable area

Vaadin 8

```
Panel panel= new Panel();
```

Vaadin 14

```
component.getElement().getStyle().set(
    "overflow", "auto");

// or, since 14.2.
Scroller scroller =
    new Scroller(myContent, ScrollDirection.BOTH);
```

Composite

Vaadin 8

```
public class MyComposite extends  
    CustomComponent {  
    // ...  
    setCompositionRoot(layout);  
}
```

Vaadin 14

```
public class MyComposite extends  
    Composite<Div> {  
  
}
```

Components to display text

Label in Vaadin 14 is based on the HTML **<label>** element. It should be used ONLY in connection to input elements - otherwise use **Paragraph**, **Span** or **Div**.

Components to display text

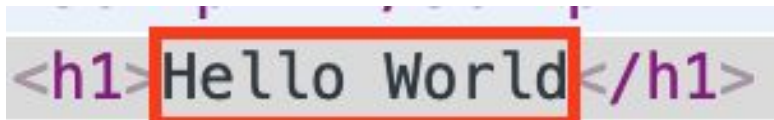
Paragraph: default choice for plain text.

Span or **Div:** when styling is needed.

Span is for small chunks (usually: a single line)

Div is for larger chunks.

There is also the **Text** class, which represents a text node. It should not be used as a standalone component.

A screenshot of a code editor showing the HTML code <h1>Hello World</h1>. The text 'Hello World' is highlighted with a red rectangular box, indicating it is the content of a text node.

```
<h1>Hello World</h1>
```

```
new H1(new Text("Hello World!"));
```


HTML-formatted text

Vaadin 8

```
new Label("<h1>HTML content</h1>",  
          ContentMode.HTML)
```

Vaadin 14

```
new HTML("<h1>HTML content</h1>")
```

Components in Vaadin platform

For the complete list, visit <https://vaadin.com/docs/flow/migration/5-components.html>

API Changes

API Changes

Vaadin 8

`addComponents()` / `addComponent()`

Vaadin 14

`add()`

API Changes

Vaadin 8

`addStyleName()`

`setStyleName()`

Vaadin 14

`addClassName()`

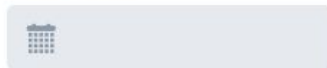
`setClassName()`

Layouting

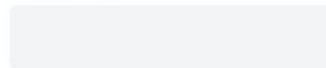
Old Friends

```
Layout layout = new xLayout();  
layout.add(new DatePicker("DateField"));  
layout.add(new TextField("TextField"));  
layout.add(new Select("Combobox"));
```

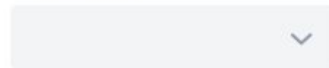
DatePicker

A light gray rectangular widget with a calendar icon on the left side, representing a date picker.

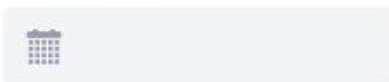
TextField

A light gray rectangular widget representing a text input field.

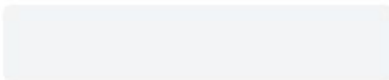
ComboBox

A light gray rectangular widget with a downward arrow icon on the right side, representing a combobox.

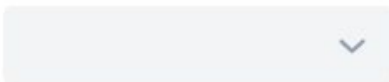
DatePicker

A light gray rectangular widget with a calendar icon on the left side, representing a date picker.

TextField

A light gray rectangular widget representing a text input field.

ComboBox

A light gray rectangular widget with a downward arrow icon on the right side, representing a combobox.

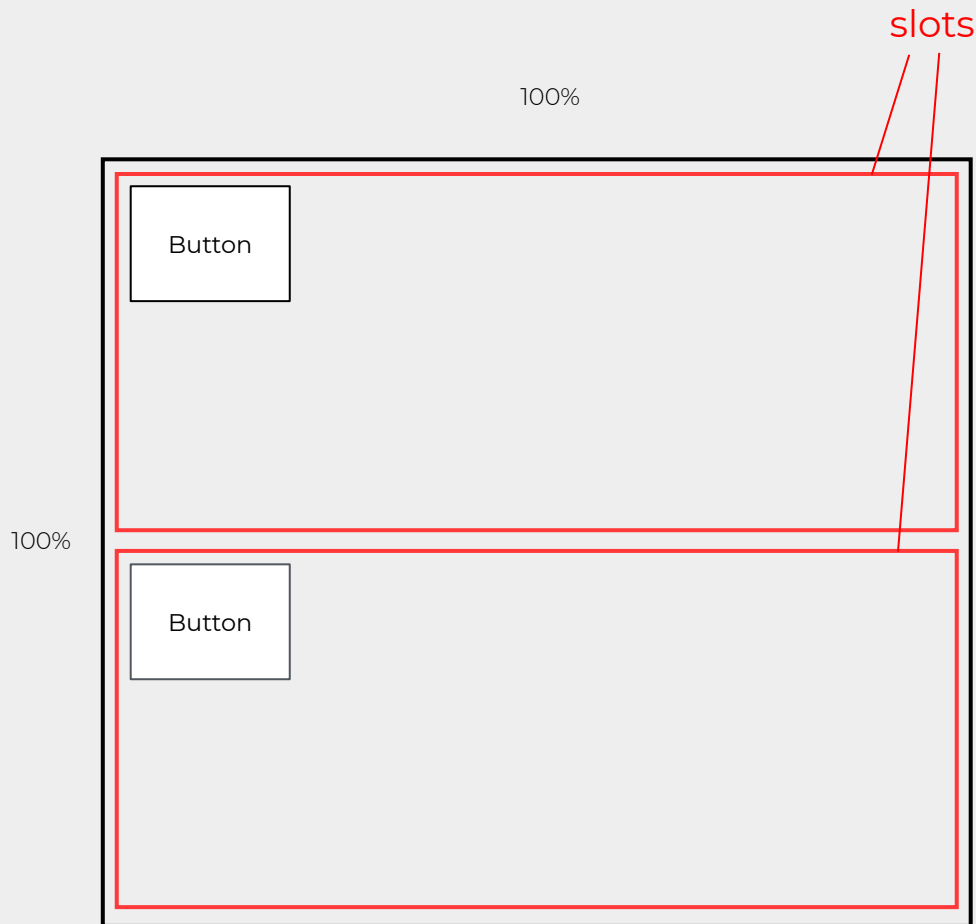


No more layout slots, Vaadin 14
layouting is based on **Flexbox**

Vaadin 8

```
VerticalLayout layout = ...  
layout.setSizeFull();
```

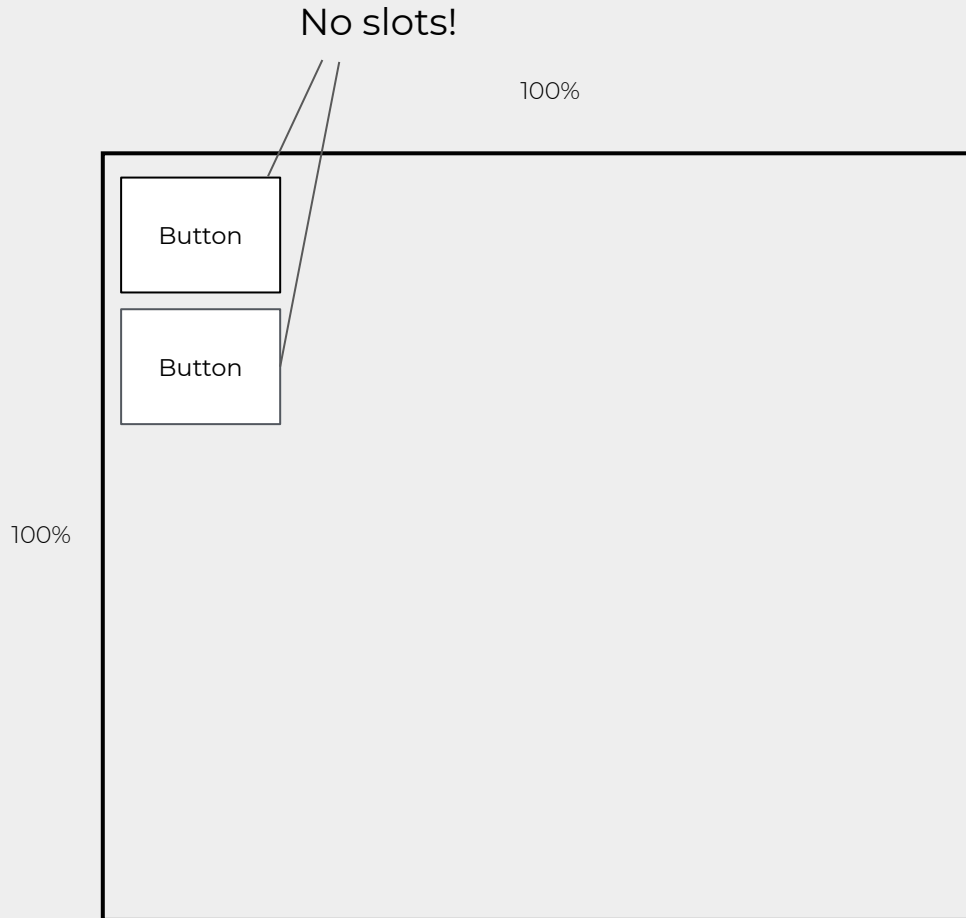
```
Button button1 = ...  
layout.addComponent(button1);  
Button button2 = ...  
layout.addComponent(button2);
```



Vaadin 14

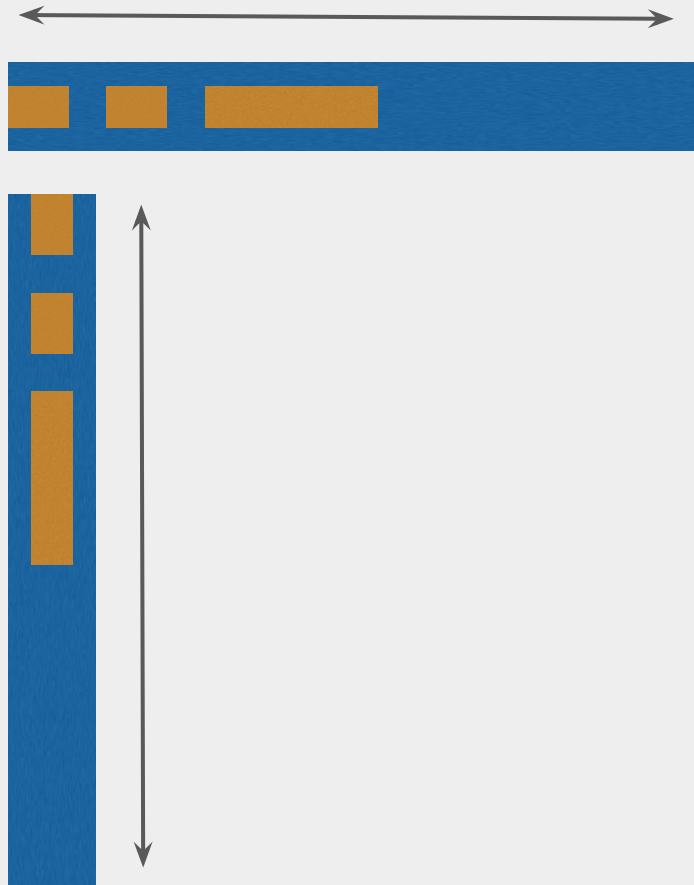
```
VerticalLayout layout = ...  
layout.setSizeFull();
```

```
Button button1 = ...  
layout.add(button1);  
Button button2 = ...  
layout.add(button2);
```



Alignment

justify-content determines where to put items on the **primary axis**.



Alignment

Primary Axis



`justify-content: flex-start`



`justify-content: flex-end`



`justify-content: flex-center`

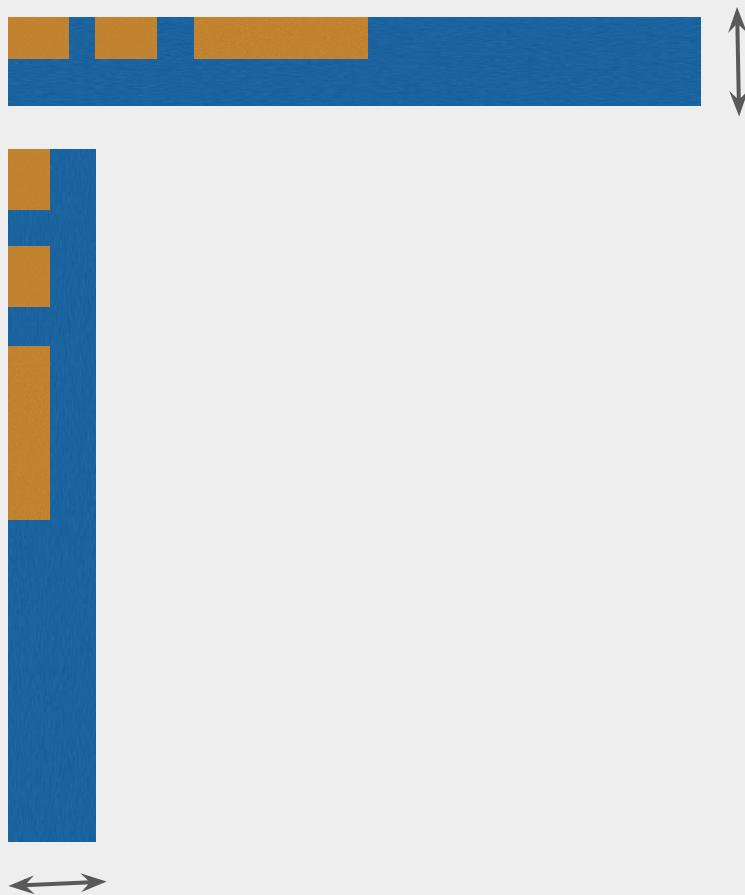
Alignment

Java API for alignment on Primary Axis

```
flexComponent.setJustifyContentMode(FlexComponent.JustifyContentMode.START);
```

Alignment

align-items determines where to put items on the **secondary axis**.



Alignment

Secondary Axis



align-items: flex-start



align-items: flex-end



align-items: flex-center



align-items: stretch

Alignment

Java API for aligning **all** items on Secondary Axis

```
flexLayout.setAlignItems(Alignment.CENTER);  
horizontalLayout.setDefaultVerticalComponentAlignment(Alignment.CENTER);  
verticalLayout.setDefaultHorizontalComponentAlignment(Alignment.CENTER);
```


Alignment

Vaadin 8

```
child1.setHeight("100%");  
child2.setHeight("100%");  
child3.setHeight("100%");
```

Vaadin 14

```
layout.setAlignItems(Alignment.STRETCH);
```



Sizing

```
setWidth()  
setHeight()  
setSizeFull()  
setSizeUndefined()
```

These methods look and mostly work the same, but there are some difference because there are no more slots

Sizing

Note when migrating layouts: relative size (e.g. 100% width/height) is relative to the **parent element** and this affects how Component sizing behaves when there is more than one component inside a Layout.

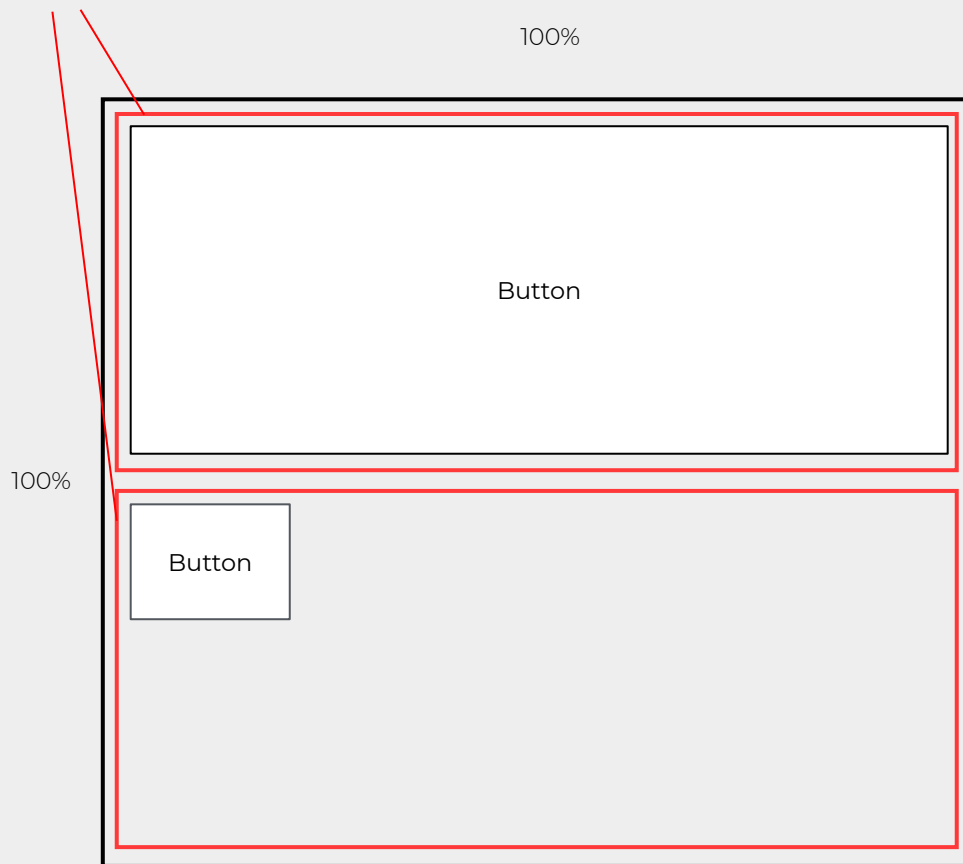
Vaadin 8: sizing is relative to the slot

```
VerticalLayout layout = ...  
layout.setSizeFull();
```

```
Button button1 = ...  
layout.addComponent(button1);  
Button button2 = ...  
layout.addComponent(button2);
```

```
button1.setSizeFull();
```

Parent element is the slot

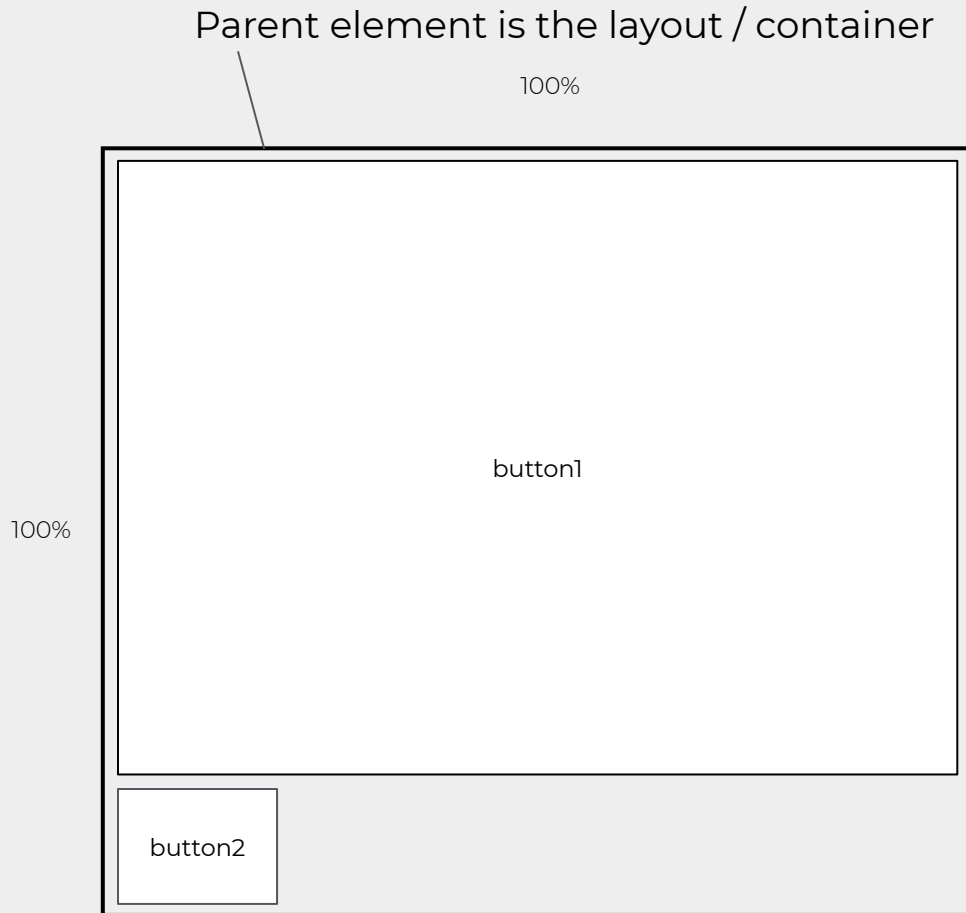


Vaadin 14: sizing is relative to the containing layout

```
VerticalLayout layout = ...  
layout.setSizeFull();
```

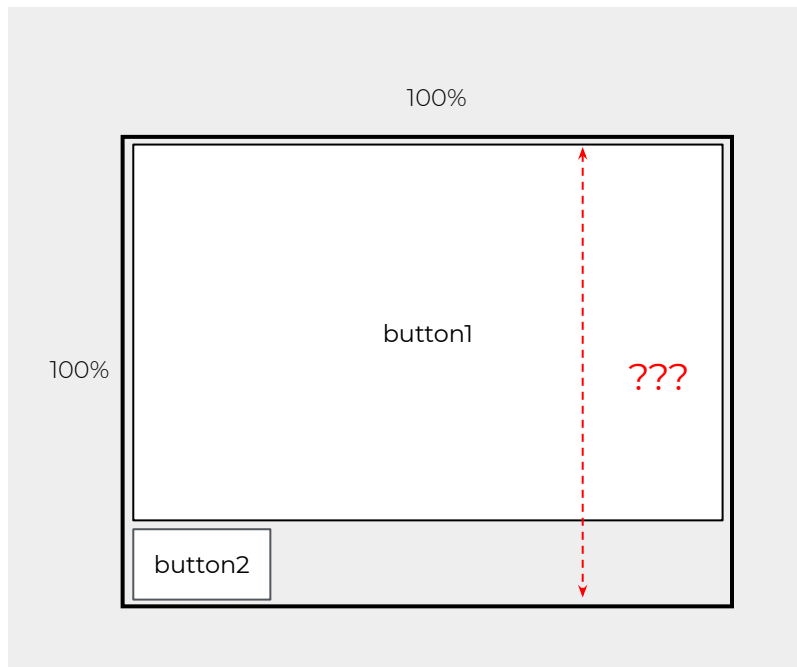
```
Button button1 = ...  
layout.add(button1);  
Button button2 = ...  
layout.add(button2);
```

```
button1.setSizeFull();
```



Sizing

But, shouldn't `button1` have the same height as the parent layout?



Sizing

Good catch. The answer is “no” because size (**both** relative and absolute) is also affected by **flex-shrink** and **flex-grow**

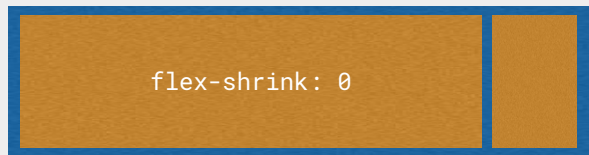
flex-shrink

Default value of `flex-shrink` is 1 -
meaning all items shrink equally to fit
into the container.



flex-shrink

Setting `flex-shrink` to 0 will prevent a child item from shrinking.

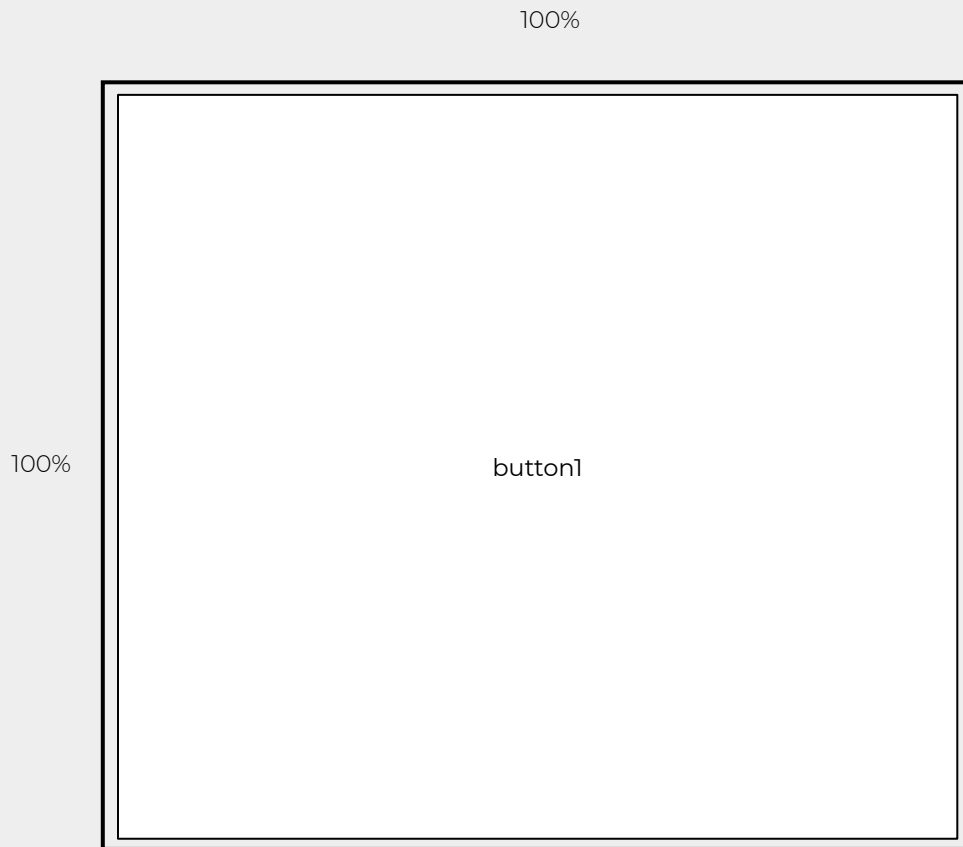


Explicitly disabling flex-shrink

```
VerticalLayout layout = ...  
layout.setSizeFull();
```

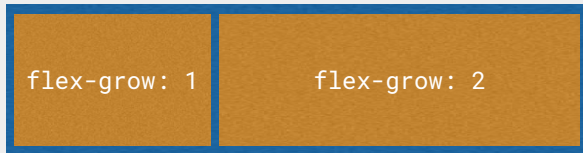
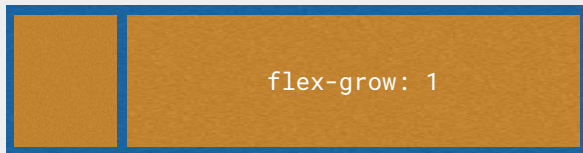
```
Button button1 = ...  
layout.add(button1);  
Button button2 = ...  
layout.add(button2);
```

```
button1.setSizeFull();  
button1.getStyle().set("flex-shrink", 0);
```



flex-grow

How to distribute free space. The default value is 0 - the item doesn't grow even if there is room in the container.



flex-grow (Java API)

```
flexLayout.setFlexGrow(3, childButton);
```

flex-grow shorthand API

```
flexLayout.expand(childButton);  
=  
flexLayout.setFlexGrow(1, childButton);
```

flex-grow vs Vaadin 8 Expand Ratio

Vaadin 14: `setFlexGrow()`

≠

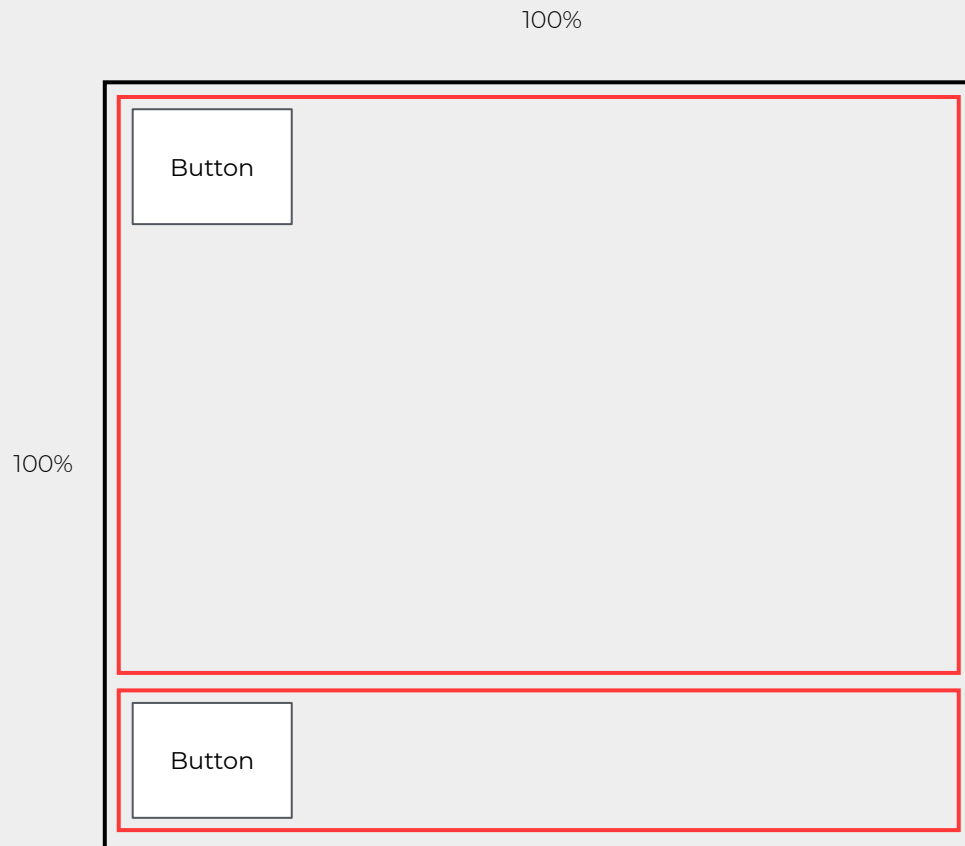
Vaadin 8: `setExpandRatio()`

Vaadin 8: setExpandRatio grows the slot

```
VerticalLayout layout = ...  
layout.setSizeFull();
```

```
Button button1 = ...  
layout.addComponent(button1);  
Button button2 = ...  
layout.addComponent(button2);
```

```
layout.setExpandRatio(button1, 1);
```

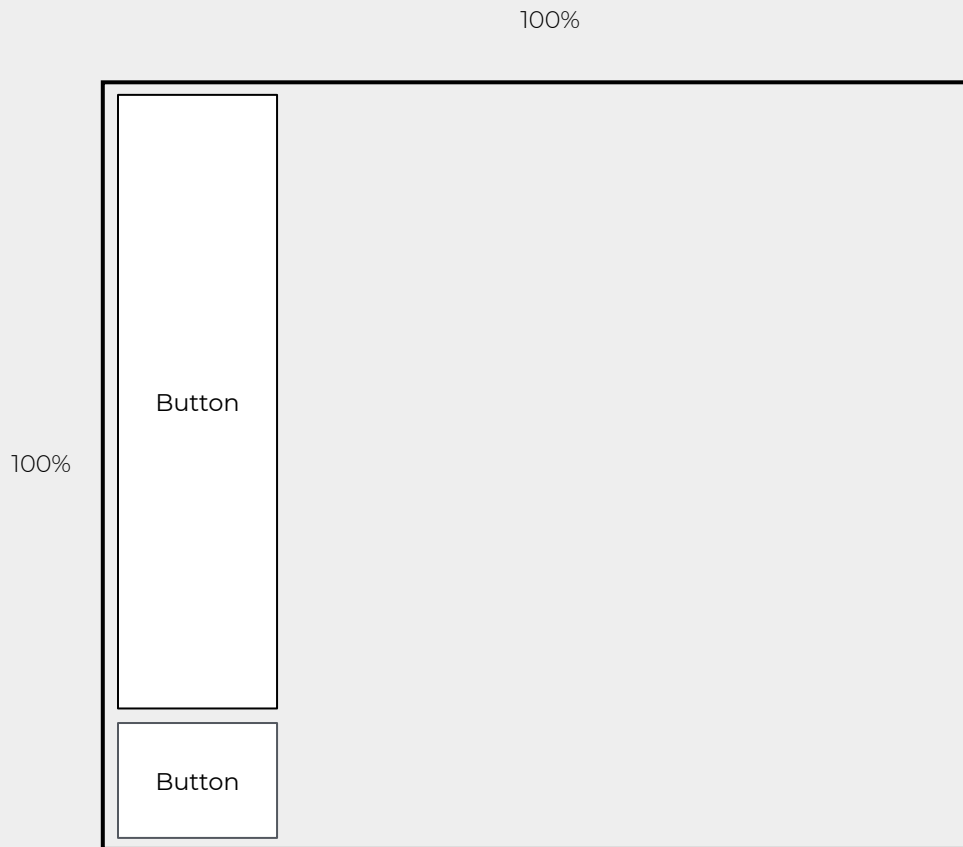


Vaadin 14: expand sets flex-grow to 1

```
VerticalLayout layout = ...  
layout.setSizeFull();
```

```
Button button1 = ...  
layout.add(button1);  
Button button2 = ...  
layout.add(button2);
```

```
layout.expand(button1);
```



Use Case #1

Vaadin 8

```
// expand the last slot  
layout.setExpandRatio(child4, 1);  
layout.setComponentAlignment(child4,  
    Alignment.BOTTOM_RIGHT);
```

Vaadin 14

```
// wrap the last component (create a slot!)  
FlexLayout wrapper = new FlexLayout(child4);  
// expand the wrapper  
layout.expand(wrapper);  
wrapper.setJustifyContentMode(  
    JustifyContentMode.END);
```



Use Case #2

Vaadin 8

```
layout.setExpandRatio(child2, 1);  
child2.setSizeFull();
```

Vaadin 14

```
layout.expand(child2);
```



More layouting

More details about layouting in Vaadin 14 can be found in the Layouting training:

<https://vaadin.com/learn/training/v14-layouting>

Exercise

Step 3

Theming

Theming

Valo -> Lumo

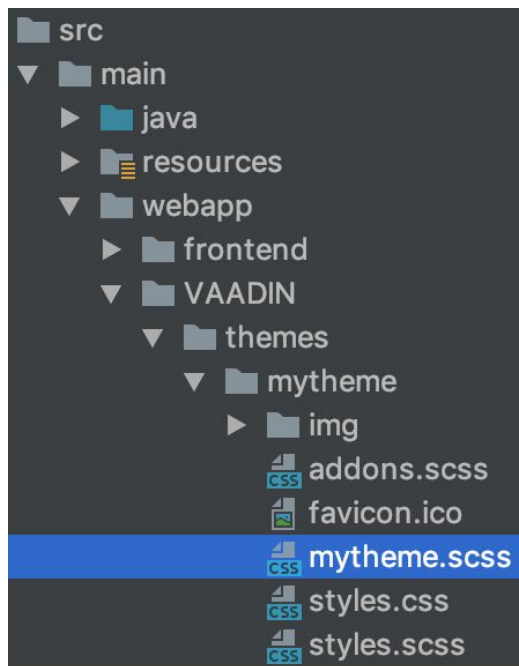
Default theme in Vaadin 14 is Lumo, there's no need to declare it explicitly.

Each Component has been rebuilt from scratch - there's no easy way to replicate the Vaadin 8 look and feel of an application.

The files

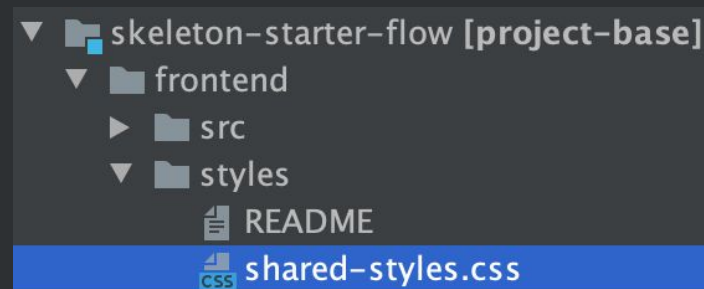
Vaadin 8

src->main->webapp->VAADIN->themes



Vaadin 14

frontend



Style content

Vaadin 8: SCSS

```
@mixin mytheme {  
  @include valo;  
  .login-screen {  
    background: #3b3f42  
  }  
}
```

Vaadin 14: CSS

```
.login-screen {  
  background: #3b3f42;  
}
```


Including to app

Vaadin 8

```
@Theme("mytheme")  
public class MyUI extends UI
```

Vaadin 14

```
@CssImport("styles/shared-styles.css")  
public class MainView extends Div
```

Built in style variants

Vaadin 8

```
btn.addStyleName(VaioTheme.BUTTON_LINK);
```

Vaadin 14

```
btn.addThemeVariants(ButtonVariant.LUMO_TERTIARY);
```

Migrating styles

You can copy some styles from styles.css. Remove the preceding theme selector like `.mytheme`

```
.mytheme .login-screen .login-form {  
  background: white;  
  color: #474747;  
  border-radius: 4px;  
  border: 1px solid #d5d5d5;  
  -webkit-box-shadow: 0 2px 3px rgba(0, 0, 0, 0.05);  
  box-shadow: 0 2px 3px rgba(0, 0, 0, 0.05);  
  border: none;  
  padding: 21px 21px 21px 21px;  
  -webkit-animation: valo-animate-in-fade 1s backwards;  
  -moz-animation: valo-animate-in-fade 1s backwards;  
  animation: valo-animate-in-fade 1s backwards;  
}
```

Migrating styles

Old Valo related styles won't work

```
mytheme .login-screen .login-form {  
  background: white;  
  color: #474747;  
  border-radius: 4px;  
  border: 1px solid #d5d5d5;  
  -webkit-box-shadow: 0 2px 3px rgba(0, 0, 0, 0.05);  
  box-shadow: 0 2px 3px rgba(0, 0, 0, 0.05);  
  border: none;  
  padding: 21px 21px 21px 21px;  
-webkit-animation: valo-animate-in fade 1s backwards;  
-moz-animation: valo-animate-in fade 1s backwards;  
animation: valo-animate-in fade 1s backwards;  
}
```

Migrating styles

Styles with classname `.v-xxx` won't work, because `v-xx` styles rely on the internal structure of V8 components

```
.mytheme .login-screen .centering-layout .v-slot {  
  height: 100%;  
}
```

Grid dynamic styling

Month	Expenses
January	451
February	544
March	369
April	747
May	744
June	482
July	387
August	660
September	436
October	422
November	615
December	396

setClassNameGenerator()

Style the row

```
grid.setClassNameGenerator(expense ->  
    expense.getValue() > 500 ?  
        "warn" : null  
);
```

Month	Expenses
January	342
February	461
March	641
April	324
May	747
June	492
July	670
August	671
September	335
October	557
November	661
December	306

Style a cell

```
Grid.Column column =  
    grid.getColumnByKey("mycolumn");  
  
column.setClassNameGenerator(expense ->  
    expense.getValue() > 500 ?  
        "warn" : null  
);
```

Month	Expenses
January	451
February	544
March	369
April	747
May	744
June	482
July	387
August	660
September	436
October	422
November	615
December	396

Then add a style module for Grid

frontend/styles/my-grid.css

```
.warn {  
  color: red;  
}
```

MainLayout.java

```
@CssImport(value = "styles/my-grid.css", themeFor = "vaadin-grid")  
public class MainLayout
```

More styling

Check the Vaadin 14 Theming training for more details on styling your app.

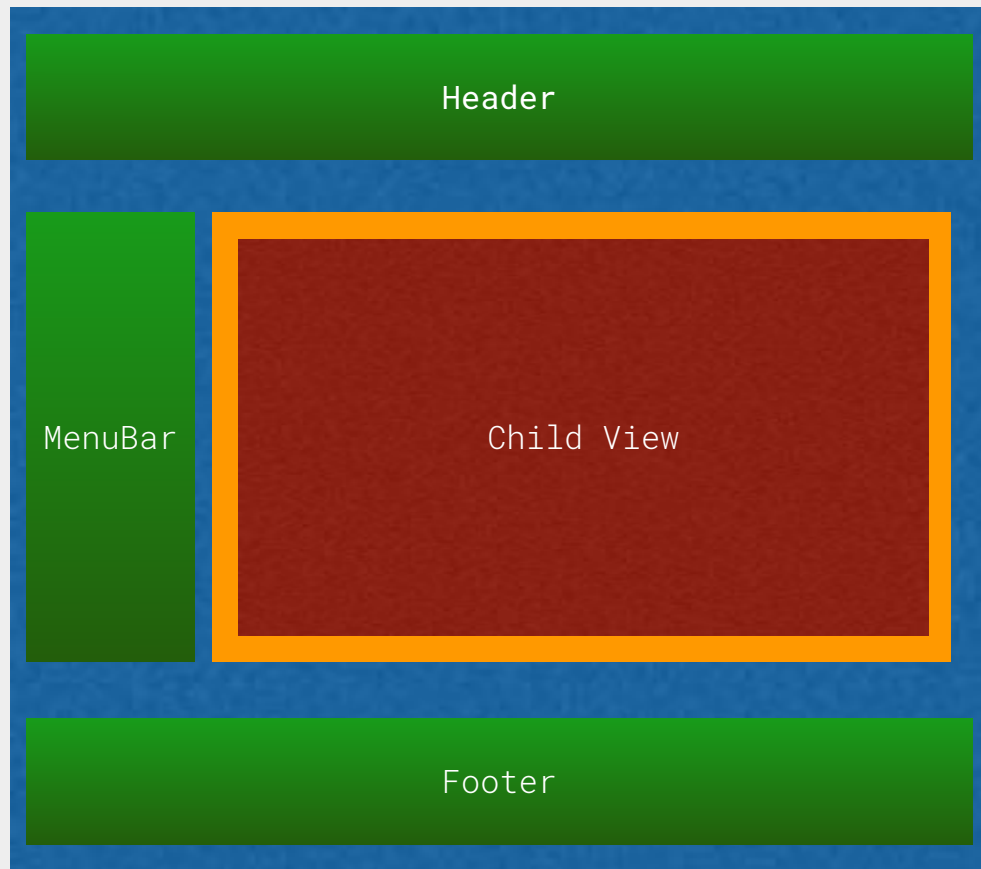
Exercise

Step 4

Application Layout

Application Layout

```
public class MainLayout extends VerticalLayout{  
    private HorizontalLayout header;  
    private HorizontalLayout footer;  
    private VerticalLayout menu;  
    //Vaadin 8  
    private CssLayout viewDisplay;  
    //Vaadin 14  
    private Div viewDisplay;  
}
```



The View

Vaadin 8

```
public class ChildView extends VerticalLayout implements View
```

Vaadin 14

(No more View interface)

```
public class ChildView extends VerticalLayout
```

View Registration

Vaadin 8, with Navigator

```
public class ChildView extends VerticalLayout  
    implements View
```

```
// elsewhere, initializing the Navigator
```

```
Navigator navigator = new Navigator(ui,  
viewDisplay);  
  
navigator.addView("child",ChildView.class);
```

Vaadin 14, with the @Route annotation

```
@Route("child", layout = MainLayout.class)  
public class ChildView extends VerticalLayout
```


Navigate

Vaadin 8

```
UI.getCurrent().getNavigator().navigateTo("child");
```

Vaadin 14

```
UI.getCurrent().navigate("child");  
// or:  
UI.getCurrent().navigate(ChildView.class);
```

Each layout used as a parent layout must implement the **RouterLayout** interface

```
public class MainLayout extends Div implements RouterLayout{  
}
```

The RouterLayout interface has a default method showRouterLayoutContent(), which will append the content as its child.

```
public interface RouterLayout extends HasElement {  
  
    default void showRouterLayoutContent(HasElement content) {  
        if (content != null) {  
            //No need to take care of the old content, Router automatically removes it.  
            getElement().appendChild(content.getElement());  
        }  
    }  
}
```

Override the showRouterLayoutContent() according to your need

```
public class MainLayout extends VerticalLayout implements RouterLayout {  
    private Div viewDisplay = new Div();  
    @Override  
    public void showRouterLayoutContent(HasElement content) {  
        // put all content inside our viewDisplay  
        viewDisplay.getElement().appendChild(content.getElement());  
    }  
}
```

Exercise

Step 5

Removing legacy stuff

Update POM file

Remove `<vaadin8.version>` property from you POM file, also all the Vaadin 8 dependencies

```
<vaadin8.version>8.6.2</vaadin8.version>
```

```
<dependency>
```

```
  <groupId>com.vaadin</groupId>
```

```
  <artifactId>vaadin-server</artifactId>
```

```
  <version>${vaadin8.version}</version>
```

```
</dependency>
```

Update POM file

Remove Vaadin 8 Maven plugin

```
<plugin>  
  <groupId>com.vaadin</groupId>  
  <artifactId>vaadin-maven-plugin</artifactId>  
  <executions>  
    <execution>  
      <goals>  
        <goal>update-theme</goal>  
        <!-- Comment out compile-theme goal to use on-the-fly theme compilation -->  
        <goal>compile-theme</goal>  
      </goals>  
    </execution>  
  </executions>  
</plugin>
```


Update POM file

Update the flow-maven-plugin to the Vaadin 14 maven plugin

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

Remove widgetset

Widgetset is no longer needed

Remove legacy code

Finally we can remove the legacy code outside the 'flow' packages.

Custom component

Custom components need to be rewritten with Polymer, check out

Vaadin 14+: Creating Web components with Polymer

Custom component

Or to use MPR (Multiplatform Runtime) to wrap legacy custom components to Vaadin 10+ application.
check out

Vaadin 14+: Migration with MPR

Exercise

Step 6

Summary

- Setup V10+
- Migrate UI class
- Component Comparison
- Layouting
- Theming
- Application Layout

bit.ly/vaadin-training

