VAADIN TRAINING

Theming

Overview

A Vaadin theme consists of style rules, images and fonts. The theme changes the look and feel of the application, but (generally) not the structure. Responsive design can also be achieved using CSS, but is not covered in this course.

Vaadin includes a single theme by default, Valo. It can be found in the `vaadin-themes` jar file. When using the compatibility components package, Vaadin 8 also supports older themes such as Reindeer and Runo, which are optional dependencies found in the `vaadin-compatibility-themes` jar file.

Built-in theme features

You can customize the visuals of any Vaadin component by modifying it's CSS rules. This can be done in a few ways. Either you can apply styles to a single component by defining an extra style name for it, or you apply it to all components of a certain type by overriding the vaadin-specific style names for that component in CSS.

All vaadin components have CSS style names applied based on this pattern:

```
v-widget v-{component}-{attribute}
```

for instance:

```
v-widget v-label
v-widget v-button-caption
```

To add a style name to a Component you can use either of these methods:

JAVA

```
addStyleName(String)
setStyleName(String)
```

and the result when adding e.g. "`mybuttonstyle`" would be:

```
v-widget v-button mybuttonstyle v-button-mybuttonstyle
```

You can then add CSS to your style file using any of these selectors. You can add as many style names as you want, all of them will be added to the Component so that you can use them.

CSS

```
.mybuttonstyle .v-button-caption {
  font-weight: bold;
}
```

There are some built-in style names you can use in the ValoTheme class. For instance, to make a button more distinct, you could call:

JAVA
```
button.addStyleName(ValoTheme.BUTTON_PRIMARY);
```

This would result in the following selectors being applied:

```
v-widget v-button primary v-button-primary
```

These style names have built-in rules in the Valo theme, making the button look a specific way without the need for the developer to write any CSS. There are many more built-in style names, they are documented in the ValoTheme class.

To use your own style names, you need to define the CSS in your own custom theme. In the style file, you can also achieve the second type of style override, for all components of a specific type. For instance, if you want all TextFields to have a bold font, you can create this CSS rule in your style file:

CSS
```
.v-textfield {
  font-weight: bold;
}
```

Since the rule doesn't have any specific qualifiers, it is applied to all TextFields in your application, including the ones you have defined additional style names for.

Font Icons

With Vaadin, you can use any Resource as a Component icon. This can be a `ThemeResource` such as a image, but we recommend using Font Icons for this purpose. Font Icons are actual font characters that scale extremely well on high-DPI screens and can be fully controlled using CSS rules such as size, color and position. Vaadin includes our own font icon library, Vaadin Icons, that can be used through the `VaadinIcons` helper class.

JAVA
```
inboxLink.setIcon(VaadinIcons.MAILBOX);
```

You can theme the icons as you would any text:

CSS
```
.icon {
  font-size: 16px;
  color: navy;
}
```

More info about Vaadin Icons can be found here: https://vaadin.com/icons

Styling Grid cells

Styling Grid cells and rows is a bit more difficult than just using style names. Grid still has the option for style names, but they cannot distinguish between different cells, for instance. For this, you need to add code that can identify cells or rows and that adds style names for them. With Grid, you can style by row or by column/cell. For styling Rows, you need a `StyleGenerator`:

JAVA

```
Grid<Person> grid = new Grid<>();
grid.getColumn(“age”).setStyleGenerator(item -> "rightalign");
```

The code above adds the style name "rightalign" to all cells in the particular column. You can also easily use the lambda call to check some property in the item itself:

JAVA

```
grid.getColumn(“age”).setStyleGenerator(item -> {
  if (i.getAge() > 42) {
    return "green";
  }
  return "";
});
```

With this, you can target those individual cells from your CSS:

CSS

```
.v-grid-cell.rightalign {
  text-align: right;
}

.v-grid-cell.green {
  color: $green; // SCSS variable for the color
}
```

How to create a Vaadin theme

Typically you don't need to create the base files yourself, since they are created by the various Vaadin Plugins when creating a new project. A Vaadin theme needs to have a single `styles.css` or `styles.scss` file in a certain folder inside your project. The folder should be:

`src/main/webapp/` (depending on your build tool)
`└ VAADIN/themes/mytheme`

The default way a theme is created actually creates two files. The main file is `styles.scss`, that imports all necessary files and acts as the style root, and `mytheme.scss`, that imports the base theme (usually Valo) and where you should add your own style rules. If you are using 3rd party widgets, the plugins will additionally create a addons.scss file with styles from the addons.

To use your own theme, you need to define it in your UI class with either an annotation:

JAVA

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

Or using the `setTheme(String)` method in the UI.

When creating your own theme, remember to test your CSS in all browsers you want to support. There are subtle differences in how each browser renders CSS.

SASS

SASS (Syntactically Awesome Style Sheets) is a language for writing more structured CSS rules. It is a superset of CSS and adds for instance the following:

- block nesting
- variables
- mixins (re-usable snippets with parameters)
- built-in methods (lighten, darken, etc.)
- selector inheritance

You can find out more about SASS at http://sass-lang.com

The Valo theme is built using SASS and uses most of it's features. To extend Valo, it is recommended to use SASS for your own theme as well instead of pure CSS.

Vaadin includes a SASS-to-CSS compiler in it's dependencies (vaadin-theme-compiler jar file). During development, if a `styles.css` file is not present, the compiler will automatically recompile your SASS theme and serve the compiled theme to the browser; this allows for quick development of your theme. When Vaadin is set to Production Mode the compiler is disabled, regardless of the presence of a styles.css file or not. To create a compiled styles.css file for production, you can either run Maven:

```
mvn vaadin:compile-theme
```

Or you can run the Java Application class found in the jar file manually.