

GETTING STARTED WITH

Vaadin 10

WRITTEN BY ALEJANDRO DUARTE, DEVELOPER ADVOCATE, VAADIN

CONTENTS

- > WHAT IS VAADIN
- > COMPONENTS
- > LAYOUT
- > GRID AND LAZY LOADING
- > DATA BINDING
- > ROUTING AND NOTIFICATIONS
- > CUSTOM COMPONENTS
- > ELEMENT API
- > THEMES
- > DEVELOPMENT

WHAT IS VAADIN?

Vaadin is an open-source web development platform for implementing HTML user interfaces (UI) using Java. Vaadin's strength is in its server-side API, Vaadin Flow, which allows not only the composition and implementation of UI components but also the direct manipulation of the Document Object Model (DOM) from the server.

With Vaadin Flow, developers implement the UI in Java running on the server side by using components such as text fields and buttons, arranging them into layouts, and connecting them to business logic through event listeners.

Vaadin Flow provides a variety of high-level ready-to-use UI components optimized for mobile and responsive design. It also includes data binding helpers, URL-based navigation, Web Components integration, and support for HTML templates.

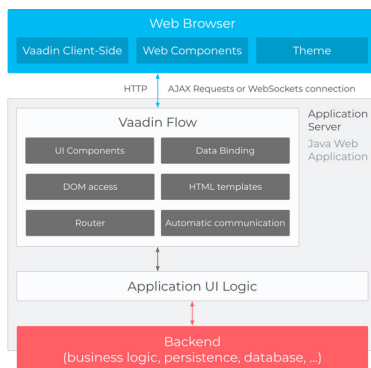


Figure 1: Vaadin client-server architecture

Since modern IDEs can list classes, methods, and even Javadocs, this Refcard is a reference guide to things IDEs can't quickly or directly tell you.

IMPORTANT LINKS

 Website: vaadin.com

 Vaadin Flow's information page: vaadin.com/flow

 Blog: vaadin.com/blog

 Documentation: vaadin.com/docs

 List of components: vaadin.com/components/browse

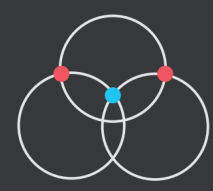
 Components directory: vaadin.com/directory

 Forum: vaadin.com/forum

 GitHub: github.com/vaadin


"HELLO, WORLD" WITH VAADIN FLOW

The following example shows how to implement a simple UI with a text field and a button that, when clicked, invokes server-side logic that adds a greeting text to the layout:



PRIME

Ensure your success with our support.
vaadin.com/pricing





BECOMING A VAADIN EXPERT

While Vaadin has the lowest learning curve on the market, the web application ecosystem keeps evolving continuously.

With our subscription based offerings you'll make sure to always benefit from the newest technologies and stay ahead of the learning curve.

Benefit from the easy integration of our charts library or speed up your development with the visual designer and our various other tools.

Vaadin also offers exceptionally good documentation, tutorials and official hands-on trainings all around the world.

Attend our official Vaadin trainings to learn about Vaadin Best Practices, the right application architecture or how to best benefit from the latest features in browsers or our tools.

We offer a broad selection of over 30 courses that will help you stay a web development expert.

CORE

- ✓ Vaadin Framework
- ✓ Access to 600+ add-ons
- ✓ Discussion forum
- ✓ Extensive documentation

PRO

- ✓ All CORE features
- ✓ Designer
- ✓ TestBench
- ✓ Charts
- ✓ Spreadsheet
- ✓ Board

PRIME

- ✓ All CORE + PRO features
- ✓ Expert Chat
- ✓ Development on Demand
- ✓ Warranty

AVAILABLE FOR PRO AND PRIME



TRAINING SUBSCRIPTION

Unlimited access to expert lead trainings and certifications online and live.

```
@Route("hello-world")
public class HelloWorld extends VerticalLayout {
    public HelloWorld() {
        // two components:
        TextField name = new TextField("Name");
        Button greet = new Button("Greet");

        // add them to "this" layout (a VerticalLayout)
        add(name, greet);

        // add logic through event listeners
        greet.addClickListener(event ->
            add(new Span("Hello, " + name.getValue())));
    }
}
```

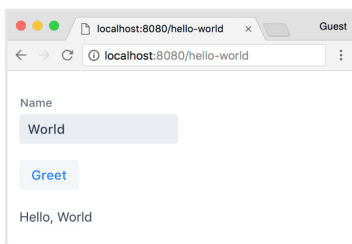


Figure 2: The "hello, world" application rendered as HTML in the browser

You implement user interfaces by extending an existing UI component such as `VerticalLayout` and annotating it with `@Route`. You can build up the UI by combining Vaadin components such as `TextField`, `Button`, `Span`, and many others to which you can add behavior by defining event listeners. The code runs on the server, which allows you to easily connect the UI to business logic. Optionally, you can customize the look and feel with CSS, bind UI components to backend data, manipulate the DOM from the server side in Java, and integrate Web Components.

BOOTSTRAPPING A PROJECT

The easiest way to bootstrap a new Vaadin Flow project is by using the project templates at vaadin.com/start. Select any of the templates for Vaadin version 10 or later, download and extract the zip file, and import it into your favorite IDE.

Alternatively, you can add the `vaadin-core` dependency in any Java web application and start using Vaadin Flow right away. With Maven, you can add the following to the `pom.xml` file:

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

Vaadin Flow requires the Servlet API version 3.1 or later. You can add it with Maven as follows:

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</dependency>
```

COMPONENTS

All Vaadin UI components directly or indirectly extend the `Component` class. You can add visual components and layouts into other layouts. There's always one root component for each view that is mapped to a URL with a `@Route` annotation. Figure 3 shows the `Component` abstract class, interfaces it implements, and some of the relevant methods. All of the 3 interfaces are also functional interfaces.

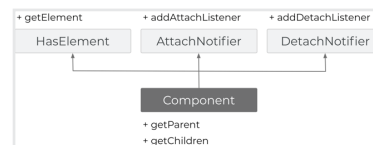


Figure 3: The Component Interface

You'll see this API pattern throughout the framework: abstract and concrete classes implementing *mixin* interfaces to build up the functionality of the component. For example, the `CheckBox` class directly or indirectly implements the `HasSize`, `HasStyle`, `Focusable`, `ClickNotifier`, and `HasValueAndElement` interfaces. You don't need to memorize or even use this interface when creating a UI. However, knowing about them and their methods will help you to understand the features available in the UI components. The following table shows the mixin interfaces, their purposes, and their methods:

OPERATOR	DEFINITION	EXAMPLE
BlurNotifier	Handle blur events	<code>addBlurListener</code>
ClickNotifier	Handle click events	<code>addClickListener</code>
CompositionNotifier	Handle DOM Composition events (text input)	<code>addCompositionStartListener</code> <code>addCompositionUpdateListener</code> <code>addCompositionEndListener</code>
Focusable	Focus, blur, and set tab order (extends HasElement, BlurNotifier, FocusNotifier, and HasEnabled)	<code>setTabIndex</code> <code>getTabIndex</code> <code>focus</code> <code>blur</code>

table continued on next page

FocusNotifier	Handle focus events	addFocusListener
HasAutocapitalize	Enables usage of the autocapitalize attribute	setAutocapitalize getAutocapitalize
HasAutocomplete	Enables usage of the autocomplete attribute	setAutocomplete getAutocomplete
HasAutocorrect	Enables usage of the autocorrect attribute	setAutocorrect isAutocorrect
HasComponents	Add and remove child components (extends HasElements and HasEnabled)	add remove removeAll
HasDataProvider	Enable data providers	setDataProvider setItems
HasElement	Retrieve underlying element	getElement
HasEnabled	Enable or disable components (extends HasElement)	setEnabled isEnabled
HasOrderedComponents	Enable component order (extends HasComponents)	replace indexOf getComponentCount getComponentAt
HasPrefixAndSuffix	Enable prefix and suffix slots for inserting components	setPrefixComponent getPrefixComponent setSuffixComponent getSuffixComponent
HasSize	Enable component sizing	setWidth getWidth setHeight getHeight setSizeFull setSizeUndefined
HasStyle	Enable CSS styles and CSS class names	addClassName removeClassName setClassName getClassName hasClassName addClassNames removeClassNames getStyle

HasText	Enable content text	setText getText
HasValidation	Enable input validation	setErrorMessage getErrorMessage setInvalid isInvalid
HasValue	Enable user-editable values	setValue getValue addValueChangeListener getEmptyValue getOptionalValue isEmpty clear setReadOnly isReadOnly setRequiredIndicatorVisible isRequiredIndicatorVisible
HasValueAndElement	Extends HasValue, HasElement, and HasEnabled	See extended interfaces
InputNotifier	Handle input events	addInputListener
KeyNotifier	Handle keyboard events	addKeyDownListener addKeyPressListener addKeyUpListener
PollNotifier	Handle events for asynchronous UI updates	addPollListener
SortNotifier	Handle sort events	addSortListener

INPUT COMPONENTS

All input components implement `HasSize` (except `ListBox`), `HasStyle`, and, for most of them, `HasValue` and `Focusable`. The following table shows other of the mixin interfaces implemented by each input component:

COMPONENT	MIXIN INTERFACES
Button	ClickNotifier, HasText, Focusable
CheckBox	Focusable, ClickNotifier, HasValueAndElement

table continued on next page

ComboBox	HasValidation, HasDataProvider, Focusable, HasValueAndElement
DatePicker	HasValidation, Focusable, HasValueAndElement
Grid	HasDataProvider, Focusable, SortNotifier
IronList	HasDataProvider, Focusable
ListBox	HasItemsAndComponents, HasDataProvider, HasValueAndElement
TextField	HasSize, HasValidation, HasPrefixAndSuffix, InputNotifier, KeyNotifier, CompositionNotifier, HasStyle, Focusable, HasValueAndElement
TextArea	HasSize, HasValidation, HasPrefixAndSuffix, InputNotifier, KeyNotifier, CompositionNotifier, HasAutocomplete, HasAutocapitalize, HasAutocorrect, HasStyle, Focusable, HasValueAndElement
PasswordField	HasSize, HasValidation, HasPrefixAndSuffix, InputNotifier, KeyNotifier, CompositionNotifier, HasAutocomplete, HasAutocapitalize, HasAutocorrect, HasStyle, HasValueAndElement
RadioButton Group	HasItemsAndComponents, HasDataProvider, HasStyle, HasValueAndElement
Upload	HasSize, HasStyle

HTML COMPONENTS

There are UI components that represent most common elements in HTML and that extend the `HtmlComponent` class, for example: `Div`, `Span`, `Section`, `Header`, `Footer`, `Article`, `H1`, `H2`, `H3`, `H4`, `H5`, `H6`, `Paragraph`, `Label`, `Anchor`, `OrderedList`, `ListItems`, `Image`, and several others.

Tip: If you come from previous versions of Vaadin, note that the `Label` class is mapped to the `label` HTML element which is intended to be a label for an input element. Use components such as `Span`, `Text`, or `Div`, instead of `Label` when migrating from previous versions of Vaadin.

NOTIFICATIONS

You can show notifications using the `Notification` class or its static `show` method. Notifications accept text or UI components:

```
new Notification(
    new VerticalLayout(
        new Span("Terms accepted."),
        new RouterLink("Learn More", TermsView.class)
    )
).open();
```

SIZING

You can adjust the size of any Component that implements `HasSize`. The size of components can be set in fixed or relative units by either dimension (width or height) or be undefined to shrink to fit the content. You can either use explicit size or define it relatively from the area provided by the parent component. Notice that if all the components in a layout have relative size in a particular direction, the layout may not have undefined size in that direction!

METHOD	DESCRIPTION
<code>setWidth</code> <code>setHeight</code>	Set the component size in either fixed units (px, pt, pc, cm, mm, in, em, or rem) or as a relative percentage (%) of the available area provided by the containing layout. The null value or 1 means undefined size (see below), causing the component to shrink to fit the content.
<code>setSizeFull</code>	Sets both dimensions to 100% relative size.
<code>setSizeUndefined</code>	Sets both dimensions as undefined, causing the component to shrink to its minimum size.

LAYOUTS

A UI is built hierarchically from layout components, or more generally component containers, with the actual interaction components as the leaf nodes of the components tree. Some layouts accept multiple components and others a limited number.

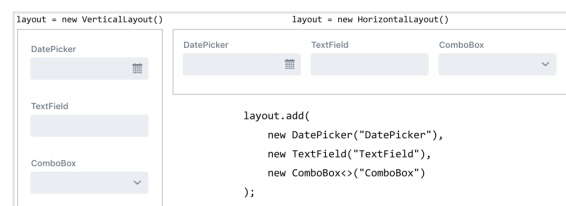


Figure 4: Vertical and horizontal layouts

MARGIN AND PADDING

Certain layouts (those implementing `ThemableLayout`) support setting margin and padding programmatically. Setting `setMargin(true)` and `setPadding(true)` enables all paddings and

margins respectively. The margin and padding sizes can be adjusted with a CSS rule. Example:

The Java class:

```
@HtmlImport("styles/shared-styles.html").
public class MyComponent extends VerticalLayout {
    public MyComponent() {
        addClassName("custom");
        ...
    }
}
```

And the corresponding styles file (src/main/webapp/frontend/styles/shared-styles.html):

```
<custom-style>
<style>
    .custom {
        margin-left: 10px;
        margin-right: 20px;
        margin-top: 30px;
        margin-bottom: 40px;
    }
</style>
</custom-style>
```

SPACING

Certain layout components also support the `setSpacing(true)` method that controls the space between the contained components. Spacing can be adjusted with CSS properties (for example, `--lumo-space-m: 20px`).

ALIGNMENT

Alignment is based in the Flexbox layout mode of CSS. With Flexbox, you have two axis: main axis and cross axis. In a `VerticalLayout`, the main axis is a column that runs from top to bottom (`flex-direction: column`, in CSS). In a `HorizontalLayout`, the main layout is a row that runs from left to right (`flex-direction: row`, in CSS). You can override to `column-reverse` or `row-reverse` with CSS if needed. Since the axes have a direction, you specify the alignment in terms of start/end instead of left/right or top/bottom. To set the alignment in the main axis you can use the `setJustifyContentMode` method.

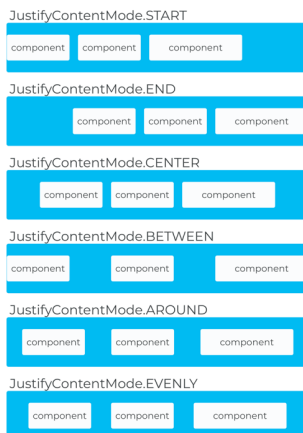


Figure 5: Main axis alignments

To set the alignment in the cross axis you can use the following methods:

LAYOUT	FOR ALL COMPONENTS	FOR INDIVIDUAL COMPONENTS
VerticalLayout	<code>setAlignItems(Alignment)</code> <code>setDefaultHorizontalAlignment(Alignment)</code>	<code>setAlignSelf(Alignment, HasElement)</code> <code>setHorizontalComponentAlignment(Alignment, Component)</code>
HorizontalLayout	<code>setAlignItems(Alignment)</code> <code>setDefaultVerticalAlignment(Alignment)</code>	<code>setAlignSelf(Alignment, HasElement)</code> <code>setVerticalComponentAlignment(Alignment, Component)</code>

GROW (EXPANDING COMPONENTS)

The `setFlexGrow` method allows you to configure how to distribute the space in a layout for each component. The flex grow property specifies the amount of the available space inside the layout a component should take up, proportionally to the other components. For example, if all components have a flex grow value of 1, the remaining space in the layout will be distributed equally to all components inside the layout. If you set the flex grow value of one component to 2, that component will take twice the available space as the other components, and so on.

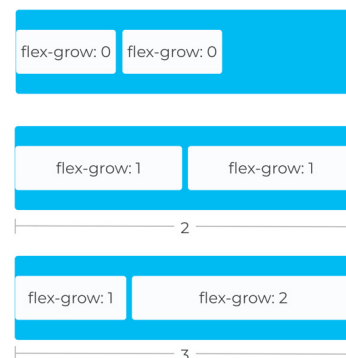


Figure 6: Flex grow example

FORMLAYOUT

`FormLayout` is a responsive layout that helps making forms readable in all screen sizes. The following example sets a `FormLayout` that uses 3 columns for widths of 22em or more, 2 columns for widths between 21em and 22em, and 1 column for widths of less than 21em:

```
formLayout.setResponsiveSteps(
    new ResponsiveStep("0", 1),
    new ResponsiveStep("21em", 2),
    new ResponsiveStep("22em", 3));
```

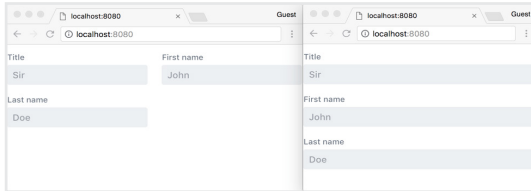


Figure 7: The same form layout rendered in different screen sizes

DIALOG

Dialog is a popup layout you can show in the UI using its open method. For example:

```
Dialog dialog = new Dialog(
    new Span("Press ESC to close"));
dialog.setCloseOnOutsideClick(false);
dialog.open();
```

GRID AND LAZY LOADING

Grid, the component that shows your data in a tabular format, often plays a central part in business applications. The easiest way to use the **Grid** component is to pass the data as a list, stream, or array. In the following code example, **Person** is the domain object (a Java Bean):

```
Grid grid = new Grid<>(Person.class);
grid.setItems(listOfPersons);
// define columns and the order as bean properties
// (default: show all)
grid.setColumns("name", "email");
```

Alternatively, you can define columns programmatically. In this case, you don't need to pass the domain type as a constructor parameter:

```
Grid grid = new Grid<>();
grid.setItems(people);
grid.addColumn(Person::getName)
    .setCaption("Name");
...
```

The **addColumn** method returns a **Column** object that can be used to further configure the column and the data representation. In the above example, we only configure the caption of the column. With bean-based listing, you can get a reference to the **Column** with **getColumn("propertyName")**.

The **setItems** method stores the given data in your server memory for rapid access. If your data grid contains a lot of data and you wish to save server memory, you can also do a lazy binding with your backend using the **setDataProvider** method:

```
grid.setDataProvider(DataProvider.fromCallbacks(
    query -> service.findAll(query.getOffset(),
        query.getLimit()),
    query -> service.count()
));
```

The example above uses the easiest solution, in which you only pass two lambda expressions: the first to provide the given range of items, and the second to provides the total number of items available. Alternatively, you can provide a custom implementation of the **DataProvider** interface.

DATA BINDING

Binder is a helper class that allows you to bind Java properties to UI components so that the input made by the end user automatically gets written to your domain model. The Binder class also helps you to convert values and validate the data. The following binds **firstName (String)** and **age (Integer)** to two text fields:

```
Binder<Person> b = new Binder<>();
b.forField(firstNameField)
    // additional configuration
    .asRequired("First name must be defined")
    .bind(Person::getFirstName, Person::setFirstName);
b.forField(ageField)
    .withNullRepresentation("")
    .withConverter(new StringToIntegerConverter(
        "Must be valid integer !"))
    .withValidator(integer -> integer > 0,
        "Age must be positive")
    .bind(p -> p.getAge(), (p, i) -> p.setAge(i));
b.setBean(person);
```

You can also use the **bindInstanceFields** method to bind all the Java member fields that are also Vaadin fields (those that extend **HasValue**):

```
public class Form extends VerticalLayout {
    private TextField name = new TextField("Name");
    @PropertyId("phoneNumber") // optional
    private TextField phone = new TextField("Phone");

    Form form = new Form();
    ...
    binder.bindInstanceFields(form);
}
```

ROUTING AND NAVIGATION

You can link UI components to URLs using the **@Route** annotation. Optionally, you can specify a layout (that must implement **RouterLayout**) where the annotated component should be rendered, and implement **HasURLParameter** if the view accepts parameters:

```
@Route(layout=MainLayout.class)
public class GreetingView extends VerticalLayout
    implements HasUrlParameter<String> {
    @Override
    public void setParameter(BeforeEvent event,
                            String parameter) {
        add(new Span("Hello, " + parameter));
    }
}
```

To pass the value world as a parameter, you can request the application with a URL like `http://yourdomain.com/greet/world`. The route name is derived from the class name removing any trailing "View". If the class is named `MainView` or `View`, it will be mapped to a root URL (""). You can override this by using the value property of the `@Route` annotation.

The following interfaces allows you to interact with the navigation lifecycle. You can implement observers in a UI component or manually register listeners using the UI class.

INTERFACE	DESCRIPTION
BeforeLeaveObserver BeforeLeaveListener	ClickNotifier, HasText, Focusable
BeforeEnterObserver BeforeEnterListener	Allows changing the navigation to a different destination
AfterNavigationObserver AfterNavigationListener	Allows updating parts of the UI once the navigation has been completed

CUSTOM COMPONENTS

You can create custom components by using existing `components` and extending `Composite`, which allows you to hide the API of the actual root level component:

```
public class CustomTextField extends Composite<Div> {
    private Label label;
    private Input input;

    public CustomTextField(String caption, String value) {
        label = new Label();
        label.setText(caption);
        input = new Input();
        input.setValue(value);

        getContent().add(label, input);
    }
}
```

Implementing custom input fields is done by extending the `AbstractCompositeField`, `AbstractSinglePropertyField`, or `AbstractField` depending on your requirements.

To implement component "containers," implement the `HasComponents` interface. It includes default implementations for `add(Component...)` and `remove(Component...)` that you can override if needed:

```
@Tag("div")
public class CustomContainer extends Component
    implements HasComponents {
    @Override
    public void add(Component... components) {
        for (Component c : components) {
            getElement().appendChild(c.getElement());
        }
    }
}
```

ELEMENT API

The *Element API* of Vaadin Flow allows you to manipulate the DOM from server-side Java. Most often you should use the Element API to create custom components or do small modifications to existing ones. Since the `Component` class implements the `HasElement` interface, all UI components have an underlying `Element` object that you can get with `component.getElement()`. The `Element` class represents an HTML element in the DOM and contains methods to update and query various parts of an element, such as attributes and child elements.

The following example shows how to create an input field with a companion label and how to set properties and styles:

```
Element label = ElementFactory.createLabel("Name:");
label.setProperty("for", "name");

Element input = ElementFactory.createInput();
input.setProperty("id", "name");
input.setProperty("value", "Jane");
input.getStyle().set("font-size", "28px");
input.getStyle().set("margin-left", ".5em");
input.getStyle().set("background-color", "deepskyblue");
input.getStyle().set("color", "#fff");

Element div = ElementFactory.createDiv();
div.appendChild(label, input);
```

Notice that an `Element` is not a `Component`. You can however add `Elements` into `Components` as follows:

```
VerticalLayout layout = new VerticalLayout();
layout.getElement().appendChild(div);
```

HTML TEMPLATES

You can define the UI, or parts of it using, HTML:


```
<link rel="import" href="../../bower_components/polymer/
polymer-element.html">
<link rel="import" href="../../bower_components/
vaadin-text-
field/vaadin-text-field.html">
<link rel="import" href="../../bower_components/vaadin-
button/vaadin-button.html">
<dom-module id="example-template">
  <template>
    <vaadin-text-field label="Name:" value="{{name}}"></
vaadin-text-field>
    <vaadin-button onclick="{{buttonClicked}}">Send</
vaadin-button>
  </template>
<script>
  class ExampleTemplate extends Polymer.Element {
    static get is() {
      return 'example-template'
    }
  }
  customElements.define(ExampleTemplate.is,
ExampleTemplate);
</script>
</dom-module>
```

Tip: You can visually edit HTML templates in Eclipse or IntelliJ with Vaadin Designer.

The previous template imports the Polymer library and two Web Components: `vaadin-text-field` and `vaadin-button`. These are static resources packaged as a WebJAR and included with Vaadin (from a Bower-based installation).

The name value and `buttonClicked` method can be used server-side as follows:

```
@Tag("example-template")
@HtmlImport("src/example-template.html")
public class ExampleTemplate extends
PolymerTemplate<ExampleModel> {
  public interface ExampleModel extends TemplateModel {
    void setName(String value);
    String getName();
  }
  @EventHandler
  public void buttonClicked() {
    Notification.show("Hello, " + getModel().getName());
  }
}
```

The `ExampleModel` interface defines getters and setters for the HTML attributes defined in the template (for example `{{name}}` in the HTML file).

THEMES

Vaadin separates the appearance of the user interface from its logic using themes. Theming is done, as with any HTML content, by using

CSS. Vaadin 10 includes the Lumo theme. It has a light (default) and a dark variant:

```
@Route(value = "")
@Theme(value = Lumo.class, variant = Lumo.DARK)
public class DarkApplication extends Div {
}
```

UI components have variants you can set with the Element API:

```
Button button = new Button("Save");
button.getElement().setAttribute("theme", "contrast
primary");
```

CUSTOM STYLES

Since Vaadin components are based in Web Components, you have to put the CSS in an HTML file and load it with the

`@HtmlImport` annotation:

```
@HtmlImport("frontend://styles/shared-styles.html")
public class MainLayout extends Div implements
RouterLayout {
}
```

In the `shared-styles.html` file (located in `src/main/webapp/frontend/styles`) you can add any CSS rules as follows:

```
<custom-style>
  <style>
    ... CSS here ...
  </style>
</custom-style>
```

Lumo defines CSS custom properties that allow you to quickly modify the look and feel of an app:

```
<custom-style>
  <style>
    html {
      --lumo-font-family: Courier, monospace;
      --lumo-border-radius: 20px;
      --lumo-base-color: #fafbfc;
      --lumo-primary-color: #00b4f0;
    }
  </style>
</custom-style>
```

You can add CSS class names to any UI component using the `addClassName` and `addClassNames` methods:

```
addClassName("custom");
```

This class name can be used in a selector in the `shared-styles.html` file:

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

DEPLOYMENT

To deploy a Vaadin Flow application in a production environment, you must add the `vaadin-maven-plugin` in a Maven profile in the `pom.xml` file and compile the project specifying the name of the profile (for example, `mvn clean package -PproductionMode`). This performs three main tasks:

- Transpilation: Converts ES6 JavaScript to ES5 JavaScript to support older browsers.
- Minimization: Makes JavaScript files smaller.
- Bundling: Bundling: Stitching together a group of modules into a single file.

For details about how to add the `vaadin-maven-plugin`, see vaadin.com/docs/v10/flow/production/tutorial-production-mode-basic.html.

In order to use the resources created by the `vaadin-maven-plugin`, you have to enable production mode using the `@VaadinServletConfiguration` annotation as follows:

```
@WebServlet(urlPatterns = "/*", asyncSupported = true,
  initParams = { // optional
    @WebInitParam(name = "frontend.url.es6", value =
      "http://mydomain.com/es6/"),
    @WebInitParam(name = "frontend.url.es5", value =
      "http://mydomain.com/es5/") }
)
@VaadinServletConfiguration(productionMode = true)
public static class MainVaadinServlet extends
  VaadinServlet {}
```

You can also use system properties to override the `@WebInitParam` annotations. For example, you can pass the properties to the Jetty Maven Plugin as follows:

```
mvn jetty:run -Dvaadin.frontend.url.es6=http://mydomain.
com/es6/ -Dvaadin.frontend.url.es5=http://mydomain.com/es5/
```

Notice that if you run the application using the Jetty Maven Plugin in production mode, you have to run it using `mvn jetty:run-exploded`.



Written by Alejandro Duarte, Developer Advocate, Vaadin

Alejandro Duarte is a Developer Advocate at Vaadin Ltd and author of "Data-Centric Applications with Vaadin 8" and "Vaadin 7 UI Design by Example: Beginner's Guide." He is passionate about Java technologies and game development with C++. When not coding, Alejandro splits his free time between his family, his beautiful girlfriend, and his passion for the electric guitar. If you are feeling social, you can follow him on Twitter: [@alejandro_du](https://twitter.com/alejandro_du).



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more. "DZone is a developer's dream," says PC Magazine.

DZone, Inc.

150 Preston Executive Dr. Cary, NC 27513

888.678.0399 919.678.0300

Copyright © 2017 DZone, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.