Vaadin Cheat Sheet

A quick reference guide for Vaadin, covering essential components, layouts, data binding, and themes.

Core Components

Basic UI Components

`Button`	A clickable button that triggers an action.
	`Button button = new Button("Click Me");`
	`button.addClickListener(event -> {`
	`  Notification.show("Button Clicked!");`
	`});`

`TextField`	A text input field for user input.
	`TextField textField = new TextField("Enter Text:");`
	`textField.addValueChangeListener(event -> {`
	`  System.out.println("Text changed: " + event.getValue());`
	`});`

`TextArea`	A multi-line text input area.
	`TextArea textArea = new TextArea("Enter Description:");`
	`textArea.setRows(5);`

`Label`	Displays static text. `Label label = new Label("This is a label.");`

`Checkbox`	A boolean selection component.
	`Checkbox checkbox = new Checkbox("Agree to terms");`
	`checkbox.addValueChangeListener(event -> {`
	`  System.out.println("Checkbox state: " + event.getValue());`
	`});`

`ComboBox`	A dropdown list for selecting from predefined options.
	`ComboBox<String> comboBox = new ComboBox<>("Select Option:");`
	`comboBox.setItems("Option 1", "Option 2", "Option 3");`

Data Input Components

`DatePicker`	Allows users to select a date from a calendar.
	`DatePicker datePicker = new DatePicker("Select Date:");`
	`datePicker.setValue(LocalDate.now());`

`TimePicker`	Allows users to select a specific time.
	`TimePicker timePicker = new TimePicker("Select Time:");`
	`timePicker.setValue(LocalTime.now());`

`Upload`	Allows users to upload files.
	`Upload upload = new Upload("Upload File");`
	`upload.addSucceededListener(event -> {`
	`  Notification.show("File uploaded successfully!");`
	`});`

Layouts

Layout Managers

`VerticalLayout`	Arranges components vertically.
	`VerticalLayout layout = new VerticalLayout();`
	`layout.add(new Label("Label 1"), new Label("Label 2"));`

`HorizontalLayout`	Arranges components horizontally.
	`HorizontalLayout layout = new HorizontalLayout();`
	`layout.add(new Button("Button 1"), new Button("Button 2"));`

`FlexLayout`	A flexible layout that allows for complex arrangements.
	`FlexLayout layout = new FlexLayout();`
	`layout.setFlexGrow(1, component);`

`GridLayout`	Arranges components in a grid.
	`GridLayout layout = new GridLayout(2, 2);`
	`layout.add(new TextField(), new TextField());`
	`layout.add(new Button(), new Button());`

`FormLayout`	Arranges components in a form-like structure, typically for data entry.
	`FormLayout layout = new FormLayout();`
	`layout.addFormItem(new TextField(), "Name");`
	`layout.addFormItem(new EmailField(), "Email");`

Layout Properties

`setMargin(boolean)`	Adds margin around the layout. `layout.setMargin(true);`
`setPadding(boolean)`	Adds padding inside the layout. `layout.setPadding(true);`
`setSpacing(boolean)`	Adds spacing between components in the layout. `layout.setSpacing(true);`
`setWidth(String)`	Sets the width of the layout (e.g., "100%", "500px"). `layout.setWidth("100%");`
`setHeight(String)`	Sets the height of the layout (e.g., "300px"). `layout.setHeight("300px");`

Data Binding

Binder API

`Binder<T>`	Binds UI components to data fields in a Java bean. `Binder<Person> binder = new Binder<>(Person.class);`
`bind(Component, String)`	Binds a component to a field in the bean. `binder.bind(textField, "firstName");`
`readBean(T)`	Reads the values from the bean and sets them to the bound components. `binder.readBean(person);`
`writeBean(T)`	Writes the values from the components back to the bean. `binder.writeBean(person);`
`validate()`	Validates the bound values. `ValidationResult result = binder.validate();`

Validation

`@NotNull`	Bean Validation annotation to ensure a field is not null.
	`@NotNull`
	`private String firstName;`

`@Size(min, max)`	Bean Validation annotation to specify the size of a string field.
	`@Size(min = 2, max = 50)`
	`private String lastName;`

`EmailField`	A specialized text field for email input with built-in validation. `EmailField emailField = new EmailField("Email");`

Theming

Styling with CSS

`@Theme(themeName)`	Annotation to define the theme for a Vaadin view.
	`@Theme("mytheme")`
	`public class MyView extends VerticalLayout {`
	`  ...`
	`}`

Theme Folder	CSS files are located in the `themes/mytheme/styles.css` directory.

`@CssImport(url = "./styles/my-styles.css")`	Import CSS files directly into a component.
	`@CssImport(url = "./styles/my-styles.css")`
	`public class MyComponent extends Div {`
	`  ...`
	`}`

CSS Properties	Use standard CSS properties to style components, such as `color`, `background-color`, `font-size`, etc.

Custom Themes

Create custom themes by defining your own CSS styles and applying them to components. This allows for complete control over the look and feel of your Vaadin application.
Vaadin provides a set of built-in themes that can be customized or extended to create your own unique theme.