Layouting

Vaadin 14

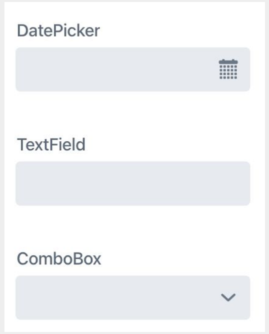
Agenda

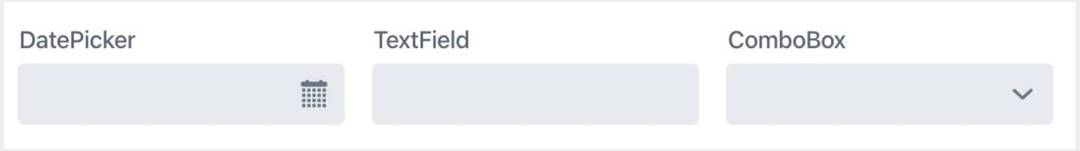
- HorizontalLayout & VerticalLayout
- FlexLayout
- Exercise 1
- FormLayout
- Exercise 2
- Vaadin Board
- Exercise 3
- App Layout

HorizontalLayout & VerticalLayout

VerticalLayout

HorizontalLayout





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

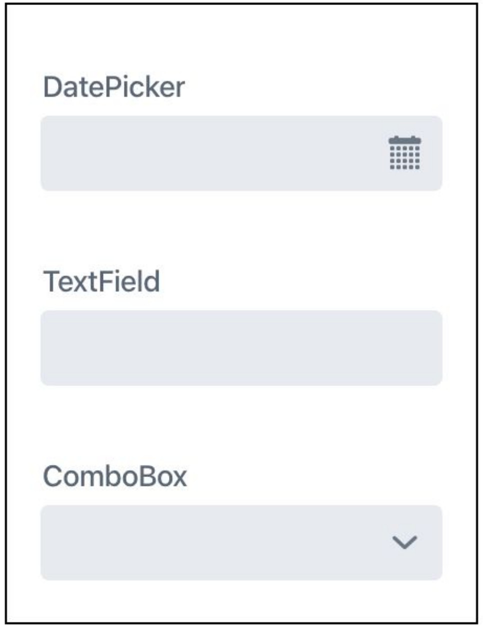
Padding

Padding means space around the **inner** side of the border of the layout.

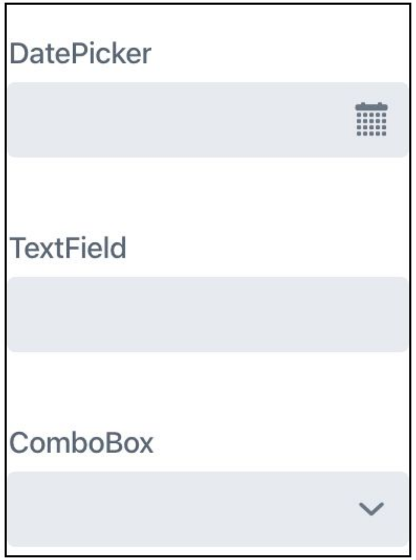
Padding can be turned on and off with setPadding()

```
layout.setPadding(false);
layout.setPadding(true);
```

With Padding



Without Padding

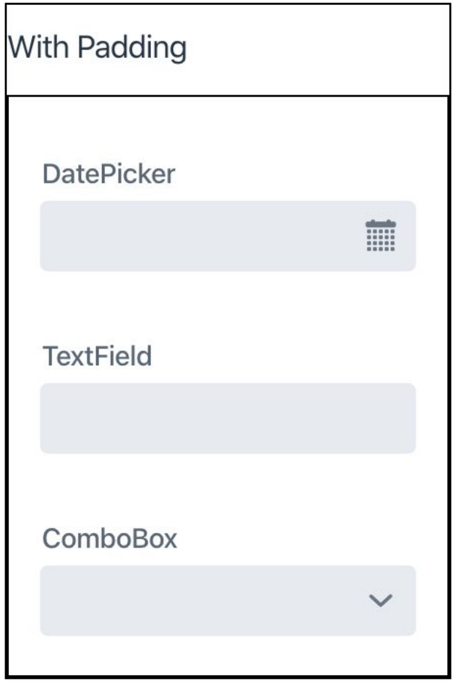


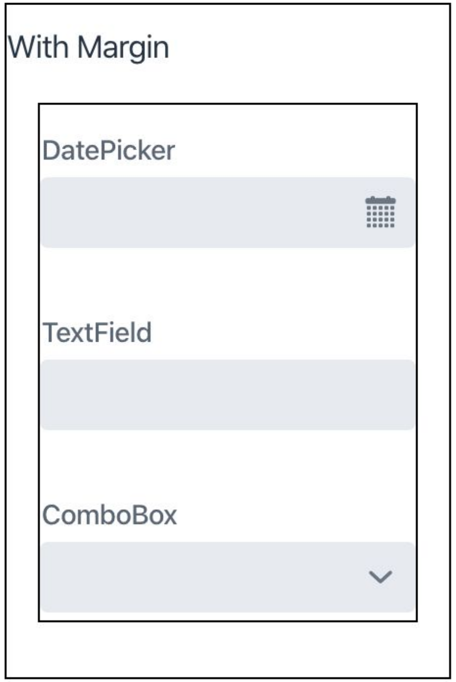
Margin

Margin means space around the **outer** side of the border of the layout.

Margin can be turned on and off with setMargin()

```
layout.setMargin(false);
layout.setMargin(true);
```



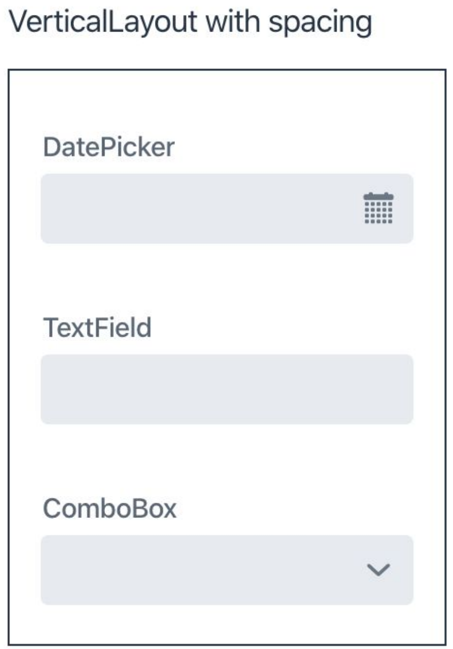


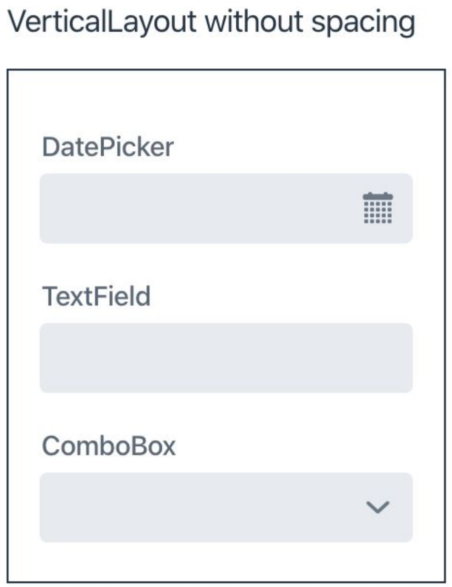
Spacing

Spacing means the space between the components in the layout.

Can turn the spacing on and off with

```
layout.setSpacing(false);
layout.setSpacing(true);
```





Margin vs Padding

Padding is part of the layout while margin is outside the layout, So if you set the background color to the layout, padding will extend the background color while margin will not.

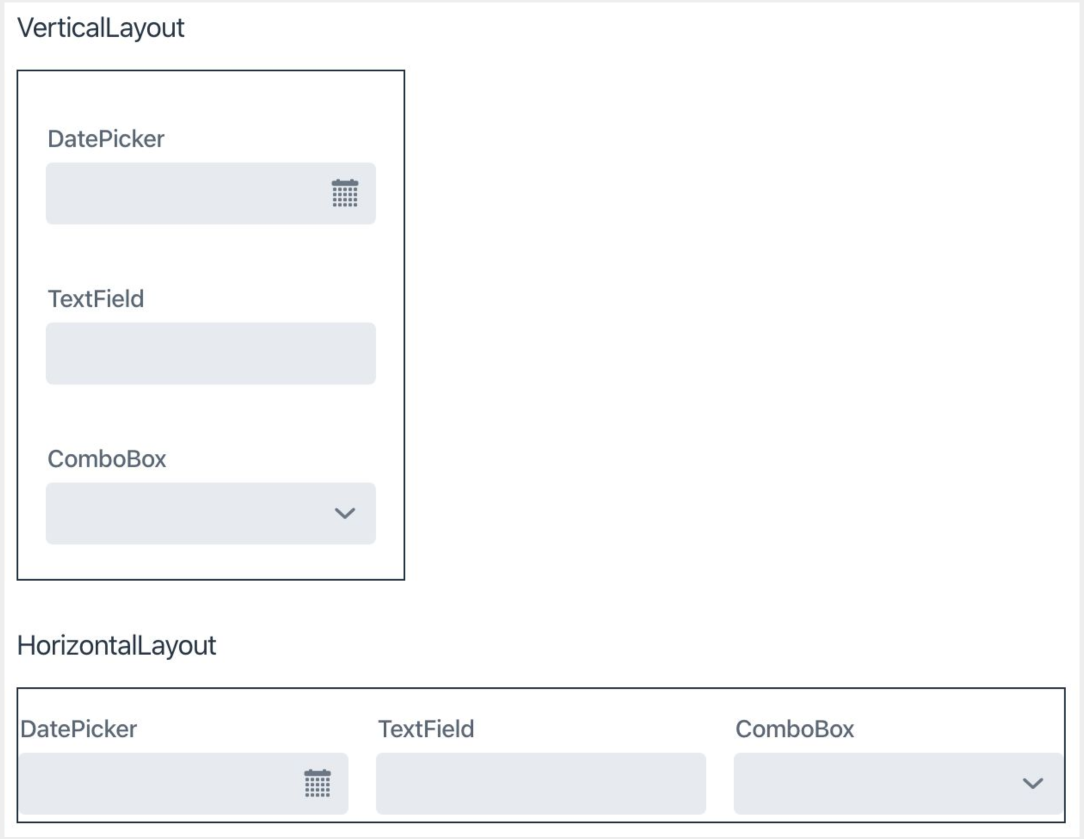
Use padding when you want it to be inside the borders and extend the background. Use margin when it should be outside the borders.

Default values

By default, VerticalLayout has both padding and spacing;

HorizontalLayout has spacing but not padding.

Neither Verticallayout nor HorizontalLayout has margin by default.



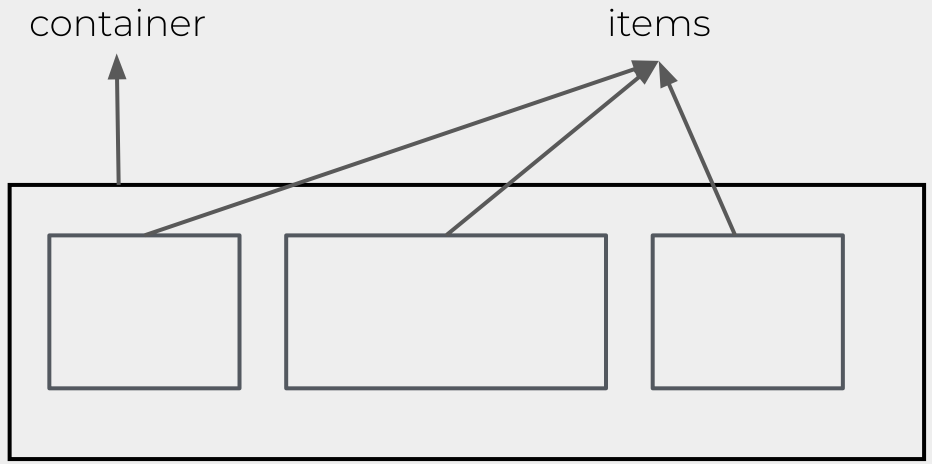
FlexLayout

FlexLayout is a component that implements **Flexbox**.

Flexbox

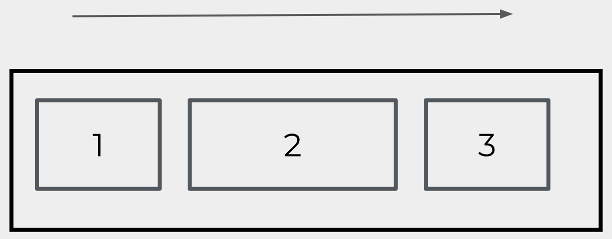
Flexbox is a CSS layouting feature.

It has two main concepts: container and items.

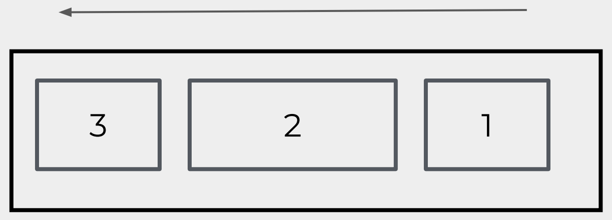


flex-direction

flex-direction is a CSS property that applies to the container.



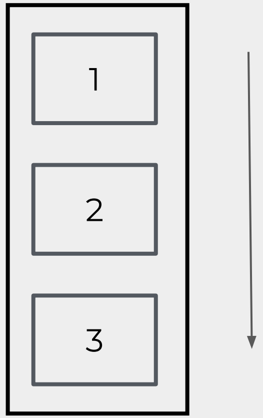
flex-direction:row



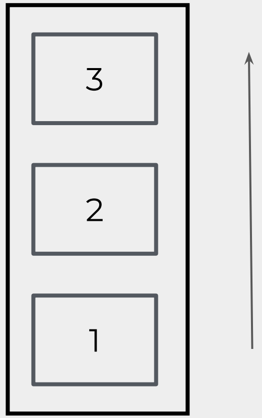
flex-direction:row-reverse

flex-direction

flex-direction is a CSS property that applies to the container.



flex-direction:column



flex-direction:column-reverse

Java API for flex-direction

There is a **FlexDirection** enum and a Java API for setting the flex-direction of a FlexLayout since **14.1**

```
flexLayout.setFlexDirection(FlexDirection.ROW);
flexLayout.setFlexDirection(FlexDirection.ROW_REVERSE);
flexLayout.setFlexDirection(FlexDirection.COLUMN);
flexLayout.setFlexDirection(FlexDirection.COLUMN_REVERSE);
```

Java API for flex-direction

For older versions, can use Element API

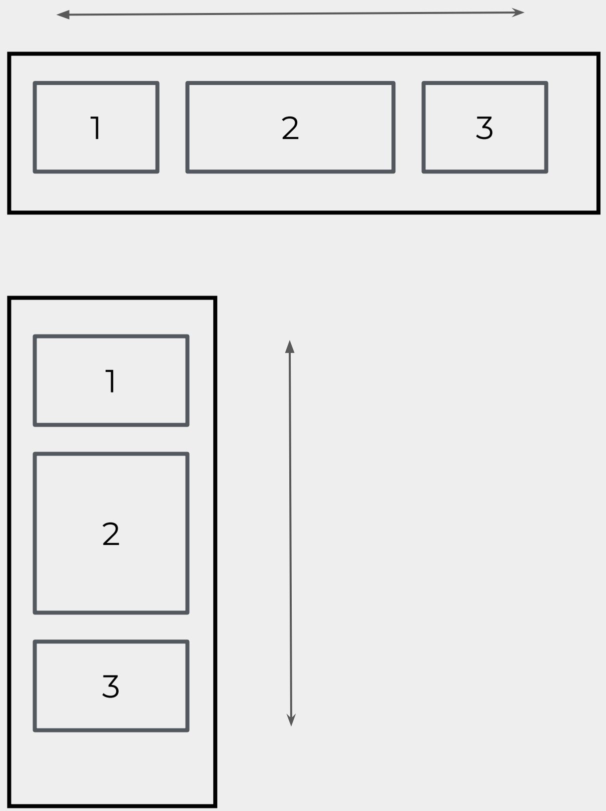
```
flexLayout.getStyle().set("flex-direction", "row");
flexLayout.getStyle().set("flex-direction", "row-reverse");
flexLayout.getStyle().set("flex-direction", "column");
flexLayout.getStyle().set("flex-direction", "column-reverse");
```

Alignment

justify-content determines how the items are positioned on the **primary** axis.

For a horizontal layout, it's the horizontal axis.

For a vertical layout, it's the vertical axis.

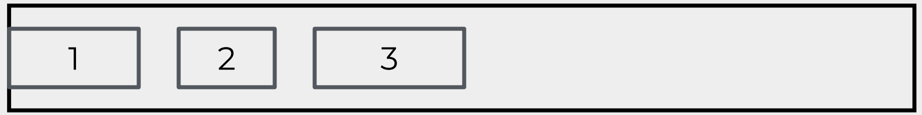


Alignment on the primary axis

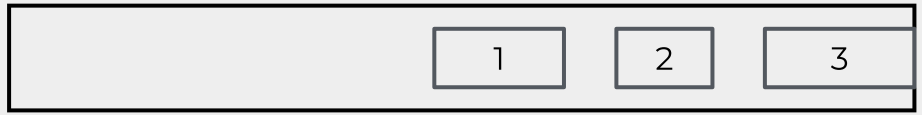
flex-start: Default value. Items are positioned at the beginning of the primary axis.

flex-end: Items are positioned at the end of the primary axis.

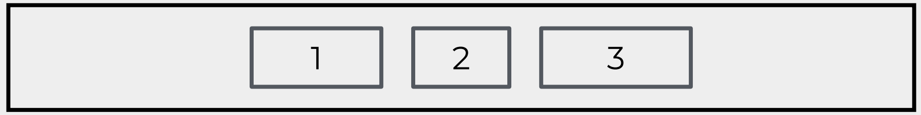
center: Items are positioned at the center of the primary axis.



justify-content: flex-start



justify-content: flex-end



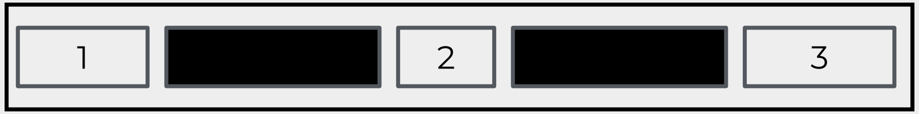
justify-content: center

Alignment on the primary axis

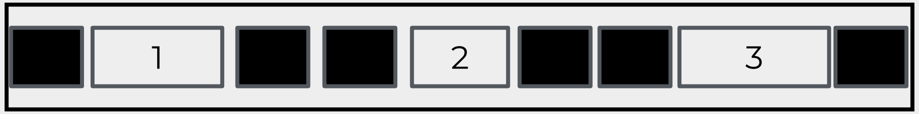
space-between: items are evenly distributed in the primary axis; the first item is on the start line, the last item on the end line.

space-around: items are evenly distributed in the primary axis with equal space around them.

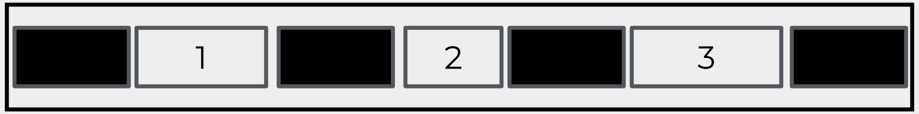
space-evenly: items are distributed so that the spacing between any two items (and the space to the edges) is equal.



justify-content: space-between



justify-content: space-around



justify-content: center

Alignment on the primary axis

There is a **JustifyContentMode** enum and a Java API for doing the alignment on the primary axis

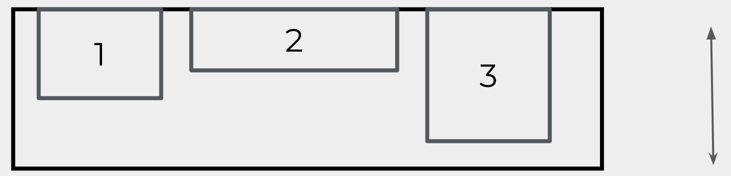
```
layout.setJustifyContentMode(FlexComponent.JustifyContentMode.AROUND);
layout.setJustifyContentMode(FlexComponent.JustifyContentMode.BETWEEN);
layout.setJustifyContentMode(FlexComponent.JustifyContentMode.CENTER);
layout.setJustifyContentMode(FlexComponent.JustifyContentMode.END);
layout.setJustifyContentMode(FlexComponent.JustifyContentMode.EVENLY);
layout.setJustifyContentMode(FlexComponent.JustifyContentMode.START);
```

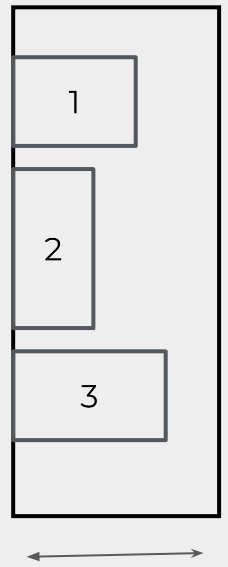
Alignment

align-items determines how the items are positioned on the **secondary** axis.

For a horizontal layout, it’s the vertical axis

For a vertical layout, it’s the horizontal axis.



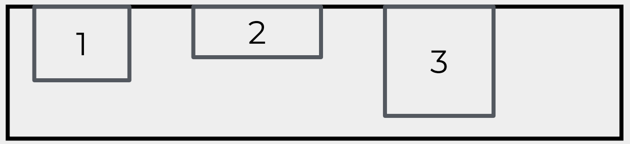


Alignment on the secondary axis

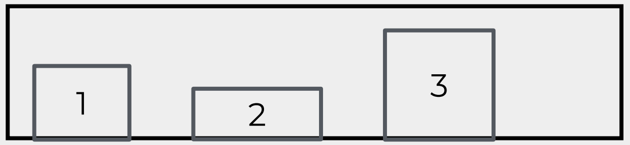
flex-start: Items are positioned at the start of the secondary axis.

flex-end: Items are positioned at the end of the secondary axis.

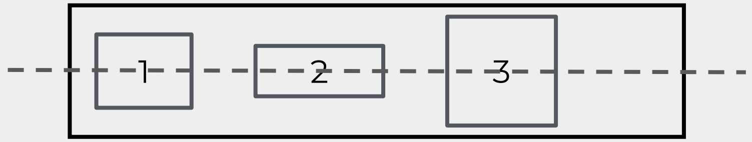
center: Items are positioned at the center of the secondary axis.



align-items: flex-start



align-items: flex-end

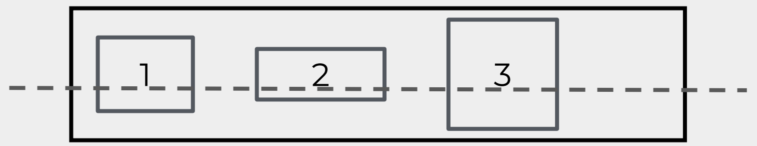


align-items: flex-center

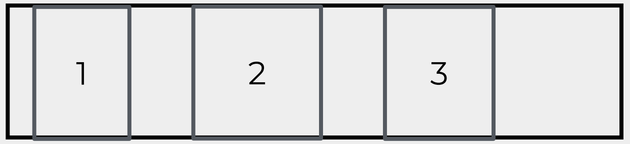
Alignment on the secondary axis

flex-baseline: Items are aligned on the baseline.

flex-stretch: Items are stretched across the whole secondary axis.



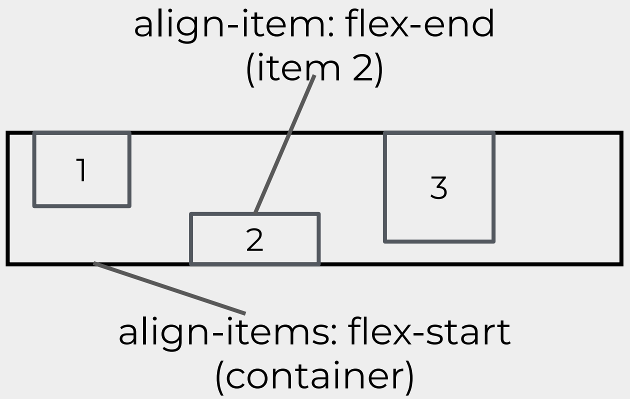
align-items: baseline



align-items: stretch

Alignment on the secondary axis

Also possible to align an individual item differently with **align-item**



Alignment on the secondary axis

There is a **Alignment** enum and a Java API for doing the alignment on the secondary axis

```
//For the container, to align all the items
layout.setAlignItems(FlexComponent.Alignment.BASELINE);
layout.setAlignItems(FlexComponent.Alignment.CENTER);
layout.setAlignItems(FlexComponent.Alignment.END);
layout.setAlignItems(FlexComponent.Alignment.START);
layout.setAlignItems(FlexComponent.Alignment.STRETCH);
```

Alignment on the secondary axis

There are also helper methods for HorizontalLayout and VerticalLayout to do the alignment on the secondary axis.

```
//To align all the items on the vertical direction for a horizontal layout
horizontalLayout.setDefaultVerticalComponentsAlignment(FlexComponent.Alignment.BASELINE);

//To align all the items on the horizontal direction for a vertical layout
verticalLayout.setDefaultHorizontalComponentsAlignment(FlexComponent.Alignment.BASELINE);
```

Alignment on the secondary axis

There is a **Alignment** enum and a Java API for doing the alignment on the secondary axis

```
//For individual item(s)
layout.setAlignSelf(FlexComponent.Alignment.BASELINE, item);
layout.setAlignSelf(FlexComponent.Alignment.CENTER, item);
layout.setAlignSelf(FlexComponent.Alignment.END, item);
layout.setAlignSelf(FlexComponent.Alignment.START, item);
layout.setAlignSelf(FlexComponent.Alignment.STRETCH, item);
```

Alignment on the secondary axis

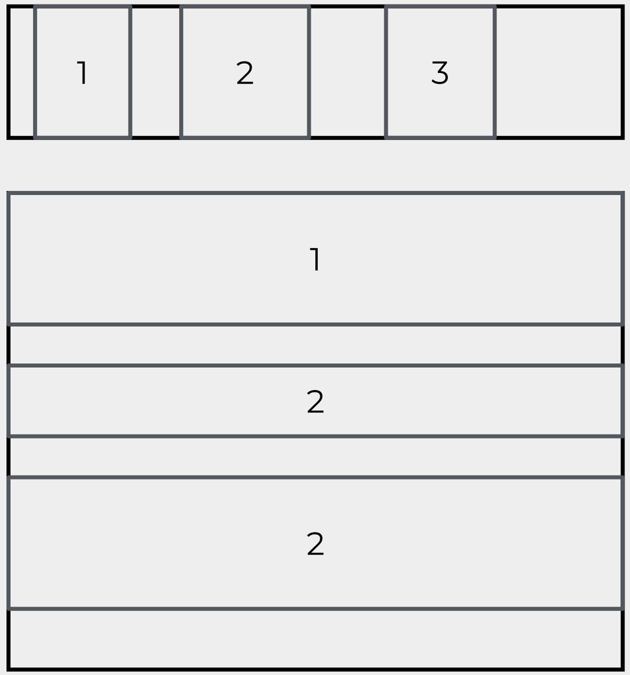
There are also helper methods for HorizontalLayout and VerticalLayout to do the alignment for individual items on the secondary axis.

```
//To align an individual item on the vertical direction for a horizontal layout
horizontalLayout.setVerticalComponentsAlignment(FlexComponent.Alignment.END, component);

//To align an individual item on the horizontal direction for a vertical layout
verticalLayout.setHorizontalComponentsAlignment(FlexComponent.Alignment.END, component);
```

Use case - full width/height

To make all the child items have full height in a horizontal layout or full width in a vertical layout, use

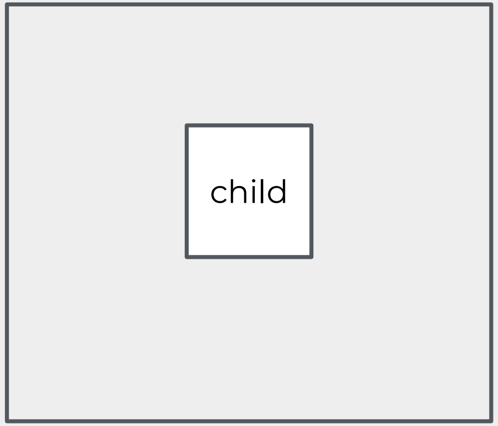
```
layout.setAlignItems(
       FlexComponent.Alignment.STRETCH);
```



Use case - centering

To center a child item, you can combine the Alignment and JustifyConentMode

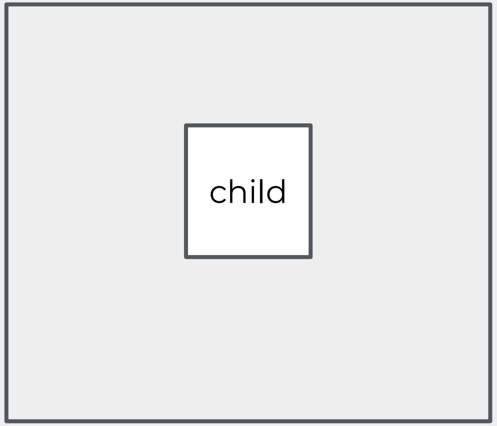
```
layout.setAlignItems(
     FlexComponent.Alignment.CENTER);
layout.setJustifyContentMode(
     FlexComponent.JustifyContentMode.CENTER);
```



Use case - centering

Could also use a CSS trick

```
child.getElement().getStyle()
        .set("margin", "auto");
```



Sizing

There are convenient APIs for setting the size of a component

```
//set width/height of a component, e.g. setWidth/Height(“200px”), setWidth/Height(“100%”)
component.setWidth()
component.setHeight()

//A shorthand for setWidth/Height(“100%”)
component.setWidthFull()
component.setHeightFull()
```

Sizing

There are convenient APIs for setting the size of a component

```
//Set the min/max height/width of a component, could be useful for responsive layouting
component.setMinWidth()
component.setMinHeight()
component.setMaxWidth()
component.setMaxHeight()
```

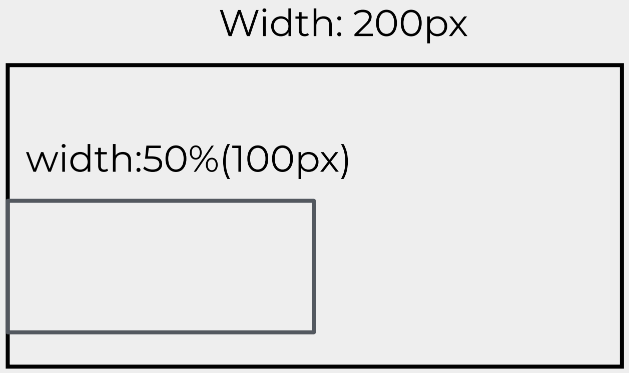
Sizing

There are convenient APIs for setting the size of a component

```
//A shorthand for setWidth(“100%”) and setHeight(“100%”)
component.setSizeFull()

//Remove the height and width of the component
component.setSizeUndefined()
```

Relative size

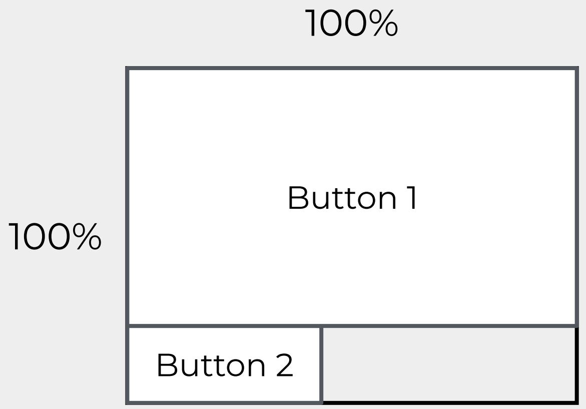
Relative size is relative to the parent component. E.g., 50% means 50% of the parent component’s height/width.

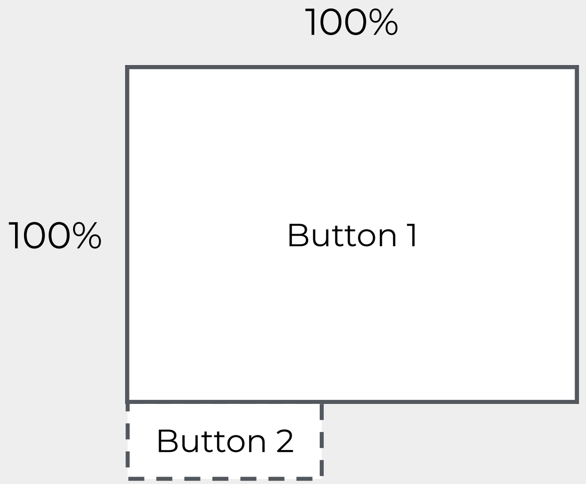


Sizing

Given the code below, which layouting on the right side is correct?

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

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

button1.setSizeFull();
```

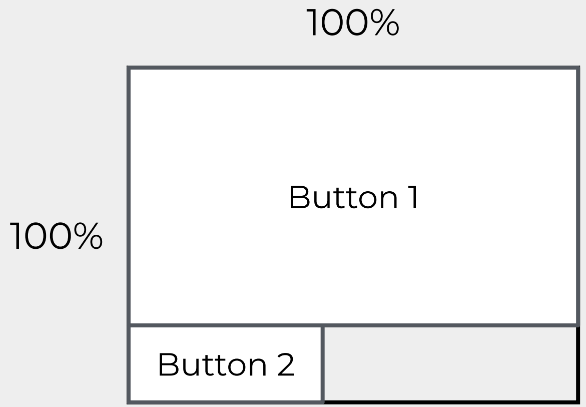




Sizing

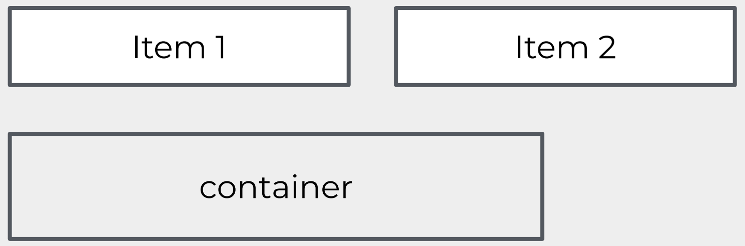
Size is also affected by **flex-shrink** and **flex-grow**.

Note that it affects not only relative size but also absolute size, e.g. 100px.



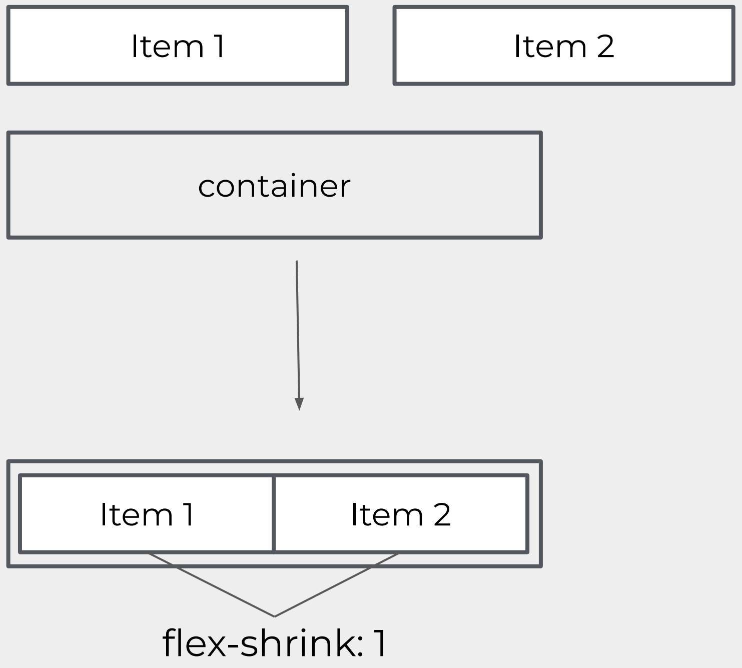
flex-shrink

It defines how items should shrink when there isn’t enough space in the container.



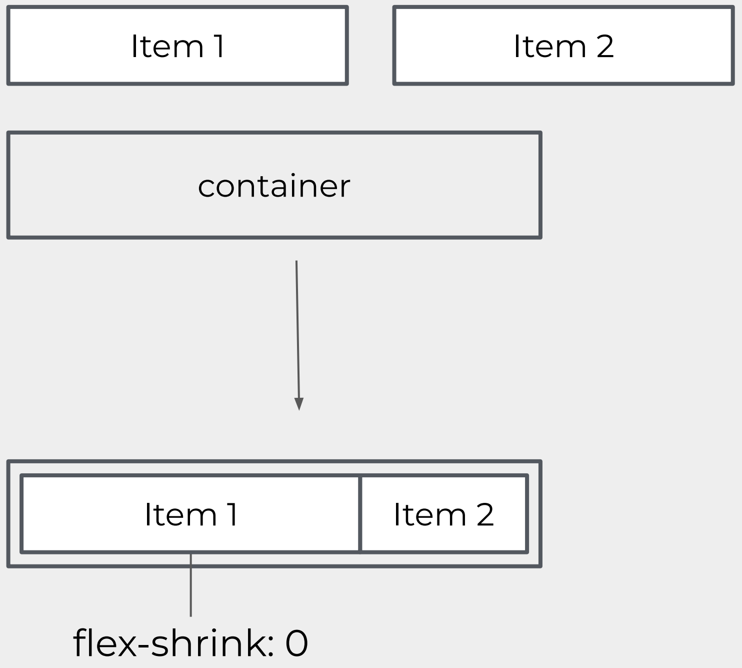
flex-shrink

The default value is 1, which means all items will shrink equally to fit into the container.



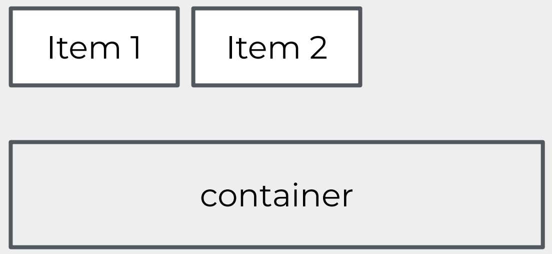
flex-shrink

Setting flex-shrink to 0 will prevent an item from shrinking.



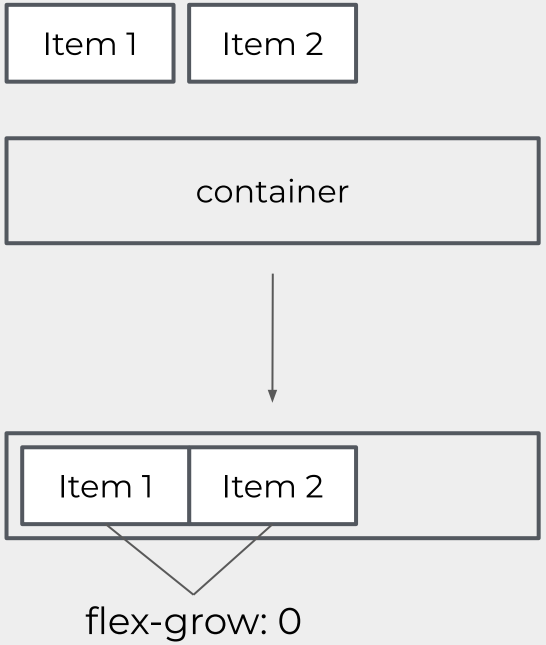
flex-grow

Defines how to distribute **free space**, when the container is bigger than the size of the items.



flex-grow

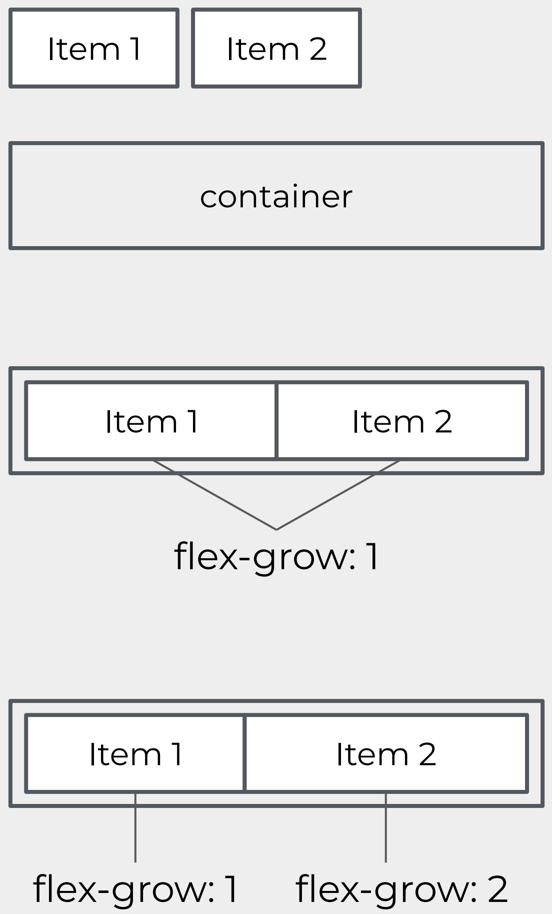
The default value is 0, which means an item won’t take up any free space.



flex-grow

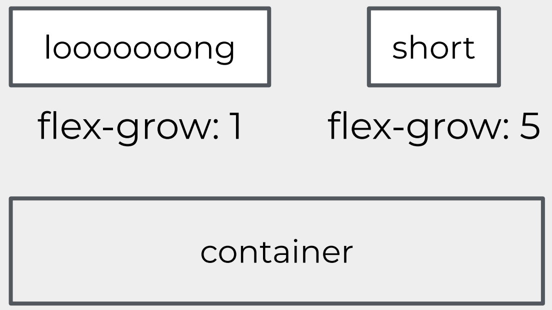
Define flex-grow: 1 for each item will make them grow equally.

Define flex-grow with a different value for each item will make them grow proportionally



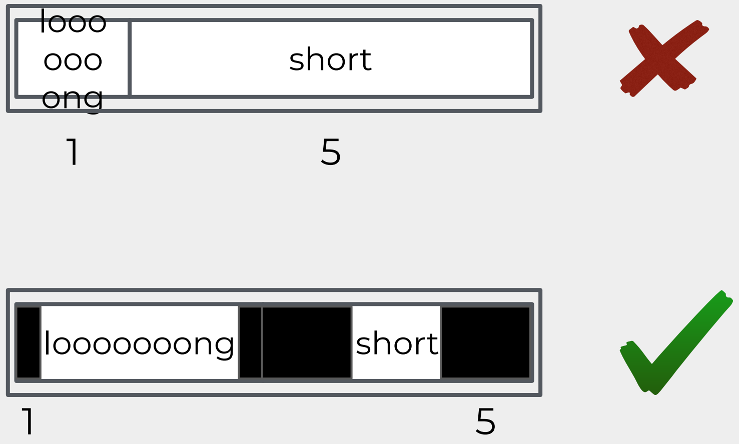
Quiz

How will the items fit into the container?



Quiz

Free space is distributed according to 1:5 ratio, not the width of the items.



Java API for flex-grow

You can set the flex-grow of an item from the parent layout via the setFlexGrow() Java API

```
layout.setFlexGrow(3, item);
```

Java API for flex-grow

There is also a shorthand method expand() for setting the flex-grow to 1.

```
layout.expand(item);
=
layout.setFlexGrow(1, item);
```

Java API for flex-shrink

There is Java API for setting the flex-shrink for child components **on the layout** since version **14.1**.

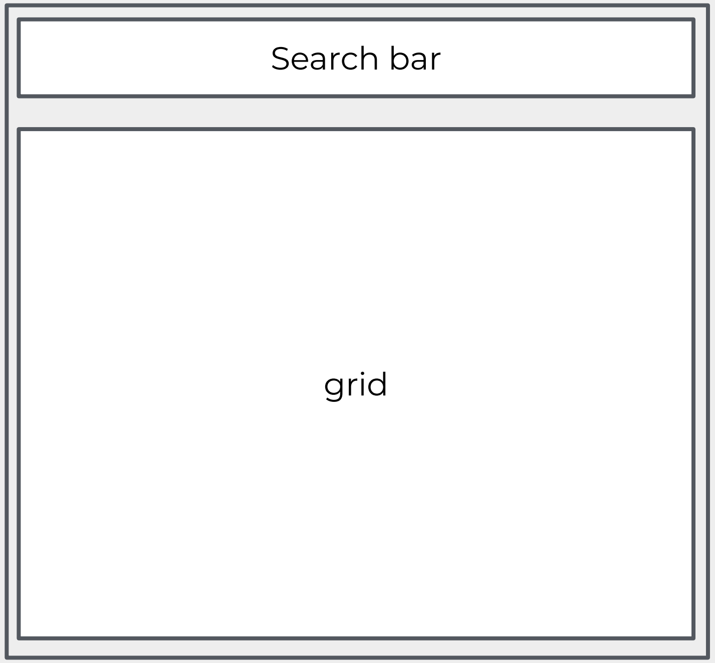
```
layout.setFlexShrink(0, component);
```

For older versions, can use Element API to set the flex-shrink **on the child component**.

```
item.getElement().getStyle().set("flex-shrink", "0");
```

Use case #1

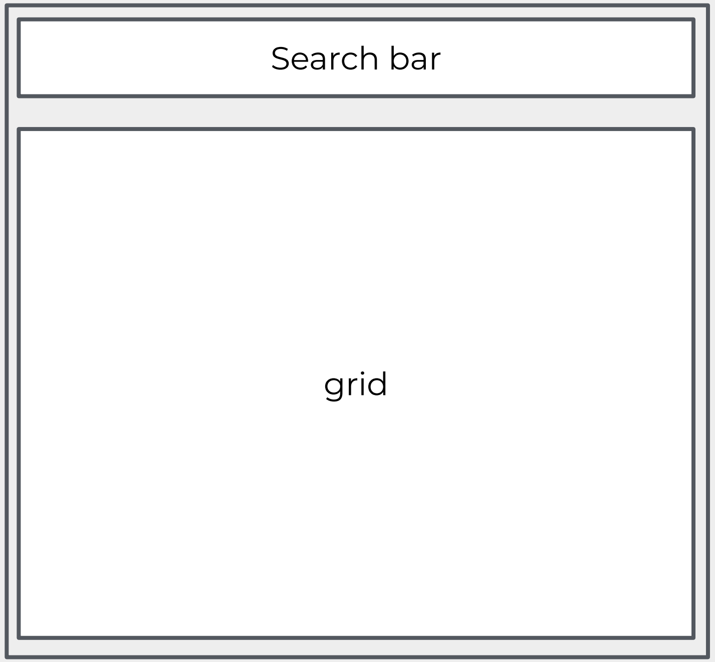
How to expand a component?



Use case #1

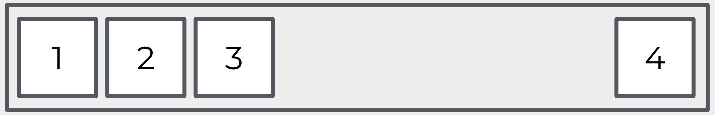
How to expand a component?

```
layout.setAlignItems(Alignment.STRETCH);
layout.setHeightFull();
layout.expand(grid);
```



Use case #2

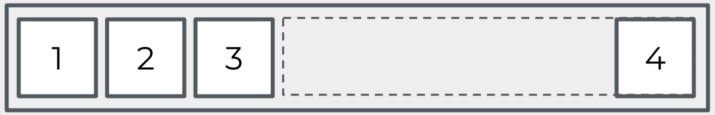
How to implement layout like this?



Use case #2

One way is to wrap item 4 into a FlexLayout.

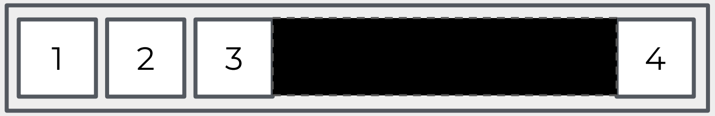
```
FlexLayout wrapper = new FlexLayout(item4);
layout.expand(wrapper);
wrapper.setJustifyContentMode(
      FlexComponent.JustifyContentMode.END);
```



Use case #2

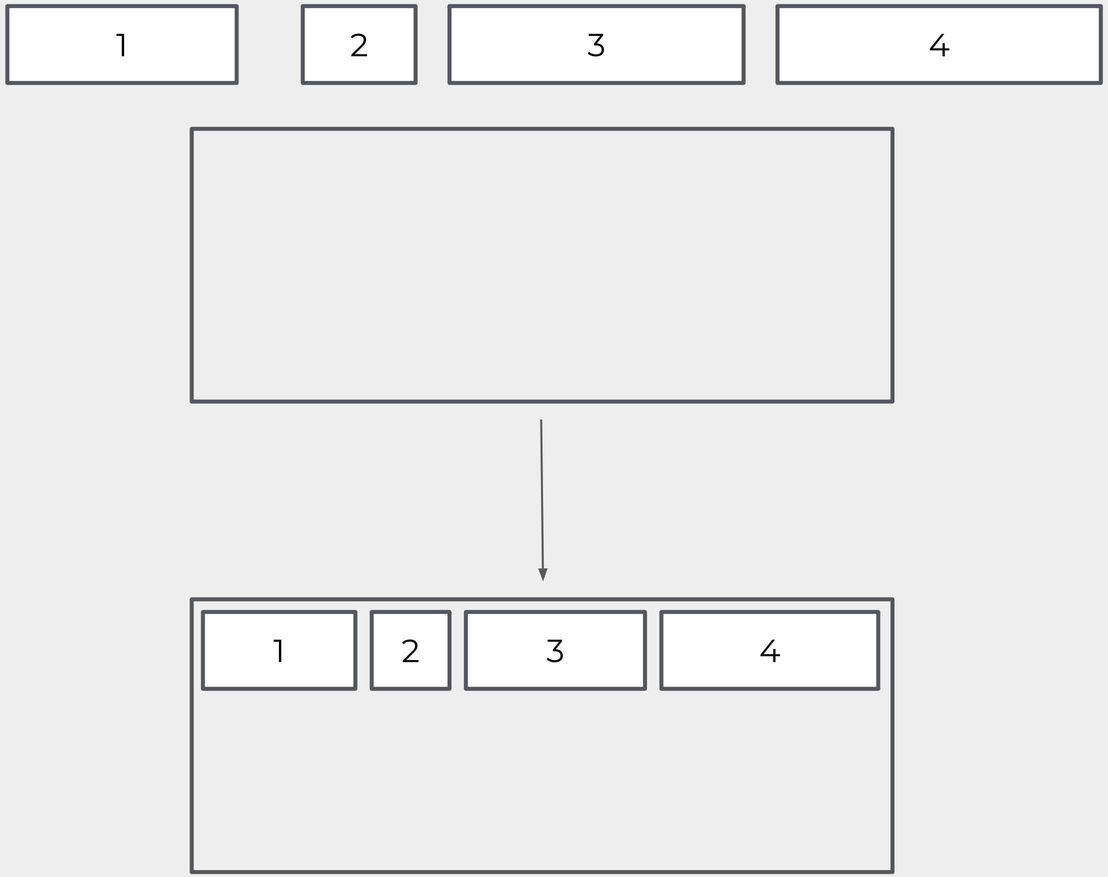
Could also just use some CSS trick.

```
child4.getStyle().set(
      "margin-left", "auto");
```



Wrap items

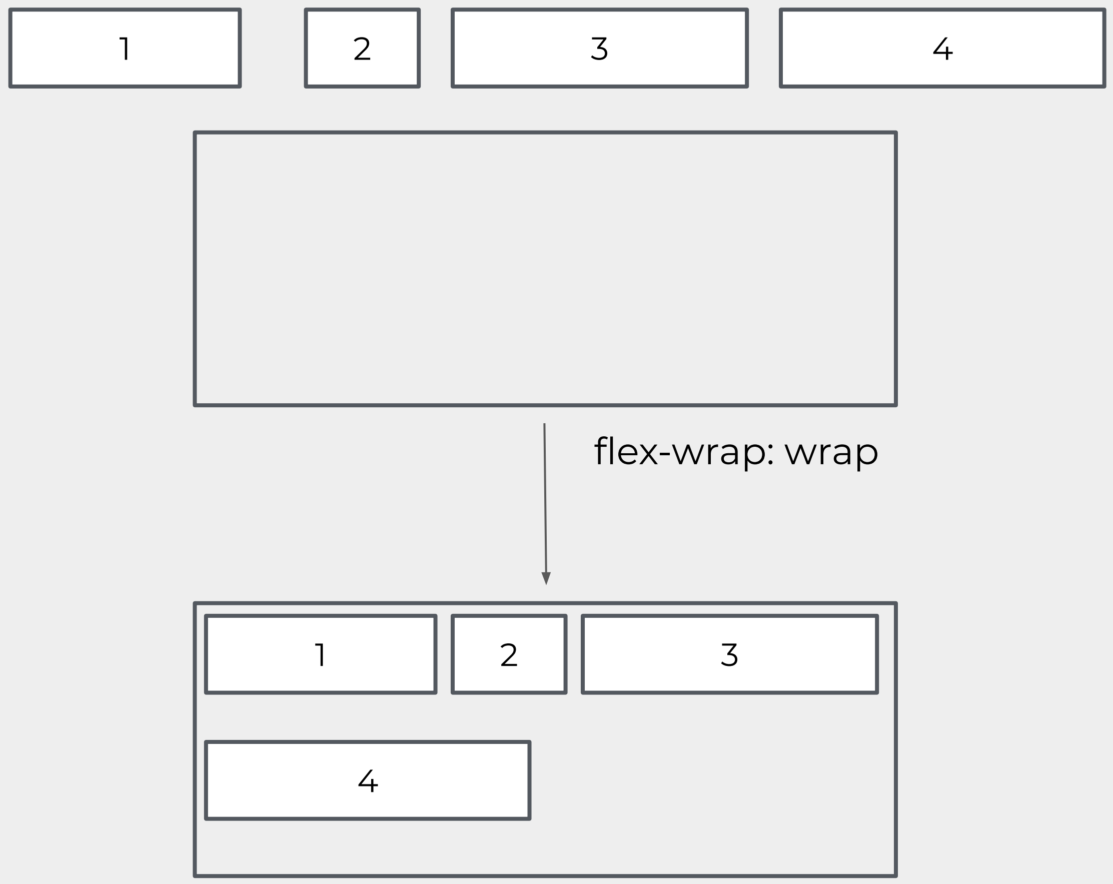
By default, items will shrink themselves to fit into one line, even though there is enough space to start a new line.



Wrap items

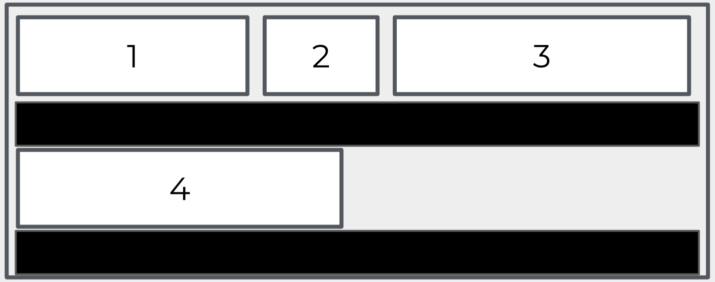
You can configure the layout by setting **flex-wrap** to **wrap** to make the items wrap into new lines.

```
layout.setWrapMode(FlexLayout.WrapMode.WRAP);
```



Free space between lines

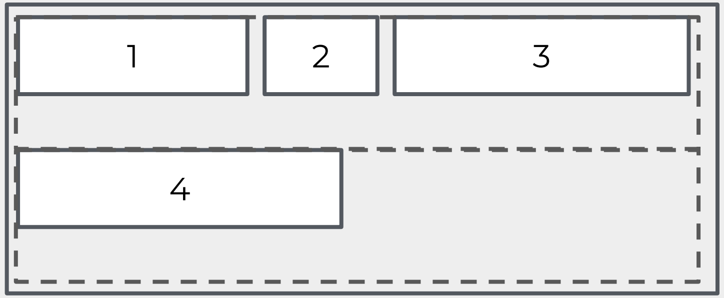
When there are multiple lines in the container, you can decide how to distribute the free space between the lines with the **align-content** property.



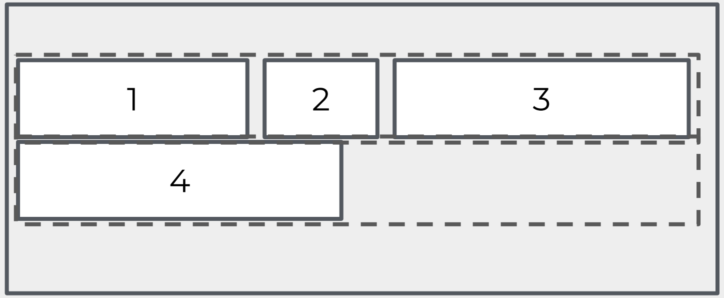
align-content

stretch: The **default** value. Each row (dashed lines in the picture) will stretch equally to take free space. So there will be space inside the row if the items are smaller than the row.

center: rows (dashed lines in the picture) are packed to the center of the container.



align-content: stretch

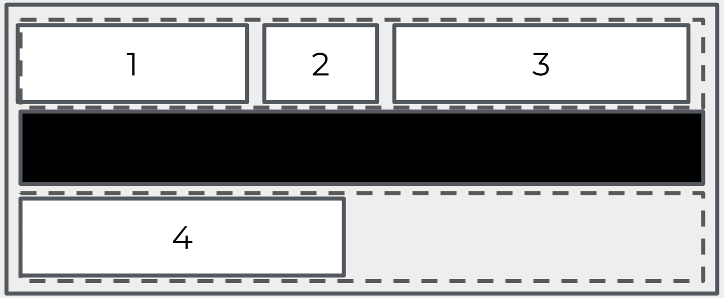


align-content: center

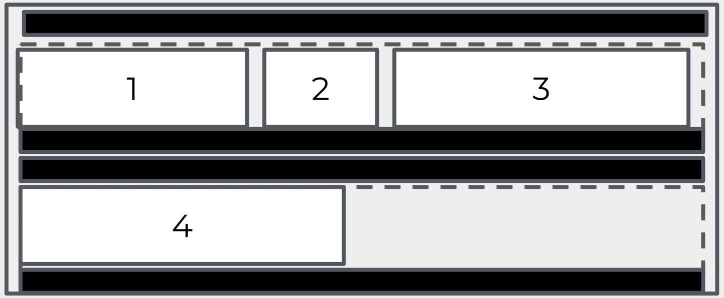
align-content

space-between: rows are evenly distributed; the first row is at the start of the container while the last row is at the end.

space-around: rows are evenly distributed with equal space between them.



align-content: space-between

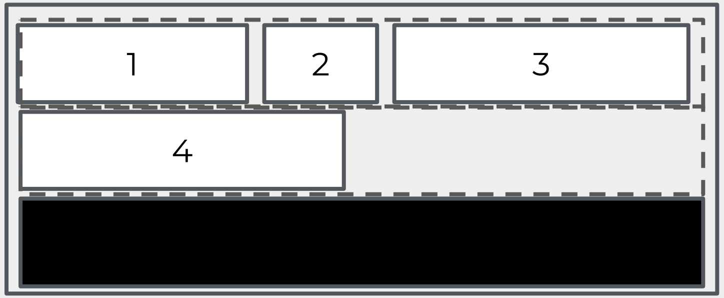


align-content: space-around

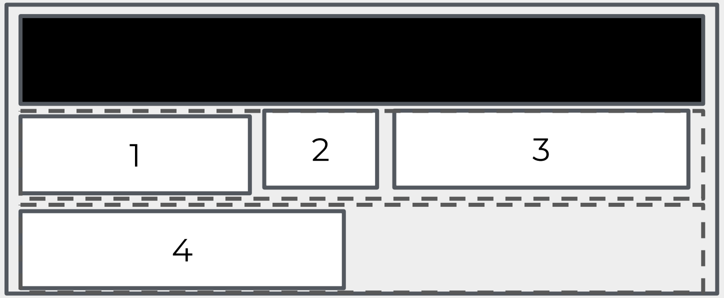
align-content

flex-start: rows are packed to the start of he container.

flex-end: rows are packed to the end of he container.



align-content: flex-start



align-content: flex-end

Java API

There is a Java API for FlexLayout to set the align-content since **14.1**

```
flexLayout.setAlignContent(FlexLayout.ContentAlignment.START);
flexLayout.setAlignContent(FlexLayout.ContentAlignment.END);
flexLayout.setAlignContent(FlexLayout.ContentAlignment.CENTER);
flexLayout.setAlignContent(FlexLayout.ContentAlignment.SPACE_BETWEEN);
flexLayout.setAlignContent(FlexLayout.ContentAlignment.SPACE_AROUND);
flexLayout.setAlignContent(FlexLayout.ContentAlignment.STRECTH);
```

Java API

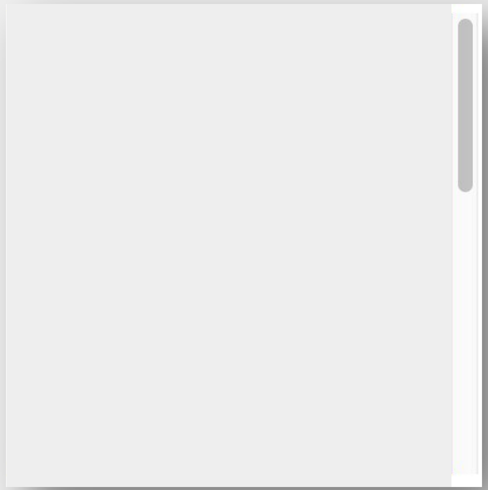
For older versions, can use Element API.

```
layout.getElement().getStyle().set("align-content", "flex-start");
layout.getElement().getStyle().set("align-content", "flex-end");
layout.getElement().getStyle().set("align-content", "center");
layout.getElement().getStyle().set("align-content", "space-between");
layout.getElement().getStyle().set("align-content", "space-around");
layout.getElement().getStyle().set("align-content", "stretch");
```

Scrolling

There is no Java API to enable scrolling yet, but can be done with Element API and CSS

```
//enable vertical scroll bar
layout.getStyle().set("overflow-y", "auto");
//enable horizontal scroll bar
layout.getStyle().set("overflow-x", "auto");
//enable both horizontal and vertical scroll bars
layout.getStyle().set("overflow", "auto");
```



Exercise 1

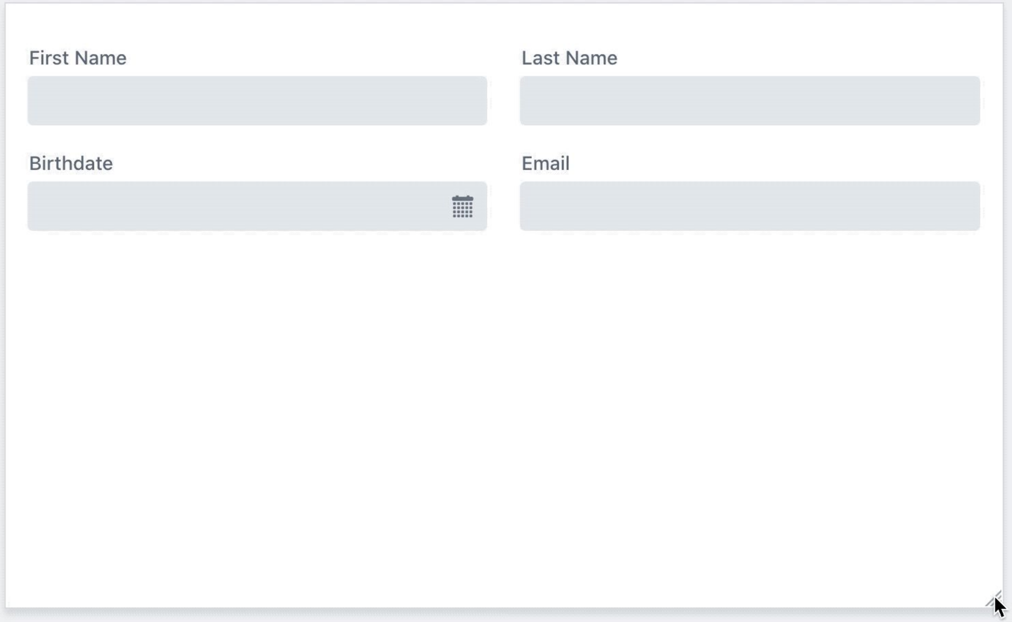
Compose an application layout

FormLayout

A responsive layout designed for displaying forms

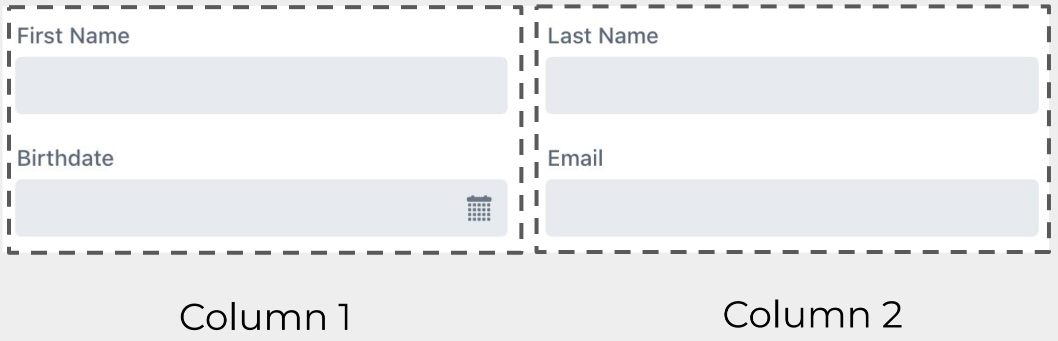
Responsive

One significant benefit of using FormLayout is that it's responsive out of the box.



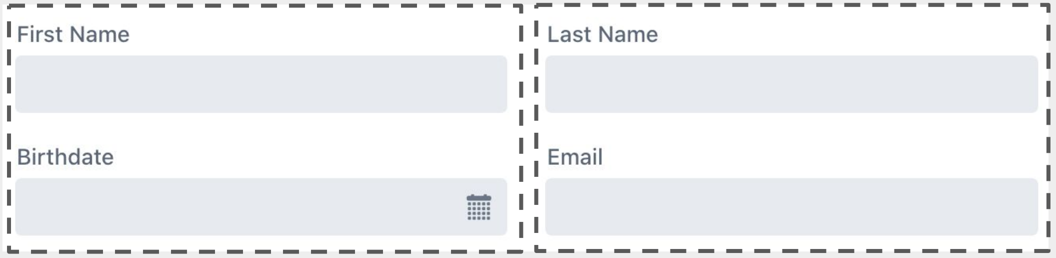
Responsive

The responsiveness works by showing a different number of columns depending on the width of the Formlayout.



Responsive

By default, it shows 2 columns when the width is more than 40em, only 1 column otherwise.

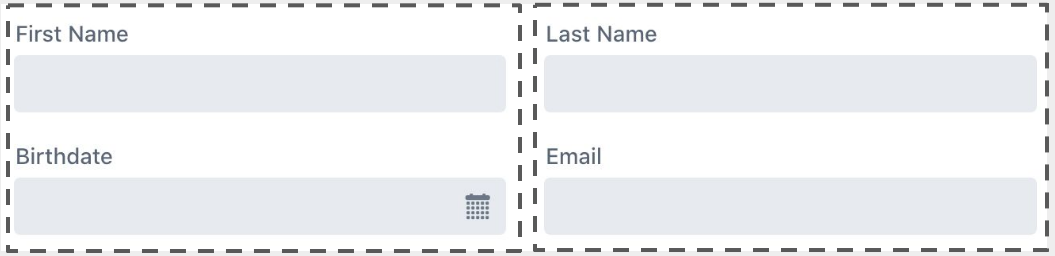


Column 1

Column 2

Responsive Step

The number of columns can also be customised via setting the responsive steps.



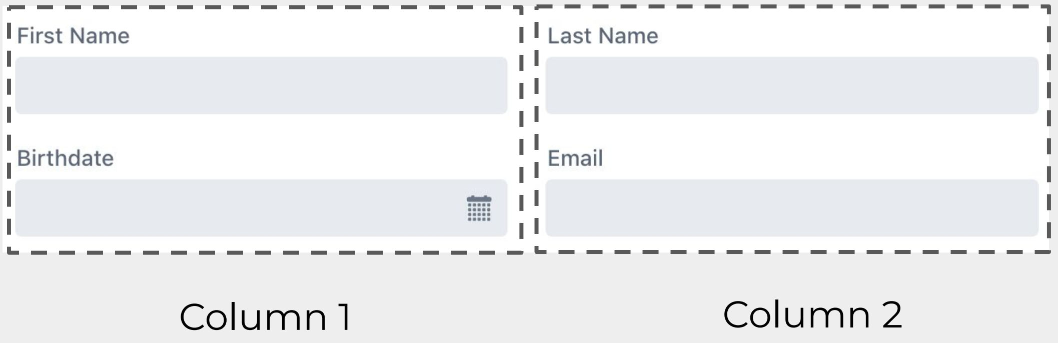
Column 1

Column 2

Responsive Step

A responsive step works as when the width is more than the [first parameter], then there should be [second parameter] columns.

```
/**
* Show 2 columns when the width is >= 50em.
* Show 1 column when the width is [0-50) em
*/
formLayout.setResponsiveSteps(
       new ResponsiveStep("0", 1),
       new ResponsiveStep("50em", 2));
```

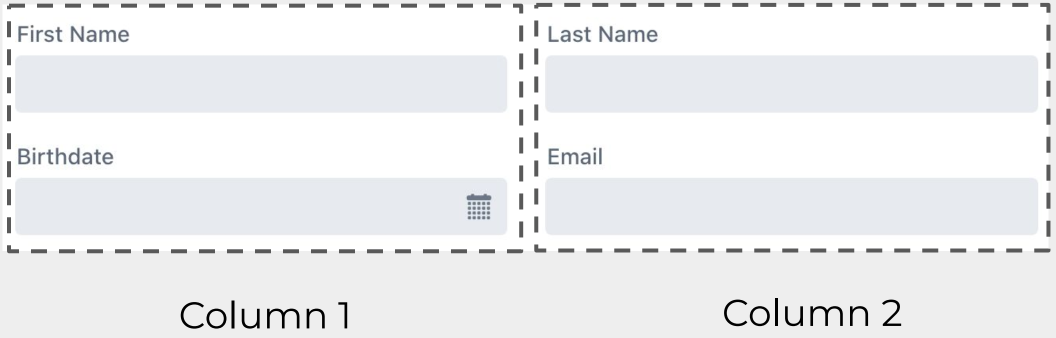


Add components

The first way of adding components to a FormLayout is to use the **add()** method, as in any other type of layouts.

In this way, a **label** is set for the added component and displays on **top** of the component.

```
FormLayout formLayout = new FormLayout();

TextField firstName = new TextField("First Name");
formLayout.add(firstName);
```



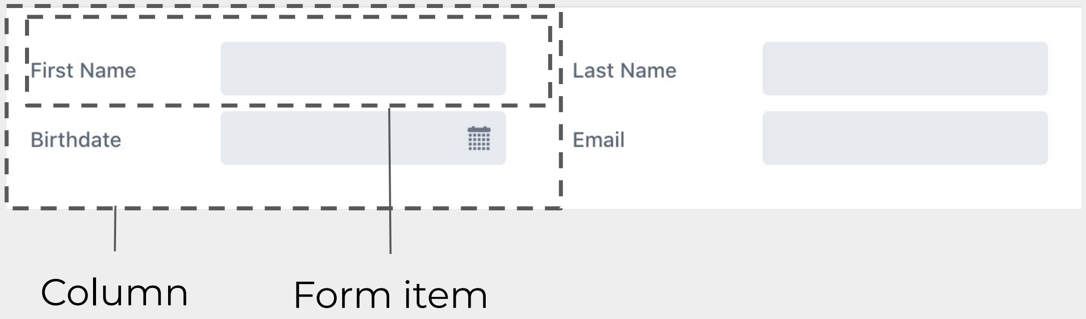
Add components

The second way of adding components to a FormLayout is to use the **addFormItem()** method

In this way, the added component is wrapped into a **form item**, and the label should be set for the form item.

A **label** displays to the **left** side of a component.

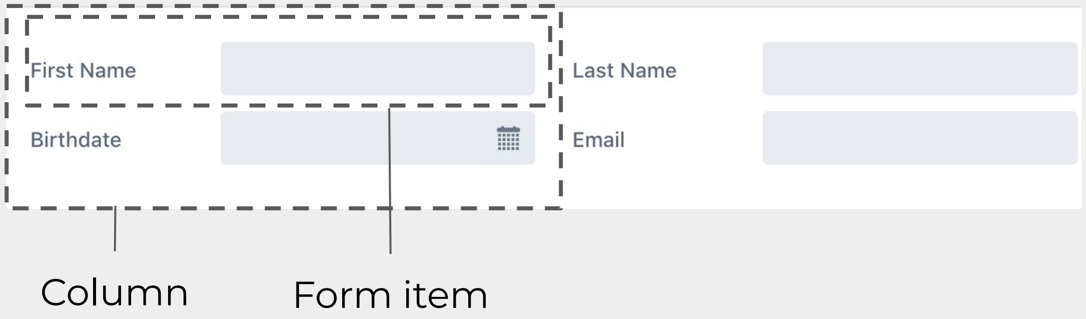
```
FormLayout formLayout = new FormLayout();

TextField firstName = new TextField();
formLayout.addFormItem(firstName, "First Name");
...
```



Add components

When using addFormItem(), it’s normally good to set full width for the component, so the component will take the full width of the available space in the form item

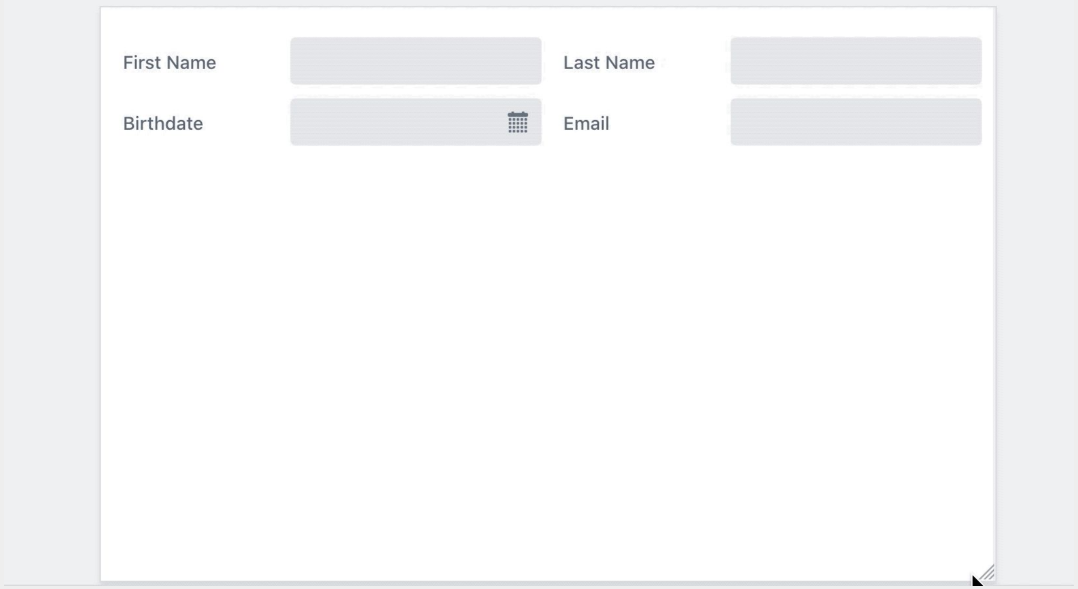
```
FormLayout formLayout = new FormLayout();

TextField firstName = new TextField();
formLayout.addFormItem(firstName, "First Name");
firstName.setWidthFull();
...
```



Responsive label position

The most significant benefit of using addFormItem() is that the position of the labels can be changed responsively.

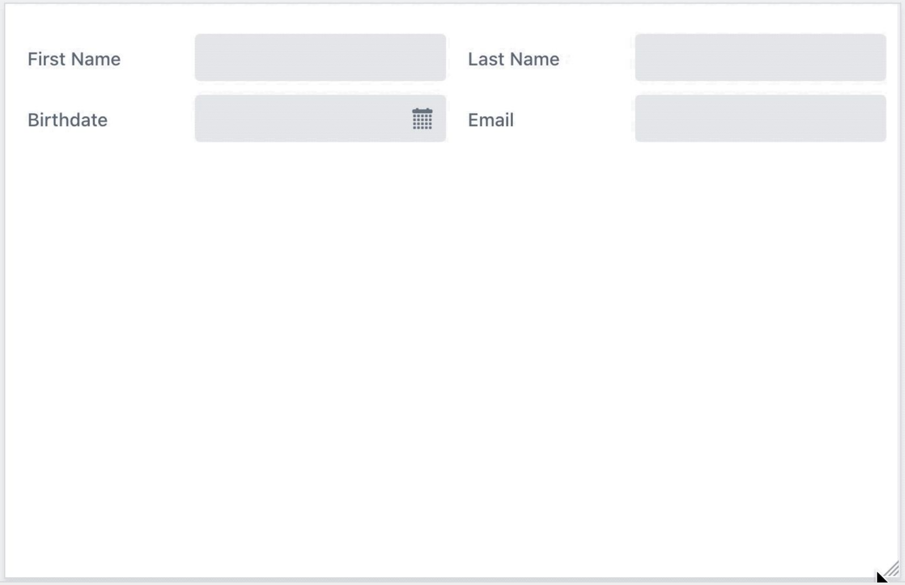
The default is that when the width is less than 20em, the labels will come to the top of the components.



Responsive label position

The position of the labels can also be customised via setting the responsive steps.

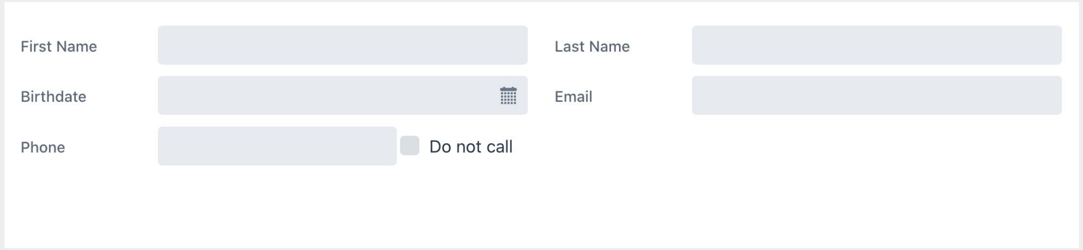
```java
/**
* Show 2 columns when the width is >= 50em.
* Show 1 column with label on the left side
* when the width is [20-50) em.
* Show 1 column with label on the top
* when the width is [0-20) em.
*/
fl.setResponsiveSteps(
       new ResponsiveStep("0", 1,
             LabelsPosition.TOP),
       new ResponsiveStep("20em", 1),
       new ResponsiveStep("50em", 2));
```



Add complex components

Components can be grouped into a layout and then add the layout to the FormLayout.

```
FormLayout formLayout = new FormLayout();

FlexLayout phoneLayout = new FlexLayout();
phoneLayout.setWidthFull();
TextField phone = new TextField();
Checkbox noCall = new Checkbox("Do not call");
phoneLayout.add(phone, noCall);
phoneLayout.expand(phone);
phoneLayout.setAlignItems(Alignment.CENTER);

formLayout.addFormItem(phoneLayout, "Phone");
...
```



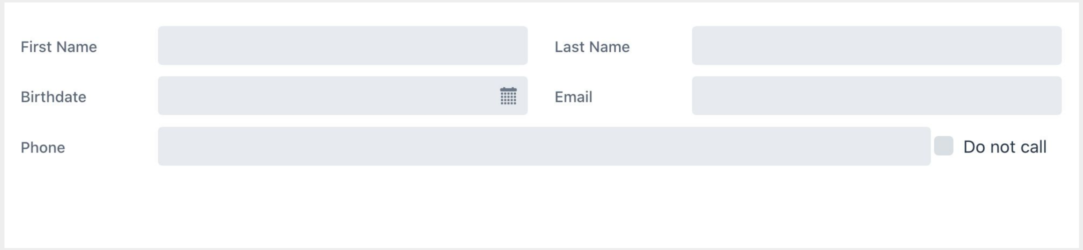
Column span

Could also let a component span multiple columns by using **formLayout.setColspan()**

When using add(), set column span on the component.

When using addFormItem(), set column span on the form item.

```
formLayout.setColspan(formItem, 2);
formLayout.setColspan(component, 2);
```

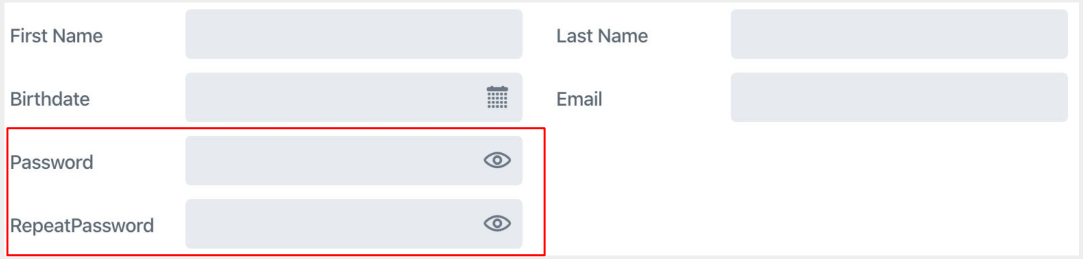


Force a new row

Sometimes, instead of showing a component on the second column, you might want to force it to a new row.

You can achieve this by adding a <br>.

```
PasswordField password = new PasswordField();
formLayout.addFormItem(password, "Password");

formLayout.getElement().appendChild(
      ElementFactory.createBr());

PasswordField repeatPwd = new PasswordField();
formLayout.addFormItem(
      repeatPwd, "RepeatPassword");
```



Exercise 2

Build a form with FormLayout

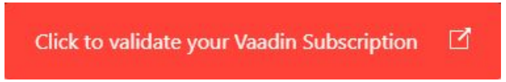
Vaadin Board

Automatic responsive layout

Vaadin Board

Vaadin Board is a **commercial** component.

Start a **free trial** by clicking on the prompt in the browser when seeing one



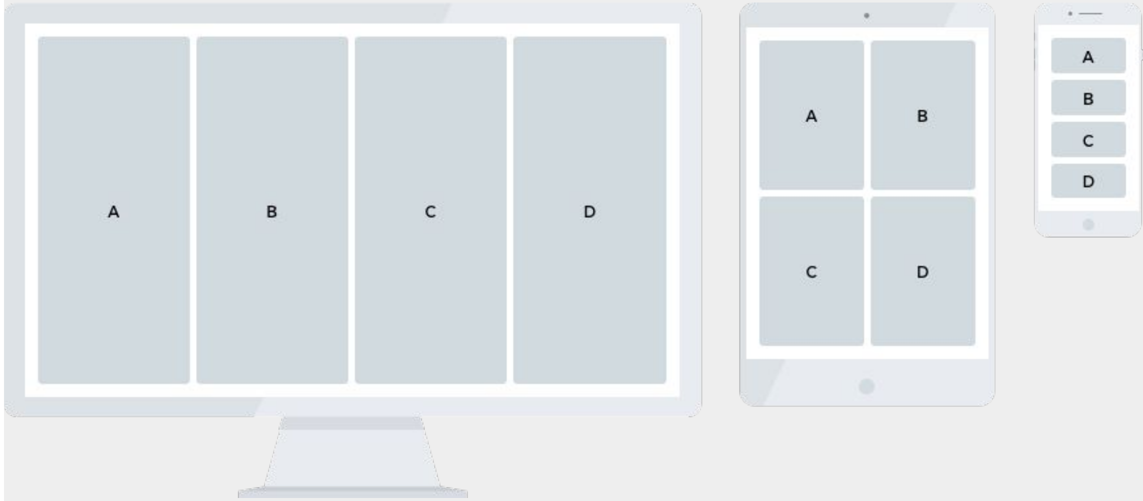
Click to validate your Vaadin Subscription

Vaadin Board

Vaadin Board is based on rows and columns.

You add components to the rows, Vaadin will make it responsive for you.

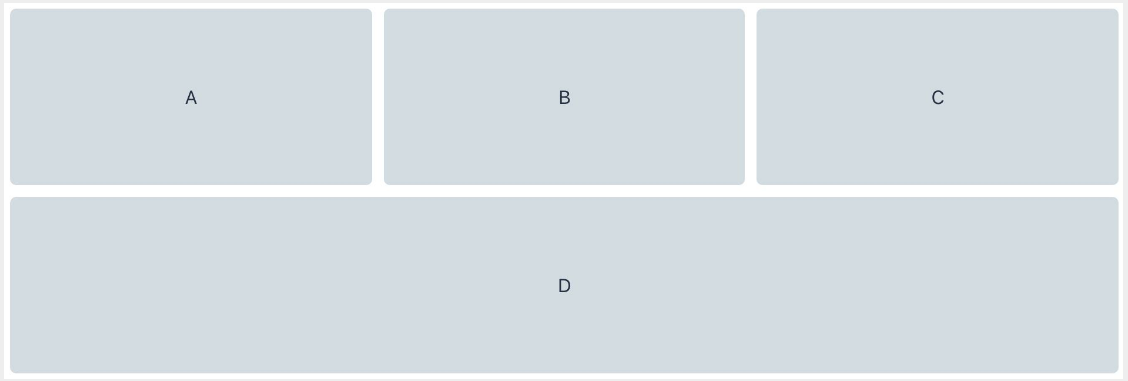
```
Board board = new Board();
Component a = createComponent("A");
Component b = createComponent("B");
Component c = createComponent("C");
Component d = createComponent("D");

board.addRow(a, b, c, d);
```



Vaadin Board

Could also add multiple rows

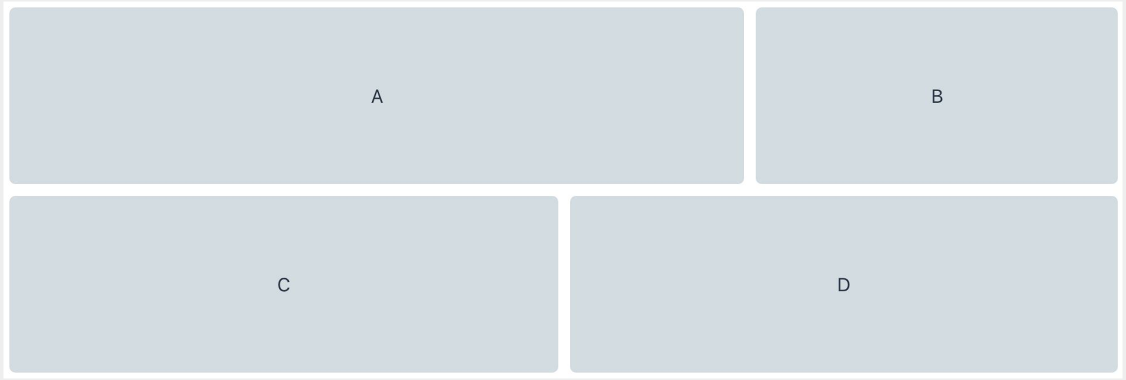
```
Board board = new Board();
Component a = createComponent("A");
Component b = createComponent("B");
Component c = createComponent("C");
Component d = createComponent("D");

board.addRow(a, b, c);
board.addRow(d);
```



Vaadin Board

Could also set column span

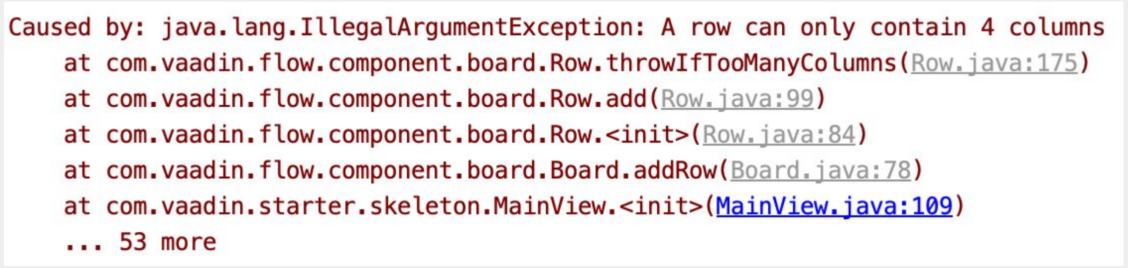
```
Board board = new Board();
Component a = createComponent("A");
Component b = createComponent("B");
Component c = createComponent("C");
Component d = createComponent("D");

Row row1 = board.addRow(a, b);
row1.setComponentSpan(a, 2);

board.addRow(c, d);
```



Vaadin Board

Limitation: A row can only have up to **4** columns.

```
Board board = new Board();
Component a = createComponent("A");
Component b = createComponent("B");
Component c = createComponent("C");
Component d = createComponent("D");
Component e = createComponent("E");
board.addRow(a, b, c, d, e);
```



```
Caused by: java.lang.IllegalArgumentException: A row can only contain 4 columns
    at com.vaadin.flow.component.board.Row.throwIfTooManyColumns(Row.java:175)
    at com.vaadin.flow.component.board.Row.add(Row.java:99)
    at com.vaadin.flow.component.board.Row.<init>(Row.java:84)
    at com.vaadin.flow.component.board.Board.addRow(Board.java:78)
    at com.vaadin.starter.skeleton.MainView.<init>(MainView.java:109)
    ... 53 more
```

Exercise 3

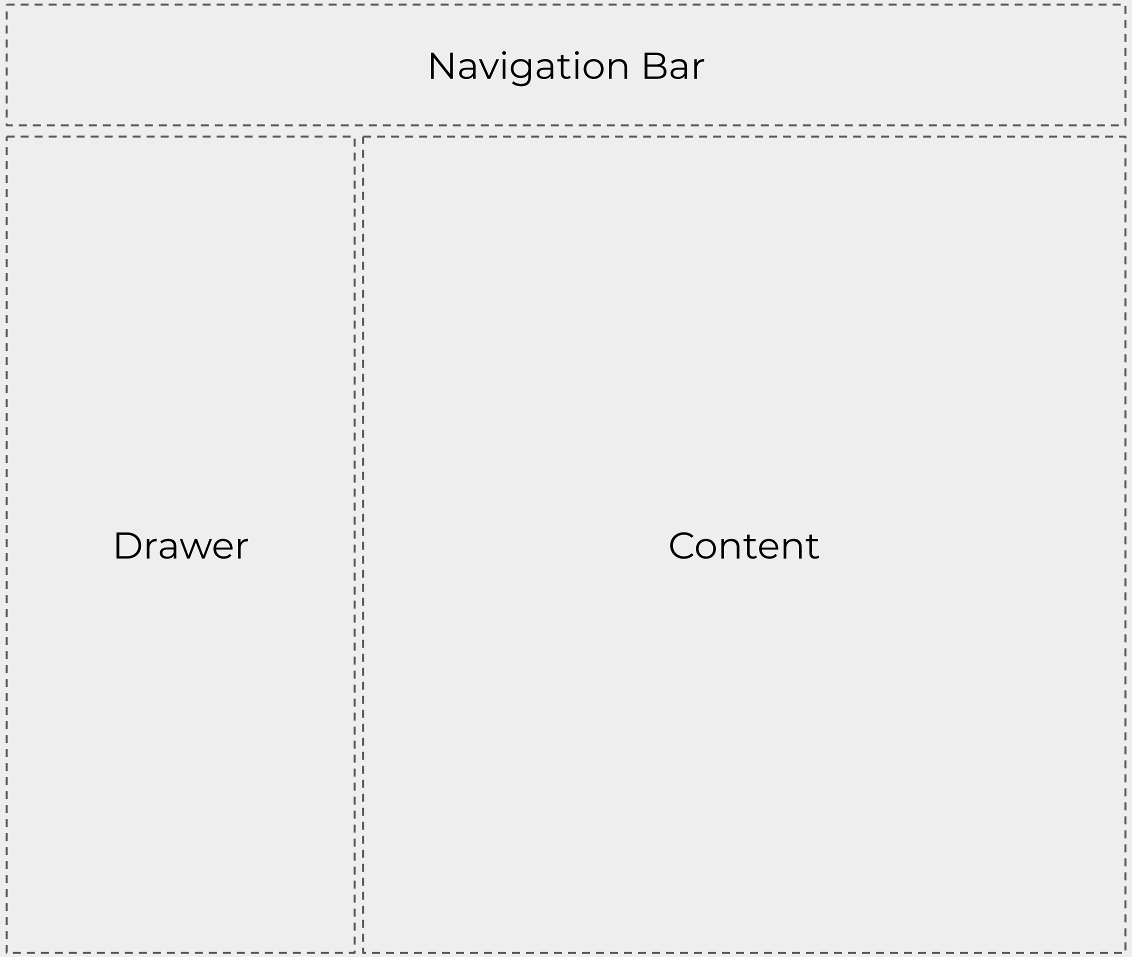
Use Board

App Layout

A quick and easy way to get a common application layout

App Layout

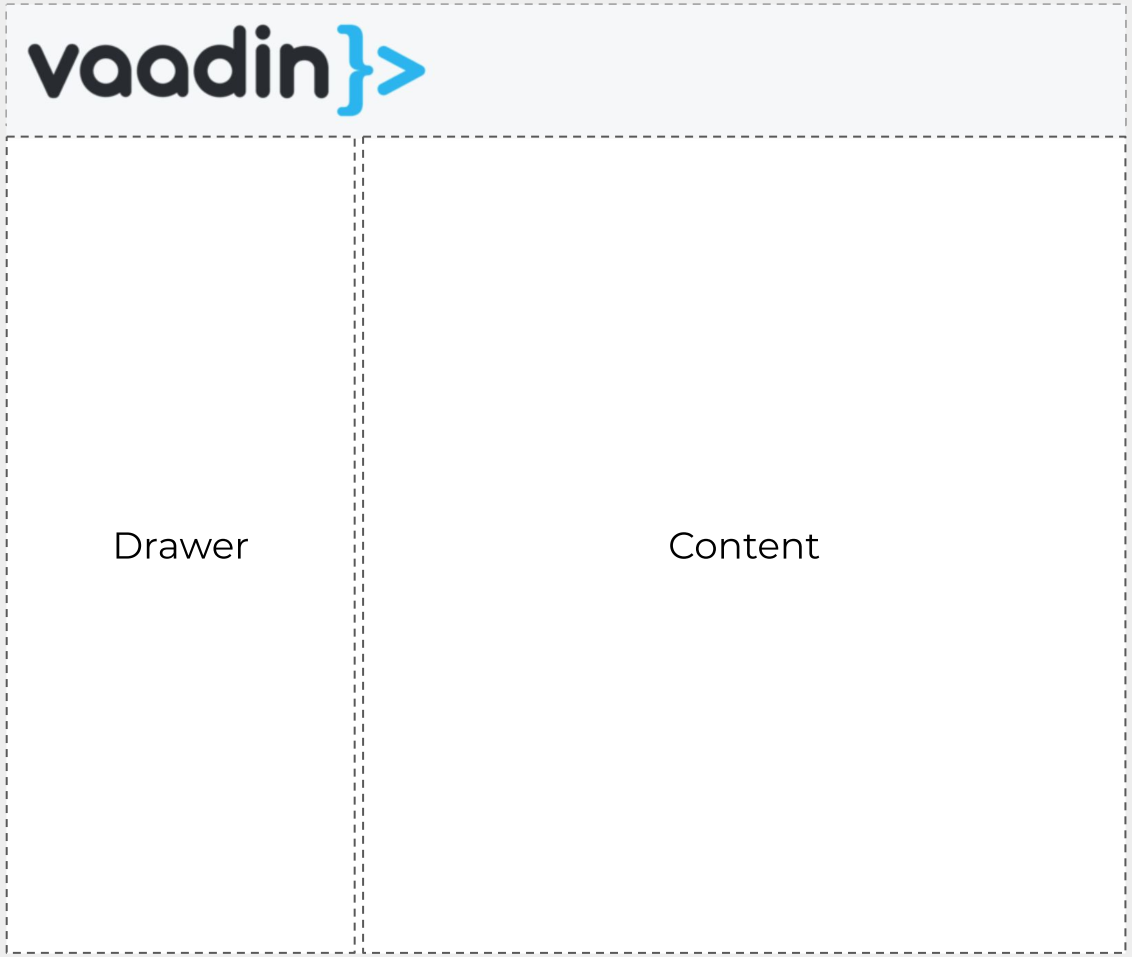
An App Layout contains 3 parts: the navigation bar, the drawer and the content.



App Layout

Add components to the navigation bar

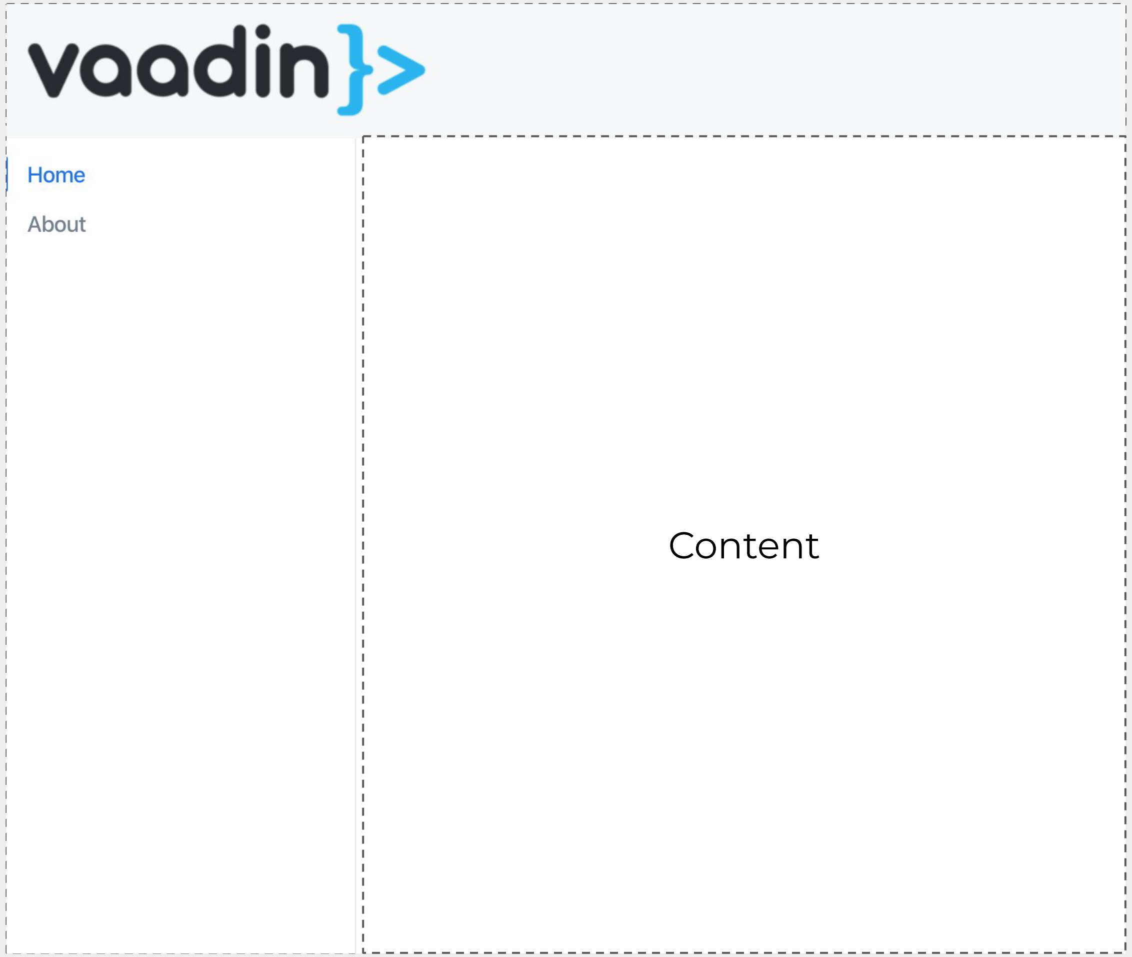
```
AppLayout appLayout = new AppLayout();

Image img = new Image("logo-url", "Logo");
appLayout.addToNavbar(img);
```



App Layout

Add components to the drawer

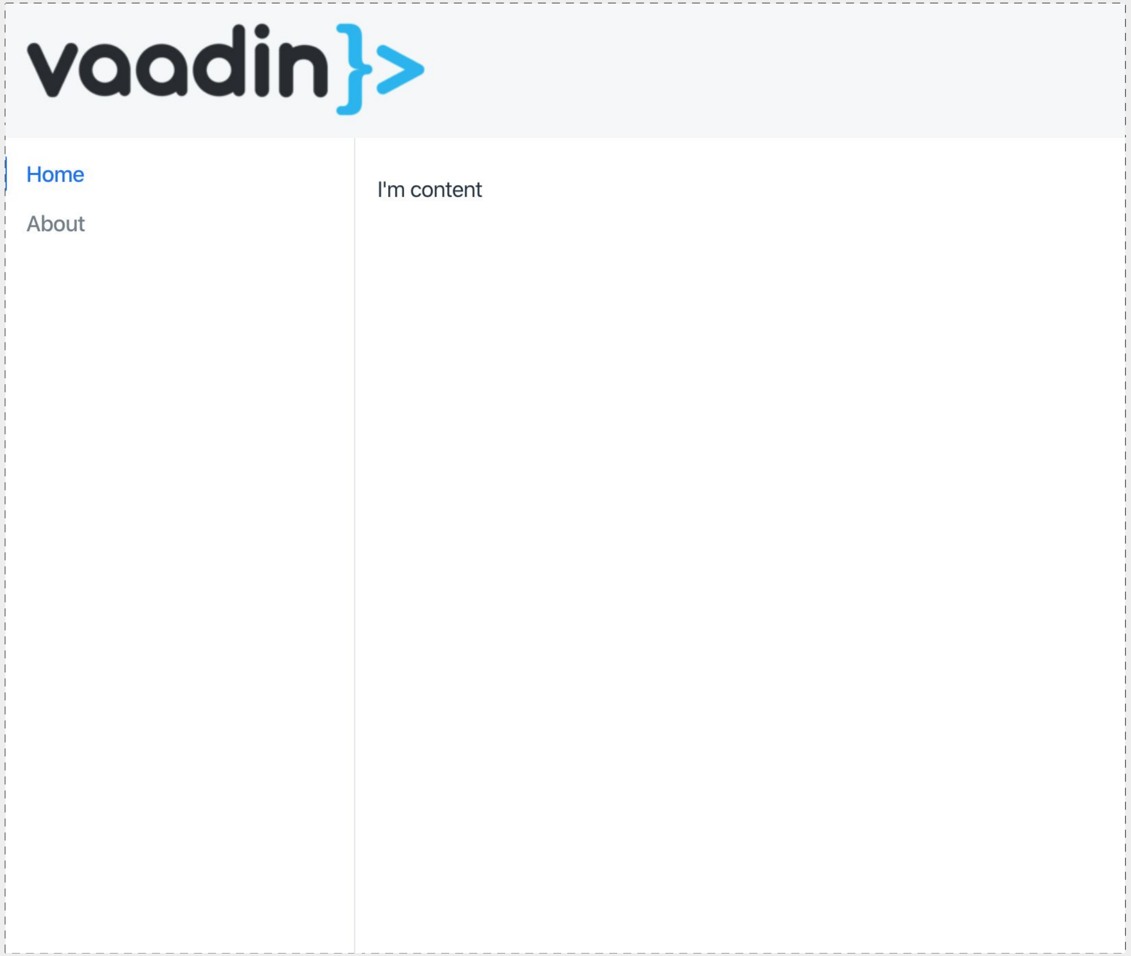
```
AppLayout appLayout = new AppLayout();

Tabs tabs = new Tabs(
        new Tab("Home"), new Tab("About"));
tabs.setOrientation(Tabs.Orientation.VERTICAL);
appLayout.addToDrawer(tabs);
```



App Layout

Set content

```
AppLayout appLayout = new AppLayout();

Component content = new Paragraph("I'm content");
appLayout.setContent(content);
```

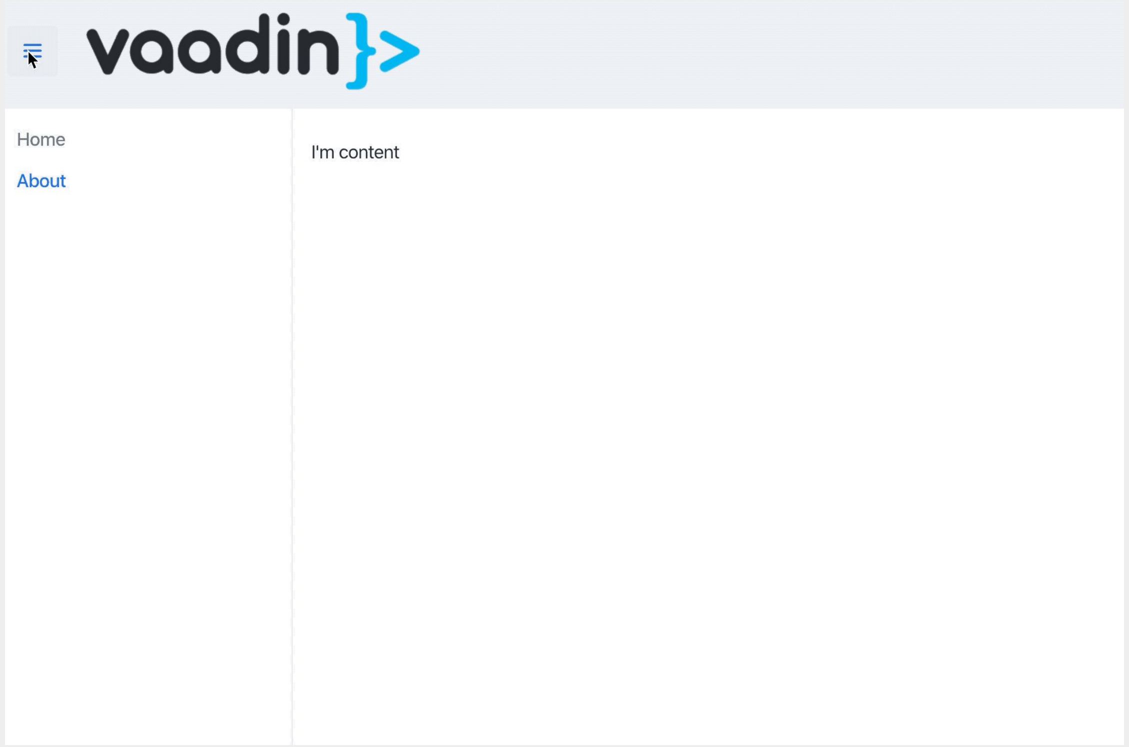


I'm content

App Layout

Add a drawer toggle button to control the visibility of the drawer.

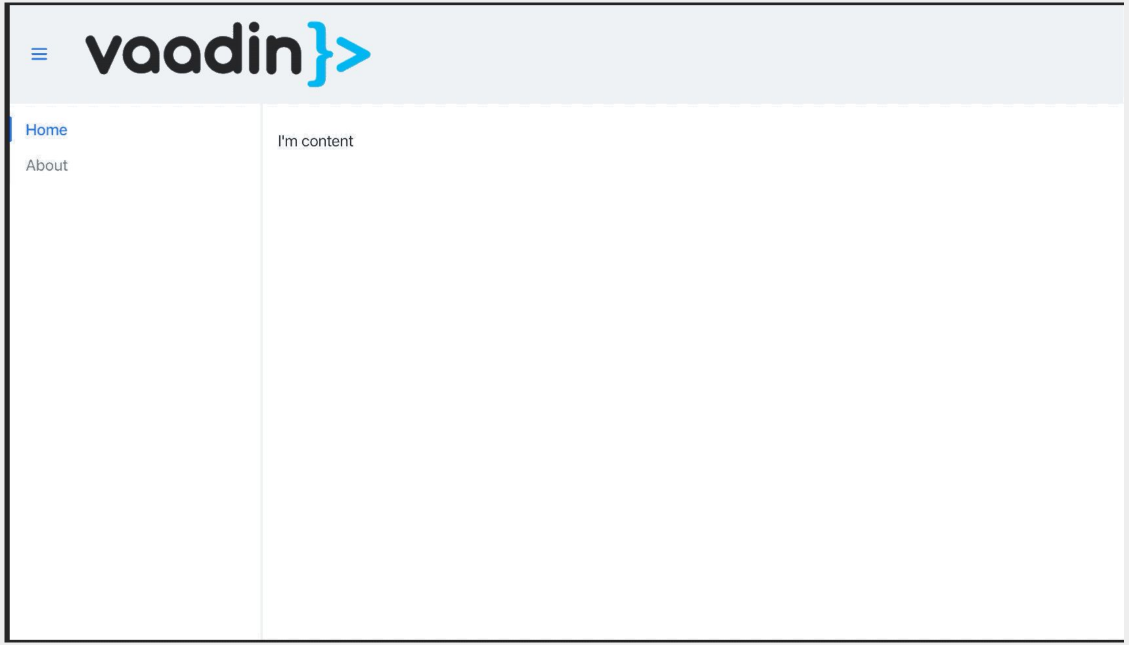
```
AppLayout appLayout = new AppLayout();

appLayout.addToNavbar(new DrawerToggle(),
      new Image("logo-url", "Logo"));
```



I'm content

App Layout

It’s responsive out-of-the-box



I'm content

Summary

- HorizontalLayout & VerticalLayout
- FlexLayout
- FormLayout
- Vaadin Board
- App Layout

Feedback

bit.ly/vaadin-training