Below are some websites and videos we watched to learn how to do and expand sections of the code we were unfamiliar with, and a brief summary of what came from that specific website.

1. Linear Regression in Java, medium.com
   https://arnavkartikeya.medium.com/multivariable-linear-regression-in-java-part-1-6bd2a4bf0b22
   This website is a good review over 2D arrays and how to use them in the ways that we need. It explains matrix multiplications, scaling operations, and how to build and manage the systems that expanded our knowledge and ability in the code.
2. CodeSignal, implementing regression
   https://codesignal.com/learn/courses/regression-and-gradient-descent/lessons/implementing-multiple-linear-regression-from-scratch
   This mostly covers implementation into python, but the concepts were useful and it made it far easier to manually implement into java.

Below are the sites referenced for the vaadin work, primarily taken from our own assignments 6 and 7.

- https://vaadin.com/docs/latest/getting-started
- https://vaadin.com/blog/learn-java-web-app-development-new-tutorial-series

Source: ChatGPT

What Vaadin component do I use to accept number input from a user?

In Vaadin, the standard component for numeric input is:

👉 **NumberField**

It's specifically designed for numbers (integers or decimals), and it handles validation, formatting, and user-friendly controls like step buttons.

Example:

```
NumberField numberField = new NumberField("Enter a number");
numberField.setMin(0);
numberField.setMax(100);
numberField.setStep(1);
numberField.setValue(10.0);
```

How do I display an image in a Vaadin view and set a default source?

In Vaadin, you display images using the Image component.

Basic example (with a default source)

```
Image image = new Image("images/default.png", "Default image");
image.setWidth("300px"); // optional
add(image);
```

Where to put the image file

- Place your image in:
  src/main/resources/META-INF/resources/images/default.png

Vaadin automatically serves files from that location, so "images/default.png" works as the path.

Setting or changing the source later

```
image.setSrc("images/another-image.png");
```

Using a default fallback image

If you want a “default” image when something else isn’t available:

```
String imagePath = (userImage != null) ? userImage : "images/default.png";
Image image = new Image(imagePath, "Profile image");
```

Bonus: loading from external URL

```
Image image = new Image("https://example.com/image.jpg", "External image");
```

If you’re planning to update the image dynamically (like based on user input or database values), I can show you how to wire that into your existing MainView setup.

How do I change the source of an existing Vaadin Image component after it has been created?

In Vaadin, you can change the source of an existing Image component using the setSrc() method.

Simple example

```
image.setSrc("images/new-image.png");
```

That's it — Vaadin will immediately update the image in the UI.

Example with user interaction

```
Button button = new Button("Change Image", e -> {
  image.setSrc("images/updated.png");
});
```

Using a dynamic value (e.g., from input)

```
TextField urlField = new TextField("Image URL");

Button update = new Button("Update Image", e -> {
  image.setSrc(urlField.getValue());
});
```

Notes

- The string you pass can be:
    - A **relative path** (e.g., "images/pic.png" from your resources folder)
    - A **full URL** ("https://...")
- The change is reactive—no refresh needed.

How do I place two Vaadin components side by side instead of stacked vertically?

In Vaadin, components stack vertically by default when you use something like VerticalLayout. To place them **side by side**, you want a horizontal container.

👉 Use HorizontalLayout

```
HorizontalLayout layout = new HorizontalLayout(component1, component2);
add(layout);
```

That's the simplest way—anything inside HorizontalLayout is arranged left-to-right.

Example

```
Button button = new Button("Click me");
TextField textField = new TextField("Name");

HorizontalLayout row = new HorizontalLayout(button, textField);
add(row);
```

Optional spacing & alignment

```
row.setSpacing(true);
row.setAlignItems(FlexComponent.Alignment.CENTER);
```

Alternative: FlexLayout (more control)

If you want finer control (wrapping, alignment, responsiveness):

```
FlexLayout flex = new FlexLayout(component1, component2);
flex.setFlexDirection(FlexLayout.FlexDirection.ROW);
add(flex);
```

Quick rule of thumb

- VerticalLayout → top to bottom
- HorizontalLayout → side by side
- FlexLayout → advanced/custom layouts

In Vaadin, you combine VerticalLayout and HorizontalLayout by **nesting them**—think of it like building rows inside columns.

- VerticalLayout = stacks sections (top → bottom)
- HorizontalLayout = arranges items within a section (left → right)

Basic pattern (most common)

```
VerticalLayout mainLayout = new VerticalLayout();

// Top section
H1 title = new H1("My App");

// Row with side-by-side components
TextField name = new TextField("Name");
Button submit = new Button("Submit");
HorizontalLayout formRow = new HorizontalLayout(name, submit);

// Another row
Image image = new Image("images/default.png", "Example");
Button change = new Button("Change Image");
HorizontalLayout imageRow = new HorizontalLayout(image, change);

// Add everything vertically
mainLayout.add(title, formRow, imageRow);
add(mainLayout);
```

How to think about it

You’re basically designing like this:

[Title]

[Name field | Submit button]

[Image | Change button]

Each **row** is a HorizontalLayout, and all rows are stacked in a VerticalLayout.

Controlling positioning & spacing

Center everything

```
mainLayout.setAlignItems(FlexComponent.Alignment.CENTER);
```

Space items in a row

```
formRow.setSpacing(true);
formRow.setJustifyContentMode(FlexComponent.JustifyContentMode.BETWEEN);
```

Make components expand

```
name.setWidthFull();
formRow.setWidthFull();
formRow.expand(name); // name takes remaining space
```

Example: Sidebar + content layout

```
VerticalLayout page = new VerticalLayout();

H1 header = new H1("Dashboard");

VerticalLayout sidebar = new VerticalLayout(new Button("Menu 1"), new Button("Menu 2"));
VerticalLayout content = new VerticalLayout(new Text("Main content here"));

HorizontalLayout body = new HorizontalLayout(sidebar, content);
body.setSizeFull();
body.expand(content); // content grows, sidebar stays fixed

page.add(header, body);
page.setSizeFull();
add(page);
```

Key idea

- Use **VerticalLayout for page structure**
- Use **HorizontalLayout for rows within that structure**

How do I use a string file path to load and display different images in Vaadin based on a condition?

In Vaadin, you can absolutely use a String file path (or URL) and swap images based on a condition using Image.setSrc().

The key idea is: **decide the path first, then assign it to the Image component**.

Basic conditional image loading

```
Image image = new Image();
image.setAlt("Dynamic image");

// some condition
boolean isHappy = true;

String path;
if (isHappy) {
   path = "images/happy.png";
} else {
   path = "images/sad.png";
}

image.setSrc(path);

add(image);
```

Cleaner version (ternary operator)

```
Image image = new Image();
```

```
boolean isHappy = true;

image.setSrc(isHappy ? "images/happy.png" : "images/sad.png");

add(image);
```

If your path comes from a variable (database, input, etc.)

```
String imagePathFromDb = getImagePath(); // e.g. "images/user1.png"

Image image = new Image(imagePathFromDb, "User image");
add(image);
```

Switching image later (dynamic update)

If the condition changes after the UI is already built:

```
if (score > 80) {
   image.setSrc("images/good.png");
} else {
   image.setSrc("images/try_again.png");
}
```

Important note about file locations

For local project images, use:

src/main/resources/META-INF/resources/images/

Then reference them like:

"images/file.png"

Vaadin serves that directory automatically.

What import do I need to use the so I can center my alignment in Vaadin HorizontalLayout?

To center alignment in a HorizontalLayout in Vaadin, you use the **FlexComponent alignment enums**.

Required import

import com.vaadin.flow.component.orderedlayout.FlexComponent;

Example: center items horizontally aligned layout

HorizontalLayout layout = new HorizontalLayout();
layout.setAlignItems(FlexComponent.Alignment.CENTER);

If you also want full centering (both axes)

You typically combine:

import com.vaadin.flow.component.orderedlayout.FlexComponent;
import com.vaadin.flow.component.orderedlayout.HorizontalLayout;

HorizontalLayout layout = new HorizontalLayout();
layout.setJustifyContentMode(FlexComponent.JustifyContentMode.CENTER);
layout.setAlignItems(FlexComponent.Alignment.CENTER);

- justifyContentMode → horizontal centering (left/right axis)
- alignItems → vertical centering (cross-axis)

Quick reference

Goal	Method
Center vertically in row	setAlignItems(CENTER)
Center horizontally in row	setJustifyContentMode(CENTER)
Fully center both ways	use both together

How do I toggle between light and dark mode in a Vaadin application using a button?

In Vaadin, dark/light mode is controlled through the **Lumo theme variants**. You toggle it by adding or removing a theme attribute on the UI root element.

✅ Basic idea

- **Light mode** = default (no dark theme)
- **Dark mode** = add Lumo.DARK class

🔧 Step 1: Import theme constants

```
import com.vaadin.flow.component.UI;
import com.vaadin.flow.theme.lumo.Lumo;
```

🔘 Step 2: Create a toggle button

```
Button themeToggle = new Button("Toggle theme");
```

🌓 Step 3: Toggle logic

```
themeToggle.addClickListener(e -> {
  UI ui = UI.getCurrent();

  String theme = ui.getElement().getAttribute("theme");

  if (theme != null && theme.contains(Lumo.DARK)) {
    ui.getElement().setAttribute("theme", "");
  } else {
    ui.getElement().setAttribute("theme", Lumo.DARK);
  }
});
```

🧱 Step 4: Add to layout

```
add(themeToggle);
```

💡 Cleaner version (recommended)

```
private void toggleTheme() {
   UI ui = UI.getCurrent();
   boolean dark = ui.getElement().getAttribute("theme") != null
         && ui.getElement().getAttribute("theme").contains(Lumo.DARK);

   ui.getElement().setAttribute("theme", dark ? "" : Lumo.DARK);
}
```

Then:

```
Button toggle = new Button("Toggle theme", e -> toggleTheme());
add(toggle);
```

⚠️ Optional improvement: persist user preference

If you want the theme to stay after refresh, you'd typically store it in:

- VaadinSession
- browser local storage (via JavaScript)
- or backend user settings

How do I print the full error details to the console when catching an exception in Java?

In Java, when you catch an exception, you have a few good ways to print the **full error details (stack trace)** to the console.

✅ 1. Best simple method: printStackTrace()

```
try {
   // code that may fail
} catch (Exception e) {
   e.printStackTrace();
}
```

What it does:

- Prints the exception type

- Prints the message
- Prints the full stack trace (line-by-line call path)

✅ 2. Print as a string (useful for logs)

```
try {
   // code
} catch (Exception e) {
   System.out.println(e.toString());
}
```

⚠️ This is **not full detail**—only the exception type and message.

✅ 3. Proper logging approach (recommended)

If you're using a logger:

```
import java.util.logging.Logger;

private static final Logger logger = Logger.getLogger(MyClass.class.getName());

try {
   // code
} catch (Exception e) {
   logger.severe("Error occurred:");
   e.printStackTrace();
}
```

Or better (logger-only stack trace):

```
logger.log(Level.SEVERE, "Error occurred", e);
```

✅ 4. Why printStackTrace() is useful

It shows:

- Where the error happened
- What method chain led to it

- Exact file + line number

Example output:

```
java.lang.NullPointerException
   at com.example.MyClass.method(MyClass.java:25)
   at com.example.App.main(App.java:10)
```

🚀 **Best practice**

For real applications (including Vaadin apps), prefer:

```
logger.log(Level.SEVERE, "Something failed", e);
```

because it integrates better with servers and logs.