Vaadin Flow Workshop

Anatol, Fabian, Lawrence

Agenda

- Introduction
- What is Vaadin Flow?
- Key Features
- Vaadin compared to alternatives
- Creating a Vaadin Flow application

What is Vaadin Flow?

Vaadin Flow helps you to quickly build web applications in pure Java — without writing any HTML or JavaScript

Key Features

SERVER-SIDE DEVELOPMENT

SEAMLESS INTEGRATION OF EXISTING JAVA LIBRARIES

RESPONSIVE DESIGN

THEMING AND STYLING

Vaadin vs other **Java** Web Frameworks

Vaadin Flow

- 100% development in **Java**
- Works with components
- **Server-sided** architecture
- High abstraction -> easy to learn and use

Google Web Toolkit

- Development in **Java** + **XML**
- Works with components
- **Client-sided** architecture
- Low abstraction -> hard to learn and use

Jakarta Faces

- Development in **Java** + **HTML**
- Works with HTML templates
- **Server-sided** architecture
- Medium abstraction -> easier to learn and use

Thymeleaf +

Spring Boot

- Development in **Java** + **HTML**
- Works with HTML templates
- **Server-sided** architecture
- Medium abstraction -> easier to learn and use

Vaadin vs other Web Frameworks

Vaadin Flow

- 100% development in **Java**
- Works with components
- **Server-sided** architecture
- High abstraction -> easy to learn and use

Vue

- Development in JavaScript + HTML
- Works with components
- **Client-sided** architecture
- Lightweight / Little overhead
- Offers little room to expand to big applications

React

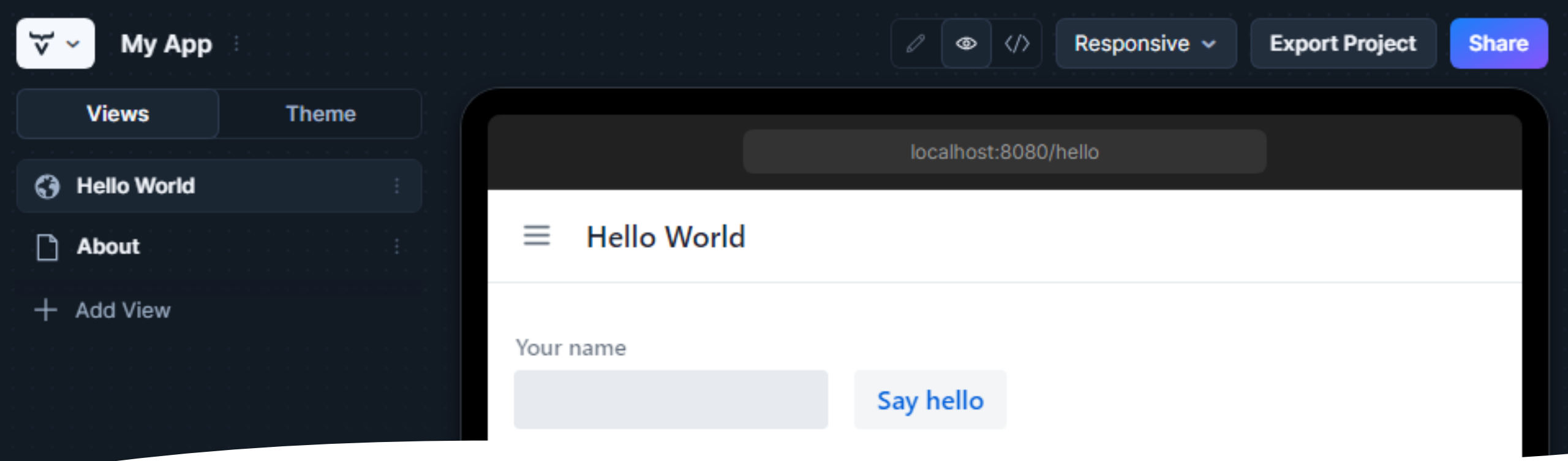
- Development in JavaScript + HTML
- Works with components
- **Client-sided** architecture
- Offers more control
- More complex concepts (e.g. states)

Angular

- Development in JavaScript + HTML
- Works with components
- **Client-sided** architecture
- Offers a lot of functionality
- Heavyweight

Setting up a project

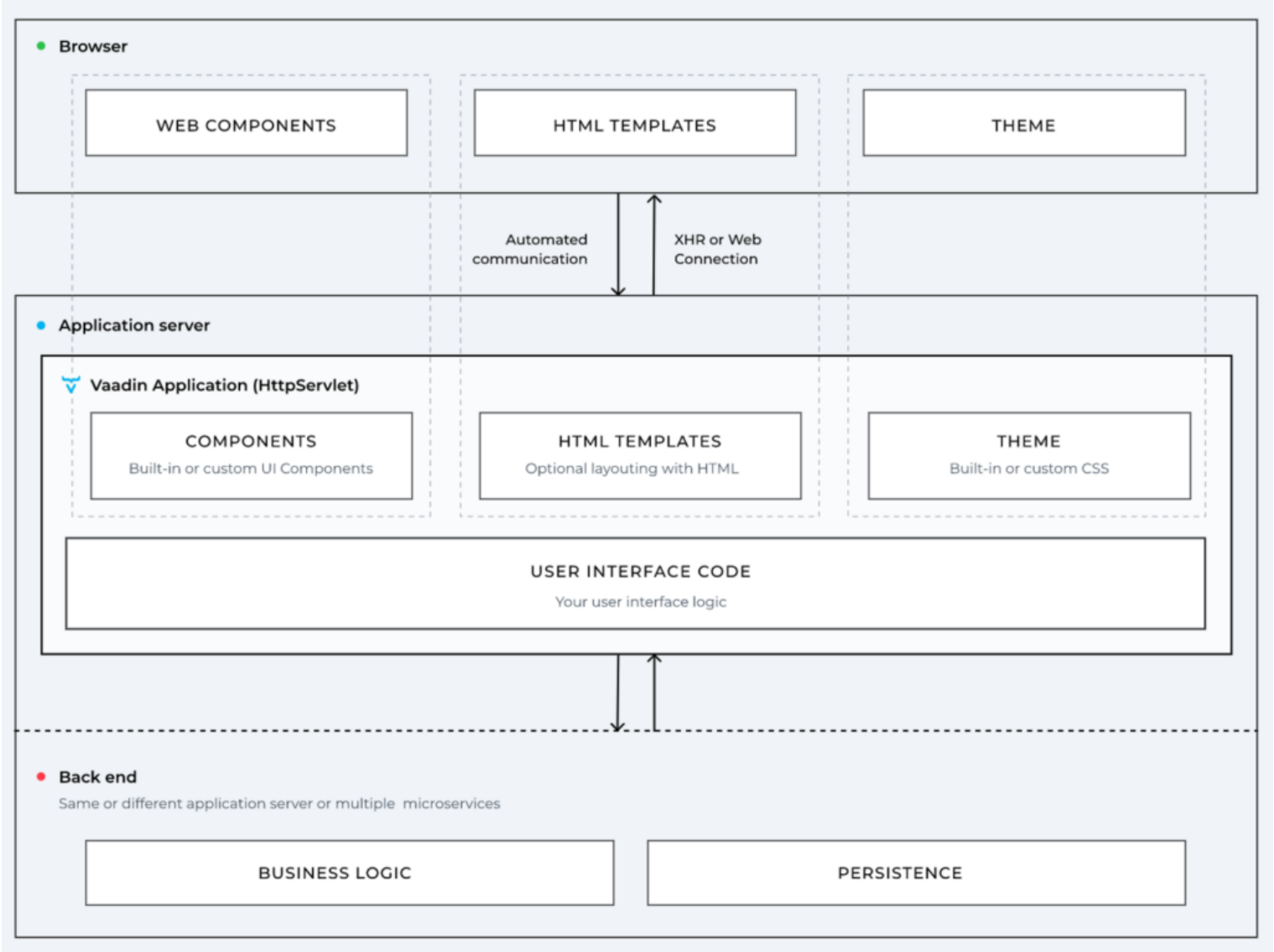
Requirements	•JDK 17 or higher
Starter Project	•https://start.vaadin.com/
Maven Archetype/ Gradle	



Vaadin Starter Project

- Configure and download core of Vaadin project
- Add several views
- Selection of more than 15 templates
- Add and modify JPA entities
- Security and configure access control
- Change look and feel
- Add helpful project settings

Architecture



Views

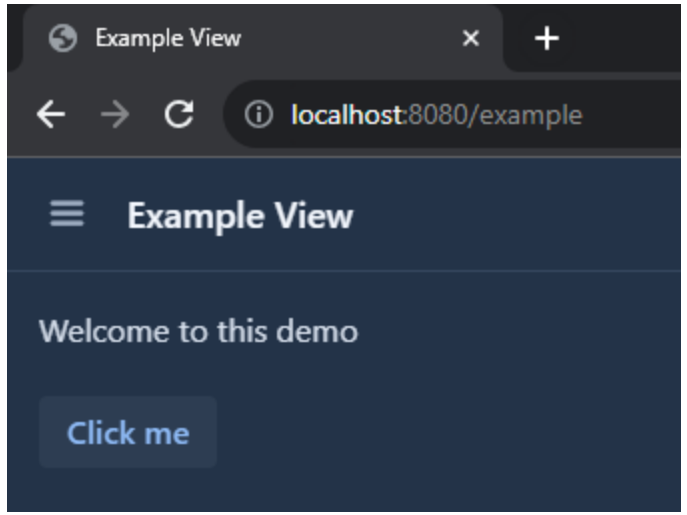
Views in Vaadin Flow

- Represents a part or section of a web application
- Help organize UI components with different layouts
- Each view corresponds to a URL

Creating a View

```java
@Route(value = "example")
@PageTitle("Example View")
public class ExampleView extends VerticalLayout {
    public ExampleView() {
        Text titleLabel = new Text("Welcome to this demo");
        Button clickButton = new Button( text: "Click me");

        add(titleLabel, clickButton);
    }
}
```



Navigating between Views

Using RouterLinks

```
RouterLink mainLink = new RouterLink( text: "Main View", MainView.class);
RouterLink secondaryLink = new RouterLink( text: "Secondary View", ExampleView.class);
```

Using UI

```
Button goToSecondaryViewButton = new Button( text: "Go to Secondary View");
goToSecondaryViewButton.addClickListener(e ->
        getUI().ifPresent(ui -> ui.navigate(ExampleView.class))
);
```

Components

Components in Vaadin Flow

- Building blocks used to create visual interactive elements
- Available pre-defined components

Creating a Component

```
public class ContactForm extends FormLayout {
    public ContactForm() {
        TextField nameField = new TextField( label: "Name");
        EmailField emailField = new EmailField( label: "Email");
        TextArea messageField = new TextArea( label: "Message");
        Button save = new Button( text: "Save");

        add(nameField, emailField, messageField, save);

    }
}
```

Styling Components

- Vaadin components come with built-in style variants *(Lumo)* that can be used to change the color, size, or other visual aspects
- Default styling of Vaadin components is based on **CSS** style properties that can be customized

Styling Components

Theme

```java
// Theme variants give you predefined extra styles for components.
// Example: Primary button has a more prominent look.
button.addThemeVariants(LUMO_PRIMARY);
```

Inline Styling

```java
// Set the inline style to change styling properties.
button.getStyle().set("color", "red");
```

CSS Class + File

```java
// Use custom CSS classes to apply styling. This is defined in
// styles.css.
addClassName("centered-content");
```

In styles.css

```css
/* Example: CSS class name to center align the content . */
.centered-content {
  margin: 0 auto;
  max-width: 250px;
}
```

Database Access

Database with JPA - Entities

- Represent Java objects that are mapped to database tables
- Entities encapsulate data and behavior
- Enables interaction with the database using object-oriented principles

Creating Entities

```
@Entity
public class Status extends AbstractEntity {
    private String name;

    public Status() {

    }

    public Status(String name) { this.name = name;

    public String getName() { return name; }

    public void setName(String name) { this.name =

}
```

Validating data

- Define data validation rules as Java Bean Validation annotations
- E.g @NotNull, @Size, @Past

```
@Email
@NotEmpty
private String email = "";
```

Database with JPA - Repositories

- Provides set of methods for performing common database operations on entities
- Simplifies data access and reduces the amount of boilerplate code

Creating Repositories

```
public interface ContactRepository extends JpaRepository<Contact, Long> {
        @Query("select c from Contact c " +
                "where lower(c.firstName) like lower(concat('%', :searchTerm, '%')) " +
                "or lower(c.lastName) like lower(concat('%', :searchTerm, '%'))")
        List<Contact> search(@Param("searchTerm") String searchTerm);
```

Database with JPA - Services

- Encapsulates business logic
- Act as layer between data access layer and presentation layer

Creating Services

```
@Service
public class CrmService {

    private final ContactRepository contactRepository;
    private final CompanyRepository companyRepository;
    private final StatusRepository statusRepository;

    public CrmService(ContactRepository contactRepository,
                      CompanyRepository companyRepository,
                      StatusRepository statusRepository) {
        this.contactRepository = contactRepository;
        this.companyRepository = companyRepository;
        this.statusRepository = statusRepository;
    }

    public List<Contact> findAllContacts(String stringFilter) {
        if (stringFilter == null || stringFilter.isEmpty()) {
            return contactRepository.findAll();
        } else {
            return contactRepository.search(stringFilter);
        }
    }
}
```

Service Annotation

Repository Declaration & Initialization

Logic

Data Binding

Binding data objects to input

- Binder class allows to define how the values in an object are bound to fields in the UI
- Allows for seamless synchronization of data between the UI components and the backend

Binding data objects to input

```
TextField nameField = new TextField( label: "Name");
Text nameLabel = new Text("Person Name: ");

Person person = new Person();
Binder<Person> binder = new Binder<>(Person.class);

binder.bind(nameField, Person::getName, Person::setName);

binder.addValueChangeListener(event -> {
    String updatedName = event.getValue().toString();
    nameLabel.setText("Person Name: " + updatedName);
});
```

Our Java BEAN

UI Components

Binding UI to BEAN

Validating input

Validator Class

```
binder.forField(emailField)
        // Explicit validator instance
        .withValidator(new EmailValidator(
                errorMessage: "This doesn't look like a valid email address"))
        .bind(Person::getEmail, Person::setEmail);
```

Field to validate

Predefined Validator Class

BEAN Binding

Lambda

```
binder.forField(nameField)
        // Validator defined based on a lambda
        // and an error message
        .withValidator(
                name -> name.length() >= 3,
                message: "Name must contain at least three characters")
        .bind(Person::getName, Person::setName);
```

True = valid, False = invalid

Message if invalid

Required

```
binder.forField(titleField)
        // Shorthand for requiring the field to be non-empty
        .asRequired("Every employee must have a title")
        .bind(Person::getTitle, Person::setTitle);
```

Not-Null Check

Thanks for your attention

Sources

- https://vaadin.com/docs/latest/overview
- https://www.baeldung.com/vaadin
- https://72.services/en/vaadin-community-award/
- Comparison
  - https://www.gwtproject.org/doc/latest/DevGuide.html
  - https://blog.payara.fish/getting-started-with-jakarta-ee-9-jakarta-faces-jsf
  - https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html
  - https://vaadin.com/comparison

How to get started

- Visit https://sebivenlo.github.io/ESD-2023-Vaadin-Flow/