| Fontys University of Applied Sciences

Design Document

Himalayan Bus booking

Himal Aryal

2023

Table of Contents

[Architecture Constraints and Design Decisions 3](#_Toc152349659)

[What is Spring Boot? 3](#_Toc152349660)

[Why am I using Spring Boot? 3](#_Toc152349661)

[What is React? 4](#_Toc152349662)

[Why am I using React? 4](#_Toc152349663)

[What is MySQL? 5](#_Toc152349664)

[Why am I using MySQL? 5](#_Toc152349665)

[C4 Model Diagrams 6](#_Toc152349666)

[Level 1: System Context Diagram 6](#_Toc152349667)

[Level 2: Container diagram 8](#_Toc152349668)

[Level 3: Component Diagram 9](#_Toc152349669)

[Connecting with software principles: 12](#_Toc152349670)

[SOLID Principles: 12](#_Toc152349671)

[Dependency Inversion Principle (DIP): 12](#_Toc152349672)

[KISS (Keep It Simple, Stupid): 12](#_Toc152349673)

[Separation of Concerns 12](#_Toc152349674)

[DRY (Don't Repeat Yourself): 12](#_Toc152349675)

[Centralized Data Handling: 12](#_Toc152349676)

[YAGNI (You Ain't Gonna Need It): 12](#_Toc152349677)

[Minimal Components and Services: 12](#_Toc152349678)

[CI Setup Diagram 13](#_Toc152349679)

[Graphical user interface, text, application, chat or text message

Description automatically generated 13](#_Toc152349680)

# Architecture Constraints and Design Decisions

## What is Spring Boot?

Spring Boot is an open-source Java framework developed by Pivotal (now part of VMware) that simplifies the process of building and deploying Java applications. It is a part of the broader Spring ecosystem, which provides a comprehensive platform for developing Java-based enterprise applications.

Spring Boot is designed to make it easier to create stand-alone, production-grade Spring-based applications with minimum configuration and effort. It achieves this by providing a set of pre-configured templates, commonly used libraries, and a convention-over-configuration approach.

### Why am I using Spring Boot?

I'm using Spring Boot for several key reasons:

1. **Streamlined Development:** Spring Boot simplifies Java application development by providing opinionated defaults and eliminating much of the manual configuration work. This allows me to focus on writing application logic rather than boilerplate code.
2. **Rapid Prototyping:** Spring Boot's quick setup and auto-configuration help me build prototypes and proof-of-concept applications efficiently.
3. **Microservices:** Spring Boot is an excellent choice for developing microservices, thanks to its support for standalone applications and integration with Spring Cloud for microservices-specific features.
4. **Production-Ready:** It includes built-in features for production readiness, such as health checks and externalized configuration, which are essential for deploying robust applications.
5. **Community Support:** The large and active Spring Boot community provides extensive documentation, tutorials, and a wealth of third-party libraries and extensions, making it easier to find solutions to common problems.
6. **Integration:** Spring Boot's seamless integration with various technologies and frameworks makes it versatile for integrating with databases, messaging systems, REST APIs, and more.
7. **Auto-Configuration:** The auto-configuration feature saves time by automatically configuring application components based on dependencies, reducing manual setup.
8. **Stand-Alone Applications:** Spring Boot allows me to package my applications as standalone JAR files, enabling easy deployment without external application servers.
9. **Enterprise-Grade:** It provides features for building enterprise-grade applications, including robust security options, transaction management, and a wide array of data access methods.
10. **Ecosystem of Spring Projects:** Spring Boot seamlessly integrates with other Spring projects, offering a comprehensive ecosystem for building and maintaining Java applications.

In summary, Spring Boot simplifies and accelerates Java application development, making it a versatile choice for a wide range of projects, from small prototypes to large-scale, mission-critical systems.

## What is React?

React, often referred to as React.js or ReactJS, is an open-source JavaScript library for building user interfaces (UIs) or user interface components. Developed and maintained by Facebook, react is one of the most popular and widely used libraries for front-end web development.

React is commonly used for building web applications, particularly single-page applications (SPAs), where the UI can change dynamically without requiring a full page reload. It can be combined with other libraries and technologies, such as React Router for routing, Redux for state management, and Axios for making HTTP requests, to create robust web applications. Additionally, React Native is a related technology that allows developers to build mobile applications for iOS and Android using React principles.

### Why am I using React?

I'm using React for several reasons:

1. **Component-Based Architecture:** React component-based architecture allows me to create reusable UI components, making it easier to build and maintain complex user interfaces.
2. **Virtual DOM:** React use of the virtual DOM results in efficient UI updates, minimizing direct manipulation of the actual DOM, and thus improving performance.
3. **Declarative Approach:** React declarative style of programming simplifies UI development. I describe what the UI should look like and react handles the underlying updates for me.
4. **Reactivity:** React enables the creation of responsive and interactive user interfaces by managing events, state, and rendering based on data and user actions.
5. **Unidirectional Data Flow:** React one-way data flow simplifies data management and makes it clear how changes propagate through the UI.
6. **JSX for Structure:** JSX makes it convenient to define the structure and appearance of components, combining HTML-like syntax within JavaScript.
7. **Ecosystem:** React has a rich ecosystem of libraries and tools that complement its functionality, including state management solutions, routing libraries, and various UI component libraries.
8. **Community and Support:** React has a vibrant community with extensive documentation, tutorials, and resources, making it easier to learn and work with.

In summary, I'm using React because it provides a powerful and efficient framework for building dynamic and interactive user interfaces, and it is supported by a thriving community and a wealth of resources.

## What is MySQL?

MySQL is an open-source relational database management system (RDBMS). It is one of the most widely used database systems in the world and is popular for a variety of applications, from small personal projects to large-scale enterprise systems. MySQL is known for its speed, reliability, and ease of use.

MySQL is commonly used in web applications, content management systems (CMS), e-commerce platforms, and various other software projects that require robust data storage and retrieval capabilities. It's an essential component in the technology stack of many organizations and developers.

### Why am I using MySQL?

I'm using MySQL for several reasons:

1. **Relational Database:** MySQL is a mature and well-established relational database management system (RDBMS) that is trusted by a large user base and has a proven track record of reliability.
2. **Open Source:** MySQL is open-source software, which means it's freely available, and I can benefit from a large and active open-source community for support and development.
3. **Scalability:** MySQL can scale to handle large datasets and high-traffic websites when properly configured, making it suitable for a wide range of applications, from small projects to enterprise-level systems.
4. **ACID Compliance:** MySQL is ACID-compliant, ensuring the reliability and consistency of data, which is crucial for applications where data integrity is a priority.
5. **Performance:** With proper optimization and indexing, MySQL offers robust performance for data retrieval and manipulation operations.
6. **Cross-Platform:** MySQL is compatible with multiple platforms and operating systems, making it versatile for various deployment scenarios.
7. **Security:** MySQL provides features like user authentication, access control, and encryption to secure data and protect against unauthorized access.
8. **Community Support:** MySQL has a large and active user community, which means that there is an abundance of documentation, forums, and resources available to help resolve issues and find answers to questions.
9. **Integration:** MySQL can be easily integrated with various programming languages, frameworks, and tools, making it suitable for a broad spectrum of applications.

In summary, I'm using MySQL because it is a reliable, open-source relational database system that offers performance, scalability, and a strong community of users and developers. It's well-suited for a wide range of applications, from small projects to enterprise-level database systems.

## C4 Model Diagrams

## Level 1: System Context Diagram

A diagram of a system

Description automatically generated

At Level 1, we have the "Bus Booking System" as our core system. It interacts with two external systems: "Client-Side Application" and "Mainframe Booking System."

* **Bus Booking System**: This represents the core of our application, responsible for managing bus bookings. It connects to the "Client-Side Application" and the "Mainframe Booking System."
* **Client-Side Application**: This is the user-facing part of our system, built using React. It enables users to access system functionalities based on their roles. It connects to the internal Microsoft Exchange e-mail system for sending emails.
* **Mainframe Booking System**: This system handles essential booking data and connects to the Bus Booking System for data exchange. It is connected to the internal Microsoft Exchange e-mail system.

## Level 2: Container diagram

A diagram of a diagram

Description automatically generated

At Level 2, we break down the "Bus Booking System" into several containers or subsystems, and we introduce the "E-mail System" and "Database."

* **Bus Booking System**: This container represents the core of the system. It includes the following components: "User Controller," "Bus Controller," "Booking Controller," "Route Controller," "Feedback Controller," and "Admin Controller." Each controller corresponds to CRUD operations related to its respective domain.
* **E-mail System**: This container connects to the "Client-Side Application" and "Mainframe Booking System." It is responsible for sending emails. We can apply the SOLID principle, particularly the Single Responsibility Principle (SRP), by keeping the email-sending functionality separate from the main application logic.
* **Database**: The database container stores essential booking data and handles interactions with components within the "Bus Booking System." It adheres to the Single Responsibility Principle by centralizing data storage and retrieval.

## Level 3: Component Diagram

A screenshot of a computer

Description automatically generated

At Level 3, we further break down the components within the "Bus Booking System" container, showing how they interact with services and repositories.

* **User Controller**: This component connects to the "User Service" and "User Repository." It handles CRUD operations related to users. The use of the Dependency Inversion Principle can be noted here, as the controller depends on abstractions (services and repositories) rather than concrete implementations.
* **Bus Controller**: Similarly, the "Bus Controller" connects to "Bus Service" and "Bus Repository" and is responsible for CRUD operations related to buses.
* **Booking Controller**: This component is responsible for handling booking operations and connects to "Booking Service" and "Booking Repository."
* **Route Controller**: This component connects to "Route Service" and "Route Repository" to manage routes.
* **Feedback Controller**: Handles user feedback and connects to "Feedback Service" and "Feedback Repository."
* **Admin Controller**: Manages admin-related functionality and connects to "Admin Service" and "Admin Repository."
* **E-mail System**: This component in the "E-mail System" container sends emails to various recipients.
* **Database**: The "Database" component represents interactions with the MySQL database and includes "User Repository," "Bus Repository," "Booking Repository," "Route Repository," "Feedback Repository," and "Admin Repository."

A diagram of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Connecting with software principles:

## SOLID Principles:

Dependency Inversion Principle (DIP): This principle encourages high-level modules to depend on abstractions rather than concrete implementations. In the component diagram, you can see this principle at work as the controllers (e.g., User Controller, Bus Controller) depend on abstractions (services and repositories) rather than concrete database implementations. This allows for greater flexibility and ease of testing because you can swap out implementations without affecting the controllers.

## KISS (Keep It Simple, Stupid):

Separation of Concerns**:** KISS advocates breaking your system down into smaller, simpler components with well-defined responsibilities. In the C4 diagram, you can see this with the separation of the "E-mail System" from the core "Bus Booking System." Keeping email functionality separate from the main application logic makes it easier to manage and maintain, adhering to the principle of keeping things simple and focused.

## DRY (Don't Repeat Yourself):

Centralized Data Handling: In the component diagram, the "Database" component centralizes data storage and retrieval logic for user, bus, booking, route, feedback, and admin data. This prevents the need to duplicate database-related code in multiple controllers. Instead of rewriting the same database access code in each controller, it's centralized in one place, which reduces redundancy and ensures consistency in data handling. This adherence to DRY can lead to more maintainable code.

## YAGNI (You Ain't Gonna Need It):

Minimal Components and Services: YAGNI is a principle that suggests not adding features or components to your system until they are needed. In the component diagram, the components and services are minimal and focused on their specific responsibilities. This means that unnecessary complexity or features are avoided until there is a clear requirement for them. It helps keep the system lightweight and avoids overengineering.

In summary, applying these software principles in the C4 model diagram ensures that the architecture is well-structured, easy to maintain, and adaptable to future changes. It encourages a clean separation of concerns, reduces redundancy, and keeps the system simple and focused on the most essential functionality, preventing unnecessary complexity or feature bloat.

# CI Setup Diagram

# Graphical user interface, text, application, chat or text message Description automatically generated