Plan To Code

# User Interfaces

**Finance Dashboard**

**HR Dashboard**

**Supply Chain Management Dashboard**

**Production and Operations Dashboard**

**Environmental Management Dashboard**

**Project Management Dashboard**

# Technologies to Use

* **Backend:** Java with Spring Boot
* **Database:** MySQL
* **Authentication:** Spring Security
* **APIs:** RESTful APIs with Spring Boot
* **Deployment:** Docker and Kubernetes
* **CI/CD:** Jenkins or GitLab CI

# Initialize Spring Boot Project:

* Go to [Spring Initializer](https://start.spring.io/).
* Choose the following dependencies:
  + Spring Web
  + Spring Data JPA
  + MySQL Driver
  + Spring Boot DevTools

# Project Structure

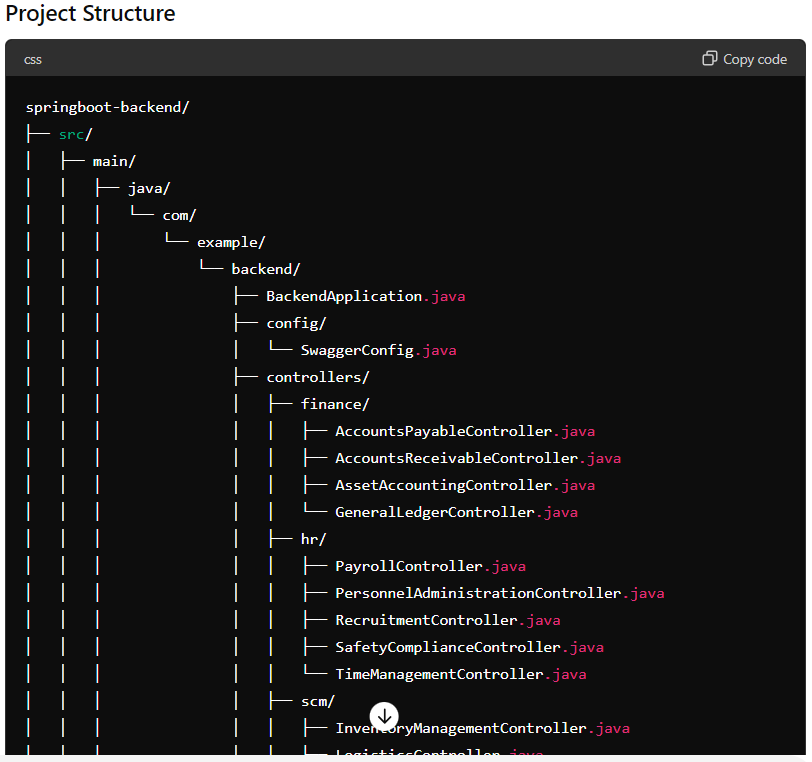


Figure 1

A screenshot of a computer program

Description automatically generated

Figure 2

A screenshot of a computer program

Description automatically generated

Figure 3

A screenshot of a computer program

Description automatically generated

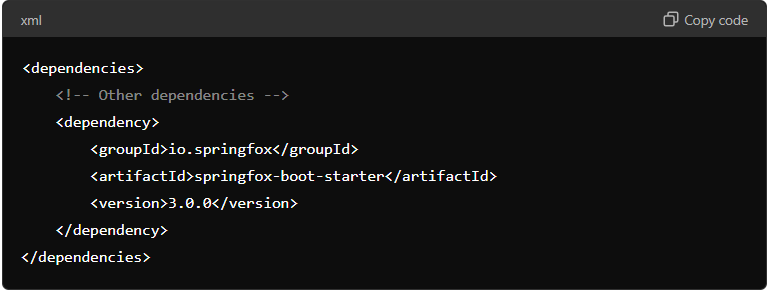
Figure 4

A screenshot of a computer program

Description automatically generated

Figure 5

**Add Springfox Dependencies**: Add the necessary dependencies in your pom.xml file.

****

**Configure Application Properties (Optional)**: You can customize Swagger's behavior by adding properties in application. Properties.

A black and white text

Description automatically generated

The Swagger UI provides a web interface for exploring and testing your API endpoints.

**Access Swagger UI**: Once you run your application, you can access the Swagger UI at <http://localhost:8080/swagger-ui/>

# Purpose of Entities

Entities in the context of a Spring Boot application using JPA (Java Persistence API) represent the data model. An entity is a lightweight, persistent domain object that typically corresponds to a table in a relational database. Here’s a detailed explanation of why we create entities and what they are used for:

* **Mapping to Database Tables**: Entities represent the structure of the data in your application and are directly mapped to database tables. Each entity corresponds to a table, and each instance of the entity corresponds to a row in that table.
* **Object-Relational Mapping (ORM)**: Entities facilitate ORM, which allows developers to interact with the database using Java objects instead of writing raw SQL queries. ORM frameworks like Hibernate (used by JPA) handle the mapping between Java classes and database tables.
* **Data Manipulation**: Entities are used to perform CRUD (Create, Read, Update, Delete) operations. JPA repositories provide methods for these operations, making it easier to manage database interactions.
* **Business Logic Encapsulation**: Entities often contain business logic that pertains to the data they represent. This can include validation rules, calculated properties, and methods that operate on the data.
* **Transaction Management**: Entities participate in transactions. JPA ensures that data integrity is maintained by managing transactions that involve one or more entities.

# Purpose of Repositories

Repositories in the context of Spring Data JPA are interfaces that provide methods for performing CRUD operations on entities. They act as a data access layer, encapsulating the logic required to interact with the database. By using repositories, you can perform database operations without writing boilerplate code.

### Key Points about Repositories

1. **CRUD Operations**: Repositories provide methods for Create, Read, Update, and Delete operations out-of-the-box.
2. **Custom Queries**: You can define custom query methods using query derivation from method names or by using JPQL (Java Persistence Query Language) or native SQL queries.
3. **Transactional Management**: Spring Data JPA handles transactions for repository methods automatically, ensuring data consistency.

# Service

In a Spring Boot application, the service layer is responsible for implementing the business logic. It interacts with the repository layer to fetch or store data in the database and then performs the necessary operations before passing the data to the controller layer, which handles HTTP requests and responses.

### Key Points about the Service Layer

1. **Business Logic**: The service layer contains the core business logic of the application.
2. **Transaction Management**: It can manage transactions, ensuring data consistency.
3. **Decoupling**: Separates the business logic from the controller and repository layers, promoting a clean architecture.

# Controller

The controller layer in a Spring Boot application is responsible for handling HTTP requests, processing them (often delegating to the service layer), and returning HTTP responses. It serves as the entry point for the client's interaction with the backend application.

## Summary

* **Controller**: Handles HTTP requests and responses, delegating business logic to the service layer.
* **Service**: Contains business logic, interacts with the repository layer.
* **Repository**: Directly interacts with the database, performing CRUD operations.