# Cargo Load Management System

Creating a detailed implementation of a Spring Boot MVC web project with JSP pages for cargo, truck, driver, and route management is a comprehensive task. Below, I'll provide a step-by-step guide to building this project. Note that this is a high-level overview, and you should implement each step with proper coding practices, folder structure, and separation of concerns.

Step 1: Project Setup

1. Create a new Spring Boot project using Spring Initializer with the following dependencies:

- Spring Web

- Spring DevTools

-lombok

-tomcat server

Spring data jpa

Mysql

2. Generate the project and open it in your preferred IDE.

Step 2: Define Domain Models

Define the domain models for Cargo, Truck, Driver, and Route. Create separate Java classes for each entity with appropriate attributes, getters, setters, and relationships.(JPA)

Step 3: Create Repositories (Excluded in this Assignment)

In a real-world scenario, you would create repositories for each entity using Spring Data JPA. However, this assignment excludes JPA repositories.

Step 4: Create Controllers

Create controllers for cargo, truck, driver, and route management. Each controller should have methods to handle various operations (e.g., create, read, update, delete).

Step 5: Create JSP Views

For each entity (cargo, truck, driver, route), create JSP views that allow users to perform CRUD operations. Here's a basic structure for each entity's JSP:

- Cargo Management:

- `cargo-list.jsp`: Display a list of cargos.

- `cargo-create.jsp`: Create a new cargo.

- `cargo-edit.jsp`: Edit an existing cargo.

- `cargo-details.jsp`: Show cargo details.

- Truck Management:

- `truck-list.jsp`: Display a list of trucks.

- `truck-create.jsp`: Create a new truck.

- `truck-edit.jsp`: Edit an existing truck.

- `truck-details.jsp`: Show truck details.

- Driver Management:

- `driver-list.jsp`: Display a list of drivers.

- `driver-create.jsp`: Create a new driver.

- `driver-edit.jsp`: Edit an existing driver.

- `driver-details.jsp`: Show driver details.

- Route Management:

- `route-list.jsp`: Display a list of routes.

- `route-create.jsp`: Create a new route.

- `route-edit.jsp`: Edit an existing route.

- `route-details.jsp`: Show route details.

Step 6: Configure View Resolver

Configure the Spring MVC view resolver to resolve JSP views. Set up the `InternalResourceViewResolver` in your `application.properties` or `application.yml` file.

Step 7: Implement Controller Logic

In each controller, implement the logic to handle CRUD operations. Use appropriate service classes to perform business logic. Example methods might include:

- `listEntities()`: To list all entities.

- `showEntityForm()`: To display the form for creating or editing an entity.

- `saveEntity()`: To save a new entity or update an existing one.

- `deleteEntity()`: To delete an entity.

Step 8: Create Service Classes (Excluded in this Assignment)

In a real-world scenario, you would create service classes to handle business logic, including validations, calculations, and database interactions. However, this assignment excludes service classes for simplicity.

Step 9: Define Routes

Create routes in your controllers to map URLs to controller methods. Ensure that routes are RESTful and follow best practices.

Step 10: Frontend Design

Design the frontend of your JSP pages with HTML and CSS. Use forms to capture user input and display data in tables. Make use of JSTL (JavaServer Pages Standard Tag Library) for control structures and expression language (EL) for displaying data.

Step 11: Run and Test the Application

Start your Spring Boot application and test the functionality of each module (cargo, truck, driver, route) through a web browser.