

Topic :- Train ticket system

Introduction:

The Train Ticket Reservation System is a comprehensive web-based platform engineered to modernize and centralize the lifecycle of railway travel management. While the core of the application focuses on the efficient handling of passenger records—including adding, viewing, and updating reservations—it extends beyond a simple data repository to serve as a high-performance transactional engine. By integrating **Spring Security** for role-based authentication and **Spring JDBC** for optimized database communication, the system ensures that sensitive passenger data is handled securely while maintaining real-time accuracy of seat inventory.

The application's architecture is designed to bridge the gap between complex backend business logic and an intuitive user experience. Utilizing a responsive frontend built with **Thymeleaf** and **Bootstrap**, the system provides a seamless interface for both passengers searching for available journeys and administrators managing fleet schedules. This transition from manual paper-based tracking to an automated **PostgreSQL**-driven environment significantly reduces human error, prevents overbooking through automated constraint logic, and provides a scalable foundation for high-volume ticketing operations.

Application Flow :

User Browser



PassengerController (home method)



PassengerRepository.findAll() → Database



Model { passengers: [...] }



Thymeleaf renders index.html



HTML page displayed with passenger table

Step1:-

Go on spring initializer to create the spring project and add dependency

The screenshot shows the Spring Initializer web interface. On the left, there's a sidebar with a green circular icon and the text "spring initializr". The main area is divided into sections:

- Project**: Options for Gradle - Groovy, Gradle - Kotlin, Java (selected), Kotlin, Groovy, and Maven.
- Language**: Java (selected), Kotlin, Groovy.
- Dependencies**: A button labeled "ADD DEPENDENCIES... CTRL + B".
- Spring Web [WEB]**: Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Thymeleaf [TEMPLATE ENGINES]**: A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.
- PostgreSQL Driver [SQL]**: A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.
- JDBC API [SQL]**: Database Connectivity API that defines how a client may connect and query a database.

Below these sections, there are fields for Project Metadata, Configuration, and Java version selection.

Project Metadata fields:
Group: com.example, Artifact: demo
Name: demo
Description: Demo project for Spring Boot
Package name: com.example.demo
Packaging: Jar (selected), War
Configuration: Properties (selected), YAML
Java: 25, 21, 17 (17 is selected)

Step 2:-

Create database table

CREATE DATABASE train_ticket_db;

```
CREATE TABLE passengers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    age INT NOT NULL,
    gender VARCHAR(10),
    seat_number VARCHAR(10)
);
```

The screenshot shows a PostgreSQL database interface. On the left, a tree view of the database schema is displayed, including Materialized Views, Operators, Procedures, Sequences, Tables (3), Triggers, Functions, Types, Views, Subscriptions, and various system roles and programs. The 'passengers' table is selected under the 'Tables (3)' section.

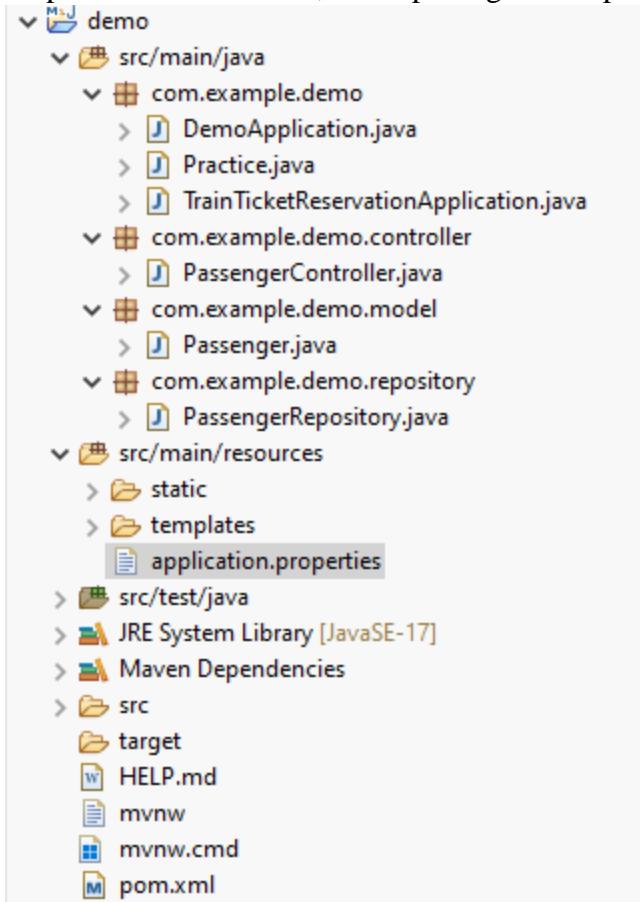
In the center, a query editor window shows the SQL command:

```
1 SELECT * FROM public.passengers
2 ORDER BY id ASC
```

Below the query editor, a data grid displays the contents of the 'passengers' table:

	id [PK] integer	name character varying (100)	age integer	gender character varying (10)	seat_number character varying (10)
1	1	sahil	21	Male	A4258959
2	3	susmit	21	Male	A42141
3	4	atharv	21	Male	A522213

Step 3 - create controller,model package in the prebuild package



- **PassengerRepository.java** - Data Access Layer
- **RowMapper** - Database Row to Object Conversion
- **@Repository** -Data Access Object (DAO)

Application.properties

```
1 server.port=8086
2
3 spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
4 spring.datasource.username=postgres
5 spring.datasource.password=password
6 spring.datasource.driver-class-name=org.postgresql.Driver
7
```

Application

workflow:- Home

Page

User lands on the main dashboard displaying an overview of recent entries and navigation options

Add Entry

User navigates to the entry form to create a new journal entry with title, content, and mood selection

Save to Database

System validates and persists the entry to MySQL database with automatic timestamp generation

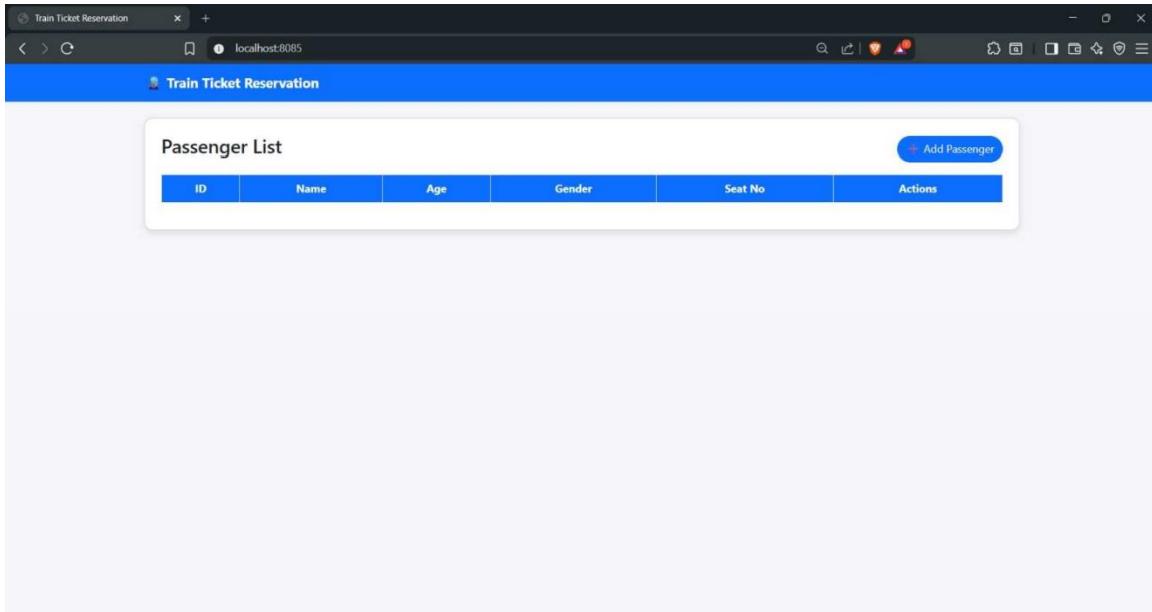
View Entries

User browses through all saved journal entries in chronological order with mood indicators

Delete Entry

User can remove unwanted entries with confirmation, triggering database deletion operation

Output-



Pic 1.0

This output screen shows the Passenger List page of the Train Ticket Reservation System. It displays passenger details such as ID, Name, Age, Gender, and Seat Number in a tabular format. The “Add Passenger” button allows users to add new passenger details for ticket reservation.

A screenshot of a web browser window titled "Train Ticket Reservation" with the URL "localhost:8085/add". The main content area is titled "Add Passenger" and contains four input fields: Name (Millie), Age (21), Gender (F), and Seat Number (50). Below the form are two buttons: "Back" and "Save".

Pic 1.1

This screen shows the **Add Passenger** form of the Train Ticket Reservation System. It allows users to enter passenger details such as **Name, Age, Gender, and Seat Number**. After filling the form, the **Save** button stores the passenger information in the system, while **Back** returns to the passenger list page.

The screenshot shows a web browser window titled "Train Ticket Reservation". The main content area is titled "Passenger List". A table displays the following data:

ID	Name	Age	Gender	Seat No	Actions
1	sahil	21	Male	A4258959	<button>Edit</button> <button>Delete</button>
3	susmit	21	Male	A42141	<button>Edit</button> <button>Delete</button>
4	atharv	21	Male	A522213	<button>Edit</button> <button>Delete</button>

At the top right of the table, there is a blue button labeled "+ Add Passenger".

Pic 1.2

This screen displays the **updated Passenger List** after adding a passenger. It shows stored details such as **ID, Name, Age, Gender, and Seat Number** retrieved from the database. The **Edit** and **Delete** buttons allow users to update or remove passenger records, supporting full CRUD operations.

The screenshot shows a web browser window titled "Train Ticket Reservation". The main content area is titled "Passenger List". A table displays the following data:

ID	Name	Age	Gender	Seat No	Actions
8	Max	23	F	20	<button>Edit</button> <button>Delete</button>
9	Willi	22	M	20	<button>Edit</button> <button>Delete</button>

At the top right of the table, there is a blue button labeled "+ Add Passenger".

Pic 1.3

This screen shows the Passenger List page with multiple records in the Train Ticket Reservation System. It displays details of all registered passengers, including Name, Age, Gender, and SeatNumber ,fetched from the database. The Add Passenger, Edit, and Delete options allow users to manage passenger information efficiently.

The screenshot shows a REST API testing interface. At the top, the URL `http://localhost:8085/` is entered into the address bar. Below the address bar, a search bar contains the text `http://localhost:8085/`. The main interface has tabs for `Docs`, `Params`, `Authorization`, `Headers (6)`, `Body`, `Scripts`, `Tests`, and `Settings`. A `Send` button is located at the top right. Below these tabs, there is a section for `Query Params` with a table. The table has columns for `Key`, `Value`, `Description`, and a `Bulk Edit` button. There is one row with the key `Key` and value `Value`, with the description `Description`. Below this, there are tabs for `Body`, `Cookies`, `Headers (6)`, and `Test Results`. The `Test Results` tab is selected, showing a green `200 OK` status, a response time of `48 ms`, and a size of `2.12 KB`. Below the status, there are buttons for `HTML`, `Preview`, and `Visualize`. The preview area shows the `Train Ticket Reservation` page. The page title is `Passenger List`. It features a table with columns: `ID`, `Name`, `Age`, `Gender`, `Seat No`, and `Actions`. One record is listed: `ID: 9, Name: Will, Age: 22, Gender: M, Seat No: 20`. To the right of the table is a blue `Add Passenger` button. Below the table are two buttons: `Edit` (yellow) and `Delete` (red). At the bottom of the interface, there are links for `Runner`, `Start Proxy`, `Cookies`, `Vault`, `Trash`, and a refresh icon.

Pic 1.4

This output shows the **GET API response** of the Train Ticket Reservation System tested using an API tool. The request returns a **200 OK** status and successfully loads the **Passenger List page**, displaying passenger data retrieved from the backend. It confirms proper integration between the **Spring Boot backend** and the frontend for fetching and displaying records.