

Project Preparation Questionnaire So, my project is on train reservation system made on spring boot, reacts js on frontend, my SQL as database and we have 7 tables 1. train, 2. user, 3. calendar, 4. passenger details and a 5. hold passenger details temp table, 6. payment details, 7. ticket details. and with 2 roles user and admin with their functionality. also have a dummy payment page I have written apis for each method also used jwt for token-based authentication. We used rest apis. in the frontend side we used axios for fetching data also used cors and toastify.

Sure, here is a more detailed and thorough answer on how to explain your project to an interviewer in a proper manner.

Introduction: I have worked on a Train Reservation System that enables users to search and book train tickets, as well as make payments online. The system uses Spring Boot for the backend, ReactJS for the frontend, and MySQL as the database.

Architecture: The Train Reservation System is designed as a RESTful API that interacts with the frontend using Axios library. The APIs are hosted on a Spring Boot application server, which communicates with the database to store and retrieve data. We used JWT-based token authentication to ensure the security of the system.

Technologies used: We used ReactJS to build a responsive and interactive user interface. For data fetching and handling, we utilized Axios, which enabled us to fetch data asynchronously and handle errors more efficiently. We used MySQL as our database to ensure data consistency and scalability.

Authentication and Authorization: We implemented token-based authentication using JWT to ensure that only authorized users could access the system. The JWT tokens were generated and validated on the server-side, and we utilized Spring Security to implement the authorization and access control.

User interface: We designed the UI to be intuitive and user-friendly, with clear call-to-actions and easy navigation. We utilized ReactJS's component-based architecture to make the UI more modular and reusable. We also implemented Toastify to display notifications to users about their booking status.

Challenges and Learning: One of the biggest challenges we faced was implementing JWT-based authentication, but we were able to overcome it by thoroughly researching and understanding the implementation process. We also learned how to work with the different technologies, and how to integrate them effectively to build a robust and scalable system.

Why did you choose spring boot only why not dot net?

The reason we chose Spring Boot for our Train Reservation System is that Spring Boot is a lightweight, flexible, and widely-used Java framework that makes it easy to build scalable and robust applications. Additionally, Spring Boot provides a rich set of features and libraries that make it easier to build and deploy web applications quickly. That being said, .NET is also a great framework for building web applications and has its own set of advantages. The choice of the framework ultimately depends on various factors, such as the development team's expertise, project requirements, and business needs.

What do you mean by scalable and robust application

Scalability refers to the ability of an application to handle an increasing number of users, data, or traffic without compromising its performance. In other words, a scalable application is one that can grow and expand seamlessly to meet the changing demands of the business. Robustness refers to the ability of an application to handle unexpected or erroneous inputs or situations without crashing or causing errors. A

robust application is one that can withstand various kinds of errors or exceptions and continue to function normally. Therefore, building a scalable and robust application is critical to ensure that the application can handle the increasing demands of the business and provide a reliable user experience.

What did you do to make your project scalable and robust? Great question! We took several steps to make our Train Reservation System scalable and robust: Use of Spring Boot framework: We chose Spring Boot because it provides a lightweight, flexible, and scalable platform for building web applications. It also has several built-in features that help with scalability, such as connection pooling and caching. Database optimization: We optimized our database schema, indexing, and query performance to ensure that the database could handle a large amount of data and queries efficiently.

Load testing: We performed load testing to simulate a high volume of traffic and users to identify and address any bottlenecks or performance issues.

Error handling: We implemented robust error handling mechanisms, such as exception handling and logging, to ensure that the application could handle unexpected or erroneous inputs or situations without crashing or causing errors.

Security measures: We implemented various security measures, such as JWT-based token authentication, to ensure that the system is secure and protected against malicious attacks.

What does spring auto configuration do

Spring Auto Configuration is a feature of the Spring Framework that automatically configures beans and other components based on their dependencies and the contents of the classpath. This feature greatly simplifies the process of setting up a Spring application, as it allows developers to rely on sensible defaults and convention over configuration.

When a Spring application starts up, the Spring container scans the classpath for any components that can be automatically configured. It then uses various techniques, such as reflection and bytecode analysis, to determine the dependencies of these components and the most appropriate way to configure them.

For example, if an application includes a datasource driver in its classpath, Spring Auto Configuration can automatically configure a datasource bean based on the properties specified in the application's configuration files.

Why did you choose axios for fetching data

We chose to use Axios for fetching data in our Train Reservation System because it is a popular and widely-used library for making HTTP requests from the client-side. Axios provides a simple and easy-to-use API for making asynchronous HTTP requests, including GET, POST, PUT, and DELETE requests, among others. It also supports features such as interceptors, which allow for easy customization and error handling. One of the main reasons we chose Axios is because it is compatible with most modern web browsers and supports a wide range of use cases, from simple data fetching to complex API interactions. Additionally, it provides a robust and efficient solution for handling network requests, with support for features such as caching and request cancellation. Another reason we chose Axios is because it is lightweight and easy to integrate with other libraries and frameworks. For example, we were able to easily integrate Axios with React JS and other popular client-side libraries.

Why didnt you use fetch() method

The Fetch API is another widely-used method for making HTTP requests from the client-side, and it is built into modern web browsers. While the Fetch API provides a similar functionality to Axios, we chose to use Axios for fetching data in our Train Reservation System for several reasons:

Consistency: Axios provides a consistent API for making HTTP requests across different browsers and platforms, which can help to simplify development and reduce potential issues. **Error Handling:** Axios provides a robust error handling mechanism, with built-in support for handling common HTTP errors, such as 404 Not Found or 500 Internal Server Error. This can help to improve the reliability and stability of the application.

Interceptors: Axios provides a powerful feature called interceptors, which allow you to intercept and modify HTTP requests and responses. This can be useful for adding custom headers, handling authentication, or implementing caching.

Flexibility: Axios provides a range of options for configuring and customizing HTTP requests, such as setting headers, timeouts, and response types. This can make it easier to integrate with different APIs and services.

While the Fetch API is a valid option for making HTTP requests from the client-side, we found that Axios provided a more comprehensive and flexible solution for our needs in the Train Reservation System.

Why have you used cors and how it is working

I have used CORS (Cross-Origin Resource Sharing) to allow the frontend ReactJS application to communicate with the backend Spring Boot application. CORS is a security feature built into web browsers that prevents websites from making requests to another domain. To enable CORS, I added the appropriate configuration in the Spring Boot backend application. This allowed the ReactJS frontend to make requests to the backend API. In simple terms, CORS is working by allowing cross-origin requests from the frontend to the backend server. Without CORS, the frontend would not be able to communicate with the backend due to security restrictions.

Describe your Role in Project As a member of the development team, my role in the project involved designing and implementing several key features, such as the user authentication system, the train booking module with managing passenger details, and the payment processing functionality. I also contributed to the overall architecture and design of the application, working closely with the project lead and other team members.

How you all done the project module distribution? We distributed the project modules among the team members based on their skill sets and expertise. We followed an Agile methodology and used a Kanban board to track the progress of each module and ensure timely delivery. We also conducted regular team meetings and stand-ups to ensure effective communication and collaboration.

AS a leader which are the project leader qualities are? Some of the key project leader qualities that I believe are essential include strong communication skills, effective decision-making abilities, a clear vision and strategy for the project, and the ability to motivate and inspire team members. A project leader should also have a deep understanding of the project requirements, technologies, and constraints, and be able to balance competing priorities and demands.

Have you used any Software Life cycle Model in your project?

We followed an Agile methodology and used a Kanban board to manage the project tasks and modules. We did not follow any specific software life cycle model such as Waterfall or Agile SCRUM, but rather adapted our approach based on the project requirements and constraints. We made sure to conduct regular testing and

quality checks throughout the development cycle to ensure that the project met the required standards and specifications.

Is it possible to convert your project from Java to .NET or vice versa?

Converting a project from Java to .NET or vice versa would require significant effort, as the two technologies have different syntax, libraries, and frameworks. It may be possible to reuse some of the business logic and algorithms, but the application would need to be re-architected and re-implemented in the new technology.

Have you used any OOPs concepts in your Project?

I have used Object-Oriented Programming (OOPs) concepts in my project. The Spring Boot framework itself is based on the principles of OOPs, and I have utilized several OOPs concepts while writing code for my project.

For example, I have created classes for each entity in my database schema such as Train, User, Calendar, Passenger Details, Payment Details, and Ticket Details, which are used to define the attributes and methods associated with these entities. I have also used inheritance, polymorphism, and encapsulation to organize my code, make it modular, and easy to maintain.

Encapsulation: It is the concept of hiding the implementation details of an object from the user and only exposing the necessary functionality through methods or interfaces. In a project, we can apply encapsulation by defining classes with private data members and public methods to access or modify those data members.

Inheritance: It is the concept of defining a new class based on an existing class, inheriting all the properties and methods of the existing class. In a project, we can apply inheritance to create a hierarchy of classes with shared properties and methods.

Polymorphism: It is the concept of using a single interface to represent multiple classes. In a project, we can apply polymorphism by defining an interface or abstract class and implementing it in multiple concrete classes.

Abstraction: It is the concept of defining the essential features of an object and ignoring the details that are not important. In a project, we can apply abstraction by defining abstract classes or interfaces that provide a high-level view of the system without getting into the implementation details.

Modularity: It is the concept of breaking down a large system into smaller, more manageable modules or components. In a project, we can apply modularity by dividing the system into logical components that can be developed and tested independently.

Have you used any APIs in your project? Yes, I have used APIs in my project. In fact, my entire project is based on REST APIs that are used to perform various operations on the database and interact with the frontend.

Here are some examples of the APIs that I have used in my project: **CRUD APIs:** I have created APIs for performing CRUD (Create, Read, Update, Delete) operations on all entities in my database schema. These APIs are used to add new records, retrieve existing records, update records, and delete records.

Authentication APIs: I have created APIs for user registration, user login, and token validation. These APIs are used to authenticate users and generate JWT tokens that are used for subsequent requests.

Payment Gateway APIs: I have integrated payment gateway APIs into my project to enable users to make online payments for booking train tickets. These APIs are used to validate user payment information and process payments securely.

Why did you choose rest. what is it? what are the other alternatives?

I chose to use REST (Representational State Transfer) in my project because it is a widely adopted and standardized approach for building web services that can be easily consumed by a variety of clients, including web browsers, mobile devices, and other applications. REST provides a simple, stateless, and scalable architecture that allows resources to be manipulated using standard HTTP methods such as GET, POST, PUT, DELETE, etc.

REST architecture has several advantages, including: Easy to implement: REST is based on the standard HTTP protocol, which makes it easy to implement and understand. It also provides a clear separation of concerns between the server and the client, which improves scalability and maintainability

Platform independent: RESTful web services can be accessed from any platform or programming language that supports HTTP requests and JSON (or XML) payloads. This makes them ideal for building APIs that can be used by a wide range of clients.

Caching: REST supports caching of resources, which can improve performance and reduce server load by reducing the number of requests to the server.

Security: RESTful web services can be secured using standard security mechanisms such as SSL/TLS, OAuth, or JWT tokens.

There are other alternatives to REST, such as SOAP (Simple Object Access Protocol) and GraphQL. SOAP is a more complex protocol that uses XML for data exchange and provides a standardized set of rules for message exchange. GraphQL is a query language for APIs that allows clients to specify exactly what data they need and receive a JSON response containing only that data. However, these alternatives have their own trade-offs and may not be suitable for all use cases.

Have you used any Payment Gateway in your Project?

To integrate a payment gateway in the project, the first step is to sign up for an account with the chosen payment gateway provider and obtain API keys. Once the API keys are obtained, the payment gateway API can be integrated into the application to enable customers to make payments. The integration involves creating a form for collecting payment information from customers and submitting the information to the payment gateway API for processing. The payment gateway API typically provides a response containing the status of the transaction, which can be used to update the ticket and payment details in the application's database. It's important to ensure that the payment gateway integration is secure and follows best practices to protect customer data and prevent fraudulent transactions.

Any errors that you face during project?

NullPointerException: This exception occurs when a variable or object reference is null and an attempt is made to invoke a method or access an attribute of that null reference.

DataAccessException: This exception is thrown when there is a failure in accessing the database, such as when the connection is lost or the SQL query is invalid.

`InvalidFormatException`: This exception is thrown when there is an error in parsing the data, such as when an invalid date or time format is entered.

`DuplicateKeyException`: This exception is thrown when trying to insert data into the database with a primary key or unique constraint that already exists in the database.

`InvalidTokenException`: This exception is thrown when there is an issue with the JWT token used for authentication and authorization.

`BadRequestException`: This exception is thrown when the request made to the server is invalid, such as when required parameters are missing.

Some potential errors or issues that could arise during the development of a project like the train reservation system you described:

Integration errors: While integrating multiple technologies and platforms, integration errors can occur due to differences in syntax and data types.

Authentication and authorization errors: Implementing token-based authentication can be challenging and errors can occur when implementing it.

Data inconsistency errors: As multiple tables are involved in the train reservation system, it is important to ensure data consistency across all tables, especially when updating or deleting data.

Input validation errors: Input validation is essential to prevent SQL injection attacks and ensure the proper functioning of the system. Failing to validate input can lead to unexpected errors and system crashes.

Draw the architecture of your project.

The architecture can be divided into three main layers: the presentation layer, the application layer, and the data layer.

Presentation layer: This layer is responsible for rendering the UI components and handling user interactions. In our example, we are using ReactJS as our front-end technology, which runs in the browser and communicates with the server through REST APIs. The presentation layer communicates with the application layer through the REST API endpoints.

Application layer: This layer contains the business logic of the application and manages the communication between the presentation layer and the data layer. In our example, we are using Spring Boot to build the back-end, which handles the REST API requests from the front-end, processes the data, and returns the results to the front-end. The application layer is responsible for the authentication, authorization, and validation of user requests. It also performs data processing and coordination between the different components of the system.

Data layer: This layer is responsible for managing the data storage and retrieval. In our example, we are using MySQL as our relational database. The data layer stores the data related to trains, users, tickets, and payments. The application layer interacts with the data layer through the Spring Data JPA library, which provides an abstraction layer over the database and simplifies the data access operations.

The following diagram illustrates the high-level architecture of the system:

+-----+ | Presentation | | (ReactJS) | +-----+ | +-----+ | Application | | (Spring Boot) | +-----+ | +-----+ | Data | | (MySQL) | +-----+ This architecture allows for a clear separation of concerns between the different layers and facilitates scalability, maintainability, and testability of the system.

Design flow diagram for your project.

The following is a high-level flow diagram for the train reservation system:

User registration and login: The user can register by providing their details like name, email, and password. After registration, the user can log in to the system using their credentials.

Search trains: The user can search for available trains by providing the source and destination stations, travel date, and the number of passengers. The system will return a list of available trains with their details like train number, name, departure and arrival time, and fare.

Select train and book tickets: The user can select a train from the list and book tickets for the desired number of passengers. The system will ask for passenger details like name, age, gender, and seat preference. The user can then make the payment for the tickets using a preferred payment method.

Ticket confirmation and cancellation: After successful payment, the system will generate a ticket with the details of the journey, passengers, and seat numbers. The user can view and print the ticket. If the user wants to cancel the ticket, they can do so by providing the ticket number and the system will refund the amount based on the cancellation policy.

The following is a basic flow diagram that illustrates the above steps:

+-----+ | User Registration | +-----+ | | +-----+ | User Authentication | +-----+ | | +-----+ | Search Trains | +-----+ | | +-----+ | Select Train and | | Book Tickets | +-----+ | | +-----+ | Payment Gateway | +-----+ | | +-----+ | Ticket Confirmation | and Cancellation | +-----+ This flow diagram represents a simplified version of the train reservation system and can be expanded and customized based on the specific requirements of the project.

Which UML Diagrams you have designed and why? In the case of a train reservation system, some of the UML diagrams that could be designed are:

Use Case Diagram: A use case diagram provides a high-level view of the functionality provided by the system and the actors who interact with the system. This diagram can help in identifying the various use cases and actors involved in the train reservation system.

Activity Diagram: An activity diagram is a behavior diagram that shows the flow of activities or processes in a system. This diagram can help in understanding the various processes involved in the train reservation system, such as ticket booking, payment processing, and cancellation.

Architectural Diagram

Explain the ER diagram of your project.

The ER (Entity-Relationship) diagram of the train reservation system would illustrate the relationships between the various entities in the system, such as users, trains, bookings, payments, etc. Here's a high-level overview

of the ER diagram for the system:

The main entities in the system are Users, Trains, Bookings, Payments, and Tickets. The Users entity stores information about the users of the system, such as their name, email, password, and contact details. The Trains entity stores information about the various trains that are available for booking, such as their name, number, route, and schedule. The Bookings entity stores information about the bookings made by users, such as the train number, passenger details, and payment status. The Payments entity stores information about the payments made by users, such as the payment amount, transaction ID, and status. The Tickets entity stores information about the tickets issued to users, such as the booking ID, passenger details, and seat number.

Explain Tables with its relationship.

The train reservation system has several tables, each storing different types of data. Here is an overview of the tables and their relationships:

Train Table: This table stores information about all the trains available for booking, such as the train name, train number, source, destination, departure time, arrival time, and fare.

User Table: This table stores information about all the users of the system, such as their name, email address, password, and contact details.

Calendar Table: This table stores information about the availability of trains on different dates.

Passenger Details Table: This table stores information about all the passengers who have booked a ticket, such as their name, age, gender, and seat number.

Hold Passenger Details Table: This table stores temporary information about passengers who have started the booking process but have not yet completed payment. Once payment is confirmed, the data from this table is moved to the Passenger Details Table.

Payment Details Table: This table stores information about all the payments made by users, such as the payment ID, payment amount, payment date, and payment status.

Ticket Details Table: This table stores information about all the tickets issued, such as the ticket ID, passenger details, train details, payment details, and status.

The relationships between these tables can be defined as follows:

The Train Table has a one-to-many relationship with the Calendar Table, as there can be multiple availability entries for a single train.

The User Table has a one-to-many relationship with the Ticket Details Table, as a user can make multiple bookings and each booking will have a separate ticket issued.

The Passenger Details Table has a many-to-one relationship with the Ticket Details Table, as multiple passengers can be associated with a single ticket.

The Payment Details Table has a one-to-one relationship with the Ticket Details Table, as there will be only one payment associated with each ticket.

The Hold Passenger Details Table has a one-to-one relationship with the Passenger Details Table, as the data from this table is moved to the Passenger Details Table once payment is confirmed.

Normalization is applicable to your project? (Yes/no). If so, up to what level.

Yes, normalization is applicable to the train reservation system project. The project involves the creation of several tables that store data related to trains, users, passengers, bookings, payments, and tickets, and each table has multiple fields that need to be properly organized to ensure data integrity and efficiency.

Normalization is a process of organizing the data in a database to reduce redundancy and dependency. It is applicable up to the third normal form (3NF) in this project. The tables have been designed with proper normalization to minimize data redundancy and ensure data consistency.

For example, the Passenger Details Table contains only passenger-related information, while the Ticket Details Table contains only ticket-related information. This ensures that there is no duplication of data, as the passenger details are linked to the ticket details through foreign keys, rather than being stored in the same table.

Similarly, the Payment Details Table contains only payment-related information, and is linked to the Ticket Details Table through a foreign key. This ensures that payment details are not repeated in multiple tables, which would have led to data redundancy and inconsistency.

Overall, normalization helps to ensure that the database is well-structured, efficient, and easy to maintain, which is critical for the successful functioning of the train reservation system.

Describe database connection of your project?

The train reservation system project uses MySQL as the database management system, and the connection to the database is established using the JDBC API in Java.

The database connection is typically established in the backend code of the project, using a connection string that contains the database URL, the username, and the password. The connection string is then passed to the JDBC driver, which manages the connection to the database.

In the train reservation system project, the connection to the database is established in the Spring Boot backend code. The application.properties file in the backend code contains the database connection settings, including the database URL, username, and password.

For example, the following is a sample database connection string for the train reservation system project:

```
spring.datasource.url=jdbc:mysql://localhost:3306/train_reservation_system
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

This configuration sets up a connection to a MySQL database running on the local machine on port 3306, with the username 'root' and password '1234', and the name of the database as 'train_reservation_system'.

Once the connection is established, the backend code can perform CRUD (create, read, update, delete) operations on the database, such as inserting new data, retrieving existing data, updating data, or deleting data, as required by the train reservation system.

Already so many applications are in market, how your application is different from market applications?

Unique Features: The application may have unique features that are not available in other applications. These features can provide additional value to the users and can make the application stand out from the competition.

User Experience: The user experience of the application can be superior to that of other applications. The application can be designed to be intuitive, easy-to-use, and visually appealing. This can lead to better user engagement and increased user satisfaction.

Performance: The application can have better performance than other applications in terms of speed, scalability, and reliability. This can lead to faster response times, reduced downtime, and better user experience.

Security: The application can have better security features than other applications. This can include measures such as encryption, access control, and authentication. Better security can lead to increased user trust and reduced risk of data breaches.

Customization: The application can provide a high degree of customization options to the users. This can allow users to tailor the application to their specific needs and preferences.

Integration: The application can integrate with other applications and services to provide a seamless user experience. This can include integration with social media platforms, payment gateways, and other third-party services.

Overall, the key to differentiating an application from other applications in the market is to provide additional value to the users in a way that is not currently being provided by the competition.

What all concepts of PG-DAC courses are applied to your Project?

Object-Oriented Programming: OOP concepts like classes, objects, inheritance, encapsulation, and polymorphism can be used to design and implement the project.

Database Management: Knowledge of database design, normalization, and SQL queries can be applied to design and manage the project database.

Web Development: Concepts of web development like HTML, CSS, JavaScript, AJAX, REST APIs, and web frameworks like Spring Boot and React can be applied to develop the project's frontend and backend.

Software Engineering: Concepts of software engineering like software development life cycle, requirements gathering, design patterns, testing, and project management can be used to plan and execute the project.

Data Structures and Algorithms: Knowledge of data structures and algorithms can be applied to optimize the performance of the project.

Security: Concepts of security like authentication, authorization, encryption, and secure coding practices can be applied to ensure the project is secure.

Cloud Computing: Concepts of cloud computing like virtualization, containerization, and cloud services can be used to deploy and host the project on the cloud.

Can you mention any test case from the project?

Test case: Book a ticket for a specific train and verify that the seat is reserved and the ticket is generated correctly.

Steps:

Login to the system using valid credentials. Search for the desired train by entering the source and destination stations and the travel date. Select the desired train from the search results. Select the desired seat from the available seats and click on the "Book" button. Enter the passenger details and click on the "Proceed to Payment" button. Enter the payment details and click on the "Pay" button. Verify that the ticket is generated with the correct details, such as the train name, passenger name, seat number, travel date, and fare. Verify that the seat is reserved and marked as booked in the database.

35.Explain your testing methodology with its type?

Testing Methodologies:

Waterfall Model: This is a sequential approach to software development, where testing is done at the end of the development cycle.

Agile Model: In this model, testing is done in parallel with development, and the feedback from testing is used to improve the software continuously.

DevOps: This is a software development approach that emphasizes collaboration and communication among developers, testers, and operations teams to achieve faster and more reliable software delivery.

Testing Types:

Unit Testing: This type of testing involves testing individual units or components of the software, such as functions or methods, in isolation from the rest of the software.

Integration Testing: This type of testing involves testing how different units or components of the software work together as a system.

how did you implemented authorisation and authentication in your project tell me in details I

implemented authorization and authentication in my project using JWT (JSON Web Tokens). First, let's start with Authentication: User registration and login: When a user registers or logs in, their credentials (username and password) are sent to the backend. The backend verifies the user's credentials against the database and generates a JWT token. JWT Token: The JWT token contains information about the user, such as their ID, username, and role. It is then returned to the frontend, where it is stored in local storage or cookies.

Authorization header: For each subsequent request made by the user, the frontend attaches the JWT token to the Authorization header of the HTTP request. Token Validation: The backend validates the token by checking its signature, expiration date, and whether it has been tampered with. If the token is valid, the user is authenticated and allowed to access the requested resource. Now, let's move on to Authorization: Role-based access control: In my project, each user has a role (e.g., admin, customer, or agent) that determines what actions they can perform and what resources they can access. Permission checks: For each protected resource or action, the backend checks whether the user's role allows them to access it. If the user is authorized, the request is allowed to proceed; otherwise, an error response is returned. To summarize, I implemented authentication in my project by using JWT tokens and verifying them on the backend, while authorization was implemented using role-based access control and permission checks. This approach helped me to ensure that

only authenticated and authorized users can access the resources and perform the actions they are allowed to do. 42. write me a code for jwt implementation Sure, here is an example code for JWT implementation in a Spring Boot application using the jjwt library:

First, add the jjwt dependency to your pom.xml file:

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.2</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.2</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.11.2</version>
  <scope>runtime</scope>
</dependency>
```

Next, create a class to handle the JWT generation and validation:

```
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Service;

import java.security.Key;
import java.util.Date;

@Service
public class JwtUtils {

    @Value("${jwt.secret}")
    private String jwtSecret;

    @Value("${jwt.expirationMs}")
    private int jwtExpirationMs;

    public String generateJwtToken(Authentication authentication) {

        UserDetailsImpl userDetails = (UserDetailsImpl)
authentication.getPrincipal();
```

```

        Date now = new Date();
        Date expiration = new Date(now.getTime() + jwtExpirationMs);

        Key key = Keys.hmacShaKeyFor(jwtSecret.getBytes());

        return Jwts.builder()
            .setSubject(userDetails.getUsername())
            .setIssuedAt(now)
            .setExpiration(expiration)
            .signWith(key)
            .compact();
    }

    public boolean validateJwtToken(String token) {
        try {
            Key key = Keys.hmacShaKeyFor(jwtSecret.getBytes());
            Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    public String getUsernameFromJwtToken(String token) {
        Claims claims =
            Jwts.parserBuilder().setSigningKey(jwtSecret.getBytes()).build().parseClaimsJws(token).getBody();
        return claims.getSubject();
    }
}

```

In this example code, the JwtUtils class generates a JWT token using the user's authentication credentials, sets an expiration time for the token, and signs it using the secret key. It also includes methods for validating the token and retrieving the username from the token.

so you said hold passenger table is your auxiliary table and the datta in it will be entered in the passenegr table only when payment will be done. so how it is working actually. explain with diagram and codes.

When a user selects a train and provides the passenger details, a new record is created in the hold passenger details table with the necessary details such as passenger name, age, gender, train number, seat number, etc. This record will be used to hold the reservation until the payment is made.

Here is an example SQL query to create the hold passenger details table:

```

Copy code
CREATE TABLE hold_passenger_details (
    hold_id INT NOT NULL AUTO_INCREMENT,
    train_number VARCHAR(50),
    passenger_name VARCHAR(255),

```

```
age INT,
gender VARCHAR(20),
seat_number VARCHAR(50),
PRIMARY KEY (hold_id)
);
```

Once the payment is completed, the details from the hold passenger details table are transferred to the passenger details table. Here is an example SQL query to transfer the data:

```
INSERT INTO passenger_details (train_number, passenger_name, age, gender,
seat_number)
SELECT train_number, passenger_name, age, gender, seat_number
FROM hold_passenger_details
WHERE hold_id = <hold_id>;
```

The hold_passenger_details table is then cleared for the next reservation. Here is an example SQL query to clear the table:

```
DELETE FROM hold_passenger_details WHERE hold_id = <hold_id>;
```

A user logs in to the system and searches for available trains based on the source and destination.

The system displays the available trains and the number of seats available for each train.

The user selects a train and enters the passenger details such as name, age, and gender, and clicks on "Reserve Seats".

The system checks the availability of seats and if seats are available, it creates a new record in the Hold Passenger Details table with the passenger details and the train details. The system then displays a confirmation message to the user indicating that the seats have been temporarily reserved.

The system waits for a certain amount of time (for example, 10 minutes) for the user to complete the payment process. If the payment is not completed within this time, the system deletes the record from the Hold Passenger Details table and releases the seats for booking by other users.

If the payment is completed within the specified time, the system creates a new record in the Passenger Details table with the passenger details and the train details, and deletes the corresponding record from the Hold Passenger Details table. The system then displays a confirmation message to the user indicating that the seats have been booked successfully.

Here's an example of how the Hold Passenger Details table might look:

Hold Passenger ID Train Number Passenger Name Seat Number Hold Expiry Time
 1 123 John Doe A1 2023-03-28 10:30:00
 2 456 Jane Smith B2 2023-03-28 10:35:00
 3 789 Bob Johnson C3 2023-03-28 10:40:00

The Hold Passenger Details table contains the following columns:

Hold Passenger ID: A unique identifier for each record in the table.

Train Number: The number of the train for which the seats are being reserved.

Passenger Name: The name of the passenger who is reserving the seats.

Seat Number: The seat number(s) that are being reserved by the passenger.

Hold Expiry Time: The time when the hold on the seats will expire.

Here's an example of how the Hold Passenger Details table might be implemented in SQL:

```
CREATE TABLE hold_passenger_details (  
  hold_passenger_id INT PRIMARY KEY,  
  train_number INT,  
  passenger_name VARCHAR(255),  
  seat_number VARCHAR(10),  
  hold_expiry_time DATETIME  
);
```

This creates a new table called "hold_passenger_details" with columns for the hold passenger ID, train number, passenger name, seat number, and hold expiry time. The hold passenger ID column is set as the primary key for the table to ensure that each record is unique.

How the details are transferring from this auxiliary table to permanent table In the train reservation system, the details from the Hold Passenger Details table are transferred to the Passenger Details table only when the payment is completed successfully by the user. Here's how the process might work:

When the user completes the payment process, the system verifies the payment and confirms that it has been successful.

The system retrieves the corresponding record from the Hold Passenger Details table, based on the hold passenger ID.

The system then creates a new record in the Passenger Details table with the details from the Hold Passenger Details table, such as the passenger name, train number, seat number, and other relevant information.

After creating the new record in the Passenger Details table, the system deletes the corresponding record from the Hold Passenger Details table to free up the temporarily held seats.

Here's an example of how the transfer of details from the Hold Passenger Details table to the Passenger Details table might be implemented in SQL:

Copy code

```
-- Insert a new record into the Passenger Details table  
INSERT INTO passenger_details (train_number, passenger_name, seat_number, ...)  
SELECT train_number, passenger_name, seat_number, ...  
FROM hold_passenger_details  
WHERE hold_passenger_id = <hold_passenger_id>;  
  
-- Delete the corresponding record from the Hold Passenger Details table  
DELETE FROM hold_passenger_details WHERE hold_passenger_id = <hold_passenger_id>;
```

This SQL code selects the relevant details from the Hold Passenger Details table based on the hold passenger ID, and inserts them into the Passenger Details table to create a new record. Once the new record has been inserted, the corresponding record is deleted from the Hold Passenger Details table using the same hold passenger ID.

RESPONSE ENTITY

ResponseEntity represents the whole HTTP response: status code, headers, and body. As a result, we can use it to fully configure the HTTP response.

If we want to use it, we have to return it from the endpoint; Spring takes care of the rest.

ResponseEntity is a generic type. Consequently, we can use any type as the response body:

```
@GetMapping("/hello")
ResponseEntity<String> hello() {
    return new ResponseEntity<>("Hello World!", HttpStatus.OK);
}
```

Since we specify the response status programmatically, we can return with different status codes for different scenarios:

```
@GetMapping("/age")
ResponseEntity<String> age(
    @RequestParam("yearOfBirth") int yearOfBirth) {

    if (isInFuture(yearOfBirth)) {
        return new ResponseEntity<>(
            "Year of birth cannot be in the future",
            HttpStatus.BAD_REQUEST);
    }

    return new ResponseEntity<>(
        "Your age is " + calculateAge(yearOfBirth),
        HttpStatus.OK);
}
```

Since we specify the response status programmatically, we can return with different status codes for different scenarios:

```
@GetMapping("/age")
ResponseEntity<String> age(
    @RequestParam("yearOfBirth") int yearOfBirth) {

    if (isInFuture(yearOfBirth)) {
        return new ResponseEntity<>(
            "Year of birth cannot be in the future",
```



```

        HttpStatus.BAD_REQUEST);
    }

    return new ResponseEntity<>(
        "Your age is " + calculateAge(yearOfBirth),
        HttpStatus.OK);
}

```

WRITE ME SOME APIS FOR USER LIKE SEARCH TRAIN OR ADD PASSENGER IN SPRING BOOT Sure, here are some example APIs for user functionalities in Spring Boot:

Search trains API:

Copy code

```

@RequestMapping(value = "/trains", method = RequestMethod.GET)
public List<Train> searchTrains(@RequestParam("source") String source,
    @RequestParam("destination") String destination,
    @RequestParam("date") @DateTimeFormat(pattern = "dd-MM-yyyy") Date date) {
    // Call a service method to search for trains based on the source, destination,
    and date
    List<Train> trains = trainService.searchTrains(source, destination, date);
    return trains;
}

```

Add passenger API:

```

@RequestMapping(value = "/passengers", method = RequestMethod.POST)
public Passenger addPassenger(@RequestBody Passenger passenger) {
    // Call a service method to add the passenger to the database
    Passenger savedPassenger = passengerService.addPassenger(passenger);
    return savedPassenger;
}

```

In the first API, the user can search for trains based on the source, destination, and date. The API takes in these parameters as query parameters and returns a list of trains that match the search criteria.

In the second API, the user can add a passenger by sending a POST request with the passenger details in the request body. The API takes in the passenger object as a request body and returns the saved passenger object with an ID assigned by the database. Note that you will need to define the Passenger class with appropriate fields and getters/setters for this API to work.

WHAT DOES REQUEST PARAM DO In Spring Boot, @RequestParam is an annotation that is used to bind a request parameter to a method argument. It allows you to retrieve a query parameter or form parameter from the request URL and pass it to the controller method as an argument.

For example, consider the following controller method:

Copy code

```
@RequestMapping(value = "/greeting", method = RequestMethod.GET)
public String greeting(@RequestParam("name") String name) {
    return "Hello " + name + "!";
}
```

Here, the `@RequestParam` annotation is used to bind the value of the "name" query parameter in the request URL to the name method argument. When a GET request is sent to `/greeting?name=John`, the value "John" will be passed to the `greeting()` method as the name parameter.

You can also specify optional parameters by setting the required attribute of the `@RequestParam` annotation to false, like this:

Copy code

```
@RequestMapping(value = "/greeting", method = RequestMethod.GET)
public String greeting(@RequestParam(value = "name", required = false) String
name) {
    if (name != null) {
        return "Hello " + name + "!";
    } else {
        return "Hello!";
    }
}
```

Here, the name parameter is optional, and if it is not present in the request URL, the name argument will be null. The required attribute defaults to true, so if you don't set it to false, Spring Boot will throw a `MissingServletRequestParameterException` if the parameter is not present in the request.

CONTROLLERS

user login

```
@Autowired
private UserServiceImpl userService;

@PostMapping("/user/login")
public ResponseEntity<?> signInUser(@RequestBody CredentialsDTO cred) {
    User user = userService.findUserByEmailAndPassword(cred.getEmail(),
cred.getPassword());
    if (user == null) {
        return Response.error("User not found");
    } else if (user.getRole() != Role.USER) {
        return Response.error("User not allowed to sign in");
    } else {
        return Response.success(user);
    }
}
```

SEARCH TRAIN

```

@GetMapping("/{source}/{destination}/{date}")
public ResponseEntity<?> searchTrain(@PathVariable("source") String source,
    @PathVariable("destination") String destination, @PathVariable("date")
Date date) {
    List<Object[]> train = trainService.searchTrain(source, destination,
date);
    if (!train.isEmpty())
        return Response.success(train);
    return Response.error("train not found");
}

```

```

@PostMapping("/signup")
public ResponseEntity<?> SaveUser(@RequestBody User user) {
    User result = userService.saveUser(user);
    return Response.success(result);
}

@DeleteMapping("/deleteUser/{id}")
public ResponseEntity<?> DeleteUser(@PathVariable("id") int id) {
    int result = userService.DeleteUser(id);

    return Response.success(result);
}

@PostMapping("/AddPassenger")
public ResponseEntity<?> AddPassenger(@RequestBody Passenger psgr) {
    Passenger pas = userService.AddPassenger(psgr);
    return Response.success(pas);
}
....

```

****SEARCHTRAIN In Dao****

```

....java

```

@Repository

```

public interface TrainDao extends JpaRepository<Train, Integer> {
    @Query(value = " SELECT t.TrainNo, t.TrainName, t.Source,
t.ArrivalTime,t.Destination,t.DepartureTime, t.TotalSeats, t.Fair,
c.AvailableSeats\r\n"
        + "      From Train  t inner join Calender c ON t.TrainNo=c.TrainNo"
        + "      WHERE t.Source=?1 AND t.Destination=?2 AND c.TrainDate=?3 ",
nativeQuery = true)
    List<Object[]> SearchTrain(String source, String Destination, Date date);
}
....

```

TESTCASE

```
```JAVA
@Test
void testauthenticateadmin() {
 User admin=adminService.findAdminByEmailAndPassword("y@gmail.com", "1234");
 System.out.println("Admin="+admin);
 //assertThat(list).isNotEmpty();
}
}
```
```

****Write me a single component for user login****

```
```javascript

import { useState } from "react";
import axios from "axios";
import { toast } from "react-toastify";

const Login = () => {
 const [username, setUsername] = useState("");
 const [password, setPassword] = useState("");

 const handleSubmit = async (e) => {
 e.preventDefault();
 try {
 const response = await axios.post("http://example.com/api/login", {
 username,
 password,
 });
 const token = response.data.token;
 localStorage.setItem("token", token);
 toast.success("Login successful!");
 } catch (error) {
 console.error(error);
 toast.error("Login failed.");
 }
 };

 return (
 <form onSubmit={handleSubmit}>
 <label>

 </label>
 export default login;
 </form>
);
};

```
```