**BLA BLA PRO MAX Spring Boot Guide**

### **1. Project Initialization**

* Use **Spring Initializr** to generate a basic project:
  + Go to [Spring Initializr](https://start.spring.io/).
  + Select:
    - **Project**: Maven or Gradle (based on preference).
    - **Language**: Java.
    - **Spring Boot Version**: Latest stable version.
  + Add dependencies:
    - **Spring Web**: For building REST APIs.
    - **Spring Data JPA**: For database interaction.
    - **Spring Security**: For authentication and authorization.
    - **Spring Boot DevTools**: For development ease.
    - **MySQL Driver** (or other databases you plan to use).
    - **Lombok** (optional): To reduce boilerplate code.
  + Generate and download the project.

### **2. Directory Structure**

A typical directory structure:

src/

main/

java/com/example/ridesharing/

controller/

service/

repository/

model/

dto/

resources/

application.properties

test/

java/com/example/ridesharing/

### **3. Configure application.properties**

Set up database connection and other configurations:

spring.datasource.url=jdbc:mysql://localhost:3306/ridesharing

spring.datasource.username=your\_username

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

### **4. Create Database Models**

Define entities such as User, Ride, Booking, and Vehicle.

Example: User entity

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String phone;

// Getters and Setters

}

### **5. Create Repositories**

For database operations:

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByEmail(String email);

}

### **6. Create Services**

Handle business logic.

Example: UserService

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User registerUser(User user) {

return userRepository.save(user);

}

}

### **7. Create Controllers**

Handle API endpoints.

Example: UserController

@RestController

@RequestMapping("/api/users")

public class UserController {

@Autowired

private UserService userService;

@PostMapping("/register")

public ResponseEntity<User> registerUser(@RequestBody User user) {

return ResponseEntity.ok(userService.registerUser(user));

}

}

### **8. Add Authentication and Authorization**

* Use Spring Security to secure APIs.
* Implement JWT for token-based authentication.

Example dependencies:  
 <dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

### **9. Add Core Features**

* **Ride Management**:
  + Create a Ride entity with details like origin, destination, date, price, and available seats.
  + Develop APIs for creating and fetching rides.
* **Booking Management**:
  + Create a Booking entity linking users to rides.
  + Develop APIs for booking a ride and checking ride availability.
* **Search**:
  + Add filtering logic in RideRepository to find rides by origin, destination, and date.

### **10. Test APIs**

Use tools like **Postman** or **Swagger** to test the endpoints.

### **11. Frontend Integration**

Integrate with a front-end framework (e.g., React or Angular) or expose APIs for mobile apps.

### **12. Deployment**

* Use Docker to containerize the application.
* Deploy on a cloud platform like AWS, Azure, or GCP.

Let me know if you'd like detailed code for any part of this setup!