**Đầu tiên ta dùng Spring Initializr để tạo qua 1 project cơ bản với các** Dependencies:

Spring Web → Bắt buộc nếu viết API.

Spring Data JPA → Nếu muốn làm việc với database bằng Hibernate.

MySQL Driver → Nếu database là MySQL.

Spring Boot DevTools → Giúp auto-restart khi code thay đổi.

Spring Security → Nếu cần bảo mật API.

Spring Boot Actuator → Nếu cần giám sát API.

JavaMailSender → Gửi OTP qua email

**Tiếp theo, chuẩn bị môi trường:**

Trước tiên, bạn cần môi trường để lập trình API. Nếu dùng Spring Boot (Java), cần:

JDK (Java Development Kit). Nên dùng JDK 17 hoặc 21 để tránh lỗi.

Maven/Gradle (để quản lý thư viện)

Spring Boot (cấu hình project nhanh với spring-boot-starter-web)

Postman (để test API)

Database (MySQL)

**Các bước mở project Spring Boot trong VS Code**

Mở VS Code

Chọn File → Open Folder

Chọn thư mục chứa project Spring Boot

**Trên VS Code, nhấn Ctrl + Shift + X (mở Extensions) và cài đặt:**

Extension Pack for Java (Gói cài đặt đầy đủ cho Java)

Spring Boot Extension Pack (Hỗ trợ Spring Boot)

Maven for Java (Quản lý Maven)

Debugger for Java (Hỗ trợ Debug)

Lombok

**Tiếp theo, vào file application.properties để thêm các thông tin sau:**

# =========================

# Thông tin ứng dụng

# =========================

spring.application.name=Tên ứng dụng

# =========================

# Cấu hình Server

# =========================

server.port=8080

# =========================

# Cấu hình kết nối Database (MySQL)

# =========================

spring.datasource.url=jdbc:mysql://localhost:3306/<Tên database>?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=Asia/Ho\_Chi\_Minh

spring.datasource.username=root

spring.datasource.password=Mật khẩu

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# =========================

# Cấu hình JPA (Hibernate)

# =========================

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# =========================

# Cấu hình gửi mail (Gmail SMTP)

# =========================

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=abc@gmail.com

spring.mail.password=mật khẩu (16 ký tự)

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

spring.mail.properties.mail.smtp.starttls.required=true

spring.mail.properties.mail.smtp.connectiontimeout=5000

spring.mail.properties.mail.smtp.timeout=5000

spring.mail.properties.mail.smtp.writetimeout=5000

**Sau khi đã hoàn tất các bước trên, vào file, mở terminal của file pom.xml (vào file pom, ấn Ctrl + ~) sau đó chạy lệnh mvn clean install, nếu không có lỗi thì đến các bước kế tiếp.**

|  |  |
| --- | --- |
| **Model (Entity)** | |
| Account | accountId (PK)  username  email  password  isActive  OneToOne với User |
| User (các field trong table user tùy thuộc vào dự án đang làm) | userId (PK)  name  Phone  Address  Birthdate  gender  role (default: "customer")  createdAt  sumPoints  OneToOne với Account |
| Otp | id (PK)  email  otpCode  createdAt  expiresAt (1 phút)  isUsed |

|  |  |
| --- | --- |
| **Repository** | |
| AccountRepository | existsByUsername, existsByEmail,  findByUsername, findByEmail |
| UserRepository |  |
| OtpRepository | findByEmailAndOtpCodeAndIsUsedFalse  findTopByEmailOrderByCreatedAtDesc (dùng chống spam) |

|  |  |
| --- | --- |
| **Service** | |
| register(...) | Tạo Account (chưa active), tạo OTP, gửi mail  Kiểm tra spam OTP: chỉ được gửi 1 lần mỗi 1 phút |
| verifyOtp(...) | Kiểm tra mã OTP, active account, tạo User |
| login(...) | Đăng nhập nếu tài khoản active, kiểm tra mật khẩu |
| forgotPassword(...) | Gửi lại OTP để đặt lại mật khẩu  Chặn spam OTP trong 1 phút |
| resetPassword(...) | Xác minh OTP rồi cập nhật password cho Account |
| sendOtpEmail(...) | Gửi OTP thực qua Gmail (SMTP) |

|  |  |
| --- | --- |
| **Controller** | |
| POST | /api/auth/register |
| POST | /api/auth/verify-otp |
| POST | /api/auth/login |
| POST | /api/auth/forgot-password |
| POST | /api/auth/reset-password |

Đã tích hợp:

- Mã hóa password khi lưu vào database, khi login sẽ lấy password nhập từ textbox đi mã hóa để so sánh với pass trong db.

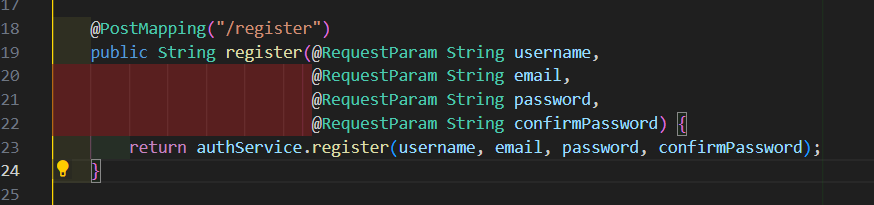
- Đã tích hợp JWT để xác thực người dùng: sau khi login sẽ gửi về một token được mã hóa sh256 dùng để xác minh người dùng bất kỳ hành động lúc sau.

- Đã format lại cấu trúc của mẫu email được gửi.

- Đã phân quyền role - user/admin với SecurityConfig.java, phân quyền bằng cách kiểm tra các yêu cầu gọi api đến url có chứa /admin sẽ check token có trả về role admin hay không.

- Đã Custom Response Format giúp không return về chuỗi kết quả mà có cả succes là true hoặc false với hành động gọi api, message kết quả và data nếu là list car, hay cụ thể là token khi login.

- Đã Tích hợp @RequestBody + DTO + @Valid cho tất cả các API còn lại trong AuthController

Cụ thể: trước đó dùng 

Việc này gây khó khăn khi phải thêm quá nhiều tham số và chưa check trong request, do đó cần tạo dto để valid request và truyền dữ liệu tốt hơn, dễ bảo trì và nâng cấp, nếu thêm field chỉ cần thêm vào trong request của api đó, không cần vào code thêm.



Hiện tại mới chỉ có package Request, chưa có response

Nên có dto.response khi:

- /api/user/profile → trả về thông tin user: name, email, phone → cần UserProfileResponse

- /api/cars/list → trả về danh sách xe → cần CarListResponse

- /api/bookings/history → trả danh sách lịch sử đặt xe → cần BookingHistoryResponse

➡ Đó là lúc tách riêng package dto.response cho đẹp và rõ ràng.