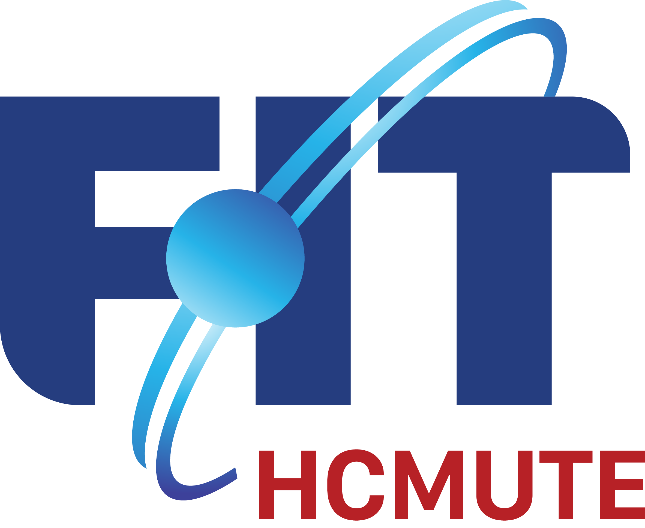


**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KỲ**

**MÔN LẬP TRÌNH DI ĐỘNG**

**XÂY DỰNG ỨNG DỤNG**

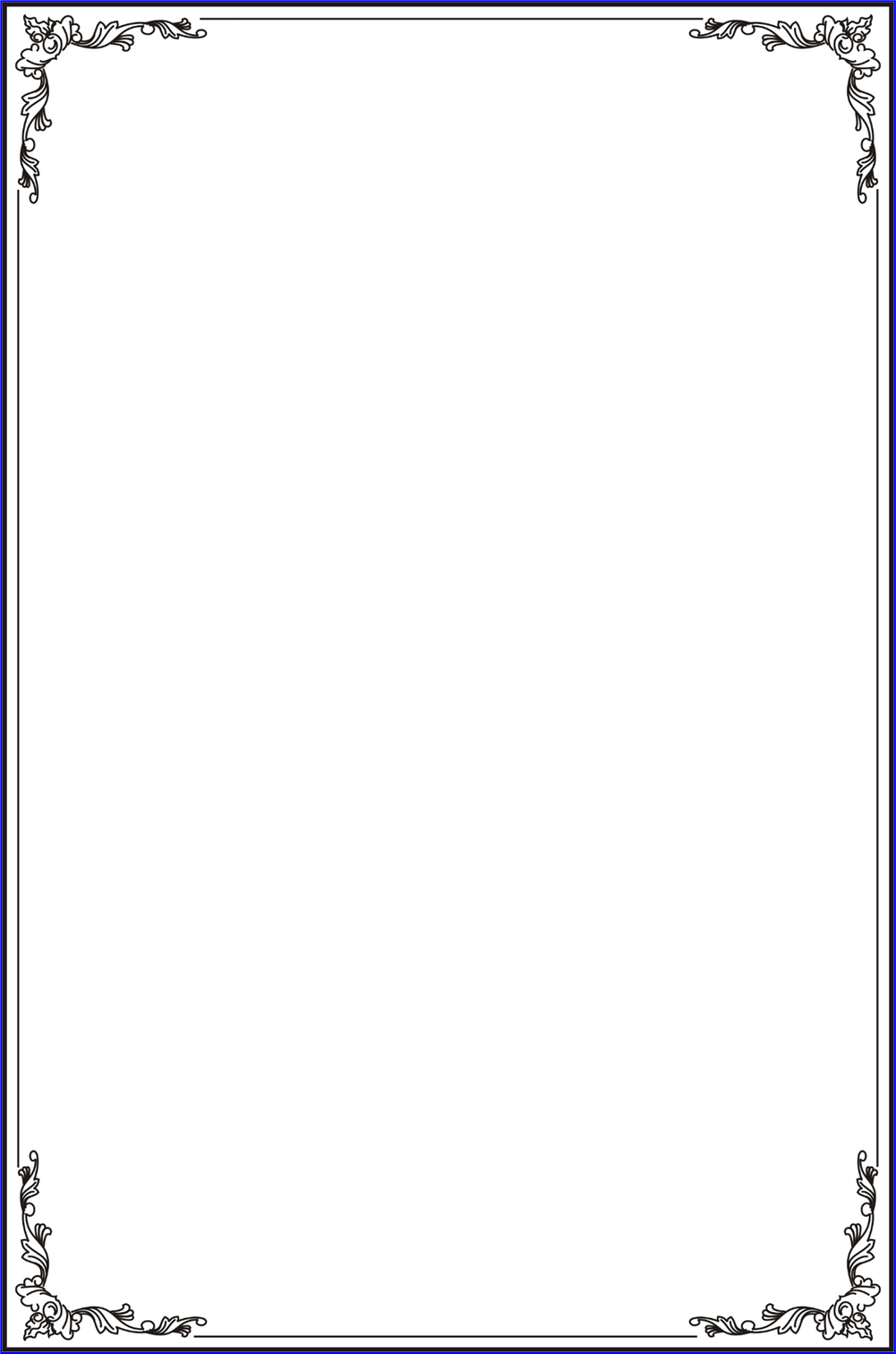
**THUÊ XE OTO**

**GIẢNG VIÊN HƯỚNG DẪN: THS. NGUYỄN HỮU TRUNG**

**SINH VIÊN THỰC HIỆN: NHÓM 13**

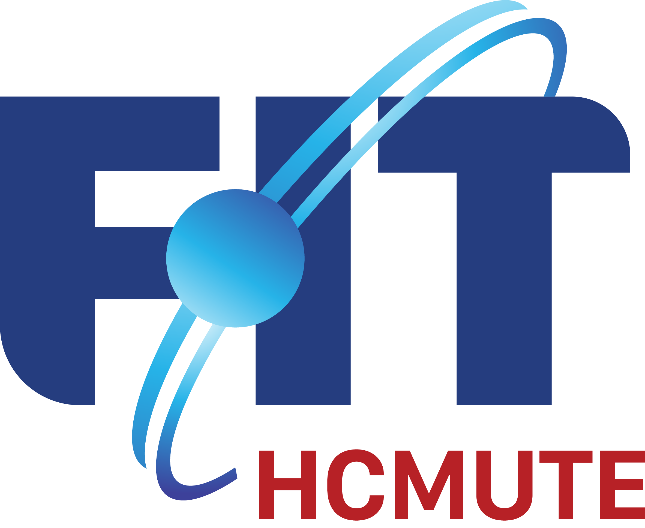
| Đỗ Phú Luân | 22110372 |
| --- | --- |
| Huỳnh Minh Mẫn | 22110377 |

**TP. Hồ Chí Minh, tháng 5 năm 2025**



**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KỲ**

**MÔN LẬP TRÌNH DI ĐỘNG**

**XÂY DỰNG ỨNG DỤNG**

**THUÊ XE OTO**

**GIẢNG VIÊN HƯỚNG DẪN: THS. NGUYỄN HỮU TRUNG**

**SINH VIÊN THỰC HIỆN: NHÓM 13**

| Đỗ Phú Luân | 22110372 |
| --- | --- |
| Huỳnh Minh Mẫn | 22110377 |

**TP. Hồ Chí Minh, tháng 5 năm 2025**

# 

[**PHẦN MỞ ĐẦU 3**](#_heading=h.lfn8orp3o46u)

[1. Đặt vấn đề 3](#_heading=h.3rl9zwi6fu96)

[2. Mục tiêu đề tài 3](#_heading=h.ja44fofgkehb)

[3. Phạm vi thực hiện 3](#_heading=h.206i9mcwvs2b)

[4. Ý nghĩa thực tiễn 3](#_heading=h.suztjxwxw6g1)

[**PHẦN NỘI DUNG 5**](#_heading=h.olq25y99tjy2)

[1. Bảng phân công nhiệm vụ 5](#_heading=h.m8awx675zccf)

[2. Lược đồ Use case 6](#_heading=h.juwgonubfwke)

[3. Cấu trúc Project 6](#_heading=h.babbswcqdssv)

[3.1. Frontend (Ứng dụng di động) 6](#_heading=h.utv0artb11cx)

[3.1.1. Giới thiệu chung 6](#_heading=h.3exk9m6bz3q2)

[3.1.2. Cấu trúc 7](#_heading=h.l9qiqyh65irt)

[3.2. Backend (API - ThueXeAPI) 7](#_heading=h.eqjoz4gvx0a8)

[3.2.1. Giới thiệu chung 7](#_heading=h.7btj5z8cd29i)

[3.2.2. Cấu trúc project 7](#_heading=h.d51ai3j4flll)

[4. Mô tả các chức năng 8](#_heading=h.opj0bznbujwz)

[4.1. Đăng ký, đăng nhập và quên mật khẩu cho người dùng tích hợp xác nhận OTP qua email 8](#_heading=h.pmfqd6w9r0zi)

[4.2. Nhắn tin tích hợp WebSocket thời gian thực 9](#_heading=h.225mrvl65098)

[4.3. Đăng ký xe, xem các xe đã đăng ký, chỉnh sửa thông tin xe tích hợp Cloudinary 11](#_heading=h.iqbrpcni9l34)

[4.4. Quản lý tài khoản và thông tin cá nhân 12](#_heading=h.pp8aaqfnute0)

[4.5. Tìm kiếm, lọc và xem danh sách xe 12](#_heading=h.v329z9tps9zq)

[4.6. Xem chi tiết thông tin xe, xem đánh giá xe 13](#_heading=h.84uzeug5meff)

[4.7. Tạo đơn đặt xe, xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe 15](#_heading=h.5pbidmy3cva2)

[4.8. Đánh giá dịch vụ thuê xe 16](#_heading=h.xl13tgs96tnz)

[5. Giao diện các chức năng 18](#_heading=h.xk9i1dug7ly9)

[5.1. Đăng ký, đăng nhập và quên mật khẩu cho người dùng tích hợp xác nhận OTP qua email 18](#_heading=h.hqcl1q85fhy5)

[5.2. Nhắn tin tích hợp WebSocket thời gian thực 20](#_heading=h.y7y9sq9ithum)

[5.3. Đăng ký xe, xem các xe đã đăng ký, chỉnh sửa thông tin xe tích hợp Cloudinary 21](#_heading=h.nl39xm3q2jxc)

[5.4. Quản lý tài khoản và thông tin cá nhân 23](#_heading=h.3rflcmtpy3rk)

[5.5. Tìm kiếm, lọc và xem danh sách xe 24](#_heading=h.gx84uanml2z4)

[5.6. Xem chi tiết thông tin xe, xem đánh giá xe 25](#_heading=h.m85z2arymz3v)

[5.7. Tạo đơn đặt xe, xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe 27](#_heading=h.7snks3u3z9p2)

[5.8. Đánh giá dịch vụ thuê xe, chỉnh sửa, xóa đánh giá. 29](#_heading=h.125b1c2l5a2i)

[6. Mô tả code của chức năng 30](#_heading=h.vz1sjhdkd8da)

[6.1. Đăng ký, đăng nhập và quên mật khẩu cho người dùng tích hợp xác nhận OTP qua email 30](#_heading=h.dkyb77oqahm9)

[6.1.1. Đăng ký tài khoản 30](#_heading=h.oi3swvjmixvm)

[6.1.2. Xác minh OTP 35](#_heading=h.da5sd1pxefza)

[6.1.3. Đăng nhập 39](#_heading=h.oih8emj7kcp0)

[6.1.4. Quên mật khẩu và đặt lại mật khẩu 42](#_heading=h.fhl8apef73ol)

[6.1.5. Kiểm tra JWT Token: 49](#_heading=h.bb561rfjz6ps)

[6.2. Nhắn tin tích hợp WebSocket thời gian thực 51](#_heading=h.e8wxndb3nxcj)

[6.2.1. Tải danh sách hội thoại: 51](#_heading=h.7quc1mqnk15s)

[6.2.2. Xem chi tiết hội thoại và tải tin nhắn 55](#_heading=h.rug7w2usir16)

[6.2.3. Gửi tin nhắn 58](#_heading=h.4s63yaw3va62)

[6.2.4. WebSocket - Gửi và nhận tin nhắn 63](#_heading=h.rr7z4ujob8qm)

[6.3. Đăng ký xe, xem các xe đã đăng ký, chỉnh sửa thông tin xe tích hợp Cloudinary 66](#_heading=h.orss692s69kw)

[6.3.1. Đăng ký xe 66](#_heading=h.syw90p4je5lz)

[6.3.2. Xem các xe đã đăng ký 72](#_heading=h.ng32xq2bdufm)

[6.3.3. Chỉnh sửa thông tin xe 76](#_heading=h.13ad70fs0tlw)

[6.4. Quản lý tài khoản và thông tin cá nhân 82](#_heading=h.7xll5ts8docu)

[6.4.1. Xem thông tin tài khoản 82](#_heading=h.m2iz47vs1gnw)

[6.4.2. Chỉnh sửa thông tin cá nhân 86](#_heading=h.clni2a6rg0o)

[6.5. Tìm kiếm, lọc và xem danh sách xe 91](#_heading=h.rp92d4wsrp41)

[6.5.1. Tìm kiếm xe 91](#_heading=h.4c6omhkwt71f)

[6.5.2. Lọc xe theo yêu cầu 97](#_heading=h.kxjmv0cedx6r)

[6.6. Xem chi tiết thông tin xe, xem đánh giá xe 100](#_heading=h.v6dsr4icz0a0)

[6.6.1. Xem chi tiết thông tin xe 100](#_heading=h.5elnfxpxtlsk)

[6.6.2. Xem đánh giá xe 106](#_heading=h.snj6y64idkza)

[6.7. Tạo đơn đặt xe, xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe 107](#_heading=h.bylx2u42rp3j)

[6.7.1. Tạo đơn đặt xe 107](#_heading=h.hd7xpioun66c)

[6.7.2. Xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe 112](#_heading=h.xnlia3uw90ul)

[6.8. Đánh giá dịch vụ thuê xe, chỉnh sửa, xóa đánh giá. 117](#_heading=h.xx0mo4hhjopi)

[**PHẦN KẾT LUẬN 124**](#_heading=h.pqlpu5g7f9ad)

[1. Kết quả đạt được 124](#_heading=h.kxgw2z7zdc7q)

[2. Kỹ năng đạt được 125](#_heading=h.8i0i1xq16h6v)

[3. Ưu điểm 125](#_heading=h.8j0ww1lmapx4)

[4. Nhược điểm 125](#_heading=h.71hr7lg8w8xg)

[5. Hướng phát triển 125](#_heading=h.n4nmymlqdelp)

# 

# PHẦN MỞ ĐẦU

## Đặt vấn đề

Trong thời đại số hóa hiện nay, nhu cầu di chuyển bằng oto ngày càng gia tăng, đặc biệt là trong các thành phố lớn và các khu du lịch phát triển. Tuy nhiên, không phải ai cũng có đủ điều kiện sở hữu xe oto riêng. Chính vì vậy, dịch vụ thuê oto đã trở thành một giải pháp tiện lợi, tiết kiệm và linh hoạt cho người dùng.

Bên cạnh các nền tảng lớn như Grab, BeCar hay Turo, nhu cầu phát triển các ứng dụng thuê oto nhỏ, dễ sử dụng và có tính tùy biến cao vẫn rất lớn, đặc biệt đối với các doanh nghiệp nhỏ hoặc các cá nhân kinh doanh dịch vụ thuê xe. Do đó, việc xây dựng một ứng dụng thuê oto với các tính năng cơ bản, dễ sử dụng và có khả năng mở rộng là một hướng đi thiết thực, phù hợp với khả năng triển khai của sinh viên và đáp ứng đúng yêu cầu môn học.

## Mục tiêu đề tài

Đề tài “Xây dựng ứng dụng thuê xe oto” được thực hiện với mục tiêu phát triển một ứng dụng hoạt động trên nền tảng Android, cho phép người dùng:

* Đăng ký, xác thực và đăng nhập tài khoản.
* Tìm kiếm và đặt thuê oto theo các tiêu chí: loại xe, giá thuê, thời gian thuê, vị trí.
* Quản lý thông tin xe, cập nhật thông tin xe.
* Đánh giá, nhận xét và phản hồi sau khi sử dụng dịch vụ.
* Quản lý thông tin cá nhân, lịch sử đặt thuê và chi tiết thanh toán.

## Phạm vi thực hiện

Phạm vi của đề tài tập trung vào:

* Phát triển ứng dụng frontend trên nền tảng Android bằng Java và XML.
* Xây dựng backend sử dụng Spring Boot kết hợp với WebSocket để giao tiếp thời gian thực.
* Thiết kế cơ sở dữ liệu MySQL để quản lý thông tin xe, người dùng và đơn thuê.

## Ý nghĩa thực tiễn

Việc thực hiện đề tài mang lại nhiều giá trị cả về kiến thức lẫn kỹ năng thực hành.

Về mặt học thuật: Nhóm được rèn luyện khả năng tổ chức dự án, làm việc theo nhóm, phân tích hệ thống và triển khai ứng dụng thực tế.

Về mặt thực tiễn: Sản phẩm tạo ra có thể trở thành nền tảng để phát triển thêm các tính năng nâng cao như thanh toán trực tuyến, định vị GPS hoặc tích hợp API bên thứ ba. Đồng thời, ứng dụng này có thể được sử dụng làm tài liệu tham khảo cho các nhóm sinh viên khác trong các học kỳ sau.

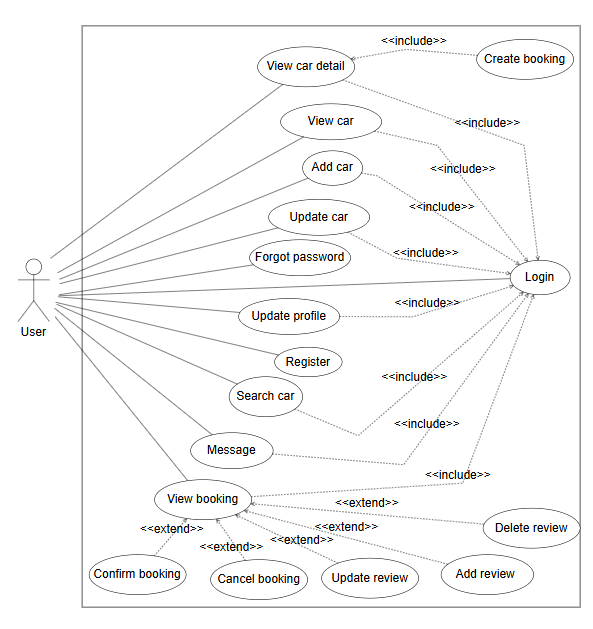
Quá trình xây dựng ứng dụng giúp nhóm hiểu rõ hơn về vòng đời phát triển phần mềm và tích lũy kinh nghiệm giải quyết vấn đề trong môi trường dự án thực tế.

# PHẦN NỘI DUNG

## Bảng phân công nhiệm vụ

| STT | Tên thành viên | Chức năng |
| --- | --- | --- |
| 1 | Huỳnh Minh Mẫn | - Đăng ký, đăng nhập và quên mật khẩu cho người dùng tích hợp xác nhận OTP qua email.  - Nhắn tin tích hợp WebSocket thời gian thực.  - Đăng ký xe, xem các xe đã đăng ký, chỉnh sửa thông tin xe tích hợp Cloudinary.  - Quản lý tài khoản và thông tin cá nhân. |
| 2 | Đỗ Phú Luân | - Tìm kiếm, lọc và xem danh sách xe  - Xem chi tiết thông tin xe, xem đánh giá xe  - Tạo đơn đặt xe, xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe  - Đánh giá đơn thuê xe. Chỉnh sửa, xóa đánh giá. |

## Lược đồ Use case



## Cấu trúc Project

## Frontend (Ứng dụng di động)

### Giới thiệu chung

Ứng dụng di động Lotom là một ứng dụng quản lý đặt xe, cho phép người dùng đăng ký, đăng nhập, xem danh sách xe, đặt xe, và theo dõi các booking của mình phát triển bằng Java trên nền tảng Android.

Ứng dụng sử dụng kiến trúc MVVM để tách biệt các lớp View, ViewModel và Model nhằm tối ưu hóa mã nguồn và tăng khả năng tái sử dụng.

### Cấu trúc

Thư mục activity: Chứa các Activity chính như LoginActivity, RegisterActivity, HomeActivity.

Thư mục fragment: Quản lý các Fragment như ProfileFragment, CarListFragment, BookingFragment.

Thư mục adapter: Adapter cho RecyclerView như CarAdapter, BookingAdapter, MessageAdapter.

Thư mục layout: Các file XML định nghĩa giao diện người dùng.

Thư mục model: Lớp dữ liệu (User.java, Car.java, Booking.java).

Thư mục repository: Xử lý dữ liệu, kết nối API và database.

Thư mục viewmodel: Cầu nối giữa View và Model, chứa các lớp ViewModel cho từng tính năng.

## **Backend (API - ThueXeAPI)**

### Giới thiệu chung

API được phát triển bằng Spring Boot và tuân thủ kiến trúc nhiều lớp (Layered Architecture).

### Cấu trúc project

Thư mục controller:

* AuthController.java: Điều khiển luồng xác thực và đăng nhập.
* UserController.java: Điều khiển các thao tác người dùng và booking.
* ChatWebSocketController.java: Xử lý kết nối WebSocket cho tính năng chat.

Thư mục service:

* AuthService.java: Xử lý logic xác thực.
* BookingService.java: Xử lý logic booking.
* CarService.java: Quản lý thông tin xe.
* MessageService.java: Xử lý tin nhắn chat.
* UserService.java: Xử lý thông tin người dùng.

Thư mục repository:

* UserRepository.java: Tương tác với bảng User.
* CarRepository.java: Tương tác với bảng Car.
* BookingRepository.java: Tương tác với bảng Booking.
* MessageRepository.java: Tương tác với bảng Message.

Thư mục dto/request:

* LoginRequest.java: Dữ liệu yêu cầu đăng nhập.
* RegisterRequest.java: Dữ liệu yêu cầu đăng ký.
* BookingRequest.java: Dữ liệu yêu cầu đặt xe.
* AddCarRequest.java: Dữ liệu yêu cầu thêm xe.

Thư mục dto/response:

* UserProfileResponse.java: Dữ liệu trả về thông tin người dùng.
* CarResponse.java: Dữ liệu trả về thông tin xe.
* BookingDetailResponse.java: Dữ liệu trả về chi tiết booking.

Thư mục security:

* SecurityConfig.java: Cấu hình bảo mật JWT.
* JwtAuthFilter.java: Kiểm tra token JWT.

application.properties: Cấu hình cơ sở dữ liệu, bảo mật, WebSocket.

pom.xml: Quản lý dependency, plugin và các cấu hình Maven.

## **Mô tả** các **chức năng**

## **Đăng ký, đăng nhập và quên mật khẩu cho người dùng tích hợp xác nhận OTP qua email**

Ứng dụng Thuê xe oto tích hợp tính năng đăng ký, đăng nhập và quên mật khẩu cho người dùng, kết hợp cơ chế xác thực OTP qua email để tăng cường tính bảo mật và đảm bảo tính xác thực cho từng tài khoản.

* Đăng ký tài khoản:

Người dùng thực hiện đăng ký bằng cách cung cấp tên tài khoản, email và mật khẩu.

Sau khi gửi yêu cầu đăng ký, hệ thống tự động gửi mã OTP gồm 6 chữ số tới email đã đăng ký.

Người dùng nhập mã OTP để xác thực email và kích hoạt tài khoản.

Nếu mã OTP không hợp lệ hoặc hết hạn, người dùng có thể yêu cầu gửi lại mã OTP mới.

* Đăng nhập:

Người dùng đăng nhập bằng tên tài khoản và mật khẩu đã đăng ký.

Hệ thống sẽ xác thực thông tin đăng nhập dựa trên dữ liệu người dùng đã lưu trữ.

Nếu thông tin đăng nhập chính xác, người dùng sẽ được chuyển đến trang chính của ứng dụng.

* Quên mật khẩu:

Khi người dùng quên mật khẩu, hệ thống cho phép họ nhấn vào mục “Quên mật khẩu”.

Người dùng cần nhập email đã đăng ký để nhận mã OTP xác nhận.

Sau khi nhập đúng mã OTP, người dùng sẽ được phép thiết lập mật khẩu mới.

* Ý nghĩa và lợi ích:

Bảo mật tài khoản: Việc tích hợp OTP giúp ngăn ngừa các hành vi truy cập trái phép và bảo vệ tài khoản người dùng.

Xác thực nhanh chóng: Cơ chế OTP qua email đảm bảo tính nhanh chóng và tiện lợi trong việc xác thực người dùng.

Tính khả dụng cao: Người dùng có thể dễ dàng khôi phục mật khẩu mà không cần sự can thiệp từ bộ phận hỗ trợ.

Trải nghiệm người dùng nâng cao: Quy trình đăng ký và đăng nhập trở nên dễ dàng và an toàn hơn nhờ OTP.

## **Nhắn tin tích hợp WebSocket thời gian thực**

Ứng dụng Thuê xe oto cung cấp tính năng nhắn tin thời gian thực giữa người thuê và chủ xe, sử dụng WebSocket để đảm bảo thông tin được truyền tải nhanh chóng, tức thời mà không cần gửi yêu cầu HTTP liên tục. Đây là một chức năng quan trọng, giúp tối ưu hóa trải nghiệm người dùng khi trao đổi thông tin trong quá trình thuê xe.

* Khởi tạo kết nối WebSocket:

Khi người dùng đăng nhập thành công, hệ thống tự động thiết lập kết nối WebSocket để duy trì phiên giao tiếp liên tục.

Mỗi người dùng sẽ được cấp một session ID duy nhất, cho phép hệ thống định danh và quản lý các phiên làm việc.

* Gửi tin nhắn:

Người dùng có thể gửi tin nhắn văn bản tới người thuê hoặc chủ xe thông qua giao diện chat.

Mỗi tin nhắn sẽ bao gồm nội dung, thời gian gửi

Khi người dùng nhấn nút gửi, tin nhắn sẽ được truyền qua WebSocket và chuyển tới người nhận ngay lập tức mà không cần tải lại trang.

* Nhận tin nhắn:

Khi nhận được tin nhắn mới, hệ thống sẽ tự động cập nhật giao diện chat.

Người dùng sẽ thấy tin nhắn ngay lập tức trong giao diện chat, đảm bảo tính tức thời và không bỏ lỡ thông tin quan trọng.

* Lưu trữ và quản lý tin nhắn:

Tất cả các tin nhắn sẽ được lưu trữ vào cơ sở dữ liệu MySQL, kèm theo các thông tin: người gửi, người nhận, nội dung, thời gian gửi và trạng thái.

Khi người dùng đăng xuất và đăng nhập lại, hệ thống sẽ tự động tải lại toàn bộ lịch sử tin nhắn từ cơ sở dữ liệu để đảm bảo sự liên tục trong trải nghiệm chat.

* Ý nghĩa và lợi ích:

Tính năng nhắn tin thời gian thực qua WebSocket giúp cải thiện hiệu suất giao tiếp giữa người thuê và chủ xe, đặc biệt trong các trường hợp cần phản hồi ngay lập tức.

Cơ chế WebSocket giảm tải cho server khi không cần gửi yêu cầu HTTP liên tục, đồng thời tăng tốc độ truyền tải dữ liệu.

Đảm bảo tính nhất quán và liền mạch trong trải nghiệm người dùng khi trao đổi thông tin trong thời gian thực.

## **Đăng ký xe, xem các xe đã đăng ký, chỉnh sửa thông tin xe tích hợp Cloudinary**

Ứng dụng Thuê xe oto cung cấp chức năng quản lý xe dành cho chủ xe, cho phép đăng ký xe mới, xem danh sách xe đã đăng ký và chỉnh sửa thông tin xe. Đặc biệt, ứng dụng tích hợp Cloudinary để lưu trữ hình ảnh xe trên cloud, giúp tối ưu hóa không gian lưu trữ và tăng cường khả năng quản lý dữ liệu hình ảnh.

* Đăng ký xe mới:

Người dùng cung cấp các thông tin chi tiết về xe, bao gồm: tên xe, hãng xe, loại nhiên liệu, số ghế, giá thuê, vị trí và mô tả chi tiết.

Hệ thống yêu cầu người dùng tải lên ít nhất một hình ảnh xe. Hình ảnh này sẽ được tải lên Cloudinary và trả về URL lưu trữ.

Sau khi hoàn tất thông tin, người dùng nhấn nút “Đăng ký xe” để lưu thông tin xe vào cơ sở dữ liệu cùng với URL hình ảnh từ Cloudinary.

* Xem danh sách xe đã đăng ký:

Người dùng có thể truy cập mục “Danh sách xe” để xem toàn bộ các xe đã đăng ký.

Danh sách xe hiển thị các thông tin cơ bản bao gồm: tên xe, hãng xe, giá thuê, trạng thái xe và hình ảnh xe (lấy từ Cloudinary).

Người dùng có thể nhấn vào từng xe để xem chi tiết thông tin và hình ảnh.

* Chỉnh sửa thông tin xe:

Người dùng có thể chọn xe cần chỉnh sửa thông tin.

Tại giao diện chỉnh sửa, người dùng có thể cập nhật các thông tin như giá thuê, mô tả chi tiết, hình ảnh xe.

Khi cập nhật hình ảnh, hệ thống sẽ tự động thay thế ảnh cũ bằng ảnh mới trên Cloudinary và cập nhật URL mới vào cơ sở dữ liệu.

* Cloudinary - Lưu trữ hình ảnh trên cloud:

Mọi hình ảnh xe khi được tải lên sẽ được lưu trữ tại Cloudinary, giúp giảm tải dung lượng lưu trữ trên server và tăng tốc độ tải ảnh.

URL của ảnh được trả về từ Cloudinary sẽ được lưu vào cơ sở dữ liệu và sử dụng để hiển thị hình ảnh xe trong ứng dụng.

Các ảnh cũ sẽ được xóa khỏi Cloudinary khi người dùng cập nhật hình ảnh mới để tối ưu không gian lưu trữ.

* Ý nghĩa và lợi ích:

Tích hợp Cloudinary giúp tối ưu hóa việc lưu trữ hình ảnh, tăng cường khả năng truy xuất nhanh và giảm tải cho server.

Chức năng đăng ký xe cho phép chủ xe dễ dàng quản lý danh sách xe và cập nhật thông tin nhanh chóng.

Người thuê xe có thể dễ dàng tiếp cận thông tin xe và hình ảnh chi tiết, nâng cao trải nghiệm người dùng.

## **Quản lý tài khoản và thông tin cá nhân**

Ứng dụng Thuê xe oto cung cấp tính năng quản lý tài khoản và thông tin cá nhân, cho phép người dùng cập nhật các thông tin cơ bản như tên, email, số điện thoại, mật khẩu và ảnh đại diện. Chức năng này giúp người dùng duy trì và quản lý hồ sơ cá nhân một cách linh hoạt và bảo mật.

* Cập nhật thông tin cá nhân:

Người dùng có thể cập nhật các thông tin cá nhân bao gồm: tên, số điện thoại, địa chỉ email và giới tính.

Khi người dùng nhấn “Lưu thay đổi”, hệ thống sẽ thực hiện cập nhật dữ liệu vào cơ sở dữ liệu và phản hồi thông báo thành công hoặc thất bại.

* Ý nghĩa và lợi ích:

Tính năng quản lý tài khoản giúp người dùng dễ dàng duy trì hồ sơ cá nhân và kiểm soát thông tin cá nhân.

Việc tích hợp email xác thực khi thay đổi thông tin nhạy cảm như mật khẩu và email giúp tăng cường tính bảo mật.

Lưu trữ ảnh đại diện trên Cloudinary giúp giảm tải cho server và cải thiện tốc độ tải ảnh.

## **Tìm kiếm, lọc và xem danh sách xe**

Ứng dụng Thuê xe oto cung cấp chức năng tìm kiếm và lọc xe, giúp người dùng dễ dàng tiếp cận các xe cho thuê dựa trên nhiều tiêu chí khác nhau như giá thuê, loại xe, vị trí, số ghế và trạng thái xe. Tính năng này góp phần nâng cao trải nghiệm người dùng khi tìm kiếm xe phù hợp với nhu cầu cá nhân.

* Tìm kiếm xe:

Người dùng có thể tìm kiếm xe liên quan đến địa chỉ.

Hệ thống sẽ thực hiện truy vấn dữ liệu và trả về danh sách xe phù hợp

Kết quả tìm kiếm sẽ hiển thị các thông tin cơ bản như: tên xe, giá thuê, số ghế, hình ảnh xe.

* Lọc xe theo tiêu chí:

Người dùng có thể áp dụng bộ lọc theo các tiêu chí sau:

* Hãng xe: Lọc theo hãng xe như Toyota, Vinfast, …
* Số ghế: Lọc xe theo số ghế ngồi (4 chỗ, 7 chỗ, 16 chỗ, ...).
* Số tự động hay số sàn.
* Loại nhiên liệu tiêu thụ (Dầu, Diesel, EV).
* Giá thuê: Lọc xe theo mức giá thuê mong muốn.

Sau khi áp dụng bộ lọc, hệ thống sẽ cập nhật danh sách xe tương ứng và hiển thị các xe phù hợp.

* Ý nghĩa và lợi ích:

Tính năng tìm kiếm và lọc xe giúp người dùng tiết kiệm thời gian và nhanh chóng tiếp cận các xe phù hợp với nhu cầu.

Khả năng lọc xe theo nhiều tiêu chí giúp cải thiện trải nghiệm người dùng và tăng khả năng kết nối giữa người thuê và chủ xe.

Giao diện danh sách xe được thiết kế trực quan, dễ sử dụng và hiển thị thông tin xe một cách rõ ràng, giúp người dùng dễ dàng so sánh và lựa chọn.

## **Xem chi tiết thông tin xe, xem đánh giá xe**

Ứng dụng Thuê xe oto cung cấp chức năng xem chi tiết thông tin xe và đánh giá xe, cho phép người dùng tiếp cận các thông tin cụ thể về xe cũng như tham khảo các đánh giá từ những người dùng trước đó. Đây là một tính năng quan trọng, giúp người thuê xe có cái nhìn tổng quan về xe trước khi quyết định đặt thuê.

* Xem chi tiết thông tin xe:

Khi người dùng nhấn vào một xe trong danh sách xe, hệ thống sẽ hiển thị trang chi tiết thông tin xe.

Trang chi tiết bao gồm các thông tin sau: hhình ảnh xe, tên xe, hãng xe, giá thuê theo ngày số ghế, loại nhiên liệu và mức tiêu thụ nhiên liệu.

Ngoài ra, người dùng có thể xem thông tin liên hệ của chủ xe để liên hệ trực tiếp khi có nhu cầu thuê xe.

* Xem đánh giá xe:

Bên dưới phần chi tiết xe, hệ thống sẽ hiển thị danh sách đánh giá từ những người dùng đã từng thuê xe đó.

Mỗi đánh giá bao gồm các thông tin sau:

Tên người đánh giá, hình ảnh đại diện (nếu có).

Nội dung đánh giá và số sao (từ 1 đến 5 sao).

Ngày đánh giá và thời gian thuê xe.

Người dùng có thể xem tổng điểm đánh giá trung bình và số lượng đánh giá đã nhận.

* Giao diện đánh giá xe:

Giao diện đánh giá được thiết kế trực quan, sắp xếp theo thời gian để người dùng dễ dàng theo dõi các đánh giá mới nhất.

Nếu có nhiều đánh giá, hệ thống sẽ hỗ trợ tính năng phân trang và tải thêm để người dùng có thể xem thêm các đánh giá cũ.

* Ý nghĩa và lợi ích:

Chức năng xem chi tiết xe cung cấp cho người thuê cái nhìn đầy đủ về xe, giúp họ dễ dàng ra quyết định thuê xe phù hợp.

Tính năng đánh giá xe giúp người thuê tham khảo ý kiến từ những người dùng trước đó, tăng tính minh bạch và độ tin cậy của thông tin xe.

Hệ thống đánh giá cũng tạo động lực cho chủ xe duy trì chất lượng dịch vụ, từ đó nâng cao trải nghiệm người dùng.

## **Tạo đơn đặt xe, xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe**

Ứng dụng Thuê xe oto cung cấp chức năng tạo đơn đặt xe, xem chi tiết thông tin đơn đặt và hủy đơn đặt xe. Bên cạnh đó, hệ thống cũng cho phép chủ xe xác nhận hoặc từ chối yêu cầu thuê xe, đảm bảo quá trình thuê xe diễn ra minh bạch và rõ ràng.

* Tạo đơn đặt xe:

Người thuê xe chọn xe cần thuê từ danh sách xe.

Tại trang chi tiết xe, người thuê nhấn vào nút “Đặt xe” để mở giao diện tạo đơn đặt xe.

Người dùng cung cấp các thông tin cần thiết cho đơn đặt xe, bao gồm:

Ngày và giờ bắt đầu thuê.

Ngày và giờ kết thúc thuê.

Địa điểm nhận xe và trả xe.

Tùy chọn bổ sung: thuê tài xế, yêu cầu giao xe tận nơi, áp dụng bảo hiểm xe hằng ngày.

Sau khi xác nhận các thông tin, người dùng nhấn “Gửi yêu cầu” để tạo đơn đặt xe.

Đơn đặt xe sẽ được lưu trữ vào cơ sở dữ liệu và gửi thông báo tới chủ xe để chờ xác nhận.

* Xem thông tin đơn đặt xe:

Người dùng có thể truy cập vào mục “Đơn đặt xe” để xem danh sách các đơn đặt xe hiện tại.

Tại đây, hệ thống sẽ hiển thị các thông tin tóm tắt bao gồm: tên xe, ngày thuê, ngày trả, trạng thái đơn đặt (chờ xác nhận, đã xác nhận, đã hủy).

Người dùng có thể nhấn vào từng đơn đặt để xem chi tiết, bao gồm:

Thông tin xe và chủ xe.

Tổng chi phí thuê xe.

Tùy chọn thuê tài xế và giao xe tận nơi.

Thời gian thuê xe và trạng thái đơn đặt.

* Hủy đơn đặt xe:

Người thuê xe có thể hủy đơn đặt xe trước khi chủ xe xác nhận.

Khi nhấn “Hủy đơn đặt”, hệ thống sẽ hiển thị ô để nhập lý do hủy và nút xác nhận.

Nếu người thuê xác nhận hủy, đơn đặt xe sẽ được cập nhật trạng thái thành “Đã hủy bởi chủ xe”.

* Xác nhận cho thuê xe:

Chủ xe có thể xem xét thông tin đơn đặt và chọn một trong hai tùy chọn:

Xác nhận cho thuê: Đơn đặt xe sẽ chuyển sang trạng thái “Đã xác nhận”.

Từ chối yêu cầu: Đơn đặt xe sẽ được cập nhật trạng thái “Đã từ chối”.

* Ý nghĩa và lợi ích:

Tính năng đặt xe giúp người thuê dễ dàng lên lịch thuê xe và chủ xe có thể quản lý các yêu cầu thuê xe một cách hiệu quả.

Cơ chế xác nhận và hủy đơn đặt giúp duy trì sự minh bạch và tránh các trường hợp xung đột hoặc hiểu nhầm giữa hai bên.

Giao diện quản lý đơn đặt xe rõ ràng, chi tiết giúp người thuê và chủ xe dễ dàng theo dõi tình trạng và lịch sử thuê xe.

## **Đánh giá dịch vụ thuê xe**

Ứng dụng Thuê xe oto cung cấp chức năng đánh giá dịch vụ thuê xe, cho phép người thuê để lại nhận xét và đánh giá về xe và chủ xe sau khi hoàn tất quá trình thuê. Đây là một tính năng quan trọng giúp xây dựng uy tín cho chủ xe và cung cấp thông tin tham khảo cho các người thuê xe sau.

* Để lại nhận xét:

Sau khi hoàn tất quá trình thuê xe, người dùng có thể truy cập vào mục “Chuyến của tôi” để xem lại các đơn đặt.

Tại mỗi đơn thuê đã hoàn thành, người dùng sẽ có tùy chọn để để lại nhận xét.

Người dùng có thể nhập nội dung nhận xét và đánh giá số sao (từ 1 đến 5 sao).

Khi người dùng nhấn nút “Gửi nhận xét”, hệ thống sẽ lưu nhận xét vào cơ sở dữ liệu, đồng thời hiển thị lên màn hình.

* Hiển thị nhận xét:

Tại trang chi tiết xe, hệ thống sẽ hiển thị danh sách các nhận xét từ người thuê trước đó.

Mỗi nhận xét sẽ bao gồm các thông tin:

Tên người đánh giá và hình ảnh đại diện (nếu có).

Số sao đánh giá.

Nội dung nhận xét chi tiết.

Thời gian đánh giá..

* Quản lý nhận xét:

Chủ xe có thể xem các nhận xét đã nhận được thông qua xem chi tiết đơn đặt. Sử dụng tính năng nhắn tin để liên hệ về nhận xét và đơn đặt.

Người dùng đặt xe có thể chỉnh sửa hoặc xóa nhận xét trước đó để thêm nhận xét mới

* Ý nghĩa và lợi ích:

Tính năng nhận xét giúp xây dựng hệ thống đánh giá công khai, minh bạch, từ đó tạo niềm tin cho người thuê và chủ xe.

Thông qua các nhận xét, chủ xe có thể cải thiện chất lượng dịch vụ dựa trên phản hồi từ người thuê.

Người thuê xe có thêm nguồn thông tin tham khảo trước khi quyết định thuê xe, từ đó đưa ra lựa chọn phù hợp hơn.

## **Giao** diện **các chức năng**

## **Đăng** ký**, đăng nhập và quên mật khẩu cho người dùng tích hợp xác nhận OTP qua email**

|  |  |  |
| --- | --- | --- |

|  |  |
| --- | --- |

## Nhắn **tin tích hợp WebSocket thời gian thực**

|  |  |
| --- | --- |

## **Đăng** ký **xe, xem các xe đã đăng ký, chỉnh sửa thông tin xe tích hợp Cloudinary**

|  |  |
| --- | --- |

|  |  |
| --- | --- |

## Quản **lý tài khoản và thông tin cá nhân**

|  |  |  |
| --- | --- | --- |

## Tìm **kiếm, lọc và xem danh sách xe**

|  |  |  |
| --- | --- | --- |

## Xem **chi tiết thông tin xe, xem đánh giá xe**

|  |  |
| --- | --- |

|  |  |
| --- | --- |

## Tạo **đơn đặt xe, xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe**

|  |  |
| --- | --- |

|  |  |  |
| --- | --- | --- |

## Đánh **giá dịch vụ thuê xe, chỉnh sửa, xóa đánh giá.**

|  |  |  |
| --- | --- | --- |

|  |  |
| --- | --- |

## **Mô tả code của chức năng**

## **Đăng ký,** đăng **nhập và quên mật khẩu cho người dùng tích hợp xác nhận OTP qua email**

### *Đăng ký tài khoản*

* Luồng từ Android:

Người dùng nhập thông tin đăng ký gồm username, email, password, confirmPassword tại RegisterActivity.

| String username = usernameEditText.getText().toString().trim();  String email = emailEditText.getText().toString().trim();  String password = passwordEditText.getText().toString().trim();  String confirmPassword = confirmPasswordEditText.getText().toString().trim(); |
| --- |

Tạo đối tượng RegisterRequest và gọi phương thức apiService.register(request) trong ApiService.java.

| RegisterRequest request = new RegisterRequest(username, email, password, confirmPassword);  apiService.register(request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<String> apiResponse = response.body();  if (apiResponse.isSuccess()) {  Toast.makeText(RegisterActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  Intent intent = new Intent(RegisterActivity.this, VerifyOTPActivity.class);  intent.putExtra("email", email);  startActivity(intent);  finish();  } else {  Toast.makeText(RegisterActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  }  } else {  Toast.makeText(RegisterActivity.this, "Đăng ký thất bại. Vui lòng thử lại.", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(RegisterActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  }); |
| --- |

Nếu đăng ký thành công, API trả về thông báo "Đăng ký thành công. Vui lòng kiểm tra email để xác nhận OTP." và chuyển người dùng đến VerifyOTPActivity cùng với email.

* Backend - AuthController.java:

Endpoint: /api/auth/register

Controller nhận yêu cầu và gọi phương thức register() trong AuthService.java.

| @PostMapping("/register")  public ApiResponse<String> register(@RequestBody @Valid RegisterRequest request) {  String message = authService.register(  request.getUsername(),  request.getEmail(),  request.getPassword(),  request.getConfirmPassword()  );  // Trả về success=false nếu có lỗi  if (message.contains("đã tồn tại") || message.contains("không khớp") ||  message.contains("đã được sử dụng") || message.contains("chưa xác minh")) {  return new ApiResponse<>(false, message);  }  return new ApiResponse<>(true, message);  } |
| --- |

* Backend - AuthService.java:

Kiểm tra password và confirmPassword có khớp không.

Kiểm tra email đã tồn tại hay chưa:

Nếu đã tồn tại và chưa xác minh, gửi lại OTP mới.

Nếu chưa tồn tại, tạo tài khoản mới và gửi OTP qua email.

| public String register(String username, String email, String password, String confirmPassword) {  if (!password.equals(confirmPassword)) {  return "Mật khẩu và nhập lại mật khẩu không khớp.";  }  Optional<Account> existingByEmail = accountRepo.findByEmail(email);  if (existingByEmail.isPresent()) {  Account existingAccount = existingByEmail.get();  if (existingAccount.isActive()) {  return "Email đã được sử dụng.";  }  if (isOtpRecentlySent(email)) {  return "Tài khoản đã tồn tại nhưng chưa xác minh. Vui lòng đợi 5 phút để gửi lại mã OTP.";  }  Otp otp = new Otp();  otp.setEmail(email);  otp.setOtpCode(generateOtp());  otp.setCreatedAt(LocalDateTime.now());  otp.setExpiresAt(LocalDateTime.now().plusMinutes(5));  otp.setUsed(false);  otpRepo.save(otp);  sendOtpEmail(email, otp.getOtpCode());  return "Tài khoản đã tồn tại nhưng chưa xác minh. Đã gửi lại mã OTP.";  }  if (accountRepo.existsByUsername(username)) {  return "Username đã tồn tại.";  }  Account account = new Account();  account.setUsername(username);  account.setEmail(email);  account.setPassword(passwordEncoder.encode(password));  account.setActive(false);  accountRepo.save(account);  Otp otp = new Otp();  otp.setEmail(email);  otp.setOtpCode(generateOtp());  otp.setCreatedAt(LocalDateTime.now());  otp.setExpiresAt(LocalDateTime.now().plusMinutes(5));  otp.setUsed(false);  otpRepo.save(otp);  sendOtpEmail(email, otp.getOtpCode());  return "Đăng ký thành công. Vui lòng kiểm tra email để xác nhận OTP.";  } |
| --- |

* Backend - Gửi OTP qua Email:

Tạo đối tượng Otp và lưu vào bảng otp.

Gửi email OTP qua JavaMailSender với nội dung OTP và thời gian hiệu lực 5 phút.

### *Xác minh OTP*

* Luồng từ Android:

Người dùng nhập mã OTP tại VerifyOTPActivity.

Tạo đối tượng VerifyOtpRequest và gọi phương thức apiService.verifyOtp(request) trong ApiService.java.

| private void verifyOtp() {  String otpCode = otpEditText.getText().toString().trim();  if (otpCode.isEmpty()) {  Toast.makeText(this, "Vui lòng nhập mã OTP", Toast.LENGTH\_SHORT).show();  return;  }  VerifyOtpRequest request = new VerifyOtpRequest(email, otpCode);  apiService.verifyOtp(request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<String> apiResponse = response.body();  if (apiResponse.isSuccess()) {  Toast.makeText(VerifyOTPActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  // Xác thực thành công, quay về LoginActivity  Intent intent = new Intent(VerifyOTPActivity.this, LoginActivity.class);  startActivity(intent);  finish();  } else {  Toast.makeText(VerifyOTPActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  }  } else {  Toast.makeText(VerifyOTPActivity.this, "Xác minh thất bại. Vui lòng thử lại.", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(VerifyOTPActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - AuthController.java:

Endpoint: /api/auth/verify-otp

Controller gọi verifyOtp() trong AuthService.java.

| @PostMapping("/verify-otp")  public ApiResponse<String> verifyOtp(@RequestBody @Valid VerifyOtpRequest request) {  String message = authService.verifyOtp(request.getEmail(), request.getOtpCode());  // Kiểm tra lỗi, trả về success=false nếu có vấn đề  if (message.contains("không hợp lệ") || message.contains("không tồn tại") || message.contains("hết hạn")) {  return new ApiResponse<>(false, message);  }  return new ApiResponse<>(true, message);  } |
| --- |

* Backend - AuthService.java:

Kiểm tra OTP có tồn tại và chưa được sử dụng.

Kiểm tra thời gian hết hạn OTP.

Nếu hợp lệ, kích hoạt tài khoản và đánh dấu OTP là "used".

| public String verifyOtp(String email, String code) {  Optional<Otp> otpOpt = otpRepo.findByEmailAndOtpCodeAndIsUsedFalse(email, code);  if (otpOpt.isEmpty()) return "OTP không hợp lệ hoặc đã được sử dụng";  Otp otp = otpOpt.get();  if (otp.getExpiresAt().isBefore(LocalDateTime.now())) return "OTP đã hết hạn";  Optional<Account> accountOpt = accountRepo.findByEmail(email);  if (accountOpt.isEmpty()) return "Tài khoản không tồn tại";  Account account = accountOpt.get();  account.setActive(true);  accountRepo.save(account);  otp.setUsed(true);  otpRepo.save(otp);  User user = new User();  user.setAccount(account);  user.setRole("customer");  userRepo.save(user);  return "Xác thực thành công. Bạn có thể đăng nhập.";  } |
| --- |

### *Đăng nhập*

* Luồng từ Android:

Người dùng nhập username và password tại LoginActivity.

Tạo đối tượng LoginRequest và gọi apiService.login(request) trong ApiService.java.

Nếu thành công, lưu JWT token vào SharedPrefManager và chuyển đến HomeActivity.

| private void login() {  String username = usernameEditText.getText().toString().trim();  String password = passwordEditText.getText().toString().trim();  if (username.isEmpty() || password.isEmpty()) {  Toast.makeText(this, "Vui lòng nhập đầy đủ thông tin", Toast.LENGTH\_SHORT).show();  return;  }  LoginRequest request = new LoginRequest(username, password);  Gson gson = new Gson();  String json = gson.toJson(request);  Log.d("LoginRequest", "Request: " + new Gson().toJson(request));  Log.d("LoginRequest", "URL: " + apiService.login(request).request().url());  apiService.login(request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<String> apiResponse = response.body();  if (apiResponse.isSuccess()) {  // ✅ Lưu token vào SharedPreferences  String token = apiResponse.getData();  SharedPrefManager.getInstance(getApplicationContext()).saveToken(token);  Toast.makeText(LoginActivity.this, "Đăng nhập thành công", Toast.LENGTH\_SHORT).show();  Intent intent = new Intent(LoginActivity.this, HomeActivity.class);  startActivity(intent);  finish();  } else {  Toast.makeText(LoginActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  }  } else {  Toast.makeText(LoginActivity.this, "Đăng nhập thất bại. Vui lòng thử lại.", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(LoginActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_LONG).show();  }  });  } |
| --- |

* Backend - AuthController.java:

Endpoint: /api/auth/login

Controller gọi login() trong AuthService.java.

| @PostMapping("/login")  public ApiResponse<String> login(@RequestBody @Valid LoginRequest request) {  String message = authService.login(request.getUsername(), request.getPassword());  // Nếu thông báo có lỗi, trả về success=false và lỗi  if (message.contains("không tồn tại") ||  message.contains("chưa được kích hoạt") ||  message.contains("không đúng") ||  message.contains("Không tìm thấy")) {  return new ApiResponse<>(false, message);  }  // Nếu login thành công, trả về success=true và token  return new ApiResponse<>(true, "Đăng nhập thành công", message);  } |
| --- |

* Backend - AuthService.java:

Kiểm tra tài khoản có tồn tại và đã kích hoạt.

Kiểm tra username và password có khớp không.

Nếu hợp lệ, tạo JWT token thông qua JwtService.java và trả về cho Android.

| public String login(String username, String password) {  Optional<Account> accountOpt = accountRepo.findByUsername(username);  if (accountOpt.isEmpty()) return "Tài khoản không tồn tại";  Account account = accountOpt.get();  if (!account.isActive()) return "Tài khoản chưa được kích hoạt";  if (!passwordEncoder.matches(password, account.getPassword())) {  return "Mật khẩu không đúng";  }    Optional<User> userOpt = userRepo.findAll().stream()  .filter(u -> u.getAccount().getUsername().equals(username))  .findFirst();  if (userOpt.isEmpty()) return "Không tìm thấy thông tin người dùng";  String role = userOpt.get().getRole();  String token = jwtService.generateToken(username, role, userOpt.get().getUserId());  return token;  } |
| --- |

### *Quên mật khẩu và đặt lại mật khẩu*

* Luồng từ Android:

Người dùng nhập email tại ForgotPasswordActivity và yêu cầu gửi OTP qua email.

Gọi apiService.forgotPassword(request).

Sau khi nhận OTP, người dùng nhập otpCode và newPassword để đặt lại mật khẩu.

Gọi apiService.resetPassword(request).

| private void resetPassword() {  String email = emailEditText.getText().toString().trim();  String otpCode = otpEditText.getText().toString().trim();  String newPassword = newPasswordEditText.getText().toString().trim();  if (email.isEmpty() || otpCode.isEmpty() || newPassword.isEmpty()) {  Toast.makeText(this, "Vui lòng nhập đầy đủ thông tin", Toast.LENGTH\_SHORT).show();  return;  }  ResetPasswordRequest request = new ResetPasswordRequest(email, otpCode, newPassword);  apiService.resetPassword(request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<String> apiResponse = response.body();  Toast.makeText(ForgotPasswordActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  if (apiResponse.isSuccess()) {  Intent intent = new Intent(ForgotPasswordActivity.this, LoginActivity.class);  startActivity(intent);  finish();  }  } else {  Toast.makeText(ForgotPasswordActivity.this, "Đổi mật khẩu thất bại. Vui lòng thử lại.", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(ForgotPasswordActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

| private void sendOtp() {  String email = emailEditText.getText().toString().trim();  if (email.isEmpty()) {  Toast.makeText(this, "Vui lòng nhập email", Toast.LENGTH\_SHORT).show();  return;  }  ForgotPasswordRequest request = new ForgotPasswordRequest(email);  apiService.forgotPassword(request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<String> apiResponse = response.body();  Toast.makeText(ForgotPasswordActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  if (apiResponse.isSuccess()) {  startOtpCountdown();  }  } else {  Toast.makeText(ForgotPasswordActivity.this, "Gửi OTP thất bại. Vui lòng thử lại.", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(ForgotPasswordActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - AuthController.java:

Endpoint: /api/auth/forgot-password và /api/auth/reset-password

| @PostMapping("/forgot-password")  public ApiResponse<String> forgotPassword(@RequestBody @Valid ForgotPasswordRequest request) {  String message = authService.forgotPassword(request.getEmail());  // Nếu email không tồn tại trong hệ thống  if (message.contains("không tồn tại") || message.contains("vừa yêu cầu")) {  return new ApiResponse<>(false, message);  }  return new ApiResponse<>(true, message);  } |
| --- |

| @PostMapping("/reset-password")  public ApiResponse<String> resetPassword(@RequestBody @Valid ResetPasswordRequest request) {  String message = authService.resetPassword(  request.getEmail(),  request.getOtpCode(),  request.getNewPassword()  );  // Kiểm tra nếu OTP không hợp lệ hoặc hết hạn  if (message.contains("không hợp lệ") || message.contains("hết hạn") || message.contains("không tồn tại")) {  return new ApiResponse<>(false, message);  }  return new ApiResponse<>(true, message);  } |
| --- |

* Backend - AuthService.java:

Kiểm tra email có tồn tại.

Nếu có, gửi OTP qua email và lưu vào bảng otp.

Khi đặt lại mật khẩu, kiểm tra otpCode có hợp lệ và chưa hết hạn.

Nếu hợp lệ, cập nhật password và đánh dấu OTP là "used".

| public String forgotPassword(String email) {  Optional<Account> accountOpt = accountRepo.findByEmail(email);  if (accountOpt.isEmpty()) return "Email không tồn tại trong hệ thống";  if (isOtpRecentlySent(email)) {  return "Bạn vừa yêu cầu OTP. Vui lòng đợi 5 phút để gửi lại.";  }  Otp otp = new Otp();  otp.setEmail(email);  otp.setOtpCode(generateOtp());  otp.setCreatedAt(LocalDateTime.now());  otp.setExpiresAt(LocalDateTime.now().plusMinutes(5));  otp.setUsed(false);  otpRepo.save(otp);  sendOtpEmail(email, otp.getOtpCode());  return "Đã gửi mã OTP để đặt lại mật khẩu. Vui lòng kiểm tra email.";  } |
| --- |

| public String resetPassword(String email, String otpCode, String newPassword) {  Optional<Otp> otpOpt = otpRepo.findByEmailAndOtpCodeAndIsUsedFalse(email, otpCode);  if (otpOpt.isEmpty()) return "OTP không hợp lệ hoặc đã được sử dụng";  Otp otp = otpOpt.get();  if (otp.getExpiresAt().isBefore(LocalDateTime.now())) return "OTP đã hết hạn";  Optional<Account> accountOpt = accountRepo.findByEmail(email);  if (accountOpt.isEmpty()) return "Tài khoản không tồn tại";  Account account = accountOpt.get();  account.setPassword(passwordEncoder.encode(newPassword));  accountRepo.save(account);  otp.setUsed(true);  otpRepo.save(otp);  return "Đặt lại mật khẩu thành công!";  } |
| --- |

| private void sendOtpEmail(String toEmail, String otpCode) {  try {  MimeMessage message = mailSender.createMimeMessage();  MimeMessageHelper helper = new MimeMessageHelper(message, true);    helper.setTo(toEmail);  helper.setSubject("Mã OTP Xác Thực Tài Khoản");    String htmlContent = "<html><body style='font-family: Arial, sans-serif;'>"  + "<h2 style='color: #2e6c80;'>Xác thực tài khoản</h2>"  + "<p>Xin chào,</p>"  + "<p>Mã OTP của bạn là: <strong style='color: #d9534f; font-size: 20px;'>" + otpCode + "</strong></p>"  + "<p>Mã OTP có hiệu lực trong vòng <strong>5 phút</strong>.</p>"  + "<hr>"  + "<p style='font-size: 12px; color: gray;'>Đây là email tự động, vui lòng không trả lời email này.</p>"  + "</body></html>";    helper.setText(htmlContent, true); // ⚡ true để mail hiểu đây là HTML    mailSender.send(message);  } catch (MessagingException | MailException e) {  e.printStackTrace();  }  } |
| --- |

### 

### *Kiểm* tra JWT Token:

Mỗi request cần xác thực sẽ được kiểm tra JWT token qua JwtAuthFilter.java.

Nếu token hợp lệ, request sẽ tiếp tục. Nếu không, trả về lỗi "Unauthorized".

| @Override  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)  throws ServletException, IOException {  final String authHeader = request.getHeader("Authorization");  final String jwt;  final String username;  if (authHeader == null || !authHeader.startsWith("Bearer ")) {  filterChain.doFilter(request, response);  return;  }  jwt = authHeader.substring(7);  username = jwtService.extractUsername(jwt);  if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {  Optional<Account> accountOpt = accountRepo.findByUsername(username);  if (accountOpt.isPresent() && jwtService.isTokenValid(jwt)) {  String role = jwtService.extractRole(jwt);  List<GrantedAuthority> authorities = List.of(new SimpleGrantedAuthority("ROLE\_" + role.toUpperCase()));  UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(  username,  null,  authorities  );  authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  SecurityContextHolder.getContext().setAuthentication(authToken);  }  }  filterChain.doFilter(request, response);  } |
| --- |

## **Nhắn tin tích hợp WebSocket thời gian thực**

### *Tải danh sách hội thoại:*

* Luồng từ Android:

Người dùng mở ChatFragment, ứng dụng sẽ tự động gọi loadConversations() để tải danh sách hội thoại.

Tạo đối tượng ApiService và gọi phương thức apiService.getConversations().

API trả về List<ConversationResponse>, dữ liệu này sẽ được đổ vào RecyclerView thông qua ConversationAdapter.

| private void loadConversations() {  String token = SharedPrefManager.getInstance(getContext()).getToken();  apiService.getConversations("Bearer " + token)  .enqueue(new Callback<ApiResponse<List<ConversationResponse>>>() {  @Override  public void onResponse(Call<ApiResponse<List<ConversationResponse>>> call, Response<ApiResponse<List<ConversationResponse>>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<List<ConversationResponse>> apiResponse = response.body();  if (apiResponse.isSuccess()) {  allConversations = apiResponse.getData();  showConversationList(allConversations);  } else {  Toast.makeText(getContext(), apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  }  } else {  Toast.makeText(getContext(), "Không tải được danh sách hội thoại", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<List<ConversationResponse>>> call, Throwable t) {  Toast.makeText(getContext(), "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: /api/user/conversations

Controller nhận yêu cầu và gọi phương thức getAllConversationsForUser() trong ConversationService.java.

| /\*\*  \* API: GET /api/user/conversations  \* Lấy tất cả đoạn hội thoại của user đang login  \*/  @GetMapping("/conversations")  public ResponseEntity<ApiResponse<List<ConversationResponse>>> getAllConversations(Authentication authentication) {  try {  List<ConversationResponse> conversations = conversationService.getAllConversationsForUser(authentication);  return ResponseEntity.ok(new ApiResponse<>(true, "Lấy danh sách hội thoại thành công", conversations));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi lấy danh sách hội thoại", null));  }  } |
| --- |

* Backend - ConversationService.java:

Lấy danh sách hội thoại dựa trên người dùng hiện tại.

Với mỗi hội thoại, lấy tin nhắn cuối cùng (lastMessageContent, lastMessageTimestamp) để hiển thị thông tin sơ lược.

| public List<ConversationResponse> getAllConversationsForUser(Authentication authentication) {  String username = authentication.getName();  User currentUser = userRepository.findByAccount\_Username(username)  .orElseThrow(() -> new ResourceNotFoundException("User không tồn tại"));  List<Conversation> conversations = conversationRepository.findAllByUser(currentUser);  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");  return conversations.stream()  .map(c -> {  User receiver = c.getUser1().equals(currentUser) ? c.getUser2() : c.getUser1();  Message lastMessage = messageRepository.findTopByConversationOrderByTimestampDesc(c).orElse(null);  String lastContent = lastMessage != null ? lastMessage.getContent() : "";  String lastTime = lastMessage != null ? lastMessage.getTimestamp().atZoneSameInstant(ZoneId.of("Asia/Ho\_Chi\_Minh")).toLocalDateTime().format(formatter): "";  return new ConversationResponse(  c.getConversationId(),  receiver.getUserId(),  receiver.getName(),  receiver.getImageUrl(),  lastContent,  lastTime  );  })  .collect(Collectors.toList());  } |
| --- |

### *Xem chi tiết hội thoại và tải tin nhắn*

* Luồng từ Android:

Khi người dùng click vào 1 hội thoại trong ChatFragment, ChatActivity sẽ được khởi tạo.

ChatActivity nhận conversationId và receiverId từ Intent.

| private void showConversationList(List<ConversationResponse> list) {  adapter = new ConversationAdapter(getContext(), list, conversation -> {  Intent intent = new Intent(getContext(), ChatActivity.class);  intent.putExtra("conversationId", conversation.getConversationId());  intent.putExtra("receiverId", conversation.getReceiverId());  intent.putExtra("receiverName", conversation.getReceiverName());  intent.putExtra("receiverImage", conversation.getReceiverImage());  startActivity(intent);  });  recyclerView.setAdapter(adapter);  } |
| --- |

Gọi API /api/user/messages/{conversationId} để tải toàn bộ tin nhắn của hội thoại qua loadMessages().

API trả về List<MessageResponse>, dữ liệu này sẽ được hiển thị qua MessageAdapter.

| private void loadMessages() {  String token = "Bearer " + SharedPrefManager.getInstance(this).getToken();  apiService.getMessagesByConversation(conversationId, token).enqueue(new Callback<ApiResponse<List<MessageResponse>>>() {  @Override  public void onResponse(Call<ApiResponse<List<MessageResponse>>> call, Response<ApiResponse<List<MessageResponse>>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<List<MessageResponse>> apiResponse = response.body();  if (apiResponse.isSuccess()) {  messageList.clear();  messageList.addAll(apiResponse.getData());  adapter.notifyDataSetChanged();  recyclerMessages.scrollToPosition(messageList.size() - 1);  } else {  Toast.makeText(ChatActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  }  } else {  Toast.makeText(ChatActivity.this, "Không tải được tin nhắn", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<List<MessageResponse>>> call, Throwable t) {  Toast.makeText(ChatActivity.this, "Lỗi mạng: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: /api/user/messages/{conversationId}

Controller gọi MessageService.getMessagesByConversation() để lấy tin nhắn từ DB.

| /\*\*  \* API: GET /api/user/messages/{conversationId}  \* Lấy danh sách tin nhắn trong một đoạn hội thoại  \*/  @GetMapping("/messages/{conversationId}")  public ResponseEntity<ApiResponse<List<MessageResponse>>> getMessagesByConversation(  @PathVariable Long conversationId,  Authentication authentication) {  try {  List<MessageResponse> messages = messageService.getMessagesByConversation(conversationId, authentication);  return ResponseEntity.ok(new ApiResponse<>(true, "Lấy danh sách tin nhắn thành công", messages));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi lấy tin nhắn", null));  }  } |
| --- |

* Backend - MessageService.java:

Tìm tất cả tin nhắn thuộc conversationId.

Sắp xếp tin nhắn theo thời gian (timestamp) để đảm bảo thứ tự.

| public List<MessageResponse> getMessagesByConversation(Long conversationId, Authentication authentication) {  Conversation conversation = conversationRepository.findById(conversationId)  .orElseThrow(() -> new ResourceNotFoundException("Không tìm thấy cuộc hội thoại"));  List<Message> messages = messageRepository.findByConversationOrderByTimestampAsc(conversation);  return messages.stream().map(msg -> new MessageResponse(  msg.getMessageId(),  conversationId,  msg.getSender().getUserId(),  msg.getReceiver().getUserId(),  msg.getContent(),  msg.getSharedPostId(),  msg.getMessageType(),  msg.getStatus(),  msg.getTimestamp()  )).collect(Collectors.toList());  } |
| --- |

### *Gửi tin nhắn*

* Luồng từ Android:

Người dùng nhập nội dung tin nhắn và nhấn nút gửi (btnSend).

Tạo đối tượng MessageRequest với receiverId, content, messageType.

Gọi API /api/user/message để gửi tin nhắn.

Nếu gửi thành công, ChatActivity sẽ nhận tin nhắn mới qua WebSocket và tự động cập nhật RecyclerView.

| private void sendMessage() {  String content = edtMessage.getText().toString().trim();  if (content.isEmpty()) return;  String token = "Bearer " + SharedPrefManager.getInstance(this).getToken();  MessageRequest request = new MessageRequest(receiverId, content);  apiService.sendMessage(request, token).enqueue(new Callback<ApiResponse<MessageResponse>>() {  @Override  public void onResponse(Call<ApiResponse<MessageResponse>> call, Response<ApiResponse<MessageResponse>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<MessageResponse> apiResponse = response.body();  if (apiResponse.isSuccess()) {  edtMessage.setText("");  } else {  Toast.makeText(ChatActivity.this, apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  }  } else {  Toast.makeText(ChatActivity.this, "Gửi tin nhắn thất bại", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<MessageResponse>> call, Throwable t) {  Toast.makeText(ChatActivity.this, "Lỗi mạng khi gửi: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: /api/user/message

Controller nhận MessageRequest và gọi MessageService.sendMessage().

| /\*\*  \* API: POST /api/user/message  \* Gửi tin nhắn mới (TEXT hoặc POST)  \*/  @PostMapping("/message")  public ResponseEntity<ApiResponse<MessageResponse>> sendMessage(@RequestBody MessageRequest request, Authentication authentication) {  try {  MessageResponse response = messageService.sendMessage(request, authentication);  messagingTemplate.convertAndSend("/topic/conversations/" + response.getConversationId(), response);  return ResponseEntity.ok(new ApiResponse<>(true, "Gửi tin nhắn thành công", response));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi gửi tin nhắn", null));  }  } |
| --- |

* Backend - MessageService.java:

Kiểm tra receiverId có tồn tại.

Nếu chưa có hội thoại (Conversation), tạo mới.

Lưu Message vào DB và đánh dấu status = SENDING.

Gửi tin nhắn qua WebSocket /topic/conversations/{conversationId} để mọi client đang theo dõi nhận được tin nhắn.

| public MessageResponse sendMessage(MessageRequest request, Authentication authentication) {  String username;    // 1. Ưu tiên lấy từ token nếu có (REST API)  if (authentication != null) {  username = authentication.getName();  }  // 2. Nếu không có (WebSocket), lấy từ request  else if (request.getSenderUsername() != null) {  username = request.getSenderUsername(); // thêm field này vào DTO  } else {  throw new IllegalArgumentException("Không xác định được người gửi.");  }    User sender = userRepository.findByAccount\_Username(username)  .orElseThrow(() -> new ResourceNotFoundException("Người gửi không tồn tại"));    User receiver = userRepository.findById(request.getReceiverId())  .orElseThrow(() -> new ResourceNotFoundException("Người nhận không tồn tại"));    if (sender.getUserId().equals(receiver.getUserId())) {  throw new IllegalArgumentException("Không thể gửi tin nhắn cho chính mình.");  }    Conversation conversation = conversationRepository  .findByUser1AndUser2OrUser2AndUser1(sender, receiver, sender, receiver)  .orElseGet(() -> {  Conversation newConversation = new Conversation();  newConversation.setUser1(sender);  newConversation.setUser2(receiver);  return conversationRepository.save(newConversation);  });    Message message = new Message();  message.setConversation(conversation);  message.setSender(sender);  message.setReceiver(receiver);  message.setContent(request.getContent());  message.setMessageType(request.getMessageType());  message.setSharedPostId(request.getSharedPostId());  message.setStatus(MessageStatus.SENDING);    Message saved = messageRepository.save(message);    return new MessageResponse(  saved.getMessageId(),  conversation.getConversationId(),  sender.getUserId(),  receiver.getUserId(),  saved.getContent(),  saved.getSharedPostId(),  saved.getMessageType(),  saved.getStatus(),  saved.getTimestamp()  );  } |
| --- |

### *WebSocket - Gửi và nhận tin nhắn*

* Luồng từ Android:

Khi ChatActivity được mở, ChatWebSocketClient sẽ kết nối tới /ws-chat.

Thực hiện SUBSCRIBE tới /topic/conversations/{conversationId} để lắng nghe tin nhắn mới.

Khi có tin nhắn mới, onMessage() sẽ được gọi và cập nhật giao diện.

| socketClient = new ChatWebSocketClient(conversationId, json -> {  runOnUiThread(() -> {  try {  // Parse từ JSON và thêm vào danh sách  long msgId = json.getLong("messageId");  String content = json.getString("content");  long senderId = json.getLong("senderId");  MessageResponse newMsg = new MessageResponse(  msgId,  conversationId,  senderId,  receiverId,  content,  null,  "TEXT",  "SENDING",  json.getString("timestamp")  );  messageList.add(newMsg);  adapter.notifyItemInserted(messageList.size() - 1);  recyclerMessages.scrollToPosition(messageList.size() - 1);  } catch (Exception e) {  Log.e("ChatActivity", "Lỗi parse tin nhắn WebSocket", e);  }  });  }); |
| --- |

* Backend - WebSocketConfig.java:

Cấu hình WebSocket tại /ws-chat và broker /topic.

Định nghĩa /app là prefix khi client gửi tin nhắn.

| @Override  public void configureMessageBroker(MessageBrokerRegistry config) {  config.enableSimpleBroker("/topic"); // nơi client sẽ nhận tin nhắn  config.setApplicationDestinationPrefixes("/app"); // prefix khi client gửi tin  }  @Override  public void registerStompEndpoints(StompEndpointRegistry registry) {  registry.addEndpoint("/ws-chat")  .setAllowedOriginPatterns("\*") // ✅ dùng patterns thay cho setAllowedOrigins  .withSockJS();  } |
| --- |

* Backend - ChatWebSocketController.java:

Lắng nghe tại /app/chat/{conversationId}.

Khi có tin nhắn mới từ client, gọi MessageService.sendMessage() để lưu vào DB và gửi tới WebSocket.

| /\*\*  \* Nhận tin nhắn qua WebSocket, lưu DB và gửi lại cho các client trong hội thoại  \*/  @MessageMapping("/chat/{conversationId}") // client gửi vào đây: /app/chat/{id}  public void sendMessage(@DestinationVariable Long conversationId, @Payload MessageRequest request) {  // Giả định không có Authentication từ STOMP -> dùng tạm userId từ request nếu cần  MessageResponse savedMessage = messageService.sendMessage(request, null); // null nếu chưa hỗ trợ auth ở đây  // Gửi tin nhắn này đến tất cả client đang theo dõi cuộc hội thoại  messagingTemplate.convertAndSend(  "/topic/conversations/" + conversationId,  savedMessage  );  } |
| --- |

## **Đăng ký xe, xem các xe đã đăng ký, chỉnh sửa thông tin xe tích hợp Cloudinary**

### *Đăng ký* xe

* Luồng từ Android:
  + Người dùng mở màn hình AddCarActivity, nhập thông tin xe gồm: tên xe, hãng xe, mô tả, loại hộp số, số chỗ ngồi, loại nhiên liệu, mức tiêu thụ nhiên liệu, giá thuê, yêu cầu tài xế, vị trí và tải lên hình ảnh xe.
  + Khi nhấn nút Thêm xe, ứng dụng thực hiện:

Tạo đối tượng AddCarRequest từ thông tin đã nhập.

Gọi phương thức apiService.addCar(request) trong ApiService.java.

Ảnh xe sẽ được upload lên Cloudinary thông qua phương thức uploadToCloudinary().

Khi upload thành công, API trả về đường dẫn URL của ảnh.

Gửi request lên server bao gồm URL ảnh và thông tin xe.

Nếu thêm thành công, trả về thông báo "Thêm xe thành công".

| private void registerCar() {  String token = "Bearer " + SharedPrefManager.getInstance(this).getToken();  String name = edtCarName.getText().toString().trim();  String brand = edtBrand.getText().toString().trim();  String description = edtDescription.getText().toString().trim();  String gearType = spnGearType.getSelectedItem().toString();  String location = edtDetailLocation.getText().toString().trim() + ", " + spnLocation.getSelectedItem().toString();  double price = Double.parseDouble(edtPrice.getText().toString().trim());  int seats = Integer.parseInt(((EditText) findViewById(R.id.spnSeats)).getText().toString().trim());  String fuelType = spnFuelType.getSelectedItem().toString();  double energyConsumption = Double.parseDouble(edtEnergyConsumption.getText().toString().trim());  boolean driverRequired = cbDriverRequired.isChecked();  List<String> selectedFeatures = new ArrayList<>();  for (Feature feature : featureList) {  if (feature.isSelected()) {  selectedFeatures.add(feature.getName());  }  }  // Kiểm tra nếu chưa chọn ảnh  if (selectedImageUri == null) {  Toast.makeText(this, "Vui lòng chọn ảnh trước khi đăng ký.", Toast.LENGTH\_SHORT).show();  return;  }  // Nếu chưa upload ảnh, tiến hành upload  if (imageUrl == null) {  Toast.makeText(this, "Đang upload ảnh...", Toast.LENGTH\_SHORT).show();  CloudinaryManager.uploadImage(this, selectedImageUri, new UploadCallback() {  @Override  public void onStart(String requestId) {}  @Override  public void onProgress(String requestId, long bytes, long totalBytes) {}  @Override  public void onSuccess(String requestId, Map resultData) {  imageUrl = (String) resultData.get("secure\_url");  proceedToRegister(name, brand, description, gearType, seats, fuelType, energyConsumption, driverRequired, location, price, selectedFeatures, imageUrl, token);  }  @Override  public void onError(String requestId, ErrorInfo error) {  Toast.makeText(AddCarActivity.this, "Upload ảnh thất bại: " + error.getDescription(), Toast.LENGTH\_SHORT).show();  }  @Override  public void onReschedule(String requestId, ErrorInfo error) {}  });  } else {  // Nếu đã có `imageUrl`, tiến hành đăng ký  proceedToRegister(name, brand, description, gearType, seats, fuelType, energyConsumption, driverRequired, location, price, selectedFeatures, imageUrl, token);  }  } |
| --- |

| private void proceedToRegister(String name, String brand, String description, String gearType, int seats,  String fuelType, double energyConsumption, boolean driverRequired,  String location, double price,  List<String> features, String imageUrl, String token) {  AddCarRequest request = new AddCarRequest(  features,  price,  location,  driverRequired,  imageUrl,  energyConsumption,  fuelType,  seats,  gearType,  description,  brand,  name  );  apiService.addCar(request, token).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful() && response.body() != null) {  Toast.makeText(AddCarActivity.this, "Đăng ký xe thành công", Toast.LENGTH\_SHORT).show();  finish();  } else {  Toast.makeText(AddCarActivity.this, "Đăng ký xe thất bại", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(AddCarActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: /api/user/add-car

Controller nhận AddCarRequest và gọi phương thức addCar() trong CarService.java.

| /\*\*  \* API: POST /api/user/add-car  \* Thêm xe mới vào hệ thống  \*/  @PostMapping("/add-car")  public ResponseEntity<ApiResponse<String>> addCar(@RequestBody @Valid AddCarRequest request, Authentication authentication) {  try {  carService.addCar(request, authentication);  return ResponseEntity.ok(new ApiResponse<>(true, "Thêm xe thành công", null));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi thêm xe", null));  }  } |
| --- |

* Backend - CarService.java:

Kiểm tra authentication để xác định người dùng.

Tạo đối tượng Car mới từ AddCarRequest và thông tin người dùng.

Gọi phương thức carRepository.save() để lưu thông tin xe vào DB.

| public void addCar(AddCarRequest request, Authentication authentication) {  String username = authentication.getName();  User owner = userRepository.findByAccount\_Username(username)  .orElseThrow(() -> new ResourceNotFoundException("User not found"));  Car car = new Car(  owner,  request.getName(),  request.getBrand(),  request.getDescription(),  request.getGearType(),  request.getSeats(),  request.getFuelType(),  request.getEnergyConsumption(),  request.getHasEtc(),  request.getLocation(),  request.getPrice(),  request.getImageUrl(),  "Available",  request.isDriverRequired(),  LocalDateTime.now().toString()  );  carRepository.save(car);  } |
| --- |

### *Xem các xe đã đăng ký*

* Luồng từ Android:

Người dùng mở màn hình ProfileFragment và chọn registerCarLayout, ứng dụng thực hiện:

Gọi phương thức apiService.getMyCars() trong ApiService.java.

API trả về List<CarResponse>, danh sách này được hiển thị thông qua CarAdapter.

Khi người dùng click vào 1 xe, chuyển đến CarDetailActivity và gửi carId qua Intent.

| private void showMyCarsDialog() {  BottomSheetDialog dialog = new BottomSheetDialog(requireContext());  View dialogView = LayoutInflater.from(requireContext()).inflate(R.layout.dialog\_my\_cars, null);  RecyclerView rvMyCars = dialogView.findViewById(R.id.rv\_my\_cars);  rvMyCars.setLayoutManager(new LinearLayoutManager(requireContext()));  String token = "Bearer " + SharedPrefManager.getInstance(requireContext()).getToken();  apiService.getMyCars(token).enqueue(new Callback<ApiResponse<List<CarResponse>>>() {  @Override  public void onResponse(Call<ApiResponse<List<CarResponse>>> call, Response<ApiResponse<List<CarResponse>>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<List<CarResponse>> apiResponse = response.body();  if (apiResponse.isSuccess()) {  List<CarResponse> carList = apiResponse.getData();  MyCarsAdapter adapter = new MyCarsAdapter(carList, car -> {  Log.e("Đã chọn xe có id", "ID: " + car.getCarId());  Intent intent = new Intent(requireContext(), EditCarActivity.class);  intent.putExtra("carId", car.getCarId().longValue()); // Chuyển sang Long  startActivity(intent);  dialog.dismiss();  });  rvMyCars.setAdapter(adapter);  } else {  Toast.makeText(requireContext(), apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  }  } else {  Toast.makeText(requireContext(), "Không tải được danh sách xe", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<List<CarResponse>>> call, Throwable t) {  Toast.makeText(requireContext(), "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  dialog.setContentView(dialogView);  dialog.show();  } |
| --- |

* Backend - UserController.java:

Endpoint: /api/user/my-cars

Controller gọi getAllCarsByOwner() trong CarService.java để lấy danh sách xe.

| /\*\*  \* API: GET /api/user/my-cars  \* Lấy danh sách xe đã đăng của người dùng hiện tại  \*/  @GetMapping("/my-cars")  public ResponseEntity<ApiResponse<List<CarResponse>>> getMyCars(Authentication authentication) {  try {  List<CarResponse> cars = carService.getAllCarsByOwner(authentication);  return ResponseEntity.ok(new ApiResponse<>(true, "Lấy danh sách xe thành công", cars));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi lấy danh sách xe", null));  }  } |
| --- |

* Backend - CarService.java:

Lấy thông tin user từ authentication.

Sử dụng carRepository.findByOwner() để lấy danh sách xe thuộc người dùng.

Chuyển đổi mỗi Car thành CarResponse để trả về cho Android.

| public List<CarResponse> getAllCarsByOwner(Authentication authentication) {  String username = authentication.getName();  User owner = userRepository.findByAccount\_Username(username)  .orElseThrow(() -> new ResourceNotFoundException("User not found"));  List<Car> cars = carRepository.findByOwner(owner);  return cars.stream().map(car -> new CarResponse(  car.getCarId(),  car.getName(),  car.getBrand(),  car.getDescription(),  car.getGearType(),  car.getSeats(),  car.getFuelType(),  car.getEnergyConsumption(),  car.getHasEtc(),  car.getLocation(),  car.getPrice(),  car.getImageUrl(),  car.getStatus(),  car.isDriverRequired(),  car.getCreatedAt()  )).collect(Collectors.toList());  } |
| --- |

### *Chỉnh sửa thông tin xe*

* Luồng từ Android:

Người dùng mở EditCarActivity, ứng dụng thực hiện:

Hiển thị thông tin xe đã đăng ký từ CarResponse.

Người dùng chỉnh sửa thông tin và nhấn Cập nhật.

Tạo EditCarRequest và gọi apiService.editCar(carId, request).

Nếu ảnh xe thay đổi, ứng dụng thực hiện upload lại lên Cloudinary và lấy URL mới.

| private void updateCar() {  String name = edtCarName.getText().toString().trim();  String brand = edtBrand.getText().toString().trim();  String description = edtDescription.getText().toString().trim();  double price = Double.parseDouble(edtPrice.getText().toString().trim());  double energyConsumption = Double.parseDouble(edtEnergyConsumption.getText().toString().trim());  boolean driverRequired = cbDriverRequired.isChecked();  String seats = spnSeats.getText().toString().trim();  String gearType = spnGearType.getSelectedItem().toString();  String fuelType = spnFuelType.getSelectedItem().toString();  String location = edtDetailLocation.getText().toString().trim() + ", " + spnLocation.getSelectedItem().toString();  List<String> selectedFeatures = new ArrayList<>();  for (Feature feature : featureList) {  if (feature.isSelected()) {  selectedFeatures.add(feature.getName());  }  }  if (selectedImageUri != null) {  CloudinaryManager.uploadImage(this, selectedImageUri, new UploadCallback() {  @Override  public void onStart(String requestId) {  Toast.makeText(EditCarActivity.this, "Bắt đầu upload ảnh...", Toast.LENGTH\_SHORT).show();  }  @Override  public void onProgress(String requestId, long bytes, long totalBytes) {  Log.d("EditCarActivity", "Uploading: " + bytes + "/" + totalBytes);  }  @Override  public void onSuccess(String requestId, Map resultData) {  String newImageUrl = (String) resultData.get("secure\_url");  sendUpdateRequest(selectedFeatures, name, brand, description, price, energyConsumption, driverRequired, newImageUrl, seats, gearType, fuelType, location);  }  @Override  public void onError(String requestId, ErrorInfo error) {  Toast.makeText(EditCarActivity.this, "Upload ảnh thất bại: " + error.getDescription(), Toast.LENGTH\_SHORT).show();  }  @Override  public void onReschedule(String requestId, ErrorInfo error) {  Toast.makeText(EditCarActivity.this, "Upload bị tạm hoãn: " + error.getDescription(), Toast.LENGTH\_SHORT).show();  }  });  } else {  sendUpdateRequest(selectedFeatures, name, brand, description, price, energyConsumption, driverRequired, imageUrl, seats, gearType, fuelType, location);  }  } |
| --- |

| private void sendUpdateRequest(List<String> features, String name, String brand, String description, double price, double energyConsumption, boolean driverRequired, String imageUrl, String seats, String gearType, String fuelType, String location) {  String token = "Bearer " + SharedPrefManager.getInstance(this).getToken();  EditCarRequest request = new EditCarRequest(  features, price, location, driverRequired, imageUrl, energyConsumption, fuelType, Integer.parseInt(seats), gearType, description, brand, name  );  apiService.updateCar(carId, request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful() && response.body() != null) {  Toast.makeText(EditCarActivity.this, "Cập nhật thành công", Toast.LENGTH\_SHORT).show();  finish();  } else {  Toast.makeText(EditCarActivity.this, "Cập nhật thất bại", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(EditCarActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: /api/user/edit-car/{carId}

Controller nhận EditCarRequest và gọi editCar() trong CarService.java.

| @PutMapping("/edit-car/{carId}")  public ResponseEntity<ApiResponse<String>> editCar(  @PathVariable Long carId,  @RequestBody @Valid EditCarRequest request,  Authentication authentication) {  try {  carService.editCar(carId, request, authentication);  return ResponseEntity.ok(new ApiResponse<>(true, "Cập nhật xe thành công", null));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi cập nhật xe", null));  }  } |
| --- |

* Backend - CarService.java:

Kiểm tra quyền sở hữu xe dựa trên carId và userId.

Nếu người dùng là chủ xe, cập nhật các thông tin xe và upload lại ảnh nếu có thay đổi.

Gọi carRepository.save() để lưu các thay đổi vào DB.

| @Transactional  public void editCar(Long carId, EditCarRequest request, Authentication authentication) {  String username = authentication.getName();  User owner = userRepository.findByAccount\_Username(username)  .orElseThrow(() -> new RuntimeException("Người dùng không tồn tại"));  Car car = carRepository.findById(carId)  .orElseThrow(() -> new RuntimeException("Xe không tồn tại"));  if (!car.getOwner().equals(owner)) {  throw new RuntimeException("Bạn không có quyền chỉnh sửa xe này");  }  car.setName(request.getName());  car.setBrand(request.getBrand());  car.setDescription(request.getDescription());  car.setGearType(request.getGearType());  car.setSeats(request.getSeats());  car.setFuelType(request.getFuelType());  car.setEnergyConsumption(request.getEnergyConsumption());  car.setLocation(request.getLocation());  car.setPrice(request.getPrice());  car.setImageUrl(request.getImageUrl());  car.setDriverRequired(request.isDriverRequired());  // Cập nhật tính năng (hasEtc)  car.getHasEtc().clear();  if (request.getHasEtc() != null) {  car.getHasEtc().addAll(request.getHasEtc());  }  carRepository.save(car);  } |
| --- |

## 

## **Quản lý tài khoản và thông tin cá nhân**

### Xem *thông tin tài khoản*

* Luồng từ Android:

Người dùng mở màn hình ProfileFragment. Ứng dụng tự động gọi loadUserProfile() để tải thông tin người dùng.

Tạo đối tượng ApiService và gọi phương thức apiService.getUserProfile(token) để lấy thông tin người dùng hiện tại.

API trả về ApiResponse, dữ liệu này sẽ được hiển thị lên các TextView như tvUsername, tvPhone.

Nếu thông tin chưa được cập nhật (ví dụ: chưa có số điện thoại), các trường sẽ hiển thị thông báo như "Chưa liên kết số điện thoại".

| private void loadUserProfile() {  String token = "Bearer " + SharedPrefManager.getInstance(requireContext()).getToken();  apiService.getUserProfile(token).enqueue(new Callback<ApiResponse<UserProfileResponse>>() {  @Override  public void onResponse(Call<ApiResponse<UserProfileResponse>> call, Response<ApiResponse<UserProfileResponse>> response) {  if (response.isSuccessful() && response.body() != null) {  ApiResponse<UserProfileResponse> apiResponse = response.body();  if (apiResponse.isSuccess()) {  UserProfileResponse profile = apiResponse.getData();  tvUsername.setText(profile.getName());  String phone = profile.getPhone();  if (phone == null || phone.trim().isEmpty()) {  tvPhone.setText("Chưa liên kết số điện thoại");  } else {  // Đổi 0377179132 → +84 377179132  if (phone.startsWith("0") && phone.length() > 1) {  phone = "+84 " + phone.substring(1);  }  tvPhone.setText(phone);  }  } else {  Toast.makeText(requireContext(), apiResponse.getMessage(), Toast.LENGTH\_SHORT).show();  }  } else {  Toast.makeText(requireContext(), "Không lấy được thông tin người dùng", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<UserProfileResponse>> call, Throwable t) {  Toast.makeText(requireContext(), "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: /api/user/profile

Controller nhận yêu cầu từ client và gọi phương thức getProfile() trong UserService.java để lấy thông tin người dùng.

| /\*\*  \* API: GET /api/user/profile  \* Lấy thông tin người dùng hiện tại (username từ token)  \*/  @GetMapping("/profile")  public ResponseEntity<ApiResponse<UserProfileResponse>> getProfile(Authentication authentication) {  try {  UserProfileResponse profile = userService.getProfile(authentication);  return ResponseEntity.ok(new ApiResponse<>(true, "Lấy thông tin người dùng thành công", profile));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi lấy thông tin người dùng", null));  }  } |
| --- |

* Backend - UserService.java:

Xác định username từ Authentication (JWT Token).

Lấy thông tin User dựa trên username từ UserRepository.

Tạo đối tượng UserProfileResponse và trả về cho client.

| /\*\*  \* Trả về profile cơ bản của người dùng đang login  \*/  public UserProfileResponse getProfile(Authentication authentication) {  String username = authentication.getName();    Account account = accountRepo.findByUsername(username)  .orElseThrow(() -> new ResourceNotFoundException("Không tìm thấy tài khoản"));    User user = userRepo.findByAccount\_Username(username)  .orElseThrow(() -> new ResourceNotFoundException("Không tìm thấy thông tin người dùng"));    return new UserProfileResponse(  user.getUserId(),  user.getName(),  account.getEmail(),  user.getAddress(),  user.getPhone(),  user.getGender(),  user.getBirthdate() != null ? user.getBirthdate().toString() : null,  user.getImageUrl(),  user.getRole(),  user.getCreatedAt() != null ? user.getCreatedAt().toString() : null  );  } |
| --- |

### *Chỉnh sửa thông tin cá nhân*

* Luồng từ Android:

Người dùng mở màn hình AccountActivity và nhấn vào nút "Chỉnh sửa".

Ứng dụng hiển thị các trường EditText gồm: Tên, Số điện thoại, Ngày sinh, Giới tính, Địa chỉ.

Người dùng nhập thông tin và nhấn "Lưu". Ứng dụng thực hiện:

Tạo đối tượng EditProfileRequest với các thông tin đã nhập.

Gọi phương thức apiService.editProfile(request, token) trong ApiService.java.

Nếu cập nhật thành công, API trả về thông báo "Cập nhật thành công" và gọi lại loadUserProfile() để cập nhật giao diện.

| private void showEditProfileDialog(String name, String phone, String birthdate, String gender, String address) {  BottomSheetDialog dialog = new BottomSheetDialog(this);  View view = LayoutInflater.from(this).inflate(R.layout.dialog\_edit\_profile, null);  EditText edtName = view.findViewById(R.id.edt\_name);  EditText edtPhone = view.findViewById(R.id.edt\_phone);  EditText edtBirthdate = view.findViewById(R.id.edt\_birthdate);  EditText edtGender = view.findViewById(R.id.edt\_gender);  EditText edtAddress = view.findViewById(R.id.edt\_address);  Button btnSave = view.findViewById(R.id.btn\_save);  edtName.setText(name);  edtPhone.setText(phone);  edtBirthdate.setText(birthdate); // Dữ liệu dạng "yyyy-MM-dd"  edtGender.setText(gender);  edtAddress.setText(address);  btnSave.setOnClickListener(v -> {  String newName = edtName.getText().toString().trim();  String newPhone = edtPhone.getText().toString().trim();  String newBirthdate = edtBirthdate.getText().toString().trim();  String newGender = edtGender.getText().toString().trim();  String newAddress = edtAddress.getText().toString().trim();  if (newName.isEmpty() || newPhone.isEmpty() || newBirthdate.isEmpty() || newAddress.isEmpty()) {  Toast.makeText(this, "Vui lòng điền đủ thông tin", Toast.LENGTH\_SHORT).show();  return;  }  if (!newBirthdate.matches("\\d{4}-\\d{2}-\\d{2}")) {  Toast.makeText(this, "Ngày sinh phải có định dạng yyyy-MM-dd", Toast.LENGTH\_SHORT).show();  return;  }  EditProfileRequest request = new EditProfileRequest(newName, newPhone, newBirthdate, newGender, newAddress);  updateProfile(request);  dialog.dismiss();  });  dialog.setContentView(view);  dialog.show();  } |
| --- |

| private void updateProfile(EditProfileRequest request) {  String token = "Bearer " + SharedPrefManager.getInstance(this).getToken();  Gson gson = new Gson();  String jsonRequest = gson.toJson(request);  Log.d("JSON Request", jsonRequest);  apiService.editProfile(request, token).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful()) {  Toast.makeText(AccountActivity.this, "Cập nhật thành công", Toast.LENGTH\_SHORT).show();  loadUserProfile(); // Refresh dữ liệu sau khi cập nhật  } else {  Toast.makeText(AccountActivity.this, "Cập nhật thất bại", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(AccountActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: /api/user/edit-profile

Controller nhận EditProfileRequest và gọi phương thức editProfile() trong UserService.java để xử lý yêu cầu.

| /\*\*  \* API: POST /api/user/edit-profile  \* Chỉnh sửa thông tin cá nhân  \*/  @PostMapping("/edit-profile")  public ResponseEntity<ApiResponse<String>> editProfile(@RequestBody @Valid EditProfileRequest request, Authentication authentication) {  try {  userService.editProfile(request, authentication);  return ResponseEntity.ok(new ApiResponse<>(true, "Cập nhật thông tin thành công", null));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi cập nhật thông tin", null));  }  } |
| --- |

* Backend - UserService.java:

Kiểm tra quyền truy cập thông qua JWT Token.

Tìm kiếm User bằng username và cập nhật các trường có giá trị trong EditProfileRequest.

Lưu thông tin người dùng vào DB thông qua UserRepository.

| public void editProfile(EditProfileRequest request, Authentication authentication) {  String username = authentication.getName();  User user = userRepo.findByAccount\_Username(username)  .orElseThrow(() -> new ResourceNotFoundException("Không tìm thấy người dùng"));  if (request.getName() != null && !request.getName().isEmpty()) {  user.setName(request.getName());  }  if (request.getPhone() != null && !request.getPhone().isEmpty()) {  user.setPhone(request.getPhone());  }  if (request.getBirthdate() != null) {  user.setBirthdate(request.getBirthdate());  }  if (request.getGender() != null && !request.getGender().isEmpty()) {  user.setGender(request.getGender());  }  if (request.getAddress() != null && !request.getAddress().isEmpty()) {  user.setAddress(request.getAddress());  }  userRepo.save(user);  } |
| --- |

## **Tìm kiếm, lọc và xem danh sách xe**

### *Tìm kiếm xe*

* Luồng từ Android:

Người dùng vào CarListActivity từ HomeFragment.

Người dùng chọn quận, bật/tắt yêu cầu tài xế, và chọn các bộ lọc khác (số chỗ ngồi, giá thuê, loại hộp số, loại nhiên liệu).

Tạo đối tượng SearchCarRequest từ dữ liệu đã chọn.

Gọi API /api/user/cars/list thông qua phương thức apiService.searchCars(request).

API trả về danh sách xe (List<CarListResponse>), hiển thị qua CarAdapter trong RecyclerView.

| private void setupListeners() {  btnSelfDrive.setOnClickListener(v -> {  viewFlipper.setDisplayedChild(0);  setActiveTab(btnSelfDrive, btnWithDriver);  driverRequired = false;  });  btnWithDriver.setOnClickListener(v -> {  viewFlipper.setDisplayedChild(1);  setActiveTab(btnWithDriver, btnSelfDrive);  driverRequired = true;  });  iconHeart.setOnClickListener(v -> startActivity(new Intent(getActivity(), FavoritesActivity.class)));  iconNotify.setOnClickListener(v -> startActivity(new Intent(getActivity(), NotifyActivity.class)));  requireView().findViewById(R.id.btnFindCar).setOnClickListener(v -> {  Intent intent = new Intent(getActivity(), CarListActivity.class);  intent.putExtra("location", tvLocation.getText().toString());  intent.putExtra("driverRequired", driverRequired);  startActivity(intent);  });  } |
| --- |

| private void searchCars(SearchCarRequest request) {  showLoading(true);  Gson gson = new Gson();  Log.d("SearchCarRequest", gson.toJson(request));  apiService.searchCars(request).enqueue(new Callback<ApiResponse<List<CarListResponse>>>() {  @Override  public void onResponse(Call<ApiResponse<List<CarListResponse>>> call, Response<ApiResponse<List<CarListResponse>>> response) {  showLoading(false);  if (response.isSuccessful() && response.body() != null && response.body().isSuccess()) {  carList.clear();  carList.addAll(response.body().getData());  carAdapter.notifyDataSetChanged();  } else {  carList.clear();  Toast.makeText(CarListActivity.this, "Không tìm thấy xe phù hợp", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<List<CarListResponse>>> call, Throwable t) {  showLoading(false);  Toast.makeText(CarListActivity.this, "Lỗi mạng hoặc server!", Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: POST /api/user/cars/list

Controller nhận SearchCarRequest và gọi phương thức searchCars() trong CarService.java.

| /\*\*  \* API: POST /api/user/cars/list  \* Tìm kiếm xe theo điều kiện  \*/  @PostMapping("/cars/list")  public ResponseEntity<ApiResponse<List<CarListResponse>>> searchCars(@RequestBody SearchCarRequest request, Authentication authentication) {  try {  List<Car> cars = carService.searchCars(request);  if (cars.isEmpty()) {  return ResponseEntity.ok(new ApiResponse<>(false, "Không tìm thấy xe nào", null));  }  List<CarListResponse> response = cars.stream()  .map(car -> {  Double avgRating = carService.getAvgRating(car.getCarId());  Long tripCount = carService.getTripCount(car.getCarId());  return new CarListResponse(car, avgRating != null ? avgRating : 0.0, tripCount != null ? tripCount : 0L);  })  .toList();  return ResponseEntity.ok(new ApiResponse<>(true, "Tìm kiếm thành công", response));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi tìm kiếm xe", null));  }  } |
| --- |

* Backend - CarService.java:

Phương thức searchCars() nhận SearchCarRequest và chuyển dữ liệu thành các tham số cho CarRepository.java.

Thực hiện truy vấn dựa trên location, seats, brand, priceFrom, priceTo, gearType, fuelType, driverRequired.

Trả về List<Car> và chuyển đổi thành List<CarListResponse>.

Lấy thêm avgRating và tripCount cho mỗi xe.

| public List<Car> searchCars(SearchCarRequest searchCarRequest) {  String location = (searchCarRequest.getLocation() != null && !searchCarRequest.getLocation().isEmpty())  ? searchCarRequest.getLocation()  : null;  Integer seats = searchCarRequest.getSeats();  String brand = (searchCarRequest.getBrand() != null && !searchCarRequest.getBrand().isEmpty())  ? searchCarRequest.getBrand()  : null;  Double priceFrom = searchCarRequest.getPriceFrom();  Double priceTo = searchCarRequest.getPriceTo();  String gearType = (searchCarRequest.getGearType() != null && !searchCarRequest.getGearType().isEmpty())  ? searchCarRequest.getGearType()  : null;  String fuelType = (searchCarRequest.getFuelType() != null && !searchCarRequest.getFuelType().isEmpty())  ? searchCarRequest.getFuelType()  : null;  Boolean driverRequired = searchCarRequest.getDriverRequired();  return carRepository.searchCars(  location,  seats,  brand,  priceFrom,  priceTo,  gearType,  fuelType,  driverRequired,  "Available"  );  } |
| --- |

* Backend - CarRepository.java:

Truy vấn searchCars() thực hiện tìm kiếm xe với các điều kiện lọc.

Sử dụng các tham số từ SearchCarRequest để xây dựng câu truy vấn động.

| List<Car> searchCars(@Param("location") String location,  @Param("seats") Integer seats,  @Param("brand") String brand,  @Param("priceFrom") Double priceFrom,  @Param("priceTo") Double priceTo,  @Param("gearType") String gearType,  @Param("fuelType") String fuelType,  @Param("driverRequired") Boolean driverRequired,  @Param("status") String status); |
| --- |

### *Lọc xe theo yêu cầu*

* Luồng từ Android:

Người dùng nhấn nút lọc (btnFilter) trong CarListActivity.

FilterBottomSheet hiển thị và cho phép người dùng chọn các tiêu chí lọc (số chỗ ngồi, loại nhiên liệu, hộp số, giá thuê).

Sau khi nhấn nút áp dụng lọc, dữ liệu sẽ được đóng gói vào SearchCarRequest và gửi lại CarListActivity.

| private void openFilterBottomSheet() {  FilterBottomSheet filterBottomSheet = new FilterBottomSheet();  filterBottomSheet.setFilterListener(request -> {  String selectedDistrict = actvDistrict.getText().toString();  request.setLocation(selectedDistrict);  request.setDriverRequired(driverRequired);  searchCars(request);  });  filterBottomSheet.show(getSupportFragmentManager(), filterBottomSheet.getTag());  } |
| --- |

CarListActivity gửi yêu cầu API /api/user/cars/list với các tiêu chí lọc mới.

Danh sách xe cập nhật và hiển thị qua CarAdapter.

* Backend - CarService.java:

searchCars() cập nhật query để áp dụng các tiêu chí lọc từ SearchCarRequest.

Sử dụng điều kiện động cho từng tiêu chí: location, seats, brand, priceFrom, priceTo, gearType, fuelType, driverRequired.

| public List<Car> searchCars(SearchCarRequest searchCarRequest) {  String location = (searchCarRequest.getLocation() != null && !searchCarRequest.getLocation().isEmpty())  ? searchCarRequest.getLocation()  : null;  Integer seats = searchCarRequest.getSeats();  String brand = (searchCarRequest.getBrand() != null && !searchCarRequest.getBrand().isEmpty())  ? searchCarRequest.getBrand()  : null;  Double priceFrom = searchCarRequest.getPriceFrom();  Double priceTo = searchCarRequest.getPriceTo();  String gearType = (searchCarRequest.getGearType() != null && !searchCarRequest.getGearType().isEmpty())  ? searchCarRequest.getGearType()  : null;  String fuelType = (searchCarRequest.getFuelType() != null && !searchCarRequest.getFuelType().isEmpty())  ? searchCarRequest.getFuelType()  : null;  Boolean driverRequired = searchCarRequest.getDriverRequired();  return carRepository.searchCars(  location,  seats,  brand,  priceFrom,  priceTo,  gearType,  fuelType,  driverRequired,  "Available"  );  } |
| --- |

* Backend - CarRepository.java:

Phương thức searchCars() xử lý logic lọc động thông qua câu lệnh @Query.

| List<Car> searchCars(@Param("location") String location,  @Param("seats") Integer seats,  @Param("brand") String brand,  @Param("priceFrom") Double priceFrom,  @Param("priceTo") Double priceTo,  @Param("gearType") String gearType,  @Param("fuelType") String fuelType,  @Param("driverRequired") Boolean driverRequired,  @Param("status") String status); |
| --- |

## **Xem chi tiết thông tin xe, xem đánh giá xe**

### *Xem chi tiết thông tin xe*

* Luồng từ Android:

Người dùng nhấn vào một xe trong CarListActivity.

CarDetailActivity sẽ được mở và gửi carId qua Intent.

| @Override  public void onBindViewHolder(@NonNull CarViewHolder holder, int position) {  CarListResponse car = carList.get(position);  Glide.with(context).load(car.getImageUrl()).into(holder.imgCar);  holder.txtName.setText(car.getName());  holder.txtSeats.setText(car.getSeats() + " chỗ");  holder.txtGear.setText(car.getGearType().equals("Manual") ? "Số sàn" : "Tự động");  holder.txtFuel.setText(car.getFuelType().equals("Diesel") ? "Dầu diesel" : car.getFuelType());  holder.txtLocation.setText(car.getLocation());  holder.txtPrice.setText(String.format("%.0fK/ngày", car.getPrice()));  holder.txtRating.setText(String.format("%.1f ★", car.getAvgRating() != null ? car.getAvgRating() : 5.0));  holder.txtTrips.setText(String.format("%d chuyến", car.getTripCount() != null ? car.getTripCount() : 0));  holder.itemView.setOnClickListener(v -> {  Intent intent = new Intent(context, CarDetailActivity.class);  intent.putExtra("carId", car.getCarId());  context.startActivity(intent);  });  } |
| --- |

Gọi API /api/user/car/{id} thông qua phương thức apiService.getCarDetail(carId).

API trả về CarDetailResponse bao gồm thông tin chi tiết xe, đánh giá xe và thông tin chủ xe.

CarDetailActivity nhận dữ liệu và cập nhật giao diện thông qua các TextView, ImageView và RecyclerView để hiển thị thông tin xe và danh sách đánh giá.

| public void getCarDetail(Context context, Long carId) {  isLoading.setValue(true);  ApiService apiService = RetrofitClient.getRetrofit(context).create(ApiService.class);  apiService.getCarDetail(carId).enqueue(new Callback<ApiResponse<CarDetailResponse>>() {  @Override  public void onResponse(Call<ApiResponse<CarDetailResponse>> call, Response<ApiResponse<CarDetailResponse>> response) {  isLoading.setValue(false);  if (response.isSuccessful() && response.body() != null) {  ApiResponse<CarDetailResponse> apiResponse = response.body();  Log.d("CarViewModel", "API Response Success: " + new Gson().toJson(apiResponse));  Log.d("CarViewModel", "API Response JSON: " + new Gson().toJson(response.body()));  if (response.body().isSuccess()) {  carDetailLiveData.setValue(response.body().getData());  } else {  errorMessage.setValue("API Error: " + apiResponse.getMessage());  Log.e("CarViewModel", "API Error Message: " + apiResponse.getMessage());  }  } else {  errorMessage.setValue("Lỗi khi tải dữ liệu");  Log.e("CarViewModel", "API Error: " + response.message());  }  }  @Override  public void onFailure(Call<ApiResponse<CarDetailResponse>> call, Throwable t) {  isLoading.setValue(false);  errorMessage.setValue("Lỗi kết nối");  Log.e("CarViewModel", t.getMessage());  }  });  } |
| --- |

* Backend - UserController.java:

Endpoint: GET /api/user/car/{id}

Controller nhận carId và gọi phương thức getCarDetail() trong CarService.java để lấy thông tin chi tiết xe.

| /\*\*  \* API: GET /api/user/car/{id}  \* Lấy chi tiết xe  \*/  @GetMapping("/car/{id}")  public ResponseEntity<ApiResponse<CarDetailResponse>> getCarDetail(@PathVariable Long id) {  CarDetailResponse carDetail = carService.getCarDetail(id);  if (carDetail == null) {  ApiResponse<CarDetailResponse> response = new ApiResponse<>(false, "Không tìm thấy xe với ID: " + id);  return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(response);  }  ApiResponse<CarDetailResponse> response = new ApiResponse<>(true, "Lấy chi tiết xe thành công", carDetail);  return ResponseEntity.ok(response);  } |
| --- |

* Backend - CarService.java:

Phương thức getCarDetail() nhận carId và thực hiện:

Lấy thông tin xe từ CarRepository.

Lấy thông tin trung bình đánh giá (avgRating), số chuyến (tripCount) từ ReviewRepository và BookingRepository.

Lấy thông tin đánh giá từ ReviewRepository.

Tạo CarDetailResponse với thông tin xe, đánh giá và thông tin chủ xe và trả về cho Android.

| public CarDetailResponse getCarDetail(Long carId) {  Car car = carRepository.findById(carId)  .orElseThrow(() -> new RuntimeException("Car not found"));  // Đánh giá & chuyến của xe  Double avgRating = reviewRepository.findAverageRatingByCarId(carId);  Long tripCount = bookingRepository.countCompletedTripsByCarId(carId);  // Thống kê của chủ xe  Long ownerId = car.getOwner().getUserId();  Double ownerAvgRating = reviewRepository.findAverageRatingByOwnerId(ownerId);  Long ownerTripCount = bookingRepository.countCompletedTripsByOwnerId(ownerId);  // Danh sách đánh giá  List<ReviewDTO> reviews = reviewRepository.findByCar\_CarId\_dto(carId)  .stream()  .map(r -> new ReviewDTO(  r.getReviewId(),  carId,  r.getRating(),  r.getComment(),  r.getName(),  r.getCreatedAt(),  r.getImageUrl()  ))  .collect(Collectors.toList());    // Trả về DTO  CarDetailResponse response = new CarDetailResponse();  response.setId(car.getCarId());  response.setName(car.getName());  response.setBrand(car.getBrand());  response.setDescription(car.getDescription());  response.setGearType(car.getGearType());  response.setSeats(car.getSeats());  response.setFuelType(car.getFuelType());  response.setEnergyConsumption(car.getEnergyConsumption());  response.setHasEtc(car.getHasEtc());  response.setLocation(car.getLocation());  response.setPrice(car.getPrice());  response.setImageUrl(car.getImageUrl());  response.setDriverRequired(car.isDriverRequired());  response.setStatus(car.getStatus());  response.setDriverRequired(car.isDriverRequired());  response.setCreatedAt(car.getCreatedAt());  response.setAvgRating(avgRating != null ? avgRating : 0.0);  response.setTripCount(tripCount != null ? tripCount : 0L);  response.setOwnerName(car.getOwner().getName());  response.setOwnerImage(car.getOwner().getImageUrl());  response.setOwnerAvgRating(ownerAvgRating != null ? ownerAvgRating : 0.0);  response.setOwnerTripCount(ownerTripCount != null ? ownerTripCount : 0L);  response.setReviews(reviews);  return response;  } |
| --- |

### *Xem đánh giá xe*

* Luồng từ Android:

Người dùng truy cập vào CarDetailActivity và kéo xuống phần đánh giá.

CarDetailActivity sẽ hiển thị danh sách đánh giá thông qua RecyclerView.

Khi CarDetailActivity được mở, phương thức getCarDetail(carId) sẽ lấy về danh sách đánh giá từ API.

Dữ liệu đánh giá sẽ được đổ vào Adapter để hiển thị trên giao diện.

* Backend - UserController.java:

Endpoint: GET /api/user/review/car/{carId}

Controller nhận carId và gọi phương thức getReviewsByCarId() trong ReviewService.java để lấy danh sách đánh giá.

| @GetMapping("/review/car/{carId}")  public ResponseEntity<ApiResponse<List<Review>>> getReviewsByCarId(@PathVariable Long carId) {  try {  List<Review> reviews = reviewService.getReviewsByCarId(carId);  if (reviews.isEmpty()) {  return ResponseEntity.ok(new ApiResponse<>(false, "Không có review cho xe này", null));  }  return ResponseEntity.ok(new ApiResponse<>(true, "Lấy danh sách review thành công", reviews));  } catch (Exception e) {  e.printStackTrace();  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi lấy danh sách review", null));  }  } |
| --- |

* Backend - ReviewService.java:

Phương thức getReviewsByCarId() nhận carId và thực hiện truy vấn lấy danh sách ReviewDTO từ ReviewRepository.

Chuyển dữ liệu đánh giá thành ReviewDTO và trả về cho Android dưới dạng List.

| public List<Review> getReviewsByCarId(Long carId) {  return reviewRepository.findByCar\_CarId(carId);  } |
| --- |

| List<Review> findByCar\_CarId(Long carId); |
| --- |

## **Tạo đơn đặt xe, xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe**

### *Tạo đơn đặt xe*

* Luồng từ Android:

Người dùng mở màn hình BookingActivity.

BookingActivity nhận dữ liệu từ Intent: thông tin xe (CarDetailResponse), thời gian thuê (startDate, endDate), địa điểm nhận/trả xe (pickupLocation, dropoffLocation), và các tùy chọn khác (insuranceSelected, deliverySelected, driverRequired).

Khi người dùng nhấn nút Thuê xe (btnRentCar), ứng dụng thực hiện phương thức sendBookingConfirmRequest().

| private void sendBookingConfirmRequest() {  String startDateFormatted = convertToApiFormat(startDate);  String endDateFormatted = convertToApiFormat(endDate);  if (startDateFormatted == null || endDateFormatted == null) {  Toast.makeText(this, "Lỗi định dạng thời gian", Toast.LENGTH\_SHORT).show();  return;  }  BookingPreviewRequest bookingRequest = new BookingPreviewRequest(  carDetail.getId(),  startDateFormatted,  endDateFormatted,  pickupLocation,  dropoffLocation,  insuranceSelected,  deliverySelected,  driverRequired  );  apiService.confirmBooking(bookingRequest).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful() && response.body() != null) {  Toast.makeText(BookingActivity.this, "Đặt xe thành công!", Toast.LENGTH\_SHORT).show();  finish();  } else {  Toast.makeText(BookingActivity.this, "Không thể xác nhận đặt xe.", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(BookingActivity.this, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - UserController.java

Endpoint: POST /api/user/booking/confirm

Nhận BookingRequest từ Android và xác thực người dùng qua Authentication.

Gọi phương thức confirmBooking() trong BookingService.java.

| @PostMapping("/booking/confirm")  public ResponseEntity<ApiResponse<String>> confirmBooking(  @RequestBody BookingRequest request,  Authentication authentication) {  if (authentication == null || authentication.getPrincipal() == null) {  return ResponseEntity.status(HttpStatus.UNAUTHORIZED)  .body(new ApiResponse<>(false, "Chưa xác thực người dùng", null));  }  String username = authentication.getPrincipal().toString();  try {  User user = userRepository.findByAccount\_Username(username)  .orElseThrow(() -> new RuntimeException("Không tìm thấy người dùng"));  Long bookingId = bookingService.confirmBooking(request, user.getUserId());  return ResponseEntity.ok(new ApiResponse<>(true, "Đặt xe thành công", bookingId.toString()));  } catch (Exception e) {  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Đã xảy ra lỗi khi xác nhận đặt xe", null));  }  } |
| --- |

* Backend - BookingService.java:

Phương thức xác nhận đặt xe (confirmBooking):

* Kiểm tra xe (carId) và người dùng (userId) trong DB.
* Tính toán số ngày thuê xe và chi phí.
* Tạo Booking mới và lưu vào DB.

| @Transactional  public Long confirmBooking(BookingRequest request, Long userId) {  Car car = carRepository.findById(request.getCarId())  .orElseThrow(() -> new RuntimeException("Không tìm thấy xe"));  User user = userRepository.findById(userId)  .orElseThrow(() -> new RuntimeException("Không tìm thấy người dùng"));  LocalDateTime start = request.getStartDate();  LocalDateTime end = request.getEndDate();  if (!end.isAfter(start)) {  throw new RuntimeException("Ngày thuê không hợp lệ");  }  double days = Duration.between(start, end).toMinutes() / 1440.0;  double pricePerDay = car.getPrice();  double insuranceFee = request.isInsuranceSelected() ? 90\_000 : 0;  double deliveryFee = request.isDeliverySelected() ? 30\_000 : 0;  double driverFee = request.getDriverRequired() ? (pricePerDay \* 0.2) : 0;  double total = (pricePerDay + insuranceFee) \* days + deliveryFee + driverFee;  Booking booking = new Booking();  booking.setUser(user);  booking.setCar(car);  booking.setStartDate(start);  booking.setEndDate(end);  booking.setPickupLocation(request.getPickupLocation());  booking.setDropoffLocation(request.getDropoffLocation());  booking.setInsuranceSelected(request.isInsuranceSelected());  booking.setDeliverySelected(request.isDeliverySelected());  booking.setDriverRequired(request.getDriverRequired());  booking.setTotalPrice(total);  booking.setStatus("Pending");  bookingRepository.save(booking);  return booking.getBookingId();  } |
| --- |

### *Xem thông tin đơn đặt xe và hủy thông tin đơn đặt. Xác nhận cho thuê xe*

* Luồng từ Android

Người dùng mở màn hình BookingDetailActivity.

BookingDetailActivity nhận bookingId và role từ Intent.

Gọi phương thức loadBookingDetail() để gửi yêu cầu lấy thông tin chi tiết đơn đặt xe.

BookingViewModel gửi yêu cầu lấy chi tiết đơn đặt xe lên server thông qua API /api/user/booking/{bookingId}.

| public void getBookingDetail(Context context, Long bookingId) {  isLoading.setValue(true);  ApiService apiService = RetrofitClient.getRetrofit(context).create(ApiService.class);  apiService.getBookingDetail(bookingId).enqueue(new Callback<ApiResponse<BookingDetailResponse>>() {  @Override  public void onResponse(Call<ApiResponse<BookingDetailResponse>> call, Response<ApiResponse<BookingDetailResponse>> response) {  isLoading.setValue(false);  if (response.isSuccessful() && response.body() != null) {  bookingDetailLiveData.setValue(response.body().getData());  } else {  errorMessage.setValue("Lỗi khi tải dữ liệu");  }  }  @Override  public void onFailure(Call<ApiResponse<BookingDetailResponse>> call, Throwable t) {  isLoading.setValue(false);  errorMessage.setValue("Lỗi kết nối: " + t.getMessage());  }  });  } |
| --- |

* Hủy thông tin đơn đặt:

Người dùng nhấn nút Hủy (hoặc Từ chối nếu là chủ xe).

Gọi phương thức showCancelDialog() để mở dialog nhập lý do hủy.

| private void showCancelDialog(String status) {  AlertDialog.Builder builder = new AlertDialog.Builder(this);  View view = LayoutInflater.from(this).inflate(R.layout.dialog\_cancel\_reason, null);  builder.setView(view);  EditText edtCancelReason = view.findViewById(R.id.edtCancelReason);  Button btnSubmitReason = view.findViewById(R.id.btnSubmitReason);  AlertDialog dialog = builder.create();  btnSubmitReason.setOnClickListener(v -> {  String reason = edtCancelReason.getText().toString().trim();  bookingViewModel.updateBookingStatus(this, bookingId, status, reason);  dialog.dismiss();  });  dialog.show();  } |
| --- |

BookingViewModel gửi yêu cầu cập nhật trạng thái lên server qua API /api/user/booking/update-status.

| public void updateBookingStatus(Context context, Long bookingId, String status, String reason) {  UpdateStatusRequest request = new UpdateStatusRequest(bookingId, status, reason);  apiService.updateBookingStatus(request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful()) {  Toast.makeText(context, "Cập nhật thành công", Toast.LENGTH\_SHORT).show();  loadBookingDetail();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(context, "Lỗi kết nối", Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Xác nhận cho thuê xe:

Khi chủ xe nhấn nút Xác nhận, phương thức updateBookingStatus() được gọi với trạng thái "Confirmed".

| btnConfirm.setOnClickListener(v -> {  bookingViewModel.updateBookingStatus(this, bookingId, "Confirmed", null);  }) |
| --- |

* Backend - UserController.java:

Endpoint: GET /api/user/booking/{bookingId}

Lấy chi tiết đơn đặt xe.

Kiểm tra quyền truy cập dựa trên Authentication.

| @GetMapping("/booking/{bookingId}")  public ResponseEntity<ApiResponse<BookingDetailResponse>> getBookingDetail(  @PathVariable Long bookingId,  Authentication authentication) {  try {  User user = userRepository.findByAccount\_Username(authentication.getPrincipal().toString())  .orElseThrow(() -> new RuntimeException("Không tìm thấy người dùng"));  BookingDetailResponse response = bookingService.getBookingDetail(bookingId, user.getUserId());  return ResponseEntity.ok(new ApiResponse<>(true, "Lấy thông tin booking thành công", response));  } catch (Exception e) {  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Lỗi khi lấy thông tin booking", null));  }  } |
| --- |

Endpoint: PATCH /api/user/booking/update-status

Cập nhật trạng thái đơn đặt xe (Hủy, Xác nhận).

| @PatchMapping("/booking/update-status")  public ResponseEntity<ApiResponse<String>> updateBookingStatus(  @RequestBody UpdateStatusRequest request,  Authentication authentication) {  try {  bookingService.updateBookingStatus(request);  return ResponseEntity.ok(new ApiResponse<>(true, "Cập nhật trạng thái thành công", "Success"));  } catch (Exception e) {  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  .body(new ApiResponse<>(false, "Lỗi khi cập nhật trạng thái", null));  }  } |
| --- |

* BookingService.java:

Phương thức cập nhật trạng thái:

| @Transactional  public void updateBookingStatus(UpdateStatusRequest request) {  Booking booking = bookingRepository.findById(request.getBookingId())  .orElseThrow(() -> new RuntimeException("Không tìm thấy đơn đặt xe"));  booking.setStatus(request.getStatus());  booking.setUpdatedAt(LocalDateTime.now());  bookingRepository.save(booking);  } |
| --- |

## **Đánh giá dịch vụ thuê xe, chỉnh sửa, xóa đánh giá.**

* Luồng từ Android:

Người dùng mở màn hình BookingDetailActivity.

Nếu trạng thái đơn đặt xe là "Completed" và chưa có đánh giá (review == null), nút Đánh giá sẽ xuất hiện.

Khi người dùng nhấn Đánh giá, ứng dụng mở ReviewFragment.

| private void openReviewFragment(int carId, long bookingId) {  ReviewFragment reviewFragment = new ReviewFragment(carId, bookingId, new ReviewFragment.ReviewCallback() {  @Override  public void onReviewSubmitted() {  loadBookingDetail(); // Tải lại chi tiết booking sau khi đánh giá  }  });  reviewFragment.show(getSupportFragmentManager(), "ReviewFragment");  } |
| --- |

* Gửi đánh giá:

Người dùng nhập rating và comment trong ReviewFragment.

Khi nhấn Gửi, ứng dụng gửi yêu cầu POST /api/user/review/add với ReviewRequest.

| public void submitReview(Context context, int carId, long bookingId, Integer rating, String comment) {  ReviewRequest request = new ReviewRequest(carId, bookingId, rating, comment);  apiService.submitReview(request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful()) {  Toast.makeText(context, "Đã gửi đánh giá", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(context, "Lỗi kết nối: " + t.getMessage(), Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Chỉnh sửa đánh giá:

Khi người dùng nhấn nút Sửa đánh giá, openEditReviewDialog() sẽ được gọi.

| private void openEditReviewDialog(ReviewDTO review) {  AlertDialog.Builder builder = new AlertDialog.Builder(this);  View view = LayoutInflater.from(this).inflate(R.layout.fragment\_rating, null);  builder.setView(view);  EditText edtFeedback = view.findViewById(R.id.edtFeedback);  RatingBar ratingBar = view.findViewById(R.id.ratingBar);  Button btnSubmit = view.findViewById(R.id.btnSubmit);  edtFeedback.setText(review.getComment());  ratingBar.setRating((float) review.getRating());  AlertDialog dialog = builder.create();  btnSubmit.setOnClickListener(v -> {  String updatedComment = edtFeedback.getText().toString().trim();  int updatedRating = (int) ratingBar.getRating();  bookingViewModel.updateReview(this, review.getReviewId(), updatedRating, updatedComment);  dialog.dismiss();  });  dialog.show();  } |
| --- |

* Phương thức gửi yêu cầu chỉnh sửa:

| public void updateReview(Context context, long reviewId, int rating, String comment) {  ReviewRequest request = new ReviewRequest(rating, comment);  apiService.updateReview(reviewId, request).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful()) {  Toast.makeText(context, "Đánh giá đã được cập nhật", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(context, "Lỗi kết nối", Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Xóa đánh giá:

Khi người dùng nhấn Xóa, confirmDeleteReview() sẽ được gọi.

| private void confirmDeleteReview(long reviewId, Long bookingId) {  AlertDialog.Builder builder = new AlertDialog.Builder(this);  View view = LayoutInflater.from(this).inflate(R.layout.dialog\_confirm\_delete, null);  builder.setView(view);  Button btnConfirmDelete = view.findViewById(R.id.btnConfirmDelete);  AlertDialog dialog = builder.create();  btnConfirmDelete.setOnClickListener(v -> {  bookingViewModel.deleteReview(this, reviewId, bookingId);  dialog.dismiss();  });  dialog.show();  } |
| --- |

Phương thức gửi yêu cầu xóa đánh giá:

| public void deleteReview(Context context, long reviewId, long bookingId) {  apiService.deleteReview(reviewId).enqueue(new Callback<ApiResponse<String>>() {  @Override  public void onResponse(Call<ApiResponse<String>> call, Response<ApiResponse<String>> response) {  if (response.isSuccessful()) {  Toast.makeText(context, "Đánh giá đã được xóa", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<ApiResponse<String>> call, Throwable t) {  Toast.makeText(context, "Lỗi kết nối", Toast.LENGTH\_SHORT).show();  }  });  } |
| --- |

* Backend - ReviewController.java:

Endpoint: POST /review/add - Gửi đánh giá

| @PostMapping("/review/add")  public ResponseEntity<ApiResponse<String>> addReview(  @Valid @RequestBody ReviewRequest reviewRequest,  Authentication authentication) {  reviewService.addReview(authentication, reviewRequest);  return ResponseEntity.ok(new ApiResponse<>(true, "Đánh giá thành công", null));  } |
| --- |

Endpoint: PUT /review/update/{reviewId} - Cập nhật đánh giá

| @PutMapping("/review/update/{reviewId}")  public ResponseEntity<ApiResponse<String>> updateReview(  @PathVariable Long reviewId,  @Valid @RequestBody ReviewRequest reviewRequest,  Authentication authentication) {  reviewService.updateReview(authentication, reviewId, reviewRequest);  return ResponseEntity.ok(new ApiResponse<>(true, "Sửa đánh giá thành công", null));  } |
| --- |

Endpoint: DELETE /review/delete/{reviewId} - Xóa đánh giá

| @DeleteMapping("/review/delete/{reviewId}")  public ResponseEntity<ApiResponse<String>> deleteReview(  @PathVariable Long reviewId,  Authentication authentication) {  reviewService.deleteReview(authentication, reviewId);  return ResponseEntity.ok(new ApiResponse<>(true, "Xóa đánh giá thành công", null));  } |
| --- |

# **PHẦN KẾT LUẬN**

Trong quá trình phát triển ứng dụng “Thuê xe oto”, nhóm đã tập trung xây dựng hệ thống với các tính năng cơ bản nhưng thiết thực, đáp ứng nhu cầu thuê xe của người dùng. Ứng dụng được triển khai trên nền tảng Android, sử dụng Java cho frontend và Spring Boot cho backend, kết nối với MySQL để quản lý dữ liệu xe, người dùng và đơn thuê.

## Kết quả đạt được

* Về kiến thức:
  + Nắm vững cách xây dựng ứng dụng di động Android, từ thiết kế giao diện đến xử lý sự kiện.
  + Hiểu rõ mô hình RESTful API trong Spring Boot, tích hợp WebSocket để triển khai tính năng nhắn tin thời gian thực.
  + Nắm bắt cách Hibernate tự động sinh cơ sở dữ liệu từ các Entity, đảm bảo tính đồng bộ giữa backend và MySQL.
* Về kỹ năng:
  + Cải thiện kỹ năng lập trình Java và XML cho Android, đặc biệt là thiết kế giao diện người dùng và xử lý logic ứng dụng.
  + Tích lũy kinh nghiệm về tích hợp WebSocket để triển khai tính năng nhắn tin thời gian thực giữa người dùng và chủ xe.
  + Học hỏi cách xây dựng API RESTful, xử lý request/response và tối ưu truy vấn MySQL.
* Về sản phẩm:
  + Ứng dụng Thuê xe oto với các tính năng cơ bản như:
  + Đăng ký tài khoản, đăng nhập và quản lý thông tin người dùng.
  + Tìm kiếm xe, xem chi tiết xe và đặt thuê xe.
  + Nhắn tin thời gian thực giữa người thuê và chủ xe thông qua WebSocket.
  + Quản lý đơn thuê và cập nhật trạng thái xe.
* Ứng dụng đã được kiểm thử và hoạt động ổn định trong môi trường giả lập.

## Kỹ năng đạt được

* Kỹ năng lập trình Android: Thiết kế giao diện bằng XML và xử lý logic bằng Java.
* Kỹ năng thiết kế cơ sở dữ liệu: Sử dụng Hibernate để tự động sinh các bảng từ Entity và tối ưu hóa truy vấn MySQL.
* Kỹ năng xử lý giao tiếp thời gian thực: Tích hợp WebSocket để thực hiện chức năng nhắn tin.
* Kỹ năng làm việc nhóm: Quản lý mã nguồn trên GitHub, phối hợp xử lý lỗi và triển khai tính năng một cách nhịp nhàng.

## Ưu điểm

* Giao diện người dùng dễ sử dụng, thiết kế trực quan, phù hợp với người dùng phổ thông.
* Tính năng đặt thuê xe hoạt động chính xác, cho phép người dùng xem chi tiết xe và đặt thuê nhanh chóng.
* Nhắn tin thời gian thực hoạt động ổn định, giúp người dùng trao đổi nhanh chóng khi có đơn thuê.
* Hibernate tự động sinh cơ sở dữ liệu, đảm bảo tính đồng bộ và dễ dàng mở rộng thêm các tính năng mới.

## Nhược điểm

* Chưa xem xét đến vấn đề bảo mật: Hiện tại, ứng dụng chưa triển khai các biện pháp bảo mật nâng cao như xác thực hai yếu tố (2FA), kiểm soát truy cập chi tiết.
* Tính năng thanh toán chưa triển khai: Ứng dụng hiện tại chưa tích hợp các phương thức thanh toán trực tuyến như MoMo, ZaloPay hoặc thẻ tín dụng.
* Chưa triển khai tính năng thông báo: Ứng dụng chưa có hệ thống thông báo đẩy để cập nhật trạng thái đơn thuê hoặc thông báo khi có tin nhắn mới.

## Hướng phát triển

* Khắc phục các vấn đề bảo mật: Tích hợp xác thực hai yếu tố (2FA), mã hóa dữ liệu người dùng và triển khai các cơ chế kiểm soát truy cập để đảm bảo an toàn cho hệ thống.
* Phát triển tính năng thanh toán trực tuyến: Tích hợp các phương thức thanh toán qua MoMo, ZaloPay và thẻ tín dụng để tăng tính tiện ích và mở rộng khả năng thanh toán cho người dùng.
* Triển khai hệ thống thông báo đẩy: Phát triển tính năng thông báo đẩy để cập nhật kịp thời trạng thái đơn thuê xe và tin nhắn mới từ chủ xe hoặc người thuê.
* Tối ưu hóa cơ sở dữ liệu: Rà soát và cải tiến các truy vấn SQL để tăng tốc độ truy xuất dữ liệu khi số lượng người dùng và dữ liệu tăng lên.