HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

CƠ SỞ THÀNH PHỐ HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN 2

∞Ω∞

BÁO CÁO KIỂM THỬ VÀ CHẤT LƯỢNG ỨNG DỤNG

HỌC PHẦN: NHẬP MÔN CÔNG NGHỆ PHẦN MỀM

ĐỀ TÀI: Web App Quản Lý Chấm Công Nhân Viên

Nhóm sinh viên: Nhóm 3

Phụ trách kiểm thử: Nguyễn Ngọc Gia Hân

Giảng viên hướng dẫn đồ án: Châu Văn Vân

TP. HCM 12/2025

MỤC LỤC

LỜI MỞ ĐẦU

Trong quá trình phát triển phần mềm, hoạt động kiểm thử (Testing) đóng vai trò quan trọng nhằm đảm bảo hệ thống vận hành đúng chức năng, ổn định và đáp ứng yêu cầu nghiệp vụ. Đối với dự án Mini App Chấm Công Nhân Viên, việc kiểm thử càng trở nên cần thiết bởi hệ thống liên quan đến dữ liệu thời gian thực, vị trí GPS, thông tin nhân viên và các thao tác mang tính nhạy cảm như Check-in, Check-out. Nếu không được kiểm thử đầy đủ, hệ thống có thể phát sinh sai lệch dữ liệu chấm công, ảnh hưởng trực tiếp đến hiệu suất làm việc và quá trình tính lương của doanh nghiệp.

Mục tiêu của quá trình kiểm thử trong dự án này là:

- Xác minh độ chính xác của các API chấm công.
- Đảm bảo giao diện người dùng hoạt động ổn định, dễ sử dụng và hiển thị đúng dữ liệu.
- Kiểm tra logic xử lý thời gian làm việc, điều kiện check-in/check-out và định vị GPS.
- Phát hiện, ghi nhận và đề xuất hướng khắc phục các lỗi có thể xảy ra trước khi triển khai thực tế.

Để đảm bảo chất lượng toàn diện, báo cáo sử dụng ba công cụ kiểm thử phổ biến:

1. **Postman** – dùng để kiểm thử API (API Testing), đảm bảo các endpoint như Login, Check-in, Check-out và Báo cáo trả về đúng dữ liệu và status code.
2. **Selenium** – dùng cho kiểm thử giao diện (UI Testing), mô phỏng thao tác người dùng trên trình duyệt nhằm kiểm tra độ ổn định, tính tương tác và tính chính xác của các trang chấm công và báo cáo.
3. **Jest** – dùng để kiểm thử đơn vị (Unit Testing), đảm bảo các hàm xử lý logic như tính giờ làm, xác thực điều kiện check-in/check-out và kiểm tra vị trí GPS hoạt động đúng trong mọi trường hợp.

Thông qua việc kết hợp cả ba công cụ trên, quá trình kiểm thử giúp đánh giá toàn diện hệ thống từ backend, frontend đến logic xử lý, góp phần nâng cao chất lượng và độ tin cậy của ứng dụng chấm công.

I. THÔNG TIN DỰ ÁN

1. Tổng quan dự án

Tên hệ thống: Hệ thống Chấm công & Quản lý Nhân sự (Timekeeping & HR Management System).

Dự án Mini App Chấm Công Nhân Viên được xây dựng nhằm hỗ trợ doanh nghiệp quản lý thời gian làm việc của nhân viên một cách chính xác và tiện lợi. Hệ thống cho phép nhân viên thực hiện Check-in/Check-out thông qua quét khuôn mặt và vị trí GPS, đồng thời cung cấp giao diện báo cáo để theo dõi thời gian làm việc theo ngày, tuần và tháng. Mục tiêu chính của dự án là tối ưu hóa quá trình chấm công thủ công truyền thống, giảm sai sót và nâng cao tính minh bạch trong việc quản lý nhân sự.

Ứng dụng gồm ba thành phần chính:

- Frontend Web: giao diện người dùng, hỗ trợ thao tác chấm công và xem báo cáo. Chuyển đổi giữa tài khoản người dùng và người quản lý.
- Backend API: cung cấp các dịch vụ xử lý xác thực người dùng, ghi nhận thời điểm check-in/check-out và truy xuất báo cáo.
- Cơ sở dữ liệu: lưu trữ thông tin nhân viên, phiên chấm công và lịch sử làm việc, quản lý ca làm và quản lý hệ thống nhân viên.

Phạm vi của dự án tập trung vào:

- Quét mặt để xác thực nhân viên.
- Ghi nhận tọa độ GPS trong mỗi phiên chấm công.
- Tính toán thời gian làm việc trong ngày.
- Xuất báo cáo theo từng khoảng thời gian trong tháng.

Việc kiểm thử dự án là bước quan trọng nhằm đảm bảo hệ thống hoạt động đúng như yêu cầu, dữ liệu chấm công được ghi nhận chính xác và giao diện đáp ứng tốt nhu cầu sử dụng của nhân viên và quản trị viên.

2. Yêu cầu chức năng

- Đăng nhập hệ thống bằng tài khoản nhân viên.
- Check-in:

  + Xác thực qua camera (Face Scan).

  + Ghi nhận vị trí GPS tại thời điểm check-in.
- Check-out:

  + Lưu thời gian kết thúc ca làm.

  + Kiểm tra điều kiện hợp lệ (ví dụ: không check-out trước khi check-in).

- Xem báo cáo:

  + Báo cáo theo ngày/tháng.

  + Thống kê thời gian làm việc
- Quản lý thông tin nhân viên, thông tin ca làm.

II. CÔNG CỤ KIỂM THỬ

1. Jest – framework kiểm thử đơn vị (Unit Test)

- Jest là một framework kiểm thử JavaScript phổ biến, được phát triển bởi Meta (Facebook) và được sử dụng rộng rãi trong các dự án web hiện đại. Jest được thiết kế chuyên biệt cho việc kiểm thử logic xử lý (Unit Testing) và hoạt động hiệu quả với các ứng dụng Node.js cũng như các thư viện frontend như React.
- Trong dự án Mini App Chấm Công Nhân Viên, Jest được sử dụng để kiểm thử các hàm logic chấm công, bao gồm: kiểm tra điều kiện check-in/check-out, tính toán thời gian làm việc, và xác định tính hợp lệ của vị trí GPS.
- Bộ test suite hiện tại bao phủ các chức năng cốt lõi của hệ thống chấm công, bao gồm: xác thực (Auth), phân quyền (RBAC), nghiệp vụ chấm công (Attendance), quản lý ca làm việc (Shifts) và nhận diện khuôn mặt (FaceID).
- Kiểm thử được thực hiện theo mô hình **Mocking**, **Time Travel** giả lập các tương tác với cơ sở dữ liệu (Firebase Firestore) và các thư viện bên ngoài (jsonwebtoken) để đảm bảo tốc độ và tính cô lập của test case.

a. Lý do lựa chọn Jest

- Dễ cấu hình: cài đặt và sử dụng đơn giản, không yêu cầu nhiều thiết lập
- Hiệu năng cao: Jest chạy test rất nhanh nhờ cơ chế chạy song song.
- Tích hợp tốt với Node.js: phù hợp để kiểm thử các module logic backend.

b. Jest trong dự án

- **Tầng Middleware (Bảo mật & Phân quyền)**
- Các file: **auth.middleware.test.js, admin.middleware.test.js, selfOrAdmin.middleware.test.js.**
- Xác thực (Authentication): Kiểm tra tính hợp lệ của JWT Token (Header Bearer, Query param). Xử lý các trường hợp Token hết hạn, sai chữ ký, hoặc format không đúng.Hàm kiểm tra điều kiện check-out (check-out trước khi check-in…).

- Phân quyền Admin (Authorization): Đảm bảo chỉ các role được cấp phép (admin, System Admin, manager) mới truy cập được. Chặn các role không hợp lệ (user, employee) hoặc user null.
- Quyền truy cập dữ liệu (Data Privacy): Kiểm thử logic selfOrAdmin - cho phép User thường xem dữ liệu của chính mình, nhưng Admin có thể xem dữ liệu của bất kỳ ai.

- **Tầng Service (Nghiệp vụ cốt lõi)**
- Các file: **attendance.service.test.js, shifts.service.test.js, face.service.test.js**

Attendance (Chấm công):

- Logic Check-in/Check-out: Ngăn chặn check-in 2 lần trong ngày, tính toán giờ làm việc (workSeconds).
- Xử lý lịch sử & thống kê: Tính số ngày công, ngày nghỉ chính xác dựa trên dữ liệu giả lập
- Admin Update: Kiểm tra quyền sửa đổi bản ghi chấm công.

Shifts (Ca làm việc):

- CRUD: Tạo, sửa, xóa ca làm việc. Tự động sinh mã ca (CA001) và tên ca nếu thiếu.
- Phân ca (Assign): Gán ca cho một hoặc nhiều nhân viên, xử lý định dạng ngày tháng
- Validation: Bắt lỗi thiếu giờ bắt đầu/kết thúc, trùng lặp nhân viên trong ca.

FaceID (Nhận diện):

- Thuật toán: Tính khoảng cách Euclidean giữa các vector khuôn mặt (Embedding 128 chiều).
- GPS: Tính khoảng cách địa lý (Haversine) để đảm bảo nhân viên chấm công trong bán kính cho phép.
- Cấu hình: Kiểm tra việc đọc FACE_THRESHOLD từ biến môi trường.

- **Tầng Controller (Xử lý Request/Response)**
- Các file: **attendance.controllers.test.js, face.controller.test.js**

- Đảm bảo Controller gọi đúng Service tương ứng.
- Kiểm tra định dạng phản hồi (Response format): Trả về JSON chuẩn { success: true, data: ... }
- Mã lỗi HTTP: Kiểm tra việc trả về đúng status code:
  + 200: Thành công.
  + 400: Lỗi dữ liệu đầu vào (thiếu param, sai format).
  + 404: Không tìm thấy tài nguyên.

+ 500: Lỗi Server/Service.

- Điểm nổi bật: Test kỹ việc validate dữ liệu đầu vào tại controller (ví dụ: embedding phải là array, thiếu month trong query param).

c. Vai trò của Jest trong dự án

- Đảm bảo độ chính xác của nghiệp vụ chấm công, tránh sai lệch giờ làm.
- Ngăn lỗi logic phát sinh khi cập nhật tính năng mới
- Tăng độ tin cậy của hệ thống vì logic cốt lõi đã được kiểm tra đầy đủ.
- Hỗ trợ Debug nhanh khi phát hiện lỗi liên quan đến quy trình xử lý dữ liệu.

2. Selenium – Kiểm thử UI

- Selenium là bộ công cụ tự động hóa kiểm thử giao diện người dùng (UI Testing) trên trình duyệt. Selenium mô phỏng các thao tác của người dùng như click, nhập liệu, chuyển trang và kiểm tra nội dung hiển thị trên web.
- Trong dự án Mini App Chấm Công Nhân Viên, Selenium được sử dụng để kiểm thử hai giao diện chính: **Giao diện chấm công** và **Giao diện báo cáo**. Đây là những phần người dùng thao tác trực tiếp nên việc kiểm thử UI đóng vai trò quan trọng để đảm bảo trải nghiệm mượt mà và không lỗi.
- Bộ test suite này thực hiện kiểm thử đầu cuối (End-to-End Testing), mô phỏng hành vi thực tế của người dùng trên trình duyệt để xác minh các luồng nghiệp vụ quan trọng nhất: Chấm công vào (Check-in), Chấm công ra (Check-out) và Xem báo cáo thống kê (Manager Report).

a. Lý do lựa chọn Selenium

- Tự động hóa kiểm thử UI trên nhiều trình duyệt khác nhau.
- Mô phỏng thao tác thực của người dùng rất chính xác.
- Hỗ trợ đa ngôn ngữ lập trình như JavaScript, Java, Python,…
- Tích hợp tốt với quy trình CI/CD.
- Phù hợp để kiểm thử các luồng thao tác lặp đi lặp lại như chấm công hằng ngày.

b. Selenium trong dự án

- **Test checkin.test.js**

- Mục tiêu: Xác minh quy trình chấm công khuôn mặt từ lúc đăng nhập đến khi dữ liệu được ghi nhận vào lịch sử.
- Kiểm thử các tác vụ:

  + Xử lý quyền Camera: Chờ và xử lý trạng thái sẵn sàng của giao diện trước khi gọi Camera.

+ Kích hoạt FaceID: Nhấn nút Check-in -> Chờ Modal xuất hiện -> Kiểm tra Video stream hiển thị.

+ Xác nhận: Đóng Modal (giả lập nhận diện thành công) -> Chờ hệ thống đồng bộ dữ liệu.+ Kiểm tra dữ liệu hiển thị trên trang báo cáo.

- Điểm kỹ thuật: Sử dụng kỹ thuật đếm dòng (rows.length) để verify thay vì chỉ kiểm tra thông báo UI, giúp đảm bảo dữ liệu thực sự được lưu xuống Database.

- **Test checkout.test.js**

- Mục tiêu: Xác minh quy trình chấm công ra và điều hướng trang.
- Kiểm thử các tác vụ:

+ Kích hoạt FaceID: Nhấn nút Check-out -> Kiểm tra Modal và Video stream hoạt động.

+ Xử lý Modal: Giả lập thao tác đóng modal sau khi nhận diện xong -> Kiểm tra Modal đã biến mất (stalenessOf).

+ Kiểm chứng UI: Tự động điều hướng sang Tab Lịch sử -> Xác minh bảng lịch sử chấm công (Table) có hiển thị và chứa dữ liệu.

- Điểm kỹ thuật: Kiểm tra kỹ trạng thái của Modal (Visible/Invisible) để đảm bảo trải nghiệm người dùng không bị treo (freeze).

- **Test report.test.js**

- Mục tiêu: Kiểm tra chức năng xem và trích xuất báo cáo dành cho Admin/Manager.
- Kiểm thử các tác vụ:

+ Đăng nhập Admin: Sử dụng tài khoản quản lý (admin@timekeeping.com).

+ Bộ lọc dữ liệu: Truy cập trang Báo cáo -> Nhập tháng cần xem (2025-01) -> Nhấn nạp dữ liệu.

+ Kiểm tra chi tiết (Detailed View): Mở xem chi tiết ca làm việc của nhân viên.

+ Kiểm tra thống kê (Aggregated Stats):

+ Chức năng Xuất (Export): Kiểm tra nút "Xuất báo cáo" có hiển thị và cho phép click.

- Điểm kỹ thuật: Sử dụng Wait thông minh để chờ dữ liệu từ API tải xong trước khi thực hiện các phép Assert trên các ô thống kê.

c. Vai trò của Selenium trong dự án

- Xác thực toàn bộ trải nghiệm UI của người dùng.
- Đảm bảo giao diện không gặp lỗi hiển thị hay lỗi thao tác.
- Kiểm tra sự tương thích giao diện trên nhiều kích thước màn hình.
- Giảm thời gian kiểm thử thủ công trong các tác vụ lặp lại.

3. Postman – Công cụ kiểm thử API

- Postman là một công cụ mạnh mẽ và phổ biến dùng để kiểm thử API trong quá trình phát triển phần mềm. Công cụ cho phép gửi request trực tiếp đến server, kiểm tra dữ liệu trả về và mô phỏng toàn bộ quy trình hoạt động của các API mà không cần xây dựng giao diện người dùng.
- Trong dự án Mini App Chấm Công Nhân Viên, Postman được sử dụng để kiểm thử các API quan trọng như **Check-in**, **Check-out**. Những API này đóng vai trò nền tảng trong việc ghi nhận và xử lý dữ liệu chấm công, vì vậy việc kiểm thử bằng Postman giúp đảm bảo tính ổn định và độ chính xác của backend.

a. Lý do lựa chọn Postman

- Giao diện thân thiện, dễ sử dụng cho cả tester và developer.
- Hỗ trợ nhiều loại request (GET, POST, PUT, DELETE,…)
- Có thể viết test script (JavaScript) để kiểm tra tự động response.
- Hỗ trợ chạy Collection theo bộ giúp tự động hóa và tăng tốc kiểm thử.
- Cho phép mô phỏng nhiều môi trường (local, staging, production).

b. Postman trong dự án

- Module Xác Thực (Authentication)
- **API:** POST /api/auth/login
- Đánh giá: API xử lý tốt việc cấp phát Token JWT và bảo mật thông tin (password trả về dạng hash hoặc ẩn đi - trong log thấy passwordHash được trả về, lưu ý bảo mật nên ẩn field này ở response nếu không cần thiết)
- Module Đăng Ký (Enrollment)
- **API:** POST /api/face/enroll
- Đánh giá: API yêu cầu xác thực (Authorization: Bearer Token) trước khi cho phép đăng ký, đảm bảo tính bảo mật.
- Module Chấm Công (Check-in / Check-out)
- **API:** POST /api/face/check-in & POST /api/face/check-out
- Đánh giá: Đây là phần logic nghiệp vụ phức tạp nhất, được kiểm thử rất kỹ với nhiều trường hợp ngoại lệ.

c. Vai trò của Postman trong dự án

- Phát hiện lỗi nghiệp vụ trước khi đưa lên giao diện.
- Đảm bảo các API hoạt động ổn định và chính xác.

- Kiểm tra tính an toàn của API thông qua token và header.
- Đảm bảo dữ liệu chấm công được ghi nhận đúng và nhất quán.

III. THỰC HIỆN KIỂM THỬ

1. Kế hoạch kiểm thử

a. Mục tiêu kiểm thử

Kế hoạch kiểm thử được xây dựng nhằm đảm bảo rằng toàn bộ các chức năng của hệ thống Mini App Chấm Công Nhân Viên hoạt động chính xác theo yêu cầu đề ra. Cụ thể, mục tiêu của quá trình kiểm thử là:

- Xác minh tính đúng đắn của API chấm công (check-in, check-out, báo cáo).
- Đảm bảo giao diện người dùng UI hoạt động ổn định và không phát sinh lỗi trong quá trình thao tác.
- Kiểm thử các logic xử lý cốt lõi của hệ thống như kiểm tra điều kiện check-in/chế độ GPS và tính toán thời gian làm việc.
- Phát hiện lỗi sớm, giảm rủi ro khi triển khai và đảm bảo chất lượng hệ thống trước khi đưa vào sử dụng thực tế.

b. Phạm vi kiểm thử

- **Chức năng được kiểm thử**

- API Backend: Login, Check-in, Check-out.
- Giao diện Check-in/Check-out: nút chức năng, camera quét mặt, thông báo.
- Giao diện báo cáo: bảng dữ liệu, lọc theo tháng, chi tiết ngày công nhân viên.
- Logic xử lý: điều kiện chấm công, tính giờ làm, xác thực GPS.
- Chức năng phân quyền nâng cao (nếu chưa phát triển), quản lý ca làm.

c. Phương pháp kiểm thử

- **Kiểm thử đơn vị (Unit Test) – Jest**

- Kiểm tra logic check-in/check-out
- Kiểm tra xử lý tính toán giờ làm.
- Kiểm tra tính hợp lệ GPS.
- Kiểm tra luồng phân quyền, xác thực
- Kiểm tra quản lý ca làm.

- **Kiểm thử API – Postman**

- Gửi request và đánh giá response Module Check-in/Check-out, Login.
- Kiểm tra token và xác thực.
- Kiểm tra error handling và status code.

- **Kiểm thử giao diện – Selenium**

- Mô phỏng thao tác người dùng thật.
- Kiểm tra hiển thị bảng báo cáo, chi tiết chấm công.
- Kiểm tra popup camera và các hành động click.

d. Môi trường kiểm thử

- **Phần mềm**

- Trình duyệt: Microsoft Edge 143v
- Node.js v24
- Postman v11
- Selenium WebDriver
- Jest v30

- Phần cứng

- Máy tính hỗ trợ camera.
- GPS mô phỏng qua trình duyệt hoặc thiết bị di động thực.

- Dữ liệu kiểm thử

- Tài khoản nhân viên mẫu, quản lý mẫu.
- Dữ liệu GPS hợp lệ / không hợp lệ.
- Các trường hợp chấm công hợp lệ và bất thường.

e. Lịch trình kiểm thử

Giai đoạn	Nội dung	Thời gian
Chuẩn bị	Cấu hình môi trường, tạo dữ liệu kiểm thử	1 ngày
Kiểm thử API	Test Login, Check-in, Check-out	1 ngày
Kiểm thử UI	Test giao diện chấm công + báo cáo	2 ngày
Unit test với Jest	Kiểm thử logic xử lý	2 ngày
Tổng hợp kết quả	Ghi lỗi, báo cáo, retest	1 ngày

f. Tiêu chí bắt đầu và kết thúc kiểm thử

- **Tiêu chí bắt đầu**

- Môi trường đã được cấu hình hoàn chỉnh.
- Backend và frontend chạy ổn định.
- Tất cả test case đã được chuẩn bị.

- **Tiêu chí kết thúc**

- Tất cả test case đã được thực thi 100%
- Tỷ lệ pass ≥ 90%.
- Không còn lỗi nghiêm trọng (critical/blocker).
- Các lỗi mức trung bình đã được ghi lại và xác nhận.

2. Test Case

a. Kiểm thử logic – Jest

ID	Chức năng	Kịch bản kiểm thử (Test Scenario)	Input / Điều kiện	Kết quả mong đợi (Expected)
Test Case phân hệ Xác thực & Phân quyền: auth.middleware.test.js, admin.middleware.test.js và selfOrAdmin.middleware.test.js				
AUTH_01	JWT Auth	Token hợp lệ trong Header	Header: Bearer valid_token	next() được gọi, req.user có dữ liệu
AUTH_02	JWT Auth	Token hợp lệ trong Query Param	Query: ?token=valid_token	next() được gọi

AUTH_03	JWT Auth	Token hết hạn hoặc sai chữ ký	Token hết hạn / sai secret key	Trả về 401 Invalid token
AUTH_04	JWT Auth	Không có token	Header/Query trống	Trả về 401 Missing token
RBAC_01	Admin Role	Người dùng có quyền Admin truy cập	Role: admin, System Admin, manager	next() được gọi
RBAC_02	Admin Role	Người dùng thường truy cập trang Admin	Role: user, employee	Trả về 403 Admin only
SOA_01	Self/Admin	User xem dữ liệu của chính mình	req.user.id trùng req.params.userId	next() được gọi
SOA_02	Self/Admin	Admin xem dữ liệu người khác	User role: admin, Params: ID người khác	next() được gọi
SOA_03	Self/Admin	User thường xem dữ liệu người khác	User role: employee, Params: ID người khác	Trả về 403 Access denied
Test Case phân hệ Chấm công: attendance.controllers.test.js và attendance.service.test.js				
ATT_01	Check-in	Check-in lần đầu trong ngày	Chưa có bản ghi Check-in hôm nay	Success: True, tạo docId mới
ATT_02	Check-in	Check-in lại (Duplicate)	Đã tồn tại bản ghi Check-in hôm nay	Success: False, báo lỗi đã check-in
ATT_03	Check-out	Check-out thành công	Đã Check-in, chưa Check-out	Success: True, cập nhật checkOutAt
ATT_04	Check-out	Check-out khi chưa Check-in	Không tìm thấy bản ghi Check-in	Success: False, báo lỗi chưa check-in
ATT_05	History	Lấy lịch sử chấm công	User ID hợp lệ	Trả về danh sách, sắp xếp theo ngày giảm dần

ATT_06	Summary	Tính tổng ngày công	Có dữ liệu shifts và attendance	Trả về đúng số daysWorked và daysOff
ATT_07	Calendar	Lấy lịch làm việc tháng	Month: YYYY-MM	Trả về object map theo ngày với status (checked-full, absent, v.v.)
ROUTE_01	API Route	Gọi API sai phương thức (Method)	Gọi POST vào endpoint chỉ nhận GET	Trả về 404
ROUTE_02	Admin API	Admin cập nhật chấm công	Body chứa workSeconds, note	Gọi Controller update thành công (200)
Test Case phân hệ Quản lý Ca làm: shifts.service.test.js				
SFT_01	Tạo ca	Tạo ca với đầy đủ thông tin	Date, Name, Start, End hợp lệ	Tạo thành công, trả về Shift Code (VD: CA001)
SFT_02	Tạo ca	Tạo ca thiếu giờ	Thiếu startTime hoặc endTime	Ném lỗi (Throw Error)
SFT_03	Tạo ca	Tự động sinh tên ca	Không gửi name	Tên tự động: Ca ngày... (Giờ-Giờ)
SFT_04	Gán ca	Gán nhiều NV vào nhiều ca	userIds [], shiftIds []	Tạo đủ số lượng bản ghi trong user_shifts
SFT_05	Gán ca	Format ngày tháng	Input 28/11/2025	Lưu vào DB dạng chuẩn 2025-11-28
SFT_06	Xoá NV	Xoá nhân viên khỏi ca	ShiftId và UserId hợp lệ	Xoá bản ghi tương ứng trong DB
SFT_07	Xoá NV	Xoá nhân viên không có trong ca	ShiftId/UserId không khớp	Ném lỗi Nhân viên không tồn tại trong ca
Test Case phân hệ Nhận diện Khuôn mặt & GPS: face.service.test.js và face.controller.test.js				
FACE_01	Enroll	Đăng ký khuôn mặt mới	Embedding (vector 128 số) hợp lệ	Lưu thành công vào Firestore

FACE_02	Validate	Xác thực khuôn mặt + GPS hợp lệ	Face Distance < 0.6 VÀ GPS Distance < Radius	Success: True
FACE_03	Validate	Khuôn mặt không khớp	Face Distance > Threshold (0.6)	Success: False, message "Khuôn mặt không khớp"
FACE_04	Validate	Sai vị trí địa lý	GPS Distance > Radius (Company Settings)	Success: False, message "Ngoài khu vực làm việc"
FACE_05	Validate	Môi trường biến thiên (Env Var)	Chỉnh process.env.FACE_THRESHOLD thấp xuống	Logic so sánh phải dùng giá trị mới từ Env
CTR_01	Check-in API	Check-in bằng FaceID thành công	Service Face trả về True + Service Checkin trả về True	Trả về 200 + Data checkin
CTR_02	Check-in API	Check-in thất bại do logic Face	Service Face trả về False	Trả về 400 + Message lỗi từ Service

b. Kiểm thử giao diện – Selenium

ID	File Nguồn	Tên Kịch Bản (Scenario)	Các Bước Thực Hiện (Test Steps)	Dữ Liệu Kiểm Thử (Test Data)	Kết Quả Mong Đợi (Expected Result)
TC-01	**checkin.test.js**	Check-in thành công và ghi nhận lịch sử	1. Đăng nhập tài khoản nhân viên.	Email: nv2@example.com	- Modal FaceID bật lên.
			2. Vào tab Home, nhấn nút "Face Check-in".	Pass: 12345678	- Video stream hiển thị.
			3. Chờ Modal và Video hiển thị.		- Số lượng bản ghi trong History tăng lên 1 dòng so với trước khi check-in.
			4. Đóng Modal (giả lập nhận diện xong).		
			5. Chuyển sang tab History.		

			6. Đếm số dòng dữ liệu.		
TC-02	**checkin.test.js**	Kiểm tra hiển thị giao diện Camera Check-in	1. Đăng nhập.	N/A	- Phần tử #faceModal có thuộc tính open.
			2. Nhấn nút "Face Check-in".		- Phần tử #faceVideo hiển thị (isDisplayed = true).
			3. Kiểm tra phần tử Modal và Video.		
TC-03	**checkout.test.js**	Check-out thành công và điều hướng	1. Đăng nhập tài khoản nhân viên.	Email: nv2@example.com	- Modal Check-out hiển thị và đóng lại được.
			2. Điều hướng về Tab Home.	Pass: 12345678	- Điều hướng thành công sang trang History.
			3. Nhấn nút "Face Check-out".		- Bảng #histTable hiển thị.
			4. Chờ Modal hiển thị, sau đó đóng Modal.		- Có dữ liệu dòng (tr) trong bảng.
			5. Chờ Modal biến mất hoàn toàn.		
			6. Điều hướng sang Tab History.		
TC-04	**report.test.js**	Xem báo cáo chấm công theo tháng (Manager)	1. Đăng nhập tài khoản Quản lý (Admin).	Email: admin@...	- Nút Summary (#btnLoadSummary) chuyển trạng thái enabled.
			2. Vào tab "Statistics & Report".	Pass: admin123	- Bảng chi tiết nhân viên (#employeeSummaryTable) có dữ liệu (innerHTML không rỗng).
			3. Nhập tháng vào ô chọn tháng.	Tháng: 2025-01	
			4. Nhấn nút "Nạp dữ liệu" (Reload).		
			5. Nhấn nút "Xem chi tiết" (Summary).		
TC-05	**report.test.js**	Kiểm tra hiển thị các	1. Thực hiện các bước nạp dữ liệu như TC-04.	N/A	Các ô chỉ số sau không được rỗng:

		chỉ số thống kê tổng hợp	2. Đọc giá trị các ô thống kê: Tổng ca, Có mặt, Vắng mặt, Tỷ lệ.		- Tổng ca (#sumTotal)
					- Có mặt (#sumPresent)
					- Vắng mặt (#sumAbsent)
					- Tỷ lệ (#sumRate)
TC-06	**report.test.js**	Kiểm tra trạng thái nút Xuất báo cáo	1. Thực hiện nạp dữ liệu thành công.	N/A	- Nút xuất báo cáo phải hiển thị (isDisplayed).
			2. Kiểm tra trạng thái nút Xuất báo cáo (nút Summary).		- Nút phải cho phép bấm (isEnabled).

c. Kiểm thử API – Postman

TC ID	Test Scenario	Test Data	Header & Body	Output
LOG 01	Login thành công	**HTTP method:** POST **Path**: http://localhost:5000/api/auth/login	**Content-Type:** application/json **Body:** { "email": "nv1@example.com", "password": "123" }	{ "success": true, "data": { "token": "eyJhbGciOiJIUz…", "refreshToken": "eyJhbGciOiJIUz…", "user": { "id": "oHnlk8LlMMdSEHgxeDoP", "employeeCode": "NV013", "name": "Nhân viên 3", "email": "nv3@example.com", "passwordHash": "$2b$10$BqW9ygwcFG/fV7vQkoWTcuIvd3HCcIKZw7JVoW0q/OT1sZB/mImGu", "role": "user", "position": "employee", "dept": "", "status": "active",

				"workStatus": "dang_lam", "createdAt": "2025-12-04T19:40:58.403Z", "updatedAt": "2025-12-04T19:40:58.403Z" } } }

LOG 02	Sai mật khẩu		**Content-Type:** application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002
			]
			}
				{
				"success": false,
				"message": "Email hoặc mật khẩu không đúng"
				}

ER 01	Enroll thành công	**HTTP method:** POST
		Path: http://localhost:5000/api/face/enroll
			Authorization: Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002]
			}
				{
				"success": true,
				"message": "Đăng ký FaceID thành công"
				}

CI 01	Check-in thành công	**HTTP method:** POST
		Path: http://localhost:5000/api/face/check-in
			Authorization: Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002],
			"lat": 10.123456,
			"lng": 106.123456
			}
				{
				"success": true,
				"message": "Check-in FaceID thành công",
				"data": {
				"docId": "oHnlk8LlMMdSEHgxeDoP_2025-12-07",
				"faceDist": 0,
				"gpsDist": 0
				}
				}

CI 02	Chưa enroll mặt		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002],
				{
				"success": false,
				"message": "Chưa đăng ký khuôn mặt"
				}

			"lat": 10.123456, "lng": 106.123456 }	

CI 03	Sai mặt		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			-0.881, 0.554, 0.221, 0.992, -0.337, 0.998, -0.445, 0.002
			], "lat": 10.123456,
			"lng": 106.123456
			}
				{
				"success": false,
				"message": "Khuôn mặt không khớp"
				}

CI 04	Thiếu GPS		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002
			]
			}
				{
				"success": false,
				"message": "Thiếu tọa độ GPS"
				}

CI 05	Ngoài vùng GPS		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002],
			"lat": 11.000000,
			"lng": 107.000000
			}
				{
				"success": false,
				"message": "Bạn đang ngoài khu vực làm việc (cách 136676m, giới hạn 100m)."
				}

CI 06	Check-in lần 2		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002
				{
				"success": false,
				"message": "Bạn đã check-in rồi."
				}

			], "lat": 10.123456, "lng": 106.123456 }	

CO 01	Check-out thành công	**HTTP method:** POST
		Path: http://localhost:5000/api/face/check-out
			Authorization: Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002
			], "lat": 10.123456,
			"lng": 106.123456
			}
				{
				"success": true,
				"message": "Check-out FaceID thành công",
				"data": {
				"docId": "oHnlk8LlMMdSEHgxeDoP_2025-12-07",
				"faceDist": 0,
				"gpsDist": 0
				}
				}

CO 02	Sai mặt		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			-0.991, 0.442, 0.330, -0.828
			], "lat": 10.123456,
			"lng": 106.123456
			}
				{
				"success": false,
				"message": "Khuôn mặt không khớp"
				}

CO 03	Thiếu GPS		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002]
			}
				{
				"success": false,
				"message": "Thiếu tọa độ GPS"
				}

CO 04	Ngoài vùng GPS		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002
			], "lat": 11.000000,
			"lng": 107.000000
			}
				{
				"success": false,
				"message": "Bạn đang ngoài khu vực làm việc (cách 136676m, giới hạn 100m)."
				}

CO 05	Check-out khi chưa check-in		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002],
			"lat": 10.123456,
			"lng": 106.123456
			}
				{
				"success": false,
				"message": "Bạn chưa check-in hôm nay)."
				}

CO 06	Check-out lần 2		**Authorization:** Bearer eyJhbGciOiJI...
			Content-Type: application/json
			Body:
			{
			"embedding": [
			0.121, 0.234, 0.556, -0.223, 0.112, 0.998, -0.445, 0.002
			], "lat": 10.123456,
			"lng": 106.123456
			}
				{
				"success": false,
				"message": "Bạn đã check-out rồi."
				}

3. Kết quả kiểm thử

a. Postman

- LOG-01 (Success): Đăng nhập đúng thông tin -> Trả về HTTP 200 kèm token, refreshToken và thông tin user đầy đủ
- LOG-02 (Fail): Đăng nhập sai password -> Trả về HTTP 400 và thông báo lỗi chính xác.
- ER-01: (Success): Gửi vector đặc trưng khuôn mặt (embedding array) kèm Token hợp lệ -> Trả về HTTP 200 "Đăng ký FaceID thành công".

- **Happy Path (Luồng thành công):**

- CI-01, CO-01: Input hợp lệ (Face Embedding khớp, GPS trong vùng cho phép) -> Trả về HTTP 200, ghi nhận checkInAt/checkOutAt, trả về docId

- **Validation & Error Handling (Luồng lỗi):**

- Chưa đăng ký (Unenrolled): CI-02 -> Chặn người dùng chưa có dữ liệu khuôn mặt (HTTP 400).
- Nhận diện sai (Face Mismatch): CI-03, CO-03 -> Embedding gửi lên không khớp với Database -> Trả về lỗi "Khuôn mặt không khớp" (HTTP 400).
- Ràng buộc GPS:

+ CI-04, CO-04 (Thiếu GPS): Hệ thống từ chối nếu không gửi tọa độ.

CI-05, CO-05 (Ngoài vùng): Tọa độ gửi lên nằm ngoài bán kính cho phép -> Trả về lỗi (HTTP 400).

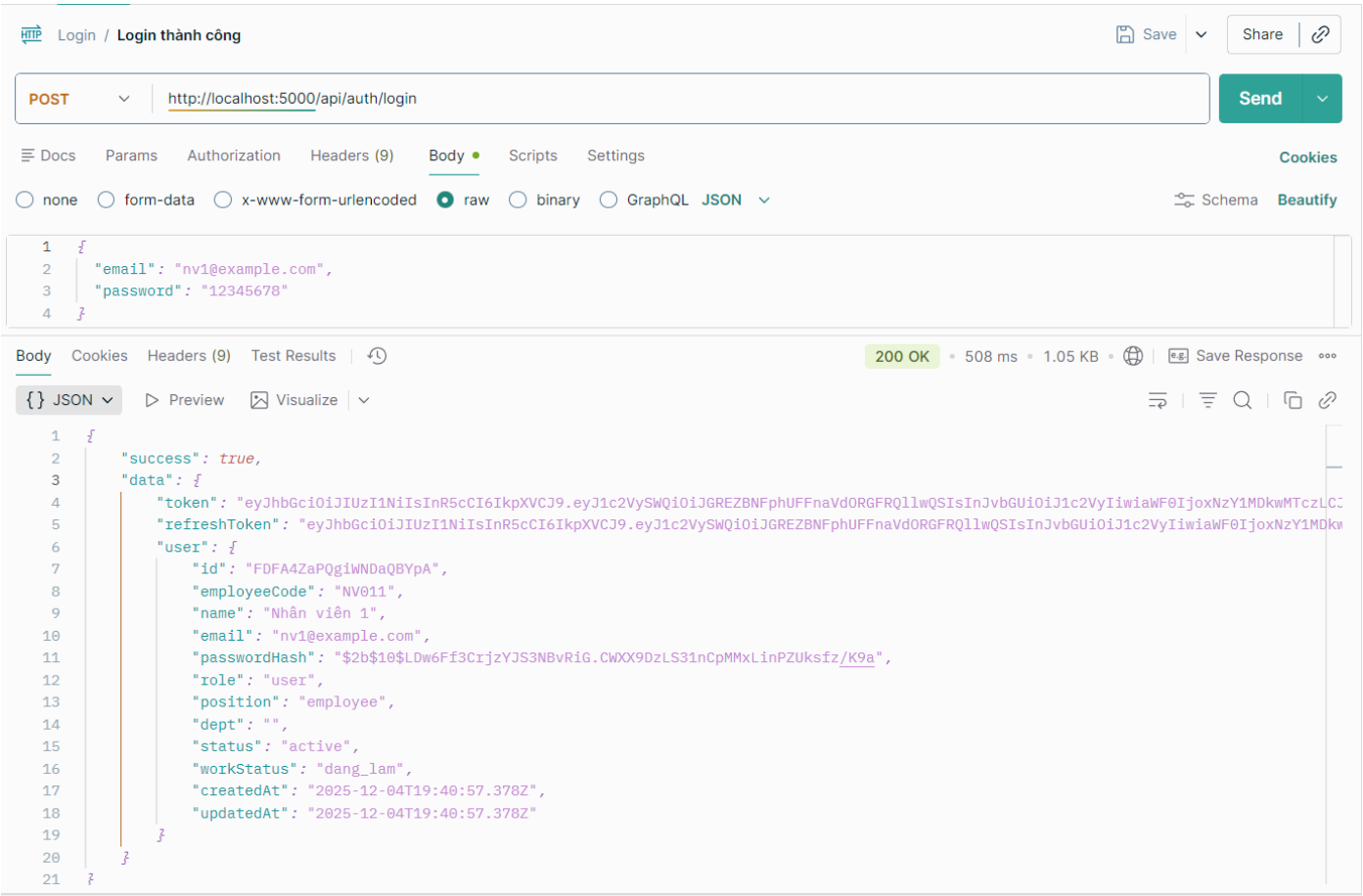
- **Logic trùng lặp (Duplicate/Sequence):**

  + CI-06: Check-in lần 2 trong ngày -> Báo lỗi "Đã check-in hôm nay".

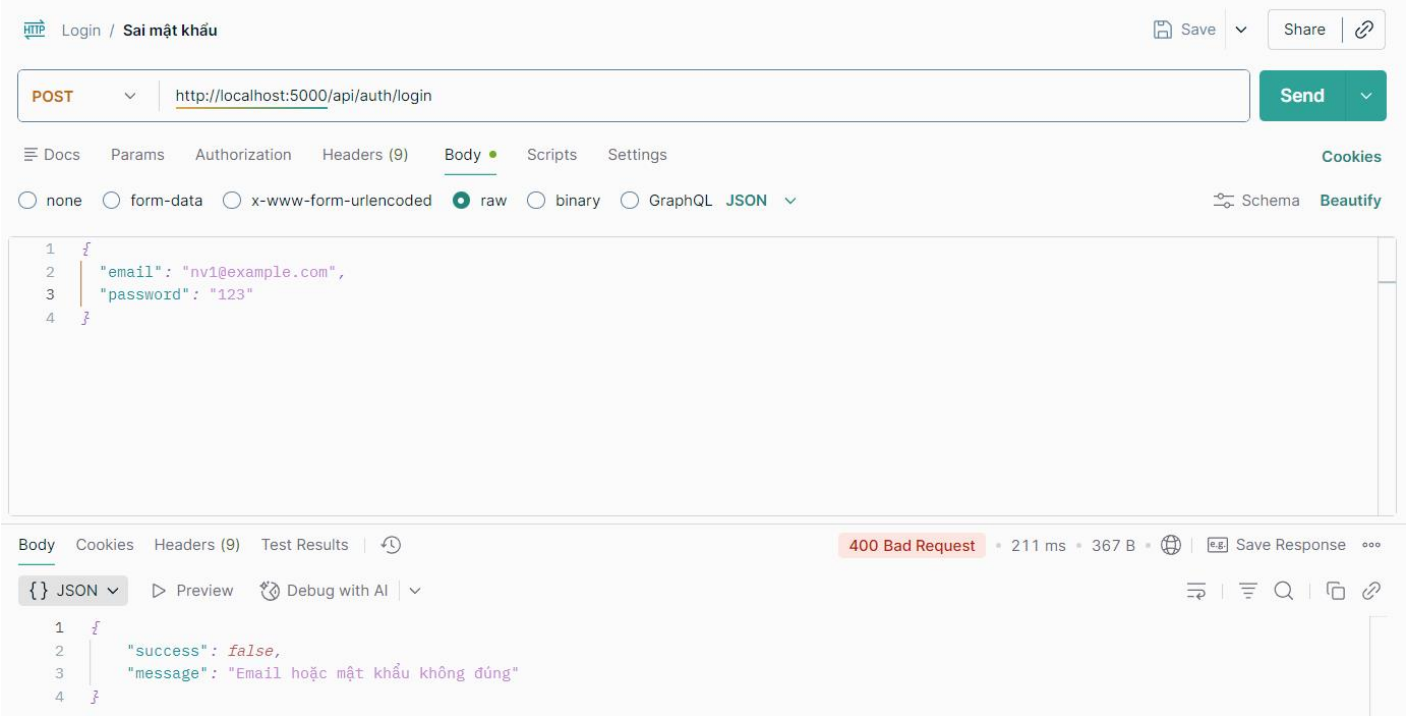
  + CO-06: Check-out khi chưa Check-in -> Báo lỗi logic nghiệp vụ.

  + CO-07: Check-out lần 2 -> Báo lỗi "Đã check-out rồi"

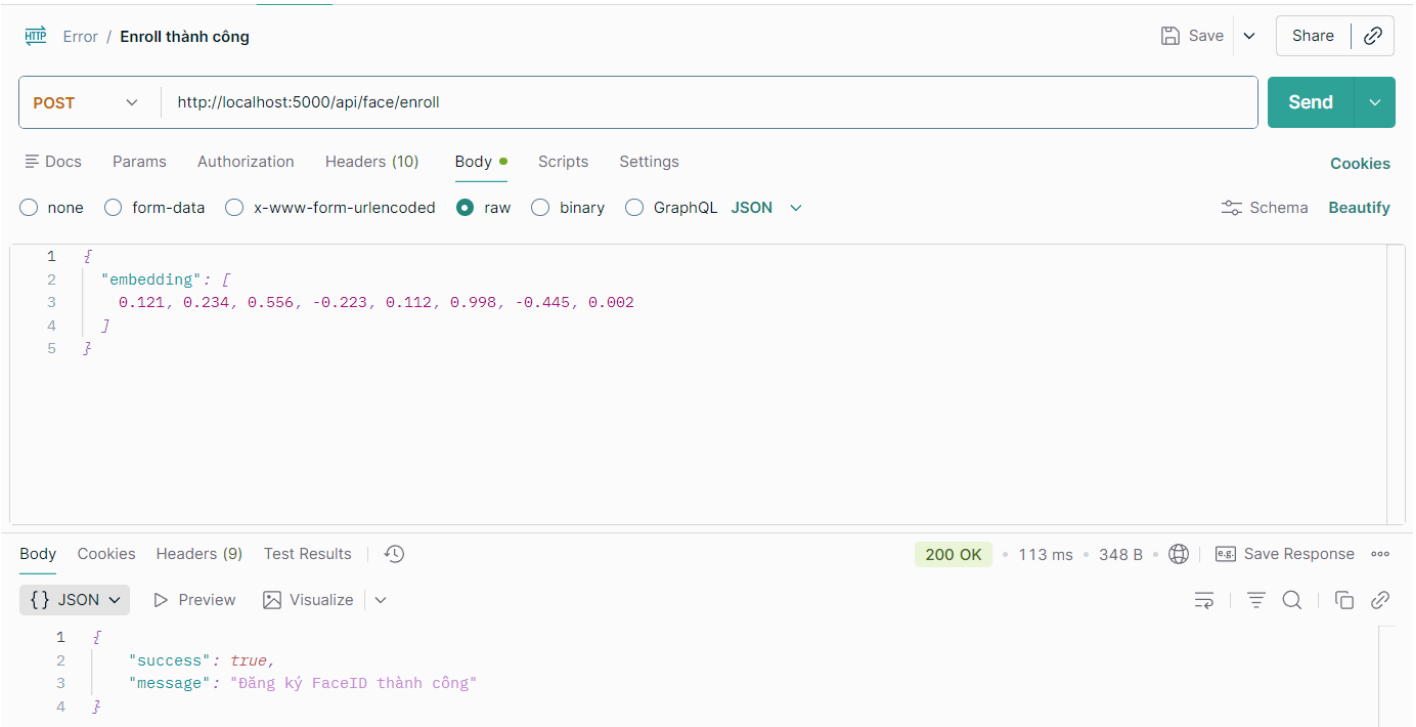
- ***Ảnh chụp màn hình hiển thị kết quả test***



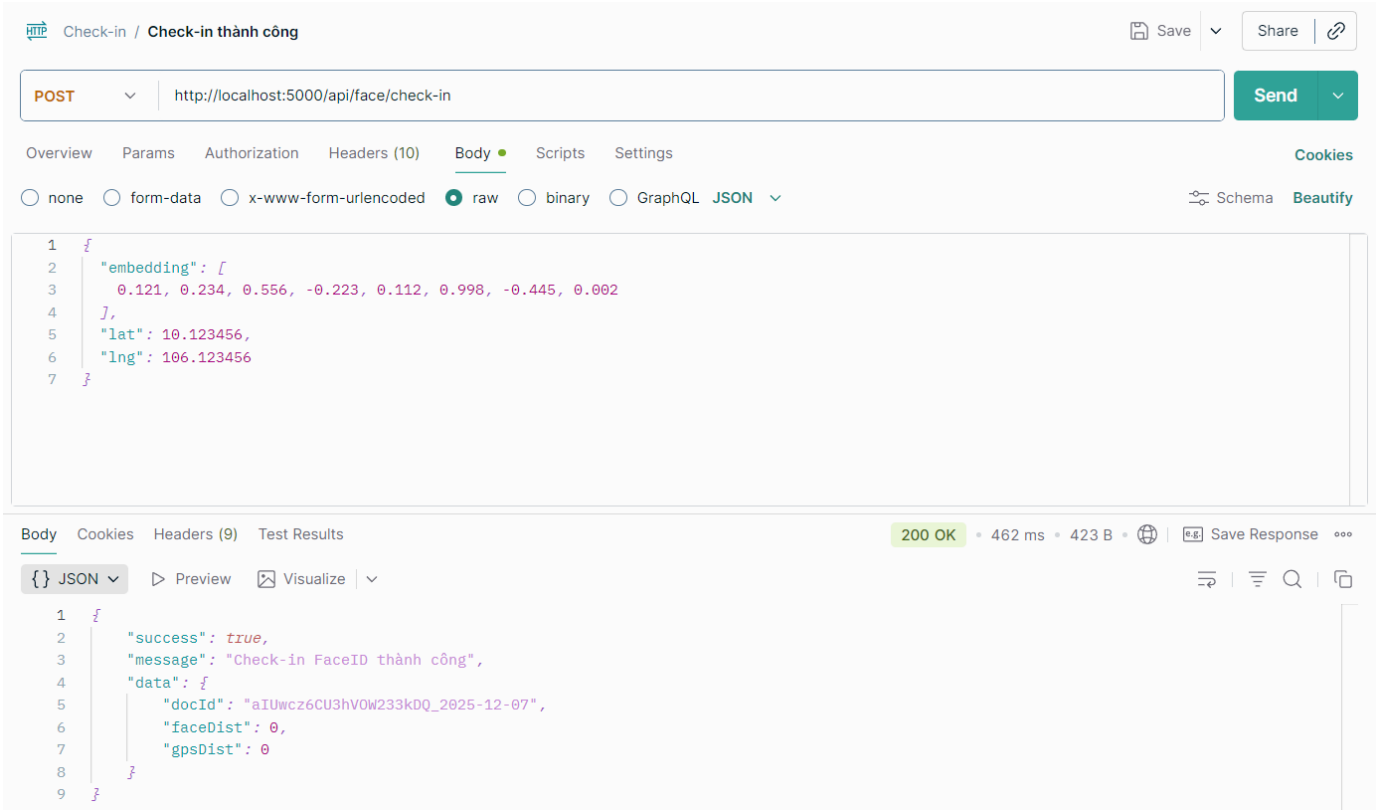
LOG 01 – Login thành công



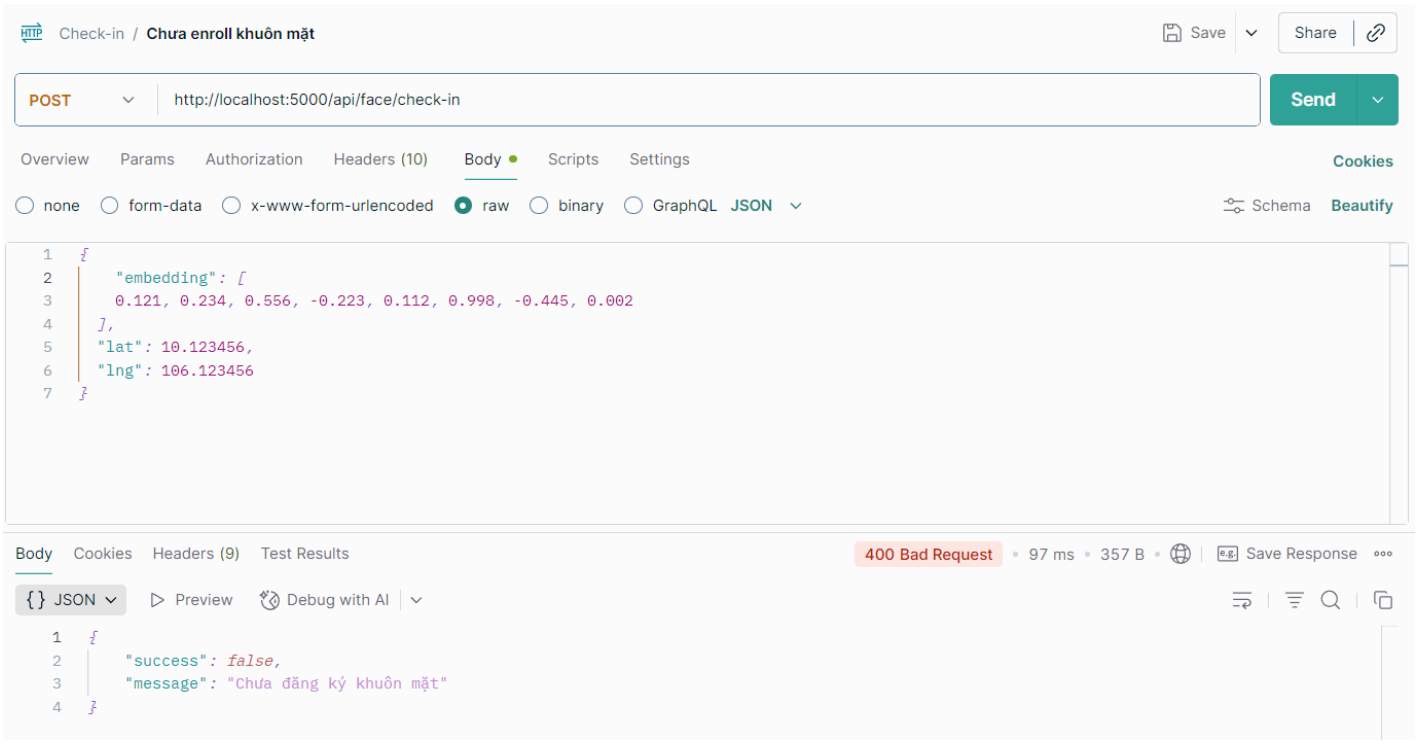
LOG 02 – Sai mật khẩu



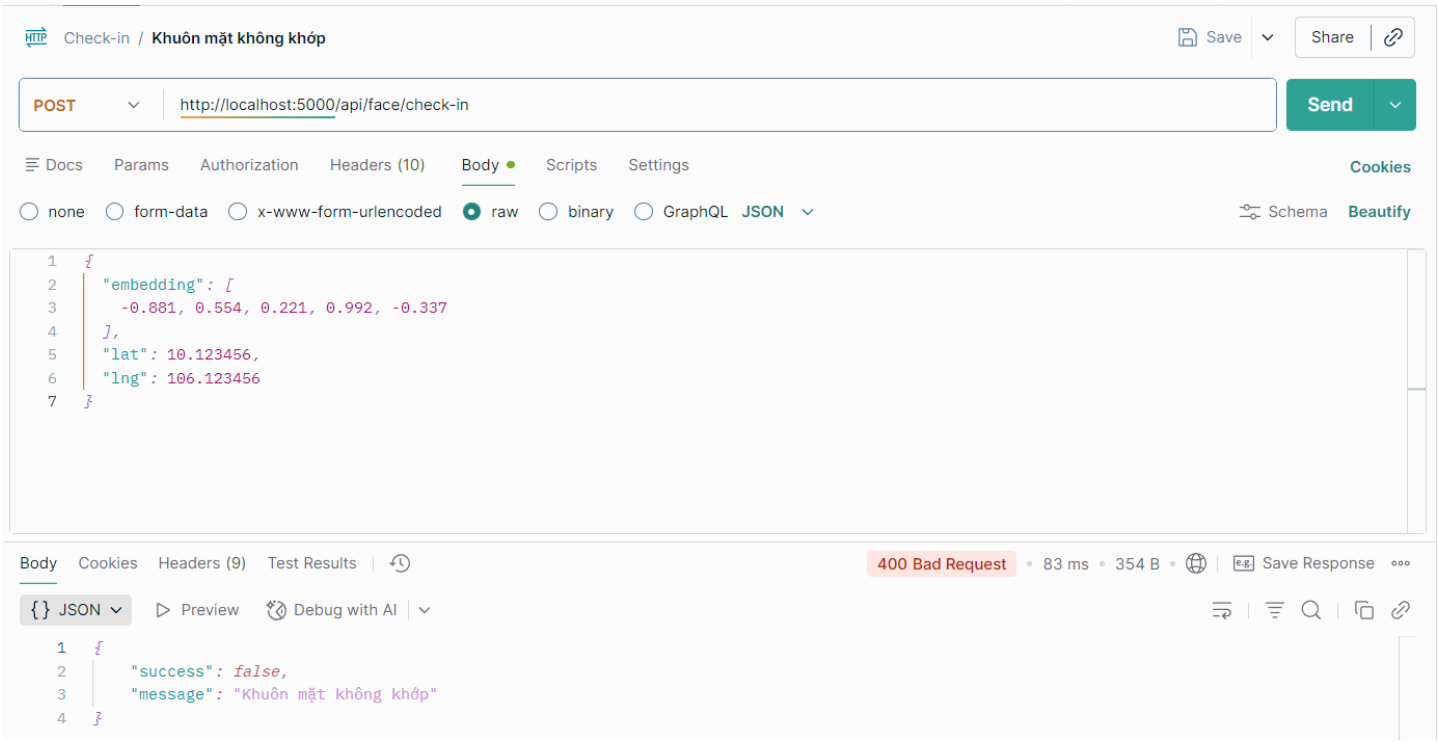
ER 01 – Enroll thành công



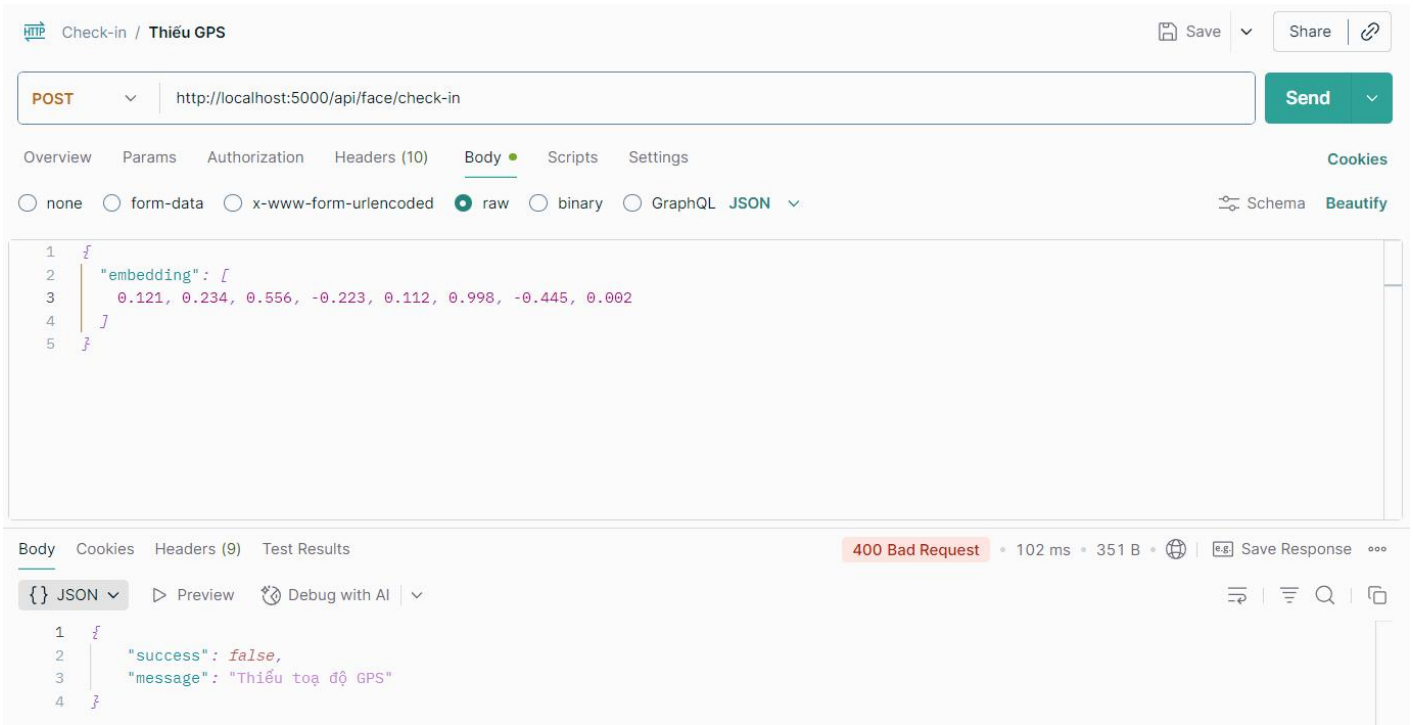
CI 01 – Check-in thành công



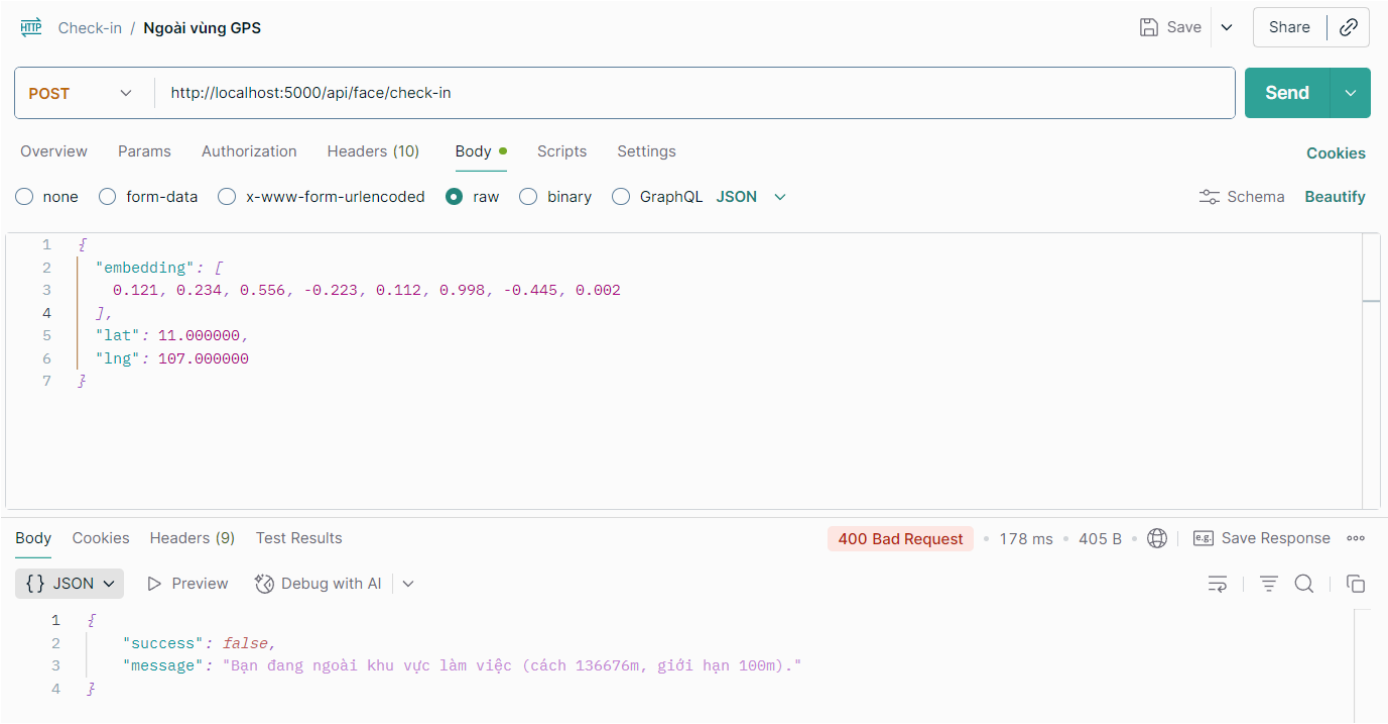
CI 02 – Chưa enroll mặt



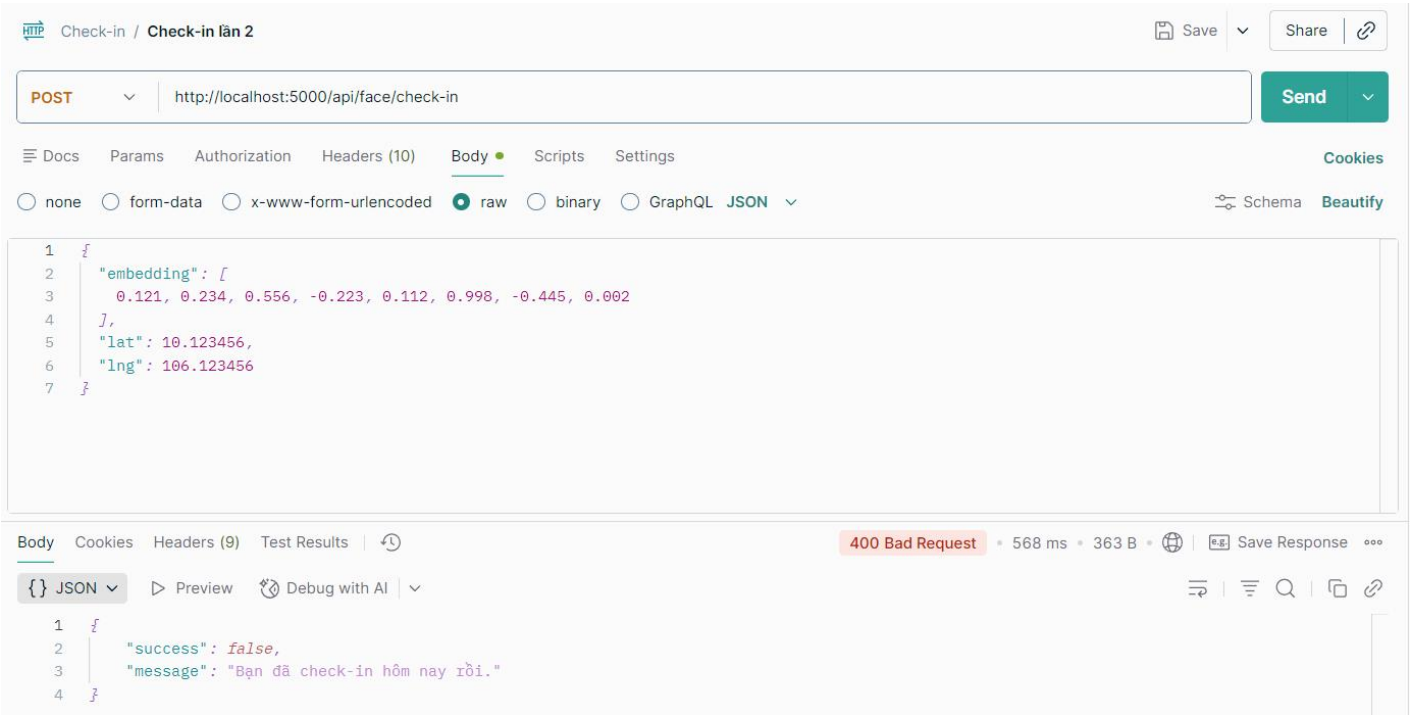
CI 03 – Khuôn mặt không khớp



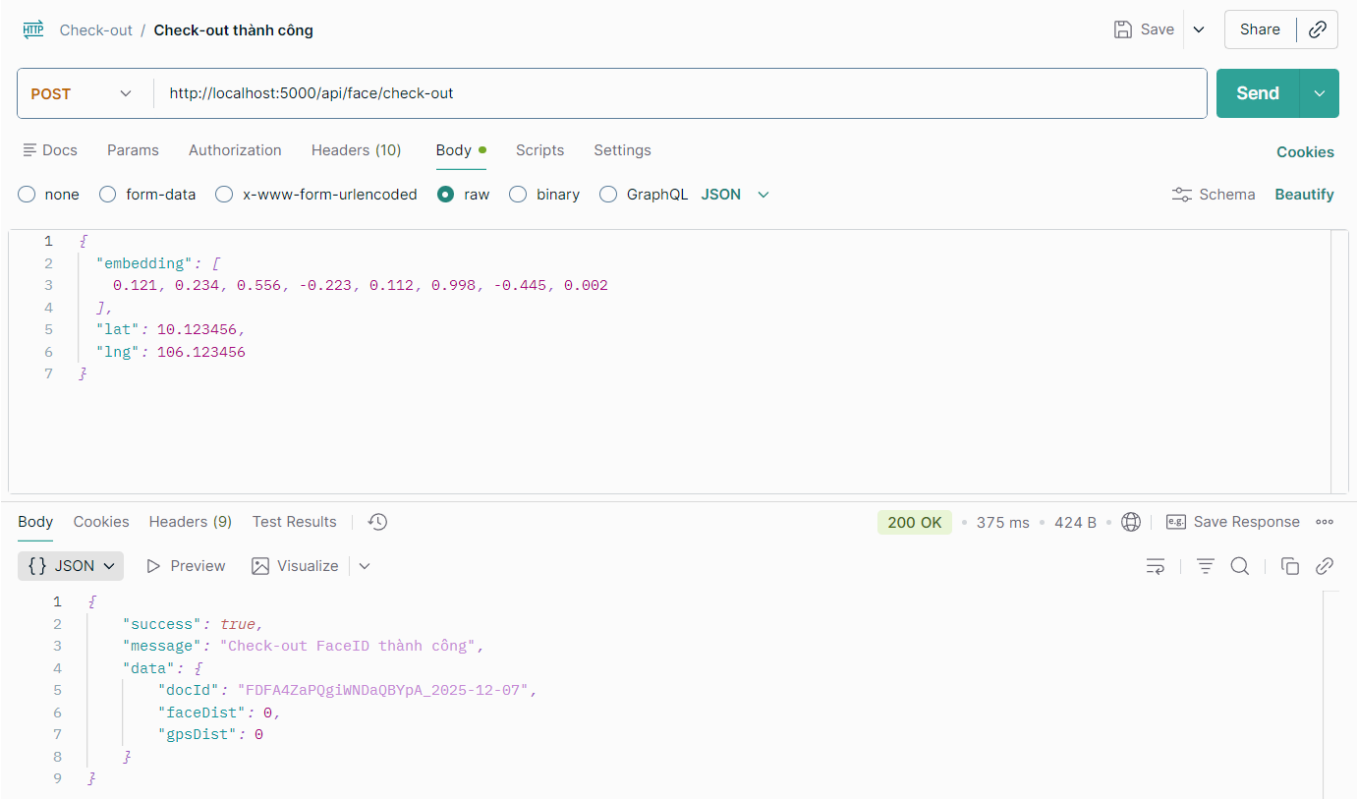
CI 04 – Thiếu GPS



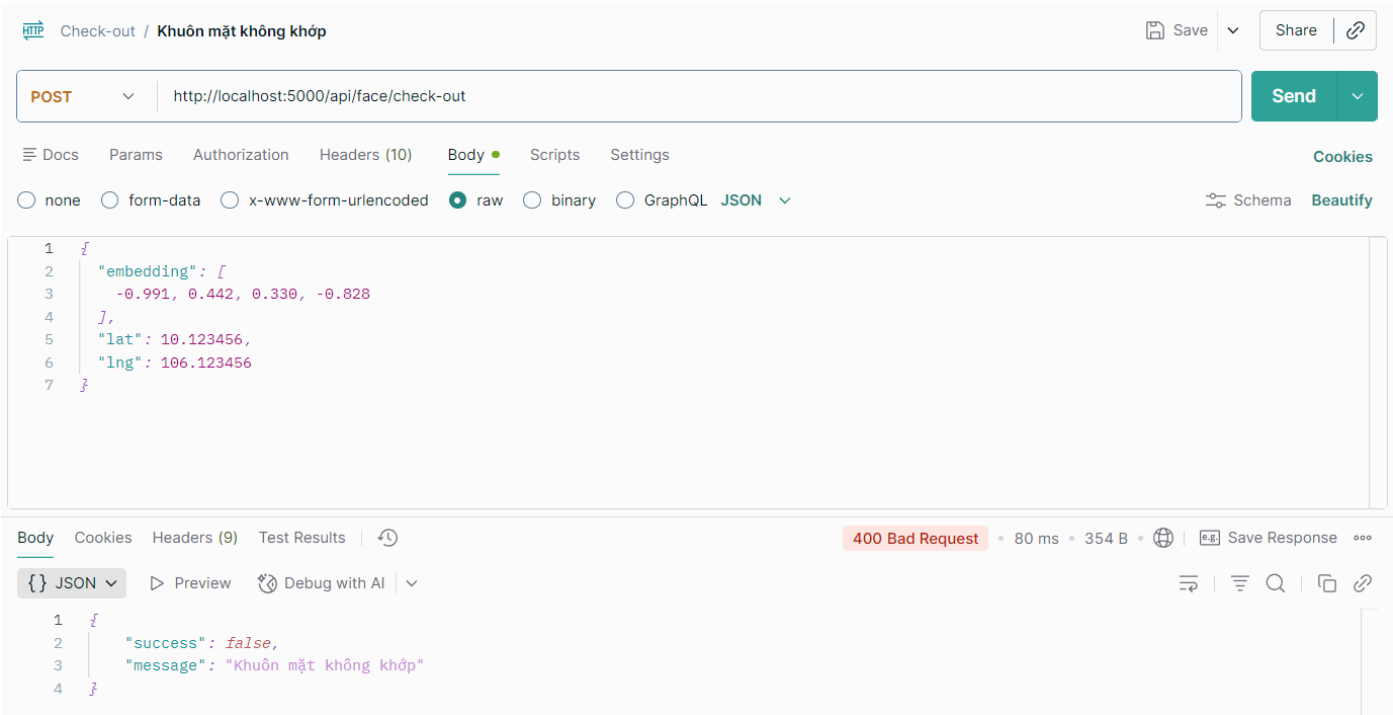
CI 05 – Ngoài vùng GPS



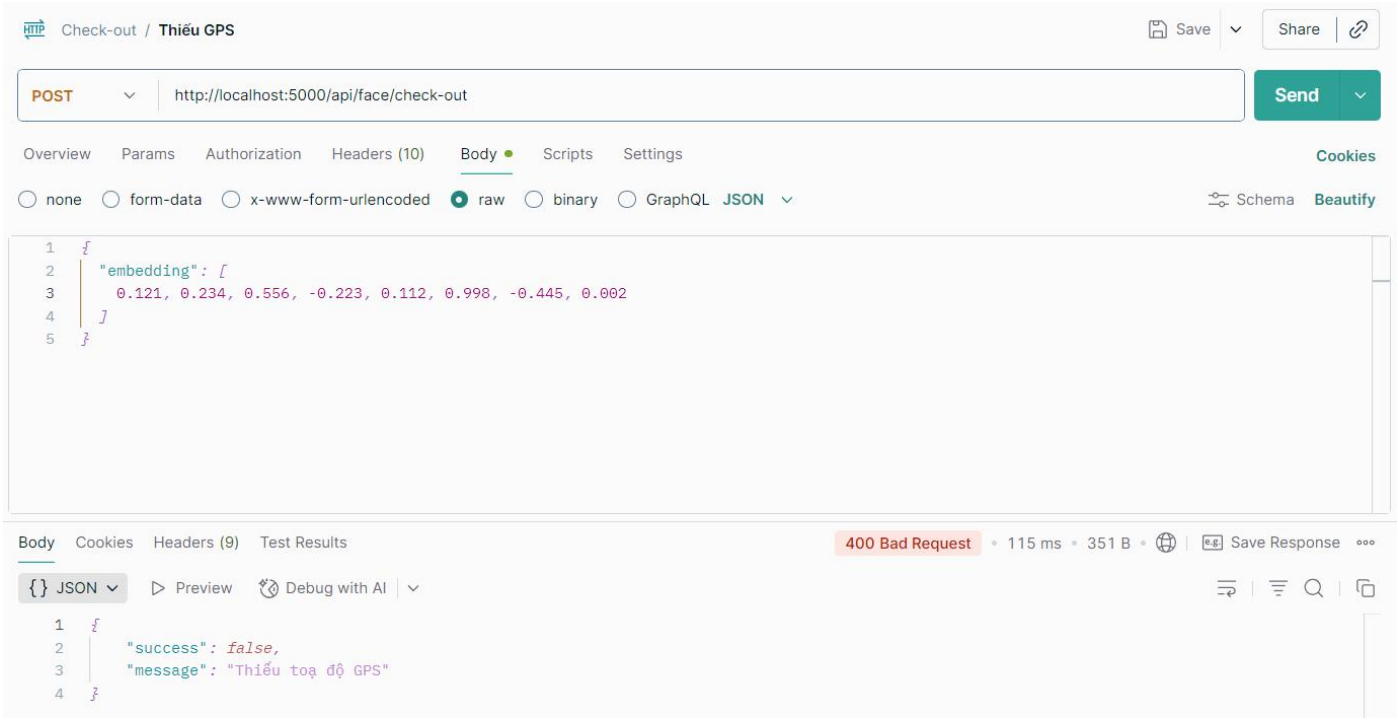
CI 06 – Check-in lần 2



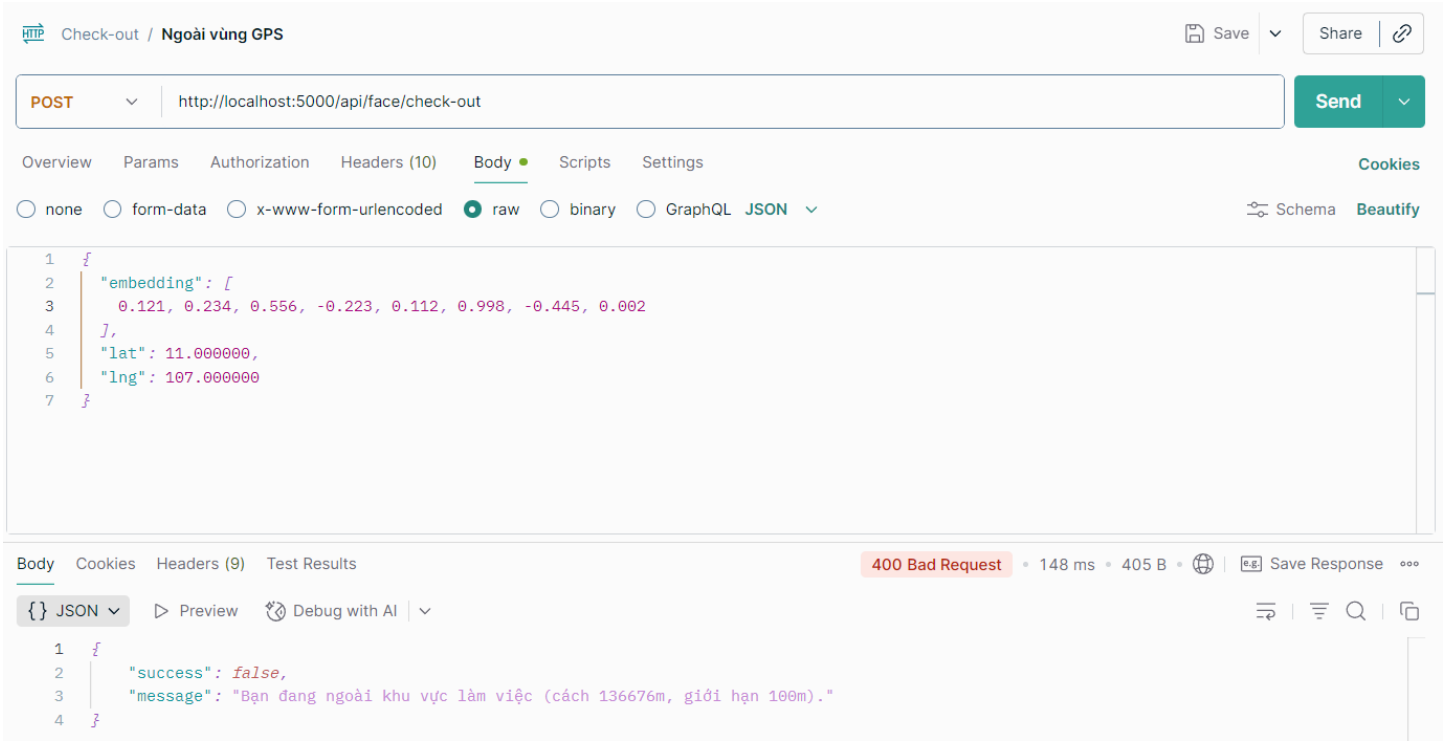
CO 01 – Check-out thành công



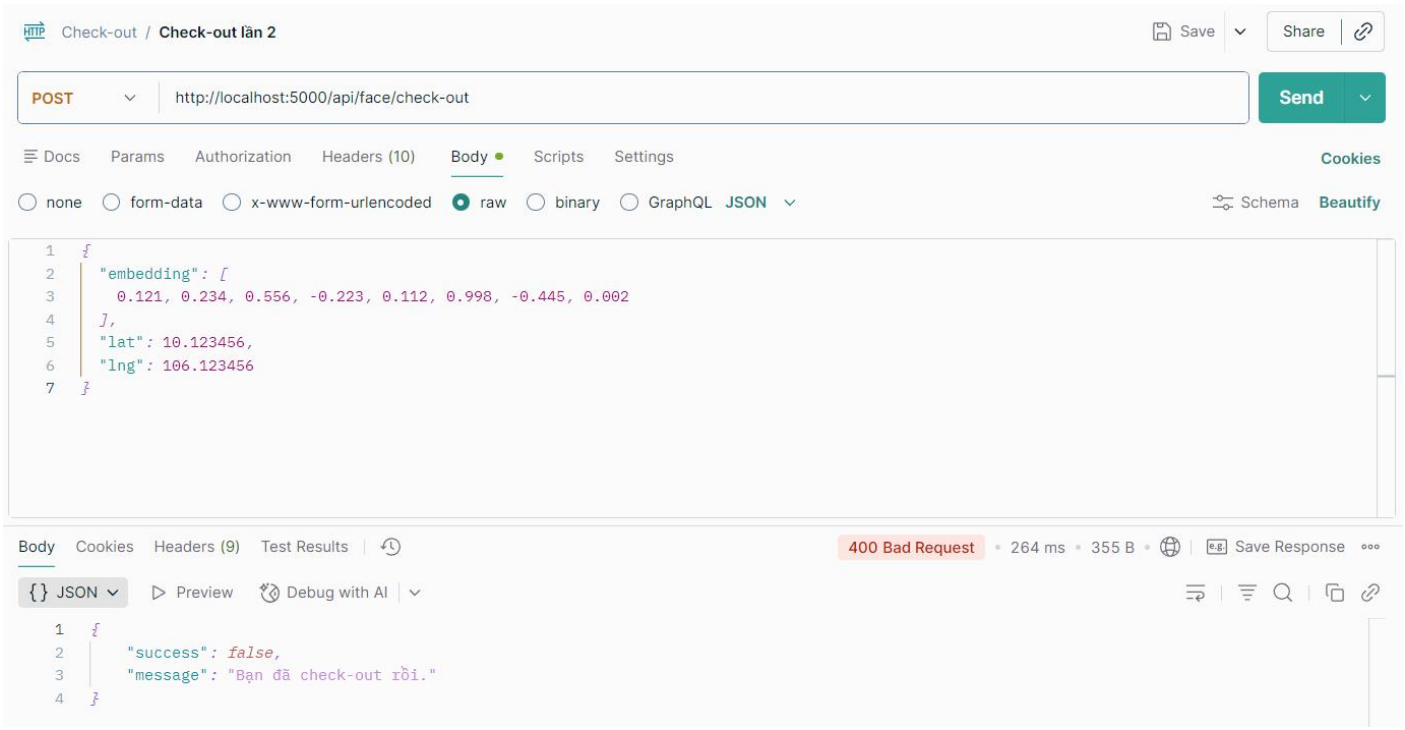
CO 02 – Khuôn mặt không khớp



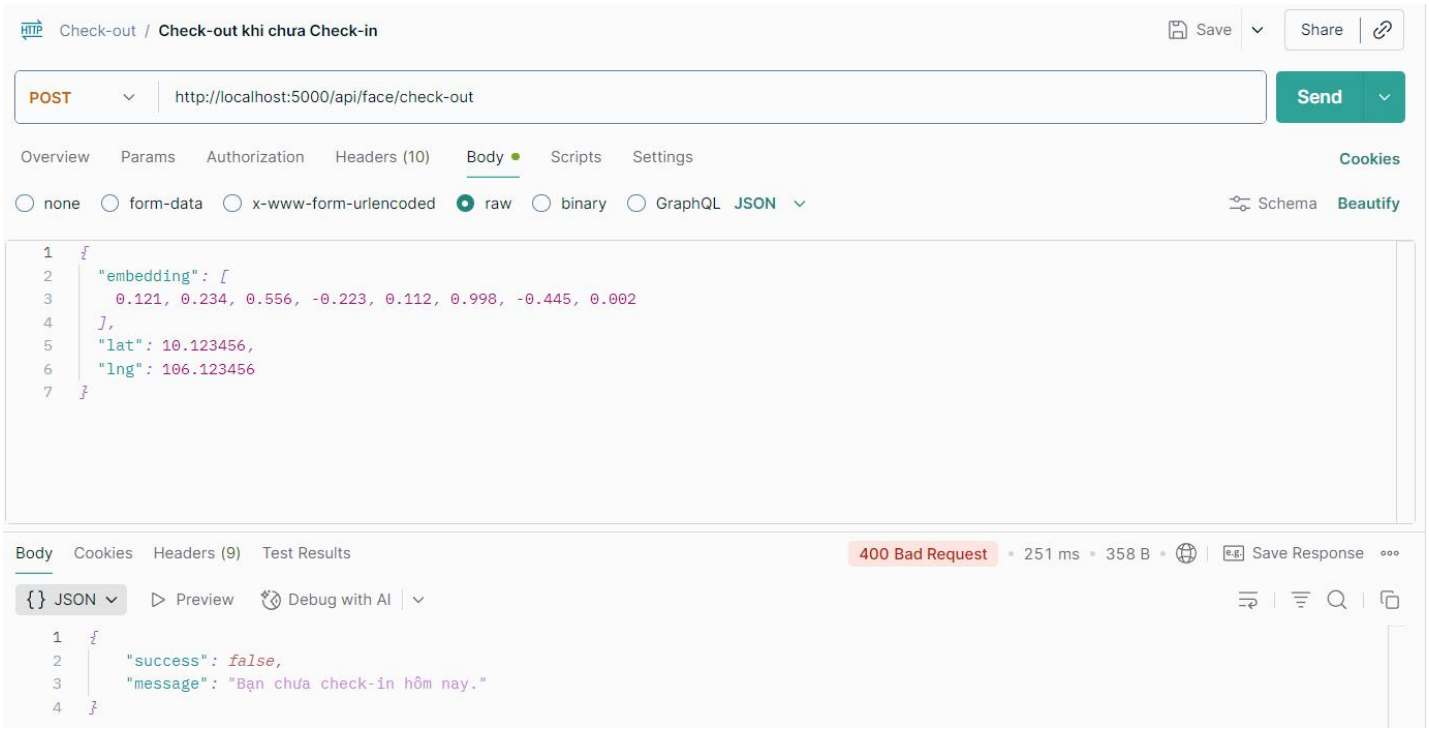
CO 03 – Thiếu GPS



CO 04 – Ngoài vùng GPS



CO 05 – Check-out lần 2



CO 06 – Check-out khi chưa Check-in

b. Jest

- **attendance.service.test.js**

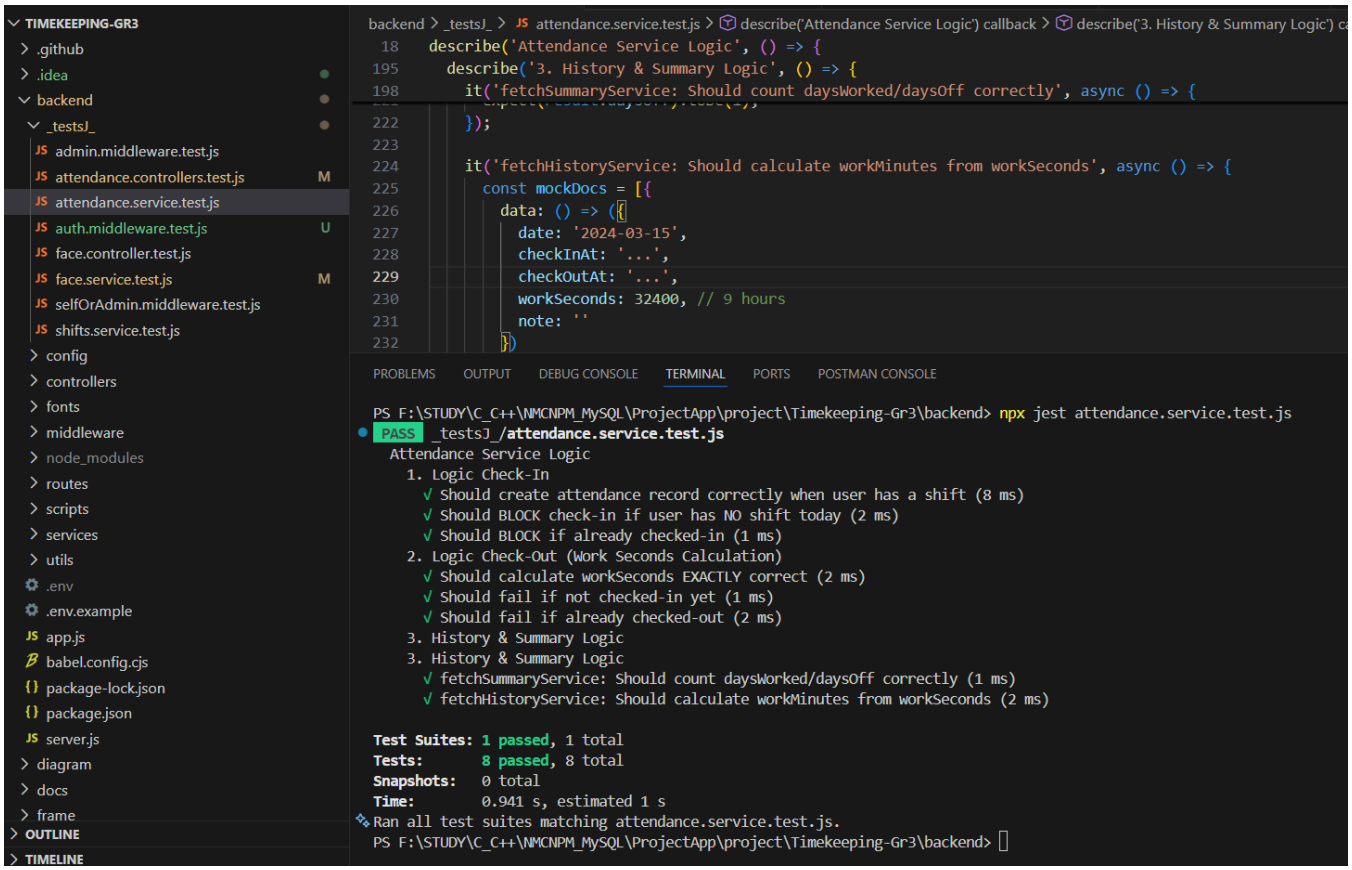
- **Chức năng:** Kiểm thử logic nghiệp vụ chấm công (Check-in/Check-out) và thống kê.
- **Đối tượng test**: handleCheckInService, handleCheckOutService, fetchSummaryService, v.v.
- **Phân tích kịch bản:**

  + Trạng thái chấm công: Ngăn chặn check-in 2 lần trong ngày, hoặc check-out khi chưa check-in.

  + Tính toán: Logic tính daysWorked (ngày làm) và daysOff (ngày nghỉ) dựa trên dữ liệu giả lập từ Firebase

  + Lịch làm việc: Logic phức tạp để tổng hợp dữ liệu từ 4 nguồn (Ca làm, Chấm công, Yêu cầu, Nghỉ phép) để tạo ra trạng thái hiển thị trên lịch (ví dụ: checked-full, absent, pending-request).

- **Kết quả:**



- **attendance.controllers.test.js**

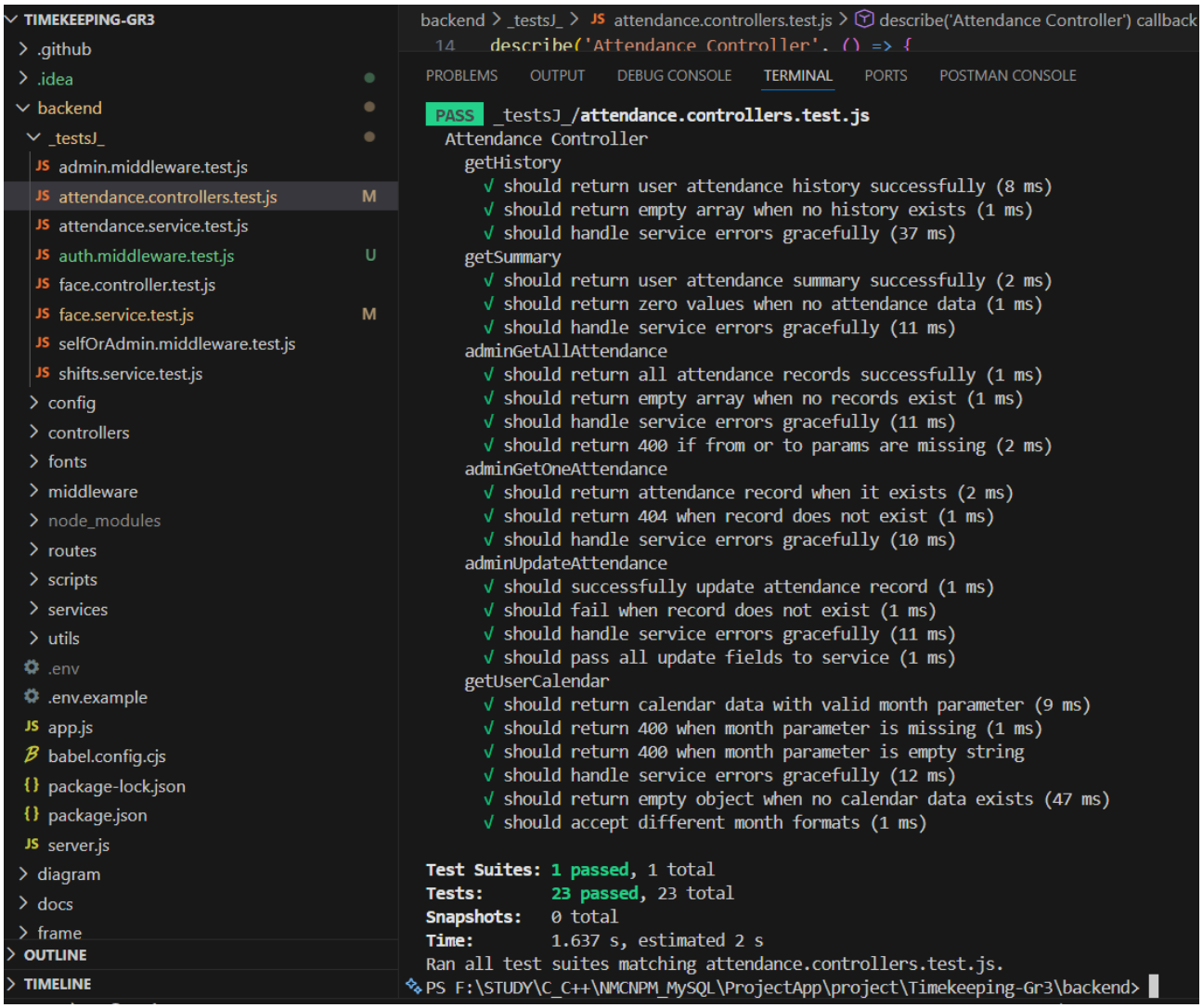
- **Chức năng:** Kiểm thử các hàm điều khiển liên quan đến quản lý và xem báo cáo chấm công.

- **Đối tượng test**: getHistory, getSummary, adminGetAllAttendance, adminUpdateAttendance, getUserCalendar
- **Phân tích kịch bản:**

  + Xử lý dữ liệu: Kiểm tra việc controller nhận dữ liệu từ service và trả về JSON đúng cấu trúc.

  + Query Params: Test logic xử lý tham số trên URL, ví dụ: bắt lỗi nếu thiếu tham số month khi gọi API lấy lịch làm việc (getUserCalendar).

  + Quyền Admin: Test chức năng admin cập nhật chấm công (adminUpdateAttendance), đảm bảo dữ liệu sửa đổi được gửi đúng xuống service.
- **Kết quả:**



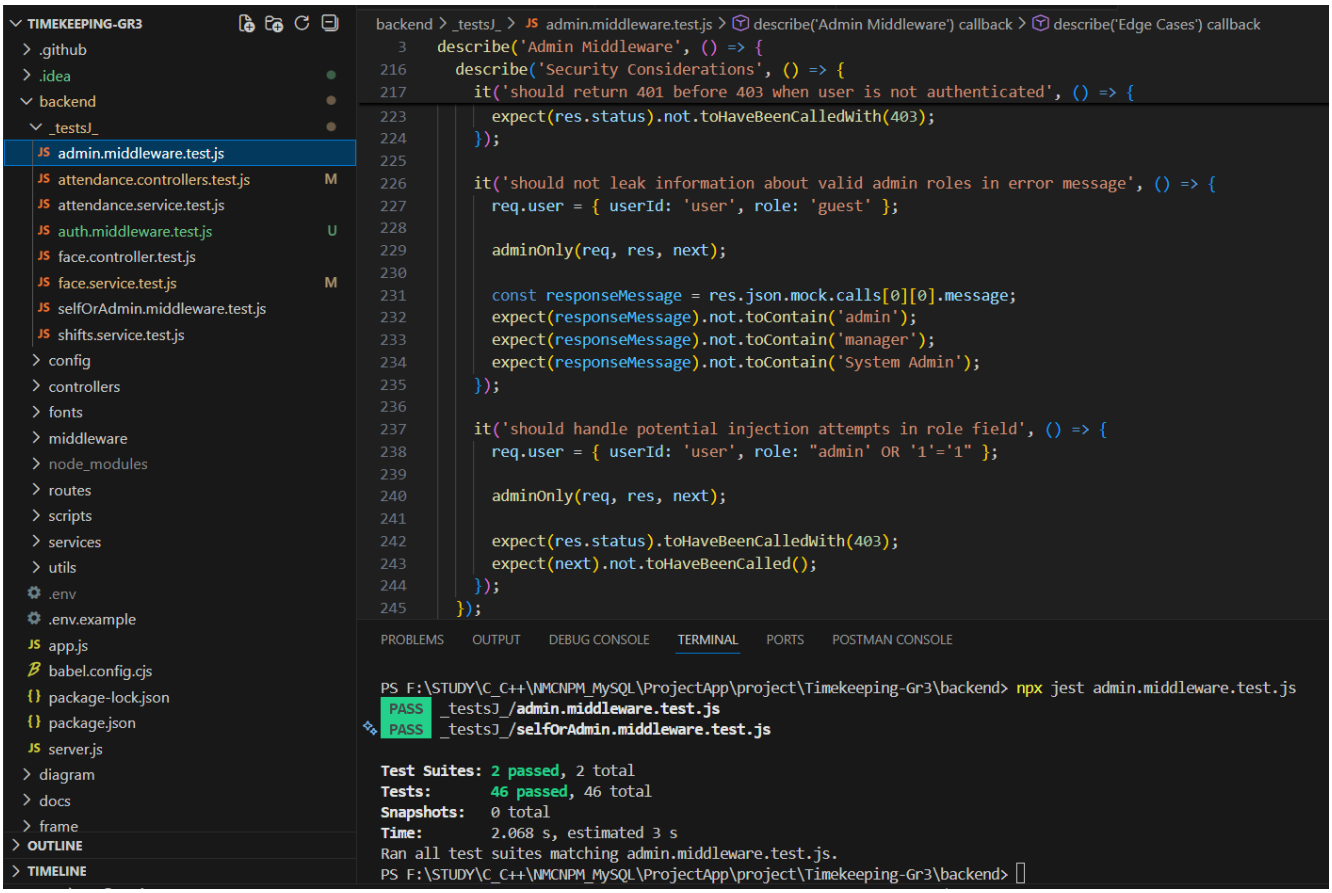
- **admin.middleware.test.js**

- **Chức năng:** Kiểm thử lớp bảo vệ quyền truy cập dành riêng cho Admin.
- **Đối tượng test:** Middleware adminOnly.
- **Phân tích kịch bản:**

  + Role hợp lệ: Chấp nhận 3 role: 'admin', 'System Admin', 'manager'

  + Chặn truy cập: Từ chối các role 'user', 'employee', hoặc user không tồn tại (null/undefined)

  + Edge Cases (Trường hợp biên): Đây là file test rất kỹ về bảo mật, kiểm tra các tình huống như: tên role viết sai hoa/thường (Admin), role có khoảng trắng thừa (admin), hoặc nỗ lực tấn công injection (admin' OR '1'='1).
- **Kết quả:**



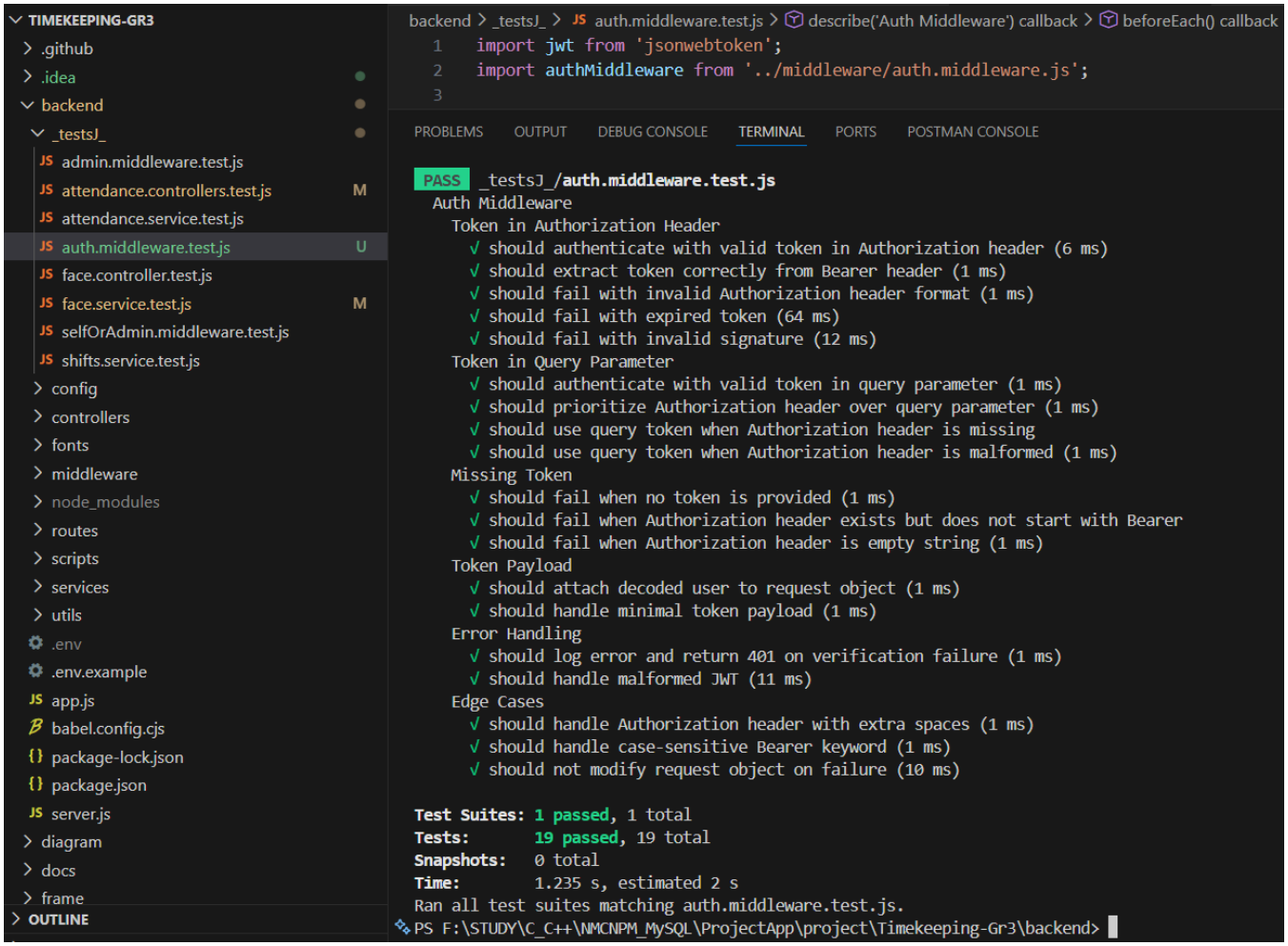
- **auth.middleware.test.js**
- **Chức năng:** Kiểm thử cơ chế xác thực người dùng qua JWT.
- **Đối tượng test:** Middleware authMiddleware.
- **Phân tích kịch bản:**

+ Nguồn Token: Kiểm tra khả năng lấy token từ cả Header (Authorization: Bearer ...) và Query Param (?token=...). Ưu tiên Header hơn Query.

+ Giải mã JWT: Sử dụng jest.mock để giả lập jsonwebtoken.verify. Test các lỗi: Token hết hạn, sai chữ ký, format header sai (thiếu từ khóa 'Bearer').

+ Gán User: Đảm bảo sau khi xác thực thành công, thông tin user được gán vào req.user để các middleware sau sử dụng.

- **Kết quả:**



- **selfOrAdmin.middleware.test.js**

- **Chức năng:** Kiểm thử quyền riêng tư dữ liệu (Data Privac

- **Đối tượng test**: Middleware selfOrAdmin.

- **Phân tích kịch bản:**

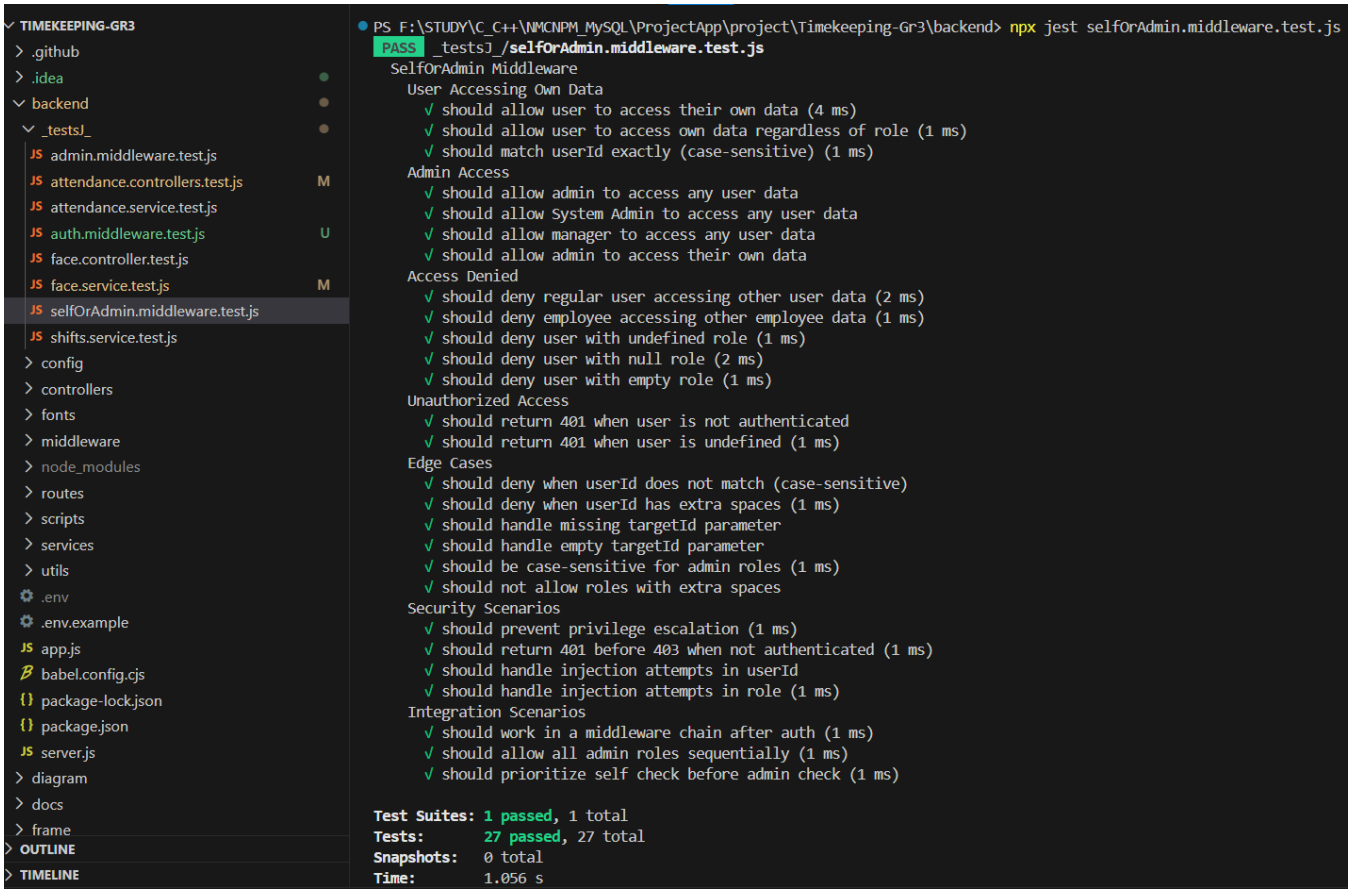
+ Quyền chính chủ: User A được phép xem dữ liệu của User A (so sánh req.user.userId với req.params.userId)

+ Quyền Admin: Admin được phép xem dữ liệu của bất kỳ ai.

+ Chặn truy cập chéo: User A không được xem dữ liệu của User B.

+ Bảo mật: Kiểm tra kỹ tính Case-sensitive (phân biệt hoa thường) của ID và ngăn chặn các nỗ lực leo thang đặc quyền.

- **Kết quả:**



- **face.controller.test.js**

- **Chức năng:** Kiểm thử các hàm điều khiển việc nhận diện khuôn mặt và chấm công bằng khuôn mặt

- **Đối tượng test**: Hàm enrollFace (đăng ký), faceCheckIn (vào), faceCheckOut (ra).

- **Phân tích kịch bản:**

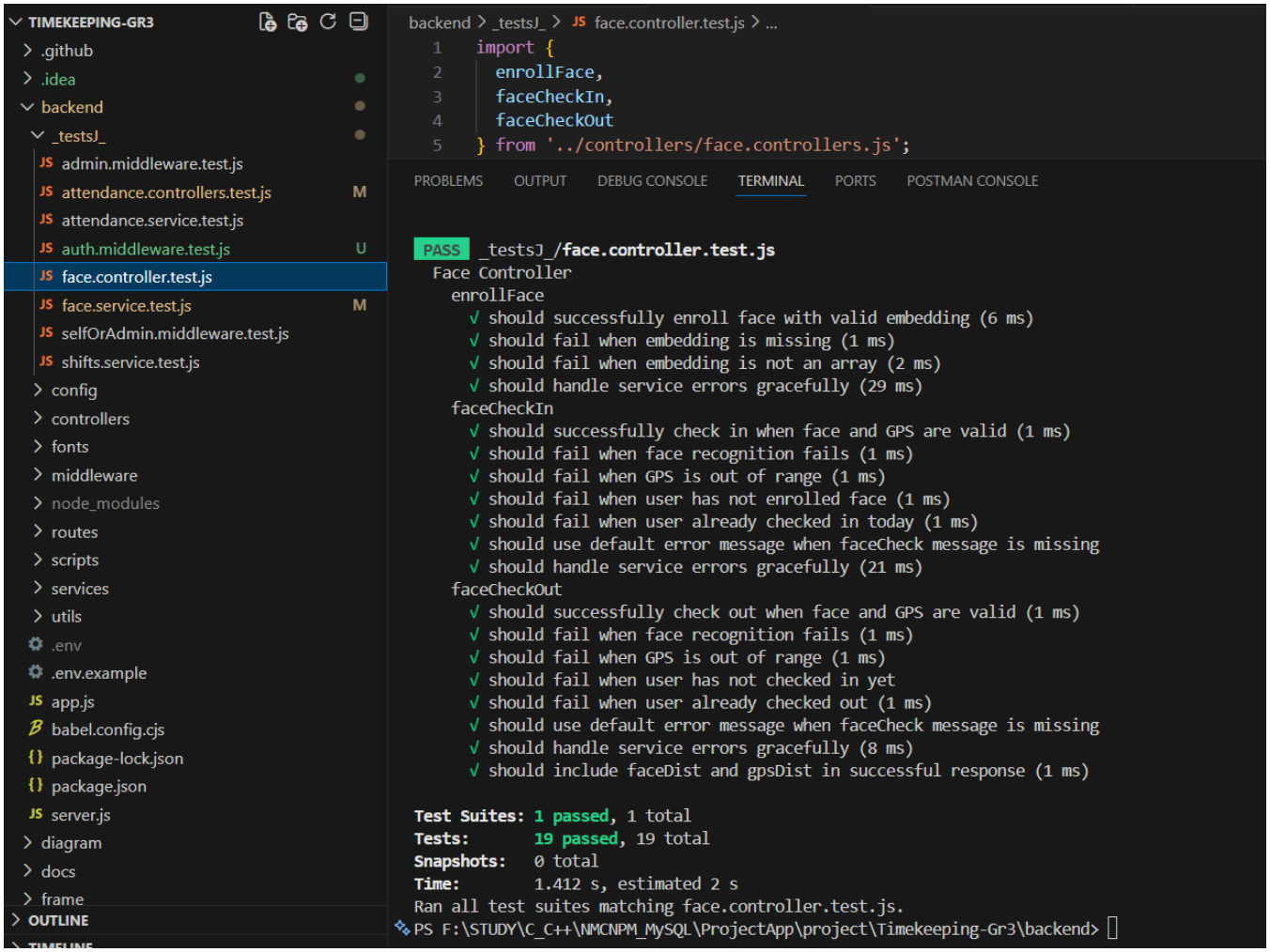
+ Mocking: Giả lập toàn bộ faceServices và attendanceService để cô lập logic của controller.

+ Validation: Kiểm tra kỹ các trường hợp thiếu dữ liệu đầu vào (thiếu embedding, embedding không phải là mảng)

+ Phản hồi HTTP: Đảm bảo controller trả về đúng mã lỗi: 200 (thành công), 400 (lỗi dữ liệu/logic như GPS xa, mặt không khớp), 500 (lỗi server).

+ Dữ liệu trả về: Kiểm tra response check-in/out có chứa đủ thông tin quan trọng như khoảng cách khuôn mặt (faceDist) và khoảng cách GPS (gpsDist) hay không.

- **Kết quả:**



- **face.service.test.js**

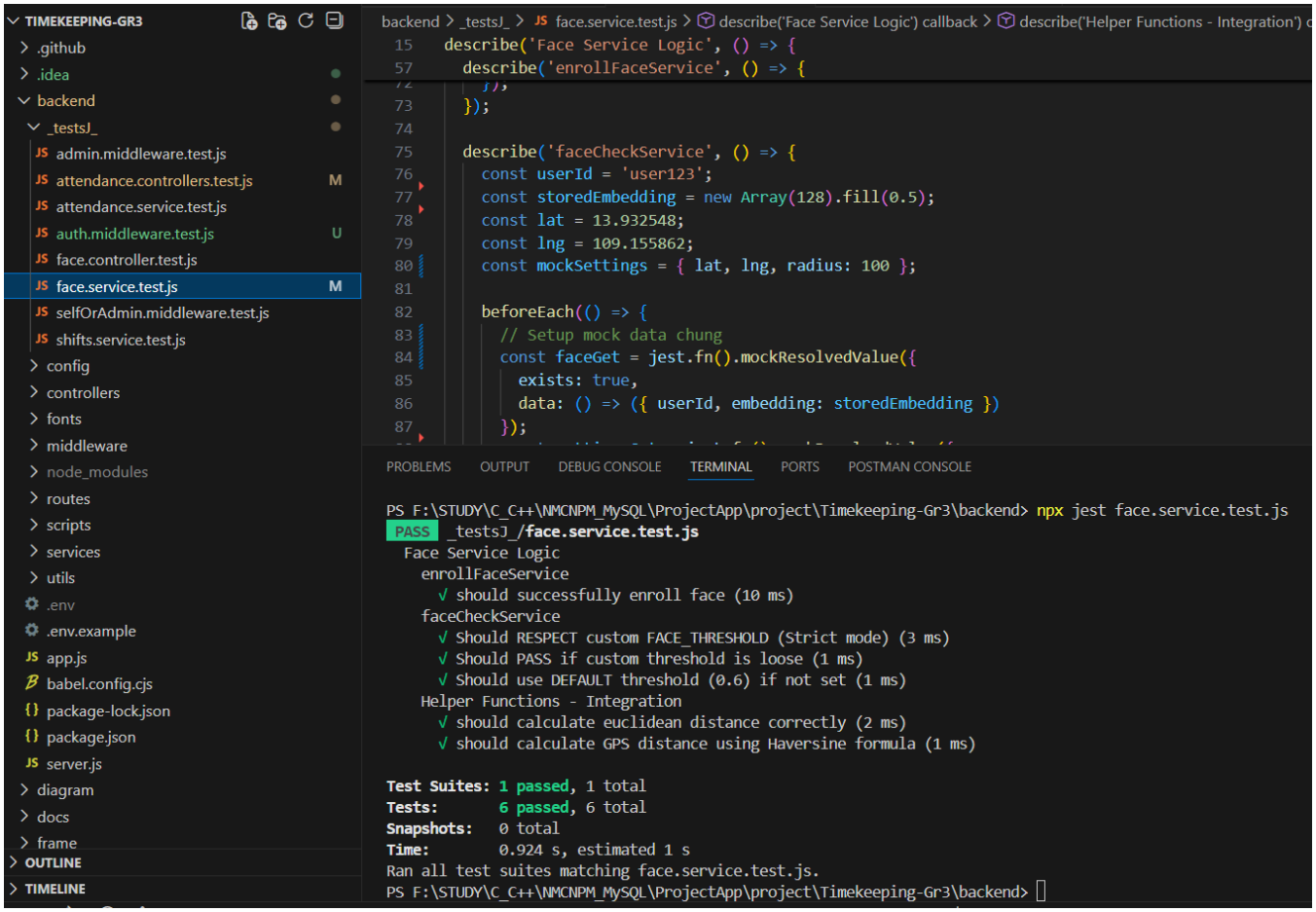
- **Chức năng:** Kiểm thử logic nghiệp vụ cốt lõi về xử lý vector khuôn mặt và vị trí địa lý.
- **Đối tượng test:** enrollFaceService và faceCheckService
- **Phân tích kịch bản:**

+ Thuật toán: Kiểm tra logic so khớp khuôn mặt (tính khoảng cách Euclidean giữa 2 vector 128 chiều).

+ Cấu hình môi trường: Test việc đọc biến FACE_THRESHOLD từ process.env. Test này thử nghiệm việc hạ thấp threshold để xem logic so khớp có thất bại đúng như mong đợi không.

+ GPS: Giả lập tọa độ GPS để tính toán khoảng cách (công thức Haversine), đảm bảo người dùng chấm công trong bán kính cho phép.

- **Kết quả:**



- **shifts.service.test.js**

- **Chức năng:** Kiểm thử logic quản lý ca làm việc và phân ca.

- **Đối tượng test:** CRUD Shift, assignShiftService, getEmployeesInShiftService.

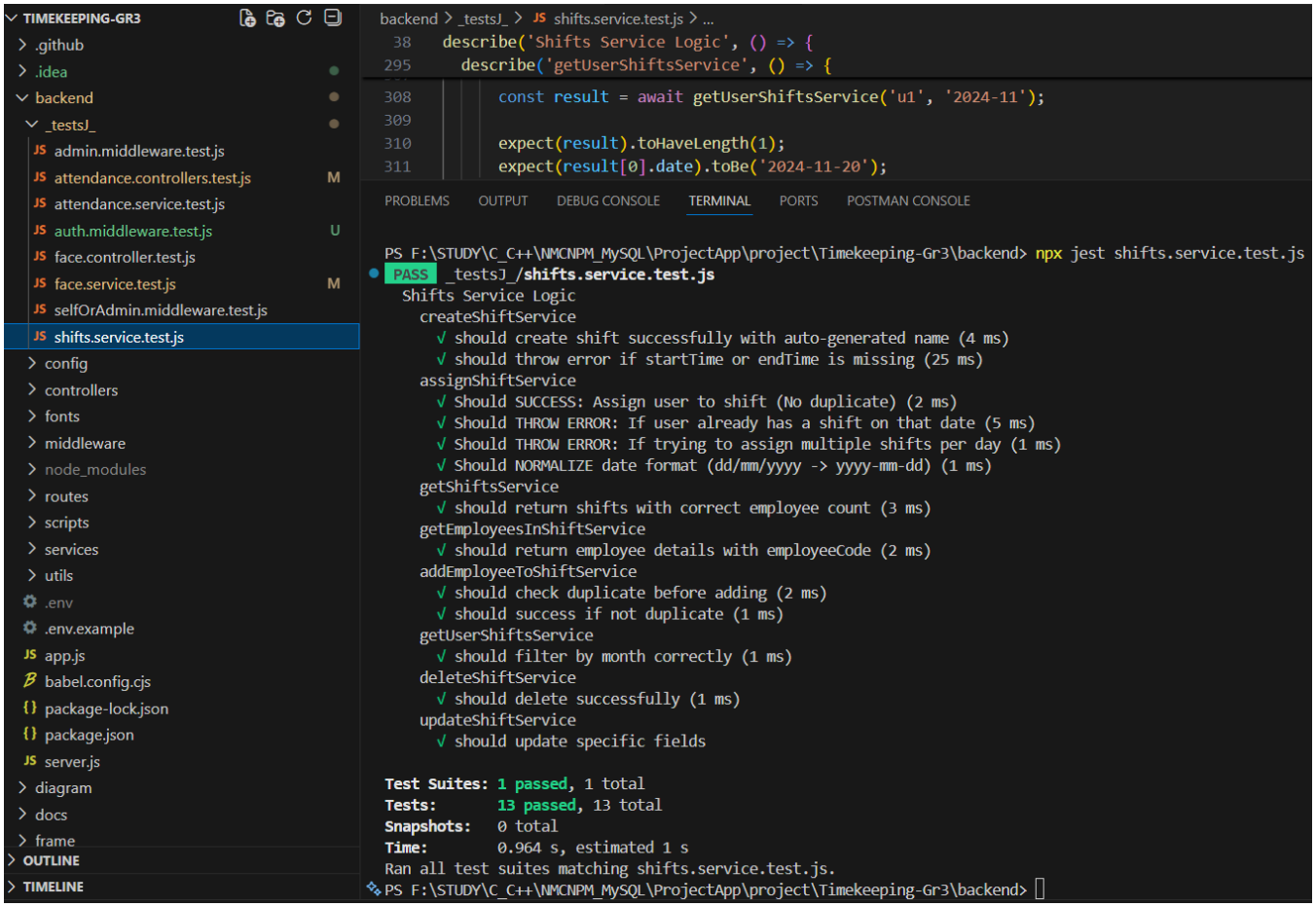
- **Phân tích kịch bản:**

+ Tự động hóa: Test việc tự động sinh mã ca (CA001) và tên ca nếu người dùng không nhập.

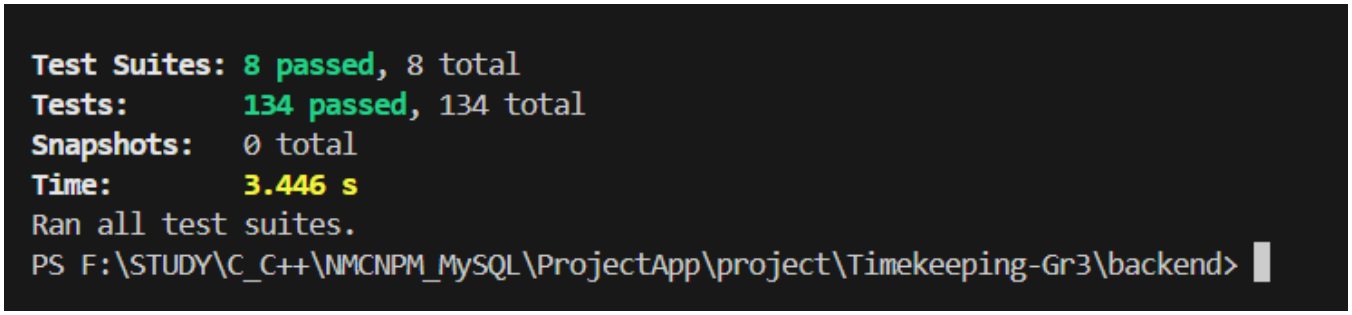
+ Batch Processing: Kiểm tra logic gán ca hàng loạt (nhiều user, nhiều ca) xem có gọi đúng hàm batch.commit() của Firestore không.

+ Validation: Kiểm tra định dạng ngày tháng (tự động convert dd/mm/yyyy sang yyyy-mm-dd) và bắt lỗi trùng lặp nhân viên trong ca.

- **Kết quả:**



- Kết quả chạy cho 8 trường hợp test trên: **npm run dev**



c. Selenium

- **Checkin.test.js**

- **Mục tiêu chính**: Kiểm tra quy trình chấm công vào (Check-in) từ giao diện người dùng, xác nhận luồng dữ liệu thành công từ Frontend đến Backend và ghi nhận vào lịch sử
- **Vai trò người dùng:** Nhân viên (giahan1835@gmail.com).
- **Các bước thực hiện (Flow):**

  + Cấu hình: Khởi tạo WebDriver cho trình duyệt Microsoft Edge với các tùy chọn để tự động cấp quyền Camera (--use-fake-ui-for-media-stream) và sử dụng video giả (--use-fake-device-for-media-stream).

  + Đăng nhập: Đăng nhập thành công bằng tài khoản nhân viên.

  + Thao tác: Điều hướng đến Tab Home và nhấn nút "Face Check-in" (#btnFaceCheckin).

  + Kiểm tra UI/Camera:

  . Đợi modal FaceID (#faceModal[open]) hiển thị.

  . Đợi video camera (#faceVideo) hoạt động và có ít nhất 1 video track.

  + Kiểm tra Phản hồi AI: Đợi trạng thái nhận diện (#faceStatus) thay đổi từ "Đang chuẩn bị..." sang thông báo thành công hoặc thất bại.

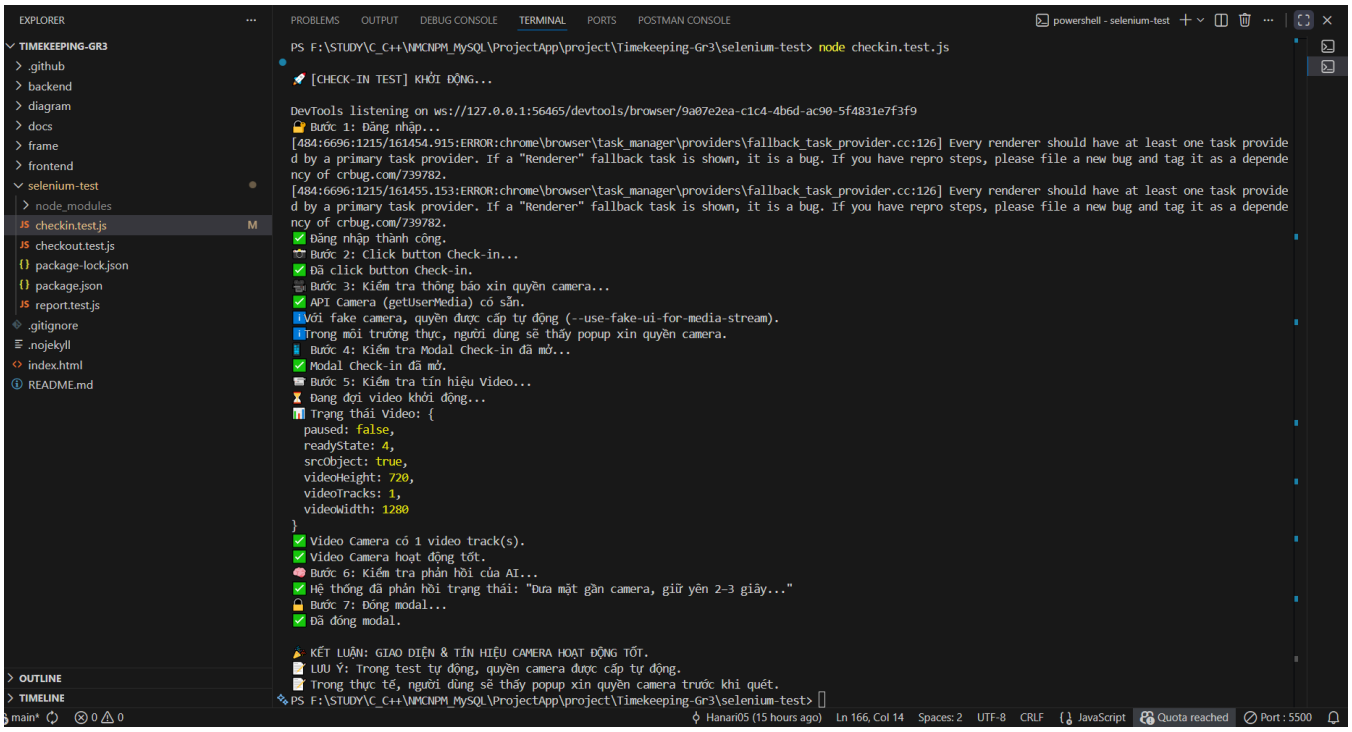
  + Đóng Modal: Click nút đóng (#faceCloseBtn).

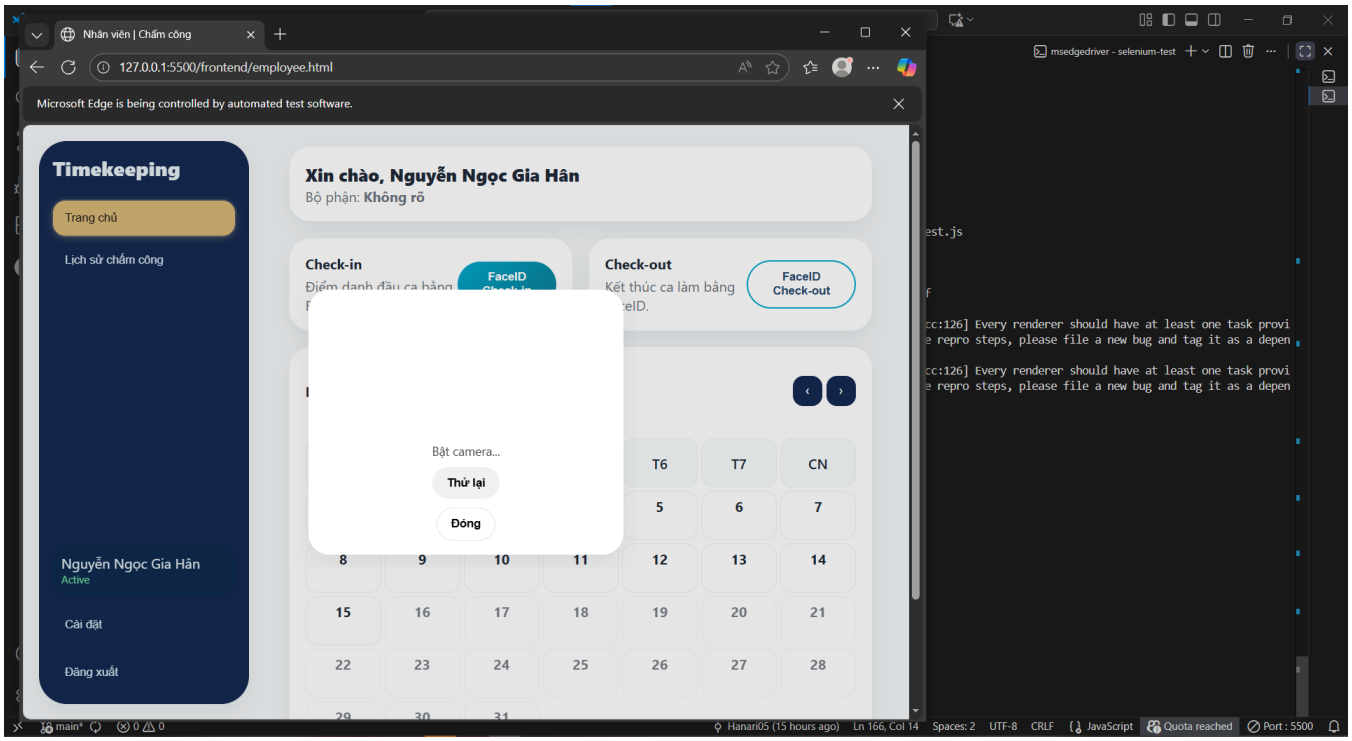
  + Xác minh (Assertion Logic):

  . Lấy số lượng bản ghi lịch sử ban đầu (#histTable tbody tr).

  . Thực hiện check-in thành công.

  . Lấy số lượng bản ghi lịch sử cuối cùng và xác nhận số lượng bản ghi phải tăng lên (tối thiểu 1), chứng minh dữ liệu đã được ghi nhận vào hệ thống.
- **Kết quả kiểm thử:**





- **Checkout.test.js**

- **Mục tiêu chính:** Kiểm tra quy trình chấm công ra (Check-out) và xác nhận dữ liệu check-out được cập nhật chính xác trên bảng lịch sử.

- **Vai trò người dùng**: Nhân viên (giahan1835@gmail.com)
- **Các bước thực hiện (Flow):**

  + Cấu hình & Đăng nhập: Tương tự như Check-in.

  + Thao tác: Điều hướng đến Tab Home và nhấn nút "Face Check-out" (#btnFaceCheckout).

  + Kiểm tra UI/Camera: Đợi modal Check-out hiển thị và xác nhận camera hoạt động.

  + Đóng Modal: Click nút đóng để giả lập quá trình nhận diện thành công.

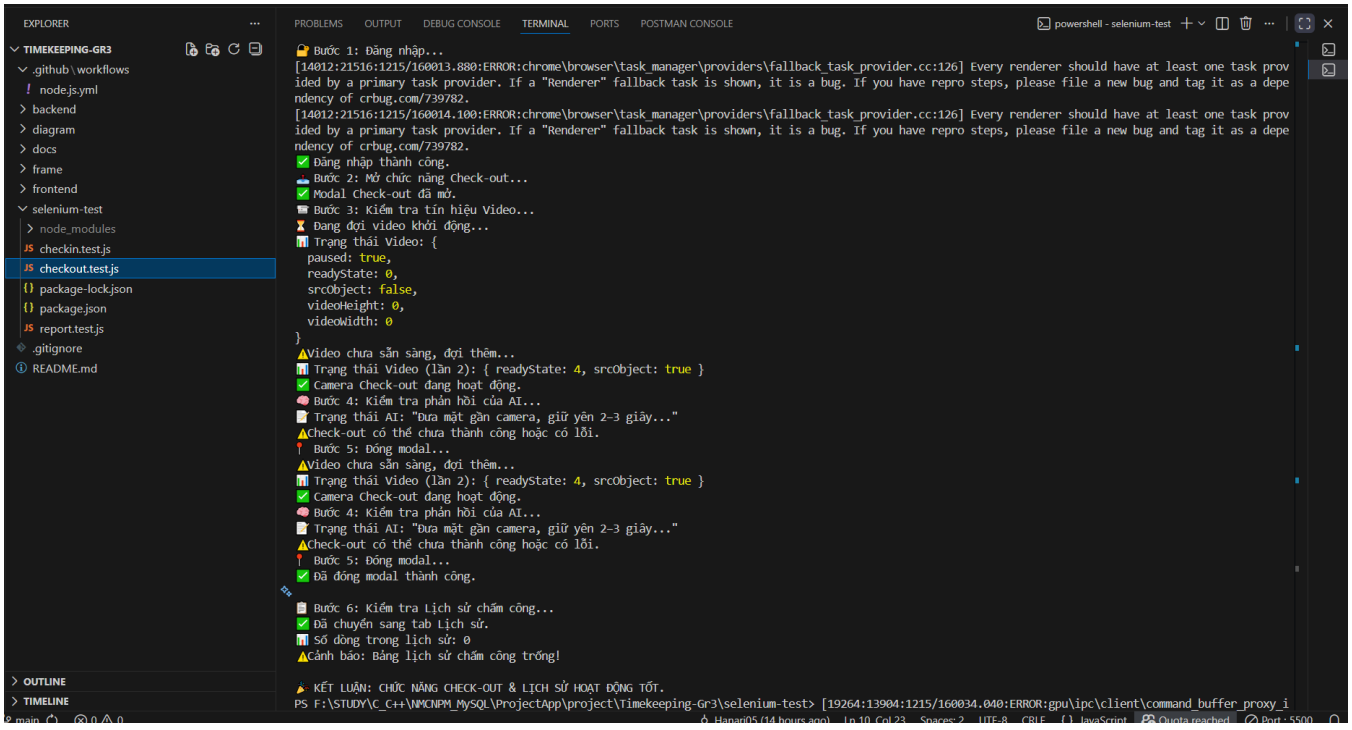
  + Điều hướng: Chuyển sang Tab History.

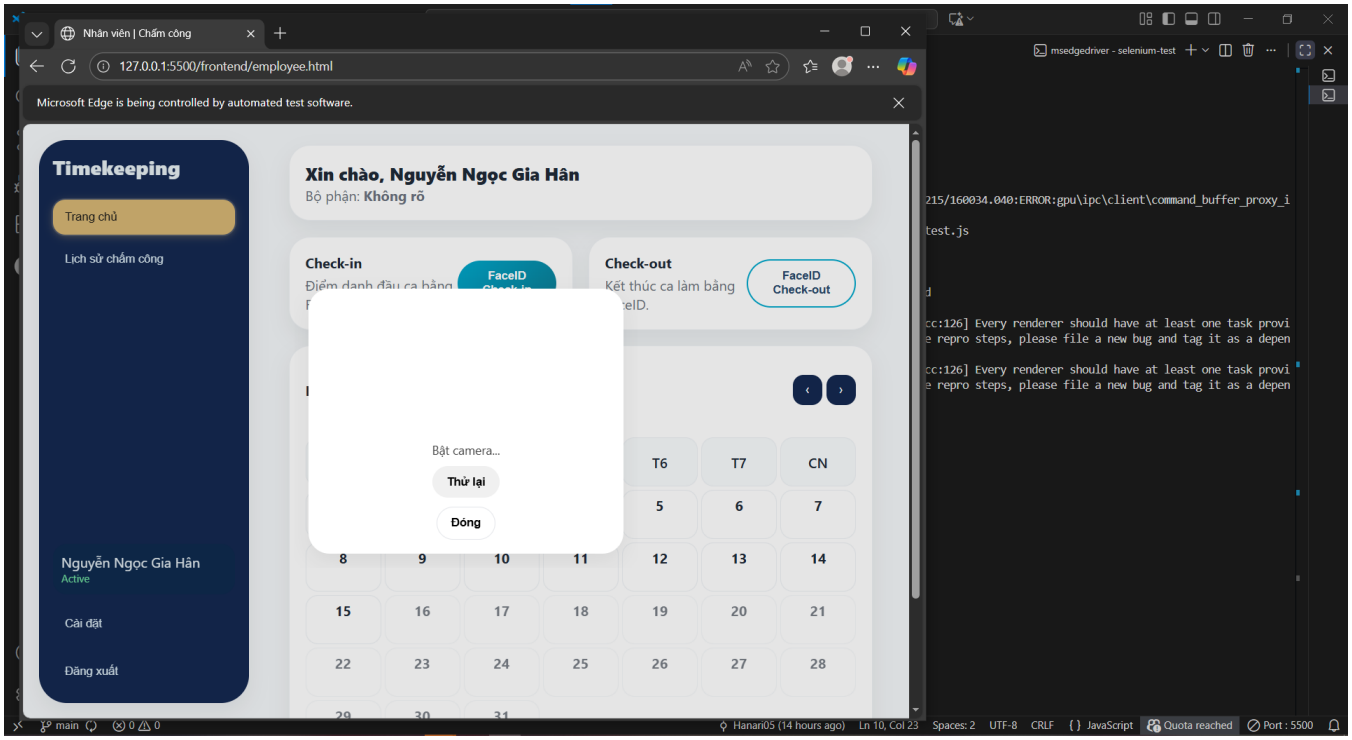
  + Xác minh (Assertion Logic):

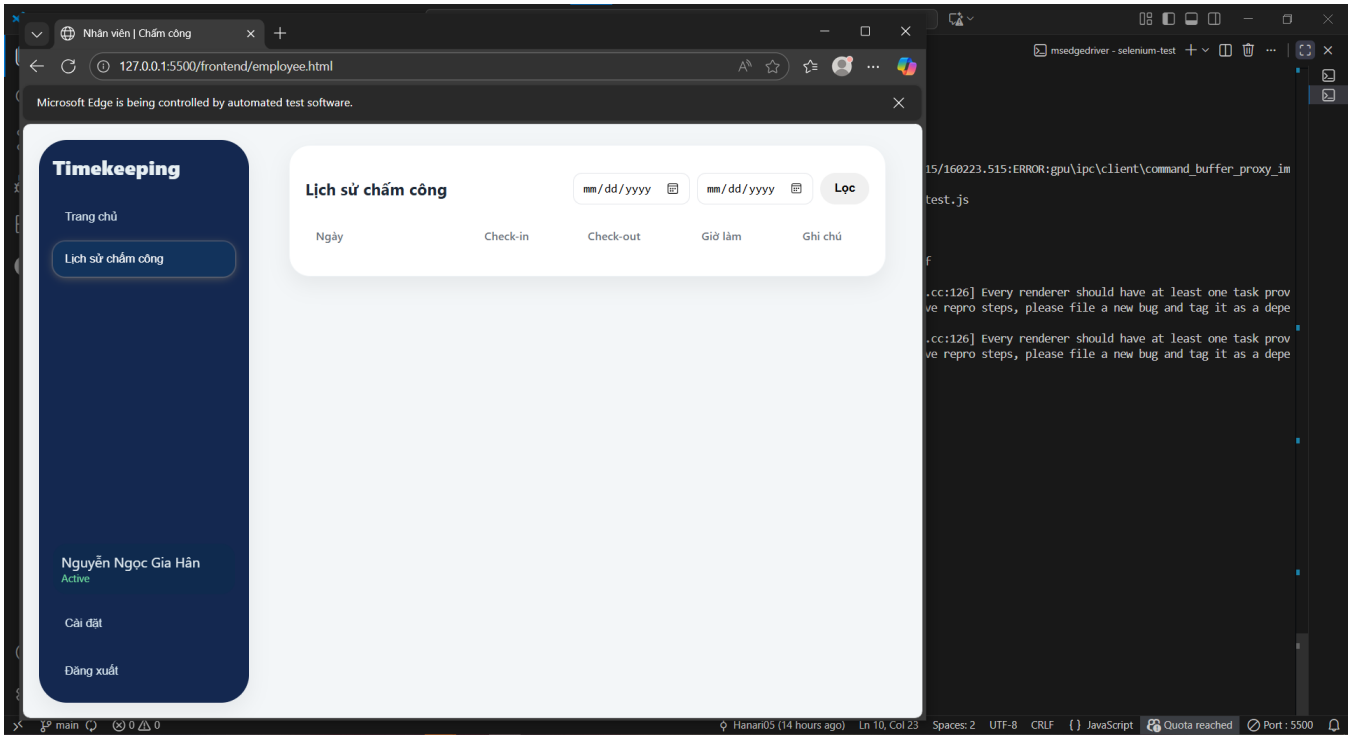
  . Kiểm tra bảng lịch sử (#histTable) có tồn tại dữ liệu.

  . Lấy dòng bản ghi mới nhất (dòng đầu tiên).

  . Xác nhận cột Check-out trong bản ghi mới nhất phải có giá trị (không rỗng hoặc không phải "-"), chứng tỏ quá trình Check-out đã cập nhật thành công bản ghi trong ngày.
- **Kết quả kiểm thử:**







- **Report.test.js**

- **Mục tiêu chính:** Kiểm tra luồng truy xuất và hiển thị dữ liệu báo cáo thống kê, đảm bảo dữ liệu được tải và các chỉ số tổng hợp được hiển thị chính xác.

- **Vai trò người dùng**: Admin/Quản lý (admin@timekeeping.com).
- **Các bước thực hiện (Flow):**

  + Đăng nhập: Đăng nhập thành công bằng tài khoản Admin/Manager.

  + Điều hướng: Chuyển đến Tab Báo cáo (.nav-item[data-route='reports']).

  + Truy vấn dữ liệu:

  . Nhập tháng báo cáo (2025-01) vào ô input (#repMonth).

  . Nhấn nút "Tải dữ liệu" (#btnReloadReports).

  . Nhấn nút "Xem chi tiết" (#btnLoadSummary).

  + Kiểm tra Thống kê: Đọc và ghi nhận giá trị của các chỉ số tổng hợp (#sumTotal, #sumPresent, #sumAbsent, #sumRate).

  + Kiểm tra Bảng chi tiết: Đợi bảng chi tiết nhân viên (#employeeSummaryTable) hiển thị và có dữ liệu (ít nhất 1 dòng).

  + Kiểm thử chức năng Xuất báo cáo:

  . Xác nhận nút "Xuất báo cáo" hiển thị và được kích hoạt (isEnabled).

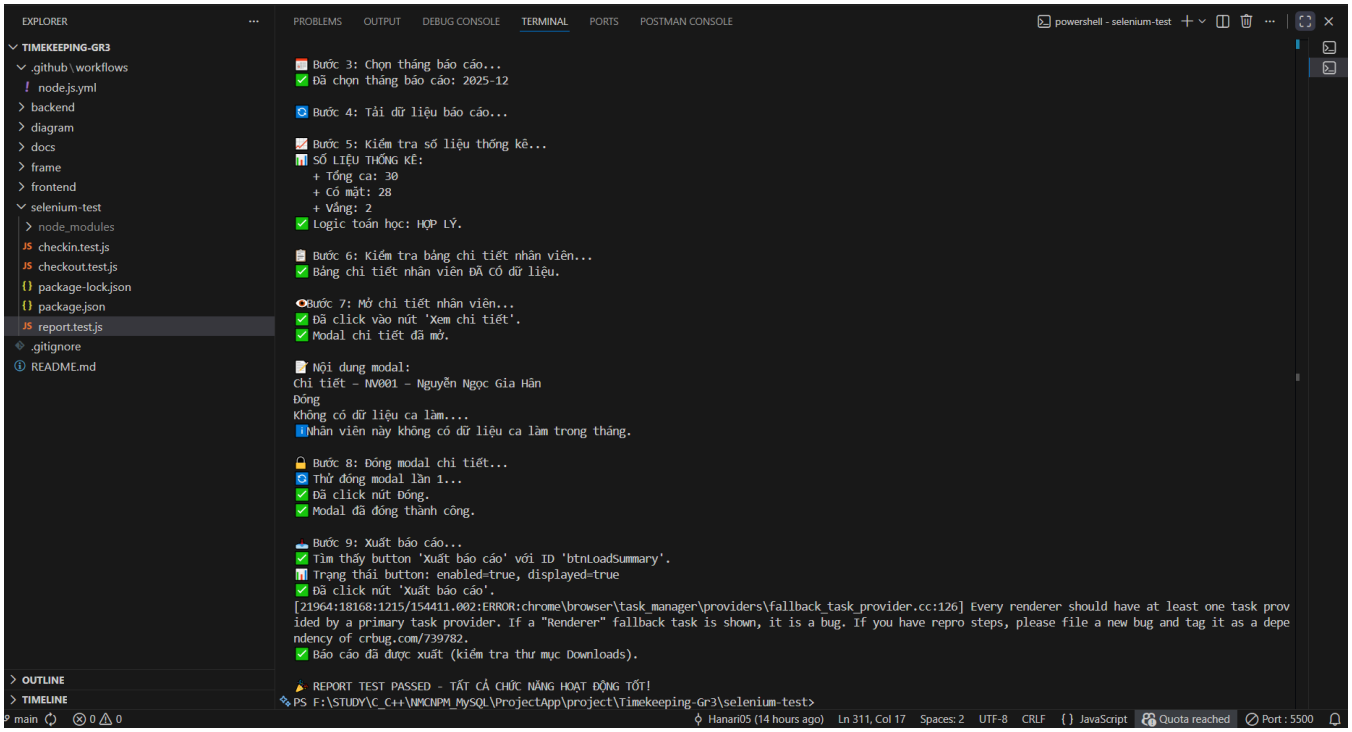
  . Click nút "Xuất báo cáo" (sử dụng JavaScript click để đảm bảo).

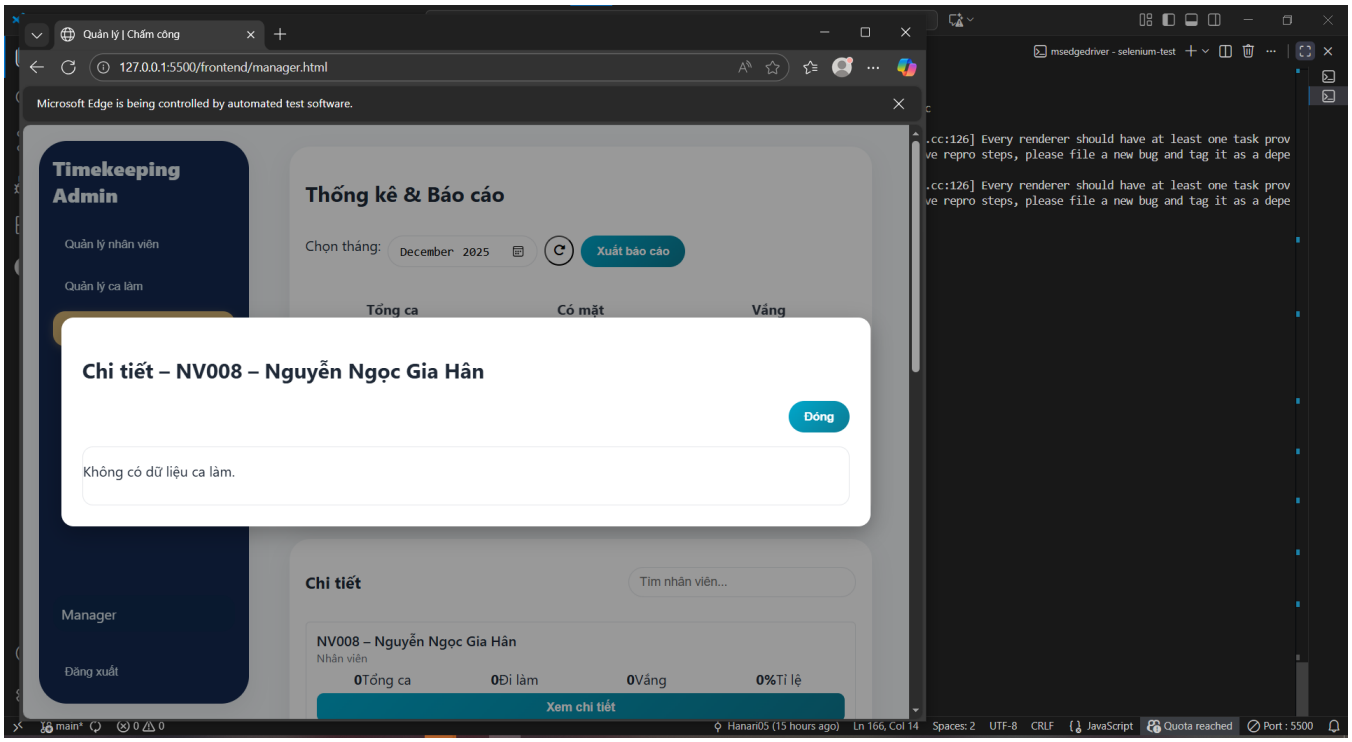
  + Xác minh (Assertion Logic):

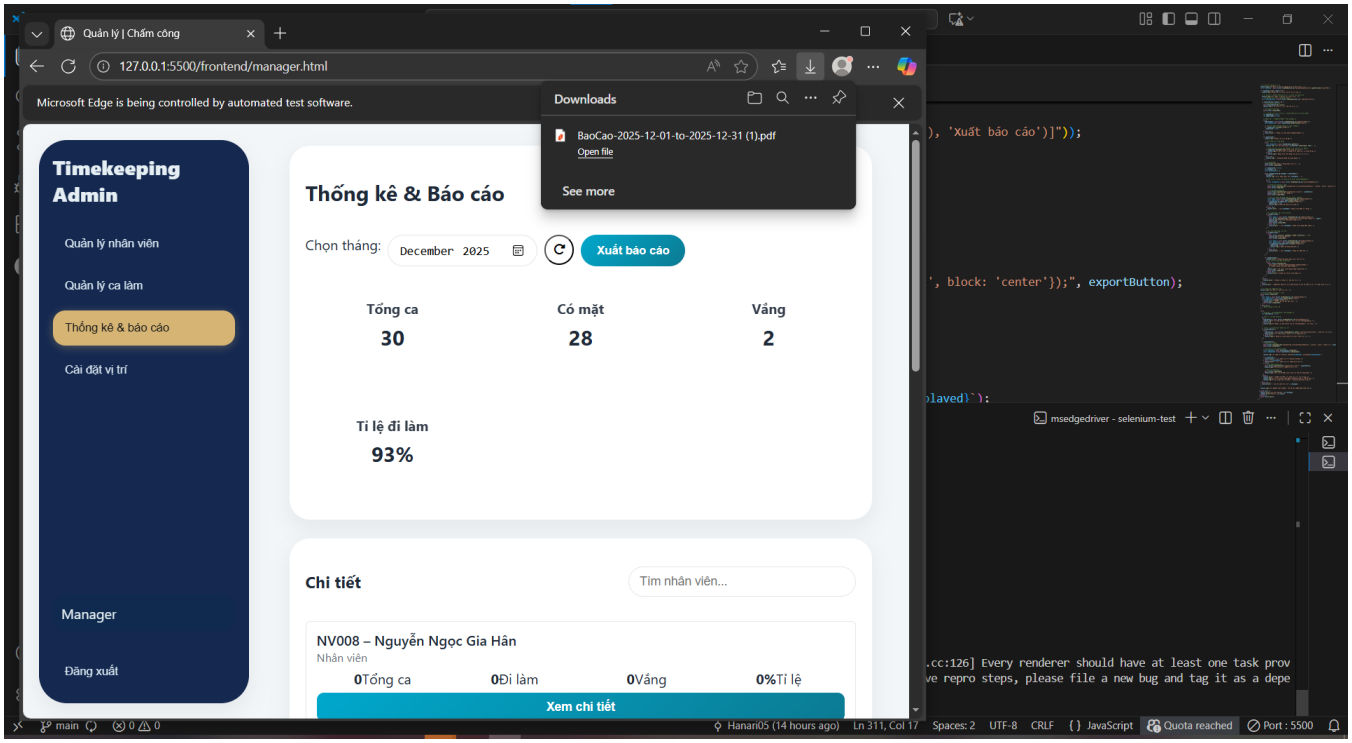
  . Bảng chi tiết nhân viên phải chứa dữ liệu (rows.length > 0).

  . Các chỉ số thống kê tổng hợp (Total, Present, Absent, Rate) phải được hiển thị và có giá trị hợp lệ (kiểm tra nội dung HTML không rỗng).

  . Nút "Xuất báo cáo" phải hoạt động (enabled).
- **Kết quả kiểm thử:**


EXPLORER
TIMEKEEPING-GR3
.github\workflows
node.js.yml
backend
diagram
docs
frame
frontend
selenium-test
node_modules
checkin.test.js
checkout.test.js
package-lock.json
package.json
report.test.js
.gitignore
README.md
OUTLINE
TIMELINE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
POSTMAN CONSOLE
powershell - selenium-test
Bước 3: Chọn tháng báo cáo...
Đã chọn tháng báo cáo: 2025-12
Bước 4: Tải dữ liệu báo cáo...
Bước 5: Kiểm tra số liệu thống kê...
SỐ LIỆU THỐNG KÊ:
+ Tổng ca: 30
+ Có mặt: 28
+ Vắng: 2
Logic toán học: HỢP LỆ.
Bước 6: Kiểm tra bảng chi tiết nhân viên...
Bảng chi tiết nhân viên ĐÃ CÓ dữ liệu.
Bước 7: Mở chi tiết nhân viên...
Đã click vào nút 'Xem chi tiết'.
Modal chi tiết đã mở.
Nội dung modal:
Chi tiết – NV001 – Nguyễn Ngọc Gia Hân
Đóng
Không có dữ liệu ca làm....
Nhân viên này không có dữ liệu ca làm trong tháng.
Bước 8: Đóng modal chi tiết...
Thử đóng modal lần 1...
Đã click nút Đóng.
Modal đã đóng thành công.
Bước 9: Xuất báo cáo...
Tìm thấy button 'Xuất báo cáo' với ID 'btnLoadSummary'.
Trạng thái button: enabled=true, displayed=true
Đã click nút 'Xuất báo cáo'.
[21964:18168:1215/154411.002:ERROR:chrome\browser\task_manager\providers\fallback_task_provider.cc:126] Every renderer should have at least one task provided by a primary task provider. If a "Renderer" fallback task is shown, it is a bug. If you have repro steps, please file a new bug and tag it as a dependency of crbug.com/739782.
Báo cáo đã được xuất (kiểm tra thư mục Downloads).
REPORT TEST PASSED - TẤT CẢ CHỨC NĂNG HOẠT ĐỘNG TỐT!
PS F:\STUDY\C_C++\NMCNPM_MySQL\ProjectApp\project\Timekeeping-Gr3\selenium-test>
main
Hanari05 (14 hours ago)
Ln 311, Col 17
Spaces: 2
UTF-8
CRLF
JavaScript
Quota reached
Port : 5500


Quản lý | Chấm công
127.0.0.1:5500/frontend/manager.html
Microsoft Edge is being controlled by automated test software.
Timekeeping Admin
Quản lý nhân viên
Quản lý ca làm
Manager
Đăng xuất
Thống kê & Báo cáo
Chọn tháng:
December 2025
Xuất báo cáo
Tổng ca
Có mặt
Vắng
Chi tiết – NV008 – Nguyễn Ngọc Gia Hân
Đóng
Không có dữ liệu ca làm.
Chi tiết
Tìm nhân viên...
NV008 – Nguyễn Ngọc Gia Hân
Nhân viên
0Tổng ca
0Đi làm
0Vắng
0%Tỉ lệ
Xem chi tiết
msedgedriver - selenium-test
.cc:126] Every renderer should have at least one task prov
ve repro steps, please file a new bug and tag it as a depe
.cc:126] Every renderer should have at least one task prov
ve repro steps, please file a new bug and tag it as a depe
main*
Hanari05 (15 hours ago)
Ln 166, Col 14
Spaces: 2
UTF-8
CRLF
JavaScript
Quota reached
Port : 5500



IV. ĐÁNH GIÁ, CẢI TIẾN

1. Tổng Quan Chiến Lược Kiểm Thử

Dự án đã áp dụng mô hình **Kim tự tháp kiểm thử (Testing Pyramid)** khá hoàn chỉnh, bao phủ ba tầng quan trọng:

- **Unit/Integration Test (Jest):** Kiểm tra logic nghiệp vụ nội tại và tích hợp logic giữa các module backend.
- **API Test (Postman):** Kiểm tra giao diện lập trình ứng dụng, xác thực hợp đồng dữ liệu (Input/Output).
- **End-to-End Test (Selenium):** Kiểm tra luồng nghiệp vụ thực tế trên giao diện người dùng.

2. Đánh Giá Chi Tiết

a. Tầng Unit Test (Backend - Jest)

- Điểm mạnh:

- Độ bao phủ logic cao: Các test case bao phủ sâu các thuật toán quan trọng như tính khoảng cách vector khuôn mặt (Euclidean) và khoảng cách địa lý (Haversine).

- Kỹ thuật Mocking hiệu quả: Sử dụng jest.mock để giả lập Firebase và các thư viện bên ngoài. Điều này giúp test chạy nhanh, cô lập lỗi và không phụ thuộc vào kết nối mạng.
- Kiểm soát bảo mật tốt: Có các file test riêng biệt cho Middleware để kiểm tra phân quyền (RBAC) và xác thực JWT, bao gồm cả các trường hợp biên như sai chữ hoa/thường trong Role.

- Hạn chế:

- Việc Mocking quá nhiều đôi khi dẫn đến tình trạng "False Positive" (Test pass nhưng chạy thật vẫn lỗi) do hành vi của Mock Object không hoàn toàn giống Database thực tế (ví dụ: các ràng buộc Firestore rules chưa được test ở đây).

b. Tầng API Test (Postman)

- Điểm mạnh:

- Kiểm thử biên (Boundary Testing): Đã kiểm tra các trường hợp check-in ngoài vùng GPS, chưa đăng ký khuôn mặt, hoặc check-in lặp lại.
- Xác thực phản hồi chuẩn: Các test case đều kiểm tra mã trạng thái HTTP (200 vs 400) và cấu trúc JSON trả về.

- Hạn chế:

- Hiện tại việc kiểm thử đang thực hiện thủ công (dựa trên file Excel export). Chưa thấy dấu hiệu của việc tự động hóa chạy collection này trong quy trình CI/CD.

c. Tầng E2E Test (Frontend - Selenium)

- Điểm mạnh:

- Xử lý bất đồng bộ tốt: Sử dụng WebDriverWait (Explicit Wait) để chờ các phần tử UI (như Modal, Video) xuất hiện, giúp test ổn định hơn.
- Kiểm chứng dữ liệu thực tế: Kịch bản check-in có bước so sánh số lượng bản ghi trong lịch sử trước và sau khi thực hiện, đảm bảo dữ liệu đã đi trọn vẹn một vòng (Full-cycle).

- Hạn chế:

- Hardcode dữ liệu nhạy cảm: Tài khoản và mật khẩu đang được viết trực tiếp trong code (CREDENTIALS object), gây rủi ro bảo mật.
- Giả lập hành vi: Test case hiện tại chỉ mở modal và đóng lại (giả lập check-in thành công), chưa thực sự test được việc camera capture và gửi ảnh đi (do hạn chế của môi trường automation).

3. Nhận Xét Chung

Hệ thống kiểm thử này đáp ứng tốt các tiêu chí của một đồ án tốt nghiệp hoặc dự án học thuật chất lượng cao:

- **Tính đầy đủ:** Có cả kiểm thử hộp trắng (Jest) và hộp đen (Selenium/Postman).
- **Tính khoa học:** Áp dụng các kỹ thuật kiểm thử tiêu chuẩn (Mocking, Assertions, Async Handling).
- **Tính thực tế:** Tập trung vào các tính năng cốt lõi và rủi ro cao nhất (Bảo mật, Tính chính xác của chấm công).

4. Đề Xuất Cải Tiến

Để nâng cấp đồ án lên mức chuyên nghiệp hơn, tác giả có thể thực hiện các cải tiến sau:

a. Cải thiện quản lý dữ liệu (Data Management)

- Vấn đề: Hiện tại dữ liệu test (Test Data) như tài khoản admin, password đang bị hardcode rải rác trong các file.
- Giải pháp:

  + Sử dụng biến môi trường (.env) để lưu trữ thông tin đăng nhập.

  + Tạo script "Seed Data" để khởi tạo một bộ dữ liệu sạch trước khi chạy test và xóa đi sau khi chạy xong (Teardown), tránh làm rác database.

b. Tăng cường Automation cho Postman

- Vấn đề: File Excel cho thấy kết quả chạy thủ công.
- Giải pháp: Sử dụng thư viện Newman để chạy bộ Collection Postman tự động bằng dòng lệnh (Command Line) và tích hợp vào script npm test.

c. Nâng cấp E2E Test

- Vấn đề: Phụ thuộc vào trình duyệt biên dịch sẵn (MicrosoftEdge) và giao diện đồ họa.
- Giải pháp:

  + Cấu hình chạy ở chế độ Headless (không giao diện) để tích hợp vào CI/CD (GitHub Actions/GitLab CI).

  + Sử dụng tùy chọn --use-fake-device-for-media-stream của trình duyệt để giả lập Camera input, giúp test chức năng nhận diện khuôn mặt tự động hoàn toàn mà không cần người ngồi trước máy

d. Bổ sung Performance Test (Nâng cao)

- Dùng công cụ (như JMeter hoặc k6) để test khả năng chịu tải của API /api/face/check-in khi có 100 nhân viên cùng chấm công một lúc (Concurrency Testing).

V. TỔNG KẾT

1. Tổng Quan Quá Trình Kiểm Thử

Dự án đã áp dụng chiến lược kiểm thử toàn diện theo mô hình Kim tự tháp kiểm thử (Testing Pyramid), đảm bảo chất lượng từ mã nguồn (Source Code) đến trải nghiệm người dùng cuối (User Experience).

- **Giai đoạn 1**: Kiểm thử Đơn vị & Tích hợp (Unit & Integration Testing) – Backend

- Công cụ: Jest Framework.
- Phạm vi: Kiểm tra logic nghiệp vụ nội tại của các Service, Controller và Middleware.
- Kết quả đạt được:

  + Độ tin cậy thuật toán: Đã kiểm chứng tính chính xác của thuật toán tính khoảng cách Euclidean (so khớp khuôn mặt) và công thức Haversine (tính khoảng cách GPS).

  + Xử lý ngoại lệ: Hệ thống xử lý tốt các tình huống lỗi như mất kết nối Database (giả lập qua Mocking), dữ liệu đầu vào sai định dạng.

  + Bảo mật: Middleware đã chặn thành công các truy cập trái phép, phân biệt rõ ràng quyền hạn của Admin và User, ngăn chặn các nỗ lực tấn công cơ bản (Injection).

- **Giai đoạn 2:** Kiểm thử API (API Testing) - Giao tiếp dữ liệu

- Công cụ: Postman.
- Phạm vi**:** Kiểm tra các điểm cuối (Endpoints) phục vụ tính năng Xác thực và Chấm công.
- Kết quả đạt được:

  + Đảm bảo tuân thủ chuẩn RESTful (Status code 200, 400, 401, 500 được trả về đúng ngữ cảnh).

  + Xác minh tính chặt chẽ của nghiệp vụ: Ngăn chặn chấm công khi chưa đăng ký khuôn mặt, chấm công ngoài vùng cho phép, hoặc chấm công lặp lại nhiều lần trong ngày.

- **Giai đoạn 3:** Kiểm thử Chấp nhận & Giao diện (E2E Testing) – Frontend

- Công cụ: Selenium WebDriver (JavaScript).
- Phạm vi: Mô phỏng hành vi thực tế của người dùng trên trình duyệt Microsoft Edge.
- Kết quả đạt được:

  + Xác thực luồng nghiệp vụ (Business Flow): Đã kiểm thử thành công quy trình trọn vẹn: Đăng nhập -> Mở Camera -> Check-in/Check-out -> Dữ liệu hiển thị lên bảng Lịch sử.

  + Trải nghiệm người dùng (UX): Đảm bảo các thành phần giao diện như Modal, Video stream, và các bảng biểu báo cáo hiển thị đúng, không bị lỗi render.

2. Đánh Giá Chất Lượng Dự Án

- **Tính Đúng Đắn (Correctness):** Hệ thống hoạt động đúng theo yêu cầu thiết kế. Các thuật toán tính toán chấm công và nhận diện hoạt động chính xác.

- **Tính Ổn Định (Robustness):** Nhờ việc Mocking trong Unit Test và xử lý Async Wait trong E2E Test, hệ thống chứng tỏ khả năng hoạt động ổn định, ít xảy ra lỗi vặt (flaky bugs).
- **Tính Bảo Mật (Security):** Cơ chế xác thực JWT và phân quyền Middleware được kiểm thử kỹ lưỡng, đảm bảo an toàn cho dữ liệu chấm công nhạy cảm.
- **Khả Năng Bảo Trì (Maintainability):** Mã nguồn kiểm thử được tổ chức tách biệt, rõ ràng, giúp dễ dàng mở rộng tính năng hoặc refactor code sau này mà không lo phá vỡ các chức năng cũ (Regression).

LỜI KẾT

Qua quá trình thực hiện hoạt động kiểm thử cho dự án **Mini App Chấm Công Nhân Viên**, nhóm đã tiến hành kiểm tra toàn diện từ tầng giao diện, tầng API đến tầng logic nghiệp vụ bên trong hệ thống. Ba công cụ kiểm thử là **Jest**, **Postman** và **Selenium** đã hỗ trợ hiệu quả trong việc đảm bảo chất lượng sản phẩm trước khi đưa vào triển khai thực tế.

Kết quả kiểm thử cho thấy hầu hết các chức năng chính của hệ thống đều hoạt động ổn định, bao gồm: thao tác check-in/check-out của nhân viên, hiển thị dữ liệu báo cáo, định vị GPS, xử lý logic ca làm, và sự tương tác của người dùng trên giao diện web. Một số lỗi nhỏ đã được phát hiện và khắc phục kịp thời, góp phần nâng cao độ tin cậy và tính chính xác của hệ thống.

Quá trình kiểm thử không chỉ đảm bảo chất lượng sản phẩm mà còn giúp nhóm hiểu rõ hơn về quy trình phát triển phần mềm, vai trò của từng loại kiểm thử cũng như tầm quan trọng của việc phối hợp giữa lập trình và kiểm thử trong một dự án thực tế. Những kinh nghiệm tích lũy được từ dự án sẽ là nền tảng quan trọng cho các dự án sau này.

Cuối cùng, nhóm xin gửi lời cảm ơn đến giảng viên đã hướng dẫn và hỗ trợ. Đồng thời cảm ơn các thành viên trong nhóm đã nỗ lực hoàn thành tốt nhiệm vụ và hỗ trợ nhau trong từng giai đoạn của dự án.

Tài liệu tham khảo

Kinh nghiệm viết Test Report chuẩn nhất cho Tester - Test Mentor

[API Postman] Bài 4 - Phân tích tài liệu API và viết API test case

Jest Testing Framework

How to fix 401 unauthorized error in Postman?What is HTTP code 401 unauthorized? #infysky #code

React testing library and jest in Hindi #1 Introduction of React Testing

Google Gemini

ChatGPT

Claude

github.com/copilot