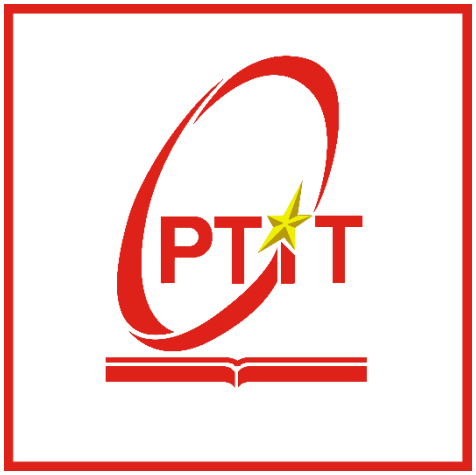
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

CƠ SỞ TẠI THÀNH PHỐ HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN 2



BÁO CÁO KẾT MÔN

Đề tài số 3: Mini App Quản Lý Chấm Công Nhân Viên

Giảng viên hướng dẫn: Châu Văn Vân

Môn: Nhập môn Công nghệ phần mềm

Nhóm: 3

Lớp: D23CQPTUD01-N

TP.HCM 12/2025

LỜI MỞ ĐẦU

Trong kỷ nguyên công nghệ 4.0 hiện nay, việc ứng dụng công nghệ thông tin vào quản lý doanh nghiệp không còn là một lựa chọn mà là một yêu cầu cấp thiết. Đối với bất kỳ tổ chức nào, công tác quản lý nhân sự và chấm công luôn đóng vai trò then chốt trong việc đảm bảo kỷ luật và tính công bằng. Các phương pháp chấm công thủ công truyền thống thường tốn nhiều thời gian, dễ xảy ra sai sót và khó kiểm soát tình trạng gian lận.

Nhận thức được tầm quan trọng đó, trong khuôn khổ học phần **Nhập môn Công nghệ Phần mềm**, nhóm 3 - chúng em đã quyết định lựa chọn và thực hiện đề tài: **"Xây dựng Mini App Quản Lý Chấm Công Nhân Viên"**.

Mục tiêu chính của đề tài là xây dựng một hệ thống phần mềm quản lý chấm công hiện đại và chính xác. Điểm nhấn của dự án là việc **nghiên cứu và tích hợp công nghệ nhận diện khuôn mặt (FaceID) vào quy trình Check-in/Check-out**. Giải pháp này không chỉ giúp tối ưu hóa trải nghiệm người dùng, rút ngắn thời gian thao tác mà còn giải quyết triệt để vấn đề chấm công hộ, đảm bảo tính minh bạch cao nhất cho doanh nghiệp.

Dự án được phát triển theo quy trình **Agile - Scrum**, giúp nhóm làm quen với môi trường làm việc chuyên nghiệp, linh hoạt và đề cao tinh thần cộng tác. Báo cáo này là kết quả của quá trình tìm hiểu lý thuyết, phân tích yêu cầu và hiện thực hóa sản phẩm của nhóm. Nội dung báo cáo sẽ trình bày chi tiết về cơ sở lý thuyết, công nghệ sử dụng (Node.js, Firebase), quá trình huấn luyện mô hình nhận diện khuôn mặt và kết quả kiểm thử hệ thống.

Trong quá trình thực hiện, mặc dù đã rất cố gắng, nhóm khó tránh khỏi những thiếu sót về mặt kiến thức cũng như kinh nghiệm thực tế. Chúng em rất mong nhận được những ý kiến đóng góp quý báu của Thầy Châu Văn Vân để đề tài được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

THÀNH VIÊN

STT	Họ và tên	Chức năng	MSSV
1	**Lê Thị Bảo Diệp**	**Backend**	**N23DCPT006**
2	**Nguyễn Ngọc Gia Hân**	**Testing**	**N23DCPT019**
3	**Nông Thị Hồng Lan**	**CD/CI**	**N23DCPT029**
4	**Thạch Gia Uy**	**Frontend**	**N23DCPT056**

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

1. Tên đề tài:

Xây dựng Mini App Quản Lý Chấm Công Nhân Viên (Timekeeping App)

2. Lý do chọn đề tài:

Trong môi trường doanh nghiệp hiện đại, việc quản lý nhân sự và theo dõi thời gian làm việc (chấm công) là một trong những nhiệm vụ quan trọng hàng đầu. Tuy nhiên, nhiều doanh nghiệp vừa và nhỏ vẫn đang sử dụng các phương pháp thủ công như ghi chép sổ sách hoặc sử dụng Excel rời rạc. Các phương pháp này bộc lộ nhiều hạn chế:

- **Tốn kém thời gian:** Việc tổng hợp dữ liệu cuối tháng mất nhiều công sức.
- **Dễ sai sót:** Quá trình nhập liệu thủ công dễ dẫn đến nhầm lẫn.
- **Khó kiểm soát gian lận:** Tình trạng chấm công hộ khó bị phát hiện nếu chỉ dùng thẻ từ hoặc điểm danh thông thường.

Xuất phát từ thực tế đó, nhóm quyết định xây dựng ứng dụng Web Timekeeping. Đặc biệt, nhóm tích hợp công nghệ **nhận diện khuôn mặt (FaceID)** để giải quyết triệt để bài toán chấm công hộ, đồng thời mang lại trải nghiệm công nghệ hiện đại, nhanh chóng cho người dùng.

Ngoài ra, đây cũng là cơ hội để nhóm áp dụng quy trình phát triển phần mềm **Agile - Scrum** và các công nghệ mới (Node.js, Cloud Database) đã được học vào thực tế.

3. Mục tiêu của đề tài:

Dự án hướng tới các mục tiêu cụ thể sau:

- **Về mặt chức năng:**

Dự án hướng tới việc xây dựng một hệ thống hoàn chỉnh với các nhóm chức năng cụ thể sau:

❖ **Quản lý tài khoản và bảo mật:**

- Tích hợp xác thực hai lớp (2FA): Đăng nhập hệ thống thông qua tài khoản Gmail để đảm bảo tính chính danh, kết hợp với xác thực sinh trắc học (FaceID) khi thực hiện chấm công.
- Phát triển chức năng "Quên mật khẩu" (Forgot Password) gửi mã xác nhận qua email, giúp người dùng khôi phục tài khoản an toàn khi bị mất quyền truy cập.

❖ **Quản lý nhân sự và ca làm việc:**

- Cung cấp công cụ quản lý nhân sự toàn diện: Cho phép xem, chỉnh sửa thông tin cá nhân và cập nhật trạng thái làm việc (Đang hoạt động/Đã nghỉ việc) của nhân viên.
- Xây dựng chức năng quản lý ca làm việc (Shift Management) linh hoạt: Cho phép người quản lý thêm mới, chỉnh sửa thời gian hoặc xóa các ca làm việc phù hợp với lịch trình của doanh nghiệp.

❖ **Báo cáo và thống kê:**

- Tự động thống kê số liệu chấm công theo thời gian thực (số giờ làm, đi muộn, về sớm).
- Tích hợp chức năng xuất báo cáo chấm công ra định dạng PDF, giúp bộ phận kế toán/nhân sự dễ dàng lưu trữ và in ấn phục vụ việc tính lương.

- **Về mặt kỹ thuật:**

- Làm chủ công nghệ Backend với Node.js và cơ sở dữ liệu thời gian thực Firestore.
- Xây dựng giao diện Frontend thân thiện, dễ sử dụng.
- Biết cách triển khai và kiểm thử hệ thống tự động.

4. Phạm vi nghiên cứu:

Đối tượng sử dụng: Hệ thống hướng tới hai nhóm người dùng chính là Người quản lý (Admin) và Nhân viên (Employee).

Nền tảng công nghệ: Ứng dụng Web (Web Application) chạy trên trình duyệt.

Giới hạn đề tài: Mặc dù đã hoàn thiện các luồng nghiệp vụ cốt lõi, trong khuôn khổ thời gian và nguồn lực của một đồ án môn học, hệ thống vẫn tồn tại một số giới hạn nhất định:

- **Độ chính xác của FaceID trong môi trường phức tạp:** Công nghệ nhận diện khuôn mặt hiện tại hoạt động tốt trong điều kiện ánh sáng tiêu chuẩn. Tuy nhiên, độ chính xác có thể bị giảm sút trong điều kiện thiếu sáng, ngược sáng, hoặc khi người dùng đeo các phụ kiện che khuất khuôn mặt.
- **Khả năng chịu tải và mở rộng:** Hệ thống hiện đang được triển khai trên môi trường Server đơn lẻ (Single Node). Do đó, chưa được tối ưu hóa để xử lý hàng nghìn yêu cầu đồng thời (High Concurrency), có thể dẫn đến độ trễ nếu áp dụng cho quy mô tập đoàn lớn ngay lập tức.
- **Phụ thuộc vào kết nối Internet:** Do sử dụng cơ sở dữ liệu đám mây (Firebase Firestore) và thư viện nhận diện trực tuyến, hệ thống yêu cầu kết nối mạng ổn định để hoạt động. Nhóm chưa phát triển cơ chế "Offline Mode" để lưu trữ dữ liệu tạm thời khi mất mạng.
- **Quy trình phê duyệt phức tạp:** Các chức năng quản lý ca làm việc hiện tại đang thực hiện theo cơ chế một chiều từ quản lý. Hệ thống chưa hỗ trợ các luồng nghiệp vụ nhân sự nâng cao như: nhân viên tự đăng ký đổi ca, quy trình phê duyệt nghỉ phép nhiều cấp (Leader duyệt → Manager duyệt → HR duyệt).
- **Bảo mật nâng cao:** Dữ liệu khuôn mặt hiện được lưu trữ dưới dạng các vector đặc trưng số học. Tuy nhiên, hệ thống chưa áp dụng các chuẩn mã hóa cấp cao (như Enterprise Grade Security) để chống lại các cuộc tấn công giả mạo (Deepfake) tinh vi.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ SỬ DỤNG

1. Tổng quan về kiến trúc hệ thống

Hệ thống Timekeeping được xây dựng dựa trên mô hình kiến trúc **Client-Server (Khách - Chủ)** kết hợp với kiến trúc **RESTful API**. Mô hình này giúp tách biệt hoàn toàn giữa giao diện người dùng (Frontend) và logic xử lý nghiệp vụ (Backend), tạo điều kiện thuận lợi cho việc phát triển, bảo trì và mở rộng hệ thống.

- **Client Side:** Giao diện tương tác người dùng, gửi yêu cầu HTTP và hiển thị dữ liệu.
- **Server Side:** Tiếp nhận yêu cầu, xử lý nghiệp vụ và tương tác với cơ sở dữ liệu.

2. Các công nghệ nền tảng (Frontend & Backend)

❖ Nền tảng Backend: Node.js và Express.js

- **Node.js:** Môi trường chạy mã JavaScript (Runtime Environment) phía Server, nổi bật với cơ chế xử lý bất đồng bộ (Asynchronous) và hướng sự kiện (Event-driven), phù hợp cho các ứng dụng yêu cầu tốc độ phản hồi nhanh.
- **Express.js:** Web Framework tối giản trên nền Node.js, hỗ trợ mạnh mẽ việc xây dựng API, quản lý định tuyến (Routing) và tích hợp các Middleware xử lý bảo mật.

❖ Nền tảng Frontend: HTML5, CSS3 và JavaScript

- **HTML5 & CSS3:** Xây dựng cấu trúc và định dạng giao diện, đảm bảo tính tương thích trên nhiều thiết bị (Responsive Design).
- **JavaScript (ES6+):** Xử lý logic phía Client, sử dụng Fetch API để tương tác bất đồng bộ với Backend.

3. Hệ quản trị Cơ sở dữ liệu: Google Firebase Firestore

Thay vì sử dụng các cơ sở dữ liệu quan hệ (SQL) truyền thống, nhóm lựa chọn **Google Cloud Firestore** - một cơ sở dữ liệu NoSQL linh hoạt dựa trên nền tảng đám mây của Google.

- **Mô hình dữ liệu:** Dữ liệu được lưu trữ dưới dạng các **Documents (Tài liệu)** và được tổ chức thành các **Collections (Bộ sưu tập)**. Cấu trúc này rất linh hoạt, phù hợp với đối tượng dữ liệu JSON.
- **Tính năng nổi bật:**
    - **Real-time Updates:** Hỗ trợ đồng bộ dữ liệu thời gian thực giữa các client, rất hữu ích cho tính năng chấm công và thông báo.
    - **Khả năng mở rộng:** Tự động mở rộng (Auto-scaling) theo lưu lượng truy cập mà không cần cấu hình server phức tạp.
    - **Bảo mật:** Tích hợp sẵn cơ chế Security Rules để phân quyền truy cập dữ liệu chặt chẽ.

4. Công nghệ CI/CD và Quản lý mã nguồn

Để hiện đại hóa quy trình phát triển và vận hành, nhóm áp dụng quy trình **CI/CD (Tích hợp và Triển khai liên tục)**. *(Chi tiết về quy trình triển khai sẽ được trình bày sâu hơn ở Chương 4)*.

Các công nghệ sử dụng bao gồm:

- **Git & GitHub:** Sử dụng để quản lý phiên bản mã nguồn (Source Control) và làm kho lưu trữ trung tâm.
- **GitHub Actions:** Công cụ tự động hóa được dùng để build và deploy phần Frontend lên GitHub Pages.
- **Render Auto-Deploy:** Nền tảng Cloud dùng để hosting Backend, hỗ trợ tự động cập nhật Server ngay khi có thay đổi mới trên mã nguồn.

5. Các công cụ và phương pháp Kiểm thử (Testing)

Để đảm bảo chất lượng phần mềm và giảm thiểu lỗi trước khi đưa vào vận hành, nhóm sử dụng kết hợp các phương pháp kiểm thử sau:

- **Unit Test (Kiểm thử đơn vị):**
    - *Mục đích:* Kiểm tra tính đúng đắn của các hàm logic xử lý nghiệp vụ nhỏ nhất (ví dụ: logic tính giờ công, logic phân quyền).
    - *Phương pháp:* Viết các kịch bản test để đảm bảo mỗi hàm hoạt động đúng với các đầu vào khác nhau.
    - *Mô hình 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.
    - Bộ test suite gồm các chức năng hệ thống chấm công: xác thực (Auth), phân quyền (RBAC), nghiệp vụ chấm công (Attendance), quản lý ca làm việc (Shifts) và nhận diện khuôn mặt (FaceID).
- **API Testing với Postman:**
    - *Mục đích:* Kiểm thử các điểm cuối (Endpoints) của Backend.
    - *Cách thức:* Sử dụng công cụ **Postman** để gửi các request (GET, POST, PUT, DELETE) kèm theo dữ liệu giả lập (Body, Header) và kiểm tra mã trạng thái (Status Code) cũng như dữ liệu trả về từ Server.
    - Bộ test suite 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.
- **UI/Functional Testing với Selenium:**
    - *Mục đích:* Kiểm thử các điểm cuối (Endpoints) của Backend.
    - *Cách thức:* Sử dụng công cụ **Postman** để gửi các request (GET, POST, PUT, DELETE) kèm theo dữ liệu giả lập (Body, Header) và kiểm tra mã trạng thái (Status Code) cũng như dữ liệu trả về từ Server.
    - Bộ test suite 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.

6. Các dịch vụ tích hợp (Third-party APIs)

- **Mailjet API:** Dịch vụ gửi Email Transactional qua giao thức HTTP, được sử dụng để gửi mã OTP xác thực tài khoản, đảm bảo tốc độ và độ tin cậy cao trên môi trường Cloud.
- **Face-API.js:** Thư viện JavaScript chạy trên trình duyệt, hỗ trợ nhận diện khuôn mặt thông qua Webcam để thực hiện chức năng điểm danh.

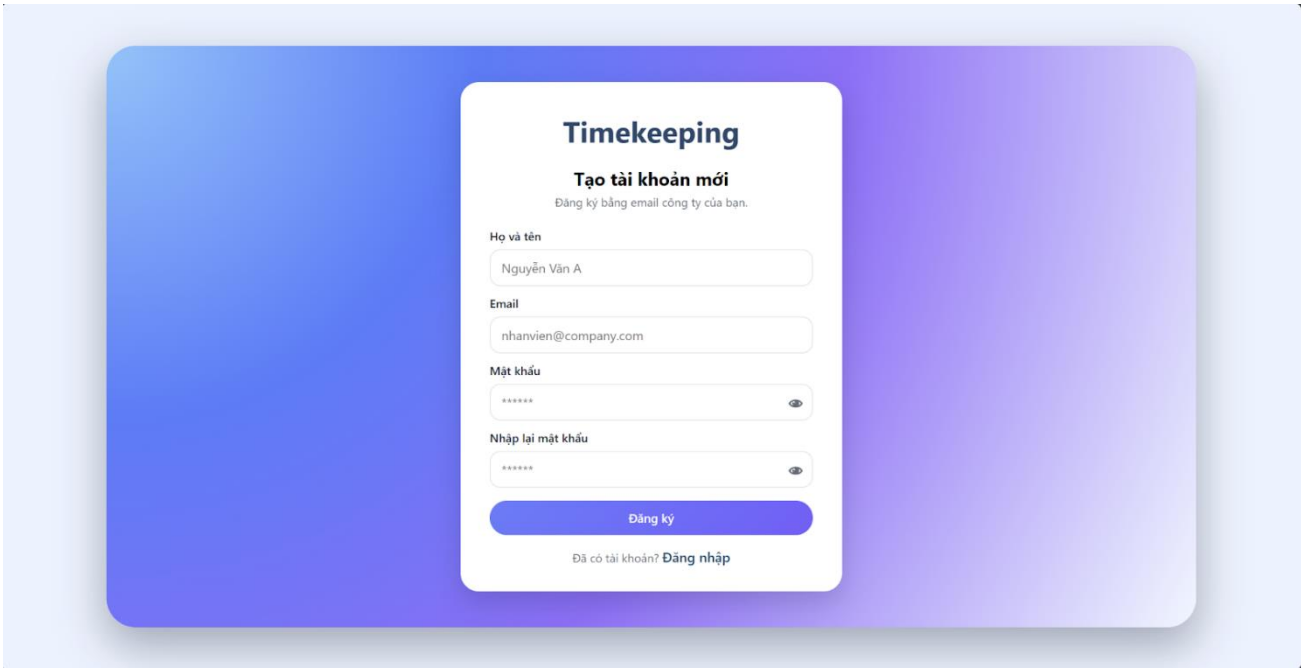
CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

1. Phân tích yêu cầu hệ thống

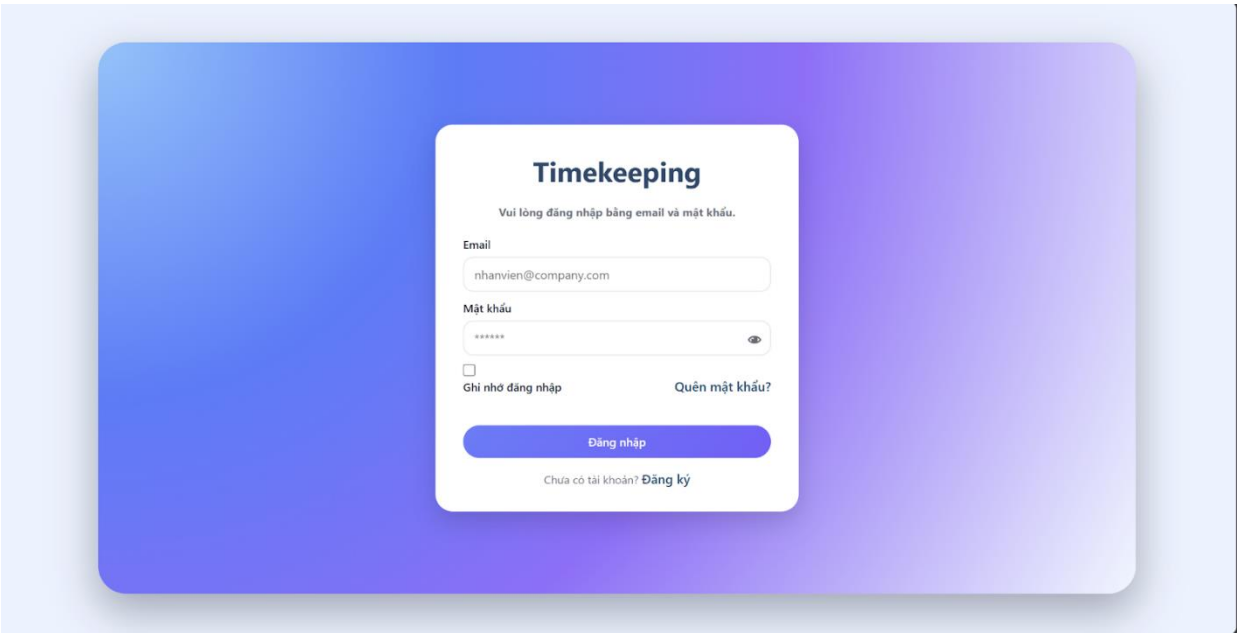
A. Yêu cầu chức năng (Functional Requirements)

Hệ thống được thiết kế để phục vụ hai nhóm đối tượng chính: Quản trị viên (Admin) và Nhân viên (User).

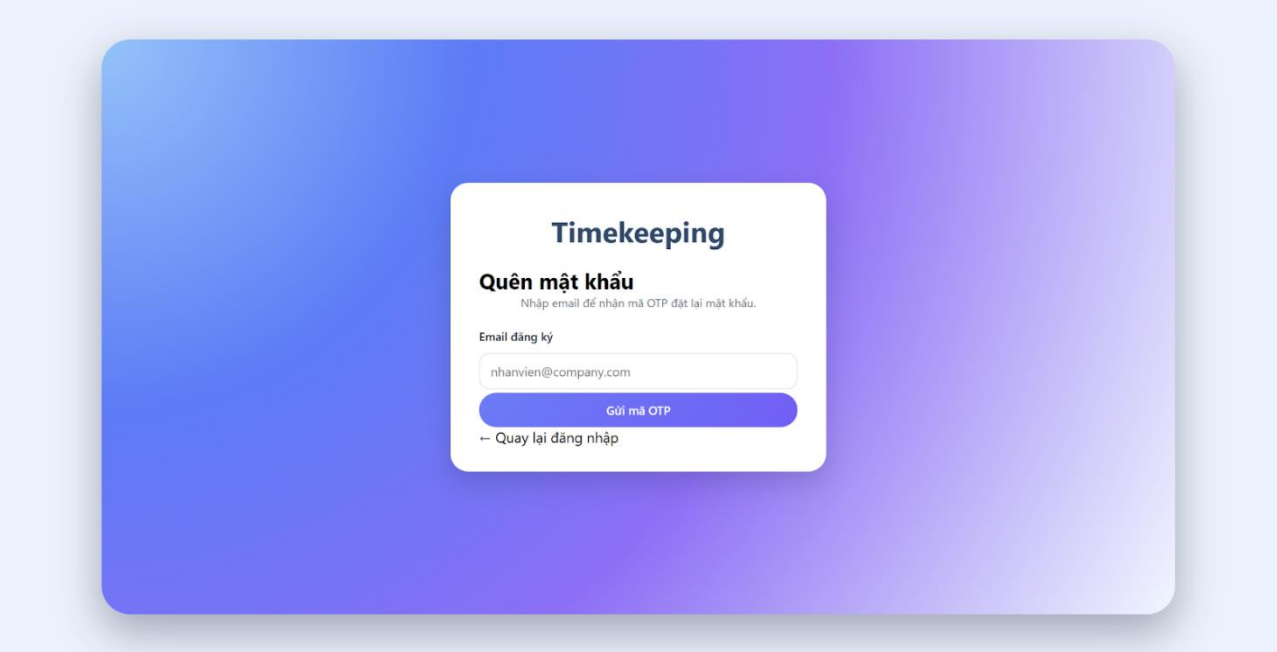
- **Nhóm chức năng Xác thực (Authentication):**
    - Đăng ký tài khoản mới: Người dùng nhập thông tin và nhận mã OTP xác thực qua Email (sử dụng Mailjet).



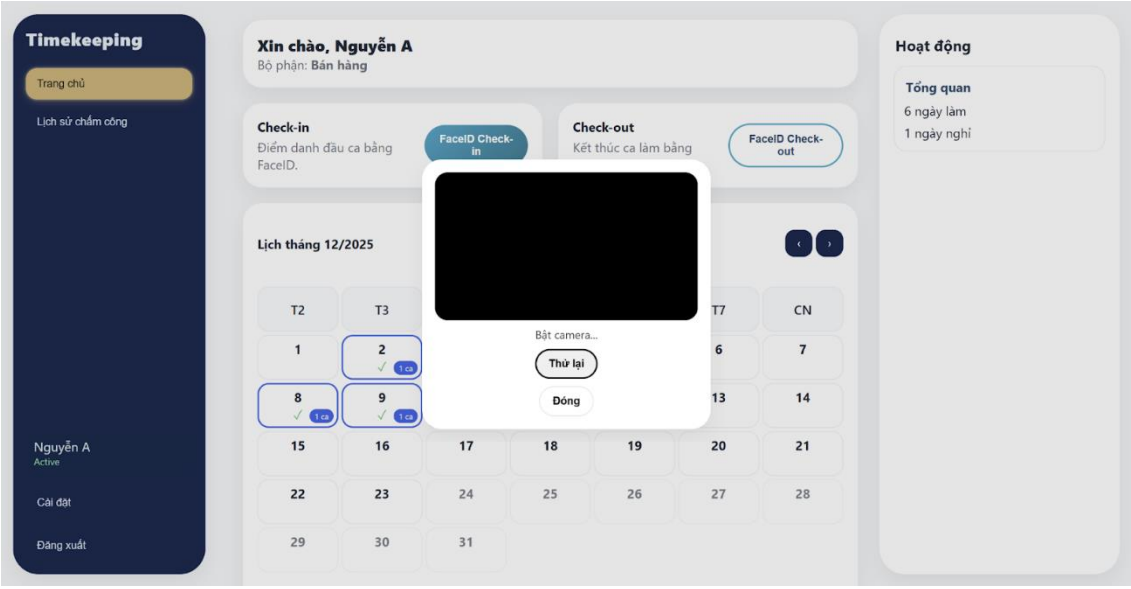
    - Đăng nhập: Hỗ trợ đăng nhập bằng Email/Mật khẩu.



    - Quên mật khẩu: Gửi link hoặc mã reset về email.

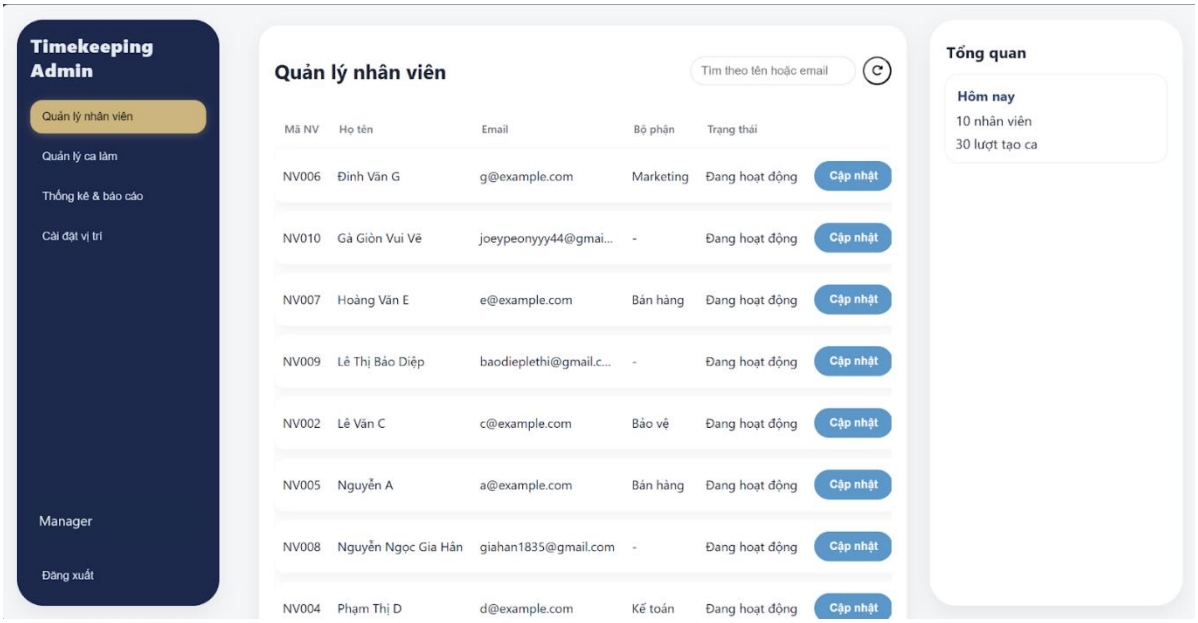


- **Nhóm chức năng Chấm công (Timekeeping):**
    - Check-in/Check-out: Thực hiện điểm danh bằng công nghệ nhận diện khuôn mặt (FaceID) kết hợp với chức năng GPS để kiểm tra vị trí của người chấm công.

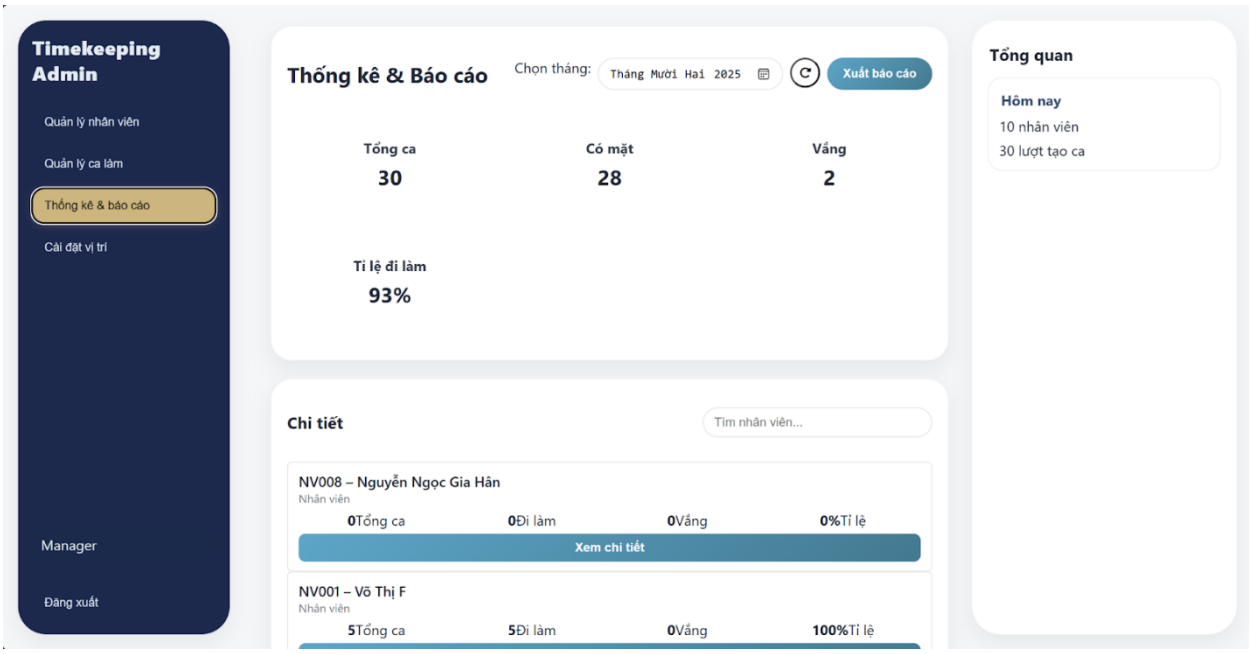


    - Ghi nhận thời gian thực: Hệ thống tự động lấy giờ hệ thống (Server time) để đảm bảo tính trung thực.

- **Nhóm chức năng Quản lý (Management - Dành cho Admin):**
    - Quản lý nhân viên: Thêm, sửa, xóa, xem danh sách nhân viên.



- Quản lý bảng công: Xem lịch sử chấm công của toàn bộ nhân viên, lọc theo ngày/tháng.



B. Yêu cầu phi chức năng (Non-Functional Requirements)

- **Hiệu năng:** Hệ thống phản hồi các API trong vòng dưới 2 giây. Chức năng nhận diện khuôn mặt xử lý trong vòng 3-5 giây.
- **Bảo mật:** Mật khẩu người dùng được mã hóa (Hash) trước khi lưu vào Firestore. API được bảo vệ bằng JWT (JSON Web Token).
- **Tính sẵn sàng:** Hệ thống hoạt động 24/7 trên nền tảng Cloud (Render & GitHub Pages).

2. Biểu đồ Ca sử dụng (Use Case Diagram)

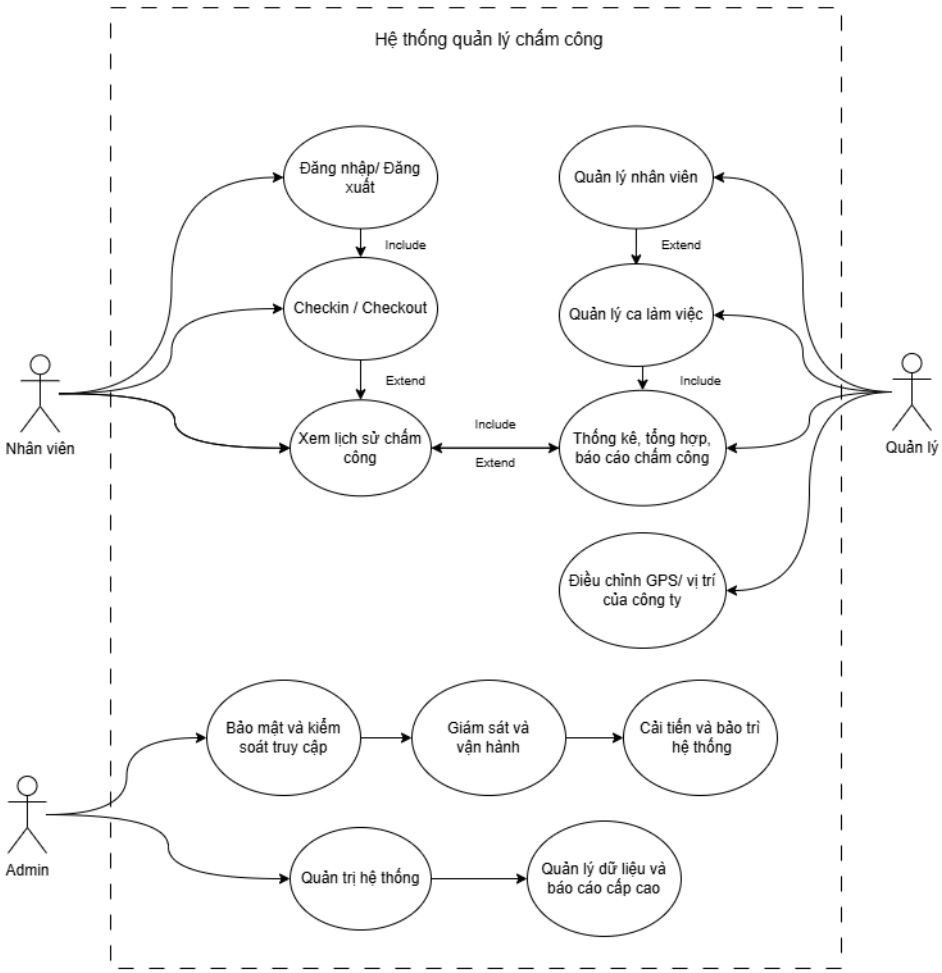
Biểu đồ này phân tách rõ ràng quyền hạn của từng tác nhân và các mối quan hệ `<<include>>` (bắt buộc có) hoặc `<<extend>>` (mở rộng/tùy chọn).

Các tác nhân chính:

- **Admin:** Quản trị viên hệ thống.
- **Employee:** Nhân viên.
- **Mailjet API (External System):** Hệ thống bên ngoài hỗ trợ gửi mail.

Danh sách các ca sử dụng chi tiết:

1. Nhóm **Quản lý tài khoản:**
    - **Đăng ký tài khoản:** Bao gồm `<<include>>` ca sử dụng **Xác thực OTP**.
    - **Xác thực OTP:** Liên kết với **Mailjet API** để gửi mã.
    - **Đăng nhập/Đăng xuất:** Bảo mật bằng JWT.
2. Nhóm **Chấm công:**
    - **Chấm công khuôn mặt (FaceID):** Chức năng lõi dành cho nhân viên.
    - **Xem lịch sử chấm công cá nhân:** Nhân viên tự theo dõi công.
3. Nhóm **Quản trị (Chỉ dành cho Admin):**
    - **Quản lý nhân viên:** CRUD (Thêm, sửa, xóa, truy vấn) hồ sơ nhân viên và dữ liệu khuôn mặt mẫu.
    - **Quản lý bảng công tổng hợp:** Xem và xuất báo cáo chấm công của toàn bộ công ty.
    - **Cấu hình hệ thống:** Thiết lập giờ làm việc, ngày lễ.



3. Thiết kế Cơ sở dữ liệu (Firestore NoSQL)

Do sử dụng **Google Cloud Firestore**, cơ sở dữ liệu được tổ chức dưới dạng các **Collections (Bộ sưu tập)** và **Documents (Tài liệu)** thay vì các bảng (Tables) quan hệ truyền thống. Cấu trúc dữ liệu dạng JSON linh hoạt.

❖ ***Collection: users (Người dùng)***

Lưu thông tin tài khoản và vai trò người dùng.

Vai trò trong hệ thống: Xác thực người dùng, phân quyền truy cập liên kết với các dữ liệu khác (attendance, shifts, faceEmbeddings)

```
{
"uid": "String (Auto ID)",
"email": "String (Unique)",
"password": "String (Hashed)",
"role": "String ('admin' | 'employee')",
"isVerified": "Boolean",
"createdAt": "Timestamp"
}
```

❖ ***Collection: employees (Hồ sơ nhân viên)***

Lưu trữ thông tin chi tiết của nhân viên (tách biệt với tài khoản login để bảo mật).

```
{
"employeeId": "String (Auto ID)",
"userId": "String (Reference to users.uid)",
"fullName": "String",
"phone": "String",
"department": "String",
"faceDescriptor": "Array (Vector dữ liệu khuôn mặt)",
"avatarUrl": "String"
}
```

❖ ***Collection: attendance (Dữ liệu chấm công)***

Lưu dữ liệu chấm công của nhân viên.

```
{
```

```
"attendanceId": "String (Auto ID)",
"employeeId": "String",
"checkInTime": "Timestamp",
"checkOutTime": "Timestamp (Optional)",
"date": "String (Format: YYYY-MM-DD)",
"status": "String ('Late' | 'OnTime')"
}
```


- ***Collection: otps (Mã xác thực tạm thời)***

Dùng để lưu mã OTP, có thời gian hết hạn (TTL).


```
{
  "email": "String",
  "otpCode": "String",
  "expiredAt": "Timestamp"
}
```


- ***Collection shifts***

Lưu thông tin ca làm việc.

Vai trò: Quản lý thời gian làm việc, gán ca cho nhân viên, hỗ trợ đối chiếu attendance

Các trường tiêu biểu:

- name
- startTime, endTime
- assignedUsers
- createdAt

- ***Collection faceEmbeddings***

Lưu dữ liệu FaceID của nhân viên.

Vai trò: So khớp khuôn mặt khi check-in, mỗi user tương ứng một embedding.

Các trường tiêu biểu:

- userId
- embedding (mảng số)
- createdAt

❖ ***Collection company_settings***

Lưu vị trí của công ty

Vai trò: Kiểm tra vị trí người dùng thực hiện checkin/checkout có khớp với vị trí được cài đặt hay không.

4. Thiết kế Kiến trúc hệ thống và API

❖ ***Kiến trúc phần mềm***

Hệ thống áp dụng mô hình **MVC (Model - View - Controller)** trên nền tảng Node.js:

- **Model:** Các Schema định nghĩa dữ liệu (tương tác với Firestore).
- **View:** Phía Client (GitHub Pages) chịu trách nhiệm hiển thị.
- **Controller:** Xử lý logic nghiệp vụ (Tính toán lương, xác thực OTP, xử lý ảnh).

❖ ***Thiết kế RESTful API***

Các API chính được xây dựng để giao tiếp giữa Frontend và Backend:

Phương thức	Endpoint (Đường dẫn)	Chức năng	Input (Body/Query)
POST	/api/auth/register	Đăng ký tài khoản	{email, password, name}
POST	/api/auth/verify-otp	Xác thực OTP	{email, otp}
POST	/api/auth/login	Đăng nhập	{email, password}
GET	/api/employees	Lấy danh sách NV	Header: Authorization (Token)
POST	/api/attendance	Check-in (Chấm công)	{employeeId, faceData}

5. Biểu đồ Tuần tự (Sequence Diagram)

Chúng ta sẽ chi tiết hóa 2 luồng xử lý phức tạp nhất trong hệ thống.

A. Luồng Đăng ký tài khoản và Xác thực OTP qua Mailjet

Luồng này thể hiện sự phối hợp giữa Frontend, Backend, Firestore và dịch vụ bên thứ ba.

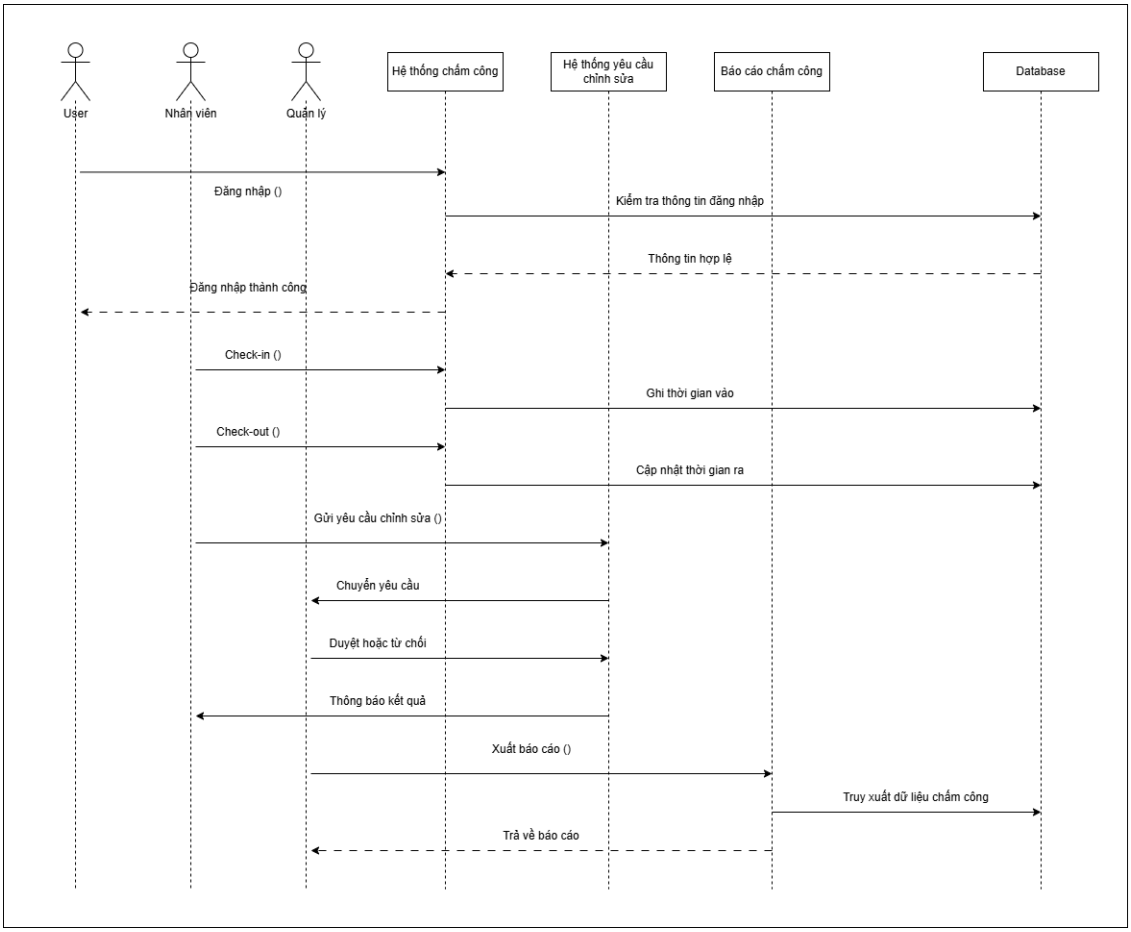
1. **User** gửi yêu cầu Đăng ký (Email, Password) từ giao diện.
2. **Frontend** gửi dữ liệu đến **AuthController** tại Backend.
3. **AuthController** gọi **OTPService** để tạo mã ngẫu nhiên.
4. **OTPService** thực hiện 2 việc song song:
    - Lưu mã OTP và thời gian hết hạn vào **Firestore (Collection otps)**.
    - Gọi **Mailjet API** thông qua HTTP Post để gửi mail cho User.
5. **User** nhận mã từ Email và nhập vào trang Xác thực trên **Frontend**.

6. **Frontend** gửi mã OTP về **AuthController**.
7. **AuthController** đối soát với dữ liệu trong **Firestore**.
8. Nếu khớp, **AuthController** cập nhật trạng thái isVerified: true trong **Collection users** và thông báo thành công.

B. Luồng Chấm công bằng nhận diện khuôn mặt (FaceID)

Đây là quy trình kết hợp giữa xử lý AI tại Client và lưu trữ dữ liệu tại Server.

1. **Employee** đứng trước Webcam, **Frontend** gọi thư viện face-api.js.
2. **face-api.js** trích xuất vector khuôn mặt (Descriptor) từ hình ảnh thực tế.
3. **Frontend** gửi ID nhân viên và Vector này về **AttendanceController**.
4. **AttendanceController** gọi **EmployeeService** để lấy vector mẫu của nhân viên đó từ **Firestore**.
5. **AttendanceController** thực hiện thuật toán so khớp (Euclidean Distance).
6. Nếu độ tương đồng đạt ngưỡng (ví dụ > 80%):
    - **AttendanceController** gọi **AttendanceService** để ghi dữ liệu (giờ vào, ngày) vào **Firestore (Collection attendance)**.
    - Phản hồi "Chấm công thành công" về **Frontend**.
7. Nếu không khớp, phản hồi lỗi "Khuôn mặt không hợp lệ".



6. Sơ đồ lớp của hệ thống (Class Diagram)

Vì hệ thống viết bằng Node.js (hướng đối tượng hoặc hướng module), sơ đồ lớp này mô tả cấu trúc các lớp logic và các thực thể dữ liệu (Entities) tương ứng với Firestore Documents.

❖ ***Các lớp thực thể (Data Entities)***

Các lớp này đại diện cho cấu trúc dữ liệu lưu trữ:

- **User:** id, email, password, role, isVerified.
- **Employee:** id, name, department, faceVector, avatar.
- **AttendanceRecord:** id, employeeId, date, checkInTime, checkOutTime, status.

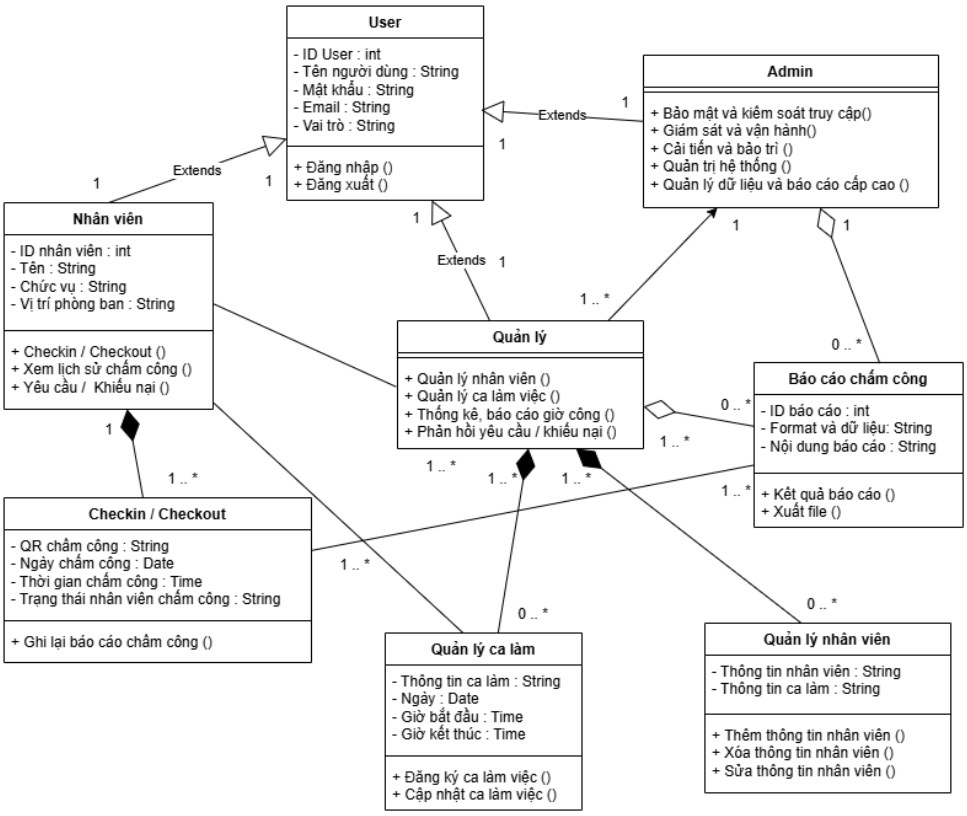
❖ ***Các lớp xử lý nghiệp vụ (Services/Controllers)***

Đây là nơi chứa logic chính của dự án:

- **AuthService:**
  - register(data): Tạo tài khoản.
  - login(email, pass): Xác thực và cấp JWT.
  - verifyOTP(email, code): Kiểm tra mã xác thực.
- **EmailService (Mailjet Adapter):**
  - sendOTP(email, code): Chịu trách nhiệm gọi Mailjet API.
- **FaceRecognitionService:**
  - compareFace(inputVector, savedVector): So sánh 2 khuôn mặt.
- **AttendanceService:**
  - processCheckIn(empId): Xử lý ghi nhận giờ vào.
  - getMonthlyReport(month): Tổng hợp dữ liệu bảng công.

Mối quan hệ:

- AuthService sử dụng EmailService để gửi mã.
- AttendanceService sử dụng FaceRecognitionService để xác thực trước khi ghi công.
- Một User có quan hệ 1-1 với một Employee.
- Một Employee có quan hệ 1-n với AttendanceRecord.



7. Sơ đồ trạng thái của bản ghi công việc (State Diagram)

A. Mục đích

Sơ đồ trạng thái mô tả các giai đoạn khác nhau của một bản ghi chấm công/công việc trong hệ thống Timekeeping. Việc này giúp đảm bảo tính minh bạch giữa nhân viên và quản lý, đồng thời cho phép xử lý các sai sót hoặc khiếu nại phát sinh trong quá trình vận hành thực tế.

B. Mô tả các trạng thái chính

Dựa trên luồng nghiệp vụ, bản ghi sẽ trải qua các trạng thái sau:

1. **Khởi tạo (Initialization):** Trạng thái bắt đầu khi nhân viên mở ứng dụng và chuẩn bị cho phiên làm việc.
2. **Đang làm việc (In Progress):** Trạng thái sau khi nhân viên thực hiện **Check-in** thành công. Lúc này hệ thống bắt đầu tính giờ làm việc thực tế.
3. **Hoàn thành (Completed):** Sau khi nhân viên thực hiện **Check-out**. Bản ghi chứa đầy đủ thông tin về giờ bắt đầu và giờ kết thúc.
4. **Chờ xác nhận (Pending Confirmation):** Dữ liệu được gửi lên hệ thống để chờ Admin hoặc quy trình tự động kiểm tra tính hợp lệ.
5. **Đã xác nhận (Confirmed):** Trạng thái khi bản ghi được duyệt là hợp lệ, không có sai sót về thời gian hay vị trí.

6. **Lưu trữ (Archived):** Trạng thái cuối cùng của vòng đời. Bản ghi được lưu trữ vĩnh viễn vào **Firestore** để phục vụ việc tổng hợp bảng lương sau này.

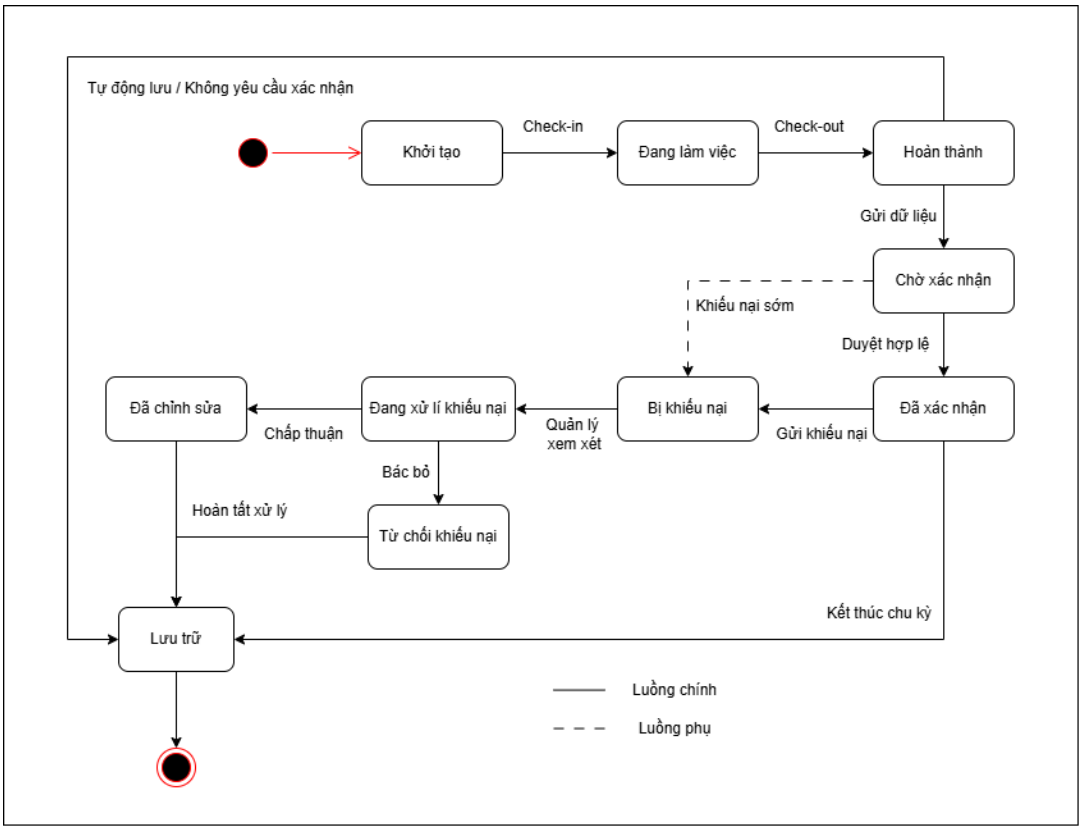
C. Quy trình xử lý khiếu nại (Dispute Flow)

Hệ thống cho phép linh hoạt xử lý các trường hợp bất thường (như quên check-in/out hoặc lỗi kỹ thuật) thông qua luồng phụ:

- **Bị khiếu nại (Disputed):** Trạng thái xảy ra khi nhân viên gửi yêu cầu chỉnh sửa thông tin (Khiếu nại sớm hoặc Gửi khiếu nại sau khi đã xác nhận).
- **Đang xử lý khiếu nại:** Quản lý (Admin) tiến hành xem xét các bằng chứng hoặc lý do nhân viên đưa ra.
- **Kết quả xử lý:**
    - **Chấp thuận:** Trạng thái chuyển sang **Đã chỉnh sửa**, cập nhật lại dữ liệu đúng và chuyển đến bước **Lưu trữ**.
    - **Bác bỏ:** Trạng thái chuyển sang **Từ chối khiếu nại**, giữ nguyên dữ liệu gốc và chuyển đến bước **Lưu trữ**.

D. Các quy tắc chuyển trạng thái đặc biệt

- **Tự động lưu / Không yêu cầu xác nhận:** Đối với các bản ghi không có dấu hiệu bất thường, hệ thống có cơ chế tự động chuyển thẳng từ các bước ban đầu sang trạng thái **Lưu trữ** để tối ưu hóa hiệu suất quản lý.
- **Tính toàn vẹn:** Một khi bản ghi đã ở trạng thái **Lưu trữ**, mọi thay đổi sẽ được ghi lại nhật ký (Log) để đảm bảo tính trung thực của hệ thống chấm công.



CHƯƠNG 4: KẾT QUẢ CÀI ĐẶT VÀ KIỂM THỬ HỆ THỐNG

1. Môi trường triển khai thực tế

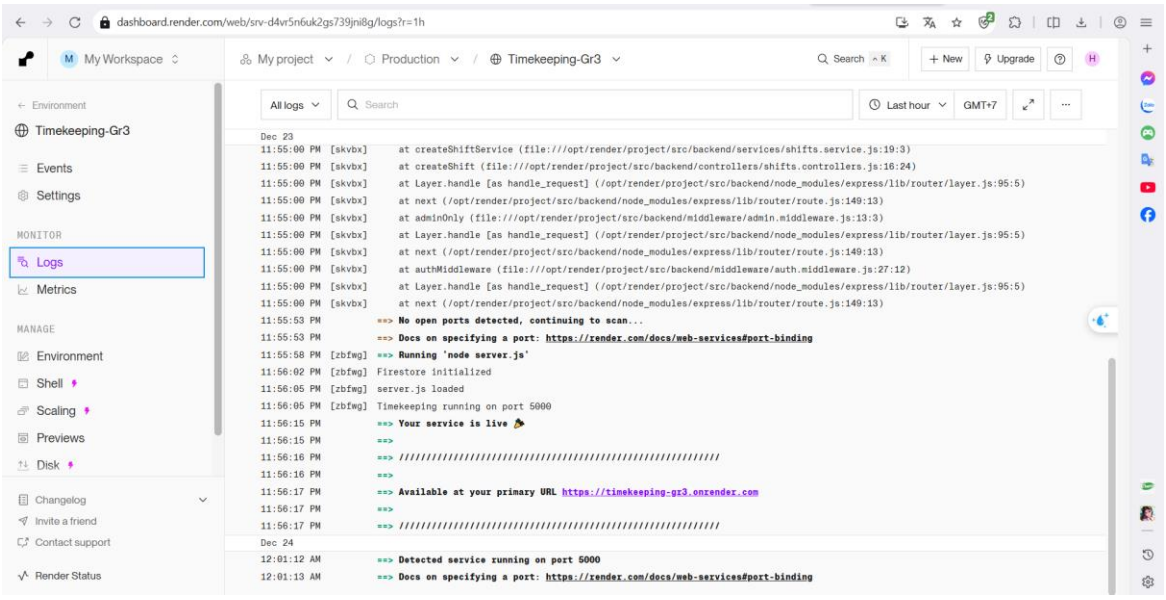
Hệ thống Timekeeping đã được triển khai hoàn chỉnh trên môi trường điện toán đám mây (Cloud Computing) với cấu hình cụ thể như sau:

- **Backend (Server-side):** Triển khai trên nền tảng **Render** (Web Service). Môi trường chạy mã: Node.js phiên bản 18.x trở lên.
- **Cơ sở dữ liệu (Database):** Sử dụng **Google Firebase Firestore** (Cloud Firestore). Dữ liệu được lưu trữ và đồng bộ hóa thời gian thực trên hạ tầng của Google Cloud.
- **Frontend (Client-side):** Triển khai trên **GitHub Pages**. Đây là dịch vụ lưu trữ web tĩnh, đảm bảo tốc độ truy cập nhanh và ổn định.
- **Dịch vụ Email API:** Sử dụng **Mailjet API** để xử lý các luồng gửi email xác thực OTP qua giao thức HTTP (Cổng 443).

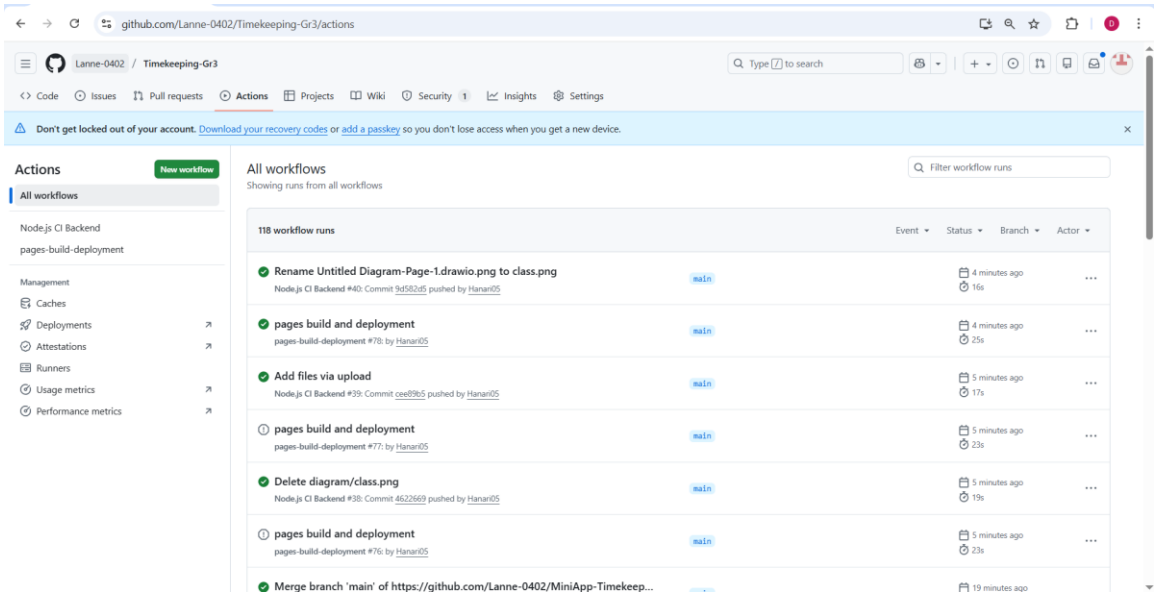
2. Kết quả triển khai quy trình CI/CD

Một trong những thành công lớn của dự án là việc áp dụng thành công quy trình tự động hóa CI/CD, giúp tối ưu hóa việc cập nhật và vận hành hệ thống.

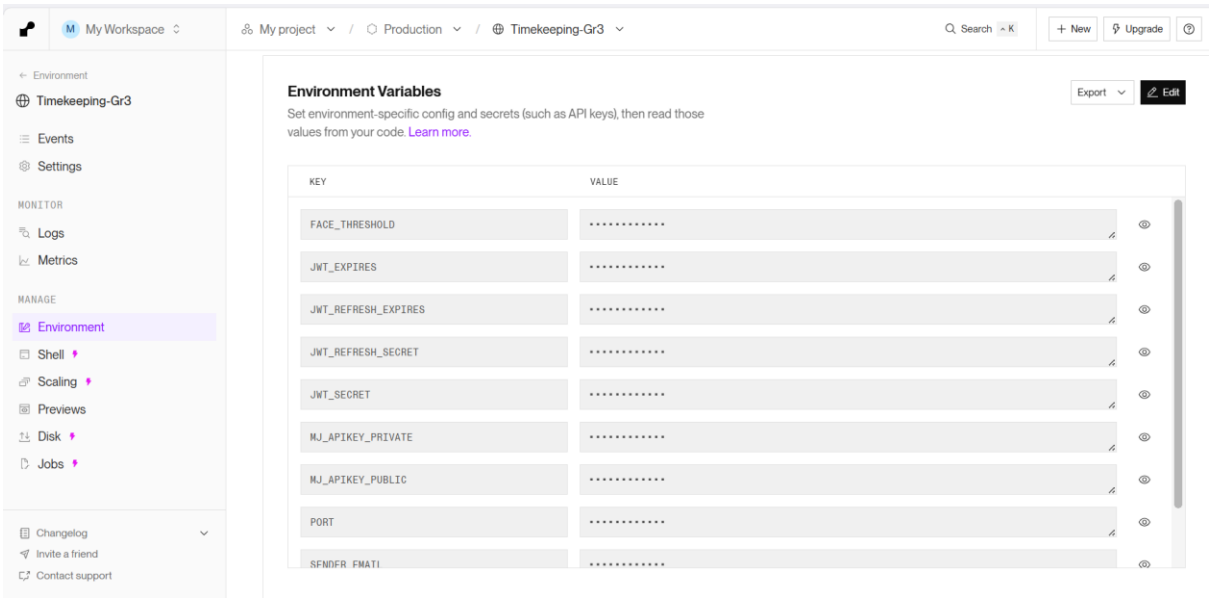
- **Tự động hóa Backend:** Thiết lập kết nối giữa GitHub và Render. Mỗi khi mã nguồn được đẩy lên nhánh main, Render sẽ tự động thực hiện quy trình Build (cài đặt dependencies) và Deploy phiên bản mới nhất mà không cần can thiệp thủ công.



- **Tự động hóa Frontend:** Sử dụng **GitHub Actions** với file cấu hình .github/workflows/deploy.yml. Quy trình này tự động hóa việc kiểm tra mã nguồn, đóng gói và cập nhật nội dung lên GitHub Pages ngay sau khi có sự thay đổi.



- **Quản lý biến môi trường:** Toàn bộ thông tin nhạy cảm (Firebase Config, Mailjet API Keys, JWT Secret) được cấu hình an toàn trong phần *Environment Variables* của Render, đảm bảo tính bảo mật tuyệt đối cho hệ thống.

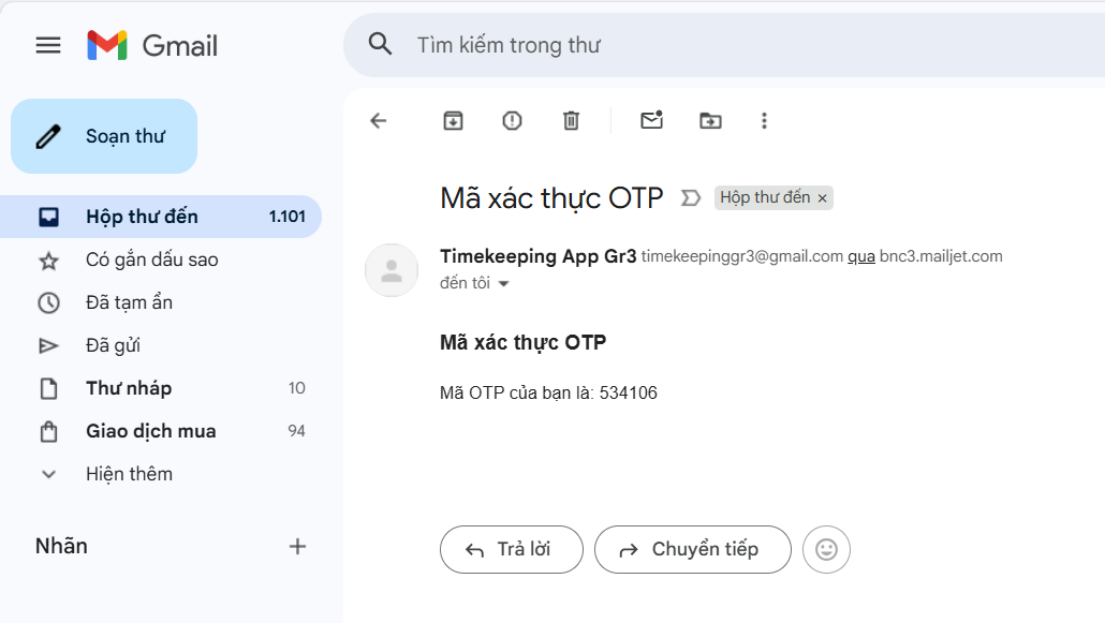


3. Kết quả thực hiện các chức năng chính

Hệ thống đã hoàn thiện và vận hành ổn định các chức năng cốt lõi sau:

❖ ***Chức năng Đăng ký tài khoản và Xác thực OTP***

- **Mô tả:** Người dùng đăng ký tài khoản và phải xác thực thông qua mã OTP gửi về email cá nhân.
- **Kết quả:** Hệ thống tích hợp thành công Mailjet API, khắc phục hoàn toàn lỗi chặn cổng SMTP trên các server Cloud. Mã OTP được gửi đi trong vòng 2-5 giây, đảm bảo trải nghiệm người dùng không bị gián đoạn.



Mã xác thực OTP được gửi về gmail của người đăng ký

❖ ***Chức năng Đăng nhập và Phân quyền***

- **Mô tả:** Hệ thống phân định rõ quyền hạn giữa Admin và Nhân viên.
- **Kết quả:** Sử dụng JWT để duy trì phiên đăng nhập. Admin có quyền truy cập vào các trang quản lý chuyên sâu, trong khi nhân viên chỉ có quyền thực hiện chấm công và xem lịch sử cá nhân.

❖ ***Chức năng Chấm công bằng nhận diện khuôn mặt (FaceID)***

- **Mô tả:** Sử dụng AI để xác thực danh tính nhân viên khi điểm danh.
- **Kết quả:** Tích hợp thành công thư viện face-api.js. Hệ thống có khả năng nhận diện chính xác khuôn mặt thông qua Webcam và đối chiếu với dữ liệu vector đã lưu trên Firestore để ghi nhận giờ công tự động.

❖ ***Chức năng Chấm công bằng nhận diện khuôn mặt kết hợp định vị GPS***

- **Mô tả** Khi nhân viên thực hiện lệnh "Check-in", trình duyệt sẽ kích hoạt đồng thời **Camera** (để quét khuôn mặt) và **Geolocation API** (để lấy tọa độ GPS thực tế).
- **Kết quả:** Hệ thống chỉ ghi nhận kết quả chấm công nếu:
  1. Khuôn mặt khớp với dữ liệu mẫu trên hệ thống.
  2. Tọa độ GPS của nhân viên nằm trong bán kính cho phép (100m) so với tọa độ đã được cấu hình của công ty.

❖ ***Chức năng Quản lý (Dashboard)***

- **Mô tả:** Hiển thị danh sách nhân viên và bảng tổng hợp công.
- **Kết quả:** Admin có thể theo dõi biến động nhân sự và tình hình đi làm theo thời gian thực nhờ khả năng cập nhật tức thời của Firebase Firestore.

4. Kết quả Kiểm thử (Testing)

Hệ thống đã trải qua quy trình kiểm thử nghiêm ngặt để đảm bảo chất lượng phần mềm.

❖ ***Kiểm thử đơn vị (Unit Test)***

- **Nội dung:** Kiểm tra các hàm xử lý logic (tạo mã OTP, kiểm tra định dạng email, tính toán thời gian chấm công).
- **Phương pháp:** Sử dụng phương pháp *Mocking* và *Time travel* để giả lập Database và đưa ra các điều kiện cùng thời gian thích hợp nhằm chạy các bộ test một cách mượt mà và trơn tru.
- **Kết quả:** 100% các hàm logic chính vượt qua các kịch bản kiểm thử, đảm bảo không có lỗi tính toán sai lệch.

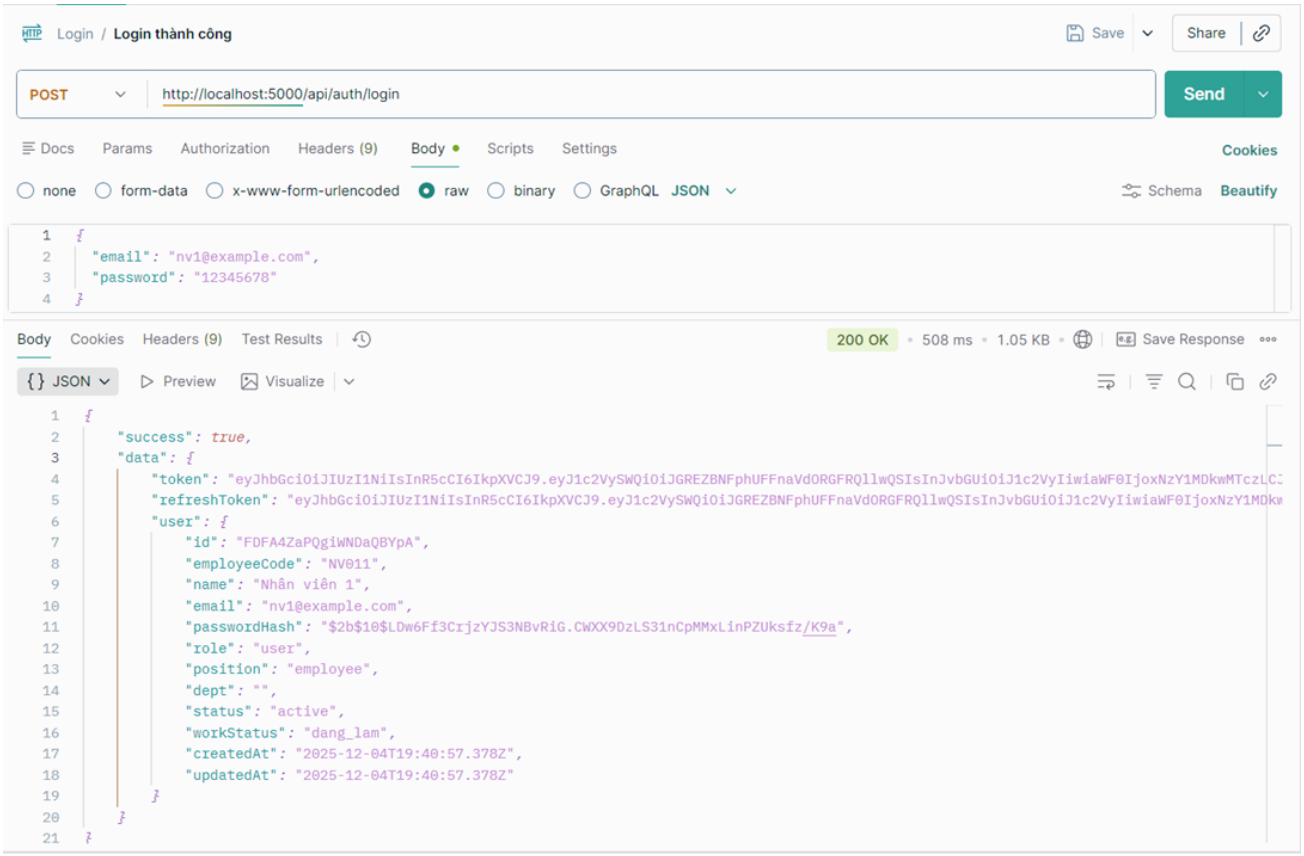
```
Test Suites: 8 passed, 8 total
Tests:       134 passed, 134 total
Snapshots:   0 total
Time:        3.446 s
Ran all test suites.
PS F:\STUDY\C_C++\NMCNPM_MySQL\ProjectApp\project\Timekeeping-Gr3\backend>
```

Kết quả toàn bộ trường hợp kiểm thử của Jest - Logic chấm công

(các kết quả chi tiết của từng trường hợp sẽ được trình bày chi tiết trong file Báo cáo kiểm thử đính kèm)

❖ ***Kiểm thử API bằng Postman***

- **Nội dung:** Gửi các request giả lập tới Backend để kiểm tra tính ổn định của các Endpoint.
- **Phương pháp:** khởi tạo Body, Header và Scripts; HTTP… để chạy và khởi động API, kiểm tra kết nối Database với Backend qua request POST.
- **Kết quả:** Tất cả các API (Auth, Employee, Attendance) đều trả về mã trạng thái 200 OK với dữ liệu JSON đúng cấu trúc thiết kế.

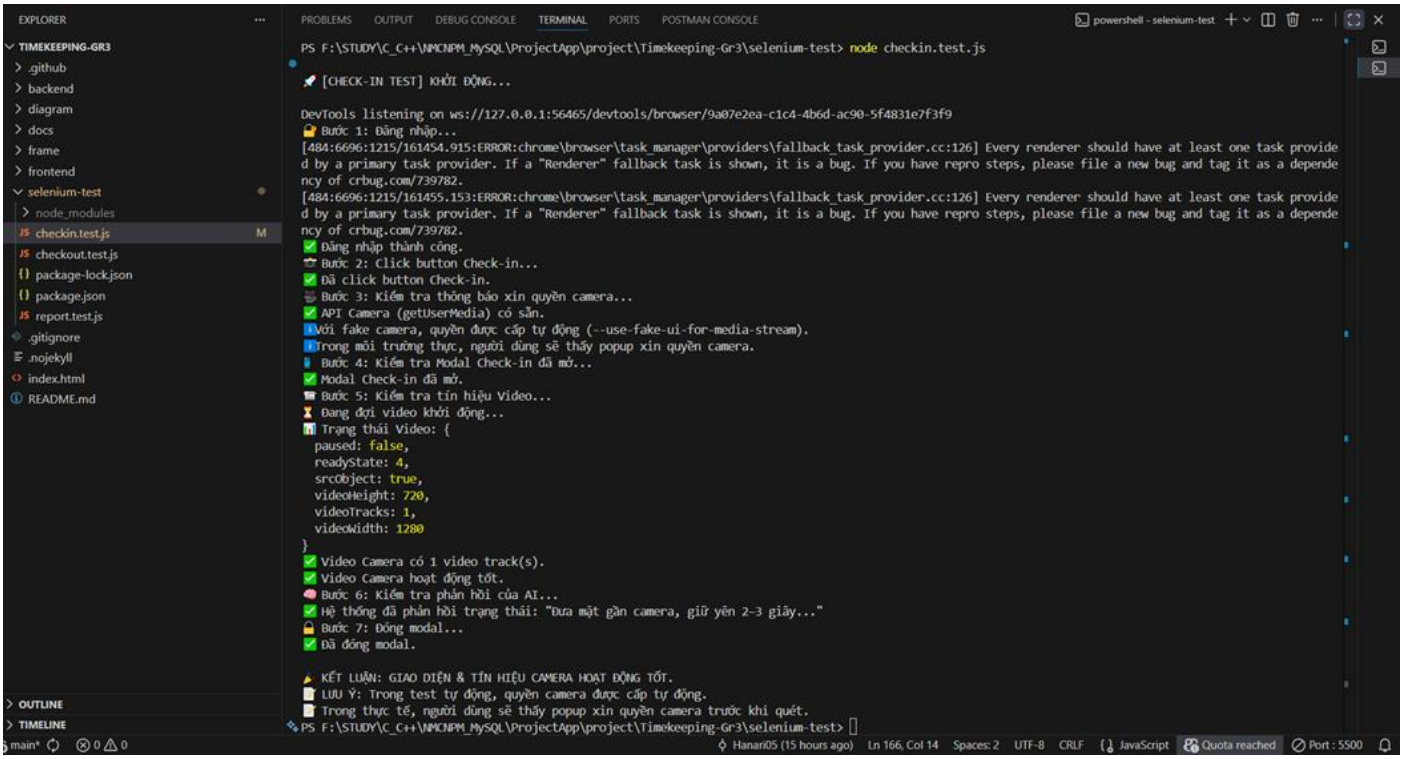


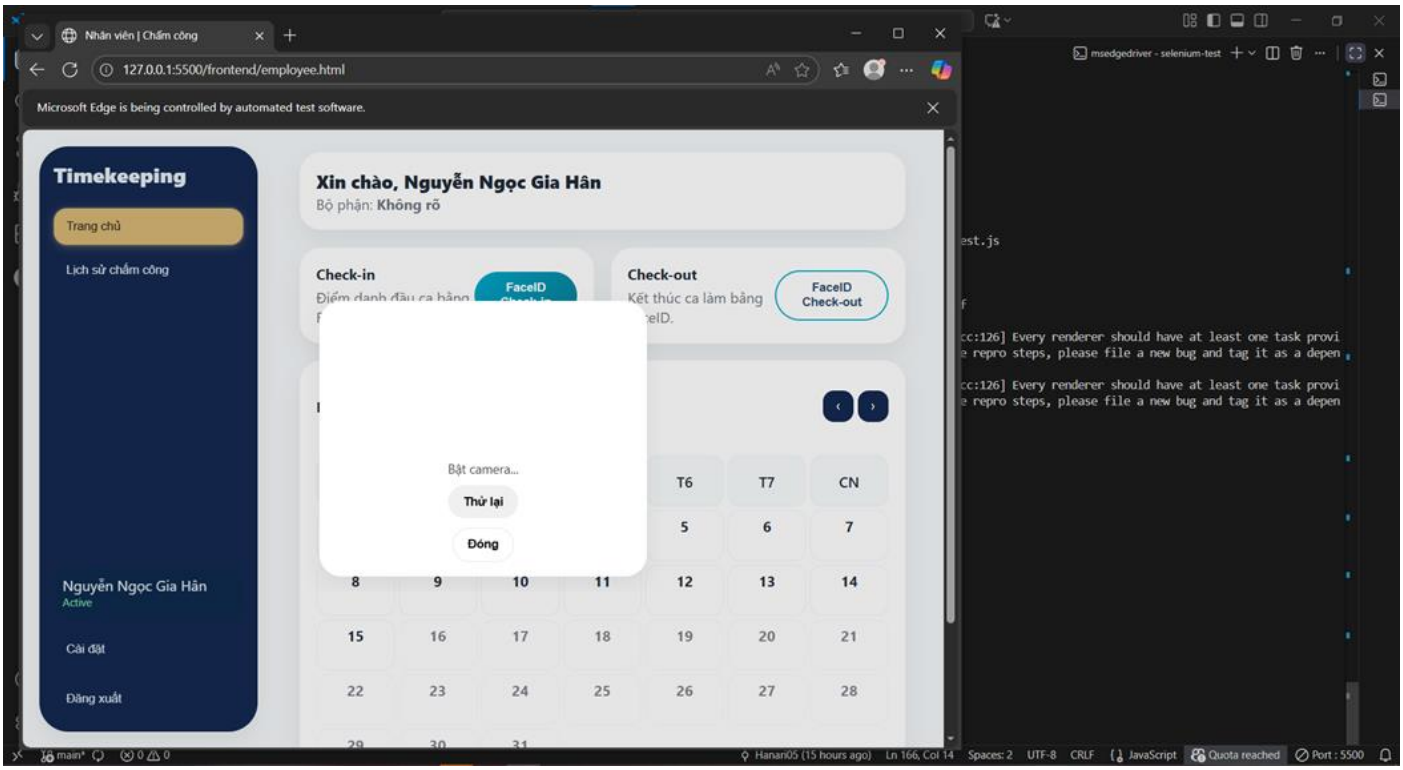
Kết quả kiểm thử của API - Đăng nhập login thành công

(các kết quả khác sẽ được trình bày chi tiết trong file Báo cáo kiểm thử đính kèm)

❖ ***Kiểm thử giao diện và chức năng bằng Selenium***

- **Nội dung:** Tự động hóa các thao tác người dùng (nhấp chuột, nhập liệu, chuyển trang) trên trình duyệt.
- **Phương pháp:** giả lập phần cứng và luồng người dùng, cho phép thao tác ảo của người dùng để chạy test và thực hiện các chức năng cốt lõi của Frontend và giao diện.
- **Kết quả:** Các luồng nghiệp vụ chính (Login -> Chấm công -> Logout) hoạt động liền mạch, không xảy ra lỗi vỡ giao diện hay treo trang web.





Kết quả kiểm thử của Selenium - Kiểm tra giao diện Check-in

(các kết quả khác sẽ được trình bày chi tiết trong file Báo cáo kiểm thử đính kèm)

5. Hạn chế của hệ thống

Mặc dù hệ thống đã đáp ứng được các yêu cầu cơ bản, nhưng do giới hạn về thời gian thực hiện, tài nguyên phần cứng và các gói dịch vụ miễn phí, hệ thống vẫn tồn tại một số hạn chế sau:

a. Hạn chế về Hạ tầng và Hiệu năng (Infrastructure & Performance)

- **Vấn đề "Cold Start" (Khởi động nguội) trên Render:** Do sử dụng gói dịch vụ miễn phí (Free Tier), Server Backend sẽ tự động rơi vào trạng thái "ngủ đông" nếu không có truy cập trong vòng 15 phút. Điều này dẫn đến việc người dùng đầu tiên truy cập sau một khoảng thời gian nghỉ sẽ gặp độ trễ lớn (có thể lên tới 30-50 giây) để Server khởi động lại.
- **Khả năng chịu tải vào giờ cao điểm:** Hệ thống chưa được kiểm thử tải (Load Testing) với lượng người dùng lớn đồng thời. Trong kịch bản thực tế tại doanh nghiệp, khi hàng trăm nhân viên cùng thực hiện Check-in vào lúc 8:00 sáng, Backend hiện tại (đơn luồng) có thể bị quá tải hoặc phản hồi chậm.

b. Hạn chế về Công nghệ Nhận diện khuôn mặt (AI/FaceID)

- **Thiếu cơ chế chống giả mạo (Liveness Detection):** Hệ thống hiện tại sử dụng thư viện face-api.js để so sánh vector khuôn mặt, nhưng chưa tích hợp các thuật toán phát hiện sự sống (Liveness Detection). Do đó, hệ thống vẫn có rủi ro bảo mật nếu người dùng sử dụng ảnh chụp chất lượng cao hoặc video của nhân viên khác để điểm danh hộ.
- **Độ phụ thuộc vào môi trường ánh sáng:** Độ chính xác của việc nhận diện bị ảnh hưởng lớn bởi điều kiện ánh sáng tại nơi đặt thiết bị (Webcam). Trong điều kiện ngược sáng hoặc thiếu sáng, khả năng nhận diện giảm đáng kể.
- **Phụ thuộc vào cấu hình thiết bị Client:** Do sử dụng mô hình xử lý AI phía Client (Client-side rendering), tốc độ nhận diện phụ thuộc vào cấu hình máy tính/điện thoại của người dùng. Các thiết bị cũ, RAM thấp có thể gặp tình trạng giật, lag khi xử lý luồng video.

c. Hạn chế về Nghiệp vụ và Báo cáo (Business Logic)

- **Khả năng tùy biến báo cáo còn thấp:** Hệ thống hiện chỉ hỗ trợ xem danh sách chấm công cơ bản. Chưa hỗ trợ các báo cáo phức tạp như: tính tổng giờ làm việc thực tế, tính lương tự động, xuất báo cáo ra file Excel/PDF để phục vụ kế toán.
- **Phụ thuộc vào giới hạn của bên thứ ba:**
    - **Mailjet API:** Gói miễn phí giới hạn số lượng email gửi trong ngày (200 emails/ngày). Nếu quy mô doanh nghiệp lớn hơn, hệ thống sẽ bị gián đoạn dịch vụ gửi OTP.
    - **Firestore:** Việc truy vấn dữ liệu phức tạp (Complex Queries) để tổng hợp báo cáo trên NoSQL khó khăn hơn so với SQL truyền thống, đôi khi phải xử lý logic lọc dữ liệu tại phía Client gây tốn băng thông.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận

Sau quá trình nghiên cứu, phân tích và triển khai, đồ án **"Xây dựng hệ thống chấm công trực tuyến (Timekeeping)"** đã hoàn thành các mục tiêu đề ra ban đầu. Nhóm thực hiện đã xây dựng thành công một ứng dụng web hoàn chỉnh, áp dụng các công nghệ hiện đại và quy trình phát triển phần mềm tiên tiến.

Những kết quả đạt được:

- **Về mặt chức năng:** Hệ thống đáp ứng đầy đủ các nghiệp vụ cốt lõi như Đăng ký/Đăng nhập bảo mật, Xác thực OTP qua Email và Chấm công bằng nhận diện khuôn mặt (FaceID).
- **Về mặt kỹ thuật:**
    - Làm chủ được kiến trúc **Client-Server** với Node.js và React/HTML5.
    - Triển khai thành công cơ sở dữ liệu đám mây **Google Firebase Firestore** với khả năng đồng bộ thời gian thực.
    - Giải quyết triệt để bài toán gửi Email trên hạ tầng Cloud bằng việc tích hợp **Mailjet API**.
- **Về mặt quy trình (DevOps):** Thiết lập thành công đường ống **CI/CD** tự động hóa từ GitHub sang Render và GitHub Pages. Đây là điểm sáng của đồ án, giúp giảm thiểu thao tác thủ công và lỗi trong quá trình vận hành.

Thông qua đồ án này, sinh viên không chỉ củng cố kiến thức về lập trình mà còn tích lũy được kinh nghiệm thực tế về quản lý dự án (Jira), kiểm thử hệ thống và giải quyết các vấn đề phát sinh khi triển khai ứng dụng lên môi trường Internet.

2. Hướng phát triển

Để hệ thống có thể áp dụng rộng rãi trong các doanh nghiệp thực tế, nhóm đề xuất các hướng nâng cấp và phát triển trong tương lai:

1. **Nâng cấp thuật toán AI:**
    - Tích hợp công nghệ **Liveness Detection** (Phát hiện thực thể sống) để ngăn chặn việc gian lận chấm công bằng hình ảnh hoặc video (Spoofing attacks).
    - Cải thiện độ chính xác của model nhận diện trong điều kiện ánh sáng yếu.
2. **Tối ưu hóa hạ tầng:**
    - Nâng cấp gói dịch vụ Cloud (Render Paid Plan) hoặc chuyển sang kiến trúc **Serverless** để loại bỏ hoàn toàn vấn đề "Cold Start" (khởi động chậm), đảm bảo hệ thống luôn sẵn sàng 24/7.
    - Sử dụng Redis để Caching (lưu bộ nhớ đệm) dữ liệu nhân viên, giúp giảm tải cho Firestore và tăng tốc độ truy vấn.
3. **Mở rộng nền tảng:**

- Phát triển phiên bản ứng dụng di động (Mobile App) sử dụng **React Native** để nhân viên có thể chấm công tiện lợi qua GPS và Camera điện thoại.
- Xây dựng module **Báo cáo chuyên sâu**: Cho phép xuất dữ liệu bảng công ra file Excel (.xlsx) hoặc PDF để bộ phận kế toán - nhân sự dễ dàng tính lương.

4. **Link GitHub Pages Web App Timekeeping:**
   - ***Link Web App Timekeeping***
   - ***Link Repo Timekeeping-Gr3***

LỜI KẾT

Đồ án **'Xây dựng hệ thống Timekeeping'** là cơ hội quý giá để nhóm em hệ thống hóa toàn bộ quy trình phát triển phần mềm (SDLC) đã được học, từ khâu thu thập yêu cầu, phân tích thiết kế, lập trình kiểm thử cho đến triển khai vận hành.

Trong quá trình thực hiện, chúng em đã nỗ lực áp dụng các tiêu chuẩn công nghệ hiện đại sát với thực tế doanh nghiệp nhất có thể. Về mặt kỹ thuật, việc làm chủ kiến trúc **Client-Server** với **Node.js** và **Firebase** đã giúp tạo ra một hệ thống có tính tương tác cao. Về mặt quản lý, việc áp dụng mô hình **Agile/Scrum** và quản lý tiến độ qua **Jira** đã giúp nhóm duy trì được sự nhịp nhàng trong công việc. Đặc biệt, việc triển khai thành công quy trình **DevOps (CI/CD)** không chỉ giúp tối ưu hóa thời gian phát triển mà còn thay đổi tư duy của chúng em từ việc 'viết code cho chạy' sang 'xây dựng hệ thống để vận hành bền vững'.

Nhìn lại toàn bộ quá trình, chúng em nhận thấy mình đã đạt được các mục tiêu sau:

1. Hiểu sâu sắc về cách tích hợp các dịch vụ bên thứ ba (Third-party APIs) vào hệ thống lõi.
2. Nắm vững quy trình kiểm thử và đảm bảo chất lượng phần mềm trước khi phát hành.
3. Hình thành tư duy xử lý sự cố linh hoạt khi môi trường triển khai thay đổi.

Mặc dù sản phẩm vẫn còn những hạn chế về mặt tài nguyên phần cứng và cần tối ưu thêm về thuật toán AI, nhưng đây sẽ là nền tảng vững chắc để chúng em tiếp tục phát triển các ứng dụng phức tạp hơn trong tương lai.

Chúng em xin chân thành cảm ơn **Học viện Công nghệ Bưu chính Viễn thông** và đặc biệt là **Thầy Châu Văn Vân** đã tận tình hướng dẫn, tạo điều kiện tốt nhất để chúng em hoàn thành đồ án này.

TÀI LIỆU THAM KHẢO

1. ***[API Postman] Bài 4 - Phân tích tài liệu API và viết API test case***
2. ***Jest Testing Framework***
3. ***Web Services*** *(Host dynamic web apps (Express, Django, etc.) at a public URL.)*
4. ***Triển khai ứng dụng Node Express trên Render***
5. ***Securing your GitHub Pages site with HTTPS***
6. ***Kiểm thử API bằng Postman***