TRƯỜNG ĐẠI HỌC TRÀ VINH
KHOA KỸ THUẬT VÀ CÔNG NGHỆ
BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

CÔNG NGHỆ PHẦN MỀM

ỨNG DỤNG QUẢN LÝ LỊCH TRÌNH CÁ NHÂN

Giáo viên giảng dạy:

TS. Nguyễn Bảo Ân

Sinh viên thực hiện:

110122193 Thạch Thị Huệ Trinh DA22TTC

110122196 Lê Xuân Trường DA22TTC

Trà Vinh, tháng 7 năm 2025

TRƯỜNG ĐẠI HỌC TRÀ VINH
KHOA KỸ THUẬT VÀ CÔNG NGHỆ
BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

CÔNG NGHỆ PHẦN MỀM

ỨNG DỤNG QUẢN LÝ LỊCH TRÌNH CÁ NHÂN

Giáo viên giảng dạy:

TS. Nguyễn Bảo Ân

Sinh viên thực hiện:

110122193 Thạch Thị Huệ Trinh DA22TTC

110122196 Lê Xuân Trường DA22TTC

Trà Vinh, tháng 7 năm 2025

NHẬN XÉT CỦA GIÁO VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Trà Vinh, ngày ..... tháng …… năm ……

Giáo viên giảng dạy

(Ký tên và ghi rõ họ tên)

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

CHƯƠNG 1. GIỚI THIỆU

1.1. Tên đề tài và chủ đề

Tên đề tài: Ứng dụng quản lý lịch trình cá nhân

Chủ đề nghiên cứu: Phát triển một ứng dụng web nhằm hỗ trợ người dùng trong việc quản lý và theo dõi lịch trình cá nhân với các chế độ hiển thị theo tháng. Hệ thống cho phép thực hiện các thao tác nghiệp vụ cơ bản như tạo mới, chỉnh sửa, xoá sự kiện, đồng thời đảm bảo tính nhất quán và liên tục của dữ liệu thông qua cơ chế đồng bộ với máy chủ thông qua giao tiếp API.

Trong quá trình xây dựng hệ thống, đề tài ứng dụng các công nghệ và công cụ tiêu biểu trong lĩnh vực công nghệ phần mềm hiện nay, cụ thể như sau:

- Thiết kế giao diện người dùng (Frontend): Xây dựng giao diện trực quan, tối ưu trải nghiệm người dùng, hỗ trợ tương tác lịch trình thông qua trình duyệt web, sử dụng các công nghệ như HTML, CSS và JavaScript.

- Phát triển hệ thống API theo kiến trúc RESTful (Backend): Thiết kế và triển khai các điểm cuối (endpoint) phục vụ cho việc lưu trữ, truy xuất và xử lý dữ liệu lịch trình một cách hiệu quả.

- Quản lý mã nguồn với GitHub: Đảm bảo kiểm soát phiên bản, đồng bộ hóa tiến độ phát triển và hỗ trợ làm việc nhóm trong môi trường phân tán.

Ứng dụng các công cụ hỗ trợ phát triển phần mềm, bao gồm:

- **Jira**: Quản lý tiến độ và công việc theo mô hình Agile, hỗ trợ phân chia nhiệm vụ và theo dõi tiến trình phát triển.

- **Docker**: Tạo lập môi trường phát triển và triển khai nhất quán, đảm bảo tính đồng bộ giữa các môi trường hệ thống.

- **Postman**: Hỗ trợ kiểm thử và mô phỏng các truy vấn đến API một cách trực quan và hiệu quả.

- **Swagger**: Tài liệu hóa API và cung cấp giao diện kiểm thử tự động, nâng cao khả năng kiểm tra và bảo trì hệ thống.

1.2. Mục tiêu của ứng dụng

Mục tiêu chính của đề tài là xây dựng một hệ thống ứng dụng web cho phép người dùng quản lý lịch trình cá nhân một cách hiệu quả, trực quan và có khả năng mở rộng. Ứng dụng đáp ứng các chức năng cốt lõi bao gồm:

- Hiển thị lịch trình theo chế độ tháng, hỗ trợ quan sát tổng thể các sự kiện đã được thiết lập.

- Cung cấp khả năng tạo mới, chỉnh sửa, xoá sự kiện tương ứng với từng ngày cụ thể trên giao diện lịch.

- Hiển thị chi tiết nội dung của các sự kiện khi tương tác với từng ô ngày.

- Tự động cập nhật và hiển thị lại giao diện khi có sự thay đổi dữ liệu.

- Đồng bộ hoá dữ liệu với máy chủ thông qua hệ thống API, đảm bảo tính nhất quán, an toàn và khả năng lưu trữ lâu dài.

1.3. Lí do chọn đề tài

Quản lý lịch trình là một nhu cầu phổ biến trong nhiều lĩnh vực như giáo dục, hành chính, doanh nghiệp và đời sống cá nhân. Việc ghi nhớ và sắp xếp thời gian một cách hợp lý đóng vai trò quan trọng trong việc nâng cao hiệu suất làm việc và tổ chức công việc khoa học. Tuy nhiên, phần lớn các giải pháp hiện tại thường tích hợp trong các hệ sinh thái phức tạp, yêu cầu tài khoản đồng bộ, hoặc không phù hợp với các tình huống sử dụng độc lập, đơn giản.

Với mục tiêu xây dựng một ứng dụng có giao diện trực quan, chức năng tập trung, dễ sử dụng và dễ triển khai, đề tài hướng đến việc giải quyết một cách cụ thể bài toán quản lý lịch trình cá nhân theo cách tiếp cận nhẹ, hiệu quả và khả thi.

Đồng thời, việc lựa chọn đề tài còn nhằm hiện thực hóa một hệ thống phần mềm hoàn chỉnh theo đúng các giai đoạn của quy trình phát triển phần mềm hiện đại, từ đặc tả yêu cầu, thiết kế, lập trình đến kiểm thử và triển khai. Đây là một trong những hướng tiếp cận phù hợp để đánh giá khả năng áp dụng lý thuyết vào thực tiễn cũng như kiểm chứng các nguyên tắc thiết kế phần mềm thông qua sản phẩm cụ thể.

1.4. Tính cần thiết của công nghệ Cloud / Microservices

Trong xu hướng phát triển phần mềm hiện nay, các hệ thống ngày càng yêu cầu khả năng hoạt động ổn định, linh hoạt và hỗ trợ truy cập từ nhiều thiết bị. Việc ứng dụng công nghệ **Cloud** và **Microservices** trở nên cần thiết để đáp ứng những yêu cầu đó.

Cloud cho phép lưu trữ và xử lý dữ liệu tập trung, giúp người dùng dễ dàng truy cập và đồng bộ hóa thông tin mọi lúc, mọi nơi. Với ứng dụng quản lý lịch trình cá nhân, Cloud hỗ trợ tăng khả năng lưu trữ, bảo mật và dễ dàng mở rộng khi số lượng người dùng tăng lên.

Microservices giúp tách hệ thống thành các dịch vụ nhỏ độc lập, mang lại tính linh hoạt, dễ bảo trì và mở rộng. Mô hình này phù hợp với các hệ thống định hướng phát triển lâu dài, khi cần nâng cấp hoặc bổ sung các chức năng mới.

Mặc dù hệ thống hiện tại được xây dựng theo mô hình nguyên khối (Monolithic) để phù hợp với phạm vi đồ án, nhưng việc ứng dụng Cloud và Microservices trong thực tế là hướng đi phù hợp nhằm đảm bảo hiệu quả vận hành và khả năng mở rộng trong tương lai.

1.5. Các khía cạnh về bảo mật và an toàn phần mềm

Để đảm bảo an toàn thông tin người dùng và nâng cao chất lượng hệ thống, đề tài đã tích hợp các giải pháp bảo mật phù hợp trong quá trình thiết kế và phát triển. Cụ thể:

- **Bảo mật mật khẩu người dùng:** Hệ thống sử dụng thư viện bcrypt để băm mật khẩu trước khi lưu trữ vào cơ sở dữ liệu, nhằm hạn chế nguy cơ rò rỉ thông tin nếu xảy ra sự cố bảo mật.
- **Xác thực người dùng bằng Token:** Quá trình xác thực và quản lý phiên làm việc của người dùng được thực hiện thông qua JWT (JSON Web Token). Việc sử dụng token giúp kiểm soát quyền truy cập, đảm bảo chỉ những người dùng hợp lệ mới có thể thao tác với dữ liệu cá nhân.

- **Bảo vệ dữ liệu cá nhân:** Dữ liệu liên quan đến lịch trình cá nhân được tổ chức và quản lý theo từng tài khoản riêng biệt, đảm bảo không có sự chia sẻ hoặc truy cập trái phép giữa các người dùng.
- **Đảm bảo chất lượng hệ thống:** Tất cả chức năng chính bao gồm đăng ký, đăng nhập, quản lý lịch trình (thêm, sửa, xóa) đều được kiểm thử đầy đủ, đảm bảo hệ thống hoạt động đúng yêu cầu và hạn chế lỗi phát sinh.

Việc áp dụng các biện pháp này không chỉ nâng cao tính bảo mật cho hệ thống mà còn giúp đáp ứng các yêu cầu cơ bản về an toàn thông tin trong phát triển phần mềm.

CHƯƠNG 2. PHÂN TÍCH YÊU CẦU

2.1. Các chức năng chính của hệ thống (Functional Requirements).

Các yêu cầu chức năng mô tả các hành vi, tác vụ mà hệ thống cần thực hiện nhằm đáp ứng các mục tiêu sử dụng của người dùng. Đối với ứng dụng quản lý lịch trình cá nhân, các chức năng chính bao gồm:

2.1.1. Đăng ký tài khoản người dùng

- Cho phép người dùng tạo tài khoản bằng tên, số điện thoại, mật khẩu và các thông tin cơ bản.

- Kiểm tra tính hợp lệ của thông tin đầu vào (số điện thoại hợp lệ).

- Thông báo cho người dùng khi đăng ký thành công hoặc khi xảy ra lỗi.

2.1.2. Đăng nhập và xác thực người dùng

- Cho phép người dùng đăng nhập bằng tên đăng nhập, mật khẩu đã đăng ký.

- Xác thực thông tin đăng nhập qua hệ thống backend.

- Cung cấp chức năng lưu phiên đăng nhập và đăng xuất an toàn.

2.1.3. Khôi phục mật khẩu

- Cung cấp chức năng "Quên mật khẩu" cho người dùng nhập số điện thoại để khôi phục tài khoản.

- Gửi mã xác nhận hoặc liên kết đặt lại mật khẩu qua số điện thoại.

2.1.4. Quản lý lịch trình cá nhân

- Thêm sự kiện: Cho phép người dùng thêm mới sự kiện với các thông tin như tiêu đề, mô tả, ngày, giờ.

- Hiển thị lịch: Giao diện lịch tháng trực quan với khả năng hiển thị sự kiện tương ứng với từng ngày.

- Sửa sự kiện: Cho phép người dùng chỉnh sửa lại các sự kiện phù hợp với mục đính cá nhân.

- Xóa sự kiện: Hỗ trợ người dùng xóa sự kiện không còn cần thiết.

2.1.5. Quản lý nhiều sự kiện trong cùng một ngày

- Cho phép người dùng thêm nhiều sự kiện vào cùng một ngày.

- Giao diện popup hiện danh sách sự kiện của ngày được chọn.

2.1.6. Cài đặt tài khoản

- Hiển thị tên tài khoản: Cho phép người dùng kiểm tra nhanh thông tin tên tài khoản (username) hiện tại đang được dùng trong ứng dụng.

- Đổi mật khẩu: Cho phép người dùng chủ động thay đổi mật khẩu để đảm bảo an toàn bảo mật tài khoản. Việc thay đổi mật khẩu yêu cầu người dùng cung cấp đầy đủ thông tin như mật khẩu hiện tại, mật khẩu mới. Hệ thống thực hiện kiểm tra hợp lệ trước khi cập nhật.

- Đăng xuất: Cho phép người dùng kết thúc phiên làm việc hiện tại và quay trở lại giao diện đăng nhập. Việc đăng xuất giúp đảm bảo an toàn thông tin, đặc biệt trong trường hợp người dùng sử dụng thiết bị công cộng hoặc không thuộc sở hữu cá nhân.

2.1.7. Giao diện người dùng thân thiện

- Hệ thống cung cấp giao diện dạng lịch.

- Các sự kiện hiển thị trực tiếp trong ô ngày tương ứng.

- Giao diện phản hồi nhanh với các thao tác: click vào ngày, chọn sự kiện,...

2.1.8. Đồng bộ hóa dữ liệu với backend

- Tất cả các thao tác tạo, sửa, xóa, lấy lịch trình đều được xử lý thông qua các API RESTful kết nối với backend.

- Đảm bảo dữ liệu được lưu trữ an toàn và đồng bộ giữa các thiết bị.

2.2. Các yêu cầu phi chức năng (Non-functional Requirements).

Hệ thống Quản lý lịch trình cá nhân cần đáp ứng một số yêu cầu phi chức năng nhằm đảm bảo chất lượng và hiệu quả hoạt động như sau:

- Hiệu năng (Performance): Hệ thống đảm bảo khả năng phản hồi nhanh chóng. Thời gian xử lý các thao tác chính như đăng nhập, hiển thị lịch, thêm, sửa, xoá sự kiện không vượt quá 2-3 giây để đảm bảo trải nghiệm người dùng mượt mà.

- Bảo mật (Security): Hệ thống đảm bảo an toàn cho dữ liệu người dùng. Thông tin tài khoản như tên đăng nhập, mật khẩu được mã hoá khi lưu trữ. Đồng thời, hệ thống phải có cơ chế xác thực, đảm bảo mỗi người chỉ truy cập được dữ liệu lịch trình cá nhân của chính họ.

- Khả năng mở rộng (Scalability): Hệ thống được xây dựng theo mô hình RESTful API tách biệt frontend và backend, thuận lợi cho việc mở rộng chức năng hoặc tích hợp với các hệ thống khác trong tương lai.

- Tính dễ sử dụng (Usability): Giao diện người dùng thân thiện, trực quan, dễ sử dụng với người không am hiểu công nghệ. Các thao tác như thêm sự kiện, chỉnh sửa, xoá hay chuyển đổi các tháng rõ ràng, dễ thực hiện. Hệ thống cung cấp thông báo, cảnh báo hoặc xác nhận khi người dùng thao tác.

- Khả năng hoạt động ổn định (Reliability): Hệ thống đảm bảo hoạt động liên tục, hạn chế lỗi phát sinh trong quá trình sử dụng. Các chức năng quan trọng như lưu dữ liệu, hiển thị lịch phải luôn sẵn sàng và chính xác để người dùng có thể sử dụng mọi lúc, mọi nơi.

- Khả năng bảo trì (Maintainability): Mã nguồn cần được tổ chức rõ ràng, dễ đọc, dễ bảo trì. Các thành phần như giao diện người dùng, API được thiết kế riêng biệt giúp thuận lợi trong việc sửa lỗi, nâng cấp hoặc thay đổi công nghệ trong tương lai.

CHƯƠNG 3. THIẾT KẾ HỆ THỐNG

3.1. Kiến trúc tổng thể

3.1.1. Mô hình kiến trúc hệ thống

Hệ thống được xây dựng dựa trên mô hình Client - Server kết hợp RESTful API nhằm đảm bảo sự phân tách rõ ràng giữa phần giao diện người dùng (Frontend) và phần xử lý nghiệp vụ, truy xuất dữ liệu (Backend). Hệ thống sử dụng cơ sở dữ liệu MongoDB để lưu trữ các thông tin liên quan đến tài khoản người dùng và lịch trình sự kiện.

Mô hình tổng thể bao gồm các thành phần chính sau:

- **Frontend (Client) - Giao diện người dùng:**

+ Được xây dựng bằng các công nghệ HTML, CSS (Tailwind CSS), JavaScript.

+ Giao diện cho phép người dùng thao tác trực tiếp với hệ thống: đăng ký, đăng nhập, tạo mới, chỉnh sửa, xoá sự kiện, và quản lý lịch trình cá nhân.

+ Kết nối tới Backend thông qua API HTTP.

- **Backend (Server API) - Xử lý nghiệp vụ:**

+ Sử dụng Node.js và Express.js làm nền tảng chính.

+ Áp dụng kiến trúc MVC (Model - View - Controller) giúp quản lý mã nguồn rõ ràng, dễ bảo trì.

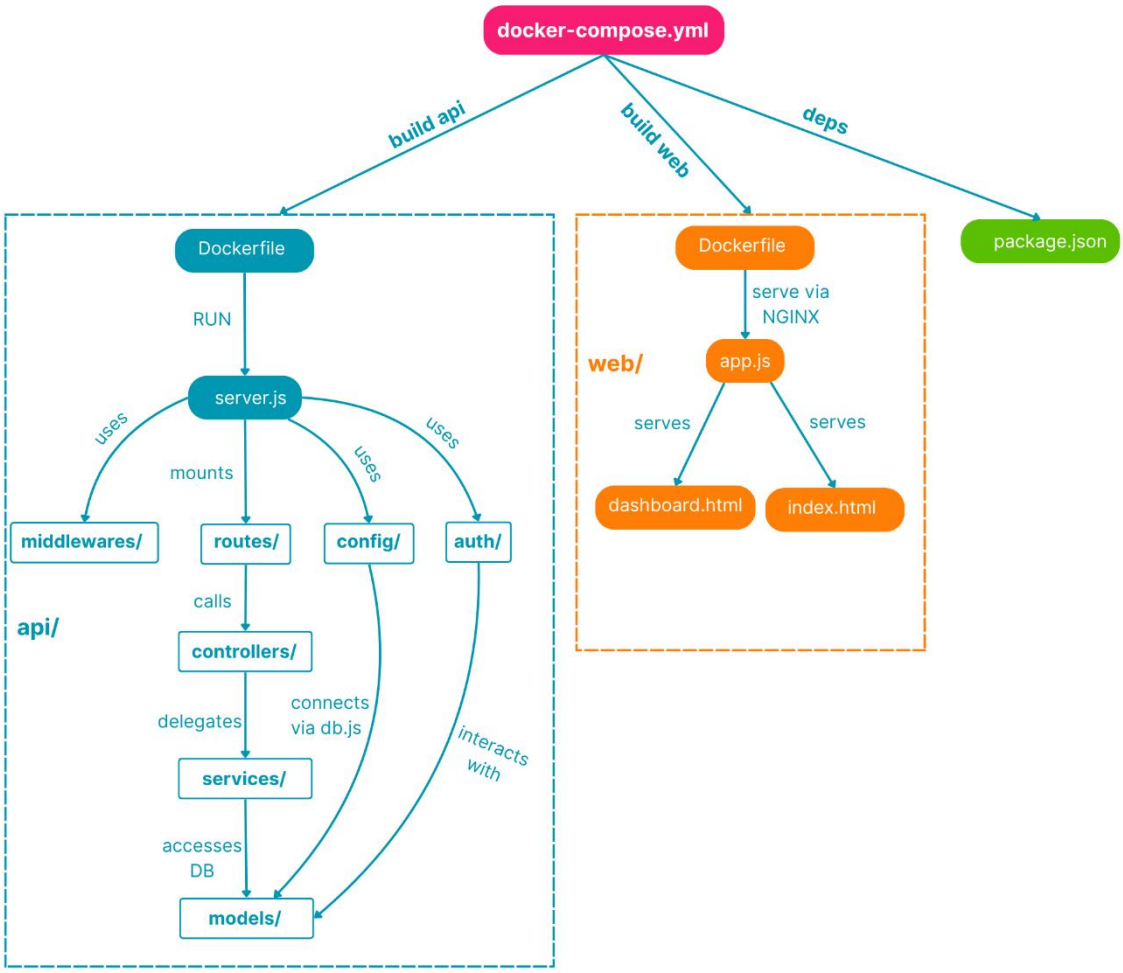
+ Xây dựng các API dạng RESTful cho phép frontend gửi request và nhận response dưới dạng JSON.

+ Bảo mật xác thực người dùng bằng JWT (JSON Web Token).

- **Database (Cơ sở dữ liệu):**

Lưu trữ dữ liệu thông qua MongoDB, một hệ quản trị cơ sở dữ liệu NoSQL linh hoạt, phù hợp cho việc lưu trữ lịch trình và tài khoản người dùng.

3.1.2. Sơ đồ kiến trúc hệ thống



Hình 3.1 Sơ đồ kiến trúc hệ thống

Mô tả hoạt động:

Sơ đồ trên minh họa hoạt động của một ứng dụng web được container hóa, phân tách thành hai dịch vụ chính: API Backend và Web Frontend, được điều phối bởi Docker Compose.

Khởi tạo và Triển khai:

- docker-compose.yml là file cấu hình trung tâm. Nó chỉ dẫn Docker "build" (xây dựng) các image (ảnh) cho cả dịch vụ API và dịch vụ Web từ các Dockerfile tương ứng. Quá trình này cũng xử lý các "deps" (phụ thuộc) được định nghĩa trong package.json.

- Khi docker-compose khởi chạy, các container riêng biệt cho API và Web sẽ được tạo và khởi động dựa trên các image đã xây dựng.

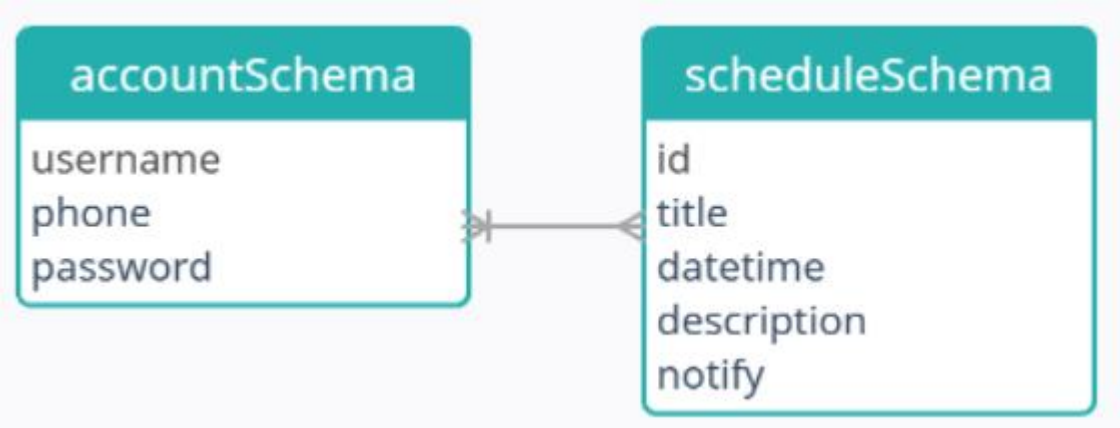
Hoạt động của API Backend (api/):

- Container API khởi động bằng cách "RUN" tệp server.js.
- server.js thiết lập server và quản lý các yêu cầu đến. Nó "uses" middlewares/ để xử lý các tác vụ chung (ví dụ: xác thực, ghi log) và config/ để tải cài đặt.
- Các yêu cầu HTTP được "mounts" (định tuyến) đến routes/.
- routes/ sẽ "calls" các chức năng tương ứng trong controllers/.
- controllers/ xử lý logic request ban đầu và "delegates" (ủy quyền) logic nghiệp vụ phức tạp hơn cho services/.
- services/ chứa logic nghiệp vụ chính và "accesses DB" (truy cập cơ sở dữ liệu) thông qua models/.
- models/ quản lý tương tác với cơ sở dữ liệu, sử dụng thông tin "connects via db.js" từ config/.
- Module auth/ xử lý xác thực người dùng và "interacts with" models/ để lưu trữ/truy xuất thông tin xác thực (credentials).

Hoạt động của Web Frontend (web/):

- Container Web khởi chạy, thường bao gồm một web server như NGINX.
- Dockerfile của Web chỉ dẫn NGINX "serve" (phục vụ) tệp app.js và các tài nguyên tĩnh khác.
- app.js là logic phía client, chịu trách nhiệm "serves" (tải hoặc render) các tệp HTML giao diện như dashboard.html và index.html để hiển thị cho người dùng.
- Người dùng tương tác với giao diện do Web Frontend cung cấp, và giao diện này sẽ gửi các yêu cầu API đến API Backend để thực hiện các tác vụ dữ liệu hoặc nghiệp vụ.

3.2. Thiết kế cơ sở dữ liệu



Hình 3.2 Mô hình ERD

3.3. Thiết kế API

3.3.1. Các endpoint chính

Bảng 3.1 Các endpoint chính

Method	Endpoint	Chức năng
POST	/auth/register	Đăng ký tài khoản
POST	/auth/login	Đăng nhập
POST	/accounts	Tạo tài khoản
GET	/accounts/:username	Lấy thông tin tài khoản
DELETE	/accounts/:username	Xóa tài khoản
PUT	/accounts/change-password	Đổi mật khẩu người dùng
POST	/accounts/:username/schedules	Thêm lịch trình mới
DELETE	/accounts/:username/schedules/:id	Xóa lịch trình
PATCH	/accounts/:username/schedules/:id	Cập nhật lịch trình

3.3.2. Cấu trúc request/response

Bảng 3.2 Cấu trúc request/response

API	Request	Response

POST /auth/register	{
	"username": "string",
	"phone": "string",
	"password": "string"
	}
		201: User đăng ký thành công
		400: Lỗi đăng ký

POST /auth/login	{
	"username": "string",
	"password": "string"
	}
		Token
		401: Lỗi đăng nhập

POST /accounts	{
	"username": "string",
	"phone": "string",
	"password": "string"
	}
		201: Tạo tài khoản thành công
		400: Lỗi tạo tài khoản

GET /accounts/:username		200: Thông tin tài khoản 404: Không tìm thấy tài khoản
DELETE /accounts/:username		204: Xóa thành công 404: Không tìm thấy tài khoản
PUT /accounts/change-	{ "currentPassword":	200: Đổi mật khẩu thành công 400: Lỗi đổi mật khẩu

password	"string", "newPassword": "string" }	401: Chưa đăng nhập hoặc token không hợp lệ

POST /accounts/:username/ schedules	{
	"title": "string",
	"datetime": "YYYY-MM-DDThh:mm:ss.xxxx",
	"description": "string"
	}
		201: Thêm lịch trình thành công
		400: Lỗi thêm lịch trình

DELETE /accounts/:username/ schedules/:id		204: Xóa lịch trình thành công 404 Không tìm thấy lịch trình

PATCH /accounts/:username/ schedules/:id	{
	"title": "string",
	"datetime": "YYYY-MM-DDThh:mm:ss.xxxx",
	"description": "string"
	}
		200: Cập nhật thành công
		400: Lỗi cập nhật lịch trình

3.4. Thiết kế giao diện (UI/UX)

3.4.1. Giới thiệu về Figma

Figma là công cụ thiết kế giao diện người dùng (UI) và trải nghiệm người dùng (UX) dựa trên nền tảng web. Figma cho phép các thành viên trong nhóm làm việc trực tuyến, thiết kế, chỉnh sửa và phản hồi ý tưởng trực tiếp trên cùng một bản thiết kế mà không cần cài đặt phần mềm cục bộ.

Một số tính năng nổi bật của Figma:

- Hỗ trợ thiết kế từ khung wireframe cơ bản đến các prototype tương tác phức tạp.

- Cho phép cộng tác theo thời gian thực (giống Google Docs cho thiết kế).

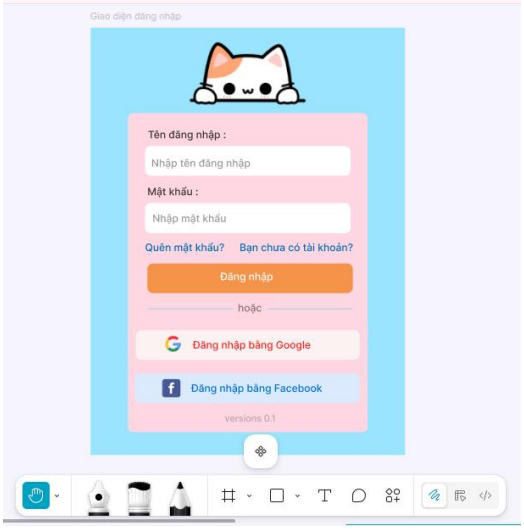
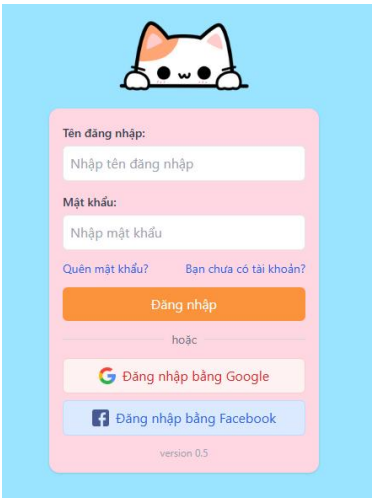
- Thư viện biểu tượng, thành phần (Component), hệ thống lưới (Grid System) giúp đảm bảo tính thống nhất giao diện.

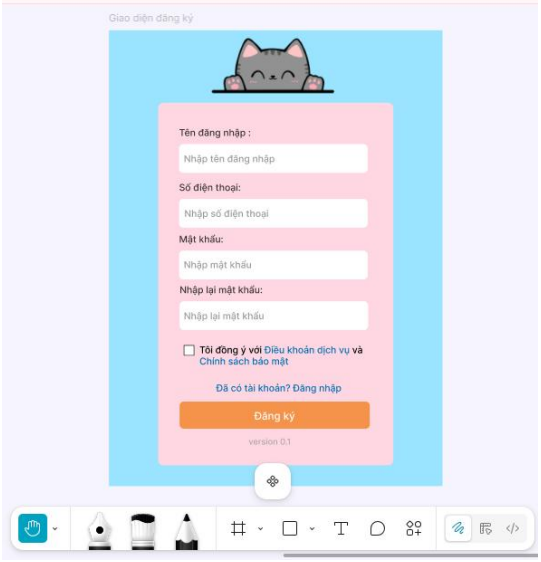
Figma là công cụ phổ biến trong phát triển phần mềm hiện nay nhờ sự tiện lợi, dễ sử dụng và khả năng đồng bộ làm việc nhóm tốt.

Hình 3.3 Công cụ thiết kế giao diện Figma

3.4.2. Giao diện của ứng dụng

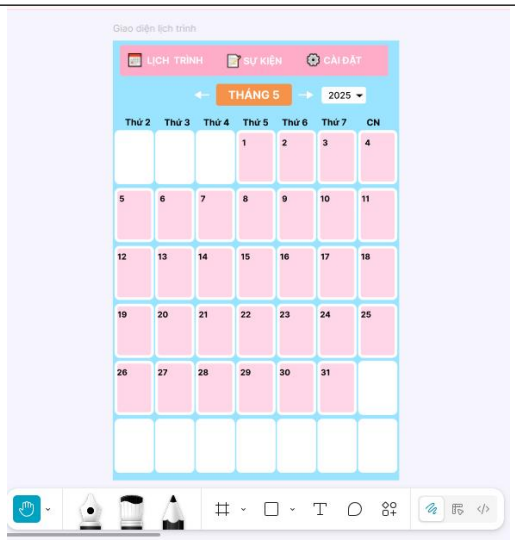
Bảng 3.3 Liên kết bản thiết kế Figma và giao diện thực tế

Bản thiết kế Figma	Giao diện thực tế
	
Hình 3.4 Giao diện đăng nhập trên Figma	*Hình 3.5 Giao diện đăng nhập thực tế*

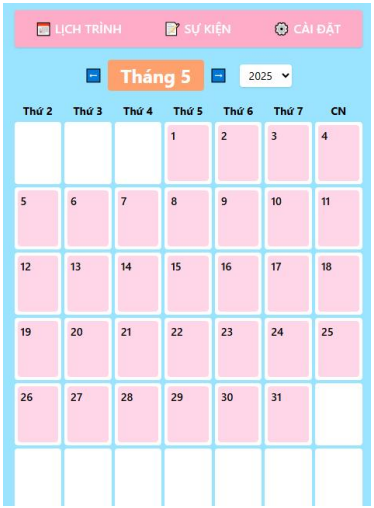


Hình 3.6 Giao diện đăng nhập trên Figma

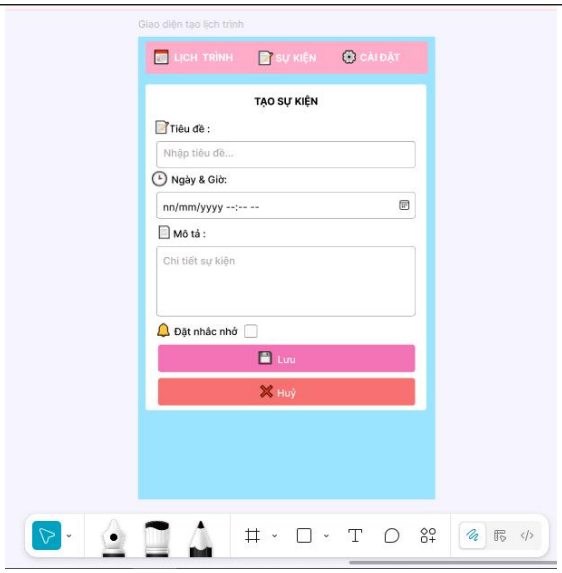
Hình 3.7 Giao diện đăng ký thực tế



Hình 3.8 Giao diện lịch trình trên Figma

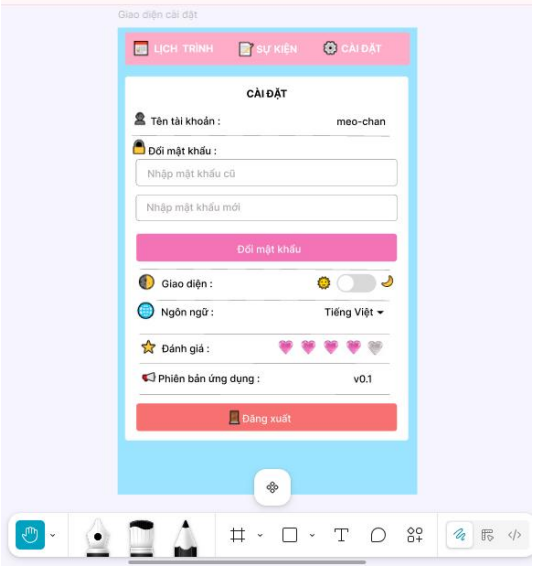


Hình 3.9 Giao diện lịch trình thực tế

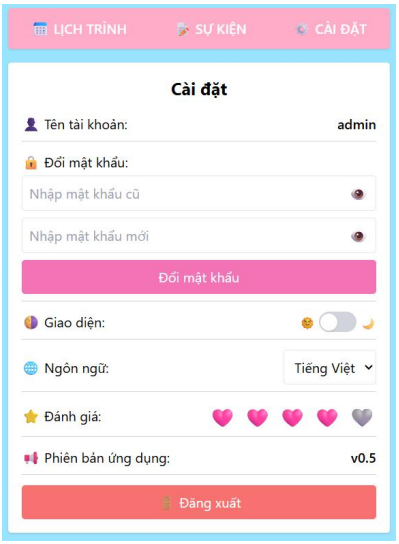


Hình 3.10 Giao diện tạo lịch trình trên Figma

Hình 3.11 Giao diện tạo lịch trình thực tế



Hình 3.12 Hình ảnh giao diện cài đặt trên Figma



Hình 3.13 Giao diện cài đặt thực tế

CHƯƠNG 4. TRIỂN KHAI VÀ CÔNG NGHỆ SỬ DỤNG

4.1. Các công nghệ sử dụng

Bảng 4.1 Danh sách các công nghệ sử dụng

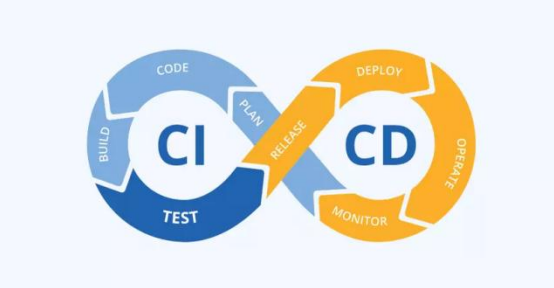
Thành phần	Công nghệ	Chức năng
Backend	Node.js	Môi trường chạy JavaScript phía server.
	express	Framework chính để tạo API RESTful nhanh gọn và linh hoạt.
	dotenv	Thư viện giúp tải các biến môi trường từ file .env vào process.env trong Node.js.
	cors	Middleware trong Node.js cho phép xử lý CORS (Cross-Origin Resource Sharing) cho frontend gọi API
	body-parser	Middleware trong Node.js dùng với Express để phân tích dữ liệu body của request, giúp dễ dàng truy cập nội dung gửi từ client.
	jsonwebtoken	Còn gọi là JWT, một thư viện trong Node.js dùng để tạo và xác thực token, hash mật khẩu người dùng trước khi lưu vào CSDL, đảm bảo bảo mật.
	swagger-ui-express	Tích hợp Swagger UI để mô tả và thử nghiệm API thông qua giao diện web.
	swagger-jsdoc	Công cụ giúp tạo file Swagger spec (OpenAPI) từ các comment trong mã nguồn JavaScript.

Fontend	HTML/CSS/JavaScript	Giao diện nhẹ, đơn giản, dễ tùy biến.
Cơ sở dữ liệu	MongoDB	Hệ quản trị cơ sở dữ liệu NoSQL.
	Mongoose	Thư viện ODM (Object Data Modeling) cho Node.js dùng để giao tiếp với MongoDB, hỗ trợ schema và mô hình hóa dữ liệu.
Kiểm thử	Jest	Framework kiểm thử phổ biến được phát triển bởi Meta, chủ yếu dùng để kiểm thử JavaScript và Node.js.
	Supertest	Thư viện trong Node.js dùng để kiểm thử các API HTTP (RESTful API) của ứng dụng Express hoặc các ứng dụng web khác một cách dễ dàng và chính xác.
	Postman	Công cụ hỗ trợ phát triển và kiểm thử API rất phổ biến hiện nay. Nó giúp bạn dễ dàng gửi yêu cầu HTTP đến máy chủ và xem phản hồi mà API trả về.

4.2. Quy trình CI/CD với GitHub Action

CI (Continuous Integration): Là một quá trình tự động hóa trong phát triển phần mềm, cho phép các thành viên trong nhóm kiểm tra và hợp nhất mã nguồn của họ vào kho chung một cách tự động và định kỳ. Mỗi lần được hợp nhất thì được xây dựng một cách tự động để phát hiện lỗi phát sinh một cách nhanh nhất có thể.

CD (Continuos Deployment): Là quá trình triển khai tự động sau khi quá trình CI được diễn ra thành công. Quá trình CD sẽ được kích hoạt để triển khai ứng dụng của chúng ta vào các môi trường của dự án. Các gói phần mềm đã build thành công được triển khai bằng cách sử dụng các công cụ tự động hoặc được triển khai thủ công bằng tay. Điều này đảm bảo rằng phần mềm được cài đặt đúng cách và hoạt động tốt trên các môi trường.



Hình 4.1 Quy trình CI/CD

GitHub Actions là một nền tảng miễn phí do GitHub cung cấp để giúp chúng ta tự động hoá quá trình CI/CD, cho phép người dùng định nghĩa các workflow tự động hoá các hoạt động trong phát triển phần mềm. Mỗi workflow trong GitHub Actions là một tập hợp các hành động (actions) được định nghĩa trong file YAML. Các actions này có thể là các lệnh cụ thể như: build, test, triển khai ứng dụng.

4.3. Cấu hình Docker và quy trình triển khai ứng dụng

4.3.1. Cấu hình Docker

Bảng 4.2 Cấu hình docker-compose.yml

Cấu hình CSDL	Cấu hình Backend	Cấu hình Fontend
mongo: image: mongo container_name: mongo ports: - '27017:27017' volumes: - mongo-data:/data/db networks: - app-network	api: build: ./api container_name: api ports: - '5000:5000' volumes: - ./api:/app - /app/node_modules environment: - MONGO_URI=mongodb://mongo:27017/schedule-db - JWT_SECRET=secret_key_2CATs depends_on: - mongo networks: - app-network	web: build: ./web container_name: web ports: - '8080:80' networks: - app-network

4.3.2. Quy trình triển khai

- Khởi động Docker Desktop.

- Thực thi lệnh **docker-compose up --build** trong Terminal của Visual Studio Code.

- Docker khởi tạo 3 service:

    + CSDL: MongoDB chạy trên cổng http://localhost:27017/

    + Fontend: Web chạy trên cổng http://localhost:8080/

    + Backend: Server chạy trên cổng http://localhost:5000/

CHƯƠNG 5. QUẢN LÝ DỰ ÁN

5.1. Giới thiệu về Jira

Jira là một phần mềm quản lý dự án mạnh mẽ và linh hoạt được phát triển bởi Atlassian – một công ty nổi tiếng chuyên cung cấp các công cụ hỗ trợ phát triển phần mềm. Jira được sử dụng phổ biến trong các nhóm phát triển phần mềm, CNTT, kiểm thử phần mềm, và quản lý dự án Agile như Scrum và Kanban.

Jira tổ chức công việc dưới dạng:

- **Project**: gồm nhiều Issue.

- **Issue:** một công việc cụ thể (bug, task, story...).

- **Board**: hiển thị trạng thái các issue theo cột (To Do, In Progress, Done...).

- **Sprint**: một chu kỳ làm việc ngắn theo Scrum (thường 1–2 tuần).

Lợi ích khi sử dụng Jira: Rõ ràng tiến độ và công việc từng người, hỗ trợ nhóm làm việc theo mô hình Agile dễ dàng hơn, giảm thời gian quản lý, tăng hiệu suất nhóm, dễ tích hợp với các công cụ CI/CD như GitHub Actions, Jenkins...

Hình 5.1 Phần mềm quản lý dự án Jira

5.2. Phân công nhiệm vụ

Bảng 5.1 Bảng phân công nhiệm vụ từng thành viên trong nhóm

STT	Nhiệm vụ	Thời gian thực hiện	Người thực hiện
1	Thiết kế giao diện đăng ký người dùng	Từ ngày 22/3/2025 đến ngày 29/3/2025	Thạch Thị Huệ Trinh

2	Phát triển chức năng đăng ký người dùng	Từ ngày 30/3/2025 đến ngày 06/4/2025	Lê Xuân Trường
3	Thiết kế giao diện đăng nhập	Từ ngày 22/3/2025 đến ngày 29/3/2025	Thạch Thị Huệ Trinh
4	Phát triển chức năng đăng nhập	Từ ngày 30/3/2025 đến ngày 06/4/2025	Lê Xuân Trường
5	Phát triển chức năng quên mật khẩu	Từ ngày 07/4/2025 đến ngày 14/4/2025	Thạch Thị Huệ Trinh
6	Thiết kế giao diện thời gian biểu	Từ ngày 07/4/2025 đến ngày 07/5/2025	Thạch Thị Huệ Trinh
7	Thiết kế giao diện tạo lịch trình	Từ ngày 07/4/2025 đến ngày 07/5/2025	Thạch Thị Huệ Trinh
8	Phát triển chức năng thêm, sửa, xóa lịch trình	Từ ngày 07/5/2025 đến ngày 07/6/2025	Thạch Thị Huệ Trinh
9	Thiết kế giao diện cài đặt	Từ ngày 07/6/2025 đến ngày 21/6/2025	Thạch Thị Huệ Trinh
10	Phát triển chức năng cập nhật mật khẩu	Từ ngày 21/6/2025 đến ngày 28/6/2025	Lê Xuân Trường
11	Phát triển chức năng thông báo nhắc nhở	Từ ngày 28/6/2025 đến ngày 05/7/2025	Lê Xuân Trường
12	Phát triển chức năng thay đổi giao diện (dark mode)	Từ ngày 28/6/2025 đến ngày 05/7/2025	Lê Xuân Trường
13	Kiểm thử hệ thống	Từ ngày 22/3/2025 đến ngày 15/7/2025	Lê Xuân Trường

5.3. Kế hoạch phát triển phần mềm

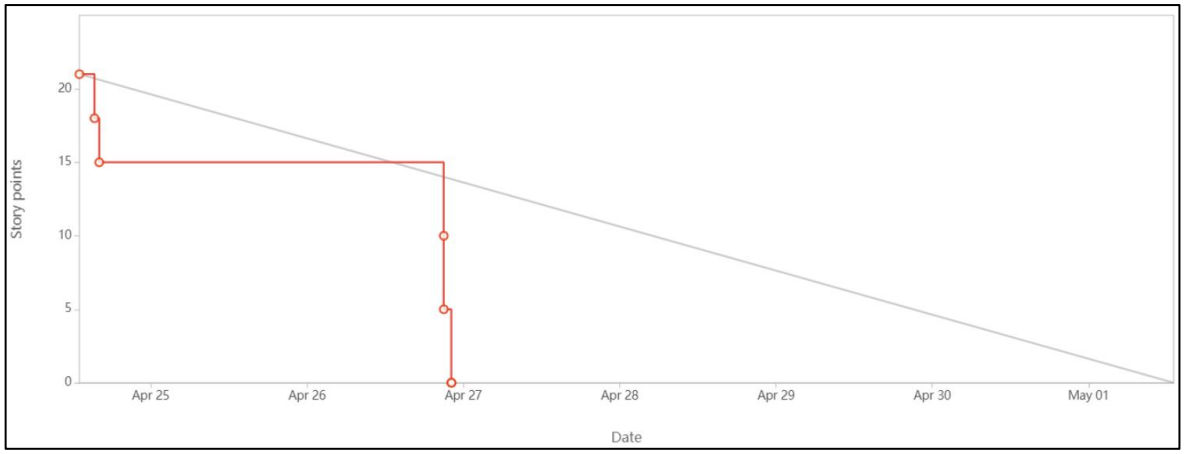
Quy trình phát triển phần mềm được thực hiện theo mô hình Scrum với 3 sprint, mỗi sprint kéo dài 1 tuần. Mỗi chức năng được chia thành các task cụ thể (gồm thiết kế giao diện và phát triển chức năng), đảm bảo phân tách rõ frontend và backend.

- **Sprint 1**: Đảm bảo người dùng có thể đăng ký và đăng nhập ứng dụng. Thời gian thực hiện: 24/4/2025 - 01/5/2025, gồm có 5 tasks với tổng cộng 21 points.

Completed work items — View in All work navigator

Key	Summary	Work type	Epic	Status	Assignee	Story points
SCRUM-1	Thiết kế giao diện đăng ký người dùng	Task		DONE	HT	3
SCRUM-2	Phát triển chức năng đăng ký người dùng	Task		DONE	TL	5
SCRUM-3	Thiết kế giao diện đăng nhập	Task		DONE	HT	3
SCRUM-4	Phát triển chức năng đăng nhập	Task		DONE	TL	5
SCRUM-5	Phát triển chức năng quên mật khẩu	Task		DONE	HT	5

Hình 5.2 Product Backlog của Sprint 1



Hình 5.3 Burndown Chart của Sprint 1

Từ Burndown Chart của Sprint 1, trong ngày đầu tiên (24/04) nhóm đã hoàn thành một phần nhỏ (từ 21 → 15 points). Sau đó, hai ngày tiếp theo không có task nào được hoàn thành trong khoảng này. Đến ngày 27/04, có một đợt hoàn thành lớn (từ 15 → 0 point) chỉ trong một ngày.

Điểm tích cực: Toàn bộ công việc đã hoàn thành đúng trước hạn sprint kết thúc. Sprint không bị kéo dài, backlog không tăng, nhóm vẫn kiểm soát tốt.

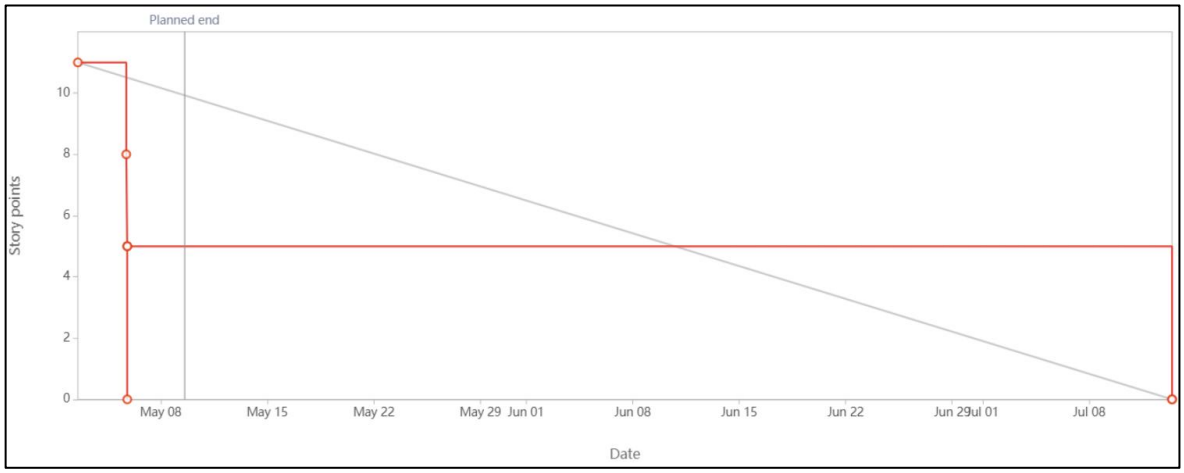
Điểm hạn chế: Tiến độ không đều, có sự trì hoãn giữa các ngày (không có công việc nào hoàn thành trong 2 ngày), sau đó dồn nhiều task vào một ngày cuối.

- **Sprint 2**: Cung cấp khả năng thêm, sửa, xóa lịch trình trong thời gian biểu. Thời gian thực hiện: 02/5/2025 - 09/5/2025, gồm có 3 tasks với tổng cộng 11 points.

Completed work items — View in All work navigator

Key	Summary	Work type	Epic	Status	Assignee	Story points
SCRUM-6	Thiết kế giao diện thời gian biểu	Task		DONE	HT	3
SCRUM-7	Thiết kế giao diện tạo lịch trình	Task		DONE	HT	3
SCRUM-8	Phát triển chức năng thêm, sửa, xóa lịch trình	Task		DONE	HT	5

Hình 5.4 Product Backlog của Sprint 2



Hình 5.5 Burndown Chart của Sprint 2

Từ Burndown Chart của Sprint 2, có một đợt giảm mạnh vào ngày đầu tiên (từ 11 → 5 points). Sau đó đường phẳng hoàn toàn suốt một thời gian dài do ảnh hưởng thời gian học và thi các môn khác cũng như gặp phải nhiều vấn đề về xử lí đồng bộ Backend và Fontend (gần 2 tháng). Đến khoảng ngày 08/07, toàn bộ phần còn lại được hoàn thành (từ 5 → 0 point).

Điểm tích cực: Công việc cuối cùng vẫn được hoàn thành và không bị bỏ dở. Nhóm đã ghi nhận lại đúng khối lượng công việc cần hoàn thành và đã rút kinh nghiệm cho các dự án khác trong tương lai.

Điểm hạn chế: Phân bổ thời gian thực hiện của Sprint sai nên đã kéo dài gấp gần 8 lần thời gian dự kiến.

- **Sprint 3**: Bổ sung thêm các chức năng nâng cao khác. Thời gian thực hiện: 13/7/2025 - 20/7/2025, gồm có 3 tasks với tổng cộng 11 points.

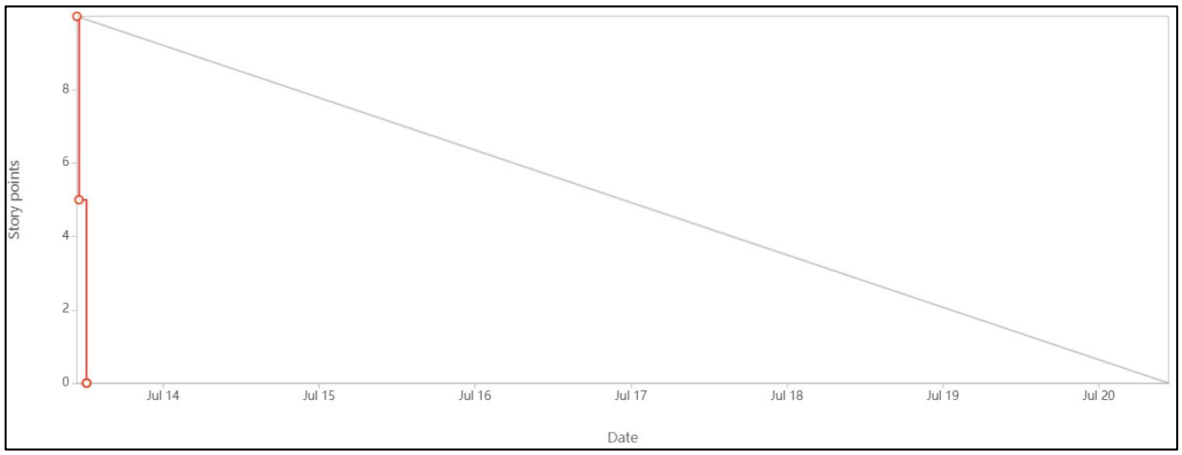
Completed work items — View in All work navigator

Key	Summary	Work type	Epic	Status	Assignee	Story points
SCRUM-10	Phát triển chức năng thông báo nhắc nhở	Task		DONE	TL	5
SCRUM-9	Phát triển chức năng cập nhật mật khẩu	Task		DONE	TL	5

Work items completed outside of sprint — View in All work navigator

Key	Summary	Work type	Epic	Status	Assignee	Story points
SCRUM-11	Phát triển chức năng thay đổi giao diện (dark mode)	Task		DONE	TL	5

Hình 5.6 Product Backlog của Sprint 3



Hình 5.7 Burndown Chart của Sprint 3

Từ Burndown Chart của Sprint 3, ngày đầu tiên toàn bộ story points giảm mạnh (từ 9 → 0 point), nghĩa là hoàn thành 100% ngay từ đầu. Không có thay đổi nào trong những ngày còn lại.

Điểm tích cực: Tất cả các task đã được hoàn thành rất sớm, Sprint không bị trễ hoặc tồn đọng backlog.

Điểm hạn chế: Sprint được lên kế hoạch chưa tối ưu.

CHƯƠNG 6. KIỂM THỬ

6.1. Chiến lược kiểm thử và công cụ sử dụng

Chiến lược kiểm thử thường bao gồm nhiều cấp độ để đảm bảo chất lượng phần mềm toàn diện:

Kiểm thử đơn vị (Unit Testing):

+ Mục tiêu: Kiểm thử từng hàm/mô-đun riêng lẻ.

+ Áp dụng: Các hàm logic, controller, service trong backend hoặc frontend.

+ Công cụ: Jest.

Kiểm thử tích hợp (Integration Testing):

+ Mục tiêu: Kiểm thử sự kết nối giữa các thành phần: controller → service → database.

+ Áp dụng: API endpoint, truy vấn DB qua mongoose.

+ Công cụ: Supertest + Jest (kiểm thử API Node.js).

Kiểm thử thủ công hoặc tự động bằng Postman:

+ Mục tiêu: Kiểm thử API từ phía người dùng hoặc QA.

+ Áp dụng: Gửi request GET/POST/PUT/DELETE, kiểm tra response trả về.

+ Công cụ: Postman.

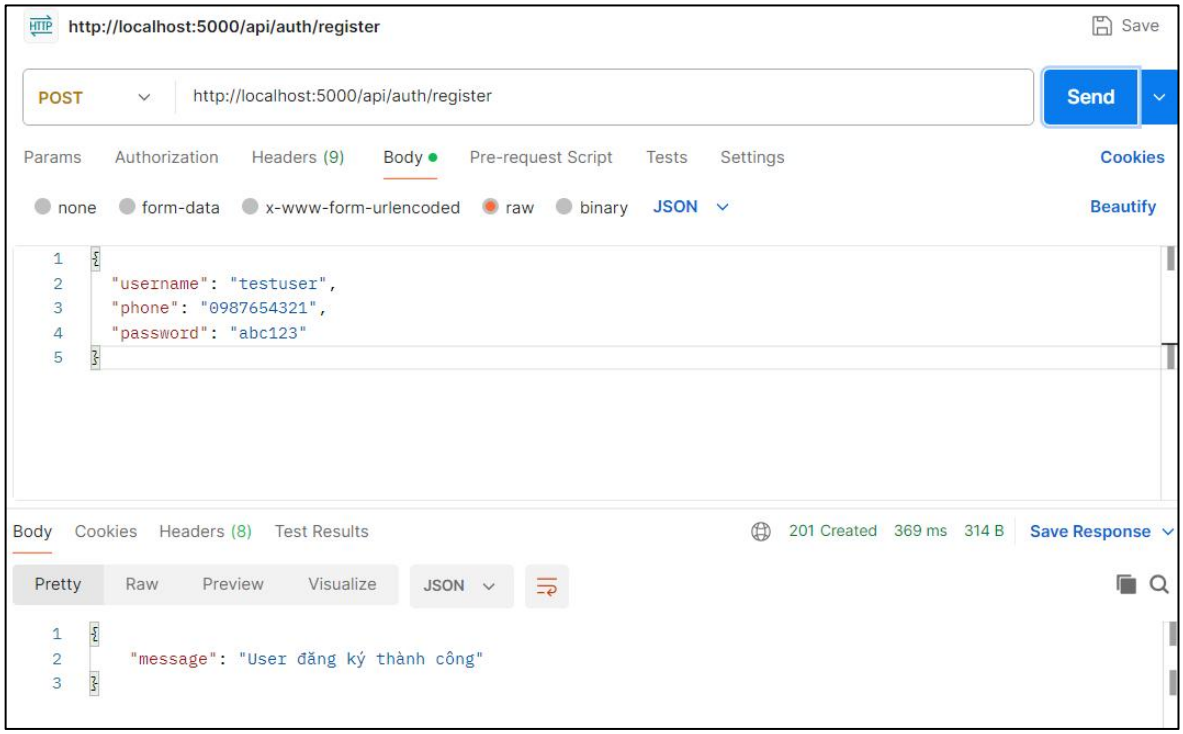
Kiểm thử hồi quy, CI/CD:

+ Mục tiêu: Đảm bảo thay đổi không phá vỡ hệ thống, kiểm thử liên tục khi code được đẩy lên GitHub.

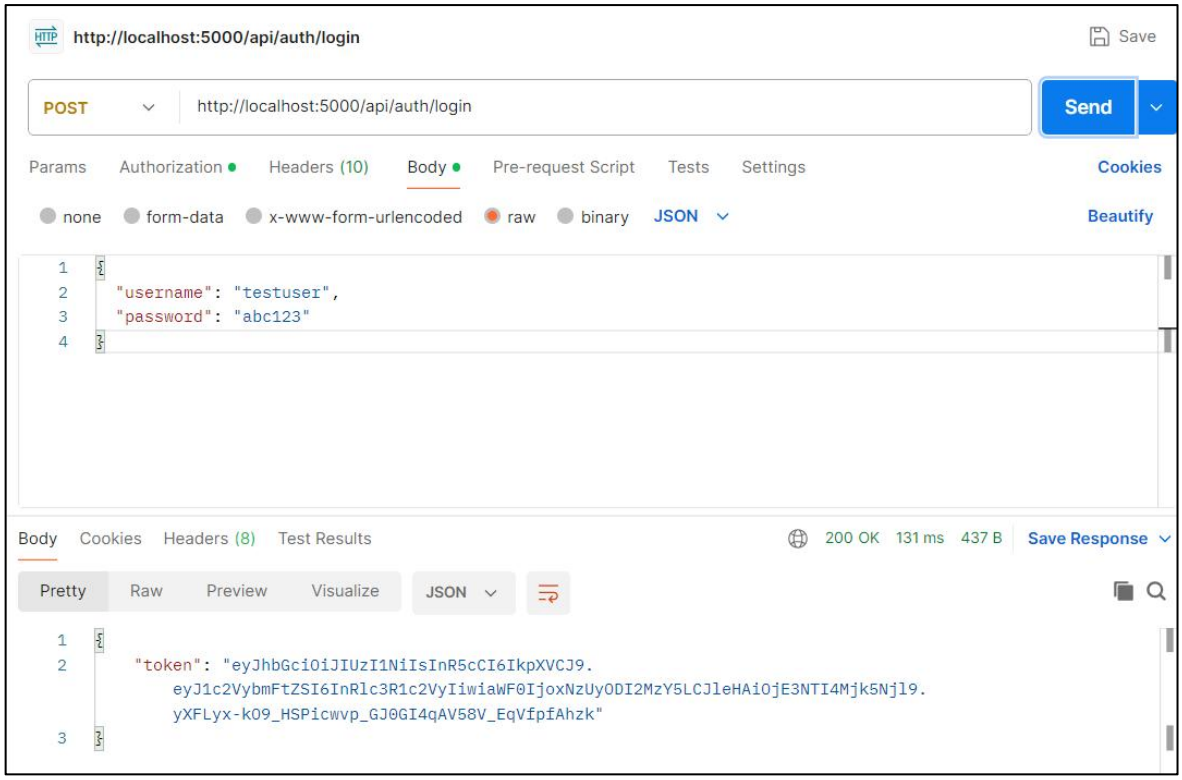
+ Thực thi tự động unit test, integration test, kiểm thử Newman khi git push.

+ Công cụ: GitHub Actions (tự động hóa test và deploy).

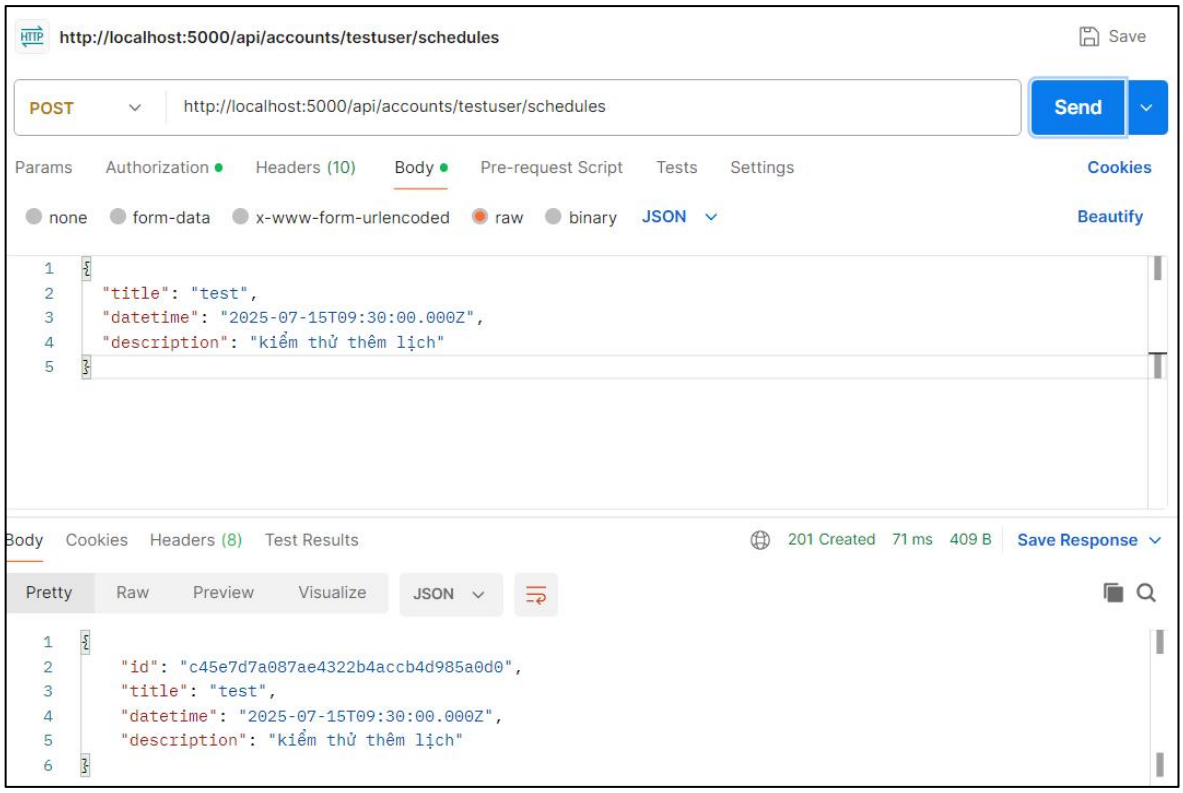
6.2. Kết quả kiểm thử API



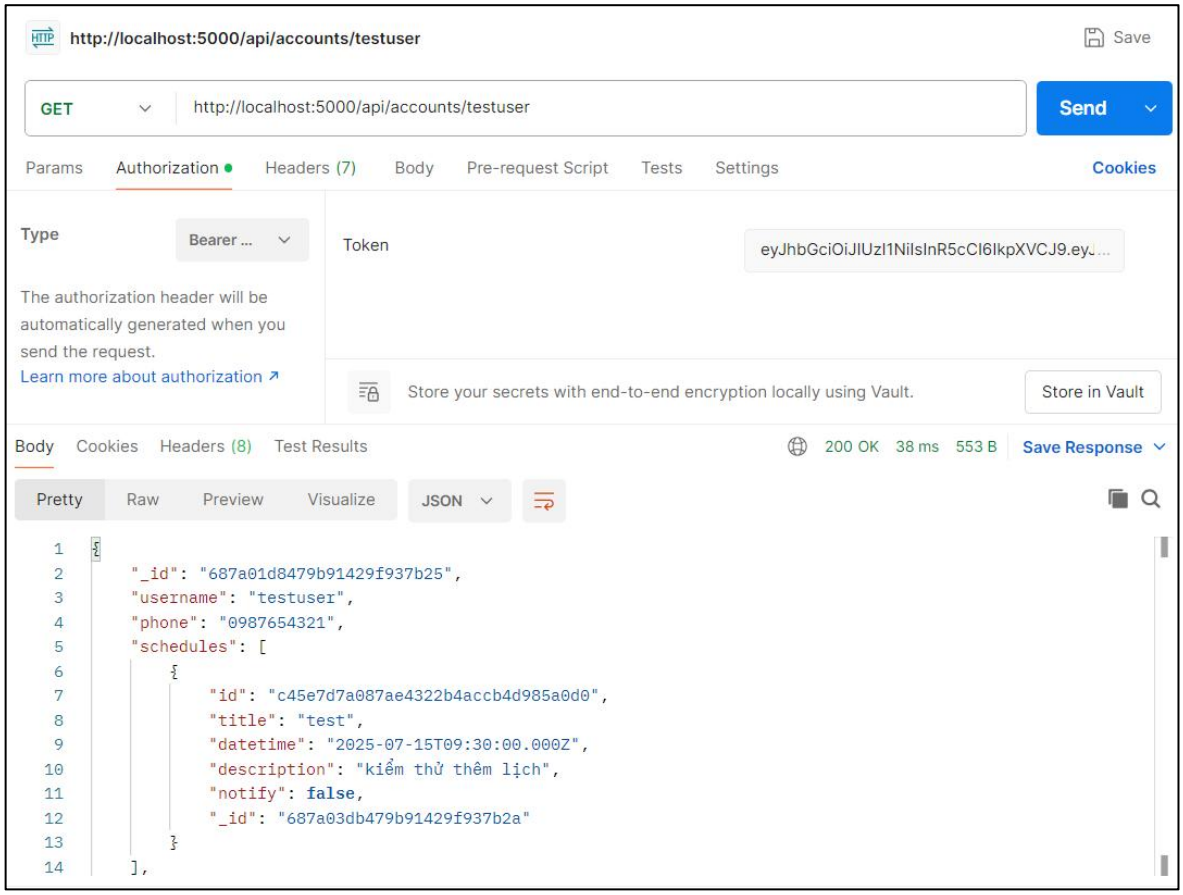
Hình 6.1 Kết quả kiểm thử đăng ký qua Postman



Hình 6.2 Kết quả kiểm thử đăng nhập qua Postman



Hình 6.3 Kết quả kiểm thử thêm lịch qua Postman



Hình 6.4 Kết quả kiểm thử lấy thông tin tài khoản qua Postman

CHƯƠNG 7. ĐÁNH GIÁ VÀ KẾT LUẬN

7.1. Những khó khăn gặp phải trong quá trình thực hiện.

- Thiếu kinh nghiệm thực tế về cách tổ chức kiến trúc dự án Web (Frontend, Backend, API, Database), dẫn đến mất nhiều thời gian tra cứu, thử nghiệm.

- Khó khăn khi kết nối API giữa frontend và backend do ban đầu chưa nắm rõ quy trình thiết kế API chuẩn RESTful.

- Thiếu kinh nghiệm với Figma nên thời gian thiết kế UI/UX ban đầu còn chậm, chưa tối ưu được khả năng tái sử dụng Component.

- Trong giai đoạn kiểm thử, một số lỗi liên quan đến hiển thị lịch, popup và dữ liệu không đồng bộ (thêm, sửa, xóa sự kiện) gây khó khăn trong việc xử lý và sửa lỗi.

7.2. Bài học rút ra và đề xuất cải thiện trong tương lai.

Bài học kinh nghiệm:

- Nắm được cách xây dựng hệ thống ứng dụng web hoàn chỉnh, bao gồm Frontend, Backend, API và cơ sở dữ liệu theo đúng quy trình phát triển phần mềm.

- Hiểu được quy trình thiết kế RESTful API, sử dụng hiệu quả các công cụ hỗ trợ như Postman, Swagger, Docker, GitHub.

- Nâng cao kỹ năng thiết kế giao diện UI/UX với Figma, biết cách áp dụng hệ thống lưới, component và nguyên tắc thiết kế trực quan.

Đề xuất cải thiện:

- Nâng cấp giao diện hiển thị lịch trình trực quan hơn, bổ sung chế độ xem theo tuần, tháng hoặc timeline.

- Thêm chức năng nhắc nhở tự động qua email hoặc thông báo.

- Tăng cường bảo mật hệ thống, quản lý phân quyền và phiên đăng nhập chặt chẽ hơn.

CHƯƠNG 8. PHỤ LỤC

8.1. Hướng dẫn cài đặt và chạy ứng dụng

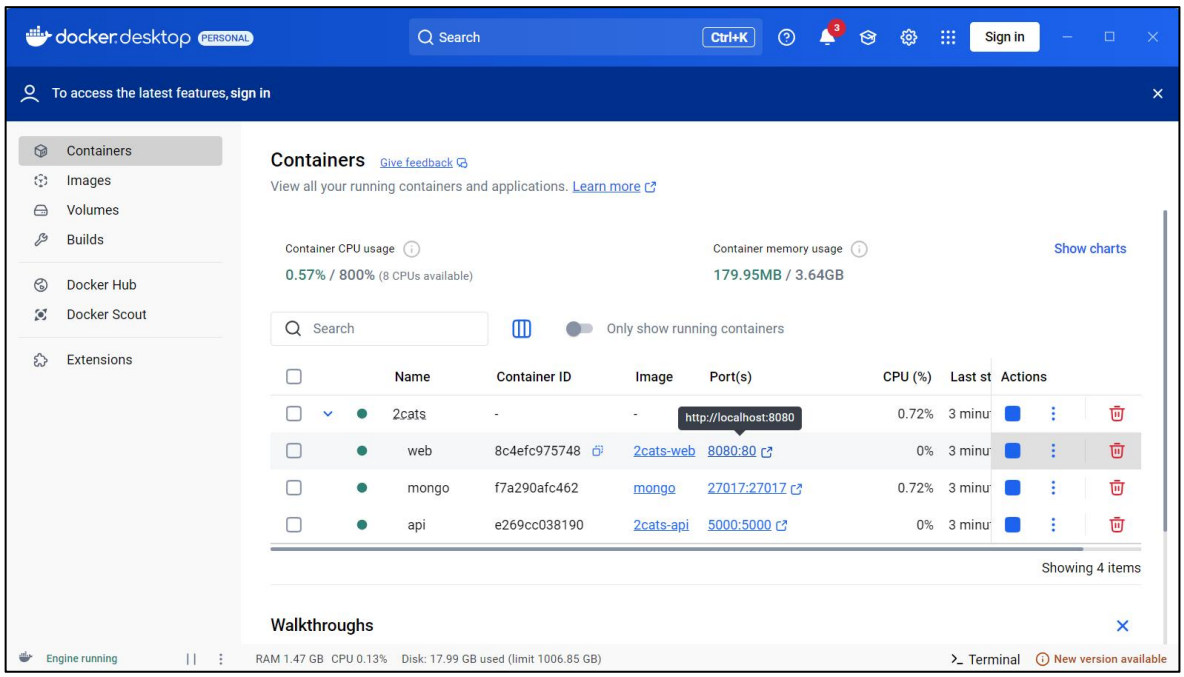
Bước 1: Clone dự án về bằng lệnh

clone https://github.com/lexuantruongtv/2CATs

trong Terminal của Visual Studio Code.

Bước 2: Khởi động Docker Desktop và thực thi lệnh **docker-compose up --build** trong Terminal của Visual Studio Code.

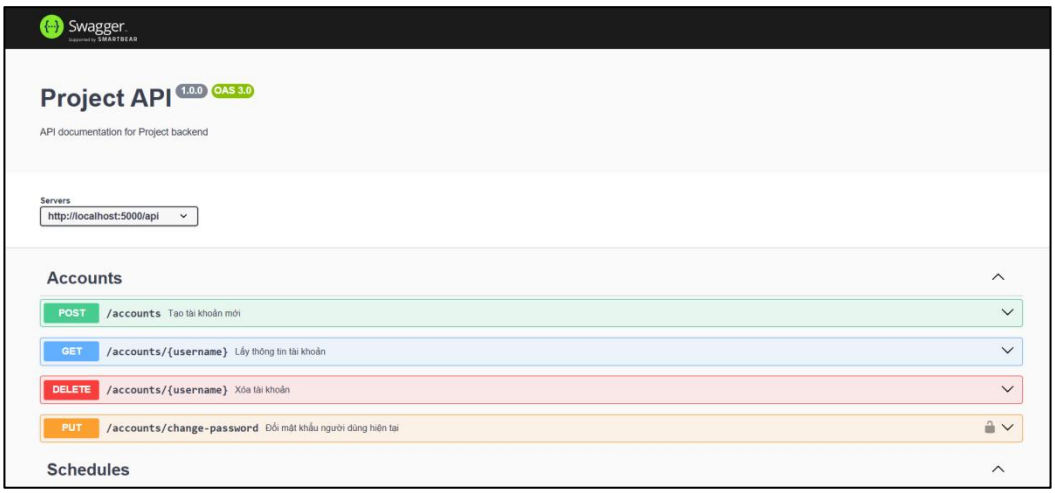
Bước 3: Sau khi khởi tạo thành công, tại cửa sổ Docker Desktop nhấp chọn **Port(s) 8080:80** hoặc sử dụng đường dẫn http://localhost:8080/ trên trình duyệt để mở ứng dụng.



Hình 8.1 Mở ứng dụng trên Docker Desktop

Bước 4: Thao tác đăng ký tài khoản và đăng nhập để vào trang của người dùng, tại đây có thể thêm, sửa, xóa lịch trình và cập nhật mật khẩu.

Bước 5: Có thể sử dụng đường dẫn http://localhost:5000/api-docs/ trên trình duyệt để mở giao diện Swagger của ứng dụng.



Hình 8.2 Giao diện Swagger của ứng dụng

8.2. Tài liệu Swagger của ứng dụng

8.2.1. Swagger tạo tài khoản mới

```
/**
 * @swagger
 * /accounts:
 *   post:
 *     summary: Tạo tài khoản mới
 *     tags: [Accounts]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required: [username, phone, password]
 *             properties:
 *               username:
 *                 type: string
 *               phone:
 *                 type: string
 *               password:
 *                 type: string
 *     responses:
 *       201:
```

```
 *         description: Tạo tài khoản thành công
 *       400:
 *         description: Lỗi tạo tài khoản
 */
router.post("/accounts", auth, controller.create);
```

8.2.2. Swagger lấy thông tin tài khoản

```
/**
 * @swagger
 * /accounts/{username}:
 *   get:
 *     summary: Lấy thông tin tài khoản
 *     tags: [Accounts]
 *     parameters:
 *       - in: path
 *         name: username
 *         required: true
 *         schema:
 *           type: string
 *         description: Username của tài khoản
 *     responses:
 *       200:
 *         description: Thông tin tài khoản
 *       404:
 *         description: Không tìm thấy tài khoản
 */
router.get("/accounts/:username", auth, controller.get);
```

8.2.3. Swagger xóa tài khoản

```
/**
 * @swagger
 * /accounts/{username}:
 *   delete:
 *     summary: Xóa tài khoản
 *     tags: [Accounts]
```

```
 *     parameters:
 *       - in: path
 *         name: username
 *         required: true
 *         schema:
 *           type: string
 *         description: Username của tài khoản
 *     responses:
 *       204:
 *         description: Xóa thành công
 *       404:
 *         description: Không tìm thấy tài khoản
 */
router.delete("/accounts/:username", auth, controller.delete);
```

8.2.4. Swagger thêm lịch trinh mới cho tài khoản

```
/**
 * @swagger
 * /accounts/{username}/schedules:
 *   post:
 *     summary: Thêm lịch trình mới cho tài khoản
 *     tags: [Schedules]
 *     parameters:
 *       - in: path
 *         name: username
 *         required: true
 *         schema:
 *           type: string
 *         description: Username của tài khoản
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
```

```
 *             required: [title, datetime]
 *             properties:
 *               title:
 *                 type: string
 *               datetime:
 *                 type: string
 *                 format: date-time
 *               description:
 *                 type: string
 *     responses:
 *       201:
 *         description: Thêm lịch trình thành công
 *       400:
 *         description: Lỗi thêm lịch trình
 */
router.post("/accounts/:username/schedules", auth, controller.addSchedule);
```

8.2.5. Swagger xóa lịch trinh của tài khoản

```
/**
 * @swagger
 * /accounts/{username}/schedules/{id}:
 *   delete:
 *     summary: Xóa lịch trình của tài khoản
 *     tags: [Schedules]
 *     parameters:
 *       - in: path
 *         name: username
 *         required: true
 *         schema:
 *           type: string
 *         description: Username của tài khoản
 *       - in: path
 *         name: id
 *         required: true
 *         schema:
```

```
 *           type: string
 *         description: ID lịch trình
 *     responses:
 *       204:
 *         description: Xóa lịch trình thành công
 *       404:
 *         description: Không tìm thấy lịch trình
 */
router.delete("/accounts/:username/schedules/:id", auth, controller.deleteSchedule);
```

8.2.6. Swagger cập nhật lịch trinh của tài khoản

```
/**
 * @swagger
 * /accounts/{username}/schedules/{id}:
 *   patch:
 *     summary: Cập nhật lịch trình của tài khoản
 *     tags: [Schedules]
 *     parameters:
 *       - in: path
 *         name: username
 *         required: true
 *         schema:
 *           type: string
 *         description: Username của tài khoản
 *       - in: path
 *         name: id
 *         required: true
 *         schema:
 *           type: string
 *         description: ID lịch trình
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
```

```
 *             type: object
 *             properties:
 *               title:
 *                 type: string
 *               datetime:
 *                 type: string
 *                 format: date-time
 *               description:
 *                 type: string
 *     responses:
 *       200:
 *         description: Cập nhật thành công
 *       400:
 *         description: Lỗi cập nhật lịch trình
 */
router.patch("/accounts/:username/schedules/:id", auth, controller.updateSchedule);
```

8.2.7. Swager đổi mật khẩu người dùng hiện tại

```
/**
 * @swagger
 * /accounts/change-password:
 *   put:
 *     summary: Đổi mật khẩu người dùng hiện tại
 *     tags: [Accounts]
 *     security:
 *       - bearerAuth: []
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               currentPassword:
 *                 type: string
```

```
 *               newPassword:
 *                 type: string
 *     responses:
 *       200:
 *         description: Đổi mật khẩu thành công
 *       400:
 *         description: Lỗi đổi mật khẩu
 *       401:
 *         description: Chưa đăng nhập hoặc token không hợp lệ
 */
router.put("/accounts/change-password", auth, controller.changePassword);
```