

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



**BÁO CÁO KẾT THÚC MÔN HỌC  
CÔNG NGHỆ PHẦN MỀM  
(MSHP: 220055)**

**TÊN ĐỀ TÀI  
XÂY DỰNG WEBSITE QUẢN LÍ CHI TIÊU**

**Sinh viên thực hiện:**

110122008	Phạm Hoàng Kha	DA22TTA
110122221	Nguyễn Thanh Hiếu	DA22TTA
110122041	Nguyễn Trí Cường	DA22TTA

**Giáo viên hướng dẫn: TS. Nguyễn Bảo Ân**

**Trà Vinh, tháng 7 năm 2025**

## **NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

*Trà Vinh, ngày ..... tháng ..... năm .....*

Giáo viên hướng dẫn

(Ký tên và ghi rõ họ tên)

## **NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG**

*Trà Vinh, ngày ..... tháng ..... năm .....*  
**Thành viên hội đồng**  
(Ký tên và ghi rõ họ tên)

## LỜI CẢM ƠN

Nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến Thầy Nguyễn Bảo Ân – giảng viên đã tận tình hướng dẫn, hỗ trợ và định hướng cho nhóm trong suốt quá trình thực hiện đề tài "**Xây dựng website quản lý chi tiêu**", thuộc môn *Công nghệ phần mềm*.

Nhờ sự chỉ dẫn tận tâm và những góp ý chuyên môn quý báu của Thầy, nhóm đã có thể vượt qua nhiều khó khăn, hoàn thiện sản phẩm đúng tiến độ và đạt được nhiều trải nghiệm thực tế bổ ích.

Nhóm cũng xin chân thành cảm ơn Trường Đại học Trà Vinh – Trường Kỹ thuật và Công nghệ, Khoa Công nghệ Thông tin đã tạo điều kiện học tập thuận lợi, cung cấp cơ sở vật chất và môi trường học tập năng động để chúng em có thể phát triển kỹ năng chuyên môn và hoàn thành tốt đề tài này.

Chúng em cũng xin gửi lời cảm ơn sâu sắc đến gia đình, những người luôn âm thầm ủng hộ, động viên và tiếp thêm tinh thần cho chúng em trong suốt quá trình học tập và thực hiện đề tài.

Bên cạnh đó, nhóm cũng trân trọng cảm ơn các bạn bè, người thân đã luôn sẵn sàng hỗ trợ, chia sẻ kiến thức và đóng góp ý kiến giúp nhóm hoàn thiện sản phẩm một cách hiệu quả nhất.

Một lần nữa, xin chân thành cảm ơn tất cả!

Kí tên

Sinh viên 1

Sinh viên 2

Sinh viên 3

**Nguyễn Thanh Hiếu**

**Phạm Hoàng Kha**

**Nguyễn Trí Cường**

# MỤC LỤC

Danh mục	Trang
LỜI CẢM ƠN-----	4
MỤC LỤC-----	5
CHƯƠNG 1 : TỔNG QUAN -----	12
1.1    Đặt vấn đề -----	12
1.2    Cơ sở hình thành đề tài -----	13
1.3    Mục tiêu đề tài -----	13
1.4    Phạm vi nghiên cứu-----	14
1.4.1    Về chức năng hệ thống-----	15
1.4.2    Về công nghệ áp dụng-----	15
1.4.3    Về đối tượng sử dụng-----	16
1.4.4    Về thời gian thực hiện-----	16
1.4.5    Về tài nguyên thực hiện-----	16
1.5    Đối tượng nghiên cứu -----	17
1.5.1    Người dùng hệ thống -----	17
1.5.2    Hệ thống phần mềm -----	17
1.5.3    Công nghệ và công cụ -----	17
1.6    Phương pháp nghiên cứu-----	17
1.6.1    Phương pháp khảo sát và phân tích yêu cầu -----	17
1.6.2    Phương pháp phát triển linh hoạt Agile (Scrum)-----	18
1.6.3    Phương pháp thiết kế nguyên mẫu (Prototyping) -----	18
1.6.4    Phương pháp kiểm thử phần mềm (Testing)-----	19
1.6.5    Phương pháp triển khai và tích hợp liên tục (CI/CD) -----	19
1.6.6    Phương pháp đánh giá và cải tiến -----	19
CHƯƠNG 2 : CƠ SỞ LÍ THUYẾT -----	20
2.1    Các khái niệm cơ bản về công nghệ phần mềm -----	20
2.1.1    Công nghệ phần mềm là gì?-----	20
2.1.2    Phần mềm là gì?-----	21
2.2    Các mô hình phát triển phần mềm-----	21
2.3    Tổng quan về Agile -----	24
2.3.1    Agile là gì?-----	24
2.3.2    Đặc trưng của Agile -----	25
2.3.3    Các nguyên tắc của Agile -----	26
2.3.4    Ưu và nhược điểm của Agile -----	27
2.3.4.1    Ưu điểm của Agile -----	27
2.3.4.2    Nhược điểm của Agile -----	28
2.3.5    Các phương pháp Agile phổ biến-----	28

2.3.6	Phương pháp Scrum-----	29
2.4	Mô hình kiến trúc Microservice -----	31
2.4.1	Microservices là gì? -----	31
2.4.2	Các đặc trưng của mô hình Microservice -----	32
2.4.3	Các ưu và nhược điểm của Microservice-----	33
2.4.3.1	Ưu điểm của Microservice architecture-----	33
2.4.3.2	Nhược điểm của Microservice-----	33
2.4.4	Thiết kế phần mềm theo kiến trúc Microservice -----	34
<b>CHƯƠNG 3</b>	<b>: PHÂN TÍCH YÊU CẦU -----</b>	<b>37</b>
3.1	Xác định nhu cầu-----	37
3.2	Các chức năng của hệ thống-----	37
3.2.1	Các chức năng chính của hệ thống-----	37
3.2.2	Các yêu cầu phi chức năng-----	38
3.3	Quản lý dự án-----	39
3.3.1	Kế hoạch để phân tích và thiết kế giao diện-----	39
3.3.2	Kế hoạch xây dựng API-----	41
3.3.3	Lập kế hoạch phát triển giao diện người dùng-----	43
3.3.4	Lập kế hoạch "Phân tích & quản lý chi tiêu"-----	45
3.3.5	Lập kế hoạch "Kiểm thử, CI/CD và Demo"-----	46
3.3.6	Bảng phân công công việc-----	47
<b>CHƯƠNG 4</b>	<b>: THIẾT KẾ HỆ THỐNG-----</b>	<b>51</b>
4.1	Kiến trúc tổng thể của hệ thống-----	51
4.2	Thiết kế cơ sở dữ liệu cho hệ thống-----	54
4.2.1	Sơ đồ Use case-----	54
4.2.2	Mô hình quan hệ dữ liệu của hệ thống-----	56
4.3	Thiết kế API cho hệ thống-----	57
4.3.1	Xác thực và phân quyền (auth):-----	57
4.3.2	Quản lý thông tin người dùng (Users)-----	59
4.3.3	Quản lý danh mục thu chi (Categories)-----	63
4.3.4	Quản lý tài khoản ngân hàng (Accounts)-----	67
4.3.5	Quản lý giao dịch thu/chi-----	70
4.3.6	Quản lý mục tiêu tiết kiệm-----	74
4.3.7	Quản lý thông kê và báo cáo-----	80
4.3.8	Trợ lí AI thông minh -----	84
4.4	Thiết kế giao diện (UI/UX)-----	88
4.4.1	Giao diện Đăng ký/ Đăng nhập-----	88
4.4.2	Giao diện trang chủ-----	92
4.4.3	Giao diện danh mục -----	95
4.4.4	Giao diện thông tin tài khoản -----	97

4.4.5	Giao diện trang mục tiêu-----	99
4.4.6	Giao diện giao dịch -----	101
4.4.7	Giao diện trang Thông kê-----	103
4.4.8	Giao diện trang cá nhân-----	107
4.4.9	Giao diện chat bot-----	108
<b>CHƯƠNG 5</b>	<b>: TRIỀN KHAI VÀ CÔNG NGHỆ SỬ DỤNG -----</b>	<b>110</b>
5.1	Các công nghệ sử dụng-----	110
5.1.1	Nhóm Frontend (ReactJS)-----	110
5.1.2	Nhóm Backend (Node.js/Express) -----	112
5.1.3	Nhóm Cơ sở dữ liệu (MongoDB) -----	113
5.1.4	Nhóm Trí Tuệ Nhân Tạo (Gemini API)-----	116
5.2	CI/ CD với Github Actions -----	118
5.2.1	CI/ CD là gì?-----	118
5.2.2	Giới thiệu về Github Actions-----	119
5.2.3	Quy trình CI/ CD vào hệ thống với Github Actions-----	121
5.3	Cấu hình Docker và quy trình triển khai ứng dụng -----	125
5.3.1	Cấu hình Docker -----	125
5.3.2	Quy trình triển khai -----	125
<b>CHƯƠNG 6</b>	<b>: KIỂM THỬ -----</b>	<b>127</b>
6.1	Chiến lược kiểm thử -----	127
6.2	Kết quả kiểm thử bằng Postman -----	127
6.3	Kiểm thử bằng unittest -----	131
<b>CHƯƠNG 7</b>	<b>: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN-----</b>	<b>134</b>
7.1	Kết luận-----	134
7.2	Hạn chế -----	134
7.3	Hướng phát triển -----	135
<b>TÀI LIỆU THAM KHẢO -----</b>	<b>136</b>	
<b>PHỤ LỤC-----</b>	<b>137</b>	

## DANH MỤC HÌNH ẢNH

Danh mục	Trang
Hình 2.1 Tìm hiểu về Công nghệ phần mềm .....	20
Hình 2.2 Mô hình thác nước .....	22
Hình 2.3 Mô hình chữ V.....	22
Hình 2.4 Mô hình Incremental Model.....	23
Hình 2.5 Mô hình cách thức hoạt động của Agile .....	23
Hình 2.6 Mô hình DevOps .....	24
Hình 2.7 Tìm hiểu về Agile .....	24
Hình 2.7 Các phương pháp phổ biến trong Agile .....	28
Hình 2.8 Hình ảnh về SRUM .....	30
Hình 2.9 Mô hình kiến trúc Microservice .....	31
Hình 2.10 Hình ảnh về 1 data riêng biệt.....	34
Hình 3.1 Các Story và task của Sprint 1.....	41
Hình 3.2 Biểu đồ Burndown của Sprint 1.....	41
Hình 3.3 Các Story và task của Sprint 2.....	42
Hình 3.4 Burnchart của kế hoạch xây dựng API .....	43
Hình 3.5 Các Story và task của Sprint 3.....	44
Hình 3.6 Burnchart của kế hoạch phát triển giao diện UI .....	44
Hình 3.7 Các Story, task, bug của Sprint 4.....	45
Hình 3.8 Burnchart của kế hoạch Phân tích chi tiêu .....	45
Hình 3.9 Các Story, task, bug của Sprint 5.....	46
Hình 3.10 Biểu đồ Burnchat của Kiểm thử CI CD .....	47
Hình 4.1 Sơ đồ kiến trúc hệ thống .....	51
Hình 4.2 Luồng thêm giao dịch mới và cập nhật mục tiêu .....	52
Hình 4.3 Luồng tương tác Goal Service .....	52
Hình 4.4 Luồng xem báo cáo tháng .....	53
Hình 4.5 Luồng tạo mục tiêu tiết kiệm .....	54
Hình 4.6 Sơ đồ Use case của hệ thống.....	55
Hình 4.7 Mô hình ERD của hệ thống.....	56
Hình 4.7 Luồng xác thực .....	59
Hình 4.8 Trang chào mừng khi được thiết kế trên Figma.....	89
Hình 4.9 Trang chào mừng khi đã triển khai lên website .....	89
Hình 4.10 Giao diện đăng nhập khi thiết kế trên Figma .....	90
Hình 4.11 Giao diện đăng nhập khi được triển khai thực tế .....	90
Hình 4.12 Giao diện đăng ký khi được thiết kế trên Figma .....	91
Hình 4.13 Giao diện đăng ký khi được triển khai thực tế .....	91
Hình 4.14 Giao diện khi đăng nhập thành công .....	92

<i>Hình 4.15 Giao diện trang chủ khi được thiết kế trên Figma .....</i>	93
<i>Hình 4.16 Giao diện trang chủ khi được triển khai lên React .....</i>	94
<i>Hình 4.17 Khi thực hiện thêm giao dịch ở trang chủ.....</i>	94
<i>Hình 4.18 Phần Chart của trang chủ.....</i>	95
<i>Hình 4.19 Phần Recent transaction của trang chủ .....</i>	95
<i>Hình 4.20 Giao diện thêm giao dịch.....</i>	96
<i>Hình 4.21 Cơ cấu thu chi của người dùng mẫu .....</i>	96
<i>Hình 4.22 Giao diện khi thiết kế trên Figma .....</i>	97
<i>Hình 4.23 Giao diện thêm nguồn tiền.....</i>	98
<i>Hình 4.24 Giao diện trang tài khoản khi thiết kế trên Figma .....</i>	99
<i>Hình 4.25 Giao diện trang tài khoản khi triển khai trên website .....</i>	99
<i>Hình 4.26 Giao diện thêm mục tiêu .....</i>	100
<i>Hình 4.27 Phần mục tiêu tài chính .....</i>	100
<i>Hình 4.28 Phần thêm tiền cho mục tiêu .....</i>	101
<i>Hình 4.29 Banner của trang giao dịch .....</i>	101
<i>Hình 4.30 Phần quản lý giao dịch của trang giao dịch.....</i>	102
<i>Hình 4.31 Phần khung lịch sử giao dịch và bộ lọc .....</i>	102
<i>Hình 4.32 Giao diện trang giao dịch khi thiết kế trên Figma .....</i>	103
<i>Hình 4.33 Giao diện trang giao dịch khi được triển khai thực tế .....</i>	103
<i>Hình 4.34 Giao diện trang thống kê .....</i>	104
<i>Hình 4.35 Phần tổng quan trong khung thống kê và phân tích .....</i>	104
<i>Hình 4.36 Phần cơ cấu trong khung thống kê và phân tích của chi tiêu .....</i>	105
<i>Hình 4.37 Phần tổng quan trong khung thống kê và phân tích của thu nhập.....</i>	105
<i>Hình 4.38 Giao diện trang thống kê và phân tích khi được thiết kế trên Figma .....</i>	106
<i>Hình 4.39 Giao diện trang thống kê khi được triển khai thực tế .....</i>	106
<i>Hình 4.40 Giao diện trang cá nhân .....</i>	107
<i>Hình 4.41 Giao diện trang cá nhân ghi lại log đăng nhập .....</i>	107
<i>Hình 4.42 Ánh đại diện của chat bot .....</i>	108
<i>Hình 4.43 Giao diện chat khi được sử dụng .....</i>	108
<i>Hình 5.1 Quy trình CI/ CD của một công ty mẫu .....</i>	118
<i>Hình 5.2 Cách Action một command .....</i>	120
<i>Hình 5.3 Các thành phần của Github Actions .....</i>	121
<i>Hình 5.4 Kiến trúc CI CD của hệ thống .....</i>	121
<i>Hình 5.5 Luồng CI .....</i>	123
<i>Hình 5.6 Đẩy Docker image lên GitHub Packages .....</i>	123
<i>Hình 5.7 Đẩy Docker image frontend lên GitHub Packages .....</i>	124
<i>Hình 5.8 Đẩy Docker image backend lên GitHub Packages .....</i>	124
<i>Hình 5.9 Luồng CD.....</i>	125
<i>Hình 6.1 Test đăng nhập bằng Postman .....</i>	127

<i>Hình 6.2 Test thay đổi mật khẩu của hệ thống bằng Postman.....</i>	128
<i>Hình 6.3 Xem giao dịch từ hệ thống bị lỗi token .....</i>	129
<i>Hình 6.4 Xem giao dịch từ hệ thống .....</i>	130
<i>Hình 6.5 Kiểm tra xóa giao dịch.....</i>	130
<i>Hình 6.6 Test Unittest cho backend .....</i>	132
<i>Hình 6.7 Test Unittest cho frontend .....</i>	132

## **DANH MỤC BẢNG BIỂU**

Danh mục	Trang
<i>Bảng 1 Công việc &amp; Phân công chi tiết cho phân tích và thiết kế giao diện .....</i>	<i>48</i>
<i>Bảng 2 Công việc &amp; Phân công chi tiết cho kế hoạch xây dựng API.....</i>	<i>49</i>
<i>Bảng 3.Công việc &amp; Phân tích chi tiêu cho phát triển giao diện người dùng.....</i>	<i>49</i>
<i>Bảng 4 Công việc &amp; phân công chi tiết cho phân tích và quản lí chi tiêu. ....</i>	<i>50</i>
<i>Bảng 5 Công việc và phân công chi tiết cho kế hoạch kiểm thử CI/ CD và Demo .....</i>	<i>50</i>
<i>Bảng 6 Các phương thức và Endpoint của xác thực và phân quyền .....</i>	<i>57</i>
<i>Bảng 7 Các phương thức và Endpoint của quản lí người dùng .....</i>	<i>59</i>
<i>Bảng 8 Các phương thức và endpoint của quản lí thu chi .....</i>	<i>63</i>
<i>Bảng 9 Các phương thức và endpoint của quản lí ngân hàng .....</i>	<i>67</i>
<i>Bảng 10 Các phương thức và endpoint của mục tiêu tiết kiệm .....</i>	<i>70</i>
<i>Bảng 12 Các phương thức và endpoint quản lí mục tiêu.....</i>	<i>75</i>
<i>Bảng 13 Các phương thức và endpoint của thống kê và báo cáo .....</i>	<i>80</i>
<i>Bảng 13 Các phương thức và endpoint của trợ lí AI.....</i>	<i>84</i>

# CHƯƠNG 1: TỔNG QUAN

## 1.1 Đặt vấn đề

Trong xã hội hiện đại, việc quản lý chi tiêu cá nhân đóng vai trò rất quan trọng đối với mỗi cá nhân, bất kể mức thu nhập cao hay thấp. Không ít người làm tướng rằng chỉ những người có thu nhập trung bình hoặc thấp mới cần theo dõi tài chính chặt chẽ, nhưng thực tế cho thấy ngay cả những người có thu nhập cao cũng có thể rơi vào tình trạng mất cân đối tài chính nếu thiếu thói quen và công cụ quản lý chi tiêu hiệu quả. Do đó, việc quản lý tài chính cá nhân không chỉ là nhu cầu, mà còn là một kỹ năng sống cần thiết trong thời đại ngày nay.

Tuy nhiên, đa số người dùng hiện nay vẫn đang sử dụng các phương pháp thủ công như ghi chép vào sổ tay hoặc bảng tính Excel để theo dõi thu – chi hàng ngày. Cách làm này tuy đơn giản nhưng còn nhiều hạn chế: khó truy xuất dữ liệu, thiếu trực quan, không thể tự động thống kê theo thời gian và không phù hợp với người dùng bận rộn hoặc thường xuyên sử dụng thiết bị di động. Trong khi đó, các ứng dụng quản lý chi tiêu trên thị trường lại phần lớn là sản phẩm quốc tế, giao diện bằng tiếng Anh, tích hợp nhiều tính năng nâng cao không cần thiết, gây khó khăn cho người dùng phổ thông. Một số ứng dụng còn yêu cầu trả phí nếu muốn sử dụng đầy đủ tính năng.

Từ những vấn đề thực tiễn đó, nhóm nhận thấy rằng việc xây dựng một website quản lý chi tiêu cá nhân đơn giản, thân thiện, miễn phí và hỗ trợ tiếng Việt sẽ là một giải pháp phù hợp, giúp người dùng dễ dàng ghi lại các khoản thu – chi, phân loại khoản chi theo danh mục, theo dõi thống kê tài chính qua biểu đồ trực quan và đưa ra nhận định về thói quen chi tiêu của bản thân. Giải pháp này không chỉ đáp ứng nhu cầu cơ bản mà còn có thể mở rộng thêm các tính năng nâng cao như gợi ý tiết kiệm, cảnh báo vượt hạn mức hoặc lập kế hoạch tài chính theo từng giai đoạn.

Bên cạnh ý nghĩa thực tiễn, đề tài còn giúp người học vận dụng kiến thức đã tiếp thu trong môn *Công nghệ phần mềm* vào một dự án cụ thể. Trong quá trình thực hiện, sinh viên sẽ trải qua đầy đủ các bước trong quy trình phát triển phần mềm như khảo sát, phân tích yêu cầu, thiết kế hệ thống, xây dựng giao diện, xử lý backend, thiết kế và quản lý cơ sở dữ liệu, kiểm thử và triển khai ứng dụng. Đồng thời, đây cũng là cơ hội để rèn luyện kỹ năng làm việc nhóm, tư duy hệ thống, khả năng giải quyết vấn đề và ứng dụng các công nghệ hiện đại vào sản phẩm thực tế.

Chính vì những lý do nêu trên, đề tài “Xây dựng website quản lý chi tiêu” được lựa chọn với kỳ vọng không chỉ mang lại giải pháp hữu ích cho người dùng, mà còn tạo môi trường học tập thực hành toàn diện cho sinh viên trong quá trình rèn luyện chuyên môn.

## 1.2 Cơ sở hình thành đề tài

Hiện nay, khi mức sống và thu nhập của người dân ngày càng tăng lên, nhu cầu về việc quản lý tài chính cá nhân cũng trở nên quan trọng và phổ biến hơn. Dù là sinh viên, nhân viên văn phòng, hay người đã có gia đình và thu nhập ổn định, việc nắm rõ dòng tiền – bao gồm các khoản thu nhập và chi tiêu hàng ngày – là điều cần thiết để xây dựng kế hoạch tài chính hiệu quả, tiết kiệm hợp lý và hướng tới các mục tiêu dài hạn.

Tuy nhiên, theo khảo sát thực tế và quan sát từ môi trường xung quanh, vẫn còn rất nhiều người chưa có thói quen ghi chép, thống kê chi tiêu. Một phần do thiếu công cụ phù hợp, phần khác do các ứng dụng hiện có chưa đáp ứng được nhu cầu sử dụng cơ bản hoặc không thân thiện với người dùng Việt Nam. Việc thiếu một nền tảng đơn giản, dễ sử dụng, không yêu cầu kiến thức công nghệ cao đã vô tình tạo ra khoảng trống trong thói quen quản lý tài chính cá nhân, đặc biệt là trong giới trẻ.

Mặt khác, trong chương trình học phần *Công nghệ phần mềm*, sinh viên được tiếp cận với quy trình phát triển phần mềm một cách bài bản, từ phân tích yêu cầu, thiết kế hệ thống, lập trình frontend/backend, đến kiểm thử và triển khai sản phẩm. Việc lựa chọn một đề tài mang tính thực tiễn cao, có thể áp dụng ngay vào đời sống sẽ giúp sinh viên không chỉ hiểu sâu lý thuyết mà còn rèn luyện tư duy hệ thống và kỹ năng thực hành phát triển phần mềm hoàn chỉnh.

Chính từ nhu cầu thực tế kết hợp với kiến thức chuyên môn đã học, đề tài “**Xây dựng website quản lý chi tiêu**” được lựa chọn nhằm mục tiêu tạo ra một công cụ hữu ích, hỗ trợ người dùng theo dõi tài chính cá nhân một cách dễ dàng, đồng thời là cơ hội thực hành xây dựng phần mềm từ đầu đến cuối một cách toàn diện.

## 1.3 Mục tiêu đề tài

Đề tài “Xây dựng website quản lý chi tiêu” hướng đến các mục tiêu sau:

+ Xây dựng một nền tảng website giúp người dùng ghi nhận, quản lý và thống kê các khoản thu – chi một cách trực quan, thuận tiện và khoa học. Người dùng có thể

dễ dàng theo dõi lịch sử giao dịch, phân loại chi tiêu theo nhóm/danh mục, và tạo các báo cáo giúp kiểm soát tài chính cá nhân hoặc nhóm.

+ Sử dụng Docker để đóng gói từng dịch vụ backend và frontend dưới dạng container, giúp quá trình deploy hệ thống trở nên dễ dàng, nhất quán và linh hoạt hơn giữa các môi trường (dev, test, production). Docker cũng giúp nhóm phát triển kiểm soát tốt hơn các phụ thuộc và cấu hình của hệ thống.

+ Cung cấp giao diện người dùng thân thiện, hiện đại, tối ưu trải nghiệm sử dụng trên nhiều thiết bị thông qua thiết kế UI/UX trên Figma, mang lại sự tiện lợi trong thao tác và dễ dàng tiếp cận cho nhiều đối tượng người dùng.

+ Ứng dụng kiến trúc microservice vào xây dựng backend hệ thống, chia nhỏ các chức năng (người dùng, giao dịch, danh mục, báo cáo...) thành các dịch vụ độc lập, giúp dễ dàng mở rộng, bảo trì, nâng cấp và đảm bảo tính ổn định cho toàn hệ thống.

+ Tối ưu hóa quy trình phát triển và triển khai phần mềm bằng CI/CD pipeline hiện đại (sử dụng GitHub Actions/GitLab CI/CD hoặc tương đương), đảm bảo việc kiểm thử, đóng gói và triển khai ứng dụng được tự động hóa, giảm thiểu lỗi và nâng cao chất lượng sản phẩm.

+ Sử dụng Jira để quản lý dự án, phân chia công việc và theo dõi tiến độ nhóm thực hiện. Từ đó phát triển kỹ năng teamwork, đảm bảo tiến độ và hiệu quả trong suốt quá trình thực hiện đề tài.

+ Đảm bảo các tiêu chí về bảo mật, quyền riêng tư và an toàn dữ liệu cá nhân khi người dùng sử dụng hệ thống, đồng thời đặt nền tảng cho việc mở rộng và phát triển các tính năng mới trong tương lai như kết nối với ngân hàng, tư vấn tài chính thông minh (AI), đa nền tảng.

Nhìn chung, đề tài hướng tới xây dựng một sản phẩm thực tiễn, ứng dụng công nghệ hiện đại để hỗ trợ quản lý chi tiêu thông minh, chuẩn kỹ năng phát triển phần mềm chuyên nghiệp, đồng thời tạo nền tảng cho các ý tưởng nâng cao trong lĩnh vực Fintech sau này.

## 1.4 Phạm vi nghiên cứu

Dự án “Xây dựng website quản lý chi tiêu cá nhân” sử dụng kiến trúc monolithic được triển khai trong một phạm vi rõ ràng, phù hợp với mục tiêu đề tài và năng lực của nhóm thực hiện. Phạm vi nghiên cứu được xác định cụ thể theo các khía cạnh sau:

### **1.4.1 Về chức năng hệ thống**

Dự án tập trung phát triển các chức năng cốt lõi, phục vụ nhu cầu quản lý tài chính cá nhân một cách hiệu quả và thông minh. Cụ thể bao gồm:

- + Quản lý thu/chi cá nhân: Cho phép người dùng thực hiện đầy đủ các thao tác thêm, sửa, xóa, xem (CRUD) các giao dịch thu – chi.
- + Phân loại giao dịch theo danh mục tùy chỉnh: Người dùng có thể tạo và quản lý các nhóm danh mục chi tiêu (như ăn uống, học tập, giải trí...) để tổ chức dữ liệu một cách hợp lý.
- + Xem báo cáo và thống kê chi tiêu: Cung cấp các bảng thống kê, biểu đồ trực quan theo các tiêu chí như thời gian (ngày, tháng, năm), danh mục, tỷ lệ thu – chi, giúp người dùng dễ dàng theo dõi tình hình tài chính.
- + Quản lý tài khoản người dùng: Bao gồm các chức năng như đăng ký, đăng nhập, xác thực, cập nhật thông tin cá nhân và đổi mật khẩu.

Các chức năng chưa nằm trong phạm vi thực hiện ở giai đoạn hiện tại gồm:

- + Tích hợp với ngân hàng số để tự động đồng bộ giao dịch.
- + Hỗ trợ thanh toán trực tuyến thông qua các cổng thanh toán như Momo, ZaloPay, VNPay...
- + Ứng dụng trí tuệ nhân tạo (AI) để phân tích thói quen chi tiêu, phát hiện các bất thường và đưa ra gợi ý điều chỉnh, dự báo xu hướng chi tiêu, giúp người dùng kiểm soát tài chính một cách khoa học và cá nhân hóa.

### **1.4.2 Về công nghệ áp dụng**

Đề tài áp dụng các công nghệ hiện đại, phổ biến, có khả năng mở rộng và cộng đồng hỗ trợ mạnh:

- + Frontend: Sử dụng ReactJS kết hợp với các thư viện hỗ trợ như React Router, Axios, Tailwind CSS, Chart.js để xây dựng giao diện hiện đại, trực quan và phản hồi nhanh.
- + Backend: Áp dụng kiến trúc monolithic với nền tảng chính là Node.js (NestJS). Mỗi chức năng (người dùng, giao dịch, báo cáo, AI gợi ý...) được triển khai như một dịch vụ độc lập, dễ dàng mở rộng và bảo trì.
- + Cơ sở dữ liệu: Sử dụng Mongo DB làm hệ quản trị cơ sở dữ liệu chính để lưu trữ giao dịch và thông tin người dùng.

- + Tích hợp AI hỗ trợ: Trong quá trình phát triển hệ thống, nhóm đã sử dụng Gemini AI (Google Generative AI) để xử lý các yêu cầu liên quan đến tài chính thông minh.
- + Kiểm thử API: Sử dụng Postman để viết và chạy các tập lệnh kiểm thử API, đảm bảo các luồng dữ liệu hoạt động đúng như thiết kế.
- + Triển khai và CI/CD: Sử dụng Docker để container hóa toàn bộ ứng dụng và thiết lập pipeline CI/CD cơ bản thông qua GitHub Actions, hỗ trợ kiểm thử và triển khai tự động.

#### **1.4.3 Về đối tượng sử dụng**

Dự án hướng đến các đối tượng người dùng cụ thể như sau:

- + Đối tượng chính: Người dùng cá nhân trong độ tuổi từ 16 đến 35, có nhu cầu theo dõi và tối ưu hóa chi tiêu hằng ngày.
- + Đối tượng mở rộng: Các hộ gia đình nhỏ, có thể sử dụng hệ thống như một công cụ chia sẻ và đồng quản lý tài chính nội bộ.
- + Chưa hỗ trợ: Các doanh nghiệp, tổ chức lớn chưa nằm trong phạm vi của hệ thống ở giai đoạn này do đặc thù nghiệp vụ và quy mô phức tạp hơn.

#### **1.4.4 Về thời gian thực hiện**

Dự án được chia thành hai giai đoạn chính để đảm bảo tiến độ và chất lượng:

- + Giai đoạn 1 (1 tháng): Xây dựng phiên bản MVP (Minimum Viable Product) bao gồm các chức năng cơ bản, giao diện nền tảng và kiểm thử nội bộ.
- + Giai đoạn 2 (1 tháng): Tối ưu hóa hiệu năng, giao diện người dùng, bảo mật, đồng thời tích hợp AI, CI/CD và triển khai thực tế trên môi trường cloud.

#### **1.4.5 Về tài nguyên thực hiện**

- + Nhân lực: Nhóm thực hiện gồm 3 thành viên, đảm nhiệm các vai trò thiết kế, lập trình frontend/backend, xây dựng AI và triển khai hệ thống.
- + Ngân sách: Tận dụng tối đa các công cụ và nền tảng mã nguồn mở, miễn phí để tiết kiệm chi phí nhưng vẫn đảm bảo hiệu quả và độ tin cậy.
- + Hạ tầng triển khai: Sử dụng các nền tảng cloud miễn phí (free-tier) như Render, mongodb atlas hoặc tương đương để triển khai hệ thống và dịch vụ.

## 1.5 Đối tượng nghiên cứu

Đối tượng nghiên cứu bao gồm các thành phần liên quan trực tiếp đến việc xây dựng, vận hành và khai thác hệ thống quản lý chi tiêu trên nền tảng web. Cụ thể:

### 1.5.1 Người dùng hệ thống

+ Người dùng cá nhân: Là đối tượng chính mà hệ thống hướng đến. Họ có nhu cầu ghi nhận, theo dõi và phân tích chi tiêu hàng ngày. Đối tượng này chủ yếu nằm trong độ tuổi từ 16 đến 35 tuổi, có hiểu biết cơ bản về công nghệ và thường xuyên sử dụng thiết bị di động hoặc máy tính để quản lý tài chính cá nhân.

+ Hộ gia đình nhỏ: Là nhóm người dùng có thể sử dụng hệ thống như một công cụ đồng quản lý tài chính giữa các thành viên, nhằm chia sẻ thông tin thu/chi và thống nhất kế hoạch chi tiêu gia đình.

### 1.5.2 Hệ thống phần mềm

+ Website quản lý chi tiêu cá nhân: Là sản phẩm trung tâm của đề tài, bao gồm giao diện người dùng (frontend), hệ thống xử lý nghiệp vụ (backend), cơ sở dữ liệu, và các thành phần hỗ trợ khác như API, module AI gợi ý.

+ Hệ thống monolithic: Là mô hình kiến trúc được nghiên cứu và áp dụng để triển khai backend, đảm bảo khả năng mở rộng, phân tách chức năng rõ ràng, dễ bảo trì và thuận tiện trong việc tích hợp với các hệ thống khác.

### 1.5.3 Công nghệ và công cụ

+ Công nghệ phát triển phần mềm: Bao gồm các công nghệ chính như ReactJS, Node.js (NestJS), Docker, GitHub Actions và các công cụ hỗ trợ kiểm thử, thiết kế như Postman, Figma, Jira.

+ Công nghệ triển khai và kiểm thử: Là các công cụ và nền tảng phục vụ việc container hóa, CI/CD, kiểm thử API, kiểm thử người dùng và triển khai hệ thống lên môi trường thực tế (cloud).

## 1.6 Phương pháp nghiên cứu

### 1.6.1 Phương pháp khảo sát và phân tích yêu cầu

Mục tiêu: Xác định nhu cầu thực tiễn của người dùng và xây dựng bộ yêu cầu phần mềm đầy đủ, rõ ràng.

Cách thực hiện:

+ Khảo sát người dùng tiềm năng: Nhóm tiến hành phỏng vấn và thu thập phản hồi từ các đối tượng người dùng mục tiêu như sinh viên, nhân viên văn phòng, người đi làm có nhu cầu quản lý tài chính cá nhân.

+ Nghiên cứu đối thủ cạnh tranh: Tiến hành phân tích các ứng dụng phổ biến trong cùng lĩnh vực như *Money Lover*, *Spendee*, từ đó rút ra ưu/nhược điểm và các tính năng cần thiết.

+ Tổng hợp và mô hình hóa yêu cầu: Các dữ liệu thu được được tổng hợp thành tài liệu đặc tả yêu cầu phần mềm (SRS – Software Requirements Specification), đóng vai trò nền tảng cho toàn bộ quá trình thiết kế và phát triển.

### **1.6.2 Phương pháp phát triển linh hoạt Agile (Scrum)**

Mục tiêu: Tối ưu hóa quy trình phát triển phần mềm bằng cách chia nhỏ dự án thành các chu kỳ phát triển ngắn, có khả năng phản hồi linh hoạt với thay đổi.

Cách thực hiện:

+ Phân chia Sprint: Dự án được chia thành các Sprint kéo dài từ 2 đến 4 tuần, mỗi Sprint tập trung vào phát triển và hoàn thiện một tập hợp chức năng cụ thể.

+ Quản lý công việc bằng công cụ chuyên dụng: Sử dụng *Jira* hoặc *GitHub Projects* để phân chia, theo dõi và đánh giá tiến độ công việc.

+ Họp nhóm thường xuyên: Các buổi họp Daily Standup giúp cập nhật tiến độ, xử lý trở ngại nhanh chóng và điều chỉnh kế hoạch linh hoạt.

### **1.6.3 Phương pháp thiết kế nguyên mẫu (Prototyping)**

Mục tiêu: Tạo ra các bản thiết kế sơ bộ (prototype) nhằm kiểm tra và điều chỉnh giao diện người dùng trước khi tiến hành lập trình chính thức.

Cách thực hiện:

+ Sử dụng công cụ thiết kế Figma: Nhóm đã sử dụng Figma để thiết kế wireframe và mockup, mô phỏng các màn hình và luồng chức năng chính của hệ thống.

+ Thu thập phản hồi từ người dùng: Các mẫu thiết kế được trình bày cho người dùng mục tiêu để kiểm thử sơ bộ và thu thập góp ý nhằm cải tiến trải nghiệm người dùng (UX) và giao diện người dùng (UI).

#### **1.6.4 Phương pháp kiểm thử phần mềm (Testing)**

Mục tiêu: Đảm bảo tính chính xác, ổn định và hiệu năng của hệ thống trước khi triển khai thực tế.

Cách thực hiện:

+ Kiểm thử đơn vị (Unit Test): Áp dụng các framework như *Jest* để kiểm tra logic của từng hàm, từng module trong hệ thống một cách độc lập.

+ Kiểm thử tích hợp (Integration Test): Dùng công cụ *Postman* để kiểm thử các API, đảm bảo các thành phần frontend – backend hoạt động đúng và tương tác chính xác.

+ Kiểm thử chấp nhận người dùng (UAT): Tổ chức các buổi thử nghiệm thực tế với người dùng nhằm kiểm tra xem hệ thống có đáp ứng đúng yêu cầu và trải nghiệm sử dụng có thuận tiện không.

#### **1.6.5 Phương pháp triển khai và tích hợp liên tục (CI/CD)**

Mục tiêu: Tự động hóa quy trình build, kiểm thử và triển khai ứng dụng nhằm giảm thiểu lỗi và đẩy nhanh tiến độ phát hành.

Công cụ sử dụng:

+ GitHub Actions: Thiết lập pipeline CI để tự động kiểm tra code, chạy test mỗi khi có commit mới được đẩy lên repository.

+ Docker: Đóng gói toàn bộ ứng dụng vào container, đảm bảo tính đồng nhất trong môi trường chạy giữa development và production.

+ Render / Heroku: Triển khai hệ thống demo trực tuyến trên nền tảng cloud giúp việc thử nghiệm và đánh giá sản phẩm dễ dàng hơn.

#### **1.6.6 Phương pháp đánh giá và cải tiến**

Mục tiêu: Nâng cao chất lượng sản phẩm bằng việc đánh giá sau mỗi chu kỳ phát triển và điều chỉnh liên tục theo phản hồi thực tế.

Cách thực hiện:

+ Sprint Retrospective: Sau mỗi Sprint, nhóm tổ chức họp đánh giá lại quá trình làm việc, chỉ ra điểm mạnh/yếu và đề xuất cải tiến cho Sprint kế tiếp.

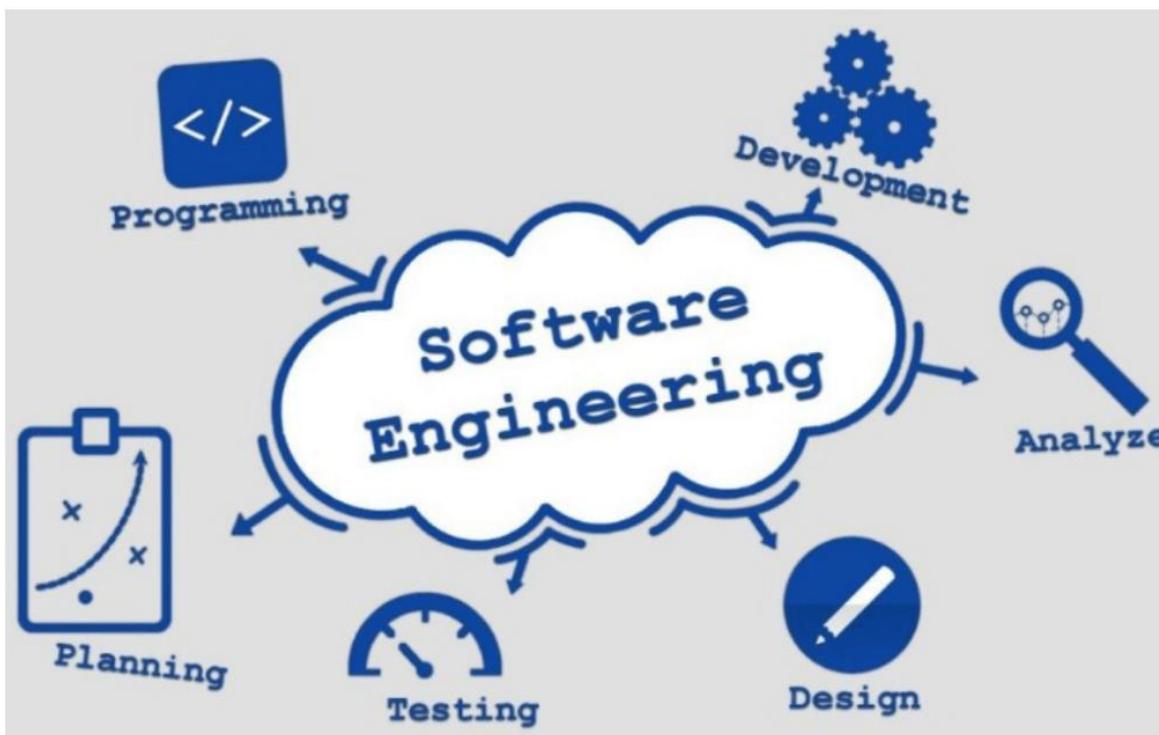
+ Sử dụng công cụ phân tích người dùng (*nếu có*): Theo dõi hành vi người dùng qua các công cụ Analytics để nhận diện các vấn đề tiềm ẩn và điều chỉnh hệ thống cho phù hợp.

## CHƯƠNG 2: CƠ SỞ LÍ THUYẾT

### 2.1 Các khái niệm cơ bản về công nghệ phần mềm

#### 2.1.1 Công nghệ phần mềm là gì?

Công nghệ phần mềm (hay còn được gọi kỹ thuật phần mềm). Đó là những khái niệm trong ngành công nghệ thông tin, có sự liên quan mật thiết tới các khía cạnh của quá trình sản xuất phần mềm. Công nghệ được áp dụng một cách có hệ thống cho sự phát triển, sử dụng cũng như để bảo trì các phần mềm hệ thống.



Hình 2.1 Tìm hiểu về Công nghệ phần mềm

Công nghệ phần mềm được xem là một bộ phận của quy trình công nghệ hệ thống, có liên quan tới sự phát triển của các ứng dụng, hạ tầng, cơ sở dữ liệu và điều khiển hệ thống. Các kỹ sư phần mềm luôn phải tuân thủ quy định của hệ thống, tổ chức trong công việc cũng như khi sử dụng kỹ thuật, công cụ phù hợp với từng vấn đề, tài nguyên sẵn có.

Khác với khoa học máy tính, công nghệ phần mềm không chỉ đề cập tới lý thuyết và các vấn đề cơ bản, mà nó còn tập trung vào hoạt động xây dựng chế tạo ra các sản phẩm phần mềm hệ thống hay phần mềm ứng dụng hữu ích với con người. Sự phát triển mạnh mẽ của ngành kỹ thuật phần mềm đã vượt xa hơn hẳn những lý thuyết khoa học máy tính tích góp nhỏ giọt.

### **2.1.2 Phần mềm là gì?**

Phần mềm (software) là tập hợp các chương trình, thủ tục, tài liệu liên quan và dữ liệu được thiết kế để thực hiện một hoặc nhiều nhiệm vụ cụ thể trên hệ thống máy tính. Phần mềm là yếu tố phi vật lý, trái ngược với phần cứng – tức các thiết bị vật lý như CPU, RAM, ổ cứng,...

Phần mềm giúp điều khiển phần cứng, hỗ trợ người dùng thao tác và khai thác hệ thống máy tính hiệu quả. Trong một hệ thống tin học hoàn chỉnh, phần mềm đóng vai trò trung gian giữa người dùng và phần cứng, cho phép thực hiện các tác vụ từ đơn giản (soạn thảo văn bản) đến phức tạp (quản lý hệ thống doanh nghiệp, trí tuệ nhân tạo,...).

Phần mềm thường được chia thành ba loại chính:

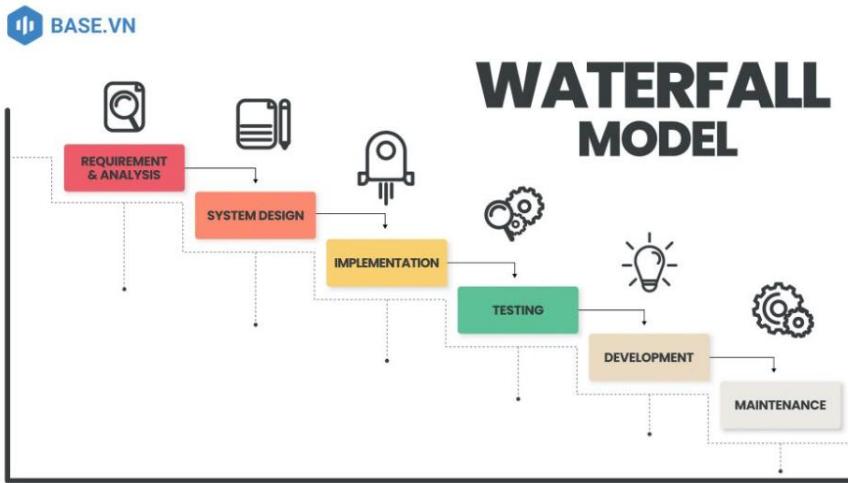
- + Phần mềm hệ thống: Bao gồm hệ điều hành và các phần mềm quản lý phần cứng.
- + Phần mềm ứng dụng: Được thiết kế để phục vụ nhu cầu cụ thể của người dùng, như phần mềm kế toán, quản lý bán hàng, trình duyệt web,...
- + Phần mềm lập trình: Hỗ trợ các lập trình viên trong việc viết, kiểm thử và bảo trì mã nguồn, bao gồm các IDE, trình biên dịch,...

Với sự phát triển mạnh mẽ của công nghệ thông tin, phần mềm đã trở thành một yếu tố không thể thiếu trong mọi lĩnh vực đời sống – từ giáo dục, y tế, tài chính đến thương mại điện tử, truyền thông và giải trí.

## **2.2 Các mô hình phát triển phần mềm**

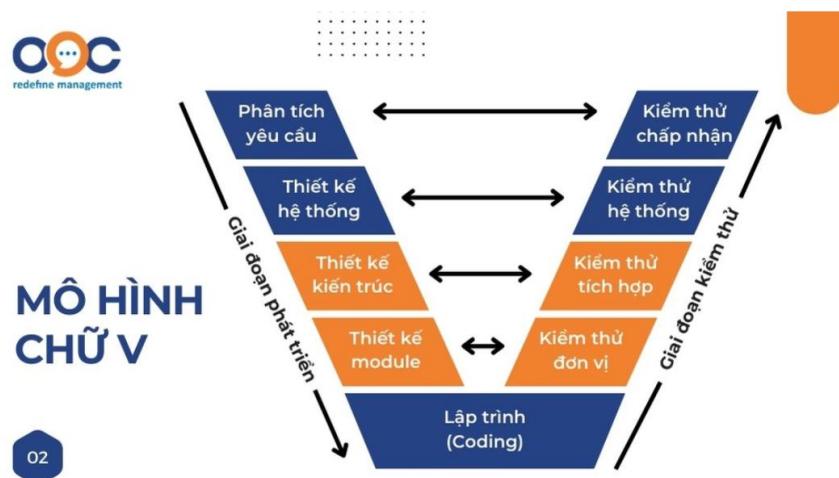
Trong kỹ thuật phần mềm, mô hình phát triển phần mềm (Software Development Model) là một khung quy trình mô tả cách tổ chức các hoạt động trong vòng đời phát triển phần mềm, từ khâu yêu cầu đến bảo trì. Việc lựa chọn mô hình phù hợp ảnh hưởng lớn đến hiệu quả, chất lượng và khả năng kiểm soát dự án phần mềm. Dưới đây là một số mô hình phổ biến:

- Mô hình Thác nước – Waterfall: Mô hình Waterfall, hay còn được gọi là Mô hình thác nước, là một trong những mô hình quản lý dự án dễ sử dụng nhất hiện nay. Nó tập trung vào việc quản lý dự án dựa trên tiến trình tuần tự và liên tiếp, dự án chỉ bước vào giai đoạn mới khi các giai đoạn trước đó đã hoàn thành.



Hình 2.2 Mô hình thác nước

- Mô hình chữ V (V model): là mô hình mở rộng của mô hình thác nước, mô hình chữ V tổ chức quy trình phát triển phần mềm theo dạng chữ V, trong đó mỗi giai đoạn phát triển đều có giai đoạn kiểm thử tương ứng ở phía đối xứng. Mô hình nhấn mạnh việc kiểm thử song song với phát triển, đảm bảo chất lượng ngay từ đầu.



Cấu trúc mô hình chữ V

Hình 2.3 Mô hình chữ V

Mô hình tăng trưởng hay còn được gọi là mô hình gia tăng (Incremental model) là một mô hình phát triển phần mềm theo từng giai đoạn, trong đó mỗi giai đoạn được gọi là một "nấc thang". Mỗi nấc thang sẽ bao gồm các hoạt động như thu thập yêu cầu, phân tích yêu cầu, thiết kế, xây dựng, kiểm thử và triển khai

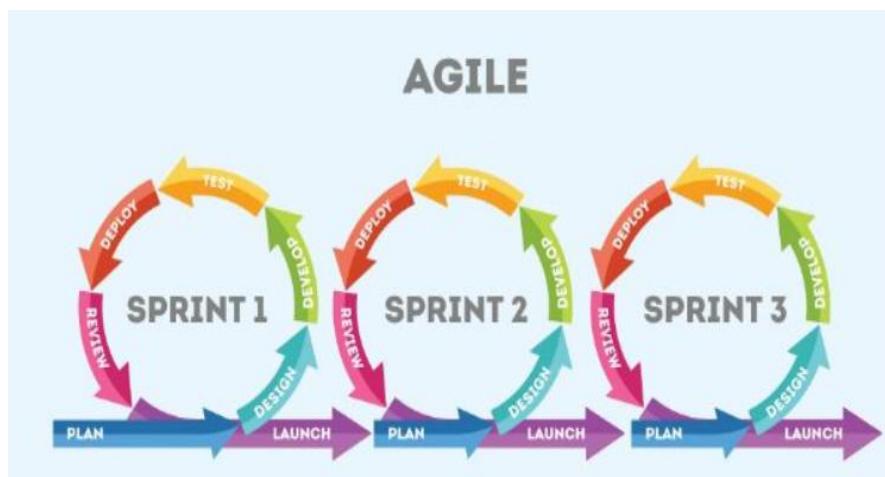


Hình 2.4 Mô hình Incremental Model

- Mô hình Agile là một phương pháp quản lý được áp dụng trong lĩnh vực phát triển sản phẩm hoặc dự án CNTT. Phương pháp này hoạt động dựa trên việc chia dự án thành các giai đoạn khác nhau và nhấn mạnh vào sự hợp tác, giao tiếp cởi mở, thích ứng và tin tưởng lẫn nhau giữa các thành viên trong một nhóm sản phẩm.

**Đặc điểm:**

- + Linh hoạt, dễ thích nghi với thay đổi.
- + Mỗi chu kỳ ngắn (sprint) đều tạo ra phiên bản phần mềm có thể sử dụng được.
- + Đòi hỏi khách hàng và nhóm phát triển phối hợp chặt chẽ, liên tục.



Hình 2.5 Mô hình cách thức hoạt động của Agile

- Mô hình DevOps: là sự kết hợp giữa Development (phát triển) và Operations (vận hành), nhằm xây dựng một quy trình làm việc khép kín, liền mạch giữa các bộ phận trong quá trình phát triển phần mềm. Đây là xu hướng của hiện đại, nhấn mạnh tự động hóa, triển khai liên tục và tích hợp liên tục (CI/CD).



Hình 2.6 Mô hình DevOps

## 2.3 Tổng quan về Agile

### 2.3.1 Agile là gì?

Agile (viết tắt của Agile Software Development) là một phương thức phát triển phần mềm linh hoạt, được thực hiện bằng cách sử dụng các bước lặp ngắn từ 1 đến 4 tuần. Mục tiêu của Agile là giúp rút ngắn thời gian phát triển sản phẩm, đưa sản phẩm đến với tay khách hàng càng sớm càng tốt.

Về bản chất, Agile giống như một phương pháp luận, một triết lý dựa trên nguyên tắc phân đoạn vòng lặp (iterative) và tăng trưởng (incremental) nên sở hữu tính linh hoạt cao. Tính chất này đi ngược lại với các phương pháp quản lý dự án truyền thống – vốn dĩ triển khai các giai đoạn một cách tuyến tính và vô cùng bị động trước các thay đổi bất ngờ.



Hình 2.7 Tìm hiểu về Agile

### 2.3.2 Đặc trưng của Agile

Agile có các đặc trưng sau:

+ Tính lặp (Iterative): Dự án sẽ được thực hiện trong các phân đoạn lặp đi lặp lại, thường có khung thời gian ngắn (từ 1-4 tuần). Trong mỗi phân đoạn đó, nhóm phát triển dự án sẽ thực hiện đầy đủ các công việc cần thiết như lập kế hoạch, phân tích yêu cầu, thiết kế, triển khai, kiểm thử để cho ra các phần nhỏ của sản phẩm.

+ Tính tăng trưởng và tiến hóa (Incremental & Evolutionary): Cuối các phân đoạn, nhóm cho ra các phần nhỏ của sản phẩm cuối cùng, thường là đầy đủ, có khả năng chạy tốt, được kiểm thử cẩn thận và có thể sử dụng. Theo thời gian, phân đoạn này tiếp nối phân đoạn kia, các phần chạy được này sẽ được tích lũy, lớn dần lên cho tới khi toàn bộ yêu cầu của khách hàng được thỏa mãn.

+ Tính thích nghi (adaptive): Do các phân đoạn chỉ kéo dài trong một khoảng thời gian ngắn và việc lập kế hoạch cũng được điều chỉnh liên tục, nên các thay đổi trong quá trình phát triển (yêu cầu thay đổi, thay đổi công nghệ, thay đổi định hướng về mục tiêu,...) đều có thể được đáp ứng theo cách thích hợp.

+ Nhóm tự tổ chức và liên chức năng: Các cấu trúc nhóm này tự phân công công việc mà không dựa trên các mô tả cứng nhắc về chức danh hay một sự phân cấp rõ ràng. Nhóm tự tổ chức đã đủ các kỹ năng cần thiết để có thể được trao quyền tự ra quyết định, tự quản lý và tổ chức lấy công việc của chính mình để đạt được hiệu quả cao nhất.

+ Quản lý tiến trình thực nghiệm (Empirical Process Control): Các nhóm Agile ra các quyết định dựa trên các dữ liệu thực tiễn (data-driven) thay vì tính toán lý thuyết hay các tiền giả định. Agile rút ngắn vòng đời phản hồi để dễ dàng thích nghi và gia tăng tính linh hoạt nhờ đó có thể kiểm soát được tiến trình, và nâng cao năng suất lao động.

+ Giao tiếp trực diện (face-to-face communication): Agile không phản đối việc tài liệu hóa, nhưng đánh giá cao hơn việc giao tiếp trực diện thay vì thông qua giấy tờ. Trong giao tiếp giữa nội bộ nhóm, Agile khuyến khích trực tiếp trao đổi và thống nhất với nhau về thiết kế của hệ thống và cùng nhau triển khai thành các chức năng theo yêu cầu.

+ Phát triển dựa trên giá trị (value-based development): Một trong các nguyên tắc cơ bản của Agile là “sản phẩm chạy tốt chính là thước đo của tiến độ”. Nhóm Agile

thường cộng tác trực tiếp và thường xuyên với khách hàng để biết yêu cầu nào có độ ưu tiên cao hơn, mang lại giá trị hơn sớm nhất có thể cho dự án.

### 2.3.3 Các nguyên tắc của Agile

Nguyên tắc trong phát triển phần mềm Agile được trình bày trong Manifesto Agile và mô tả các giá trị và nguyên tắc cốt lõi của phương pháp phát triển Agile. Dưới đây là 12 nguyên tắc chính trong Agile software development:

- + Khách hàng hài lòng bằng cách cung cấp phần mềm có giá trị cao: Cung cấp phần mềm có giá trị cao, liên tục và sớm để đáp ứng nhu cầu của khách hàng và cải thiện sự hài lòng của họ.
- + Thay đổi yêu cầu là điều bình thường: Đón nhận và thích ứng với thay đổi yêu cầu, ngay cả khi phát triển ở giai đoạn muộn. Điều này giúp cung cấp giá trị tốt nhất cho khách hàng.
- + Phát triển phần mềm liên tục và thường xuyên: Cung cấp phần mềm có thể hoạt động một cách thường xuyên và liên tục, từ vài tuần đến vài tháng, với một chu kỳ phát triển ngắn hơn.
- + Hợp tác chặt chẽ giữa các bên liên quan: Hợp tác liên tục giữa các nhà phát triển và các bên liên quan (bao gồm khách hàng và các thành viên nhóm) trong suốt quá trình phát triển.
- + Xây dựng dựa trên đội ngũ tự quản lý và có động lực cao: Xây dựng dự án xung quanh các đội ngũ tự quản lý và có động lực cao. Đảm bảo rằng các thành viên đội ngũ có thể tự tổ chức và có đủ kỹ năng và sự tự tin để hoàn thành công việc.
- + Tạo ra phần mềm có thể hoạt động được: Phần mềm hoạt động là mục tiêu chính. Các bản phát hành nhỏ và có thể hoạt động là ưu tiên hàng đầu.
- + Thiết kế đơn giản và tối ưu: Thiết kế phần mềm đơn giản, với tối đa tính năng cần thiết để đáp ứng nhu cầu hiện tại. Tập trung vào việc giảm thiểu công việc chưa cần thiết và tối ưu hóa hiệu suất.
- + Tốc độ phát triển và khả năng duy trì cao: Duy trì khả năng phát triển nhanh chóng và có khả năng thay đổi dễ dàng. Điều này bao gồm việc duy trì mã nguồn đơn giản, dễ hiểu và có thể dễ dàng thay đổi khi cần thiết.

+ Phản hồi sớm và liên tục từ khách hàng: Nhận phản hồi thường xuyên từ khách hàng để đảm bảo rằng phần mềm đáp ứng nhu cầu và mong muốn của họ. Phản hồi nhanh chóng giúp điều chỉnh và cải thiện sản phẩm liên tục.

+ Kỹ thuật và thiết kế tốt: Khuyến khích kỹ thuật tốt và thiết kế phần mềm tốt, giúp duy trì sự nhất quán và chất lượng của mã nguồn. Đầu tư vào kỹ thuật và thiết kế sẽ giúp sản phẩm cuối cùng có chất lượng cao hơn.

+ Tạo môi trường làm việc tích cực: Tạo ra một môi trường làm việc tích cực, nơi các thành viên trong nhóm có thể giao tiếp mở, hỗ trợ lẫn nhau và làm việc cùng nhau hiệu quả.

+ Đánh giá và cải tiến thường xuyên: Đánh giá thường xuyên các quy trình làm việc, công cụ và kỹ thuật. Cải tiến liên tục giúp nâng cao hiệu suất nhóm và chất lượng sản phẩm.

### **2.3.4 Ưu và nhược điểm của Agile**

#### **2.3.4.1 Ưu điểm của Agile**

Mô hình Agile ngày càng trở nên phổ biến, được nhiều doanh nghiệp áp dụng để phát triển sản phẩm của mình bởi nó sở hữu những ưu điểm vượt trội so với những phương pháp truyền thống.

+ Dễ dàng thực hiện thay đổi ở bất kỳ giai đoạn nào của dự án, thích nghi nhanh và hiệu quả với sự thay đổi, bao gồm cả yêu cầu sửa lại sản phẩm, sự biến động từ thị trường,...

+ Phát triển và bàn giao sản phẩm nhanh hơn, bởi việc chia nhỏ dự án cho phép đội ngũ có thể tiến hành kiểm tra theo từng phần, phát hiện sự cố và sửa chữa vấn đề nhanh hơn.

+ Sản phẩm đạt chất lượng tốt hơn, do luôn nhận được phản hồi ngay lập tức từ phía khách hàng và được tối ưu lại ngay sau đó.

+ Lãng phí ít tài nguyên hơn vì nhóm luôn thực hiện các công việc đã được cập nhật, giảm tải thời gian chờ, giảm tải thời gian sửa lỗi sản phẩm, giảm tải số lượng lớn giấy tờ tài liệu,...

+ Người tham gia dự án không cần phải nắm mọi thông tin ngay từ đầu, phù hợp với những dự án chưa xác định rõ ràng được mục tiêu cuối cùng.

#### 2.3.4.2 Nhược điểm của Agile

Bên cạnh những ưu điểm vượt trội, cũng không khó để nhận ra một số nhược điểm còn tồn tại khi triển khai mô hình Agile trong thực tế:

+ Khó lên kế hoạch dự án, đặc biệt là khó xác định rõ ràng thời gian bàn giao sản phẩm cuối cùng, không nắm rõ được chi phí thực sự của dự án là bao nhiêu,... vì dự án được chia nhỏ thành các vòng lặp khác nhau.

+ Bắt buộc phải hướng dẫn và đào tạo chi tiết, thì thành viên dự án mới có thể hiểu rõ được mô hình Agile và thực hiện theo nó một cách rõ ràng, đặc biệt là trong thời gian đầu.

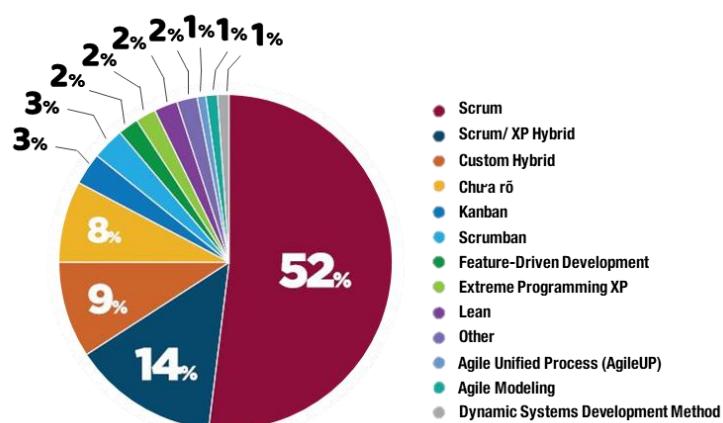
+ Ít tài liệu hướng dẫn về dự án và không xác định rõ được kỳ vọng và thành phẩm ngay từ đầu, do người thực thi mô hình Agile thường tin rằng mọi thứ sẽ thay đổi rất nhiều, không cần thiết phải ghi chép và lưu trữ tài liệu.

+ Đòi hỏi sự cam kết về thời gian và công sức từ các bên hơn, vì tất cả mọi người cần phải liên tục tương tác với nhau trong suốt quá trình thực thi dự án.

+ Chi phí thực hiện dự án theo mô hình Agile thường cao hơn so với các phương pháp phát triển khác.

#### 2.3.5 Các phương pháp Agile phổ biến

Bảng thống kê dưới đây liệt kê 13 phương pháp họ Agile, cho thấy phần lớn các công ty hiện nay đã sử dụng Scrum như một cách tiếp cận cơ bản. Bên cạnh đó, nhiều công ty đã kết hợp các phương pháp lại với nhau. Ví dụ 44.4% các công ty có sử dụng Waterfall, có nghĩa là một tỉ lệ nhất định nào đó vừa dùng Waterfall, vừa sử dụng Scrum trong hoạt động của mình.



Hình 2.7 Các phương pháp phổ biến trong Agile

- Scrum là một khung tổ chức công việc (framework) hoạt động dựa trên cơ chế lặp và tăng trưởng, cho phép nhóm dự án tập trung vào việc tối ưu hóa giá trị đạt được trong các vòng lặp cố định về mặt thời gian. Điểm nổi bật của Scrum chính là các Sprint – các giai đoạn phát triển và vòng lặp và việc tối đa hóa thời gian phát triển sản phẩm để có thể đạt được mục tiêu – các Product Goal.

- Phương pháp Kanban: được tổ chức trên một tấm bảng trực quan chia thành các cột khác nhau, thể hiện các bước liên tiếp trong luồng công việc của dự án sản xuất. Bạn có thể bắt đầu một dự án Kanban đơn giản nhất với 3 cột cơ bản: To-do (sẽ làm), Doing (đang làm) và Done (hoàn thành). Trong quá trình triển khai, các thông tin trong bảng sẽ liên tục thay đổi. Và mỗi khi có nhiệm vụ mới xuất hiện, một “thẻ” mới sẽ được tạo.

- Lean là một mô hình quản trị theo triết lý tinh gọn, bắt đầu xuất hiện từ lĩnh vực sản xuất và dần được ứng dụng rộng rãi trong nhiều ngành nghề khác nhau. Mô hình này dựa trên ý tưởng tăng năng suất và giảm lãng phí trong sản xuất.

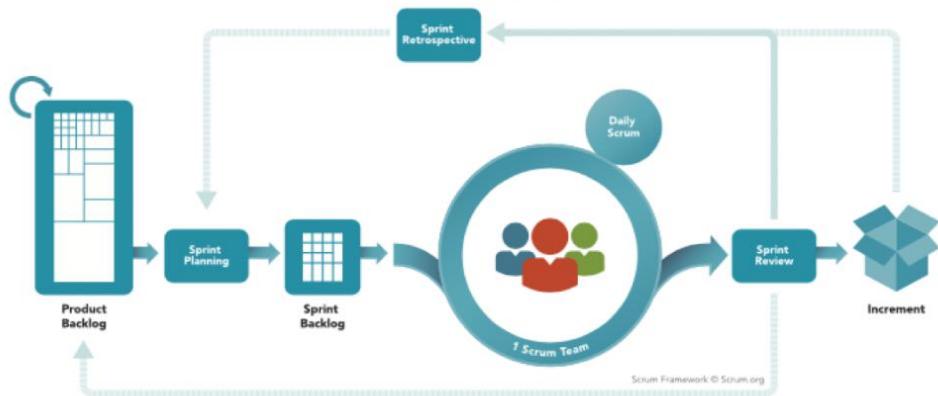
- Extreme Programming (XP) là một framework điển hình dựa trên phương pháp Agile, có thể điều chỉnh cho phù hợp với các quy mô công ty khác nhau. XP hoạt động dựa trên ý tưởng khám phá “điều đơn giản nhất khả thi” mà không đặt quá nhiều công sức vào các tầm nhìn dài hạn cho sản phẩm.

- Crystal: là một framework làm việc theo mô hình Agile linh hoạt nhất, mang đến sự tự do trong việc phát triển cho nhóm triển khai. Phương pháp này tập trung nhiều vào các cá nhân và cách họ tương tác hơn là vào quy trình và các công cụ.

### 2.3.6 Phương pháp Scrum

Scrum là một “bộ khung làm việc” cơ bản để tiếp cận những công việc phức tạp. Dựa trên bộ khung này, nhóm làm việc có thể áp dụng những quy trình, kỹ thuật khác nhau cho công việc của mình... Nó là một thành viên của họ Agile.

## SCRUM FRAMEWORK



Hình 2.8 Hình ảnh về SCRUM

Các giá trị cốt lõi của Scrum:

Ba yếu tố nòng cốt tạo thành một mô hình quản lý tiến trình thực nghiệm gồm: sự minh bạch (transparency), thanh tra (inspection) và thích nghi (adaptation).

+ Minh bạch: Muốn áp dụng thành công Scrum, các thông tin liên quan đến quá trình phải minh bạch và thông suốt. Các thông tin có thể là tầm nhìn của sản phẩm, yêu cầu của khách hàng, tiến độ công việc, các rào cản khác... Từ đó mọi thành viên ở vai trò khác nhau có đầy đủ thông tin cần có để tiến hành quyết định trong việc nâng cao hiệu quả công việc.

+ Thanh tra: Phải thường xuyên thanh tra các hoạt động trong Scrum và tiến độ đến đích để phát hiện các bất thường không theo ý muốn. Tần suất thanh tra không nên quá dày để khỏi ảnh hưởng đến công việc. Công tác thanh tra khi được thực hiện bởi người có kỹ năng tại các điểm quan trọng của công việc sẽ giúp cải tiến liên tục trong Scrum.

+ Thích nghi: Scrum mang lợi thế là tính linh hoạt rất cao, nhờ đó mang lại tính thích nghi cao. Dựa vào thông tin liên tục và minh bạch từ quá trình thanh tra và làm việc, Scrum có thể cho lại các thay đổi tích cực, nhờ đó mang lại thành công cho dự án.

Lợi ích của Scrum:

- + Cải thiện chất lượng phần mềm, dễ học và dễ sử dụng.
- + Rút ngắn thời gian phát hành phần mềm, cho phép khách hàng sử dụng sản phẩm sớm hơn.
- + Nâng cao tinh thần đồng đội, tối ưu hóa hiệu quả và nỗ lực của đội phát triển.

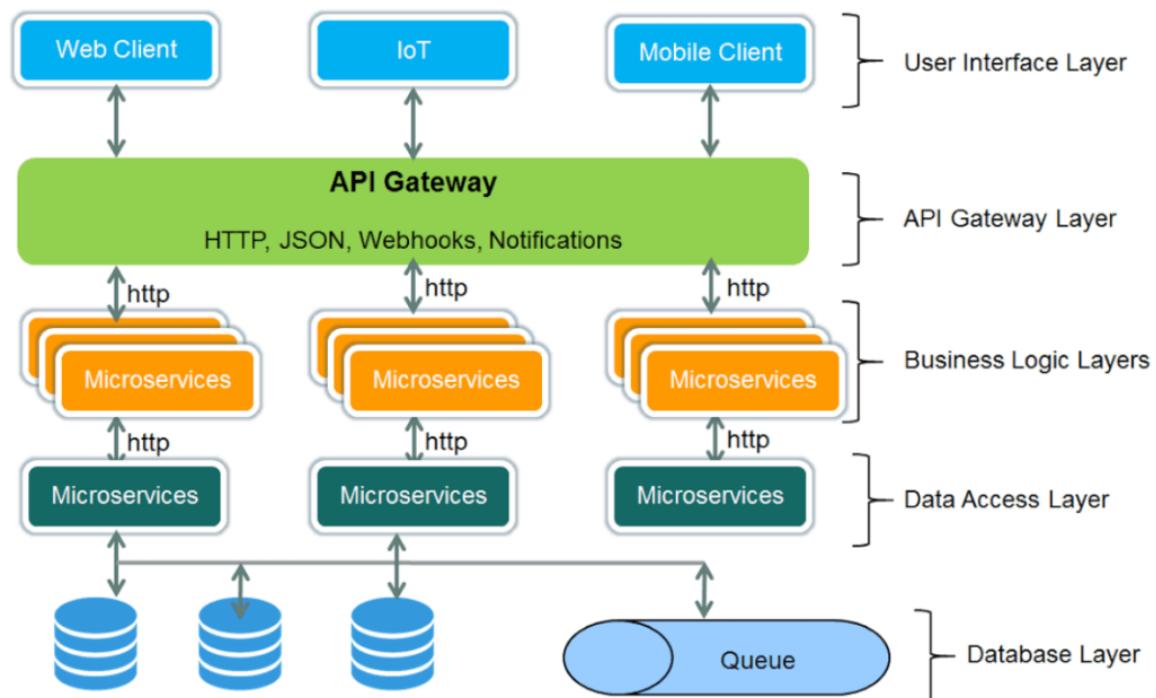
- + Gia tăng tỷ suất hoàn vốn đầu tư (ROI)
- + Tăng mức độ hài lòng của khách hàng
- + Kiểm soát dự án tốt, cải tiến liên tục
- + Giảm thiểu rủi ro khi xây dựng sản phẩm

## 2.4 Mô hình kiến trúc Microservice

### 2.4.1 Microservices là gì?

Microservices là các module trong hệ thống được chia thành nhiều services nhỏ. Mỗi service sẽ thực hiện các chức năng chuyên biệt, như quản lý đơn hàng hoặc quản lý khách hàng,... và được đặt tại một server riêng, cho phép nâng cấp chỉnh sửa một cách độc lập. Các server này có thể giao tiếp thông qua các phương thức như gRPC, Rest API, lambda và không bị ảnh hưởng bởi nhau.

Việc áp dụng kiến trúc microservices cho phép chia nhỏ chức năng của ứng dụng thành các dịch vụ nhỏ, tối ưu hóa trải nghiệm và tốc độ cho từng người dùng. Thiết kế giao diện dựa trên từng đối tượng giúp cải thiện tương thích và tốc độ, đồng thời giảm thiểu các chức năng không cần thiết.



Hình 2.9 Mô hình kiến trúc Microservice

### **2.4.2 Các đặc trưng của mô hình Microservice**

- Micro-service: Đặc trưng này được thể hiện ngay từ tên của kiến trúc. Nó là microservice chứ không phải là miniservice hay nanoservice. Trên thực tế không tồn tại mô hình kiến trúc cho miniservice hay nanoservice. Từ microservice được sử dụng để giúp người thiết kế có cách tiếp cận đúng đắn. Một ứng dụng lớn cần được chia nhỏ ra thành nhiều thành phần, các thành phần đó cần tách biệt về mặt dữ liệu (database) và phải đủ nhỏ cả về mặt kích cỡ và độ ảnh hưởng của nó trong hệ thống, khi thêm một microservice vào hệ thống cũng nên đảm bảo rằng nó đủ nhỏ để dễ dàng tháo gỡ, xóa bỏ khỏi hệ thống mà không ảnh hưởng nhiều tới các thành phần khác.

- Tính độc lập:

+ Các microservice hoạt động tách biệt nhau trong hệ thống, do vậy việc build một microservice cũng độc lập với việc build các microservice khác. Thông thường, để tiện cho việc phát triển và duy trì các microservice, người phát triển nên viết các built script khác nhau cho mỗi microservice.

+ Do tính tách biệt này mà mỗi microservice đều dễ dàng thay thế và mở rộng. Hơn thế nữa, nó còn giúp việc phát triển các microservice linh động hơn, các microservice có thể được phát triển bởi các team khác nhau, dùng các ngôn ngữ khác nhau và tiến độ phát triển dự án cũng nhanh hơn do không có sự phụ thuộc giữa các team, mỗi team có thể chủ động quản lý phần việc riêng của mình.

- Tính chuyên biệt: Mỗi microservice là một dịch vụ chuyên biệt, có thể hoạt động độc lập, thông thường mỗi microservice đại diện cho một tính năng mà các công ty/ doanh nghiệp muốn cung cấp tới người dùng, do vậy người thiết kế hệ thống microservice cần hiểu rõ về hoạt động kinh doanh của công ty. Các đầu vào đầu ra và chức năng của mỗi microservice cần được định nghĩa rõ ràng.

- Phòng chống lỗi:

+ Kiến trúc microservice sinh ra là để dành cho các hệ thống từ lớn đến vô cùng lớn. Nó áp dụng phương pháp chia để trị, phương pháp này giúp việc áp dụng các công cụ, kỹ thuật cho việc giám sát, phòng chống lỗi phần mềm, lỗi hệ thống hiệu quả.

+ Khi một thành phần trong hệ thống bị lỗi, nó có thể được thay thế bằng các thành phần dự phòng một cách dễ dàng, trong quá trình thay thế thành phần bị lỗi, các

thành phần khác vẫn hoạt động bình thường, do vậy hoạt động của toàn bộ hệ thống sẽ không hoặc ít bị gián đoạn.

### **2.4.3 Các ưu và nhược điểm của Microservice**

#### **2.4.3.1 Ưu điểm của Microservice architecture**

+ Cho phép lập trình viên linh động hơn trong việc lựa chọn ngôn ngữ, công cụ và nền tảng để phát triển và triển khai các microservice (tuy nhiên trong một hệ thống, việc lựa chọn các ngôn ngữ khác nhau để phát triển các microservice không được khuyến khích)

+ Một microservice có thể được phát triển bởi một team nhỏ. Do vậy việc quản lý sẽ dễ dàng hơn.

+ Dễ dàng thực hiện tự động tích hợp và tự động triển khai (CI-CD) bằng cách sử dụng một số công cụ như Jenkins, Hudson ...

+ Mỗi microservice có kích thước nhỏ, giúp cho các lập trình viên dễ tiếp cận, đọc hiểu source code. Do vậy các thành viên mới tham gia team sẽ hòa nhập và đóng góp cho team nhanh hơn.

+ Các microservice khởi động nhanh giúp quá trình phát triển, kiểm thử cũng nhanh hơn.

+ Dễ dàng mở rộng và tích hợp với các dịch vụ của bên thứ ba.

+ Cố lập lỗi tốt hơn, khi một microservice bị lỗi và ngừng hoạt động thì các microservice khác vẫn có thể hoạt động bình thường. Với mô hình nguyên khôi, một lỗi nhỏ có thể làm cả hệ thống ngừng hoạt động.

+ Khi cần thay đổi một thành phần, thì chỉ cần sửa đổi, cập nhật và triển khai lại thành phần đó chứ không cần triển khai lại toàn bộ hệ thống.

#### **2.4.3.2 Nhược điểm của Microservice**

+ Nhược điểm của kiến trúc microservice đến từ bản chất của hệ thống phân tán.

+ Việc triển khai hệ thống microservice phức tạp hơn nhiều so với việc triển khai hệ thống nguyên khôi.

+ Các lập trình viên phải tốn nhiều công sức hơn để thực hiện phần giao tiếp giữa các microservice, với kiến trúc nguyên khôi có khi họ chỉ cần gọi hàm để thực hiện việc này.

+ Các microservice thường (nên) được triển khai bên trong docker container và giao tiếp với nhau qua REST API. Việc này làm hiệu năng của toàn bộ chương trình ứng dụng giảm xuống đáng kể do giới hạn tốc độ truyền tải của các giao thức và tốc độ mạng. Hơn nữa việc giao tiếp giữa các microservice có thể bị lỗi khi các kết nối bị lỗi.

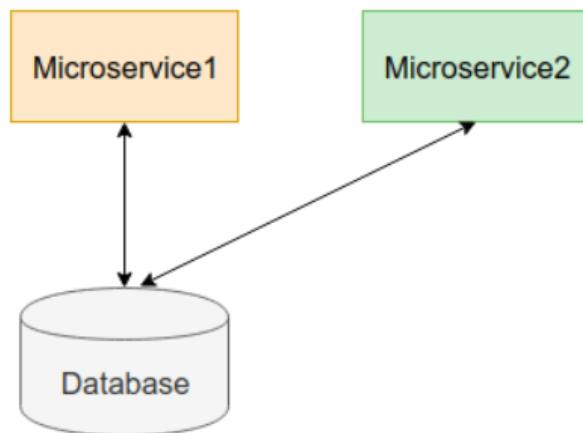
+ Cần tính toán kích cỡ của một microservice. Nếu một microservice quá lớn, bản thân nó trở thành một ứng dụng theo kiến trúc nguyên khối. Nếu một microservice quá nhỏ thì độ phức tạp của hệ thống tăng lên rất nhiều, làm cho hệ thống trở lên khó hiểu, lúc này việc quản lý giám sát và triển khai hệ thống sẽ khó khăn hơn.

+ Khi ứng dụng ngày càng lớn lên, số lượng microservice ngày càng nhiều, các lập trình viên thường có xu hướng sử dụng sự hỗ trợ từ các công cụ mã nguồn mở, hoặc của bên thứ 3, việc sử dụng, tích hợp các công cụ này làm cho hệ thống khó kiểm soát và có thể bị dính các mã độc làm cho hệ thống kém an toàn.

#### 2.4.4 Thiết kế phần mềm theo kiến trúc Microservice

##### - Mỗi microservice nên có một database riêng biệt:

Việc này đảm bảo cho microservice có tính đóng gói cao. Tuy vậy việc phân tách nơi chứa dữ liệu cho mỗi microservice không hề đơn giản, nó là phần khó nhất trong việc thiết kế ứng dụng phần mềm theo kiến trúc microservice. Do đó, rất nhiều người chọn lựa thiết kế sử dụng chung database cho nhiều microservice.

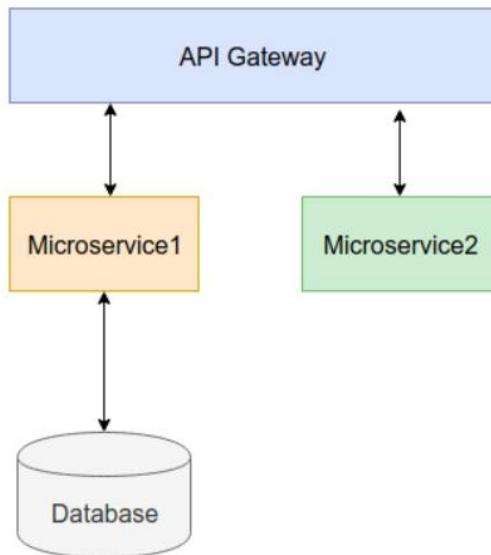


Hình 2.10 Hình ảnh về 1 data riêng biệt

Cách này tồn tại một hạn chế là khi database schema cần thay đổi cho microservice1 thì nó sẽ làm ảnh hưởng và gây lỗi cho các microservice khác sử dụng

chung database này. Để sửa lỗi, ta cần sửa đổi, cập nhật toàn bộ microservice còn lại để chúng có thể hoạt động với schema mới.

Có một cách tiếp cận khác, giúp tránh được hạn chế trên.



Hình 2.11 Mô phỏng cách thực hiện còn lại

Sơ đồ trên, microservice2 không truy cập trực tiếp vào database, thay vào đó nó truy cập database thông qua microservice1. Do đó việc thay đổi schema của database sẽ không ảnh hưởng tới microservice2. Tuy nhiên với cách tiếp cận này thì microservice2 phụ thuộc vào microservice1, khi microservice1 có lỗi thì microservice2 cũng bị lỗi theo.

Việc chọn cách tiếp cận nào phụ thuộc rất nhiều vào tình hình thực tế của dự án. Cần cân nhắc thiệt hơn của mỗi phương án để đưa ra lựa chọn cuối cùng.

#### - Giữ source code của microservice ở mức hợp lý:

Như đã đề cập ở phần ưu và nhược điểm của microservice. Kích thước source code của một microservice không nên quá nhỏ hoặc quá lớn. Tuy nhiên cái khó ở đây là không có một con số định lượng cho kích thước của một microservice, nên thông thường việc quyết định kích thước của một microservice là do kinh nghiệm, cảm tính.

#### - Triển khai mỗi microservice bên trong một app (docker container):

Việc triển khai mỗi microservice trong một docker container đem lại rất nhiều lợi ích cho việc triển khai và mở rộng ứng dụng cũng như việc phân chia tài nguyên phần cứng cho mỗi microservice. Hiện nay có rất nhiều công cụ hỗ trợ cho việc liên tục tích hợp, liên tục triển khai hệ thống microservice. Các công cụ này giúp tăng hiệu quả

làm việc cho các lập trình viên, giảm thời gian phôi phổi sản phẩm phần mềm, và các công cụ này đòi hỏi mỗi microservice được đóng gói trong một docker image và triển khai trên app.

**- Stateless server:**

Khi một yêu cầu được gửi đến server thì một phiên làm việc (session) được mở ra, kèm theo đó là các thông tin của phiên. Stateless server là server không lưu thông tin của phiên. Mà thông tin về phiên được lưu ở một nơi khác, như caching server chẳng hạn.

Việc này rất quan trọng, bởi vì mỗi microservice thường được đóng gói thành một docker image. Khi muốn cập nhật một microservice, ta cập nhật docker image của nó, và khi chạy docker image mới (xóa bỏ container cũ và tạo container mới dựa trên image mới) thì toàn bộ thông tin của các phiên hoạt động trên container cũ sẽ bị mất, thông tin phiên thường bao gồm thông tin mà client gửi tới server, mất thông tin này là một lỗi vô cùng nghiêm trọng. Nếu container là stateless thì nó không lưu thông tin của các phiên nên không có gì để mất cả.

## CHƯƠNG 3: PHÂN TÍCH YÊU CẦU

### 3.1 Xác định nhu cầu

Để đảm bảo quá trình phát triển phần mềm đáp ứng đúng mục tiêu và mong đợi của người dùng cũng như các bên liên quan, nhóm phát triển đã tiến hành xác định và ghi nhận các nhu cầu chính trong suốt quá trình làm việc theo phương pháp lặp – tăng dần trong mô hình Agile Scrum.

Các nhu cầu được xác định dựa trên vai trò của người dùng, người quản trị và chủ doanh nghiệp, với mục tiêu cuối cùng là xây dựng một hệ thống website quản lý chi tiêu cá nhân dễ sử dụng, tiện lợi, và hiệu quả. Việc xác định nhu cầu được chia theo từng tuần làm việc cụ thể như sau:

- + Tuần 1: Người dùng cần thao tác dễ dàng với các trang trên website.
- + Tuần 2: Người dùng cần có khả năng tạo tài khoản để lưu trữ thông tin và dữ liệu chi tiêu.
- + Tuần 3: Người dùng mong muốn có thể tìm kiếm và tương tác với sản phẩm (hoặc các mục chi tiêu) một cách nhanh chóng.
- + Tuần 4: Người dùng muốn có chức năng lưu trữ và thanh toán; người quản trị cần giao diện dễ quản lý thông tin hệ thống.
- + Tuần 5: Chủ doanh nghiệp mong muốn hệ thống dễ triển khai, bảo trì và vận hành.

### 3.2 Các chức năng của hệ thống

#### 3.2.1 Các chức năng chính của hệ thống

- Quản lý tài khoản người dùng
- + Đăng ký/Đăng nhập:
- + Hồ sơ cá nhân:
- + Cập nhật thông tin (tên, avatar, ngân hàng liên kết).
- + Thiết lập ngân sách mặc định theo tháng.
- Quản lý giao dịch
- + Thêm/sửa/xóa giao dịch:
- + Nhập số tiền, danh mục (ăn uống, di chuyển, giải trí...), ngày tháng, ghi chú.
- + Tự động phân loại:
- + Gợi ý danh mục dựa trên mô tả (VD: "Starbucks" → "Cà phê").

- Thống kê & Báo cáo

- + Tổng quan chi tiêu:

- Biểu đồ cột so sánh thu/chi theo tháng.

- Xu hướng chi tiêu 6 tháng gần nhất.

- + Xuất dữ liệu.

- Quản lý ngân sách

- + Thiết lập ngân sách:

- Theo danh mục (VD: 2 triệu/tháng cho "Ăn uống").

- Cảnh báo khi chi tiêu vượt 80% ngân sách.

- + Theo dõi tiến độ: Hiển thị % đã sử dụng bằng progress bar.

- Mục tiêu tài chính

- + Tạo mục tiêu. Ví dụ: Tiết kiệm 10 triệu trong 3 tháng.

- + Tính toán tự động: Dự đoán thời gian đạt mục tiêu dựa trên thu nhập/chi tiêu hiện tại.

- + Hiển thị % hoàn thành bằng progress bar, kèm gợi ý hành vi chi tiêu.

- Tìm kiếm & Lọc

- + Lọc giao dịch: Theo khoảng thời gian, danh mục, khoảng tiền.

- + Tìm kiếm nhanh: Nhập từ khóa (VD: "tiền điện") để lọc giao dịch liên quan.

- Trợ lý tài chính AI (AI Assistant):

- + Cho phép giao tiếp với Chatbox để giảm thời gian nhập liệu và xử lý các giao dịch.

### **3.2.2 Các yêu cầu phi chức năng**

- Hiệu suất (Performance)

- + Tốc độ phản hồi: Load trang dashboard < 1.5s, API xử lý < 500ms.

- Bảo mật (Security)

- + Mã hóa dữ liệu: Mật khẩu hash (bcrypt), HTTPS cho mọi kết nối.

- + Bảo vệ thông tin: Không lưu số tài khoản ngân hàng, chỉ lưu 4 số cuối.

- Khả năng mở rộng (Scalability)

- + Kiến trúc backend: Sử dụng REST API, dễ dàng tích hợp thêm dịch vụ ngân hàng.

+ MongoDB: Lưu giao dịch với schema linh hoạt (VD: { amount, category, date, user\_id }).

- Tính khả dụng (Availability)

+ Uptime: Đảm bảo 99.5% thời gian hoạt động.

- Giao diện người dùng (Usability)

Thiết kế:

+ Giao diện tối giản, hỗ trợ dark mode.

+ Hiển thị biểu đồ trực quan (Pie chart, Line chart).

+ Đa nền tảng: Web responsive

- Khả năng bảo trì (Maintainability)

+ Tài liệu: Swagger cho API.

+ Logging: Ghi log lỗi vào hệ thống Sentry để debug.

### 3.3 Quản lý dự án

#### 3.3.1 Kế hoạch để phân tích và thiết kế giao diện

Dựa trên nhu cầu người dùng, các thành viên trong nhóm đã lên kế hoạch như sau:

- SCRUM-11: Phân tích các yêu cầu về chức năng & phi chức năng của Website

Công việc thực hiện:

+ Khảo sát nhu cầu thực tế của người dùng

+ Liệt kê, mô tả các chức năng hệ thống cần có (đăng ký, đăng nhập, quản lý giao dịch, thống kê, báo cáo...)

+ Phân tích các yêu cầu phi chức năng: bảo mật, hiệu năng, giao diện người dùng, khả năng mở rộng...

+ Viết tài liệu yêu cầu phần mềm (SRS) hoặc ghi vào Confluence/Jira

Kết quả: Nhóm trưởng hoặc người phụ trách tập hợp ý kiến, mô tả cụ thể yêu cầu cho các nhóm phát triển bám sát.

- SCRUM-6: Thiết kế CSDL (Cơ sở dữ liệu)

Công việc thực hiện:

+ Xác định các thực thể chính (user, transaction, category,...)

+ Vẽ sơ đồ quan hệ (ERD) hoặc schema cho MongoDB/PostgreSQL

+ Định nghĩa các trường, kiểu dữ liệu, khóa chính/phụ

- + Xây dựng tài liệu hướng dẫn thiết kế CSDL

Kết quả: Có file schema hoặc bản thiết kế sẵn sàng để các dev backend bắt đầu code chức năng.

- SCRUM-12: Thiết kế giao diện UI

Công việc thực hiện:

- + Dựng wireframe, mockup (bản vẽ/phác giao diện màn hình)

- + Chọn bảng màu, phong cách, bố cục UX/UI

- + Dùng Figma hoặc công cụ tương đương để minh họa từng màn hình (đăng nhập, dashboard, thêm giao dịch...)

- + Thông nhất thiết kế với cả nhóm trước khi lập trình frontend

Kết quả: Có file thiết kế giao diện, hướng dẫn chi tiết cho frontend lập trình đồng nhất.

- SCRUM-13: Setup môi trường làm việc (Jira, Github, Visual Studio Code)

Công việc thực hiện:

- + Tạo repo trên GitHub, phân quyền thành viên

- + Cấu hình Jira/Trello để quản lý công việc

- + Cài đặt các extension, plugin hỗ trợ code, quản lý task

- + Thông nhất các tiêu chuẩn commit, quy ước làm việc nhóm

Kết quả: Mọi người có đầy đủ tools, môi trường nhất quán để bắt đầu code và quản lý tiến độ.

- SCRUM-17: Tạo Swagger API draft

Công việc thực hiện:

- + Thiết kế các endpoint (API đường dẫn – ví dụ /api/user, /api/transaction,...)

- + Viết mô tả API chuẩn OpenAPI/Swagger cho các chức năng chính (input/output)

- + Triển khai Swagger UI (nếu có) để dễ test, kiểm thử

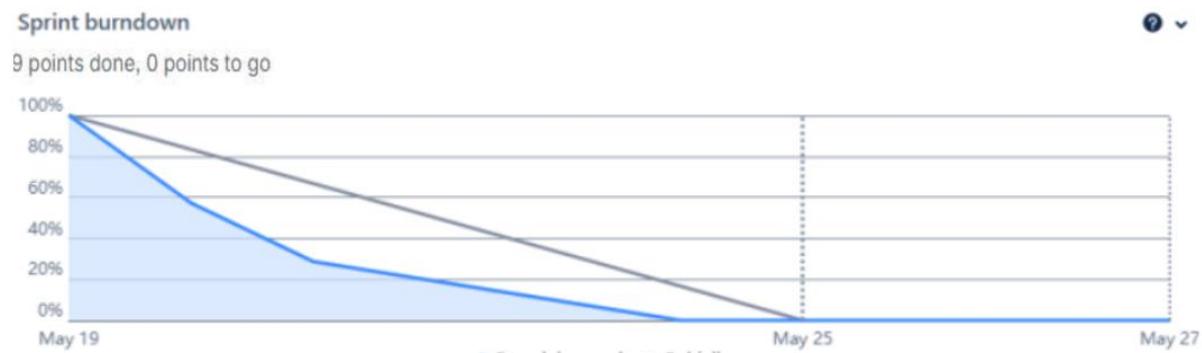
- + Chia sẻ tài liệu API cho toàn nhóm

Kết quả: Các dev frontend & backend thông nhất cách gọi API ngay từ đầu, tránh sai sót.

<input checked="" type="checkbox"/> SCRUM-11 Phân tích các yêu cầu về chức năng & phi chức năng của Website	PHÁT TRIỂN UI CHO W...	DONE	1	TN
<input type="checkbox"/> SCRUM-6 Thiết kế CSDL	HOÀN THIỆN CÁC THI...	DONE	3	TH
<input type="checkbox"/> SCRUM-12 Thiết kế giao diện UI	PHÁT TRIỂN UI CHO W...	DONE	2	KH
<input checked="" type="checkbox"/> SCRUM-13 Setup môi trường làm việc (Jira, Github, Visual Studio Code)	HOÀN THIỆN CÁC THI...	DONE	1	KH
<input checked="" type="checkbox"/> SCRUM-17 Tạo Swagger API draft	XÂY DỰNG BACKEND ...	DONE	2	TH

Hình 3.1 Các Story và task của Sprint 1

Biểu đồ Sprint Burndown Chart của lập kế hoạch “Phân tích và thiết kế giao diện”.



Hình 3.2 Biểu đồ Burndown của Sprint 1

+ Biểu đồ cho thấy nhóm đã hoàn thành tất cả công việc 9 point đều được hoàn thành, nhóm đã hoàn thành trước thời gian đặt ra.

### 3.3.2 Kế hoạch xây dựng API

SCRUM-15: API Đăng ký và Đăng nhập

Công việc chi tiết:

+ Thiết kế endpoint /register (POST) với các trường: email, password, name, phone.

+ Thiết kế endpoint /login (POST) với xác thực JWT.

+ Validate dữ liệu đầu vào (regex email, độ dài password).

+ Mã hóa password bằng bcrypt.

+ Xử lý lỗi trùng email, thông tin đăng nhập sai.

Kết quả: API trả về token JWT khi thành công, HTTP status code rõ ràng.

- SCRUM-16: API CRUD khoản thu/chi

Công việc chi tiết:

+ Tạo model Transaction với các trường: amount, category, date, description.

+ Xây dựng các endpoint:

- GET /transactions: Lọc theo tháng/category.
- POST /transactions: Thêm giao dịch mới.
- PUT /transactions/:id: Cập nhật giao dịch.
- DELETE /transactions/:id: Xóa giao dịch.
- Phân quyền (chỉ user sở hữu được sửa/xóa).

Kết quả: API đầy đủ chức năng, tích hợp với CSDL.

- SCRUM-18: Cập nhật tài liệu Swagger

Công việc chi tiết:

- + Mô tả chi tiết request/response cho từng API.
- + Thêm ví dụ payload (JSON) cho POST/PUT.
- + Triển khai Swagger UI tại /api-docs.

Kết quả: Tài liệu tự động cập nhật khi deploy.

- SCRUM-19: Kiểm thử API bằng Postman

Công việc chi tiết:

- + Tạo collection Postman với các test case:
- + Test đăng ký với email không hợp lệ.
- + Test CRUD transaction thiếu token.
- + Thiết lập automated test trong GitHub Actions.

Kết quả: 100% API pass test, coverage > 90%.

- SCRUM-47/48/49:

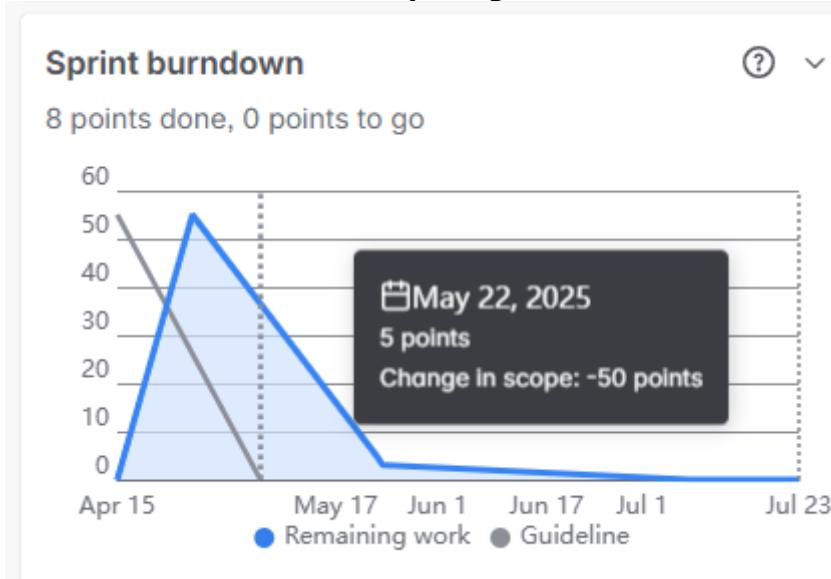
Bổ sung các thành phần sau:

- + Ghi log lỗi vào file error.log (SCRUM-47).
- + API /login trả về thông tin user kèm token (SCRUM-48).
- + API /register gửi email xác nhận (SCRUM-49).



Hình 3.3 Các Story và task của Sprint 2

Biểu đồ BurnChart của kế hoạch xây dựng API:



Hình 3.4 Burnchart của kế hoạch xây dựng API

+ Từ biểu đồ trên cho thấy, đây là sprint mà nhóm đã chưa hoàn thành đúng thời gian đặt ra, do lần đầu tiếp xúc với quản lí dự án nên nhóm đã đặt ra thời gian hoàn thành chưa được hợp lí, và với một phần phải làm song song và thời gian hoàn thành báo cáo được tăng lên nên nhóm vẫn hoàn thành hết các công việc.

### 3.3.3 Lập kế hoạch phát triển giao diện người dùng

- SCRUM-22: Giao diện đăng nhập

Công việc chi tiết:

+ Thiết kế form với validation real-time:

- Hiển thị lỗi khi email sai định dạng.
- Nút "Hiển thị mật khẩu".

+ Xử lý redirect sau khi login thành công.

Kết quả: Giao diện responsive, tích hợp API.

- SCRUM-39: Dashboard tổng quan

Công việc chi tiết:

+ Hiển thị các card thống kê:

- Tổng thu/chi tháng hiện tại.
- So sánh với tháng trước.

+ Biểu đồ đường xu hướng chi tiêu.

Kết quả: Data được lấy từ API thực tế.

- SCRUM-44/45: Layout & Mock data

Bổ sung:

- Sử dụng thư viện Material-UI cho layout.
- Mock data bằng JSON server để test UI độc lập.

- SCRUM-41/42: Giao diện ngân sách

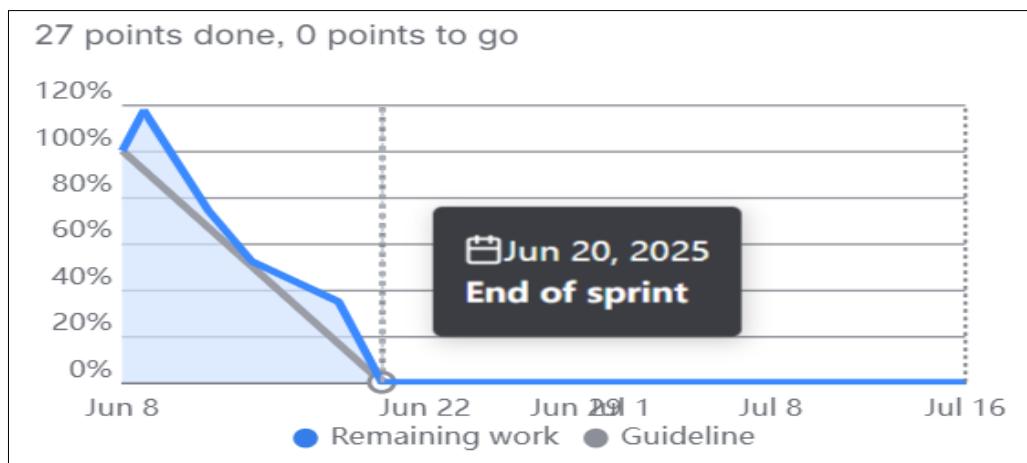
Chi tiết:

- + Progress bar hiển thị % sử dụng ngân sách.
- + Form thêm/sửa ngân sách với dropdown category.

Phát triển giao diện UI   6 May – 27 May (9 work items)			
		0   0   12	Complete sprint
Tiến hành thiết kế và phát triển các giao diện như: - Giao diện Đăng nhập. - Giao diện Trang chủ (tổng quan). - Giao diện Quản lý tài khoản... - Routing các trang, Layout tổng quát...			
<input type="checkbox"/> SCRUM-22	Giao diện đăng nhập	PHÁT TRIỂN UI CHO W...	DONE ✓
<input type="checkbox"/> SCRUM-39	Giao diện Trang chủ (dashboard tổng quan)	PHÁT TRIỂN UI CHO W...	DONE ✓
<input checked="" type="checkbox"/> SCRUM-44	Tạo layout tổng quát	PHÁT TRIỂN UI CHO W...	DONE ✓
<input checked="" type="checkbox"/> SCRUM-45	Gắn dữ liệu mock	PHÁT TRIỂN UI CHO W...	DONE ✓
<input type="checkbox"/> SCRUM-40	Giao diện Quản lý tài khoản/thẻ	PHÁT TRIỂN UI CHO W...	DONE ✓
<input type="checkbox"/> SCRUM-44	Giao diện Mục tiêu tài chính	PHÁT TRIỂN UI CHO W...	DONE ✓
<input type="checkbox"/> SCRUM-42	Giao diện Ngân sách	PHÁT TRIỂN UI CHO W...	DONE ✓
<input checked="" type="checkbox"/> SCRUM-43	Routing giữa các trang	PHÁT TRIỂN UI CHO W...	DONE ✓
<input checked="" type="checkbox"/> SCRUM-46	Tạo file component	PHÁT TRIỂN UI CHO W...	DONE ✓

Hình 3.5 Các Story và task của Sprint 3

+ Biểu đồ Burn Chart:



Hình 3.6 Burnchart của kế hoạch phát triển giao diện UI

+ Từ biểu đồ cho thấy nhóm đã hoàn thành khá tốt quản lí các công việc, việc nhóm giữa tìm hiểu các công nghệ mới và các phần mềm mới khiến nhóm vẫn còn khá chậm trong công việc, tuy nhiên vẫn đảm bảo hoàn thành công việc đúng thời gian đã đặt ra.

### 3.3.4 Lập kế hoạch "Phân tích & quản lý chi tiêu"

- SCRUM-26/27: Quản lý ngân sách (IN PROGRESS)

- Công việc đang triển khai:

- + Tính toán % tiêu vượt ngân sách theo công thức:

$$\% = (\text{Tổng chi thực tế} / \text{Ngân sách}) * 100 \quad (1)$$

- + Hiển thị cảnh báo màu đỏ khi vượt > 100%.

- SCRUM-28/29: Biểu đồ (DONE)

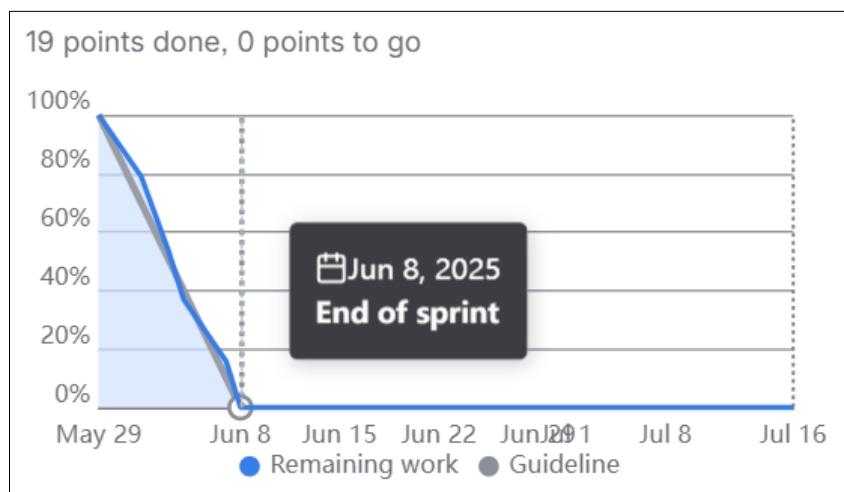
- + Sử dụng Chart.js vẽ biểu đồ cột so sánh 6 tháng.

- + Fix lỗi reload biểu đồ bằng cách cache data.

Tên Task	Trạng thái	Số lượng	Mô tả
SCRUM-24 Giao diện thống kê theo thời gian	DONE	1	
SCRUM-25 Biểu diễn xu hướng chi tiêu	DONE	1	
SCRUM-26 Giao diện quản lý ngân sách	IN PROGRESS	1	KH
SCRUM-27 Hiển thị % tiêu vượt ngân sách	IN PROGRESS	1	TH
SCRUM-28 Tao biểu đồ so sánh giữa các tháng	DONE	1	TH
SCRUM-29 Biểu đồ bị vỡ khi resize màn hình	DONE	1	TH

Hình 3.7 Các Story, task, bug của Sprint 4

- Biểu đồ Sprint Burndown Chart của lập kế hoạch “ Phân tích quản lí chi tiêu”.



Hình 3.8 Burnchart của kế hoạch Phân tích chi tiêu

- + Biểu đồ này cho thấy nhóm đã hoàn thành xuất sắc Sprint với 19 điểm công việc (story points) đạt 100% và không còn tồn đọng (0 points to go). Không bị "spillover" (công việc dời sang Sprint sau) → Kế hoạch tốt, ước lượng chính xác.

- + Từ biểu đồ cho thấy nhóm đã rút kinh nghiệm hơn so với các biểu đồ chưa hoàn thành tốt, tốt độ thực hiện công việc và phân chi ổn định, biểu đồ gàn như trùng so với Guide line cho thấy nhóm đã kiểm soát công việc rất tốt..

### 3.3.5 Lập kế hoạch "Kiểm thử, CI/CD và Demo"

- SCRUMP-32: Docker hóa

- Công việc chi tiết:

- + Tạo Dockerfile cho frontend (build React) và backend (Node.js).
- + Docker Compose kết nối các service: FE, BE, MongoDB.

- SCRUMP-33: GitHub Actions

- Quy trình CI/CD:

- + Tự động chạy test khi push code.
- + Deploy lên Heroku nếu pass test.
- + Gửi notification qua Slack.

- SCRUMP-36: Kiểm thử API

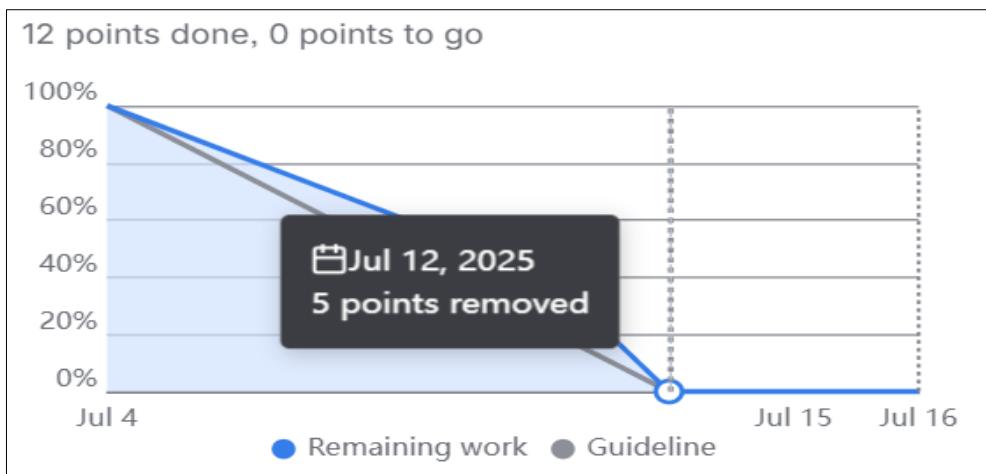
- Test cases:

- + Kiểm tra hiệu năng.

Kiểm thử, CI/CD và Demo 11 Jun – 18 Jun (6 work items)		0	2	6	Complete sprint	...
Quá trình hoàn thiện sẽ diễn ra như sau: - Tiến hành Docker hóa cho backend và frontend. - Thiết lập các CI/CD trên GitHub. - Kiểm thử lại toàn bộ API. - Viết báo cáo/Slide báo cáo và chuẩn bị báo cáo.						
<input checked="" type="checkbox"/> SCRUM-32	Docker hóa backend & frontend	CI/CD & TRIỂN KHAI	DONE	1	KH	
<input checked="" type="checkbox"/> SCRUM-33	Thiết lập CI/CD GitHub Actions	CI/CD & TRIỂN KHAI	IN PROGRESS	1	TH	
<input checked="" type="checkbox"/> SCRUM-35	Viết báo cáo	KIỂM THỬ VÀ BÁO CÁO	DONE	3	KH	
<input checked="" type="checkbox"/> SCRUM-36	Làm slide và chuẩn bị demo	KIỂM THỬ VÀ BÁO CÁO	DONE	1	TM	
<input checked="" type="checkbox"/> SCRUM-37	Kiểm thử toàn bộ API với Postman	KIỂM THỬ VÀ BÁO CÁO	IN PROGRESS	1	TH	
<input checked="" type="checkbox"/> SCRUM-38	CI/CD không tự build khi push	CI/CD & TRIỂN KHAI	DONE	1	KH	

Hình 3.9 Các Story, task, bug của Sprint 5

Biểu đồ Sprint Burndown Chart của lập kế hoạch “Kiểm thử, CI, CD và Demo”:



Hình 3.10 Biểu đồ Burnchart của Kiểm thử CI CD

Biểu đồ trên cho thấy nhóm hoàn thành vừa khớp với khoảng thời gian đã đặt ra, 12 point đều được hoàn thành tuy tốc độ công việc chưa được gần nhất với Guide line nhưng vẫn ổn định.

### 3.3.6 Bảng phân công công việc

- Công việc & Phân công chi tiết cho Kế hoạch phân tích và thiết kế giao diện:

STT	Công việc (Task)	Mô tả	Người phụ trách	Hỗ trợ	Kết quả đầu ra
1	SCRUM-11: Phân tích yêu cầu	- Khảo sát nhu cầu người dùng. - Liệt kê chức năng (đăng nhập, giao dịch...). - Phân tích yêu cầu phi chức năng (bảo mật, UX...).	Nguyễn Thanh Hiếu	Phạm Hoàng Kha	Tài liệu SRS/Confluence
2	SCRUM-6: Thiết kế CSDL	- Xác định thực thể (user, transaction...).	Nguyễn Trí Cường	Nguyễn Thanh Hiếu	Sơ đồ ERD/Schema MongoDB/PostgreSQL
3	SCRUM-12: Thiết kế UI/UX	- Dựng wireframe, mockup bằng Figma. - Chọn bảng màu, bố cục UX.	Phạm Hoàng Kha	Nguyễn Trí Cường	File thiết kế Figma
4	SCRUM-13: Setup môi trường	- Tạo repo GitHub, cấu hình Jira.	Nguyễn Trí Cường	Cả nhóm	Repo GitHub, Jira ready

STT	Công việc (Task)	Mô tả	Người phụ trách	Hỗ trợ	Kết quả đầu ra
		- Cài đặt VSCode extensions, tiêu chuẩn commit.			
5	SCRUM-17: Thiết kế API (Swagger)	- Thiết kế endpoint (e.g., /api/user). -- Viết mô tả input/output API.	Nguyễn Thanh Hiếu	Phạm Hoàng Kha	Tài liệu Swagger/Open API

Bảng 1 Công việc & Phân công chi tiết cho phân tích và thiết kế giao diện

- Công việc & Phân công chi tiết cho kế hoạch xây dựng API:

STT	Công việc (Task)	Mô tả	Người phụ trách	Hỗ trợ	Kết quả đầu ra
1	SCRUM-15: API Đăng ký/Đăng nhập	- Thiết kế endpoint / register, /login với JWT. - Validate email/password, mã hóa bcrypt. - Xử lý lỗi trùng email, thông tin sai.	Nguyễn Thanh Hiếu	Phạm Hoàng Kha	API trả về token, status code chuẩn
2	SCRUM-16: API CRUD giao dịch	- Tạo model Transaction (amount, category...). - Xây dựng endpoints GET/POST/PUT/DELETE. - Lọc theo tháng/category, kiểm tra quyền sở hữu.	Nguyễn Trí Cường	Cả nhóm	API CRUD hoàn chỉnh, phân quyền user
3	SCRUM-18: Cập nhật Swagger	- Mô tả request/response cho từng API. - Thêm ví dụ JSON payload.	Phạm Hoàng Kha	Nguyễn Thanh Hiếu	Swagger UI tại /api-docs
4	SCRUM-19: Kiểm thử Postman	- Tạo collection test case (invalid email, thiếu token...). - Tích hợp automated test với GitHub Actions.	Nguyễn Trí Cường	Nguyễn Thanh Hiếu	100% API pass test, coverage > 90%
5	SCRUM-47/48/49: Bổ sung	- Ghi log lỗi vào error.log (SCRUM-47). - API /login trả về thông tin user + token (SCRUM-48).	Phạm Hoàng Kha	Nguyễn Trí Cường	Log hệ thống, email xác nhận

STT	Công việc (Task)	Mô tả	Người phụ trách	Hỗ trợ	Kết quả đầu ra
		- API /register gửi email xác nhận (SCRUM-49).			

Bảng 2 Công việc & Phân công chi tiết cho kế hoạch xây dựng API

- Công việc & Phân tích chi tiêu cho phát triển giao diện người dùng

STT	Công việc (Task)	Mô tả	Người phụ trách	Hỗ trợ	Kết quả đầu ra
1	SCRUM-22: Giao diện đăng nhập	- Thiết kế form đăng nhập với validation real-time - Xử lý redirect sau login	Nguyễn Trí Cường	Phạm Hoàng Kha	Giao diện responsive, tích hợp API
2	SCRUM-39: Dashboard tổng quan	- Thiết kế card thống kê - Xây dựng biểu đồ xu hướng chi tiêu	Phạm Hoàng Kha	Nguyễn Thanh Hiếu	Dashboard hiển thị data từ API
3	SCRUM-44/45: Layout & Mock data	- Sử dụng Material-UI - Tạo mock data bằng JSON server	Nguyễn Thanh Hiếu	Cả nhóm	Hệ thống layout thống nhất, mock data
4	SCRUM-41/42: Giao diện ngân sách	- Progress bar % ngân sách - Form thêm/sửa ngân sách	Phạm Hoàng Kha	Nguyễn Trí Cường	Giao diện quản lý ngân sách

Bảng 3.Công việc & Phân tích chi tiêu cho phát triển giao diện người dùng

- Công việc & phân công chi tiết cho phân tích và quản lí chi tiêu.

STT	Công Việc (Task)	Mô Tả	Người Phụ Trách	Hỗ Trợ	Kết Quả Đầu Ra
1	SCRUM-26/27: Quản lý ngân sách	- Tính toán % vượt ngân sách - Hiển thị cảnh báo màu đỏ khi >100%	Nguyễn Thanh Hiếu	Phạm Hoàng Kha	Tính năng cảnh báo ngân sách

STT	Công Việc (Task)	Mô Tả	Người Phụ Trách	Hỗ Trợ	Kết Quả Đầu Ra
2	SCRUM-28/29: Biểu đồ thống kê	- Vẽ biểu đồ cột 6 tháng bằng Chart.js - Fix lỗi reload bằng cache data	Nguyễn Trí Cường	Cả nhóm	Biểu đồ mượt, ổn định

Bảng 4 Công việc & phân công chi tiết cho phân tích và quản lý chi tiêu.

- Công việc và phân công chi tiết cho kế hoạch kiểm thử CI/ CD và Demo:

STT	Công Việc (Task)	Mô Tả	Người Phụ Trách	Hỗ Trợ	Kết Quả Đầu Ra
1	SCRUM-32: Docker hóa	- Tạo Dockerfile cho FE (React) và BE (Node.js) - Cấu hình Docker Compose kết nối FE, BE, MongoDB	Nguyễn Trí Cường	Phạm Hoàng Kha	Hệ thống chạy ổn định trên môi trường container
2	SCRUM-33: GitHub Actions CI/CD	- Tự động chạy test khi push code - Deploy lên Heroku nếu pass test - Gửi notification Slack	Nguyễn Thanh Hiếu	Cả nhóm	Quy trình CI/CD tự động, thông báo real-time
3	SCRUM-36: Kiểm thử API	- Kiểm tra hiệu năng API - Viết test case cho các endpoint chính	Phạm Hoàng Kha	Nguyễn Thanh Hiếu	Báo cáo hiệu năng, test coverage > 85%

Bảng 5 Công việc và phân công chi tiết cho kế hoạch kiểm thử CI/ CD và Demo

## CHƯƠNG 4: THIẾT KẾ HỆ THỐNG

### 4.1 Kiến trúc tổng thể của hệ thống

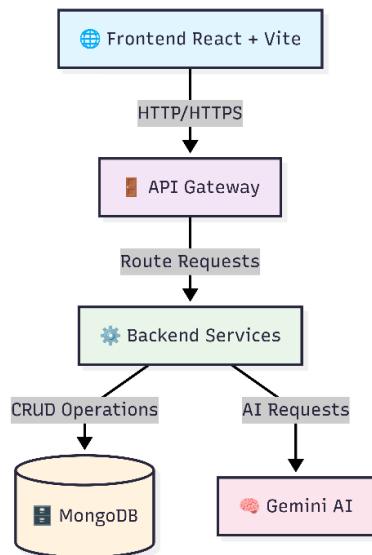
Hệ thống của bạn sử dụng kiến trúc Client-Server với sự phân tách rõ ràng giữa frontend (React) và backend (Node.js + Express), kết hợp các dịch vụ monolithic.

Các thành phần chính:

- + Client: Ứng dụng công nghệ React.js để hiển thị giao diện, xử lý các tương tác người dùng như thêm giao dịch, xem báo cáo.
- + API Gateway: Sử dụng Node.js để định tuyến request, xử lý CORS, authentication, logging.
- + Server: Sử dụng Node.js monolithic để xử lý logic nghiệp vụ (Auth, User, Transaction, Goal...).
- + Database: Sử dụng hệ quản trị cơ sở dữ liệu MongoDB để lưu trữ các loại dữ liệu có cấu trúc (users) và kể cả các dữ liệu phi cấu trúc (transactions)
- + Cache: Sử dụng redis để tăng tốc độ truy cập dữ liệu thông kê thường xuyên hơn.

Sơ đồ kiến trúc hệ thống:

Data Flow Diagram - Expense Management System



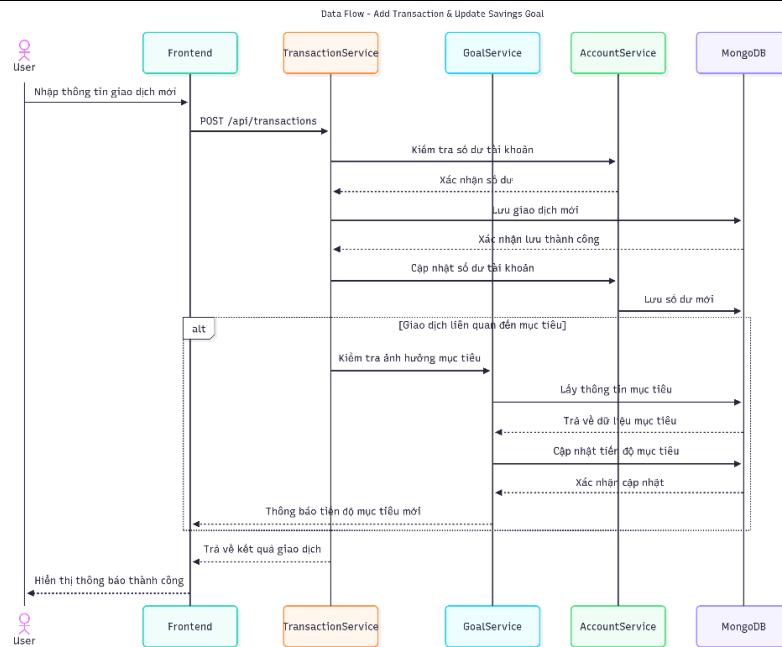
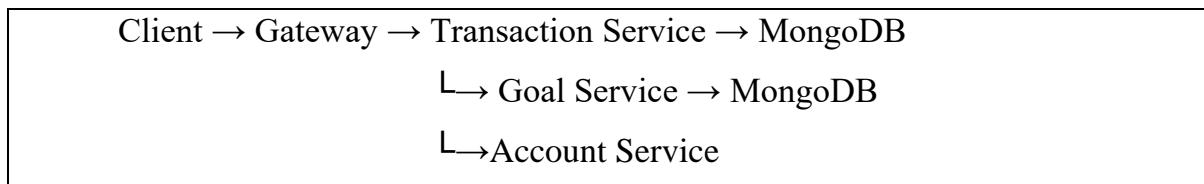
Hình 4.1 Sơ đồ kiến trúc hệ thống

Các luồng giao dịch điển hình:

Thêm giao dịch mới và cập nhật mục tiêu tiết kiệm:

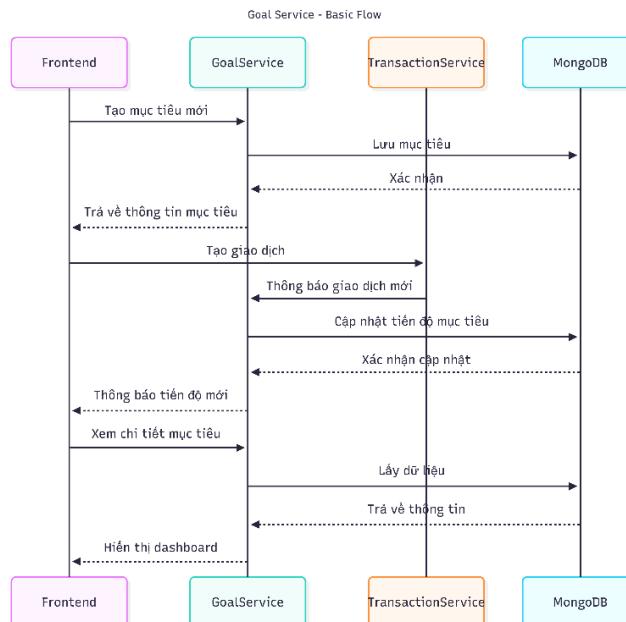
- Key Points:

+ Data Flow:



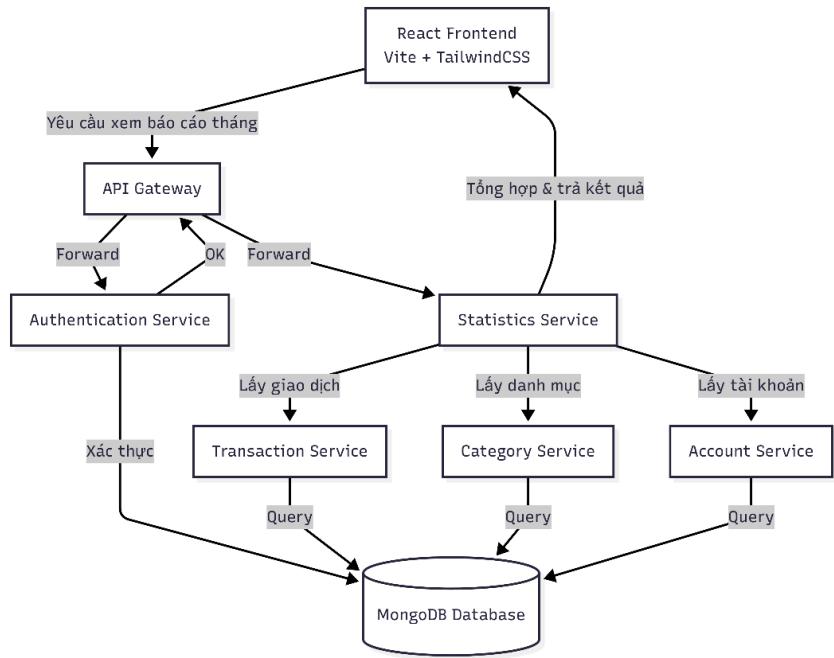
Hình 4.2 Luồng thêm giao dịch mới và cập nhật mục tiêu

- Tương tác Goal Service



Hình 4.3 Luồng tương tác Goal Service

- Luồng xem báo cáo tháng: giúp hiển thị tổng quan danh mục.

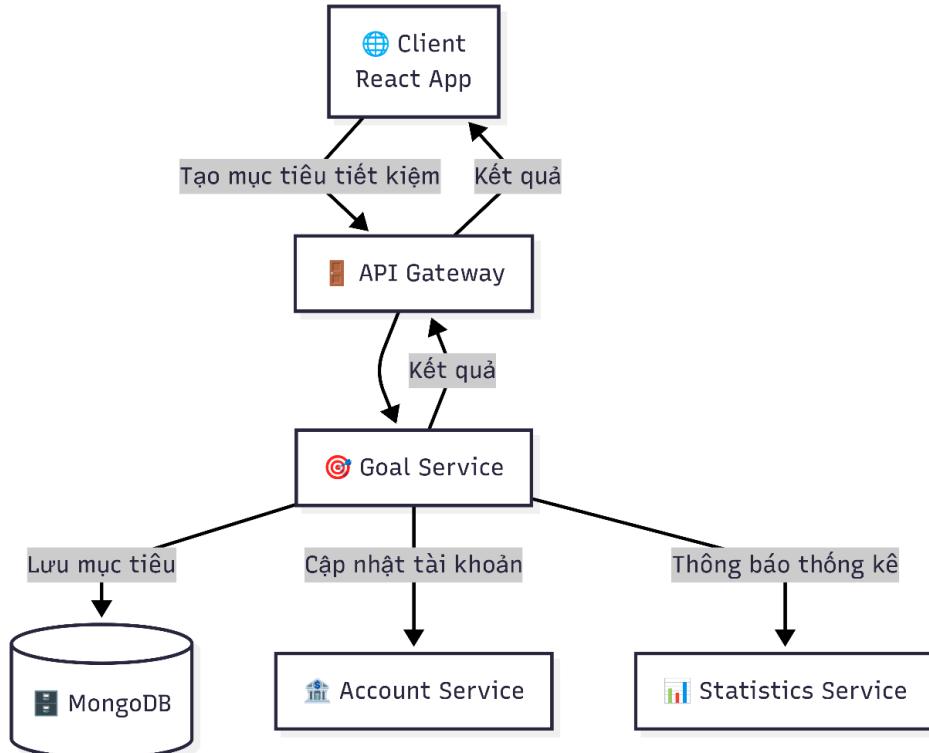


Hình 4.4 Luồng xem báo cáo tháng

- Luồng Tạo Mục Tiêu Tiết Kiệm: giúp người dùng đặt mục tiêu tiết kiệm, hệ thống liên kết với các giao dịch tương ứng.

+ Goal Service kiểm tra các giao dịch có category="Tiết kiệm" trong 30 ngày gần nhất.

+ Tự động cộng dồn vào currentAmount của goal mới.



#### *Hình 4.5 Luồng tạo mục tiêu tiết kiệm*

##### **Đặc điểm kiến trúc:**

Client (React):

- Single Page Application (SPA):

- + Tải 1 lần duy nhất, giao tiếp với server qua REST API.

- + Sử dụng Axios để gọi API, Redux quản lý state.

- Tối ưu hiệu năng:

- + Code-splitting (React.lazy) để giảm thời gian tải trang.

- + Service Worker (PWA) hỗ trợ offline mode.

Server (Node.js):

- API Gateway:

- + Xác thực tập trung: Kiểm tra JWT trước khi định tuyến.

- + Load Balancing: Phân phối request đến các service.

- Services:

- + Auth Service: Đăng nhập, xác thực JWT.

- + Transaction Service: CRUD giao dịch, phân loại tự động.

- + Goal Service: Theo dõi mục tiêu tiết kiệm

- + Statistics Service: Tổng hợp báo cáo

Database:

- MongoDB:

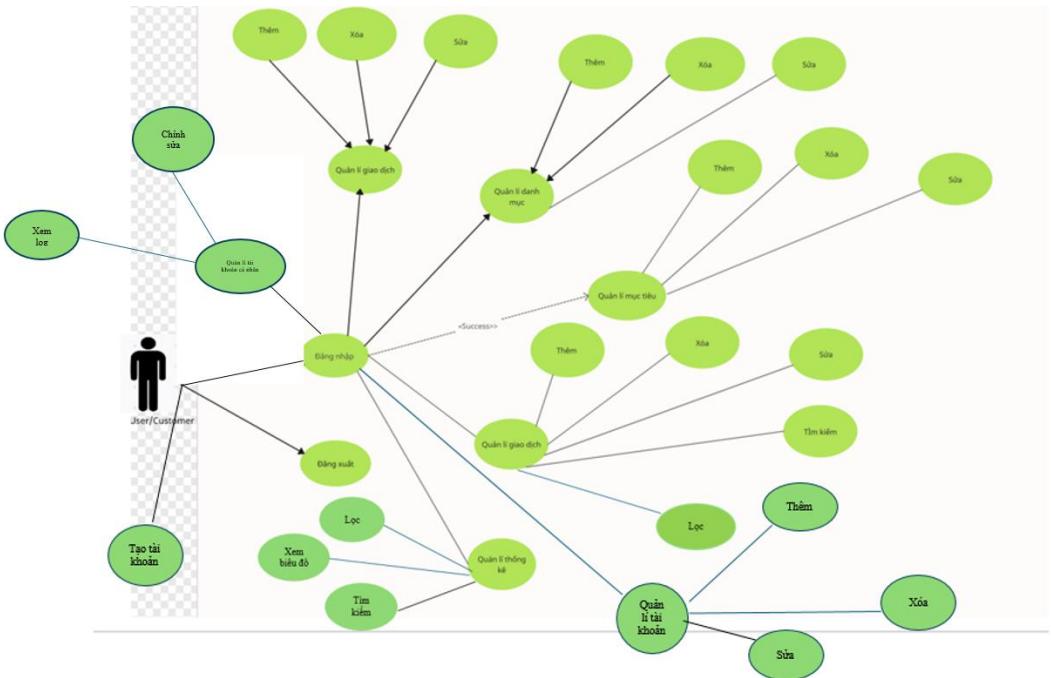
- + Ưu điểm: Linh hoạt với dữ liệu giao dịch (amount, category, date).

- + Document ví dụ:

```
{  
  "_id": "txn_123",  
  "user_id": "user_456",  
  "amount": 500000,  
  "category": "Ăn uống",  
  "date": "2023-10-20T00:00:00Z"  
}
```

## **4.2 Thiết kế cơ sở dữ liệu cho hệ thống**

### **4.2.1 Sơ đồ Use case**



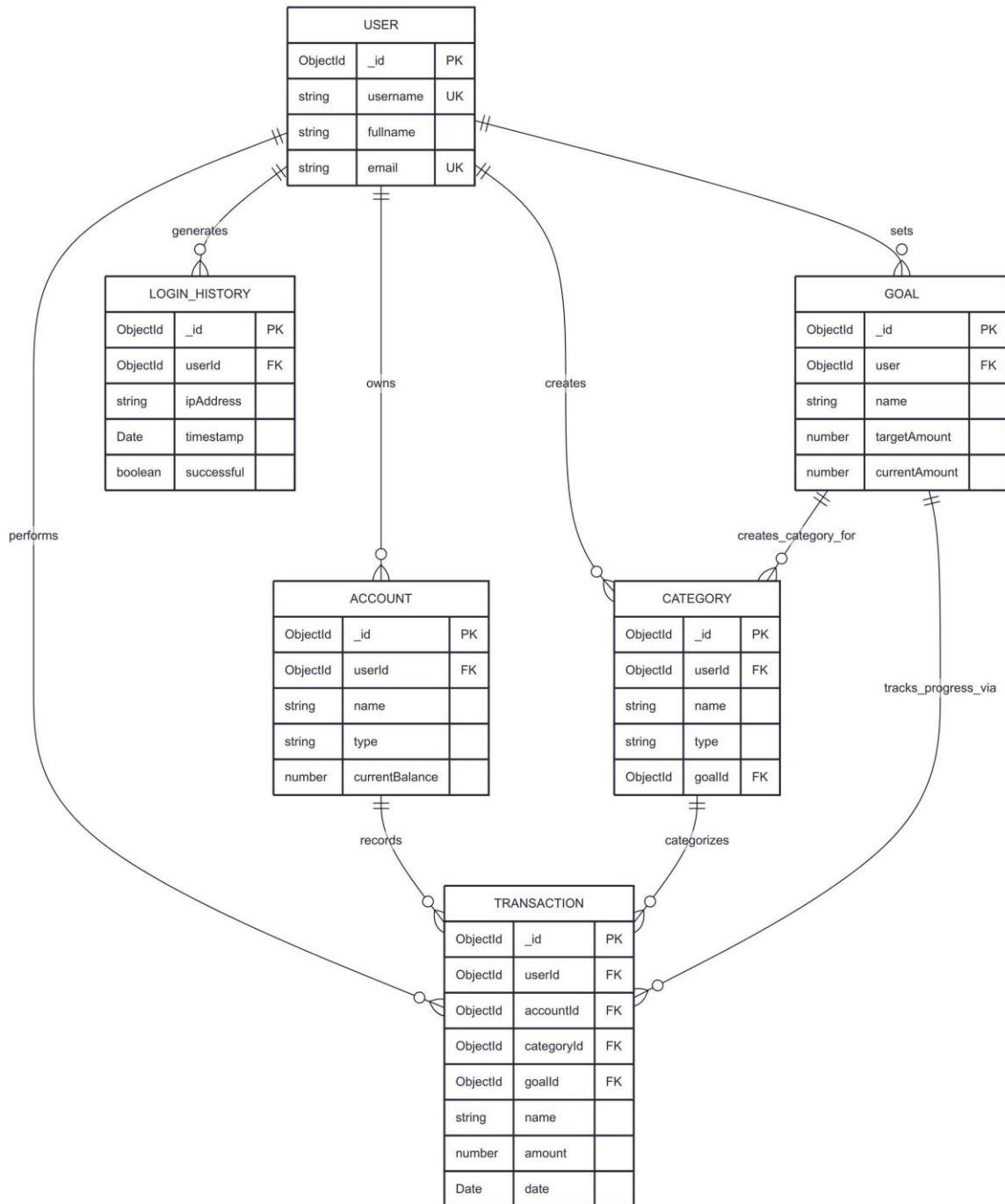
Hình 4.6 Sơ đồ Use case của hệ thống

### Các chức năng (Use Cases):

- Đăng ký tài khoản: Cho phép người dùng tạo tài khoản mới để bắt đầu sử dụng hệ thống.
- Đăng nhập / Đăng xuất: Xác thực người dùng để truy cập hệ thống, và thoát phiên khi cần.
- Quản lý giao dịch thu/chi
  - + Thêm giao dịch (thu hoặc chi)
  - + Chính sửa giao dịch
  - + Xóa giao dịch
  - + Tìm kiếm và lọc giao dịch theo ngày, danh mục, loại (thu/chí)
- Quản lý danh mục thu/chi
  - + Thêm danh mục mới (Ăn uống, Di chuyển, Học tập...)
  - + Cập nhật hoặc xóa danh mục
  - + Thông kê và báo cáo chi tiêu
  - + Xem tổng thu, tổng chi
  - + Hiển thị biểu đồ chi tiêu theo tháng, theo danh mục
  - + So sánh thu – chi qua từng thời kỳ
- Thiết lập mục tiêu tài chính (tùy chọn)
  - + Đặt mục tiêu tiết kiệm

- + Cảnh báo khi vượt hạn mức chi
- Quản lý tài khoản cá nhân
- + Cập nhật thông tin người dùng
- + Đổi mật khẩu

#### 4.2.2 Mô hình quan hệ dữ liệu của hệ thống



Hình 4.7 Mô hình ERD của hệ thống

## 4.3 Thiết kế API cho hệ thống

Hệ thống cung cấp RESTful API theo chuẩn OpenAPI 3.0, tài liệu hóa bằng Swagger, dưới đây là bảng mô tả tổng quan của từng API:

### 4.3.1 Xác thực và phân quyền (auth):

Phương thức	Endpoint	Mô tả
POST	/api/auth/register	Đăng ký tài khoản mới
POST	/api/auth/login	Đăng nhập
GET	/api/auth/me	Lấy thông tin từ token

Bảng 6 Các phương thức và Endpoint của xác thực và phân quyền

#### 1. POST /api/auth/register <a id="register"></a>

**Chức năng:** Tạo một tài khoản người dùng mới.

+Request (phải có required):

```
{  
    "username": "string",  
    "fullname": "string",  
    "password": "string"  
}
```

- username: tên đăng nhập
- fullname: tên đầy đủ
- password: mật khẩu

+Response: **201** – "Đăng ký thành công": nghĩa là người dùng đã được tạo.

```
{  
    "user": {  
        "_id": "65a1bc...",  
        "email": "user@example.com",  
        "name": "Nguyễn Văn A",  
        "avatar": null  
    },  
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}
```

Các lỗi thường gặp:

- + 400 Bad Request: Thiếu trường bắt buộc
- + 409 Conflict: Email đã tồn tại

+ 422 Unprocessable Entity: Password yếu (dưới 8 ký tự)

## 2. POST /api/auth/login <a id="login"></a>

**Chức năng:** Đăng nhập bằng tài khoản đã đăng ký.

+Request:

```
{  
    "username": "string",  
    "password": "string"  
}
```

+Response : **200** – "Đăng nhập thành công": thường trả về **access token** (JWT token) để sử dụng cho các API yêu cầu xác thực.

```
{  
    "user": {  
        "_id": "65a1bc...",  
        "email": "user@example.com",  
        "name": "Nguyễn Văn A",  
        "avatar": null  
    },  
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}
```

Các lỗi phổ biến:

+ 401 Unauthorized: Sai email/mật khẩu

+ 403 Forbidden: Tài khoản bị khóa

## 3. GET /api/auth/me <a id="me"></a>

**Chức năng:** Lấy thông tin người dùng dựa vào token gửi trong header (Authorization: Bearer <token>).

**Yêu cầu:** Gửi access token đã nhận được sau khi đăng nhập.

Header:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Response (Thành công - 200): Trả về thông tin của người dùng hiện tại như:

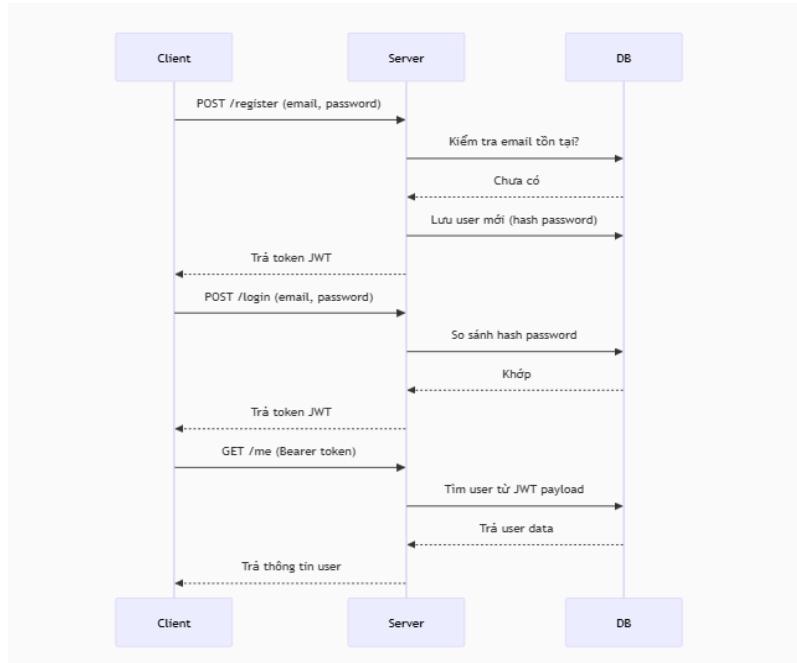
```
{  
    "_id": "65a1bc...",  
    "email": "user@example.com",  
    "name": "Nguyễn Văn A",  
    "avatar": "https://cdn.example.com/avatars/u123.jpg",  
    "createdAt": "2023-10-20T03:24:00.000Z"
```

}

Lỗi phổ biến:

- + 401 Unauthorized: Token không hợp lệ/hết hạn
- + 404 Not Found: User không tồn tại

Sơ đồ luồng xác thực:



Hình 4.7 Luồng xác thực

#### 4.3.2 Quản lý thông tin người dùng (Users)

Phương thức	Endpoint	Mô tả
GET	/api/users/profile	Lấy thông tin hồ sơ người dùng
PUT	/api/users/profile	Cập nhật thông tin hồ sơ người dùng
PUT	/api/users/avatar	Cập nhật avatar người dùng
PUT	/api/users/change-password	Đổi mật khẩu người dùng
GET	/api/users/login-history	Lấy lịch sử đăng nhập của người dùng
DELETE	/api/users/me	Xóa tài khoản người dùng hiện tại

Bảng 7 Các phương thức và Endpoint của quản lý người dùng

**Xác thực:** Tất cả API đều yêu cầu header Authorization: Bearer <token>

1.GET /api/users/profile

- Chức năng: Lấy thông tin cá nhân của người dùng đang đăng nhập (tên, email, số điện thoại, avatar, v.v.).

- Response Thành Công (200 OK):

- + Trả về thông tin chi tiết của người dùng dưới dạng JSON.

- + Hành động tiếp theo: Hiển thị thông tin trên giao diện hồ sơ cá nhân.

- Lỗi có thể xảy ra: 401 Unauthorized:

- + Nguyên nhân: Người dùng chưa đăng nhập hoặc token hết hạn.

- + Xử lý: Yêu cầu đăng nhập lại.

```
{  
  "status": "error",  
  "message": "Unauthorized - Token không hợp lệ hoặc hết hạn"  
}
```

2. PUT /api/users/profile

- Chức năng: Cập nhật thông tin cá nhân (tên, số điện thoại, email, v.v.).

- Request Body:

```
{  
  "fullname": "string",  
  "email": "user@example.com"  
}
```

- Response Thành Công (200 OK):

- + Hiển thị thông báo "Cập nhật thành công".

- + Cập nhật lại giao diện với dữ liệu mới.

```
{  
  "_id": "string",  
  "fullname": "Nguyễn Văn A",  
  "username": "nguyenvana",  
  "email": "nguyenvana@email.com",  
  "avatar": "/uploads/avatars/avatar.jpg",  
  "createdAt": "2025-07-21T16:20:54.936Z"  
}
```

Lỗi có thể xảy ra: 409 Conflict:

- + Nguyên nhân: Email mới đã tồn tại trong hệ thống.

- + Xử lý: Yêu cầu chọn email khác.

### 3. PUT /api/users/avatar

- Chức năng: Thay đổi avatar người dùng (upload ảnh mới).
- Request: Gửi file ảnh qua multipart/form-data.
- Response Thành Công (200 OK):
  - + Hiển thị ảnh mới trên giao diện.
  - + Thông báo "Đổi avatar thành công".

```
{  
  "_id": "string",  
  "fullname": "Nguyễn Văn A",  
  "username": "nguyenvana",  
  "email": "nguyenvana@email.com",  
  "avatar": "/uploads/avatars/avatar.jpg",  
  "createdAt": "2025-07-21T16:30:36.086Z"  
}
```

- Lỗi có thể xảy ra: 400 Bad Request
  - + Nguyên nhân: File upload không phải định dạng ảnh (JPEG/PNG) hoặc dung lượng quá lớn (>5MB).

- + Xử lý: Yêu cầu chọn file khác.

### 4. PUT /api/users/change-password

- Chức năng: Đổi mật khẩu của người dùng.

- Request Body:

```
{  
  "oldPassword": "string",  
  "newPassword": "string"  
}
```

- Response Thành Công (200 OK):

```
{  
  "status": "success",  
  "message": "Đổi mật khẩu thành công"  
}
```

- Hành động tiếp theo:
  - + Hiển thị thông báo thành công.
  - + Có thể yêu cầu đăng nhập lại.
- Lỗi có thể xảy ra: 400 Bad Request:

```
{
```

```
        "status": "error",
        "message": "Mật khẩu mới phải có ít nhất 8 ký tự"
    }
```

- + Nguyên nhân: Nhập sai mật khẩu hiện tại.
- + Xử lý: Yêu cầu nhập lại.

## 5. GET /api/users/login-history

- Chức năng: Lấy lịch sử đăng nhập (thiết bị, địa chỉ IP, thời gian).
- Response Thành Công (200 OK):

```
{
    "_id": "string",
    "userId": "string",
    "loginTime": "2025-07-21T19:11:04.132Z",
    "ipAddress": "string",
    "userAgent": "string"
}
```

## 6. GET /api/users/login-history

- Chức năng: Xóa vĩnh viễn tài khoản người dùng đang đăng nhập (kèm theo tất cả dữ liệu liên quan như bài đăng, lịch sử, v.v.).

- Request:

```
DELETE /api/users/me
Authorization: Bearer <access_token>
Content-Type: application/json
{
    "password": "currentPassword123" // Xác thực mật khẩu hiện tại
}
```

- Response Thành Công (200 OK):

```
{
    "message": "Tài khoản và toàn bộ dữ liệu đã được xóa thành công."
}
```

Lỗi có thể xảy ra:

- + **401 Unauthorized:**

```
{
    "status": "error",
    "message": "Token không hợp lệ hoặc hết hạn"
}
```

- **Nguyên nhân:** Người dùng chưa đăng nhập hoặc token hết hạn.

- **Xử lý:** Yêu cầu đăng nhập lại.

+ **403 Forbidden:**

```
{
  "status": "error",
  "message": "Mật khẩu không chính xác"
}
```

- **Nguyên nhân:** Nhập sai mật khẩu xác nhận.
- **Xử lý:** Yêu cầu nhập lại mật khẩu.

+ **404 Not Found:**

```
{
  "status": "error",
  "message": "Người dùng không tồn tại"
}
```

- **Nguyên nhân:** Tài khoản đã bị xóa trước đó hoặc không tìm thấy.
- **Xử lý:** Chuyển hướng về trang đăng ký.

### 4.3.3 Quản lý danh mục thu chi (Categories)

Phương thức	Endpoint	Mô tả
GET	/api/categories	Lấy toàn bộ danh mục của người dùng (kèm tổng số tiền theo từng danh mục)
POST	/api/categories	Tạo mới một danh mục chi tiêu
PUT	/api/categories/{id}	Cập nhật thông tin danh mục theo ID
DELETE	/api/categories/{id}	Xóa danh mục cụ thể theo ID
DELETE	/api/categories/all	Xóa toàn bộ danh mục của người dùng (yêu cầu xác nhận)

Bảng 8 Các phương thức và endpoint của quản lý thu chi

#### 1. GET /api/categories

- **Mục đích:** Lấy toàn bộ danh mục thu/chi của người dùng, kèm tổng số tiền theo từng danh mục.
- **Request:**
  - + Method: GET
  - + Headers:

**Authorization:** Bearer <access\_token>

- + Body: Không cần
- **Response thành công (200 OK):**

```
{
  "categories": [
    {
      "id": 1,
      "name": "Ăn uống",
      "type": "expense",
      "total_amount": 2000000,
      "created_at": "2023-10-01T00:00:00Z"
    },
    {
      "id": 2,
      "name": "Lương",
      "type": "income",
      "total_amount": 15000000,
      "created_at": "2023-09-15T00:00:00Z"
    }
  ]
}
```

- **Lỗi:**
  - + 401 Unauthorized: Thiếu hoặc token không hợp lệ
  - + 404 Not Found: Người dùng chưa có danh mục nào

## 2. POST /api/categories

- **Mục đích:** Tạo mới một danh mục chi tiêu
- **Request:**
  - + Method: POST
  - + Headers:

**Authorization:** Bearer <access\_token>

**Content-Type:** application/json

- + Body:

```
{
  "name": "Du lịch",
  "type": "expense"
}
```

- Response thành công (201 Created):

```
{
  "id": 3,
  "name": "Du lịch",
  "type": "expense",
  "message": "Danh mục đã được tạo thành công",
  "created_at": "2023-10-05T10:30:00Z"
}
```

- Lỗi:

- + 400 Bad Request: Thiếu trường bắt buộc

```
{
  "error": "MISSING_REQUIRED_FIELD",
  "message": "Trường 'type' là bắt buộc"
}
```

- + 409 Conflict: Tên danh mục đã tồn tại

```
{
  "error": "DUPLICATE_CATEGORY",
  "message": "Danh mục 'Du lịch' đã tồn tại"
}
```

### 3. PUT /api/categories/{id}

- Mục đích: Cập nhật thông tin danh mục
- Request:
  - + Method: PUT
  - + Headers:

**Authorization:** Bearer <access\_token>

**Content-Type:** application/json

- + Body:

```
{
  "name": "Ăn uống gia đình",
```

```
    "type": "expense"
}
```

- **Response thành công (200 OK):**

```
{
  "id": 1,
  "message": "Cập nhật danh mục thành công",
  "updated_at": "2023-10-05T11:00:00Z"
}
```

**Lỗi:**

- + 403 Forbidden: Không có quyền chỉnh sửa
- + 404 Not Found: Không tìm thấy danh mục

```
{
  "error": "CATEGORY_NOT_FOUND",
  "message": "Không tìm thấy danh mục với ID 999"
}
```

#### 4. DELETE /api/categories/{id}

- **Mục đích:** Xóa danh mục cụ thể

- **Request:**

- + Method: DELETE

- + Headers:

```
Authorization: Bearer <access_token>
```

- **Response thành công (200 OK):**

```
{
  "message": "Đã xóa danh mục thành công",
  "deleted_at": "2023-10-05T11:30:00Z"
}
```

- **Lỗi đặc biệt:** 423 Locked: Danh mục đang được sử dụng

```
{
  "error": "CATEGORY_IN_USE",
  "message": "Không thể xóa vì có 5 giao dịch đang sử dụng danh mục này",
  "solution": "Hãy cập nhật các giao dịch sang danh mục khác trước"
}
```

#### 5. DELETE /api/categories/all

- **Mục đích:** Xóa toàn bộ danh mục (yêu cầu xác nhận)

- **Request:**

+ Method: DELETE

+ Headers:

```
Authorization: Bearer <access_token>
```

```
Content-Type: application/json
```

+ Body:

```
{
  "confirmation": "CONFIRM_DELETE_ALL_CATEGORIES"
}
```

- **Response thành công (200 OK):**

```
{
  "message": "Đã xóa toàn bộ 3 danh mục",
  "deleted_at": "2023-10-05T12:00:00Z"
}
```

- **Lỗi:** 412 Precondition Failed: Xác nhận không đúng

```
{
  "error": "INVALID_CONFIRMATION",
  "message": "Yêu cầu xác nhận bằng chuỗi "
}
```

#### 4.3.4 Quản lý tài khoản ngân hàng (Accounts)

Phương thức	Endpoint	Mô tả
GET	/api/accounts	Lấy tất cả tài khoản của người dùng
POST	/api/accounts	Tạo tài khoản mới
PUT	/api/accounts/(id)	Cập nhật thông tin tài khoản (theo ID)
DELETE	/api/accounts/(id)	Xóa tài khoản (theo ID)
DELETE	/api/accounts/all	Xóa tất cả tài khoản của người dùng

Bảng 9 Các phương thức và endpoint của quản lý ngân hàng

## 1. GET /api/accounts

- Mục đích: Lấy danh sách tất cả tài khoản của người dùng
- Request:

```
GET /api/accounts
```

Authorization: Bearer <token>

- Response thành công (200 OK):

```
{  
  "accounts": [  
    {  
      "id": "acc_123",  
      "account_number": "123456789",  
      "type": "savings",  
      "balance": 5000000,  
      "currency": "VND",  
      "created_at": "2023-10-01T00:00:00Z"  
    }  
  ]  
}
```

- Lỗi:

- + 401 Unauthorized: Token không hợp lệ
- + 403 Forbidden: Không có quyền truy cập

## 2. POST /api/accounts

- Mục đích: Tạo tài khoản mới

- Request:

```
POST /api/accounts  
Authorization: Bearer <token>  
Content-Type: application/json  
{  
  "type": "savings",  
  "initial_balance": 1000000,  
  "currency": "VND"  
}
```

- Response thành công (201 Created):

```
{  
  "id": "acc_456",  
  "account_number": "987654321",  
  "message": "Tài khoản được tạo thành công"  
}
```

- **Lỗi:**

- + 400 Bad Request: Thiếu trường bắt buộc
- + 409 Conflict: Số tài khoản đã tồn tại

### 3. PUT /api/accounts/(id)

- **Mục đích:** Cập nhật thông tin tài khoản

- **Request:**

```
PUT /api/accounts/acc_123
Authorization: Bearer <token>
Content-Type: application/json
{
    "type": "current",
    "status": "active"
}
```

- **Response thành công (200 OK):**

```
{
    "message": "Cập nhật tài khoản thành công",
    "updated_fields": ["type", "status"]
}
```

- **Lỗi đặc biệt:** 423 Locked: Tài khoản đang bị khóa

```
{
    "error": "ACCOUNT_LOCKED",
    "message": "Tài khoản đang bị phong tỏa đến 2023-12-31"
}
```

### 4. DELETE /api/accounts/(id)

- **Mục đích:** Xóa tài khoản

- **Request:**

```
DELETE /api/accounts/acc_123
```

```
Authorization: Bearer <token>
```

- **Response thành công (200 OK):**

```
{
    "message": "Tài khoản đã được xóa",
    "deleted_at": "2023-10-05T14:30:00Z"
}
```

- **Lỗi:**

- + 404 Not Found: Không tìm thấy tài khoản

+ 403 Forbidden: Không đủ quyền xóa

## 5. DELETE /api/accounts/all

- Mục đích: Xóa toàn bộ tài khoản

- Request:

```
DELETE /api/accounts/all
Authorization: Bearer <token>
Content-Type: application/json
{
  "confirmation": "CONFIRM_DELETE_ALL_ACCOUNTS"
}
```

- Response thành công (200 OK):

```
{
  "message": "Đã xóa 3 tài khoản",
  "deleted_ids": ["acc_123", "acc_456", "acc_789"]
}
```

- Lỗi đặc biệt:: 412 Precondition Failed: Thiếu xác nhận

```
{
  "error": "CONFIRMATION_REQUIRED",
  "message": "Yêu cầu xác nhận bằng chuỗi 'CONFIRM_DELETE_ALL_ACCOUNTS'"
}
```

### 4.3.5 Quản lý giao dịch thu/chi

Phương thức	Endpoint	Mô tả
GET	/api/transactions	Lấy danh sách toàn bộ giao dịch của người dùng
POST	/api/transactions	Tạo mới một giao dịch
PUT	/api/transactions/(id)	Cập nhật thông tin giao dịch theo ID
DELETE	/api/transactions/(id)	Xóa giao dịch cụ thể theo ID
DELETE	/api/transactions/all	Xóa toàn bộ giao dịch của người dùng (yêu cầu xác nhận)

Bảng 10 Các phương thức và endpoint của mục tiêu tiết kiệm

#### 1. GET /api/transactions

- **Chức năng:** Lấy danh sách **toàn bộ giao dịch** của người dùng hiện tại (có thể phân trang, lọc theo thời gian/loại giao dịch).

- Request Query (Optional):

```
GET /api/transactions?page=1&limit=10&type=income&startDate=2023-01-01&endDate=2023-12-31
```

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "data": {
    "transactions": [
      {
        "id": "txn_123",
        "amount": 1000000,
        "type": "income",
        "category": "Lương",
        "description": "Lương tháng 10",
        "createdAt": "2023-10-01T00:00:00Z"
      }
    ],
    "pagination": {
      "total": 15,
      "page": 1,
      "limit": 10
    }
  }
}
```

- **Hành động tiếp theo:** Hiển thị danh sách giao dịch trên UI (kèm phân trang).

- **Lỗi có thể xảy ra:**

- + 401 Unauthorized: Token không hợp lệ → Yêu cầu đăng nhập lại.
- + 400 Bad Request: Query param không hợp lệ (ví dụ: startDate > endDate).

## 2. POST /api/transactions

- **Chức năng:** Tạo mới một giao dịch (thu/chi).

- Request Body:

```
{
  "amount": 500000,
  "type": "expense",
```

```

    "category": "Ăn uống",
    "description": "Tiền nhậu với team"
}

```

- Response Thành Công (201 Created):

```

{
  "status": "success",
  "message": "Tạo giao dịch thành công",
  "data": {
    "id": "txn_456",
    "amount": 500000,
    "type": "expense",
    "category": "Ăn uống",
    "description": "Tiền nhậu với team",
    "createdAt": "2023-10-05T15:30:00Z"
  }
}

```

- Hành động tiếp theo:

- + Cập nhật danh sách giao dịch trên UI.
- + Hiển thị thông báo: "Đã thêm giao dịch mới!".

- Lỗi có thể xảy ra:

- + 400 Bad Request: Thiếu trường bắt buộc (ví dụ: amount, type).

```

{
  "status": "error",
  "message": "Trường 'amount' là bắt buộc"
}

```

+ 422 Unprocessable Entity: Dữ liệu không hợp lệ (ví dụ: amount là số âm).

```

{
  "status": "error",
  "message": "Số tiền phải lớn hơn 0"
}

```

### 3. PUT /api/transactions/(id)

- Chức năng: Cập nhật thông tin giao dịch theo ID (chỉ giao dịch của người dùng hiện tại).

- Request:

```

PUT /api/transactions/txn_123
Authorization: Bearer <access_token>
Content-Type: application/json

```

```
{  
    "description": "Cập nhật mô tả mới"  
}
```

- Response Thành Công (200 OK):

```
{  
    "status": "success",  
    "message": "Cập nhật giao dịch thành công",  
    "data": {  
        "id": "txn_123",  
        "description": "Cập nhật mô tả mới"  
    }  
}
```

- **Hành động tiếp theo:** Cập nhật lại giao dịch trong danh sách trên UI.

- Lỗi có thể xảy ra:

+ **404 Not Found:** Không tìm thấy giao dịch với ID cung cấp.

```
{  
    "status": "error",  
    "message": "Giao dịch không tồn tại"  
}
```

+ **403 Forbidden:** Giao dịch không thuộc về người dùng hiện tại.

```
{  
    "status": "error",  
    "message": "Bạn không có quyền chỉnh sửa giao dịch này"  
}
```

#### 4. DELETE /api/transactions/(id)

- Chức năng: Xóa một giao dịch cụ thể theo ID.

- Request:

```
DELETE /api/transactions/txn_123
```

```
Authorization: Bearer <access_token>
```

Response Thành Công (200 OK):

```
{  
    "status": "success",  
    "message": "Đã xóa giao dịch thành công"  
}
```

- **Hành động tiếp theo:**

+ Xóa giao dịch khỏi danh sách trên UI.

- + Hiển thị thông báo: "Đã xóa giao dịch!".

- **Lỗi có thể xảy ra:**

- + 404 Not Found: ID giao dịch không tồn tại.

- + 403 Forbidden: Không có quyền xóa giao dịch.

### 5. DELETE /api/transactions/all

- **Chức năng:** Xóa toàn bộ giao dịch của người dùng (yêu cầu xác nhận mật khẩu).

- **Request:**

```
DELETE /api/transactions/all
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "password": "currentPassword123"
}
```

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "message": "Đã xóa toàn bộ 15 giao dịch"
}
```

- Hành động tiếp theo:

- + Xóa toàn bộ danh sách giao dịch trên UI.

- + Hiển thị cảnh báo: "Tất cả giao dịch đã bị xóa vĩnh viễn!".

- **Lỗi có thể xảy ra:**

- + 403 Forbidden: Sai mật khẩu xác nhận.

```
{
  "status": "error",
  "message": "Mật khẩu không chính xác"
}
```

- + 400 Bad Request: Chưa cung cấp mật khẩu.

### 4.3.6 Quản lý mục tiêu tiết kiệm

Endpoint	Method	Chức năng
/api/goals	GET	Lấy tất cả mục tiêu
/api/goals	POST	Tạo mục tiêu mới

Endpoint	Method	Chức năng
/api/goals/all	DELETE	Xóa tất cả mục tiêu
/api/goals/fix-categories-icon	PATCH	Sửa icon danh mục mục tiêu
/api/goals/:id	PUT	Cập nhật toàn bộ thông tin mục tiêu
/api/goals/:id	DELETE	Xóa mục tiêu cụ thể
/api/goals/:id/add-funds	POST	Thêm tiền vào mục tiêu
/api/goals/:id/archive	PATCH	Thay đổi trạng thái lưu trữ
/api/goals/:id/pin	PATCH	Thay đổi trạng thái ghim

Bảng 12 Các phương thức và endpoint quản lý mục tiêu

### 1. GET /api/goals

- **Chức năng:** Lấy danh sách tất cả mục tiêu tiết kiệm của người dùng hiện tại (có thể bao gồm mục tiêu đã ghim, đã lưu trữ).

- Request:

GET /api/goals

Authorization: Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "data": [
    {
      "id": "goal_123",
      "name": "Mua laptop mới",
      "targetAmount": 20000000,
      "currentAmount": 5000000,
      "deadline": "2023-12-31",
      "isPinned": true,
      "isArchived": false,
      "category": {
        "name": "Đồ điện tử",
        "icon": "💻"
      }
    }
  ]
}
```

```
}
```

- Hành động tiếp theo: Hiển thị danh sách mục tiêu trên UI.
- Lỗi có thể xảy ra: **401 Unauthorized**: Token không hợp lệ → Yêu cầu đăng nhập lại.

## 2. POST /api/goals

- Chức năng: Tạo mục tiêu tiết kiệm mới.
- Request Body:

```
{
  "name": "Du lịch Đà Lạt",
  "targetAmount": 10000000,
  "deadline": "2024-06-01",
  "category": "Du lịch"
}
```

- Response Thành Công (201 Created):

```
{
  "status": "success",
  "message": "Tạo mục tiêu thành công",
  "data": {
    "id": "goal_456",
    "name": "Du lịch Đà Lạt",
    "targetAmount": 10000000,
    "currentAmount": 0,
    "deadline": "2024-06-01"
  }
}
```

- **Hành động tiếp theo:**

- + Thêm mục tiêu mới vào danh sách trên UI.
- + Hiển thị thông báo: "Đã tạo mục tiêu!".
- Lỗi có thể xảy ra:

+ **400 Bad Request**: Thiếu trường bắt buộc (ví dụ: name, targetAmount).

```
{
  "status": "error",
  "message": "Trường 'targetAmount' là bắt buộc"
}
```

+ **422 Unprocessable Entity**: Dữ liệu không hợp lệ (ví dụ: deadline trong quá khứ).

## 3. DELETE /api/goals/all

- **Chức năng:** Xóa **toàn bộ** mục tiêu của người dùng (yêu cầu xác nhận mật khẩu).

- Request:

```
DELETE /api/goals/all  
Authorization: Bearer <access_token>  
Content-Type: application/json
```

```
{  
  "password": "currentPassword123"  
}
```

- Response Thành Công (200 OK):

```
{  
  "status": "success",  
  "message": "Đã xóa 5 mục tiêu"  
}
```

- Hành động tiếp theo:

- + Xóa toàn bộ mục tiêu trên UI.
- + Hiển thị cảnh báo: "Tất cả mục tiêu đã bị xóa!".

- Lỗi có thể xảy ra:

- + 403 Forbidden: Sai mật khẩu.

```
{  
  "status": "error",  
  "message": "Mật khẩu không chính xác"  
}
```

---

#### 4. PATCH /api/goals/fix-categories-icon

- **Chức năng:** Cập nhật icon cho các danh mục mục tiêu (dành cho admin/quản trị viên).

- Request Body:

```
{  
  "category": "Du lịch",  
  "newIcon": "✈"  
}
```

- Response Thành Công (200 OK):

```
{  
  "status": "success",  
}
```

```
        "message": "Đã cập nhật icon cho danh mục 'Du lịch'"  
    }
```

- **Hành động tiếp theo:** Cập nhật icon trên UI.

- Lỗi có thể xảy ra:

+ **403 Forbidden:** Không có quyền admin.

+ **404 Not Found:** Danh mục không tồn tại.

5. PUT /api/goals/{id}

- **Chức năng:** Cập nhật thông tin mục tiêu (tên, số tiền mục tiêu, deadline).

- **Request:**

```
PUT /api/goals/goal_123
```

**Authorization:** Bearer <access\_token>

**Content-Type:** application/json

```
{  
    "name": "Mua MacBook Pro",  
    "targetAmount": 25000000  
}
```

- Response Thành Công (200 OK):

```
{  
    "status": "success",  
    "message": "Cập nhật mục tiêu thành công",  
    "data": {  
        "id": "goal_123",  
        "name": "Mua MacBook Pro",  
        "targetAmount": 25000000  
    }  
}
```

- **Hành động tiếp theo:** Cập nhật thông tin mục tiêu trên UI.

- Lỗi có thể xảy ra:

+ **404 Not Found:** Mục tiêu không tồn tại.

+ **403 Forbidden:** Mục tiêu không thuộc về người dùng.

6. DELETE /api/goals/{id}

- **Chức năng:** Xóa một mục tiêu cụ thể.

- **Request:**

```
DELETE /api/goals/goal_123
```

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "message": "Đã xóa mục tiêu 'Mua laptop mới'"
}
```

- **Hành động tiếp theo:** Xóa mục tiêu khỏi danh sách trên UI.

- Lỗi có thể xảy ra: **404 Not Found**: Mục tiêu không tồn tại.

#### 7. POST /api/goals/{id}/add-funds

- **Chức năng:** Nạp tiền vào mục tiêu tiết kiệm.

- **Request Body:**

```
{
  "amount": 1000000,
  "description": "Tiền thưởng tháng 10"
}
```

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "message": "Đã nạp 1,000,000đ vào mục tiêu",
  "data": {
    "currentAmount": 6000000
  }
}
```

- **Hành động tiếp theo:**

+ Cập nhật số dư hiện tại của mục tiêu trên UI.

+ Hiển thị thông báo: "Nạp tiền thành công!".

- Lỗi có thể xảy ra: 400 Bad Request: Số tiền  $\leq 0$  hoặc vượt quá giới hạn.

#### 8. PATCH /api/goals/{id}/archive

- **Chức năng:** Lưu trữ/bỏ lưu trữ mục tiêu (ẩn khỏi danh sách chính).

- **Request:**

**PATCH /api/goals/goal\_123/archive**

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "message": "Đã lưu trữ mục tiêu",
  "data": {
    ...
  }
}
```

```

    "isArchived": true
  }
}

```

- Hành động tiếp theo: Âm mục tiêu khỏi danh sách chính (chuyển sang tab "Đã lưu trữ").

#### 9. PATCH /api/goals/{id}/pin

- Chức năng: Ghim/bỏ ghim mục tiêu (hiển thị lên đầu danh sách).

- Request:

**PATCH /api/goals/goal\_123/pin**

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```

{
  "status": "success",
  "message": "Đã ghim mục tiêu",
  "data": {
    "isPinned": true
  }
}

```

- + Hành động tiếp theo: Di chuyển mục tiêu lên đầu danh sách.

#### 4.3.7 Quản lý thống kê và báo cáo

STT	Endpoint	Method	Mô Tả
1	/api/statistics/overview	GET	Lấy thống kê tổng quan
2	/api/statistics/trend	GET	Lấy xu hướng thu chi theo thời gian
3	/api/statistics/by-category	GET	Thống kê theo danh mục
4	/api/statistics/calendar	GET	Lấy dữ liệu giao dịch cho lịch tháng
5	/api/statistics/summary	GET	Lấy báo cáo tổng hợp

Bảng 13 Các phương thức và endpoint của thống kê và báo cáo

#### 1. GET /api/statistics/overview

- Chức năng: Lấy thống kê tổng quan về thu nhập, chi tiêu, tiết kiệm trong một khoảng thời gian (mặc định: 30 ngày gần nhất).

- Request Query (Optional):

```
GET /api/statistics/overview?startDate=2023-10-01&endDate=2023-10-31
```

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "data": {
    "totalIncome": 15000000,
    "totalExpense": 8000000,
    "savingRate": 46.67,
    "mostSpentCategory": "Ăn uống",
    "transactionsCount": 28
  }
}
```

- Hành động tiếp theo: Hiển thị các chỉ số tổng quan dưới dạng card/bảng trên UI.
- Lỗi có thể xảy ra: 400 Bad Request: Định dạng ngày không hợp lệ (ví dụ: startDate=abc).

```
{
  "status": "error",
  "message": "Định dạng ngày phải là YYYY-MM-DD"
}
```

## 2. GET /api/statistics/trend

- Chức năng: Lấy xu hướng thu/chi theo thời gian (tuần/tháng/năm), có thể nhóm theo ngày/tuần/tháng.

- Request Query:

```
GET /api/statistics/trend?groupBy=month&type=expense&year=2023
```

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "data": [
    {
      "period": "Tháng 1",
      "amount": 2500000
    },
    {
      "period": "Tháng 2",
      "amount": 1800000
    }
  ]
}
```

```
    }
]
}
```

- Hành động tiếp theo: Vẽ biểu đồ đường (line chart) thể hiện xu hướng.
- Lỗi có thể xảy ra: 400 Bad Request: Giá trị groupBy không hợp lệ (ví dụ: groupBy=decade).

```
{
  "status": "error",
  "message": "Giá trị groupBy phải là day/week/month"
}
```

### 3. GET /api/statistics/by-category

- Chức năng: Thống kê chi tiêu theo danh mục (có thể lọc theo loại giao dịch: income/expense).

- Request Query:

```
GET /api/statistics/by-category?type=expense&limit=5
```

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "data": [
    {
      "category": "Ăn uống",
      "amount": 3000000,
      "percentage": 37.5
    },
    {
      "category": "Di chuyển",
      "amount": 1500000,
      "percentage": 18.75
    }
  ]
}
```

- Hành động tiếp theo: Vẽ biểu đồ tròn (pie chart) hoặc bảng phân bố.
- Lỗi có thể xảy ra: 400 Bad Request: Thiếu tham số type.

```
{
  "status": "error",
  "message": "Tham số 'type' là bắt buộc (income/expense)"
}
```

#### 4. GET /api/statistics/calendar

- Chức năng: Lấy dữ liệu giao dịch theo ngày để hiển thị trên lịch (ví dụ: tổng thu/chi mỗi ngày).

- Request Query:

```
GET /api/statistics/calendar?month=10&year=2023
```

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "data": {
    "2023-10-01": {
      "income": 500000,
      "expense": 200000
    },
    "2023-10-02": {
      "income": 0,
      "expense": 150000
    }
  }
}
```

- Hành động tiếp theo: Hiển thị các chấm màu (dot indicator) trên lịch UI.
- Lỗi có thể xảy ra: 400 Bad Request: Tháng/năm không hợp lệ (ví dụ: month=13).

#### 5. GET /api/statistics/summary

- Chức năng: Lấy báo cáo tổng hợp đa chiều (thu nhập, chi tiêu, tiết kiệm, so sánh theo kỳ trước).

- Request Query:

```
GET /api/statistics/summary?compareWith=last_month
```

**Authorization:** Bearer <access\_token>

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "data": {
    "currentPeriod": {
      "income": 15000000,
      "expense": 8000000
    }
  }
}
```

```

    "previousPeriod": {
        "income": 12000000,
        "expense": 9000000
    },
    "difference": {
        "income": "+25%",
        "expense": "-11.11%"
    }
}
}

```

- Hành động tiếp theo: Hiển thị bảng so sánh + biểu đồ kép (dual-axis chart).
- Lỗi có thể xảy ra: 400 Bad Request: Giá trị compareWith không hỗ trợ (ví dụ: compareWith=last\_decade).

#### 4.3.8 Trợ lí AI thông minh

Phương thức	Endpoint	Mô tả
POST	/api/ai-assistant	Khởi tạo AI Assistant - "Go! I'm mhán dán AI Assistant"
POST	/api/ai-assistant/create-transaction	Tạo giao dịch tự động từ đề xuất AI (tiền, danh mục, thời gian tự động)
POST	/api/ai-assistant/create-category	Tạo danh mục mới từ phân tích thói quen chi tiêu
POST	/api/ai-assistant/create-account	Tạo tài khoản ngân hàng tự động với thông số tối ưu từ AI
POST	/api/ai-assistant/create-goal	Tạo mục tiêu tài chính thông minh dựa trên thu nhập/chi tiêu hiện tại

Bảng 13 Các phương thức và endpoint của trợ lí AI

##### 1. POST /api/ai-assistant

- Chức năng: Gửi tin nhắn đến AI Assistant để nhận phản hồi thông minh (hỗ trợ tư vấn tài chính, phân tích giao dịch, v.v.).

##### - Request Body:

```

{
    "message": "Tôi đã chi tiêu quá nhiều vào ăn uống tháng này, làm sao để cải thiện?",
    "context": {
        "lastTransactions": ["Ăn uống: 500,000đ", "Mua sắm: 300,000đ"]
    }
}

```

```
}
```

- Response Thành Công (200 OK):

```
{
  "status": "success",
  "data": {
    "response": "Bạn nên giới hạn 300,000đ/ngày cho ăn uống,
    "suggestions": [
      { "type": "create_budget", "category": "Ăn uống", "amount": 9000000 },
      { "type": "insight", "content": "Chi ăn uống chiếm 40% tổng chi tháng này" }
    ]
  }
}
```

- Hành động tiếp theo: Hiển thị phản hồi AI và gợi ý trên UI.

- Lỗi có thể xảy ra:

+ 400 Bad Request: Thiếu trường message.

```
{
  "status": "error",
  "message": "Trường 'message' là bắt buộc"
}
```

+ 429 Too Many Requests: Gửi quá nhiều request liên tiếp.

```
{
  "status": "error",
  "message": "Vui lòng đợi 10s trước khi gửi tin nhắn mới"
}
```

## 2. POST /api/ai-assistant/create-transaction

- Chức năng: Tạo giao dịch tự động từ gợi ý của AI (ví dụ: AI phát hiện tin nhắn ngân hàng chứa thông tin giao dịch).

- Request Body:

```
{
  "aiSuggestionId": "sugg_123",
  "details": {
    "amount": 200000,
    "description": "Mua sách tại Tiki",
    "category": "Giáo dục"
  }
}
```

- Response Thành Công (201 Created):

```
{
```

```

    "status": "success",
    "message": "Đã tạo giao dịch từ gợi ý AI",
    "data": {
        "transactionId": "txn_789",
        "amount": 200000,
        "category": "Giáo dục"
    }
}

```

- Hành động tiếp theo: Thêm giao dịch vào danh sách và hiển thị thông báo thành công.

- Lỗi có thể xảy ra:

- + 404 Not Found: aiSuggestionId không tồn tại.
- + 400 Bad Request: Thiếu trường amount hoặc category.

### 3. POST /api/ai-assistant/create-category

- Chức năng: Tạo danh mục mới từ gợi ý của AI (ví dụ: AI đề xuất danh mục mới dựa trên thói quen chi tiêu).

- Request Body:

```

{
    "name": "Đầu tư chứng khoán",
    "icon": "📈",
    "color": "#4CAF50"
}

```

- Response Thành Công (201 Created):

```

{
    "status": "success",
    "message": "Đã tạo danh mục 'Đầu tư chứng khoán'",
    "data": {
        "categoryId": "cat_456",
        "icon": "📈"
    }
}

```

- Hành động tiếp theo: Cập nhật danh sách danh mục trên UI.
- Lỗi có thể xảy ra: 409 Conflict: Danh mục đã tồn tại.

```

{
    "status": "error",
    "message": "Danh mục 'Đầu tư chứng khoán' đã tồn tại"
}

```

### 4. POST /api/ai-assistant/create-account

- **Chức năng:** Tạo tài khoản ngân hàng/ví điện tử từ gợi ý AI (ví dụ: AI phát hiện thông tin tài khoản trong email).

- Request Body:

```
{
  "accountName": "Techcombank",
  "accountNumber": "190123456789",
  "balance": 5000000
}
```

- Response Thành Công (201 Created):

```
{
  "status": "success",
  "message": "Đã thêm tài khoản Techcombank",
  "data": {
    "accountId": "acc_789",
    "balance": 5000000
  }
}
```

- Hành động tiếp theo: Hiển thị tài khoản mới trong danh sách.
- Lỗi có thể xảy ra: 400 Bad Request: Số tài khoản không hợp lệ.

```
{
  "status": "error",
  "message": "Số tài khoản phải có ít nhất 10 chữ số"
}
```

## 5. POST /api/ai-assistant/create-goal

- **Chức năng:** Tạo mục tiêu tiết kiệm tự động từ gợi ý AI (ví dụ: AI đề xuất mục tiêu dựa trên thu nhập hiện tại).

- Request Body:

```
{
  "goalName": "Mua ô tô cũ",
  "targetAmount": 200000000,
  "deadline": "2025-12-31"
}
```

- Response Thành Công (201 Created):

```
{
  "status": "success",
  "message": "Đã tạo mục tiêu 'Mua ô tô cũ'",
  "data": {
    "goalId": "goal_999",
  }
}
```

```
        "monthlySaving": 5000000
    }
}
```

- Hành động tiếp theo: Thêm mục tiêu vào danh sách và tính toán kế hoạch tiết kiệm.

- Lỗi có thể xảy ra: 422 Unprocessable Entity Deadline trong quá khứ.

```
{
    "status": "error",
    "message": "Hạn chót phải lớn hơn ngày hiện tại"
}
```

## 4.4 Thiết kế giao diện (UI/UX)

Giao diện người dùng được lên kế hoạch và thực hiện trên Figma trước khi đưa vào lập trình từ đó giúp đảm bảo tính trực quan và nhất quán. Các yếu tố về màu sắc, bố cục các chức năng và khả năng tương tác được cải thiện cao giúp người dùng tối ưu được trải nghiệm của mình.

### 4.4.1 Giao diện Đăng ký/ Đăng nhập

Giao diện đầu tiên mà người dùng tiếp cận là trang chào mừng – được thiết kế tối giản nhưng hiện đại, giúp mang lại trải nghiệm thân thiện ngay từ ánh nhìn đầu tiên. Với tông màu ấm áp, nhẹ nhàng, kết hợp cùng các yếu tố đồ họa mềm mại, giao diện tạo cảm giác gần gũi và dễ sử dụng cho mọi đối tượng người dùng.

Giao diện này không chỉ là một phần mở đầu mà còn là cầu nối giữa trải nghiệm người dùng (UX) và giao diện người dùng (UI). Mục tiêu là tạo cảm giác chuyên nghiệp, đơn giản và dễ tiếp cận — giúp người dùng nhanh chóng hiểu được giá trị của ứng dụng và bắt đầu hành trình sử dụng một cách suôn sẻ.

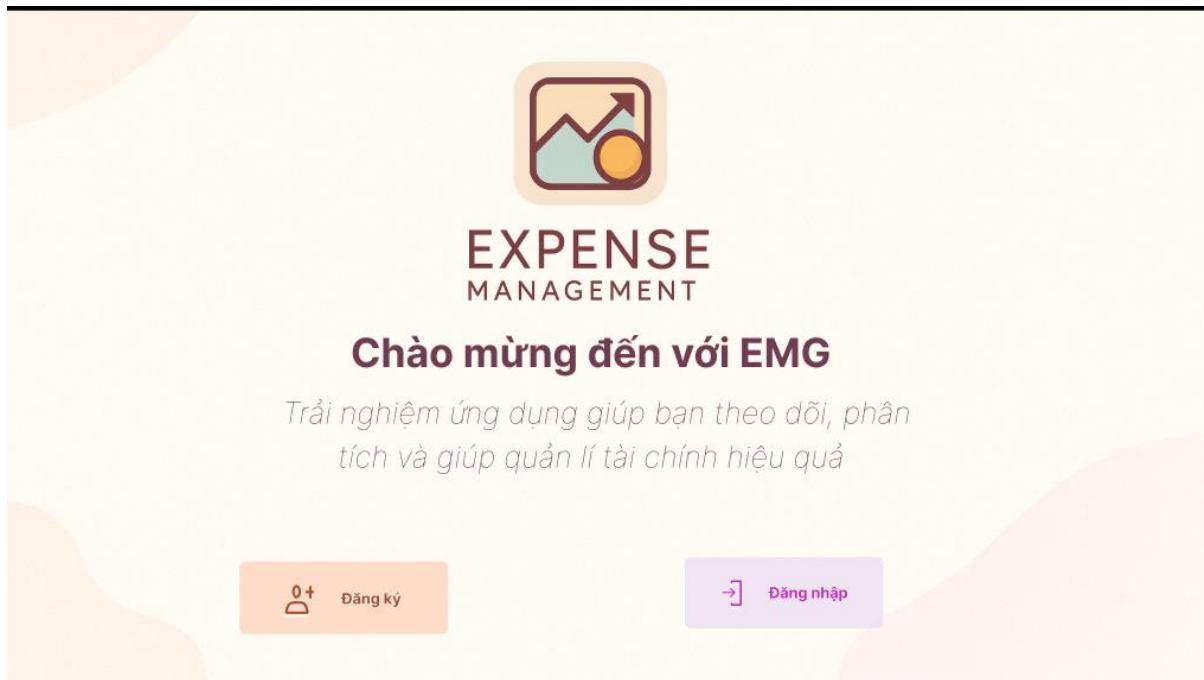
- Logo biểu tượng tài chính được đặt ở vị trí trung tâm, thể hiện rõ mục đích của ứng dụng: quản lý chi tiêu một cách hiệu quả và trực quan.

- Thông điệp "Chào mừng đến với EMG" được nhấn mạnh bằng kiểu chữ nổi bật, tạo sự chuyên nghiệp và thân thiện.

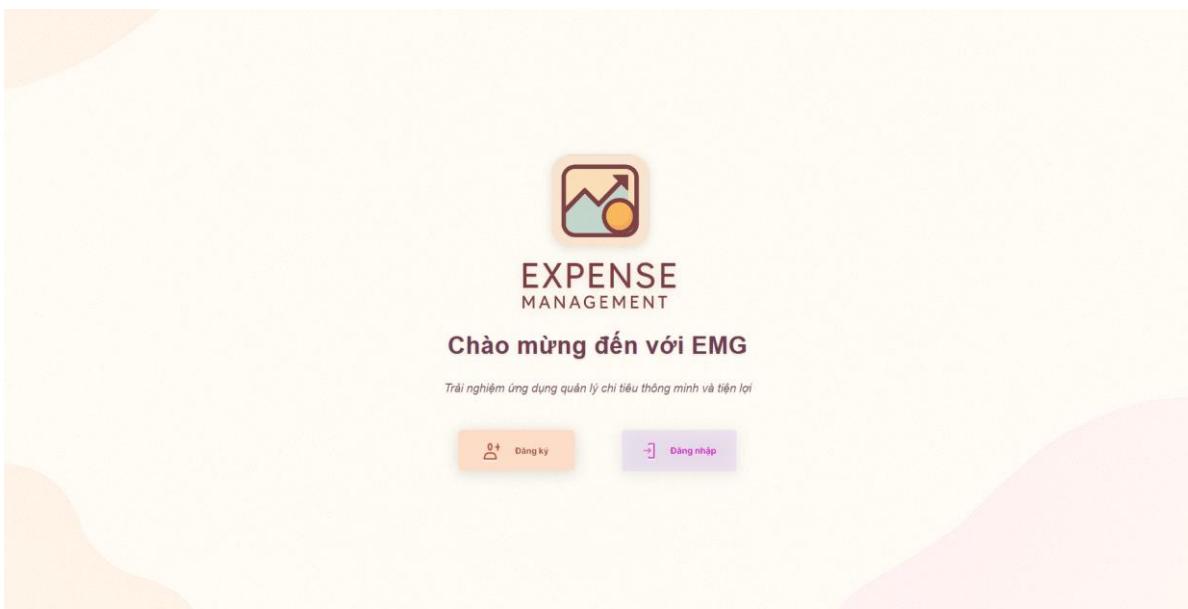
- Mô tả ngắn gọn bên dưới đóng vai trò như một lời mời trải nghiệm ứng dụng: “Trải nghiệm ứng dụng quản lý chi tiêu thông minh và tiện lợi”.

- Hai nút chức năng chính ("Đăng ký" và "Đăng nhập") được đặt song song với màu sắc đối lập, dễ phân biệt và điều hướng:

- + Nút “Đăng ký” sử dụng màu cam nhạt, thu hút người dùng mới.
- + Nút “Đăng nhập” có màu tím nhạt, thể hiện sự ổn định và tin cậy.



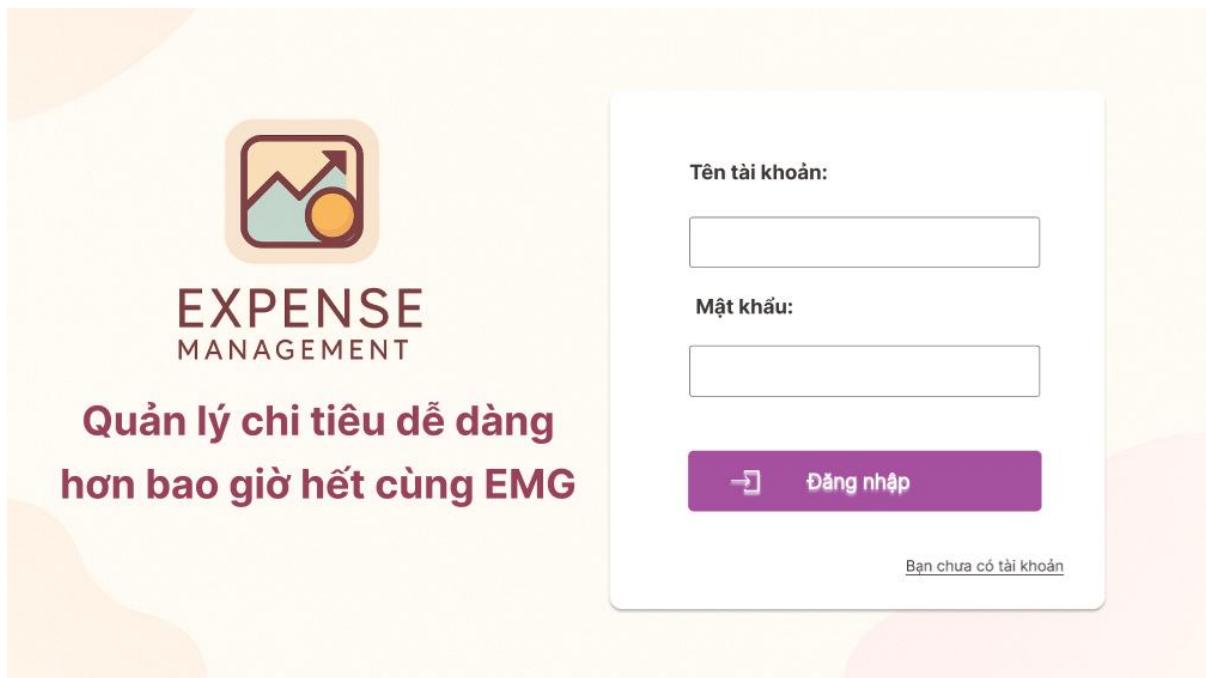
Hình 4.8 Trang chào mừng khi được thiết kế trên Figma



Hình 4.9 Trang chào mừng khi đã triển khai lên website

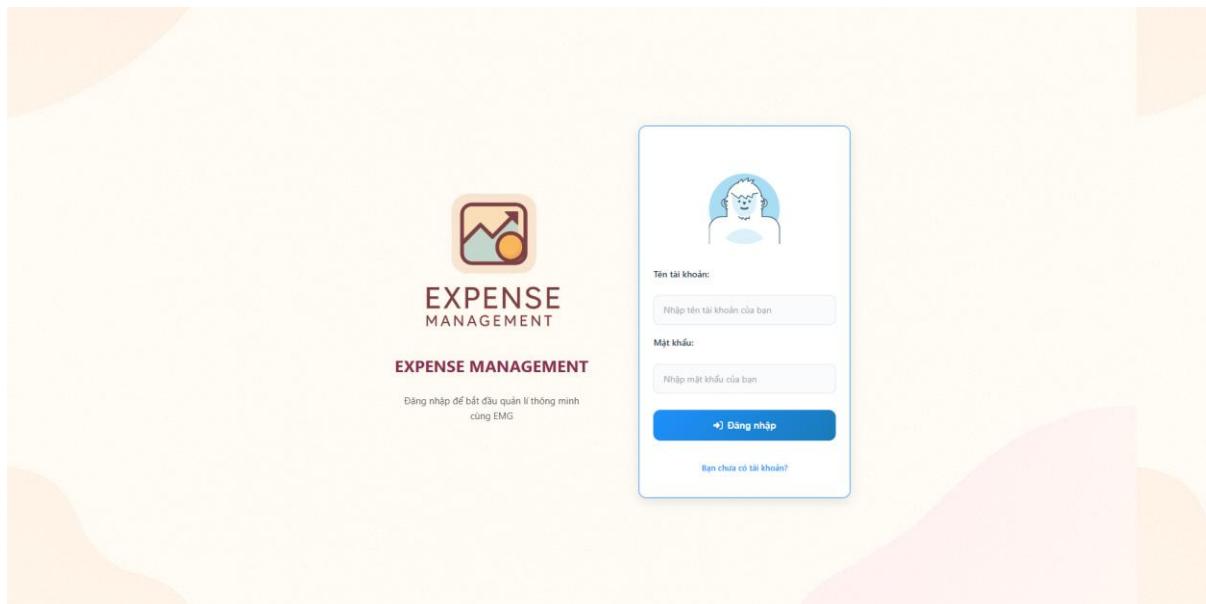
Người dùng bấm vào hai nút Đăng nhập hoặc Đăng ký để chuyển đến trang tương ứng. Nếu ở form đăng ký và sau khi bấm nút “ĐĂNG KÝ” sẽ chuyển sang form đăng nhập.

Khi người dùng lỡ ấn Đăng nhập mà chưa có tài khoản thì có thể ngay lập tức chuyển sang giao diện Đăng ký bằng nút chưa có tài khoản ở phía dưới nút Đăng nhập.

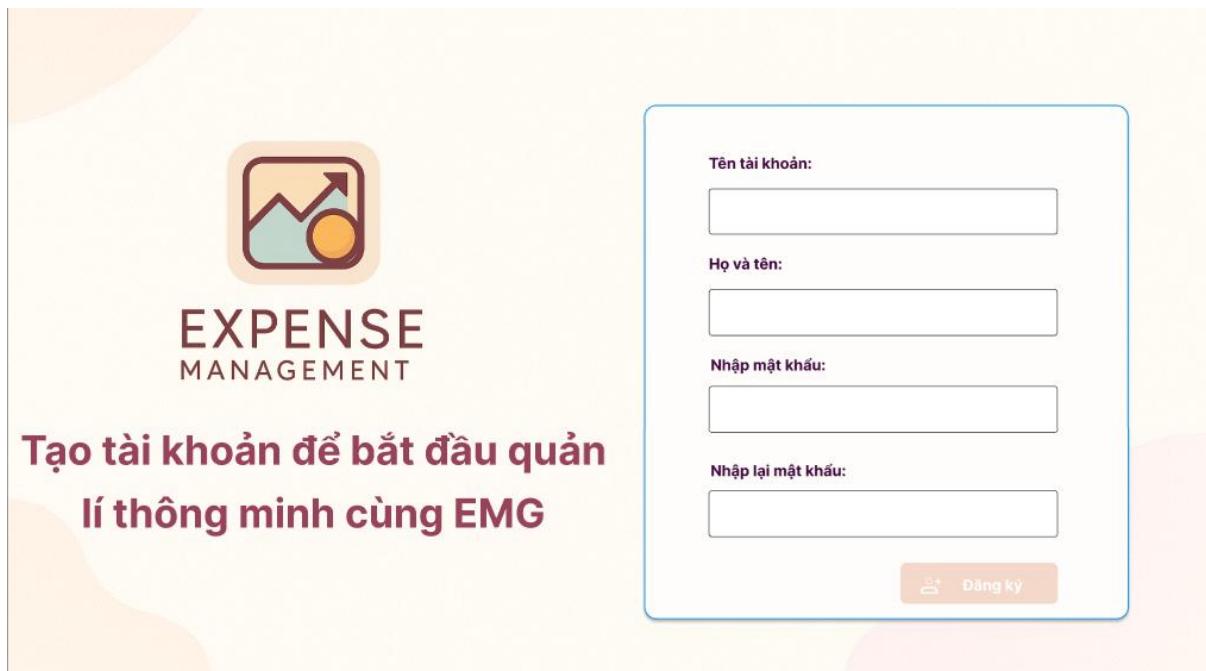


Hình 4.10 Giao diện đăng nhập khi thiết kế trên Figma

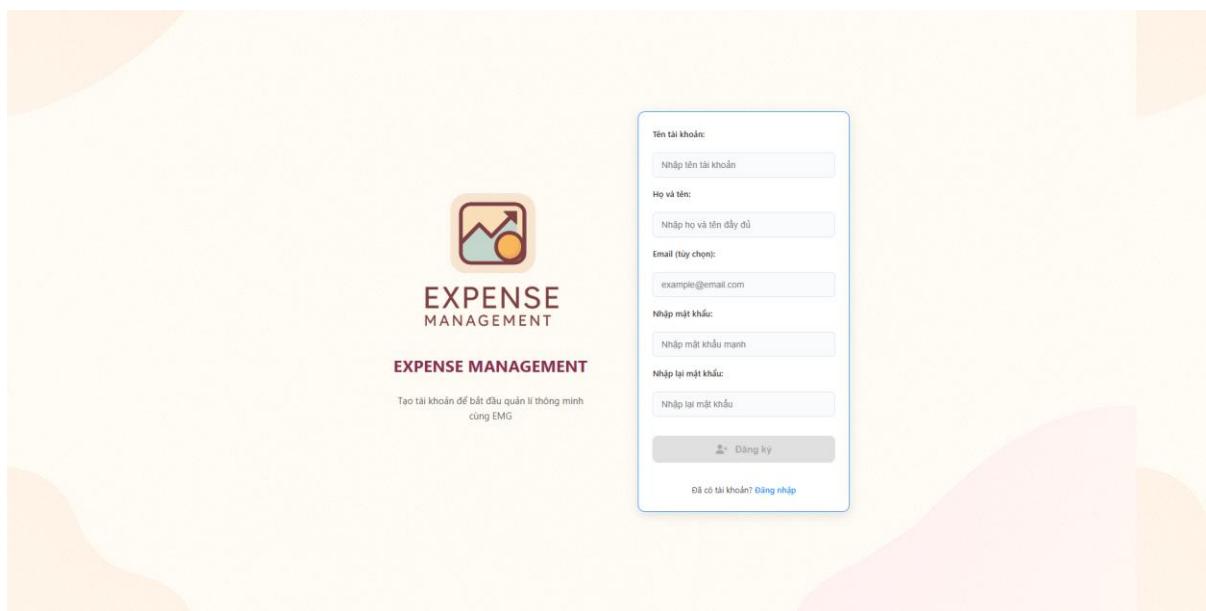
Dựa vào các bản thiết kế cơ bản trên Figma để hiện thực hóa thành các giao diện hoàn chỉnh, sinh động và bắt mắt cho website và tiến hành thêm mới một số icon để thêm phù hợp với đề tài.



Hình 4.11 Giao diện đăng nhập khi được triển khai thực tế



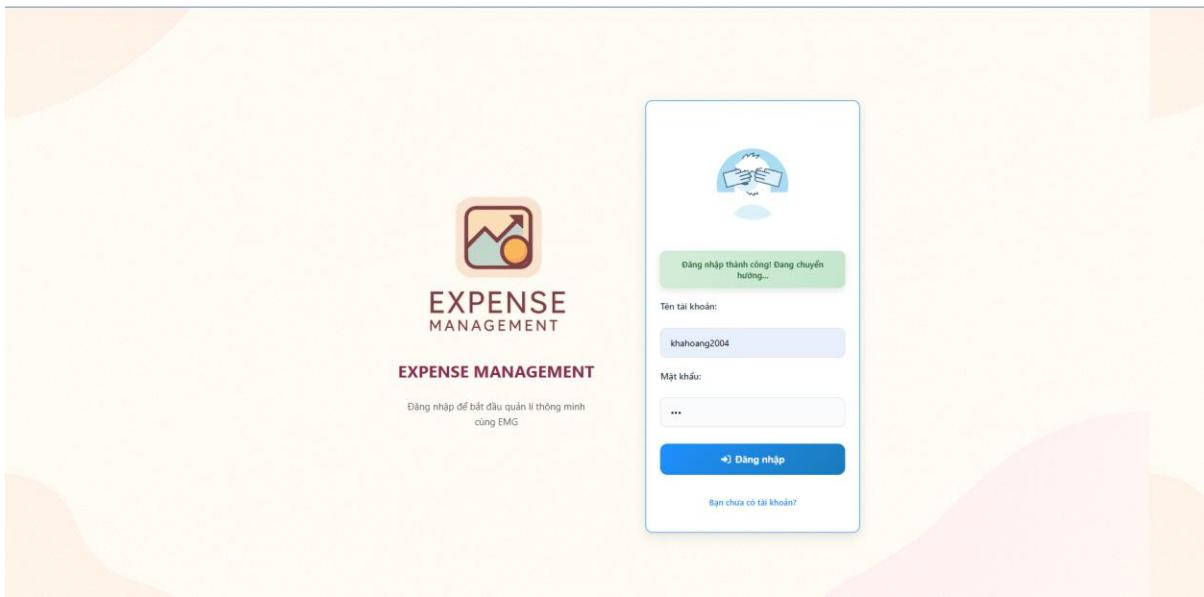
Hình 4.12 Giao diện đăng ký khi được thiết kế trên Figma



Hình 4.13 Giao diện đăng ký khi được triển khai thực tế

Khi đăng ký thành công thì sẽ chuyển sang giao diện đăng nhập với đường link URL: <http://localhost:5173/login>

Tiếp tục, đăng nhập thì sẽ chuyển hướng sang trang chủ với đường link URL: <http://localhost:5173/homepage>



Hình 4.14 Giao diện khi đăng nhập thành công

#### 4.4.2 Giao diện trang chủ

Giao diện chính của ứng dụng quản lý chi tiêu được thiết kế nhằm mang lại cái nhìn trực quan và đầy đủ về tình hình tài chính của người dùng. Tại đây, người dùng có thể dễ dàng theo dõi thu nhập, chi tiêu, số dư, cũng như truy cập nhanh đến các chức năng quản lý khác nhau.

Component Header và Footer được sử dụng cho toàn bộ các thành phần của website để đảm bảo tính nhất quán, cân đối và tiện lợi cho người dùng.

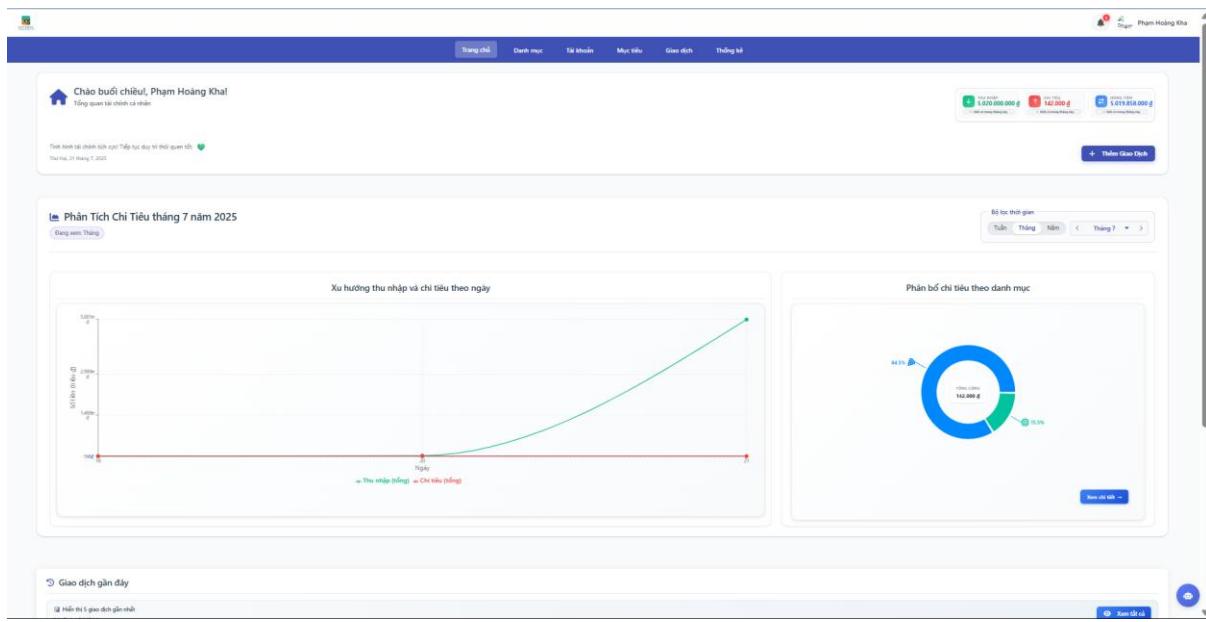
Thanh Header gồm các thành phần sau nơi xem xem thông tin tài khoản và các thông báo về mục tiêu chi tiêu hiện tại.

Tiếp đến sẽ là menu cho phép chuyển hướng đến các giao diện khác của website gồm các mục như: Trang chủ, Danh mục, Tài khoản, Mục tiêu, Giao dịch, Thống kê, ...

Thời gian	Danh mục	Mô tả	Phương thức thanh toán	Số tiền	Hành động
10/4/2025	Ăn uống	Đặt đồ ăn online	BDV	-150.000đ	
11/4/2025	Mua sắm	Mua điện thoại mới	Tienmat	-2.150.000đ	
12/4/2025	Lương	Nhiệm vụ	Vietcombank	+15.500.000đ	

Hình 4.15 Giao diện trang chủ khi được thiết kế trên Figma

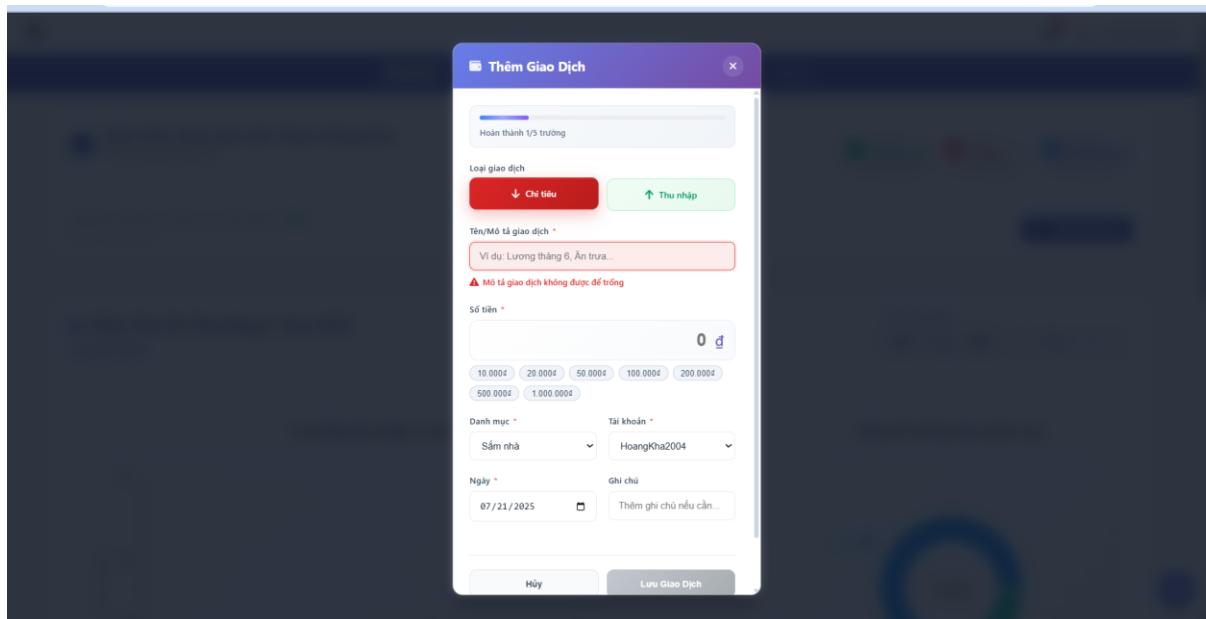
Dựa vào các thành phần đã thiết kế, tiến hành phát triển giao diện trang chủ thực tế cho website và tùy chỉnh để màu sắc phù hợp và cân đối hơn.



Hình 4.16 Giao diện trang chủ khi được triển khai lên React

Tiếp theo, ở phần content gồm các thành phần như:

- Banner: Hiển thị số dư tài khoản mức chi tiêu và thu nhập trong tháng hiện tại. Kèm theo cho phép thực hiện thêm giao dịch với nút Thêm giao dịch giúp thêm giao dịch nhanh chóng giúp tiết kiệm thời gian.

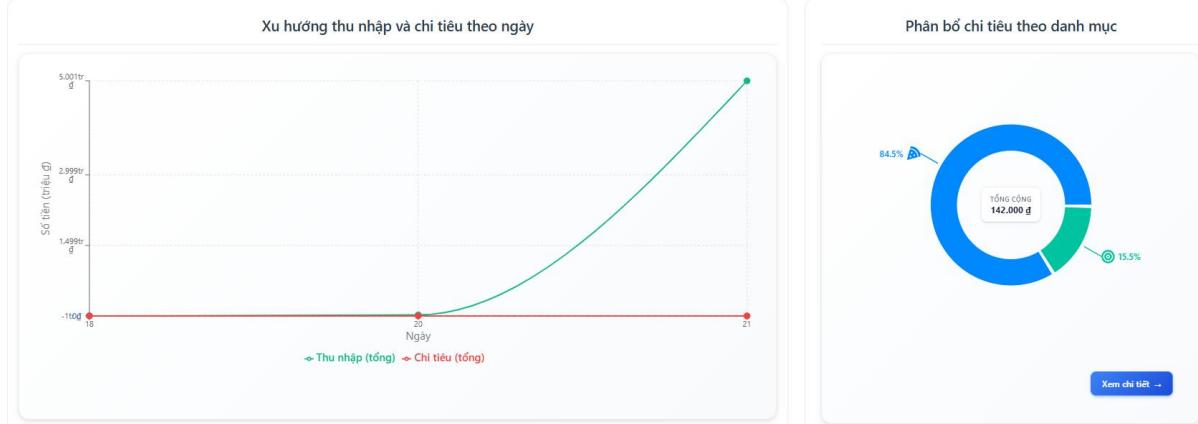


Hình 4.17 Khi thực hiện thêm giao dịch ở trang chủ

- Chart:

+ Line chart: hiển thị biểu đồ đường cho biết xu hướng thu nhập, chi tiêu trực quan theo hàng ngày, người dùng có thể so sánh dễ dàng và trực quan hơn.

+ Pie chart showing the structure: thể hiện cơ cấu chi tiêu theo từng loại danh mục.



Hình 4.18 Phàn Chart của trang chủ

- Recent tracsaction: thể hiện các giao dịch mà người dùng đã thực hiện gần đây và cho phép chỉnh sửa các giao dịch đó.

The table displays five recent transactions:

Thời gian	Danh mục	Mô tả	Phương thức TT	Số tiền	Hành động
21/07/2025 Tạo lúc: 10:36	Lương	thu lương 5 tỷ	HoangKha2004	5.000.000.000 đ	
21/07/2025 Tạo lúc: 01:23	Sắm nhà	Tiết kiệm: Sắm nhà	HoangKha2004	22.000 đ	
20/07/2025 Tạo lúc: 08:27	Lương	thu 10 triệu lương	HoangKha2004	10.000.000 đ	
20/07/2025 Tạo lúc: 08:26	Lương	thu 10 triệu lương	HoangKha2004	10.000.000 đ	
20/07/2025 Tạo lúc: 08:24	Ăn uống	ăn sáng	Ví cá nhân	50.000 đ	

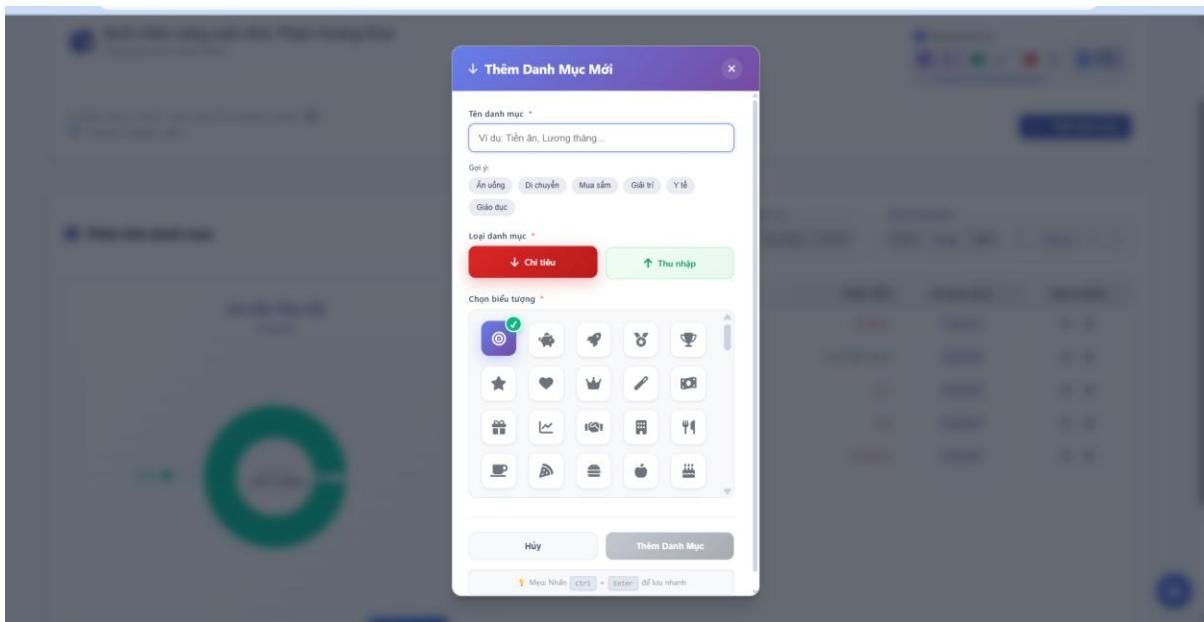
Hình 4.19 Phàn Recent tracsaction của trang chủ

#### 4.4.3 Giao diện danh mục

Khi ấn sang phần danh mục của thanh Header thì website sẽ chuyển hướng sang link URL: <http://localhost:5173/categories>

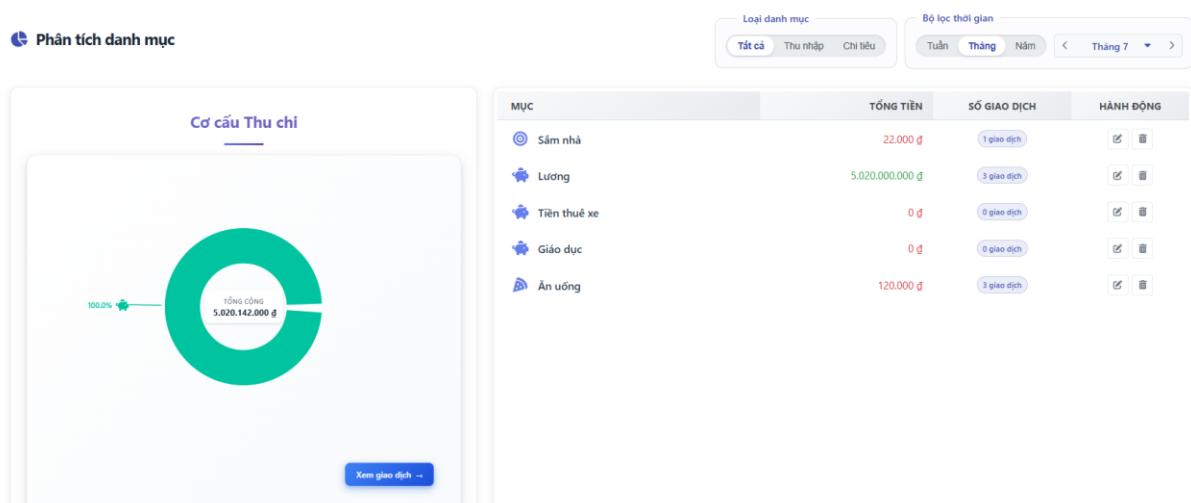
Giao diện danh mục sẽ hiện phần content gồm 2 thành phần là khung Banner và khung Phân tích danh mục:

- Khung Banner: chỉ khác với giao diện Trang chủ là nút Thêm giao dịch được thay bằng nút Thêm danh mục cho phép người dùng có thể thêm các danh mục mua sắm chi tiêu, thu nhập mới với list biểu tượng phong phú và sinh động phù hợp với nhu cầu của người dùng.



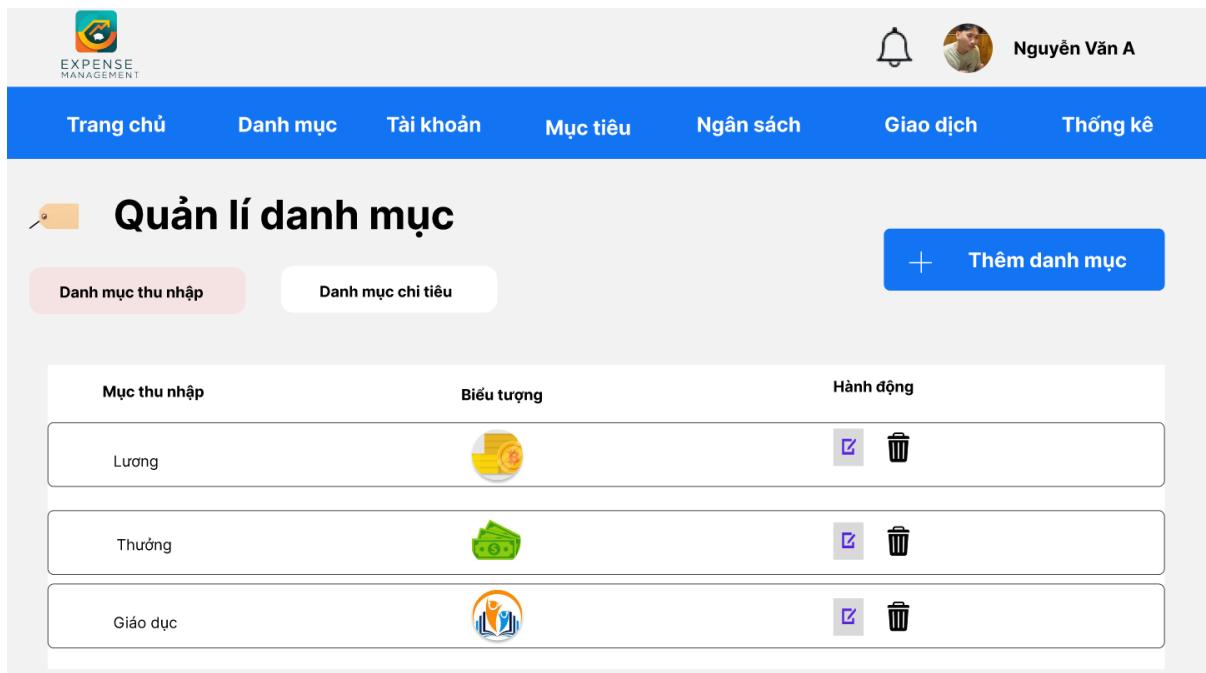
Hình 4.20 Giao diện thêm giao dịch

- Khung phân tích danh mục:
  - + Trực quan cơ cấu thu chi bằng đồ tròn thể hiện số tiền đã thu và chi tiêu, khi click vào nút Xem giao dịch sẽ chuyển đến link URL: <http://localhost:5173/transactions> của giao diện Giao dịch.
  - + Trực quan hóa danh sách các danh mục hiện tại mà người dùng đã sử dụng, cho phép lọc và xem danh mục thuộc chi tiêu và thu nhập, lọc các danh mục theo thời gian theo tuần, tháng, năm.



Hình 4.21 Cơ cấu thu chi của người dùng mẫu

Giao diện người dùng khi mô tả ở Figma đã được đặc tả và cụ thể hơn đi sâu vào khi triển khai thực tế trên website.



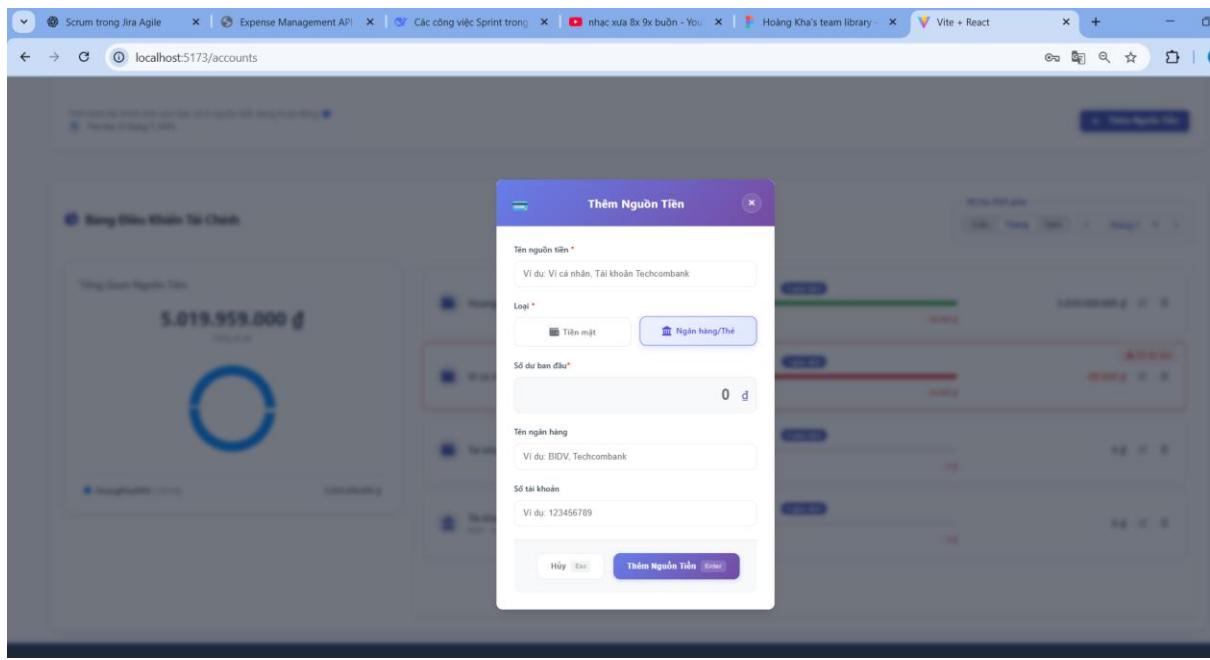
Hình 4.22 Giao diện khi thiết kế trên Figma

#### 4.4.4 Giao diện thông tin tài khoản

Khi ấn sang phần Tài khoản của thanh Header thì website sẽ chuyển hướng sang link URL: <http://localhost:5173/accounts>.

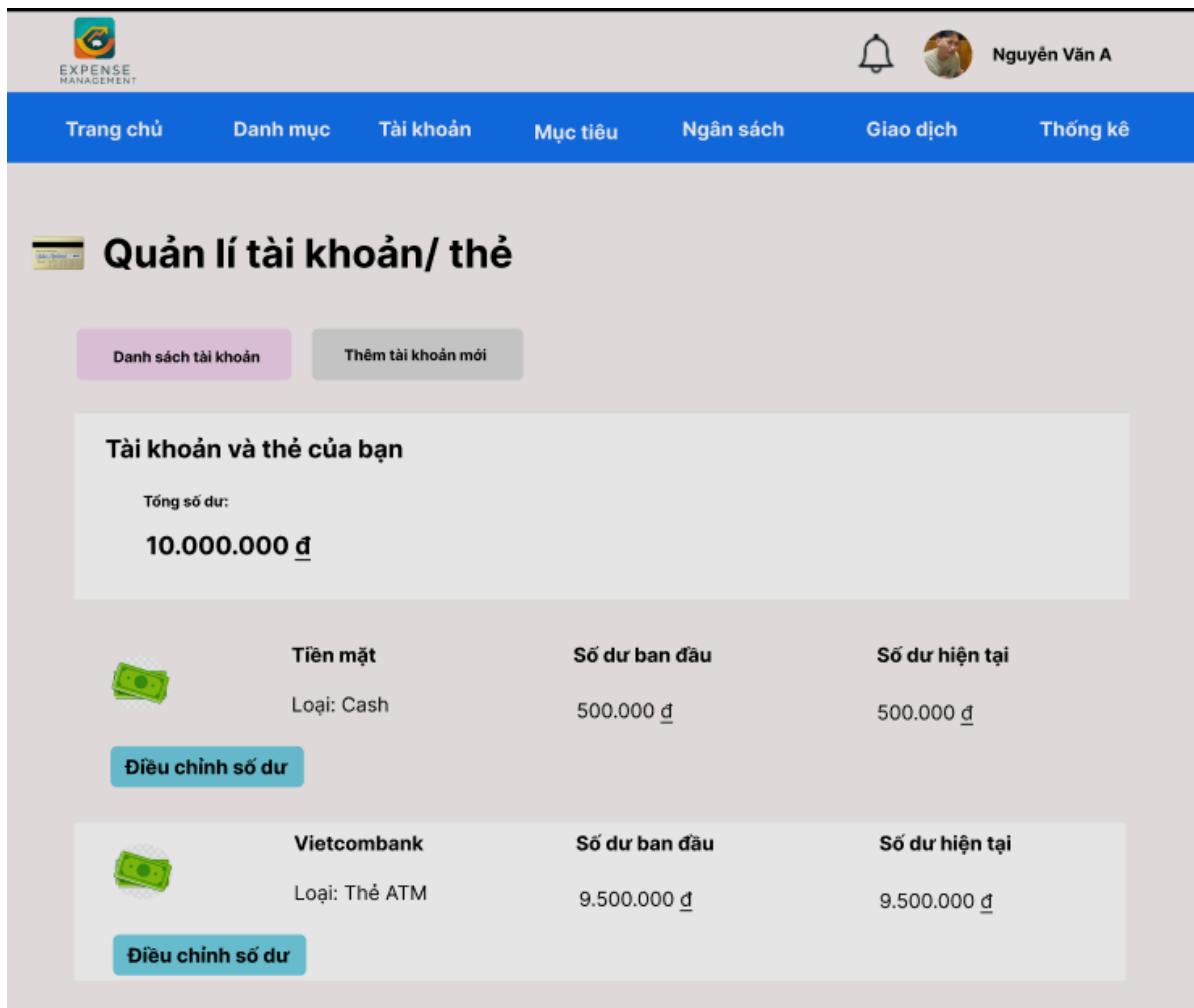
Giao diện tài khoản sẽ hiện phần content gồm 2 thành phần là khung Banner và khung Điều khiển tài chính.

- Khung Banner: cũng giống với các mục trước đó nhưng đổi thành nút Thêm nguồn tiền, cho phép bạn có thể dễ dàng thêm nguồn tiền từ các nguồn mà bạn muốn gồm có tiền mặt và từ tài khoản ngân hàng.

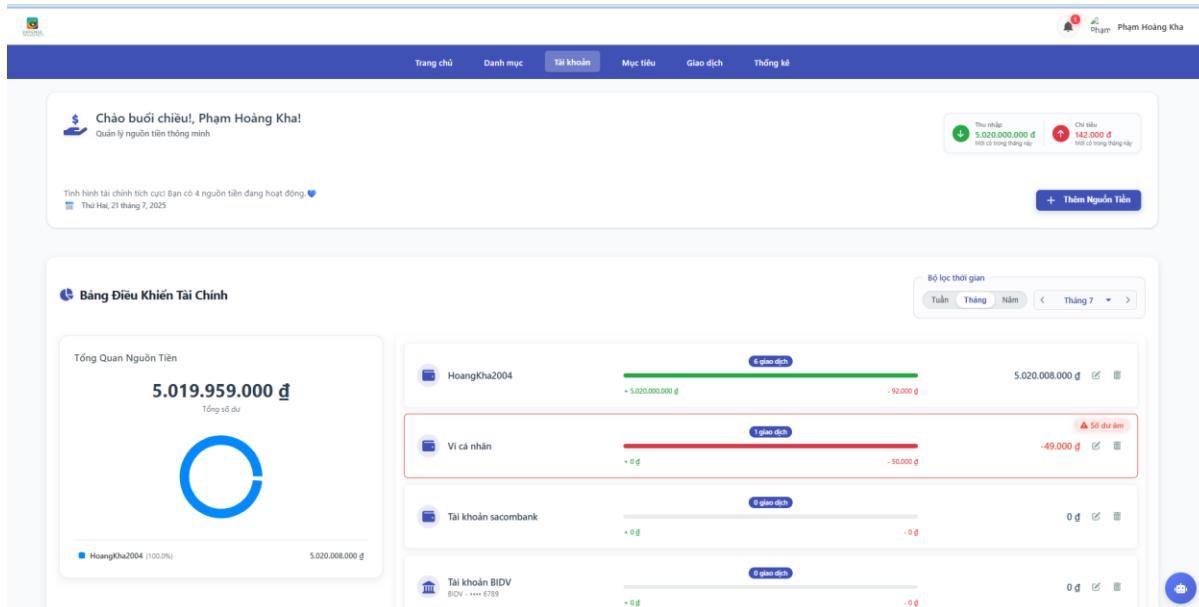


Hình 4.23 Giao diện thêm nguồn tiền

- Khung Điều khiển tài chính: bao gồm tròn hiển thị tổng quan số tiền mà nguồn đó đang giữ đồng thời hiển thị số lượng giao dịch đưa ra màu cảnh báo để người dùng dễ dàng trực quan bằng biểu đồ thanh ở mục bên phải, cho phép chỉnh sửa các nguồn tiền đó.



Hình 4.24 Giao diện trang tài khoản khi thiết kế trên Figma



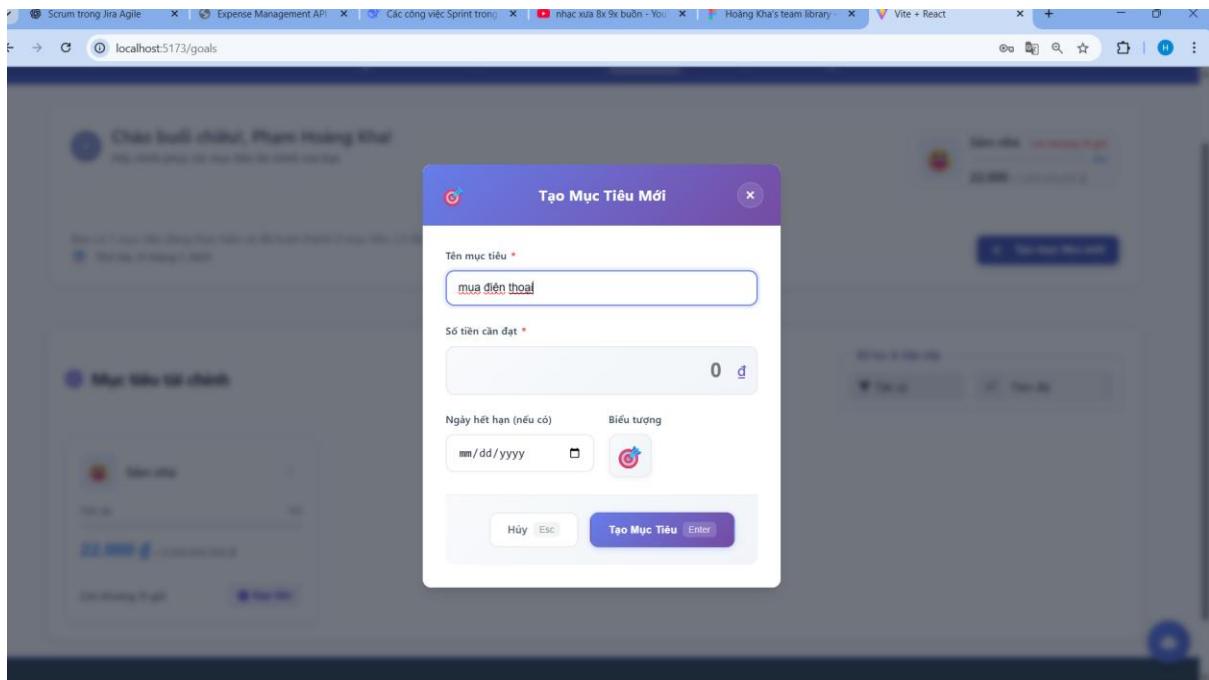
Hình 4.25 Giao diện trang tài khoản khi triển khai trên website

#### 4.4.5 Giao diện trang mục tiêu

Khi ấn sang phần danh mục của thanh Header thì website sẽ chuyển hướng sang link URL: <http://localhost:5173/goals>

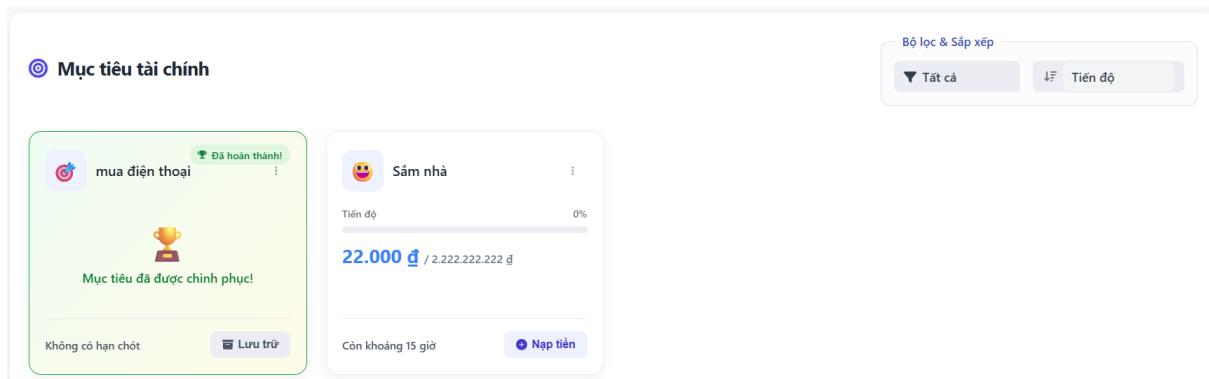
Giao diện mục tiêu sẽ hiện phần content gồm 2 thành phần là khung Banner và khung Phân tích danh mục:

- Khung Banner: cũng tương tự với các trang còn lại cho phép tạo mới một mục tiêu mà bạn muốn để xác định mục tiêu tiến độ đó hoàn thành hay chưa.

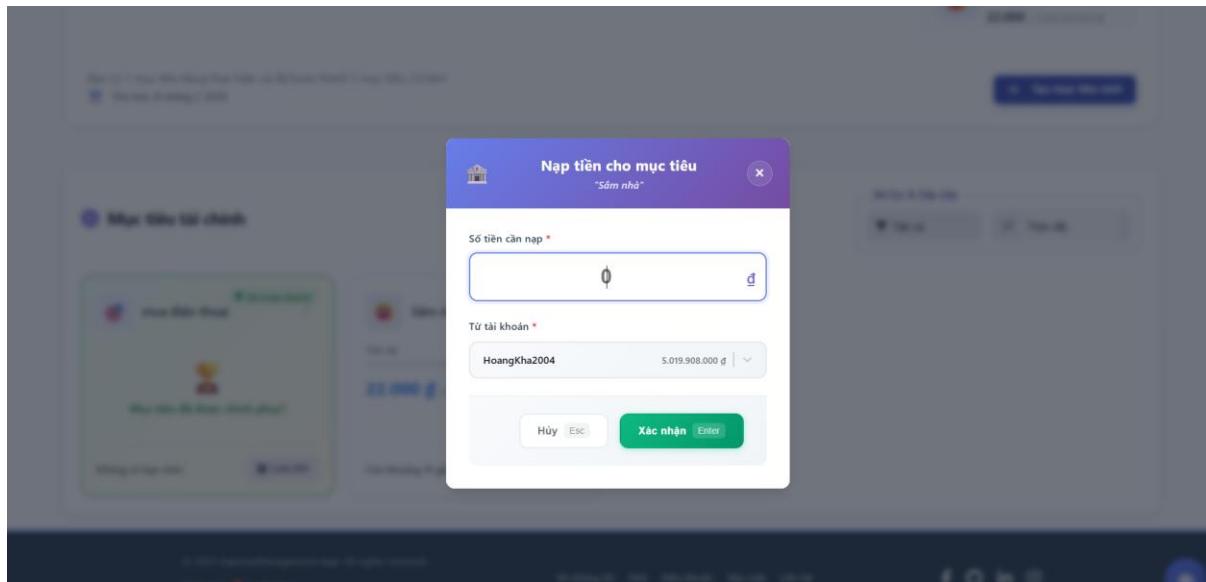


Hình 4.26 Giao diện thêm mục tiêu

- Phần mục tiêu tài chính: cho phép bạn xem đã hoàn thành mục tiêu hay chưa và tiến độ đã đạt bao nhiêu trong tổng số tiền mà bạn muốn đặt ra bằng biểu đồ thanh ngang, cho phép nạp số dư từ các nguồn tiền vào để hoàn thành mục tiêu. Cung cấp bộ lọc các danh mục theo từng loại: tiến độ, hạn chót và ngày tạo.



Hình 4.27 Phân mục tiêu tài chính



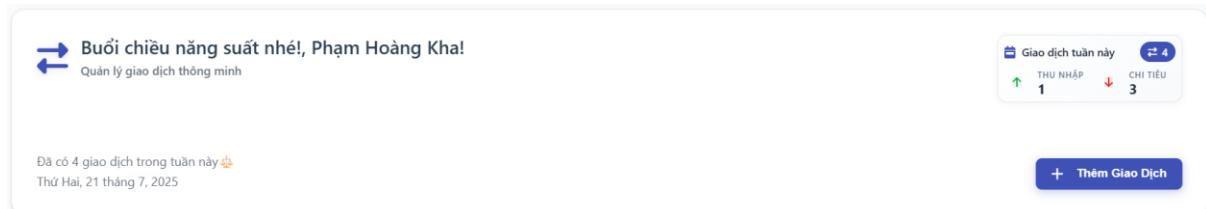
Hình 4.28 Phân thêm tiền cho mục tiêu

#### 4.4.6 Giao diện giao dịch

Khi ấn sang phần Tài khoản của thanh Header thì website sẽ chuyển hướng sang link URL: <http://localhost:5173/transactions>.

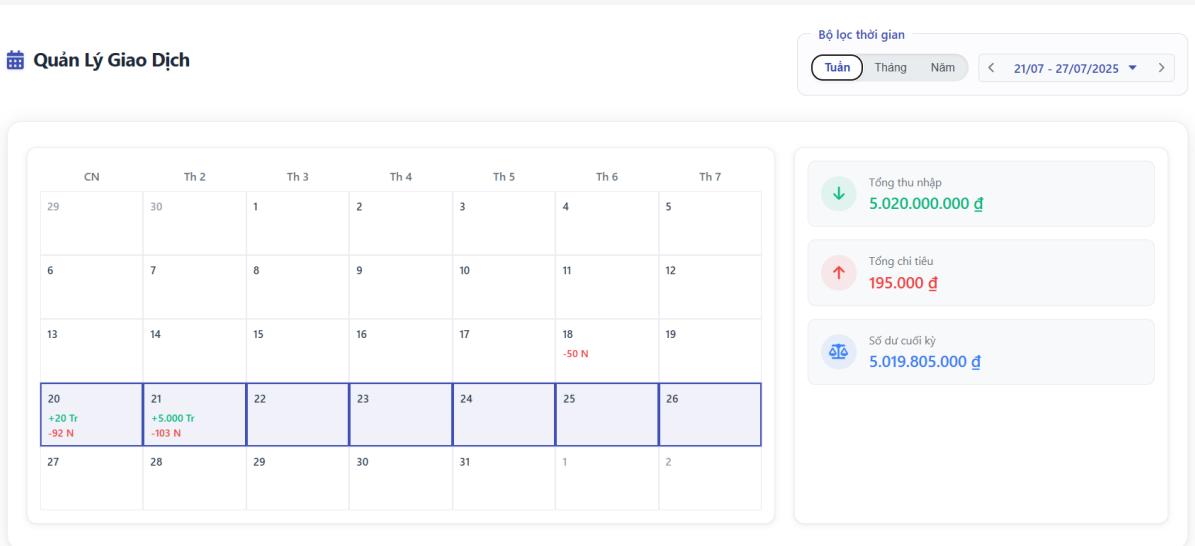
Giao diện tài khoản sẽ hiện phần content gồm 2 thành phần là khung Banner và khung Quản lý giao dịch, lịch sử và bộ lọc giao dịch.

- Khung Banner: cũng giống với khung Banner của trang chủ giao diện hài hòa giúp bạn tăng động hơn bằng những nội dung khích lệ.



Hình 4.29 Banner của trang giao dịch

- Khung Quản lý giao dịch: Hiển thị tổng số giao dịch và tổng số tiền giao dịch theo tuần, theo tháng và theo năm.



Hình 4.30 Phần quản lý giao dịch của trang giao dịch

- Khung lịch sử và bộ lọc giao dịch: Cho phép tìm kiếm lịch sử giao dịch theo từng yêu cầu mà người dùng cần thiết để đưa ra các giao dịch phù hợp và giúp người dùng tìm kiếm lịch sử giao dịch nhanh hơn.

**Bộ Lọc Giao Dịch**

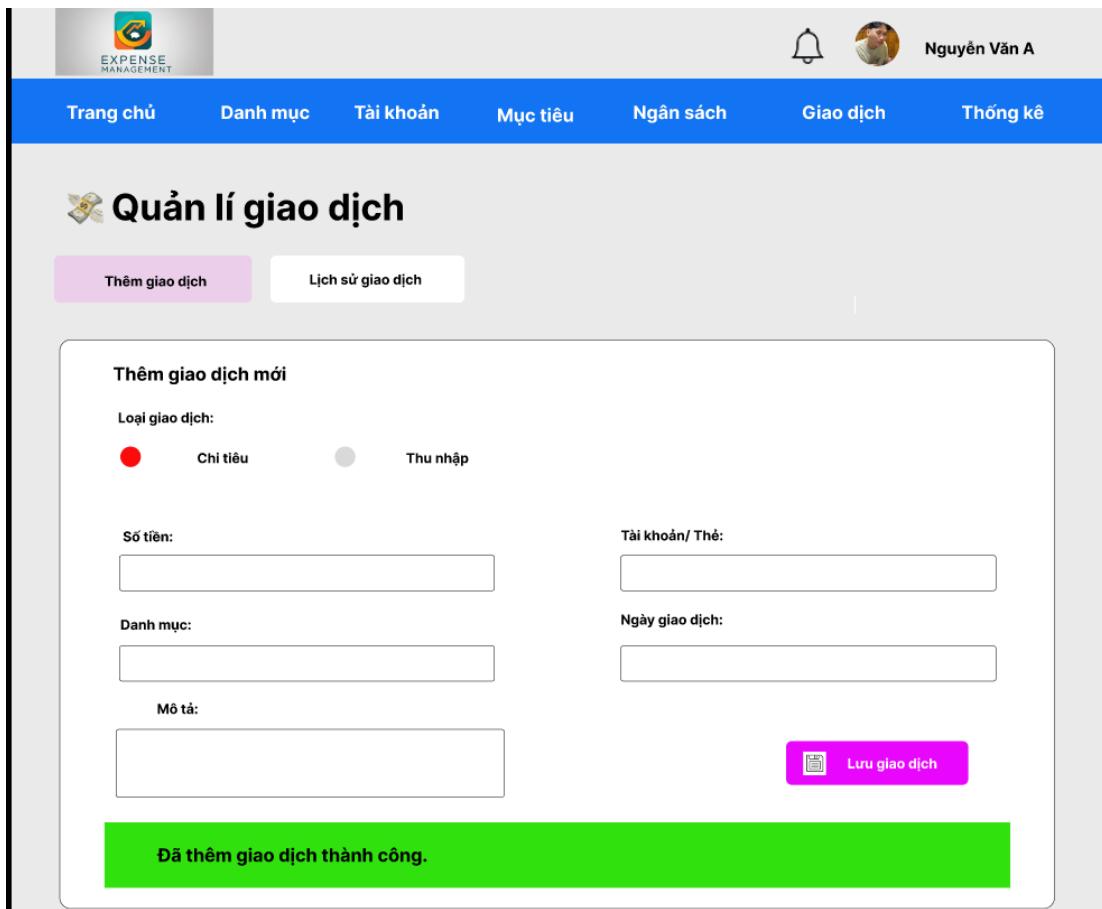
Lọc và tìm kiếm giao dịch theo tiêu chí mong muốn

**Lịch sử Giao dịch**

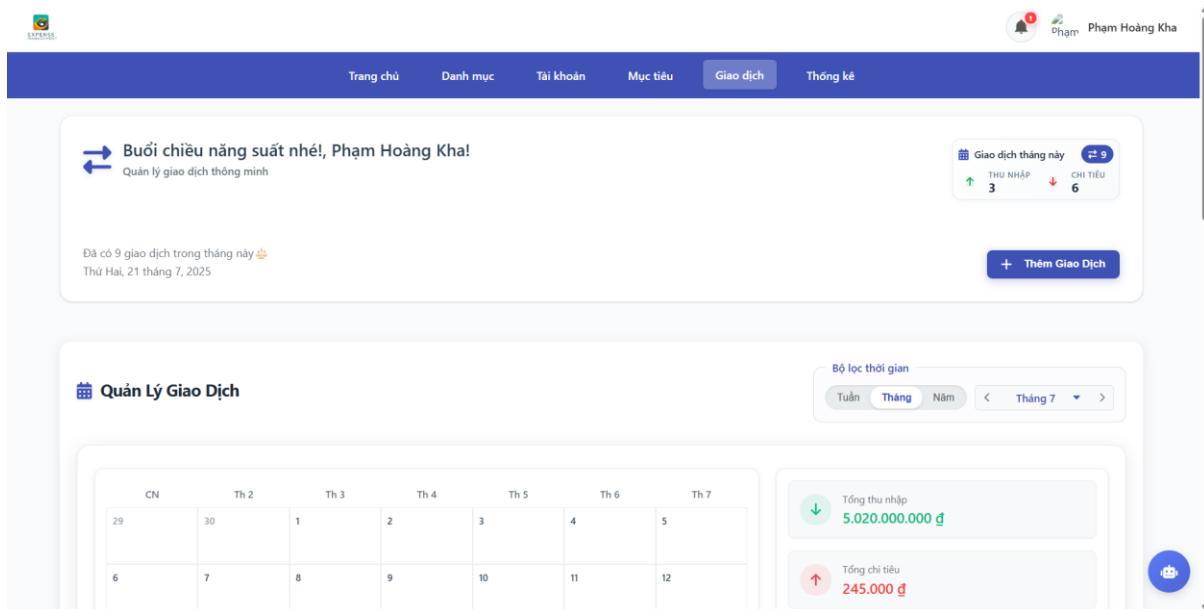
Thời gian	Danh mục	Mô tả	Tài khoản	Số tiền	Hành động
21/07/2025 Tạo lúc: 16:12	⌚ Sắm nhà	Tiết kiệm: Sắm nhà	HoangKhu2004	3.000 đ	[Edit, Delete]
21/07/2025 Tạo lúc: 16:04	⌚ mua điện thoại	Tiết kiệm: mua điện thoại	HoangKhu2004	100.000 đ	[Edit, Delete]
21/07/2025 Tạo lúc: 10:36	⭐ Lương	thu lương 5 tỷ	HoangKhu2004	5.000.000.000 đ	[Edit, Delete]
21/07/2025 Tạo lúc: 01:23	⌚ Sắm nhà	Tiết kiệm: Sắm nhà	HoangKhu2004	22.000 đ	[Edit, Delete]

Hình 4.31 Phần khung lịch sử giao dịch và bộ lọc

Trang quản lý giao dịch được triển khai đầy đủ và màu sắc phù hợp hơn khi triển khai vào thực tế.



Hình 4.32 Giao diện trang giao dịch khi thiết kế trên Figma

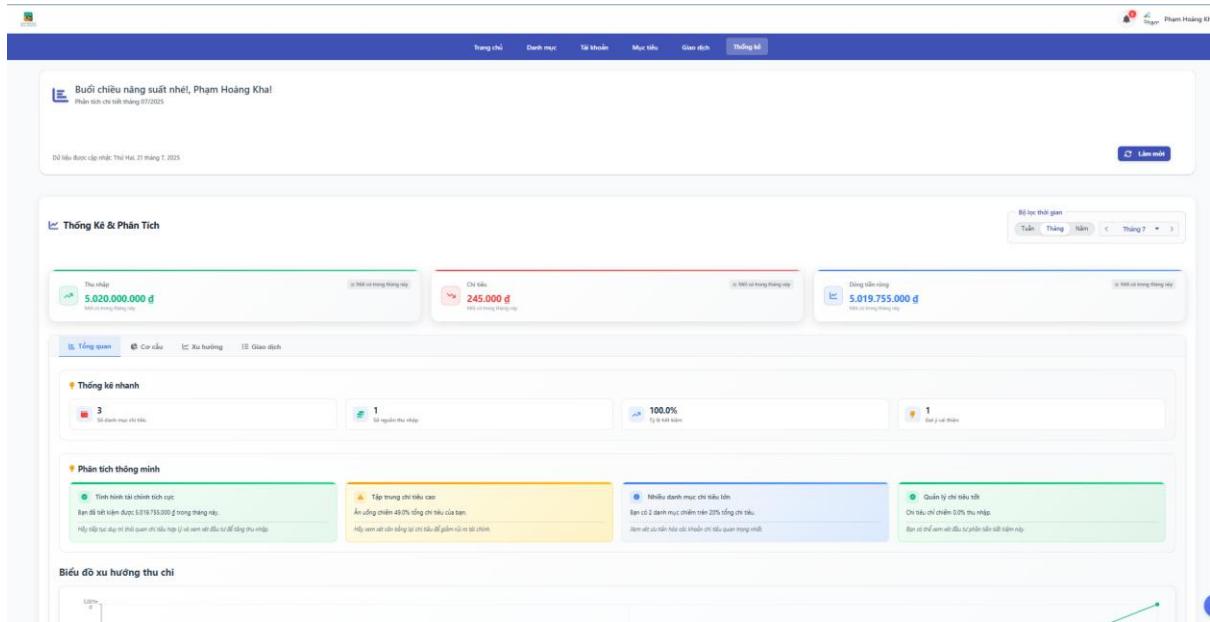


Hình 4.33 Giao diện trang giao dịch khi được triển khai thực tế

#### 4.4.7 Giao diện trang Thống kê

Giao diện thống kê đưa ra các phân tích và thống kê cùng biểu đồ để người dùng có thể tìm hiểu và kết xuất thông tin trực quan hơn.

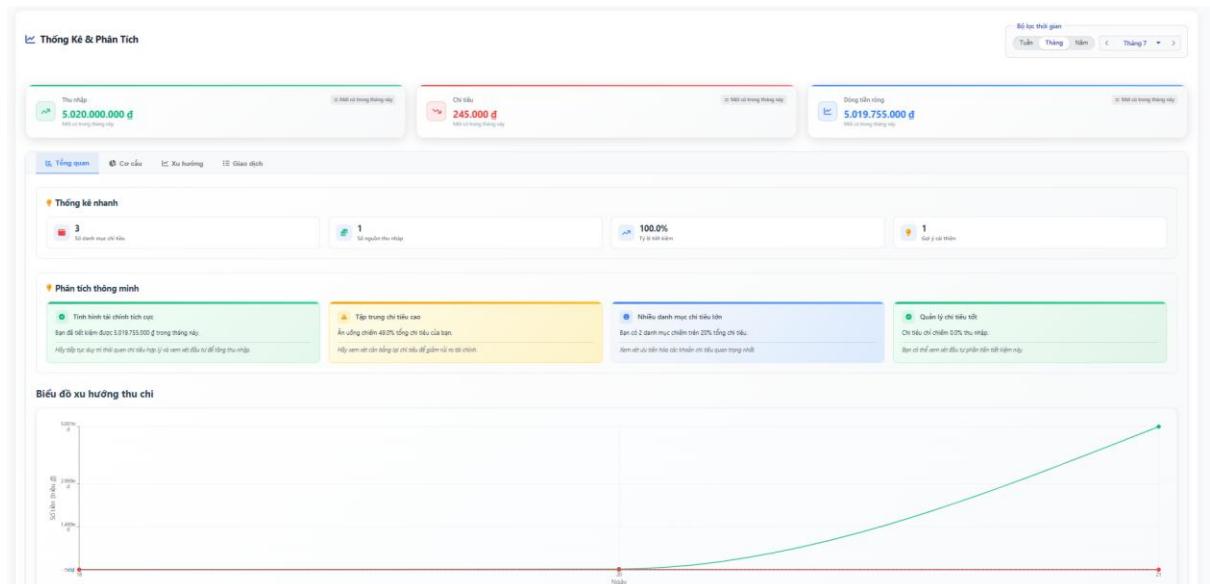
- Phần Banner: banner thân thiện hài hoà, phù hợp với ứng dụng.



Hình 4.34 Giao diện trang thống kê

- Phần Thống kê và Phân tích: Cho phép xem tổng thu nhập và chi tiêu theo tháng và dòng tiền ròng (số tiền chênh lệch giữa thu và chi trong tháng).

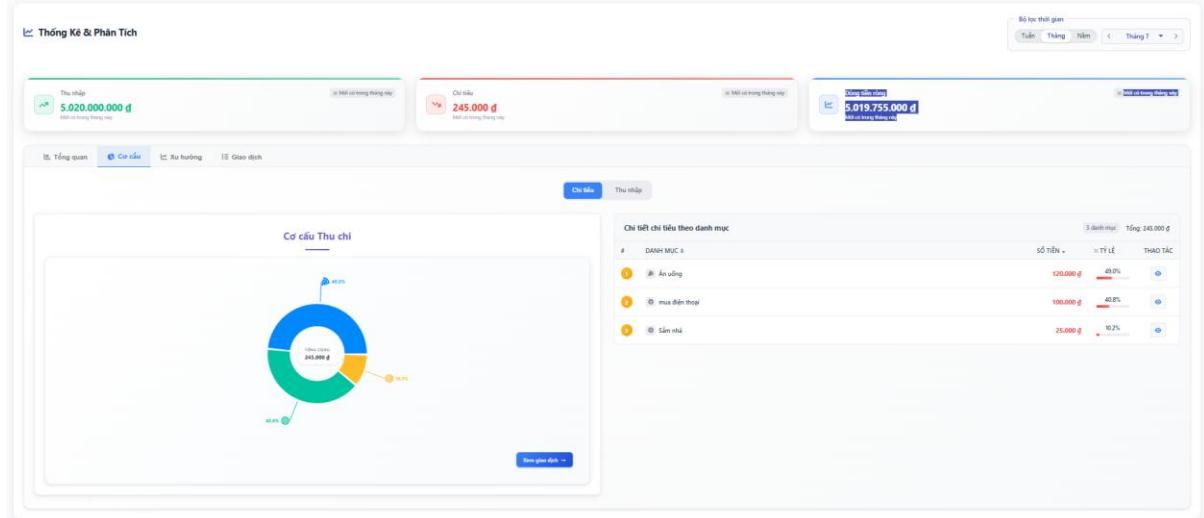
+ Với phần Tổng quan: sẽ cho ta biết thống kê nhanh và phân tích về nguồn tiền và số lượng sử dụng trong tháng và đưa ra biểu đồ thu chi (biểu đồ đường) cho người dùng trực quan hơn) và hệ thống vẫn sẽ ở link URL hiện tại của giao diện Thống kê là: <http://localhost:5173/statistics?tab=overview>



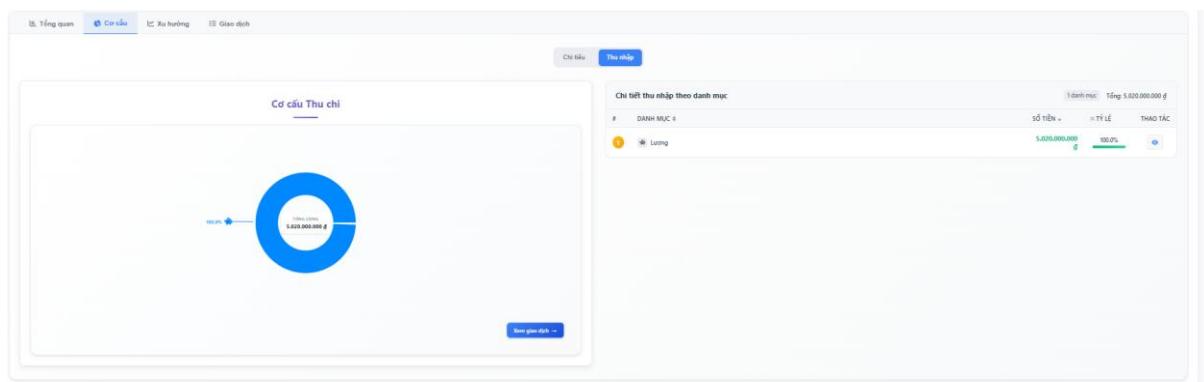
Hình 4.35 Phàn tổng quan trong khung thống kê và phân tích

+ Với mục cơ cấu: Dựa vào hệ thống sử dụng các component nên trang website được load nhanh chóng và chuyển sang đường link URL liên kết là:

<http://localhost:5173/statistics?tab=structure>. Thông kê thu nhập và chi tiêu cho biết cơ cấu chi tiêu một cách hoàn chỉnh và đầy đủ nhất bằng các biểu đồ pie và danh sách chi tiết các chi tiêu.



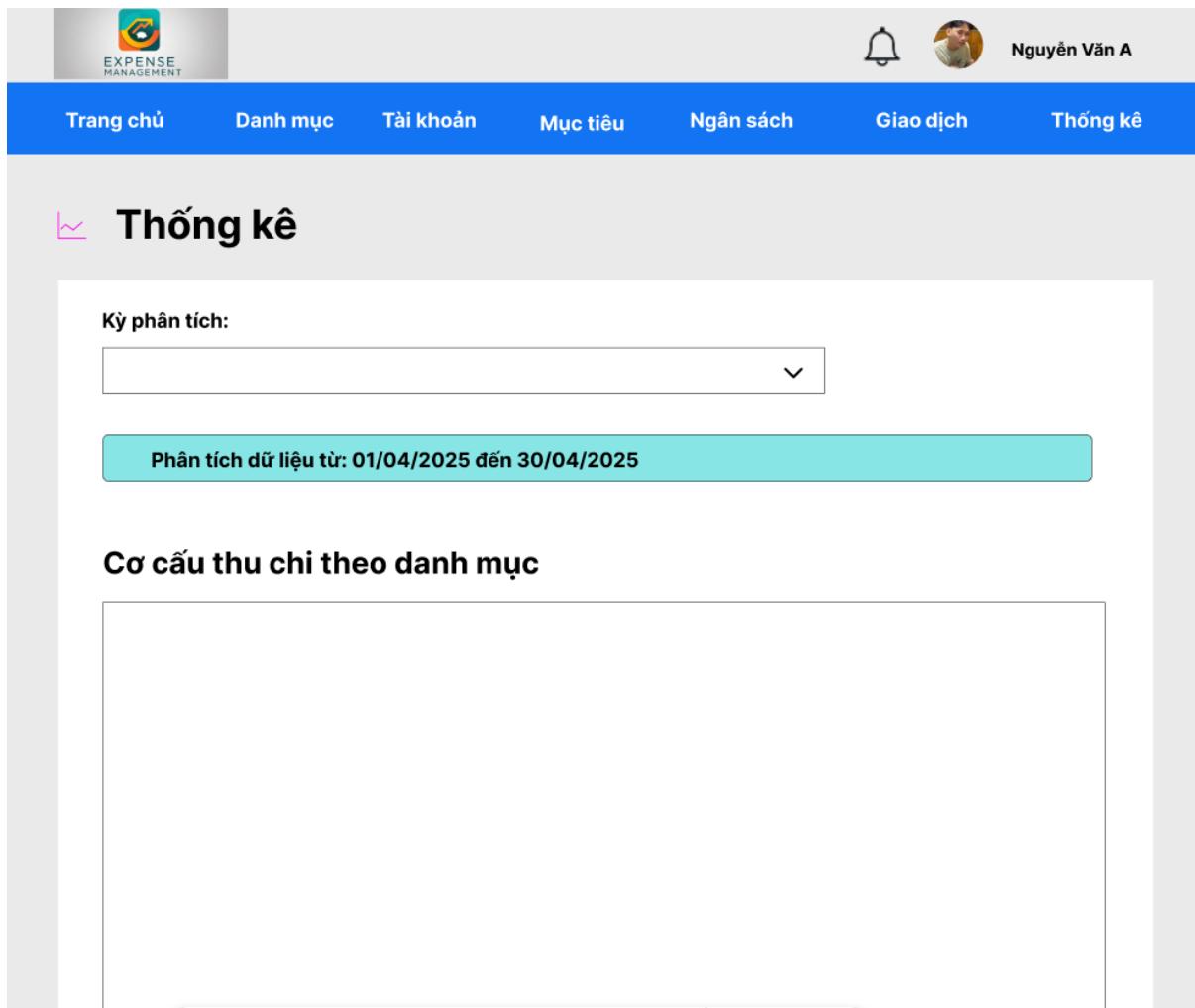
Hình 4.36 Phàn cơ cấu trong khung thống kê và phân tích của chi tiêu



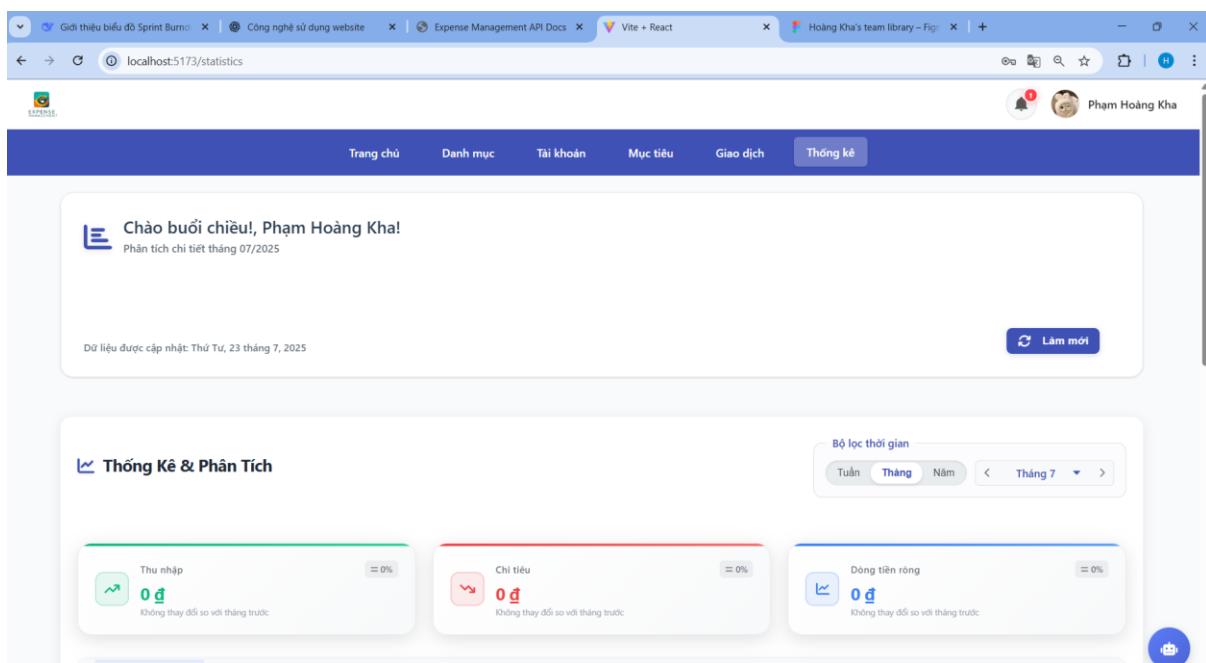
Hình 4.37 Phàn tổng quan trong khung thống kê và phân tích của thu nhập

+Với mục xu hướng: Nhanh chóng load lại chuyển liên kết sang URL: <http://localhost:5173/statistics?tab=trends&subTab=expense>. Đưa ra xu hướng chi tiêu bằng biểu đồ đường và biểu đồ tròn, và sẽ chuyển liên kết chi tiết đang trang danh mục nếu bạn muốn xem chi tiết: <http://localhost:5173/categories?focus=list>.

+ Với mục giao dịch: Nhanh chóng tải lại trang và chuyển liên kết sang URL: <http://localhost:5173/statistics?tab=transactions&subTab=expense> hiển thị đầy đủ các giao dịch mà bạn đã xử lý nhưng không cho phép chỉnh sửa.



Hình 4.38 Giao diện trang thống kê và phân tích khi được thiết kế trên Figma



Hình 4.39 Giao diện trang thống kê khi được triển khai thực tế

#### 4.4.8 Giao diện trang cá nhân

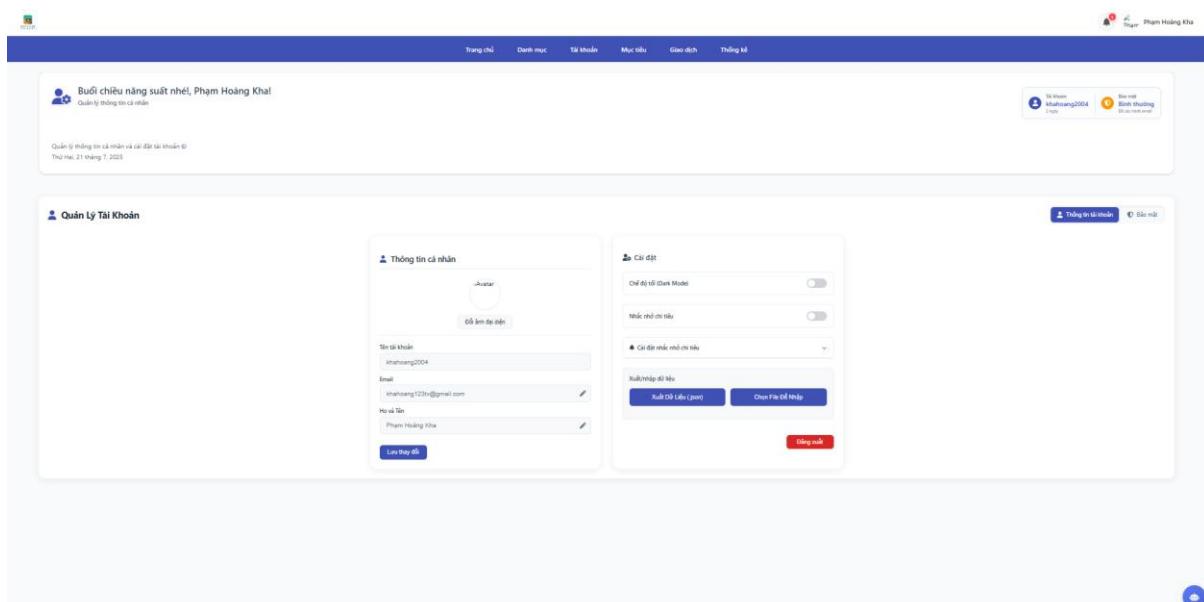
- Cho phép xem thông tin cá nhân mà bạn đã đăng ký, đồng thời cho phép chỉnh sửa thông tin cá nhân cũng như là avata đại diện của người dùng đó.

- Tùy chỉnh setting:

+ Cho phép chuyển đổi nền từ Light -> Dark và ngược lại phù hợp với sở thích và nhu cầu bảo vệ mắt của người dùng.

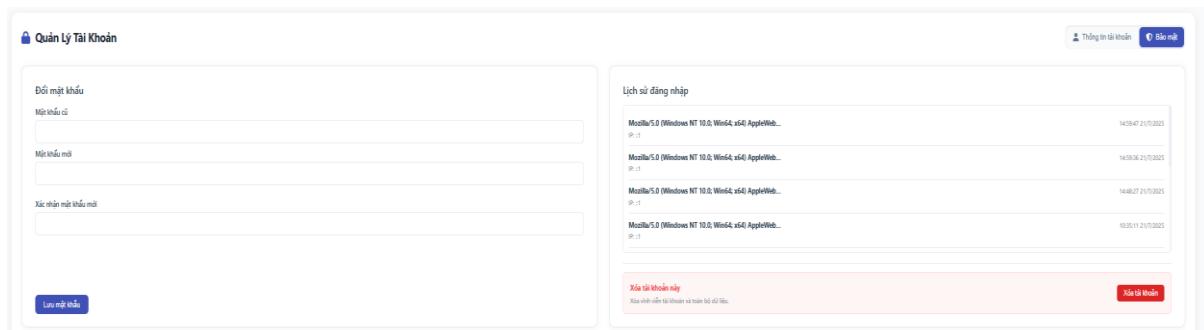
+ Cài đặt nhắc nhở chi tiêu gửi thông báo về cho người dùng.\

+ Cho phép xuất/ nhập dữ liệu một cách nhanh chóng phù hợp cho nhu cầu kết xuất thông tin và import dữ liệu vào của người dùng.



Hình 4.40 Giao diện trang cá nhân

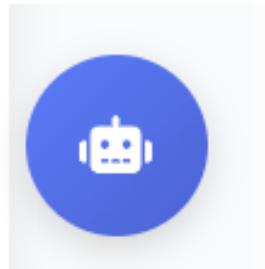
+ Tăng cường bảo mật với cho phép xem log làm việc àm người dùng đã đăng nhập trên thiết bị nào, đổi mật khẩu và xóa để bảo vệ thông tin khỏi các cuộc tấn công mạng.



Hình 4.41 Giao diện trang cá nhân ghi lại log đăng nhập

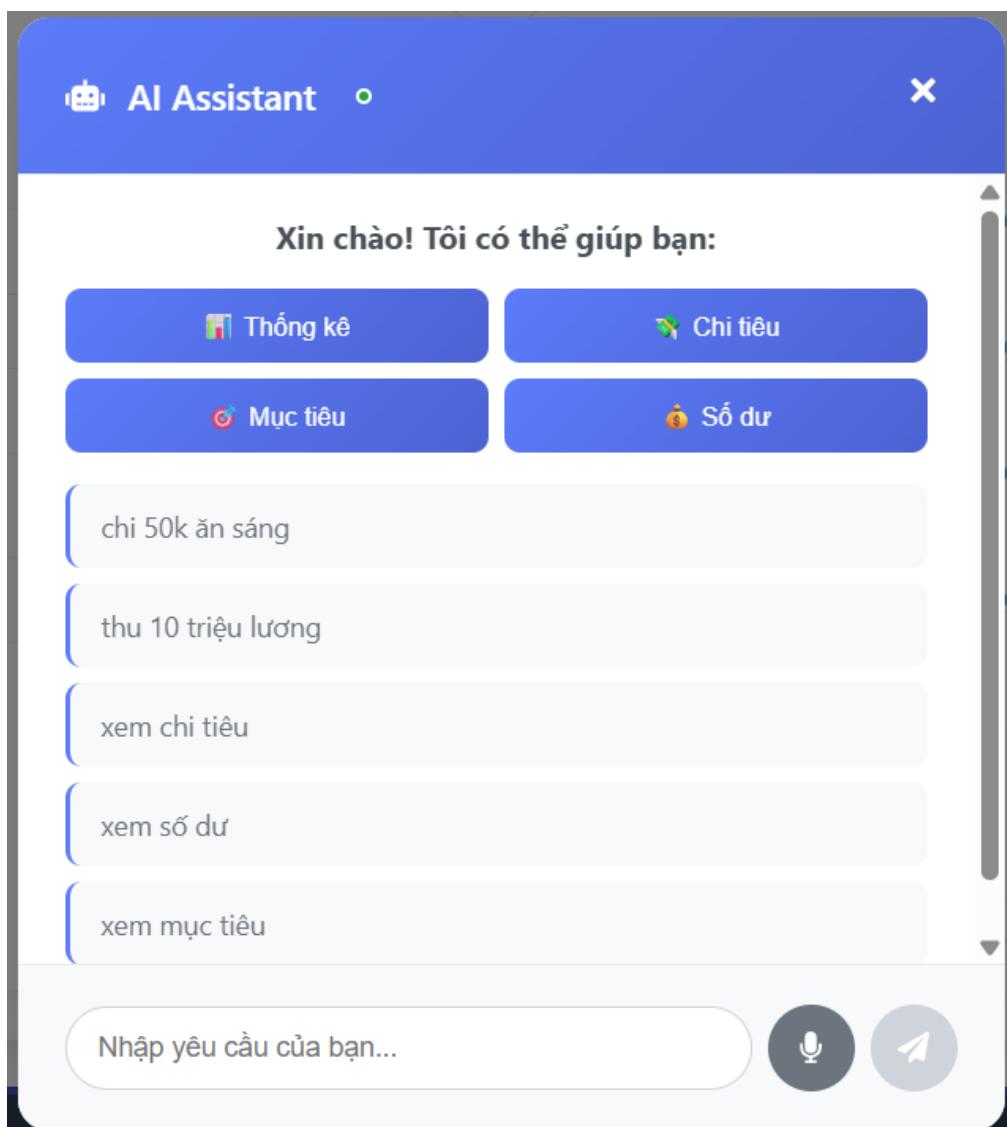
#### 4.4.9 Giao diện chat bot

Giao diện nổi hình chat bot ở dưới góc phải của mỗi giao diện cung cấp một trợ lí giao tiếp thông minh hỗ trợ cho người dùng, tư vấn nhập dữ liệu thu chi nhanh chóng một cách tự động.



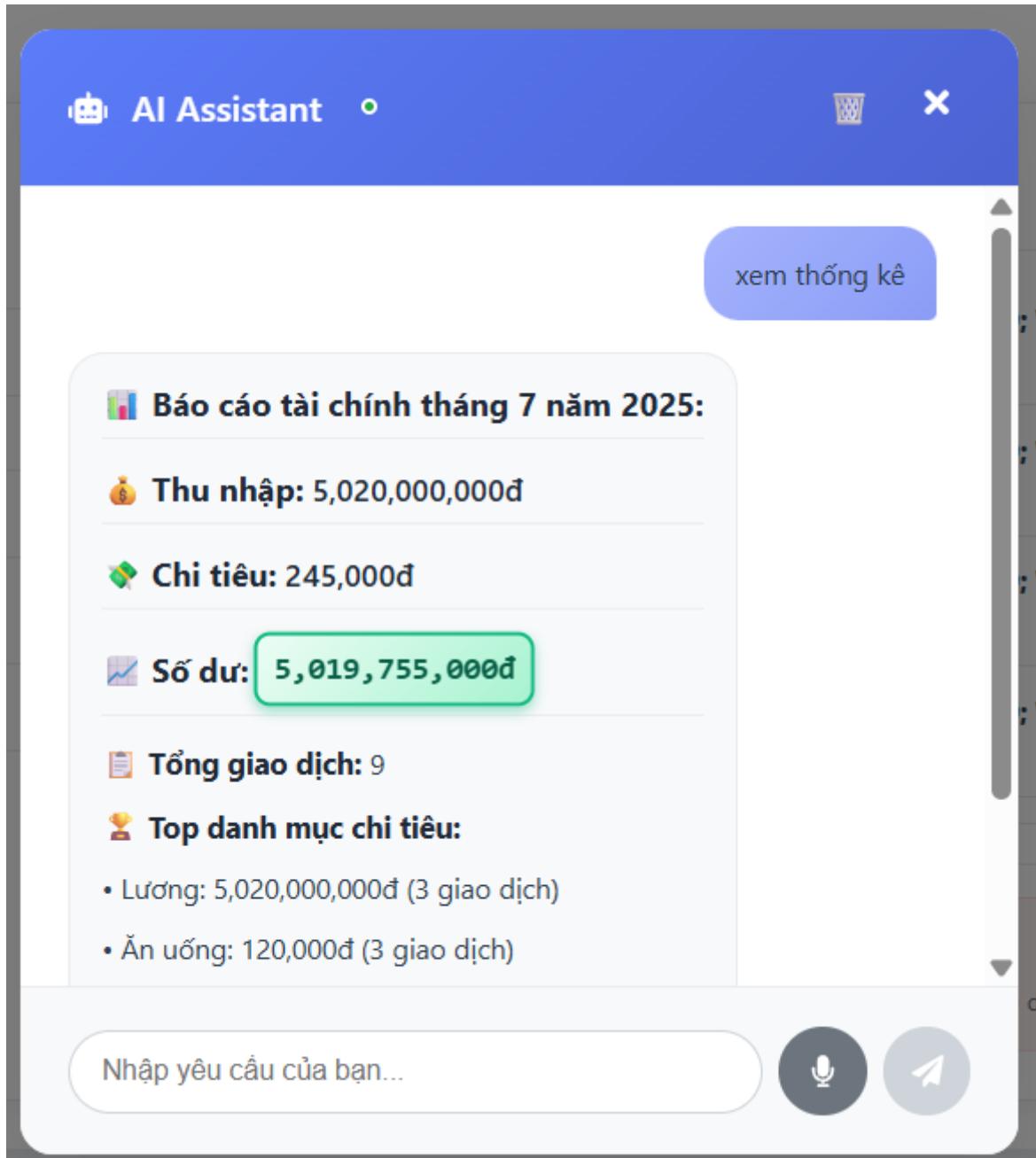
Hình 4.42 Ảnh đại diện của chat bot

+ Giao diện trò chuyện trực quan có những tip thông giúp bạn trao đổi và biết cách trò chuyện nhanh chóng



Hình 4.43 Giao diện chat khi được sử dụng

+ Cho phép hiển thị số liệu thống kê, tự động thu chi theo giao tiếp của bạn với trợ lí ảo.



Hình 4.44 Giao diện khi giao tiếp với chat và xác nhận giao dịch

## CHƯƠNG 5: TRIỂN KHAI VÀ CÔNG NGHỆ SỬ DỤNG

### 5.1 Các công nghệ sử dụng

Trong dự án “Xây dựng website quản lý chi tiêu” các công nghệ phát triển được gom nhóm theo vai trò và nhóm chức năng để đảm bảo hiệu quả tối ưu nhất trong thiết kế, triển khai, quản trị dự án.

#### 5.1.1 Nhóm Frontend (ReactJS)

React còn được gọi là ReactJS hoặc React.js, là 1 thư viện JavaScript mã nguồn mở được phát triển bởi đội ngũ kỹ sư đến từ Facebook; nó được giới thiệu vào năm 2011, tuy nhiên đến năm 2013 mới được giới thiệu cho cộng đồng lập trình viên. React nổi bật với khả năng xây dựng giao diện người dùng một cách hiệu quả, nhanh chóng và có khả năng tái sử dụng cao thông qua kiến trúc component-based.

##### a. Kiến trúc component-based:

- Ứng dụng được chia thành các component độc lập, mỗi component đảm nhiệm một chức năng cụ thể, giúp:

- + Tăng khả năng tái sử dụng (reusability): ví dụ như component danh sách giao dịch, biểu đồ thống kê, thanh điều hướng.
- + Dễ bảo trì, mở rộng: các thay đổi chỉ cần thực hiện tại component liên quan mà không ảnh hưởng đến toàn hệ thống.
- + Tối ưu hiệu năng: React chỉ re-render những component bị thay đổi nhờ Virtual DOM.

##### - Một số component chính:

- + TransactionList: Hiển thị danh sách các giao dịch thu/chi.
- + TransactionForm: Nhập liệu cho giao dịch mới (có validation).
- + ChartSummary: Hiển thị biểu đồ thống kê thu/chi.
- + Navbar và Sidebar: Quản lý điều hướng người dùng trong giao diện.

b. Quản lý trạng thái (State Management): Để đảm bảo dữ liệu giữa các component được đồng bộ và mượt mà, nhóm sử dụng:

##### - React Hooks:

- + useState: quản lý trạng thái nội bộ của từng component, chẳng hạn như trạng thái nhập liệu, thông báo lỗi.

+ `useEffect`: xử lý các tác vụ bất đồng bộ như gọi API để lấy danh sách giao dịch sau khi giao diện render.

- **Context API:** Được sử dụng để quản lý trạng thái toàn cục như:

+ Dữ liệu người dùng sau khi đăng nhập.

+ Thông tin danh mục, giao dịch được chia sẻ giữa các component.

+ Trạng thái loading, thông báo toàn hệ thống (global alert/snackbar).

+ Context API giúp loại bỏ việc truyền props rườm rà giữa các component con – cha, tăng tính gọn gàng và hiệu quả khi ứng dụng mở rộng.

c. Giao diện người dùng (UI/UX)

Để xây dựng giao diện đẹp, thân thiện và tương thích trên nhiều loại thiết bị, nhóm sử dụng:

- **Material-UI (MUI):**

+ Thư viện giao diện theo chuẩn Material Design do Google phát triển.

+ Cung cấp sẵn các component giao diện như Button, Dialog, TextField, Card,...

với khả năng tùy biến cao.

+ Đảm bảo tính responsive (tương thích di động, tablet, desktop).

- **Responsive Design:**

+ Áp dụng nguyên tắc thiết kế responsive bằng Grid System và Media Queries từ MUI.

+ Giao diện tự điều chỉnh bố cục khi thay đổi độ phân giải, đảm bảo trải nghiệm người dùng ổn định trên mọi thiết bị.

d. Trực quan hóa dữ liệu

- **Chart.js:**

+ Thư viện được sử dụng để trực quan hóa dữ liệu thu/chi dưới dạng biểu đồ cột, biểu đồ tròn hoặc biểu đồ đường.

+ Giao diện biểu đồ giúp người dùng dễ dàng theo dõi xu hướng chi tiêu theo tháng, danh mục hoặc so sánh giữa các giai đoạn.

- Biểu đồ được tích hợp trực tiếp trong Dashboard, cập nhật động theo dữ liệu từ API backend, giúp tăng giá trị sử dụng và trực quan hóa thông tin tài chính hiệu quả.

### 5.1.2 Nhóm Backend (Node.js/Express)

Backend của hệ thống được xây dựng dựa trên nền tảng Node.js kết hợp với framework ExpressJS, nhằm mục tiêu cung cấp một hệ thống xử lý dữ liệu nhanh chóng, mở rộng linh hoạt và dễ dàng tích hợp với frontend thông qua các API RESTful.

a. Tổng quan công nghệ

- **Node.js:**

Là môi trường chạy JavaScript phía server, có khả năng xử lý bát đồng bộ và chịu tải tốt nhờ cơ chế event-driven, rất phù hợp với các ứng dụng web có nhiều thao tác I/O như: truy vấn cơ sở dữ liệu, xử lý request song song.

- **ExpressJS:**

Là một framework tối giản nhưng mạnh mẽ của Node.js, hỗ trợ xây dựng các tuyến đường (routes), middleware và quản lý logic nghiệp vụ rõ ràng, hiệu quả.

b. Kiến trúc API RESTful

Backend được thiết kế theo kiến trúc **RESTful**, phân tách rõ ràng các chức năng thông qua các endpoint có cấu trúc thống nhất. Dữ liệu được trao đổi giữa frontend và backend dưới dạng **JSON**.

Các nhóm API chính bao gồm:

- **Xác thực và phân quyền:**

- + POST /api/auth/register: Đăng ký tài khoản mới.
- + POST /api/auth/login: Đăng nhập, trả về token JWT.
- + GET /api/auth/profile: Lấy thông tin người dùng hiện tại.

- **Quản lý giao dịch:**

- + GET /api/transactions: Lấy danh sách giao dịch của người dùng.
- + POST /api/transactions: Tạo mới giao dịch.
- + PUT /api/transactions/:id: Cập nhật thông tin giao dịch.
- + DELETE /api/transactions/:id: Xóa giao dịch.

- **Quản lý danh mục:**

- + GET /api/categories: Lấy danh sách danh mục thu/chi.
- + POST /api/categories: Thêm danh mục mới.
- + PUT /api/categories/:id: Chính sửa danh mục.
- + DELETE /api/categories/:id: Xóa danh mục.

**- Mục tiêu tài chính:**

- + GET /api/goals: Lấy danh sách mục tiêu.
- + POST /api/goals: Tạo mục tiêu mới.
- + PUT /api/goals/:id: Cập nhật tiến độ hoặc thông tin mục tiêu.

**- Thống kê và báo cáo:**

- + GET /api/statistics/monthly: Tổng hợp thu/chi theo tháng.
- + GET /api/statistics/yearly: Tổng hợp thu/chi theo năm.
- + GET /api/statistics/by-category: Phân tích chi tiêu theo danh mục.

c. Cơ chế xác thực và bảo mật

Hệ thống sử dụng JWT (JSON Web Token) để xác thực người dùng:

- + Sau khi đăng nhập thành công, server tạo một access token và gửi về cho frontend.
- + Token này sẽ được lưu trữ phía client (thường là localStorage) và gửi kèm trong header Authorization của các request sau.

+ Ở phía backend, middleware kiểm tra và giải mã token để xác định quyền truy cập trước khi xử lý logic nghiệp vụ.

Việc sử dụng JWT giúp:

- + Đảm bảo an toàn khi truy cập dữ liệu cá nhân.
- + Tăng hiệu suất vì không cần truy vấn database mỗi lần xác thực.
- + Dễ tích hợp với frontend SPA như React.

d. Middleware

ExpressJS hỗ trợ middleware, cho phép xử lý logic trung gian trong pipeline request-response. Nhóm đã xây dựng các middleware như:

- + Xác thực token: Kiểm tra token JWT để xác định người dùng hợp lệ.
- + Kiểm tra quyền truy cập: Một số hành động chỉ dành cho chủ sở hữu dữ liệu.
- + Xử lý lỗi tập trung: Tự động log lỗi và trả về thông báo rõ ràng cho người dùng.

Middleware giúp backend gọn gàng, tách biệt giữa logic nghiệp vụ và logic bảo mật hoặc kiểm tra điều kiện.

### **5.1.3 Nhóm Cơ sở dữ liệu (MongoDB)**

a. Lý do lựa chọn MongoDB

+ MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL dạng document phổ biến, cho phép lưu trữ dữ liệu dưới dạng BSON (Binary JSON). Hệ thống được lựa chọn sử dụng MongoDB vì những ưu điểm sau:

+ Linh hoạt trong việc định nghĩa dữ liệu: Không giống như các hệ quản trị CSDL quan hệ (RDBMS), MongoDB không bắt buộc phải có schema cố định, rất phù hợp với mô hình dữ liệu có thể thay đổi theo thời gian như các mục tiêu tài chính, phân loại giao dịch,...

+ Tốc độ truy xuất nhanh: Với khả năng xử lý tập trung vào đọc/ghi nhanh, MongoDB đáp ứng tốt yêu cầu về hiệu năng khi người dùng truy vấn nhiều lần trong ngày để xem báo cáo thu/chi, thống kê, biểu đồ,...

+ Tích hợp tốt với Node.js: MongoDB hoạt động rất hiệu quả khi đi cùng Node.js, nhờ vào bộ thư viện ORM như Mongoose, giúp việc thao tác dữ liệu trở nên trực quan và đồng bộ về mặt ngôn ngữ (JavaScript từ frontend đến backend đến database).

### b. Các collection chính trong hệ thống

Dữ liệu được tổ chức thành các **collection**, tương đương với các bảng trong RDBMS, với mỗi **document** trong collection là một bản ghi dữ liệu.

#### 1. users – Thông tin người dùng

```
{  
  "_id": ObjectId,  
  "username": "hoangkha",  
  "email": "example@gmail.com",  
  "password": "hashed_password",  
  "createdAt": Date,  
  "avatar": "url",  
  "roles": ["user"]  
}
```

+ Chứa dữ liệu cá nhân và thông tin đăng nhập.

+ Mật khẩu được mã hóa bằng **bcrypt** trước khi lưu vào CSDL.

#### 2. transactions – Giao dịch thu/chi

```
{  
  "_id": ObjectId,  
  "userId": ObjectId,  
  "type": "income" | "expense",  
  "amount": 100000,  
  "categoryId": ObjectId,  
  "note": "Tiền lương tháng 7",  
  "date": ISODate("2025-07-01T00:00:00Z")  
}
```

+ Mỗi giao dịch liên kết với một người dùng (userId) và một danh mục (categoryId).

+ Có phân biệt loại thu nhập (income) và chi tiêu (expense).

+ Trường note giúp mô tả thêm chi tiết về giao dịch.

### 3. categories – Danh mục thu/chi

```
{  
    "_id": ObjectId,  
    "userId": ObjectId,  
    "name": "Ăn uống",  
    "type": "expense",  
    "color": "#ff5733",  
    "icon": ""  
}
```

+ Các danh mục tùy chỉnh cho từng người dùng (có thể trùng tên nhưng khác userId).

+ Có thể chia thành danh mục thu hoặc chi.

### 4. goals – Mục tiêu tài chính

```
{  
    "_id": ObjectId,  
    "userId": ObjectId,  
    "title": "Tiết kiệm mua laptop",  
    "targetAmount": 20000000,  
    "currentAmount": 5000000,  
    "dueDate": ISODate("2025-12-31T00:00:00Z"),  
    "note": " "  
}
```

+ Hỗ trợ người dùng lập kế hoạch tài chính dài hạn.

+ Có thể cập nhật tiến độ dần theo thời gian.

### c. Mongoose – ORM cho MongoDB

+ Để tương tác với MongoDB một cách hiệu quả và dễ bảo trì, nhóm sử dụng Mongoose, một thư viện ORM (Object Relational Mapping) giúp:

+ Định nghĩa schema rõ ràng: Mỗi collection có một file định nghĩa schema riêng, quy định kiểu dữ liệu, ràng buộc (validation), và mối quan hệ giữa các bảng.

+ Tự động hóa các thao tác CRUD: Mongoose cung cấp các phương thức như .find(), .create(), .updateOne(), .deleteMany() giúp thao tác với MongoDB đơn giản hơn.

+ Tích hợp middleware và hook: Cho phép chèn các xử lý trung gian (pre/post) như mã hóa mật khẩu trước khi lưu, kiểm tra hợp lệ dữ liệu, tính toán tự động,...

d. Các chỉ mục và tối ưu truy vấn

+ Các collection quan trọng như transactions và users được đánh index theo userId và date, giúp tối ưu các truy vấn thống kê theo tháng/năm hoặc lọc giao dịch theo người dùng.

+ Với các API thống kê phức tạp, MongoDB sử dụng Aggregation Framework để gom nhóm (\$group), lọc (\$match), và tính toán tổng số tiền theo thời gian hoặc danh mục.

```
[  
  { $match: { userId: ..., type: "expense" } },  
  { $group: { _id: { month: { $month: "$date" } }, total: { $sum: "$amount" } } }  
]
```

e. Bảo mật và sao lưu dữ liệu

+ Dữ liệu nhạy cảm (mật khẩu, token) được **mã hóa và xác thực kỹ lưỡng**.

+ MongoDB được cấu hình chạy trong môi trường Docker, hỗ trợ backup/restore định kỳ.

+ Có thể triển khai MongoDB Atlas (cloud) để đảm bảo uptime và khả năng mở rộng cao hơn khi cần.

#### 5.1.4 Nhóm Trí Tuệ Nhân Tạo (Gemini API)

a. Công nghệ sử dụng

Trong quá trình phát triển hệ thống quản lý chi tiêu, nhóm đã tích hợp Gemini API – một nền tảng Trí tuệ nhân tạo do Google phát triển, chuyên về Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP). API này hỗ trợ tạo ra các mô hình hội thoại thông minh có khả năng hiểu và phản hồi câu hỏi của người dùng một cách tự nhiên và linh hoạt.

Gemini có thể được tích hợp thông qua RESTful API hoặc sử dụng SDK chính thức từ Google, cụ thể là thư viện @google/generative-ai, giúp quá trình tương tác và kết nối với hệ thống backend trở nên dễ dàng và hiệu quả hơn.

b. Mục đích tích hợp

Việc tích hợp Gemini API trong hệ thống có những mục tiêu cụ thể như sau:

+ Hỗ trợ người dùng bằng Chatbot thông minh: Cho phép người dùng đặt các câu hỏi dạng tự nhiên.

+ Phân tích xu hướng chi tiêu: AI có khả năng xử lý dữ liệu chi tiêu theo thời gian và đưa ra nhận xét, gợi ý, hoặc cảnh báo.

+ Nâng cao trải nghiệm người dùng: Việc tương tác bằng ngôn ngữ tự nhiên giúp người dùng cảm thấy thoải mái hơn so với việc phải tra cứu và thao tác qua nhiều menu thủ công.

#### c. Cách thức triển khai

Để tích hợp Gemini API vào hệ thống, nhóm đã thực hiện các bước triển khai sau:

#### 1. Đăng ký và lấy API Key từ Google AI Studio

- + Truy cập trang Google AI Studio, đăng nhập bằng tài khoản Google.
- + Tạo một project mới và sinh khóa API để truy cập dịch vụ Gemini.
- + Cấu hình biến môi trường GEMINI\_API\_KEY trong file .env để bảo mật.

2. Kết nối với Backend Node.js: Nhóm sử dụng thư viện chính thức @google/generative-ai để thực hiện các truy vấn NLP. Dưới đây là đoạn mã mẫu được sử dụng:

```
const { GoogleGenerativeAI } = require("@google/generative-ai");

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);

async function askGemini(prompt) {
  const model = genAI.getGenerativeModel({ model: "gemini-pro" });
  const result = await model.generateContent(prompt);
  const response = await result.response;
  return response.text();
}
```

#### 3. Xử lý yêu cầu từ Frontend

+ Frontend (React) cung cấp giao diện chat để người dùng nhập câu hỏi tự nhiên.

+ Câu hỏi được gửi đến API backend, nơi hàm askGemini(prompt) xử lý nội dung và gửi đến Gemini.

+ Kết quả từ Gemini được backend trả về giao diện người dùng dưới dạng phản hồi thân thiện.

**4. Giao tiếp giữa hệ thống và AI** đảm bảo phản hồi thời gian thực, có tính cá nhân hóa theo dữ liệu người dùng đã lưu trong hệ thống (như các giao dịch trước đó, mục tiêu chi tiêu, v.v.).

d. Lợi ích mang lại

+ Tối ưu hóa trải nghiệm người dùng thông qua tương tác nhanh chóng, dễ hiểu và không cần kỹ năng kỹ thuật cao.

+ Tiết kiệm thời gian tra cứu và tổng hợp báo cáo thủ công.

+ Mở rộng tiềm năng hệ thống theo hướng cá nhân hóa và thông minh hóa, phù hợp xu thế công nghệ hiện đại.

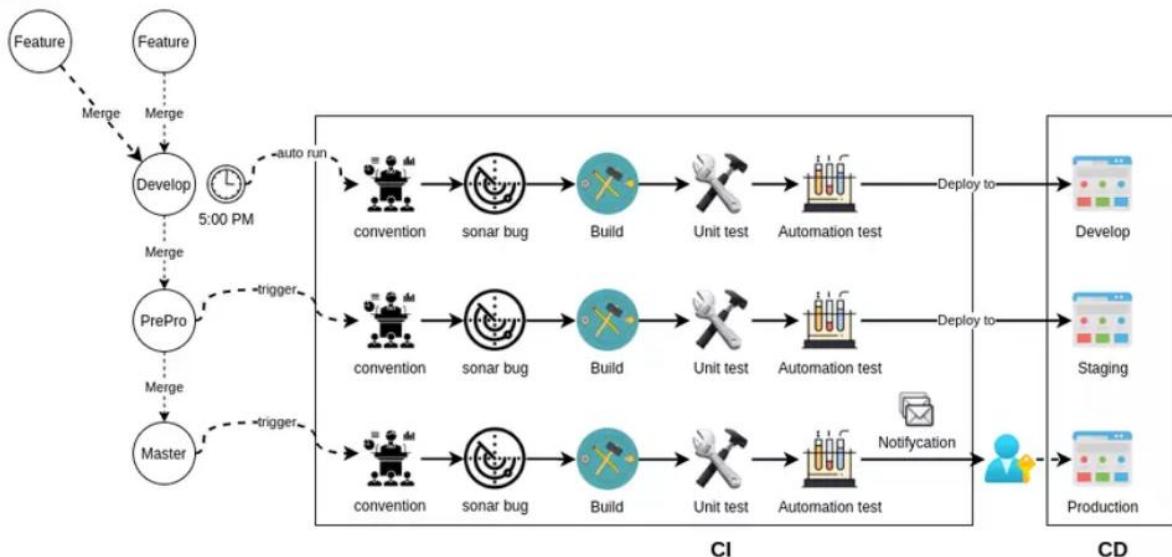
## 5.2 CI/CD với Github Actions

### 5.2.1 CI/CD là gì?

Có rất nhiều khái niệm cả chính thống và cả tà đạo về CI/CD nhưng CI/CD hiểu đơn giản nhất là CI (Continuous Integration) và CD (Continuous Delivery) tức là quá trình tích và triển khai hợp nhanh và liên tục.

Về mặt khái niệm là vậy nhưng về mặt triển khai thì CI/CD là quá trình tự động thực hiện các quá trình build, test, release, deploy khi có các trigger như commit/merge code lên một branch định sẵn hoặc có thể là tự động chạy theo một lịch cố định.

Dưới đây là một mô hình mà một công ty đã apply cho dự án:



Hình 5.1 Quy trình CI/CD của một công ty mẫu

+ Khi hoàn thành một feature thì teamlead tạo merge request rồi merge vào branch develop, đúng 5h chiều hàng ngày hệ thống sẽ tự động build, test, quét sonar.... và deploy lên develop environment (quá trình này là CD), không trigger merger code để deploy với branch này vì code được merge vào đây liên tục nếu trigger merger code sẽ dẫn đến việc build liên tục, làm chậm hệ thống.

+ Với branch prepro thì sẽ được trigger mỗi lần có sự thay đổi về code sẽ tự động thực hiện các bước như với branch develop.

+ Với branch master thì có hơi khác một chút, Git cũng sẽ tự động trigger và tiến hành các bước build, run unit test, quét sonar.... nhưng không tiến hành deploy (quá trình này chỉ là CI) lên server mà chỉ được deploy khi có confirm từ một người có quyền hoặc deploy bằng tay để đảm bảo quá trình delivery sản phẩm không xảy ra lỗi và đúng với thời gian khách hàng mong muốn.

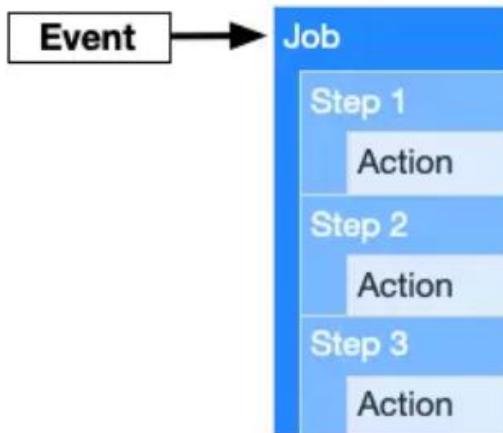
### 5.2.2 Giới thiệu về Github Actions

GitHub Actions là một nền tảng tích hợp liên tục và phân phối liên tục (CI/CD) cho phép bạn tự động hóa quy trình xây dựng, kiểm thử và triển khai. Bạn có thể tạo các quy trình công việc để xây dựng và kiểm thử mọi yêu cầu kéo đến kho lưu trữ của mình, hoặc triển khai các yêu cầu kéo đã hợp nhất lên môi trường sản xuất.

GitHub Actions không chỉ dừng lại ở DevOps mà còn cho phép bạn chạy quy trình công việc khi các sự kiện khác xảy ra trong kho lưu trữ của bạn. Ví dụ: bạn có thể chạy quy trình công việc để tự động thêm nhãn phù hợp mỗi khi có người tạo sự cố mới trong kho lưu trữ của bạn.

GitHub cung cấp máy ảo Linux, Windows và macOS để chạy quy trình làm việc của bạn hoặc bạn có thể lưu trữ trình chạy tự lưu trữ của riêng mình trong trung tâm dữ liệu hoặc cơ sở hạ tầng đám mây của riêng bạn.

Mô hình mô tả cách một git actions có thể tiến hành một công việc bất kì (như trong ví dụ trên là run các unit test case có sẵn). Một sự kiện sẽ tự động kích hoạt workflow đã được định nghĩa sẵn trong một job. Mỗi job sử dụng steps control để kiểm soát actions. Actions là commands thực hiện một hành động cụ thể nào đó (run các unit test case)



*Hình 5.2 Cách Action một command*

- Các thành phần của một Github Actions:

+ Workflows là một tập các hành động mà bạn thêm vào repository của mình để định nghĩa các hành động. Các jobs trong workflows có thể được thực thi theo lịch hoặc dựa vào một trigger nào đó. Workflows có thể được định nghĩa để build, test, release, deploy.... một dự án trên Github. Một workflows được định nghĩa bằng file yml.

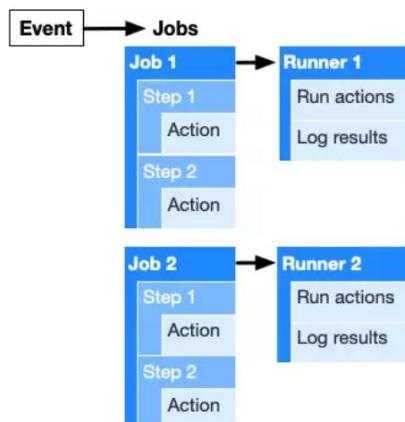
+ Events là một trigger đặc biệt để workflow bắt đầu. Ví dụ như, bạn có thể cấu hình để workflow bắt đầu khi có một người nào đó push code hoặc tạo merger request lên branch develop. Bạn cũng có thể sử dụng repository dispatch webhook để trigger một workflow khi một sự kiện bên ngoài xảy ra .

+ Jobs là tập hợp các bước thực hiện một công việc của một runner. Mặc định thì các jobs trong một workflow được chạy song song. Bạn cũng có thể cấu hình để các jobs chạy một cách tuần tự. Ví dụ trong một workflow có thể có jobs là build, run test case. Nhưng nếu build fails thì test case sẽ không được run.

+ Steps là một tác vụ độc lập nó có thể là một command trong một jobs. Mỗi steps có thể là một action hoặc một command để thực hiện một hành động nào đó. Mỗi step trong một job thực thi trong cùng một runner, có thể share data từ steps này với step khác.

+ Actions là một command độc lập khi kết hợp lại tạo thành một steps để tạo ra jobs trong workflow. Actions là đơn vị nhỏ nhất của một workflow là thành phần trực tiếp thực hiện các tác vụ mong muốn.

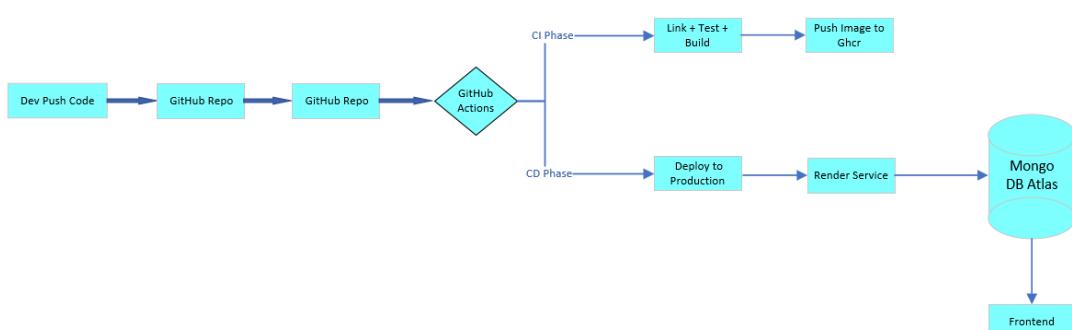
+ Runners là một server được cài đặt sẵn GitHub Actions runner application. Bạn có thể sử dụng runner hosted bởi GitHub hoặc bạn có thể tự host cho mình để sử dụng. Một runner luôn sẵn sàng lắng nghe các jobs, run một job tại một thời điểm, report process, logs và trả kết quả về cho GitHub. Với GitHub-hosted runner mỗi job được runs trên một máy ảo hoàn toàn mới (Điều này có nghĩa là mỗi job bạn đều cần có bước setup môi trường từ đầu ).



Hình 5.3 Các thành phần của Github Actions

### 5.2.3 Quy trình CI/ CD vào hệ thống với Github Actions

Tổng quan về kiến trúc:



Hình 5.4 Kiến trúc CI CD của hệ thống

Tổng quan về quy trình thực hiện:

Để tối ưu hóa quá trình phát triển, kiểm thử và triển khai ứng dụng, nhóm đã thiết lập một quy trình CI/CD (Continuous Integration / Continuous Deployment) sử dụng công cụ GitHub Actions. Mục tiêu của quy trình này là:

- + Tự động hóa toàn bộ luồng kiểm tra và triển khai.
- + Đảm bảo tính ổn định, an toàn của mã nguồn.
- + Rút ngắn thời gian đưa sản phẩm đến người dùng cuối.
- **Continuous Integration (CI) – Tích hợp liên tục**

CI là bước kiểm tra và xác thực mã nguồn ngay sau mỗi lần thay đổi (push hoặc tạo pull request) vào nhánh chính như main hoặc production.

Các bước cụ thể trong CI:

1. Lắng nghe sự kiện (Trigger): Quy trình CI được kích hoạt khi:

- + Có hành động push code mới lên GitHub.
- + Có pull request gửi đến các nhánh như production.

2. Checkout mã nguồn: Sử dụng actions/checkout@v3 để lấy mã nguồn từ GitHub về máy ảo runner.

3. Cài đặt môi trường:

- + Cài đặt Node.js cho frontend/backend.
- + Cài đặt các thư viện (npm install hoặc yarn install).

4. Kiểm tra mã nguồn (Lint): Chạy các công cụ như eslint, prettier để kiểm tra coding convention và format.

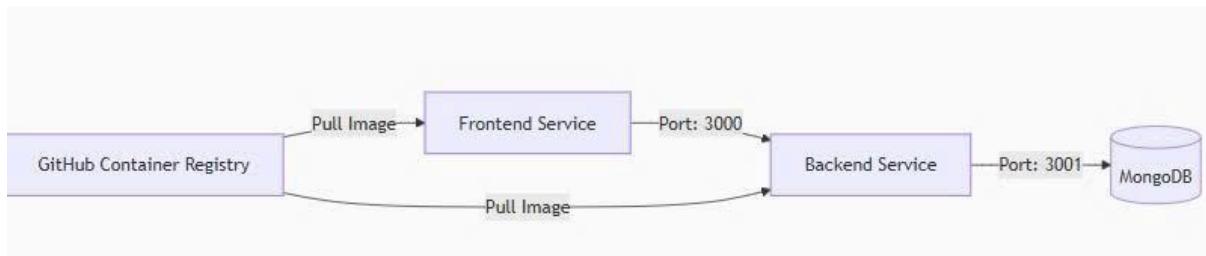
5. Build project:

- + Chạy npm run build để build frontend (React/Vite) và backend (Node.js).
- + Nếu quá trình build bị lỗi, workflow sẽ dừng tại đây để đảm bảo code chưa sẵn sàng để triển khai.

6. Tạo Docker Image:

- + Sử dụng Docker để đóng gói mã nguồn vào image cho frontend và backend.
- + Đặt tag theo phiên bản, ví dụ: expense-management-frontend: prod - <commit-hash>.

Sơ đồ luồng CI:



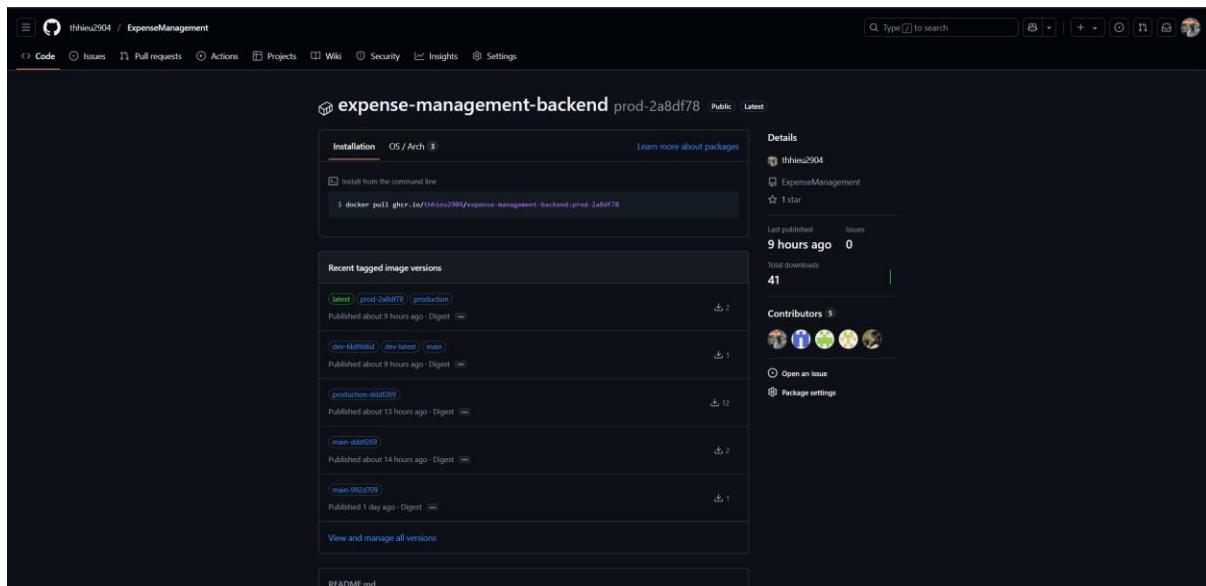
Hình 5.5 Luồng CI

### - Continuous Deployment (CD) – Triển khai liên tục

Sau khi CI hoàn tất thành công, GitHub Actions tiếp tục quy trình triển khai tự động lên môi trường production.

+ Các bước trong CD:

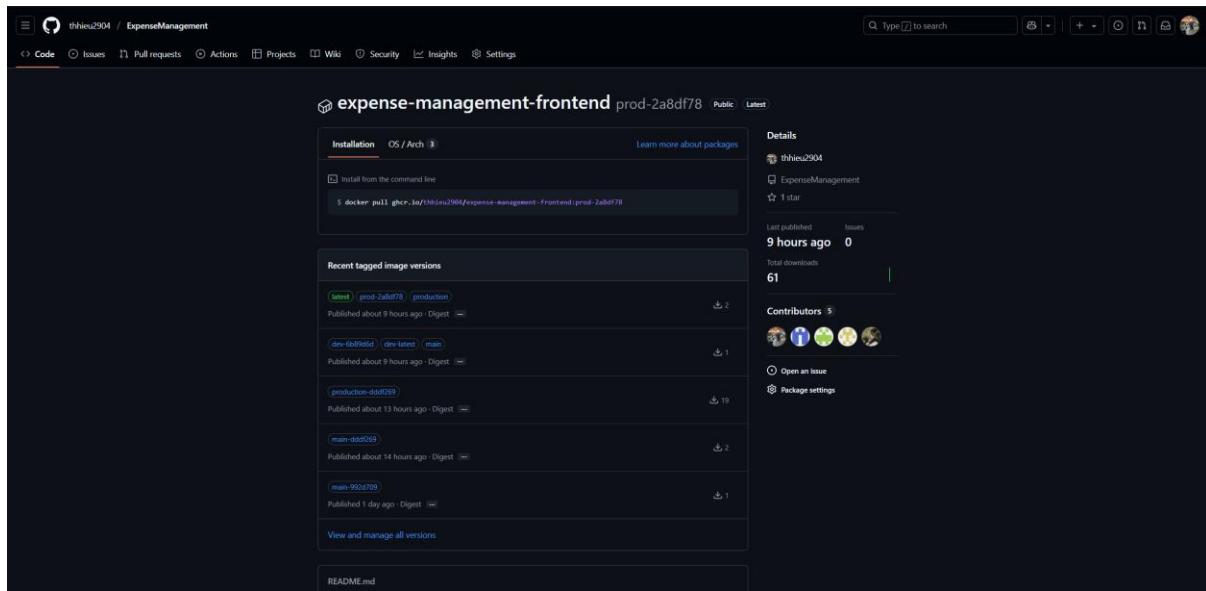
- Đăng nhập GitHub Container Registry (ghcr.io): Sử dụng token để xác thực và đẩy Docker image lên GitHub Packages.



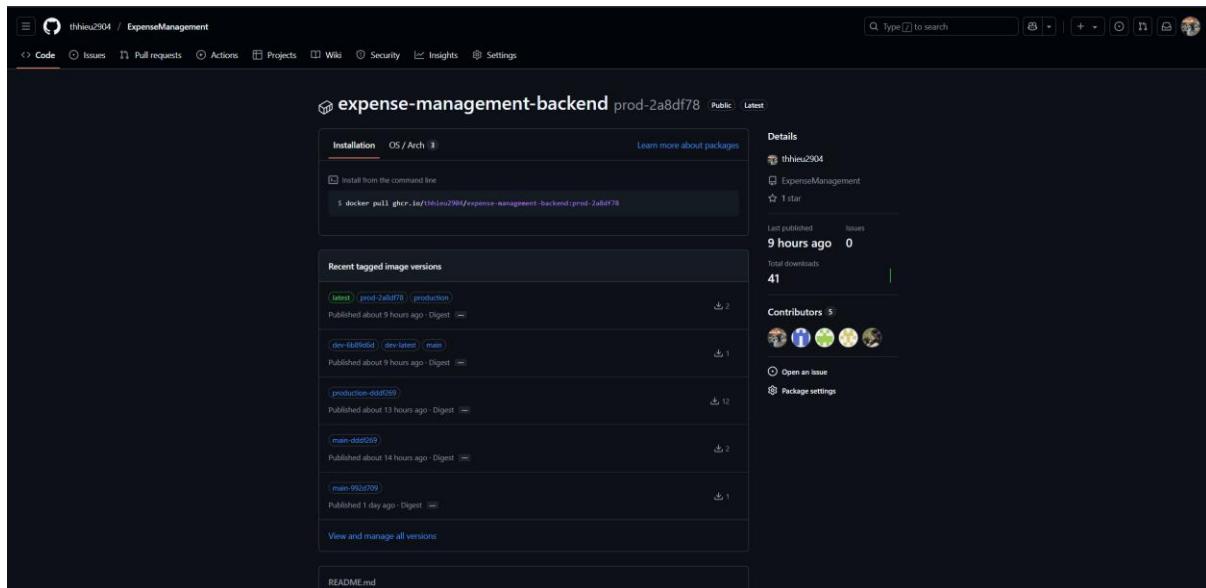
Hình 5.6 Đẩy Docker image lên GitHub Packages

2.Push Docker Image:

- Đẩy 2 image đã build (frontend & backend) lên GitHub Packages.



Hình 5.7 Đây Docker image frontend lên GitHub Packages



Hình 5.8 Đây Docker image backend lên GitHub Packages

+ Lưu trữ version để tiện rollback nếu cần.

3. Trigger deploy frontend:

+ Gửi HTTP POST đến webhook của Render.com (được lưu trong biến môi trường RENDER\_DEPLOY\_HOOK\_FRONTEND) để Render tự động lấy source mới và triển khai frontend.

+ Quá trình này đảm bảo người phát triển không cần thao tác tay.

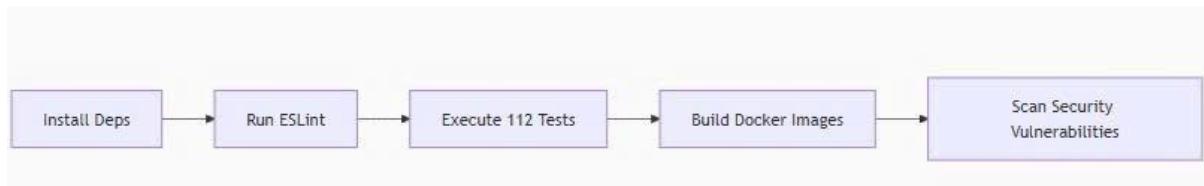
4. Triển khai backend: Có 2 cách:

+ Tự động: Tương tự frontend, backend cũng có thể được deploy thông qua Render webhook (RENDER\_DEPLOY\_HOOK\_BACKEND).

+ Thủ công: Kéo Docker image từ GitHub và deploy trên dịch vụ cloud riêng hoặc Docker host.

5. Thông báo hoặc ghi log: Sau khi deploy, GitHub Actions có thể gửi thông báo qua Slack, Email, hoặc Telegram (tùy cấu hình mở rộng).

Sơ đồ luồng CD:



Hình 5.9 Luồng CD

### 5.3 Cấu hình Docker và quy trình triển khai ứng dụng

#### 5.3.1 Cấu hình Docker

Ứng dụng "Website quản lý chi tiêu" được container hóa bằng Docker, với cấu hình được định nghĩa trong tệp docker-compose.yml. Cấu hình này bao gồm ba container chính:

- + Frontend: Chạy ứng dụng React SPA, phục vụ giao diện người dùng cho việc quản lý thu – chi, truy cập tại localhost:3000.
- + Backend: Chạy ứng dụng ExpressJS, xử lý các API như danh mục thu/chi, người dùng, thống kê, và kết nối đến cơ sở dữ liệu.
- + Database: Chạy MongoDB, lưu trữ dữ liệu hệ thống bao gồm thông tin người dùng, danh mục, giao dịch thu – chi, ghi chú và biểu đồ thống kê.

#### 5.3.2 Quy trình triển khai

+ Đóng gói ứng dụng bằng Docker: Nhóm đã container hóa toàn bộ hệ thống bằng Docker. Tệp docker-compose.yml định nghĩa các dịch vụ (frontend, backend, database) cùng các cổng tương ứng, các biến môi trường và liên kết phụ thuộc giữa backend và database. Câu lệnh docker-compose up --build được sử dụng để khởi tạo và chạy toàn bộ ứng dụng trong môi trường nhất quán.

+ Triển khai và kiểm thử ứng dụng: Sau khi build thành công, ứng dụng được chạy trên máy chủ cục bộ, với giao diện người dùng có thể truy cập tại localhost:3000. Người dùng có thể thực hiện các thao tác như thêm giao dịch thu/chi, xem báo cáo biểu

đồ, hoặc thống kê chi tiêu theo thời gian. Toàn bộ quá trình được tích hợp CI đơn giản thông qua GitHub Actions, tự động build lại Docker Image khi có thay đổi mã nguồn.

+ Định hướng triển khai trên môi trường cloud: Ở thời điểm hiện tại, ứng dụng đang được triển khai trên môi trường cục bộ. Nhóm đang lập kế hoạch mở rộng triển khai lên các nền tảng cloud như Render, Railway hoặc AWS ECS. Khi đó, việc cấu hình Docker sẽ được điều chỉnh để đảm bảo bảo mật, hiệu năng và tính linh hoạt khi triển khai thực tế.

## CHƯƠNG 6: KIỂM THỬ

### 6.1 Chiến lược kiểm thử

Để đảm bảo hệ thống hoạt động ổn định và đúng yêu cầu, nhóm đã áp dụng chiến lược kiểm thử kết hợp giữa kiểm thử thủ công và tự động như sau:

- + Unit Test: Sử dụng để kiểm tra các chức năng logic cốt lõi như xử lý đặt vé, xác thực tài khoản. Các test case được viết bằng Jest và chạy tự động trong quá trình phát triển.
- + Postman: Kiểm thử API thủ công để xác minh các endpoint phản hồi đúng dữ liệu và trạng thái, bao gồm đăng nhập, lấy danh sách phim, đặt vé,...
- + GitHub Actions: Tích hợp quy trình CI/CD, giúp tự động kiểm thử và build hệ thống mỗi khi có thay đổi mã nguồn.

Chiến lược này giúp phát hiện lỗi sớm, duy trì tính ổn định và đảm bảo chất lượng cho toàn bộ hệ thống.

### 6.2 Kết quả kiểm thử bằng Postman

Tất cả API endpoint của backend đều được test bằng Postman trước khi tạo tài liệu API với Swagger, một số ví dụ về kết quả kiểm thử:

- API đăng nhập: <http://localhost:5000/api/auth/login>
- Method: POST

The screenshot shows the Postman interface with the following details:

- URL:** http://localhost:5000/api/auth/login
- Method:** POST
- Body (JSON):**

```
1 {
2   "username": "Kha",
3   "fullname": "Hoàng Kha",
4   "password": "123",
5   "avatar": ""
6 }
```
- Response (JSON):**

```
1 {
2   "message": "Đăng nhập thành công",
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZCI6IjY4MmFmM100DY1ZDNmZTQ0ZmRlYTBmMyIsInVzZXJuYW1lIjois2hhIiwiawF0IjoxNzUzMjE0OTA1LCJleHA10jE3NTMyNTgxMDV9.
Inltet4us6AmoP86wLhDhdUSXXTtlCQTQn9dCtZI7CzU",
4   "account": {
5     "id": "602af2b4865d3fe44fdea0f3",
6     "username": "Kha",
7     "fullname": "Hoàng Kha",
8     "avatar": ""
9   }
10 }
```

Hình 6.1 Test đăng nhập bằng Postman

+ Request body:

```
{  
    "username": "Kha",  
    "fullname": "Hoàng Kha",  
    "password": "123",  
    "avatar": ""  
}
```

+ Phản hồi từ server: Status “**200 OK**” cho thấy yêu cầu đăng nhập đã thành công và trả về thông tin kèm token như mã lỗi err bằng 0, thông báo đăng nhập thành công.

Tiến hành đăng nhập để lấy token với phương thức POST với đường liên kết:

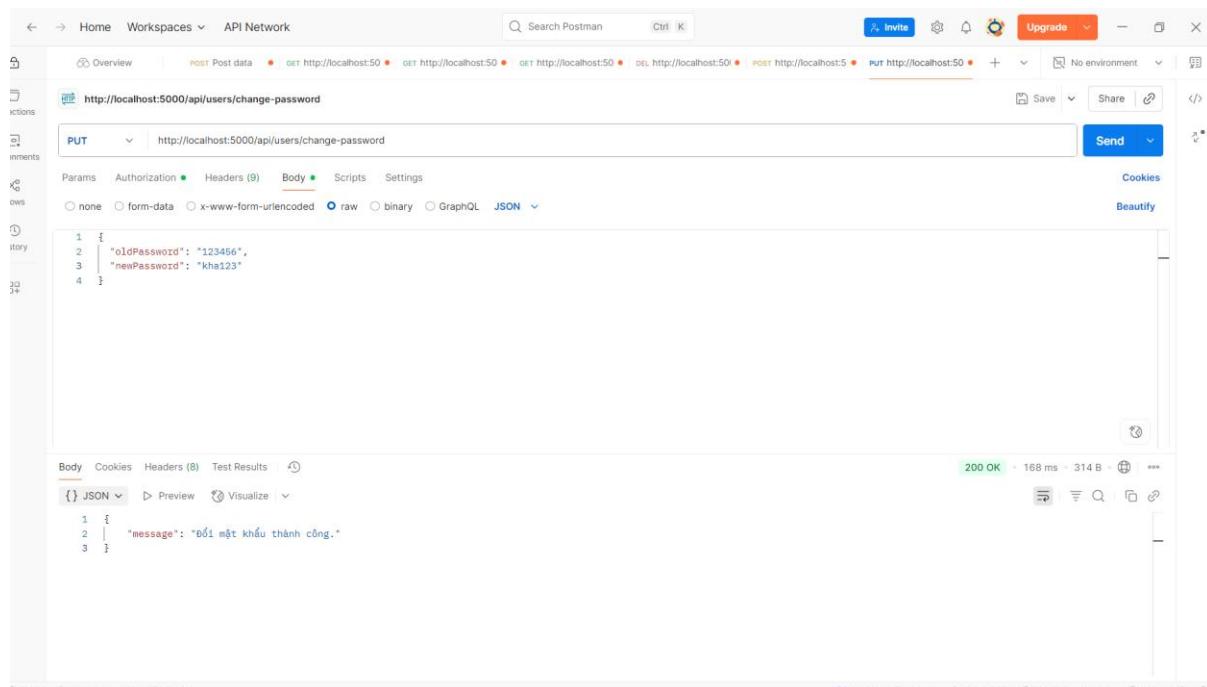
<http://localhost:5000/api/auth/login>

- API User: <http://localhost:5000/api/users/change-password>

+ Method: PUT.

+ Request body:

```
{  
    "oldPassword": "123456",  
    "newPassword": "kha123"  
}
```

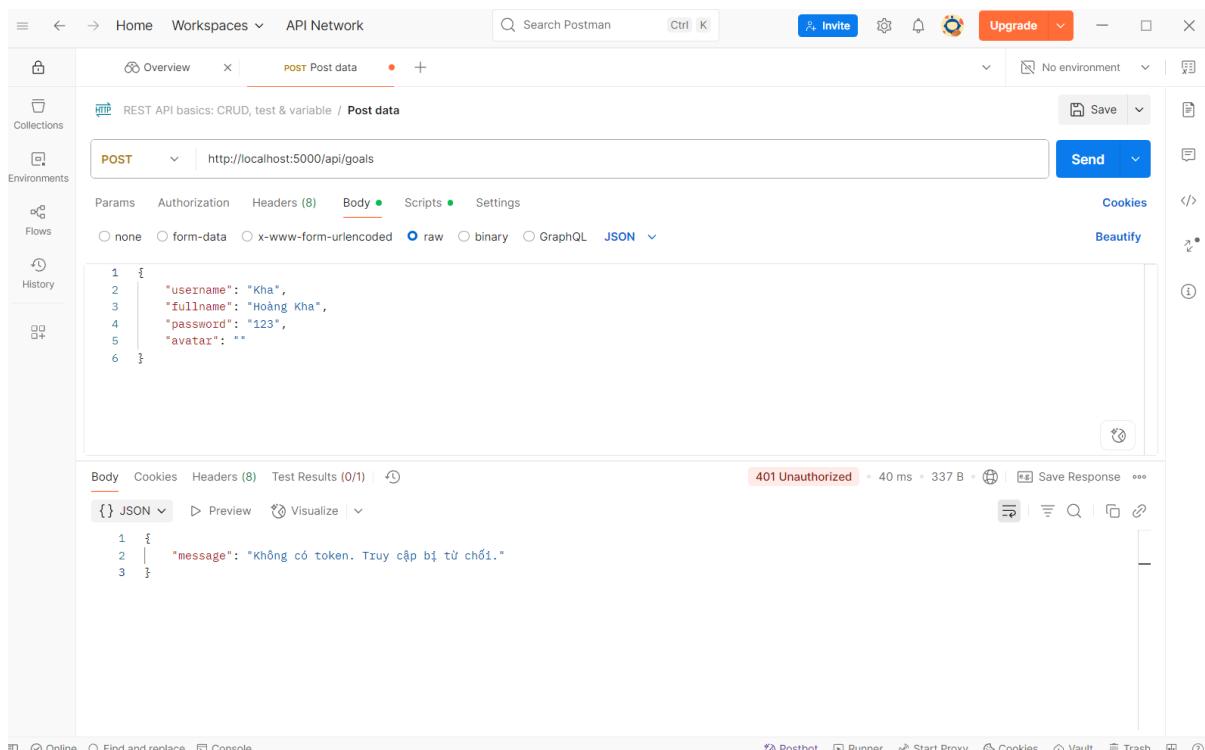


Hình 6.2 Test thay đổi mật khẩu của hệ thống bằng Postman

- API mục tiêu: <http://localhost:5000/api/goals>

- Method: GET

+ Trường hợp thiếu token hay chưa đăng nhập trên website: Postman sẽ trả về status 401 Unauthorized có ý nghĩa là server từ chối yêu cầu vì người dùng chưa xác thực hoặc xác thực không hợp lệ (do middleware xử lí).



Hình 6.3 Xem giao dịch từ hệ thống bị lỗi token

- Sử dụng phương thức GET để xem mục tiêu đã có sẵn : <http://localhost:5000/api/goals>.

The screenshot shows the Postman interface with a GET request to `http://localhost:5000/api/goals`. The 'Authorization' field is set to 'Bearer Token' with a yellow placeholder box containing a token. The response is a 200 OK status with JSON data showing a single goal record:

```

1 {
2   "data": [
3     {
4       "_id": "688086dbe2f3bf79bf56cf4b",
5       "user": "6880879ae2f3bf79bf56cdcc2",
6       "name": "Sắm nhà",
7       "targetAmount": 22222222,
8       "currentAmount": 0,
9       "deadline": "2025-07-24T00:00:00Z",
10      "status": "in-progress",
11      "icon": "\ud83d\udcbb"
12    }
13  ]
14}

```

Hình 6.4 Xem giao dịch từ hệ thống

- **API giao dịch:** <http://localhost:5000/api/transactions/6880842ae2f3bf79bf56cf1c>
- Method: DELETE
- + Hãy đảm duy trì token để có thể xem giao dịch thành công, đảm bảo giao dịch đã tồn tại.
- + Đưa id của giao dịch vào đường liên kết để thực hiện phương thức DELETE.

The screenshot shows the Postman interface with a DELETE request to `http://localhost:5000/api/transactions/6880842ae2f3bf79bf56cf1c`. The 'Body' tab shows a JSON payload with the id of the transaction to be deleted:

```

1 {
2   "_id": "68808409e2f3bf79bf56ce09",
3   "name": "Mua sắm",
4   "type": "CHITIEU",
5   "icon": "fa-pizza-slice"
6 }

```

The response is a success message indicating the transaction was deleted successfully:

```

1 {
2   "message": "Xóa giao dịch thành công"
3 }

```

Hình 6.5 Kiểm tra xóa giao dịch

- **API Goals:** <http://localhost:5000/api/goals/688086dbe2f3bf79bf56cf4b/pin>

+ Method: PATCH

+ Dùng phương thức PATCH gọi tới trạng thái và thay đổi trạng thái của id mà bạn gọi.

- + API này dùng để thay đổi trạng thái ghim (pin) của một mục tiêu (goal) cụ thể trong hệ thống. Khi một mục tiêu được ghim, nó sẽ:
  - Được ưu tiên hiển thị ở vị trí nổi bật (ví dụ: đầu danh sách hoặc giao diện riêng).
  - Giúp người dùng dễ dàng theo dõi các mục tiêu quan trọng.

The screenshot shows the Postman application interface. A PATCH request is being made to the URL `http://localhost:5000/api/goals/688086dbe2f3bf79bf56cf4b/pin`. The request body contains the following JSON:

```
1 {  
2   "oldPassword": "123456",  
3   "newPassword": "kha123"  
4 }
```

The response status is 200 OK, with a response time of 12 ms and a response size of 594 B. The response body is identical to the request body, indicating successful update.

Hình 6.5 Thay đổi trạng thái của mục tiêu

### 6.3 Kiểm thử bằng Unitest

Nội dung kiểm thử: 68

Backend:

- API quản lý người dùng:

- + Kiểm tra đăng ký tài khoản mới.
- + Kiểm tra đăng nhập và xác thực JWT token.
- + Kiểm tra thêm, sửa, xóa các giao dịch tài chính.

- Xác thực và phân quyền:

- + Đảm bảo các API yêu cầu token hợp lệ mới cho phép truy cập.
- + Trả về lỗi 401 khi token không hợp lệ hoặc không cung cấp.

```

    // Mock auth middleware
    const mockAuth = (req, res, next) => {
      if (req.headers.authorization) {
        const token = req.headers.authorization.split(" ")[1];
        try {
          const decoded = jwt.verify(token, process.env.JWT_SECRET);
          res.user = decoded;
        } catch (err) {
          return res.status(401).json({ message: "Invalid token" });
        }
      }
      next();
    };

    // Create Express app for testing
    const app = express();
    app.use(express.json());

    // Mock auth middleware
    const mockAuth = (req, res, next) => {
      if (req.headers.authorization) {
        const token = req.headers.authorization.split(" ")[1];
        try {
          const decoded = jwt.verify(token, process.env.JWT_SECRET);
          res.user = decoded;
        } catch (err) {
          return res.status(401).json({ message: "Invalid token" });
        }
      }
      next();
    };
  
```

Hình 6.6 Test Unitest cho backend

## Frontend:

### - Component hiển thị danh mục:

- + Kiểm tra sự kiện click vào danh mục chi tiêu.
- + Kiểm tra dữ liệu danh mục hiển thị đúng tên và icon.

### - Biểu đồ thống kê chi tiêu: Kiểm tra dữ liệu hiển thị trên biểu đồ được cập nhật

đúng khi có giao dịch mới.

```

    // Call onCategoryClick with: cat1 Food
    // src/pages/_tests/_welcome.test.jsx (9 tests) 1 m
    // src/components/recentTransactions/_tests/_transactionItem.test.jsx (10 tests) 11 m
    // src/components/common/_tests/_basePieChart.test.jsx (22 tests) 41 m
  
```

Hình 6.7 Test Unitest cho frontend

## Kết quả kiểm thử

Sau khi viết và chạy toàn bộ các test case, kết quả kiểm thử đạt được như sau:

Phạm vi	Số lượng test	Pass	Fail	Tỉ lệ thành công
Backend	171	171	0	100%
Frontend	46	46	0	100%
Tổng cộng	217	217	0	100%

Kết quả kiểm thử chứng minh rằng các chức năng trong hệ thống hoạt động ổn định và đáp ứng đúng yêu cầu. Việc áp dụng Unit Test đã giúp nhóm phát hiện và xử lý các lỗi tiềm ẩn sớm, góp phần nâng cao độ tin cậy của phần mềm.

## CHƯƠNG 7: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 7.1 Kết luận

Đề tài “Xây dựng website quản lý chi tiêu” đã được triển khai và hoàn thiện với mục tiêu hỗ trợ người dùng theo dõi, thống kê và quản lý các khoản thu – chi trong cuộc sống hằng ngày một cách trực quan, hiệu quả.

Trong quá trình phát triển hệ thống, nhóm đã áp dụng mô hình phát triển phần mềm Agile với phương pháp Scrum nhằm tối ưu quá trình lập kế hoạch, kiểm soát tiến độ và đảm bảo chất lượng. Công cụ Jira được sử dụng xuyên suốt để chia nhỏ công việc theo Sprint, phân công rõ ràng và theo dõi mức độ hoàn thành các User Story. Nhờ đó, tiến độ dự án được đảm bảo theo đúng kế hoạch đề ra.

Về giao diện, nhóm đã sử dụng Figma trong khâu thiết kế UI/UX, từ khung wireframe đến prototype chi tiết, giúp định hướng trực quan ngay từ giai đoạn đầu và đảm bảo trải nghiệm người dùng thân thiện, dễ sử dụng.

Về mặt kỹ thuật, hệ thống được xây dựng với kiến trúc tách biệt Frontend – Backend, sử dụng ReactJS cho giao diện người dùng và NodeJS (Express) cho xử lý nghiệp vụ phía máy chủ. Cơ sở dữ liệu MongoDB được chọn nhằm đáp ứng yêu cầu linh hoạt và dễ mở rộng dữ liệu. Toàn bộ ứng dụng được đóng gói bằng Docker, giúp đảm bảo tính nhất quán trong môi trường phát triển và triển khai. Nhóm cũng đã thiết lập quy trình CI/CD thông qua GitHub Actions, giúp tự động hóa việc kiểm tra, build và triển khai ứng dụng lên môi trường thực thi.

### 7.2 Hạn chế

Dù hệ thống đã đáp ứng được các chức năng cơ bản theo yêu cầu ban đầu, nhưng vẫn còn tồn tại một số hạn chế như sau:

- + Hệ thống chưa hỗ trợ phân quyền người dùng (ví dụ: Admin và người dùng thường).
- + Giao diện chưa tối ưu hoàn toàn trên thiết bị di động, còn phụ thuộc vào độ phân giải lớn.
- + Một trong những hạn chế của hệ thống hiện tại là chưa áp dụng kiến trúc Microservices, mà vẫn hoạt động theo mô hình đơn khối (Monolithic). Điều này khiến việc bảo trì, mở rộng hay triển khai độc lập từng chức năng gặp nhiều khó khăn. Khi số lượng người dùng và dữ liệu tăng lên, toàn bộ hệ thống phải mở rộng cùng lúc thay vì

chỉ mở rộng từng phần cần thiết. Ngoài ra, việc cập nhật hoặc sửa lỗi một chức năng nhỏ cũng có thể ảnh hưởng đến toàn bộ ứng dụng, gây khó khăn trong quá trình phát triển lâu dài.

### 7.3 Hướng phát triển

Trong tương lai, để hệ thống trở nên hoàn thiện và thực tiễn hơn, nhóm đề xuất các hướng phát triển sau:

- + Phát triển thêm phiên bản ứng dụng di động sử dụng React Native hoặc Flutter để nâng cao khả năng tiếp cận và trải nghiệm người dùng.
- + Tích hợp xác thực người dùng bằng OAuth 2.0 (Google, Facebook, v.v.) để tăng tính bảo mật và thuận tiện đăng nhập.
- + Xây dựng hệ thống phân quyền, cho phép quản lý nhiều vai trò như Quản trị viên, Người dùng cơ bản.
- + Mở rộng khả năng thống kê và phân tích, với nhiều tiêu chí lọc hơn và cho phép xuất báo cáo ra định dạng PDF hoặc Excel.
- + Tích hợp tính năng thông báo và nhắc nhở định kỳ, hỗ trợ người dùng kiểm soát tài chính theo lịch trình.
- + Trong giai đoạn tiếp theo, hệ thống có thể được phát triển theo hướng kiến trúc Microservices nhằm khắc phục những hạn chế hiện tại.

# TÀI LIỆU THAM KHẢO

## 1. Sách và Bài Báo Khoa Học

- [1] Nguyễn Ngọc Xuân, 2023, Đồ án “Hệ thống quản lý chi tiêu”, Học viện công nghệ bưu chính viễn thông”, Thành phố Hồ Chí Minh.
- [2] Trần Ngọc Thanh, 2022, "Ứng dụng Machine Learning trong dự đoán chi tiêu cá nhân", Nhà xuất bản Khoa học Kỹ thuật, Hà Nội.

## 2. Tài Liệu Công Nghệ Chính Thức

- [3] MongoDB Inc., 2023, "MongoDB Aggregation Framework Documentation", [Online]. Available: <https://www.mongodb.com/docs/manual/core/aggregation-pipeline/>
- [4] React Community, 2023, "React Hooks Advanced Guide", [Online]. Available: <https://reactjs.org/docs/hooks-reference.html>
- [5] Node.js Foundation, 2023, "Node.js Security Best Practices", [Online]. Available: <https://nodejs.org/en/docs/guides/security/>

## 3. Bài Viết Kỹ Thuật và Blog

- [6] Nguyen Van Truong, "Sequelize - JavaScript ORM cho NodeJS", Viblo, 28/11/2015. [Trực tuyến]. Có tại: <https://viblo.asia/p/sequelize-javascript-orm-cho-nodejs-l0rvmmJDvyqA>. [Truy cập: 20/04/2025].
- [7] "createStore - Redux API." [Trực tuyến]. Có tại: <https://redux.js.org/api/createstore>. [Truy cập: 20/04/2025].

## 4. Tài Liệu Triển Khai Thực Tế

- [8] Render Documentation, 2023, "Deploying Docker Containers on Render", [Online]. Available: <https://render.com/docs/deploy-docker>
- [9] Vercel, 2023, "Optimizing React Performance", [Online]. Available: <https://vercel.com/guides/react-performance-optimization>

## 5. Công Cụ và Phần Mềm

- [10] Postman, 2023, "API Testing Automation Guide", [Online]. Available: <https://learning.postman.com/docs/>
- [11] Docker Inc., 2023, "Docker Multi-stage Builds", [Online]. Available: <https://docs.docker.com/build/building/multi-stage/>

## 6. Nghiên Cứu Liên Quan

- [12] Google AI, 2023, "Receipt Parsing with Computer Vision", [Online]. Available: <https://ai.google/research/pubs/pub48051>

## **PHỤ LỤC**

Hướng dẫn sử dụng cài đặt và chạy ứng dụng, cùng với code của hệ thống điều được đây lên Github với liên kết: <https://github.com/thhieu2904/ExpenseManagement>