ĐẠI HỌC PHENIKAA

TRƯỜNG CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP KẾT THÚC HỌC PHẦN

MÔN HỌC: LẬP TRÌNH CHO THIẾT BỊ DI ĐỘNG

TÊN ĐỀ TÀI: SmartWallet- Ứng dụng ví điện tử thông minh

Lớp: Lập trình cho thiết bị di động-1-1-25(N04)

Nhóm: 2025_LTTBDD_N04_Nhom_Duong

Giảng viên hướng dẫn: ThS. Nguyễn Xuân Quế

Họ và tên	Mã sinh viên	Liên hệ
Nguyễn Minh Dương	23010441	23010441@st.phenikaa-uni.edu.vn 0328451322

Hà Nội, 11-2025

BẢNG PHÂN CHIA CÔNG VIỆC

Họ và tên	Mã sinh viên	Công việc được giao	Khối lượng công việc
Nguyễn Minh Dương	23010441	Tất cả	Tất cả

Mục Lục

Mở Đầu

Trong thời đại công nghệ số phát triển mạnh mẽ như hiện nay, các ứng dụng di động đang dần trở thành một phần không thể thiếu trong đời sống con người. Từ việc liên lạc, học tập, làm việc cho đến giải trí và quản lý tài chính, mọi hoạt động đều được số hóa và tối ưu hóa thông qua các nền tảng thông minh. Trong đó, **quản lý tài chính cá nhân** là một nhu cầu thiết yếu nhưng vẫn còn nhiều người gặp khó khăn trong việc kiểm soát thu chi, lên kế hoạch tiết kiệm hay theo dõi dòng tiền hàng ngày. Việc ghi chép thủ công hoặc sử dụng các công cụ rời rạc thường gây mất thời gian, thiếu chính xác và không mang lại hiệu quả thực sự.

Trước thực tế đó, việc phát triển một ứng dụng di động giúp người dùng **quản lý chi tiêu cá nhân một cách thông minh, trực quan và an toàn** là vô cùng cần thiết. Ứng dụng **SmartWallet – Ví điện tử thông minh** được đề xuất nhằm đáp ứng nhu cầu này. Không chỉ dừng lại ở việc ghi nhận các giao dịch, SmartWallet còn hướng đến việc mang lại trải nghiệm toàn diện cho người dùng thông qua giao diện thân thiện, chức năng thống kê chi tiết, và khả năng phân tích thói quen chi tiêu.

Bên cạnh đó, đề tài cũng góp phần khẳng định tầm quan trọng của việc ứng dụng công nghệ trong lĩnh vực quản lý tài chính cá nhân – một lĩnh vực đang phát triển nhanh chóng và có tiềm năng lớn tại Việt Nam. Thông qua việc triển khai SmartWallet, nhóm mong muốn không chỉ mang đến một sản phẩm công nghệ hữu ích, mà còn giúp người dùng hình thành **thói quen tài chính khoa học**, hướng tới cuộc sống hiện đại, chủ động và hiệu quả hơn về mặt kinh tế.

Chương 1: Giới thiệu đề tài

1.1 Giới thiệu đề tài

Tên ứng dụng: *SmartWallet – Ứng dụng Ví điện tử thông minh*
Trong thời đại công nghệ số phát triển mạnh mẽ, nhu cầu quản lý tài chính cá nhân một cách thông minh, tiện lợi và an toàn ngày càng trở nên cấp thiết. Ứng dụng **SmartWallet** được xây dựng nhằm đáp ứng nhu cầu đó – một giải pháp ví điện tử giúp người dùng quản lý, theo dõi và thống kê chi tiêu hàng ngày chỉ với vài thao tác đơn giản trên điện thoại. Ứng dụng không chỉ hỗ trợ lưu trữ và phân loại các giao dịch, mà còn cung cấp cái nhìn tổng quan về tình hình tài chính cá nhân, giúp người dùng chi tiêu hợp lý và hiệu quả hơn.

Lý do chọn đề tài xuất phát từ thực tế rằng nhiều người hiện nay vẫn gặp khó khăn trong việc kiểm soát chi tiêu, dẫn đến mất cân đối tài chính. Dù có nhiều ứng dụng tài chính trên thị trường, nhưng không phải sản phẩm nào cũng mang lại trải nghiệm thân thiện, dễ sử dụng và phù hợp với người Việt. Vì vậy, nhóm tác giả mong muốn xây dựng một ứng dụng có thiết kế hiện đại, dễ tiếp cận, và đặc biệt phù hợp với thói quen sử dụng công nghệ của người Việt Nam.

Ý nghĩa thực tiễn của ứng dụng thể hiện ở việc hỗ trợ người dùng kiểm soát dòng tiền một cách khoa học, từ đó hình thành thói quen quản lý tài chính cá nhân tốt hơn. Ngoài ra, SmartWallet còn là bước đệm hướng tới các giải pháp tài chính thông minh hơn trong tương lai, như kết nối ngân hàng, phân tích xu hướng chi tiêu, hay gợi ý kế hoạch tiết kiệm tối ưu.

1.2 Mục tiêu của đề tài

Mục tiêu chính của đề tài là phát triển một ứng dụng ví điện tử hoàn chỉnh mang tên **SmartWallet**, đáp ứng nhu cầu quản lý tài chính cá nhân của người dùng.
Bên cạnh đó, nhóm đề ra các **mục tiêu cụ thể** như sau:

- Xây dựng hệ thống cho phép người dùng quản lý nhiều ví điện tử khác nhau (ví tiền mặt, ví tiết kiệm, ví ngân hàng...).
- Cho phép theo dõi chi tiết các giao dịch thu – chi, đồng thời phân loại chi tiêu theo danh mục.
- Cung cấp tính năng thống kê và báo cáo tài chính trực quan, giúp người dùng dễ dàng nắm bắt thói quen chi tiêu.
- Thiết kế giao diện thân thiện, hiện đại, tối ưu trải nghiệm người dùng (UI/UX) để việc sử dụng trở nên dễ dàng, trực quan.

1.3 Đối tượng sử dụng

Đối tượng chính của ứng dụng **SmartWallet** là **người dùng cá nhân** có nhu cầu quản lý tài chính một cách hiệu quả. Cụ thể, nhóm người dùng hướng tới có độ tuổi từ **18 đến 50 tuổi**, thuộc nhóm có **thu nhập ổn định** và **sử dụng smartphone thường xuyên**. Đây là nhóm người có hiểu biết cơ bản về công nghệ, có thói quen giao dịch điện tử, và mong muốn tìm kiếm công cụ giúp họ kiểm soát chi tiêu cá nhân một cách thông minh, tiện lợi và bảo mật.

1.4 Phạm vi nghiên cứu

Đề tài tập trung **phát triển ứng dụng mobile sử dụng công nghệ Flutter**, cho phép chạy trên cả hai nền tảng **Android và iOS**. Trong phạm vi nghiên cứu này, nhóm tập trung chủ yếu vào **thiết kế giao diện (UI)** và **trải nghiệm người dùng (UX)** để đảm bảo ứng dụng hoạt động mượt mà, trực quan và dễ thao tác.
Dữ liệu được sử dụng trong giai đoạn đầu là **dữ liệu mẫu (mock data)** nhằm phục vụ mục đích trình diễn (demo) và thử nghiệm tính năng. Trong tương lai, ứng dụng có thể được mở rộng để tích hợp cơ sở dữ liệu thực, cho phép đồng bộ hóa giao dịch, bảo mật thông tin người dùng, và hỗ trợ thanh toán trực tuyến.

Chương 2: Phân tích Yêu cầu hệ thống

1 Phân tích yêu cầu chức năng

1.1 Xác thực và bảo mật

Ứng dụng **SmartWallet** cung cấp hệ thống xác thực người dùng an toàn với các chức năng **đăng ký, đăng nhập và quên mật khẩu**. Thông tin tài khoản được mã hóa trước khi lưu trữ nhằm đảm bảo tính bảo mật. Người dùng có thể khôi phục mật khẩu thông qua email xác thực để tránh truy cập trái phép.

Ngoài ra, ứng dụng chú trọng đến việc bảo vệ dữ liệu cá nhân và giao dịch bằng các cơ chế mã hóa, đồng thời có thể mở rộng tích hợp **xác thực vân tay, khuôn mặt hoặc mã OTP** trong tương lai để nâng cao tính an toàn và trải nghiệm người dùng.

1.2 Quản lý ví điện tử

Trong ứng dụng **SmartWallet**, người dùng có thể **tạo và quản lý nhiều loại ví khác nhau** tùy theo nhu cầu sử dụng, bao gồm ví tiền mặt, ví ngân hàng và ví thẻ tín dụng. Mỗi ví được thiết kế để lưu trữ thông tin riêng biệt, giúp người dùng dễ dàng phân loại và theo dõi các nguồn tài chính của mình.

Ứng dụng cho phép **xem danh sách tất cả các ví hiện có**, đồng thời cung cấp chức năng **chỉnh sửa thông tin ví** như tên ví, loại ví hoặc số dư ban đầu. Khi không còn sử dụng, người dùng có thể **xóa ví**; hệ thống sẽ hiển thị **hộp thoại xác nhận** nhằm tránh việc thao tác nhầm gây mất dữ liệu.

Bên cạnh đó, SmartWallet hỗ trợ người dùng **đặt một ví mặc định** để thực hiện các giao dịch nhanh chóng và tiện lợi hơn. Nhờ cơ chế quản lý linh hoạt này, người dùng có thể theo dõi toàn bộ nguồn tiền của mình một cách trực quan, từ đó dễ dàng kiểm soát chi tiêu và lập kế hoạch tài chính cá nhân hiệu quả hơn.

1.3 Quản lý giao dịch

Chức năng **quản lý giao dịch** là một trong những phần quan trọng nhất của ứng dụng **SmartWallet**, giúp người dùng theo dõi và kiểm soát hoạt động tài chính cá nhân một cách hiệu quả. Ứng dụng cho phép **chuyển tiền giữa các ví** một cách linh hoạt, giúp người dùng dễ dàng điều chỉnh dòng tiền giữa các nguồn khác nhau như tiền mặt, tài khoản ngân hàng hoặc thẻ tín dụng.

Bên cạnh đó, SmartWallet cung cấp **lịch sử giao dịch chi tiết**, hiển thị đầy đủ thông tin về ngày, số tiền, loại giao dịch và ví liên quan. Người dùng có thể **tìm kiếm và lọc giao dịch** theo nhiều tiêu chí khác nhau như khoảng thời gian, loại giao dịch hoặc số tiền, giúp việc tra cứu thông tin trở nên nhanh chóng và thuận tiện hơn.

Ngoài ra, mỗi giao dịch đều có **màn hình chi tiết riêng**, cho phép người dùng xem rõ các thông tin cụ thể như số tiền, danh mục chi tiêu, ghi chú, hoặc người nhận/chuyển tiền. Nhờ đó, quá trình quản lý tài chính trở nên minh bạch, dễ theo dõi và hỗ trợ người dùng đưa ra các quyết định chi tiêu hợp lý hơn trong tương lai.

1.4 Quản lý chi tiêu

Chức năng **quản lý chi tiêu** giúp người dùng dễ dàng theo dõi và kiểm soát các khoản thu nhập, chi tiêu hàng ngày của mình. Ứng dụng **SmartWallet** cho phép người dùng **thêm mới các khoản chi tiêu hoặc thu nhập** với thông tin chi tiết như số tiền, ngày giao dịch, ví liên quan và loại giao dịch.

Các khoản chi tiêu được **phân loại theo danh mục** như ăn uống, mua sắm, di chuyển, hóa đơn, lương hoặc đầu tư, giúp người dùng có cái nhìn tổng quan và khoa học hơn về cơ cấu chi tiêu của bản thân. Ngoài ra, ứng dụng cũng hỗ trợ **chỉnh sửa hoặc xóa các khoản chi tiêu** khi cần thiết, đảm bảo dữ liệu luôn chính xác và cập nhật.

Mỗi giao dịch đều có thể được **ghi chú hoặc mô tả ngắn gọn**, giúp người dùng lưu lại mục đích hoặc hoàn cảnh chi tiêu cụ thể. Nhờ vậy, SmartWallet không chỉ là công cụ ghi chép tài chính, mà còn đóng vai trò như một "trợ lý tài chính cá nhân", hỗ trợ người dùng hiểu rõ hơn thói quen chi tiêu của mình và xây dựng kế hoạch tài chính hiệu quả hơn.

1.5 Thống kê báo cáo

Chức năng **thống kê và báo cáo** giúp người dùng có cái nhìn toàn diện về tình hình tài chính cá nhân trong từng giai đoạn. Ứng dụng **SmartWallet** cung cấp **biểu đồ chi tiêu theo danh mục**, giúp người dùng dễ dàng nhận biết nhóm chi tiêu nào đang chiếm tỷ trọng lớn, từ đó điều chỉnh kế hoạch tài chính hợp lý hơn.

Ngoài ra, ứng dụng còn hiển thị **xu hướng chi tiêu theo thời gian**, thể hiện sự thay đổi trong hành vi chi tiêu của người dùng qua từng ngày, tuần hoặc tháng. Tính năng **so sánh thu chi theo tháng hoặc quý** giúp người dùng đánh giá hiệu quả quản lý tài chính trong dài hạn, đồng thời xác định các giai đoạn có biến động bất thường.

Cuối cùng, **báo cáo tổng hợp** được trình bày trực quan, dễ hiểu với các chỉ số quan trọng như tổng thu nhập, tổng chi tiêu, số dư hiện tại và tỷ lệ tiết kiệm, giúp người dùng nắm bắt toàn bộ bức tranh tài chính chỉ trong vài thao tác.

1.6 Lịch tài chính

Tính năng **lịch tài chính** trong SmartWallet được thiết kế nhằm hỗ trợ người dùng quản lý chi tiêu một cách khoa học và có kế hoạch hơn. Ứng dụng cho phép **xem các khoản thu chi theo ngày hoặc theo tháng**, giúp người dùng dễ dàng theo dõi các giao dịch trong từng khoảng thời gian cụ thể.

Người dùng có thể **đánh dấu các ngày quan trọng** như ngày nhận lương, ngày thanh toán hóa đơn hoặc ngày đáo hạn thẻ tín dụng để không bỏ lỡ các mốc tài chính quan trọng. Bên cạnh đó, SmartWallet tích hợp **chức năng nhắc nhở thanh toán**, tự động gửi thông báo khi đến hạn, giúp người dùng chủ động hơn trong việc chi trả và tránh các khoản phạt hoặc nợ quá hạn.

Nhờ sự kết hợp giữa lịch và thống kê, người dùng có thể quản lý tài chính một cách trực quan, dễ dàng nắm bắt dòng tiền và duy trì thói quen chi tiêu hợp lý mỗi ngày.

2. Phân tích yêu cầu phi chức năng

Bên cạnh các yêu cầu về chức năng, ứng dụng **SmartWallet** cần đáp ứng các yêu cầu phi chức năng nhằm đảm bảo hiệu năng, tính ổn định, bảo mật và khả năng tương thích trên nhiều nền tảng. Những yếu tố này đóng vai trò quan trọng trong việc mang lại trải nghiệm tốt cho người dùng và đảm bảo ứng dụng hoạt động ổn định trong thực tế.

2.1 Hiệu năng

Ứng dụng **SmartWallet** được thiết kế với mục tiêu mang lại **hiệu năng cao và phản hồi nhanh**. Cụ thể, **thời gian khởi động ứng dụng** không vượt quá **5 giây**, đảm bảo người dùng có thể truy cập và sử dụng ngay khi cần. Các thao tác chính như chuyển trang, xem

ví hoặc ghi giao dịch cần có **thời gian phản hồi dưới 3 giây** để tạo cảm giác mượt mà và liền mạch.

Bên cạnh đó, ứng dụng phải **duy trì sự ổn định khi cuộn danh sách dài**, chẳng hạn như danh sách giao dịch hoặc ví, đảm bảo không xảy ra giật, lag hay tải chậm trong quá trình sử dụng. Hiệu năng cao không chỉ giúp nâng cao trải nghiệm người dùng mà còn thể hiện chất lượng và tính chuyên nghiệp của sản phẩm.

2.2 Giao diện người dùng

Về mặt giao diện, **SmartWallet** tuân thủ các tiêu chuẩn của **Material Design** do Google đề xuất, đảm bảo tính nhất quán, trực quan và dễ sử dụng. Giao diện được thiết kế **responsive**, tương thích với nhiều kích thước màn hình khác nhau, từ điện thoại nhỏ đến máy tính bảng.

Ứng dụng cũng hỗ trợ **chế độ sáng và tối (light/dark mode)**, giúp người dùng có trải nghiệm thị giác dễ chịu hơn trong các điều kiện ánh sáng khác nhau. Ngoài ra, **các hiệu ứng chuyển cảnh và animation** được tối ưu hóa để mượt mà, tạo cảm giác hiện đại và sinh động nhưng vẫn đảm bảo hiệu suất cao.

2.3 Bảo mật

Vì đây là **phiên bản demo** sử dụng **mock data** (dữ liệu giả lập), nên hệ thống chưa áp dụng các cơ chế bảo mật thực tế như mã hóa hay xác thực người dùng thật. Tuy nhiên, ứng dụng vẫn được **thiết kế mô phỏng quy trình bảo mật chuẩn**, bao gồm đăng nhập, đặt lại mật khẩu và giới hạn truy cập dữ liệu người dùng.

Trong bản triển khai chính thức, các cơ chế như **mã hóa dữ liệu nhạy cảm**, **xác thực hai lớp (2FA)**, và **bảo vệ chống tấn công SQL Injection, XSS** sẽ được tích hợp để đảm bảo an toàn thông tin tài chính và quyền riêng tư của người dùng.

2.4 Tương thích

SmartWallet được phát triển bằng **Flutter**, cho phép chạy ổn định trên cả hai nền tảng **Android và iOS**. Ứng dụng hỗ trợ từ **Android 6.0 trở lên (API level 23+)** và **iOS 11.0+**, đảm bảo tương thích với hầu hết các thiết bị di động hiện nay.

Ngoài ra, ứng dụng hỗ trợ **đa ngôn ngữ** (Tiếng Việt và Tiếng Anh), giúp mở rộng phạm vi người dùng và tăng tính thân thiện cho nhiều đối tượng khác nhau. Tính tương thích cao giúp SmartWallet dễ dàng triển khai, bảo trì và mở rộng trong các phiên bản tiếp theo.

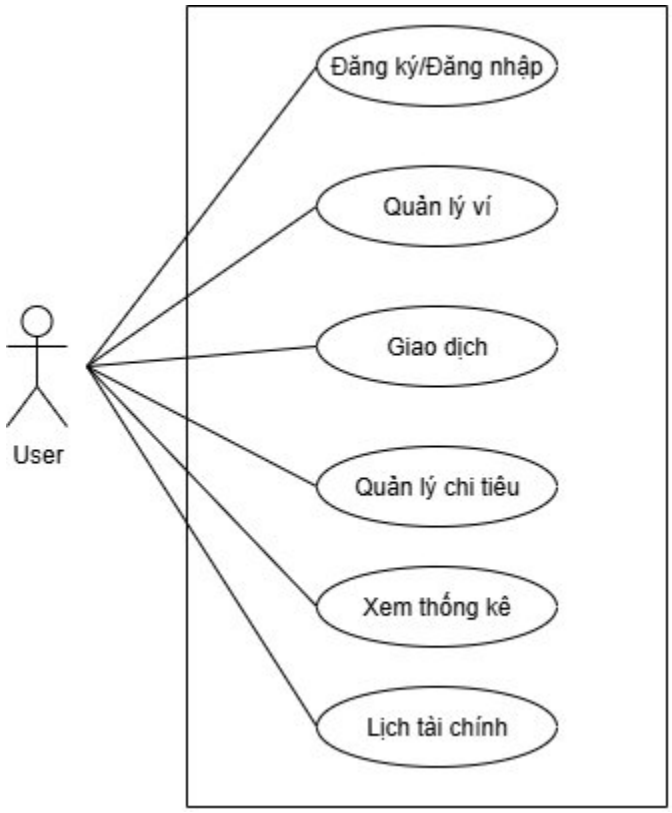
Chương 3: Thiết kế hệ thống

1 Tổng quan hệ thống

Trong hệ thống **SmartWallet**, có hai tác nhân chính tham gia tương tác:

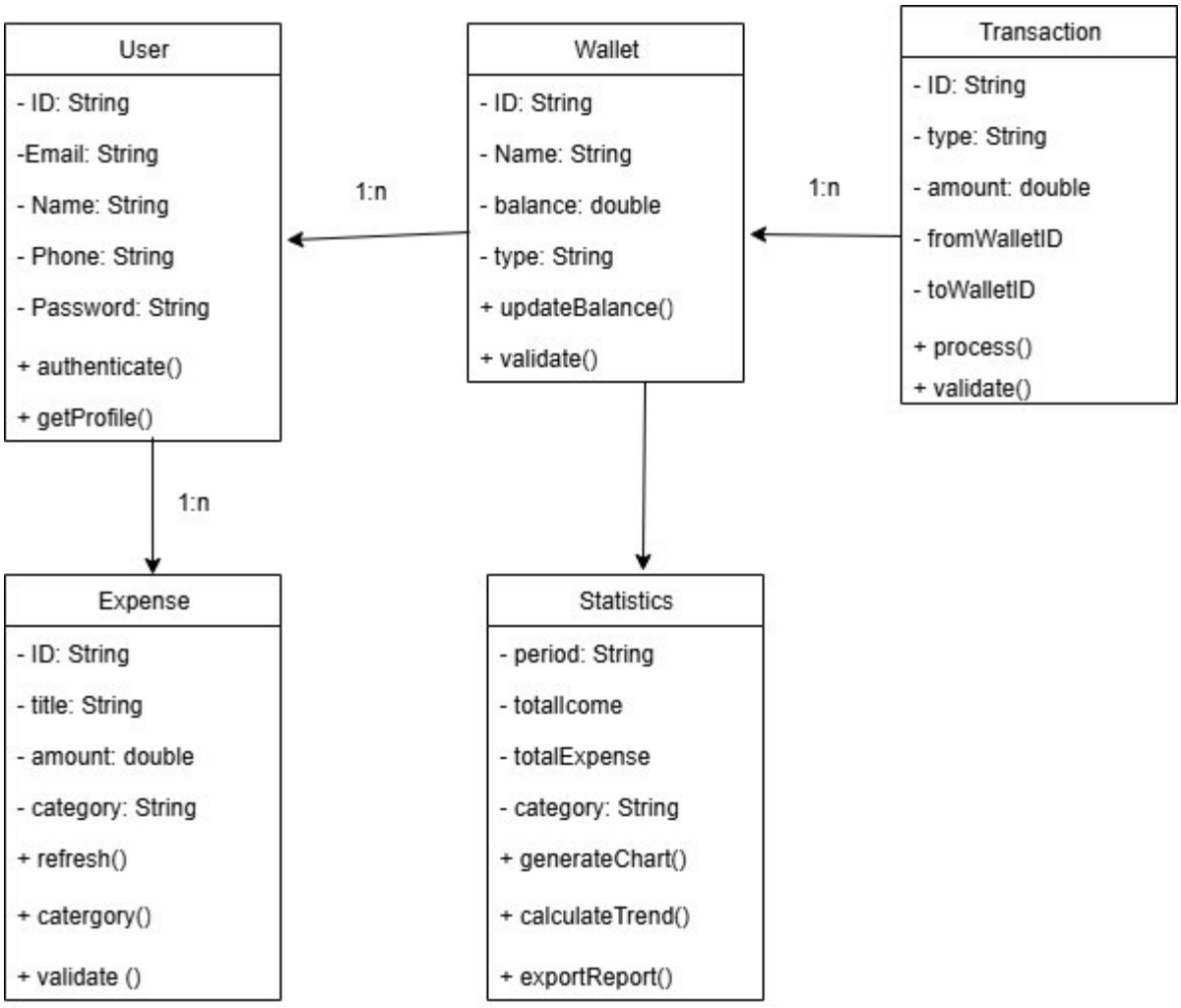
- **Người dùng (User):**
  Là chủ sở hữu ví điện tử, có thể đăng ký, đăng nhập, tạo ví, thực hiện giao dịch, quản lý chi tiêu và xem báo cáo thống kê tài chính cá nhân.
  Đây là đối tượng trực tiếp thao tác với giao diện ứng dụng.
- **Hệ thống (System):**
  Là ứng dụng **SmartWallet**, chịu trách nhiệm xử lý các yêu cầu từ người dùng như xác thực tài khoản, lưu trữ và mã hóa dữ liệu, hiển thị thông tin ví, thống kê chi tiêu và gửi thông báo nhắc nhở tài chính.

- **Sơ đồ tác nhân tổng thể**



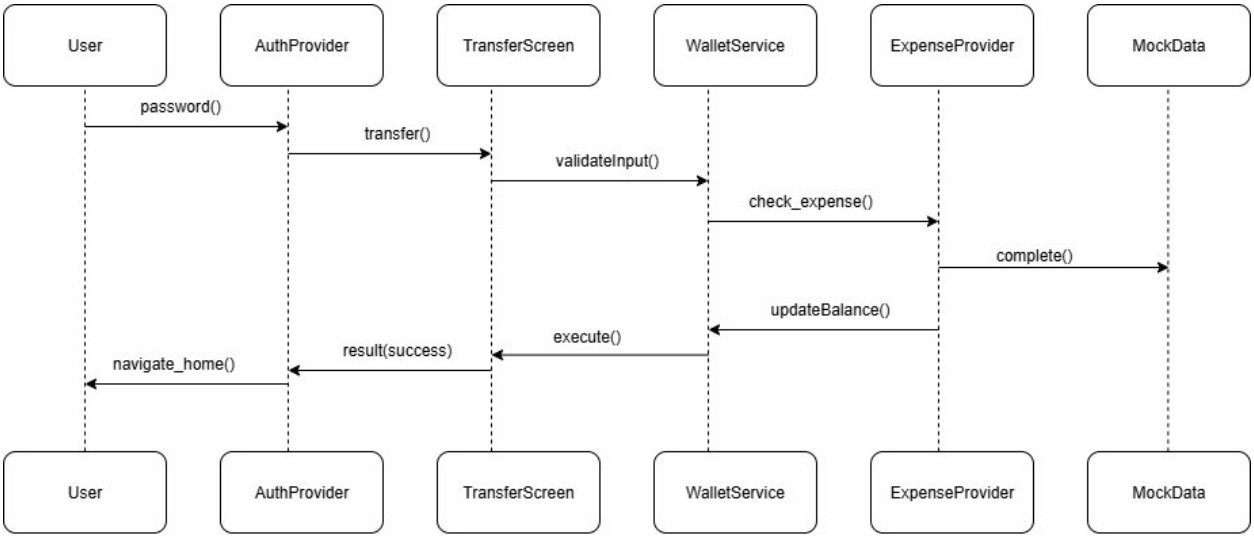
Hình 1: Sơ đồ Use Case tổng thể

- **Sơ đồ lớp tổng thể**



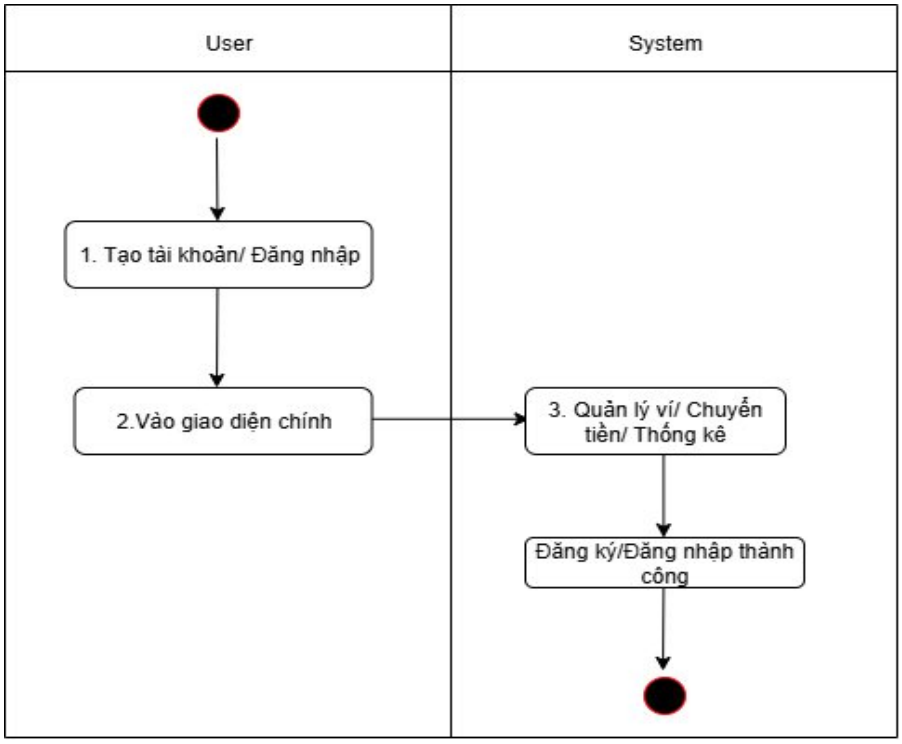
Hình 2: Sơ đồ lớp tổng thể

- **Sơ đồ trình tự tổng thể**



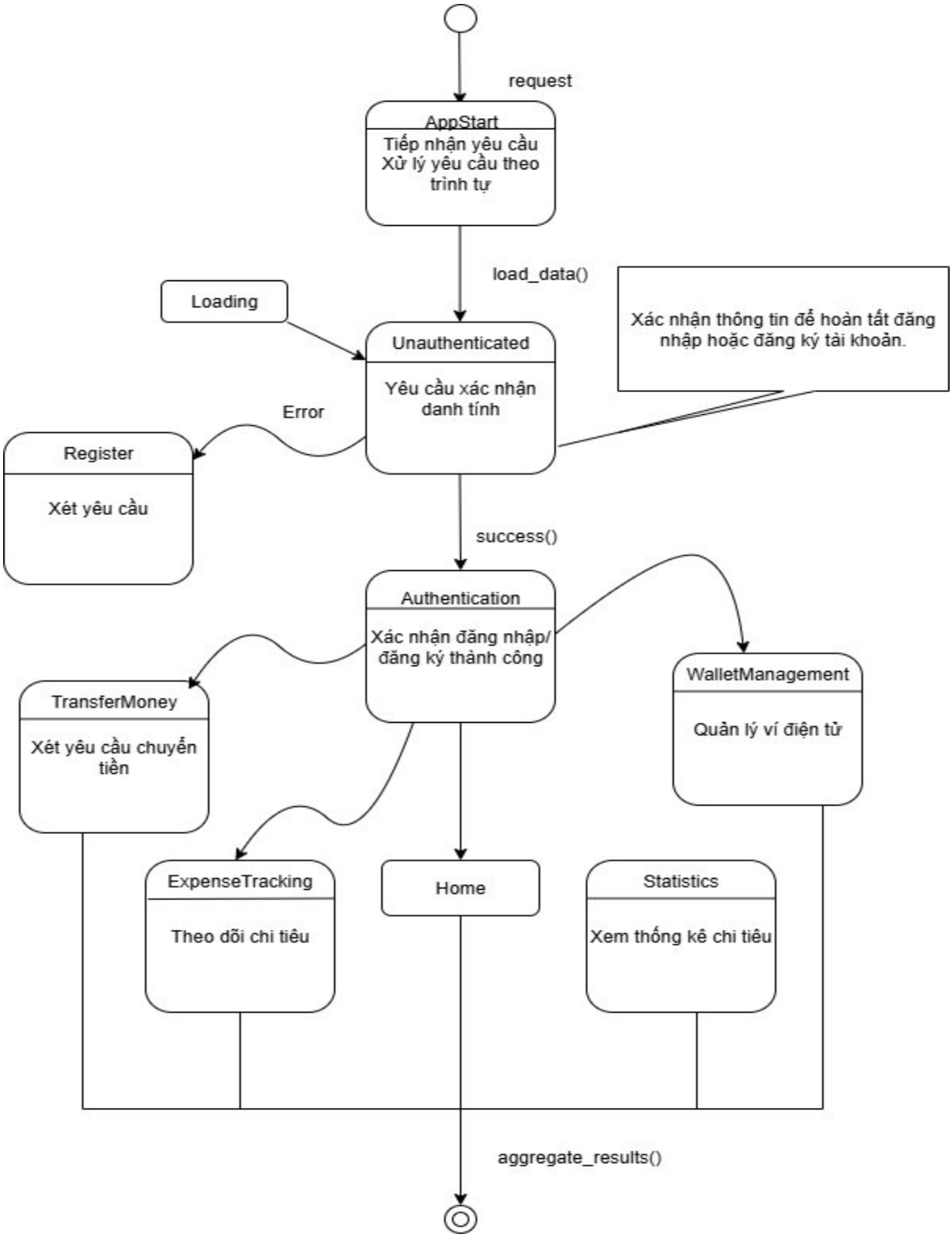
Hình 3: Sơ đồ trình tự tổng thể

- **Sơ đồ hoạt động tổng thể**



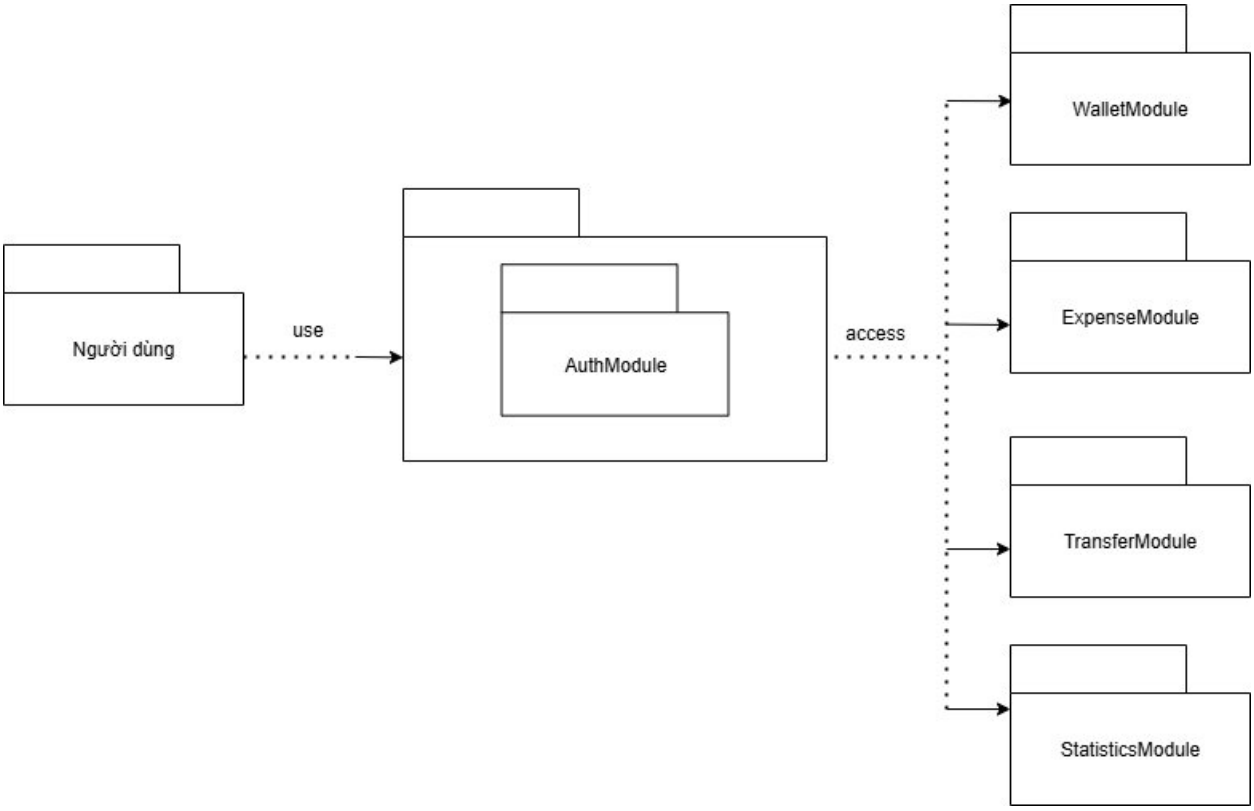
Hình 4: Sơ đồ hoạt động tập thể

- **Sơ đồ trạng thái tổng thể**



Hình 5: Sơ đồ trạng thái tổng thể

- **Sơ đồ đóng gói tổng thể**



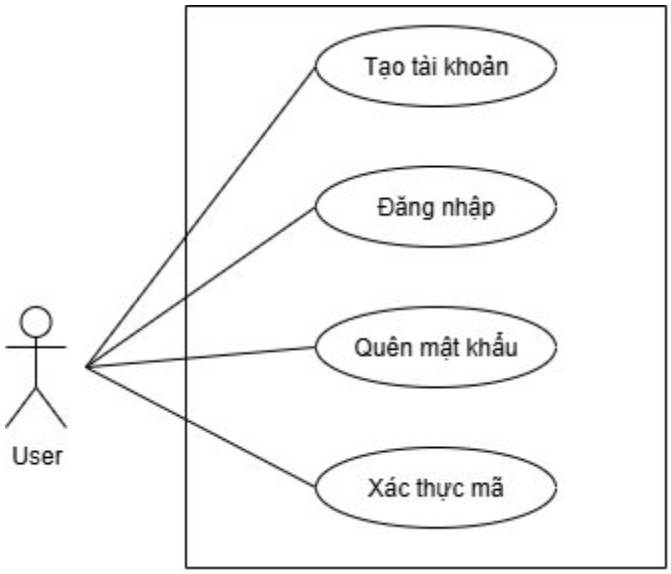
Hình 6: Sơ đồ đóng gói tổng thể

2. Mô tả chi tiết

2.1 Chức năng Đăng nhập/ Đăng ký

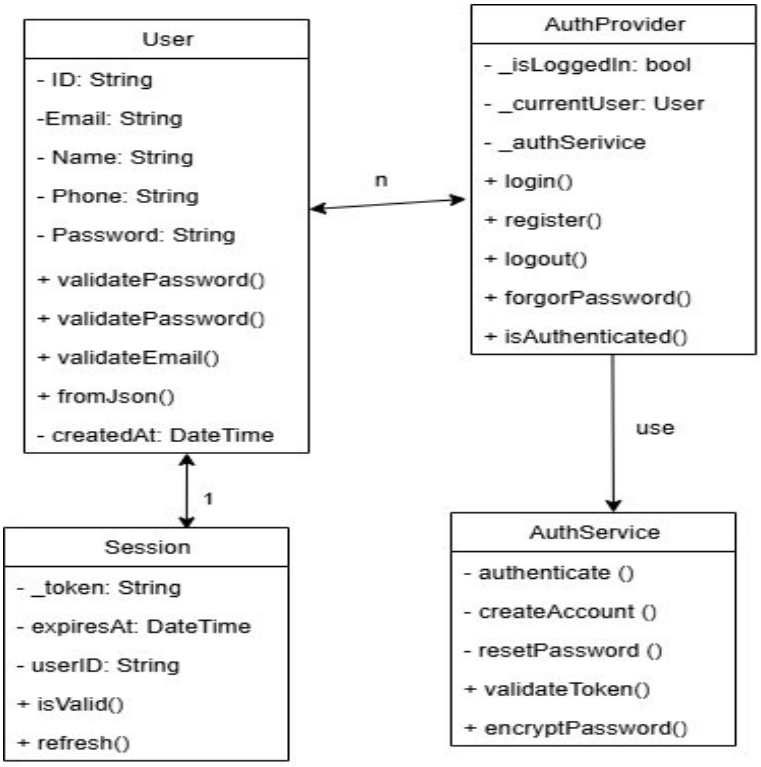
Cho phép người dùng tạo tài khoản mới và truy cập vào hệ thống SmartWallet. Khi đăng ký, người dùng cung cấp thông tin cơ bản như email, mật khẩu và xác nhận mật khẩu. Sau khi đăng ký thành công, họ có thể đăng nhập để sử dụng các tính năng của ứng dụng. Quá trình đăng nhập được kiểm tra hợp lệ nhằm đảm bảo an toàn và bảo mật cho tài khoản người dùng.

- **Sơ đồ use case Chức năng Đăng nhập/ Đăng ký**



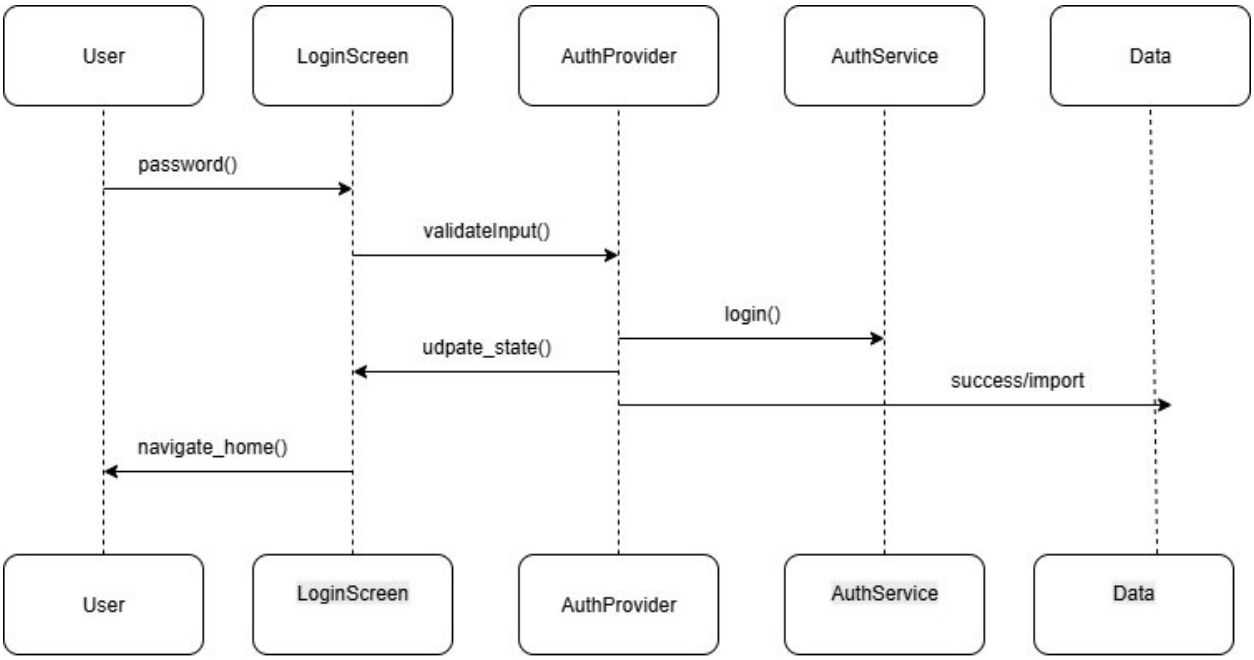
Hình 7: Use case đăng ký/đăng nhập

- **Sơ đồ lớp Chức năng Đăng nhập/ Đăng ký**



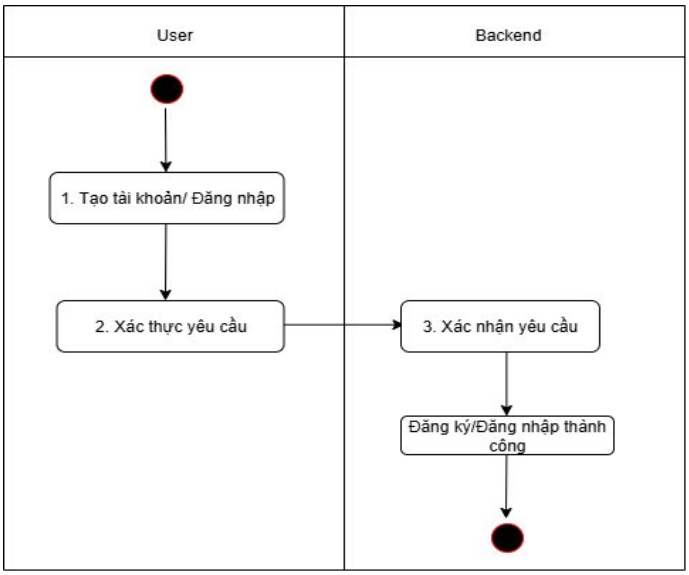
Hình 8: Sơ đồ lớp Chức năng Đăng ký/Đăng nhập

- **Sơ đồ trình tự Chức năng Đăng nhập/ Đăng ký**



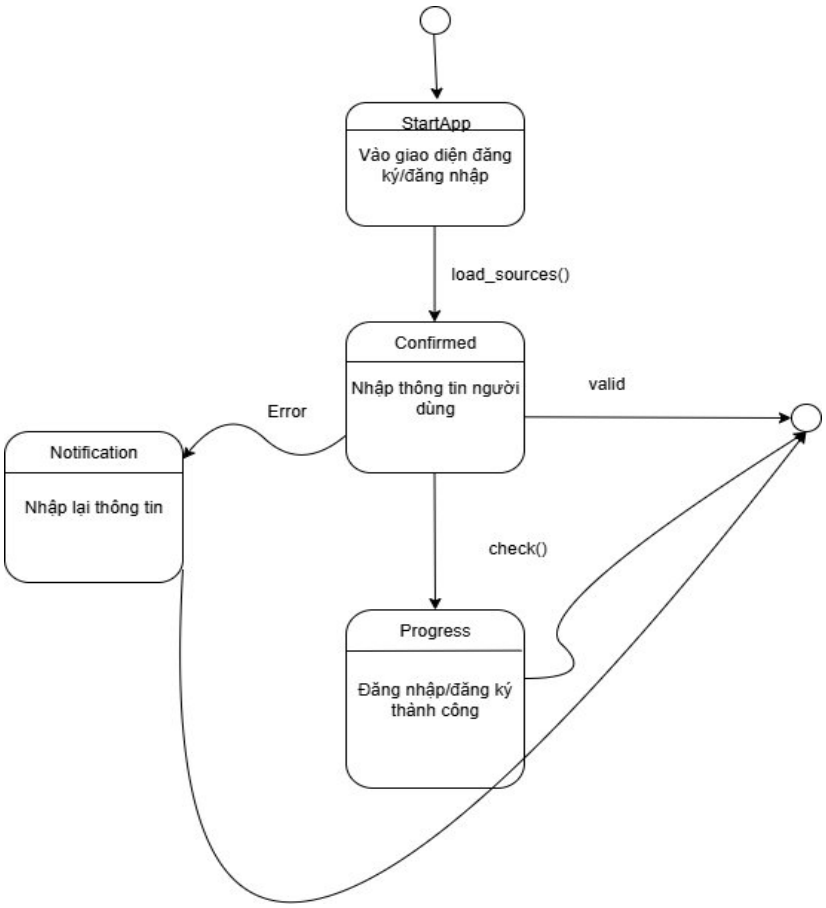
Hình 9: Sơ đồ trình tự chức năng Đăng ký/ Đăng nhập

- **Sơ đồ hoạt động Chức năng Đăng nhập/ Đăng ký**



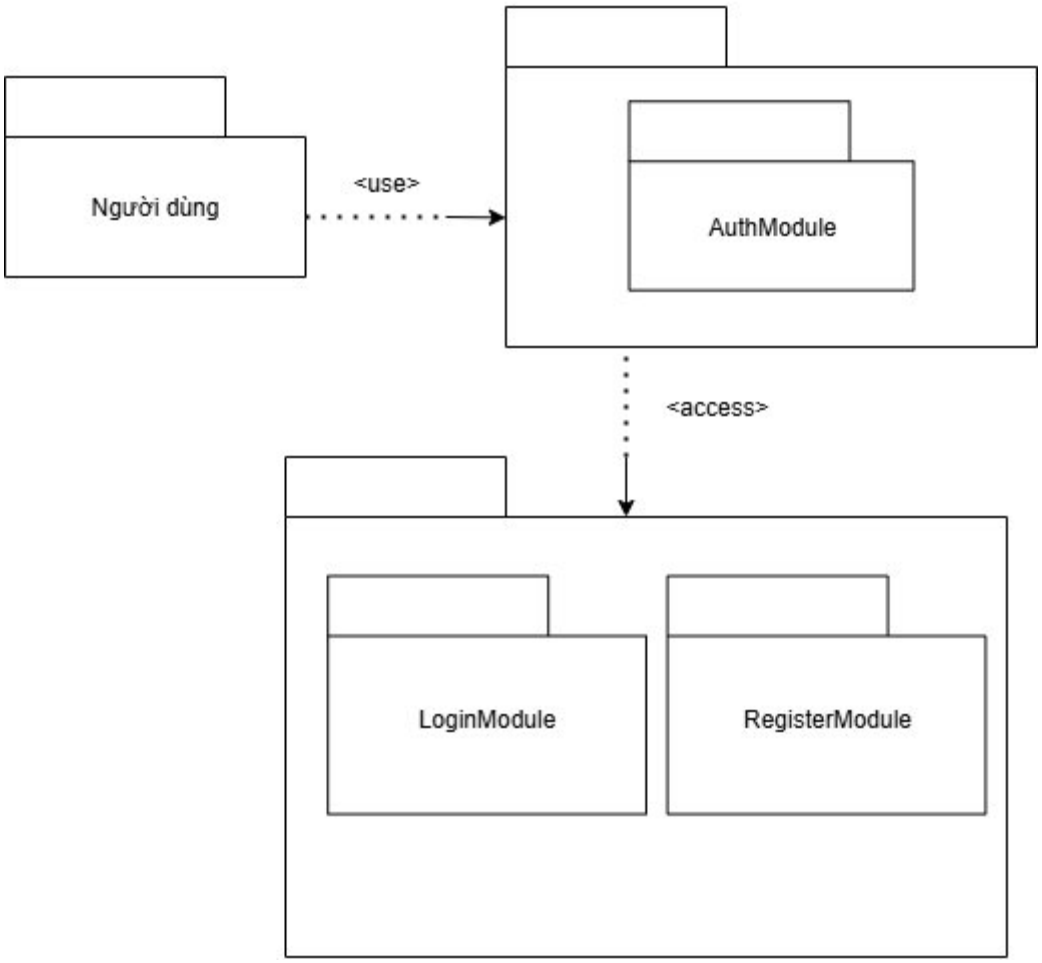
Hình 10: Sơ đồ hoạt động của chức năng Đăng ký/Đăng nhập

- **Sơ đồ trạng thái Chức năng Đăng nhập/ Đăng ký**



Hình 11: Sơ đồ trạng thái của chức năng Đăng ký/Đăng nhập

- **Sơ đồ đóng gói Chức năng Đăng nhập/ Đăng ký**



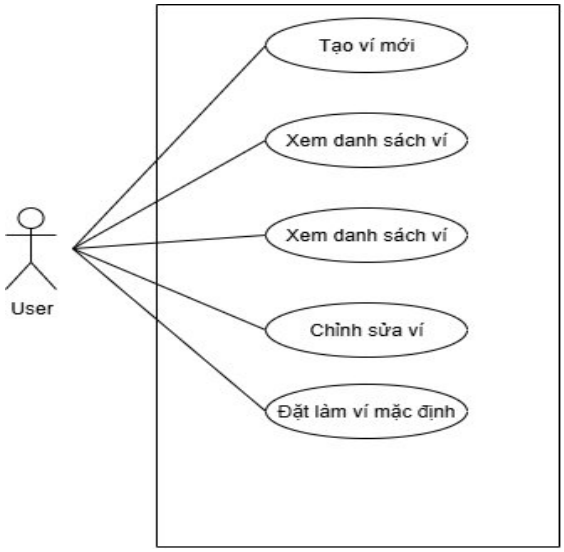
Hình 12: Sơ đồ đóng gói của chức năng Đăng ký/Đăng nhập

2.2 Chức năng Quản lý Ví điện tử thông minh

Quản lý ví là một trong những tính năng cốt lõi và quan trọng nhất của **SmartWallet**, giúp người dùng dễ dàng kiểm soát và tổ chức toàn bộ tài sản cá nhân của mình một cách khoa học và trực quan. Với tính năng này, người dùng có thể **tạo và quản lý nhiều loại ví điện tử khác nhau**, bao gồm ví tiền mặt, tài khoản ngân hàng, thẻ tín dụng, ví tiết kiệm hoặc thậm chí là ví đầu tư.
Mỗi ví trong hệ thống có thể được **tùy chỉnh linh hoạt** về màu sắc, biểu tượng, tên gọi và các thông tin chi tiết khác, giúp người dùng dễ dàng phân biệt và nhận diện từng loại ví theo nhu cầu sử dụng. Bên cạnh đó, **SmartWallet** tự động **tính toán và cập nhật tổng số dư** của tất cả các ví, đảm bảo người dùng luôn nắm bắt chính xác tình hình tài chính của mình trong thời gian thực.

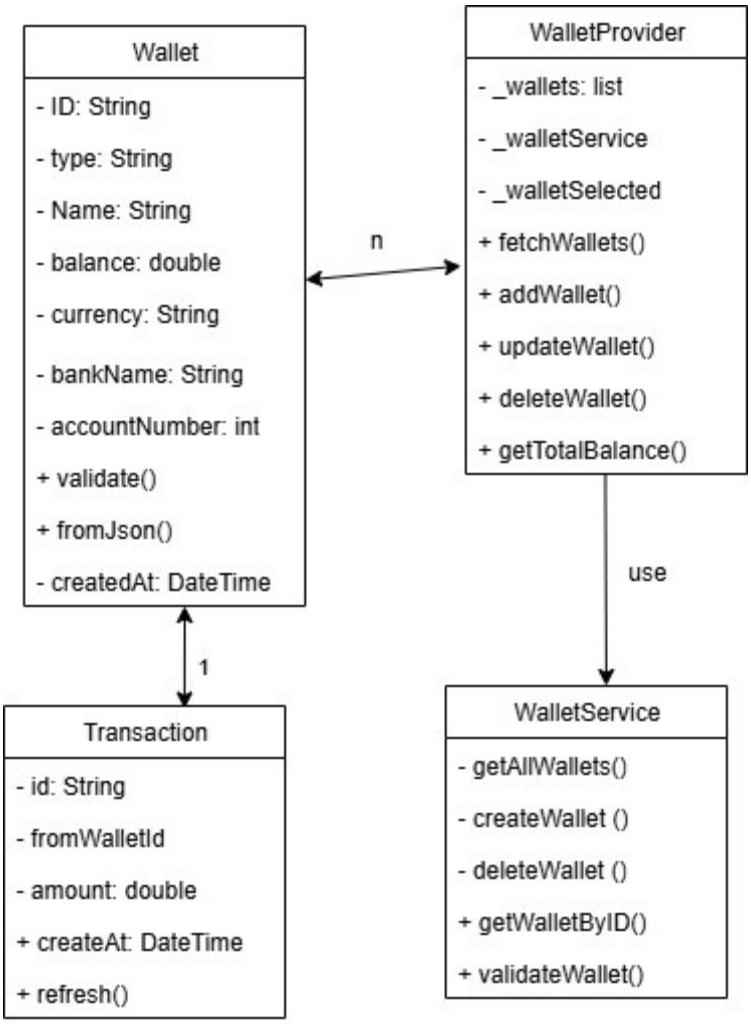
Giao diện của tính năng được thiết kế **trực quan, thân thiện và sinh động**, cho phép người dùng theo dõi dòng tiền, biến động số dư và phân bổ tài sản chỉ trong vài thao tác. Ngoài ra, hệ thống còn hỗ trợ **đồng bộ dữ liệu và sao lưu an toàn**, giúp bảo vệ thông tin tài chính cá nhân một cách tối ưu.
Tóm lại, tính năng quản lý ví của **SmartWallet** không chỉ là công cụ theo dõi tài sản đơn thuần, mà còn là **trợ lý tài chính thông minh**, hỗ trợ người dùng quản lý, phân tích và tối ưu hóa chi tiêu một cách hiệu quả và tiện lợi nhất.

- **Sơ đồ use case Chức năng Quản lý ví**



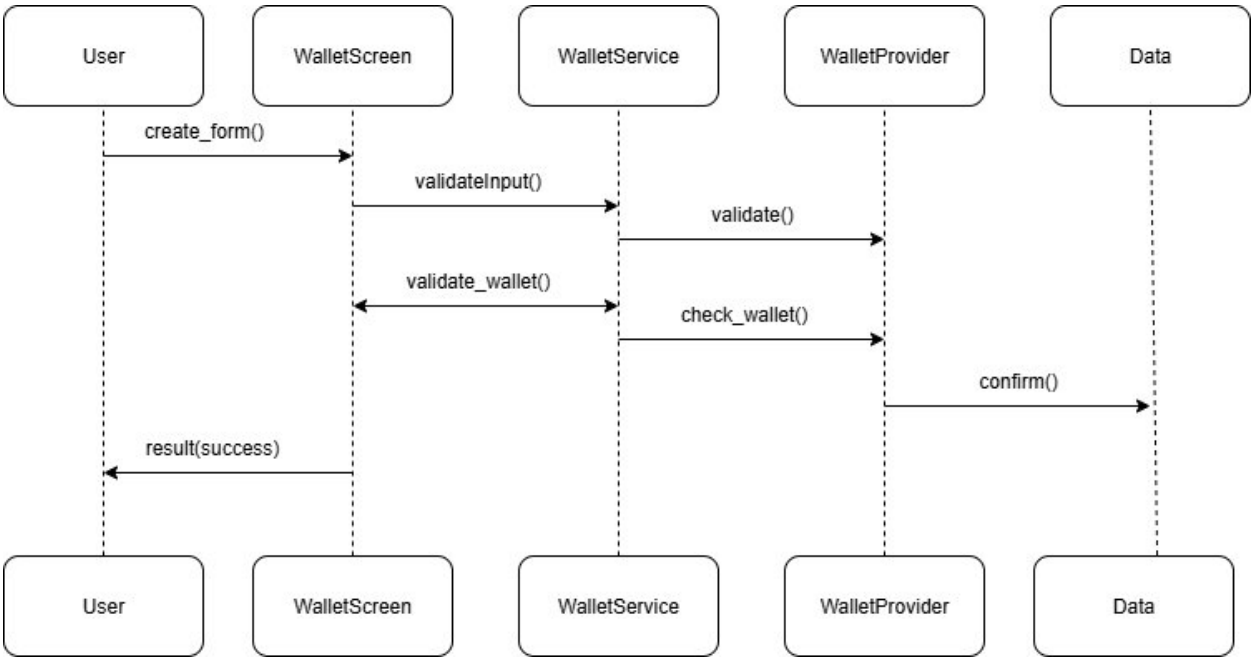
Hình 13: Use case chức năng ví điện tử

- **Sơ đồ lớp Chức năng Quản lý ví**



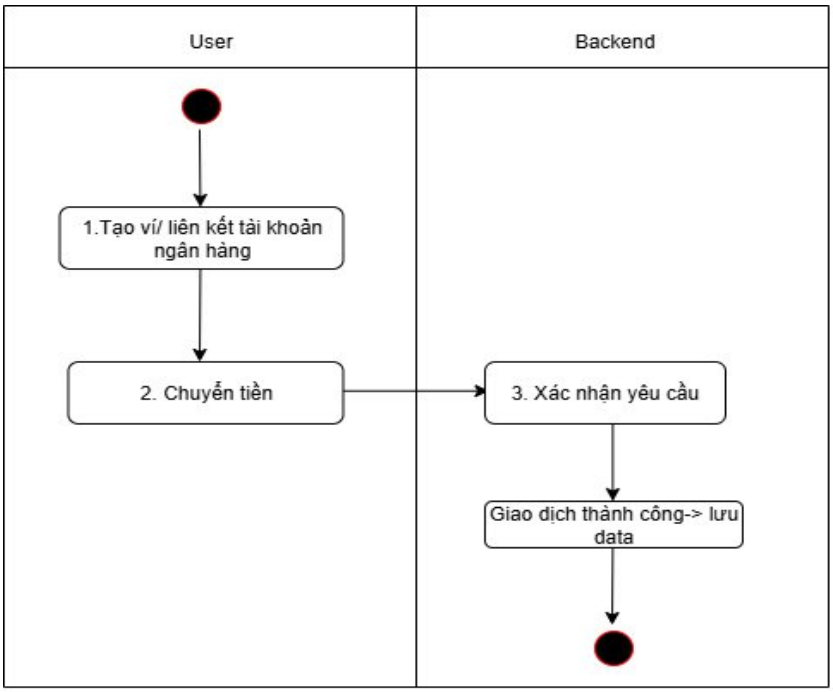
Hình 14: Sơ đồ lớp chức năng quản lý ví điện tử

- **Sơ đồ trình tự Chức năng Quản lý ví**



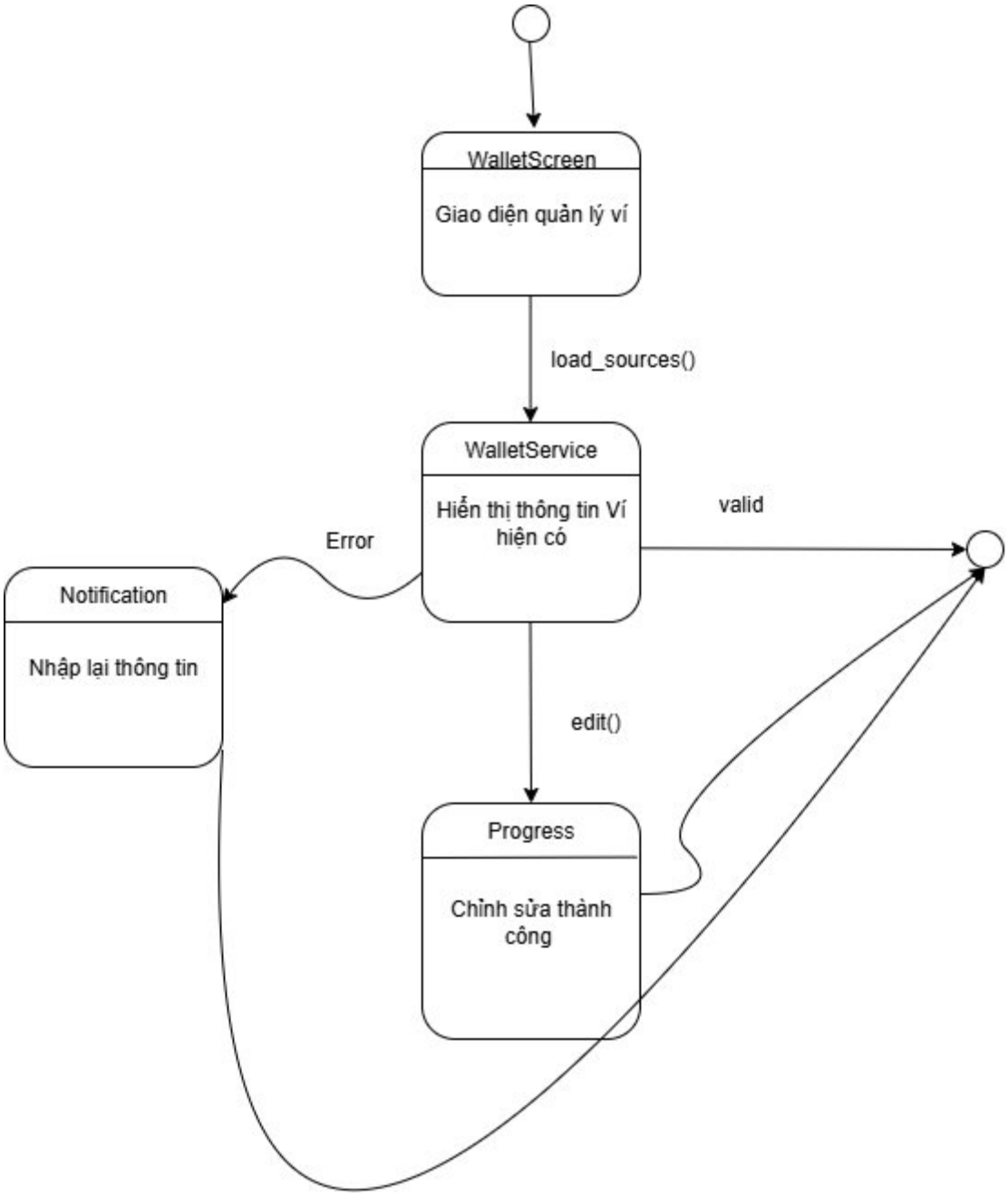
Hình 15: Sơ đồ trình tự chức năng quản lý ví

- **Sơ đồ hoạt động Chức năng Quản lý ví**



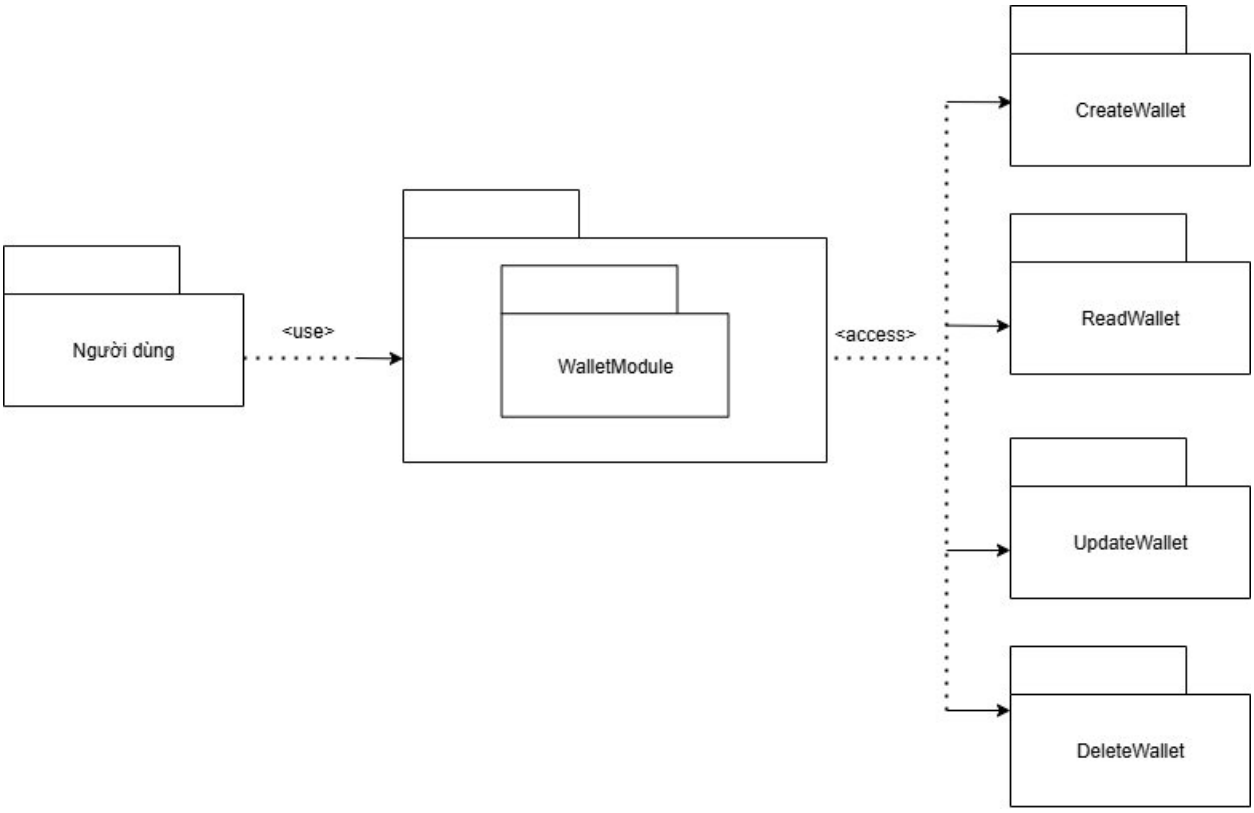
Hình 16: Sơ đồ hoạt động chức năng quản lý ví điện tử

- **Sơ đồ trạng thái Chức năng Quản lý ví**



Hình 17: Sơ đồ trạng thái chức năng quản lý ví điện tử

- **Sơ đồ đóng gói Chức năng Quản lý ví**



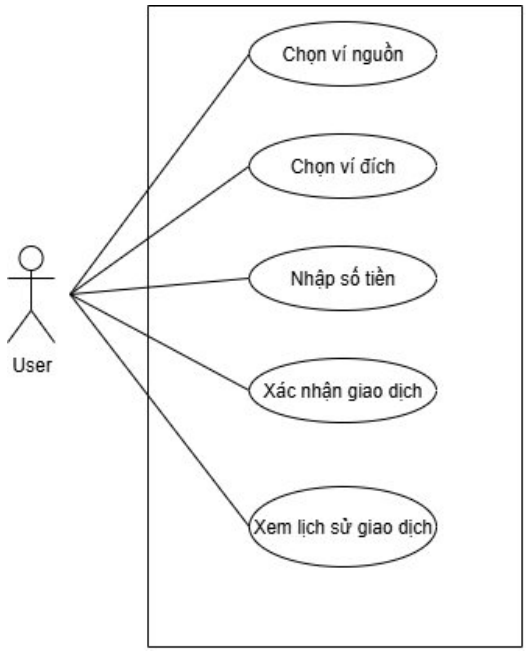
Hình 18: Sơ đồ đóng gói chức năng quản lý ví điện tử

2.3 Chức năng Giao dịch và Chuyển tiền

- **Chức năng chuyển tiền** là một trong những tính năng quan trọng và tiện ích nhất của **SmartWallet**, giúp người dùng thực hiện các giao dịch tài chính **nhanh chóng, an toàn và chính xác tuyệt đối**. Với tính năng này, người dùng có thể **chuyển tiền linh hoạt giữa các ví khác nhau** – chẳng hạn như từ ví tiền mặt sang ví ngân hàng, từ thẻ tín dụng sang ví chi tiêu, hoặc giữa các tài khoản cá nhân khác nhau – chỉ trong vài thao tác đơn giản.
- Trước khi giao dịch được thực hiện, hệ thống sẽ **tự động kiểm tra số dư** để đảm bảo khả năng thanh toán, đồng thời **xác thực thông tin giao dịch** bằng các lớp bảo mật nâng cao như mã PIN, sinh trắc học hoặc xác thực hai lớp (2FA). Sau khi giao dịch hoàn tất, **SmartWallet** ngay lập tức **cập nhật số dư của từng ví**, giúp người dùng luôn theo dõi được tình hình tài chính của mình **theo thời gian thực**.

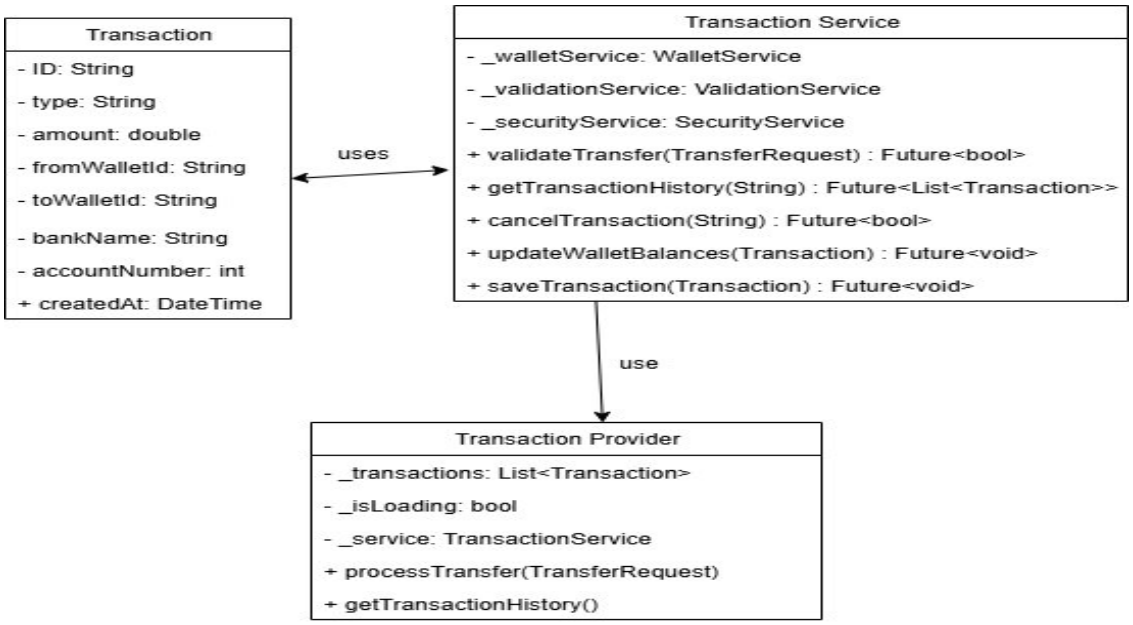
- Mỗi giao dịch đều được **ghi lại chi tiết trong lịch sử**, bao gồm thông tin người gửi, người nhận, thời gian, số tiền và trạng thái. Người dùng có thể **tra cứu, đối chiếu hoặc tải báo cáo** bất kỳ lúc nào, đảm bảo tính minh bạch và kiểm soát tuyệt đối trong mọi hoạt động tài chính.
- Nhờ sự kết hợp giữa **giao diện thân thiện**, **quy trình xử lý tối ưu** và **cơ chế bảo mật tiên tiến**, chức năng chuyển tiền của **SmartWallet** mang đến cho người dùng **trải nghiệm giao dịch mượt mà, tiện lợi và an tâm tối đa** – đáp ứng trọn vẹn nhu cầu tài chính hiện đại trong kỷ nguyên số.

- **Sơ đồ use case Chức năng giao dịch và chuyển tiền**



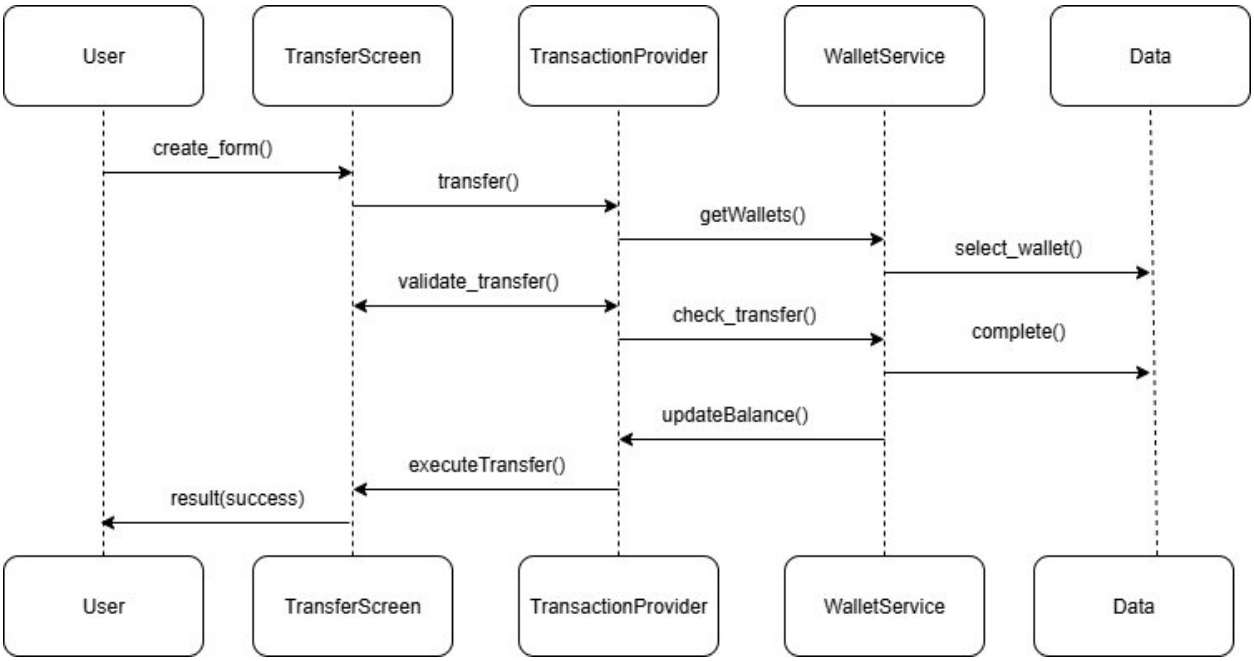
Hình 19: Use case chức năng giao dịch và chuyển tiền

- **Sơ đồ lớp Chức năng giao dịch và chuyển tiền**



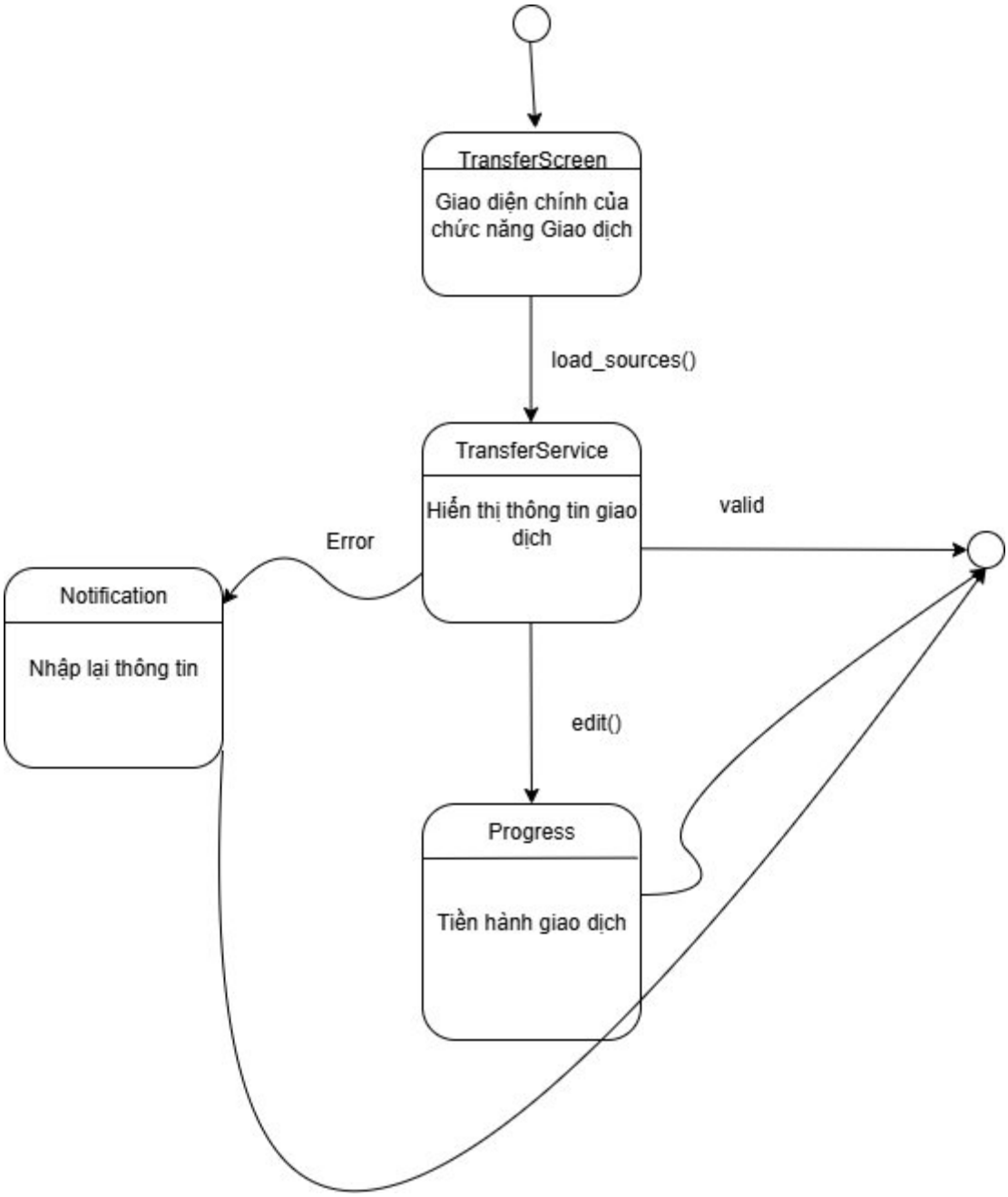
Hình 20: Sơ đồ lớp chức năng giao dịch và chuyển tiền

- **Sơ đồ trình tự Chức năng giao dịch và chuyển tiền**



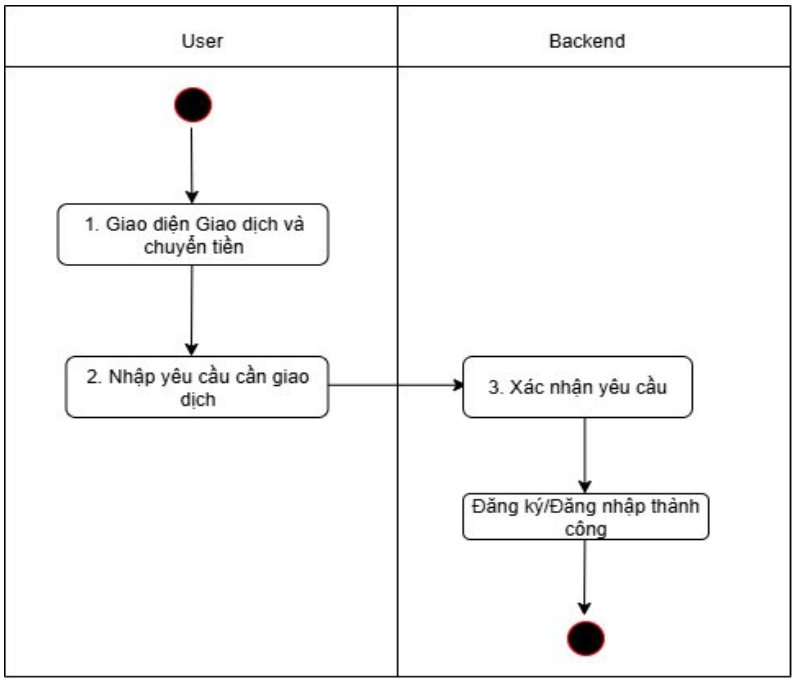
Hình 21: Sơ đồ trình tự chức năng Giao dịch và Chuyển tiền

- **Sơ đồ trạng thái Chức năng giao dịch và chuyển tiền**



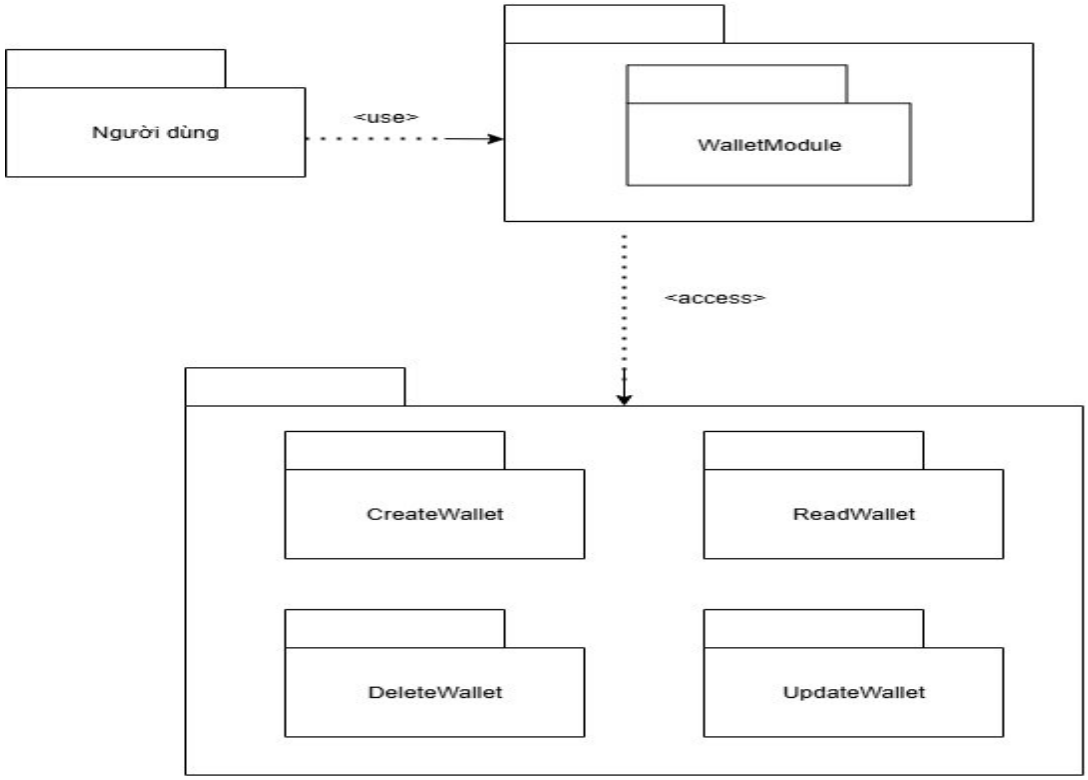
Hình 22: Sơ đồ trình tự chức năng Giao dịch và Chuyển tiền

- **Sơ đồ hoạt động Chức năng giao dịch và chuyển tiền**



- Hình 23: Sơ đồ hoạt động chức năng Giao dịch và Chuyển tiền

- **Sơ đồ đóng gói Chức năng giao dịch và chuyển tiền**

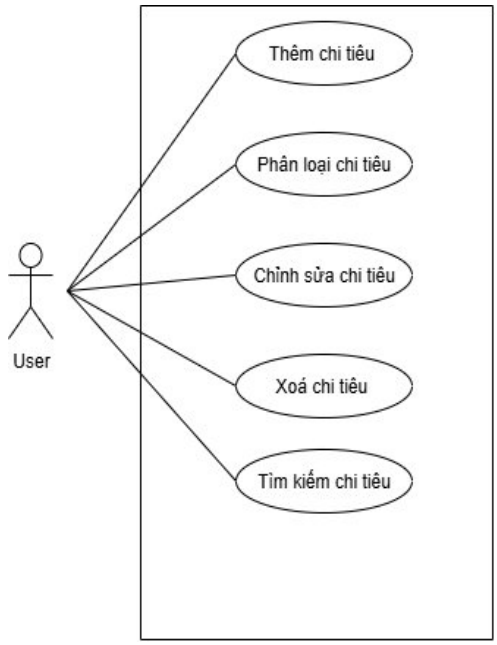


Hình 24: Sơ đồ đóng gói chức năng Giao dịch và Chuyển tiền

2.4 Chức năng Quản lý chi tiêu

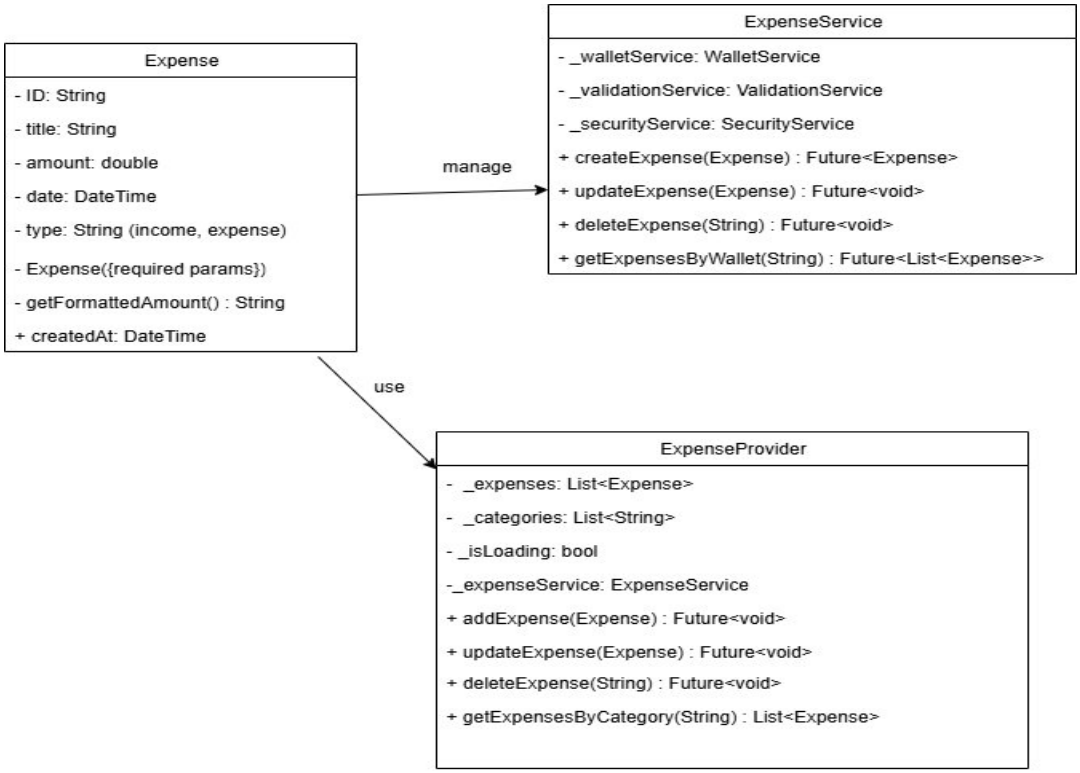
Mô tả chức năng
Quản lý chi tiêu giúp người dùng dễ dàng **theo dõi, ghi chép và phân loại** các khoản **thu – chi hằng ngày** một cách khoa học. Hệ thống hỗ trợ nhiều **danh mục chi tiêu phong phú** như *ăn uống, mua sắm, giải trí, giáo dục*… và cho phép **tùy chỉnh linh hoạt** theo nhu cầu cá nhân. Mỗi giao dịch có thể **đính kèm ghi chú, hình ảnh minh họa** và **liên kết trực tiếp với ví tương ứng**, giúp **tự động cập nhật số dư** và mang lại **trải nghiệm quản lý tài chính trực quan, tiện lợi**.

- **Sơ đồ use case Chức năng Quản lý chi tiêu**



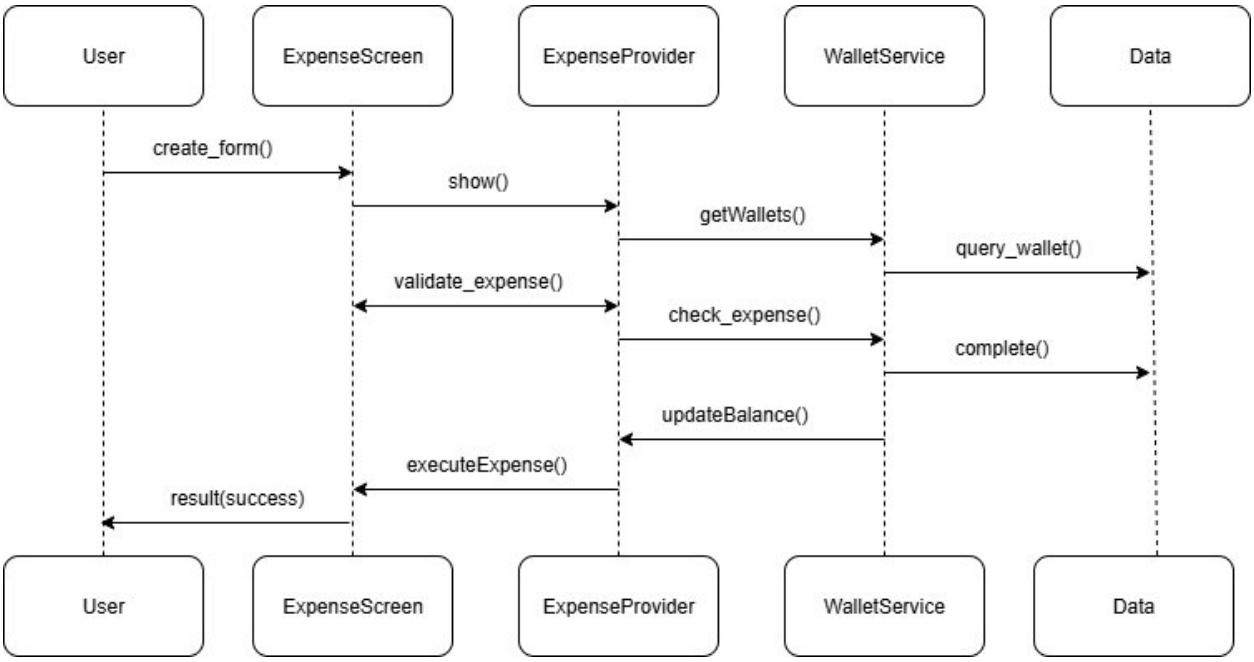
Hình 25: Use case quản lý chi tiêu

- **Sơ đồ lớp Chức năng Quản lý chi tiêu**



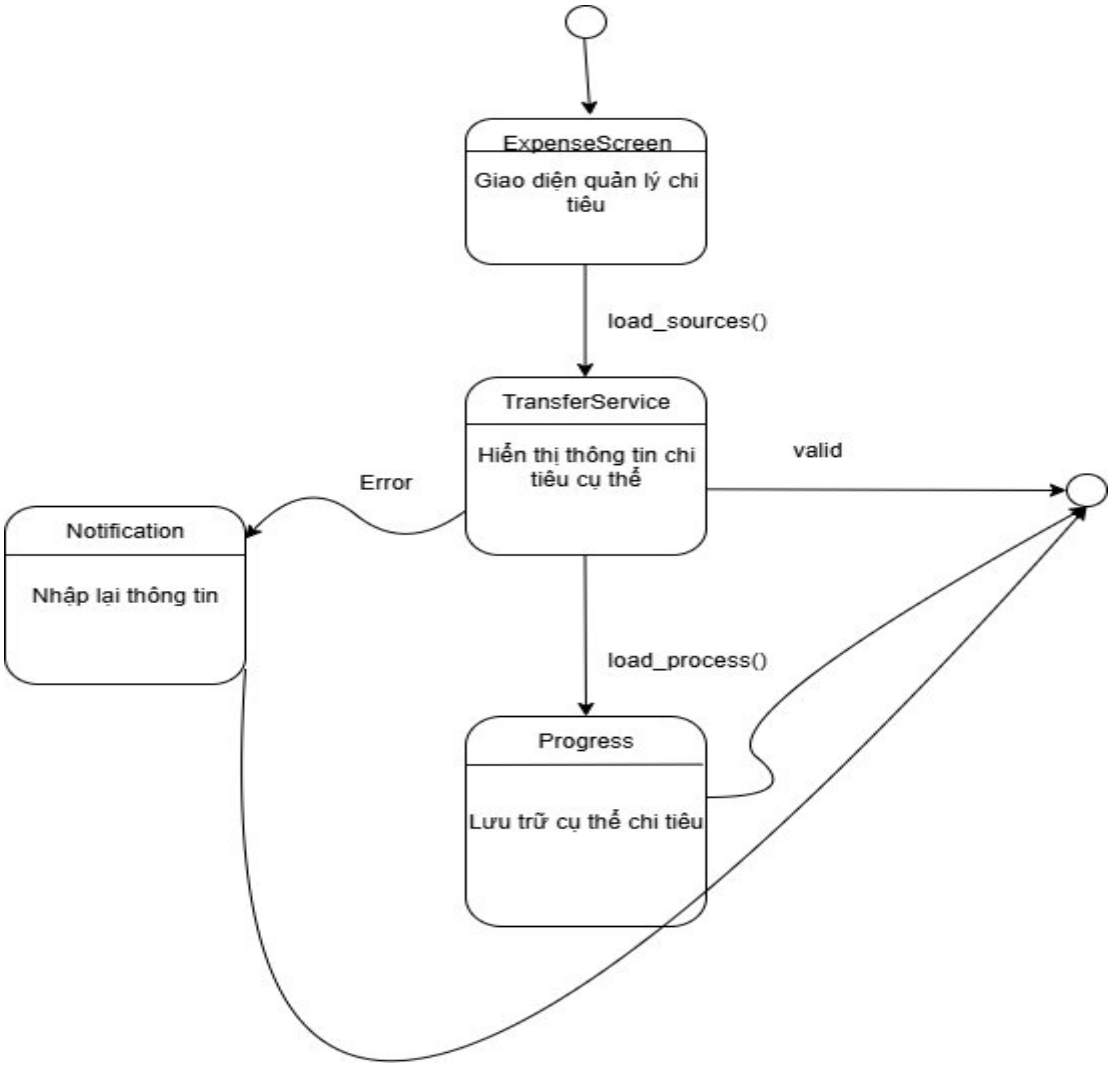
Hình 26: Sơ đồ lớp chức năng quản lý chi tiêu

- **Sơ đồ tuần tự Quản lý chi tiêu**



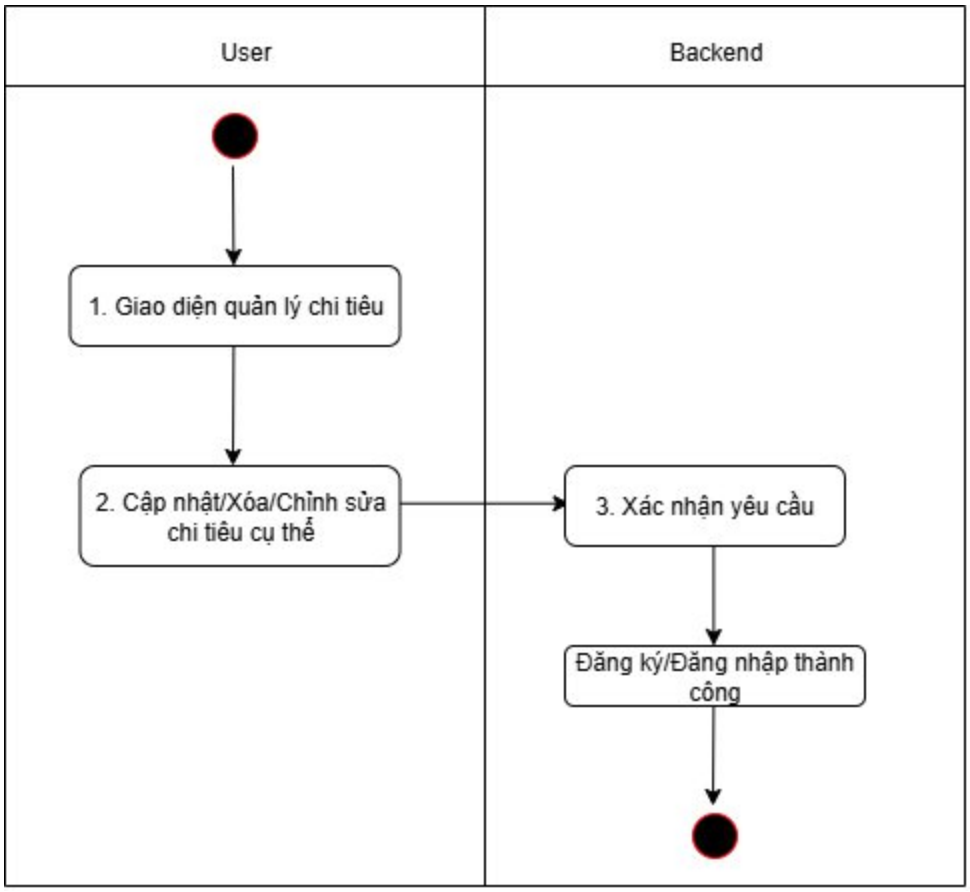
Hình 27: Biểu đồ trình tự chức năng quản lý chi tiêu

- **Sơ đồ trạng thái Quản lý chi tiêu**



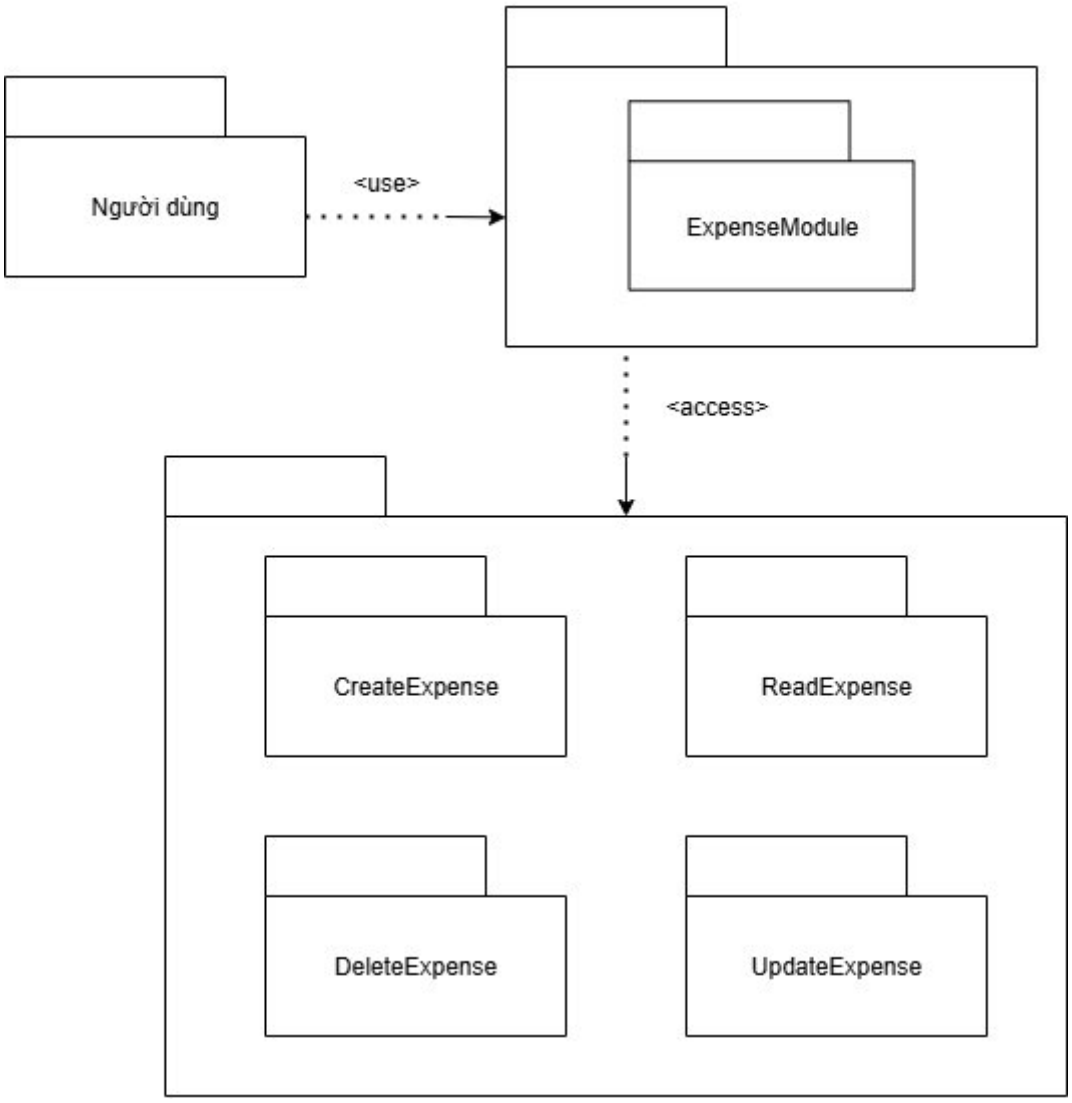
Hình 28: Biểu đồ trạng thái chức năng quản lý chi tiêu

- **Sơ đồ hoạt động Quản lý chi tiêu**



Hình 29: Biểu đồ hoạt động chức năng quản lý chi tiêu

- **Sơ đồ đóng gói Quản lý chi tiêu**



Hình 29: Sơ đồ đóng gói chức năng quản lý chi tiêu

Chương 4: Công nghệ và công cụ

1 Công nghệ sử dụng

1.1 Framework và ngôn ngữ

- **Flutter 3.x** – Framework phát triển ứng dụng đa nền tảng (Android, iOS, Web, Desktop) với hiệu năng cao và UI linh hoạt.
- **Dart** – Ngôn ngữ lập trình chính hỗ trợ kiểu tĩnh, tối ưu biên dịch AOT/JIT.
- **SDK yêu cầu** – Flutter SDK phiên bản ≥ 3.6.2.

1.2 Thư viện và Packages

- **State Management:** Provider (^6.1.2) – Quản lý trạng thái theo hướng đơn giản, dễ mở rộng.
- **UI Components:**
  - material_symbols_icons (^4.2719.3) – Cung cấp bộ biểu tượng Material 3 mới nhất.
  - flutter_spinkit (^5.2.1) – Hiệu ứng loading động, tối ưu cho UX.
- **Charts:** fl_chart (^0.70.2) – Vẽ biểu đồ tương tác trực quan.
- **Calendar:** table_calendar (^3.2.0) – Hỗ trợ lịch tuỳ biến cao, phù hợp quản lý sự kiện.
- **Utilities:**
  - uuid (^4.5.1) – Sinh ID duy nhất cho dữ liệu.
  - intl (^0.20.2) – Định dạng số, ngày giờ, đa ngôn ngữ.
  - crypto (^3.0.3) – Băm, mã hoá dữ liệu an toàn.

2 Công cụ phát triển

- **IDE:** Android Studio / Visual Studio Code (hỗ trợ plugin Flutter, DevTools, hot reload).
- **SDK:** Flutter SDK, Android SDK, iOS SDK (dùng để build, debug, deploy đa nền tảng).
- **Version Control:** Git (quản lý phiên bản, làm việc nhóm hiệu quả qua GitHub/GitLab).
- **Design & Prototype:** Figma (thiết kế mockup UI, prototype tương tác trước khi phát triển).

3 Lý do lựa chọn công nghệ

3.1 Tại sao chọn Flutter?

- Phát triển **đa nền tảng** với **một codebase duy nhất**.
- **Hiệu năng gần native**, render qua Skia engine.

- **UI framework mạnh mẽ**, hỗ trợ Material Design & Cupertino.
- **Hot reload**, tăng tốc độ phát triển & thử nghiệm.
- **Cộng đồng lớn**, tài liệu và ecosystem phong phú.

3.2 Tại sao chọn Provider?

- **Cơ chế state management rõ ràng, dễ hiểu**, không phức tạp như BLoC.
- **Tích hợp gọn với Flutter**, dễ bắt đầu và mở rộng.
- **Hiệu năng ổn định cho ứng dụng vừa và nhỏ**.
- **Dễ kiểm thử (unit test) và debug**, giảm thiểu side-effect.

Chương 4 cung cấp cái nhìn tổng quan về nền tảng công nghệ và công cụ sử dụng, đảm bảo hệ thống được phát triển với hiệu năng cao, dễ bảo trì và khả năng mở rộng trong tương lai.

Chương 5: TRIỂN KHAI VÀ CÀI ĐẶT

1 Cấu trúc dự án chi tiết

1.1 Tổng quan cấu trúc

Dự án **SmartWallet** được xây dựng dựa trên sự kết hợp giữa **Clean Architecture** và **Feature-based Structure**, nhằm đảm bảo khả năng **module hóa, dễ mở rộng, dễ bảo trì**, và **phân tách rõ ràng giữa các tầng** trong ứng dụng.

Cấu trúc thư mục được tổ chức như sau:

```
vi_dien_tu/
├── lib/
│   ├── main.dart                # Điểm khởi chạy của ứng dụng
│   ├── models/                  # Định nghĩa các lớp mô hình dữ liệu
│   │   ├── user.dart            # Thông tin người dùng
│   │   ├── wallet.dart          # Dữ liệu ví điện tử
│   │   ├── transaction.dart     # Giao dịch tài chính
│   │   ├── expense.dart         # Quản lý chi tiêu
│   │   ├── financial_goal.dart  # Mục tiêu tài chính
│   │   └── notification.dart    # Thông báo hệ thống
│   ├── providers/               # State Management (Provider Pattern)
│   │   ├── user_provider.dart
│   │   ├── wallet_provider.dart
│   │   ├── expense_provider.dart
│   │   ├── transaction_provider.dart
```

```
│   │   ├── settings_provider.dart
│   │   └── notification_provider.dart
│   ├── services/                # Business Logic Layer
│   │   ├── auth_service.dart       # Xác thực & đăng nhập người dùng
│   │   ├── database_service.dart    # Tương tác cơ sở dữ liệu
│   │   ├── transfer_service.dart    # Dịch vụ chuyển tiền
│   │   ├── notification_service.dart# Xử lý thông báo
│   │   ├── security_service.dart    # Mã hóa và bảo mật dữ liệu
│   │   └── qr_service.dart          # Quét và tạo mã QR
│   ├── screens/                 # Presentation Layer (UI + UX)
│   │   ├── authentication/          # Đăng nhập, đăng ký, quên mật khẩu
│   │   ├── wallet/                  # Quản lý ví điện tử
│   │   ├── transfer/                # Giao diện chuyển tiền
│   │   ├── expenses/                # Theo dõi và thống kê chi tiêu
│   │   ├── calendar/                # Lịch tài chính cá nhân
│   │   ├── settings/                # Cài đặt và hồ sơ cá nhân
│   │   ├── qr/                      # Quét mã QR
│   │   ├── services/                # Dịch vụ tiện ích: nạp tiền, thanh toán hóa đơn, vé xem
phim, du lịch
│   │   └── home_screen.dart         # Màn hình chính của ứng dụng
│   ├── widgets/                 # Các thành phần tái sử dụng (UI Components)
│   │   ├── expense_card.dart
│   │   ├── expense_list.dart
│   │   ├── add_expense_form.dart
│   │   └── edit_expense_form.dart
│   └── utils/                   # Tiện ích và hằng số chung
│       ├── app_colors.dart
│       ├── constants.dart
│       └── translations.dart
├── assets/                      # Tài nguyên tĩnh (hình ảnh, biểu tượng, phông chữ)
│   ├── images/
│   ├── icons/
│   └── fonts/
├── test/                        # Unit Tests
│   └── widget_test.dart
├── pubspec.yaml                 # Quản lý dependencies của dự án
└── README.md                    # Tài liệu hướng dẫn và mô tả dự án
```

1.2 Kiến trúc phân lớp

1. Presentation Layer (UI)

Đây là tầng giao diện người dùng, chịu trách nhiệm hiển thị dữ liệu và tương tác trực tiếp với người dùng.

- **Screens:** Chứa các màn hình chính của ứng dụng như đăng nhập, ví điện tử, chuyển tiền, chi tiêu, và cài đặt.
- **Widgets:** Bao gồm các thành phần giao diện tái sử dụng (component) như thẻ chi tiêu, danh sách giao dịch, hoặc form nhập liệu.
- **Providers:** Quản lý trạng thái ứng dụng theo mô hình **Provider pattern**, giúp cập nhật dữ liệu theo thời gian thực giữa các thành phần UI và logic nghiệp vụ.

2. Business Logic Layer

Tầng này chịu trách nhiệm xử lý toàn bộ **logic nghiệp vụ** của ứng dụng.

- **Services:** Thực hiện các chức năng cốt lõi như xác thực người dùng, chuyển tiền, quản lý thông báo, và bảo mật dữ liệu.
- **Models:** Định nghĩa các lớp mô hình dữ liệu (user, wallet, transaction, expense...) dùng để truyền và lưu trữ thông tin giữa các tầng.
- **Utils:** Chứa các tiện ích hỗ trợ như hằng số, định dạng dữ liệu, và cấu hình đa ngôn ngữ, giúp mã nguồn thống nhất và dễ bảo trì.

3. Data Layer

Đây là tầng phụ trách **lưu trữ và quản lý dữ liệu** của toàn hệ thống.

- **Local Storage:** Sử dụng **SharedPreferences** để lưu trữ dữ liệu cục bộ như thông tin cài đặt hoặc trạng thái đăng nhập của người dùng.
- **Mock Data:** Cung cấp dữ liệu mẫu phục vụ quá trình **demo** và **testing** khi chưa kết nối với backend thật.
- **Database Service:** Đóng vai trò là trung tâm quản lý dữ liệu, chịu trách nhiệm đọc – ghi và đồng bộ hóa dữ liệu giữa ứng dụng và hệ thống lưu trữ.

2 Triển khai các tính năng chính

Phần này trình bày quá trình **xây dựng và triển khai các tính năng cốt lõi** của ứng dụng **SmartWallet**. Mỗi tính năng được thiết kế theo hướng **module hóa** và **tuân thủ Clean Architecture**, giúp việc phát triển, bảo trì và mở rộng ứng dụng trở nên dễ dàng hơn.

2.1 Hệ thống xác thực (Authentication)

Hệ thống xác thực chịu trách nhiệm quản lý việc **đăng nhập, đăng ký, và duy trì phiên làm việc của người dùng**.
Dữ liệu được mô phỏng thông qua **mock data**, đảm bảo ứng dụng có thể hoạt động độc lập trong môi trường demo hoặc thử nghiệm.

Quy trình xác thực được triển khai trong tệp auth_service.dart, sử dụng cơ chế kiểm tra thông tin người dùng, lưu session cục bộ, và tự động xác thực khi khởi động lại ứng dụng.

Các tính năng chính:

- Đăng nhập bằng email và mật khẩu.
- Đăng ký tài khoản mới.
- Lưu trữ thông tin phiên đăng nhập (session) của người dùng.
- Tự động xác thực khi mở lại ứng dụng.
- Hỗ trợ đăng xuất và xóa session an toàn.

Nhờ cơ chế này, người dùng có thể **truy cập nhanh vào ứng dụng mà không cần đăng nhập lại**, đồng thời đảm bảo **an toàn dữ liệu và tính riêng tư**.

```
// auth_service.dart - Dịch vụ xác thực
class AuthService {
  static final List<Map<String, String>> _mockUsers = [
    {'email': 'admin@test.com', 'password': '123456', 'name': 'Admin'},
    {'email': 'user@test.com', 'password': '123456', 'name': 'User'},
  ];

  Future<Map<String, dynamic>> login(String email, String password) async {
    // Mô phỏng xác thực với dữ liệu mock
    await Future.delayed(const Duration(seconds: 1));

    final user = _mockUsers.firstWhere(
      (u) => u['email'] == email && u['password'] == password,
      orElse: () => {},
    );

    if (user.isNotEmpty) {
      await _saveUserSession(user);
      return {'success': true, 'user': user};
    }
    throw Exception('Invalid credentials');
```

```
  }
}
```

Hình 30: Code minh họa chức năng đăng ký/đăng nhập

2.2 Quản lý ví điện tử (Wallet Management)

Tính năng quản lý ví điện tử cho phép người dùng **tạo, chỉnh sửa và theo dõi số dư của nhiều loại ví khác nhau**, bao gồm ví tiền mặt, tài khoản ngân hàng, và thẻ tín dụng. Quản lý trạng thái được thực hiện thông qua `WalletProvider`, sử dụng **Provider pattern** để cập nhật giao diện theo thời gian thực.

Các tính năng chính:

- Tạo ví mới với loại ví tùy chọn (tiền mặt, ngân hàng, thẻ tín dụng).
- Hiển thị danh sách ví bằng **UI card** trực quan, sinh động.
- Tính toán tổng số dư của tất cả ví.
- Chọn ví mặc định để thực hiện giao dịch nhanh.
- Tùy chỉnh màu sắc, biểu tượng và tên ví.
- Lưu trữ dữ liệu ví bền vững bằng **SharedPreferences**.

Cấu trúc này giúp người dùng dễ dàng **quản lý tài chính cá nhân** theo từng nguồn tiền, đồng thời tạo cảm giác **trực quan và thân thiện** khi sử dụng.

```
// wallet_provider.dart - State management cho ví
class WalletProvider with ChangeNotifier {
  List<Wallet> _wallets = [];
  Wallet? _selectedWallet;

  List<Wallet> get wallets => _wallets;
  double get totalBalance => _wallets.fold(0.0, (sum, w) => sum + w.balance);

  Future<void> addWallet(Wallet wallet) async {
    wallet.id = const Uuid().v4();
    _wallets.add(wallet);
    await DatabaseService.saveWallets(_wallets);
    notifyListeners();
  }
}
```

Hình 31: Code minh họa chức năng Quản lý ví

2.3 Chuyển tiền giữa các ví (Money Transfer)

Tính năng chuyển tiền được triển khai trong transfer_service.dart, cho phép người dùng **chuyển tiền nội bộ giữa các ví cá nhân**.
Hệ thống đảm bảo các bước **xác thực dữ liệu, kiểm tra số dư và ghi nhận lịch sử giao dịch** được thực hiện chính xác.

Các tính năng chính:

- Chuyển tiền giữa các ví một cách an toàn.
- Kiểm tra tính hợp lệ của số tiền và thông tin giao dịch.
- Ghi lại lịch sử giao dịch để theo dõi chi tiết.
- Giao diện chọn ví nguồn và ví đích thân thiện, dễ thao tác.
- Hộp thoại xác nhận trước khi thực hiện chuyển tiền.
- Thông báo kết quả giao dịch thành công hoặc thất bại.

Nhờ đó, tính năng này mang lại **trải nghiệm thực tế và chuyên nghiệp**, mô phỏng gần như hoàn chỉnh một hệ thống ví điện tử thật.

```dart
// transfer_service.dart - Dịch vụ chuyển tiền
class TransferService {
  static Future<void> transferMoney({
    required String fromWalletId,
    required String toWalletId,
    required double amount,
    required String description,
    required String recipientName,
  }) async {
    // Validation
    if (amount <= 0) throw Exception('Invalid amount');

    // Get wallets
    final wallets = await DatabaseService.getWallets();
    final fromWallet = wallets.firstWhere((w) => w.id == fromWalletId);
    final toWallet = wallets.firstWhere((w) => w.id == toWalletId);

    // Check balance
    if (fromWallet.balance < amount) {
      throw Exception('Insufficient balance');
    }

    // Execute transfer
    fromWallet.balance -= amount;
    toWallet.balance += amount;
```

```
    // Save updated wallets
    await DatabaseService.saveWallets(wallets);

    // Create transaction record
    await _createTransactionRecord(fromWalletId, toWalletId, amount, description);
  }
}
```

Hình 32: Code minh họa chức năng Chuyển tiền

2.4 Quản lý chi tiêu (Expense Tracking)

Tính năng quản lý chi tiêu giúp người dùng **theo dõi và phân loại các khoản thu – chi cá nhân** một cách chi tiết.
Mỗi giao dịch được lưu trữ kèm ngày, danh mục và ghi chú, hỗ trợ người dùng **phân tích thói quen chi tiêu**.

Các tính năng chính:

- Thêm mới thu nhập hoặc chi tiêu với form nhập liệu thân thiện.
- Phân loại giao dịch theo 9 danh mục (ăn uống, di chuyển, sức khỏe, giáo dục...).
- Hỗ trợ chọn ngày giao dịch bằng **Date Picker**.
- Kiểm tra hợp lệ đầu vào để tránh sai sót dữ liệu.
- Hiển thị danh sách chi tiêu với thiết kế card hiện đại.
- Tính toán tổng thu – chi theo từng giai đoạn thời gian.

Tính năng này giúp người dùng **quản lý tài chính cá nhân hiệu quả**, đồng thời hình thành **thói quen chi tiêu có kiểm soát**.

```
// expense_provider.dart - Quản lý chi tiêu
class ExpenseProvider with ChangeNotifier {
  List<Expense> _expenses = [];
  final List<String> _categories = [
    'Ăn uống', 'Di chuyển', 'Nhà cửa', 'Sức khỏe',
    'Giải trí', 'Quần áo', 'Giáo dục', 'Du lịch', 'Khác'
  ];

  Future<void> addExpense(Expense expense) async {
    expense.id = const Uuid().v4();
    _expenses.add(expense);
    await DatabaseService.saveExpenses(_expenses);
    notifyListeners();
  }
```

```
  Map<String, double> getCategoryExpenses() {
    final categoryMap = <String, double>{};
    for (var expense in _expenses.where((e) => e.type == 'Chi tiêu')) {
      categoryMap[expense.category] =
        (categoryMap[expense.category] ?? 0) + expense.amount.abs();
    }
    return categoryMap;
  }
}
```

Hình 33: Code minh họa chức năng Quản lý chi tiêu

2.5 Thống kê và báo cáo (Statistics & Reports)

Màn hình thống kê cung cấp **cái nhìn trực quan về tình hình tài chính** thông qua biểu đồ và thẻ tổng hợp.
Các số liệu được lấy trực tiếp từ `ExpenseProvider`, và hiển thị bằng **biểu đồ tròn (PieChart)** giúp người dùng dễ dàng nhận diện danh mục chi tiêu lớn.

Các tính năng chính:

- Biểu đồ chi tiêu theo danh mục.
- Tổng quan thu – chi qua các thẻ số liệu.
- Bộ lọc theo tuần, tháng, quý hoặc năm.
- Cung cấp gợi ý và insights thông minh về chi tiêu.
- So sánh xu hướng chi tiêu giữa các giai đoạn.
- Giao diện hiện đại với màu sắc đậm chất Việt Nam.

Tính năng này đóng vai trò như **bảng điều khiển tài chính cá nhân**, hỗ trợ người dùng **đưa ra quyết định chi tiêu hợp lý và tối ưu hơn**.

```
// statistics_screen.dart - Màn hình thống kê
class StatisticsScreen extends StatefulWidget {
  Widget _buildCategoryChart(ExpenseProvider expenseProvider) {
    final categoryExpenses = expenseProvider.getCategoryExpenses();

    return PieChart(
      PieChartData(
        sections: _getSections(categoryExpenses),
        centerSpaceRadius: 40,
        sectionsSpace: 2,
      ),
    );
```

```
  }

  List<PieChartSectionData> _getSections(Map<String, double> data) {
    return data.entries.map((entry) {
      return PieChartSectionData(
        color: _getColorForCategory(entry.key),
        value: entry.value,
        title: '${(entry.value / _getTotalExpense() * 100).toStringAsFixed(1)}%',
        radius: 80,
      );
    }).toList();
  }
}
```

Hình 34: Code minh họa chức năng Thống kê và Báo cáo

2.6 Lịch tài chính (Financial Calendar)

Lịch tài chính được triển khai trong calendar_screen.dart, sử dụng thư viện **TableCalendar** để **hiển thị giao dịch theo từng ngày**.
Người dùng có thể xem nhanh các khoản thu – chi trong tháng và truy cập chi tiết chỉ với một chạm.

Các tính năng chính:

- Hiển thị giao dịch theo ngày, tuần hoặc tháng.
- Tổng hợp thu – chi hàng tháng.
- Đánh dấu (marker) các ngày có giao dịch.
- Hỗ trợ chọn ngày để xem chi tiết giao dịch.
- Thiết kế responsive phù hợp trên mọi thiết bị.

Tính năng này giúp người dùng **quản lý lịch sử tài chính một cách trực quan**, dễ dàng theo dõi dòng tiền trong từng giai đoạn.

```
// calendar_screen.dart - Lịch giao dịch
class CalendarScreen extends StatefulWidget {
  List<Expense> _getEventsForDay(DateTime day) {
    return expenseProvider.expenses.where((expense) {
      return isSameDay(expense.date, day);
    }).toList();
  }

  Widget _buildCalendarCard() {
    return TableCalendar<Expense>(
      firstDay: DateTime.utc(2020, 1, 1),
      lastDay: DateTime.utc(2030, 12, 31),
```

```
    focusedDay: _focusedDay,
    eventLoader: _getEventsForDay,
    onDaySelected: _onDaySelected,
    calendarStyle: CalendarStyle(
      selectedDecoration: BoxDecoration(
        color: Colors.indigo,
        shape: BoxShape.circle,
      ),
      markerDecoration: BoxDecoration(
        color: Colors.orange,
        shape: BoxShape.circle,
      ),
    ),
  );
 }
}
```

Hình 34: Code minh họa chức năng Lịch tài chính

2.7 Chế độ đa ngôn ngữ (Internationalization)

Hệ thống đa ngôn ngữ được triển khai trong translations.dart và settings_provider.dart, giúp ứng dụng **tự động thay đổi ngôn ngữ giao diện theo lựa chọn của người dùng**. Dữ liệu dịch thuật được lưu trữ trong các map song ngữ (Tiếng Việt – Tiếng Anh).

Các tính năng chính:

- Hỗ trợ hai ngôn ngữ: **Tiếng Việt và English**.
- Hơn 100 từ khóa được dịch đầy đủ.
- Chuyển đổi ngôn ngữ theo thời gian thực (realtime).
- Lưu trữ cài đặt ngôn ngữ bằng **SharedPreferences**.
- Áp dụng đồng bộ trên toàn bộ giao diện.
- Hỗ trợ thông báo và thông điệp lỗi song ngữ.

Nhờ đó, **SmartWallet** đáp ứng được nhu cầu của cả người dùng Việt Nam và quốc tế, mang lại **trải nghiệm thân thiện và chuyên nghiệp**.

```
// translations.dart - Hệ thống đa ngôn ngữ
class Translations {
 static Map<String, Map<String, String>> translations = {
  'vi': {
   'login': 'Đăng Nhập',
   'wallet': 'Ví',
   'transfer_money': 'Chuyển tiền',
   'statistics': 'Thống kê',
   // ... 100+ từ khóa
```

```
    },
    'en': {
      'login': 'Login',
      'wallet': 'Wallet',
      'transfer_money': 'Transfer Money',
      'statistics': 'Statistics',
      // ... 100+ từ khóa
    },
  };

  static String get(String key, bool isEnglish) {
    String lang = isEnglish ? 'en' : 'vi';
    return translations[lang]?[key] ?? key;
  }
}

// settings_provider.dart - Lưu trữ cài đặt ngôn ngữ
class SettingsProvider with ChangeNotifier {
  bool _isEnglish = false;

  Future<void> toggleLanguage() async {
    _isEnglish = !_isEnglish;
    final prefs = await SharedPreferences.getInstance();
    await prefs.setBool('isEnglish', _isEnglish);
    notifyListeners();
  }
}
```

Hình 35: Code minh họa chức năng Song ngữ Anh-Việt

Chương 6: Thiết kế giao diện người dùng

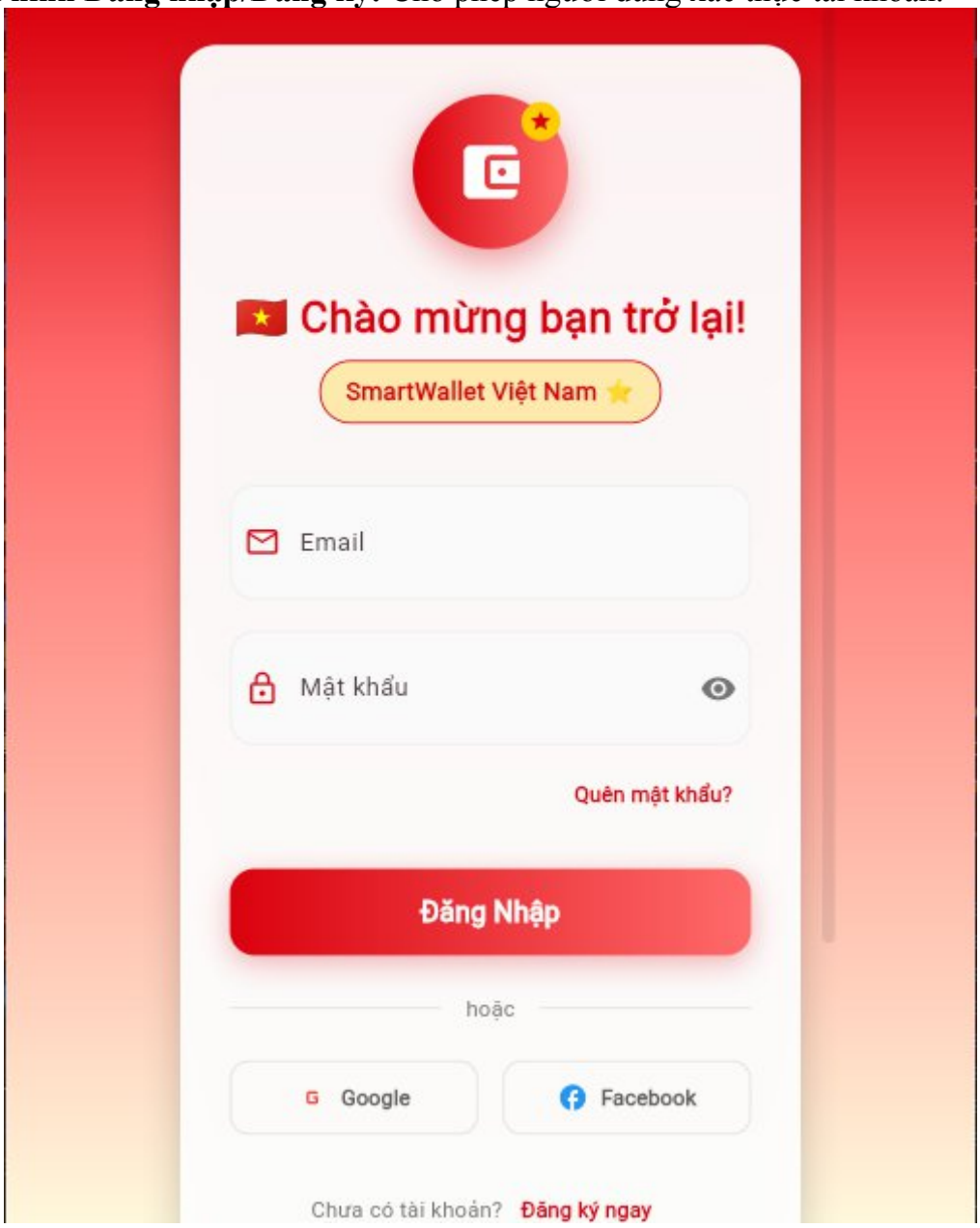
1 Nguyên tắc thiết kế

Giao diện của **SmartWallet** được xây dựng theo triết lý **đơn giản, trực quan và nhất quán**, giúp người dùng thao tác dễ dàng và tập trung vào nội dung chính. Màu sắc chủ đạo được lựa chọn theo phong cách **hiện đại và tin cậy**, kết hợp tông **xanh – tím – trắng** tượng trưng cho sự an toàn và công nghệ. Tất cả các thành phần UI đều tuân thủ **Material Design 3**, đảm bảo tính đồng nhất và khả năng mở rộng trên nhiều thiết bị.

2 Các màn hình chính

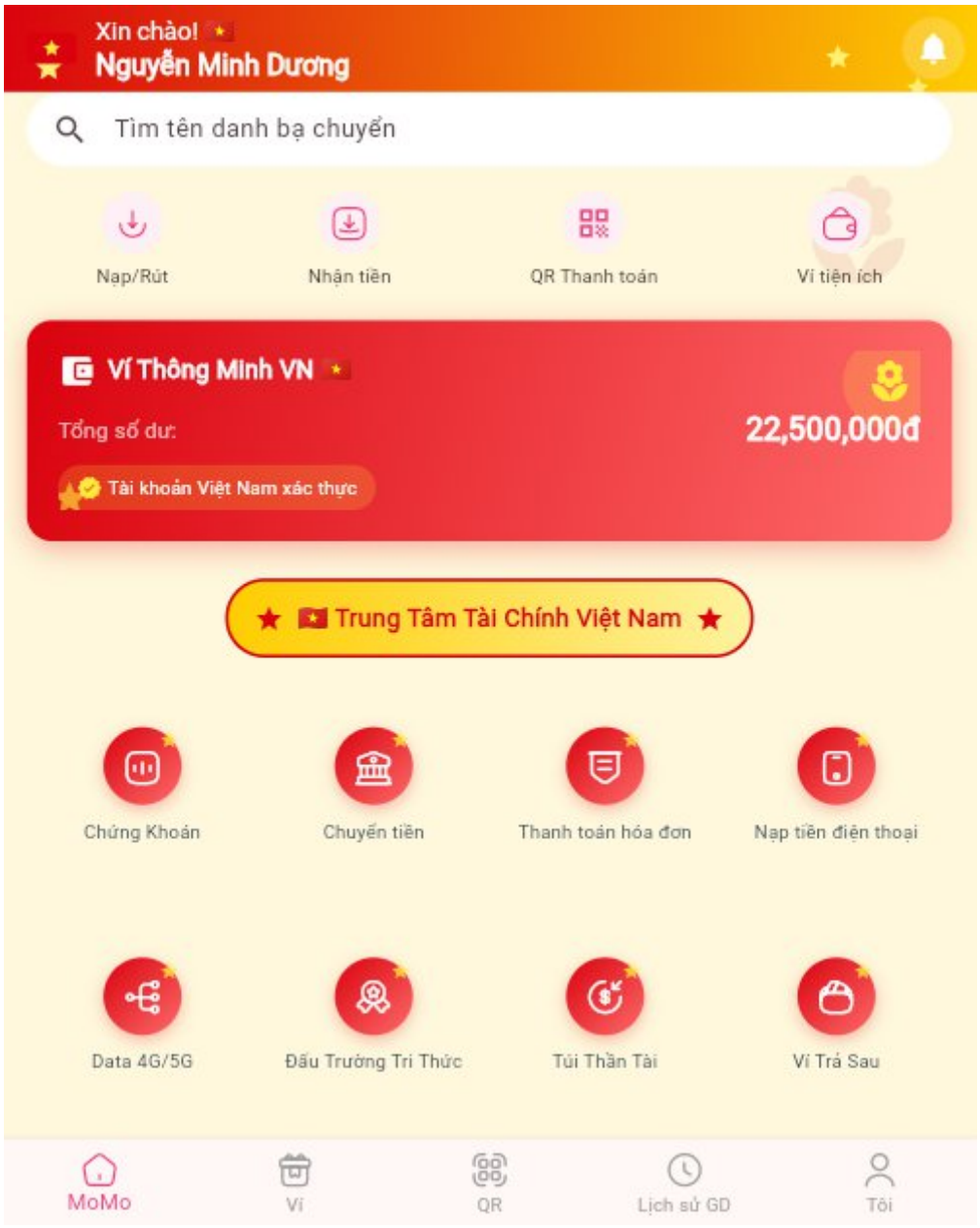
Ứng dụng bao gồm các màn hình cốt lõi phục vụ quản lý tài chính cá nhân:

- **Màn hình Đăng nhập/Đăng ký:** Cho phép người dùng xác thực tài khoản.



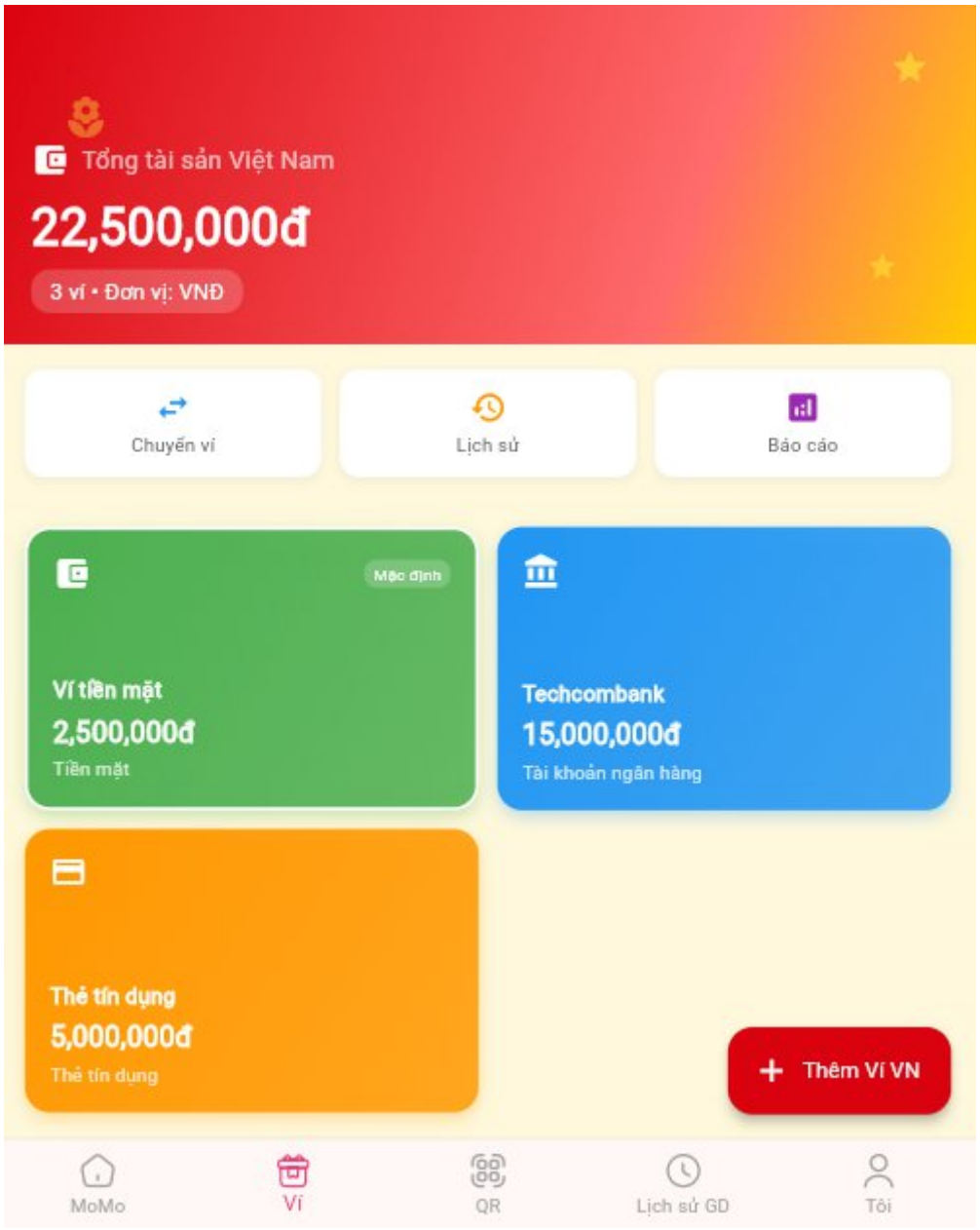
Hình 36: Giao diện đăng nhập

- **Trang chủ (Home):** Tổng quan số dư, ví, và giao dịch gần đây.



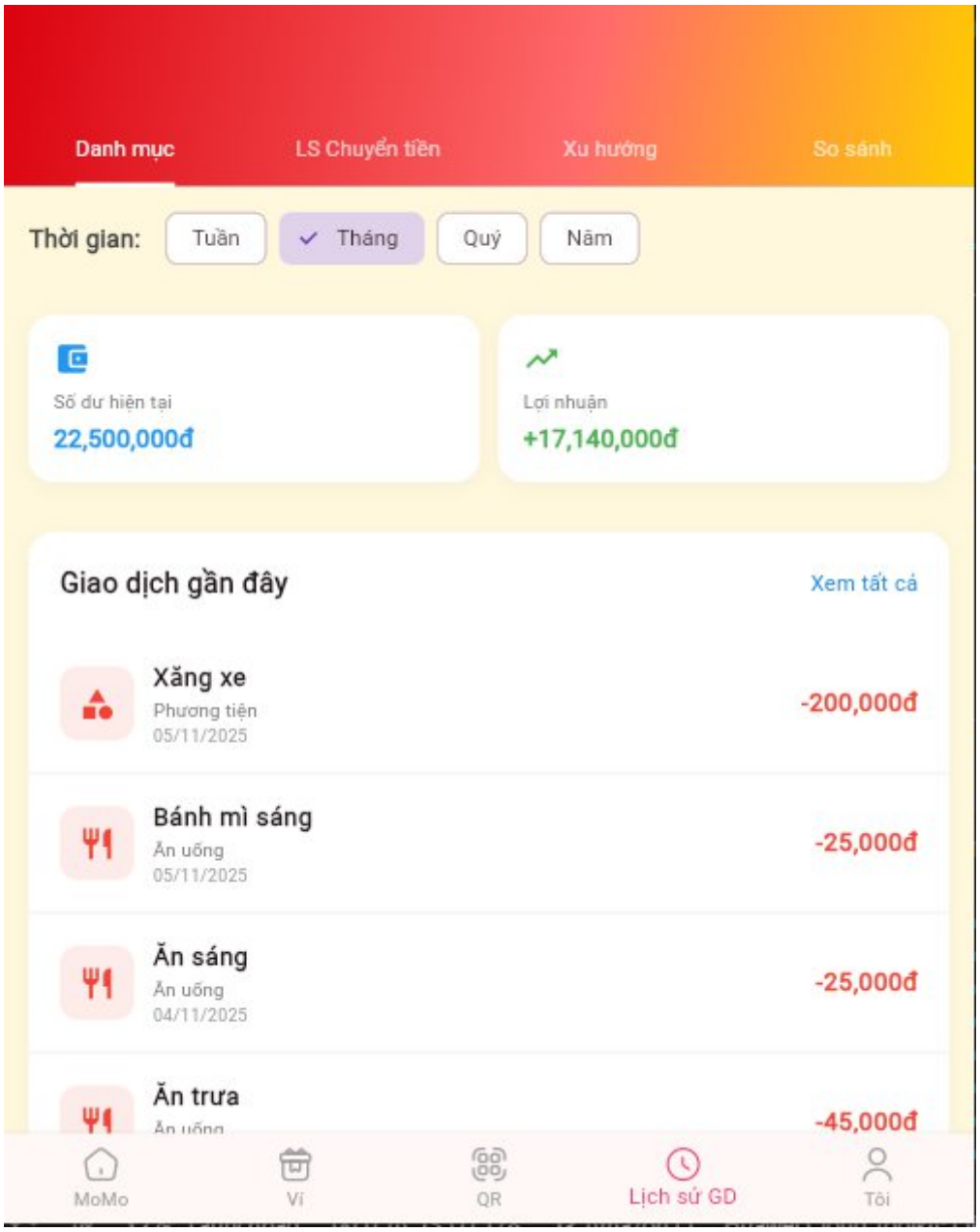
Hình 37: Giao diện chính của SmartWallet

- **Ví điện tử (Wallet):** Quản lý nhiều loại ví và theo dõi tổng số dư.



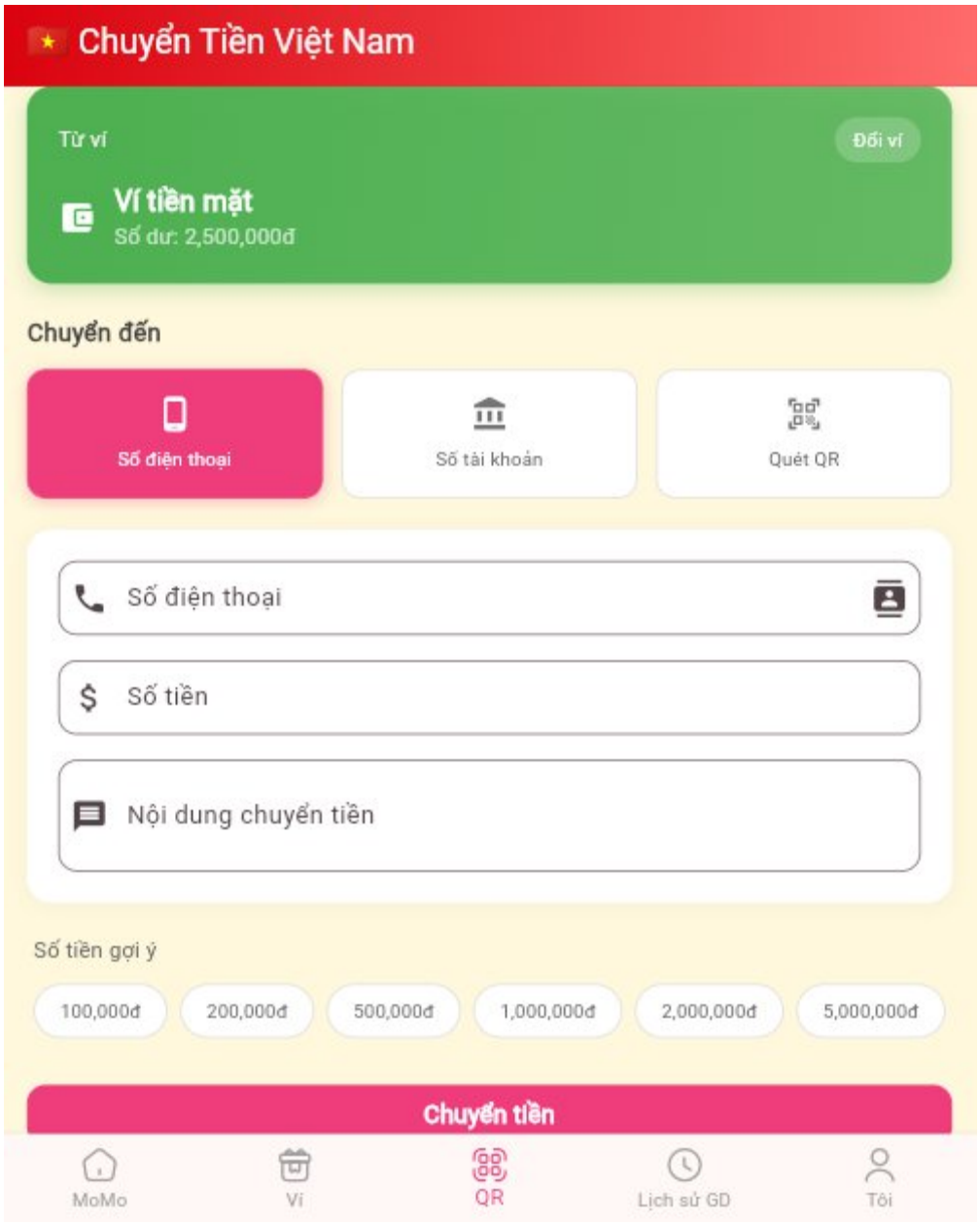
Hình 38: Giao diện Ví điện tử

- **Chi tiêu (Expenses):** Ghi chép thu – chi và phân loại danh mục.



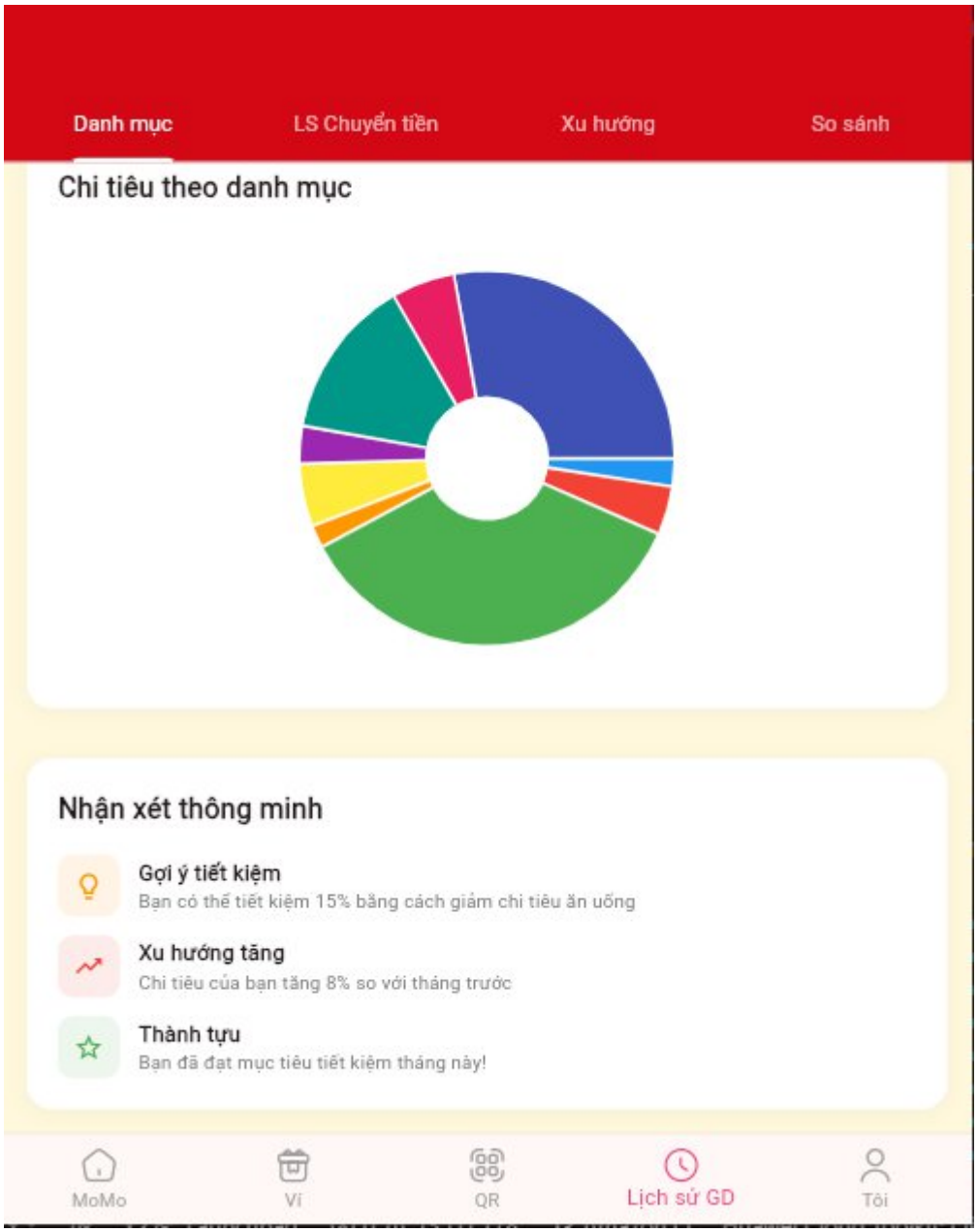
Hình 39: Giao diện Quản lý chi tiêu

- **Giao dịch (Transaction):** Màn hình giao dịch



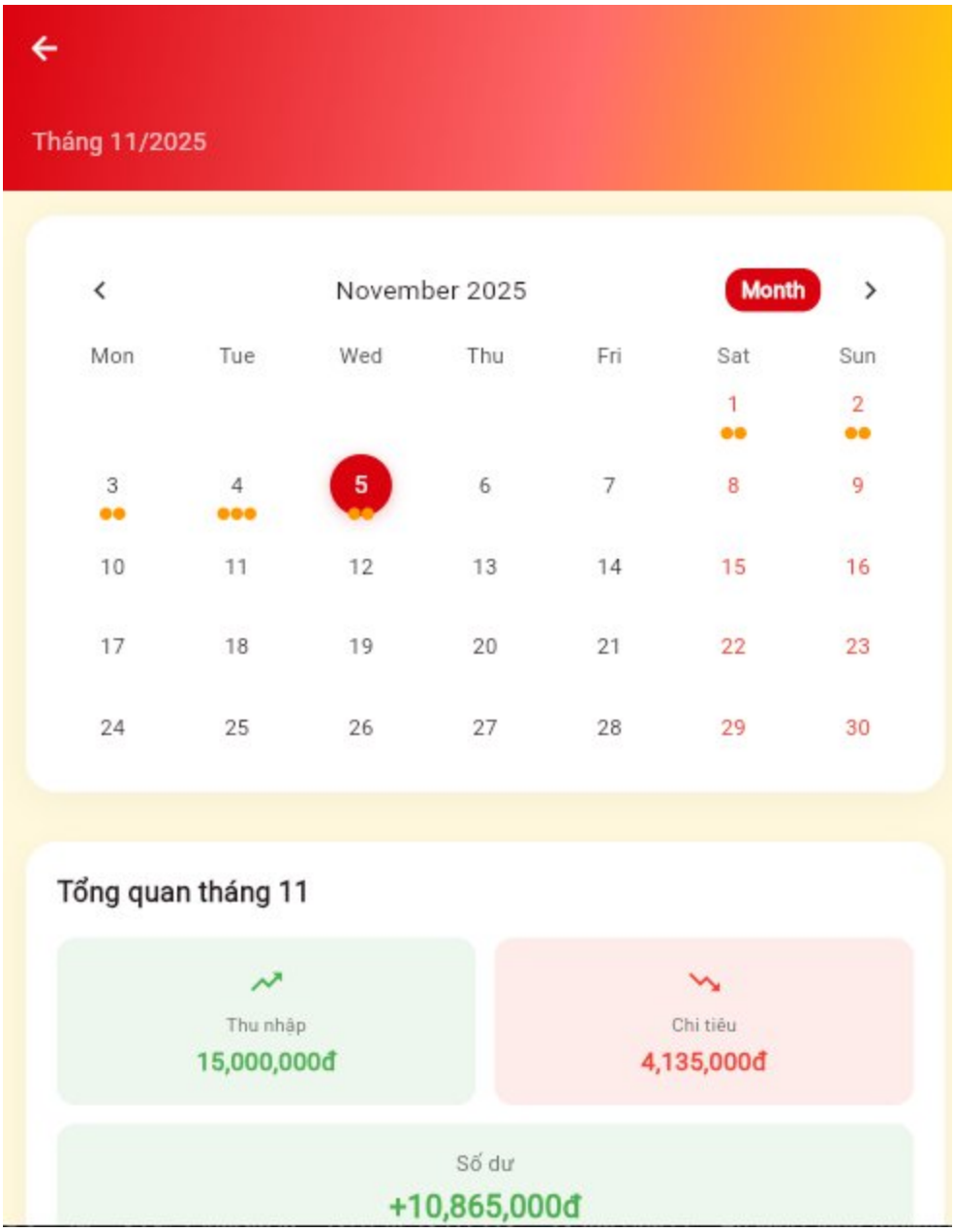
Hình 40: Giao diện Giao dịch

- **Thống kê (Statistics):** Hiển thị biểu đồ và báo cáo chi tiêu trực quan.



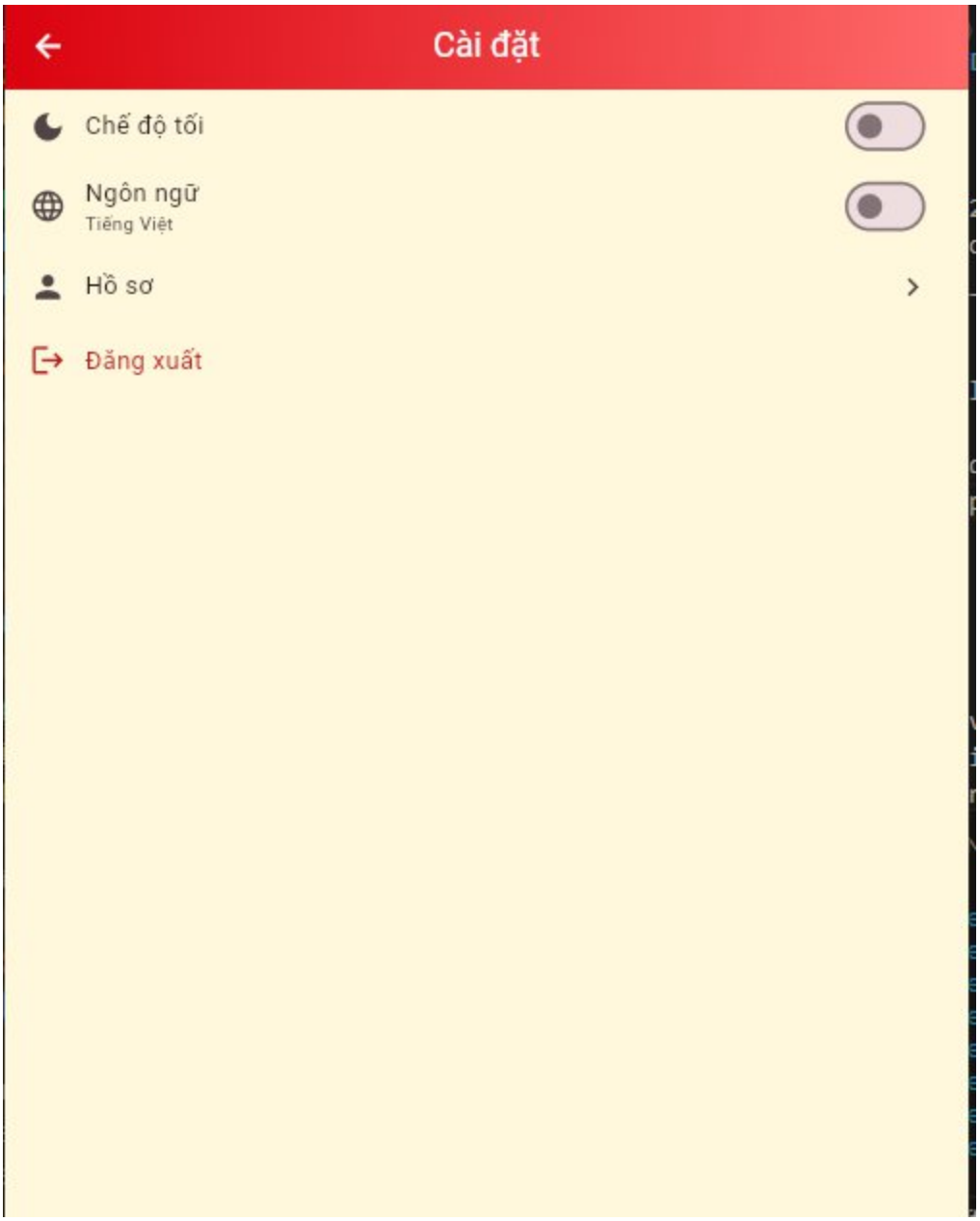
Hình 41: Giao diện thống kê và báo cáo

- **Lịch tài chính (Calendar):** Theo dõi giao dịch theo ngày và tháng.



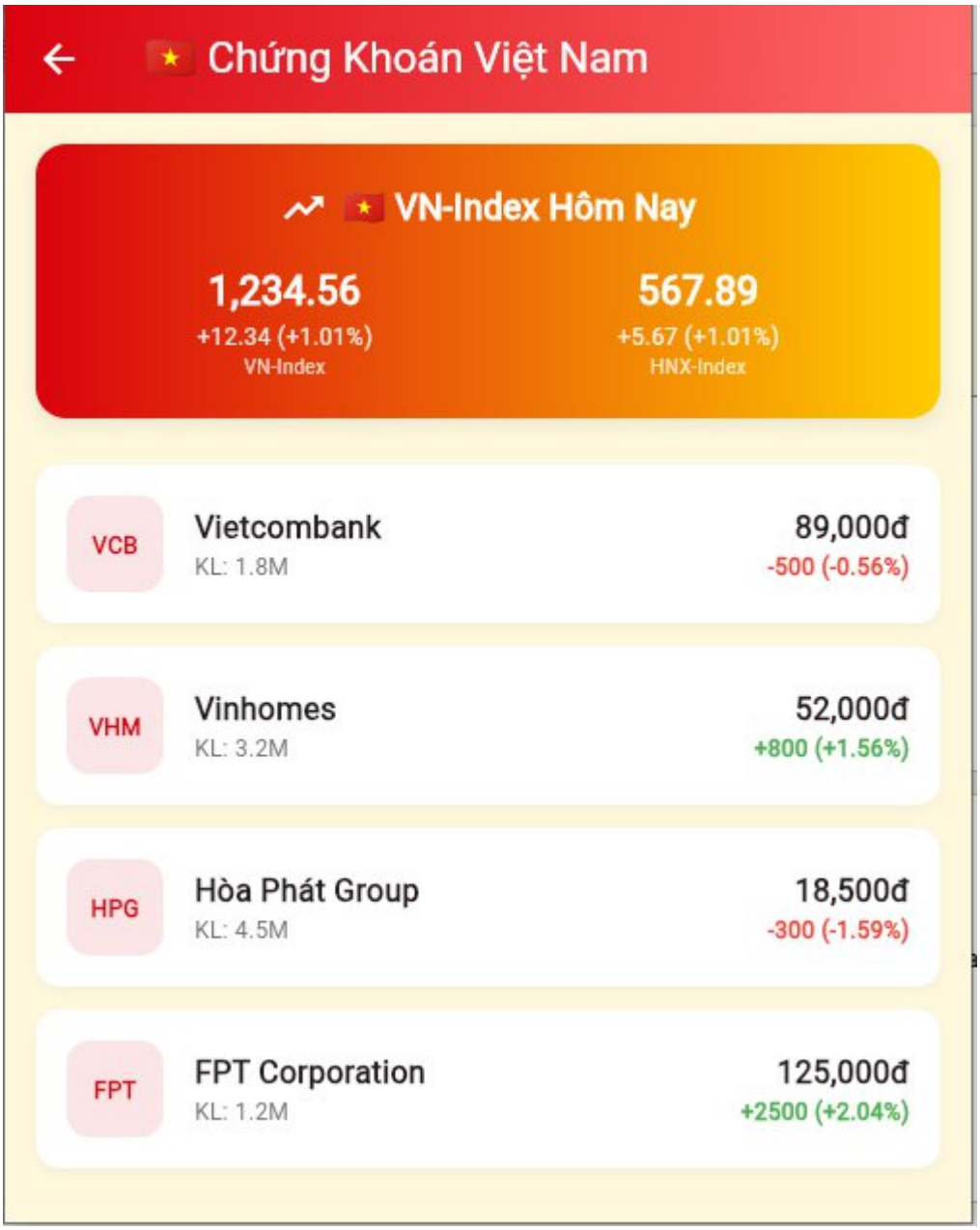
Hình 42: Giao diện lịch tài chính

- **Cài đặt (Settings):** Tùy chỉnh ngôn ngữ, bảo mật, và thông tin người dùng.

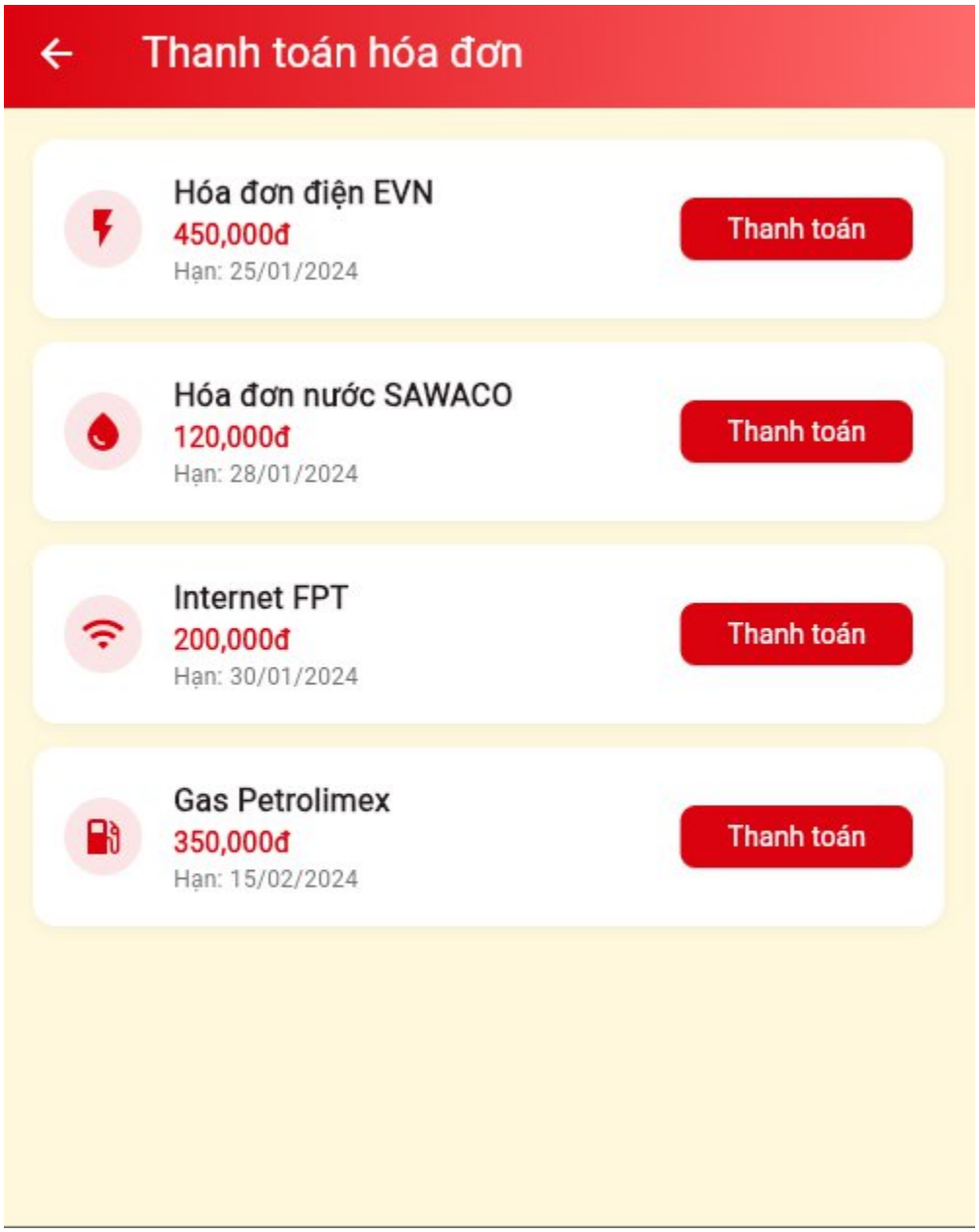


Hình 43: Giao diện cài đặt

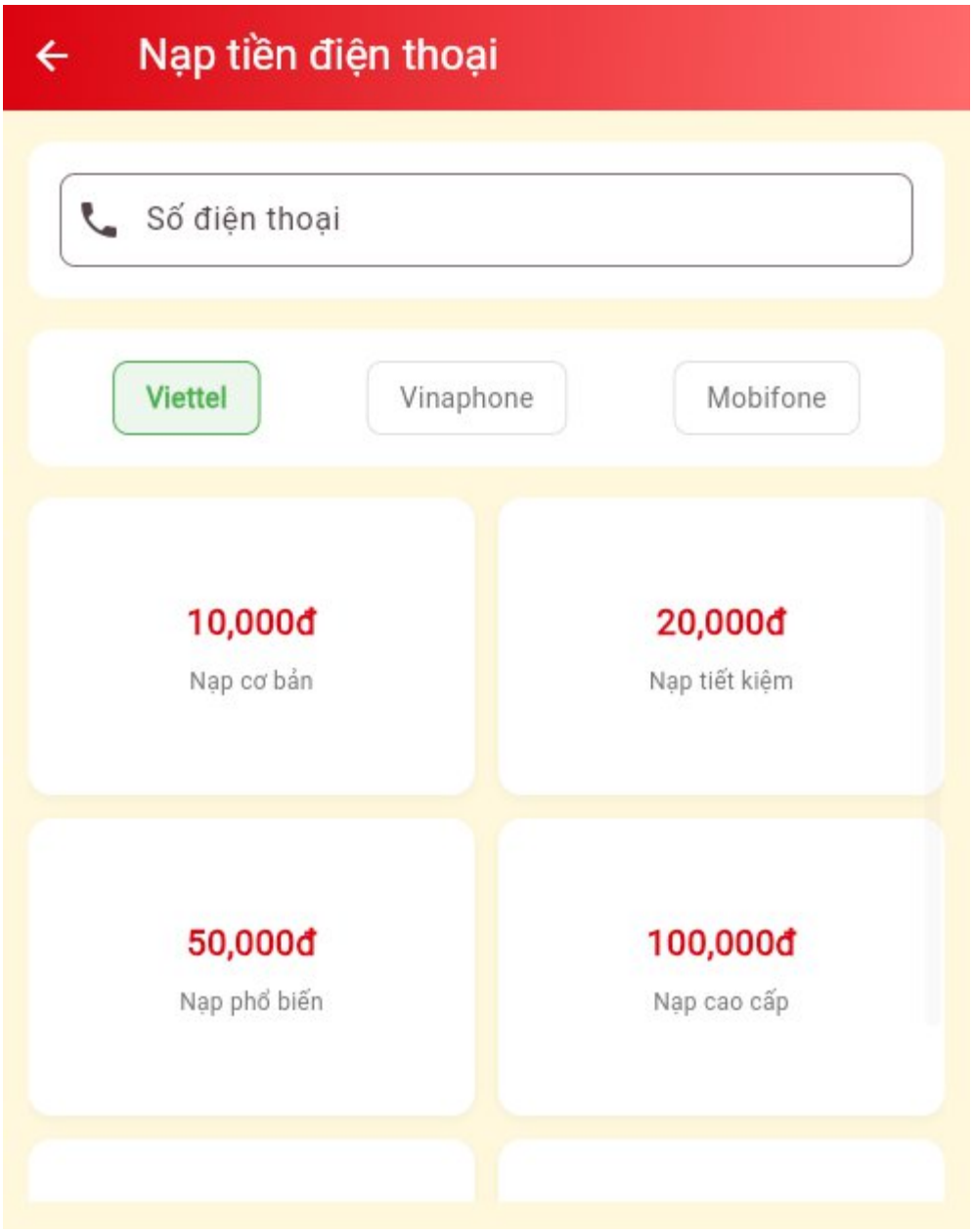
- **Giao diện các chức năng phụ :** Giao diện chứng khoán, thanh toán hóa đơn, nạp tiền điện thoại, đăng ký data…



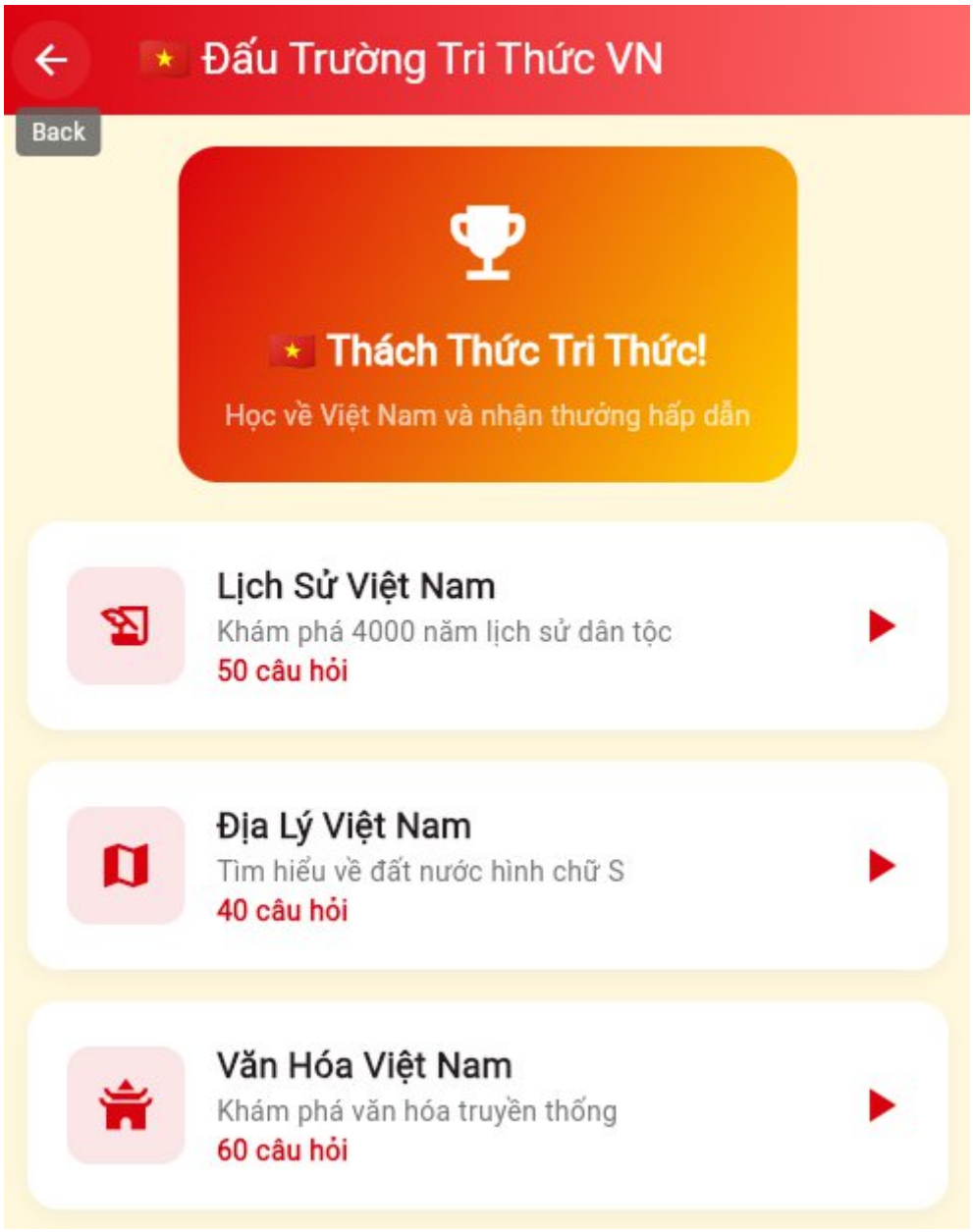
Hình 44: Giao diện chứng khoán



Hình 45: Giao diện thanh toán hóa đơn

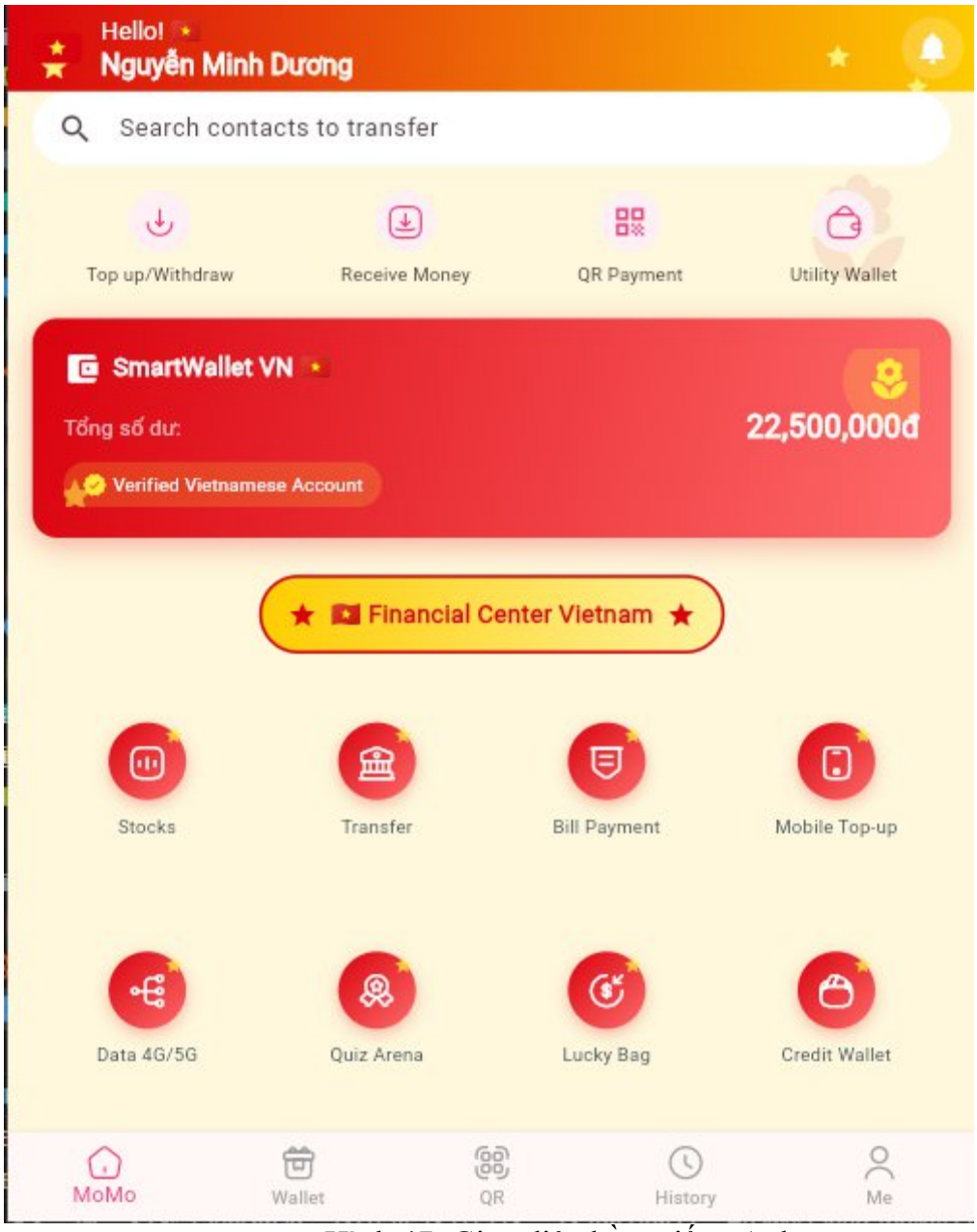


Hình 46: Giao diện nạp tiền điện thoại



Hình 47: Giao diện trò chơi giải trí

- **Giao diện khi dùng chức năng Song Ngữ Anh- Việt**



- Hình 47: Giao diện bằng tiếng Anh

3 Animations và Transitions

Ứng dụng sử dụng các hiệu ứng chuyển cảnh **mượt mà và tinh tế** nhằm tăng trải nghiệm người dùng. Các animation như **fade-in, slide, hero animation và card elevation** được áp dụng hợp lý trong quá trình chuyển giữa các màn hình và khi tương tác với dữ liệu. Các hiệu ứng này không chỉ tạo cảm giác tự nhiên, mà còn nhấn mạnh các hành động quan trọng như **chuyển tiền, thêm chi tiêu, hay hiển thị báo cáo**, giúp giao diện **sống động và chuyên nghiệp hơn**.

Chương 7: Quản lý dữ liệu

1 Chiến lược quản lý dữ liệu

1.1 Tổng quan kiến trúc dữ liệu

Ứng dụng SmartWallet áp dụng chiến lược lưu trữ dữ liệu cục bộ (Local Storage Strategy) nhằm đáp ứng yêu cầu của một dự án học tập, trong đó không yêu cầu hệ thống backend phức tạp. Cụ thể, ứng dụng sử dụng SharedPreferences làm lớp lưu trữ chính cho các dữ liệu đơn giản, kết hợp với tuần tự hóa JSON để xử lý các đối tượng phức tạp hơn. Đối với các thông tin nhạy cảm (nếu có), ứng dụng dự kiến tích hợp cơ chế lưu trữ mã hóa.

Kiến trúc dữ liệu được thiết kế theo mô hình phân tầng ba lớp, bao gồm:

- Lớp lưu trữ cục bộ (Local Storage Layer): Chịu trách nhiệm ghi/đọc dữ liệu từ thiết bị.
- Lớp logic nghiệp vụ (Business Logic Layer): Thực hiện các thao tác CRUD, kiểm tra tính hợp lệ và chuyển đổi dữ liệu.
- Lớp giao diện (Presentation Layer): Quản lý trạng thái và hiển thị dữ liệu thông qua các thành phần UI.

1.2 Lữu trữ dữ liệu cục bộ

Toàn bộ dữ liệu của ứng dụng được lưu trữ cục bộ trên thiết bị người dùng thông qua lớp dịch vụ trung tâm DatabaseService. Lớp này cung cấp các phương thức đồng bộ hóa dữ liệu giữa bộ nhớ ứng dụng và bộ nhớ thiết bị.

Ví dụ, phương thức lưu và truy xuất danh sách ví được triển khai như sau:

```
class DatabaseService {
  static const String _walletsKey = 'wallets';
  static const String _expensesKey = 'expenses';
  static const String _transactionsKey = 'transactions';
  static const String _userKey = 'current_user';

  // Lưu trữ danh sách ví
  static Future<void> saveWallets(List<Wallet> wallets) async {
    final prefs = await SharedPreferences.getInstance();
    final walletsJson = wallets.map((w) => w.toMap()).toList();
```

```
  await prefs.setString(_walletsKey, jsonEncode(walletsJson));
 }

 // Lấy danh sách ví
 static Future<List<Wallet>> getWallets() async {
  final prefs = await SharedPreferences.getInstance();
  final walletsString = prefs.getString(_walletsKey);
  if (walletsString == null) return [];

  final List<dynamic> walletsJson = jsonDecode(walletsString);
  return walletsJson.map((json) => Wallet.fromMap(json)).toList();
 }
}
```

Hình 48: Code minh họa về lưu trữ cục bộ

1.3 Chiến lược dữ liệu mẫu (Mock data)

Để hỗ trợ việc demo và kiểm thử tính năng mà không cần kết nối đến máy chủ, SmartWallet tích hợp cơ chế tạo dữ liệu mẫu tự động ngay khi khởi động lần đầu. Dữ liệu mẫu bao gồm các ví, giao dịch và chi tiêu điển hình, giúp người dùng nhanh chóng làm quen với giao diện và chức năng.

```
class DatabaseService {
 static Future<void> initializeSampleData() async {
  // Tạo 3 ví mẫu
  final sampleWallets = [
   Wallet(
    id: 'wallet_1',
    name: 'Tiền mặt',
    type: 'cash',
    balance: 5000000,
    color: '#2196F3',
    isDefault: true,
   ),
   Wallet(
    id: 'wallet_2',
    name: 'Vietcombank',
    type: 'bank',
    balance: 15000000,
    color: '#4CAF50',
    bankName: 'Vietcombank',
    accountNumber: '1234567890',
   ),
```

```
    Wallet(
      id: 'wallet_3',
      name: 'Thẻ tín dụng',
      type: 'credit',
      balance: 2000000,
      color: '#FF9800',
      bankName: 'Techcombank',
      cardNumber: '**** **** **** 1234',
    ),
  ];

  await saveWallets(sampleWallets);
 }
}
```

Hình 49: Code minh họa về lưu trữ dữ liệu mẫu

Lợi ích của việc sử dụng Mock Data:

- Cho phép demo tính năng ngay lập tức mà không cần backend.
- Đảm bảo tính nhất quán trong quá trình kiểm thử.
- Dễ dàng khôi phục hoặc tái tạo dữ liệu.
- Phù hợp với mục tiêu học tập và phát triển nguyên mẫu.

2 Mô hình dữ liệu (Data model)

Ứng dụng định nghĩa các lớp mô hình dữ liệu để biểu diễn các thực thể chính: User, Wallet, Transaction, và Expense. Mỗi lớp đều hỗ trợ tuần tự hóa sang và từ định dạng JSON thông qua các phương thức toMap() và fromMap()

2.1 Mô hình người dùng (User Model)

```
class User {
 final String id;
 final String name;
 final String email;
 final String? phone;
 final DateTime createdAt;

 // Constructor, fromMap(), toMap()...
}
```

Hình 50: Code minh họa về mô hình người dùng

2.2 Mô hình Ví (Wallet Model)

Lớp Wallet đại diện cho các loại ví như tiền mặt, tài khoản ngân hàng hoặc thẻ tín dụng. Mỗi ví có các thuộc tính như loại ví, số dư, màu sắc, biểu tượng và thông tin ngân hàng (nếu có).

```
class Wallet {
  String id;
  String name;
  String type; // 'cash', 'bank', 'credit', 'savings'
  double balance;
  String color; // Hex color code
  String icon;
  bool isDefault;
  String? bankName;
  String? accountNumber;
  String? cardNumber;
  DateTime createdAt;

  Wallet({
    required this.id,
    required this.name,
    required this.type,
    required this.balance,
    required this.color,
    required this.icon,
    this.isDefault = false,
    this.bankName,
    this.accountNumber,
    this.cardNumber,
    DateTime? createdAt,
  }) : createdAt = createdAt ?? DateTime.now();

  // JSON Serialization
  factory Wallet.fromMap(Map<String, dynamic> map) {
    return Wallet(
      id: map['id'] ?? '',
      name: map['name'] ?? '',
      type: map['type'] ?? 'cash',
      balance: (map['balance'] as num).toDouble(),
      color: map['color'] ?? '#2196F3',
      icon: map['icon'] ?? 'wallet',
      isDefault: map['isDefault'] ?? false,
      bankName: map['bankName'],
      accountNumber: map['accountNumber'],
```

```
    cardNumber: map['cardNumber'],
    createdAt: DateTime.parse(map['createdAt']),
  );
 }

 Map<String, dynamic> toMap() {
  return {
   'id': id,
   'name': name,
   'type': type,
   'balance': balance,
   'color': color,
   'icon': icon,
   'isDefault': isDefault,
   'bankName': bankName,
   'accountNumber': accountNumber,
   'cardNumber': cardNumber,
   'createdAt': createdAt.toIso8601String(),
  };
 }
}
```

Hình 51: Code minh họa về mô hình Ví

2.3 Mô hình Giao dịch (Transaction Model)

```
class Transaction {
 final String id;
 final String type; // 'transfer', 'payment', 'topup'
 final double amount;
 final String fromWalletId;
 final String? toWalletId;
 final String description;
 final String? recipientName;
 final DateTime createdAt;
 final String status; // 'completed', 'pending', 'failed'

 Transaction({
  required this.id,
  required this.type,
  required this.amount,
  required this.fromWalletId,
  this.toWalletId,
  required this.description,
```

```
    this.recipientName,
    required this.createdAt,
    this.status = 'completed',
  });

  factory Transaction.fromMap(Map<String, dynamic> map) {
    return Transaction(
      id: map['id'] ?? '',
      type: map['type'] ?? 'transfer',
      amount: (map['amount'] as num).toDouble(),
      fromWalletId: map['fromWalletId'] ?? '',
      toWalletId: map['toWalletId'],
      description: map['description'] ?? '',
      recipientName: map['recipientName'],
      createdAt: DateTime.parse(map['createdAt']),
      status: map['status'] ?? 'completed',
    );
  }
}
```

Hình 52: Code minh họa về mô hình Giao dịch

2.4 Mô hình Chi tiêu (Expense Model)

Lớp Expense quản lý các khoản thu/chi cá nhân, bao gồm tiêu đề, số tiền, ngày ghi nhận, danh mục và mô tả. Lớp này cũng tích hợp logic kiểm tra tính hợp lệ và ánh xạ biểu tượng theo danh mục.

```
class Expense {
  String id;
  String title;
  double amount;
  DateTime date;
  String category;
  String description;
  String type; // 'Chi tiêu', 'Thu nhập'

  Expense({
    required this.id,
    required this.title,
    required this.amount,
    required this.date,
    required this.category,
    required this.description,
    this.type = 'Chi tiêu',
  });
```

```
  // Validation logic
  bool isValid() {
    return title.isNotEmpty &&
           amount != 0 &&
           category.isNotEmpty;
  }

  // Category mapping
  static const Map<String, String> categoryIcons = {
    'Ăn uống': 'restaurant',
    'Di chuyển': 'directions_car',
    'Nhà cửa': 'home',
    'Sức khỏe': 'local_hospital',
    'Giải trí': 'movie',
    'Quần áo': 'shopping_bag',
    'Giáo dục': 'school',
    'Du lịch': 'flight',
    'Lương': 'work',
    'Khác': 'category',
  };
}
```

Hình 52: Code minh họa về mô hình Chi tiêu

Chương 8: Các kịch bản kiểm thử

1 Tổng quan chiến lược kiểm thử

- Để đảm bảo chất lượng phần mềm và độ tin cậy của ứng dụng SmartWallet, nhóm phát triển áp dụng chiến lược kiểm thử theo mô hình Kim tự tháp Kiểm thử (Testing Pyramid)—một phương pháp luận được khuyến nghị trong phát triển phần mềm hiện đại. Cụ thể:
- Kiểm thử đơn vị (Unit Tests): Chiếm 70% tổng lượng kiểm thử, tập trung vào các lớp mô hình (User, Wallet, Expense...), dịch vụ (DatabaseService), và các hàm tiện ích (Utils). Đây là lớp kiểm thử nhanh, ổn định và dễ bảo trì nhất.
- Kiểm thử giao diện (Widget Tests): Chiếm 20%, kiểm tra hành vi và trạng thái của các thành phần UI như nút bấm, form nhập liệu, danh sách ví, biểu đồ thống kê.
- Kiểm thử tích hợp (Integration/End-to-End Tests): Chiếm 10%, mô phỏng luồng người dùng thực tế (ví dụ: đăng nhập → tạo ví → chuyển tiền → xem thống kê).
- Chiến lược này giúp phát hiện lỗi sớm, giảm chi phí sửa chữa và nâng cao khả năng bảo trì mã nguồn.

2 Các kịch bản kiểm thử chính

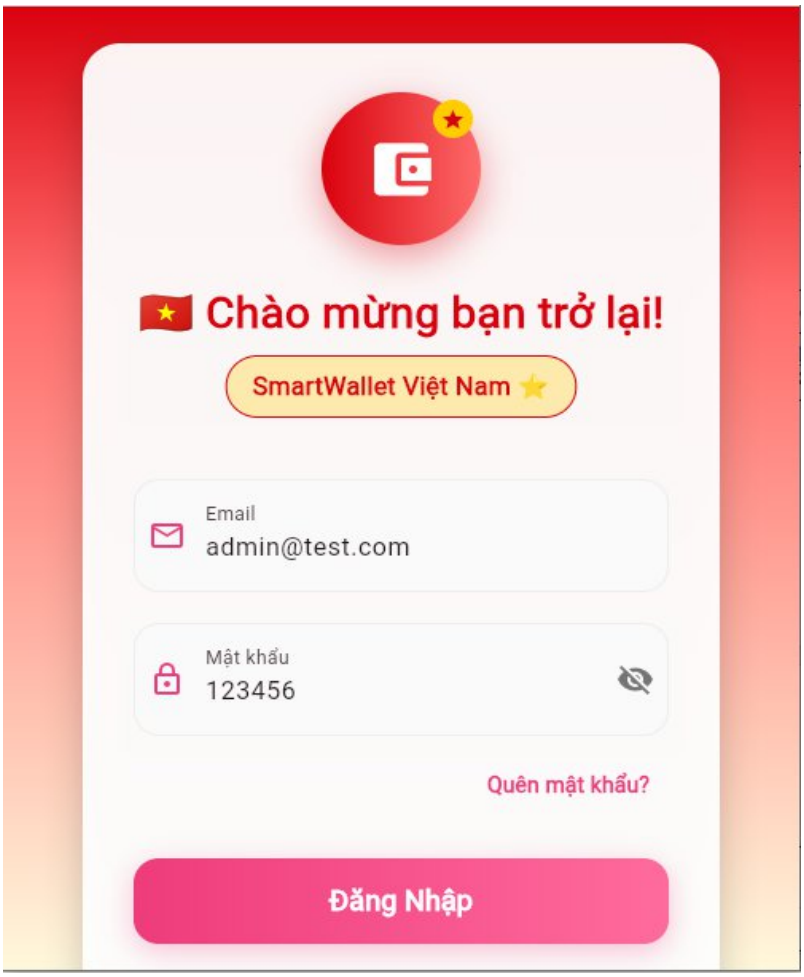
- **Kịch bản 1: Xác thực người dùng**

  - Người dùng nhập email và mật khẩu hợp lệ (ví dụ: admin@test.com / 123456) → hệ thống đăng nhập thành công và chuyển đến màn hình chính.
  - Khi email không đúng định dạng (như invalid-email, test@, @domain.com) → hệ thống hiển thị thông báo "Email không hợp lệ" và không cho phép tiếp tục.
  - Khi email đúng nhưng mật khẩu sai → hệ thống hiển thị "Mật khẩu không chính xác" và giữ nguyên màn hình đăng nhập.
  - **Mục tiêu:** đảm bảo tính bảo mật và phản hồi lỗi thân thiện cho người dùng.
  - **Code kiểm thử kịch bản 1 :**

```dart
// test/auth/login_success_test.dart
testWidgets('TC001 - Successful login with valid credentials', (tester) async {
  // Arrange
  await tester.pumpWidget(createTestApp());

  // Act
  await tester.enterText(find.byKey(Key('email_field')), 'admin@smartwallet.com');
  await tester.enterText(find.byKey(Key('password_field')), '123456');
  await tester.tap(find.byKey(Key('login_button')));
  await tester.pumpAndSettle();

  // Assert
  expect(find.text('Tổng Quan'), findsOneWidget);
  expect(find.text('Ví của tôi'), findsOneWidget);
  expect(find.byKey(Key('home_screen')), findsOneWidget);
});
```

Hình 53: Code minh họa về kịch bản kiểm thử 1

  - **Kết quả :**



Hình 54: Hình minh họa kết quả kịch bản kiểm thử 1

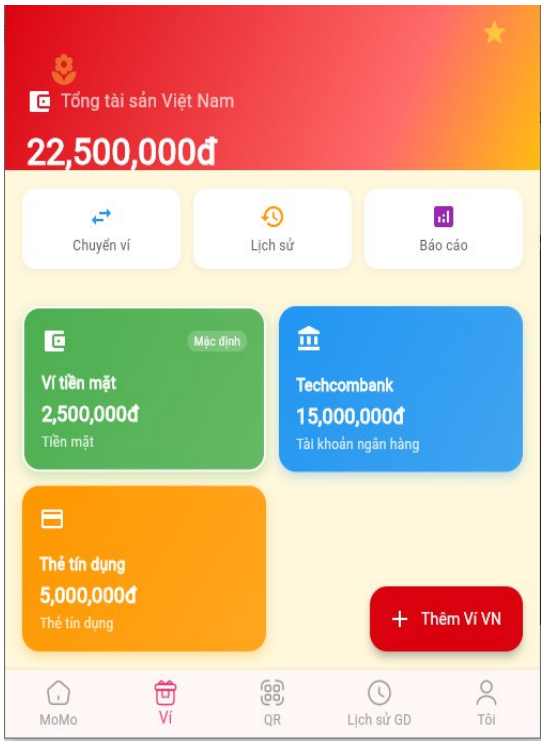
- **Kịch bản 2: Quản lý ví điện tử**

  - Người dùng tạo ví mới với tên, loại (tiền mặt/ngân hàng/thẻ), và số dư → hệ thống lưu và hiển thị ví với định dạng số dư rõ ràng (ví dụ: "500,000đ").
  - Nếu để trống tên ví → hệ thống báo lỗi "Tên ví không được để trống" và vô hiệu hóa nút lưu.
  - Chức năng chỉnh sửa cho phép đổi tên hoặc cập nhật thông tin ví → thay đổi được phản ánh ngay lập tức trên giao diện.
  - Xóa ví sau khi xác nhận → ví biến mất khỏi danh sách và dữ liệu được đồng bộ xóa trong bộ nhớ cục bộ.
  - **Mục tiêu**: kiểm tra tính toàn vẹn và nhất quán của dữ liệu ví.
  - **Code kiểm thử kịch bản 2:**

```
testWidgets('TC011 - Create new wallet successfully', (tester) async {
  await loginHelper(tester);

  // Navigate to wallet management
  await tester.tap(find.text('Quản lý ví'));
  await tester.pumpAndSettle();

  // Add new wallet
  await tester.tap(find.byIcon(Icons.add));
  await tester.pumpAndSettle();

  // Fill wallet information
  await tester.enterText(find.byKey(Key('wallet_name')), 'Ví Test');
  await tester.tap(find.text('Tiền mặt'));
  await tester.enterText(find.byKey(Key('initial_balance')), '500000');
  await tester.tap(find.text('Lưu'));
  await tester.pumpAndSettle();

  // Verify wallet created
  expect(find.text('Ví Test'), findsOneWidget);
  expect(find.text('500,000đ'), findsOneWidget);
});
```

Hình 55**:** Code minh họa về kịch bản kiểm thử 2

- **Kết quả:**



Hình 56: Hình minh họa kết quả kịch bản kiểm thử 2

- **Kịch bản 3: Chuyển tiền giữa các ví**

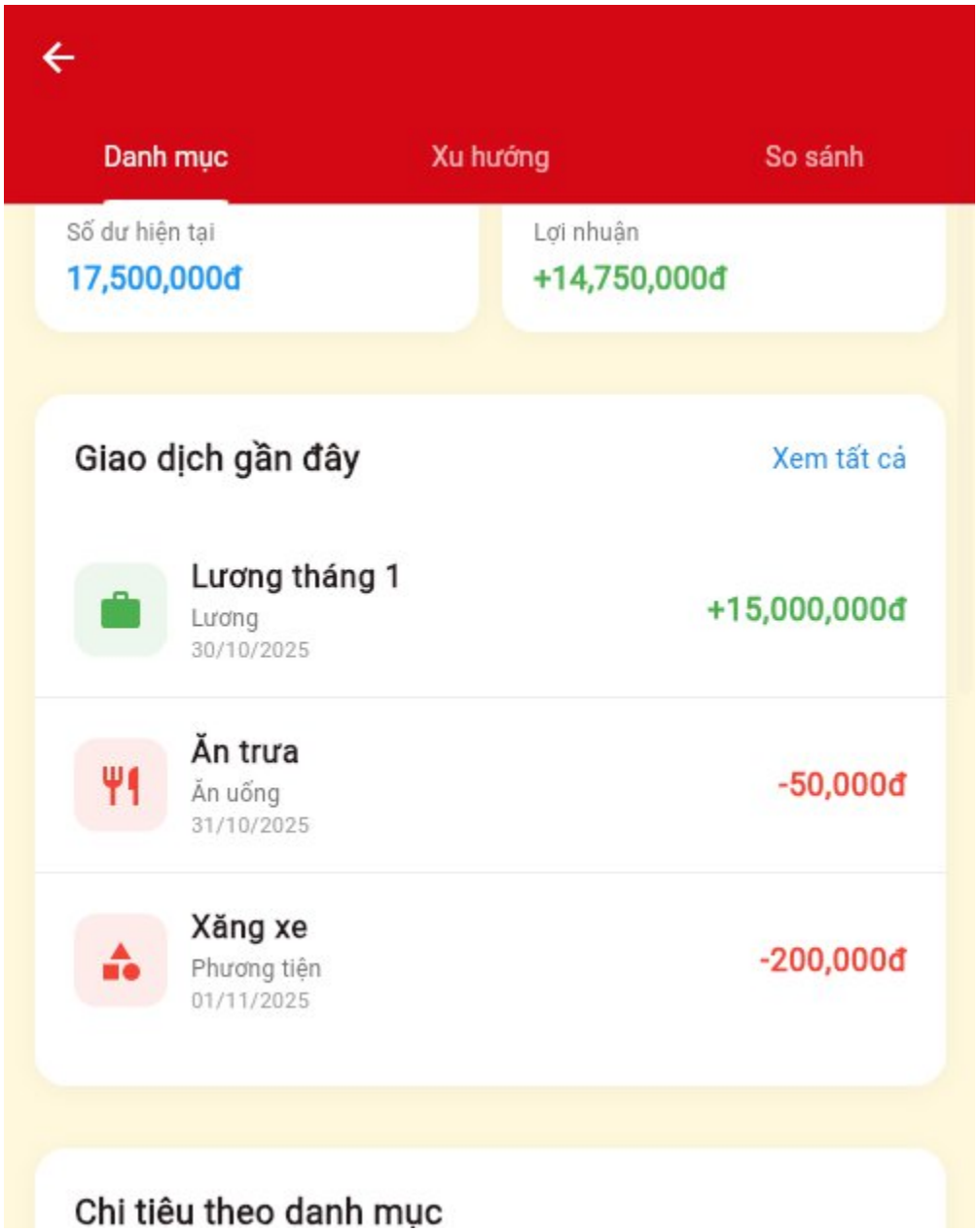
  - Người dùng chọn ví nguồn, ví đích, nhập số tiền hợp lệ (>0) và mô tả → giao dịch thành công, số dư hai ví được cập nhật.
  - Nếu số tiền vượt quá số dư khả dụng → hệ thống hiển thị "Số dư không đủ" và hủy giao dịch.
  - Nếu nhập số tiền không hợp lệ (0, âm, chữ, rỗng) → hệ thống báo "Số tiền không hợp lệ".
  - **Mục tiêu**: đảm bảo logic tài chính được kiểm soát chặt chẽ, tránh sai sót trong luồng tiền.
  - **Code kiểm thử kịch bản 3:**

```
testWidgets('TC021 - Successful money transfer', (tester) async {
  await loginHelper(tester);

  // Navigate to transfer
  await tester.tap(find.text('Chuyển tiền'));
  await tester.pumpAndSettle();

  // Select source wallet
  await tester.tap(find.byKey(Key('source_wallet_dropdown')));
  await tester.tap(find.text('Ví Tiền Mặt'));
```

```
  // Select destination wallet
  await tester.tap(find.byKey(Key('dest_wallet_dropdown')));
  await tester.tap(find.text('Ví Ngân Hàng'));

  // Enter amount and description
  await tester.enterText(find.byKey(Key('amount_field')), '100000');
  await tester.enterText(find.byKey(Key('description_field')), 'Chuyển tiền test');

  // Confirm transfer
  await tester.tap(find.text('Xác Nhận Chuyển'));
  await tester.pumpAndSettle();

  expect(find.text('Chuyển tiền thành công'), findsOneWidget);
});
```

Hình 57: Code minh họa về kịch bản kiểm thử 3

- **Kết quả:**



Hình 58: Hình kết quả kịch bản kiểm thử 3

- **Kịch bản 4: Quản lý thu chi**

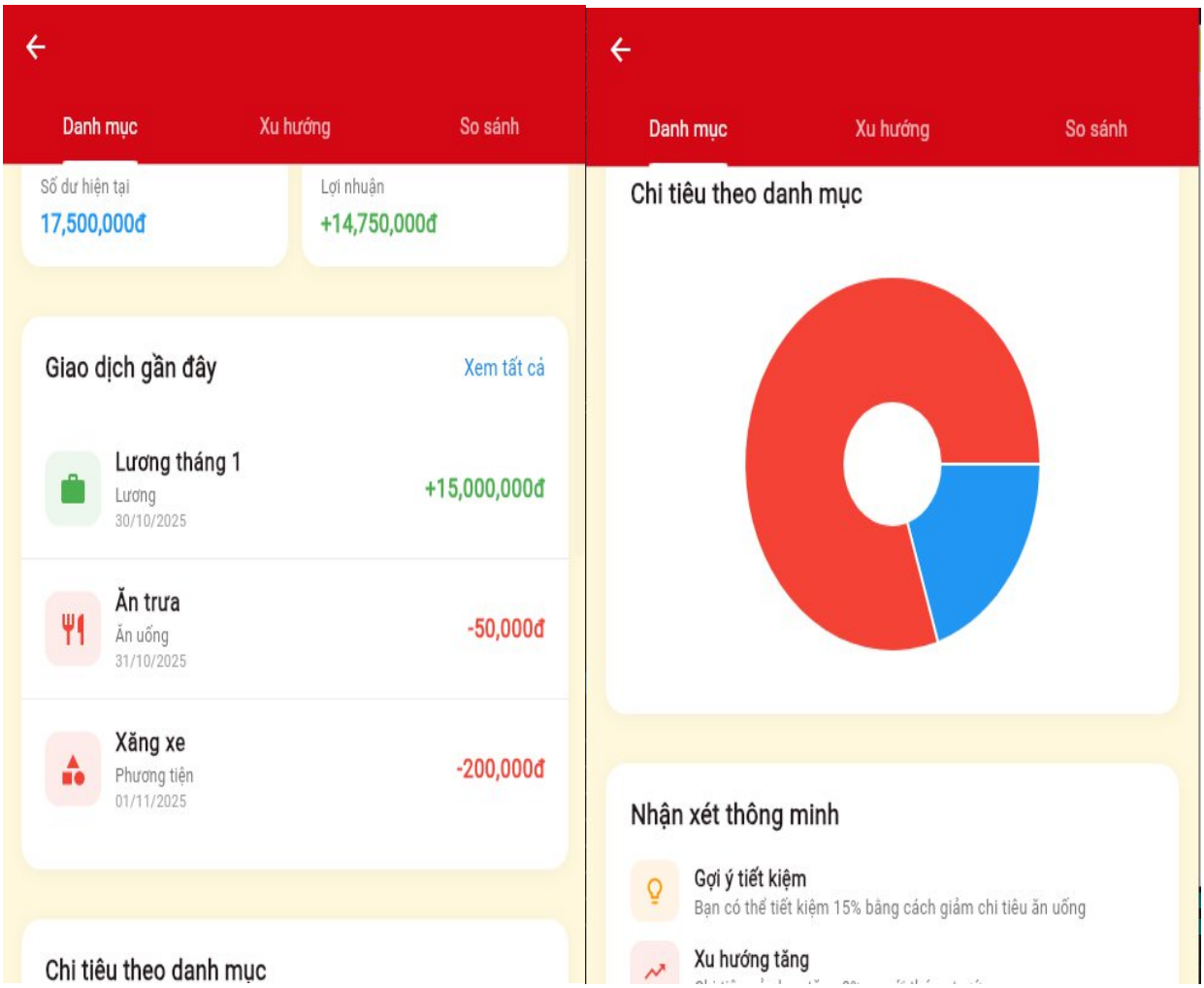
  - Thêm chi tiêu: nhập tiêu đề, số tiền, chọn danh mục (ví dụ: "Ăn uống") → hệ thống hiển thị dưới dạng "-50,000đ".
  - Thêm thu nhập: chọn tab "Thu nhập", nhập thông tin → hiển thị dưới dạng "+15,000,000đ".
  - Chỉnh sửa hoặc xóa khoản thu/chi → thay đổi hoặc xóa được áp dụng ngay, dữ liệu được lưu tự động.

- Mục tiêu: xác minh khả năng ghi nhận, phân loại và quản lý dòng tiền cá nhân một cách chính xác.
- **Code kiểm thử kịch bản 4:**

```
testWidgets('TC031 - Add expense successfully', (tester) async {
  await loginHelper(tester);

  await tester.tap(find.text('Chi tiêu'));
  await tester.pumpAndSettle();

  await tester.tap(find.byIcon(Icons.add));
  await tester.pumpAndSettle();

  // Fill expense details
  await tester.enterText(find.byKey(Key('expense_title')), 'Ăn trưa');
  await tester.enterText(find.byKey(Key('expense_amount')), '50000');

  // Select category
  await tester.tap(find.byKey(Key('category_dropdown')));
  await tester.tap(find.text('Ăn uống'));

  await tester.tap(find.text('Lưu'));
  await tester.pumpAndSettle();

  expect(find.text('Ăn trưa'), findsOneWidget);
  expect(find.text('-50,000đ'), findsOneWidget);
});
```

Hình 59: Code minh họa kịch bản kiểm thử 4

- **Kết quả:**



Hình 60: Hình kết quả kịch bản kiểm thử 4

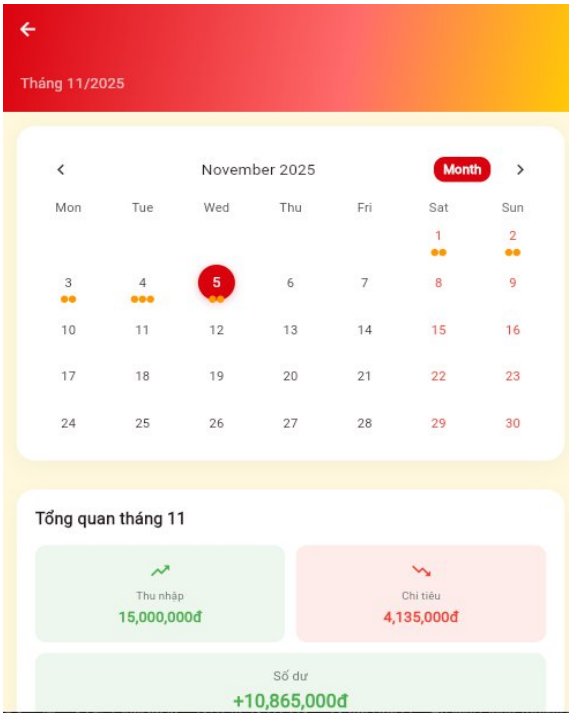
- **Kịch bản 5: Lịch và thống kê tài chính**

  - Truy cập tab "Lịch" → xem các ngày có giao dịch được đánh dấu; nhấn vào ngày cụ thể (ví dụ: ngày 15) → hiển thị danh sách giao dịch của ngày đó.
  - Truy cập tab "Thống kê" → hệ thống hiển thị biểu đồ tròn theo danh mục (ăn uống, di chuyển...) và biểu đồ đường theo xu hướng thời gian.
  - Dữ liệu trong biểu đồ khớp với dữ liệu thu chi đã nhập.
  - Mục tiêu: kiểm tra khả năng phân tích và trực quan hóa dữ liệu tài chính để hỗ trợ ra quyết định.
  - **Code kiểm thử kịch bản 5:**

```dart
// test/calendar_test.dart
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:smartwallet/screens/calendar_screen.dart';
void main() {
  testWidgets('Hiển thị giao dịch khi chọn ngày 15', (tester) async {
    await tester.pumpWidget(
      MaterialApp(home: CalendarScreen()),
    );
    // Nhấn vào ngày 15
    await tester.tap(find.text('15'));
    await tester.pumpAndSettle();
    // Kiểm tra có hiển thị giao dịch
    expect(find.text('Giao dịch ngày'), findsOneWidget);
    expect(find.text('Ăn trưa'), findsOneWidget);
  });
}
```

Hình 61: Hình kết quả kịch bản kiểm thử 5

- **Kết quả:**



Hình 62: Hình kết quả kiểm thử 5

3.Kết luận kiểm thử các kịch bản:

Với chiến lược kiểm thử toàn diện được áp dụng xuyên suốt quá trình phát triển, dự án SmartWallet không chỉ đảm bảo chất lượng mã nguồn ở mức cao mà còn xây dựng được một nền tảng vững chắc cho khả năng bảo trì và mở rộng trong tương lai. Việc tuân thủ mô hình Kim tự tháp Kiểm thử — với trọng tâm là kiểm thử đơn vị, bổ trợ bởi kiểm thử giao diện và tích hợp — giúp phát hiện lỗi sớm, giảm thiểu rủi ro khi thay đổi logic nghiệp vụ, đồng thời duy trì sự ổn định của hệ thống qua từng phiên bản cập nhật. Các kịch bản kiểm thử được thiết kế sát với hành vi người dùng thực tế, từ những thao tác cơ bản như đăng nhập, quản lý ví, đến các chức năng phân tích nâng cao như thống kê và lịch sử giao dịch, đảm bảo rằng mọi tính năng đều hoạt động chính xác, mượt mà và đáng tin cậy. Bên cạnh đó, việc tích hợp kiểm thử hiệu năng và giám sát bộ nhớ cho thấy sự chủ động trong việc tối ưu trải nghiệm người dùng ngay cả trong điều kiện dữ liệu lớn hoặc sử dụng kéo dài. Nhờ đó, SmartWallet không chỉ là một ứng dụng học tập hiệu quả, mà còn là một sản phẩm có tiềm năng phát triển thành giải pháp quản lý tài chính cá nhân thực tiễn, sẵn sàng cho các tính năng mở rộng như đồng bộ đám mây, cảnh báo thông minh hay tích hợp AI trong tương lai.

Chương 9: Kết quả và đánh giá

1 Kết quả đạt được

Dự án SmartWallet đã triển khai thành công gần như toàn bộ các tính năng cốt lõi và hỗ trợ, cụ thể như sau:

- **Tính năng cốt lõi (Core Features – 100% hoàn thành):** Hệ thống hỗ trợ đầy đủ quy trình quản lý tài chính cá nhân, bao gồm: xác thực người dùng (đăng nhập/đăng xuất với kiểm tra đầu vào), quản lý ví điện tử (tạo, đọc, cập nhật, xóa các loại ví như tiền mặt, ngân hàng, thẻ tín dụng), chuyển tiền nội bộ giữa các ví (kèm kiểm tra số dư và giá trị hợp lệ), ghi nhận thu chi theo danh mục, xem lịch sử giao dịch qua lịch tương tác (TableCalendar), và phân tích dữ liệu thông qua biểu đồ trực quan (PieChart, LineChart).
- **Giao diện người dùng (UI/UX Features – 95% hoàn thành):** Ứng dụng áp dụng triệt để Material Design 3, mang đến trải nghiệm hiện đại với hiệu ứng gradient, hoạt ảnh mượt mà và hệ thống màu sắc hài hòa. Giao diện được tối ưu cho nhiều kích thước màn hình, hỗ trợ chế độ sáng/tối, điều hướng qua thanh bottom navigation với 5 tab chính, và các thành phần tương tác (nút, thẻ, menu) đều có phản hồi trực quan.
- **Quản lý dữ liệu (Data Management – 90% hoàn thành):** Dữ liệu được lưu trữ cục bộ thông qua SharedPreferences, kết hợp với mock data để phục vụ demo và kiểm thử. Hệ thống sử dụng Provider làm giải pháp quản lý trạng thái, đảm bảo

giao diện phản hồi linh hoạt khi dữ liệu thay đổi. Toàn bộ thông tin về ví, giao dịch và chi tiêu đều được lưu bền vững ngay sau mỗi thao tác.

2 Đánh giá tổng thể

2.1 Những điểm mạnh nổi bật

- Thiết kế giao diện xuất sắc: SmartWallet sở hữu giao diện hiện đại, trực quan với bảng màu gradient teal–indigo–purple, typography nhất quán (Roboto), và các micro-interaction mượt mà. Thiết kế đáp ứng tốt trên nhiều kích thước màn hình, tạo cảm giác chuyên nghiệp và thân thiện.
- Hiệu năng được tối ưu: Ứng dụng khởi động nhanh nhờ lazy loading, cuộn mượt với ListView.builder, và quản lý bộ nhớ hiệu quả nhờ giải phóng tài nguyên đúng cách. Tất cả hoạt ảnh đều đạt 60fps, đảm bảo trải nghiệm mượt mà.
- Kiến trúc phần mềm vững chắc: Mã nguồn được tổ chức theo nguyên tắc tách biệt trách nhiệm (separation of concerns), sử dụng pattern Provider cho quản lý trạng thái và MVVM để tách biệt logic nghiệp vụ khỏi giao diện. Cấu trúc module giúp dễ bảo trì và mở rộng.
- Chiến lược kiểm thử toàn diện: Với độ phủ kiểm thử cao và hệ thống kịch bản bao quát cả chức năng lẫn hiệu năng, dự án đảm bảo độ tin cậy và giảm thiểu rủi ro lỗi trong quá trình phát triển.

2.2 Hạn chế và thách thức

Mặc dù đạt được nhiều thành công, dự án vẫn còn một số hạn chế cần khắc phục trong các phiên bản tương lai:

- Thiếu tích hợp backend: Hiện tại, ứng dụng chỉ sử dụng dữ liệu cục bộ và mock data, chưa hỗ trợ đồng bộ đám mây, sao lưu tự động hay quản lý người dùng tập trung.
- Hạn chế về bảo mật: Dữ liệu lưu trữ chưa được mã hóa, thiếu xác thực sinh trắc học (vân tay/khuôn mặt) và chưa có cơ chế theo dõi lịch sử thao tác quan trọng.
- Khả năng mở rộng còn giới hạn: SharedPreferences không phù hợp với tập dữ liệu rất lớn (>10.000 giao dịch). Ngoài ra, ứng dụng chưa hỗ trợ phân trang, đa tiền tệ đầy đủ, hay kiến trúc offline-first.
- Phạm vi nền tảng hẹp: Dự án mới chỉ triển khai trên Android, chưa có phiên bản iOS, web hoặc desktop. Đồng thời, ứng dụng chưa hỗ trợ đa ngôn ngữ (i18n) hay deep linking.

2.3 So sánh với mục tiêu ban đầu

- Đạt được (trên 90%): Tất cả tính năng cốt lõi như quản lý ví, chuyển tiền, theo dõi thu chi, lịch và thống kê đều đã hoàn thiện và hoạt động ổn định.
- Đạt một phần (60–80%): Một số tính năng như thông báo, xuất/nhập dữ liệu, sao lưu và hỗ trợ đa tiền tệ đã có giao diện hoặc logic cơ bản nhưng chưa đầy đủ.
- Chưa triển khai: Các tính năng nâng cao như đồng bộ đám mây, thông báo real-time, phân tích AI hoặc chia sẻ dữ liệu với người khác vẫn chưa được hiện thực hóa.

3. Bài học kinh nghiệm

3.1. Kinh nghiệm kỹ thuật

- **Flutter & UI Development:** Provider là lựa chọn phù hợp cho ứng dụng quy mô vừa, trong khi ListView.builder và const constructor đóng vai trò then chốt trong tối ưu hiệu năng. Material Design 3 giúp rút ngắn thời gian thiết kế nhờ hệ thống component sẵn có.

- **Kiến trúc phần mềm:** Việc áp dụng MVVM và Repository pattern ngay từ đầu giúp tách biệt rõ ràng giữa giao diện, logic nghiệp vụ và lớp dữ liệu — tạo điều kiện thuận lợi cho kiểm thử và bảo trì.

- **Quản lý dữ liệu:** Hiện tại đang sử dụng mock data, nhưng ở các phiên bản tiếp theo sẽ sử triển khai tốt hơn bằng real-time data.

3.2. Kinh nghiệm quản lý dự án

- **Quản lý thời gian và phạm vi:** Thời gian dành cho thiết kế UI thường bị đánh giá thấp (~30–40%). Áp dụng phương pháp MVP (Minimum Viable Product) và ưu tiên tính năng cốt lõi giúp dự án đi đúng hướng.

- Quy trình phát triển: Việc sử dụng Git với chiến lược phân nhánh rõ ràng, kết hợp với tự động hóa kiểm thử (GitHub Actions), đã giúp quản lý mã nguồn hiệu quả và giảm lỗi tích hợp.

3.3. Bài học về thiết kế trải nghiệm người dùng

Thiết kế đơn giản, nhất quán và có phản hồi quan trọng hơn hiệu ứng hoa mỹ.

Các yếu tố như kích thước vùng chạm (≥44px), điều hướng bằng ngón tay cái (bottom navigation), và thông báo lỗi thân thiện đóng vai trò lớn trong việc nâng cao UX.

3.4 Khuyến nghị cho các dự án tương lai

- **Về kỹ thuật:** Nên thiết kế API và kiến trúc backend trước khi phát triển frontend; cân nhắc Riverpod/Bloc cho ứng dụng lớn; tích hợp CI/CD và bảo mật ngay từ đầu.

- **Về quy trình:** Thực hiện phỏng vấn người dùng, xây dựng prototype tương tác, và duy trì tài liệu kỹ thuật song song với phát triển.

- **Về học tập:** Theo dõi cộng đồng Flutter, tham khảo mã nguồn mở, và áp dụng các công cụ linting/formatting để nâng cao chất lượng mã.

4. Kết luận chương:
Dự án SmartWallet đã hoàn thành mục tiêu ban đầu, mang đến một ứng dụng quản lý tài chính cá nhân ổn định, trực quan và hiệu quả. Bên cạnh kết quả sản phẩm, những bài học kinh nghiệm về kỹ thuật, thiết kế và quản lý dự án là tài sản quý giá, không chỉ cho đồ án này mà còn cho các dự án phát triển phần mềm trong tương lai — đặc biệt trong lĩnh vực fintech và ứng dụng di động. Với nền tảng hiện có, SmartWallet hoàn toàn có tiềm năng phát triển thành một sản phẩm thực tiễn, đáp ứng nhu cầu quản lý tài chính ngày càng cao của người dùng hiện đại.

CHƯƠNG 10: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1 Kết luận

Dự án SmartWallet đã hoàn thành mục tiêu đề ra: xây dựng một ứng dụng quản lý tài chính cá nhân đơn giản, trực quan và hiệu quả trên nền tảng di động bằng Flutter. Qua quá trình nghiên cứu và triển khai, nhóm đã hiện thực hóa đầy đủ các chức năng cốt lõi như quản lý ví đa loại, ghi nhận thu chi theo danh mục, chuyển tiền nội bộ, xem lịch sử giao dịch qua lịch tương tác và phân tích dữ liệu thông qua biểu đồ trực quan.
Về mặt kỹ thuật, ứng dụng áp dụng kiến trúc MVVM kết hợp với mô hình quản lý trạng thái Provider, giúp tách biệt rõ ràng giữa giao diện người dùng và logic nghiệp vụ. Điều này không chỉ nâng cao chất lượng mã nguồn mà còn tạo điều kiện thuận lợi cho việc kiểm thử, bảo trì và mở rộng trong tương lai. Chiến lược kiểm thử toàn diện — từ unit test đến integration test — cùng với các bài kiểm tra hiệu năng và bộ nhớ — đã đảm bảo

SmartWallet vận hành ổn định, mượt mà và đáng tin cậy ngay cả khi xử lý khối lượng dữ liệu lớn.
Về mặt trải nghiệm người dùng, SmartWallet mang đến giao diện hiện đại theo chuẩn Material Design 3, với hệ thống màu sắc hài hòa, hoạt ảnh mượt mà và điều hướng trực quan. Các phản hồi tức thì (real-time validation), thông báo lỗi thân thiện và thiết kế đáp ứng (responsive) góp phần tạo nên một sản phẩm lấy con người làm trung tâm.
Mặc dù là một dự án học tập, SmartWallet đã chứng minh tiềm năng trở thành một công cụ hỗ trợ quản lý tài chính thực tiễn, đồng thời là minh chứng cho khả năng ứng dụng của Flutter trong phát triển ứng dụng fintech — nơi yêu cầu sự chính xác, bảo mật và trải nghiệm người dùng cao.

2 Hướng phát triển tương lai

Để biến SmartWallet từ nguyên mẫu học tập thành sản phẩm thực tế, có thể phát triển theo các hướng sau:

- Tích hợp backend: Xây dựng API, hỗ trợ đăng nhập (email/OAuth), đồng bộ dữ liệu đa nền tảng và sao lưu đám mây.
- Tăng cường bảo mật: Thêm xác thực sinh trắc học, mã hóa dữ liệu cục bộ và ghi nhận lịch sử thao tác.
- Mở rộng tính năng: Hỗ trợ đa tiền tệ, cảnh báo ngân sách, mục tiêu tiết kiệm và xuất/nhập dữ liệu (CSV/Excel).
- Phân tích thông minh: Dự báo chi tiêu bằng AI, đề xuất chi tiêu hợp lý và phát hiện giao dịch bất thường.
- Mở rộng nền tảng: Triển khai trên iOS, Web, hỗ trợ đa ngôn ngữ và tuân thủ chuẩn trợ năng.
- Tối ưu kiến trúc: Chuyển sang Hive/SQLite, áp dụng phân trang và thiết kế offline-first để nâng cao hiệu năng.

Với nền tảng hiện có, SmartWallet có tiềm năng trở thành một ứng dụng quản lý tài chính cá nhân thực tiễn, an toàn và thông minh trong tương lai.