Chiến lược phát triển Frontend và lộ trình thực thi hệ thống quản lý tài chính đa tầng

Trong bối cảnh phát triển phần mềm hiện đại năm 2025 và định hướng 2026, việc xây dựng các ứng dụng web phức tạp như hệ thống quản lý chi tiêu cá nhân và gia đình đòi hỏi một cách tiếp cận kiến trúc bền vững, linh hoạt và tối ưu hóa trải nghiệm người dùng ngay từ những giai đoạn đầu tiên.[1] Xu hướng chủ đạo hiện nay chuyển dịch từ các ứng dụng đa trang (MPA) truyền thống sang ứng dụng đơn trang (SPA) dựa trên thành phần (component-based), nơi logic phía máy khách (client-side) đóng vai trò trung tâm trong việc xử lý dữ liệu và phản hồi tương tác.[1] Báo cáo này trình bày một lộ trình chi tiết nhằm hoàn thiện tầng giao diện và logic nghiệp vụ phía Frontend cho đồ án quản lý chi tiêu, áp dụng các tiêu chuẩn công nghiệp tiên tiến nhất mà không cần phụ thuộc sớm vào sự hiện diện của hệ thống Backend.[3]

Việc tách biệt hoàn toàn tiến trình phát triển Frontend cho phép các kỹ sư tập trung tối đa vào thiết kế trải nghiệm người dùng (UX), đảm bảo tính thẩm mỹ và độ tin cậy của các thuật toán tài chính trước khi thực hiện tích hợp API RESTful.[1] Chiến lược này không chỉ giảm thiểu rủi ro tắc nghẽn trong quy trình phát triển mà còn tạo điều kiện cho việc kiểm thử đơn vị (unit testing) và kiểm thử tích hợp (integration testing) diễn ra sớm thông qua các kỹ thuật giả lập dữ liệu mạnh mẽ.[3]

1. Nền tảng công nghệ và kiến trúc hệ thống cốt lõi

Việc lựa chọn một hệ sinh thái công nghệ thống nhất là bước đi chiến lược đầu tiên nhằm đảm bảo tính ổn định và khả năng bảo trì lâu dài của dự án.[2] Hệ thống được xây dựng trên nền tảng React.js, một thư viện JavaScript hàng đầu với khả năng quản lý trạng thái hiệu quả và cộng đồng hỗ trợ khổng lồ, kết hợp cùng Vite để tối ưu hóa hiệu năng phát triển thông qua cơ chế Hot Module Replacement (HMR) cực nhanh.[2]

1.1 Chi tiết Tech Stack và lý do lựa chọn

Sự kết hợp giữa React và TypeScript cung cấp một lớp bảo vệ vững chắc thông qua việc kiểm soát kiểu dữ liệu nghiêm ngặt, điều này đặc biệt quan trọng trong các ứng dụng tài chính nơi sai số nhỏ nhất cũng có thể dẫn đến hậu quả lớn.[2]

Thành phần	Lựa chọn công nghệ	Vai trò và giá trị chiến lược
Framework	React 18/19	Cung cấp mô hình lập trình hướng thành phần và khả năng render đồng thời.[2]
Công cụ xây dựng	Vite	Tăng tốc độ khởi động dự án và đóng gói tài nguyên hiệu quả hơn Webpack.[2]
Ngôn ngữ	TypeScript	Đảm bảo tính an toàn của mã nguồn và cung cấp tài liệu tự thân cho logic tài chính.[2]
Quản lý trạng thái	Redux Toolkit	Trung tâm hóa dữ liệu giao dịch, ví và ngân sách, giải quyết triệt để vấn đề Props Drilling.[3]
Thiết kế giao diện	TailwindCSS	Xây dựng giao diện responsive nhanh chóng với hệ thống utility-first tiên tiến.[3]
Thư viện UI	shadcn/ui	Cung cấp các thành phần giao diện chất lượng cao, dễ tùy biến và tuân thủ các chuẩn accessibility.[1]
Giả lập API	Mock Service Worker	Đánh chặn các yêu cầu mạng và trả về dữ liệu giả lập, cho phép FE hoạt động như khi có BE thật.[4]

1.2 Kiến trúc Component-Based và tư duy "Chia để trị"

Kiến trúc dựa trên thành phần không chỉ đơn thuần là việc chia nhỏ giao diện mà là một tư duy tổ chức mã nguồn theo hướng mô-đun hóa.[3] Mỗi thành phần trong hệ thống quản lý chi tiêu được định nghĩa là sự kết hợp giữa giao diện (UI), logic nghiệp vụ và phong cách thiết kế

riêng biệt.[3] Điều này cho phép các kỹ sư tái sử dụng các thành phần như thẻ hiển thị giao dịch (TransactionCard) ở nhiều màn hình khác nhau từ bảng điều khiển tổng quát đến chi tiết ngân sách mà vẫn giữ được tính nhất quán.[3]

Hệ thống sẽ tuân thủ mô hình thư mục Feature-based Architecture, nơi mỗi tính năng lớn như "Quản lý ví" hay "Đơn giản hóa nợ" sẽ chứa đầy đủ các thành phần giao diện, custom hooks và logic quản lý trạng thái của chính nó.[8] Cách tiếp cận này giúp cô lập lỗi và tăng tốc độ mở rộng hệ thống mà không làm ảnh hưởng đến các phần khác của ứng dụng.[1]

2. Thiết kế giao diện và trải nghiệm người dùng (UI/UX) chuyên sâu

Đối với các ứng dụng tài chính, sự minh bạch và tốc độ truy cập thông tin là yếu tố then chốt để xây dựng niềm tin nơi người dùng.[11] Giao diện không chỉ cần đẹp mà còn phải giải tỏa được sự căng thẳng tâm lý thường thấy khi quản lý tiền bạc thông qua bảng màu nhẹ nhàng và bố cục rõ ràng.[13]

2.1 Bố cục Dashboard và phân phối thông tin

Hệ thống sẽ áp dụng cấu trúc ba cột tiêu chuẩn cho màn hình chính (Dashboard), một mô hình đã được chứng minh hiệu quả trong các ứng dụng quản lý tài chính hàng đầu.[9]

Cột	Chức năng chính	Thành phần giao diện
Cột trái	Điều hướng và Chuyển đổi ngữ cảnh	Sidebar với các liên kết chính, trình chuyển đổi giữa ví cá nhân và gia đình.[3]
Cột giữa	Phân tích xu hướng và Danh sách	Biểu đồ thu chi, danh sách các giao dịch gần đây, thanh tìm kiếm đa tiêu chí.[15]
Cột phải	Tóm tắt nhanh và Hành động	Tổng số dư ví, trạng thái ngân sách hiện tại, nút hành động nhanh (Thêm giao dịch).[9]

Giao diện Dashboard cần đảm bảo người dùng có thể thấy ngay tình trạng tài chính của mình trong vòng 3 giây đầu tiên sau khi đăng nhập.[13] Các chỉ số quan trọng như "Tổng chi tiêu

trong tháng" hay "Ngân sách còn lại" phải được làm nổi bật bằng kiểu chữ rõ ràng và màu sắc tương phản phù hợp (xanh lá cho thu nhập, đỏ cho chi phí).[14]

2.2 Thiết kế Responsive và Mobile-First

Với xu hướng quản lý chi tiêu mọi lúc mọi nơi, giao diện phải hoạt động hoàn hảo trên các thiết bị di động.[3] Chiến lược Mobile-First sẽ được áp dụng, ưu tiên tối ưu hóa các thao tác chạm và vuốt.[11] Trên màn hình nhỏ, cột trái và cột phải của Dashboard sẽ được ẩn vào menu hamburger và các tabs ở dưới đáy màn hình (Bottom Navigation), tập trung không gian hiển thị cho danh sách giao dịch ở giữa.[15] Các mục tiêu chạm (tap targets) phải có kích thước tối thiểu 44px để tránh lỗi thao tác.[16]

3. Quản lý trạng thái và logic nghiệp vụ tại Client

Trong giai đoạn chưa tích hợp API, Frontend sẽ đóng vai trò là "bộ não" thực sự của ứng dụng.[3] Mọi hoạt động tính toán từ cân đối số dư đến đơn giản hóa nợ nhóm đều được xử lý trực tiếp bởi các hàm JavaScript và được lưu trữ trong Redux Store.[17]

3.1 Cấu trúc Redux Store và các Slice nghiệp vụ

Dữ liệu sẽ được tổ chức thành các "Slice" độc lập để quản lý các mảng nghiệp vụ khác nhau, đảm bảo tính bất biến (immutability) thông qua thư viện Immer tích hợp sẵn trong Redux Toolkit.[8]

Tên Slice	Chức năng	Dữ liệu mô phỏng chính
transactions	Quản lý toàn bộ hồ sơ thu/chi	Mảng các object giao dịch với UUID, số tiền, ngày, danh mục.[3]
wallets	Theo dõi số dư và loại ví	Danh sách ví CASH, BANK, EWALLET gắn với user_id hoặc family_id.[3]
families	Quản lý cấu trúc nhóm	Thông tin gia đình, danh sách thành viên, vai trò (Admin/Manager/Member).[3]

budgets	Giám sát hạn mức chi tiêu	Ngân sách theo tháng cho từng danh mục ăn uống, đi lại.[3]
auth	Mô phỏng xác thực và phân quyền	Thông tin profile, token giả lập, quyền truy cập chức năng.[7]

Sự phức tạp trong quản lý tài chính nằm ở mối quan hệ giữa các thực thể. Chẳng hạn, khi một giao dịch chi tiêu mới được thêm vào, logic phía Client phải đồng thời thực hiện ba nhiệm vụ: trừ số dư trong walletSlice, cập nhật tổng chi tiêu trong budgetSlice và thông báo cho debtSlice nếu đó là một giao dịch chia sẻ trong gia đình.[3]

3.2 Quy tắc tách biệt dữ liệu Cá nhân và Gia đình

Dựa trên đặc tả ERD, hệ thống yêu cầu một sự tách biệt nghiêm ngặt giữa ví cá nhân và ví gia đình.[3] Logic phía Frontend sẽ thực thi quy tắc XOR (loại trừ lẫn nhau): một ví chỉ được thuộc về một người dùng hoặc một gia đình, tuyệt đối không được có cả hai hoặc không thuộc về ai.[3]

Tại giao diện người dùng, điều này được thể hiện thông qua một bộ lọc toàn cục (Global Context Switcher). Khi người dùng chọn chế độ "Cá nhân", Frontend sẽ thực hiện lọc toàn bộ dữ liệu trong store để chỉ hiển thị các giao dịch liên kết với ví của cá nhân đó.[3] Ngược lại, ở chế độ "Gia đình", hệ thống sẽ chuyển sang hiển thị các ví chung mà người dùng là thành viên, đồng thời kích hoạt các tính năng đặc thù như xem nợ chung và đề xuất thanh toán.[3]

4. Thuật toán logic phức tạp: Đơn giản hóa nợ và Giao dịch định kỳ

Yêu cầu cốt lõi giúp đồ án vượt lên các ứng dụng CRUD thông thường chính là việc thực thi các thuật toán nghiệp vụ chuyên sâu ngay tại trình duyệt.[3]

4.1 Thuật toán tối ưu hóa dư nợ (Debt Simplification Algorithm)

Trong một nhóm gia đình, việc nhiều người cùng chi trả cho các mục đích chung tạo ra một mạng lưới nợ chéo phức tạp. Hệ thống sẽ cài đặt thuật toán dựa trên lý thuyết đồ thị để giảm thiểu số lượng giao dịch cần thiết để tất cả thành viên về trạng thái cân bằng.[3]

Quy trình thực thi thuật toán tại Client-side:

1. **Tính toán vị thế ròng (Net Flow):** Frontend duyệt qua danh sách thành viên gia đình và các giao dịch chung. Cho mỗi người dùng i, tính toán giá trị:

$Net_i = \sum Paid_i - \sum Owed_i$. Kết quả là một con số dương nếu người đó đang cho vay và con số âm nếu đang nợ.[19]

2. **Phân nhóm thực thể:** Chia danh sách thành hai tập hợp riêng biệt: Debtors (những người có $Net < 0$) và Creditors (những người có $Net > 0$).[21]
3. **Tối ưu hóa bằng giải thuật tham lam (Greedy):** Sắp xếp cả hai danh sách theo giá trị tuyệt đối giảm dần. Lấy người nợ nhiều nhất ghép đôi với người cho vay nhiều nhất.[3]
4. **Thanh toán ảo:** Tạo một giao dịch chuyển tiền ảo với số tiền $S = \min(|Net_{debtor}|, Net_{creditor})$. Cập nhật lại giá trị ròng của cả hai người và lặp lại cho đến khi toàn bộ số dư ròng về không.[3]

Thuật toán này đảm bảo rằng với n thành viên, số lượng giao dịch thanh toán tối đa chỉ là $n - 1$, giúp tiết kiệm thời gian và giảm bớt sự phiền hà trong quản lý tài chính nhóm.[3] Giao diện sẽ hiển thị kết quả này dưới dạng một đồ thị hoặc danh sách các bước thanh toán trực quan (ví dụ: "A hãy chuyển cho C 200.000đ") thay vì để người dùng tự tính toán thủ công.[3]

4.2 Công cụ xử lý giao dịch định kỳ (Recurring Transaction Engine)

Hệ thống cần quản lý các khoản chi cố định như tiền nhà, tiền internet hay các gói thuê bao định kỳ.[3] Để mô phỏng tính năng này mà không có Backend scheduler, Frontend sẽ thiết lập một cơ chế "Lazy Check".[3] Khi ứng dụng khởi động hoặc khi người dùng làm mới trang, một Service Layer sẽ so sánh ngày hiện tại với next_run_date của các mẫu định kỳ (Recurring Patterns) được lưu trong Store.[3] Nếu đến hạn, Frontend sẽ tự động sinh ra các bản ghi giao dịch mới và cập nhật lại ngày chạy kế tiếp, đồng thời gửi thông báo Toast đến người dùng về việc các giao dịch đã được tạo tự động.[11]

5. Chiến lược giả lập dữ liệu lớn và Kiểm thử hiệu năng

Một trong những yêu cầu bắt buộc của đồ án là khả năng xử lý từ 500 đến 1000 bản ghi dữ liệu.[3] Để đáp ứng điều này ngay trong giai đoạn Frontend-only, dự án sẽ sử dụng thư viện Faker.js để tạo ra một cơ sở dữ liệu giả lập phong phú và có cấu trúc.[3]

5.1 Xây dựng bộ Seeder dữ liệu thực tế

Dữ liệu giả lập không được là các chuỗi ngẫu nhiên vô nghĩa mà phải phản ánh chính xác hành vi chi tiêu của một hộ gia đình thực tế để phục vụ kiểm thử tính đúng đắn của các biểu đồ và báo cáo.[4]

Đối tượng	Cách thức giả lập với Faker.js	Ràng buộc nghiệp vụ cần tuân thủ
Giao dịch	faker.finance.amount() kèm mô tả từ faker.commerce.productName().[24]	Số tiền phải là numeric(18,2) để tránh sai số dấu phẩy động.[3]
Thời gian	faker.date.between({ from: '2025-01-01', to: new Date() }).[24]	Sử dụng timestamptz để đảm bảo nhất quán múi giờ.[3]
Ví tiền	Gán ngẫu nhiên cho một User hoặc một Family dựa trên logic XOR.[3]	Số dư ví phải bằng tổng thu trừ tổng chi trong lịch sử giao dịch giả lập.[3]
Danh mục	Chọn ngẫu nhiên từ bộ enum: Ăn uống, Di chuyển, Lương, Quà tặng.[3]	Giao dịch INCOME/EXPENSE bắt buộc phải có category_id.[3]

5.2 Tối ưu hóa hiệu năng render với Large Datasets

Việc hiển thị đồng thời 1000 giao dịch sẽ khiến trình duyệt trở nên nặng nề và giảm độ mượt của hoạt ảnh.[6] Frontend sẽ áp dụng kỹ thuật "Ảo hóa danh sách" (List Virtualization) thông qua thư viện react-window hoặc tanstack-virtual.[6] Cơ chế này chỉ render các dòng giao dịch hiện đang nằm trong khung nhìn (viewport) của người dùng, giúp giảm số lượng DOM node từ hàng nghìn xuống còn vài chục, qua đó duy trì tốc độ phản hồi 60fps ngay cả trên các thiết bị di động cấu hình thấp.[6]

Ngoài ra, việc tính toán các chỉ số báo cáo tổng hợp từ 1000 bản ghi sẽ được tối ưu bằng cách sử dụng useMemo của React để ghi nhớ kết quả, tránh việc thực hiện lại các phép tính nợ phức tạp mỗi khi người dùng thay đổi các yếu tố giao diện nhỏ lẻ.[2]

6. Trực quan hóa dữ liệu tài chính (Data Visualization)

Báo cáo và thống kê là giá trị gia tăng lớn nhất của ứng dụng quản lý chi tiêu.[3] Frontend sẽ tích hợp các thư viện biểu đồ hiện đại như Recharts để chuyển đổi dữ liệu giao dịch thô thành những thông tin chi tiết có giá trị.[26]

6.1 Lựa chọn thư viện và các loại biểu đồ chiến lược

Recharts được chọn vì tính tương thích tuyệt vời với React thông qua cấu trúc component-based và hỗ trợ SVG giúp các biểu đồ luôn sắc nét trên mọi độ phân giải màn hình.[6]

Loại biểu đồ	Mục đích sử dụng	Đặc điểm tương tác
Area Chart	Xu hướng thu nhập vs chi tiêu theo thời gian.[3]	Cho phép di chuột để xem chi tiết số dư ròng tại từng thời điểm.[26]
Donut Chart	Phân bổ chi tiêu theo danh mục (Cơ cấu chi phí).[3]	Nhấp vào từng phần để lọc danh sách giao dịch tương ứng bên dưới.[29]
Bar Chart	So sánh ngân sách kế hoạch và thực tế.[25]	Sử dụng màu đỏ để cảnh báo các cột chi tiêu vượt hạn mức.[3]
Network Graph	Trực quan hóa các mối quan hệ nợ trong gia đình.[26]	Hiển thị các mũi tên dòng tiền giúp người dùng hiểu tại sao hệ thống đề xuất trả nợ như vậy.[3]

Để đảm bảo biểu đồ không bị rối mắt khi có quá nhiều dữ liệu, Frontend sẽ thực hiện logic nhóm dữ liệu (Aggregation) ngay tại máy khách. Thay vì vẽ từng giao dịch đơn lẻ, hệ thống sẽ nhóm theo ngày, tuần hoặc tháng tùy theo lựa chọn của người dùng, giúp biểu đồ trở nên rõ ràng và dễ hiểu hơn.[3]

7. Xử lý biểu mẫu, Validation và Phản hồi hệ thống

Giao diện nhập liệu là nơi người dùng tương tác nhiều nhất. Sự sai sót trong dữ liệu đầu vào có thể làm hỏng toàn bộ hệ thống tính toán tài chính, vì vậy việc kiểm soát dữ liệu tại Frontend là vô cùng nghiêm ngặt.[3]

7.1 Cơ chế Validation đa tầng với Formik và Yup

Hệ thống sử dụng Formik để quản lý trạng thái form và Yup để định nghĩa các sơ đồ kiểm tra (Validation Schema) một cách khai báo.[3]

Các quy tắc kiểm tra bắt buộc:

- **Định dạng tiền tệ:** Số tiền phải lớn hơn 0, không được chứa ký tự lạ và tự động định

dạng dấu phân cách phần nghìn khi người dùng nhập liệu (ví dụ: 1.000.000).[3]

- **Logic thời gian:** Ngày giao dịch không được vượt quá ngày hiện tại đối với các khoản chi thực tế, đảm bảo tính toàn vẹn của báo cáo quá khứ.[3]
- **Ràng buộc ví và danh mục:** Nếu loại giao dịch là TRANSFER (chuyển tiền), hệ thống bắt buộc người dùng phải chọn cả ví nguồn và ví đích, đồng thời ẩn đi trường chọn danh mục.[3]
- **Chia sẻ giao dịch:** Khi tạo một giao dịch chung, tổng số tiền được gán cho các thành viên trong transaction_shares phải khớp chính xác 100% với tổng số tiền của giao dịch gốc.[3] Frontend sẽ thực hiện tính toán thời gian thực và hiển thị số tiền còn thiếu/thừa ngay trên form.[3]

7.2 Phản hồi trực quan và Trạng thái hệ thống

Người dùng luôn cần biết điều gì đang xảy ra bên trong ứng dụng.[11] Frontend sẽ cài đặt các thành phần phản hồi tức thì:

- **Skeleton Screens:** Trong các tình huống mô phỏng tải dữ liệu lớn, thay vì hiển thị biểu tượng xoay tròn (spinner) gây nhàm chán, hệ thống sẽ hiển thị các khung trống mờ để tạo cảm giác nội dung đang được chuẩn bị sẵn sàng.[3]
- **Toast Notifications:** Các thông báo thành công hoặc thất bại sẽ xuất hiện ở góc màn hình trong thời gian ngắn (200-500ms) để không làm gián đoạn luồng làm việc.[3]
- **Positive Friction (Ma sát tích cực):** Đối với các hành động nguy hiểm như xóa ví hoặc hủy nhóm gia đình, Frontend sẽ yêu cầu một bước xác nhận bổ sung "Bạn có chắc chắn không?" để ngăn chặn các sai sót vô tình.[13]

8. Bảo mật và Phân quyền người dùng (RBAC) phía Client

Dù chưa có Backend thực thi quyền hạn, Frontend vẫn phải xây dựng một hệ thống phân quyền logic hoàn chỉnh để trình diễn khả năng bảo vệ dữ liệu.[3]

8.1 Mô phỏng xác thực và Protected Routes

Hệ thống sẽ sử dụng React Router để quản lý các tuyến đường. Một Higher-Order Component (HOC) mang tên PrivateRoute sẽ bọc toàn bộ các trang yêu cầu đăng nhập.[3] Thành phần này kiểm tra trạng thái trong authSlice; nếu người dùng chưa "đăng nhập" (với một token giả lập được lưu trong LocalStorage), hệ thống sẽ tự động điều hướng họ về trang Login.[3]

8.2 Logic phân quyền chức năng (Role-Based Display)

Hệ thống hỗ trợ 3 vai trò chính, và Frontend sẽ điều chỉnh giao diện dựa trên các vai trò này

để đảm bảo người dùng chỉ thấy những gì họ được phép thực hiện.[3]

Vai trò	Giới hạn giao diện phía Frontend
System Admin	Hiển thị thêm tab "Quản trị hệ thống", cho phép xem thống kê tổng quát toàn sàn và quản lý danh sách người dùng.[3]
Family Manager	Hiển thị các nút "Mời thành viên", "Thiết lập hạn mức ngân sách" và "Phê duyệt chi tiêu lớn".[3]
Family Member	Toàn bộ giao diện ngân sách gia đình chuyển sang chế độ Read-only. Ẩn các tính năng quản lý nhóm, chỉ giữ lại quyền ghi chép giao dịch cá nhân.[3]

Mọi nỗ lực truy cập trái phép bằng cách thay đổi URL thủ công sẽ bị chặn đứng bởi logic kiểm tra vai trò tại mỗi component trang, trả về một trang thông báo lỗi 403 Forbidden chuyên nghiệp.[7]

9. Lộ trình thực thi chi tiết qua các giai đoạn (Sprint)

Lộ trình được chia thành 10 chặng nước rút (Sprints), mỗi chặng tập trung vào một nhóm giá trị cụ thể, đảm bảo sự tiến triển liên tục của tầng Frontend.[3]

Giai đoạn 1: Nền tảng và Thiết kế (Sprint 1-2)

- Khởi tạo dự án với Vite, TypeScript và cấu hình các công cụ kiểm soát chất lượng mã nguồn (ESLint, Prettier).[3]
- Xây dựng hệ thống Design System cơ sở: bảng màu, kiểu chữ và các thành phần nguyên tử (Atoms) như Button, Input, Modal.[3]
- Hoàn thiện giao diện Đăng nhập/Đăng ký và cấu hình điều hướng cơ bản.[3]

Giai đoạn 2: Lõi nghiệp vụ và Giả lập dữ liệu (Sprint 3-5)

- Thiết lập Redux Store và cài đặt logic CRUD cho các thực thể Giao dịch và Ví.[17]
- Tích hợp Faker.js để sinh 1000 bản ghi mẫu và cài đặt tính năng phân trang, tìm kiếm đa tiêu chí phía Client.[3]
- Xây dựng bộ form nhập liệu thông minh với đầy đủ validation và định dạng tiền tệ tự

động.[3]

Giai đoạn 3: Tính năng Gia đình và Thuật toán (Sprint 6-8)

- Thực hiện logic chuyển đổi ngữ cảnh Cá nhân/Gia đình và áp dụng các quy tắc XOR cho ví.[3]
- Cài đặt thuật toán Đơn giản hóa nợ (Greedy Algorithm) và xây dựng giao diện trực quan hóa nợ nhóm.[3]
- Triển khai công cụ Giao dịch định kỳ với cơ chế tự động sinh dữ liệu khi load trang.[3]

Giai đoạn 4: Trực quan hóa và Hoàn thiện (Sprint 9-10)

- Tích hợp Recharts để xây dựng Dashboard thống kê với các biểu đồ Area, Donut và Bar.[26]
- Xây dựng tính năng Xuất báo cáo (Export) ra file PDF/Excel giả lập ngay tại trình duyệt.[3]
- Tối ưu hóa hiệu năng render bằng List Virtualization và kiểm tra bảo mật XSS phía Client.[6]
- Đóng gói toàn bộ ứng dụng bằng Docker (Nginx serving static files) để sẵn sàng bàn giao.[3]

10. Chiến lược chuyển đổi sang tích hợp API Backend

Dù phần tích hợp API được tạm hoãn, Frontend vẫn cần được thiết kế theo hướng "Sẵn sàng kết nối" (Integration-Ready).[1] Toàn bộ các yêu cầu lấy dữ liệu hiện tại sẽ được đóng gói vào các Service hoặc Thunk trong Redux.[7] Khi hệ thống Backend hoàn thiện và cung cấp các Endpoint RESTful, các kỹ sư chỉ cần thay thế phần logic trả về dữ liệu giả bằng các lệnh gọi axios thực tế.[17]

Việc sử dụng Axios Interceptors sẽ cho phép hệ thống tự động đính kèm JWT vào header của mọi yêu cầu trong tương lai mà không cần sửa đổi từng component lẻ.[3] Điều này giúp quá trình tích hợp diễn ra vô cùng nhanh chóng và ít lỗi, vì toàn bộ logic hiển thị và xử lý dữ liệu đã được kiểm chứng kỹ lưỡng trong giai đoạn phát triển Frontend độc lập.[3]

11. Các biện pháp bảo mật và tối ưu hóa hiệu năng bổ sung

Ngay cả khi hoạt động độc lập, ứng dụng vẫn phải tuân thủ các tiêu chuẩn bảo mật nghiêm ngặt để bảo vệ dữ liệu tài chính mô phỏng của người dùng.[3]

- **Chống SQL Injection và XSS:** Frontend tuyệt đối không sử dụng dangerouslySetInnerHTML. Mọi dữ liệu do người dùng nhập vào đều được React tự động escape để ngăn chặn các cuộc tấn công script độc hại.[3]
- **Bảo vệ dữ liệu nhạy cảm:** Các tham số cấu hình như địa chỉ server mô phỏng hoặc các

API key của các dịch vụ bên thứ ba (nếu có) phải được lưu trữ trong tệp .env và được nạp thông qua biến môi trường, tuyệt đối không hard-code vào mã nguồn.[3]

- **Tối ưu hóa bộ nhớ:** Với khối lượng 1000 bản ghi trong Redux Store, việc quản lý bộ nhớ tại Client trở nên quan trọng. Hệ thống sẽ thực hiện dọn dẹp (cleanup) các trạng thái không còn sử dụng khi người dùng chuyển đổi giữa các module lớn để tránh tình trạng rò rỉ bộ nhớ (memory leak).[3]

Lộ trình phát triển Frontend chi tiết này không chỉ đảm bảo đồ án hoàn thiện về mặt tính năng mà còn thể hiện tư duy kiến trúc chuyên nghiệp, sẵn sàng cho việc mở rộng thành một sản phẩm thương mại thực thụ trong tương lai.[1] Việc tập trung vào logic phức tạp và trải nghiệm người dùng ngay từ đầu chính là chìa khóa để tạo ra một hệ thống quản lý chi tiêu thực sự có giá trị và đột phá.[12]

Nguồn trích dẫn

1. Modernize Frontend in 2025: Future-proof your web and app development - Anima Blog, truy cập vào tháng 2 4, 2026, https://www.animaapp.com/blog/enterprise/modernize-frontend-in-2025-future-proof-your-web-and-app-development/
2. Modern Front End Development: Complete 2025 Guide - DevCrew, truy cập vào tháng 2 4, 2026, https://devcrew.io/modern-front-end-development-guide/
3. Mini-SRS -Đặc tả yêu cầu phần mềm.docx
4. Mock Data Generator: The Key to Efficient Software Testing - DEV Community, truy cập vào tháng 2 4, 2026, https://dev.to/keploy/mock-data-generator-the-key-to-efficient-software-testing-2nac
5. Generating Realistic Test Data with Faker.js - OpenReplay Blog, truy cập vào tháng 2 4, 2026, https://blog.openreplay.com/generating-realistic-test-data-faker-js/
6. The Top 5 React Chart Libraries to Know in 2026 for Modern Dashboards | Syncfusion Blogs, truy cập vào tháng 2 4, 2026, https://www.syncfusion.com/blogs/post/top-5-react-chart-libraries
7. Create a React App with Redux and TypeScript Complete | by Sunil Nepali | Medium, truy cập vào tháng 2 4, 2026, https://medium.com/@sunilnepali844/create-a-scalable-react-app-with-redux-toolkit-and-typescript-629bf76dd81e
8. Redux Essentials, Part 2: Redux Toolkit App Structure, truy cập vào tháng 2 4, 2026, https://redux.js.org/tutorials/essentials/part-2-app-structure
9. Financial Dashboard - Free React, Tailwind Component - Purecode.AI, truy cập vào tháng 2 4, 2026, https://purecode.ai/community/financialdashboard-tailwind-financedashboard
10. I built an open-source React + Tailwind + shadcn admin dashboard — feedback welcome : r/reactjs - Reddit, truy cập vào tháng 2 4, 2026,

https://www.reddit.com/r/reactjs/comments/1pqejjp/i_built_an_opensource_react_tailwind_shadcn_admin/
11. Payment App UI Design Services for Seamless and Secure User Experiences, truy cập vào tháng 2 4, 2026, https://www.thealien.design/insights/payment-app-ui-design
12. Fintech Design: Mastering UI UX for Finance Apps - ProCreator Design, truy cập vào tháng 2 4, 2026, https://procreator.design/blog/fintech-design-expert-ui-ux-tips/
13. Fintech UX best practices for designing finance apps - Merge Rocks, truy cập vào tháng 2 4, 2026, https://merge.rocks/blog/ux-design-best-practices-for-fintech-apps
14. Financial App Design: UX Strategies for High-Performance Mobile Trading - Netguru, truy cập vào tháng 2 4, 2026, https://www.netguru.com/blog/financial-app-design
15. Portfolio Project: Build a Responsive Finance Dashboard UI with React & Tailwind CSS, truy cập vào tháng 2 4, 2026, https://www.youtube.com/watch?v=RRmSRYnwhG0
16. Mobile checkout UI: Best practices for building a better payment flow - Stripe, truy cập vào tháng 2 4, 2026, https://stripe.com/resources/more/mobile-checkout-ui
17. How to Integrate React with Redux Toolkit: A Complete Guide - Zignuts Technolab, truy cập vào tháng 2 4, 2026, https://www.zignuts.com/blog/how-to-integrate-react-with-redux-toolkit
18. Redux Toolkit Quick Start, truy cập vào tháng 2 4, 2026, https://redux-toolkit.js.org/tutorials/quick-start
19. Algorithm Behind Splitwise's Debt Simplification Feature | by Mithun Mohan K | Medium, truy cập vào tháng 2 4, 2026, https://medium.com/@mithunmk93/algorithm-behind-splitwises-debt-simplification-feature-8ac485e97688
20. How does the Splitwise algorithm work? | by Himanshi Sharma - Medium, truy cập vào tháng 2 4, 2026, https://medium.com/@howoftech/how-does-the-splitwise-algorithm-work-dc1de5eaa371
21. Algorithm to share/settle expenses among a group - Stack Overflow, truy cập vào tháng 2 4, 2026, https://stackoverflow.com/questions/974922/algorithm-to-share-settle-expenses-among-a-group
22. Algorithm to simplify a weighted directed graph of debts - Stack Overflow, truy cập vào tháng 2 4, 2026, https://stackoverflow.com/questions/15723165/algorithm-to-simplify-a-weighted-directed-graph-of-debts
23. Effortless Data Generation with Faker.js: A Developer's Guide - Vue School Articles, truy cập vào tháng 2 4, 2026, https://vueschool.io/articles/vuejs-tutorials/effortless-data-generation-with-faker-js-a-developers-guide/
24. Getting Started With Faker.js: A Developer's Guide - Testim, truy cập vào tháng 2

4, 2026, https://www.testim.io/blog/getting-started-with-faker-js/
25. Showcase of the Best React Charts and Graphs - SciChart, truy cập vào tháng 2 4, 2026, https://www.scichart.com/blog/showcase-of-the-best-react-charts-and-graphs/
26. 8 Best React Chart Libraries for Visualizing Data in 2025 - Embeddable, truy cập vào tháng 2 4, 2026, https://embeddable.com/blog/react-chart-libraries
27. Top 10 React Chart Libraries for Data Visualization in 2025 - OpenReplay Blog, truy cập vào tháng 2 4, 2026, https://blog.openreplay.com/react-chart-libraries-2025/
28. React Financial & Advanced Charts - CanvasJS, truy cập vào tháng 2 4, 2026, https://canvasjs.com/react-charts/react-financial-advanced-charts/
29. A guide to React graph visualization - Cambridge Intelligence, truy cập vào tháng 2 4, 2026, https://cambridge-intelligence.com/react-graph-visualization-library/
30. System Design of Splitwise Backend (Expense Sharing Apps) - GeeksforGeeks, truy cập vào tháng 2 4, 2026, https://www.geeksforgeeks.org/system-design/system-design-of-backend-for-expense-sharing-apps-like-splitwise/
31. Tailgrids – Tailwind React Component Library and Design System, truy cập vào tháng 2 4, 2026, https://tailgrids.com/
32. 5 Best JS Chart Libraries for Data Visualization in 2026 | Qrvey, truy cập vào tháng 2 4, 2026, https://qrvey.com/blog/js-chart-library/
33. Top React Chart Libraries to Use in 2026 - Aglowid IT Solutions, truy cập vào tháng 2 4, 2026, https://aglowiditsolutions.com/blog/react-chart-libraries/