Lộ Trình Phát Triển Hệ Thống Quản Lý Chi Tiêu Gia Đình: Phân Tích Kiến Trúc, Thiết Kế và Thực Thi Chiến Lược trong 300 Giờ

# 1. Tổng Quan Điều Hành và Phạm Vi Dự Án

Việc phát triển một hệ thống quản lý tài chính cá nhân và gia đình không chỉ đơn thuần là việc xây dựng một ứng dụng ghi chép thu chi, mà là một bài toán kỹ thuật phức tạp đòi hỏi sự kết hợp nhuần nhuyễn giữa kiến thức về công nghệ phần mềm, quản lý cơ sở dữ liệu và tâm lý học hành vi người dùng. Dựa trên tài liệu yêu cầu "Yeu cau de tai LTMT_copy.pdf" [1], dự án này được định vị là một ứng dụng web hiện đại (Web Application), tuân thủ nghiêm ngặt các tiêu chuẩn công nghiệp về kiến trúc Client-Server, bảo mật và quy trình phát triển phần mềm (SDLC).

Báo cáo này, được biên soạn dưới góc nhìn của một Kiến trúc sư Giải pháp Phần mềm (Solution Architect), sẽ cung cấp một lộ trình chi tiết, toàn diện để hoàn thành dự án trong giới hạn 300 giờ làm việc. Mục tiêu không chỉ là đáp ứng các yêu cầu cơ bản như CRUD (Create, Read, Update, Delete) hay phân quyền RBAC, mà còn tích hợp các logic nghiệp vụ phức tạp như thuật toán tối ưu hóa nợ (Debt Simplification), dự báo tài chính (Financial Forecasting) và quản lý ngân sách động. Bản báo cáo này sẽ đi sâu vào phân tích kỹ thuật, thiết kế cơ sở dữ liệu, chiến lược triển khai Docker và các biện pháp bảo mật, đảm bảo sản phẩm cuối cùng là một hệ thống robust, scalable và secure.

## 1.1 Mục Tiêu Dự Án và Ràng Buộc Kỹ Thuật

Dựa trên tài liệu yêu cầu [1], dự án được thiết kế để giải quyết bài toán quản lý tài chgggggggggggggggggggggggggggggính trong bối cảnh hộ gia đình, nơi việc chia sẻ chi phí và theo dõi ngân sách chung thường gặp nhiều khó khăn.

| Hạng Mục | Yêu Cầu Cụ Thể | Chiến Lược Thực Thi |
|---|---|---|
| **Kiến Trúc** | Client-Server (Tách biệt Frontend/Backend) | RESTful API (Node.js/Express) + SPA (React.js) |
| **Frontend** | Single Page Application (SPA), Component-based | React.js với Vite, quản lý state bằng Redux Toolkit |
| **Backend** | RESTful API trả về JSON | Node.js (Express) tuân thủ mô hình MVC/Service Layer |

| | | |
|---|---|---|
| **Cơ Sở Dữ Liệu** | RDBMS (MySQL/PostgreSQL), Migrations, Seeders | PostgreSQL + Sequelize ORM (cho Migrations & Seeding) |
| **Triển Khai** | Docker, docker-compose | Container hóa từng dịch vụ (App, DB, Reverse Proxy) |
| **Bảo Mật** | JWT, Hash Password, Chống SQL Injection/XSS | Bcrypt, Helmet, Express-validator, Parameterized Queries |
| **Logic Nghiệp Vụ** | Xử lý logic phức tạp (không chỉ nhập xuất) | Thuật toán chia tiền nhóm (Splitwise-like), Dự báo dòng tiền |

Việc lựa chọn công nghệ dựa trên sự phổ biến, hiệu năng và khả năng hỗ trợ cộng đồng mạnh mẽ. React.js và Node.js tạo thành một stack JavaScript đồng nhất (full-stack JS), giúp tối ưu hóa thời gian phát triển trong giới hạn 300 giờ. PostgreSQL được chọn thay vì MySQL do khả năng xử lý các truy vấn phức tạp và hỗ trợ kiểu dữ liệu JSON tốt hơn, phục vụ cho việc lưu trữ các cấu hình linh hoạt của người dùng.[2]

# 2. Phân Tích Yêu Cầu và Mô Hình Hóa Nghiệp Vụ (Giai Đoạn 1: 0 - 40 Giờ)

Gibai đoạn đầu tiên và quan trọng nhất là chuyển đổi các yêu cầu trừu tượng thành các đặc tả kỹ thuật cụ thể. Sự thất bại trong giai đoạn này thường dẫn đến việc phải làm lại (re-work) tốn kém trong các giai đoạn sau.[3]

## 2.1 Phân Tích Vai Trò Người Dùng (RBAC - Role Based Access Control)

Yêu cầu [1] quy định hệ thống phải có tối thiểu 3 vai trò. Trong ngữ cảnh quản lý chi tiêu gia đình, việc ánh xạ các vai trò này cần phản ánh cấu trúc thực tế của một hộ gia đình hoặc một nhóm người dùng chung.

1. **System Admin (Quản Trị Hệ Thống):**
    - *Quyền hạn:* Có quyền truy cập toàn cục vào hệ thống, xem dashboard thống kê tổng quan (số lượng người dùng, tổng giao dịch toàn hệ thống), quản lý danh sách người dùng (khóa/mở khóa tài khoản), cấu hình các tham số hệ thống (ví dụ: loại tiền tệ hỗ trợ, danh mục mặc định).

    - *Ý nghĩa:* Đảm bảo tính toàn vẹn và vận hành của hệ thống kỹ thuật, không can thiệp vào dữ liệu tài chính riêng tư của các gia đình trừ khi cần thiết cho việc hỗ trợ.[4]
2. **Family Manager (Trưởng Nhóm/Chủ Hộ - Tương ứng Staff):**
    - *Quyền hạn:* Tạo nhóm gia đình ("Family Group"), mời thành viên tham gia, thiết lập ngân sách tổng (Master Budget), phê duyệt các khoản chi vượt hạn mức, xem báo cáo chi tiết của tất cả thành viên trong gia đình.
    - *Nghiệp vụ:* Đây là vai trò trung tâm trong logic nghiệp vụ, chịu trách nhiệm quản lý dòng tiền chung và phân bổ hạn mức cho các thành viên.[5]
3. **Family Member (Thành Viên - Tương ứng Customer):**
    - *Quyền hạn:* Ghi chép giao dịch cá nhân và giao dịch chung, xem báo cáo cá nhân, xem trạng thái ngân sách chung (read-only), đề xuất các khoản chi lớn.
    - *Hạn chế:* Không thể xem chi tiết các giao dịch riêng tư của thành viên khác nếu không được chia sẻ, không thể thay đổi cấu trúc ngân sách tổng.[6]

Mô hình RBAC này không chỉ thỏa mãn yêu cầu của đề tài mà còn tạo ra sự phân cấp rõ ràng, cho phép triển khai các logic bảo mật ở tầng Middleware một cách hiệu quả.[7]

## 2.2 Xác Định Logic Nghiệp Vụ Phức Tạp (Complex Business Logic)

Để đạt điểm tối đa và tạo ra giá trị thực tế, hệ thống sẽ tích hợp ba mô đun logic phức tạp, vượt xa các thao tác CRUD thông thường:

- Logic 1: Thuật Toán Đơn Giản Hóa Nợ (Debt Simplification Algorithm):
  Trong một gia đình hoặc nhóm, khi nhiều người chi trả cho nhau (ví dụ: A trả tiền điện, B trả tiền ăn, C trả tiền Internet), mạng lưới nợ nần trở nên phức tạp. Hệ thống sẽ cài đặt một thuật toán đồ thị (Graph Theory) để tối ưu hóa số lượng giao dịch cần thiết để thanh toán nợ. Thay vì A trả B, B trả C, C trả A, thuật toán sẽ tính toán dòng tiền ròng (Net Flow) và đề xuất các giao dịch tối thiểu (ví dụ: chỉ cần A trả C).8 Đây là một tính năng cao cấp thường thấy trong các ứng dụng fintech hàng đầu như Splitwise.
- Logic 2: Dự Báo Dòng Tiền và Cảnh Báo Sớm (Cash Flow Forecasting):
  Dựa trên lịch sử chi tiêu và các khoản chi định kỳ (Recurring Expenses), hệ thống sẽ sử dụng phương pháp hồi quy tuyến tính đơn giản (Linear Regression) hoặc phân tích chuỗi thời gian (Time Series Analysis) để dự báo số dư vào cuối tháng. Nếu xu hướng chi tiêu hiện tại đe dọa vượt ngân sách, hệ thống sẽ gửi cảnh báo "Sớm" thay vì chờ đến khi đã vượt ngân sách.9
- Logic 3: Xử Lý Giao Dịch Định Kỳ (Recurring Transaction Engine):
  Hệ thống cần một cơ chế để tự động tạo ra các bản ghi giao dịch cho các khoản chi cố định (tiền nhà, Netflix, Internet) mà không cần người dùng nhập liệu thủ công. Điều này đòi hỏi việc thiết kế một Scheduler (như node-cron) chạy nền, kiểm tra cơ sở dữ liệu hàng ngày và kích hoạt logic tạo bản ghi dựa trên tần suất (frequency) và ngày bắt đầu.10

# 3. Kiến Trúc Hệ Thống và Thiết Kế Cơ Sở Dữ Liệu (Giai Đoạn 2: 41 - 90 Giờ)

Thiết kế hệ thống vững chắc là nền tảng cho việc phát triển nhanh chóng và ít lỗi. Giai đoạn này tập trung vào việc mô hình hóa dữ liệu và thiết kế API.

## 3.1 Thiết Kế Sơ Đồ Thực Thể Liên Kết (ERD - Entity Relationship Diagram)

Cơ sở dữ liệu sẽ được thiết kế ở dạng Chuẩn hóa 3 (3NF) để đảm bảo tính toàn vẹn dữ liệu. Dưới đây là các thực thể chính và mối quan hệ của chúng:

1. **Users (Người Dùng):** id, username, password_hash, email, role, created_at.
2. **Families (Gia Đình):** id, name, owner_id (FK -> Users).
3. **FamilyMembers (Thành Viên Gia Đình):** Bảng trung gian giải quyết mối quan hệ Many-to-Many giữa Users và Families. user_id, family_id, joined_at, status (active/pending).
4. **Wallets (Ví Tiền):** id, name, balance, currency, user_id (nullable - nếu là ví riêng), family_id (nullable - nếu là ví chung). Logic kiểm tra ràng buộc: Một ví phải thuộc về User HOẶC Family, không thể cả hai hoặc không có ai.[11]
5. **Categories (Danh Mục):** id, name, type (expense/income), parent_id (đệ quy cho danh mục con), icon.
6. **Transactions (Giao Dịch):** Thực thể trung tâm. id, amount, date, description, wallet_id, category_id, user_id (người thực hiện), related_transaction_id (cho các giao dịch chuyển tiền/trả nợ).
7. **Budgets (Ngân Sách):** id, amount_limit, start_date, end_date, category_id, family_id.
8. **RecurringPatterns (Mẫu Định Kỳ):** id, frequency (daily, weekly, monthly), next_run_date, amount, description.

Mối quan hệ giữa Transactions và Budgets là mối quan hệ tính toán, không phải khóa ngoại trực tiếp. Tổng amount của các Transactions trong khoảng thời gian và danh mục tương ứng sẽ được so sánh với amount_limit của Budgets.[11]

## 3.2 Thiết Kế API RESTful

API sẽ được thiết kế theo chuẩn REST, sử dụng các phương thức HTTP (GET, POST, PUT, DELETE) một cách ngữ nghĩa. Tài liệu API sẽ được tự động hóa bằng Swagger/OpenAPI.[1]

Một số Endpoint quan trọng:

- POST /api/auth/login: Xác thực và trả về JWT (Access Token + Refresh Token).
- GET /api/transactions: Lấy danh sách giao dịch, hỗ trợ query params để lọc (?startDate=...&endDate=...&categoryId=...) và phân trang (?page=1&limit=20).

- POST /api/families/:id/join: Gửi yêu cầu tham gia gia đình.
- GET /api/reports/monthly: Trả về dữ liệu đã tổng hợp cho biểu đồ (Aggregate Data). Việc tính toán tổng hợp (Aggregation) nên được thực hiện ở tầng Database (sử dụng GROUP BY trong SQL) thay vì tải toàn bộ dữ liệu về Node.js để xử lý, nhằm tối ưu hiệu năng.[2]

### 3.3 Chiến Lược Triển Khai Docker

Cấu trúc docker-compose.yml sẽ bao gồm 3 services chính:

1. **db:** Sử dụng image postgres:15-alpine. Cấu hình volume để persist dữ liệu (postgres_data:/var/lib/postgresql/data).
2. **api:** Build từ Dockerfile trong thư mục backend. Sử dụng nodemon trong môi trường dev để hot-reload.
3. **web:** Build từ Dockerfile trong thư mục frontend. Trong môi trường dev, container này chạy Vite server.

Việc sử dụng Docker đảm bảo môi trường phát triển đồng nhất giữa các thành viên trong nhóm (nếu có) và mô phỏng môi trường production chính xác, đáp ứng yêu cầu "Deployment" của đề tài.[13]

## 4. Lộ Trình Thực Thi Chi Tiết (300 Giờ)

Lộ trình được chia thành 7 Sprint, mỗi Sprint kéo dài khoảng 40-50 giờ, tập trung vào việc chuyển giao các tính năng có thể chạy được (Deliverables).

### Sprint 1: Khởi Tạo và Cấu Hình Môi Trường (Giờ 0 - 30)

- **Mục tiêu:** Thiết lập xong môi trường phát triển, Docker, Git flow và kết nối Database.
- **Chi tiết công việc:**
  1. Khởi tạo Git repo (GitHub/GitLab), thiết lập nhánh main và dev.[1]
  2. Cài đặt cấu trúc thư mục Monorepo (/client, /server, /docker).
  3. Viết docker-compose.yml để khởi chạy Postgres và Node.js server rỗng.
  4. Cài đặt các thư viện lõi: Express, Sequelize (Backend); React, TailwindCSS, Axios (Frontend).
  5. Thiết lập ESLint và Prettier để đảm bảo "Coding Convention" ngay từ đầu.

### Sprint 2: Backend Core - Auth & Database Migrations (Giờ 31 - 70)

- **Mục tiêu:** Hoàn thiện sơ đồ CSDL và hệ thống xác thực.
- **Chi tiết công việc:**
  1. Viết migration scripts cho tất cả các bảng (Users, Wallets, Transactions...).
  2. Viết Seeders sử dụng thư viện faker.js để tạo 1000 bản ghi giả lập (Transactions) phục vụ kiểm thử hiệu năng và phân trang.[1]

3. Implement module Auth: Register, Login, Refresh Token.
4. Mã hóa mật khẩu bằng bcrypt.
5. Tạo Middleware xác thực JWT (authMiddleware) và phân quyền (roleMiddleware).[7]

## Sprint 3: Backend Business Logic - CRUD & Nghiệp Vụ (Giờ 71 - 120)

- **Mục tiêu:** API hoàn chỉnh cho các chức năng chính.
- **Chi tiết công việc:**
  1. Implement CRUD cho Transactions, Wallets, Categories.
  2. Xử lý logic: Khi xóa Wallet, kiểm tra xem có Transactions liên quan không (Soft delete vs Hard delete).
  3. Xây dựng **Complex Logic 1 (Recurring Engine):** Viết cron job chạy mỗi nửa đêm, quét bảng RecurringPatterns, tạo transaction mới nếu đến hạn.[10]
  4. Xây dựng API Reports: Sử dụng Sequelize Aggregations để tính tổng thu/chi theo tháng, theo danh mục.

## Sprint 4: Frontend Core - Giao Diện & Tích Hợp (Giờ 121 - 170)

- **Mục tiêu:** Giao diện người dùng hoạt động được với dữ liệu thật từ API.
- **Chi tiết công việc:**
  1. Thiết kế Layout chung (Sidebar, Navbar) sử dụng TailwindCSS. Đảm bảo Responsive trên Mobile.[1]
  2. Implement trang Login/Register, lưu JWT vào LocalStorage hoặc HttpOnly Cookie.
  3. Xây dựng màn hình Dashboard: Hiển thị số dư, danh sách giao dịch gần đây.
  4. Tích hợp API: Sử dụng axios và interceptors để tự động đính kèm Token vào header và xử lý Refresh Token khi hết hạn.[16]

## Sprint 5: Frontend Advanced - Biểu Đồ & Logic Phức Tạp (Giờ 171 - 220)

- **Mục tiêu:** Trực quan hóa dữ liệu và tính năng nâng cao.
- **Chi tiết công việc:**
  1. Tích hợp Chart.js hoặc Recharts.[17] Vẽ biểu đồ tròn (cơ cấu chi tiêu) và biểu đồ cột (thu nhập vs chi tiêu).
  2. Xây dựng màn hình "Ngân Sách Gia Đình": Cho phép Family Manager thiết lập hạn mức.
  3. Implement **Complex Logic 2 (Debt Simplification UI):** Hiển thị đồ thị nợ trong nhóm. Ví dụ: A nợ B 100k, B nợ C 100k -> Hệ thống hiển thị đề xuất "A trả C 100k" (Backend cung cấp API tính toán, Frontend hiển thị kết quả).[8]
  4. Validation form: Sử dụng thư viện Formik và Yup để bắt lỗi nhập liệu ngay tại Frontend (ví dụ: Số tiền không được âm, ngày không được ở tương lai nếu là chi tiêu thực tế).[1]

### Sprint 6: Testing, Fix Bug & Optimization (Giờ 221 - 260)

- **Mục tiêu:** Đảm bảo chất lượng và độ ổn định.
- **Chi tiết công việc:**
  1. Viết Unit Test cho các hàm tính toán logic phức tạp ở Backend (sử dụng Jest).
  2. Integration Test cho các API quan trọng (Login, Create Transaction).
  3. Kiểm thử bảo mật: Thử nghiệm SQL Injection (đảm bảo ORM đã chặn), XSS (escape dữ liệu đầu ra).
  4. Tối ưu hiệu năng: Thêm Indexing cho các cột hay truy vấn (ví dụ: transaction_date, category_id) trong PostgreSQL.[11]

### Sprint 7: Hoàn Thiện, Tài Liệu & Đóng Gói (Giờ 261 - 300)

- **Mục tiêu:** Chuẩn bị sản phẩm để bàn giao và bảo vệ.
- **Chi tiết công việc:**
  1. Viết báo cáo thuyết minh (Technical Report): Vẽ lại sơ đồ ERD, Use Case, Architecture bằng công cụ như Draw.io hoặc StarUML.[18]
  2. Xuất tài liệu API từ Swagger.
  3. Quay Video Demo (5-7 phút) theo kịch bản: Đăng nhập -> Thêm chi tiêu -> Xem báo cáo -> Cảnh báo ngân sách -> Quản trị Admin.
  4. Soạn Slide thuyết trình.
  5. Kiểm tra lần cuối file docker-compose.yml để đảm bảo giảng viên có thể chạy dự án bằng 1 câu lệnh.[1]

# 5. Đi Sâu Vào Các Giải Pháp Kỹ Thuật (Technical Deep Dive)

## 5.1 Giải Thuật Tối Ưu Hóa Nợ (Debt Simplification)

Trong ngữ cảnh quản lý chi tiêu nhóm (Shared Wallet), vấn đề "Ai nợ ai bao nhiêu?" là bài toán phức tạp. Hệ thống sẽ áp dụng thuật toán tham lam (Greedy Algorithm) để giải quyết bài toán này.

- **Nguyên lý:** Tính toán số dư ròng (Net Balance) của mỗi người. Người có số dư dương là người cần được trả (Creditor), người có số dư âm là người nợ (Debtor).
- **Quy trình:**
  1. Tính Net Balance cho tất cả thành viên.
  2. Tách thành 2 danh sách: Debtors (số dư < 0) và Creditors (số dư > 0).
  3. Sắp xếp cả 2 danh sách theo giá trị tuyệt đối giảm dần.
  4. Lấy người nợ nhiều nhất trả cho người được nhận nhiều nhất. Số tiền giao dịch = Min(|Debit|, |Credit|).
  5. Cập nhật lại số dư và lặp lại cho đến khi danh sách rỗng.

- **Kết quả:** Giảm thiểu tối đa số lượng giao dịch chuyển tiền cần thực hiện, tối ưu hóa trải nghiệm người dùng.[19]

### 5.2 Bảo Mật Dữ Liệu Tài Chính

Vì ứng dụng xử lý dữ liệu tài chính nhạy cảm, các biện pháp bảo mật sau là bắt buộc:

- **Data Isolation (Cô lập dữ liệu):** Mặc dù sử dụng chung một Database, nhưng mọi câu truy vấn (Query) liên quan đến Transactions hoặc Wallets đều phải kèm theo điều kiện WHERE family_id = user.family_id hoặc user_id = current_user.id. Điều này ngăn chặn lỗi IDOR (Insecure Direct Object References), nơi một user có thể xem giao dịch của người khác bằng cách đổi ID trên URL.[7]
- **Sensitive Data Protection:** Các thông tin như mật khẩu phải được băm (hash) bằng thuật toán mạnh (Bcrypt/Argon2). API Key hoặc Secret Key (dùng cho JWT) không được hard-code trong mã nguồn mà phải nạp từ biến môi trường (.env), và file .env phải được thêm vào .gitignore để tránh lộ lọt trên GitHub.[1]

### 5.3 Trực Quan Hóa Dữ Liệu (Data Visualization)

Sử dụng thư viện Chart.js, hệ thống sẽ cung cấp các góc nhìn sâu sắc về tài chính:

- **Biểu đồ xu hướng (Line Chart):** Trục X là thời gian, Trục Y là số tiền. Giúp người dùng nhận diện "đỉnh" chi tiêu bất thường.
- **Biểu đồ phân bố (Doughnut Chart):** Hiển thị tỷ trọng các danh mục. Ví dụ: "Ăn uống" chiếm 40% tổng chi tiêu.
- **Kỹ thuật xử lý:** Để đảm bảo hiệu năng khi dữ liệu lên tới hàng nghìn bản ghi, Backend sẽ trả về dữ liệu đã được nhóm (grouped data) thay vì trả về raw data để Frontend tự tính toán. Điều này giảm tải cho trình duyệt và giảm băng thông mạng.[20]

## 6. Kết Luận và Hướng Phát Triển

Trong vòng 300 giờ làm việc, lộ trình này đảm bảo việc xây dựng hoàn chỉnh một Hệ thống Quản lý Chi tiêu Gia đình không chỉ đáp ứng đầy đủ các yêu cầu học thuật khắt khe (Docker, Microservices-ready architecture, Complex Logic) mà còn mang lại giá trị sử dụng thực tế cao.

Việc áp dụng các quy trình chuẩn công nghiệp như Migrations, Seeders, Dockerization và CI/CD simulation giúp sinh viên làm quen với môi trường làm việc chuyên nghiệp. Các tính năng nâng cao như thuật toán đơn giản hóa nợ và dự báo tài chính là điểm nhấn quan trọng, thể hiện tư duy thuật toán và khả năng giải quyết vấn đề sâu sắc của người phát triển.

**Hướng phát triển tương lai (Future Work):**

- Tích hợp AI/OCR (Optical Character Recognition) để tự động quét hóa đơn giấy và nhập

liệu.[21]
- Phát triển ứng dụng di động (Mobile App) sử dụng React Native, tận dụng lại hoàn toàn Backend API đã xây dựng.
- Tích hợp Open Banking API để đồng bộ giao dịch trực tiếp từ ngân hàng (nếu điều kiện pháp lý cho phép).

Báo cáo này đóng vai trò như một bản thiết kế kỹ thuật (Technical Blueprint), định hướng mọi quyết định trong quá trình phát triển, đảm bảo dự án về đích đúng hạn với chất lượng cao nhất.

---

## Bảng Phân Bổ Thời Gian Chi Tiết (Time Allocation Table)

| Giai Đoạn | Hoạt Động Chính | Thời Gian (Giờ) | Kết Quả Bàn Giao (Deliverables) |
|---|---|---|---|
| **1. Phân Tích** | Thu thập yêu cầu, định nghĩa Role, Chọn Tech Stack | 20 | Tài liệu đặc tả yêu cầu (SRS) |
| | Mô hình hóa Domain, Nghiên cứu thuật toán (Debt Algo) | 20 | Sơ đồ luồng dữ liệu, Logic Flowcharts |
| **2. Thiết Kế** | Thiết kế CSDL (ERD), Định nghĩa API Contract (Swagger) | 25 | ERD hoàn chỉnh, Tài liệu API |
| | Wireframing UI/UX (Figma/Sketch) | 25 | Mockups giao diện chi tiết |
| **3. Môi Trường** | Thiết lập Docker, Git Repo, CI/CD Linting | 20 | Môi trường Dev sẵn sàng (One-command setup) |

| | | | |
|---|---|---|---|
| | Viết DB Migration Scripts, Logic Seeding (Faker.js) | 10 | Database có dữ liệu mẫu (1000 records) |
| **4. Backend** | Module Auth (JWT, RBAC, Reset Password) | 25 | API Đăng nhập/Đăng ký bảo mật |
| | Core Transaction & Budget CRUD | 30 | API Thêm/Sửa/Xóa giao dịch |
| | Complex Logic: Recurring Payments & Debt Algorithm | 35 | Service xử lý nghiệp vụ phức tạp |
| **5. Frontend** | Khởi tạo dự án, Routing, Auth Pages | 20 | Giao diện Đăng nhập/Đăng ký |
| | Dashboard Components & Tích hợp Chart.js | 30 | Dashboard tương tác, Biểu đồ động |
| | Form quản lý giao dịch & Validation (Formik/Yup) | 30 | Giao diện nhập liệu có kiểm tra lỗi |
| **6. Đánh Bóng** | Tính năng Real-time (Socket.io) & Cảnh báo | 20 | Thông báo tức thời |
| | Export/Import Features & Tinh | 20 | Chức năng xuất Excel/PDF, UI mượt |

| | | | |
|---|---|---|---|
| | chỉnh UI | | mà |
| **7. Hoàn Thiện** | Unit & Integration Testing, Fix Bug | 30 | Bản build ổn định, không lỗi nghiêm trọng |
| | Viết báo cáo, Slide, Quay Video Demo | 20 | Gói sản phẩm nộp cuối kỳ |
| **TỔNG CỘNG** | | **400*** | *(Lưu ý: Đã bao gồm thời gian dự phòng và tự học)* |

*Lưu ý: Mặc dù yêu cầu đặt ra là 300 giờ, bảng phân bổ trên mở rộng lên 400 giờ để bao gồm cả thời gian tự nghiên cứu công nghệ mới (learning curve) và xử lý các vấn đề phát sinh không lường trước (buffer). Tuy nhiên, các đầu mục cốt lõi (Core Features) hoàn toàn có thể hoàn thành trong 300 giờ nếu tuân thủ nghiêm ngặt kế hoạch.*

**Nguồn trích dẫn**


1. Yeu cau de tai LTMT_copy.pdf
2. Building a Personal Finance Management App: Database Setup with PostgreSQL and Docker. | by George Zefkilis | Towards Data Engineering | Medium, truy cập vào tháng 1 10, 2026, https://medium.com/towards-data-engineering/building-a-personal-finance-management-app-database-setup-with-postgresql-and-docker-5075e283303e
3. How to Plan and Execute a Successful Web Application Development Project, truy cập vào tháng 1 10, 2026, https://dev.to/albert_ed/how-to-plan-and-execute-a-successful-web-application-development-project-1kkg
4. What Is Role-Based Access Control (RBAC)? Meaning & Examples - Fortinet, truy cập vào tháng 1 10, 2026, https://www.fortinet.com/resources/cyberglossary/role-based-access-control
5. Design and Implementation of a Family Expense Tracker Using MERN Stack - ijrpr, truy cập vào tháng 1 10, 2026, https://ijrpr.com/uploads/V6ISSUE8/IJRPR52224.pdf
6. Splitwise - App Store - Apple, truy cập vào tháng 1 10, 2026, https://apps.apple.com/us/app/splitwise/id458023433
7. How to Implement RBAC in an Express.js Application - Permit.io, truy cập vào

tháng 1 10, 2026, https://www.permit.io/blog/how-to-implement-rbac-in-an-expressjs-application
8. Algorithm Behind Splitwise's Debt Simplification Feature | by Mithun Mohan K | Medium, truy cập vào tháng 1 10, 2026, https://medium.com/@mithunmk93/algorithm-behind-splitwises-debt-simplification-feature-8ac485e97688
9. How to Automate Financial Forecasting: 8 Processes You Can Automate | NetSuite, truy cập vào tháng 1 10, 2026, https://www.netsuite.com/portal/resource/articles/financial-management/automate-financial-forecasting.shtml
10. System Design Pattern for Recurring Payments - GeeksforGeeks, truy cập vào tháng 1 10, 2026, https://www.geeksforgeeks.org/system-design/system-design-pattern-for-recurring-payments/
11. Designing a database schema for a budget tracker with Automigrate - bogoyavlensky.com, truy cập vào tháng 1 10, 2026, https://bogoyavlensky.com/blog/db-schema-for-budget-tracker-with-automigrate/
12. Daily expense tracker - Entity-relationship diagram example - Gleek.io, truy cập vào tháng 1 10, 2026, https://www.gleek.io/templates/expense-tracker-erd
13. Learn Docker in 3 hours. - DEV Community, truy cập vào tháng 1 10, 2026, https://dev.to/morning_redemption_3940af/learn-docker-in-3-hours-91p
14. Setting Up Docker: Node.js & PostgreSQL | FullStack Blog, truy cập vào tháng 1 10, 2026, https://www.fullstack.com/labs/resources/blog/set-up-docker-development-environment-that-matches-production
15. Migrations | Sequelize, truy cập vào tháng 1 10, 2026, https://sequelize.org/docs/v6/other-topics/migrations/
16. How to Build a Full-Stack React CRUD App with Node.js, Express and PostgreSQL: Step-by-Step Guide - DEV Community, truy cập vào tháng 1 10, 2026, https://dev.to/corbado/how-to-build-a-full-stack-react-crud-app-with-nodejs-express-and-postgresql-step-by-step-guide-323e
17. Struggling to Pick a Charting Library for React? These 10 Options Cover It All | HackerNoon, truy cập vào tháng 1 10, 2026, https://hackernoon.com/struggling-to-pick-a-charting-library-for-react-these-10-options-cover-it-all
18. Guide to Technical Report Writing - University of Sussex, truy cập vào tháng 1 10, 2026, https://www.sussex.ac.uk/ei/internal/forstudents/engineeringdesign/studyguides/techreportwriting
19. Algorithm to share/settle expenses among a group - Stack Overflow, truy cập vào tháng 1 10, 2026, https://stackoverflow.com/questions/974922/algorithm-to-share-settle-expenses-among-a-group
20. ReactJS Development for Real-Time Analytics Dashboards - Makers Den, truy cập vào tháng 1 10, 2026, https://makersden.io/blog/reactjs-dev-for-real-time-analytics-dashboards

21. (PDF) EXPENSE TRACKER APPLICATION - ResearchGate, truy cập vào tháng 1 10, 2026, https://www.researchgate.net/publication/381249579_EXPENSE_TRACKER_APPLICATION