TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP. HỒ CHÍ MINH

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ ĐIỆN ĐIỆN TỬ**

ĐỒ ÁN

**THỰC TẬP TỐT NGHIỆP**

**TÌM HIỂU FRAMEWORK LARAVEL, VUE.JS VÀ XÂY DỰNG ỨNG DỤNG MINH HỌA - TASK MANAGER**

**GVHD:** Trần Đức Doanh

**SVTH:** Huỳnh Minh Nhựt - 2251120371

# **LỜI NÓI ĐẦU**

Trong những năm gần đây, sự phát triển nhanh chóng của công nghệ thông tin và đặc biệt là các ứng dụng web đã tạo ra nhiều thay đổi sâu sắc trong đời sống cũng như hoạt động sản xuất, kinh doanh. Việc nắm vững và áp dụng các framework hiện đại trong lập trình web không chỉ giúp rút ngắn thời gian phát triển, tăng tính hiệu quả và ổn định, mà còn mở ra cơ hội tiếp cận với những công nghệ tiên tiến, phù hợp với nhu cầu thực tế của xã hội.

Xuất phát từ thực tiễn đó, em đã chọn thực hiện đề tài “Tìm hiểu framework Laravel, Vue.js và xây dựng ứng dụng minh họa – Task Manager”. Thông qua đề tài, em mong muốn tìm hiểu chuyên sâu về hai công nghệ phổ biến là Laravel (đảm nhận vai trò backend) và Vue.js (xây dựng giao diện frontend), đồng thời vận dụng để phát triển một ứng dụng quản lý công việc đơn giản nhưng đầy đủ các chức năng cơ bản, có khả năng triển khai trên môi trường thực tế.

Trong quá trình thực hiện, đồ án không chỉ giúp em củng cố và mở rộng kiến thức đã học trên giảng đường, mà còn rèn luyện thêm kỹ năng nghiên cứu, tư duy hệ thống, cũng như kỹ năng triển khai và tích hợp các công nghệ trong thực tiễn. Tuy nhiên, do hạn chế về thời gian, kinh nghiệm cũng như nguồn lực, đồ án chắc chắn không tránh khỏi những thiếu sót. Rất mong nhận được những ý kiến đóng góp quý báu của quý thầy cô và bạn bè để em có thể hoàn thiện hơn trong tương lai.

# **LỜI CẢM ƠN**

Trước tiên, em xin bày tỏ lòng biết ơn sâu sắc đến quý thầy cô Viện công nghệ thông tin điện điện tử, Trường đại học Giao thông vận tải thành phố Hồ Chí Minh đã tận tình giảng dạy, truyền đạt kiến thức và kỹ năng trong suốt quá trình học tập, tạo nền tảng vững chắc để em có thể thực hiện và hoàn thành đồ án này.

Đặc biệt, em xin gửi lời cảm ơn chân thành đến thầy Trần Đức Doanh - giảng viên hướng dẫn, người đã tận tình định hướng, chỉ bảo và đưa ra nhiều nhận xét quý báu, giúp em khắc phục khó khăn, bổ sung kiến thức và từng bước hoàn thiện đề tài.

Em cũng xin trân trọng cảm ơn gia đình đã luôn là điểm tựa tinh thần, động viên và tạo điều kiện thuận lợi trong suốt thời gian học tập và nghiên cứu. Đồng thời, em cảm ơn các anh chị, bạn bè, đồng môn đã đồng hành, chia sẻ tài liệu, kinh nghiệm và góp ý chân thành, giúp em hoàn thiện đồ án này tốt hơn.

Mặc dù đã có nhiều cố gắng, song với kiến thức và kinh nghiệm còn hạn chế, đồ án chắc chắn không tránh khỏi những sai sót. Em rất mong tiếp tục nhận được sự chỉ dẫn, đóng góp và phê bình của quý thầy cô và các bạn để em có thể rút ra nhiều bài học kinh nghiệm, phục vụ cho quá trình học tập và công việc sau này.

Xin trân trọng cảm ơn!

# **TÓM TẮT ĐỒ ÁN**

Trong bối cảnh phát triển mạnh mẽ của các ứng dụng web hiện đại, việc kết hợp giữa một framework backend vững chắc và một framework frontend linh hoạt đóng vai trò quan trọng trong việc xây dựng hệ thống hiệu quả, dễ mở rộng và thân thiện với người dùng. Đề tài “Tìm hiểu framework Laravel, Vue.js và xây dựng ứng dụng minh họa - Task Manager” được thực hiện với mục tiêu nghiên cứu hai công nghệ phổ biến hiện nay là Laravel (backend) và Vue.js (frontend), đồng thời ứng dụng vào việc xây dựng một hệ thống quản lý công việc (Task Manager) đơn giản nhưng đầy đủ chức năng.

Ứng dụng được phát triển theo kiến trúc client–server, trong đó Laravel đảm nhận vai trò backend với cơ chế RESTful API, quản lý dữ liệu qua Eloquent ORM, đồng thời triển khai authentication bằng Laravel Sanctum nhằm đảm bảo an toàn khi giao tiếp. Về phía frontend, Vue.js được sử dụng để xây dựng giao diện dạng Single Page Application (SPA), mang lại trải nghiệm người dùng mượt mà, hỗ trợ reactivity và responsive trên cả desktop lẫn thiết bị di động.

Kết quả của đồ án cho thấy toàn bộ các chức năng cốt lõi đều đã được hiện thực, bao gồm:

* Quản lý người dùng (đăng ký, đăng nhập, đăng xuất, xác thực token).
* Quản lý công việc (CRUD task, validation form, giao diện trực quan).
* Triển khai hệ thống trên môi trường cloud với Render (backend + database PostgreSQL) và GitHub Pages (frontend).

Ngoài việc đạt được mục tiêu ban đầu, đề tài còn bổ sung thêm một số cải tiến như seed dữ liệu mẫu, interceptor trong Axios, và thiết kế code backend/frontend theo hướng modular để thuận tiện cho việc mở rộng sau này.

Bên cạnh các kết quả đạt được, đồ án cũng chỉ ra một số hạn chế: chưa có phân quyền nâng cao (admin/super-admin), thiếu tính năng real-time notification, reminder, hoặc file attachment. Ngoài ra, kiểm thử tự động vẫn còn hạn chế, chủ yếu mới ở mức thủ công.

Qua quá trình nghiên cứu và triển khai, em đã nắm vững nhiều khái niệm quan trọng trong Laravel và Vue.js, rèn luyện kỹ năng tích hợp backend-frontend, cũng như làm quen với quy trình triển khai ứng dụng thực tế trên nền tảng cloud. Đây sẽ là nền tảng quan trọng để phát triển các ứng dụng web phức tạp hơn trong tương lai, chẳng hạn như mở rộng theo kiến trúc microservices, tích hợp Progressive Web App (PWA), hoặc ứng dụng machine learning trong gợi ý sắp xếp công việc.

**MỤC LỤC**

[LỜI NÓI ĐẦU i](#_Toc208597924)

[LỜI CẢM ƠN ii](#_Toc208597925)

[TÓM TẮT ĐỒ ÁN iii](#_Toc208597926)

[MỤC LỤC v](#_Toc208597927)

[DANH MỤC CÁC TỪ VIẾT TẮT ix](#_Toc208597928)

[DANH MỤC CÁC HÌNH ẢNH x](#_Toc208597929)

[CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI 1](#_Toc208597930)

[**1.1. Tên đề tài** 1](#_Toc208597931)

[**1.2. Lý do chọn đề tài (Bối cảnh và động cơ)** 1](#_Toc208597932)

[**1.3. Mục tiêu nghiên cứu** 1](#_Toc208597933)

[*1.3.1. Mục tiêu chung* 1](#_Toc208597934)

[*1.3.2. Mục tiêu cụ thể* 2](#_Toc208597935)

[**1.4. Phạm vi nghiên cứu và giới hạn đề tài** 2](#_Toc208597936)

[*1.4.1. Phạm vi* 2](#_Toc208597937)

[*1.4.2. Giới hạn* 2](#_Toc208597938)

[**1.5. Phương pháp nghiên cứu và quy trình thực hiện** 3](#_Toc208597939)

[**1.6. Công cụ và môi trường kỹ thuật** 3](#_Toc208597940)

[**1.7. Đóng góp và sản phẩm kỳ vọng** 4](#_Toc208597941)

[**1.8. Kết cấu báo cáo** 4](#_Toc208597942)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 5](#_Toc208597943)

[**2.1. Tổng quan về hệ quản lý công việc** 5](#_Toc208597944)

[**2.2. Các khái niệm cơ bản** 5](#_Toc208597945)

[**2.3. Giới thiệu Laravel** 7](#_Toc208597946)

[*2.3.1. Lịch sử & phiên bản* 7](#_Toc208597947)

[*2.3.2. Kiến trúc MVC trong Laravel* 7](#_Toc208597948)

[*2.3.3. Eloquent ORM, Migration, Seeder, Factory* 8](#_Toc208597949)

[*2.3.4. Middleware, Service Providers, Form Requests, Events/Listeners* 8](#_Toc208597950)

[*2.3.5. Laravel Sanctum (cơ chế token)* 8](#_Toc208597951)

[**2.4. Giới thiệu Vue.js** 9](#_Toc208597952)

[*2.4.1. Lịch sử & phiên bản (Vue 3, Composition API)* 9](#_Toc208597953)

[*2.4.2. Kiến trúc component, reactive data, props & emits* 9](#_Toc208597954)

[*2.4.3. Vue Router và quản lý state (composition + localStorage)* 9](#_Toc208597955)

[*2.4.4. Axios và giao tiếp với API* 10](#_Toc208597956)

[**2.5. So sánh các lựa chọn công nghệ** 10](#_Toc208597957)

[*2.5.1. Ưu/nhược của Laravel so với các framework backend khác* 10](#_Toc208597958)

[*2.5.2. Ưu/nhược của Vue so với React và Angular* 10](#_Toc208597959)

[CHƯƠNG 3: PHÂN TÍCH HỆ THỐNG 11](#_Toc208597960)

[**3.1. Yêu cầu chức năng (Functional Requirements)** 11](#_Toc208597961)

[*3.1.1. Quản lý người dùng* 11](#_Toc208597962)

[*3.1.2. Quản lý công việc (Tasks)* 11](#_Toc208597963)

[*3.1.3. Giao diện & trải nghiệm người dùng* 12](#_Toc208597964)

[**3.2. Yêu cầu phi chức năng (Non-functional Requirements)** 12](#_Toc208597965)

[*3.2.1. Bảo mật* 12](#_Toc208597966)

[*3.2.2. Hiệu năng* 13](#_Toc208597967)

[*3.2.3. Khả năng mở rộng* 13](#_Toc208597968)

[*3.2.4. Khả năng bảo trì* 13](#_Toc208597969)

[*3.2.5. Tính sẵn sàng & sao lưu* 13](#_Toc208597970)

[**3.3. Phân tích người dùng** 14](#_Toc208597971)

[**3.4. Kịch bản sử dụng (Use cases)** 14](#_Toc208597972)

[**3.5. Luồng tương tác chính** 15](#_Toc208597973)

[*3.5.1. Luồng Authentication* 15](#_Toc208597974)

[*3.5.2. Luồng Quản lý công việc (CRUD)* 16](#_Toc208597975)

[**3.6. Ràng buộc và giả định** 16](#_Toc208597976)

[CHƯƠNG 4: THIẾT KẾ HỆ THỐNG 17](#_Toc208597977)

[**4.1. Kiến trúc tổng thể (System Architecture)** 17](#_Toc208597978)

[*4.1.1. Sơ đồ kiến trúc* 17](#_Toc208597979)

[*4.1.2. Mô tả các thành phần chính* 17](#_Toc208597980)

[**4.2. Thiết kế cơ sở dữ liệu (Database Design)** 18](#_Toc208597981)

[*4.2.1. Sơ đồ ERD* 18](#_Toc208597982)

[*4.2.2. Mô tả bảng* 22](#_Toc208597983)

[**4.3. Thiết kế API (API Specification)** 23](#_Toc208597984)

[*4.3.1. Danh sách endpoint* 23](#_Toc208597985)

[*4.3.2. Request/Response mẫu* 23](#_Toc208597986)

[*4.3.3. Bảo mật API* 24](#_Toc208597987)

[**4.4. Thiết kế phía Backend (Laravel)** 25](#_Toc208597988)

[*4.4.1. Luồng xử lý request* 25](#_Toc208597989)

[*4.4.2. Middleware & Validation* 25](#_Toc208597990)

[*4.4.3. Migration, Seeder, Factory* 26](#_Toc208597991)

[**4.5. Thiết kế phía Frontend (Vue.js)** 27](#_Toc208597992)

[*4.5.2. Wireframes / Mockups* 27](#_Toc208597993)

[*4.5.3. Luồng xác thực* 27](#_Toc208597994)

[*4.5.4. Quản lý trạng thái & giao tiếp API* 28](#_Toc208597995)

[**4.6. Thiết kế giao diện người dùng (UI/UX)** 28](#_Toc208597996)

[*4.6.1. Nguyên tắc UI/UX* 28](#_Toc208597997)

[*4.6.2. Palette, Typography, Component* 29](#_Toc208597998)

[**4.7. Sequence Diagram** 30](#_Toc208597999)

[CHƯƠNG 5: TRIỂN KHAI VÀ THỰC HIỆN 32](#_Toc208598000)

[**5.1. Phương pháp tổ chức và quản lý mã nguồn** 32](#_Toc208598001)

[*5.1.1. Chiến lược tổ chức repository* 32](#_Toc208598002)

[*5.1.2. Tiêu chuẩn hóa môi trường phát triển* 32](#_Toc208598003)

[**5.2. Quy trình triển khai Backend (Laravel)** 32](#_Toc208598004)

[*5.2.1. Phương pháp khởi tạo và cấu hình* 32](#_Toc208598005)

[*5.2.2. Chiến lược quản lý cơ sở dữ liệu* 33](#_Toc208598006)

[*5.2.3. Kiến trúc API và bảo mật* 33](#_Toc208598007)

[*5.2.4. Cơ chế bảo mật và middleware* 34](#_Toc208598008)

[*5.2.5. Validation & xử lý lỗi* 34](#_Toc208598009)

[*5.2.6. Phương pháp containerization và deployment* 34](#_Toc208598010)

[**5.3. Cài đặt & cấu hình frontend (Vue.js)** 34](#_Toc208598011)

[*5.3.1. Kiến trúc Single Page Application* 35](#_Toc208598012)

[*5.3.2. Chiến lược giao tiếp API* 35](#_Toc208598013)

[*5.3.3. Tổ chức component và state management* 35](#_Toc208598014)

[*5.3.4. Route protection và quản lý session* 35](#_Toc208598015)

[*5.3.5. Quy trình kiểm thử và đảm bảo chất lượng* 36](#_Toc208598016)

[*5.3.7. Triển khai GitHub Pages* 36](#_Toc208598017)

[**5.4. Tích hợp và automation** 36](#_Toc208598018)

[*5.4.1. CI/CD pipeline* 36](#_Toc208598019)

[*5.4.2. Quản lý cấu hình môi trường* 36](#_Toc208598020)

[CHƯƠNG 6: ĐÁNH GIÁ KẾT QUẢ VÀ THẢO LUẬN 37](#_Toc208598021)

[**6.1. So sánh kết quả với mục tiêu ban đầu** 37](#_Toc208598022)

[**6.2. Đánh giá theo tiêu chí** 38](#_Toc208598023)

[**6.3. Hạn chế của hệ thống (Known limitations)** 39](#_Toc208598024)

[**6.4. Những bài học rút ra khi học Laravel & Vue** 39](#_Toc208598025)

[*6.4.1. Những khái niệm khó / cần thời gian để nắm vững* 39](#_Toc208598026)

[*6.4.2. Những điểm mạnh của Laravel* 39](#_Toc208598027)

[*6.4.3. Những điểm mạnh của Vue* 40](#_Toc208598028)

[**6.5. Gợi ý cải tiến kỹ thuật & hướng nghiên cứu tiếp theo** 40](#_Toc208598029)

[CHƯƠNG 7: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 42](#_Toc208598030)

[**7.1. Kết luận chung** 42](#_Toc208598031)

[**7.2. Đóng góp của đề tài** 42](#_Toc208598032)

[**7.3. Hạn chế còn tồn tại** 43](#_Toc208598033)

[**7.4. Hướng phát triển tương lai** 43](#_Toc208598034)

[**7.5. Tổng kết** 44](#_Toc208598035)

[TÀI LIỆU THAM KHẢO 45](#_Toc208598036)

[PHỤ LỤC 46](#_Toc208598037)

# **DANH MỤC CÁC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| Viết tắt | Thuật ngữ tiếng anh | Thuật ngữ tiếng việt |
|  |  |  |

# **DANH MỤC CÁC HÌNH ẢNH**

# **CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI**

## **1.1. Tên đề tài**

“Tìm hiểu framework Laravel, Vue.js và xây dựng ứng dụng minh họa - Task Manager”

## **1.2. Lý do chọn đề tài (Bối cảnh và động cơ)**

Trong kỷ nguyên phát triển web hiện đại, ứng dụng web một trang (SPA) và kiến trúc client–server đang trở thành xu hướng phổ biến do khả năng tương tác cao, trải nghiệm mượt mà và dễ mở rộng. Laravel (một PHP framework hiện đại) và Vue.js (framework frontend nhẹ, linh hoạt) là hai công nghệ nổi bật, được cộng đồng rộng rãi chấp nhận vì sự hiệu quả, tài liệu phong phú và hệ sinh thái hỗ trợ.

Việc kết hợp Laravel cho backend và Vue.js cho frontend mang lại nhiều lợi ích thực tiễn: Laravel cung cấp các công cụ mạnh mẽ cho quản lý dữ liệu, xác thực, migration, còn Vue.js giúp phát triển giao diện người dùng phản ứng nhanh với lượng code tối thiểu. Đề tài này chọn xây dựng một Task Manager làm ứng dụng minh họa vì tính thực tế (phù hợp nhu cầu cá nhân/văn phòng), phạm vi vừa đủ để triển khai các khái niệm cốt lõi (CRUD, auth, API, state management), đồng thời có thể mở rộng cho các tính năng nâng cao về sau (notification, sharing, PWA).

Mục tiêu học tập là vừa nắm vững lý thuyết, vừa ứng dụng thực tế—từ đó rèn luyện kỹ năng thiết kế hệ thống, viết API, quản lý trạng thái frontend, và triển khai ứng dụng trên môi trường cloud.

## **1.3. Mục tiêu nghiên cứu**

### *1.3.1. Mục tiêu chung*

Xây dựng và triển khai một ứng dụng quản lý công việc Task Manager minh họa, đồng thời tìm hiểu, so sánh và áp dụng các kỹ thuật tiêu chuẩn khi phát triển web bằng Laravel (backend) và Vue.js (frontend).

### *1.3.2. Mục tiêu cụ thể*

Nghiên cứu các khái niệm và cơ chế cơ bản của Laravel (MVC, Eloquent, migration, middleware, Sanctum).

Nghiên cứu các khái niệm của Vue.js (component, reactivity, Composition API, Vue Router) và cách tích hợp với REST API.

Thiết kế và triển khai RESTful API bảo mật cho quản lý người dùng và task.

Xây dựng SPA frontend với các chức năng: đăng ký, đăng nhập, CRUD task, lọc/sắp xếp và responsive UI.

Triển khai hệ thống thực tế (backend lên Render, frontend lên GitHub Pages) và viết tài liệu hướng dẫn cài đặt, sử dụng.

Viết test cơ bản cho backend (unit/feature) và thực hiện kiểm thử chức năng cho frontend.

## **1.4. Phạm vi nghiên cứu và giới hạn đề tài**

### *1.4.1. Phạm vi*

Ứng dụng minh họa là Task Manager dành cho người dùng cá nhân (không tập trung vào quản lý nhóm phức tạp).

Backend phát triển bằng Laravel (REST API).

Frontend phát triển bằng Vue.js (SPA), sử dụng CSS cho UI.

Triển khai: backend trên Render, database PostgreSQL; frontend deploy tĩnh trên GitHub Pages.

### *1.4.2. Giới hạn*

Chức năng tập trung vào CRUD task, xác thực đơn giản bằng token (Sanctum). Các tính năng nâng cao như phân quyền phức tạp (RBAC), nhắc việc tự động, real-time notification, upload file sẽ chỉ dừng ở mức đề xuất/định hướng và chưa được triển khai đầy đủ trong phạm vi báo cáo này.

Kiểm thử tự động chỉ ở mức cơ bản (feature tests cho backend); e2e hoặc test frontend tự động là tùy chọn, có thể mở rộng sau.

## **1.5. Phương pháp nghiên cứu và quy trình thực hiện**

Nghiên cứu tài liệu: tìm hiểu tài liệu chính thức của Laravel và Vue.js, tham khảo các tutorial, best-practice, và bài viết chuyên sâu.

Thực nghiệm (experimental development): triển khai thực tế các module nhỏ, đo lường và điều chỉnh kiến trúc.

Phân tích - Thiết kế - Triển khai - Kiểm thử: áp dụng quy trình phát triển phần mềm cơ bản: phân tích yêu cầu, thiết kế kiến trúc và cơ sở dữ liệu, cài đặt, kiểm thử và triển khai.

So sánh & đánh giá: đối chiếu kết quả với mục tiêu, nhận diện hạn chế và đề xuất cải tiến.

## **1.6. Công cụ và môi trường kỹ thuật**

Ngôn ngữ & framework: PHP 8.2, Laravel 12, JavaScript, Vue 3 (Composition API).

Build & package: Composer, Node.js (Vite), NPM.

Database: PostgreSQL.

Triển khai: Render.com (backend + DB), GitHub Pages (frontend).

Hệ thống quản lý mã nguồn: Git, cấu trúc repo chính có submodules cho backend & frontend.

Công cụ hỗ trợ: Visual Studio Code.

## **1.7. Đóng góp và sản phẩm kỳ vọng**

Sản phẩm phần mềm: mã nguồn backend và frontend, ứng dụng Task Manager chạy được trên cloud.

Tài liệu kỹ thuật: báo cáo hoàn chỉnh, file hướng dẫn cài đặt (README), và phụ lục chứa link GitHub

Bài học và kinh nghiệm thực tế: tài liệu tổng hợp các khó khăn, cách khắc phục và gợi ý cải tiến cho phát triển ứng dụng tương tự.

## **1.8. Kết cấu báo cáo**

Báo cáo được tổ chức thành các chương như sau:

* Chương 1 - Giới thiệu: nêu đề tài, mục tiêu, phương pháp và tổ chức báo cáo.
* Chương 2 - Cơ sở lý thuyết: trình bày khái niệm nền tảng.
* Chương 3 - Phân tích hệ thống: yêu cầu chức năng, kịch bản sử dụng, luồng nghiệp vụ.
* Chương 4 - Thiết kế hệ thống: kiến trúc tổng thể, ERD, API spec, UI/UX.
* Chương 5 - Triển khai và thực hiện: môi trường, cài đặt backend/frontend, deploy, test.
* Chương 6 - Đánh giá kết quả và thảo luận: so sánh mục tiêu với kết quả, hạn chế, bài học.
* Chương 7 - Kết luận và hướng phát triển: tổng kết và đề xuất mở rộng.

# **CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

## **2.1. Tổng quan về hệ quản lý công việc**

Hệ thống quản lý công việc (Task Manager) là một loại ứng dụng phần mềm hỗ trợ người dùng trong việc tổ chức, theo dõi và quản lý các nhiệm vụ trong cả công việc lẫn đời sống hằng ngày. Những ứng dụng này ra đời nhằm giải quyết tình trạng quá tải thông tin và sự phân tán công việc, giúp cá nhân và nhóm làm việc hiệu quả hơn.

Các tính năng cơ bản thường được tích hợp trong một Task Manager bao gồm: khả năng tạo mới, chỉnh sửa và xóa nhiệm vụ; đặt thời hạn hoàn thành (deadline) và trạng thái thực hiện (hoàn thành hoặc chưa hoàn thành); phân loại hoặc gán nhiệm vụ cho các dự án, nhóm hoặc người dùng cụ thể; cùng với đó là khả năng đồng bộ dữ liệu trên nhiều thiết bị và gửi thông báo nhắc nhở ở các hệ thống nâng cao.

Trong bối cảnh làm việc hiện đại, nơi mà mỗi cá nhân thường phải xử lý nhiều loại công việc khác nhau và các nhóm làm việc phân tán về không gian, Task Manager đóng vai trò quan trọng trên hai phương diện: quản lý cá nhân (Personal Productivity) và quản lý nhóm (Team Collaboration). Một số sản phẩm thương mại nổi bật như Trello, Todoist, Asana hay Microsoft To Do đã chứng minh hiệu quả của loại ứng dụng này. Tuy nhiên, hầu hết các giải pháp hiện có thường mang tính phức tạp hoặc yêu cầu trả phí để sử dụng đầy đủ tính năng. Vì vậy, việc xây dựng một ứng dụng quản lý công việc cá nhân với ưu điểm nhẹ nhàng, dễ sử dụng, miễn phí là một nhu cầu thực tiễn. Đồng thời, đây cũng là một môi trường học tập và thực hành lý tưởng để áp dụng các framework web hiện đại.

## **2.2. Các khái niệm cơ bản**

Để xây dựng một hệ thống quản lý công việc dựa trên mô hình web hiện đại, cần nắm rõ một số khái niệm nền tảng sau:

REST API (Representational State Transfer – Application Programming Interface): Đây là một kiểu thiết kế API phổ biến dựa trên các nguyên tắc REST, trong đó các phương thức HTTP (GET, POST, PUT, DELETE) được sử dụng để thao tác dữ liệu. Dữ liệu thường được trả về ở định dạng JSON, dễ dàng tích hợp với các ứng dụng frontend như Vue.js hoặc React. Nhờ sự đơn giản và tính chuẩn hóa, REST API hiện là chuẩn giao tiếp chính trong phát triển ứng dụng web.

JWT (JSON Web Token) và Laravel Sanctum: JWT là một cơ chế xác thực người dùng dựa trên token đã được ký số, thường được áp dụng trong các hệ thống phân tán hoặc microservices, nơi cần trao đổi thông tin bảo mật giữa nhiều dịch vụ. Trong khi đó, Laravel Sanctum là một giải pháp xác thực nhẹ hơn, được thiết kế đặc biệt cho SPA (Single Page Application) và ứng dụng di động. Sanctum cho phép lưu trữ token trong cơ sở dữ liệu và dễ dàng tích hợp với các ứng dụng viết bằng Laravel. Với đề tài này, Sanctum được lựa chọn vì phù hợp với mô hình SPA + API backend đơn giản.

SPA (Single Page Application): Đây là loại ứng dụng web chỉ tải một trang HTML duy nhất và cập nhật nội dung động bằng JavaScript khi người dùng tương tác. SPA giúp trải nghiệm trở nên mượt mà, không cần tải lại toàn bộ trang khi chuyển hướng giữa các chức năng. Các framework phổ biến để phát triển SPA gồm Vue.js, React và Angular.

CORS (Cross-Origin Resource Sharing): Đây là một cơ chế bảo mật của trình duyệt, quy định việc cho phép hoặc chặn các request từ một domain khác đến API. Ví dụ, trong đề tài này, frontend được triển khai trên GitHub Pages, trong khi backend chạy trên Render.com. Do đó, cần bật CORS trên backend Laravel để frontend có thể gửi và nhận dữ liệu hợp lệ.

CRUD (Create, Read, Update, Delete): Đây là bốn thao tác cơ bản trong quản lý dữ liệu và được coi là yêu cầu cốt lõi trong hầu hết các ứng dụng web. Cụ thể, Create cho phép thêm mới nhiệm vụ, Read dùng để truy xuất và hiển thị danh sách nhiệm vụ, Update dùng để chỉnh sửa thông tin hoặc trạng thái công việc, và Delete cho phép xóa bỏ nhiệm vụ không còn cần thiết. Trong một hệ thống Task Manager, việc triển khai CRUD chính là nền tảng để đảm bảo khả năng quản lý nhiệm vụ của người dùng.

## **2.3. Giới thiệu Laravel**

### *2.3.1. Lịch sử & phiên bản*

Laravel được phát triển lần đầu vào năm 2011 bởi Taylor Otwell, nhằm mang đến một framework PHP có cú pháp gọn gàng, hiện đại, thay thế cho những hạn chế của CodeIgniter. Trải qua nhiều phiên bản, Laravel đã trở thành một trong những framework PHP phổ biến nhất trên thế giới, nhờ vào cấu trúc MVC rõ ràng, cộng đồng lớn mạnh, và hệ sinh thái phong phú. Laravel tích hợp sẵn nhiều tính năng quan trọng như routing, migration và authentication, giúp lập trình viên tiết kiệm thời gian và công sức.

Trong phạm vi đề tài, phiên bản Laravel 12 được lựa chọn vì sự ổn định, tương thích tốt với PHP 8+, đồng thời được cam kết hỗ trợ lâu dài từ cộng đồng và nhà phát triển.

### *2.3.2. Kiến trúc MVC trong Laravel*

Laravel tuân thủ mô hình kiến trúc MVC (Model – View – Controller). Trong đó:

* Model chịu trách nhiệm quản lý dữ liệu và logic nghiệp vụ, thường kết nối với cơ sở dữ liệu thông qua Eloquent ORM.
* View là lớp giao diện hiển thị. Trong dự án này, View chủ yếu phục vụ phản hồi API dưới dạng JSON cho frontend.
* Controller đóng vai trò trung gian, tiếp nhận request từ người dùng (frontend), gọi Model xử lý, và trả về response thích hợp.

### *2.3.3. Eloquent ORM, Migration, Seeder, Factory*

Eloquent ORM: Cho phép thao tác dữ liệu bằng cú pháp hướng đối tượng thay vì SQL thuần, giúp code dễ hiểu và dễ bảo trì.

Migration: Cung cấp cơ chế quản lý phiên bản cơ sở dữ liệu, hỗ trợ việc tạo mới, thay đổi hoặc rollback bảng dữ liệu một cách nhất quán.

Seeder: Dùng để khởi tạo dữ liệu mẫu cho database, thuận tiện cho việc thử nghiệm.

Factory: Cho phép sinh dữ liệu giả (fake data) nhằm phục vụ quá trình kiểm thử ứng dụng.

### *2.3.4. Middleware, Service Providers, Form Requests, Events/Listeners*

Middleware: Các lớp trung gian xử lý request, ví dụ kiểm tra token trước khi cho phép truy cập một route bảo mật.

Service Providers: Là nơi đăng ký các dịch vụ vào service container của Laravel, giúp mở rộng ứng dụng linh hoạt.

Form Requests: Cung cấp lớp xác thực dữ liệu đầu vào, giúp đảm bảo request hợp lệ trước khi xử lý.

Events/Listeners: Cơ chế phát và lắng nghe sự kiện, hỗ trợ xử lý bất đồng bộ, ví dụ tự động gửi email khi người dùng đăng ký thành công.

### *2.3.5. Laravel Sanctum (cơ chế token)*

Sanctum cung cấp một cơ chế xác thực đơn giản và hiệu quả cho SPA. Quy trình hoạt động như sau:

* Người dùng đăng nhập vào hệ thống, Laravel sẽ tạo ra một token.
* Token này được lưu trong bảng personal\_access\_tokens.
* Frontend lưu token vào LocalStorage của trình duyệt và gửi kèm trong header Authorization dưới dạng Bearer Token khi gọi API.
* Backend kiểm tra token và quyết định cho phép hoặc từ chối truy cập tài nguyên.

Cơ chế này đảm bảo tính bảo mật đồng thời giữ cho hệ thống nhẹ nhàng, dễ triển khai.

## **2.4. Giới thiệu Vue.js**

### *2.4.1. Lịch sử & phiên bản (Vue 3, Composition API)*

Vue.js được phát triển bởi Evan You vào năm 2014 và nhanh chóng trở thành một trong những framework frontend phổ biến nhất nhờ tính linh hoạt và cú pháp dễ học. Phiên bản mới nhất, Vue 3, giới thiệu Composition API – một cách viết code rõ ràng hơn, dễ dàng tái sử dụng logic giữa các component.

### *2.4.2. Kiến trúc component, reactive data, props & emits*

Vue.js xây dựng ứng dụng theo kiến trúc component. Mỗi component là một khối giao diện độc lập, có khả năng tái sử dụng và dễ dàng bảo trì. Dữ liệu trong Vue mang tính reactive, tức là khi dữ liệu thay đổi thì giao diện sẽ tự động được cập nhật. Việc truyền dữ liệu giữa các component được thực hiện thông qua props (từ cha xuống con) và emits (từ con lên cha).

### *2.4.3. Vue Router và quản lý state (composition + localStorage)*

Vue Router là thư viện chính thức để quản lý điều hướng trong SPA, cho phép xây dựng nhiều trang ảo (login, tasks, profile…) mà không cần tải lại toàn bộ trang web. Ngoài ra, việc quản lý state (trạng thái ứng dụng) được thực hiện thông qua reactive kết hợp với LocalStorage, giúp lưu giữ thông tin người dùng và token ngay cả khi tải lại trang.

### *2.4.4. Axios và giao tiếp với API*

Axios là một thư viện HTTP client được sử dụng phổ biến để giao tiếp với REST API. Trong hệ thống này, Axios được dùng để thực hiện các tác vụ như: gọi API đăng nhập/đăng ký, thực hiện CRUD đối với nhiệm vụ, hoặc gửi request kèm Bearer Token để xác thực người dùng.

## **2.5. So sánh các lựa chọn công nghệ**

### *2.5.1. Ưu/nhược của Laravel so với các framework backend khác*

So với Node.js (Express): Laravel mạnh về cấu trúc, tích hợp sẵn nhiều tính năng, giúp rút ngắn thời gian phát triển. Trong khi đó, Express.js có tính linh hoạt cao và hiệu năng tốt nhờ cơ chế non-blocking, nhưng thường yêu cầu cài đặt nhiều thư viện thủ công.

So với Django (Python): Laravel có cú pháp dễ tiếp cận đối với lập trình viên PHP, cộng đồng sử dụng lớn, tài liệu phong phú. Django lại nổi bật về bảo mật và có lợi thế trong lĩnh vực phân tích dữ liệu hoặc AI.

### *2.5.2. Ưu/nhược của Vue so với React và Angular*

Vue vs React: Vue có cú pháp gần gũi với HTML, dễ học hơn cho người mới. React phổ biến hơn và sở hữu hệ sinh thái rộng lớn, nhưng đôi khi đòi hỏi nhiều cấu hình phức tạp.

Vue vs Angular: Angular là framework toàn diện, mạnh mẽ nhưng khá phức tạp, tốn nhiều thời gian học. Vue thì nhẹ nhàng, linh hoạt, phù hợp cho các dự án vừa và nhỏ.

Từ những phân tích trên có thể thấy, việc lựa chọn Laravel cho backend và Vue cho frontend là một giải pháp hợp lý. Laravel mang đến sự ổn định và đầy đủ chức năng cho API backend, trong khi Vue giúp phát triển frontend nhẹ nhàng, dễ bảo trì và mở rộng.

# **CHƯƠNG 3: PHÂN TÍCH HỆ THỐNG**

## **3.1. Yêu cầu chức năng (Functional Requirements)**

Hệ thống quản lý công việc được thiết kế với các nhóm chức năng chính, bao gồm quản lý người dùng, quản lý công việc và giao diện người dùng. Những chức năng này nhằm đảm bảo ứng dụng đáp ứng nhu cầu cốt lõi của người dùng trong việc tạo, theo dõi và hoàn thành các nhiệm vụ hằng ngày.

### *3.1.1. Quản lý người dùng*

Để đảm bảo an toàn và cá nhân hóa dữ liệu, hệ thống cung cấp đầy đủ các tính năng liên quan đến quản lý người dùng:

* Đăng ký (Register): Người dùng cần nhập thông tin cơ bản gồm tên, email và mật khẩu. Hệ thống sẽ kiểm tra tính hợp lệ (email chưa tồn tại, mật khẩu đủ mạnh) và sau đó tạo tài khoản mới. Đây là bước khởi đầu để người dùng có thể tham gia vào hệ thống.
* Đăng nhập (Login): Sau khi có tài khoản, người dùng đăng nhập bằng email và mật khẩu. Backend xác thực thông tin này và trả về một mã token (theo chuẩn Sanctum). Token này được dùng trong mọi phiên làm việc tiếp theo.
* Đăng xuất (Logout): Khi người dùng thoát, hệ thống xóa token hiện tại, đảm bảo không còn phiên đăng nhập hợp lệ, từ đó tăng cường bảo mật.
* Xác thực token: Tất cả request đến API quản lý công việc đều phải kèm Bearer token. Điều này giúp đảm bảo rằng mỗi người dùng chỉ có thể truy cập và thao tác trên dữ liệu của chính mình, tránh rò rỉ hoặc chỉnh sửa trái phép.

### *3.1.2. Quản lý công việc (Tasks)*

Đây là nhóm chức năng quan trọng nhất của hệ thống, tập trung vào CRUD (Create, Read, Update, Delete) và các thao tác mở rộng:

* Tạo (Create): Người dùng có thể nhập tiêu đề, mô tả và thời hạn hoàn thành. Hệ thống ghi nhận và lưu dữ liệu này vào cơ sở dữ liệu.
* Đọc (Read): Người dùng có thể xem danh sách toàn bộ công việc cá nhân, được phân trang để tối ưu hiệu năng hiển thị.
* Sửa (Update): Cho phép chỉnh sửa thông tin công việc như tiêu đề, mô tả, deadline và trạng thái.
* Xóa (Delete): Người dùng có thể loại bỏ những công việc không còn cần thiết, hệ thống sẽ xóa dữ liệu đó khỏi cơ sở dữ liệu.
* Thay đổi trạng thái: Chức năng đánh dấu công việc hoàn thành hoặc chưa hoàn thành (toggle status), giúp theo dõi tiến độ dễ dàng hơn.

### *3.1.3. Giao diện & trải nghiệm người dùng*

Một hệ thống quản lý công việc không chỉ cần đầy đủ tính năng mà còn phải dễ sử dụng:

* Giao diện responsive: Thiết kế giao diện thích ứng, đảm bảo hiển thị tốt trên cả máy tính lẫn thiết bị di động.
* Trải nghiệm trực quan: Danh sách công việc được trình bày rõ ràng, form nhập liệu gọn gàng, dễ hiểu.
* Phản hồi tức thời: Hệ thống hiển thị thông báo lỗi hoặc thành công sau mỗi thao tác (ví dụ: thêm công việc thành công, nhập sai email khi đăng ký).

## **3.2. Yêu cầu phi chức năng (Non-functional Requirements)**

Ngoài các chức năng cốt lõi, hệ thống còn cần đáp ứng các yêu cầu phi chức năng để đảm bảo hiệu quả, an toàn và khả năng mở rộng.

### *3.2.1. Bảo mật*

Mật khẩu được lưu trữ dưới dạng hash bằng thuật toán bcrypt, giúp giảm nguy cơ lộ lọt dữ liệu nhạy cảm.

Xác thực người dùng thông qua Laravel Sanctum, đảm bảo mỗi request đều phải kèm token hợp lệ.

CORS được bật để frontend (deploy trên GitHub Pages) có thể gọi API từ backend (Render.com).

Hỗ trợ HTTPS nhằm mã hóa dữ liệu khi truyền tải, tránh nghe lén hoặc tấn công Man-in-the-Middle.

### *3.2.2. Hiệu năng*

Hệ thống được thiết kế để các API cơ bản có thời gian phản hồi trung bình dưới 500ms.

Các trường dữ liệu quan trọng như user\_id, status, deadline, email được đánh index để tối ưu tốc độ truy vấn.

### *3.2.3. Khả năng mở rộng*

Hệ thống có thể mở rộng để phục vụ nhiều người dùng hơn bằng cách tăng tài nguyên trên Render.com.

Cấu trúc backend và frontend sẵn sàng cho việc bổ sung tính năng nâng cao như chia sẻ công việc, gán tag, hoặc gửi thông báo qua email.

### *3.2.4. Khả năng bảo trì*

Backend được phát triển theo chuẩn REST API, giúp dễ dàng thay thế hoặc nâng cấp frontend.

Code frontend tổ chức theo kiến trúc component của Vue.js, thuận tiện cho việc tái sử dụng và bảo trì về lâu dài.

### *3.2.5. Tính sẵn sàng & sao lưu*

Render.com đảm bảo uptime cao, hệ thống luôn sẵn sàng phục vụ.

Cơ sở dữ liệu PostgreSQL được hỗ trợ backup định kỳ, giúp giảm thiểu rủi ro mất dữ liệu.

## **3.3. Phân tích người dùng**

Dựa trên phạm vi đề tài, có hai nhóm đối tượng chính có thể sử dụng hệ thống:

Người dùng cá nhân (Primary User): Chủ yếu là sinh viên hoặc nhân viên văn phòng có nhu cầu quản lý công việc cá nhân. Họ cần một ứng dụng đơn giản, dễ sử dụng, miễn phí nhưng vẫn đảm bảo đầy đủ các chức năng cơ bản.

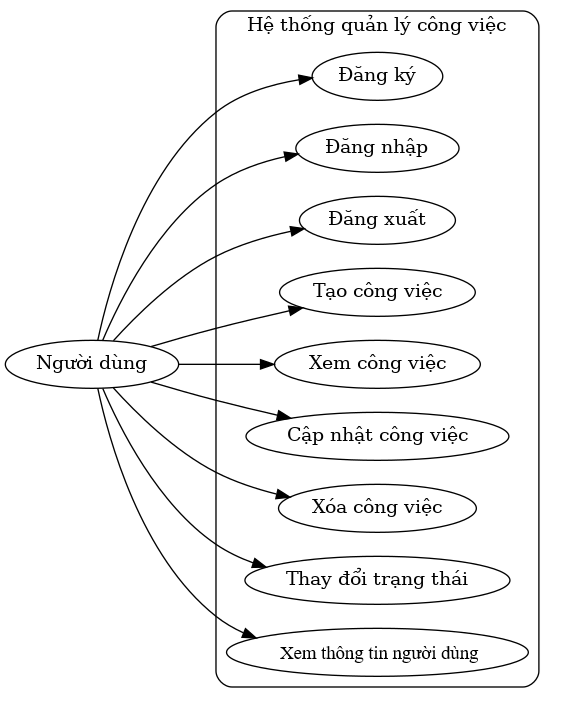
Quản trị viên hệ thống (Admin — optional): Chức năng quản trị chưa được triển khai trong giai đoạn đầu, nhưng có thể được mở rộng trong tương lai. Admin sẽ có quyền quản lý toàn bộ người dùng và công việc, hỗ trợ kiểm soát hệ thống ở mức tổng thể.

## **3.4. Kịch bản sử dụng (Use cases)**

Actor: Người dùng (User)là tác nhân duy nhất, cá nhân đăng ký và sử dụng hệ thống để quản lý công việc.

Các Use case chính:

* Đăng ký tài khoản (Register)
* Đăng nhập (Login)
* Đăng xuất (Logout)
* Quản lý công việc (Tasks CRUD)
  + Tạo công việc (Create Task)
  + Xem danh sách công việc (Read Task)
  + Cập nhật công việc (Update Task)
  + Xóa công việc (Delete Task)
  + Thay đổi trạng thái (Toggle Status)

**Hình 3.1. Sơ đồ use tổng quát**

## **3.5. Luồng tương tác chính**

### *3.5.1. Luồng Authentication*

Quy trình xác thực người dùng diễn ra như sau:

* Người dùng mở giao diện frontend và nhập email, mật khẩu.
* Frontend gửi request đến endpoint /api/login.
* Backend kiểm tra thông tin và trả về token nếu hợp lệ.
* Frontend lưu token này trong LocalStorage của trình duyệt.
* Tất cả các request tiếp theo đều gửi kèm token dưới dạng Bearer Token để được xác thực.

### *3.5.2. Luồng Quản lý công việc (CRUD)*

Người dùng đăng nhập thành công và truy cập màn hình danh sách công việc.

Frontend gọi API GET /api/tasks để lấy danh sách công việc.

Khi người dùng thêm, sửa hoặc xóa công việc, frontend gửi request tương ứng đến API.

Backend xử lý, cập nhật dữ liệu trong cơ sở dữ liệu và trả về kết quả JSON.

Fronend cập nhật lại danh sách công việc hiển thị cho người dùng.

## **3.6. Ràng buộc và giả định**

Ràng buộc:

* Backend chỉ phục vụ dưới dạng REST API, không render view HTML.
* Frontend chỉ giao tiếp với backend thông qua giao thức HTTPS.
* Người dùng phải đăng ký tài khoản trước khi có thể sử dụng chức năng quản lý công việc.

Giả định:

* Người dùng có kết nối Internet ổn định khi sử dụng ứng dụng.
* Hệ thống hoạt động với khối lượng dữ liệu nhỏ đến trung bình (100–1000 task mỗi user).
* Hạ tầng Render.com và GitHub Pages đảm bảo uptime và hiệu năng ổn định.

# **CHƯƠNG 4: THIẾT KẾ HỆ THỐNG**

## **4.1. Kiến trúc tổng thể (System Architecture)**

### *4.1.1. Sơ đồ kiến trúc*

****Hệ thống được xây dựng theo mô hình client–server với ba thành phần chính: Frontend (Vue.js), Backend (Laravel API) vàCơ sở dữ liệu (PostgreSQL). Các thành phần này được triển khai (deploy) trên các nền tảng dịch vụ khác nhau, nhưng phối hợp nhịp nhàng thông qua giao tiếp qua Internet và API RESTful.

**Hình 4.1. Sơ đồ kiến trúc hệ thống**

### *4.1.2. Mô tả các thành phần chính*

Client (Frontend – Vue.js):

* Ứng dụng frontend được xây dựng bằng framework Vue.js và triển khai trên GitHub Pages để đảm bảo miễn phí và dễ dàng phân phối.
* Thành phần này chịu trách nhiệm hiển thị giao diện cho người dùng cuối, xử lý các thao tác như đăng nhập, đăng ký, quản lý danh sách công việc.
* Token xác thực được lưu trữ an toàn trong localStorage, giúp duy trì phiên làm việc của người dùng.
* Ứng dụng được chia thành nhiều component nhỏ, cho phép tái sử dụng và bảo trì dễ dàng.

Backend (Laravel API):

* Được phát triển bằng Laravel và triển khai trên Render.com, backend đóng vai trò xử lý toàn bộ logic của hệ thống.
* Các chức năng bao gồm: xác thực người dùng, thực hiện CRUD đối với công việc, và kiểm tra dữ liệu đầu vào.
* Backend tuân thủ chuẩn RESTful API, hỗ trợ khả năng mở rộng và tích hợp với các frontend khác trong tương lai.
* Các middleware đảm bảo bảo mật và kiểm soát truy cập, đồng thời kết hợp với cơ chế validation để tránh dữ liệu không hợp lệ.

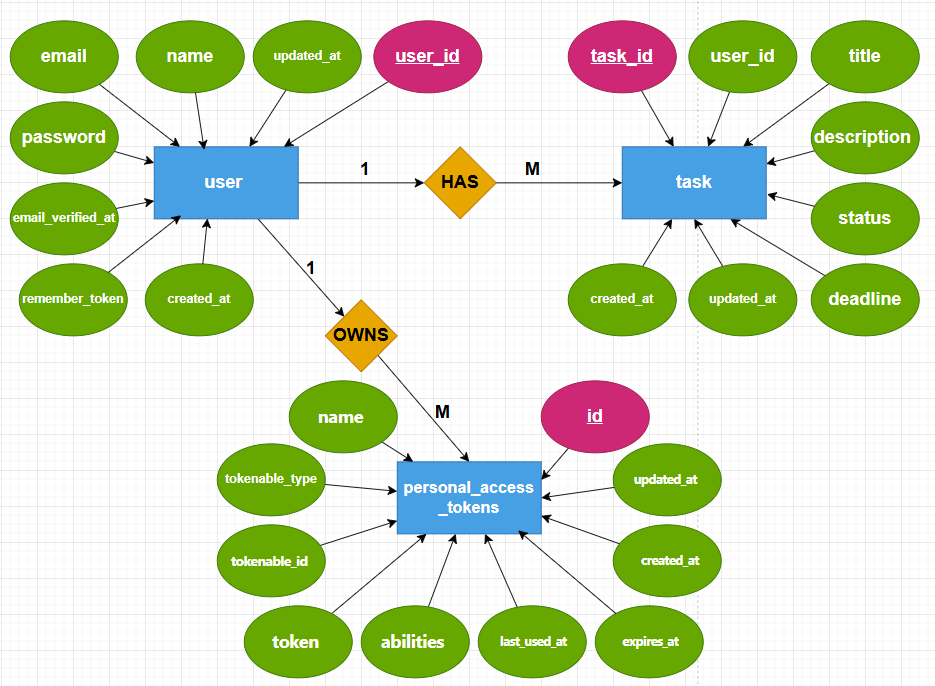
Database (PostgreSQL):

* Lưu trữ thông tin người dùng và công việc trong các bảng quan hệ.
* Sử dụng index cho các trường quan trọng như user\_id, status, deadline để tối ưu hóa tốc độ truy vấn.
* Hệ thống cơ sở dữ liệu hỗ trợ backup và khôi phục định kỳ, đảm bảo an toàn dữ liệu khi có sự cố.

## **4.2. Thiết kế cơ sở dữ liệu (Database Design)**

### *4.2.1. Sơ đồ ERD*

Sơ đồ ERD này mô tả hệ thống cơ sở dữ liệu với tính năng xác thực người dùng qua API tokens. ERD sử dụng các ký hiệu được định nghĩa như hình chữ nhật, hình thoi, hình oval và các đường kết nối để mô tả sự liên kết giữa các thực thể, mối quan hệ và thuộc tính.

**Hình 4.2. Sơ đồ ERD**

Entities (Thực thể) - Hình chữ nhật

* USERS (Người dùng)
  + Thực thể trung tâm của hệ thống
  + Lưu trữ thông tin về người dùng đã đăng ký
* TASKS (Công việc)
  + Thực thể con phụ thuộc vào USERS
  + Lưu trữ thông tin về các công việc được tạo bởi người dùng
* PERSONAL\_ACCESS\_TOKENS (Token truy cập cá nhân)
  + Thực thể hỗ trợ cho việc xác thực API
  + Lưu trữ các token để xác thực người dùng (Laravel Sanctum)

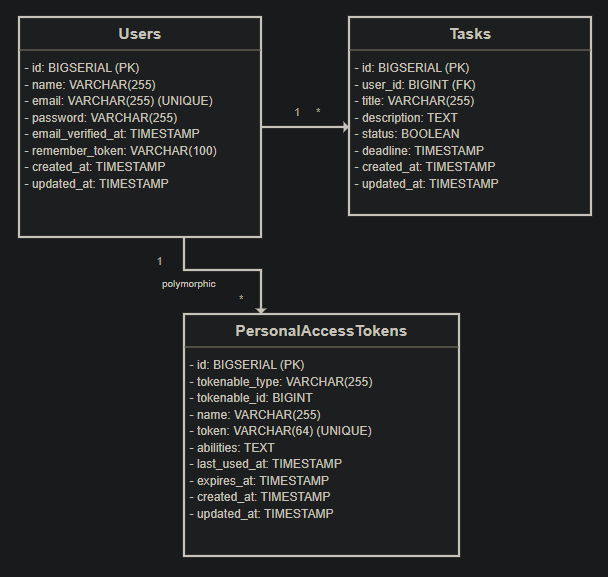
Attributes (Thuộc tính) - Hình oval

* USERS có các thuộc tính:
  + id (Primary Key - màu hồng, gạch chân): Định danh duy nhất
  + name: Tên người dùng
  + email: Email (duy nhất)
  + password: Mật khẩu đã mã hóa
  + email\_verified\_at: Thời gian xác thực email
  + remember\_token: Token ghi nhớ đăng nhập
  + created\_at, updated\_at: Timestamp
* TASKS có các thuộc tính:
  + id (Primary Key): Định danh duy nhất của task
  + title: Tiêu đề công việc
  + description: Mô tả chi tiết
  + status: Trạng thái hoàn thành (true/false)
  + deadline: Thời hạn hoàn thành
  + created\_at, updated\_at: Timestamps
* PERSONAL\_ACCESS\_TOKENS có các thuộc tính:
  + id (Primary Key): Định danh duy nhất
  + tokenable\_type: Loại đối tượng sở hữu token
  + tokenable\_id: ID của đối tượng sở hữu token
  + name: Tên token
  + token: Chuỗi token (duy nhất)
  + abilities: Quyền hạn của token
  + last\_used\_at: Lần cuối sử dụng
  + expires\_at: Thời gian hết hạn
  + created\_at, updated\_at: Timestamps
  + Relationships (Mối quan hệ) - Hình thoi

Mối quan hệ:

* HAS (Users HAS Tasks)
  + Mối quan hệ 1:M (One-to-Many)
  + 1 người dùng có thể có nhiều công việc
  + 1 công việc chỉ thuộc về 1 người dùng
  + Được thực hiện thông qua foreign key user\_id trong bảng TASKS
* OWNS (Users OWNS Personal\_Access\_Tokens)
  + Mối quan hệ 1:M (One-to-Many)
  + 1 người dùng có thể có nhiều tokens
  + 1 token chỉ thuộc về 1 người dùng
  + Được thực hiện thông qua polymorphic relationship (tokenable\_type và tokenable\_id)

### *4.2.2. Mô tả bảng*

**Hình 4.3. Các bảng chính**

Bảng users:

* Chứa thông tin tài khoản của người dùng như: id, name, email, password.
* Cột email được đánh dấu unique để tránh trùng lặp.
* Trường password lưu trữ giá trị đã được mã hóa bằng thuật toán bcrypt.

Bảng tasks:

* Chứa thông tin các công việc, bao gồm: id, title, description, status, deadline, user\_id.
* Trường status là kiểu boolean, biểu thị công việc đã hoàn thành hay chưa.
* Trường user\_id là khóa ngoại liên kết với bảng users, đảm bảo mỗi task luôn thuộc về một người dùng cụ thể.

## **4.3. Thiết kế API (API Specification)**

### *4.3.1. Danh sách endpoint*

Hệ thống API được thiết kế theo chuẩn REST, bao gồm các nhóm chính:

* Auth (xác thực và quản lý người dùng):
  + POST /api/register – Đăng ký tài khoản mới.
  + POST /api/login – Đăng nhập và nhận token xác thực.
  + POST /api/logout – Đăng xuất và hủy token hiện tại.
* Tasks (quản lý công việc):
  + GET /api/tasks – Lấy danh sách công việc của người dùng.
  + POST /api/tasks – Tạo công việc mới.
  + GET /api/tasks/{id} – Xem chi tiết công việc.
  + PUT /api/tasks/{id} – Cập nhật thông tin công việc.
  + DELETE /api/tasks/{id} – Xóa công việc.

### *4.3.2. Request/Response mẫu*

Ví dụ tạo task mới:

* Request:

{

"title": "Hoàn thành báo cáo",

"description": "Viết xong báo áo thực tập",

"deadline": "2025-09-10"

}

* Response (201 Created):

{

"id": 12,

"title": "Hoàn thành báo cáo",

"description": "Viết xong báo cáo thực tập",

"status": false,

"deadline": "2025-09-10",

"user\_id": 1

}

Mã lỗi trả về thường gặp:

* 401 Unauthorized – Token không hợp lệ.
* 403 Forbidden – Người dùng không có quyền truy cập task.
* 404 Not Found – Task không tồn tại.
* 422 Unprocessable Entity – Dữ liệu nhập sai định dạng.

### *4.3.3. Bảo mật API*

Hệ thống API được thiết kế với một cơ chế bảo mật nghiêm ngặt nhằm đảm bảo an toàn thông tin và ngăn chặn các truy cập trái phép. Cụ thể, mọi yêu cầu liên quan đến việc quản lý task đều phải được xác thực thông qua việc gửi kèm header Authorization với định dạng "Bearer <token>". Đây là một tiêu chuẩn bảo mật phổ biến trong các ứng dụng web hiện đại, giúp đảm bảo rằng chỉ những người dùng đã được xác thực mới có thể thực hiện các thao tác với dữ liệu. Token được phát hành bởi Laravel Sanctum, đảm bảo cơ chế xác thực an toàn và đáng tin cậy.

Một nguyên tắc bảo mật quan trọng khác được áp dụng là việc ràng buộc chặt chẽ giữa mỗi task và user\_id của người dùng sở hữu nó. Điều này có nghĩa là mỗi công việc trong hệ thống luôn được gắn liền với định danh của người tạo ra nó, từ đó hoàn toàn ngăn chặn khả năng truy cập dữ liệu chéo giữa các tài khoản khác nhau. Cơ chế này đảm bảo tính riêng tư tuyệt đối, người dùng A sẽ không bao giờ có thể xem, chỉnh sửa hoặc xóa các task thuộc về người dùng B.

## **4.4. Thiết kế phía Backend (Laravel)**

### *4.4.1. Luồng xử lý request*

Quy trình xử lý một yêu cầu trong hệ thống được thiết kế theo mô hình MVC chuẩn của Laravel, đảm bảo tính nhất quán và dễ bảo trì. Khi một ứng dụng khách gửi yêu cầu đến API, quá trình xử lý diễn ra theo một chuỗi các bước có trình tự rõ ràng và hợp lý:

* Client gửi request đến API.
* Request được định tuyến (Route) đến controller tương ứng.
* Dữ liệu đầu vào được kiểm tra thông qua Validation.
* Controller gọi đến Service/Model để xử lý nghiệp vụ và tương tác với cơ sở dữ liệu.
* Kết quả được trả về dưới dạng JSON Response.

### *4.4.2. Middleware & Validation*

Hệ thống bảo mật của backend được xây dựng trên nền tảng của hai thành phần chính là Middleware và Validation. Middleware auth:sanctum được triển khai như một lớp bảo vệ trung gian, đứng giữa yêu cầu từ khách và logic xử lý của controller. Cụ thể, middleware này sẽ tự động kiểm tra và xác thực token của người dùng trước khi cho phép truy cập vào bất kỳ đường dẫn nào thuộc nhóm /api/tasks. Chỉ những người dùng đã đăng nhập thành công và sở hữu token hợp lệ mới được phép thực hiện các thao tác liên quan đến quản lý task.

Bên cạnh middleware, hệ thống validation được xây dựng thông qua các lớp FormRequest chuyên biệt như StoreTaskRequest và UpdateTaskRequest. Những lớp này đóng vai trò như những cổng kiểm soát chất lượng, đảm bảo rằng mọi dữ liệu được gửi từ ứng dụng khách đều phải tuân thủ các quy tắc nghiêm ngặt về định dạng, độ dài, kiểu dữ liệu và các ràng buộc nghiệp vụ khác trước khi được lưu xuống cơ sở dữ liệu. Điều này không chỉ giúp duy trì tính toàn vẹn của dữ liệu mà còn ngăn chặn hiệu quả các cuộc tấn công chèn mã độc và làm hỏng dữ liệu.

### *4.4.3. Migration, Seeder, Factory*

Cấu trúc cơ sở dữ liệu của hệ thống được quản lý thông qua hệ thống Migration mạnh mẽ của Laravel. Các tệp Migration đóng vai trò như bản thiết kế chi tiết của cơ sở dữ liệu, định nghĩa một cách chính xác cấu trúc của các bảng chính như users và tasks.

Các lớp Seeder được phát triển nhằm mục đích tạo ra dữ liệu mẫu phong phú và đa dạng cho quá trình kiểm thử và phát triển. Thay vì phải thủ công tạo ra hàng chục người dùng và công việc để kiểm tra các tính năng, các seeder sẽ tự động tạo ra một bộ dữ liệu hoàn chỉnh bao gồm nhiều hồ sơ người dùng khác nhau cùng với các task tương ứng, đảm bảo rằng ứng dụng được kiểm tra trên một bộ dữ liệu có tính đại diện cao.

Các lớp Factory được thiết kế đặc biệt để phục vụ cho việc kiểm thử tự động và kiểm thử đơn vị. Khác với seeder tạo ra dữ liệu cố định, factory có khả năng tạo ngẫu nhiên hàng nghìn bản ghi với các thuộc tính đa dạng, giúp các nhà phát triển có thể viết các test case phong phú và toàn diện. Mẫu thiết kế Factory này đặc biệt hữu ích trong việc kiểm tra hiệu suất, các trường hợp biên và các tình huống giới hạn của ứng dụng.

## **4.5. Thiết kế phía Frontend (Vue.js)**

### *4.5.2. Wireframes / Mockups*

Giao diện người dùng được thiết kế theo nguyên tắc đơn giản và lấy người dùng làm trung tâm, tập trung vào việc tối ưu trải nghiệm người dùng thông qua các màn hình được thiết kế tinh tế và chức năng. Màn hình đăng nhập được thiết kế bao gồm một biểu mẫu đăng nhập tối giản với hai trường thông tin chính là email và mật khẩu. Giao diện này loại bỏ mọi yếu tố không cần thiết để tập trung hoàn toàn vào mục đích chính là xác thực, đồng thời vẫn đảm bảo tính thẩm mỹ và chuyên nghiệp.

Màn hình danh sách Task được thiết kế như trung tâm điều khiển của toàn bộ ứng dụng, nơi người dùng có thể xem tổng quan toàn bộ công việc của mình một cách trực quan và hiệu quả cho phép người dùng nhanh chóng cập nhật trạng thái hoàn thành. Các nút hành động như chỉnh sửa và xóa được đặt ở vị trí dễ tiếp cận nhưng không gây nhầm lẫn hoặc nhấp nhầm. Bố cục được tối ưu để hiển thị tối đa thông tin trong không gian có sẵn mà vẫn duy trì khả năng đọc và tính thẩm mỹ.

Task Form được thiết kế để xử lý cả hai chức năng tạo task mới và chỉnh sửa task hiện có trong một giao diện thống nhất. Biểu mẫu được cấu trúc với các trường hợp lý và trực quan, hỗ trợ soạn thảo văn bản phong phú và nhiều loại đầu vào khác nhau tùy thuộc vào yêu cầu của task.

### *4.5.3. Luồng xác thực*

Luồng xác thực được triển khai với sự tập trung cao vào bảo mật và trải nghiệm người dùng. Sau khi người dùng hoàn thành quá trình đăng nhập thành công, ứng dụng sẽ nhận được token xác thực từ backend và lưu trữ nó một cách an toàn trong localStorage của trình duyệt.

Cơ chế route guard được triển khai như một lớp bảo mật toàn diện, tự động giám sát trạng thái xác thực của người dùng trên mọi lần chuyển hướng đường dẫn. Khi người dùng cố gắng truy cập bất kỳ đường dẫn được bảo vệ nào mà không sở hữu token hợp lệ, bộ bảo vệ đường dẫn sẽ ngay lập tức ngắt luồng chuyển hướng và điều hướng họ về trang đăng nhập.

Quản lý token cũng bao gồm việc làm mới tự động và xử lý hết hạn, đảm bảo rằng phiên làm việc của người dùng vẫn hợp lệ trong khoảng thời gian hợp lý mà không ảnh hưởng đến bảo mật. Hệ thống cũng xử lý các trường hợp đặc biệt như token bị hỏng hoặc bị vô hiệu hóa từ phía máy chủ một cách duyên dáng.

### *4.5.4. Quản lý trạng thái & giao tiếp API*

Hạ tầng giao tiếp API được xây dựng trên nền tảng của Axios HTTP client, được nâng cấp với cơ chế interceptor tinh vi để đơn giản hóa quá trình xác thực. Axios interceptor tự động gắn header Authorization với token hợp lệ hiện tại vào mọi yêu cầu gửi đi, loại bỏ nhu cầu quản lý header thủ công ở mỗi lời gọi API. Điều này không chỉ giảm thiểu việc lặp lại mã nguồn mà còn đảm bảo tính nhất quán và độ tin cậy trong xác thực trên toàn bộ ứng dụng.

Kiến trúc trạng thái được thiết kế với sự tách biệt rõ ràng các mối quan tâm, với các module riêng biệt cho xác thực, quản lý task và trạng thái giao diện người dùng. Các hành động và đột biến được định nghĩa rõ ràng với xử lý lỗi phù hợp và trạng thái tải, cung cấp cho người dùng phản hồi rõ ràng về trạng thái của các thao tác của họ. Cập nhật theo thời gian thực và cập nhật giao diện người dùng tạm thời được triển khai để nâng cao hiệu suất cảm nhận và khả năng phản hồi.

## **4.6. Thiết kế giao diện người dùng (UI/UX)**

### *4.6.1. Nguyên tắc UI/UX*

Triết lý thiết kế giao diện người dùng được xây dựng trên nền tảng của tính khả dụng và khả năng tiếp cận, nhắm đến nhóm người dùng rộng rãi bao gồm sinh viên và nhân viên văn phòng với các mức độ chuyên môn kỹ thuật khác nhau. Phương pháp thiết kế ưu tiên sự đơn giản và trực quan hơn là các hoạt ảnh rực rỡ hoặc tương tác phức tạp, đảm bảo rằng người dùng có thể đạt được mục tiêu của họ một cách hiệu quả mà không cần phải trải qua quá trình học tập khó khăn.

Responsive được triển khai như một yêu cầu cốt lõi thay vì là suy nghĩ thứ cấp, với phương pháp tiếp cận ưu tiên di động trước đảm bảo trải nghiệm tối ưu trên tất cả các loại thiết bị từ điện thoại thông minh đến màn hình máy tính để bàn.

Accessible: được tuân thủ nghiêm ngặt để làm cho ứng dụng trở nên bao trùm cho người dùng với những khả năng khác nhau, nút bấm và màu sắc rõ ràng, đảm bảo người dùng dễ dàng nhận biết thao tác.

### *4.6.2. Palette, Typography, Component*

Hệ thống thiết kế thị giác được tuyển chọn cẩn thận để tạo ra vẻ ngoài gắn kết và chuyên nghiệp trong khi duy trì sự tương tác của người dùng. Bảng màu được neo bởi các tông màu xanh và đen, được chọn đặc biệt để gợi lên cảm giác tin cậy, đáng tin cậy và chuyên nghiệp. Màu xanh được sử dụng cho các hành động chính và điểm nhấn, trong khi màu đen cung cấp độ tương phản mạnh cho khả năng đọc văn bản và thiết lập phân cấp thị giác. Các màu hỗ trợ được chọn để bổ sung cho bảng màu chính mà không làm choáng ngợp người dùng hoặc tạo ra tiếng ồn thị giác.

Hệ thống typography được xây dựng xung quanh các phông chữ hiện đại, có độ đọc cao với sự chú ý cẩn thận đến phân cấp và khoảng cách. Việc lựa chọn phông chữ ưu tiên khả năng đọc trên các kích thước màn hình và độ phân giải khác nhau, với chiều cao dòng và khoảng cách chữ thích hợp để tối ưu hóa trải nghiệm đọc. Các cấp độ tiêu đề được phân biệt rõ ràng thông qua kích thước, trọng lượng và khoảng cách, tạo ra phân cấp thông tin rõ ràng hướng dẫn người dùng qua nội dung một cách tự nhiên.

Tích hợp framework CSS được tận dụng để tăng tốc quá trình phát triển trong khi duy trì tính nhất quán của thiết kế. Framework được tùy chỉnh với các biến và ghi đè cụ thể của dự án để đảm bảo bản sắc thương hiệu độc đáo trong khi hưởng lợi từ các thành phần và tiện ích đã được kiểm nghiệm. Thư viện component được phát triển với các yếu tố có thể tái sử dụng như nút, biểu mẫu, cửa sổ bật lên và thành phần điều hướng, đảm bảo tính nhất quán trên toàn ứng dụng và tạo thuận lợi cho việc bảo trì và cập nhật trong tương lai.

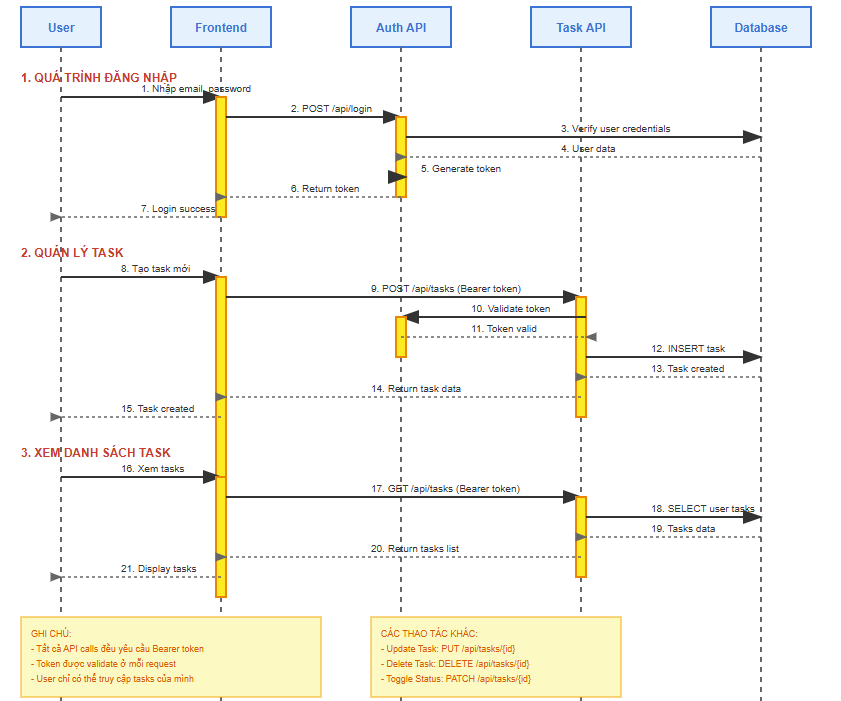
## **4.7. Sequence Diagram**

Mục đích: Mô tả tương tác giữa các actors theo thời gian

Các actors: User, Frontend, Auth API, Task API, Database

Quy trình chính:

* Đăng nhập và xác thực token
* Tạo task mới với Bearer token
* Xem danh sách tasks của user

**Hình 4.4 Sơ đồ Sequence Diagram**

# **CHƯƠNG 5: TRIỂN KHAI VÀ THỰC HIỆN**

Mục tiêu của chương này là mô tả chi tiết quá trình xây dựng và triển khai hệ thống từ môi trường phát triển, cấu hình backend (Laravel), cấu hình frontend (Vue.js) đến quy trình tích hợp và triển khai (deployment). Toàn bộ mã nguồn được tổ chức dưới dạng hai repository con (submodules) trong repo chính TaskPJ.

## **5.1. Phương pháp tổ chức và quản lý mã nguồn**

### *5.1.1. Chiến lược tổ chức repository*

Hệ thống được tổ chức theo mô hình repository chính với hai submodule độc lập. Repository chính TaskPJ đóng vai trò là container tổng thể, trong khi TaskBackend và TaskFrontend được phát triển như các module riêng biệt. Cách tiếp cận này cho phép hai team có thể làm việc song song mà không ảnh hưởng lẫn nhau, đồng thời duy trì tính nhất quán về version control.

### *5.1.2. Tiêu chuẩn hóa môi trường phát triển*

Để đảm bảo tính nhất quán giữa các môi trường khác nhau, dự án áp dụng chiến lược tiêu chuẩn hóa phiên bản nghiêm ngặt. Backend sử dụng PHP 8.2 với Laravel framework, kết hợp Composer 2.x cho quản lý dependencies. Frontend được xây dựng trên Node.js 18.x LTS với npm 9.x, đảm bảo tính ổn định và hỗ trợ dài hạn. PostgreSQL 15 được lựa chọn cho môi trường production nhằm tận dụng các tính năng performance và bảo mật mới nhất.

## **5.2. Quy trình triển khai Backend (Laravel)**

### *5.2.1. Phương pháp khởi tạo và cấu hình*

Backend được khởi tạo theo cấu trúc chuẩn của Laravel framework, với việc tùy chỉnh cấu hình để phù hợp với môi trường cloud. Quá trình setup tập trung vào việc thiết lập các biến môi trường an toàn, cấu hình database connection, và thiết lập các service cần thiết cho ứng dụng production.

### *5.2.2. Chiến lược quản lý cơ sở dữ liệu*

Database schema được quản lý thông qua hệ thống migration của Laravel, đảm bảo tính nhất quán và khả năng rollback khi cần thiết. Các bảng chính bao gồm users, tasks và personal\_access\_tokens được thiết kế với các ràng buộc phù hợp.

* Bảng users có các cột: id, name, email (unique), password, timestamps.
* Bảng tasks có các cột: id, title, description (nullable), status (boolean default false), deadline (date nullable), user\_id (foreign key), timestamps.

Seeder được sử dụng để tạo dữ liệu mẫu, hỗ trợ quá trình testing và demonstration.

### *5.2.3. Kiến trúc API và bảo mật*

AuthController: các method register, login, logout (trả token, revoke token).

TaskController: methods index, store, show, update, destroy.

Routes (routes/api.php): nhóm các route /api/tasks được bảo vệ bằng middleware auth:sanctum:

Route::post('/register', [AuthController::class, 'register']);

Route::post('/login', [AuthController::class, 'login']);

Route::middleware('auth:sanctum')->group(function() {

Route::post('/logout', [AuthController::class, 'logout']);

Route::apiResource('tasks', TaskController::class);

});

API được thiết kế theo chuẩn RESTful với Laravel Sanctum làm giải pháp authentication. Hệ thống route được tổ chức theo nhóm, với middleware auth:sanctum bảo vệ các endpoints quan trọng. AuthController xử lý các chức năng xác thực cơ bản (register, login, logout), trong khi TaskController quản lý toàn bộ CRUD operations cho tasks.

### *5.2.4. Cơ chế bảo mật và middleware*

Hệ thống áp dụng nhiều lớp bảo mật khác nhau. CORS được cấu hình để cho phép truy cập từ domain frontend, rate limiting ngăn chặn các cuộc tấn công brute force, và mật khẩu được hash bằng bcrypt. HTTPS được bắt buộc trong production để mã hóa toàn bộ dữ liệu truyền tải..

### *5.2.5. Validation & xử lý lỗi*

Sử dụng FormRequest (ví dụ StoreTaskRequest, UpdateTaskRequest) để validate dữ liệu: title required, deadline date, v.v. Đảm bảo tính toàn vẹn của dữ liệu trước khi xử lý.

Exception Handler: cấu hình trả về JSON rõ ràng (message, errors) với mã lỗi HTTP phù hợp (422 cho validation, 401 cho unauthorized,…). Exception Handler được tùy chỉnh để trả về response JSON nhất quán với các mã lỗi HTTP phù hợp, giúp frontend xử lý lỗi một cách hiệu quả.

### *5.2.6. Phương pháp containerization và deployment*

Backend được đóng gói bằng Docker với Dockerfile được tối ưu hóa cho môi trường production. Container được thiết kế lightweight, chỉ bao gồm các component cần thiết và được cấu hình để chạy trên port 8000. Render.com được lựa chọn làm platform deployment nhờ khả năng tự động scaling và managed database service.

## **5.3. Cài đặt & cấu hình frontend (Vue.js)**

Frontend là SPA dùng Vue 3 + Vite, triển khai tĩnh trên GitHub Pages.

### *5.3.1. Kiến trúc Single Page Application*

Frontend được xây dựng theo mô hình SPA sử dụng Vue 3 với Composition API, kết hợp Vite làm build tool để tối ưu performance. Cách tiếp cận này cho phép tạo ra user experience mượt mà và giảm thiểu thời gian loading.

### *5.3.2. Chiến lược giao tiếp API*

Axios được sử dụng làm HTTP client chính với việc cấu hình interceptors để tự động xử lý authentication token và error handling. Base URL được cấu hình để có thể dễ dàng chuyển đổi giữa các môi trường khác nhau. Timeout và retry logic được implement để đảm bảo reliability.

### *5.3.3. Tổ chức component và state management*

LoginForm.vue: form nhận email và password, gọi POST /api/login, lưu token về localStorage và cập nhật store.

RegisterForm.vue: form đăng ký tương tự.

TaskList.vue: gọi GET /api/tasks, hiển thị danh sách, checkbox toggle trạng thái, nút sửa/xóa.

TaskForm.vue: dùng cho thêm và sửa task (reusable).

ProtectedRoute / Route Guard: kiểm tra token trước khi cho phép truy cập trang Tasks; nếu không có, chuyển hướng về /login.

### *5.3.4. Route protection và quản lý session*

Route guard được triển khai để bảo vệ các trang yêu cầu authentication. Token được lưu trong localStorage để duy trì session qua các lần reload, với logic tự động redirect về login khi token expire hoặc invalid.

### *5.3.5. Quy trình kiểm thử và đảm bảo chất lượng*

Testing được thực hiện theo phương pháp kiểm thử thủ công với các test case bao gồm authentication flow, CRUD operations, và error handling scenarios. Mỗi chức năng được test kỹ lưỡng để đảm bảo UI response phù hợp với user expectations.

Test đăng nhập thành công và đăng nhập sai (thông báo lỗi).

Test quy trình CRUD: tạo, đọc, cập nhật, xóa task, đảm bảo phản hồi UI phù hợp.

Test hiển thị deadline và status đúng.

### *5.3.7. Triển khai GitHub Pages*

GitHub Pages được lựa chọn để host frontend nhờ tính đơn giản và tích hợp tốt với GitHub repository. Build process được tự động hóa thông qua GitHub Actions với workflow trigger khi có push lên main branch. Quá trình build sử dụng npm để tối ưu hóa code, sau đó deploy lên gh-pages branch.

## **5.4. Tích hợp và automation**

### *5.4.1. CI/CD pipeline*

GitHub Actions được thiết lập để tự động hóa quy trình build và deploy frontend. Workflow được cấu hình để chạy trên môi trường Ubuntu với các bước install dependencies, build production code, và deploy lên GitHub Pages. Điều này đảm bảo mọi thay đổi được deploy một cách nhất quán và đáng tin cậy.

### *5.4.2. Quản lý cấu hình môi trường*

Các biến môi trường được quản lý riêng biệt cho từng environment (development, staging, production). Backend sử dụng .env file với các sensitive data được bảo mật thông qua platform environment variables. Frontend được cấu hình base URL phù hợp với mục tiêu triển khai.

# **CHƯƠNG 6: ĐÁNH GIÁ KẾT QUẢ VÀ THẢO LUẬN**

Mục tiêu chính của chương này là đối chiếu kết quả đạt được so với mục tiêu ban đầu, đánh giá hệ thống theo nhiều tiêu chí khác nhau, chỉ ra hạn chế tồn tại, đồng thời rút ra các bài học kinh nghiệm trong quá trình học tập và triển khai thực tế với Laravel và Vue.js. Cuối cùng, chương cũng đưa ra một số gợi ý cải tiến kỹ thuật và định hướng phát triển trong tương lai.

## **6.1. So sánh kết quả với mục tiêu ban đầu**

Ở giai đoạn khởi đầu, đề tài được định hướng với các mục tiêu chính sau:

Xây dựng một ứng dụng quản lý công việc cá nhân (Task Manager) với các chức năng cơ bản như quản lý người dùng, quản lý công việc, giao diện trực quan và khả năng đồng bộ dữ liệu.

Nghiên cứu và thực hành hai công nghệ chủ đạo là Laravel (backend) và Vue.js (frontend).

Triển khai hệ thống hoàn chỉnh trên môi trường cloud, cụ thể là Render cho backend và GitHub Pages cho frontend.

Kết quả thực hiện cho thấy các mục tiêu trên cơ bản đã được đáp ứng đầy đủ:

Toàn bộ chức năng CRUD (Create, Read, Update, Delete) cho công việc đã được cài đặt thành công và hoạt động ổn định.

Hệ thống authentication bằng Laravel Sanctum vận hành tốt, đảm bảo người dùng chỉ có thể thao tác trên dữ liệu của chính họ.

Phần frontend viết bằng Vue.js có giao diện trực quan, thân thiện, tương thích cả trên desktop và thiết bị di động.

Ứng dụng đã được triển khai thành công, với backend chạy trên Render và frontend hiển thị trên GitHub Pages.

Ngoài ra, đề tài còn vượt ngoài phạm vi ban đầu khi bổ sung một số tính năng và kỹ thuật hỗ trợ như: seed dữ liệu mẫu để dễ kiểm thử, validation form để hạn chế lỗi nhập liệu, và sử dụng interceptor trong Axios nhằm quản lý token hiệu quả hơn.

## **6.2. Đánh giá theo tiêu chí**

Để có cái nhìn toàn diện hơn, hệ thống được đánh giá dựa trên các tiêu chí sau:

Chức năng: Ứng dụng đáp ứng đầy đủ yêu cầu cơ bản: CRUD task, đăng ký/đăng nhập/đăng xuất người dùng, xác thực token và giao diện responsive. Điều này cho phép người dùng quản lý công việc cá nhân một cách mượt mà và rõ ràng.

Hiệu năng: Các truy vấn cơ bản (SELECT, INSERT, UPDATE, DELETE) được xử lý nhanh nhờ sử dụng index trên những trường quan trọng. Với hạ tầng Render, hệ thống duy trì khả năng phản hồi ổn định đối với lượng request ở mức nhỏ đến trung bình.

Bảo mật: Hệ thống tuân thủ các nguyên tắc bảo mật quan trọng: mật khẩu người dùng được hash bằng bcrypt, xác thực bằng token thông qua Sanctum, đồng thời kích hoạt CORS và HTTPS để đảm bảo an toàn khi truyền tải dữ liệu.

Khả năng mở rộng: Cấu trúc backend theo REST API và frontend modular hóa bằng Vue component tạo tiền đề cho việc mở rộng. Trong tương lai, ứng dụng có thể bổ sung các tính năng nâng cao như phân loại công việc theo project, chia sẻ task cho nhiều người dùng, hoặc tích hợp hệ thống thông báo.

Trải nghiệm người dùng (UX): Giao diện được thiết kế gọn gàng, trực quan, hỗ trợ thao tác dễ dàng trên cả máy tính và điện thoại. Tuy nhiên, vẫn còn thiếu một số tính năng tiện ích như bộ lọc nâng cao hoặc thao tác kéo-thả (drag-drop) để sắp xếp công việc linh hoạt hơn.

## **6.3. Hạn chế của hệ thống (Known limitations)**

Mặc dù hệ thống đã đạt được nhiều kết quả khả quan, vẫn tồn tại một số hạn chế đáng lưu ý:

Chưa có phân quyền nâng cao: Hệ thống mới chỉ dừng lại ở vai trò “người dùng thông thường”. Chức năng quản trị (admin) để giám sát và quản lý toàn bộ user/task vẫn chưa được triển khai.

Thiếu các chức năng nâng cao: Các tính năng hữu ích như nhắc việc (reminder), thông báo real-time, hoặc khả năng upload file đính kèm chưa được tích hợp.

Hiệu năng phụ thuộc vào Render: Khi có nhiều người dùng truy cập đồng thời, tốc độ phản hồi có thể bị chậm do giới hạn tài nguyên.

Kiểm thử còn hạn chế: Mặc dù có unit test cơ bản, phần lớn kiểm thử vẫn được thực hiện thủ công. Hệ thống chưa có quy trình test tự động toàn diện.

## **6.4. Những bài học rút ra khi học Laravel & Vue**

### *6.4.1. Những khái niệm khó / cần thời gian để nắm vững*

Cơ chế xác thực bằng Sanctum và quy trình hoạt động của middleware trong Laravel.

Nguyên lý reactivity trong Vue và cách quản lý trạng thái qua Composition API.

Thiết lập kết nối cơ sở dữ liệu PostgreSQL trên môi trường cloud (Render) với nhiều ràng buộc thực tế.

### *6.4.2. Những điểm mạnh của Laravel*

Eloquent ORM: Cho phép thao tác dữ liệu một cách trực quan, ánh xạ tự nhiên giữa bảng và model.

Migration & Seeder: Giúp quản lý phiên bản cơ sở dữ liệu dễ dàng, đồng thời tái tạo dữ liệu mẫu nhanh chóng.

Middleware: Tiện lợi trong việc xử lý bảo mật, logging, và CORS.

Sanctum: Cung cấp cơ chế xác thực API token đơn giản nhưng hiệu quả, phù hợp với ứng dụng vừa và nhỏ.

### *6.4.3. Những điểm mạnh của Vue*

Composition API: Hỗ trợ tách biệt logic, tăng khả năng tái sử dụng code, linh hoạt hơn so với Options API.

Reactivity: UI được cập nhật tự động khi dữ liệu thay đổi, mang lại trải nghiệm mượt mà.

Vue Router: Giúp điều hướng trang đơn giản, dễ dàng bảo vệ các route yêu cầu đăng nhập.

Axios integration: Kết hợp interceptor cho phép xử lý token và lỗi một cách tập trung, tăng tính bảo mật và giảm rủi ro.

## **6.5. Gợi ý cải tiến kỹ thuật & hướng nghiên cứu tiếp theo**

Dựa trên những hạn chế đã nêu và kinh nghiệm triển khai, một số gợi ý cải tiến và định hướng nghiên cứu được đề xuất:

Cải tiến kỹ thuật:

* Tích hợp phân quyền nâng cao (Role-Based Access Control - RBAC) để phân biệt user, admin, super-admin.
* Bổ sung hệ thống notification real-time sử dụng WebSocket hoặc Laravel Echo.
* Xây dựng quy trình kiểm thử tự động với unit test, integration test kết hợp CI/CD pipeline nhằm đảm bảo chất lượng phần mềm.
* Sử dụng Redis cache để cải thiện hiệu năng truy vấn và giảm tải cho cơ sở dữ liệu.

Hướng nghiên cứu tiếp theo:

* Nghiên cứu và áp dụng kiến trúc microservices thay cho monolithic, nhằm tăng khả năng mở rộng.
* Tích hợp Progressive Web App (PWA) để hỗ trợ chế độ offline.
* Triển khai ứng dụng trên hạ tầng Kubernetes hoặc kiến trúc serverless để tối ưu chi phí và khả năng mở rộng.
* Ứng dụng machine learning để gợi ý sắp xếp công việc theo mức độ ưu tiên hoặc thói quen của người dùng.

# **CHƯƠNG 7: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

## **7.1. Kết luận chung**

Đề tài “Tìm hiểu framework Laravel, Vue.js và xây dựng ứng dụng minh họa - Task Manager” đã được triển khai thành công, từ khâu phân tích yêu cầu, thiết kế hệ thống, cho đến xây dựng, triển khai và đánh giá kết quả. Quá trình thực hiện cho thấy:

Ứng dụng đã đáp ứng được các mục tiêu cốt lõi đặt ra ban đầu, bao gồm: quản lý người dùng, quản lý công việc với đầy đủ chức năng CRUD, giao diện trực quan, bảo mật bằng Laravel Sanctum, và triển khai trên môi trường cloud.

Hệ thống đã được xây dựng trên nền tảng công nghệ hiện đại: Laravel cho backend, Vue.js cho frontend, PostgreSQL cho cơ sở dữ liệu, và được triển khai thông qua Render (backend) và GitHub Pages (frontend).

Báo cáo đã mô tả rõ ràng từ giai đoạn thiết kế kiến trúc, cơ sở dữ liệu, API, UI/UX, đến quy trình triển khai và kiểm thử, giúp đảm bảo tính hệ thống và khoa học.

Đặc biệt, quá trình thực hiện đề tài không chỉ mang lại một sản phẩm ứng dụng hoàn chỉnh mà còn giúp người thực hiện tích lũy thêm nhiều kiến thức thực tiễn về lập trình web, quản lý dự án phần mềm, cũng như kinh nghiệm triển khai trên môi trường cloud thực tế.

## **7.2. Đóng góp của đề tài**

Những kết quả chính mà đề tài đạt được có thể tóm tắt như sau:

Sản phẩm phần mềm: Một ứng dụng quản lý công việc cá nhân hoạt động ổn định, cho phép người dùng đăng ký, đăng nhập và quản lý công việc hiệu quả trên cả máy tính và điện thoại.

Đóng góp học thuật: Báo cáo đã hệ thống hóa kiến thức về quá trình phát triển ứng dụng web theo mô hình client–server, ứng dụng công nghệ hiện đại (Laravel, Vue.js, REST API).

Đóng góp thực tiễn: Sản phẩm có thể được sử dụng ngay trong thực tế cho cá nhân muốn quản lý công việc hằng ngày, đồng thời có tiềm năng mở rộng thành công cụ quản lý nhóm hoặc tổ chức nhỏ.

## **7.3. Hạn chế còn tồn tại**

Mặc dù đạt được nhiều kết quả, hệ thống vẫn còn một số điểm hạn chế đã được chỉ ra ở Chương 6, như:

Chưa có cơ chế phân quyền nâng cao (chỉ hỗ trợ người dùng thông thường, chưa có admin).

Các tính năng nâng cao như thông báo real-time, nhắc việc, upload file vẫn chưa được triển khai.

Hiệu năng còn phụ thuộc nhiều vào dịch vụ Render khi có số lượng người dùng lớn.

Kiểm thử tự động chưa đầy đủ, phần lớn vẫn dựa trên kiểm thử thủ công.

Những hạn chế này chính là tiền đề cho các hướng cải tiến trong tương lai.

## **7.4. Hướng phát triển tương lai**

Trong giai đoạn tiếp theo, hệ thống có thể được mở rộng và cải tiến theo các hướng sau:

Phát triển chức năng:

Bổ sung phân quyền nâng cao (RBAC) để quản lý nhiều vai trò khác nhau (user, admin, super-admin).

Tích hợp tính năng nhắc việc (reminder), thông báo real-time qua WebSocket hoặc Laravel Echo.

Cho phép chia sẻ công việc giữa nhiều người dùng, hỗ trợ cộng tác nhóm.

Hỗ trợ upload file đính kèm hoặc đính kèm hình ảnh minh họa cho từng công việc.

Cải tiến kỹ thuật:

Tối ưu hiệu năng bằng cách sử dụng Redis cache, CDN, và triển khai trên hạ tầng có khả năng mở rộng linh hoạt (Kubernetes, serverless).

Tích hợp CI/CD pipeline để tự động hóa quá trình build, test và deploy.

Áp dụng kiểm thử tự động (unit test, integration test, end-to-end test) toàn diện hơn.

Hướng nghiên cứu mở rộng:

Chuyển đổi sang kiến trúc microservices để tăng khả năng mở rộng và tính linh hoạt.

Tích hợp PWA (Progressive Web App) để hỗ trợ chế độ offline và thông báo đẩy.

Nghiên cứu tích hợp AI/Machine Learning nhằm gợi ý thứ tự ưu tiên công việc hoặc phân tích thói quen làm việc của người dùng.

## **7.5. Tổng kết**

Nhìn chung, đề tài đã đạt được những kết quả quan trọng, không chỉ dừng lại ở việc xây dựng một ứng dụng quản lý công việc cá nhân hoàn chỉnh mà còn giúp người thực hiện rèn luyện kỹ năng về phát triển phần mềm hiện đại. Sản phẩm này có tính ứng dụng cao, đồng thời là nền tảng vững chắc để tiếp tục nghiên cứu và phát triển những hệ thống quản lý công việc tiên tiến hơn trong tương lai.

# **TÀI LIỆU THAM KHẢO**

[1] Laravel, *Laravel Documentation: The PHP Framework for Web Artisans*. Available: <https://laravel.com/docs>, 2025.

[2] Vue.js, *Vue.js Documentation: The Progressive JavaScript Framework*. Available: <https://vuejs.org/guide>, 2025.

[3] PostgreSQL Global Development Group, *PostgreSQL Documentation*. Available: <https://www.postgresql.org/docs/>, 2025.

[4] Mozilla Developer Network (MDN), *MDN Web Docs*. Available: <https://developer.mozilla.org/>, 2025.

[5] OpenAI, *ChatGPT*. Available: <https://chat.openai.com>, 2025.

[6] Anthropic, *Claude AI*. Available: https://claude.ai, 2025.

# **PHỤ LỤC**

Frontend (Vue.js): <https://hmn011104.github.io/TaskFrontend/>

Backend API (Laravel): <https://taskbackend-vrp7.onrender.com/>

API Documentation: <https://taskbackend-vrp7.onrender.com/api/>

Live test: <https://hmn011104.github.io/TaskFrontend/>