**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**



**báo cáo môn   
Kiến trúc và thiết kế phần mềm**

**Đề tài: ĐĂNG KÝ HỌC PHẦN**

***Nhóm 35- Sinh viên thực hiện***

1. Lê Thị Thúy Kiều – 21108651
2. Nguyễn Trọng Đạt – 21113731
3. Nguyễn Trần Nhật Minh – 21003345

**MỤC LỤC**

[CHƯƠNG 1 : GIỚI THIỆU 3](#_Toc167044318)

[1.1 Mô tả yêu cầu cơ bản của hệ thống 3](#_Toc167044319)

[1.2 Mô tả yêu cầu chức năng 3](#_Toc167044320)

[1.3 Giả thuyết mới 5](#_Toc167044321)

[CHƯƠNG 2 : MÔ HÌNH MINH HOẠ CHO KIẾN TRÚC 5](#_Toc167044322)

[2.1 Các kiến trúc phần mềm phổ biến 5](#_Toc167044323)

[1. Layered architecture: 5](#_Toc167044324)

[2. Pipeline architecture 6](#_Toc167044325)

[3. Microkernel architecture 7](#_Toc167044326)

[4. Client-server architecture 8](#_Toc167044327)

[5. Service-oriented architecture 10](#_Toc167044328)

[6. Microservices architecture 11](#_Toc167044329)

[7. Space-based architecture 12](#_Toc167044330)

[2.2 Lý do chọn Architectural style là Client\_Server Architecture 13](#_Toc167044331)

[2.3 Nhược điểm của Client-server architecture 14](#_Toc167044332)

[2.4 Mô hình tổng quát 15](#_Toc167044333)

[2.5 Mô hình kiến trúc: (Client-server architecture) 15](#_Toc167044334)

[CHƯƠNG 3 : PHÂN TÍCH VÀ THIẾT KẾ 17](#_Toc167044335)

[3.1 Usecase tổng quát 17](#_Toc167044336)

[3.2 Class diagram 18](#_Toc167044337)

[3.3 Deployment diagram 19](#_Toc167044338)

[3.4 Package diagram 20](#_Toc167044339)

[3.5 Component diagram 21](#_Toc167044340)

[CHƯƠNG 4 : HIỆN THỰC 23](#_Toc167044341)

[4.1 Cấu hình phần cứng, phần mềm 23](#_Toc167044342)

[4.2 Giao diện của hệ thống 23](#_Toc167044343)

[CHƯƠNG 5 : KẾT LUẬN 27](#_Toc167044344)

[5.1 Kết quả đạt được: 27](#_Toc167044345)

[5.2 Hạn chế của đồ án: 28](#_Toc167044346)

[5.3 Hướng phát triển: 28](#_Toc167044347)

[TÀI LIỆU THAM KHẢO 28](#_Toc167044348)

# : GIỚI THIỆU

## Mô tả yêu cầu cơ bản của hệ thống

Một trường đại học cần quản lý hệ thống đăng ký học phần của sinh viên theo theo cơ chế tín chỉ. Các sinh viên sẽ được biên chế ở các khoa theo mỗi ngành học của mình đăng ký.Trong mỗi học kỳ, nhà trường sẽ chọn một số môn học của từng ngành học cho sinh viên đăng ký học. Sinh viên đăng ký theo học vào một lớp nhất định (lớp tín chỉ) với số sinh viên được đăng ký tối đa được qui định cho từng lớp.Môn học được mở có thể là môn học tự chọn Mỗi môn học có một hoặc nhiều môn học tiên quyết. Sinh viên sẽ không được đăng ký môn học mà có môn tiên quyết mình chưa học. Mỗi học kỳ sinh viên được đăng ký tối đa 30 tín chỉ. Nếu quá số này, hệ thống sẽ không cho đăng ký.

Để phòng ngừa trường hợp sinh viên đăng ký xong rồi hủy bỏ, nhà trường yêu cầu sinh viên phải xác nhận trước khi đăng ký. Vào ngày mở đăng ký, sinh viên sẽ đăng nhập vào hệ thống và sẽ nhìn thấy danh sách các môn học mà mình có khả năng đăng ký. Sinh viên chọn các môn học và tiến hành đăng ký. Trường hợp các lớp đã đầy, sinh viên sẽ được đưa vào một danh sách dự bị để nhà trường cân nhắc có mở thêm lớp hay không. Nếu không mở thêm lớp, sinh viên sẽ bị hủy đăng ký môn đó.

Sau khi đăng ký thành công, một email thông báo sẽ được gửi cho sinh viên xác nhận việc đăng ký và nhận quyết định đóng học phí.

Ngoài ra, hệ thống còn có khả năng cho phép sinh viên xem thông tin học tập của mình (số tín chỉ đã đạt, số môn đã học, điểm môn học, điểm trung bình tích lũy…), thời khóa biểu theo tuần và các tiện ích khác. Sau khi số tín chỉ đã đạt theo từng ngành, sinh viên có quyền đăng ký xét tốt nghiệp. Nếu mọi tiêu chuẩn đều thỏa, sinh viên sẽ được cấp bằng tốt nghiệp và sẽ được đưa vào danh sách các cựu sinh viên. Thông tin về bằng cấp sẽ được công khai trên trang web của nhà trường. Thông tin của cựu sinh viên sẽ được lưu giữ để theo dõi quá trình làm việc (nếu sinh viên đồng ý), làm các cuộc survey, cũng như nhiều hoạt động khác.

## Mô tả yêu cầu chức năng

1. Đăng nhập vào trang web: Sinh viên được 1 tài khoản bao gồm: tên tài khoản là mã số sinh viên, password là 123. Sinh viên được đổi mật khẩu với mail trường cấp.

2. Xem thông tin sinh viên: Sinh viên sau khi vào trang web sẽ xem được thông tin của chính mình bao gồm: mã số sinh viên, tên, niên khóa, lớp định danh, ngành học, quốc tịnh, nơi sinh,… Ngoài ra còn có phần cho sinh viên xem cái chức năng khác.

3. Xem lịch học: Sinh viên có thể xem lịch học và lịch thi của bản thân theo tuần, trong lịch bao gồm: mã lớp học phần, tên lớp, tên môn, tiết bao nhiêu bắt đầu và kết thúc để sinh viên có thể theo dõi chính xác

4. Xem điểm: Sinh viên có thể xem điểm tất cả môn học đã học theo từng kì, bao gồm điểm thường kì, thực hành(nếu có), giữa kì, cuối kì và điểm tổng trên thang 4, 10 ngay cả điểm trung bình tích lũy.

5. Đăng ký xét tốt nghiệp: Sau khi hoành thành hết tất cả các môn và chứng chỉ của ngành học thì sinh viên có quyền đăng ký xét tốt nghiệp, sẽ được cấp bằng tốt nghiệp và lưu vào danh sách cựu sinh viên (nếu sinh viên đồng ý sẽ được lưu giữ để theo dõi)

6. Đăng ký học phần:

a. Chọn lớp học phần: Khi vào trang đăng ký học phần, sinh viên sẽ được xem môn học theo kì (hiện những môn học mà sinh viên được phép đăng kí bao gồm trong chương trình khung và kì này môn đó có mở lớp và trừ những môn sinh viên đã đăng ký), sau khi chọn môn, sẽ hiện ra danh sách lớp học với các trạng thái như chưa mở, sẵn sàng, đã khóa, sinh viên chỉ được đăng kí những lớp có trạng thái sẵn sàng và số lượng chưa đủ. Còn những môn có điều kiện tiên quyết thì sinh viên phải học môn tiên quyết trước (sẽ hiện ở phần môn học). Sau khi chọn lớp đủ tiêu chí, hệ thống hiện ra chi tiết lớp học phần bao gôm cả lý thuyết lẫn thực hành.

b. Kiểm tra trùng lịch: Hệ thống sẽ kiểm tra trùng lịch của lớp học phần đang được chọn và kiểm tra trùng lịch với tất cả các lớp mà sinh viên đăng kí trong kì và năm hiện tại (lịch lý thuyết lẫn thực hành), nếu trùng thì sẽ hiện lên lịch bị trùng ấy cho sinh viên.

c. Đăng ký học phần: Sau khi chọn học phần và kiểm tra trùng lịch, sinh viên nhấn đăng ký học phần, hệ thống sẽ thêm xuống database bao gồm (dữ liệu của sinh viên và nối với lớp học phần đã chọn, tạo ra kết quả học tập mới cho lớp học phần ấy, thêm lịch học của lớp học phần vào lịch học sinh viên)

d. Xem lớp học phần đã đăng ký: Sinh viên sẽ được xem lớp học phần đã đăng kí của mình theo kì và năm hiện tại bao gồm xem chi tiết lớp học phần đó

e. Hủy lớp học phần đã đăng ký: Sinh viên sẽ được hủy lớp học phần đã đăng ký của mình (chỉ được hủy khi nó ở trang thái sẵn sàng) và sau khi hủy môn học đó sẽ hiện lại cho sinh viên đăng ký.

7. Xem danh sách môn học: admin sẽ vào giao diện admin và xem được danh sách của tất cả môn học theo ngành

8. Thêm môn học: admin sẽ thêm môn học mới với id(được tạo ra) và các thuộc tính khác như tên, ngành, số tín chỉ, ….

9. Xem danh sách lớp học phần theo môn học

10. Thêm lớp học phần: admin thêm lớp học phần với các thuộc tính của lớp như: mã, tên, sĩ số, giáo viên, lớp thực hành (nếu có), lịch học, ….

11. Xóa lớp học phần: admin có thể xóa lớp học phần nếu sĩ số đăng ký không đủ để mở lớp.

12. Thay đổi trạng thái lớp học phần: admin có thể thay đổi trạng thái lớp học phần nhất định (INACTIVE ->ACTIVE -> FINISHED)

13. Gửi mail khi đăng ký thành công: hệ thống sẽ gửi mail cho sinh viên khi đăng ký thành công bao gồm cả học phí

## Giả thuyết mới

Ứng dụng đăng ký học phần của bạn đang được sử dụng bởi một trường đại học lớn. Vào đầu mỗi học kỳ, hàng ngàn sinh viên đồng loạt truy cập hệ thống để đăng ký các môn học. Điều này gây ra tình trạng quá tải cho máy chủ, khiến ứng dụng chạy chậm, thậm chí bị treo. Sinh viên không thể đăng ký được môn học, gây ra sự bất mãn và ảnh hưởng đến tiến độ học tập.

**Và ta sử dụng 2 trong các cách để thực hiện trong tình huống này**

**1. Cân bằng tải (Load Balancing):**

Cân bằng tải là kỹ thuật phân phối lưu lượng truy cập đều đặn giữa nhiều máy chủ. Khi có một lượng lớn người dùng truy cập vào trang web, hệ thống cân bằng tải sẽ chuyển hướng yêu cầu của họ đến các máy chủ khác nhau, đảm bảo không có máy chủ nào bị quá tải.

**2. Giới hạn tốc độ (Rate Limiting):**

Giới hạn tốc độ là kỹ thuật kiểm soát số lượng yêu cầu mà một người dùng hoặc một địa chỉ IP có thể gửi đến máy chủ trong một khoảng thời gian nhất định. Điều này giúp ngăn chặn các cuộc tấn công từ chối dịch vụ (DoS) và giảm tải cho máy chủ khi có lượng truy cập đột biến.

Có nhiều cách để giới hạn tốc độ, chẳng hạn như giới hạn số lượng yêu cầu mỗi giây, mỗi phút, hoặc mỗi giờ.

**Cân bằng tải trong kiến trúc client-server:**

* **Cân bằng tải phía máy chủ:**
  + Đặt một bộ cân bằng tải (load balancer) trước các máy chủ ứng dụng. Bộ cân bằng tải này sẽ nhận các yêu cầu từ client và phân phối chúng đến các máy chủ ứng dụng khác nhau dựa trên các thuật toán cân bằng tải (như round-robin, least connections, hoặc IP hash).
  + Các máy chủ ứng dụng có thể được đặt trên cùng một máy chủ vật lý hoặc trên các máy chủ vật lý khác nhau.
  + Giải pháp này giúp tăng khả năng mở rộng và tính sẵn sàng của ứng dụng, đảm bảo rằng không có máy chủ nào bị quá tải.
* **Cân bằng tải phía client:**
  + Client có thể được cấu hình để gửi yêu cầu đến nhiều máy chủ ứng dụng khác nhau.

**Giới hạn tốc độ trong kiến trúc client-server:**

* Giới hạn tốc độ phía máy chủ:
  + Cấu hình máy chủ web (Apache) để giới hạn số lượng yêu cầu từ một địa chỉ IP hoặc một người dùng trong một khoảng thời gian nhất định.
  + Sử dụng tường lửa ứng dụng web (WAF) để giới hạn tốc độ và bảo vệ ứng dụng khỏi các cuộc tấn công.
* Giới hạn tốc độ phía client:
  + Triển khai logic giới hạn tốc độ trong ứng dụng client.
  + Giải pháp này có thể giúp giảm tải cho máy chủ, nhưng yêu cầu client phải có khả năng thực hiện giới hạn tốc độ.

**3.Tại sao client-server lại tốt trong giả thuyết này**

Kiến trúc client-server không hẳn thực hiện tốt hơn các kiến trúc khác trong việc cân bằng tải và giới hạn tốc độ, nhưng nó có những ưu điểm nhất định giúp việc triển khai các kỹ thuật này trở nên dễ dàng và hiệu quả hơn.

**Ưu điểm của kiến trúc client-server:**

* + - Phân tách rõ ràng: Kiến trúc client-server phân tách rõ ràng giữa máy chủ và máy khách, giúp việc triển khai cân bằng tải và giới hạn tốc độ trở nên dễ dàng hơn. Ví dụ, bạn có thể đặt một bộ cân bằng tải trước các máy chủ ứng dụng để phân phối lưu lượng truy cập, hoặc cấu hình máy chủ web để giới hạn tốc độ truy cập.
    - Khả năng mở rộng: Kiến trúc client-server có khả năng mở rộng tốt. Bạn có thể dễ dàng thêm nhiều máy chủ ứng dụng để đáp ứng nhu cầu ngày càng tăng. Điều này đặc biệt quan trọng khi bạn cần xử lý lượng truy cập lớn.
    - Dễ bảo trì: Kiến trúc client-server dễ bảo trì hơn so với các kiến trúc khác. Bạn có thể cập nhật hoặc thay thế các máy chủ ứng dụng mà không ảnh hưởng đến máy khách.

# : MÔ HÌNH MINH HOẠ CHO KIẾN TRÚC

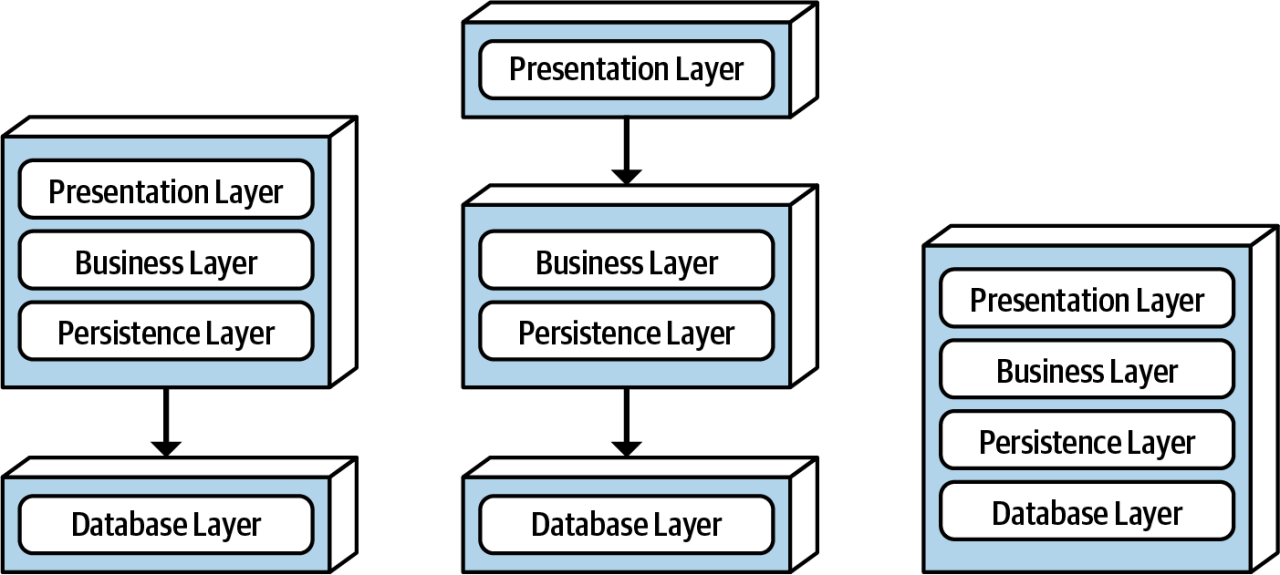
## Các kiến trúc phần mềm phổ biến

1. **Layered architecture:**

Là một cách chia ứng dụng thành các lớp logic riêng biệt để trị mối quan tâm, đóng gói và tách riêng, bằng cách nhóm các đơn vị mã bằng vai trò chức năng rõ ràng của chúng trong ứng dụng.

Thành phần chính:

* Presentation Layer (Lớp Giao diện): Còn được gọi là UI layer. Đây là lớp tương tác với người dùng, hiển thị thông tin và nhận dữ liệu đầu vào.
* Business Logic Layer (Lớp Logic Nghiệp vụ): Đảm nhiệm các quy trình nghiệp vụ phức tạp và các quy tắc kinh doanh của ứng dụng. Thường kết hợp với application layer hoặc có thể được tách riêng trong các hệ thống phức tạp.
* Data Access Layer (Lớp Truy cập Dữ liệu): Chịu trách nhiệm truy xuất dữ liệu từ các nguồn dữ liệu, như cơ sở dữ liệu, file, hoặc các dịch vụ khác. Thực hiện các thao tác CRUD (Create, Read, Update, Delete) và giao tiếp với database layer.
* Database Layer (Lớp Cơ sở Dữ liệu): Chứa các cơ sở dữ liệu và hệ thống quản lý cơ sở dữ liệu (DBMS). Lưu trữ dữ liệu thực tế của hệ thống.



*Hình 1. Mô hình Layered architecture*

Ưu điểm:

* Đơn giản, dễ học và dễ sử dụng.
* Giảm sự phụ thuộc do chức năng của mỗi lớp tách biệt với các lớp khác.
* Kiểm tra dễ dàng hơn vì các thành phần riêng biệt; mỗi thành phần có thể được kiểm tra riêng lẻ.
* Chi phí chung khá thấp.

Nhược điểm:

* Khả năng mở rộng khó khăn vì cấu trúc của framework không cho phép phát triển.
* Khó bảo trì, một thay đổi trong một lớp có thể ảnh hưởng đến toàn bộ hệ thống vì nó hoạt động như một đơn vị duy nhất.
* Có sự phụ thuộc lẫn nhau giữa các lớp vì một lớp phụ thuộc vào lớp bên trên nó để nhận dữ liệu.
* Không thể xử lý song song.

1. **Pipeline architecture**

Là một phương pháp sắp xếp các nhiệm vụ hoặc quy trình thành một chuỗi các bước.

A diagram of a flowchart

Description automatically generated

*Hình 2. Mô hình Pipeline architecture*

Có hai loại chính của kiến trúc pipeline:

* Pipeline dữ liệu: Được sử dụng để xử lý và chuyển đổi một lượng lớn dữ liệu. Chúng phổ biến trong khoa học dữ liệu và học máy.
* Pipeline hướng dẫn: Tìm thấy trong bộ xử lý máy tính. Chúng chia nhỏ các hướng dẫn thành các giai đoạn để cải thiện tốc độ xử lý.

Ưu điểm: Tăng hiệu suất, giảm thời gian chờ, tăng tính linh hoạt và dễ dàng gỡ lỗi.

Nhược điểm: Tăng độ phức tạp, yêu cầu phụ thuộc dữ liệu, giảm hiệu quả cho nhiệm vụ nhỏ và yêu cầu đồng bộ hóa.

1. **Microkernel architecture**

Bao gồm 2 thành phần: hệ thống lõi (core system) và các thành phần plug-in

Logic ứng dụng được phân chia giữa các thành phần plug-in độc lập và hệ thống lõi cơ bản, cung cấp khả năng mở rộng, khả năng thích ứng và cách ly các tính năng ứng dụng cũng như logic xử lý tùy chỉnh.

A diagram of a computer system

Description automatically generated

*Hình 3. Mô hình Microkernel architecture*

Ưu điểm: Đơn giản, dễ bảo trì và nâng cấp, chi phí chung khá thấp

Nhược điểm: Khả năng mở rộng kém, khả năng chịu lỗi thấp

1. **Client-server architecture**

Thành phần chính của Client-server architecture

* Client: Là các thiết bị hoặc phần mềm gửi yêu cầu đến server để truy cập dịch vụ hoặc tài nguyên. Ví dụ: trình duyệt web, ứng dụng di động, hoặc ứng dụng máy tính.
* Server: Là các thiết bị hoặc phần mềm cung cấp dịch vụ hoặc tài nguyên cho các client. Server xử lý các yêu cầu từ client và trả về kết quả tương ứng.

A diagram of a computer

Description automatically generated

*Hình 4. Mô hình Client-server architecture*

Cơ chế hoạt động:

* Yêu cầu (Request): Client gửi một yêu cầu đến server thông qua mạng (Internet, mạng cục bộ, …). Yêu cầu có thể là yêu cầu dữ liệu, gửi dữ liệu để xử lý, hoặc yêu cầu thực hiện một hành động nào đó.
* Xử lý (Processing): Server nhận yêu cầu từ client và xử lý yêu cầu đó. Quá trình xử lý có thể bao gồm truy vấn cơ sở dữ liệu, thực hiện tính toán, hoặc tương tác với các dịch vụ khác.
* Phản hồi (Response): Sau khi xử lý, server gửi lại kết quả cho client. Kết quả có thể là dữ liệu, trang web, hoặc thông báo về kết quả của hành động đã yêu cầu.

Ưu điểm:

* Dữ liệu và tài nguyên được lưu trữ và quản lý tập trung tại server, giúp dễ dàng bảo trì, sao lưu và bảo vệ dữ liệu.
* Các cập nhật phần mềm và bảo mật chỉ cần được thực hiện trên server thay vì từng client.
* Dữ liệu và các quy trình quan trọng được bảo vệ tại server, nơi có thể áp dụng các chính sách bảo mật nghiêm ngặt.
* Việc bảo trì, cập nhật và nâng cấp hệ thống dễ dàng hơn vì các thay đổi chỉ cần thực hiện trên server.

Nhược điểm:

* Cần các công cụ và quy trình phức tạp để giám sát, bảo mật và duy trì hệ thống.
* Nếu số lượng client tăng cao, server có thể trở thành điểm tắc nghẽn, làm giảm hiệu suất của toàn bộ hệ thống.
* Chi phí cho việc thiết lập và duy trì server có thể cao, bao gồm phần cứng, phần mềm, và nhân lực quản lý.

1. **Service-oriented architecture**

Là một phương pháp thiết kế hệ thống phần mềm mà trong đó các thành phần chính được tổ chức thành các dịch vụ độc lập có thể tương tác với nhau qua mạng. Mỗi dịch vụ thực hiện một chức năng cụ thể và có thể được sử dụng bởi các dịch vụ khác hoặc bởi các ứng dụng người dùng.

Các thành phần chính của SOA

* Giao diện người dùng (User Interface): Được triển khai riêng biệt, giao diện người dùng là phần mà người dùng tương tác trực tiếp.
* Các dịch vụ chi tiết thô từ xa (Remote Fine-Grained Services): Được triển khai riêng biệt, các dịch vụ này thực hiện các chức năng cụ thể và chi tiết. Các dịch vụ này thường nhỏ và chuyên về một nhiệm vụ cụ thể, như xác thực người dùng, xử lý thanh toán, hoặc quản lý đơn hàng.
* Cơ sở dữ liệu nguyên khối (Monolithic Database): Một cơ sở dữ liệu lớn và tập trung chứa tất cả dữ liệu của hệ thống. Dữ liệu được chia sẻ giữa các dịch vụ qua cơ sở dữ liệu này.

A diagram of a company

Description automatically generated

*Hình 5. Mô hình Service-oriented architecture*

Cách hoạt động của SOA

* Yêu cầu từ giao diện người dùng: Khi người dùng thực hiện một hành động (như đăng nhập, mua hàng), giao diện người dùng sẽ gửi yêu cầu đến các dịch vụ liên quan.
* Xử lý yêu cầu bởi các dịch vụ: Các dịch vụ chi tiết thô từ xa nhận yêu cầu và xử lý chúng. Dịch vụ có thể cần tương tác với các dịch vụ khác hoặc truy vấn cơ sở dữ liệu để hoàn thành nhiệm vụ.
* Phản hồi kết quả: Sau khi hoàn thành xử lý, dịch vụ gửi kết quả lại cho giao diện người dùng. Giao diện người dùng hiển thị kết quả cho người dùng.

Ưu điểm:

* Thêm các dịch vụ mới hoặc thay thế các dịch vụ hiện tại mà không ảnh hưởng đến toàn bộ hệ thống
* Tích hợp các hệ thống khác nhau, bất kể ngôn ngữ lập trình hay nền tảng sử dụng.

Nhược điểm:

* Quản lý phức tạp, chi phí cao
* Mặc dù SOA giúp tích hợp các hệ thống khác nhau, việc này có thể tạo ra sự phụ thuộc phức tạp giữa các dịch vụ.
* Nếu một dịch vụ thay đổi hoặc gặp sự cố, có thể ảnh hưởng đến các dịch vụ khác trong hệ thống.

1. **Microservices architecture**

Là một phong cách kiến trúc phần mềm trong đó một ứng dụng lớn được chia thành nhiều dịch vụ nhỏ, độc lập, và tự trị. Mỗi microservice thực hiện một chức năng cụ thể và giao tiếp với các microservice khác thông qua các giao thức mạng nhẹ, thường là HTTP/REST hoặc gRPC. Phong cách kiến trúc này nhằm mục đích tăng tính linh hoạt, khả năng mở rộng, và khả năng bảo trì của ứng dụng.

A diagram of a service

Description automatically generated

*Hình 6. Mô hình Microservices architecture*

Ưu điểm:

* Tính linh hoạt và khả năng mở rộng cao
* Dễ bảo trì và phát triển: Các nhóm nhỏ có thể làm việc độc lập trên các microservice khác nhau, giảm thiểu độ phức tạp và tăng tốc độ phát triển.
* Tính chịu lỗi cao: Lỗi trong một microservice không ảnh hưởng đến toàn bộ hệ thống, giúp tăng tính ổn định và khả năng chịu lỗi của ứng dụng.

Nhược điểm

* Độ phức tạp quản lý: Quản lý nhiều microservice độc lập có thể phức tạp, đòi hỏi các công cụ và quy trình chuyên biệt để quản lý triển khai, giám sát, và bảo mật.
* Giao tiếp giữa các microservice qua mạng có thể làm giảm hiệu suất và tăng độ trễ, cần có các giải pháp để quản lý hiệu quả.
* Dữ liệu được phân tán qua nhiều microservice, việc duy trì tính nhất quán và quản lý các giao dịch trở nên khó khăn hơn.
* Chi phí hạ tầng cao

1. **Space-based architecture**

Là một phương pháp thiết kế hệ thống phần mềm mà trong đó các chức năng và quy trình nghiệp vụ được triển khai dưới dạng các dịch vụ độc lập. Các dịch vụ này thường giao tiếp với nhau thông qua các giao thức mạng tiêu chuẩn và được kết hợp lại để tạo thành các ứng dụng hoàn chỉnh.

A diagram of a user interface

Description automatically generated

*Hình 7. Mô hình Space-based architecture*

Ưu điểm: Việc chia ứng dụng thành các dịch vụ miền được triển khai riêng biệt bằng cách sử dụng kiểu kiến trúc này cho phép thay đổi nhanh hơn (linh hoạt), phạm vi kiểm thử tốt hơn do phạm vi giới hạn của miền (khả năng kiểm thử) và khả năng triển khai thường xuyên hơn với ít rủi ro hơn so với một ứng dụng lớn nguyên khối (khả năng triển khai).

Nhược điểm: quản lý phức tạp, việc quản lý nhiều dịch vụ độc lập đòi hỏi các công cụ và quy trình phức tạp hơn.

## Lý do chọn Architectural style là Client\_Server Architecture

Bên phía server ta sử dụng: Architectural patternt - Layered architecture

Client-Server Architecture: Đây là mô hình chủ đạo, phân chia rõ ràng giữa giao diện người dùng (client) và xử lý logic/dữ liệu (server).

Layered Architecture (N-tier) trong Back-end (Server): Việc chia back-end thành các lớp (Controller, Service, Repository) giúp tăng tính tổ chức, dễ bảo trì. Mỗi lớp có trách nhiệm riêng, giảm sự phụ thuộc lẫn nhau.

* Controller Layer: Xử lý các yêu cầu HTTP từ client và gọi đến Service Layer.
* Service Layer: Thực hiện các logic nghiệp vụ chính của ứng dụng, tương tác với Repository Layer nếu cần dữ liệu.
* Repository Layer: Tương tác với database, thực hiện các thao tác CRUD (Create, Read, Update, Delete).

Front-end (Client): Sử dụng HTML, JS, và Fetch API là cách tiếp cận phổ biến cho các ứng dụng web hiện đại

Mở rộng theo chiều ngang (scale-out): Bạn có thể dễ dàng thêm nhiều máy chủ để xử lý lượng truy cập lớn.

Mở rộng theo chiều dọc (scale-up): Bạn có thể nâng cấp phần cứng của máy chủ hiện tại để tăng hiệu suất.

Mở rộng từng thành phần: Bạn có thể mở rộng từng lớp trong back-end (ví dụ: thêm nhiều service instances) để đáp ứng nhu cầu cụ thể

Client-Server: Giảm tải cho server bằng cách xử lý một phần logic ở phía client.

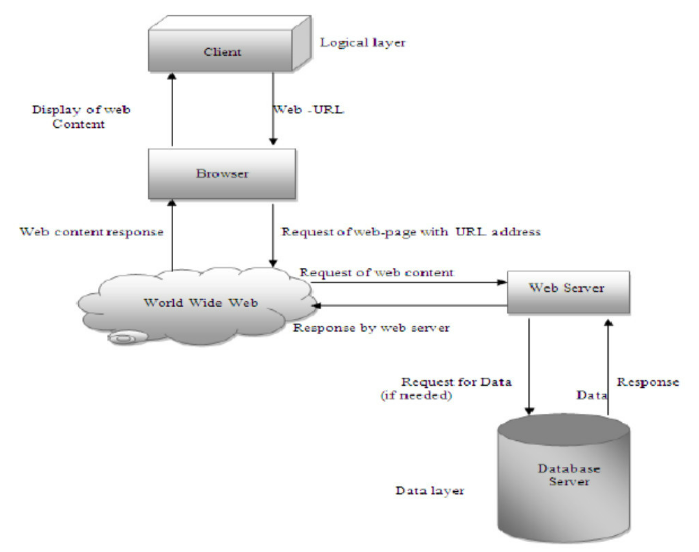
Layered Architecture**:** Giúp tối ưu hóa việc truy vấn và xử lý dữ liệu

* + - Front-end và back-end có thể phát triển độc lập, sử dụng công nghệ khác nhau.
    - Layered architecture patternt là 1 lựa chọn tốt vì nó đơn giản, dễ làm, chi phí thấp và có thể hoàn thành trong thời gian ngắn, đếu rất phù hợp với hệ thống cần làm. Và chúng tôi còn là Agile development team “just starts coding”. Hệ thống “đăng ký học phần” cũng không cần phải bảo trì và nâng cấp liên tục, việc mở rộng 1 chức năng của phần mềm này là khó có thể xảy ra.
    - Layered architecture cũng là “kiểu” kiến trúc được hiểu biết rộng rãi, có nhiều tài liệu và ví dụ phù hợp với hoàn cảnh của nhóm chúng tôi, việc đã được tiếp cận trước khiến giảm bớt gánh nặng về các khó khăn khác.

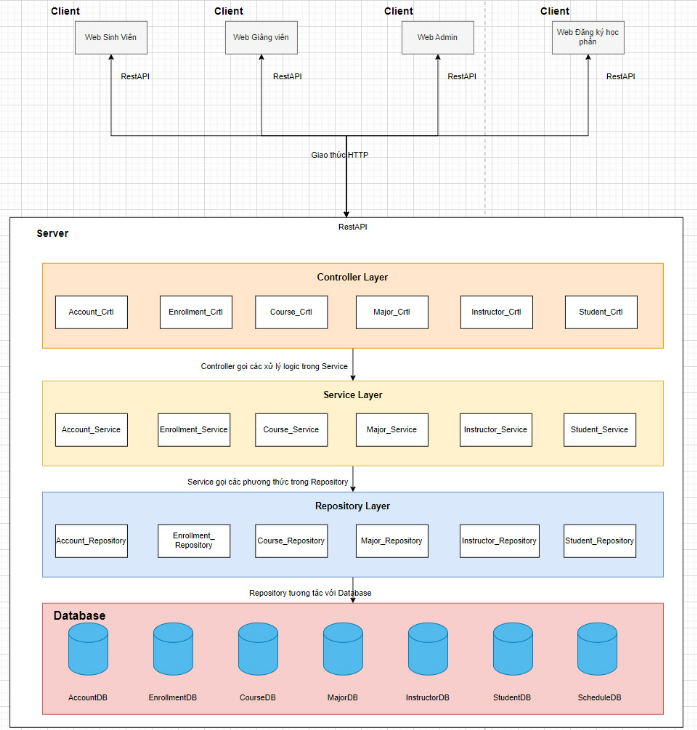
## Nhược điểm của Client-server architecture

* Độ phức tạp tăng:
  + - Nhiều lớp và thành phần: Việc chia nhỏ ứng dụng thành nhiều lớp và thành phần có thể làm tăng độ phức tạp của hệ thống, đặc biệt đối với các dự án nhỏ hoặc đơn giản. Điều này đòi hỏi sự quản lý và tổ chức cẩn thận để tránh sự rối rắm và khó hiểu.
* Giao tiếp giữa các lớp: Việc đảm bảo sự giao tiếp chính xác và hiệu quả giữa các lớp có thể là một thách thức. Cần phải thiết kế các interface và cơ chế truyền dữ liệu rõ ràng để tránh các lỗi và sự không tương thích.
* Hiệu suất có thể bị ảnh hưởng:
* Nhiều lớp xử lý: Mỗi yêu cầu từ client phải đi qua nhiều lớp xử lý trước khi đến được database và trả về kết quả. Điều này có thể làm giảm hiệu suất của ứng dụng, đặc biệt khi có nhiều lớp xử lý phức tạp hoặc khi lưu lượng truy cập lớn.
  + Tăng độ trễ: Việc giao tiếp giữa client và server, cũng như giữa các lớp trong back-end, có thể tạo ra độ trễ nhất định. Điều này có thể ảnh hưởng đến trải nghiệm người dùng, đặc biệt đối với các ứng dụng yêu cầu phản hồi nhanh.
* Khó khăn trong việc thay đổi cấu trúc:
  + Phụ thuộc giữa các lớp: Các lớp trong back-end thường có sự phụ thuộc lẫn nhau. Do đó, việc thay đổi cấu trúc của một lớp có thể ảnh hưởng đến các lớp khác, đòi hỏi phải điều chỉnh và kiểm tra lại toàn bộ hệ thống.
* Tốn kém tài nguyên:
  + - Máy chủ mạnh: Mô hình Client-Server yêu cầu một máy chủ đủ mạnh để xử lý các yêu cầu từ nhiều client cùng lúc. Điều này có thể làm tăng chi phí đầu tư và vận hành.

## Mô hình tổng quát



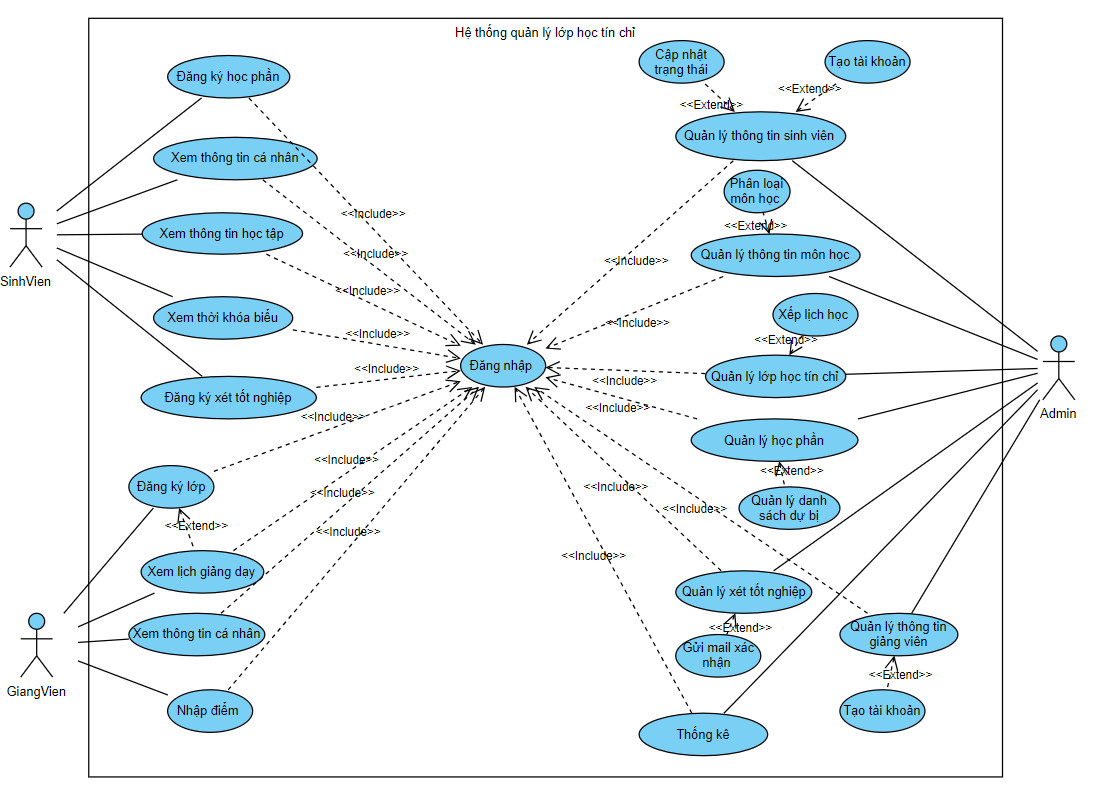
## Mô hình kiến trúc: (Client-server architecture)



*Hình 8. Mô hình kiến trúc Client-server đăng ký học phần*

# : PHÂN TÍCH VÀ THIẾT KẾ

## Usecase tổng quát

******

*Hình 9. Mô hình usecase tổng quát*

## Class diagram

A computer screen shot of a diagram

Description automatically generated

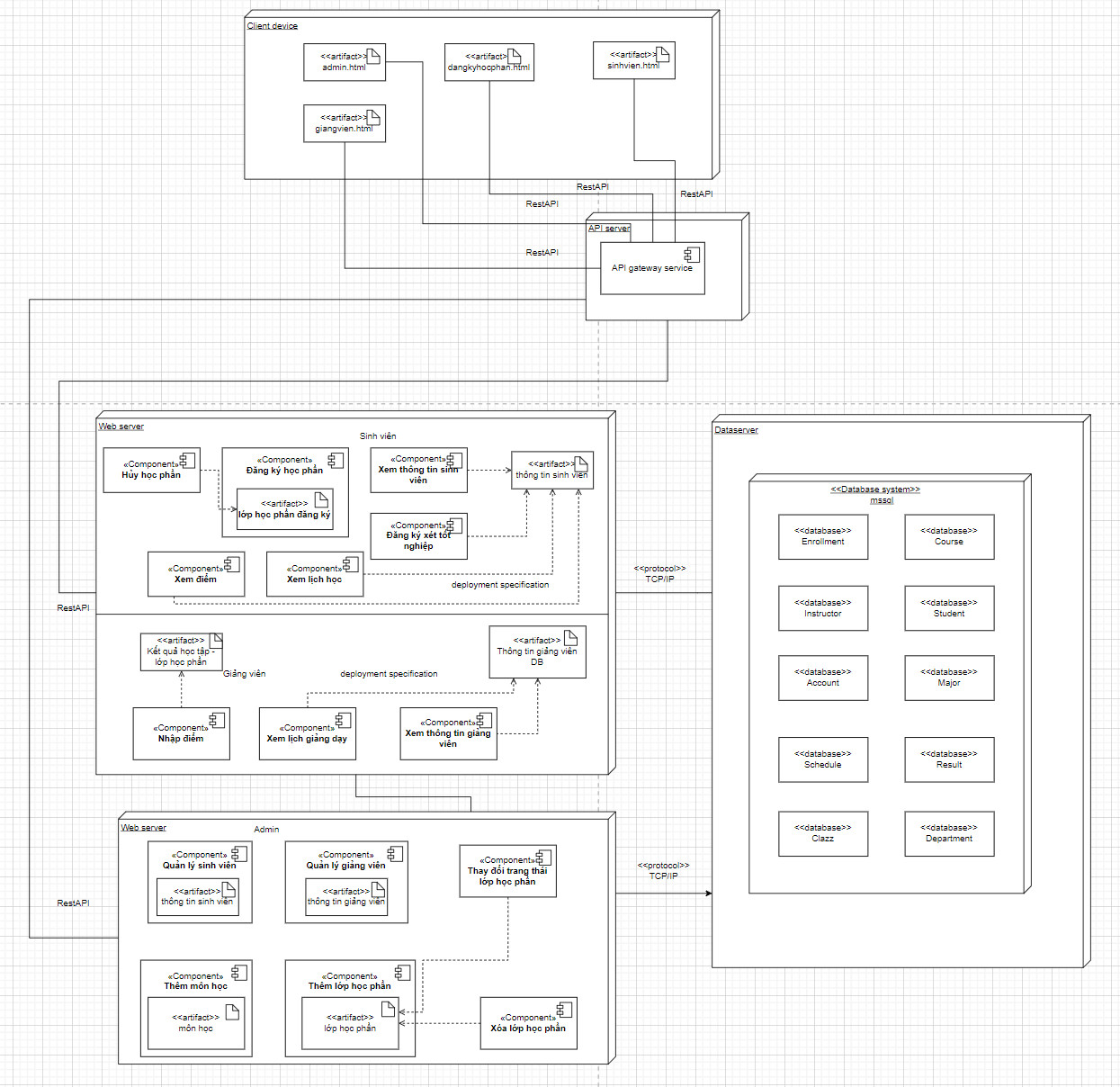
*Hình 10. Mô hình lớp*

A diagram of a computer program

Description automatically generated

## Deployment diagram

Mô hình triển khai hệ thống.



*Hình 11. Mô hình triển khai hệ thống*

## Package diagram

A diagram of a software application

Description automatically generated with medium confidence

*Hình 12. Mô hình package*

## Component diagram

A diagram of a computer

Description automatically generated

*Hình 13. Mô hình component*

# : HIỆN THỰC

## Cấu hình phần cứng, phần mềm

a. Máy Chủ (Server):

* CPU: Sử dụng CPU với ít nhất 4 lõi và hỗ trợ đa luồng để xử lý các yêu cầu đồng thời từ các layered.
* RAM: Đảm bảo có đủ bộ nhớ RAM để chạy các ứng dụng Java và các dịch vụ MSSQL một cách mượt mà. Tối thiểu 8GB RAM là một lựa chọn phù hợp.

b. Các Dịch Vụ Google Sử Dụng:

* Google Chrome: Để chạy giao diện fontend

## Giao diện của hệ thống

Màn hình chức năng đã thực hiện:

A screenshot of a login

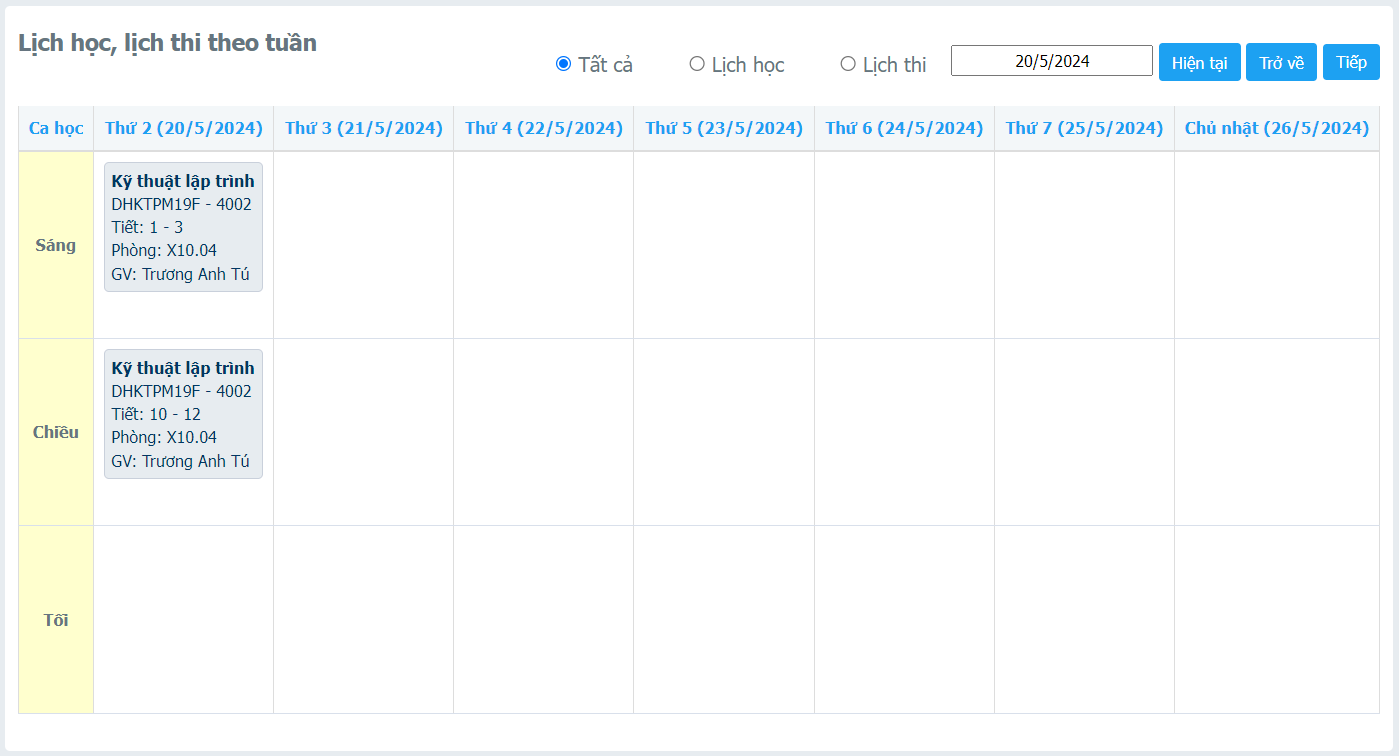
Description automatically generated

*Hình 14. Giao diện đăng nhập*

A screenshot of a computer

Description automatically generated

*Hình 15. Giao diện trang chủ trang sinh viên*



*Hình 16. Giao diện xem lịch*



*Hình 17. Giao diện xem điểm*



*Hình 18. Giao diện trang thông tin cá nhân của sinh viên*

A screenshot of a computer

Description automatically generated

*Hình 19. Giao diện đăng ký học phần của sinh viên*

A screenshot of a computer

Description automatically generated

*Hình 20. Giao diện xem chi tiết lớp học phần đã đăng ký và hủy đăng ký*

A screenshot of a computer

Description automatically generated

*Hình 21. Giao diện admin*

A screenshot of a computer

Description automatically generated

*Hình 22. Giao diện thêm môn học*

A screenshot of a computer

Description automatically generated

*Hình 23. Giao diện thêm lớp học phần*

# : KẾT LUẬN

## Kết quả đạt được:

Hoàn thành một số chức năng cơ bản như:

* Đăng ký học phần
* Hủy đăng ký
* Xem lịch học
* Đăng nhập
* Xem điểm
* Thêm môn học
* Thêm lớp học phần
* Thay đổi trạng thái lớp học phần

## Hạn chế của đồ án:

* Một số chức năng chưa hoàn thiện
* Chưa tối ưu hóa hiệu suất cho các tác vụ đồng thời.
* Cần cải thiện tính bảo mật để bảo vệ dữ liệu người dùng tốt hơn

## Hướng phát triển:

* Sử dụng Vue.js hoặc Svelte để xây dựng ứng dụng web, giúp cải thiện trải nghiệm người dùng với giao diện linh hoạt và mượt mà.
* Phát triển ứng dụng di động cho cả Android và iOS bằng Xamarin hoặc Kotlin Multiplatform Mobile (KMM) để đảm bảo hiệu suất cao và trải nghiệm người dùng đồng nhất trên các thiết bị.
* Thiết lập quy trình tự động hóa CI/CD với Jenkins hoặc GitHub Actions để duy trì hệ thống ổn định và dễ dàng triển khai các bản cập nhật.
* Tập trung vào thiết kế giao diện người dùng trực quan và dễ sử dụng với các công cụ như Figma hoặc Sketch.
* Sử dụng Keycloak hoặc Auth0 để xác thực người dùng và quản lý quyền truy cập, đảm bảo bảo mật và quản lý truy cập hiệu quả.

# TÀI LIỆU THAM KHẢO

Các tài liệu từ Internet

1. https://edwardthienhoang.wordpress.com/software-architecture/
2. Ant Design. Ant Design UI components Document [Online]. Từ <https://ant.design/components/overview/>
3. Ant Design. Ant Design UI components Document [Online]. Từ <https://ant.design/components/overview/>