BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THÀNH PHỐ HỒ CHÍ MINH

**KHOA khoa CÔNG NGHỆ THÔNG TIN**



**MÔN: KIẾN TRÚC PHẦN MỀM**

**ĐỀ CƯƠNG NGHIÊN CỨU**

***Đề tài:***

**ĐĂNG KÝ HỌC PHẦN**

***Nhóm 11 - Sinh viên thực hiện***

1. Lê Thị Kim Ngân – 20041421
2. Lê Thanh Toàn – 20040331
3. Phan Ngọc Nam – 19511731

Thành phố Hồ Chí Minh, 19 tháng 05 năm

**MỤC LỤC**

[**CHƯƠNG 1: CƠ SỞ LÝ THUYẾT** 2](#_Toc167038110)

[**1.1.** **Kiến trúc phần mềm là gì?** 2](#_Toc167038111)

[**1.1.1.** **Kiến trúc là gì?** 2](#_Toc167038112)

[**1.1.2.** **Kiến trúc phần mềm là gì?** 2](#_Toc167038113)

[**1.1.3.** **Xác định kiến trúc phần mềm** 2](#_Toc167038114)

[**1.2.** **Các mô hình kiến trúc phổ biến** 3](#_Toc167038115)

[**1.2.1.** **Mô hình kiến trúc Layered** 4](#_Toc167038116)

[**1.2.2.** **Mô hình kiến trúc Microservices** 5](#_Toc167038117)

[**1.2.3.** **Mô hình kiến trúc Monolithic** 6](#_Toc167038118)

[**CHƯƠNG 2: GIỚI THIỆU VỀ HỆ THỐNG QUẢN LÝ HỌC PHẦN TÍN CHỈ** 7](#_Toc167038119)

[**2.1.** **Mô tả yêu cầu cơ bản của hệ thống** 7](#_Toc167038120)

[**2.2. Lý do chọn mô hình Layered Architecture** 8](#_Toc167038121)

[**2.3. Ưu điểm và nhược điểm khi chọn Layered Architecture** 9](#_Toc167038122)

[**2.4. Sự đánh đổi khi chọn Layered Architecture** 9](#_Toc167038123)

[**CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG** 10](#_Toc167038124)

[**3.1. Usecase tổng quát** 10](#_Toc167038125)

[**3.2. Sơ đồ ERD** 11](#_Toc167038126)

[**3.3. Class diagram** 12](#_Toc167038127)

[**3.4. Package diagram** 14](#_Toc167038128)

[**3.5. Component diagram** 15](#_Toc167038129)

[**3.6. Deployment diagram** 15](#_Toc167038130)

[**3.7. Mô hình Layered Architecture** 16](#_Toc167038131)

[**CHƯƠNG 4: HIỆN THỰC** 16](#_Toc167038132)

[**4.1. Cấu hình phần cứng, phần mềm** 16](#_Toc167038133)

[**4.2. Giao diện của hệ thống** 17](#_Toc167038134)

[**4.2.1. Màn hình đăng nhập** 17](#_Toc167038135)

[**4.2.2. Màn hình Xem thông tin sinh viên** 18](#_Toc167038136)

[**4.2.3. Màn hình Xem kết quả học tập** 18](#_Toc167038137)

[**4.2.4. Màn hình Xem lịch học** 19](#_Toc167038138)

[**4.2.5. Màn hình Xem chương trình khung** 19](#_Toc167038139)

[**4.2.6. Màn hình Đăng ký học phần** 20](#_Toc167038140)

[**CHƯƠNG 5: KẾT LUẬN** 20](#_Toc167038141)

[**5.1. Kết quả đạt được** 20](#_Toc167038142)

[**5.2. Hạn chế** 21](#_Toc167038143)

[**5.3. Hướng phát triển** 21](#_Toc167038144)

# **CHƯƠNG 1: CƠ SỞ LÝ THUYẾT**

* 1. **Kiến trúc phần mềm là gì?**
     1. **Kiến trúc là gì?**

Kiến trúc là tổ chức cơ bản của một hệ thống, được thể hiện trong các thành phần của nó, mối quan hệ của chúng với nhau và với môi trường cũng như các nguyên tắc chi phối thiết kế và sự phát triển của nó.

* + 1. **Kiến trúc phần mềm là gì?**

Kiến trúc phần mềm của một chương trình hoặc hệ thống máy tính là cấu trúc hoặc các cấu trúc của hệ thống, bao gồm các thành phần phần mềm, các thuộc tính có thể nhìn thấy bên ngoài của các thành phần đó và mối quan hệ giữa chúng.

* + 1. **Xác định kiến trúc phần mềm**

Kiến trúc phần mềm bao gồm cấu trúc của hệ thống, kết hợp với các đặc điểm kiến trúc mà hệ thống phải hỗ trợ, các quyết định về kiến trúc và cuối cùng là các nguyên tác thiết kế.

A diagram of a structure

Description automatically generated

Hình 1: Xác định kiến trúc phần mềm

Cấu trúc của hệ thống (Structure of the system):

* Cấu trúc đề cập đến loại kiểu kiến trúc được sử dụng trong hệ thống như microservices, layered hoặc microkernel.
* Ví dụ: một hệ thống có thể được tổ chức theo mô hình microservices để tách biệt các chức năng khác nhau thành các dịch vụ độc lập, mỗi dịch vụ có thể phát triển, triển khai và mở rộng một cách độc lập.

Đặc điểm kiến trúc (architecture characteristics)

* Các đặc điểm kiến trúc xác định tiêu chí thành công của một hệ thống như: tính khả dụng, độ tin cậy, khả năng kiểm tra được, khả năng mở rộng, tính bảo mật, tính nhanh nhẹn, khả năng chịu lỗi, tính mềm dẻo, khả năng phục hồi, hiệu suất, khả năng triển khai, khả năng học hỏi.
* Ví dụ: một hệ thống ngân hàng trực tuyến có thể đặt yêu cầu về tính khả dụng và độ tin cậy cao, tính bảo mật cao cùng với hiệu suất và khả năng mở rộng linh hoạt.

Quyết định kiến trúc (architecture decisions)

* Các quyết định về kiến trúc xác định các quy tắc về cách xây dựng một hệ thống
* Ví dụ: một kiến trúc sư có thể đưa ra quyết định về kiến trúc mà chỉ các lớp nghiệp vụ và dịch vụ trong kiến trúc layered mới có thể truy cập cơ sở dữ liệu

Nguyên tắc thiết kế (design principles)

* Nguyên tắc thiết kế là các nguyên tắc tổng quát hoặc quy tắc cụ thể được áp dụng trong quá trình thiết kế hệ thống phần mềm để đảm bảo rằng hệ thống được xây dựng một cách hiệu quả, dễ bảo trì và mở rộng.
* Ví dụ:
  + Nguyên tắc open/closed (OCP): phần mềm nên được thiết kế sao cho có thể mở rộng được mà không cần sửa đổi các phần đã hoàn thành
  + Nguyên tắc single responsibility (SRP): mỗi lớp hoặc module trong hệ thống nên chỉ có một trách nhiệm duy nhất
  + Nguyên tắc liskov substitution (LSP): đối tượng thừa kế nên có thể được thay thế bằng đối tượng của lớp cơ sở mà không làm thay đổi tính đúng đắn của chương trình.
  + Nguyên tắc interface segregation (ISP): khách hàng không nên bị cưỡng chế để sử dụng các phương thức mà họ không cần
  + Nguyên tắc dependency inversion (DIP): các module cấp cao không nên phụ thuộc vào các module cấp thấp. Cả hai nên phụ thuộc vào các abstraction.
  1. **Các mô hình kiến trúc phổ biến**
     1. **Mô hình kiến trúc Layered**
        1. **Khái niệm**

Kiến trúc layerd là một kiến trúc phổ biến trong xây dựng phần mêm, trong kiến trúc này các mối quan tâm như UI, bussiness logic, lưu trữ được chia thành các tầng riêng biệt. Bằng cách này việc thay đổi ở một tầng sẽ không trực tiếp ảnh hưởng đến các tầng còn lại, tăng tính module của ứng dụng.

* + - 1. **Sơ đồ**

A screenshot of a computer application

Description automatically generated

Hình 2: Mô hình kiến trúc layered

* + - 1. **Ưu điểm**
* Đơn giản dễ hiện thực, triển khai.
* Giảm sự phụ thuộc giữa các lớp, thay đổi ở một lớp sẽ không trực tiếp tác động đến các lớp còn lại, tăng tính lỏng lẻo của ứng dụng
* Kiểm thử dễ dàng, các tầng có thể được kiểm thử riêng lẻ
* Chi phí để phát triển một ứng dụng theo kiến trúc layerd tương đối thấp khi so với các kiến trúc khác.
  + - 1. **Nhược điểm**
* Khả năng mở rộng khó khăn vì cấu trúc của layerd không cho phép scale. khó bảo trì khi code base ngày càng được mở rộng
* Vẫn có sự phụ thuôc nhất định giữa các lớp, việc thay thế hoàn toàn một lớp mà không ảnh hưởng đến các lớp còn lại là không thực sự dễ dàng.
* Xử lý song song là không thể trong ứng dụng triển khai theo kiến trúc layerd.
  + 1. **Mô hình kiến trúc Microservices**
       1. **Khái niệm**

Kiến trúc microservices chia ứng dụng thành các service nhỏ, độc lập. Giao tiếp giữa các service sẽ thông qua API hoặc messaging service.

* + - 1. **Sơ đồ**

A diagram of a process

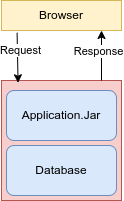
Description automatically generated

Hình 3: Mô hình kiến trúc microservices

* + - 1. **Ưu điểm**
      * Scalability: Vì các service là độc lập, việc scale một phần của ứng dụng trở nên dễ dàng.
      * Fault isolation: Việc một phần của ứng dụng bị lỗi sẽ không làm ảnh hưởng đến toàn bộ ứng dụng
      * Flexibility: Việc chia nhỏ ứng dụng thành các service nhỏ có thể giúp cho các service có thể sử dụng các công nghệ khác nhau, nhiều team với nhiều công nghệ khác nhau có thể cùng sử dụng trong quá trình phát triển microservices
      1. **Nhược điểm**
      * Latency: Việc giao tiếp giữa các service trên môi trường internet có thể làm giảm hiệu năng của ứng dụng, khi này các yếu tố khác như tốc độ mạng, băng thông có thể làm giảm đáng kể performance.
      * Testiability: Việc test trở nên khó khăn khi toàn bộ ứng dụng bị tác nhỏ thành các service nhỏ.
      * Transaction: Việc quản lý transaction là khó khăn, khi mà các database cũng bị chia nhỏ theo từng service. Ngoài ra việc debug và phát hiện lỗi cũng trở nên khó khăn hơn
    1. **Mô hình kiến trúc Monolithic**
       1. **Khái niệm**

Monolithic architecture là một kiểu kiến trúc truyền thống được sử dụng lâu đời trong ngành công nghiệp phần mềm. Các ứng dụng sử dụng được xây dựng dựa vào kiến trúc này được đóng gói tất cả thành phần vào một khối độc lập và duy nhất.

* + - 1. **Sơ đồ**



Hình 4: Mô hình kiến trúc Monolithic

* + - 1. **Ưu điểm**
* Simplicity**:** monolithic là kiến trúc dễ dàng triển khai về mặt kỹ thuật, developer có thể bắt tay ngay vào việc xây dựng sản phẩm mà không phải làm việc với các setup phức tạp.
* Performance**:** Vì tất cả các thành của một dự án được xây dựng trên kiến trúc monolithic được gói gọn, tập trung vào một codebase duy nhất, việc giao tiếp giữa các thành phần trở nên nhanh chóng khi giảm thiểu được độ trễ khi so sánh với các kiến trúc phân tán
* Deployability: Dễ dàng triển khai ứng dụng.
  + - 1. **Nhược điểm**
      * Scalability: Mở rộng một phần nhất định của ứng dụng là khó khăn khi ta cần phải mở rộng toàn bộ ứng dụng.
      * Depenability: Nếu một phần của ứng dụng gặp vấn đề, có khả năng ảnh hưởng rất lớn đến toàn bộ ứng dụng
      * Flexibility: Việc tích hợp hay sử dụng các công cụ, công nghệ khác trong một dự án là không tưởng. Về cơ bản 1 dự án sẽ bị gắn cứng với các công cụ công nghệ đã sử dụng từ lúc ban đầu.

**CHƯƠNG 2: GIỚI THIỆU VỀ HỆ THỐNG QUẢN LÝ HỌC PHẦN TÍN CHỈ**

## **2.1.** **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 cơ chế tín chỉ. Mỗi sinh viên khi đăng ký nhập học sẽ được cung cấp tài khoản đăng nhập gồm tài khoản là mã sinh viên, mật khẩu mặc định là 1111. Các sinh viên sẽ có chương trình học khác nhau tùy theo mỗi khoa và chuyên ngành mà mình đăng ký. Khi đăng nhập vào trang web của trường sinh viên có thể xem được thông tin cá nhân, kết quả học tập, lịch học theo tuần, chương trình khung, công nợ và thực hiện đăng ký học phần.

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 là 60. Trong đó, đối với môn có thực hành thì số lượng tối đa là 20 sinh viên cho mỗi nhóm. Sau một khoảng thời gian quy định, nếu số lượng đăng ký ít hơn quy định thì lớp học phần sẽ bị hủy.

Môn học được mở có thể là môn học tự chọn, và mỗi môn học có thể có một hoặc nhiều môn tiên quyết, sinh viên sẽ không được đăng ký môn học mà có môn tiên quyết mình chưa học. Và nếu như lịch học của môn học trùng với lịch học của môn khác đã đăng ký thì sinh viên sẽ không được phép đăng ký môn học đó.

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. 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.

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 mãn, 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, làm các cuộc surver cũng như nhiều hoạt động khác.

## **2.2. Lý do chọn mô hình Layered Architecture**

Sau khi phân tích mô tả yêu cầu của hệ thống thì nhóm 11 quyết định chọn mô hình kiến trúc layered với các lý do sau:

* Là một kiến trúc phù hợp với các ứng dụng vừa và nhỏ. Ngoài ra layerd là một kiến trúc phù hợp để làm “starting point” cho các dự án mà vẫn chưa xác định được kiến trúc nào sẽ là phù hợp nhất. Đặc biệt Layerd rất phù hợp cho các dự án có nguồn lực về thời gian và tiền bạc hạn chế vì tính đơn giản và dễ hiện thực.
* Simplicity (5 sao): Layered là một kiến trúc đơn giản, dễ học và implement, chính vì thế những ứng dụng được xây dựng dựa trên kiến trúc này thường có thời gian hoàn thành ngắn hơn so với những kiến trúc khác. Dựa trên điều kiện thực tế là nhóm có khoảng 1 tháng để hoàn thiện ứng dụng, nên nhóm nghĩ chọn một kiến trúc như layerd là phù hợp để theo kịp tiến độ về thời gian.
* Overall cost (5 sao): Chi phí tổng quan, cũng như chi phí dùng chuyển đổi sang một kiến trúc khác là khá thấp. Thích hợp để sử dụng trong các trường hợp trường hợp thiếu nguồn lực (Tiền, nhân lực...)
* Reliability (3 sao): Ngoài ra độ tin cậy của kiến trúc Layered là có. Hệ thống sử dụng kiến trúc layered có một độ ổn định và khả năng chịu lỗi nhất định. Kết hợp với 2 yếu tố đầu, nên nhóm quyết định dùng kiến trúc layered cho project lần này.
* Để biện luận thêm về việc chọn kiến trúc layered là khả thi cho đề tài quản lý lớp học tín chỉ, chúng ta có thể xem xét thêm tình huống sau:
  + Yêu cầu: Hệ thống cần đảm bảo tính bảo mật cao, đặc biệt là trong việc xác thực và xác định quyền truy cập của sinh viên và nhân viên trường.
  + Phân tích: Kiến trúc layered có thể được thiết kế với các lớp bảo mật riêng biệt, giúp kiểm soát và quản lý quyền truy cập của người dùng một cách hiệu quả. Ví dụ, lớp xử lý đăng nhập và xác thực có thể được tách biệt và quản lý riêng, đảm bảo tính bảo mật của hệ thống.

## **2.3. Ưu điểm và nhược điểm khi chọn Layered Architecture**

* Ưu điểm:
  + Phân chia rõ ràng các tầng: mô hình layered phân chia rõ ràng các tầng giúp dễ dàng quản lý mã nguồn và hiểu được cấu trúc của hệ thống.
  + Tính tái sử dụng: các thành phần ở mỗi tầng có thể được sử dụng lại trong các phần khác của hệ thống hoặc trong các dự án khác.
  + Dễ bảo trì và mở rộng: mỗi tầng chỉ chịu trách nhiệm cho một loại công việc cụ thể, giúp dễ dàng bảo trì và mở rộng hệ thống.
  + Tính linh hoạt: mô hình layered cho phép thay đổi ở một tầng mà không ảnh hưởng đến các tầng khác, giúp dễ dàng thay đổi và cập nhật.
* Nhược điểm:
  + Khả năng linh hoạt giảm: mô hình layered có thể làm giảm khả năng linh hoạt so với mô hình microservices vì các tầng được tích hợp chặt chẽ với nhau.
  + Hiệu suất: việc chia thành nhiều tầng có thể làm tăng độ trễ trong việc xử lý yêu cầu của người dùng.
  + Phức tạp khi phát triển lớn: khi ứng dụng phát triển lớn, mô hình kiến trúc layered có thể trở nên phức tạp và khó bảo trì.
  + Sự phụ thuộc giữa các tầng cao: các tầng trong mô hình kiến trúc layered thường phụ thuộc chặt chẽ vào nhau làm cho việc thay đổi ở một tầng có thể ảnh hưởng đến các tầng khác.

## **2.4. Sự đánh đổi khi chọn Layered Architecture**

* Overhead của giao tiếp: việc truyền dữ liệu qua lại giữa các lớp có thể tạo ra overhead, đặc biệt là trong các ứng dụng có khối lượng lớn hoặc với số lớp lớn.
* Phức tạp khi số lượng lớp tăng cao: khi số lượng lớp tăng lên, việc quản lý và hiểu cấu trúc của ứng dụng có thể trở nên phức tạp. Điều này đặc biệt đúng khi phải xử lý các sự phụ thuộc giữa các lớp.
* Khó khăn trong việc debug và test: việc phân tách các tính năng và logic vào các lớp riêng biệt có thể làm tăng độ phức tạp của quá trình debug và test. Đặc biệt là khi phải theo dõi dòng dữ liệu qua lại giữa các lớp.

# **CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG**

## **3.1. Usecase tổng quát**

**A screenshot of a computer

Description automatically generated**

Hình 5: Usecase tổng quát của hệ thống

|  |  |
| --- | --- |
| Tác nhân | Mô tả tác nhân |
| Sinh viên | Sinh viên sử dụng hệ thống để đăng ký lớp học phần và xem các thông tin |
| Hệ thống thanh toán | Hệ thống hỗ trợ sinh viên thanh toán học phí |

## **3.2. Sơ đồ ERD**

A screenshot of a computer

Description automatically generated

Hình 6: Sơ đồ ERD của hệ thống

## **3.3. Class diagram**

A screenshot of a computer

Description automatically generated

Hình 7: Sơ đồ Class của hệ thống

## **3.4. Package diagram**

A screenshot of a computer

Description automatically generated

Hình 8: Sơ đồ package diagram của hệ thống

## **3.5. Component diagram**

A screenshot of a computer

Description automatically generated

Hình 9: Sơ đồ component diagram của hệ thống

## **3.6. Deployment diagram**

A container with a box and a box with a box and a box with a box with a box and a box with a box with a box and a box with a box with a box and

Description automatically generated

Hình 10: Sơ đồ deployment diagram của hệ thống

## **3.7. Mô hình Layered Architecture**

A computer screen shot of a application

Description automatically generated

Hình 11: Mô hình kiến trúc layered của hệ thống

# **CHƯƠNG 4: HIỆN THỰC**

## **4.1. Cấu hình phần cứng, phần mềm**

**Phần cứng: Máy tính cá nhân có kết nối mạng LAN.**

|  |  |  |  |
| --- | --- | --- | --- |
| **CPU** | **RAM** | **HDD** | **Architecture** |
| Intel Core i5, 2.3 GHz | 8 GB | 360 GB | 64 bit |

**Phần mềm:**

|  |  |
| --- | --- |
| **Tên phần mềm** | **Phiên bản** |
| Visual Studio Code | 1.88 |
| IntelliJ IDEA | 2023.2.1 |
| MySQL | 8 |
| Postman | V10 |
| Microsoft Windows 10 | 10 |

## **4.2. Giao diện của hệ thống**

### **4.2.1. Màn hình đăng nhập**

A screenshot of a computer

Description automatically generated

### **4.2.2. Màn hình Xem thông tin sinh viên**

A screenshot of a computer

Description automatically generated

### **4.2.3. Màn hình Xem kết quả học tập**

A screenshot of a computer

Description automatically generated

### **4.2.4. Màn hình Xem lịch học**

A screenshot of a computer

Description automatically generated

### **4.2.5. Màn hình Xem chương trình khung**

A screenshot of a computer

Description automatically generated

### **4.2.6. Màn hình Đăng ký học phần**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# **CHƯƠNG 5: KẾT LUẬN**

## **5.1. Kết quả đạt được**

Hoàn thành một số chức năng cơ bản như:

* + Đăng nhập
  + Xem thông tin sinh viên
  + Xem kết quả học tập
  + Xem lịch học
  + Xem chương trình khung
  + Đăng ký học phần

## **5.2. Hạn chế**

- Chưa thực hiện được chức năng thanh toán học phí

- Chưa xử lý các chức năng dành cho giáo vụ khoa, giảng viên

## **5.3. Hướng phát triển**

Phiên bản đầu tiên của dự án quản lý học phần tín chỉ đã hoàn thành với những chức năng cơ bản. Nhóm sẽ tiếp tục phát triển dự án với các ý tưởng như:

* Chức năng Thanh toán học phí
* Các chức năng dành cho giáo vụ khoa và giảng viên.
* Cải thiện giao diện và trải nghiệm người dùng
* Tăng cường bảo mật và hiệu suất