

**Họ tên:** Phạm Huy Hòa – B21DCCN381

**Lớp:** D21CNPM5

## **ASSIGMENT 1 – START**

### **1.1 Why do we need to study Software Architecture and Design? (3 pages)**

- Kiến trúc phần mềm đề cập đến cách thức mà tất cả các thành phần trong ứng dụng được thiết kế, lập kế hoạch và cấu trúc. Thiết kế kiến trúc tốt trong phát triển phần mềm liên quan đến việc đưa ra quyết định đúng đắn về tổ chức, thành phần, giao diện và công nghệ. Theo cách đó, việc liên kết phần mềm với các mục tiêu kinh doanh, tích hợp các công nghệ mới khi chúng xuất hiện, tạo ra một khu đất phần mềm tổng thể có khả năng chống chịu trong tương lai và cho phép hiệu suất tốt hơn của tất cả các ứng dụng trên toàn doanh nghiệp.
- Thiết kế kiến trúc vững chắc cũng giảm thiểu thời gian chết trên tất cả các thành phần hệ thống và đảm bảo tính liên tục của hoạt động kinh doanh trong nhiều tình huống bất trắc (di chuyển, tấn công mạng, thay đổi nhân sự, nâng cấp, tích hợp, v.v.).
- Trong bối cảnh công nghệ ngày càng phát triển nhanh chóng, nhu cầu về các hệ thống phần mềm phức tạp ngày càng gia tăng. Phần mềm không chỉ đơn thuần là công cụ tự động hóa các tác vụ, mà đã trở thành nền tảng của hầu hết các doanh nghiệp, dịch vụ và tiến bộ công nghệ. Khi độ phức tạp của các hệ thống tăng lên, việc hiểu rõ về kiến trúc và thiết kế phần mềm trở nên vô cùng quan trọng. Kiến trúc và thiết kế phần mềm đóng vai trò then chốt trong việc đảm bảo rằng các hệ thống này hoạt động hiệu quả, dễ bảo trì, có thể mở rộng và bảo mật:

- **Đảm bảo chất lượng và hiệu suất hệ thống:**

Một trong những lý do chính để học kiến trúc và thiết kế phần mềm là để đảm bảo rằng hệ thống hoạt động tối ưu. Kiến trúc phần mềm xác định cấu trúc tổng thể của hệ thống, bao gồm cách các thành phần của hệ thống tương tác với nhau. Thiết kế phần mềm đi sâu vào các chi tiết cụ thể của việc triển khai từng thành phần và sự tương tác giữa chúng. Một kiến trúc phần mềm được thiết kế tốt sẽ giúp phát triển các hệ thống không chỉ hoạt động mà còn đạt hiệu suất cao trong thực tế.

Lấy ví dụ một ứng dụng quy mô lớn như nền tảng thương mại điện tử. Kiến trúc cần phải tính toán để xử lý lưu lượng truy cập cao, lưu trữ dữ liệu, xử lý giao dịch và sự tương tác giữa các dịch vụ khác nhau (chẳng hạn như thanh toán, quản lý kho, giao diện người dùng). Nếu không có sự thiết kế hợp lý, hệ thống có thể gặp phải các vấn đề về hiệu suất, như thời gian phản hồi chậm, lỗi khi tải lớn, hoặc tiêu tốn tài nguyên quá mức.

Một kiến trúc phần mềm hiệu quả sẽ giảm thiểu các rủi ro này, đảm bảo phần mềm đáp ứng được các mục tiêu về hiệu suất và mang lại trải nghiệm người dùng mượt mà.

- **Dễ bảo trì và linh hoạt:**

Trong phát triển phần mềm, sự thay đổi là điều không thể tránh khỏi. Theo thời gian, yêu cầu của người dùng thay đổi, công nghệ mới xuất hiện, và phần mềm cần được cải tiến. Chính vì thế, thiết kế và kiến trúc phần mềm đóng vai trò quan trọng trong việc giúp hệ thống có thể bảo trì và thay đổi một cách dễ dàng. Một thiết kế phần mềm tốt sẽ giúp phần mềm linh hoạt, cho phép các tính năng mới được tích hợp mà không cần phải viết lại toàn bộ hệ thống.

Ví dụ: kiến trúc microservices. Kiến trúc này phân tách một hệ thống lớn thành các dịch vụ nhỏ độc lập có thể triển khai riêng biệt. Mỗi microservice xử lý một chức năng kinh doanh cụ thể và có thể được phát triển, cập nhật hoặc thay thế mà không làm ảnh hưởng đến phần còn lại của hệ thống. Phương pháp này không chỉ hỗ trợ bảo trì dễ dàng hơn mà còn giúp tích hợp các tính năng và công nghệ mới mà không làm gián đoạn hệ thống tổng thể.

- **Khả năng mở rộng:**

Khả năng mở rộng là khả năng của một hệ thống phần mềm để xử lý được tải hoặc nhu cầu tăng lên khi hệ thống phát triển. Đây là yếu tố quan trọng đối với phần mềm hiện đại, khi các doanh nghiệp phải đối mặt với sự thay đổi về nhu cầu của người dùng, sự kiện hoặc sự phát triển toàn cầu. Các hệ thống phần mềm cần được thiết kế với khả năng mở rộng để đảm bảo có thể đáp ứng một số lượng người dùng, giao dịch hoặc dữ liệu tăng lên.

Việc học kiến trúc phần mềm giúp các lập trình viên thiết kế hệ thống có khả năng mở rộng theo chiều ngang (bằng cách thêm nhiều máy chủ hoặc phiên bản) hoặc theo chiều dọc (tăng tài nguyên của một máy chủ duy nhất). Các kỹ thuật như cân bằng tải, bộ nhớ đệm, và phân mảnh cơ sở dữ liệu thường được áp dụng trong các hệ thống mở rộng để phân phối tải đều và đảm bảo hệ thống có thể xử lý lưu lượng cao mà không làm giảm hiệu suất. Một kiến trúc phần mềm có khả năng mở rộng là cần thiết để xây dựng các hệ thống có thể phát triển cùng với doanh nghiệp, tránh sự suy giảm hiệu suất và quản lý tốt tải cao.

- **Quản lý rủi ro và bảo mật:**

Bảo mật là một yếu tố tối quan trọng đối với bất kỳ hệ thống phần mềm nào. Với những lo ngại ngày càng tăng về các mối đe dọa mạng, rủi ro lộ thông tin và tấn công từ hacker, việc đảm bảo phần mềm bảo mật ngay từ ban đầu là cực kỳ quan trọng. Học

về kiến trúc phần mềm giúp các lập trình viên đưa ra các quyết định đúng đắn về các rủi ro bảo mật và cách giảm thiểu chúng.

Bảo mật cần phải được xem xét ở mọi tầng của hệ thống. Các quyết định kiến trúc như mã hóa dữ liệu, phương thức xác thực an toàn, kiểm soát truy cập, và phân tách giữa các thành phần là rất quan trọng để bảo vệ phần mềm khỏi các mối đe dọa tiềm ẩn. Thêm vào đó, thiết kế hệ thống cần phải có tính chịu lỗi, xây dựng cơ chế phục hồi và dự phòng để đảm bảo rằng hệ thống có thể khôi phục nhanh chóng sau sự cố hoặc tấn công, giảm thiểu thời gian ngừng hoạt động.

Ngoài ra, hiểu rõ các lỗ hổng và các con đường tấn công trong phần mềm giúp kiến trúc sư phần mềm có thể thực hiện các biện pháp bảo mật chủ động. Ví dụ, nếu kiến trúc sư biết rằng một số thành phần sẽ xử lý dữ liệu nhạy cảm của người dùng, họ có thể chọn cách phân tách các thành phần này thành các dịch vụ độc lập để tránh truy cập trái phép.

- **Hiệu quả về chi phí và tối ưu hóa tài nguyên:**

Kiến trúc phần mềm hiệu quả giúp quản lý chi phí bằng cách tối ưu hóa việc sử dụng tài nguyên, bao gồm cả phần cứng và công sức con người. Những quyết định kiến trúc sai lầm thường dẫn đến các giải pháp quá phức tạp, yêu cầu hạ tầng đắt đỏ, chi phí bảo trì cao và tốn nhiều thời gian phát triển. Một kiến trúc phần mềm được thiết kế tốt sẽ tập trung vào việc tạo ra các giải pháp hiệu quả về chi phí bằng cách chọn công nghệ phù hợp, giảm thiểu việc sử dụng tài nguyên và giảm bớt sự phức tạp không cần thiết.

Ví dụ, các giải pháp dựa trên đám mây đang ngày càng trở nên phổ biến vì khả năng mở rộng tài nguyên khi cần thiết. Bằng cách học kiến trúc phần mềm, các lập trình viên có thể thiết kế các hệ thống tận dụng được sự linh hoạt của đám mây, đảm bảo tài nguyên chỉ được sử dụng khi cần và tránh phải đầu tư vào hạ tầng đắt đỏ tại chỗ. Tương tự, thiết kế phần mềm tốt có thể giảm thiểu việc phát triển thừa và tối ưu hóa thời gian sửa lỗi, từ đó giảm chi phí phát triển tổng thể.

- **Giao tiếp hiệu quả trong đội ngũ phát triển:**

Kiến trúc phần mềm đóng vai trò như bản thiết kế cho toàn bộ hệ thống. Nó cung cấp một ngôn ngữ chung và hiểu biết chung cho tất cả các thành viên trong nhóm phát triển, bao gồm lập trình viên, quản lý dự án, kỹ sư kiểm thử chất lượng và các bên liên quan. Bằng cách học về kiến trúc phần mềm, các thành viên trong nhóm có thể giao tiếp hiệu quả hơn và đảm bảo rằng tất cả các phần của hệ thống hoạt động hòa hợp với nhau.

Trong một dự án phần mềm phức tạp, các thành viên trong nhóm thường làm việc trên các thành phần khác nhau mà đôi khi không hiểu rõ cách mà công việc của họ phù hợp với bức tranh tổng thể. Một thiết kế kiến trúc rõ ràng giúp giảm thiểu sự hiểu lầm và không đồng bộ bằng cách cung cấp một tầm nhìn chung về cấu trúc hệ thống. Nó giúp các lập trình viên hiểu được sự phụ thuộc giữa các thành phần, luồng dữ liệu và hành vi dự kiến của hệ thống, từ đó thúc đẩy phát triển đồng bộ và hiệu quả hơn.

- **Phù hợp với mục tiêu kinh doanh:**

Cuối cùng, một kiến trúc phần mềm được thiết kế hợp lý có thể giúp các tổ chức đạt được các mục tiêu chiến lược của họ. Trong bất kỳ doanh nghiệp nào, phần mềm cần phải phục vụ các nhu cầu kinh doanh một cách hiệu quả và đúng đắn. Học về kiến trúc phần mềm giúp các lập trình viên tạo ra các giải pháp đáp ứng các yêu cầu kinh doanh, đồng thời có thể thay đổi linh hoạt để phù hợp với các mục tiêu hay điều kiện thị trường thay đổi.

Ví dụ, nếu một doanh nghiệp muốn mở rộng vào các khu vực hay thị trường mới, phần mềm cần được thiết kế để hỗ trợ điều này. Điều này có thể bao gồm thiết kế hỗ trợ đa ngôn ngữ, lưu trữ dữ liệu khu vực, hoặc tích hợp với các hệ thống bên thứ ba hoạt động tại các khu vực đó. Một kiến trúc phần mềm vững chắc cung cấp nền tảng cho những thay đổi này, giúp doanh nghiệp phát triển mà không cần phải thay đổi toàn bộ hệ thống.

- **Tầm quan trọng của mẫu thiết kế:**

Trong việc học kiến trúc và thiết kế phần mềm, lập trình viên cũng học về các mẫu thiết kế—những giải pháp đã được chứng minh cho các vấn đề thường gặp trong phát triển phần mềm. Các mẫu thiết kế giúp tiêu chuẩn hóa cách giải quyết các vấn đề lặp đi lặp lại, tạo ra một ngôn ngữ chung cho các lập trình viên và giúp dễ dàng giao tiếp về các giải pháp.

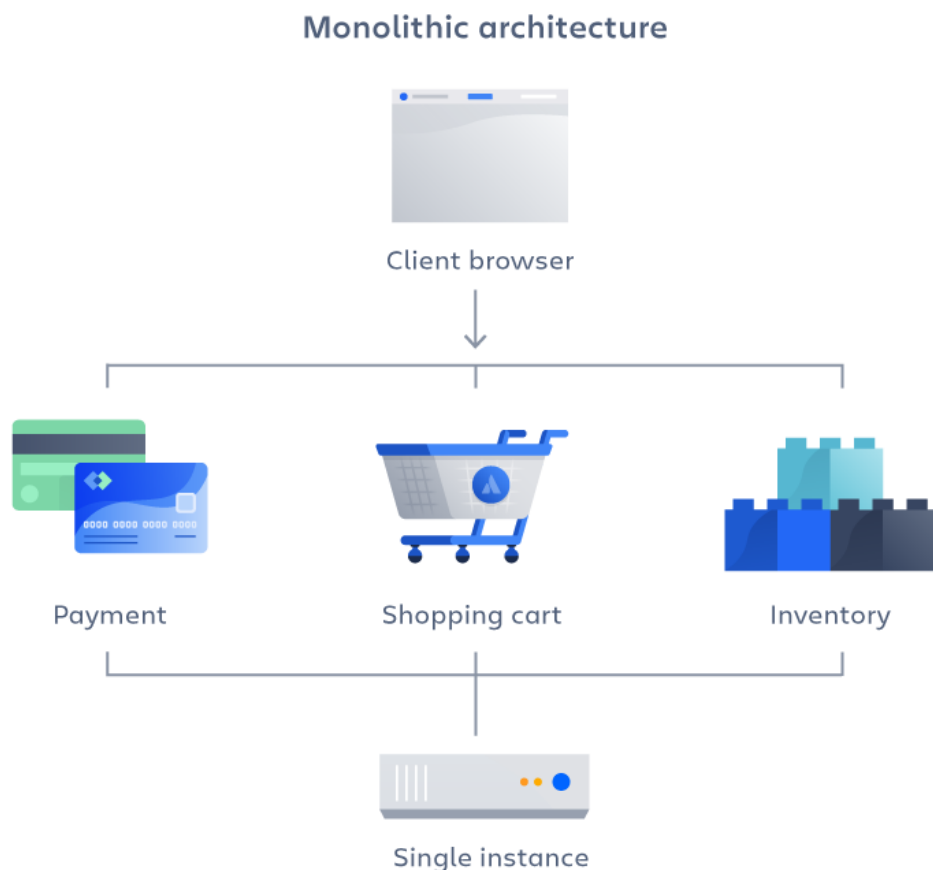
Chẳng hạn, mẫu thiết kế Singleton đảm bảo rằng một lớp chỉ có một thể hiện duy nhất, trong khi mẫu Observer cho phép giao tiếp giữa các thành phần mà không cần kết nối chặt chẽ. Bằng cách sử dụng các mẫu thiết kế này, lập trình viên có thể tránh việc làm lại từ đầu và tạo ra phần mềm dễ bảo trì, mở rộng và dễ hiểu hơn.

## 1.2 Monolithic and Microservice. Microservice and AGILE ([1] [2] write 7 pages)

### 1. Monolithic and Microservice:

#### a. Monolithic architecture

- Kiến trúc Monolithic (kiến trúc đơn khối) là một mô hình truyền thống của một chương trình phần mềm, được xây dựng như một đơn vị thống nhất, tự chứa và độc lập với các ứng dụng khác. Kiến trúc này là một mạng máy tính đơn nhất, lớn với một code base (mã nguồn) duy nhất kết nối tất cả các mối quan tâm kinh doanh lại với nhau. Để thay đổi ứng dụng kiểu này, cần phải cập nhật toàn bộ stack (ngăn xếp) bằng cách truy cập vào code base và xây dựng, triển khai một phiên bản mới của service-side interface (giao diện phía dịch vụ). Điều này khiến việc cập nhật trở nên hạn chế và tốn thời gian.
- Kiến trúc monolithic có thể thuận tiện vào đầu vòng đời của một dự án, giúp dễ dàng quản lý mã nguồn, giảm thiểu khối lượng công việc trí óc và triển khai. Điều này cho phép mọi thứ trong monolith có thể được phát hành đồng thời.

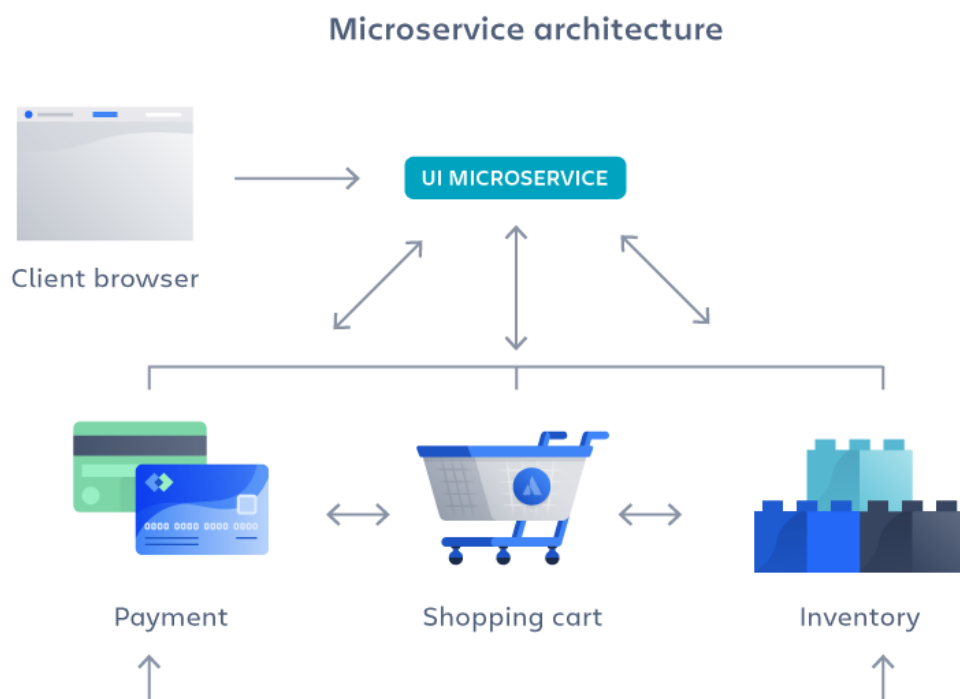


- Ưu điểm:
  - Dễ phát triển vì các stack công nghệ thống nhất ở tất cả các layer.
  - Dễ test do toàn bộ project được đóng gói trong một package nên dễ dàng chạy test integration và test end-to-end.
  - Deploy đơn giản và nhanh chóng nếu bạn chỉ có một package để bận tâm.
  - Dễ scale vì chúng ta có thể có nhiều instance cho load balancer.
  - Yêu cầu team size nhỏ cho việc maintain app.
  - Team member có thể chia sẻ ít nhiều về skill.
  - Tech stack đơn giản và đa số là dễ học.
  - Phát triển ban đầu nhanh hơn do đó có thể đem sale hoặc marketing nhanh hơn.
  - Yêu cầu cơ sở hạ tầng đơn giản, thậm chí một container đơn giản cũng đủ để chạy ứng dụng.
- Nhược điểm:
  - Các component được liên kết chặt chẽ với nhau dẫn đến side effect không mong muốn như khi thay đổi một component ảnh hưởng đến một component khác.
  - Theo thời gian thì project trở nên phức tạp và lớn dần. Các tính năng mới sẽ mất nhiều thời gian hơn để phát triển và tái cấu trúc các tính năng hiện có sẽ nhiều khó khăn hơn.
  - Toàn bộ ứng dụng cần được triển khai lại cho bất kỳ thay đổi nào.
  - Không hề dễ để hiểu project do các module liên quan chặt chẽ lẫn nhau. Một issue nhỏ cũng có thể làm chết toàn bộ ứng dụng.
  - Áp dụng công nghệ mới khó khăn vì toàn bộ ứng dụng phải thay đổi. Do đó nhiều ứng dụng một khối thường phụ thuộc một công nghệ cũ và lỗi thời.
  - Các service quan trọng không thể scale riêng dẫn đến lãng phí tài nguyên vì toàn bộ ứng dụng phải scale theo.
  - Các ứng dụng một khối lớn sẽ có thời gian khởi động lâu và tốn tài nguyên CPU cũng như bộ nhớ.
  - Các team tham gia vào dự án phải phụ thuộc lẫn nhau và rất khó để mở rộng quy mô team.
- Chúng ta thường sử dụng kiến trúc đơn khối khi:
  - Phạm vi ứng dụng là nhỏ và được xác định rõ. Ta chắc chắn ứng dụng sẽ không phát triển mạnh về các tính năng. Ví dụ: blog, web mua sắm trực tuyến đơn giản, ứng dụng CRUD đơn giản...
  - Team-size nhỏ, thường ít hơn 8 người.
  - Mặt bằng kỹ năng của các thành viên trong team thường không cao.
  - Thời gian để có thể marketing là quan trọng.

- Ta không muốn mất thời gian cho cơ sở hạ tầng, monitoring,...
- Khi người dùng thường nhỏ và ít nên ta không mong đợi họ sẽ mở rộng. Ví dụ các ứng dụng doanh nghiệp nhắm đến mục tiêu là một nhóm người cụ thể...

## b. Microservice

- Kiến trúc microservices (dịch vụ vi mô), còn được gọi đơn giản là microservices, là một phương pháp kiến trúc dựa trên một loạt các dịch vụ có thể triển khai độc lập. Các dịch vụ này có logic nghiệp vụ và cơ sở dữ liệu riêng, với mục tiêu cụ thể. Việc cập nhật, kiểm thử, triển khai và mở rộng diễn ra trong mỗi dịch vụ. Microservices phân tách các mối quan tâm lớn, chuyên biệt theo lĩnh vực kinh doanh thành các code bases (mã nguồn) độc lập. Mặc dù microservices không giảm bớt độ phức tạp, nhưng chúng làm cho mọi phức tạp trở nên rõ ràng và dễ quản lý hơn bằng cách phân chia các tác vụ thành các quy trình nhỏ hơn, hoạt động độc lập với nhau và đóng góp vào tổng thể hệ thống.
- Việc áp dụng microservices thường đi đôi với DevOps (phát triển và vận hành), vì chúng là nền tảng cho các phương pháp continuous delivery cho phép các nhóm phát triển thích ứng nhanh chóng với yêu cầu của người dùng.



- Ưu điểm:
  - Các component có kết nối lỏng lẻo dẫn đến dễ cách ly, dễ test và khởi động nhanh.
  - Vòng đời phát triển nhanh hơn. Tính năng mới được phát triển nhanh hơn và tính năng cũ được cấu trúc lại dễ hơn.
  - Các service có thể deploy độc lập nên ứng dụng dễ đọc, dễ tạo các bản vá hơn.
  - Những issue, ví dụ liên quan đến memory leak một trong các service, bị cô lập và có thể không làm sập ứng dụng.
  - Việc áp dụng các công nghệ mới dễ hơn. Các component có thể được nâng cấp độc lập với nhau.
  - Các mô hình scale phức tạp và hiệu quả hơn có thể được thiết lập. Các service quan trọng có thể scale hiệu quả hơn. Các component riêng sẽ khởi động nhanh hơn và cải thiện thời gian khởi động của cả hệ thống.
  - Các team tham gia sẽ ít phụ thuộc lẫn nhau. Kiến trúc này rất thích hợp cho các đội Agile.
- Nhược điểm:
  - Phức tạp hơn về mặt tổng thể vì các component khác nhau có các stack công nghệ khác nhau nên buộc team phải tập trung đầu tư thời gian để theo kịp công nghệ.
  - Khó thực hiện test end-to-end và integration test vì có nhiều stack công nghệ khác nhau.
  - Deploy toàn bộ ứng dụng phức tạp hơn vì có nhiều container và nền tảng ảo hóa liên quan.
  - Ứng dụng được scale hiệu quả hơn nhưng thiết lập nâng cấp sẽ phức tạp hơn vì nó sẽ yêu cầu nâng cao nhiều tính năng như truy tìm dịch vụ (service discovery), định tuyến DNS,...
  - Yêu cầu một team-size lớn để maintain ứng dụng vì có nhiều component và công nghệ khác nhau.
  - Các thành viên trong team chia sẻ các skill khác nhau dựa trên component họ làm nên sẽ tạo ra sự khó khăn khi thay thế và chia sẻ kiến thức.
  - Stack công nghệ phức tạp và khó để học hơn.
  - Thời gian phát triển ban đầu là chậm nên thời gian để có thể làm marketing lâu hơn.
  - Yêu cầu cơ sở hạ tầng phức tạp. Thông thường sẽ yêu cầu nhiều container (Docker) và nhiều máy JVM để chạy.

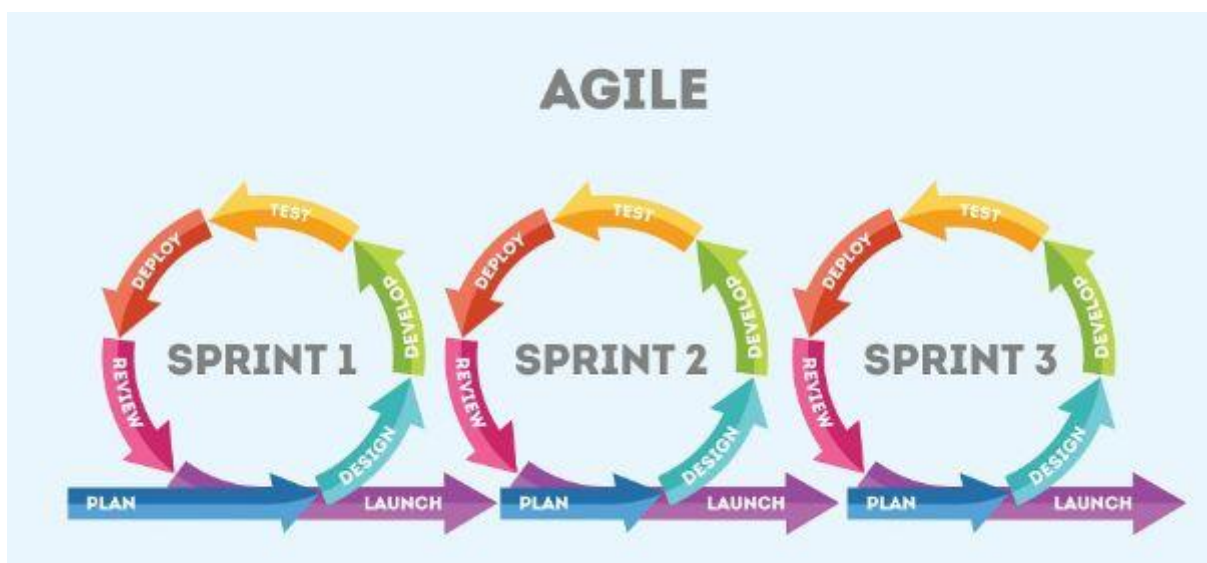


- Chúng ta thường sử dụng kiến trúc microservice khi:
  - Ứng dụng có phạm vi lớn và ta xác định các tính năng sẽ được phát triển rất mạnh theo thời gian. Ví dụ: cửa hàng thương mại điện tử trực tuyến, dịch vụ truyền thông xã hội, dịch vụ truyền phát video với số lượng người dùng lớn, dịch vụ cung cấp API,...
  - Team-size lớn, có đủ thành viên để phát triển các component riêng lẻ một cách hiệu quả.
  - Mặt bằng kỹ năng của team tốt và các thành viên tự tin về các mẫu thiết kế microservice nâng cao.
  - Thời gian để đem đi marketing không quan trọng. Kiến trúc microservice sẽ mất nhiều thời gian hơn để hoạt động được.
  - Ta sẵn sàng chi nhiều hơn cho cơ sở hạ tầng, giám sát, ... để nâng cao chất lượng sản phẩm.
  - Tiềm năng về người dùng lớn và ta kỳ vọng số lượng người dùng sẽ phát triển. Ví dụ một phương tiện truyền thông xã hội nhắm mục tiêu là người dùng trên toàn thế giới.

## 2. Microservice and AGILE

### a. Agile

- Agile (viết tắt của Agile Software Development) là một phương thức phát triển phần mềm linh hoạt, được thực hiện bằng cách sử dụng các bước lặp ngắn từ 1 đến 4 tuần. Mục tiêu của Agile là giúp rút ngắn thời gian phát triển sản phẩm, đưa sản phẩm đến với tay khách hàng càng sớm càng tốt.



- Về bản chất, Agile giống như một phương pháp luận, một triết lý dựa trên nguyên tắc phân đoạn vòng lặp (iterative) và tăng trưởng (incremental) nên sở hữu tính linh hoạt cao. Tính chất này đi ngược lại với các phương pháp quản lý dự án truyền thống








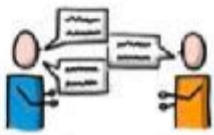




– vốn dĩ triển khai các giai đoạn một cách tuyến tính và vô cùng bị động trước các thay đổi bất ngờ.

- Ngày nay, triết lý Agile đã vượt xa khỏi khu vực truyền thống của mình là phát triển phần mềm để đóng góp sự thay đổi trong cách thức làm việc, quản lý, sản xuất ở các ngành khác như sản xuất, dịch vụ, sales, marketing, giáo dục, ... và trở thành một phương pháp quản lý dự án phổ biến nhất hiện nay với nhiều đại diện được gọi là các phương pháp “họ Agile”.
- Triết lý Agile xuất phát từ ngành công nghệ và được mô tả bằng 4 giá trị và 12 nguyên lý cốt lõi trong Tuyên ngôn phát triển phần mềm linh hoạt hay Tuyên ngôn Agile (The Manifesto for Agile Software Development).
- Đặc trưng của Agile:
  - Tính lặp (Iterative): Dự án sẽ được thực hiện trong các phân đoạn lặp đi lặp lại, thường có khung thời gian ngắn (từ 1-4 tuần). Trong mỗi phân đoạn đó, nhóm phát triển dự án sẽ thực hiện đầy đủ các công việc cần thiết như lập kế hoạch, phân tích yêu cầu, thiết kế, triển khai, kiểm thử để cho ra các phần nhỏ của sản phẩm.
  - Tính tăng trưởng và tiến hóa (Incremental & Evolutionary): Cuối các phân đoạn, nhóm cho ra các phần nhỏ của sản phẩm cuối cùng, thường là đầy đủ, có khả năng chạy tốt, được kiểm thử cẩn thận và có thể sử dụng. Theo thời gian, phân đoạn này tiếp nối phân đoạn kia, các phần chạy được này sẽ được tích lũy, lớn dần lên cho tới khi toàn bộ yêu cầu của khách hàng được thỏa mãn.
  - Tính thích nghi (adaptive): Do các phân đoạn chỉ kéo dài trong một khoảng thời gian ngắn và việc lập kế hoạch cũng được điều chỉnh liên tục, nên các thay đổi trong quá trình phát triển (yêu cầu thay đổi, thay đổi công nghệ, thay đổi định hướng về mục tiêu, ...) đều có thể được đáp ứng theo cách thích hợp.
  - Nhóm tự tổ chức và liên chức năng: Các cấu trúc nhóm này tự phân công công việc mà không dựa trên các mô tả cứng nhắc về chức danh hay một sự phân cấp rõ ràng. Nhóm tự tổ chức đã đủ các kỹ năng cần thiết để có thể được trao quyền tự ra quyết định, tự quản lý và tổ chức lấy công việc của chính mình để đạt được hiệu quả cao nhất.
  - Quản lý tiến trình thực nghiệm (Empirical Process Control): Các nhóm Agile ra các quyết định dựa trên các dữ liệu thực tiễn (data-driven) thay vì tính toán lý thuyết hay các tiên giả định. Agile rút ngắn vòng đời phản hồi để dễ dàng thích nghi và gia tăng tính linh hoạt nhờ đó có thể kiểm soát được tiến trình, và nâng cao năng suất lao động.
  - Giao tiếp trực diện (face-to-face communication): Agile không phản đối việc tài liệu hóa, nhưng đánh giá cao hơn việc giao tiếp trực diện thay vì thông qua giấy

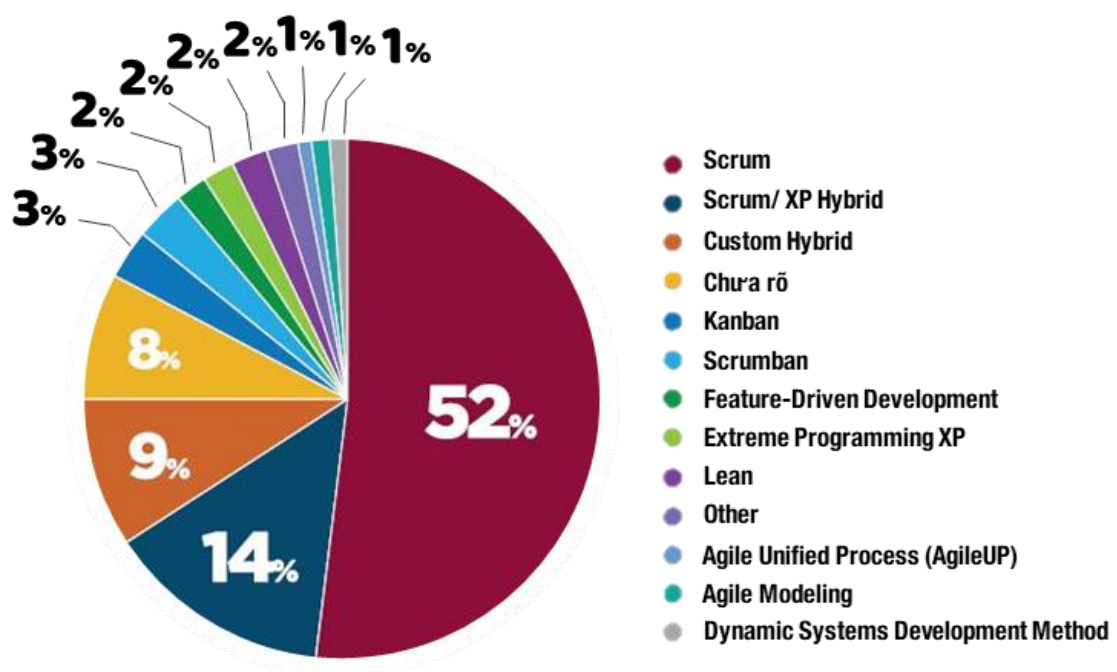
từ. Agile khuyến khích nhóm phát triển trực tiếp nói chuyện để hiểu rõ hơn về những gì khách hàng thực sự cần. Trong giao tiếp giữa nội bộ nhóm, Agile khuyến khích trực tiếp trao đổi và thống nhất với nhau về thiết kế của hệ thống và cùng nhau triển khai thành các chức năng theo yêu cầu.

- Phát triển dựa trên giá trị (value-based development): Một trong các nguyên tắc cơ bản của Agile là “sản phẩm chạy tốt chính là thước đo của tiến độ”. Nhóm Agile thường cộng tác trực tiếp và thường xuyên với khách hàng để biết yêu cầu nào có độ ưu tiên cao hơn, mang lại giá trị hơn sớm nhất có thể cho dự án.
- 4 tôn chỉ của Agile:
  - Individuals and interactions over processes and tools: Cá nhân và sự tương tác hơn là quy trình và công cụ.
  - Working software over comprehensive documentation: Phần mềm chạy tốt hơn là tài liệu đầy đủ.
  - Customer collaboration over contract negotiation: Cộng tác với khách hàng hơn là đàm phán hợp đồng.
  - Responding to change over following a plan: Phản hồi với sự thay đổi hơn là bám theo kế hoạch.
- 12 nguyên tắc trong Agile:
  - 1 – Ưu tiên cao nhất của Agile là làm hài lòng khách hàng thông qua việc cung cấp sớm và liên tục các sản phẩm có giá trị.
  - 2 – Agile luôn chào đón các yêu cầu thay đổi, kể cả khi nó xuất hiện ở giai đoạn cuối của quá trình triển khai, khi mọi thứ đã gần như hoàn thiện. Quy trình Agile luôn khai thác sự thay đổi để tạo nên lợi thế cạnh tranh cho khách hàng.
  - 3 – Giao phần mềm chạy được cho khách hàng một cách thường xuyên, từ vài tuần đến vài tháng.
  - 4 – Các nhà kinh doanh và nhà phát triển phần mềm phải làm việc cùng nhau hàng ngày trong suốt dự án.
  - 5 – Tập trung xây dựng dự án xung quanh những cá nhân có động lực, cung cấp cho họ môi trường và sự hỗ trợ cần thiết, cùng với niềm tin vào việc hoàn thành tốt công việc.
  - 6 – Phương pháp hiệu quả nhất để truyền đạt thông tin là trao đổi trực tiếp.
  - 7 – Phần mềm hoạt động tốt chính là thước đo của tiến độ triển khai Agile.
  - 8 – Quy trình Agile thúc đẩy sự phát triển bền vững.
  - 9 – Sự chú ý liên tục đến yếu tố kỹ thuật và thiết kế sẽ cải tiến sự linh hoạt.
  - 10 – Nghệ thuật tối đa hóa lượng công việc chưa xong: Sự đơn giản là cần thiết.

- 11 – Những thiết kế và yêu cầu tốt nhất chỉ xuất hiện tại những nhóm làm việc có khả năng tự tổ chức tốt.
- 12 – Nhóm thực thi cần suy nghĩ về việc làm việc hiệu quả một cách thường xuyên để có sự điều chỉnh hành vi phù hợp.

<p>Thỏa mãn khách hàng thông qua việc chuyển giao sớm và liên tục các phần mềm có giá trị.</p> 	<h2>12 NGUYÊN TẮC AGILE</h2>		<p>Nhà kinh doanh và nhà phát triển phải làm việc cùng nhau.</p> 
<p>Xây dựng các dự án xung quanh những cá nhân có động lực. Cung cấp cho họ sự hỗ trợ cần thiết. Tin tưởng họ</p> 	<p>Chào đón việc thay đổi yêu cầu, thậm chí rất muộn trong quá trình phát triển.</p> 	<p>Thường xuyên chuyển giao phần mềm chạy tốt tới khách hàng.</p> 	<p>Phần mềm chạy tốt là thước đo chính của tiến độ.</p> 
<p>Liên tục quan tâm đến các kỹ thuật và thiết kế tốt.</p> 	<p>Phương pháp hiệu quả nhất để truyền đạt thông tin là hội thoại trực tiếp.</p> 	<p>Các kiến trúc, yêu cầu và thiết kế tốt nhất đến từ các nhóm tự tổ chức.</p> 	<p>Các nhà tài trợ, nhà phát triển và người dùng nên duy trì một nhịp độ liên tục không giới hạn.</p> 
<p>Sự đơn giản - nghệ thuật tối đa hóa lượng công việc chưa xong - là căn bản.</p> 	<p>Nhóm phát triển phân tư về cách để trở nên hiệu quả hơn và điều chỉnh hành vi của mình sao cho phù hợp.</p> 		

- Các phương pháp Agile phổ biến:



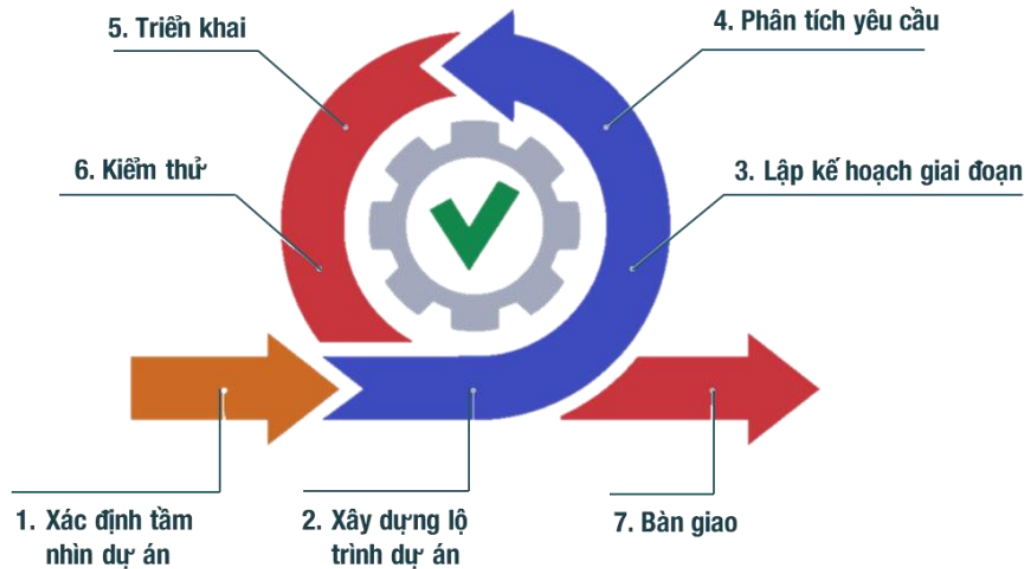
Bảng thống kê liệt kê 13 phương pháp họ Agile, cho thấy phần lớn các công ty hiện nay đã sử dụng Scrum như một cách tiếp cận cơ bản. Bên cạnh đó, nhiều công ty đã kết hợp các phương pháp lại với nhau. Ví dụ 44.4% các công ty có sử dụng Waterfall, có nghĩa là một tỉ lệ nhất định nào đó vừa dùng Waterfall, vừa sử dụng Scrum trong hoạt động của mình. Chi tiết một số phương pháp Agile nổi tiếng:

- Scrum: theo Tài liệu Hướng dẫn Scrum (The Scrum Guide) được 2 nhà đồng sáng lập Ken Schwaber and Jeff Sutherland định nghĩa, là một khung làm việc (framework) để phát triển bền vững các sản phẩm phức tạp. Có thể nói Scrum là một trong những phương pháp Agile quan trọng nhất sử dụng cơ chế lặp (iterative) và tăng trưởng (Incremental) để tối ưu hóa hiệu quả cũng như kiểm soát rủi ro.
- Kanban: là một phương pháp Agile dựa trên Phương thức Sản xuất Toyota với bốn nguyên lý: Trực quan hóa công việc, giới hạn công việc đang làm, tập trung vào luồng làm việc, cải tiến liên tục. Mô hình Kanban phù hợp cho việc hỗ trợ sản xuất trong quá trình làm việc.
- Scrumban: là một phương pháp được Corey Ladas giới thiệu vào năm 2009 trong cuốn sách với tựa đề “Scrumban – Essays on Kanban Systems for Lean Software Development”. Scrumban kết hợp được những ưu điểm của Scrum và Kanban để cho phép nhóm liên tục cải tiến quy trình và khả năng xử lý công việc.
- Lean Software Development (LSD): hay Phát triển phần mềm tinh gọn là hình thức áp dụng Tư duy tinh gọn (Lean Thinking) và các nguyên lý đặc trưng của Tinh gọn (xuất phát từ ngành sản xuất ô tô – Lean Manufacturing) cho lĩnh vực phát triển phần mềm. Thuật ngữ Lean Software Development có nguồn gốc từ một cuốn sách cùng tên của Mary Poppendieck và Tom Poppendieck. Trong đó, bảy nguyên lý diễn giải tư duy Tinh gọn bao gồm: Loại bỏ lãng phí, Khuếch trương việc học, Quyết định càng muộn càng tốt, Chuyển giao càng nhanh càng tốt, Trao quyền cho nhóm, Tạo ra tính toàn vẹn tự thân, Thấy toàn cảnh là linh hồn cho quá trình phát triển phần mềm tinh gọn.
- XP (Extreme Programming) – Hay lập trình cực hạn là một phương pháp phát triển phần mềm thuộc họ Agile được phát minh bởi Ken Beck – một kỹ sư phần mềm người Mỹ. XP hướng đến việc nâng cao chất lượng phần mềm và khả năng đáp ứng với thay đổi yêu cầu người dùng. XP chủ trương đưa ra các bản phát hành thường xuyên thông qua các chu trình phát triển ngắn. Một số các thực hành của XP như: Lập trình cặp (Pair programming), Tái cấu trúc mã nguồn (Refactoring), Kiểm thử đơn vị (Unit Testing), Tích hợp liên tục (Continuous Integration), Các bản phát hành nhỏ (Small Release)...

- Quy trình triển khai Agile:

## Quy trình Agile

---



- Giai đoạn 1: Lên ý tưởng và xây dựng lộ trình dự án
- Giai đoạn 2: Lập kế hoạch giai đoạn (Phát triển dự án)
- Giai đoạn 3: Triển khai dự án
- Giai đoạn 4: Kiểm thử
- Giai đoạn 5: Bàn giao và bảo trì sản phẩm
- Ưu điểm của Agile:
  - Dễ dàng thực hiện thay đổi ở bất kỳ giai đoạn nào của dự án, thích nghi nhanh và hiệu quả với sự thay đổi, bao gồm cả yêu cầu sửa lại sản phẩm, sự biến động từ thị trường,...
  - Phát triển và bàn giao sản phẩm nhanh hơn, bởi việc chia nhỏ dự án cho phép đội ngũ có thể tiến hành kiểm tra theo từng phần, phát hiện sự cố và sửa chữa vấn đề nhanh hơn.
  - Sản phẩm đạt chất lượng tốt hơn, do luôn nhận được phản hồi ngay lập tức từ phía khách hàng và được tối ưu lại ngay sau đó.
  - Lãng phí ít tài nguyên hơn vì nhóm luôn thực hiện các công việc đã được cập nhật, giảm tải thời gian chờ, giảm tải thời gian sửa lỗi sản phẩm, giảm tải số lượng lớn giấy tờ tài liệu,...
  - Người tham gia dự án không cần phải nắm mọi thông tin ngay từ đầu, phù hợp với những dự án chưa xác định rõ ràng được mục tiêu cuối cùng.

- Nhược điểm của Alige:
  - Khó lên kế hoạch dự án, đặc biệt là khó xác định rõ ràng thời gian bàn giao sản phẩm cuối cùng, không nắm rõ được chi phí thực sự của dự án là bao nhiêu,... vì dự án được chia nhỏ thành các vòng lặp khác nhau.
  - Bắt buộc phải hướng dẫn và đào tạo chi tiết, thì thành viên dự án mới có thể hiểu rõ được mô hình Agile và thực hiện theo nó một cách rõ ràng, đặc biệt là trong thời gian đầu.
  - Ít tài liệu hướng dẫn về dự án và không xác định rõ được kỳ vọng và thành phẩm ngay từ đầu, do người thực thi mô hình Agile thường tin rằng mọi thứ sẽ thay đổi rất nhiều, không quá cần thiết phải ghi chép và lưu trữ tài liệu.
  - Đòi hỏi sự cam kết về thời gian và công sức từ các bên hơn, vì tất cả mọi người cần phải liên tục tương tác với nhau trong suốt quá trình thực thi dự án.
  - Chi phí thực hiện dự án theo mô hình Agile thường cao hơn so với các phương pháp phát triển khác.

#### **b. Mối quan hệ giữa Microservice và AGILE**

Mặc dù Microservice và AGILE là hai phương pháp khác nhau, nhưng chúng có thể kết hợp rất tốt với nhau trong việc phát triển phần mềm. Các đặc điểm của Microservice hỗ trợ các nguyên tắc và mục tiêu của AGILE và ngược lại.

- Tính linh hoạt và khả năng thay đổi:
  - Microservice cho phép các nhóm phát triển làm việc trên các dịch vụ nhỏ và độc lập. Mỗi dịch vụ có thể được thay đổi mà không ảnh hưởng đến toàn bộ hệ thống, điều này hoàn toàn phù hợp với phương pháp AGILE, nơi yêu cầu có thể thay đổi liên tục và phần mềm được phát triển và cải thiện qua từng sprint.
  - AGILE khuyến khích thay đổi yêu cầu nhanh chóng và dễ dàng, điều này rất phù hợp với việc thay đổi hoặc cập nhật từng dịch vụ trong Microservice mà không làm gián đoạn toàn bộ hệ thống.
- Phát triển nhanh và phản hồi kịp thời:
  - Microservice cho phép triển khai nhanh chóng từng dịch vụ nhỏ, giúp phản hồi nhanh chóng với yêu cầu thay đổi và giảm thiểu thời gian phát triển cho từng tính năng.
  - AGILE thúc đẩy việc phát triển phần mềm qua các sprint ngắn, mỗi sprint kết thúc với một phần mềm có thể sử dụng được. Cả hai phương pháp đều hướng đến việc phát triển nhanh và linh hoạt, giúp các nhóm phản hồi kịp thời với thay đổi và yêu cầu từ khách hàng.

- Đảm bảo chất lượng và quản lý rủi ro:
  - Microservice giúp giảm thiểu rủi ro trong phát triển phần mềm vì mỗi dịch vụ có thể được kiểm tra và triển khai độc lập, điều này giúp phát hiện lỗi nhanh chóng và dễ dàng khắc phục mà không ảnh hưởng đến các dịch vụ khác.
  - AGILE khuyến khích kiểm thử liên tục trong mỗi sprint, giúp phát hiện lỗi sớm và điều chỉnh kịp thời. Việc kết hợp Microservice với AGILE giúp việc kiểm thử và phát triển trở nên dễ dàng hơn và giúp hệ thống ổn định hơn.
- Tự chủ và làm việc nhóm:
  - Microservice thúc đẩy sự tự chủ trong các nhóm phát triển bởi mỗi dịch vụ có thể được phát triển và triển khai độc lập. Điều này giúp các nhóm có thể làm việc hiệu quả mà không bị ràng buộc lẫn nhau.
  - AGILE cũng thúc đẩy việc làm việc nhóm và cộng tác giữa các thành viên trong đội. Các nhóm phát triển và khách hàng có thể dễ dàng trao đổi và điều chỉnh yêu cầu trong suốt quá trình phát triển.

#### **c. Lợi ích khi kết hợp Microservice và AGILE**

- Phát triển nhanh chóng và linh hoạt: Các dịch vụ nhỏ có thể được phát triển và triển khai độc lập, giúp tăng tốc quá trình phát triển và dễ dàng phản hồi với các thay đổi yêu cầu.
- Khả năng mở rộng dễ dàng: Kiến trúc Microservice giúp mở rộng các dịch vụ riêng biệt mà không ảnh hưởng đến hệ thống tổng thể, phù hợp với phương pháp AGILE khi yêu cầu thay đổi và mở rộng nhanh chóng.
- Cải thiện chất lượng phần mềm: Việc kết hợp các dịch vụ độc lập trong Microservice với kiểm thử liên tục trong AGILE giúp phát hiện lỗi sớm và đảm bảo chất lượng phần mềm cao hơn.
- Dễ dàng duy trì và cải tiến: Với Microservice, việc bảo trì các phần của hệ thống trở nên dễ dàng hơn. AGILE giúp điều chỉnh và cải tiến phần mềm nhanh chóng dựa trên phản hồi từ khách hàng.

#### **d. Một số cách kết hợp các nguyên lý AGILE với thiết kế và phát triển Microservices**

- Áp dụng phương pháp tiếp cận theo lĩnh vực (Domain-driven approach):

Một trong những yếu tố quan trọng của microservices là chúng được tổ chức xung quanh các lĩnh vực kinh doanh, chứ không phải các lớp kỹ thuật. Điều này có nghĩa là mỗi dịch vụ nên có trách nhiệm rõ ràng và hợp lý, dựa trên nhu cầu và kỳ vọng của người dùng và các bên liên quan. Phương pháp tiếp cận theo lĩnh vực giúp ta xác định và mô hình hóa các khái niệm cốt lõi, ranh giới và mối quan hệ của vấn đề kinh doanh, cũng như thiết kế các dịch vụ phản ánh chúng. Nó cũng giúp ta tránh được các phụ thuộc không cần thiết, sự trùng lặp và độ phức tạp giữa các dịch vụ của mình.



- Chấp nhận thiết kế tiến hóa (Embrace evolutionary design):

Một nguyên lý quan trọng khác của AGILE là ta nên chấp nhận sự thay đổi và phản hồi, không nên cố gắng dự đoán hoặc lập kế hoạch tất cả mọi thứ từ trước. Điều này cũng áp dụng cho microservices, vì ta cần có khả năng điều chỉnh và phát triển các dịch vụ khi yêu cầu, công nghệ và môi trường thay đổi. Để làm điều này, ta cần áp dụng phương pháp thiết kế tiến hóa, có nghĩa là thiết kế các dịch vụ với các chức năng tối thiểu và thuộc tính chất lượng đáp ứng nhu cầu hiện tại, sau đó tái cấu trúc và cải tiến chúng theo từng bước dựa trên những hiểu biết và phản hồi mới. Ta cũng cần sử dụng các công cụ và kỹ thuật hỗ trợ continuous integration (tích hợp liên tục), kiểm thử, giao hàng và triển khai dịch vụ của mình.

- Tận dụng tự động hóa và DevOps:

Microservices mang lại rất nhiều độ phức tạp và thử thách trong việc quản lý và triển khai nhiều dịch vụ trên các nền tảng và môi trường khác nhau. Để đối phó với điều này, ta cần tận dụng các phương pháp tự động hóa và DevOps, nhằm tối ưu hóa toàn bộ vòng đời phát triển phần mềm, từ lập kế hoạch đến vận hành. Tự động hóa và DevOps giúp ta giảm thiểu lỗi thủ công, tăng hiệu quả và đảm bảo tính nhất quán cũng như độ tin cậy của các dịch vụ. Chúng cũng cho phép thực hiện các phương pháp AGILE như phát hành thường xuyên, vòng phản hồi ngắn và hợp tác đa chức năng.

- Khuyến khích văn hóa tự chủ và hợp tác:

Một trong những lợi ích chính của microservices là chúng cho phép ta có các đội nhỏ, tự chủ và đa chức năng có thể làm việc độc lập và cung cấp giá trị nhanh hơn. Tuy nhiên, điều này cũng yêu cầu một văn hóa tin tưởng, trao quyền và trách nhiệm giữa các đội, cũng như giao tiếp và phối hợp hiệu quả giữa chúng. Ta cần khuyến khích một văn hóa tự chủ và hợp tác, nơi mỗi đội có quyền sở hữu và thẩm quyền để thiết kế, phát triển, kiểm thử và triển khai dịch vụ của mình, nhưng cũng chia sẻ tầm nhìn, mục tiêu và tiêu chuẩn của tổ chức. Ta cũng cần thiết lập các giao diện, hợp đồng và giao thức rõ ràng giữa các dịch vụ, và sử dụng các công cụ và nền tảng giúp chia sẻ thông tin và quản lý kiến thức.

- Cân bằng các sự đánh đổi và rủi ro:

Cuối cùng, ta cần nhận thức rằng microservices không phải là một "viên đạn bạc" (silver bullet), và chúng đi kèm với các sự đánh đổi và rủi ro mà ta cần cân bằng và giảm thiểu. Ví dụ, microservices có thể làm tăng độ trễ mạng, tiêu thụ băng thông và tỷ lệ lỗi của ứng dụng, cũng như độ phức tạp và chi phí giám sát và xử lý sự cố. Ta cũng cần phải giải quyết các vấn đề như tính nhất quán dữ liệu, bảo mật, quản trị và khả năng mở

rộng. Để giải quyết những thử thách này, cần áp dụng các mẫu thiết kế, kỹ thuật và công nghệ phù hợp, như circuit breakers (công tắc mạch), thử lại (retries), bộ nhớ đệm (caching), nhắn tin (messaging), mã hóa (encryption), xác thực (authentication) và cân bằng tải (load balancing). Ta cũng cần đo lường và đánh giá hiệu suất, chất lượng và giá trị của các dịch vụ, và sử dụng các chỉ số và thông số phù hợp với mục tiêu AGILE của mình.

**1.3** Decompose a software system in microservice and using tool such as VP to represent. Determine requirements and then draw (5 pages):

- a. Decompose e-commerce system [refer to 3.1]
- b. Decompose medicine system
- c. Decompose Tourist Assistant system
- d. Decompose University management system
- e. Decompose Grab Car Management System

### **Bài làm**

#### **a. Phân tách hệ thống thương mại điện tử (E-commerce system):**

Các chức năng chính (service):

##### **User service:**

- Đăng ký, đăng nhập, quản lý tài khoản người dùng
- Xác thực người dùng bằng JWT/OAuth2
- Lưu thông tin hồ sơ người dùng

##### **Product service:**

- Quản lý danh mục, sản phẩm, tồn kho
- Hỗ trợ tìm kiếm, lọc sản phẩm
- Cung cấp thông tin sản phẩm cho UI

##### **Order service:**

- Xử lý đơn hàng từ khách hàng
- Liên kết với Inventory service để kiểm tra tồn kho
- Gửi yêu cầu thanh toán đến Payment service

##### **Payment service:**

- Xử lý thanh toán qua thẻ, ví điện tử, COD
- Kiểm tra trạng thái thanh toán
- Thông báo kết quả thanh toán cho Order service

**Inventory service:**

- Quản lý số lượng hàng tồn kho
- Cập nhật tồn kho khi có đơn hàng mới hoặc hàng về
- Giao tiếp với Order service để xác nhận số lượng hàng

**Shipping service:**

- Xác nhận địa chỉ giao hàng.
- Gửi yêu cầu vận chuyển đến bên thứ ba (GHN, GHTK).
- Theo dõi trạng thái giao hàng.

**Notification service:**

- Gửi email, SMS xác nhận đơn hàng, thanh toán.
- Thông báo cập nhật trạng thái giao hàng.
- Nhắc nhở khách hàng về các chương trình khuyến mãi.

**Review & Rating Service:**

- Khách hàng có thể đánh giá và bình luận về sản phẩm.
- Hệ thống tính điểm trung bình đánh giá.
- Lọc nội dung bình luận không phù hợp.

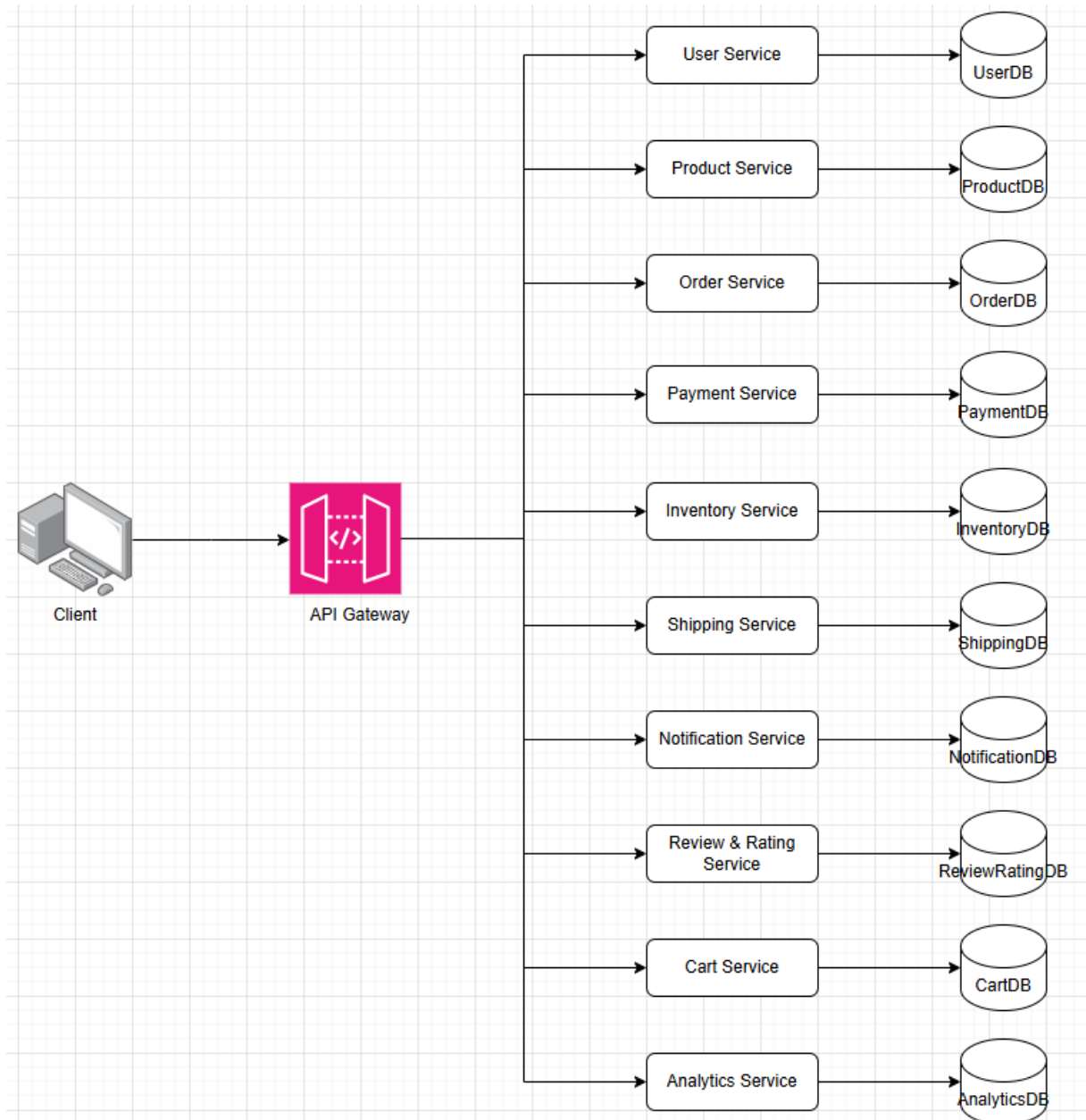
**Cart service:**

- Lưu danh sách sản phẩm khách hàng thêm vào giỏ.
- Cập nhật số lượng, xóa sản phẩm khỏi giỏ hàng.
- Tạo đơn hàng từ giỏ hàng.

**Analytics Service:**

- Thu thập và phân tích dữ liệu hành vi mua sắm.
- Đề xuất sản phẩm dựa trên sở thích khách hàng.
- Hỗ trợ báo cáo doanh thu, sản phẩm bán chạy.

## Sơ đồ Microservice:



## b. Phân tách hệ thống y tế (Medicine system):

### User service:

- Đăng ký, đăng nhập, quản lý thông tin bệnh nhân và bác sĩ.
- Xác thực người dùng bằng JWT/OAuth2
- Quản lý quyền truy cập (bệnh nhân, bác sĩ, quản trị viên)

### Patient Record Service:

- Lưu trữ hồ sơ bệnh án, lịch sử khám bệnh.
- Quản lý đơn thuốc của bệnh nhân.
- Cung cấp dữ liệu sức khỏe cho bác sĩ.

**Doctor Service:**

- Quản lý danh sách bác sĩ, lịch làm việc.
- Hỗ trợ đặt lịch hẹn với bệnh nhân.
- Cung cấp thông tin chuyên môn của bác sĩ.

**Appointment Service:**

- Cho phép bệnh nhân đặt lịch khám với bác sĩ.
- Gửi nhắc nhở lịch hẹn qua Notification Service.
- Đồng bộ với lịch làm việc của bác sĩ.

**Pharmacy Service:**

- Cho phép bác sĩ kê đơn thuốc online.
- Kiểm tra tương tác thuốc và liều lượng phù hợp.
- Đồng bộ đơn thuốc với Pharmacy Service.

**Prescription Service:**

- Xác nhận địa chỉ giao hàng.
- Gửi yêu cầu vận chuyển đến bên thứ ba (GHN, GHTK).
- Theo dõi trạng thái giao hàng.

**Payment Service:**

- Xử lý thanh toán cho dịch vụ khám bệnh và thuốc.
- Hỗ trợ thanh toán qua thẻ, ví điện tử, bảo hiểm y tế.
- Xác nhận hóa đơn với hệ thống bệnh viện.

**Notification Service:**

- Gửi email, SMS nhắc lịch hẹn khám bệnh.
- Thông báo cập nhật đơn thuốc và kết quả xét nghiệm.
- Nhắc nhở bệnh nhân uống thuốc đúng giờ.

**Lab service:**

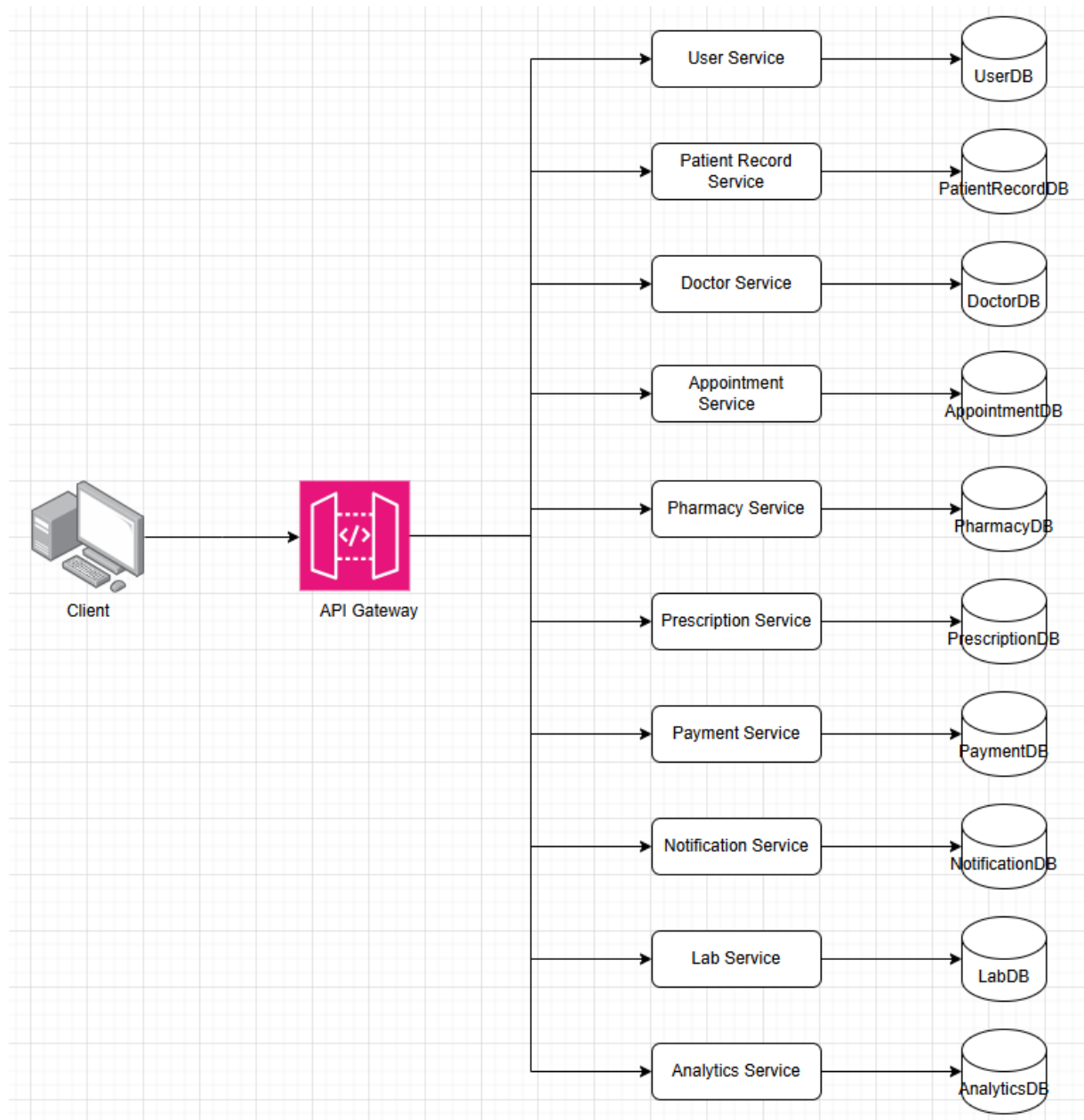
- Quản lý yêu cầu xét nghiệm từ bác sĩ.
- Lưu trữ kết quả xét nghiệm và gửi đến bệnh nhân.
- Tích hợp với các thiết bị y tế.

**Analytics Service:**

- Thu thập và phân tích dữ liệu y tế.

- Báo cáo xu hướng bệnh tật và hiệu quả điều trị.
- Hỗ trợ nghiên cứu y học.

### Sơ đồ Microservice:



### c. Phân tách hệ thống hỗ trợ du lịch (Tourist Assistant system):

#### User service:

- Đăng ký, đăng nhập và quản lý thông tin người dùng.
- Xác thực bằng JWT/OAuth2.
- Lưu trữ lịch sử đặt tour, đánh giá địa điểm.

**Destination Service:**

- Cung cấp thông tin về các điểm du lịch, danh lam thắng cảnh.
- Hỗ trợ tìm kiếm theo vị trí, loại hình du lịch.
- Cập nhật thông tin thời tiết, sự kiện tại điểm đến.

**Tour Package Service:**

- Quản lý danh sách tour du lịch có sẵn.
- Cho phép đặt tour theo sở thích cá nhân.
- Liên kết với Payment Service để xử lý giao dịch.

**Accommodation Service:**

- Cung cấp thông tin khách sạn, homestay, resort.
- Hỗ trợ đặt phòng, kiểm tra phòng trống.
- Đồng bộ với hệ thống khách sạn đối tác.

**Payment Service:**

- Xử lý thanh toán cho các dịch vụ (tour, khách sạn, vé xe).
- Hỗ trợ thanh toán qua thẻ, ví điện tử.
- Cung cấp hóa đơn cho khách hàng.

**Review & Rating Service:**

- Cho phép du khách đánh giá và bình luận về điểm đến, dịch vụ.
- Lọc nội dung không phù hợp.
- Hỗ trợ hiển thị xếp hạng địa điểm theo đánh giá.

**Recommendation Service:**

- Đề xuất tour, khách sạn, nhà hàng dựa trên lịch sử du lịch.
- Phân tích dữ liệu hành vi khách hàng.
- Gợi ý các hoạt động phù hợp theo sở thích.

**Notification Service:**

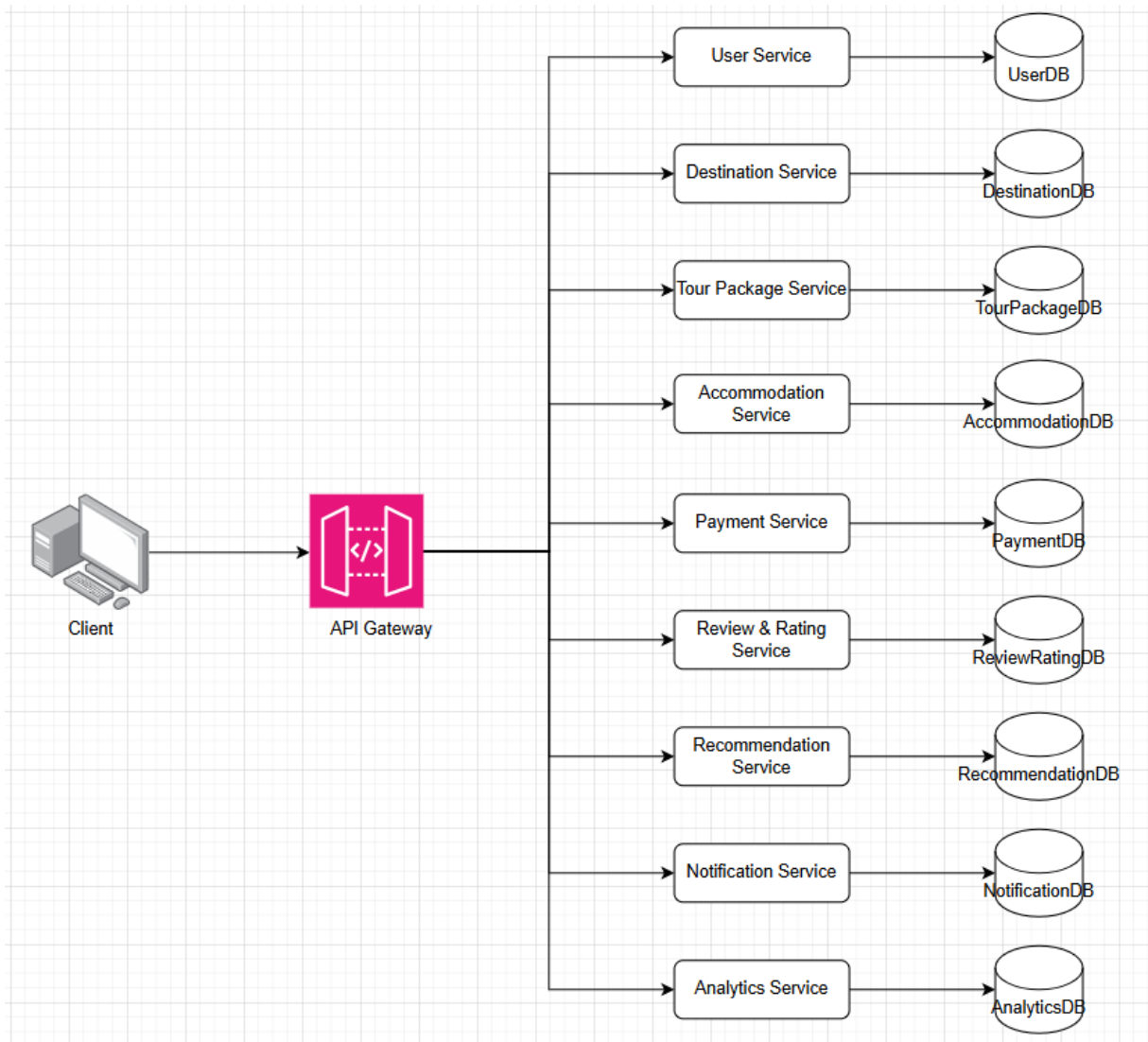
- Gửi email, SMS nhắc nhở về lịch trình du lịch.
- Cập nhật thông tin chuyến bay, thời tiết, cảnh báo du lịch.
- Thông báo ưu đãi, khuyến mãi từ đối tác.

**Analytics Service:**

- Phân tích xu hướng du lịch theo mùa, địa điểm phổ biến.

- Hỗ trợ doanh nghiệp du lịch tối ưu dịch vụ.
- Cung cấp báo cáo về hành vi khách hàng.

#### Sơ đồ Microservice:



#### d. Phân tách hệ thống quản lý đại học (University management system):

##### User service:

- Đăng ký, đăng nhập và quản lý thông tin người dùng (sinh viên, giảng viên, nhân viên).
- Xác thực bằng JWT/OAuth2.
- Quản lý quyền truy cập theo vai trò.

##### Student Service:

- Quản lý thông tin sinh viên, chương trình học.
- Lưu trữ điểm số, kết quả học tập.



- Cập nhật tình trạng học tập, cảnh báo học vụ.

**Faculty & Staff Service:**

- Quản lý danh sách giảng viên, nhân viên trường.
- Theo dõi lịch giảng dạy, hợp đồng lao động.
- Lưu trữ thông tin chuyên môn và nghiên cứu khoa học.

**Course Service:**

- Quản lý danh sách môn học, đề cương môn học.
- Hỗ trợ đăng ký, hủy môn học.
- Cập nhật lịch trình giảng dạy.

**Examination Service:**

- Quản lý lịch thi, đề thi.
- Lưu trữ điểm số, tự động tính điểm trung bình.
- Hỗ trợ nhập điểm, phúc khảo điểm.

**Library Service:**

- Quản lý sách, tài liệu học tập.
- Hỗ trợ mượn/tra sách trực tuyến.
- Cập nhật tình trạng sách trong thư viện.

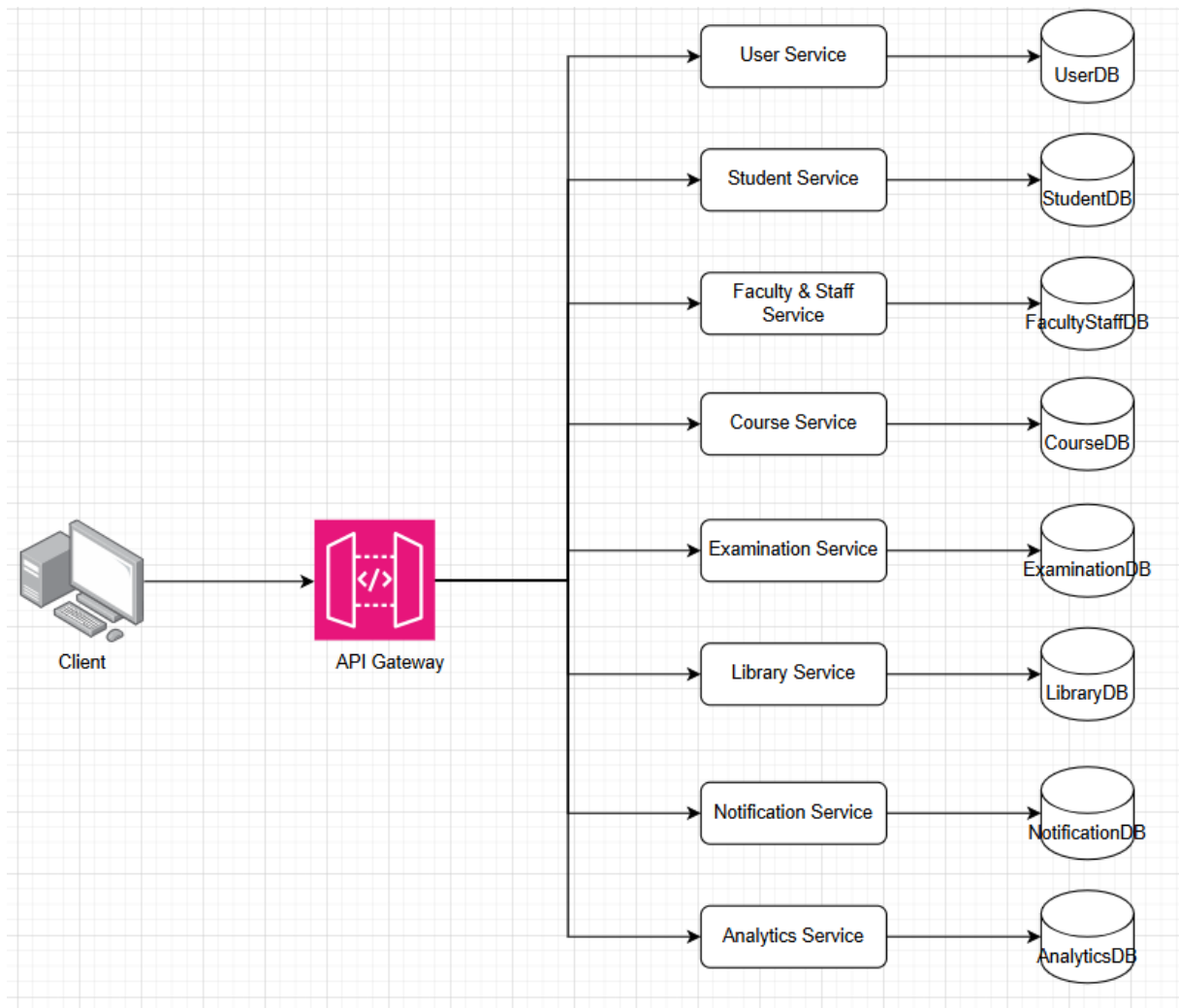
**Notification Service:**

- Gửi thông báo về lịch học, điểm số, sự kiện.
- Hỗ trợ email, SMS cho giảng viên và sinh viên.
- Cập nhật thông báo từ nhà trường.

**Analytics Service:**

- Phân tích xu hướng học tập, tỷ lệ đỗ/trượt.
- Dự báo nhu cầu khóa học trong tương lai.
- Cung cấp báo cáo cho ban quản lý nhà trường.

## Sơ đồ Microservice:



## e. Phân tách hệ thống quản lý GrabCar (Grab Car Management System):

### User service:

- Đăng ký, đăng nhập, quản lý thông tin khách hàng & tài xế.
- Xác thực người dùng bằng JWT/OAuth2.
- Quản lý quyền truy cập (tài xế, khách hàng, admin).

### Driver Management Service:

- Quản lý hồ sơ tài xế (bằng lái, giấy tờ xe, đánh giá).
- Theo dõi trạng thái tài xế (đang hoạt động, đang chờ, nghỉ).
- Quản lý lịch làm việc và thu nhập của tài xế.

### Ride Management Service:

- Theo dõi chuyến đi
- Tạo chuyến đi với giữa người đặt và tài xế, tính toán giá cước

- Hủy chuyến

#### **Payment Service:**

- Xử lý thanh toán qua ví điện tử, thẻ ngân hàng, tiền mặt.
- Quản lý hóa đơn và lịch sử giao dịch.
- Thanh toán doanh thu cho tài xế.

#### **Review & Rating Service:**

- Cho phép khách hàng và tài xế đánh giá, phản hồi về chuyến đi.
- Quản lý điểm đánh giá và cảnh báo tài xế vi phạm.
- Hỗ trợ lọc nội dung đánh giá không phù hợp.

#### **Promotion & Loyalty Service:**

- Quản lý mã giảm giá, chương trình thưởng.
- Tích điểm và ưu đãi cho khách hàng trung thành.
- Cung cấp các chương trình khuyến mãi cho tài xế.

#### **Notification Service:**

- Gửi thông báo về chuyến đi, xác nhận đặt xe.
- Cập nhật trạng thái tài xế, nhắc nhở tài xế mở ứng dụng.
- Thông báo khuyến mãi, ưu đãi cho khách hàng.

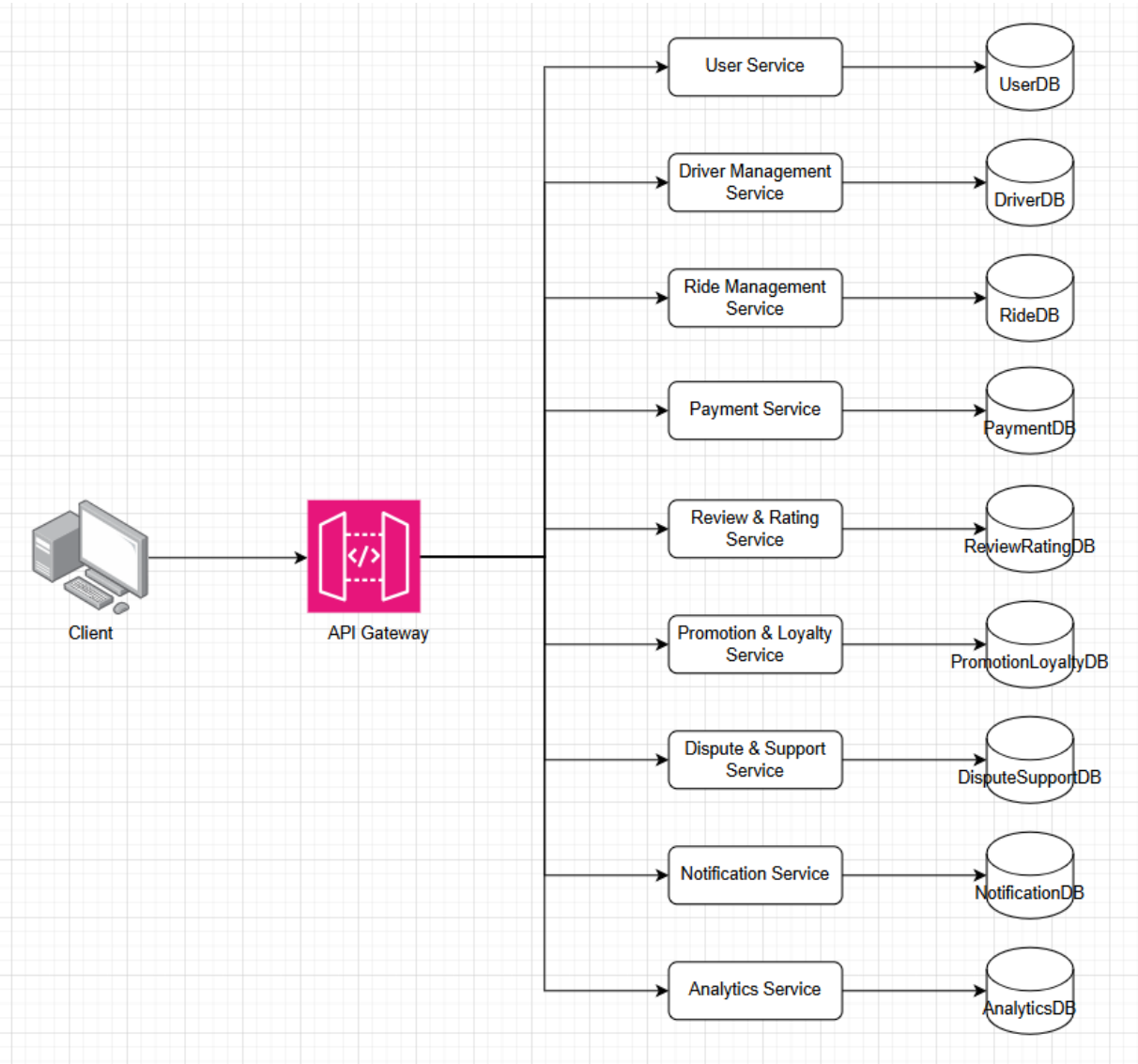
#### **Dispute & Support Service:**

- Xử lý khiếu nại về giá cước, thái độ tài xế, tai nạn.
- Hỗ trợ khách hàng trong các tình huống khẩn cấp.
- Hỗ trợ tài xế và khách qua chatbot hoặc tổng đài.

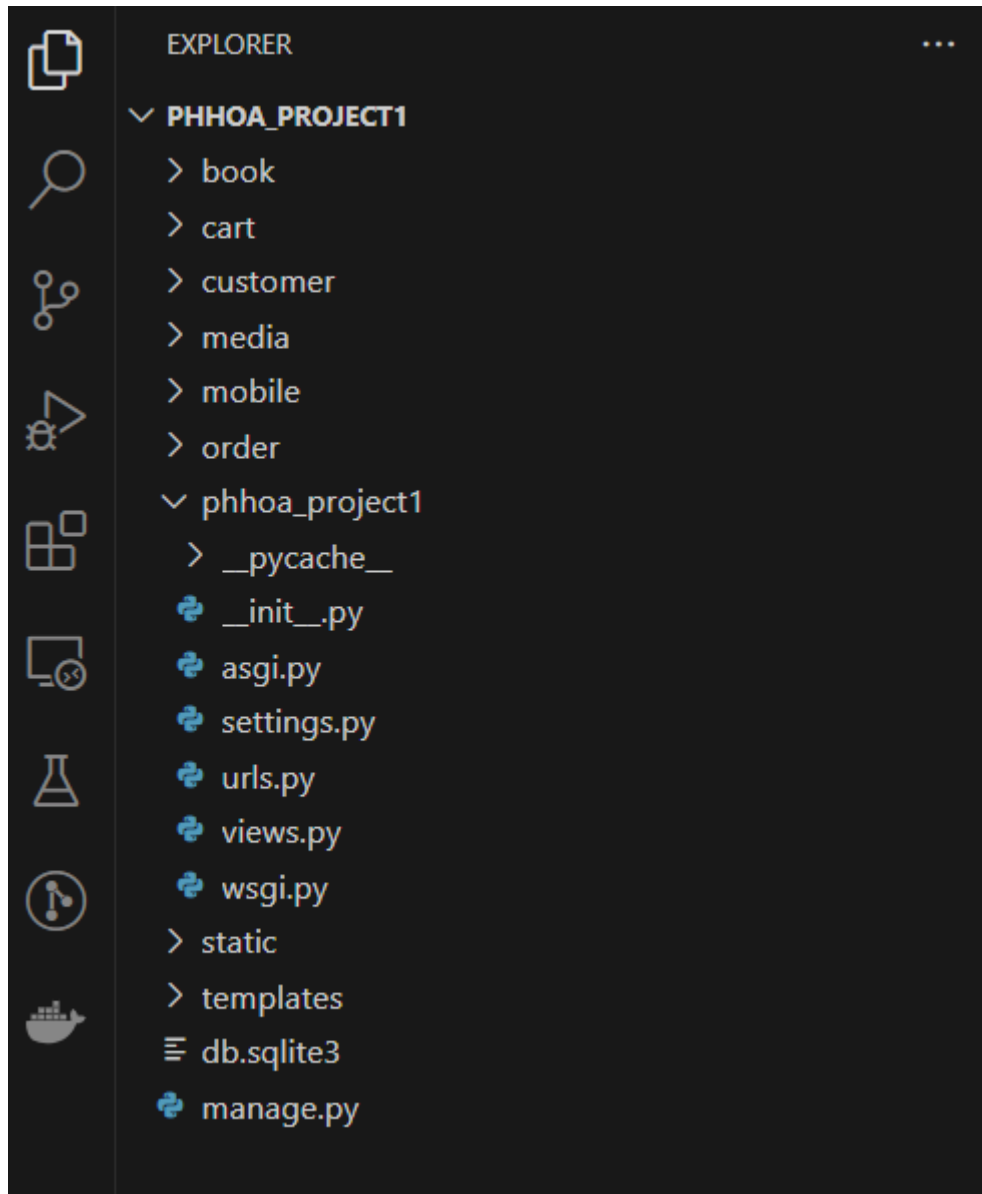
#### **Analytics Service:**

- Phân tích xu hướng đặt xe theo khu vực, thời gian.
- Dự báo nhu cầu xe để điều phối tài xế.
- Cung cấp báo cáo hiệu suất tài xế, thu nhập.

Sơ đồ Microservice:



**1.4** Install python and Django (see guide below). Create PROJECT tensinhvien\_project1 and APPLICATION customer, cart and book (Customer & Item Management in e-commerce)



## 1.5 Develop modules **customer**, **cart**, **book** in microservice with Django

// refer to ChatGPT for customer, cart and book – attributes and methods

### 1) Customer

#### a. models.py

```
customer > models.py > ...
1  from django.contrib.auth.models import User
2  from django.db import models
3
4  class Customer(models.Model):
5      user = models.OneToOneField(User, on_delete=models.CASCADE, null=True, blank=True)
6      address = models.TextField(blank=True, null=True)
7      contact = models.CharField(max_length=15, blank=True, null=True)
8
9      def __str__(self):
10         return self.user.username if self.user else "Unlinked Customer"
11
```

#### b. admin.py

```
customer > admin.py > ...
1  from django.contrib import admin
2  from .models import Customer
3
4  class CustomerAdmin(admin.ModelAdmin):
5      list_display = ('user', 'get_email', 'get_name', 'address', 'contact')
6
7      def get_name(self, obj):
8         return obj.user.username # ✅ Get name from Django User model
9         get_name.short_description = "Tên Khách Hàng"
10
11     def get_email(self, obj):
12         return obj.user.email # ✅ Get email from Django User model
13         get_email.short_description = "Email"
14
15     admin.site.register(Customer, CustomerAdmin)
16
```

#### c. apps.py

```
customer > apps.py > ...
1  from django.apps import AppConfig
2
3
4  class CustomerConfig(AppConfig):
5      default_auto_field = 'django.db.models.BigAutoField'
6      name = 'customer'
7
```

#### d. urls.py

```
customer > 📄 urls.py > ...
1  from django.urls import path
2  from .views import (
3      customer_list, signup, customer_login, customer_logout,
4      customer_main, all_books, add_to_cart, buy_now
5  )
6
7  urlpatterns = [
8      path('list/', customer_list, name='customer_list'),
9      path('signup/', signup, name='signup'),
10     path('login/', customer_login, name='customer_login'),
11     path('logout/', customer_logout, name='customer_logout'),
12     path('main/', customer_main, name='customer_main'),
13     path('books/', all_books, name='all_books'),
14     path('add_to_cart/<int:book_id>/', add_to_cart, name='add_to_cart'),
15     path('buy_now/<int:book_id>/', buy_now, name='buy_now'),
16 ]
17 |
```

#### e. views.py

```
customer > 📄 views.py > ...
1  from django.shortcuts import render, redirect, get_object_or_404
2  from django.contrib.auth import authenticate, login, logout
3  from django.contrib.auth.decorators import login_required
4  from django.contrib.auth.models import User
5  from django.contrib import messages
6  from django.http import JsonResponse, HttpResponse
7  from django.views.decorators.csrf import csrf_exempt
8
9  from .models import Customer
10 from book.models import Book
11 from order.models import Order
12
13 # 🚀 View: List all customers
14 @login_required
15 def customer_list(request):
16     customers = Customer.objects.all()
17     return render(request, 'customer_list.html', {'customers': customers})
18
19
20 # 🚀 View: User Registration
21 @csrf_exempt
22 def signup(request):
23     if request.method == 'POST':
24         username = request.POST['username']
25         email = request.POST['email']
26         password = request.POST['password']
27         address = request.POST['address']
28         contact = request.POST['contact']
29
30         # 🔥 Create User & Customer Profile
31         user = User.objects.create_user(username=username, email=email, password=password)
32         Customer.objects.create(user=user, address=address, contact=contact)
33
34         login(request, user) # Auto login after signup
35         return redirect('customer_main')
36
```

```

37     return render(request, 'customer/signup.html')
38
39
40 # 🚀 View: User Login
41 @csrf_exempt
42 def customer_login(request):
43     if request.method == 'POST':
44         username = request.POST['username']
45         password = request.POST['password']
46
47         user = authenticate(request, username=username, password=password)
48         if user is not None:
49             login(request, user)
50             return redirect('customer_main')
51         else:
52             return render(request, 'customer/login.html', {'error': 'Invalid username or password'})
53
54     return render(request, 'customer/login.html')
55
56
57 # 🚀 View: Customer Dashboard
58 @login_required
59 def customer_main(request):
60     customer = get_object_or_404(Customer, user=request.user)
61
62     # Fetch orders for the logged-in customer
63     orders = Order.objects.filter(customer=customer).order_by('-date')
64
65     return render(request, 'customer/customer_main.html', {
66         'customer_name': customer.user.username,
67         'orders': orders
68     })
69
70

```

```

71 # 🚀 View: Logout User
72 def customer_logout(request):
73     logout(request)
74     messages.success(request, "Đăng xuất thành công!")
75     return redirect('landing_page') # Redirect to home page
76
77
78 # 🚀 View: Show All Books with Filters
79 def all_books(request):
80     search_query = request.GET.get('search', '') # Get search query
81     price_filter = request.GET.get('price', '') # Get price filter
82     sort_by = request.GET.get('sort', '') # Get sorting order
83
84     books = Book.objects.all()
85
86     # 🔍 Apply Search Filter
87     if search_query:
88         books = books.filter(name__icontains=search_query)
89
90     # 💰 Apply Price Filter
91     if price_filter == 'low_to_high':
92         books = books.order_by('price')
93     elif price_filter == 'high_to_low':
94         books = books.order_by('-price')
95
96     # 📖 Apply A-Z Sorting
97     if sort_by == 'a_to_z':
98         books = books.order_by('name')
99     elif sort_by == 'z_to_a':
100         books = books.order_by('-name')
101
102     return render(request, 'customer/all_books.html', {'books': books})
103
104

```



```

105 from django.shortcuts import get_object_or_404
106 from django.http import JsonResponse
107 from django.contrib.auth.decorators import login_required
108 from django.views.decorators.csrf import csrf_exempt
109 import json
110
111 from cart.models import Cart
112 from book.models import Book
113 from customer.models import Customer
114
115
116 @csrf_exempt
117 @login_required # ✅ Ensure only logged-in users can add items to cart
118 def add_to_cart(request, book_id):
119     if request.method != "POST":
120         return JsonResponse({'message': 'Phương thức không hợp lệ'}, status=405)
121
122     try:
123         customer = get_object_or_404(Customer, user=request.user)
124         book = get_object_or_404(Book, id=book_id)
125
126         # ✅ Parse JSON request body
127         data = json.loads(request.body)
128         quantity = data.get("quantity", 1)
129
130         # ✅ Get or create cart item
131         cart_item, created = Cart.objects.get_or_create(customer=customer, book=book)
132         if not created:
133             cart_item.quantity += quantity # Update quantity if already in cart
134             cart_item.save()
135
136         # ✅ Update session cart count
137         request.session['cart_items_count'] = Cart.objects.filter(customer=customer).count()
138

```

```

138
139         return JsonResponse({
140             'message': 'Sách đã được thêm vào giỏ hàng!',
141             'cart_count': request.session['cart_items_count']
142         })
143
144     except Exception as e:
145         return JsonResponse({'message': 'Lỗi khi thêm vào giỏ hàng', 'error': str(e)}, status=500)
146
147
148
149 # 🚀 View: Buy Now (Redirect to Checkout)
150 @login_required
151 def buy_now(request, book_id):
152     book = get_object_or_404(Book, id=book_id)
153     return render(request, 'checkout.html', {'book': book})
154

```

## 2) Cart

### a. models.py

```
cart > models.py > ...
1  from django.db import models
2  from customer.models import Customer
3  from book.models import Book
4
5  class Cart(models.Model):
6      customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
7      book = models.ForeignKey(Book, on_delete=models.CASCADE)
8      quantity = models.IntegerField(default=1)
9
10     @staticmethod
11     def get_cart_items_by_customer(customer_id):
12         return Cart.objects.filter(customer=customer_id)
13
14     def __str__(self):
15         return f"Cart - {self.customer.name} - {self.book.name} ({self.quantity})"
16
```


### b. admin.py

```
cart > admin.py
1  from django.contrib import admin
2
3  # Register your models here.
4
```




### c. apps.py

```
cart > apps.py > ...
1  from django.apps import AppConfig
2
3
4  class CartConfig(AppConfig):
5      default_auto_field = 'django.db.models.BigAutoField'
6      name = 'cart'
7
```

#### d. urls.py

```
cart >  urls.py > ...
1  from django.urls import path
2  from .views import add_to_cart, view_cart, remove_from_cart
3
4  urlpatterns = [
5      path('add/<int:book_id>/', add_to_cart, name='add_to_cart'),
6      path('', view_cart, name='view_cart'),
7      path('remove/<int:cart_id>/', remove_from_cart, name='remove_from_cart'),
8  ]
9
```

#### e. views.py

```
cart >  views.py > ...
1  from django.shortcuts import render, redirect, get_object_or_404
2  from django.http import JsonResponse
3  from django.contrib.auth.decorators import login_required
4  from django.views.decorators.csrf import csrf_exempt
5  from django.utils.decorators import method_decorator
6  import json
7
8  from .models import Cart
9  from book.models import Book
10 from customer.models import Customer
11
12 @csrf_exempt
13 @login_required
14 def add_to_cart(request, book_id):
15     if request.method != "POST":
16         return JsonResponse({'message': 'Phương thức không hợp lệ'}, status=405)
17
18     try:
19         book = get_object_or_404(Book, id=book_id)
20         customer = get_object_or_404(Customer, user=request.user)
21
22         # Parse JSON data from request
23         data = json.loads(request.body)
24         quantity = data.get('quantity', 1)
25
26         #  Get or create cart item
27         cart_item, created = Cart.objects.get_or_create(customer=customer, book=book)
28         if not created:
29             cart_item.quantity += quantity # Update quantity if already in cart
30             cart_item.save()
31
32         #  Update session cart count
33         request.session['cart_items_count'] = Cart.objects.filter(customer=customer).count()
34
```

```

34
35     return JsonResponse({
36         'message': 'Sách đã được thêm vào giỏ hàng!',
37         'cart_count': request.session['cart_items_count']
38     })
39
40 except Exception as e:
41     return JsonResponse({'message': 'Lỗi khi thêm vào giỏ hàng', 'error': str(e)}, status=500)
42
43
44 @login_required
45 def view_cart(request):
46     customer = get_object_or_404(Customer, user=request.user)
47     cart_items = Cart.get_cart_items_by_customer(customer.id)
48
49     # ✅ Update session cart count
50     request.session['cart_items_count'] = cart_items.count()
51
52     total_price = sum(item.quantity * item.book.price for item in cart_items)
53
54     return render(request, 'cart.html', {
55         'cart_items': cart_items,
56         'total_price': total_price
57     })
58
59
60 from django.http import JsonResponse
61 from django.views.decorators.csrf import csrf_exempt
62 from .models import Cart
63
64 @csrf_exempt
65 def remove_from_cart(request, cart_id):
66     if not request.user.is_authenticated:
67         return JsonResponse({'message': 'Bạn cần đăng nhập để xóa sản phẩm!', 'success': False}, status=401)
68

```

```

69     try:
70         cart_item = Cart.objects.get(id=cart_id, customer=request.user.customer)
71         cart_item.delete()
72
73         # ✅ Update cart count in session
74         remaining_cart_items = Cart.objects.filter(customer=request.user.customer)
75         request.session['cart_items_count'] = sum(item.quantity for item in remaining_cart_items)
76
77         return JsonResponse({
78             'message': '✅ Sản phẩm đã bị xóa!',
79             'success': True,
80             'cart_count': request.session['cart_items_count'] # ✅ Return updated cart count
81         })
82     except Cart.DoesNotExist:
83         return JsonResponse({'message': '❌ Sản phẩm không tồn tại!', 'success': False}, status=404)
84
85
86 from django.http import JsonResponse
87 from django.views.decorators.csrf import csrf_exempt
88 from .models import Cart
89 import json
90
91 @csrf_exempt
92 def update_cart(request, cart_id):
93     if not request.user.is_authenticated:
94         return JsonResponse({'message': 'Bạn cần đăng nhập để cập nhật giỏ hàng!', 'success': False}, status=401)
95
96     try:
97         data = json.loads(request.body)
98         new_quantity = int(data.get('quantity', 1))
99
100         cart_item = Cart.objects.get(id=cart_id, customer=request.user.customer)
101         cart_item.quantity = new_quantity
102         cart_item.save()
103

```

```

104         # ✅ Update cart count in session
105         remaining_cart_items = Cart.objects.filter(customer=request.user.customer)
106         request.session['cart_items_count'] = sum(item.quantity for item in remaining_cart_items)
107
108         return JsonResponse({
109             'message': '✅ Số lượng đã cập nhật!',
110             'success': True,
111             'cart_count': request.session['cart_items_count']
112         })
113     except Cart.DoesNotExist:
114         return JsonResponse({'message': '❌ Sản phẩm không tồn tại!', 'success': False}, status=404)
115
116

```

### 3) Book

#### a. models.py

```

book > models.py > ...
1  from django.db import models
2
3  class Book(models.Model):
4      id = models.AutoField(primary_key=True)
5      name = models.CharField(max_length=100) # Book title
6      author = models.CharField(max_length=100) # Author name
7      price = models.IntegerField(default=0) # Book price
8      description = models.TextField(blank=True, null=True) # Book description
9      image = models.ImageField(upload_to='uploads/books/', blank=True, null=True) # Book cover image
10
11     @staticmethod
12     def get_books_by_id(ids):
13         return Book.objects.filter(id__in=ids)
14
15     @staticmethod
16     def get_all_books():
17         return Book.objects.all()
18
19     def __str__(self):
20         return f"{self.name} - {self.author}" # Show title & author in admin panel
21

```

#### b. admin.py

```

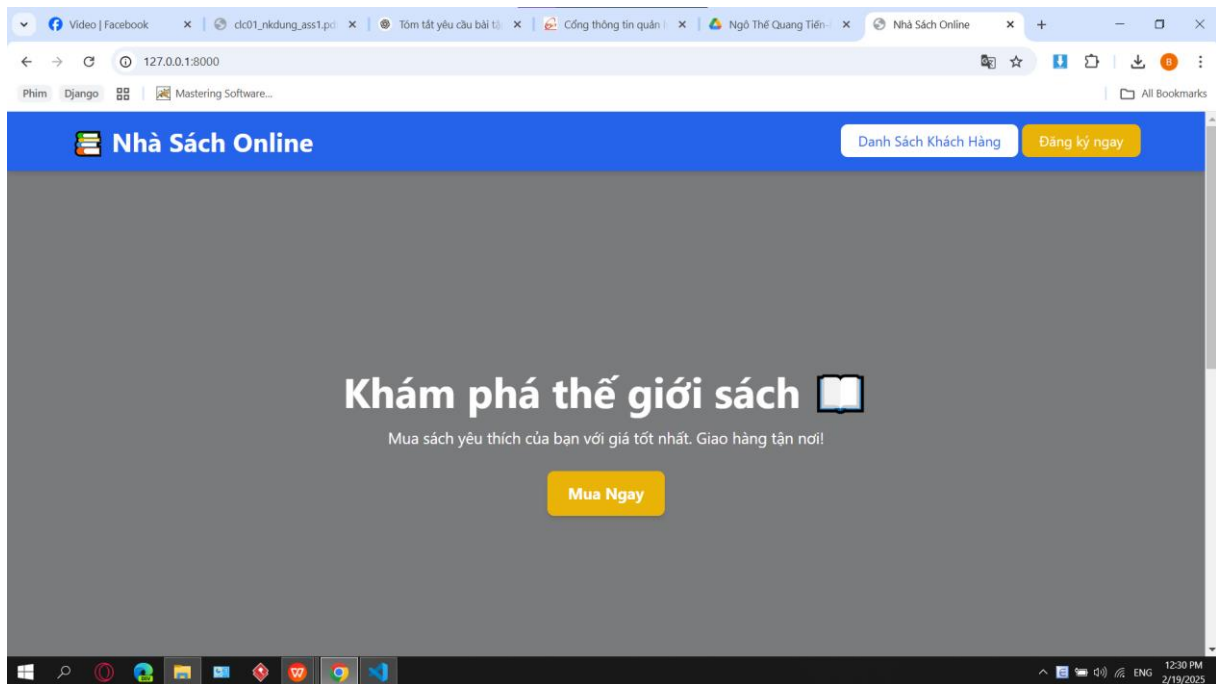
book > admin.py > ...
1  from django.contrib import admin
2  from .models import Book
3
4  @admin.register(Book)
5  class BookAdmin(admin.ModelAdmin):
6      list_display = ('id', 'name', 'author', 'price')
7      search_fields = ('name', 'author')
8

```

### c. apps.py

```
book > apps.py > ...
1  from django.apps import AppConfig
2
3
4  class BookConfig(AppConfig):
5      default_auto_field = 'django.db.models.BigAutoField'
6      name = 'book'
7
```

### Demo:



Customer List

ID	Name	Email	Contact	Address
3	Tien	tien@gmail.com	0123456789	asd

Đăng Ký

Họ và Tên

Email

Mật khẩu

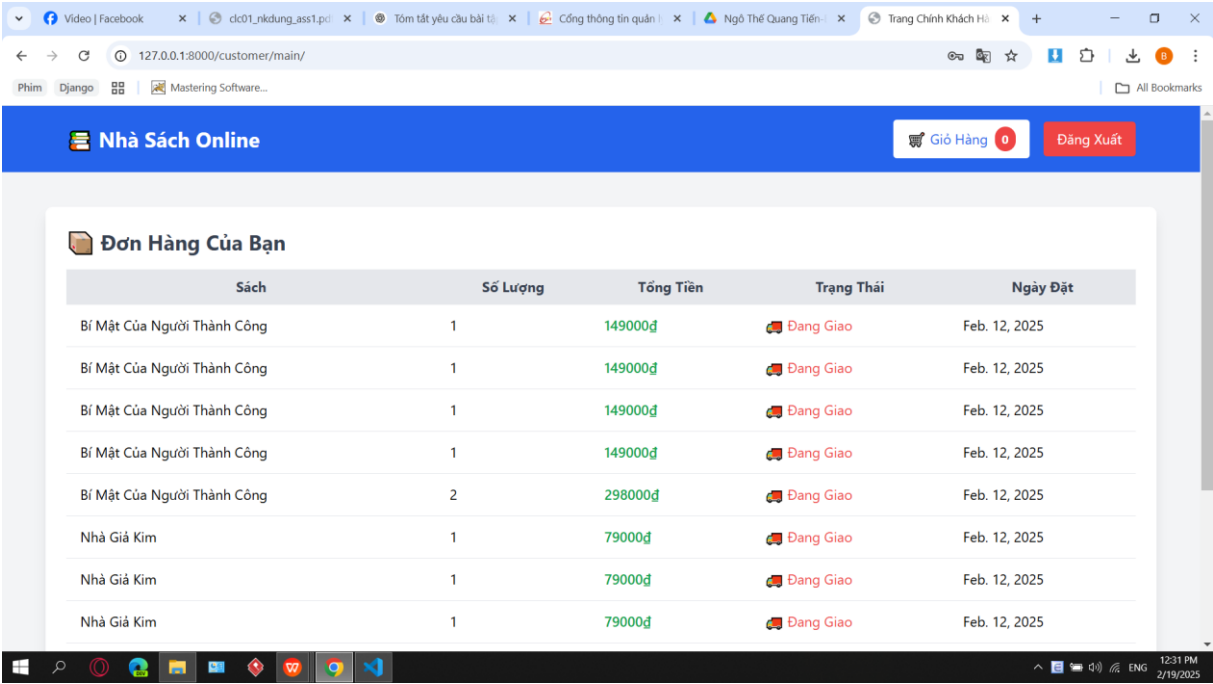
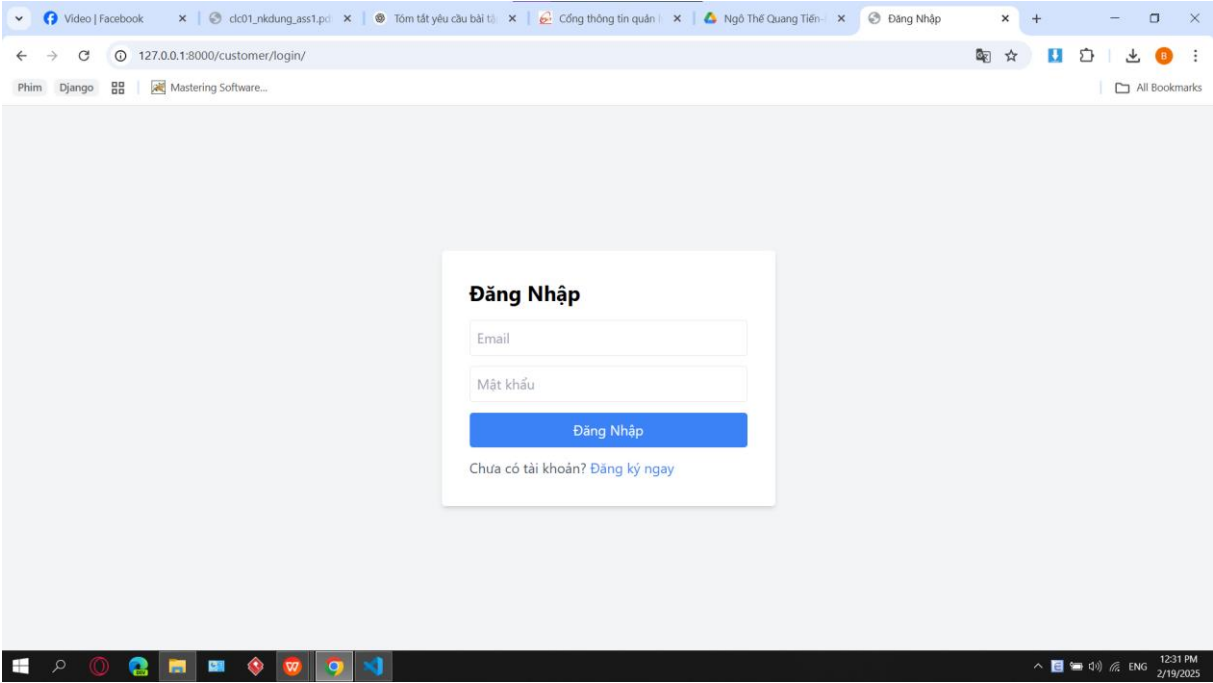
Địa chỉ

Số điện thoại

Đăng Ký

Đã có tài khoản? Đăng nhập

Please fill out this field.





Video | Facebook

cic01\_nkdung\_ass1.p...

Tóm tắt yêu cầu bài t...

Cổng thông tin quâ...

Ngô Thế Quang Tiến

Giỏ Hàng

127.0.0.1:8000/cart/

Phím Django Mastering Software...

All Bookmarks

Nhà Sách Online

Trang Chủ

Giỏ Hàng 1

### Giỏ Hàng Của Bạn

Sách	Tác Giả	Giá	Số Lượng	Tổng	Hành Động	
	Đắc Nhân Tâm	Dale Carnegie	159000đ	1	159,000đ	Xóa

Tổng Tiền: 159,000đ

Thanh Toán Ngay

Video | Facebook

cic01\_nkdung\_ass1.p...

Tóm tắt yêu cầu bài t...

Cổng thông tin quâ...

Ngô Thế Quang Tiến

Tất Cả Sách

127.0.0.1:8000/customer/books/

Phím Django Mastering Software...

All Bookmarks

Nhà Sách Online

Trang Chủ

Giỏ Hàng 1

Lọc theo giá

Sắp xếp

Áp dụng

### Danh Sách Sách

**Bi Mật Của Người Thành Công**  
Tác giả: Missing  
149000đ

Mua Ngay

**Nhà Già Kim**  
Tác giả: Paulo Coelho  
79000đ

Mua Ngay

**Đắc Nhân Tâm**  
Tác giả: Dale Carnegie  
159000đ

Mua Ngay