I. Nhiệm vụ:

1. Tìm hiểu vòng đời và quy trình phát triển phẩn mềm

2. Phân biệt quy trình phát triển phần mềm Agile (Scrum) và Waterfall

3. Phân biệt kiến trúc Microservices và kiến trúc Monolithic

II. Sinh viên thực hiện:

1. Đỗ Thành Đạt

2. Đặng Tiểu Minh

3 . Lê Hải Đăng Lâm

4. Nguyễn Trần Long Hảo

5. Đỗ Thị Khuê

6. Võ Hoài Liên

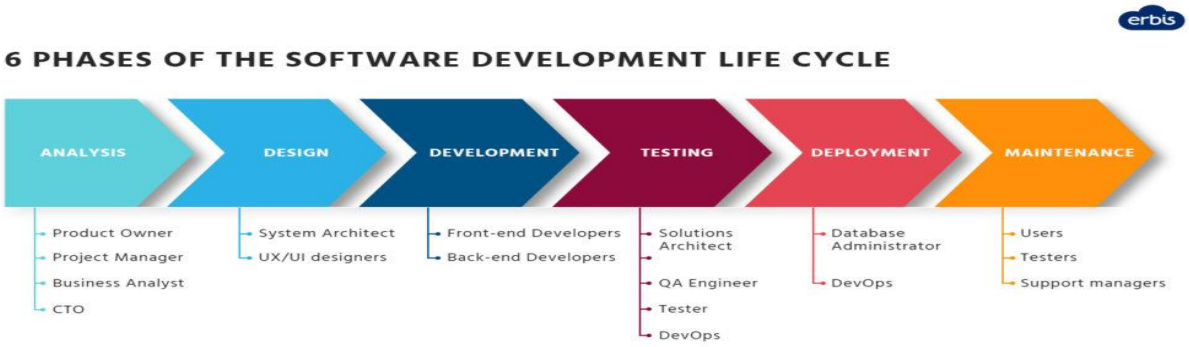
7. Lê Chí Đức

III. Nội dung báo cáo:

1. Vòng đời phát triển phần mềm là gì ?

Software Development Lifecycle (SDLC) còn gọi là vòng đời phát triển phần mềm. một vòng đời phát hành phần mềm là một sự tổng hợp các pha phát triển phần mềm từ giai đoạn sơ khai cho đến giai đoạn hoàn chỉnh, và cuối cùng là công bố phần mềm đó hoặc phiên bản nâng cấp mới. Việc chia thành nhiều giai đoạn như vậy giúp cho việc quản lý, sửa lỗi và bảo trì phần mềm dễ dàng hơn.

1.1. Giới thiệu



Quy trình phát triển phần mềm (SDLC), cũng được gọi là phát triển ứng dụng vòng đời , là một thuật ngữ được sử dụng trong hệ thống kỹ thuật , hệ thống thông tin và công nghệ phần mềm để mô tả một quá trình lập kế hoạch, tạo, thử nghiệm và triển khai một hệ thống thông tin. Một số mô hình SDLC hoặc các phương pháp đã được tạo ra, chẳng hạn như thác nước , xoắn ốc , phát triển phần mềm Agile , tạo mẫu nhanh , gia tăng và đồng bộ hóa và ổn định. SDLC được sử dụng trong quá trình phát triển một dự án CNTT, nó mô tả các giai đoạn khác nhau liên quan đến dự án từ bản vẽ, thông qua việc hoàn thành dự án.

1.2. Các giai đoạn trong quy trình

Khung phát triển vòng đời phát triển hệ thống cung cấp một chuỗi các hoạt động cho các nhà thiết kế hệ thống và các nhà phát triển để làm theo. Nó bao gồm một bộ các bước hoặc các giai đoạn trong đó mỗi giai đoạn của SDLC sử dụng các kết quả của bước trước.

SDLC tuân thủ các giai đoạn quan trọng cần thiết cho các nhà phát triển, chẳng hạn như lập kế hoạch , phân tích , thiết kế và thực hiện và được giải thích trong phần dưới đây. Nó bao gồm việc đánh giá hệ thống hiện tại, thu thập thông tin, nghiên cứu khả thi và yêu cầu phê duyệt. Một số mô hình SDLC đã được tạo ra: thác nước, đài phun nước, xoắn ốc, xây dựng và sửa chữa, tạo mẫu nhanh, gia tăng, đồng bộ hóa và ổn định. Người cao tuổi

nhất, và nổi tiếng nhất, là mô hình thác nước: một dãy các giai đoạn trong đó đầu ra của mỗi giai đoạn trở thành đầu vào cho kế tiếp. Các giai đoạn này có thể được mô tả và phân chia theo nhiều cách khác nhau, bao gồm những điều sau:

1) **Phân tích sơ bộ** : Mục tiêu của giai đoạn 1 là tiến hành phân tích sơ bộ, đưa ra các giải pháp thay thế, mô tả chi phí và lợi ích và đệ trình một kế hoạch sơ bộ với các khuyến nghị.

- Tiến hành phân tích sơ bộ: trong bước này, bạn cần phải tìm ra các mục tiêu của tổ chức và bản chất và phạm vi của vấn đề đang nghiên cứu. Ngay cả khi một vấn đề chỉ đề cập đến một phân đoạn nhỏ của tổ chức, bạn cần phải tìm ra mục đích của tổ chức chính là những gì. Sau đó, bạn cần xem vấn đề đang được nghiên cứu phù hợp với họ như thế nào.

2) **Phân tích hệ thống, định nghĩa yêu cầu** : Xác định các mục tiêu của dự án thành các chức năng được xác định và hoạt động của ứng dụng dự định. Đó là quá trình thu thập và giải thích các sự kiện, chẩn đoán các vấn đề và đề xuất cải tiến hệ thống. Phân tích nhu cầu thông tin của người dùng cuối và cũng loại bỏ bất kỳ sự không nhất quán và không đầy đủ trong các yêu cầu này.

- Thu thập Dữ kiện: Các yêu cầu của người dùng cuối có được thông qua tài liệu, phỏng vấn của khách hàng, quan sát và bảng câu hỏi,

- Kiểm tra hệ thống hiện tại: Xác định ưu và nhược điểm của hệ thống hiện tại tại chỗ, để chuyển các ưu và tránh những khuyết điểm trong hệ thống mới.

- Phân tích hệ thống được đề xuất: Giải pháp cho những thiếu sót trong bước hai được tìm thấy và bất kỳ đề xuất người dùng cụ thể được sử dụng để chuẩn bị các thông số kỹ thuật.

3) **Thiết kế hệ thống** : mô tả chi tiết các tính năng và hoạt động mong muốn, bao gồm bố cục màn hình, quy tắc kinh doanh , sơ đồ quá trình , mã giả và các tài liệu khác.

- Trong thiết kế hệ thống , các chức năng và hoạt động thiết kế được mô tả chi tiết, bao gồm bố cục màn hình, quy tắc kinh doanh, sơ đồ quá trình và các tài liệu khác. Đầu ra của giai đoạn này sẽ mô tả hệ thống mới như là một tập hợp các mô-đun hoặc các hệ thống con.

4) **Phát triển** : Mã thực được viết ở đây.

- Khi tiếp nhận tài liệu thiết kế hệ thống, công việc được chia thành các mô đun / đơn vị và bắt đầu viết mã thực. Kể từ đó, trong giai đoạn này mã được tạo ra do đó nó là

trọng tâm chính cho các nhà phát triển. Đây là giai đoạn dài nhất của vòng đời phát triển phần mềm.

5) **Tích hợp và thử nghiệm** : Mang tất cả các mảnh vào một môi trường thử nghiệm đặc biệt, sau đó kiểm tra lỗi, lỗi và khả năng tương tác. Mã được kiểm tra ở nhiều cấp độ khác nhau trong kiểm thử phần mềm . Sau đây là các loại thử nghiệm có thể có liên quan, tùy thuộc vào loại hệ thống đang được phát triển:

6) **Cài đặt, triển khai** : Giai đoạn cuối cùng của sự phát triển ban đầu, nơi phần mềm được đưa vào sản xuất và chạy kinh doanh thực tế.

- Việc triển khai hệ thống bao gồm các thay đổi và cải tiến trước khi ngừng hoạt động hoặc hoàng hôn của hệ thống. Duy trì hệ thống là một khía cạnh quan trọng của SDLC. Khi nhân sự chủ chốt thay đổi vị trí trong tổ chức, những thay đổi mới sẽ được thực hiện. Có hai cách để phát triển hệ thống; có phương pháp tiếp cận truyền thống (cấu trúc) và hướng đối tượng . Kỹ thuật Thông tin bao gồm cách tiếp cận hệ thống truyền thống, còn được gọi là kỹ thuật phân tích cấu trúc và thiết kế. Phương pháp tiếp cận hướng đối tượng xem hệ thống thông tin như một tập hợp các đối tượng được tích hợp với nhau để tạo ra một hệ thống thông tin đầy đủ và hoàn chỉnh.

7) **Bảo trì** : Trong quá trình bảo trì của SDLC, hệ thống được đánh giá để đảm bảo nó không trở nên lỗi thời. Một khi khách hàng bắt đầu sử dụng hệ thống đã phát triển thì những vấn đề thực sự xuất hiện và cần được giải quyết theo thời gian. Quá trình này, nơi chăm sóc được thực hiện cho các sản phẩm phát triển được gọi là bảo trì.

1.3. Kết luận

Một số người cho rằng SDLC không còn áp dụng cho các mô hình như Agile, nhưng nó vẫn là một thuật ngữ rộng rãi được sử dụng trong giới công nghệ.Thay vì xem SDLC từ quan điểm sức mạnh hoặc điểm yếu, điều quan trọng hơn là phải áp dụng các phương pháp hay nhất từ mô hình SDLC và áp dụng nó cho bất kỳ điều gì phù hợp nhất cho phần mềm được thiết kế.

2. Phân biệt quy trình phát triển phần mềm Agile (Scrum) và Waterfall

2.1. Thế nào là mô hình Waterfall/Thác nước:

Phương pháp mô hình thác mà còn được gọi là mô hình vòng tuần hoàn dạng vòng lặp. Mô hình thác nước theo thứ tự tuần tự và do đó nhóm phát triển dự án chỉ chuyển sang giai đoạn phát triển hoặc thử nghiệm tiếp theo nếu bước trước đó hoàn thành thành công.

Ưu điểm:

- Nó là một trong những mô hình dễ nhất để quản lý. Bởi vì bản chất của nó, mỗi giai đoạn có quá trình cụ thể .
- Nó hoạt động tốt cho các dự án có kích thước nhỏ , các yêu cầu dễ hiểu.
- Phân phối dự án nhanh hơn
- Quá trình và kết quả cũng được ghi nhận.
- Phương pháp dễ điều chỉnh cho các đội chuyển dịch
- Phương pháp quản lý dự án này có lợi cho việc quản lý các phụ thuộc.

Hạn chế:

- Nó không phải là một mô hình lý tưởng cho một dự án kích thước lớn
- Nếu yêu cầu không rõ ràng ngay từ đầu thì đó là phương pháp kém hiệu quả hơn.
- Rất khó di chuyển trở lại cái giai đoạn trước đó để thay đổi .
- Quá trình thử nghiệm bắt đầu khi quá trình phát triển kết thúc. Do đó, nó có nguy cơ cao của các lỗi được tìm thấy sau giai đoạn phát triển, và rất tốn kém để sửa các lỗi.

2.2. Mô hình Agile(Scrum) là gì?

Phương pháp nhanh là một phương phát lặp liên tục gia đoạn phát triển và thử nghiệm trong quá trình phát triển phần mềm. Trong mô hình này, các hoạt động phát triển và thử nghiệm là đồng thời, không giống như mô hình Thác. Quá trình này cho phép giao tiếp nhiều hơn giữa khách hàng, nhà phát triển, người quản lý và người thử nghiệm.

Ưu điểm:

- Nó là quá trình khách hàng tập trung. Vì vậy, nó đảm bảo rằng khách hàng liên tục tham gia trong mọi giai đoạn.
- Các nhóm agile được tạo động lực và tự tổ chức để có khả năng cung cấp kết quả tốt hơn từ các dự án phát triển.
- Phương pháp phát triển phần mềm nhanh đảm bảo rằng chất lượng của sự phát triển được duy trì

- Quá trình này hoàn toàn dựa trên tiến trình gia tăng. Vì vậy, khách hàng và nhóm biết chính xác những gì được hoàn thành và những gì không. Điều này làm giảm rủi ro trong quá trình phát triển.

Hạn chế:

- Nó không phải là phương pháp hữu ích cho các dự án phát triển nhỏ.
- Nó đòi hỏi một chuyên gia để có những quyết định quan trọng trong cuộc họp.
- Chi phí thực hiện một phương pháp nhanh hơn một chút so với các phương pháp phát triển khác.
- Dự án có thể dễ dàng đi theo chiều hướng xấu nếu người quản lý dự án không rõ ràng kết quả họ muốn.

2.3. Sự khác biệt giữa mô hình Agile và Waterfall:

Agile	Waterfall
Nó tách vòng đời phát triển dự án thành chạy nước rút.	Quá trình phát triển phần mềm được chia thành các giai đoạn riêng biệt.
Nó theo một cách tiếp cận gia tăng	Phương pháp thác nước là một quá trình thiết kế tuần tự.
Phương pháp nhanh được biết đến với tính linh hoạt của nó.	Thác là một phương pháp phát triển phần mềm có cấu trúc nên hầu hết thời gian nó có thể khá cứng nhắc.
Agile có thể được coi là một bộ sưu tập của nhiều dự án khác nhau.	Phát triển phần mềm sẽ được hoàn thành như một dự án duy nhất.
Agile là một phương pháp khá linh hoạt cho phép thay đổi được thực hiện trong các yêu cầu phát triển dự án ngay cả khi kế hoạch ban đầu đã được hoàn thành.	Không có phạm vi thay đổi các yêu cầu khi phát triển dự án bắt đầu.

Agile	Waterfall
Phương pháp nhanh , theo một cách tiếp cận phát triển lặp lại vì quy hoạch, phát triển, tạo mẫu và các giai đoạn phát triển phần mềm khác có thể xuất hiện nhiều lần.	Tất cả các giai đoạn phát triển dự án như thiết kế, phát triển, thử nghiệm, vv được hoàn thành một lần trong mô hình Thác
Kế hoạch kiểm tra được xem xét sau mỗi lần chạy nước rút	Kế hoạch kiểm tra hiếm khi được thảo luận trong giai đoạn thử nghiệm.
Phát triển nhanh là một quá trình trong đó các yêu cầu được dự kiến sẽ thay đổi và phát triển.	Phương pháp này là lý tưởng cho các dự án có yêu cầu nhất định và thay đổi không được mong đợi.
Trong phương pháp Agile, thử nghiệm được thực hiện đồng thời với phát triển phần mềm.	Trong phương pháp này, giai đoạn "Thử nghiệm" xuất hiện sau giai đoạn "Xây dựng"
Agile giới thiệu tư duy sản phẩm, nơi sản phẩm phần mềm đáp ứng nhu cầu của khách hàng cuối cùng và thay đổi chính nó theo nhu cầu của khách hàng.	Mô hình này cho thấy một tư duy dự án và đặt trọng tâm của nó hoàn toàn vào việc hoàn thành dự án.
Agat methdology hoạt động đặc biệt tốt với Time & Materials hoặc tài trợ không cố định. Nó có thể làm tăng căng thẳng trong các kịch bản giá cố định.	Giảm rủi ro trong các hợp đồng giá cố định của công ty bằng cách nhận được thỏa thuận rủi ro vào đầu quá trình.
Thích các nhóm nhỏ nhưng chuyên dụng với mức độ phối hợp và đồng bộ	Phối hợp / đồng bộ hóa nhóm rất hạn chế.

Agile	Waterfall
hóa cao.	
Chủ sở hữu sản phẩm với nhóm chuẩn bị các yêu cầu chỉ là về mỗi ngày trong một dự án.	Phân tích kinh doanh chuẩn bị các yêu cầu trước khi bắt đầu dự án.
Đội kiểm tra có thể tham gia vào các yêu cầu thay đổi mà không có vấn đề gì.	Thật khó để thử nghiệm bắt đầu bất kỳ thay đổi nào về yêu cầu.
Mô tả chi tiết dự án có thể được thay đổi bất cứ lúc nào trong quá trình SDLC.	Mô tả chi tiết cần thực hiện phương pháp tiếp cận phát triển phần mềm thác nước.
Các thành viên của Nhóm Agile có thể hoán đổi cho nhau, do đó, chúng hoạt động nhanh hơn. Cũng không cần thiết cho các nhà quản lý dự án vì các dự án được quản lý bởi toàn bộ nhóm	Trong phương pháp thác nước, quy trình luôn đơn giản như vậy, người quản lý dự án đóng một vai trò thiết yếu trong mọi giai đoạn của SDLC.

3. Phân biệt kiến trúc Microservices và kiến trúc Monolithic:

3.1. Kiến trúc Microservices:

3.1.1. Kiến trúc Microsevices là gì?

Microservice có thể hiểu là chia một khối phần mềm thành các dịch vụ - service nhỏ hơn, có thể triển khai trên các service khác nhau. Mỗi service sẽ xử lý từng phần công việc, chức năng khác nhau để đáp ứng các yêu cầu của phần mềm và được kết nối với

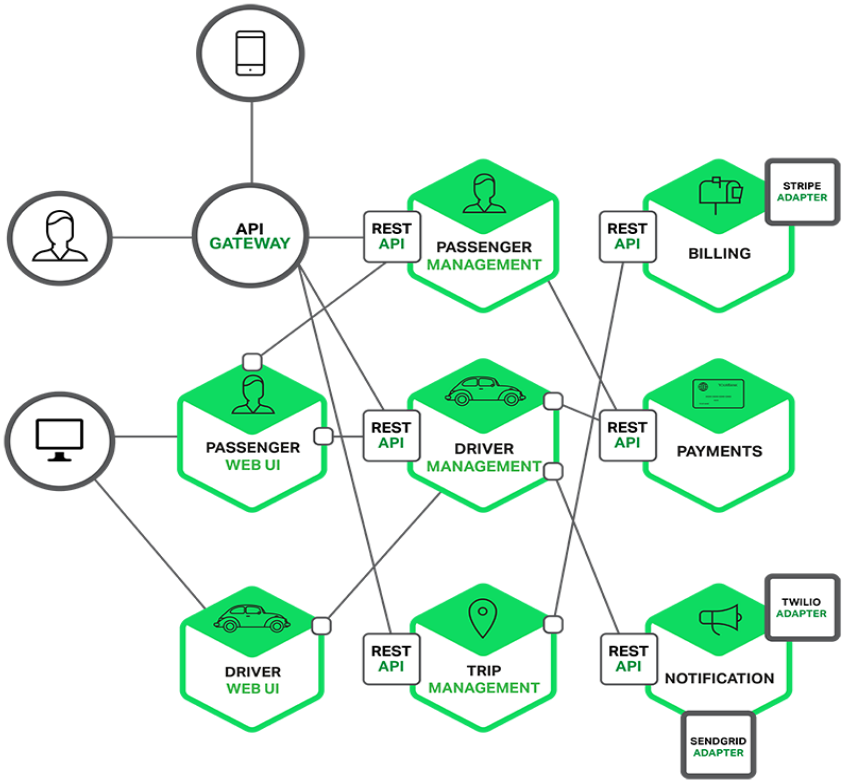
nhau thông qua các giao thức khác nhau như http, SOA, soket, Message queue (Active MQ, Kafka) … để truyền tải dữ liệu.

Từng dịch vụ có thể phát triển hoạt động độc lập nên việc xây dựng ứng dụng sẽ có tính linh động cao hơn. Mỗi dịch vụ sẽ được đảm nhận bởi những nhóm kĩ sư khác nhau với ngôn ngữ, framework khác nhau.

Với mô hình Microservices, khi muốn phát triển thêm tính năng sản phẩm, chúng ta chỉ cần phát triển dịch vụ mới cho tính năng đó và giao tiếp với những dịch vụ khác thông qua các giao tiếp - API được qui định sẵn.

*Các **đặc điểm** của **Microservice***

- **Decoupling** - Các service trong một hệ thống phần lớn được tách rời. Vì vậy, toàn bộ ứng dụng có thể dễ dàng được xây dựng, thay đổi và thu nhỏ.
- **Componentization - Microservices** được coi là các thành phần độc lập có thể dễ dàng thay thế và nâng cấp.
- **Business Capabilities** - mỗi một thành phần trong kiến trúc microservice rất đơn giản và tập trung vào một nhiệm vụ duy nhất.
- **Autonomy** - các lập trình viên hay các nhóm có thể làm việc độc lập với nhau trong quá trình phát triển.
- **Continous Delivery** - Cho phép phát hành phần mềm thường xuyên, liên tục.
- **Responsibility**.
- **Decentralized Governance** - không có mẫu chuẩn hóa hoặc bất kỳ mẫu công nghệ nào. Được tự do lựa chọn các công cụ hữu ích tốt nhất để có thể giải quyết vấn đề.
- **Agility - microservice** hỗ trợ phát triển theo mô hình Agile.



Hình 1: Kiến trúc Microservice của Uber (Source: vietnix.vn)

3.1.2. Ưu điểm, nhược điểm:

Ưu điểm

- Dễ hiểu codebase của hệ thống: Mỗi module chỉ đảm nhiệm một nhiệm vụ nhất định, do đó việc hiểu codebase và chức năng của module đó sẽ trở nên đơn giản hơn nhiều so với hệ thống monolithic.
- Dễ mở rộng quy mô: Mỗi service được thiết kế, phát triển và triển khai độc lập với nhau. Vì vậy ta có thể dễ dàng cập nhật một phần mềm riêng lẻ thông qua microservices tương ứng mà không ảnh hưởng đến toàn bộ hệ thống.
- Chống chịu lỗi tốt: Các ứng dụng trong microservices vẫn có thể hoạt động dù cho bất kỳ service nào khác gặp lỗi, bởi các service trong kiến trúc này gần như là độc lập với nhau.
- Cho phép thử nghiệm nhiều công nghệ khác nhau: Các developer có thể linh hoạt thử nghiệm nhiều loại công nghệ trong quá trình tạo ra service, sở dĩ vì có ít sự

phụ thuộc về mặt công nghệ giữa các module hơn nên việc chuyển đổi công cụ cũng không quá khó khăn.

Nhược điểm

- Phức tạp để thiết kế và triển khai: Dịch vụ được phát triển bởi nhiều công nghệ, ngôn ngữ vậy nên sẽ có nhiều thành phần phải quản lý.
- Việc kiểm tra trở nên phức tạp hơn: Testing trong các ứng dụng monolithic tương đối đơn giản khi ta chỉ cần khởi chạy ứng dụng và kiểm tra kết nối của nó với database. Mặt khác, trong kiến trúc microservices thì các service cần phải được khởi chạy và thử nghiệm riêng lẻ.
- Giao tiếp giữa các service trở nên phức tạp hơn: Việc chia một ứng dụng thành nhiều module nhỏ sẽ khiến việc giao tiếp trở nên phức tạp và tốn nhiều chi phí hơn. Ngoài ra, cách giao tiếp của từng hệ thống khác nhau cũng là khác nhau, vì vậy đôi khi có thể cần thêm một trình phiên dịch trong ứng dụng.
- Yêu cầu cơ sở hạ tầng phức tạp: Không chỉ quản lý các service trong Microservice mà phải quản lý cả các hệ thống messaging (liên quan đến việc giao tiếp giữa các service).

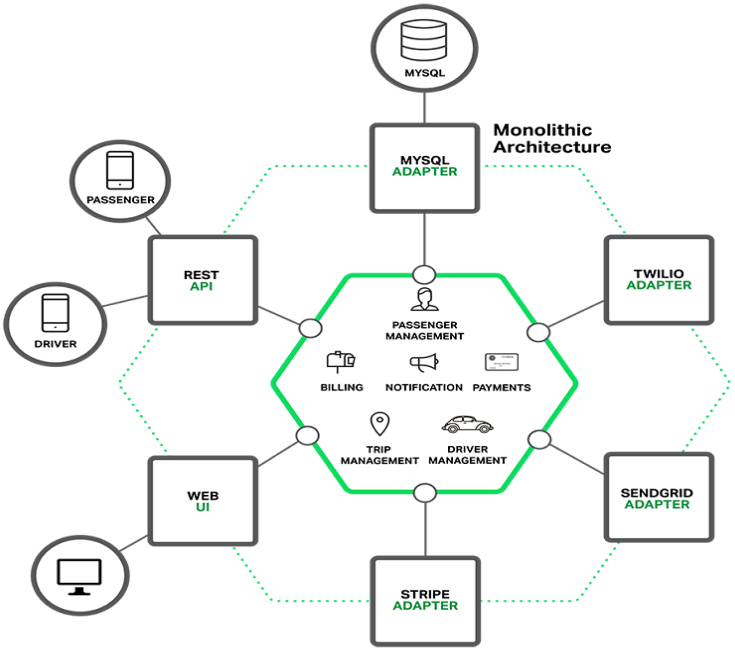
3.1.3. Trường hợp áp dụng:

- Dự án lớn, khả năng mở rộng và phát triển mạnh mẽ, đối tượng người dùng là rất nhiều hoặc có tiềm năng phát triển cực lớn trong tương lai.
- Team lớn hoặc rất lớn, nguồn nhân lực có kiến thức và kinh nghiệm tốt để phát triển.

3.2. Kiến trúc Monolithic:

3.2.1. Kiến trúc Monolithic là gì?

Monolithic là kiến trúc phần mềm dạng nguyên khối, nghĩa là mọi tính năng sẽ nằm trong một project. Giả sử mình có một project web bán hàng triển khai theo kiến trúc Monolithic, thì các module như khách hàng, đơn hàng, sản phẩm,… sẽ được gói gọn trong project đó.



Hinh 2: Kiến trúc monolithic của Uber (Source: vietnix.vn)

3.2.2. Ưu điểm, nhược điểm:

Ưu điểm:

- Đơn giản để phát triển và triển khai: Đơn ngôn ngữ, tất cả các thành phần của một khối đều được tập trung hóa, có nhiều framwork hỗ trợ.
- Dễ kiểm tra: Việc debug và test dễ dàng hơn so với Microservice vì chỉ trên một IDE, một project.
- Quản lý bảo mật đơn lẻ: Mặc dù có một số lợi ích bảo mật khi chia ứng dụng thành các microservices riêng biệt, nhưng việc sử dụng monolithic có nghĩa là bảo mật được xử lý ở một nơi, thay vì phải theo dõi các lỗ hổng trên tất cả các microservices.
- Hiệu suất tốt hơn nếu được triển khai đúng cách.
- Yêu cầu về cơ sở hạ tầng không quá phức tạp.

Nhược điểm

- Phức tạp theo thời gian: Khi một ứng dụng phát triển, source code ngày một lớn hơn (requirement change, refactor, fix bug...) dẫn đến việc khó quản lý và khiến cho người mới mất thời gian để hiểu.

- Khó mở rộng quy mô: Để mở rộng quy mô các ứng dụng nguyên khối, ứng dụng phải được mở rộng cùng một lúc bằng cách thêm các tài nguyên tính toán bổ sung, được gọi là **chia tỷ lệ dọc**. Điều này có thể tốn kém và có thể có giới hạn về mức độ một ứng dụng có thể mở rộng theo chiều dọc.
- Hạn chế về công nghệ: Việc thêm hoặc thay đổi chức năng cho một khối nguyên khối có thể cực kỳ khó khăn do các phụ thuộc lồng vào nhau được tìm thấy trong một khối nguyên khối. Tùy thuộc vào nhu cầu của ứng dụng của bạn, các nhà phát triển có thể bị giới hạn về những tính năng mới mà họ có thể triển khai với một khối.
- Điểm lỗi duy nhất: Bởi vì tất cả các phần của ứng dụng được liên kết chặt chẽ, một vấn đề ở bất kỳ vị trí nào trong mã có thể làm hỏng toàn bộ ứng dụng.

3.2.3. Trường hợp áp dụng:

- Dự án nhỏ, khả năng mở rộng trong tương lai ít, đối tượng người dùng không nhiều.
- Team nhỏ, nhân sự ít, không yêu cầu về kỹ năng và kinh nghiệm nhiều.
- Yêu cầu phát triển nhanh.