**TRƯỜNG ĐẠI HỌC ĐÀ NẴNG**
**ĐẠI HỌC BÁCH KHOA**

**TRUNG TÂM VNPT-IT KHU VỰC 3**

# BÁO CÁO THỰC TẬP TUẦN 1

## I. Nhiệm vụ:

Phân biệt kiến trúc Microservices và kiến trúc Monolithic

## II. Sinh viện thực hiện:

1. Nguyễn Trần Long Hảo
2. Đỗ Thị Khuê
3. Võ Hoài Liên
4. Lê Chí Đức

## III. Nội dung báo cáo

### 1. Kiến trúc Microservices

### 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.

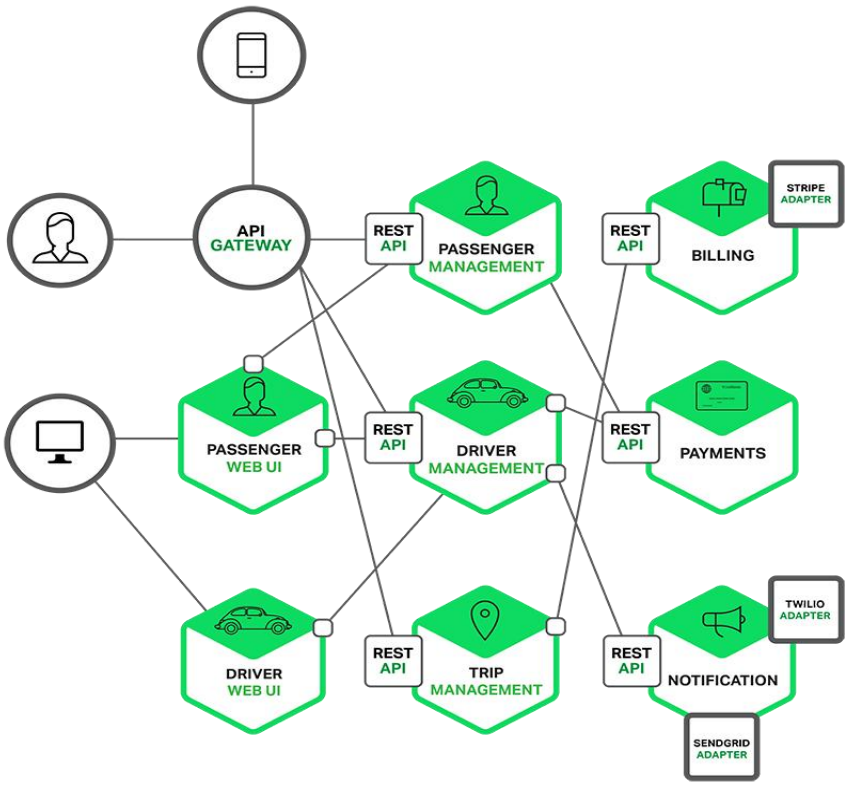
Từng dịch vụ có thể phát triển hoạt động độc lập nên việc xây dựng ứng dụng sẽ có tính linh động cao hơn. Mỗi dịch vụ sẽ được đảm nhận bởi những nhóm kĩ sư khác nhau với ngôn ngữ, framework khác nhau.

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)*

## 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).

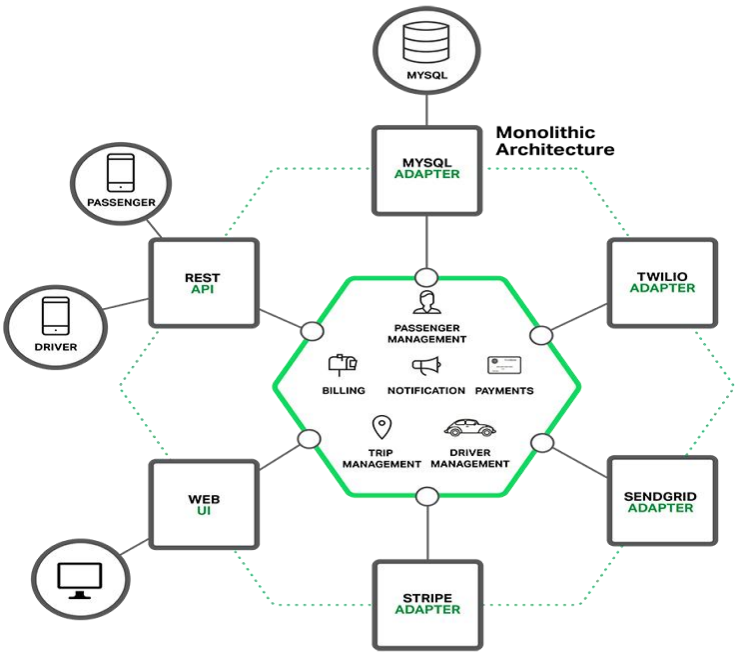
### 1.3. Trường hợp áp dụng

- 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.

## 2. Kiến trúc Monolithic

### 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.




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

## 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.

### 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.