BÁO CÁO THỰC TẬP DOANH NGHIỆP TUẦN: 1

- Họ và tên sinh viên: Trần Văn Toàn

- Mã số sinh viên: 20225937

- Lớp: Việt Nhật 01

- Tên doanh nghiệp thực tập: VCCorp

- Tuần thực tập số (Từ ngày: 09 / 06 / 2025 đến 15 / 06 / 2025

- Người hướng dẫn tại doanh nghiệp: Ngô Văn Vĩ

I. Nội dung công việc trong tuần

STT	Nội dung công việc	Kết quả đạt được	Ghi chú
01	Tìm hiểu về Microfrontend	Hiểu được Microfrontend là gì, bối cảnh để sử dụng và các hoạt động của nó	
02	Tìm hiểu về 12factor	Nắm cơ bản về 12 nguyên tắc để tạo nên 1 ứng dụng	
03	Clean Architecture	Hiểu được thêm được mô hình để xây dựng ứng dụng mới ngoài MVC đã biết từ trước	

II. Kỹ năng hoặc kiến thức đã học được

Microfrontend

1. Bối cảnh và thách thức của các dự án frontend quy mô lớn

Khi ứng dụng frontend phức tạp, quy mô lớn(có nhiều màn hình, nhiều team cùng làm dự án, mỗi team lại sử dụng 1 framework khác nhau), mã nguồn trở nên đồ sộ thời gian build/deploy tăng, dễ confict Git, build lỗi khó kiểm soát, performance UX kém -> màn hình trắng lâu, người dùng phải load tất cả những thứ có thể họ không sự dụng.

Giải pháp truyền thống (lazy-load, SSR, share UI libs, submodule git…) không giải quyết triệt để, vì **frontend đòi hỏi toàn bộ mã nguồn có mặt khi build và deploy** .

2. Microfrontend là gì

Vì phương pháp truyền thông chưa giải quyết được khi frontend vẫn đòi hỏi toán bộ mã nguồn nên Microfrontend ra đời.

Là kiến trúc đưa khái niệm microservice vào frontend: chia UI thành các **app con (widgets)** độc lập (có thể là một component nhỏ hoặc cả page) do các team khác nhau đảm nhận, deploy riêng, dùng các framework khác nhau (React, Angular, Vue…).

Mỗi widget deploy độc lập, app chính (app-shell) chỉ load những gì cần thiết, giảm thiểu bundling không cần thiết .

3. Giải pháp để ứng dụng Microfrontend: Webpack Module Federation

a. Tổng quan

Module Federation là 1 tính năng được giới thiệu trong Webpack, cho phép nhiều ứng dụng(module) chia sẻ mã nguồn một cách động tại runtime. Là một giải pháp cốt lõi giúp các phần frontend được xây dựng và deploy 1 cách độc lập, và chương trình chính có thể gọi bất kì ứng dụng nào mà người dùng cần.

b. Các hoạt động

Widget(ứng dụng con) sẽ build riêng, export những thành phần cần chia sẻ(component, fucntion, UI)

App-shell(ứng dụng mẹ) cấu hình để runtime fetch và sử dụng các module tử widget thông qua URL (remoteEntry.js)

c. Cấu hình Module Federation

- Cấu hình ứng dụng con

Widget sử dụng Webpack ModuleFederationPlugin như sau:

```
new ModuleFederationPlugin({
  name: 'widget1',
  filename: 'remoteEntry.js',
  exposes: {
    './App': './src/App',
  },
  shared: {
    react: { singleton: true },
    'react-dom': { singleton: true },
  },
});
```

Giải thích:

- name: tên định danh widget.
- filename: file đầu ra chứa module runtime.
- exposes: chỉ định component/folder nào được chia sẻ ra ngoài.
- shared: tránh load lại các thư viện lớn (như React) nhiều lần

- Cấu hình ứng dụng chính

```
new ModuleFederationPlugin({
  remotes: {
    widget1: 'widget1@http://localhost:3001/remoteEntry.js',
  },
  shared: {
    react: { singleton: true },
    'react-dom': { singleton: true },
  },
});
```

- remotes: khai báo các module từ xa cần load.
- Khi runtime, app-shell tạo một <script> để load remoteEntry.js, sau đó lấy component từ window.widget1.get(...).

d. Tích hợp và phân quyền

Giải pháp này còn hỗ trợ tải widget dự trên phân quyền người dùng, giúp giảm dung lượng ban đầu phải load vào, linh hoạt tối ưu hóa trả nghiệm cho người dùng

Các widget có thể viết bằng nhiều framework khác nhau, App-shell sử dụng các wrapper component cho từng framework để render các widget đúng cách, App-shell chỉ cần biết cách render đúng cách làm cho từng team chọn công nghệ khác nhau mà ko bị ràng buộc về công nghệ duy nhất.

12factor

1. Codebase – Một codebase, nhiều deploy khác nhau

- Codebase là tập mã nguồn được quản lý trong hệ thống kiểm soát phiên bản (Git), lưu trữ trong 1 repo.
- Một app chỉ có 1 codebase duy nhất, nhưng có thể có nhiều bản deploy khác nhau.
- Nếu có nhiều codebase đó không còn là 1 app, mà là hệ thống phân tán -> mỗi thành phần của hệ thống là 1 app riêng biệt.
- Việc chia sẻ mã nguồn giữa nhiều app là không phù hợp với nguyên tắc twelve-factor. Thay vào đó, mã dùng chung nên được tách thành library và quản lý thông qua dependency manager.
- Mọi deploy để sử dụng chung 1 codebase nhưng có thể ở các phiên bản khác nhau.

2. Dependencies – Khai báo và cách ly
- Ứng dụng twelve-factor phải khai báo rõ ràng tất cả dependencies thông qua tệp khai báo (manifest) và cách ly chúng khỏi hệ thống bằng công cụ thích hợp.
- Không được phụ thuộc vào các gói cài đặt sẵn trên hệ điều hành (system-wide packages) hoặc các công cụ hệ thống như curl, ImageMagick.
- Ví dụ về công cụ:
    - Python: pip và virtualenv
- Lợi ích: Giúp developer mới dễ dàng thiết lập môi trường làm việc chỉ với mã nguồn, trình thông dịch ngôn ngữ, và công cụ quản lý dependencies.
- Nếu cần sử dụng công cụ hệ thống, nên đóng gói (vendor) chúng trong app để đảm bảo tính nhất quán trên mọi môi trường.
3. Config
- Cấu hình là những thông tin có thể thay đổi theo từng môi trường triển khai (development, staging, production…), ví dụ: thông tin đăng nhập dịch vụ, tên máy chủ, tài nguyên kết nối (database, cache...).
- Theo mô hình twelve-factor, cấu hình phải được tách biệt hoàn toàn khỏi mã nguồn, không được hard-code trong code.
- Cấu hình nên được lưu trữ trong các biến môi trường (environment variables) – giúp dễ dàng thay đổi giữa các môi trường mà không làm thay đổi mã nguồn.
- Twelve-factor app không nhóm các cấu hình theo tên môi trường (như development, test, production), mà quản lý mỗi biến môi trường một cách độc lập, giúp mở rộng triển khai linh hoạt và tránh sự phức tạp không cần thiết khi thêm các môi trường mới.
4. Backing services
- Dịch vụ hỗ trợ (backing services) là các dịch vụ mà ứng dụng truy cập thông qua mạng, ví dụ: cơ sở dữ liệu (MySQL, CouchDB), hệ thống hàng đợi (RabbitMQ), SMTP (Postfix), bộ nhớ đệm (Memcached), hoặc dịch vụ bên thứ ba (Amazon S3, Postmark, New Relic...).
- Twelve-factor app coi tất cả các dịch vụ hỗ trợ – dù là cục bộ hay từ bên thứ ba – đều là tài nguyên bên ngoài có thể cấu hình thông qua biến môi trường (env).
- Ứng dụng có thể chuyển đổi giữa các dịch vụ cục bộ và bên ngoài mà không cần thay đổi mã nguồn – chỉ cần cập nhật cấu hình.
- Mỗi dịch vụ (dù cùng loại) được xem là một tài nguyên riêng biệt. Ví dụ: hai cơ sở dữ liệu MySQL khác nhau được coi là hai tài nguyên độc lập.
- Điều này đảm bảo rằng ứng dụng có liên kết lỏng (loose coupling) với các dịch vụ hỗ trợ, giúp linh hoạt trong triển khai và phục hồi (ví dụ: thay thế nhanh cơ sở dữ liệu khi gặp sự cố mà không cần sửa mã).
5. Build, release, run
- Xây dựng (Build):

Biến mã nguồn (ở một commit cụ thể) thành một bản dựng thực thi, bao gồm việc tải phụ thuộc và biên dịch nếu cần.

- Phát hành (Release):
  Kết hợp bản dựng với cấu hình triển khai (env vars) để tạo ra một bản phát hành sẵn sàng vận hành. Mỗi bản phát hành có ID duy nhất và không thể chỉnh sửa sau khi tạo ra.
- Vận hành (Run):
  Khởi chạy ứng dụng từ bản phát hành trong môi trường thực thi. Việc khởi động lại tiến trình (do sự cố, khởi động lại server...) phải không ảnh hưởng đến mã nguồn hoặc cấu hình.

6. Processes

- Không lưu dữ liệu phiên hay tạm thời trong bộ nhớ hay ổ đĩa cục bộ của tiến trình.
- Mọi dữ liệu cần lưu trữ lâu dài phải được đặt trong dịch vụ hỗ trợ trạng thái như cơ sở dữ liệu hoặc Redis.
- Bộ nhớ và file hệ thống chỉ dùng làm bộ đệm tạm thời, không dựa vào chúng cho các yêu cầu trong tương lai.
- Tránh sử dụng “sticky sessions” – nên lưu dữ liệu phiên trong dịch vụ ngoài như Redis.

7. Port binding

- Không phụ thuộc vào máy chủ web tích hợp sẵn như Apache hay Tomcat.
- Ứng dụng vận hành như một máy chủ độc lập, tự xử lý giao tiếp HTTP hoặc các giao thức khác.
- Ví dụ: lập trình viên có thể truy cập app tại http://localhost:5000 trong môi trường local.
- Việc cung cấp dịch vụ thông qua cổng mạng giúp ứng dụng dễ dàng triển khai, đóng gói và mở rộng.
- Cách tiếp cận này cũng cho phép ứng dụng đóng vai trò là dịch vụ hỗ trợ cho các ứng dụng khác thông qua URL.

8. Concurrency

- Tiến trình là đơn vị mở rộng chính: Mỗi loại công việc được xử lý bởi một loại tiến trình riêng biệt (ví dụ: HTTP → tiến trình web, xử lý hàng đợi → tiến trình worker).
- Không phụ thuộc vào multi-threading bên trong (như JVM), mà sử dụng nhiều tiến trình song song để xử lý tải.
- Mô hình tiến trình phù hợp với kiến trúc mở rộng theo chiều ngang, giúp chạy ứng dụng trên nhiều máy chủ vật lý dễ dàng.
- Việc mở rộng đơn giản và đáng tin cậy, gọi là "công thức tiến trình" (process formation) – biểu thị số lượng và loại tiến trình ứng dụng sử dụng.
- Không chạy như dịch vụ nền (daemon), không ghi tệp PID – thay vào đó, tiến trình được quản lý bởi hệ thống supervisor như Upstart, systemd, hoặc công cụ như Foreman (trong dev).

9. Disposability

- Khởi động nhanh
- Ngừng hoạt động ổn định

- Xủ lý nền tốt
- Khả năng chịu lỗi

10. Dev/prod parity

- Mục tiêu: giảm sự khác biệt giữa ba môi trường (dev – test – prod) hỗ trợ triển khai liên tục ít lỗi

11. Logs

- Mục tiêu: Ứng dụng không tự quản lý log files, thay vào đó xem logs là một luồng sự kiện (event stream) và ghi ra stdout (standard output).

12. Admin processes

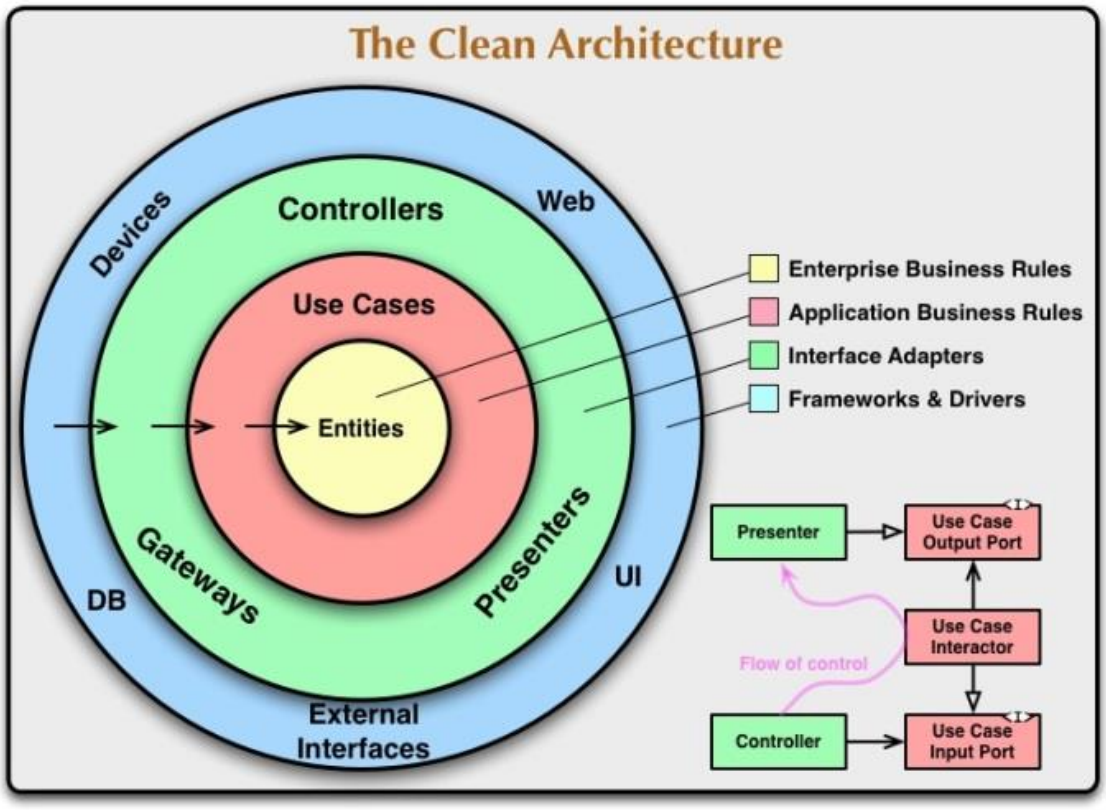
- Mụ tiêu: Thực thi các tác vụ quản trị ứng dụng (như migrate DB, fix dữ liệu, kiểm thử mô hình) như một tiến trình ngắn hạn, tách biệt, nhưng chạy trong cùng môi trường như các tiến trình chính.

Clean Architecture

1. Mục tiêu

Giúp độc lập khỏi framework, độc lập khỏi cơ chế phân phối, đơn giản hóa việc kiểm thử

2. Cấu trúc kiến trúc



Clean Architecture tổ chức hệ thống thành các vòng đồng tâm, từ lõi đến vỏ ngoài:

[Entities] ← Domain logic, data model trừu tượng nhất

[Use Cases / Business Rules] ← Điều khiển nghiệp vụ, kết nối lõi với bên ngoài

[Interface Adapters] ← Chuyển đổi dữ liệu từ ứng dụng sang các adapter như DB, UI

[Frameworks & Drivers] ← Lớp chịu trách nhiệm về framework, database, webserver, UI

3. Ưu điểm nổi bật
- Độc lập cao: không phụ thuộc vào framework hay database cụ thể
- Dễ kiểm thử và tách rời: Business logic được tách biệt, có thể mock class với bên ngoài để test nhanh, giảm phụ thuộc
- Tái sử dụng và mở rộng: Entities và use-case có thể chạy trong nhiều application hoặc giao diện khác nhau (web, mobile, CLI).

4. Hạn chế
- Phức tạp, nhiều vòng tròn trừu tưởng, có thể gây phức tạp cho các ứng dụng đơn giản
- Thiếu định hướng chi tiết
- Cồng kềnh nếu dùng sai

III. Những khó khăn gặp phải

IV. Nhận xét của người hướng dẫn (nếu có)

.....

.....