TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN CNTT2**

**XÂY DỰNG WEBSITE ĐĂNG TIN TUYỂN DỤNG CỦA DOANH NGHIỆP CHO SINH VIÊN TDTU SỬ DỤNG CÔNG NGHỆ NODEJS VÀ REACT**

*Người hướng dẫn*: **GV DOÃN XUÂN THANH**

*Người thực hiện*: **NGUYỄN HỮU TẤN ĐẠT - 51702075**

**NGÔ MINH HIẾU - 51702017**

Lớp **: 17050201**

Khoá  **: 21**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2022**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN CNTT2**

**XÂY DỰNG WEBSITE ĐĂNG TIN TUYỂN DỤNG CỦA DOANH NGHIỆP CHO SINH VIÊN TDTU SỬ DỤNG CÔNG NGHỆ NODEJS VÀ REACT**

*Người hướng dẫn*: **GV DOÃN XUÂN THANH**

*Người thực hiện*: **NGUYỄN HỮU TẤN ĐẠT - 51702075**

**NGÔ MINH HIẾU - 51702017**

Lớp **: 17050201**

Khoá  **: 21**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2022**

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành đến thầy **Doãn Xuân Thanh** – Giảng viên hướng dẫn của chúng em trong việc thực hiện dự án công nghệ thông tin 2, và khoa Công Nghệ Thông Tin đã cho em cơ hội thực hiện đề tài này và cũng như là cơ hội em được tiếp xúc với một số công nghệ được xem là trang bị cho em trong môi trường làm việc thực tế, cũng như trong việc hoàn thành đề tài xây dựng trang ứng dụng này.

Vì áp dụng 1 số kiến thức ở trường cũng như tìm hiểu và ứng dụng một số công nghệ mới nên chúng em sẽ không tránh khỏi được những sai sót và sự hạn chế của kiến thức tự tìm hiểu , đánh giá và trình bày về đề tài này. Rất mong nhận được sự quan tâm, góp ý của các thầy/cô để chung em biết mình thiếu sót những gì và giúp cho đề tài của chúng em được đầy đủ và hoành chỉnh hơn. Chúng em xin chân thành cảm ơn.

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của GV Doãn Xuân Thanh ;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 03 tháng 12 năm 2022*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Đạt*

*Nguyễn Hữu Tấn Đạt*

*Hiếu*

*Ngô Minh Hiếu*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

* Giới thiệu các công nghệ và một số thuật ngữ được sử dụng trong quá trình xây dựng trang ứng dụng
  + Docker, CI/CD, VPS, NGINX, NODEJS
* Xác định các tính năng cần có , xây dựng các bảng dữ liệu lưu trữ cần có để thực tiến hành viết API và giao diện tương ứng
* Quá trình xây dựng API và cách gọi đến API
* Quá trình xây dựng Giao diện.

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc121479656)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN ii](#_Toc121479657)

[TÓM TẮT iii](#_Toc121479658)

[MỤC LỤC 1](#_Toc121479659)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 4](#_Toc121479660)

[CHƯƠNG 1 – GIỚI THIỆU 5](#_Toc121479661)

[1. GIỚI THIỆU VỀ ĐỀ TÀI 5](#_Toc121479662)

[2. GIỚI THIỆU VỀ CÔNG NGHỆ VÀ THUẬT NGỮ 5](#_Toc121479663)

[1) Docker 5](#_Toc121479664)

[2) CI/CD 11](#_Toc121479665)

[3) VPS 18](#_Toc121479666)

[4) NODEJS(EXPRESS) 22](#_Toc121479667)

[5) ORM (PRISMA) 23](#_Toc121479668)

[6) ReactJS 30](#_Toc121479669)

[a. JSX 30](#_Toc121479670)

[b. Tại sao lại nên dùng JSX 32](#_Toc121479671)

[c. Components 33](#_Toc121479672)

[d. Lifecycle 34](#_Toc121479673)

[e. Props và State 35](#_Toc121479674)

[f. Hook 36](#_Toc121479675)

[7) Redux 44](#_Toc121479676)

[a. Action 46](#_Toc121479677)

[b. Reducers 47](#_Toc121479678)

[c. Store 49](#_Toc121479679)

[g. Sử dụng Redux 49](#_Toc121479680)

[h. Redux-toolkit 49](#_Toc121479681)

[8) Typecript 51](#_Toc121479682)

[**a.** Ưu điểm của Typescript: 51](#_Toc121479683)

[b. Basic Types: 52](#_Toc121479684)

[c. Function 54](#_Toc121479685)

[d. Type aliases 55](#_Toc121479686)

[e. Interface 56](#_Toc121479687)

[f. Types vs Interfaces 57](#_Toc121479688)

[g. Intersection 59](#_Toc121479689)

[h. Generics 60](#_Toc121479690)

[9) Antd 66](#_Toc121479691)

[a. Antd là gì 66](#_Toc121479692)

[b. Cách sử dụng 67](#_Toc121479693)

[3. Nội dung của chương này 68](#_Toc121479694)

[CHƯƠNG 2 – QUÁ TRÌNH THỰC HIỆN 69](#_Toc121479695)

[1 Quá trình thiết lập server 69](#_Toc121479696)

**DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT**

**CÁC CHỮ VIẾT TẮT**

CI/CD Chỉ quá trình tích hợp liên tục và phân phối liên tục

JS Là một ngôn ngữ lập trình JavaScript

TS Là một ngôn ngữ lập trình được phát triển bởi Microsoft và có cách viết tương tự như JavaScript nhưng chặt chẽ hơn và dễ phát hiện lỗi sai trong khi viết

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC HÌNH**

[Hình 2.1: Kiến trúc FTP 1](#_Toc387689394)

**DANH MỤC BẢNG**

[Bảng 3.1 Ví dụ cho chèn bảng 1](#_Toc387689363)

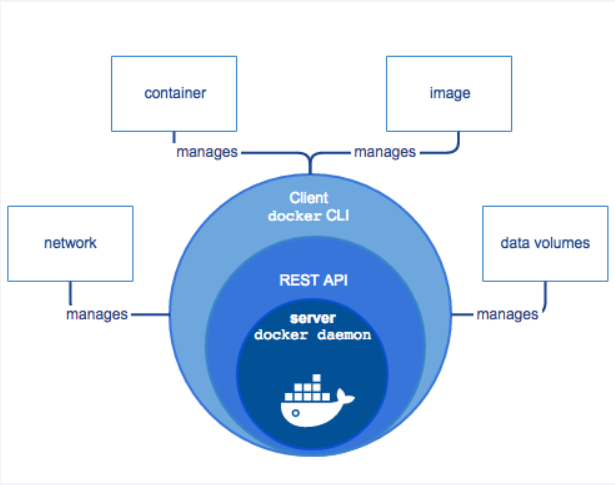
CHƯƠNG 1 – GIỚI THIỆU

1. GIỚI THIỆU VỀ ĐỀ TÀI

* Với nhu cầu tìm việc của nguồn lực trường, Trường đại học Tôn Đức Thắng muốn tăng cường cơ hội tìm việc cũng như có thêm một kênh thông tin về cơ hội việc làm cùng với các kênh khác như topcv, glint, hoặc facebook. Nhầm để đáp ứng với nhu cầu đó thì chung em sẽ xem và xây dựng một kênh tìm việc cũng như đăng các tin tuyển dụng của các nhà tuyển dụng lên để tạo cơ hội việc làm cho sinh viên của trường
* Kênh tìm kiếm việc này sẽ cho phép sinh viên tạo hồ sơ ứng tuyển của riêng mình cũng như dễ dàng tìm được công việc phù hợp dựa vào nguồn việc làm từ các đối tác lớn của nhà trường cũng như các công ty nhỏ và cần nguồn nhân lực chất lượng cao

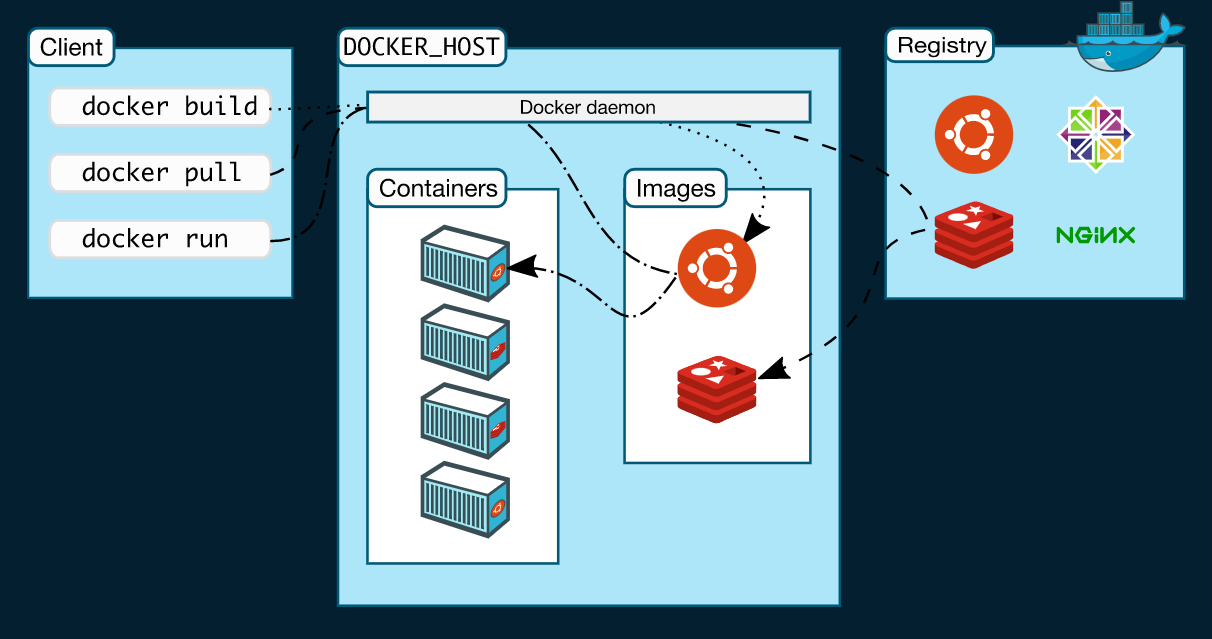
1. GIỚI THIỆU VỀ CÔNG NGHỆ VÀ THUẬT NGỮ
2. Docker

* Để một source code được khởi động và hoạt động trơn tru ở một máy tính đòi hỏi máy tính đó phải có đầy đủ tài nguyên như môi trường , các biến , một số thư viện , cũng như một số công cụ để hỗ trợ cho ngôn ngữ và công nghệ đó hoạt động. Với Docker bạn có thể quản lý các tài nguyên của source code một cách dễ dàng và giảm độ trễ giữ việc viết code với việc chạy chúng trong sản phẩm của bản thân.
  + VD: Nếu một máy tính đang chạy một phiên bản môi trường ví dụ như NodeJs ở phiên bản 16 nhưng ở một máy của người đồng nghiệp lại chạy NodeJs ở phiên bản cao hơn hoặc thấp hơn điều này sẽ dẫn đến việc là khi source code chạy ở máy mình hoạt động bình thường nhưng khi sang máy của bạn hoặc đồng nghiệp lại không được vì không hỗ trợ cách viết JS đó hoặc thiếu gì đó và dẫn đến mấy thời gian rất nhiều
* Chính vì thế thì Docker( Container ) được sinh ra để giải quyết những vấn đề này
  + Docker là một công cụ để cung cấp cho việc Build , Deploy, và Run source code ứng dụng hay là một trang web nào đó một cách dễ dàng. Và docker ban đầu được viết bằng ngôn ngữ Python và sau này là GoLang
  + Để thực hiện các việc như Build , Deploy , Run như thế thì Docker áp dụng 1 thuật ngữ là Containers . Containers cho phép người lập trình viên đóng gói source code với tất cả các điều kiện mà source code cần để chạy được như là thư viện , môi trường , hay là công cụ hỗ trợ.
  + Dựa vào Container sau khi ta khai báo và đóng gói các package cần thiết thì ta có thể đưa nó vào máy chủ chạy bất kì hệ điều hành nào cũng được và phổ biến nhất ở đấy là Linux, Ubuntu , Centos,…. Và có thể là Windows của Microsoft tùy thuộc vào máy chủ mà người tập trình viên đó thao tác với nó
  + Sự độc lập và bảo mật cho phép bạn chạy riêng biệt nhiều Container cùng một máy chủ. Các container hoạt động chiếm một lượng dung lượng của máy chủ tùy theo độ lớn nhỏ của dự án mà ta đưa vào nó.
  + Docker cung cấp các công cụ và nền tảng để quản lý quá trình hoạt động và vòng đời container của bạn.
    - Phát triển ứng dụng của bạn và các thành phần hỗ trợ trong contrainers
    - Container cũng đóng vai trò là đơn vị cung cấp và thử nghiệm ứng dụng hay trang web của bạn
    - Nếu bạn có nhu cầu đưa ứng dụng hay trạng web của mình lên môi trường sản phẩm và đưa ra cho mọi người sử dụng dưới dạng là 1 container hoặc là một dịch vụ nào đó cho dù môi trường đó là nằm ở máy của bạn hay là nằm trên một máy chủ khác ở điện toán đám mây hoặc có thể là sử dụng cả hai.
* Docker Engine là gì ?
  + Một tiến trình Docker daemon có nhiệm vụ các bước ban đầu để tạo và quản lý các container, image , volume , network
  + Docker Client sẽ giao tiếp với Docker daemon thông qua API theo chuẩn REST ( tức có nghĩa sẽ có các phương thức GET , POST , PUT , DELETE ) để yêu cầu Docker daemon thực thi các lệnh mà người dùng gọi đến
  + Docker CLI được xem là Client hay được hiểu là giao diện tương tác giữa người dùng ở đây là sử dụng các công cụ hỗ trợ như là Command Line (CMD), Window Terminal, …. Để gửi các yêu cầu tương ứng đến docker daemon thông qua API



Hình 1: Sơ đồ về kiến trúc của docker

* Kiến trúc của Docker
  + Docker có kiến trúc là Client – Server và giao tiếp qua REST API (Cổng mạng và một số phương thức khác) để có thể gửi các yêu cầu đến Docker daemon. Việc giao tiếp này thực hiện các công việc như Run , Deploy , Build các docker container. Ta có thể cấu hình cho Docker Client – Docker Daemon trên cùng 1 máy , hoặc có thể dùng Docker Client để giao tiếp từ xa với Docker Daemon. Và minh họa cho kiến trúc của docker như sau

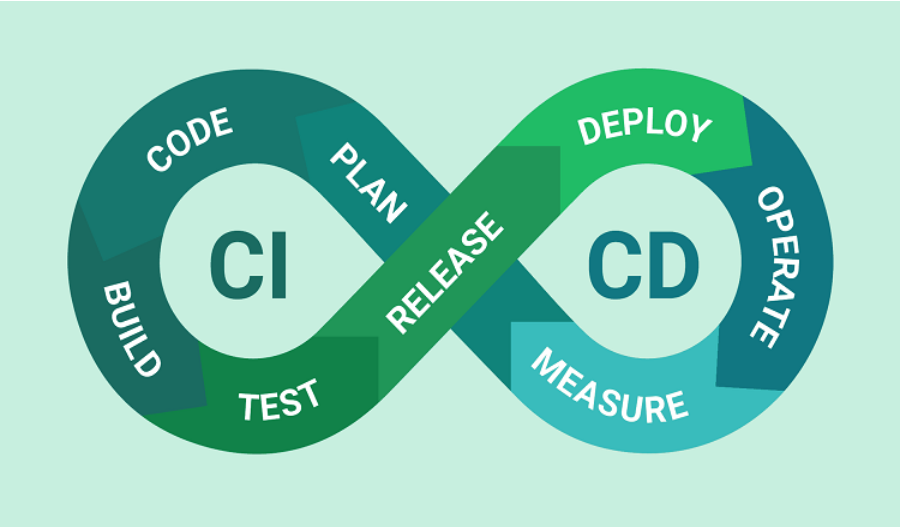


Hình 2 : Chi tiết và cách hoạt động và thao tác với docker

Một số thành phần khác của docker

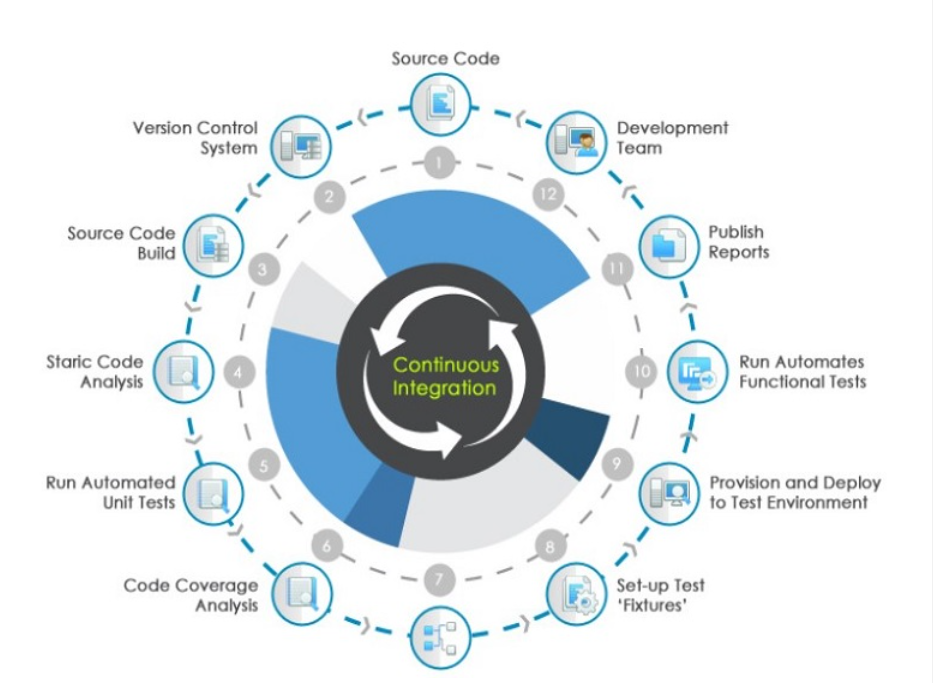
* Docker daemon
  + Daemon Docker (dockerd) lắng nghe các yêu cầu của người dùng thông qua Docker API và quản lý các Docker đối tượng như image, container, network và volume. Một daemon cũng có thể giao tiếp với các daemon khác để quản lý dịch vụ Docker.
* Docker Client
  + Docker client (docker) là cách tốt nhất để người thiết lập docker giao tiếp với no. Khi bạn sử dụng các câu lệnh như docker run, client thông qua Docker API sẽ gửi các câu lệnh này đến dockerd, thực hiện chúng. Docker client có thể giao tiếp với nhiều hơn một daemon Docker.
* Docker Registry
  + Docker registry sẽ là nơi lưu trữ các hình ảnh Docker. Docker Hub là nơi lưu trữ Docker image public (public registry) mà bất kỳ ai cũng có thể sử dụng và Docker được định cấu hình mặc định để tìm kiếm hình ảnh trên Docker Hub. Ngoài ra, người dùng có thể cấu hình các tư vấn đăng ký khác nhau để lưu trữ hình ảnh Docker.
* Khi người dùng sử dụng câu lệnh “docker pull” hoặc “docker run”, các image only sẽ được tải về dựa trên registry đã được cấu hình trước đó. Khi người dùng sử dụng câu lệnh “docker push”, image cũng sẽ được tải lên registry mà người dùng đã cấu hình từ trước đó.
* Docker image
  + Docker Image là template read-only (chỉ cho phép đọc) với các hướng dẫn để tạo Docker container. Image sẽ được sử dụng để đóng gói các ứng dụng và các thành phần đi kèm của ứng dụng, được lưu trữ ở server hoặc trên registry. Ví dụ bạn có thể sử dụng Dockerfile để tạo ra một Docker image sử dụng hệ điều hành Ubuntu và cài đặt Apache server với những cài đặt, cấu hình tùy chỉnh của riêng mình.
* Docker container
  + Container là 1 “runable instance” của image. Bạn có thể khởi tạo, dừng hoặc xóa container bằng cách sử dụng Docker API hoặc CLI. Bạn có thể kết nối container đến 1 hoặc nhiều network, thư mục lưu trữ, hoặc thậm chí tạo ra 1 image mới dựa trên tình trạng hiện tại của container. Mặc định 1 container được “cách ly” với các container và server nếu người dùng không có các cài đặt gì thêm.
* Docker volume
  + Volume được thiết kể để làm nơi lưu trữ các dữ liệu độc lập với vòng đời của container.
* Docker network
  + Cung cấp một private network mà chỉ tồn tại giữa container và server, giúp các container có thể giao tiếp được với nhau một cách dễ dàng.
* Docker service
  + Docker Service cho phép bạn mở rộng các contaner thông qua nhiều Docker daemon, chúng giao tiếp với nhau thông qua swarm cluster bao gồm nhiều manager và worker. Mỗi một node của swarm là 1 Docker daemon giao tiếp với nhau bằng cách sử dụng Docker API. Theo mặc định thì service được cân bằng tải trên các node.

1. CI/CD



Hình 3 : Quá trình hoạt động của CI/CD

* CI ( Continous Integration ) là cách thức phát triển phần mềm bằng cách tích hợp liên tục có nghĩa là nếu trong một nhóm những người cùng một dự án khi thực hiện một tính năng hay tham gia vào việc bảo trì một tính năng nào đó và sinh ra các đoạn mã mới thay cho đoạn mã cũ và đưa đoạn mã trên nhánh của người đó lên nhánh tổng ( master ) hoặc là nhánh develop thì ở đây CI sẽ thực hiện công việc của nó là tự động chạy toàn bộ source code cùng với đoạn mã của người viết đoạn mã vừa mới đưa lên và kiểm tra và check xem độ chính xác về mặt quy tắc viết mã của ngôn ngữ mà source đó sử dụng hay không. Nếu không thì sẽ tiếp tục bước tiếp theo nhưng khi có vẫn đề xảy ra thì sẽ báo lỗi cho người viết đoạn mã đó chưa khớp với đoạn mã trên nơi tổng mà các nhánh của các thành viên đổ về đó, và cũng như đưa ra những gợi ý cho người viết mã là một đề xuất để sửa. CI cũng đảm bảo source code của bạn sau gom mã của thành viên trong nhóm lại sẽ có thể hoạt động ở bất kì điều kiện nào và không xảy ra lỗi. Quá trình Build Run và Deploy sẽ được chạy và cho người quản lý source code tiến trình nó xảy ra như thế nào.
* Quá trình đưa mã code từ nhánh cá nhân lên nhánh chính ( develop hoặc master) thì quá trình CI hoạt động sẽ được hoạt động tự động nếu người quản lý source code cấu hình và cài đặt trước . Nến quá trình gom source code -> kiểm tra -> phản hồi sẽ diễn ra liền mạch và tự động. Người xây dựng mã người có thể tiết kiệm thời gian để xây dựng tính năng khác hoặc sửa lỗi nếu có và sau khi quá trình kết thúc thì người quản lý có thể xem được quá trình build và test.



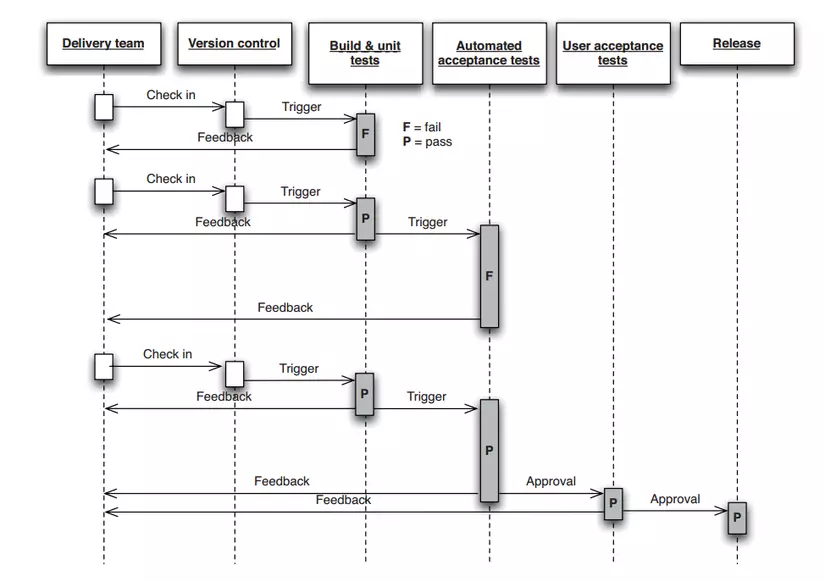
Hình 4 : Quá trình phát triển ứng dụng.

Lợi ích của việc sử dụng CI

* Giảm thiểu rủi ro nhờ việc phát hiện lỗi và fix sớm, tăng chất lượng phần mềm nhờ việc tự động test và inspect (đây cũng là một trong những lợi ích của CI, code được inspect tự động dựa theo config đã cài đặt, đảm bảo coding style, chẳng hạn một function chỉ được dài không quá 10 dòng code ...)
* Giảm thiểu những quy trình thủ công lặp đi lặp lại (build css, js, migrate, test...), thay vì đó là build tự động, chạy test tự động
* Sinh ra phần mềm có thể deploy ở bất kì thời gian, địa điểm

CD là gì : Thì đi kèm với CI là việc tích hợp liên tục thì còn một quá trình nữa gọi là chuyển giao liên tục thì nếu source code qua quá trình CI không có lỗi xảy ra thì sẽ sang một giai đoạn gọi là CD để tiến hành kiểm tra những thay đổi về mã nguồn đã được build lên và mã nguồn trong môi trường kiểm thử trước đó. CD cho phép người viết mã nguồn cài đặt những chu kì tự động ngắn và liên tục . Từ đó nhanh chống phát hiện ra lỗi sai và tiến hành chỉnh sửa ngay trong những phần tương tự.

* Do sự tiện lợi của 2 quá trình này thì người quản lý mã nguồn thường tích hợp cả hai CI/CD với nhau để tiết kiệm thời gian , chi phí và tài nguyên bộ nhớ cho 2 quá trình này, CD sẽ sử dụng giai đoạn cuối cùng của CI để bắt đầu quá trình chuyển giao của chính nó
* Trong quá trình CD thực hiện tiến trình của mình thì có thể mã nguồn sẽ trải qua một số phần testing được người quản lý mã nguồn thiết lập và một số bài test bao gồm UI Testing , API Testing, …. CD sử dụng một thuật ngữ được gọi là Deployment Pipeline giúp chia quy trình chuyển giao thành các giao đoạn. Mỗi giai đoạn sẽ đáp ứng những mục đích riêng bao gồm xác minh chất lượng của các tính năng từ một số góc độ khác nhau để kiểm định chức năng và tránh lỗi ảnh hưởng đến người dùng. Pipeline sẽ thể hiện phản hồi cho nhóm trong việc cung cấp tính năng mới hoặc sửa chữa các tính năng đã có sẵn. Deployment Pipelien là quy trình để chuyển phần mềm từ version control đến tay người dùng.
* Việc kết hợp 2 quá trình này thì ta sẽ được chu trình hoạt động của CICD gói gọn trong 3 bước là Xây dựng -> Kiểm tra -> Triển khai



Hình 5 : Cách hoạt động của CI/CD

Lợi ích của CI/CD

* **Giảm thiểu rủi ro không đáng có**

Đây có thể là một lợi ích vô cùng hữu ích của CI/CD, nó cho phép làm giảm thiểu đi những rủi ro nhờ việc phát hiện và sửa lỗi sớm, giúp tăng chất lượng sản phẩm nhờ khả năng tự động kiểm tra và quan sát. Không những vậy, những quy trình thủ công lặp đi lặp lại hằng ngày cũng được giảm tải, thay vào đó là xây dựng và kiểm thử tự động mà không cần đến sự giúp đỡ của con người. Một đặc điểm nữa của CI CD chính là có thể deploy, triển khai phần mềm ở bất cứ địa điểm và thời gian nào

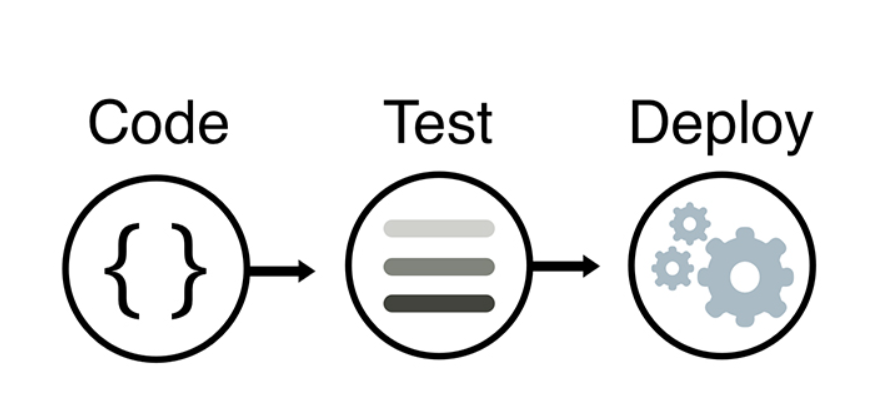
* **Thay đổi code nhỏ**

Một lợi ích vô cùng lớn của CI/CD chính là cho phép chúng ta tích hợp nhiều loại mã nhỏ cùng một lúc. Những thay đổi mã này được thực hiện một cách đơn giản và xử lý nhanh hơn so với những đoạn mã khổng lồ, từ đó làm giảm đi khả năng sinh ra những vấn đề liên quan đến việc thay đổi sau này.

Những sự thay đổi mã nhỏ này có thể được thực kiểm tra ngay sau khi chúng được tích hợp vào kho mã. Các nhà phát triển có thể dễ dàng nhận ra vấn đề trước khi lượng lớn công việc tăng lên một cách chóng mặt. Đây thực sự là một lợi thế đối với những nhóm phát triển lớn hoặc người làm việc từ xa giao tiếp được hiệu quả hơn.

* **Hạn chế những ảnh hưởng của lỗi**

CI/CD được thiết kế với hệ thống sao cho khi có lỗi nào đó xảy ra thì những kết quả tiêu cực sẽ bị giới hạn trong phạm vi ảnh hưởng nhất định nào đó. Việc hạn chế các vấn đề này giúp làm giảm khả năng hư hỏng từ đó làm cho hệ thống được bảo trì và xử lý một cách dễ dàng hơn.Với hệ thống CI CD, có thể đảm bảo cho việc cách ly lỗi sẽ được phát hiện một cách nhanh chóng và dễ dàng thực hiện hơn. Chính vì vậy, hậu quả của các lỗi trong ứng dụng sẽ được giới hạn trong phạm vi ảnh hưởng của nó.



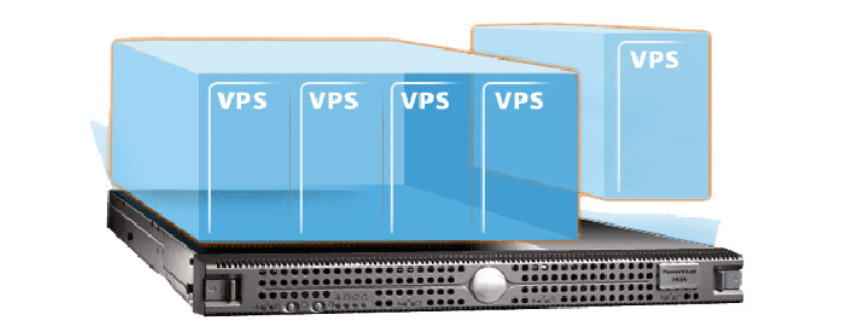
Hình 6 : Tổng quan về quá trình hoạt động với CI/CD

Nhược điểm của CI/CD là gì ?

* Bên cạnh các ưu điểm thì hệ thống CI và CD vẫn tồn tại một số điểm trừ nhất định. Vì CI/CD thường xuyên được nâng cấp để tối ưu hơn nên đòi hỏi các nhà lập trình phải nhanh chóng học hỏi các kiến thức mới.
* Ngoài ra, để quá trình tự động hóa diễn ra trơn tu, người sử dụng phải hiểu rõ cách vận hành của chương trình. Và điều này hoàn toàn không đơn giản bởi CI và CD sở hữu bộ kỹ năng khá phức tạp. Để hoàn toàn sử dụng trơn tru được CI/CD đòi hỏi quá trình rèn luyện lâu dài.
* Để có thể ứng dụng được CI và CD vào công việc đòi hỏi nhà sản xuất phải đầu tư trước một khoản phí khá lớn. Không nhiều công ty dám mạo hiểm bởi như đã nói ở trên, việc sử dụng CI và CD không dễ dàng. Và nhược điểm cuối cùng là một số hệ thống lập trình không cho phép hỗ trợ thêm CI và CD. Nếu không thể thay đổi hệ thống thì khó lòng sử dụng được chương trình này.

1. VPS

* VPS là viết tắt của ( Virtual Private Server) hay còn gọi là máy chủ riêng ảo. Được cung cấp và cho thuê bởi các công ty dịch vụ Internet Hosting được tạo ra bới công nghệ ảo hóa.
* VPS hoạt động như một server để người dùng tạo ra trang web hay ứng dụng trên các tài nguyên ta cài đặt được giành riêng
* Người cài đặt VPS có toàn quyền kiểm soát bằng superuser ( Root , quyền admin ), và thông qua SSH key để điều khiển cài đặt các thư viện, hoặc cấu hình VPS theo ý muốn của người cài đặt.
* Thường thì VPS sẽ được thuê và có những option các hệ điều hành có sẵn khi người muốn thuê lựa chọn các dịch vụ và trả tiền để thành toán phí duy trì của cái máy ảo đó ở máy chủ của công ty hosting đó một số hosting phổ biến như AWS , Digital Ocean , Microsoft Azure , Ở Việt Nam thì có Mắt Bão , ….. trên điện toán đám mây
* VPS an toàn và ổn định tùy theo tình trạng mạng của bạn kết nối và ổn định hơn so với host website vì bạn không chia sẻ không gian lưu trữ với người khác. Và nó cũng có giá thành thấp hơn so với thuê một server riêng



* Thường thì các máy chủ ảo sẽ không có tên miền để thay thế cho việc truy cập đến máy chủ đó mà phải truy cập qua IP của VPS đó được sinh ra khi bạn tiến hành thuê VPS đó
* VPS là một server ảo hoạt động tương tự như một máy chủ vật lý. Máy chủ , hay server, là một máy tinh chứa source code và cơ sở dữ liệu cho ứng dụng của bạn , Môi trường và các thư viện cần thiết. Bất kể khi nào người dùng truy cập vào ứng dụng của bạn trên trình duyệt thì sẽ gửi yêu cầu đến con server của bạn và lúc này dựa vào đường truyền internet các file cần thiết để chạy 1 trang web sẽ được truyền xuống máy tính của người dùng và hiện thị nó trên tình duyệt
* Trên thực tế thì VPS là máy chủ ảo được cấu hình qua mạng như nó sẽ được hình thành trên một server vật lý, và mỗi VPS là sẽ có phần tài nguyên riêng biệt của nó và không bị ảnh hưởng bởi các VPS khác
* Các yếu tố lưu ý đến VPS khi tiến hành thuê
  + **Ram** : Trong máy tính, RAM là loại bộ nhớ chính, nếu VPS của bạn có nhiều RAM thì khả năng truy xuất dữ liệu càng tốt. Bởi vì khi sử dụng VPS bạn sẽ cần RAM để xử lý các vấn đề như xử lý truy vấn nhập xuất của database
  + **SWAP** hay là bộ nhớ ảo có nhiệm vụ lưu lại các hành động trong quá khứ trong trường hợp bộ nhớ RAM bị quá tải (overload). Hay nói cách dễ hiểu hơn, SWAP là nơi lưu trữ trên ổ cứng thay vì một bộ nhớ độc lập. Tuy nhiên, không phải VPS nào cũng hỗ trợ SWAP
  + **Disk** : Ổ cứng/Ổ đĩa cứng hay còn gọi là Disk là không gian lưu trữ được sử dụng để lưu các file cài đặt của hệ điều hành và các file của mã nguồn website. Có 2 loại ổ đĩa được sử dụng phổ biến, đó là:
    - HDD (Hard Disk Drive): Là loại ổ đĩa thông dụng được sử dụng trên máy tính.
    - SSD (Solid State Drive): SSD hay còn gọi là ổ cứng bán dẫn được sử dụng để lưu trữ dữ liệu nhưng tốc độ truy xuất dữ liệu nhanh hơn so với HDD 300 lần.
    - Do vậy, giá thành của ổ cứng SSD sẽ có đắt hơn so với loại ổ HDD.
  + **CPU Core :** là lõi xử lý của CPU. Một máy chủ riêng sẽ có lượng core nhất định và nó được chia cho các VPS. Thông thường, số core càng cao thì khả năng xử lý dữ liệu càng tốt. Số lượng core trung bình ở các gói VPS là từ 1 core đến 3 core.
  + **Bandwidth và transfer :**  đều có nghĩa là băng thông, chính là lưu lượng được phép truyền tải dữ liệu đi.
  + **IP (Internet Protocol) :** là số lượng địa chỉ IP mà nhà cung cấp dịch vụ VPS sẽ cấp cho bạn.
* Thì các công ty cho thuê dịch vụ máy chủ ảo riêng biệt như thế sẽ cho người thuê toàn quyền quyết định về hệ điều hành, cũng như cho phép chọn cấu hình của phân vùng VPS đó như sử dụng bao nhiều RAM xử lý , Bao nhiêu GB bộ nhớ , Thì Ram và bộ nhớ càng cao ứng dụng của bạn sẽ hoạt động một cách trơn tru hơn nhưng bù lại chi phí của của người thuê bỏ ra để duy trì nó sẽ càng cao hơn.



Ưu điểm của VPS là gì ?

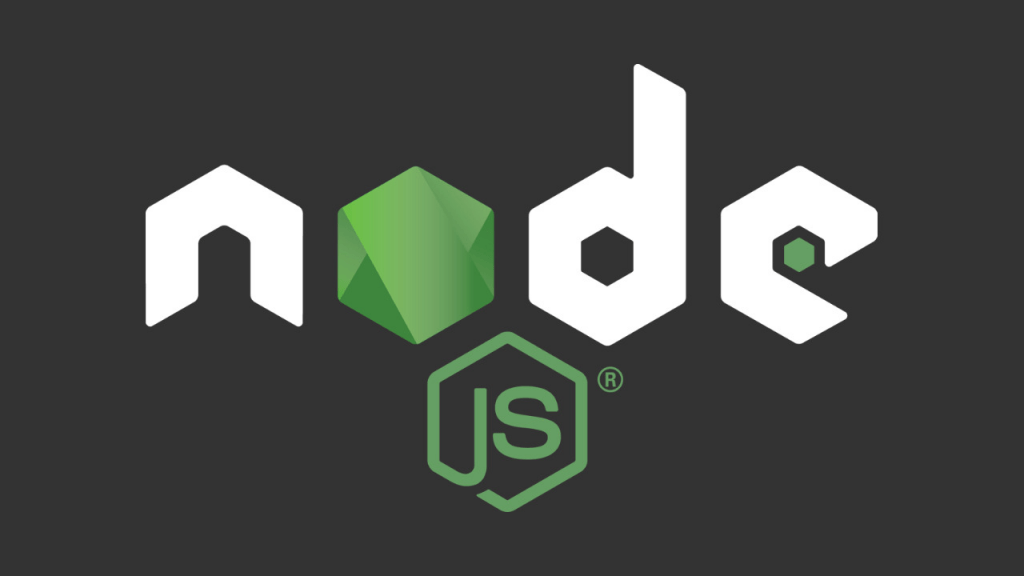
* Nhanh và đáng tin cậy hơn server shared hosting.
* Vì được đảm bảo về thông số server như bộ nhớ và sức mạnh vi xử lý, bạn sẽ không gặp phải vấn đề tài nguyên bị người khác dùng hết.
* Các vấn đề về lượng truy cập đột biến tăng cao không ảnh hưởng đến site của bạn.
* Bạn có quyền superuser (root) trên server.
* Có độ riêng tư cao hơn, vì files và databases bị khóa khỏi hệ thống server của các người dùng khác.
* Dễ dàng nâng cấp. Ngay khi website tăng trưởng, bạn chỉ cần nâng cấp gói hosting để nâng tài nguyên lên mà không phải tốn công chuyển dữ liệu hay chuyển server (RAM, CPU, disk space, bandwidth,…).
* IP riêng. Chính vì điều này khả năng chống DDoS cũng cao hơn.

Nhược điểm

* Giá cao hơn shared hosting.
* Cần ít nhiều kiến thức kỹ thuật để quản lý.
* Cấu hình server không đúng có thể tạo ra lỗ hổng bảo mật.

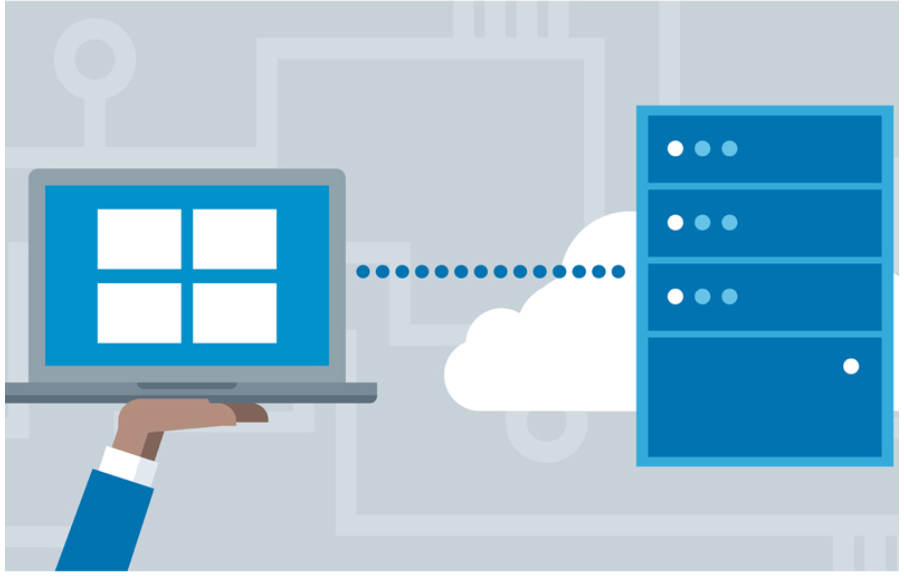
1. NODEJS(EXPRESS)

* NodeJS được xem là nền tảng để viết và chạy các mã nguồn thuộc về ngôn ngữ JavaScript, Nó cho biến ngôn ngữ JavaScript từ một ngôn ngữ chỉ hoạt động trên các client web thành một thứ mà các lập trình viên có thể sử dụng nó để tạo ra các ứng dụng độc lập trong các lĩnh vực như
  + Ứng dụng trò chuyện trong thời gian thực: Nhờ vào cấu trúc không đồng bộ đơn luồng, Node.JS rất thích hợp cho mục đích xử lý giao tiếp trong thời gian thực. Nền tảng này có thể dễ dàng mở rộng quy mô và thường dùng để tạo ra các chatbot.
  + Internet of Things (IoT): Các ứng dụng IoT thường bao gồm nhiều bộ cảm biến phức tạp để gửi những phần dữ liệu nhỏ.
  + Truyền dữ liệu: Netflix là một trong số những công ty lớn trên thế giới chuyên sử dụng Node.JS cho mục đích truyền dữ liệu.
  + Các SPA (Single-page application) phức tạp: Trong SPA, toàn bộ ứng dụng được load vào trong một trang duy nhất, do đó sẽ có một số request được thực hiện trong nền.
  + Các ứng dụng REST dựa trên API: JavaScript được sử dụng trong cả frontend lẫn backend của trang. Do đó một server có thể dễ dàng giao tiếp với frontend qua REST API bằng Node.js
* : là một Framework nhỏ, nhưng linh hoạt được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile
  + Về các package hỗ trợ: Expressjs có vô số các package hỗ trợ nên các bạn không phải lo lắng khi làm việc với Framework này.
  + Về performance: Express cung cấp thêm về các tính năng (feature) để dev lập trình tốt hơn. Chứ không làm giảm tốc độ của NodeJS.



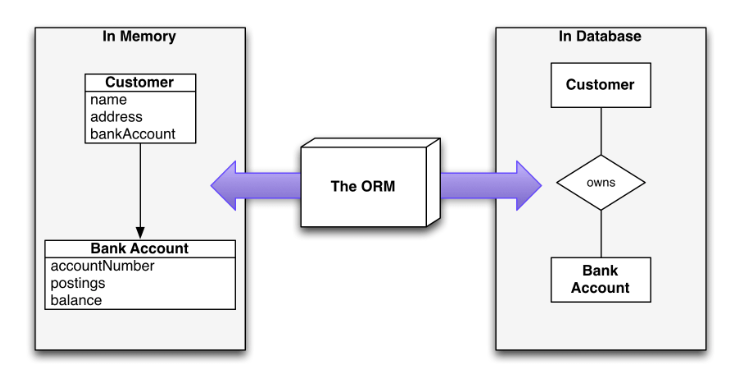
1. ORM (PRISMA)

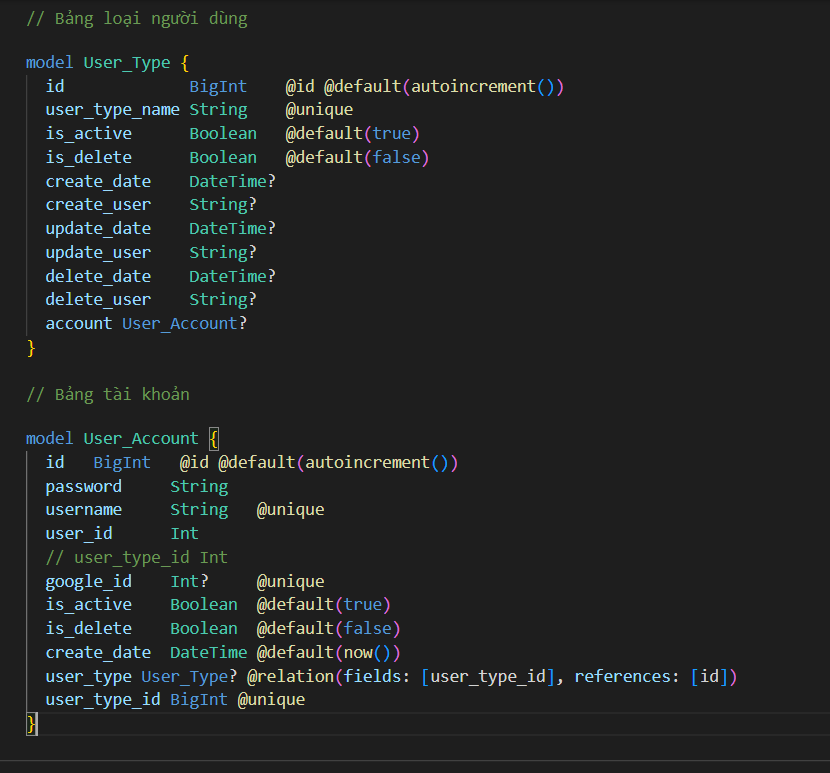
* SQL server hay còn được gọi là Microsoft SQL Server, nó từ viết tắt của MS SQL Server. Đây chính là một loại phần mềm đã được phát triển bởi Microsoft và nó được sử dụng để có thể dễ dàng lưu trữ cho những dữ liệu dựa theo tiêu chuẩn RDBMS.
* SQL Server có khả năng cung cấp đầy đủ các công cụ cho việc quản lý từ giao diện GUI đến sử dụng ngôn ngữ cho việc truy vấn SQL. Điểm mạnh của SQL điểm mạnh của nó là có thể kết hợp và hoạt động với các nền tảng khác nhau
* Và để người viết code giao tiếp với SQL server sẽ được giao tiếp với nhau thông qua một ngôn ngữ gọi là SQL , T-SQL
  + T-SQL là một trong những loại ngôn ngữ thuộc quyền sở hữu của Microsoft và được gọi với cái tên Transact-SQL. Nó thường cung cấp thêm rất nhiều cho các khả năng khai báo biến, thủ tục lưu trữ và xử lý ngoại lệ,...
  + SQL Server Management Studio là một loại công cụ giao diện chính cho máy chủ cơ sở của chính dữ liệu SQL, thông thường thì nó hỗ trợ cho cả môi trường 64 bit và 32 bit.



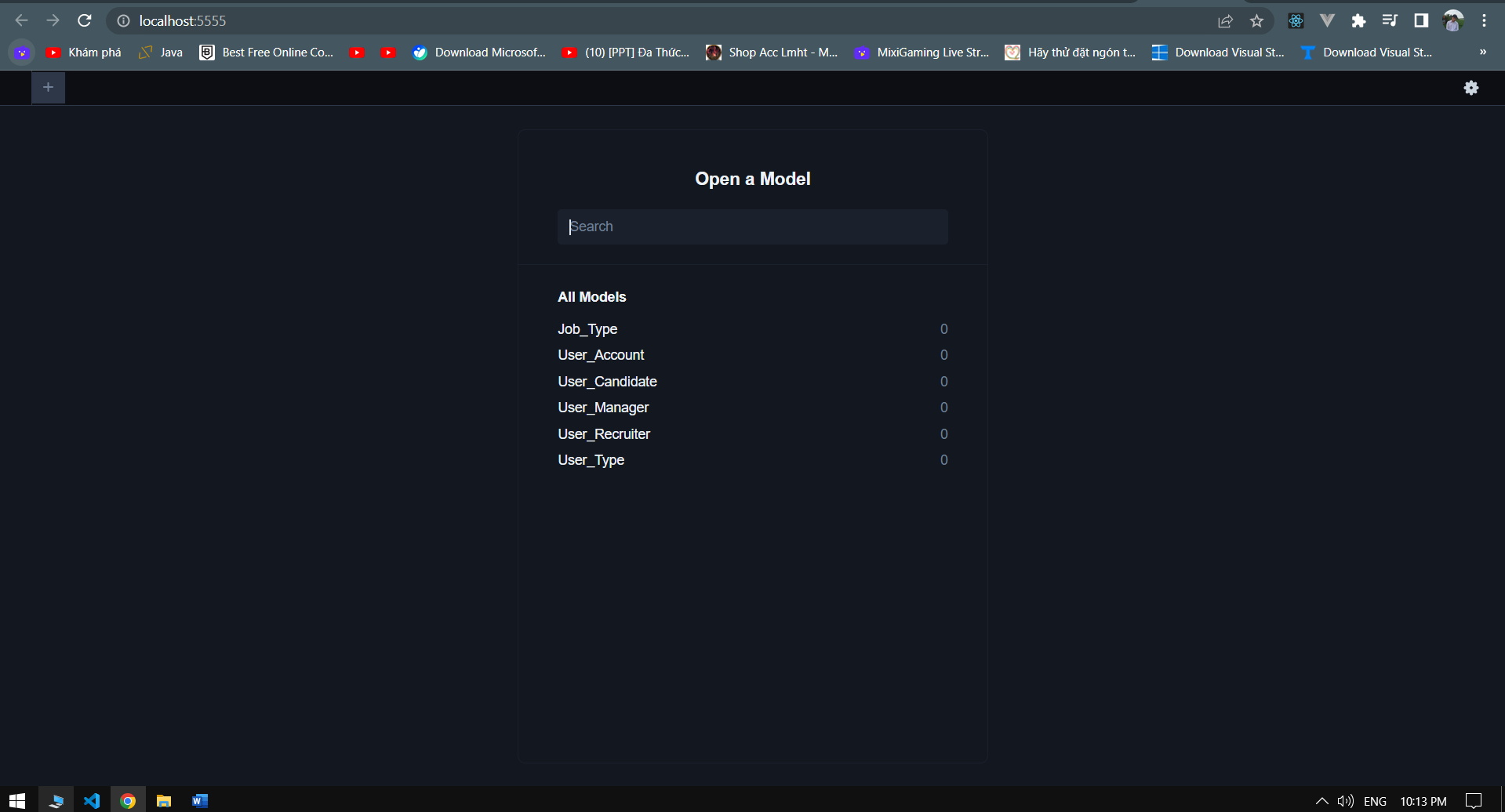
* Thì hiện nay trong thực tế có một số cách thực hiện tạo Database mà không cần thông qua GUI của SQL server mà dùng các đối tượng được định nghĩa và nhờ vào thư viện để ánh xạ các đối tượng đó vào trong cơ sở dữ liệu để hình thành các bảng và mối quan hệ của chúng thì kĩ thuật đó được gọi là ORM
* Vậy ORM là từ viết tắt của (Object Relational Mapping), là một kỹ thuật/cơ chế lập trình thực hiện ánh xạ CSDL sang các đối tượng trong các ngôn ngữ lập trình hướng đối tượng như Java, C# …(các table tương ứng các class, mối ràng buộc giữa các table tương ứng quan hệ giữa các class ‘has a’ , ‘is a’).
* Việc sử dụng ORM cho phép lập trình viên thao tác với database 1 cách hoàn toàn tự nhiên, dễ hiểu thông qua các đối tượng. Lập trình viên không cần tới loại database, kiểu dữ liệu trong database…
* Ưu điểm của ORM:
  + ORM giúp lập trình viên tập trung vào lập trình hướng đối tượng
  + Tính độc lập: Làm việc được với nhiều loại database(hệ quản trị cơ sở dữ liệu), nhiều kiểu dữ liệu khác nhau. Dễ dàng thay đổi loại database hơn. Các câu lệnh SQL không phụ thuộc vào loại database.
  + Đơn giản, dễ sử dụng: Hỗ trợ HSQL, cung cấp nhiều nhiều API truy vấn.
  + Năng suất hơn: viết code ít hơn, dễ hiểu hơn. Phù hợp các case CRUD (Create, Read, Update, Delete)
  + Khả năng sử dụng lại code.
* Nhược điểm của ORM:
  + Khả năng truy vấn bị hạn chế, nhiều trường hợp ta vẫn phải dùng native SQL để truy vấn database.
  + Khó tối ưu câu lệnh SQL (do câu lệnh SQL được ORM tự động sinh ra).
* Một thư viện hỗ trợ việc ánh xạ dữ liệu từ đối tượng trong khi xây dựng back-end bằng NodeJS đó là thư viện Prisma . Thư viện giúp ánh xạ các đối tượng ta khai báo từ dạng model được quy định thành các bảng trong cở sở dữ liệu mà ta muốn thì ở đây chúng em chọn SQL Server
* Prisma là một Object Relational Mapping (ORMs) được dùng để xây dựng các máy chủ như GraphQL Server, SQL Server, RESTful APIs, microservice, ... Prisma đơn giản là 1 layer nằm giữa Webserver và Database. Prisma giúp chúng ta giao tiếp với db một cách dễ dàng hơn.
* Prisma bao gồm ba phần chính:
  + Prisma Client : Trình tạo truy vấn an toàn và được tạo tự động cho Node.js và TypeScript.
  + Prisma Migrate : Một hệ thống di chuyển và mô hình hóa dữ liệu khai báo.
  + Prisma Studio : Một GUI để xem và chỉnh sửa dữ liệu trong cơ sở dữ liệu của bạn.
  + Cách truyền thống mà Webserver giao tiếp với Database là thông qua các câu lệnh SQL Query như SELECT, UPDATE hay DELETE. Giờ đây, nhờ vào các công cụ ORMs nói chung và Prisma nói riêng. Chúng tạo ra một tầng abstraction giữa Webserver và Database. Điều này giúp cho lập trình viên dễ dàng trong việc thao tác với Database. Thay vì viết những câu lệnh SQL khô khan, có thể sai bất cứ lúc nào thì chúng ta có thể viết các hàm tương ứng.







* Thì trên đây là 2 bảng được khởi tạo trong database được khởi tạo thành 2 bảng là User\_type và User\_Account thì các giá trị bắt buộc không được bỏ trống thì sẽ không có dấu ? sau kiểu dữ liệu
* Và cũng như khóa ngoại của bảng User\_account sẽ là id trong bảng User\_type
* Thì với thư viện Prisma ta có thể ánh xạ các model thành các bảng trong cở sở dữ liệu và thư viện này có hỗ trợ một số loại cơ sở dữ liệu phổ biến hiện này như: PostgreSQL , MySQL, SQLite, MongoDB, SQL Server
* Để tạo bảng trong các cơ sở dữ liệu ta chỉ cần thực hiện các thao tác sau
  + npx prisma studio : để mở GUI cho người viết code thấy được giao diện của prisma cùng với các bảng của mình



* + npx prisma migrate : Để khởi tạo các bảng lưu lại quá trình thay đổi cũng như các file có dạng SQL ứng với mỗi migrate sẽ có 1 file SQL sinh ra và Prisma sẽ dựa vào tất cả các file SQL ở từng thư mục trong migrate để khởi tạo cơ sở dữ liệu cũng như là cập nhật các thuộc tính trong quá trình ta thao tác với các model
  + Sau khi chạy câu lệnh này ta trong SQL Server ta khởi tạo và gán địa chỉ vào config để khi tạo từ prisma thì các bảng sẽ vào cơ sở dữ liệu đó sẽ hình thành các bản như các model mà ta đã khai báo trong Prisma ở NodeJS

1. ReactJS

ReactJS là một opensource được phát triển bởi Facebook, ra mắt vào năm 2013, bản thân nó là một thư viện JavaScript chuyên giúp các nhà phát triển xây dựng giao diện người dùng hay UI, Nó được phân loại thành kiểu “V” trong mô hình **MVC** (Model-View-Controller). Một trong những điểm nổi bật nhất của ReactJS đó là việc render dữ liệu không chỉ thực hiện được trên tầng Server mà còn ở dưới Client nữa.

React được sử dụng bởi hàng trăm công ty lớn trên thế giới, bao gồm Netflix, Airbnb, American Express, Facebook, WhatsApp, eBay, và Instagram.

Hơn nữa, để tăng tốc quá trình phát triển và giảm thiểu những rủi ro có thể xảy ra trong khi coding, React còn cung cấp cho chúng ta khả năng Reusable Code (tái sử dụng code)

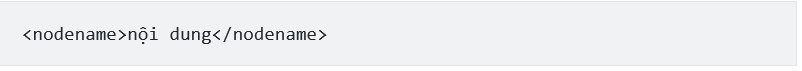
1. JSX

JSX (nói ngắn gọn cho JavaScript extension)  là một cú pháp mở rộng cho JavaScript. Nó giúp chúng ta dễ dàng thay đổi cây DOM bằng các HTML-style code đơn giản.

JSX = Javascript + XML. Nó transform cú pháp dạng gần như XML về thành Javascript. Giúp người lập trình có thể code ReactJS bằng cú pháp của XML thay vì sử dụng Javascript. Các XML elements, attributes và children được chuyển đổi thành các đối số truyền vào React.createElement.

XML là viết tắt của từ eXtensible Markup Language, hay còn gọi là ngôn ngữ đánh dấu mở rộng do W3C đề nghị với mục đích tạo ra các ngôn ngữ đánh dấu khác. Đây là một tập hợp con đơn giản có thể mô tả nhiều loại dữ liệu khác nhau nên rất hữu ích trong việc chia sẻ dữ liệu giữa các hệ thống. Điển hình nhất là ngôn ngữ đánh dấu siêu văn bản HTML sử dụng cú pháp của XML để tạo nên và nó có các bộ phần tử và thuộc tính không mềm dẻo nên chỉ có tác dụng trong việc trình bày dữ liệu trên trình duyệt Browser.

Cú pháp của tài liệu XML XML được xây dựng dựa vào cấu trúc NODE lồng nhau, mỗi node sẽ có một thẻ mở và một thẻ đóng như sau:



Trong đó:

* <nodename> là thẻ mở, tên của thẻ này do bạn tự định nghĩa.
* </nodename> là thẻ đóng, tên của thẻ này phải trùng với tên của thẻ mở.
* content là nội dung của thẻ này

Cú pháp của JSX cũng tương tự như XML.

* Ta có thẻ mở tag:

<JSXElementName JSXAttributesopt>

* Đóng tag:

</JSXElementName>

* Ở đây lưu ý tên của thẻ mở tag và đống tag phải giống nhau. ví dụ :

<MyButton color="blue" shadowSize={2}>

Click Me

</MyButton>

* Ngoài ra JSX cũng có SelfClosingElement:

<JSXElementName JSXAttributes/>

* ví dụ:

<div className="sidebar" />

1. Tại sao lại nên dùng JSX

Việc sử dụng JSX trong ReactJS là không bắt buộc. Bạn có thể sử dụng chỉ JS thuần thôi. Thay vì tách biệt các công nghệ một cách giả tạo bằng cách đưa định nghĩa giao diện và logic vào những tệp khác nhau, React [tách bạch mối quan hệ](https://en.wikipedia.org/wiki/Separation_of_concerns) bằng những đơn vị rời rạc gọi là “components” chứa cả hai cái trên. Nhưng có rất nhiều lý do cho việc nên sử dụng JSX trong ReactJS đấy.

* Thứ nhất, JSX với cú pháp gần giống XML, cấu trúc cây khi biểu thị các attributes, điều đó giúp ta dễ dàng định nghĩa, quản lý được các component phức tạp, thay vì việc phải định nghĩa và gọi ra nhiều hàm hoặc object trong javascript. Khi nhìn vào cấu trúc đó cũng dễ dàng đọc hiểu được ý nghĩa của các component. Code JSX ngắn hơn, dễ hiểu hơn code JS.
* Thứ 2, JSX không làm thay đổi ngữ nghĩa của Javascript
* Thứ 3, với cách viết gần với các thẻ HTML, nó giúp những developers thông thường (ví dụ như các designer) có thể hiểu được một cách dễ dàng, từ đó có thể viết hoặc sửa code mà không gặp nhiều khó khăn. Ví du với đoạn code JSX như sau:

const myelement = <h1>I Love JSX!</h1>;

ReactDOM.render(myelement, document.getElementById('root'));

* Không có JSX, đoạn code sẽ như sau:

const myelement = React.createElement('h1', {}, 'I do not use JSX!');

ReactDOM.render(myelement, document.getElementById('root'));

1. Components

Components cho phép bạn chia UI thành các phần độc lập, có thể tái sử dụng, và hoàn toàn tách biệt nhau. Về mặt khái niệm, components cũng giống như các hàm Javascript. Chúng nhận vào bất kì đầu vào nào (còn được gọi là “props”) và trả về các React elements mô tả những gì sẽ xuất hiện trên màn hình.

Có 2 kiểu components 1 là function components 2 là class components nhưng chúng ta chỉ bàn về function components ở đây.

Cách đơn giản nhất để định nghĩa một component đó là viết một hàm JavaScript:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

Hàm này là một React component hợp lệ về nó nhận đầu vào là một tham số object “props” (viết tắt của properties) với dữ liệu và trả về một React element. Chúng ta gọi các components này là “function components” vì chúng là các hàm JavaScript theo đúng nghĩa đen.

Component có thể sử dụng các HTML tag

function App() {

return (

<div>

<Welcome name="Sara" />

);

}

**Chú ý:** Luôn luôn bắt đầu tên của component bằng chữ in hoa.

React xử lí các components bắt đầu với chữ thường giống như các DOM tags. Ví dụ, <div /> biểu diễn HTML div tag, nhưng <Welcome /> biểu diễn một component và yêu cầu Welcome nằm trong scope.

Các components có thể tham chiếu đến các components khác tại đầu ra của chúng. Điều này cho phép chúng ta sử dụng cùng một component abstraction cho mọi mức độ chi tiết. Một button, form, dialog, màn hình: trong React apps, chúng đều được hiển thị như là các components.

Thông thường, các React apps mới tạo sẽ có một App component ở tầng cao nhất. Thế nhưng, nếu bạn tích hợp React vào ứng dụng hiện có, bạn có thể bắt đầu bằng cách tiếp cận bottom-up với một component nhỏ như là Button và dần dần đi lên các tầng trên cùng của cây kế thừa giao diện.

Đừng ngại ngần việc tách components thành các components nhỏ hơn. Chia các components ngay từ đầu là một công việc không đơn giản, nhưng bù lại chúng ta sẽ có được một tập hợp các components có thể tái sử dụng trong các ứng dụng lớn hơn khác. Một nguyên tắc quan trọng đó là nếu một phần UI của bạn được sử dụng lại nhiều lần

1. Lifecycle

Lifecycle là vòng đời của 1 component react. Chúng ta có thể tương tác lifecycle của 1 component thông qua các giai đoạn của lifecycle:

Có 3 giai đoạn chính **Mounting**, **Updating**, và **Unmounting**.

**Mounting:** Đây là giai đoạn đầu tiên của 1 component đó là khi component được khởi tạo

**Updating:** Đây là giai đoạn sau khi component được khởi tạo. mà được cập nhật mỗi khi state hoặc props thay đổi

**Unmounting:** Đây là giai đoạn khi component được xóa khởi DOM

1. Props và State

* Props là gì

Props là 1 từ viết ngắn gọn của properties. props là 1 đối tượng, nó lưu trữ các giá trị của các attribute (thuộc tính) của một thẻ (Tag).Là cách mà component có thể nhận được các giá trị của thuộc tính truyền vào từ bên ngoài vào, và là cách mà các component có thể nói chuyện với nhau.

Ví dụ :

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

const element = <Welcome name="ReactJS" />;

ReactDOM.render(

element,

document.getElementById('root')

);

Trong đó, name ở {props.name} là property, Welcome là Component. Mỗi property của Component sẽ tương ứng với 1 attribute của thẻ, giá trị của attribute sẽ được truyền vào property của Component.

* Chúng ta sử dụng props để truyền gửi dữ liệu đến các component.
* Mọi component được gọi là hàm javascript thuần khiết (pure function).
* props tương ứng với tham số của pure function javascript .
* Không thể thay đổi dữ liệu của props .

Dữ liệu của của React là một chiều có nghĩa là nó chỉ đi từ component cha xuống component con

* **State là gì**

Cũng giống như props , state cũng lưu trữ thông tin về component, nhưng là lưu trữ dữ liệu động của một component.

State là dữ liệu động , nó cho phép một component theo dõi thông tin thay đổi giữa các kết xuất (render) và làm cho nó có thể tương tác.

State chỉ có thể được sử dụng ở trong một component sinh ra nó.

Nếu dự đoán được một component cần quản lý state, thì nó nên được tạo ở trong một class component chứ không phải là một function component. Vì function components là stateless component có nghĩa là nó không có state để lưu trữ thông tin, thay vào đó nó sử dụng các hook để lưu giữ thông tin.

1. Hook

Hooks chính thức được giới thiệu trong phiên bản React 16.8. Nó cho phép chúng ta sử dụng state và các tính năng khác của React mà không phải dùng đến Class

Điều này có nghĩa là từ phiên bản 16.8 trở đi, chúng ta đã có thể sử dụng state trong stateless (functional) component

Hook là 1 function có thể gắn (hook) vào 1 function component để giúp component có thể quản lý các state hay các tính năng vòng đời của 1 component

Hook không hoạt động được trong class component

* **useState**

Hàm useState nhận đầu vào là giá trị khởi tạo của 1 state và trả ra 1 mảng gồm có 2 phần tử, phần tử đầu tiên là state hiện tại, phần tử thứ 2 là 1 function dùng để update state. Ví dụ:

const [isLoading, setLoading] = useState(false);

onClick() {

setLoading(true)

}

Trong đó:

* isLoading là state lưu trữ giá trị hiện tại
* setLoading là hàm để thay đổi giá trị của isLoading
* false là giá trị khởi tạo của state
* **useEffect**

Hàm useEffect dùng để xử lý các side effects. Nó thay thế các hàm cũ trong lifecycle.

Hàm useEffect nhận đầu vào gồm 2 tham số:

Tham số 1 là 1 hàm sẽ được chạy khi các giá trị trong chuỗi ở hàm thứ 2 thay đổi. trong hàm này nếu bạn trả về 1 hàm thì hàm trả về này sẽ được chạy khi component được update

Tham số 2 là 1 chuỗi các biến hoặc hàm, khi các các giá trị này thay đổi hàm ở tham số 1 sẽ chạy

 Ví dụ:

import { callApi } from './actions'

const App = ({ callApi, data }) => {

const callApiWhenStart = () => {

callApi('some\_payload\_')

}

useEffect(callApiWhenStart,[])

return(

<div>

{data.map(item => {// do something })}

</div>

)

}

export default App

Trong đó callApiWhenStart là hàm sẽ chạy khi 1 trong các phần tử của chuỗi ở tham số 2 thay đổi, tuy nhiên vì tham số 2 ở đây là chuỗi rỗng nên hàm callApiWhenStart sẽ chỉ chạy 1 lần khi component được tạo lần đầu tiên

Ở 1 ví dụ khác:

useEffect(

() => {

const subscription = props.source.subscribe();

return () => {

subscription.unsubscribe();

};

},

[props.source], // giá trị được subcrive

);

Trong đó mỗi khi props.source thay đổi giá trị thì hàm ở tham số 1 sẽ chạy lại và trước khi hàm ở tham số 1 sẽ chạy lại hàm return sẽ chạy để dọn dẹp những dữ liệu của hàm ở tham số 1 để tránh tốn bộ nhớ

* useMemo

useMemo giúp ta kiểm soát việc được render dư thừa của các component con bằng cách truyền vào 1 tham số thứ 2 thì chỉ khi tham số này thay đổi thì thằng useMemo mới được thực thi. Ví dụ:

const NotUsingMemo = ({ products }) => {

const soldoutProducts = products.filter(x => x.isSoldout === true); };

const UsingMemo = ({ products }) => {

const soldoutProducts = useMemo(

() => products.filter(x => x.isSoldout === true), [products]

);

};

Trong đó ở component NotUsingMemo mỗi khi component được render thì soldoutProducts sẽ được thực khi lại. Điều này dẫn đến việc phải tính toán lại cả những trường hợp đầu vào và đầu ra không thay đổi. Nếu việc tính toán là 1 hàm nặng tốn nhiều thời gian vậy sẽ gay ảnh hưởng đến trải nghiệp người dùng

Trong khi đó ở component UsingMemo soldoutProducts sẽ chỉ được tính toán lại nếu như products (tham số truyền vào ở tham số thứ 2 của hàm useMemo) thay đổi. Nó giúp ta loại bỏ được các trường hợp đầu vào không thay đổi, giúp giảm tải việc tính toán

* useCallback

useCallback có nhiệm vụ tương tự như useMemo nhưng khác ở chỗ function truyền vào useMemo bắt buộc phải ở trong quá trình render trong khi đối với useCallback đó lại là function callback của 1 event nào đó như là onClick chẳng hạn.

* useContext

Như đã nói ở trên dòng dữ liệu trong react chỉ đi 1 chiều từ cha xuống con vậy làm sao để truyền props từ con lên cha hoặc giữa các component con với nhau. Để giải quyết được vấn đề này, bạn có thể tạo một context để thêm những shared state và sau đó component nào cần thì bạn gọi context đó ra xài, không cần phải truyền từ cha xuống con nữa. Nó cũng giống như biến global được chia sẻ và sử dụng ở nhiều nơi, nhưng context được quản lí tốt hơn để maintain code dễ hơn.

Ví dụ sử dụng context:

import React from 'react';

const ExampleContext = React.createContext();

const App = () => {

return (

<ExampleContext.Provider value={{ color: 'red' }}>

<div className='App'>

<ChildComponent />

</div>

</ExampleContext.Provider>

);

};

const ChildComponent = () => {

const { color } = React.useContext(ExampleContext);

return <p style={{ color }}>This text is {color}</p>;

};

export default App;

Đầu tiên chúng ta sẽ định nghĩa 1 context:

const ExampleContext = React.createContext();

Sau đó bao bọc toàn bộ thành phần DOM của component bằng thẻ Provider, đồng thời truyền giá trị mà mình muốn chia sẻ đến các component khác

<ExampleContext.Provider value={{ color: 'red' }}>

<div className="App">

<ChildComponent />

</div>

</ExampleContext.Provider>

Như vậy là chúng ta đã có thể sử dụng context đó trong các component con thông qua useContext:

const ChildComponent = () => {

const { color } = React.useContext(ExampleContext);

return <p style={{ color }}>This text is {color}</p>;

};

Ngoài ra value của hàm Provider còn có thể chưa các function

* Các hook khác và custom hook

Ngoài ra chúng ta còn rất nhiều cái hook khác được định nghĩa sẳn bởi React mà các bạn có thể đọc thêm ở trên trang chủ của React hoặc chúng ta có thể tự viết hook cho mình trong trường hợp cái logic đó lập lại ở các component khác nhau. Các hàm hook luôn được bắt đầu với use (vd: useFetch). Ví dụ:

import { useState, useEffect } from "react";

const useFetch = (url) => {

const [data, setData] = useState(null);

useEffect(() => {

fetch(url)

.then((res) => res.json())

.then((data) => setData(data));

}, [url]);

return [data];

};

export default useFetch;

Trong ví dụ trên ta tạo sử dụng hook useFetch để lấy dữ liệu từ data. Hàm useFetch nhận vào url và trả về data và sẽ gọi api mỗi khi url thay đổi (ứng dụng useEffect đã giới thiệu ở trên). Sau đó chúng ta có thể sử dụng useFetch như các hook khác

import useFetch from "./useFetch";

const Home = () => {

const [data] = useFetch("https://jsonplaceholder.typicode.com/todos");

return (

<>

{data &&

data.map((item) => {

return <p key={item.id}>{item.title}</p>;

})}

</>

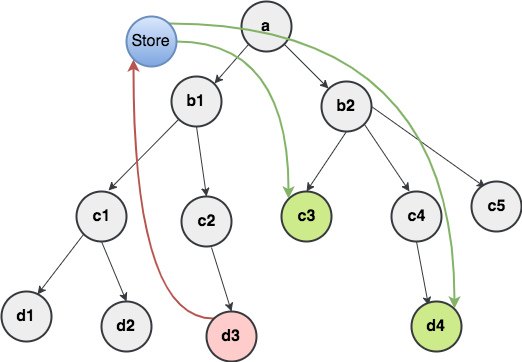
);

};

1. Redux

Redux là 1 thư viện Javascript để quản lý state của ứng dụng, thường được sử dụng với javascript framework như React.

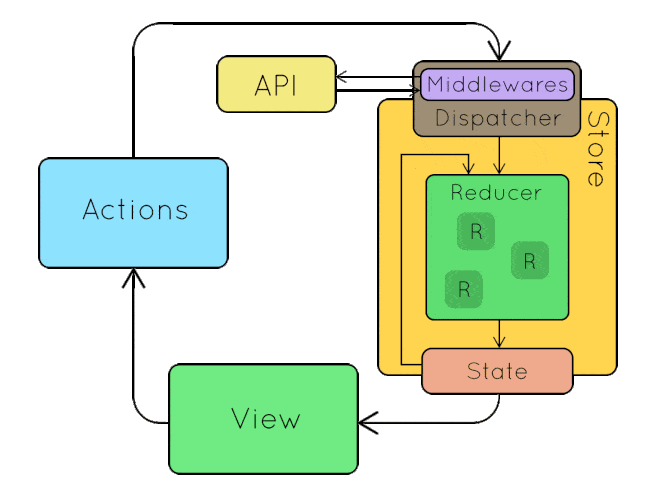
Như đã đề cập bên trên dữ liệu của react chỉ truyền 1 chiều từ cha cho con. Việc truyền dữ liệu giữa con lên cha hoặc giữ các component đồng cấp rất phức tạp. Chúng ta có thể dùng useContext đã đề cập ở trên tuy nhiên useContext sẽ render lại tất cả các component nằm trong provider khi value thay đổi điều này ổn với các ứng dụng nhỏ tuy nhiên với các ứng dụng lớn việc này sẽ giảm hiện xuất của trang web. Do đó redux là 1 giải pháp quản lý state được nhiều người sử dụng khi sử dụng React. Trái với useContext, khi sử dụng Redux các component chỉ render lại những component thay đổi props hoặc state tuy nhiên việc setup Redux đòi hỏi nhiều công sức hơn là useContext.



Hình 1 Mô hình hoạt động của Redux

Cái cách mà Redux hoạt động là khá đơn giản. Nó có 1 store lưu trữ toàn bộ state của app. Mỗi component có thể access trực tiếp đến state được lưu trữ thay vì phải truyền props từ component này đến component khác.

Cơ chế hoạt động của nó được tóm gọn trong 1 sơ đồ đơn giản:



Hình 2 Sơ đồ hoạc động của Redux

Khi người dùng tương tác với UI sẽ tạo ra 1 action action này sẽ được dispatcher gửi đến reducer (các action bất đồng bộ ví dụ như gọi api sẽ được sử lý bởi các middleware tại bước này). Reducer sẽ nhận thông tin từ action và state hiện tại và sử lý sao đó trả về một state mới. Khi state thay đổi Ui cũng sẽ cập nhật theo.

Thành phần chính của Redux gồm: Actions, Store, Reducers.

1. Action

Actions đơn giản là các events. Chúng là cách mà chúng ta send data từ app đến Redux store. Những data này có thể là từ sự tương tác của user vs app, API calls hoặc cũng có thể là từ form submission.

Actions được gửi bằng cách sử dụng store.dispatch() method, chúng phải có một type property để biểu lộ loại action để thực hiện. Chúng cũng phải có một payload chứa thông tin. Actions được tạo thông qua một action creator. Ví dụ:

const setLoginStatus = (name, password) => {

return {

type: "LOGIN",

payload: {

username: "foo",

password: "bar"

}

}

}

Trong đó setLoginStatus là tên của action (function) sẽ được gọi trong hàm dispatch. setLoginStatus trả về 1 object gồm type để reducer nhận biết phải thực hiện hành động nào và payload để truyền data xuống reducer

1. **Reducers**

Reducers là các function nguyên thủy chúng lấy state hiện tại của app, thực hiện một action và trả về một state mới. Những states này được lưu như những objects và chúng định rõ cách state của một ứng dụng thay đổi trong việc phản hồi một action được gửi đến store. 1 reducer tương đương với 1 state nhưng kèm theo các mô tả state sẽ thay đổi như thế nào khi các action khác nhau được gọi. Ví dụ:

export default function counterApp (state = initialState, action) {

switch (action.type) {

case INCREASE:

return {

increase: ++state.increase,

decrease: state.decrease

}

case DECREASE:

return {

increase: state.increase,

decrease: ++state.decrease

}

default:

return state

}

}

Trong đó counterApp là tên của reducer. Trong 1 ứng dụng có thể có nhiều reducer. Mỗi reducer có 1 tên khác nhau. Reducer nhận vào 2 tham số là state với giá trị default là initialState (giá trị khởi tạo của state) và 1 action

Reducer sẽ dựa theo action type để thay đổi state với payload (nếu có) và trả về state mới. Lưu ý state mới phải là object mới nếu không Redux sẽ không thể cập nhật lại UI

1. Store

Store là nơi lưu trạng thái của ứng dụng và chỉ có duy nhất một Store trong bất kỳ một chương trình ứng dụng Redux nào. Nhiệm vụ của store chính là quản lý, access các state được lưu, update state thông qua dispatch, cho phép truy cập state thông qua Getstate và đăng ký hoặc hủy đăng ký các listeners thông qua helper methods.

1. Sử dụng Redux

Sử dụng useSelector của react-redux để lấy state counter từ store.

Sử dụng useDispatch để trả về function dispatch, truyền increment và decrement vào dispatch để gọi 2 action này

1. **Redux-toolkit**

Như đã trình bày ở trên việc setup redux khá phức tạp và tốn thời gian. Redux-toolkit là một package được sinh ra nhằm giải quyết phần lớn những vấn đề kể trên, được phát triển bởi chính chủ reduxjs team, giúp chúng ta viết code redux nhanh gọn, hoàn chỉnh theo một quy chuẩn thống nhất.

* configureStore

Hàm configureStore là 1 hàm của redux-toolkit giúp chung ta đơn giản việc setup store. Như ví dụ ở trên, chúng ta khởi tạo một store bằng hàm createStore của redux với tham số nhận vào là một reducer, configureStore sẽ làm điều tương tự như vậy, chúng ta sẽ khởi tạo store theo phương pháp sau

import { configureStore } from '@reduxjs/toolkit';

const store = configureStore({

reducer: rootReducer

})

Về cú pháp thì không khác nhau nhiều. Tuy nhiên thay vì chỉ khởi tạo một store đơn thuần, configureStore sẽ mặc định thiết lập cho phép sử dụng redux devtool để debug và theo dõi quá trình cập nhật state cũng như thiết lập sẵn một số middleware giúp thực hiện các action bất đồng bộ.

* createSlice

createSlide là hàm của redux-toolkit giúp chúng ta định nghĩa 1 reducer và các action liên quan đến nó. Ví dụ:

const initialState = {  
 value: 0,  
}  
  
export const counterSlice = createSlice({  
 name: 'counter',  
 initialState,  
 reducers: {  
 increment: (state) => {  
 state.value += 1  
 },  
 decrement: (state) => {  
 state.value -= 1  
 },  
 incrementByAmount: (state, action) => {  
 state.value += action.payload  
 },  
 },  
})  
  
export const { increment, decrement, incrementByAmount } = counterSlice.actions  
  
export default counterSlice.reducer

createSlice nhận vào 1 object gồm:

* name: Tên của reducer
* initialState: state khởi tạo của reducer
* reducer: là 1 object định nghĩa các hàm quy định reducer sẽ cập nhật state như thế nào. Bên cạnh đó như đã nhắc ở trên reducer phải trả về 1 object mới tuy nhiên redux-toolkit đã đơn giản hóa việc đó chúng ta có thể thực hiện tính toán ngay trên object cũ

createSlide trả về 1 object chứa actions và reducer. Chúng ta có thể export các action và reducer để sử dụng. Như vậy chỉ 1 hàm đơn giản đã giúp chúng ta tiết kiệm rất nhiều thời gian bằng cách gọp bước tạo action và tạo reducer vào làm 1 .

1. Typecript

TypeScript là một phiên bản cao hơn của JavaScript, được thiết kế để xây dựng các ứng dụng lớn và phức tạp.  Nó kế thừa nhiều khái niệm từ Java và C#, TypeScript là ngôn ngữ tĩnh (Static typed) có nghĩa là nó nghiêm ngặt và có trật tự trái ngược với free-type. Nó còn được bổ sung thêm lớp hướng đối tượng mà điều này không có ở Javascript. Với TypeScript, ta có thể bê nguyên xi code JavaScript vào trong cùng một file và chạy cùng nhau bình thường, bởi vì TypeScript duy trì cú pháp của JavaScript và mở rộng nó bằng một loạt tính năng mới. Nhờ đó mà hiệu năng làm việc được tăng lên đáng kể.

1. **Ưu điểm của Typescript:**

* Dễ dàng hơn trong phát triển các dự án lớn, được hỗ trợ bởi các Javascript Framework lớn.
* Hầu hết các cú pháp hướng đối tượng đều được hỗ trợ bởi Typescript như kế thừa, đóng gói, constructor, abstract, interface, implement, override…v.v
* Cách tổ chức code rõ ràng hơn, hỗ trợ cơ chế giúp kiến trúc hệ thống code hướng module, hỗ trợ namespace, giúp xây dựng các hệ thống lớn nơi mà nhiều lập trình viên có thể làm việc cùng nhau một cách dễ dàng hơn.
* Hỗ trợ các tính năng mới nhất của Javascript. TypeScript luôn đảm bảo việc sử dụng đầy đủ các kỹ thuật mới nhất của Javascript, ví dụ như version hiện tại là ECMAScript 2015 (ES6).
* Một lợi thế của Typescript nữa là mã nguồn mở vì vậy nó miễn phí và có cộng đồng hỗ trợ rất lớn.
* Với static typing, code viết bằng TypeScript dễ dự đoán hơn, và dễ debug hơn.

1. Basic Types:

Trong TypeScript chia ra làm 7 loại cơ bản, bao gồm: boolean, number, string, array, enum, any, void.

* Boolean:

var isDone: boolean = true;

* String:

var name: boolean = "nguyen thi A";

* Number:

var height: number = 8;

* Array : có 2 kiểu khai báo tương đương với nhau trong TypeScript

1: var list: boolean[] = [true, false];

2: var isDone: Array<boolean> = [true, false];

* Enum: khi khai báo enum một cách thông thường các phần tử sẽ được đánh số từ 0 và tăng dần

enum Color{Red, Green, Blue};

var c: Color = Color.Green

var colorName = Color[1] // kết quả sẽ là Green

* Khi muốn phần tử đầu tiên là 1 chứ không phải là 0 như mặc định thì cần khai báo như sau:

enum Color{Red = 1, Green, Blue};

var c: Color = Color.Green

var colorName = Color[1] // kết quả sẽ là Red

* Any: Any là một kiểu mà bạn có thể gán bất kỳ kiểu nào cho nó.

var notSure: any = 4;

notSure = "maybe a string instead";

notSure = false; // khai báo này hoàn toàn được chấp nhận.

// nếu notSure ban đầu khai báo và number thì

// tại đây chắc chắn sẽ có lỗi

var list:any[] = [1, true, "free"]; // nếu sử dụng var list:number[] thì

// tất cả các phần tử trong list sẽ phải là kiểu number

list[1] = 100;

* Void: Cũng giống như any nhưng void được sử dụng là đầu ra của hàm.

function warnUser(): void {

alert("This is my warning message");

}

1. Function

TypeScript có khai báo function giống như javaScript. Nhưng khi khai báo function TypeScript còn hỗ chợ việc khai báo với các kiểu trả ra của function và cũng như kiểu đầu vào của dữ liệu. Không những thế khi sử dụng TypeScript ta có thể khai báo giá trị mặc định của đầu vào ngay khi khai báo function, điều mà JavaScript không có. Không dừng lại ở đó typeScript còn hỗ chợ việc bỏ qua nhập một hoặc vài đầu vào.

Ví dụ:

function buildName(firstName: string, lastName?: string):string {

if (lastName)

return firstName + " " + lastName;

else

return firstName;

}

Trong đó

* “firstName: string”: Khai báo kiểu dữ liệu string cho firstName
* “lastName?: string”: Khai báo kiểu dữ liệu string lastName tuy nhiên lastName có thể là undefined
* “:string”: là kiểu trả về của function

1. Type aliases

Trong TypeScript, có rất nhiều types cơ bản như là number, string,... Ngoài ra, trong TypeScript chúng ta có các types nâng cao và các types nâng cao này được gọi là type aliases. Với type aliases, chúng ta có thể tạo tên mới cho một type nhưng chúng ta không define một type mới. Từ khóa type trong TypeScript là một cách cung cấp type aliases cho các variables, objects và functions của chúng ta. Các type aliases này mô tả dữ liệu của chúng ta trông như thế nào. Để mô tả loại dữ liệu của mình trông như thế nào chúng ta sử dụng các type cơ bản (string, number,...) hoặc bằng cách tạo tạo các type tùy chỉnh của chúng ta.

Ví dụ:

// type của Year là một number

type Year = number;

// Biến currentYear là thuộc type Year và phải là number

let currentYear: Year = 2021;

// custom Person object type

type Person = {

name: String;

gender: String;

};

1. Interface

Interface trong typescript cho phép bạn định nghĩ thuộc tính là gì và phương thức là gì mà đối tượng cần để được implement. Nếu đối tượng tuân thủ đúng khuôn mẫu interface thì đối tượng đã implement interface ấy sẽ được thực thi đúng. Nếu interface không được thực thi đúng thì typescript sẽ phát sinh lỗi ngay lập tức.

interface Person {

name: string

age: number

}

const viblo: Person = {

name: "Viblo",

age: 31

};

1. Types vs Interfaces

* Có thể merge interfaces, types thì không

interface Person {

name: string

}

interface Person {

age: number

}

const viblo: Person = {

name: "Viblo",

age: 31

}

* Type aliases có thể sử dụng computed properties:

Từ khóa in có thể được sử dụng để iterate tất cả các item bên trong một tập hợp keys. Chúng ta có thể sử dụng tính năng này để tạo mapped types.

Ví dụ sử dụng type aliases

type keys = "firstname" | "lastname"

type Person = {

[key in keys]: string

}

const viblo: Person = {

firstname: "Viblo",

lastname: "blog"

}

* Extend và implements:

Trong TypeScript, chúng ta dễ dàng extends và implements interfaces. Nhưng không thể với types Với interface để kế thừa chúng ta sử dụng extends:

interface Animal {

name: string

}

interface Bear extends Animal {

honey: boolean

}

1. Intersection

Intersection cho phép chúng ta kết hợp nhiều types thành một types duy nhất. Để tạo một intersection types chúng ta dùng &:

type Name = {

name: string

};

type Age = {

age: number

};

type Person = Name & Age;

Chúng ta cũng có thể làm điều này với interface:

interface Name {

name: “string”

};

interface Age {

age: number

};

type Person = Name & Age;

1. Generics

Generic type trong TypeScript cho phép bạn viết các function, class và interface có thể tái sử dụng và tổng quát hóa. Trong hướng dẫn này, bạn sẽ tập trung vào việc tạo các hàm generic (generic function).

Giả sử ta gặp trường hợp chúng ta phải định nghĩa 1 function trả về ngẫu nhiên 1 phần tử trong 1 mảng. Nếu chúng ta biết mảng vào sẽ là mảng number thì chúng ta có function:

function getRandomNumberElement(items: number[]): number {

let randomIndex = Math.floor(Math.random() \* items.length);

return items[randomIndex];

}

Tuy nhiên sẽ ra sao nếu chuỗi vào có thể là mảng number hoặc là mảng string. Trong trường hợp này chúng ta có thể tạo thêm 1 hàm

function getRandomStringElement(items: string[]): string {

let randomIndex = Math.floor(Math.random() \* items.length);

return items[randomIndex];

}

Vậy nếu chuỗi vào có thể là bất cứ type nào thì sao? Chúng ta không thể tạo ra tất cả các hàm cho mỗi type được. Vậy phải làm sao đây? Bạn có thể dùng type any tuy nhiên nếu như vậy bạn sẽ không xác định được type trả về hay nói cách khác là nó không an toàn. Do đó typescript cho chúng ta 1 giải pháp cho nhưng trường hợp như vậy đó là generic type.

Chúng ta có thể định nghĩa 1 hàm generic như sau:

function getRandomElement<T>(items: T[]): T {

let randomIndex = Math.floor(Math.random() \* items.length);

return items[randomIndex];

}

Hàm này sử dụng biến kiểu T. Kiểu T cho phép bạn chỉ định kiểu dữ liệu tại thời điểm gọi hàm. Ngoài ra, hàm sử dụng biến kiểu T làm kiểu trả về của nó.

Hàm getRandomElement() là hàm generic bởi vì nó có thể làm việc với bất kỳ kiểu dữ liệu nào, ví dụ: chuỗi, số, đối tượng, ...

Theo quy ước, chúng ta sử dụng ký tự T làm kiểu dữ liệu của biến. Tuy nhiên, bạn có thể sử dụng các chữ khác như A, B C, ...

Sau đây là cách sử dụng hàm getRandomElement() với một mảng số:

let numbers = [1, 5, 7, 4, 2, 9];

let randomEle = getRandomElement<number>(numbers);

console.log(randomEle);

Ví dụ này chỉ định rõ kiểu number cho T khi gọi hàm getRandomElement().

Trong thực tế, bạn sẽ sử dụng kiểu suy luận cho đối số. Có nghĩa là bạn để trình biên dịch TypeScript tự xác định kiểu dữ liệu của T dựa trên loại đối số mà bạn truyền vào, như thế này:

let numbers = [1, 5, 7, 4, 2, 9];

let randomEle = getRandomElement(numbers);

console.log(randomEle);

Trong ví dụ này, chúng tôi đã không chỉ định kiểu number một cách rõ ràng khi gọi hàm getRandomElement(). Trình biên dịch sẽ xác định kiểu dữ liệu của T dựa vào kiểu dữ liệu của tham số truyền vào.

Bây giờ, hàm getRandomElement() cũng là kiểu an toàn. Ví dụ: nếu bạn gán giá trị trả về cho một biến kiểu chuỗi, bạn sẽ gặp lỗi:

let numbers = [1, 5, 7, 4, 2, 9];

let returnElem: string;

returnElem = getRandomElement(numbers); // compiler error

Hàm generic có nhiều kiểu dữ liệu trong TypeScript

Phần sau minh họa cách tạo một hàm generic với hai biến kiểu U và V:

function merge<U, V>(obj1: U, obj2: V) {

return {

...obj1,

...obj2

};

}

Hàm merge() kết hợp hai đối tượng với kiểu U và V. Nó kết hợp các thuộc tính của hai đối tượng thành một đối tượng duy nhất.

Kiểu suy luận suy ra giá trị trả về của hàm merge() là kiểu giao của U và V, là U & V.

Sau đây minh họa cách sử dụng hàm merge() để hợp nhất hai đối tượng:

let result = merge(

{ name: 'John' },

{ jobTitle: 'Frontend Developer' }

);

console.log(result);

Đầu ra:

{ name: 'John', jobTitle: 'Frontend Developer' }

Tuy nhiên trong ví dụ trên Hàm merge() mong đợi đầu vào là hai đối tượng. Tuy nhiên, nó không ngăn bạn truyền vào tham số như thế này:

let person = merge(

{ name: 'John' },

25

);

console.log(person);

Đầu ra:

{ name: 'John' }

Code vẫn chạy mà không có một lỗi nào, tuy nhiên đó không phải là điều chúng ta mong muốn. Chúng ta muốn đầu vào của hàm phải là 2 object vì vậy chúng ta có thể gàng buộc generic type là là type mở trọng của object với từ khóa extends.

function merge<U extends object, V extends object>(obj1: U, obj2: V) {

return {

...obj1,

...obj2

};

}

Điều này dẫn đến đoạn code dưới đây lỗi và đó là điều chúng ta mong muốn.

let person = merge(

{ name: 'John' },

25

);

TypeScript cho phép bạn khai báo một tham số có kiểu dữ liệu bị ràng buộc bởi kiểu dữ liệu của một tham số khác.

Hàm prop() sau chấp nhận hai tham số là một đối tượng và một tên thuộc tính. Nó trả về giá trị của thuộc tính.

function prop<T, K>(obj: T, key: K) {

return obj[key];

}

Trình biên dịch gặp lỗi sau:

Type 'K' cannot be used to index type 'T'.

Để khắc phục lỗi này, bạn thêm ràng buộc cho kiểu dữ liệu K để đảm bảo rằng nó là một khóa của kiểu dữ liệu T như sau:

function prop<T, K extends keyof T>(obj: T, key: K) {

return obj[key];

}

Nếu bạn truyền vào hàm prop một tên thuộc tính tồn tại trên obj, trình biên dịch sẽ không báo lỗi. Ví dụ:

let str = prop({ name: 'John' }, 'name');

console.log(str);

Đầu ra:

John

Tuy nhiên, nếu bạn truyền một khóa không tồn tại trên đối số đầu tiên, trình biên dịch sẽ phát ra lỗi:

let str = prop({ name: 'John' }, 'age');

Lỗi:

Argument of type '"age"' is not assignable to parameter of type '"name"'.

1. Antd
2. Antd là gì

Ant là tập hợp các components của React được xây dựng theo chuẩn thiết kế của Ant UED Team. Tương tự như chuẩn Material Design, Ant cung cấp hầu hết các component thông dụng trong ứng dụng web hiện đại, như Layout, Button, Icon, DatePicket, v.v…Bên cạnh đó Ant cũng có những component riêng thú vị, như LocaleProvider cho phép bạn thay đổi ngôn ngữ trên toàn ứng dụng.

Có thể coi Ant Design cho React là tập hợp của hầu hết các thư viện về React. Nó đáp ứng được hầu hết các yêu cầu của project của bạn mà ban không phải cài thêm bất cứ thư viện nào nữa. Dưới đây là danh sách các component mà nó cung cấp:

* General: Button, Icon
* Layout: Grid, Layout
* Navigation: Affix, Breadcrumb, Dropdown, Menu, Pagination, Steps
* Data Entry: AutoComplete, Checkbox, Cascader, DatePicker, Form, InputNumber, Input, Mention, Rate, Radio, Switch, Slider, Select, TreeSelect, Transfer, TimePicker, Upload
* Data Display: Avatar, Badge, Collapse, Carousel, Card, Calendar, List, Popover, Tree, Tooltip, Timeline, Tag, Tabs, Table
* Feedback: Alert, Drawer, Modal, Message, Notification, Progress, Popconfirm, Spin, Skeleton
* Other: Anchor, BackTop, Divider, LocaleProvider

1. Cách sử dụng

Import css được antd sử dụng (chỉ cần import 1 lần duy nhất ở file index.js):

import 'antd/dist/antd.css';

Import component mà ta muốn sử dụng:

import { Button } from 'antd';

Sử dụng nhưng các component khác

<Button>Default Button</Button>

Đọc thêm về tất cả các component được thư viện cung cấp tại [trang chủ của antd](https://ant.design/components/overview/).

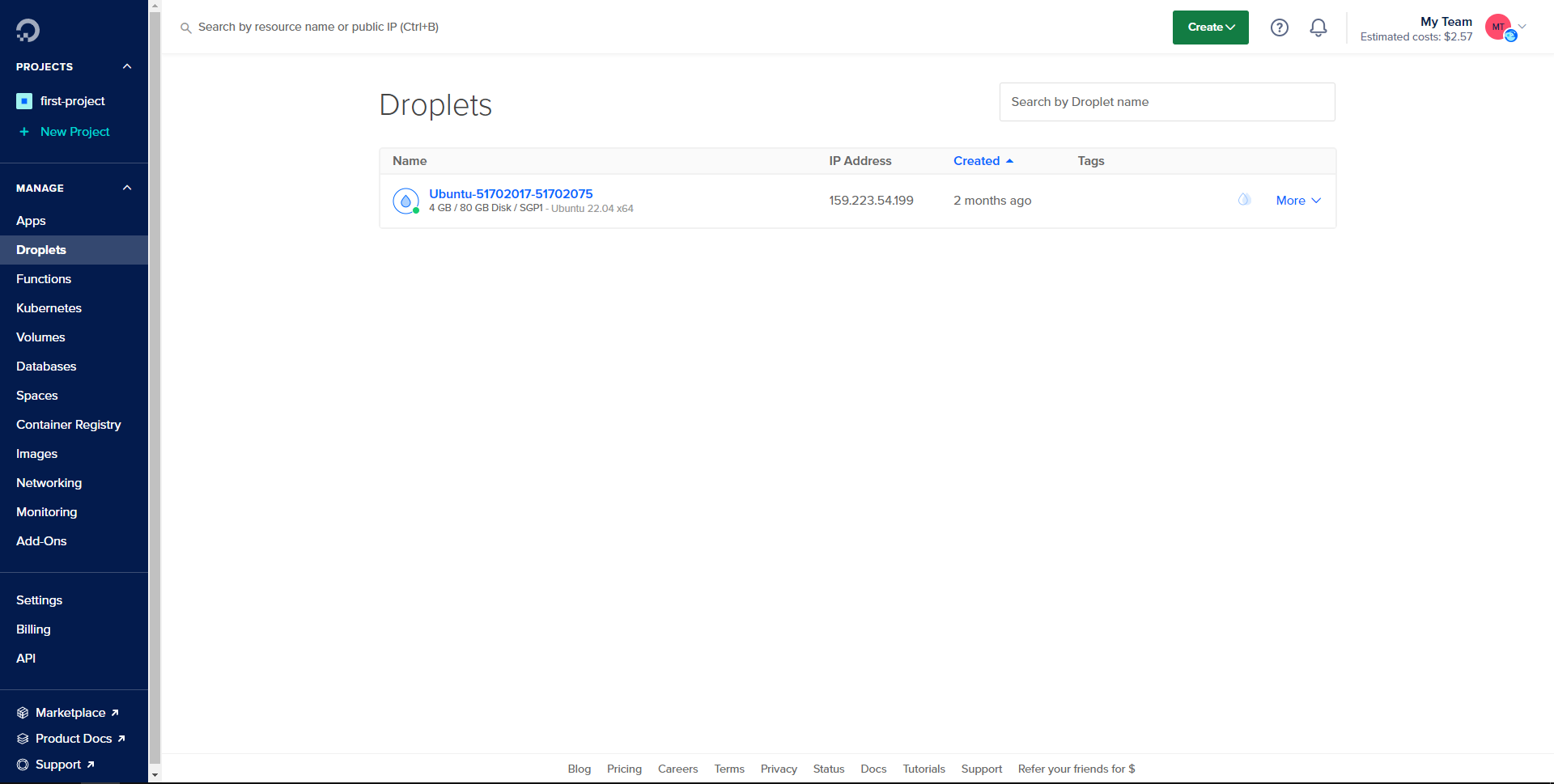
1. Nội dung của chương này

* Giới thiệu về đề tại về xây dựng trang ứng dụng đăng tin tuyển dụng cũng như ứng tuyển dành cho sinh viên trường Đại học Tôn Đức Thắng
* Giới thiệu về các công nghệ sử dụng trong quá trình xây dựng trang ứng dụng này

CHƯƠNG 2 – QUÁ TRÌNH THỰC HIỆN

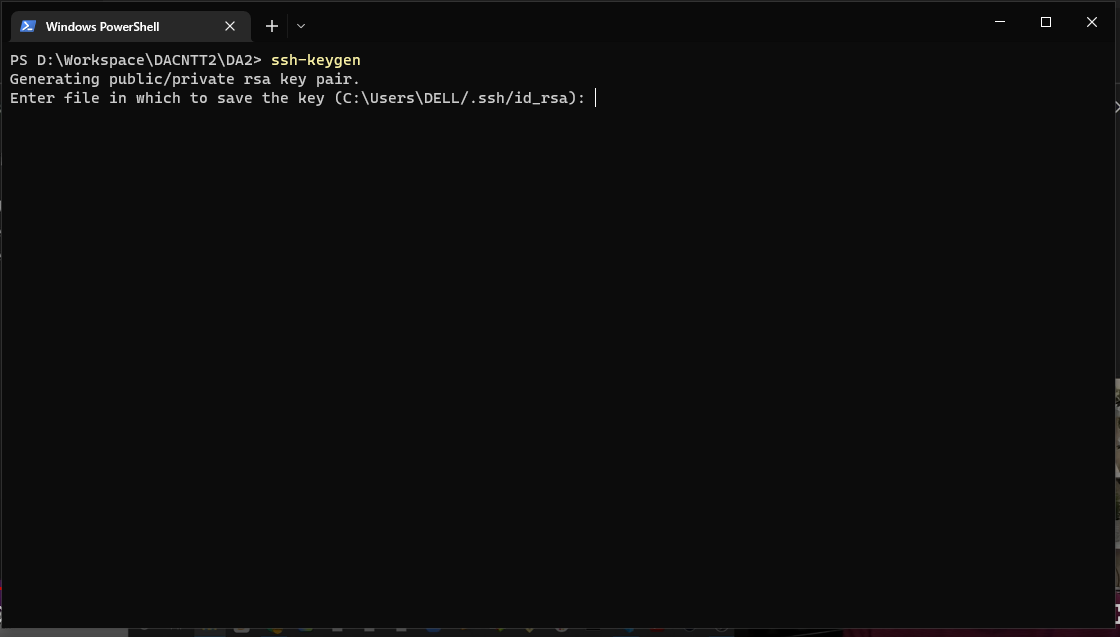
1 Quá trình thiết lập server

* Để đưa source code lên server ta cần một server ở đây chúng em thuê 1 con server chạy hệ điều hành Ubuntu với 4GB ram và 80GB bộ nhớ của Digitalocean với chi phí duy trì hàng tháng
* Quá trình thuê thì sẽ đưa thẻ visa của bản thân vào tài khoản để thuê server và mỗi tháng thì digitalocen sẽ trừ khoản tiền đó



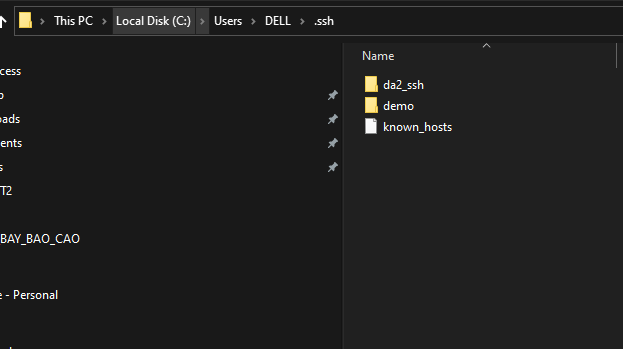
Hình 7 : Hình ảnh thuê server

* Khi đã có server thì bước tiếp theo thì sẽ cấu hình cho việc kết nối với server đó quá một số công cụ như cmd , Window Terminal, …., và các bước như sau:
  + Bước 1 : ta sẽ sử dụng ssh-keygen để tạo ra 1 chuỗi các kí tự ngẫu nhiên được gọi là key để sau khi thiết lập mỗi lần muốn kết nối với server ta chỉ cần gõ lệnh ssh@root + tên server và nhập mật khẩu của server thì lúc này ta sẽ kết nối được với server của mình vừa thuê



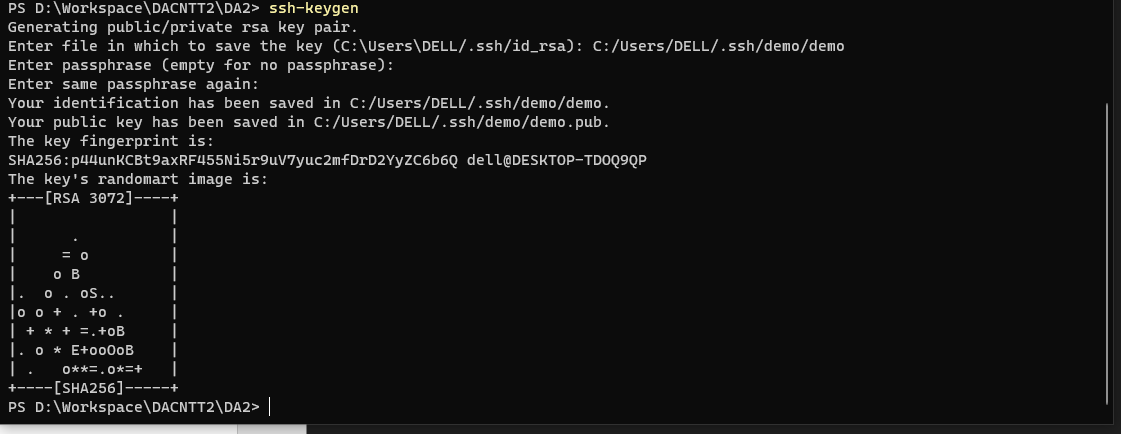
Hình 8 : Gõ ssh-keygen để tạo ra bộ mã cho riêng mình

* + Bước 2 ta vào thư mục C:/Users/(Tên của user)/.ssh và tạo ra 1 thư mục ví dụ ở đây máy cá nhân này có thư mục (demo) để mô phỏng lại quá trình tạo ssh-keygen và da2\_ssh đã tạo trong quá trình thực hiện đề tài này

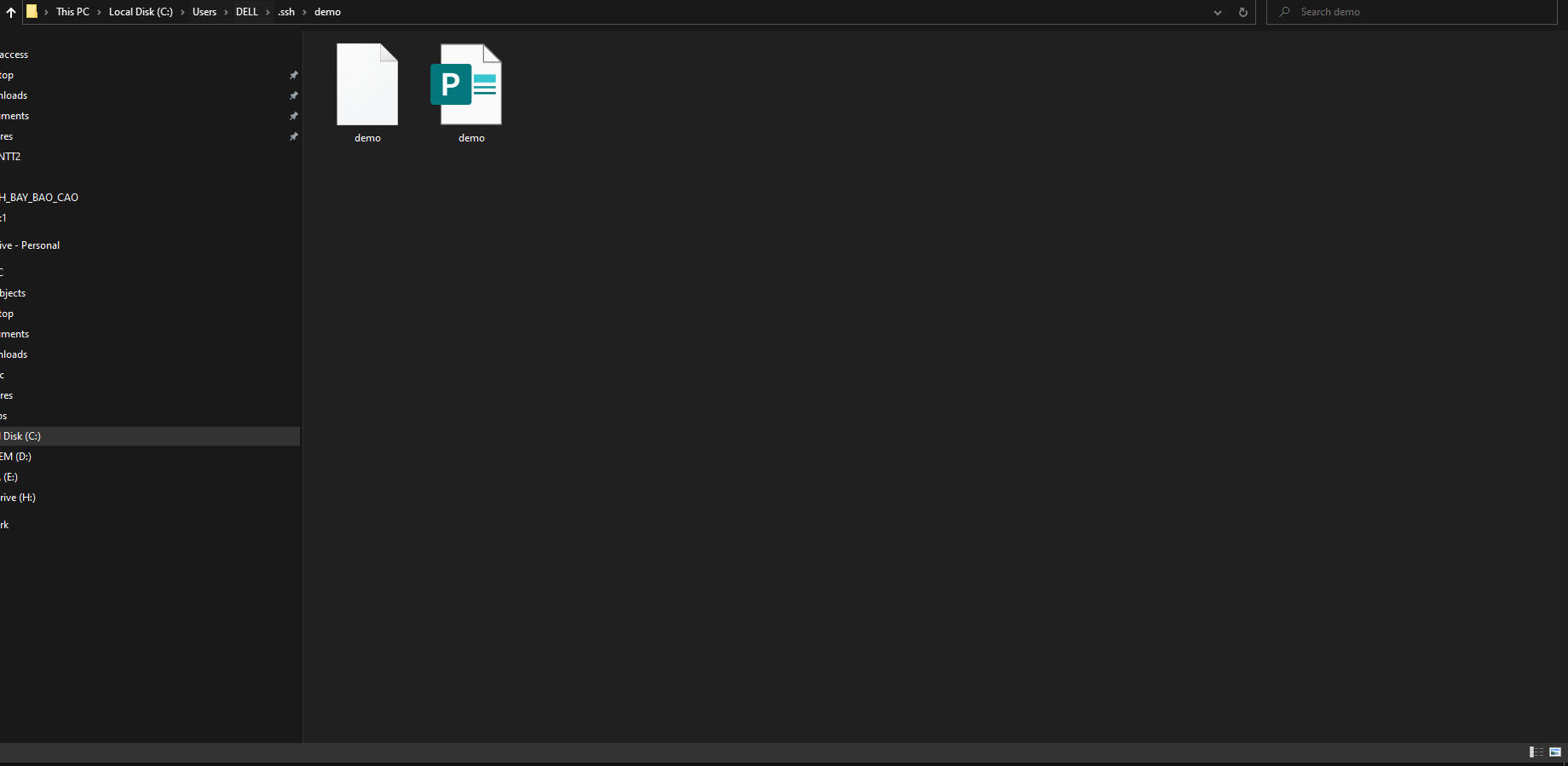


Hình 9 : Thư mục .ssh chứa ssh-keygen

* + Tiếp tục ta ở Window Terminal ta gõ lệnh để với thư mục demo ta gõ lệnh chuyển trỏ nơi tạo file chưa public key và private key

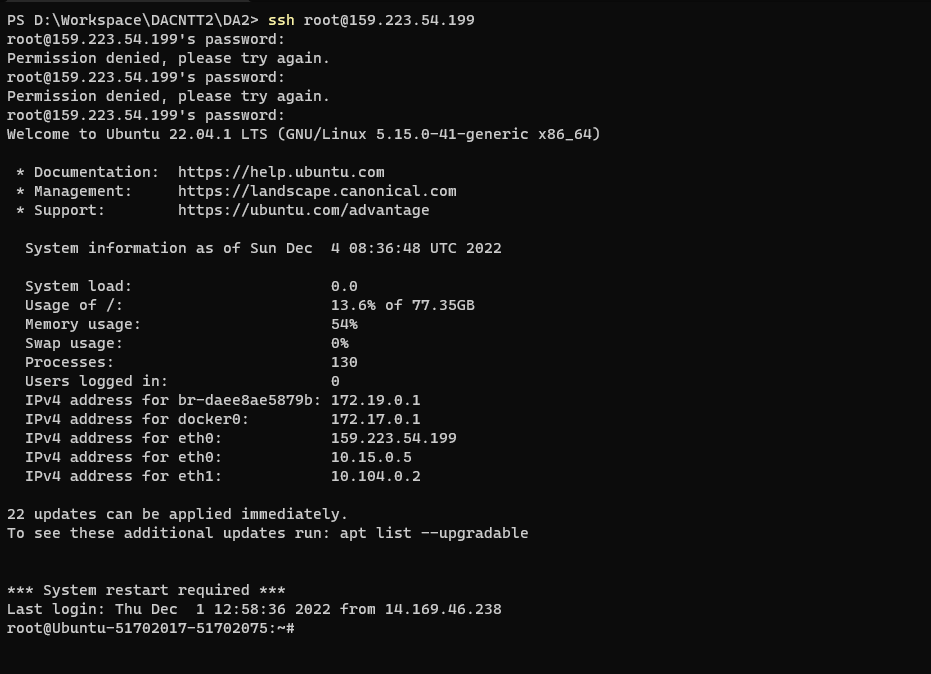


Hình 10 : Hình ảnh tạo ssh key trong thư mục demo



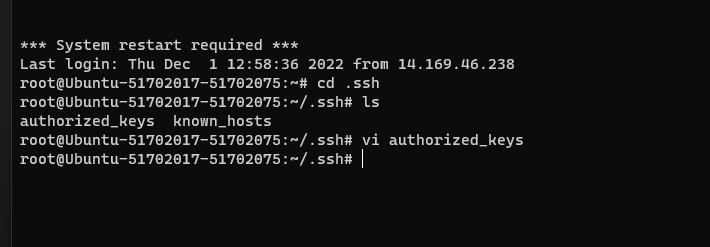
Hình 11 : Hình ảnh thư mục demo chứa ssh key ta vừa tạo với demo

* + Trong thư mục ta vừa tạo thì sẽ có 2 tập tin bào gồm 1 file demo thường và 1 file demo có đuôi là pub tức nghĩa là public key còn file còn lại là secret key
  + Sau khi khởi tạo ssh key ở máy cá nhân ta truy cập với server thông quá ssh bằng dòng lệnh



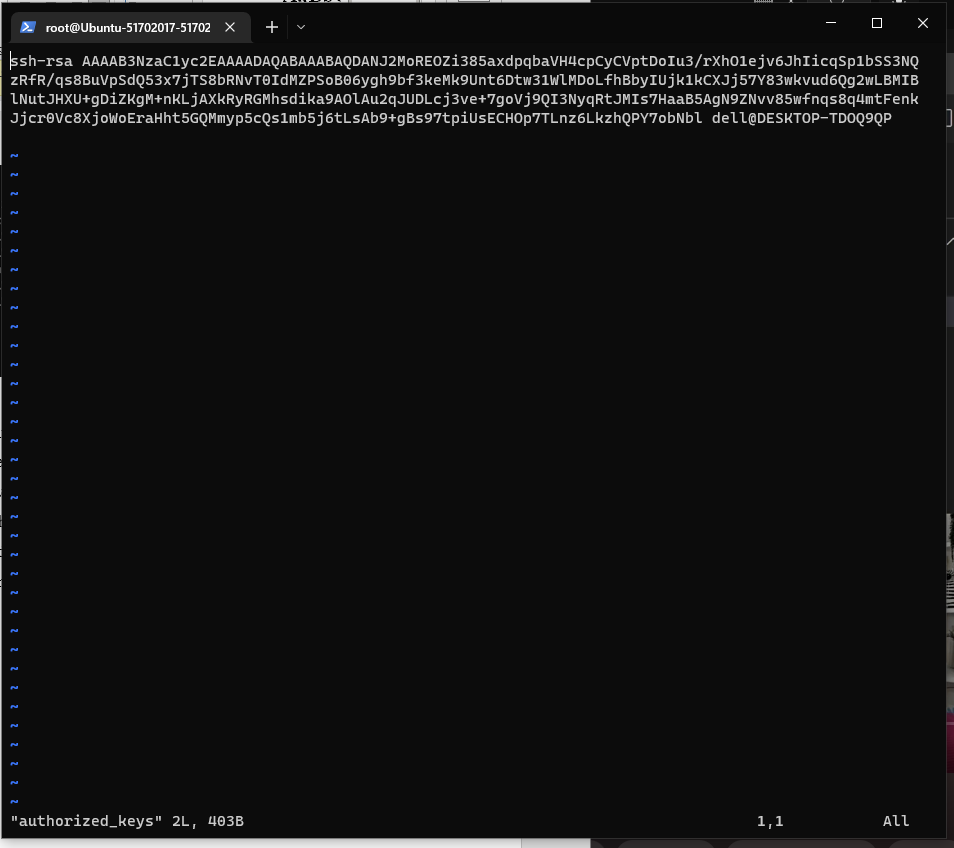
Hình 12 : Kết nối với server thuê được bằng ssh

* + Ta sẽ sử dụng ssh root@(IP) mà nhà cung cấp dịch vụ vps cung cấp cho ta để kết nối và nhập mật khẩu của server để truy cập vào server
  + Khi này ta sẽ có quyền thao tác với server lúc này ta sẽ add ssh được sinh ra ở máy cá nhân



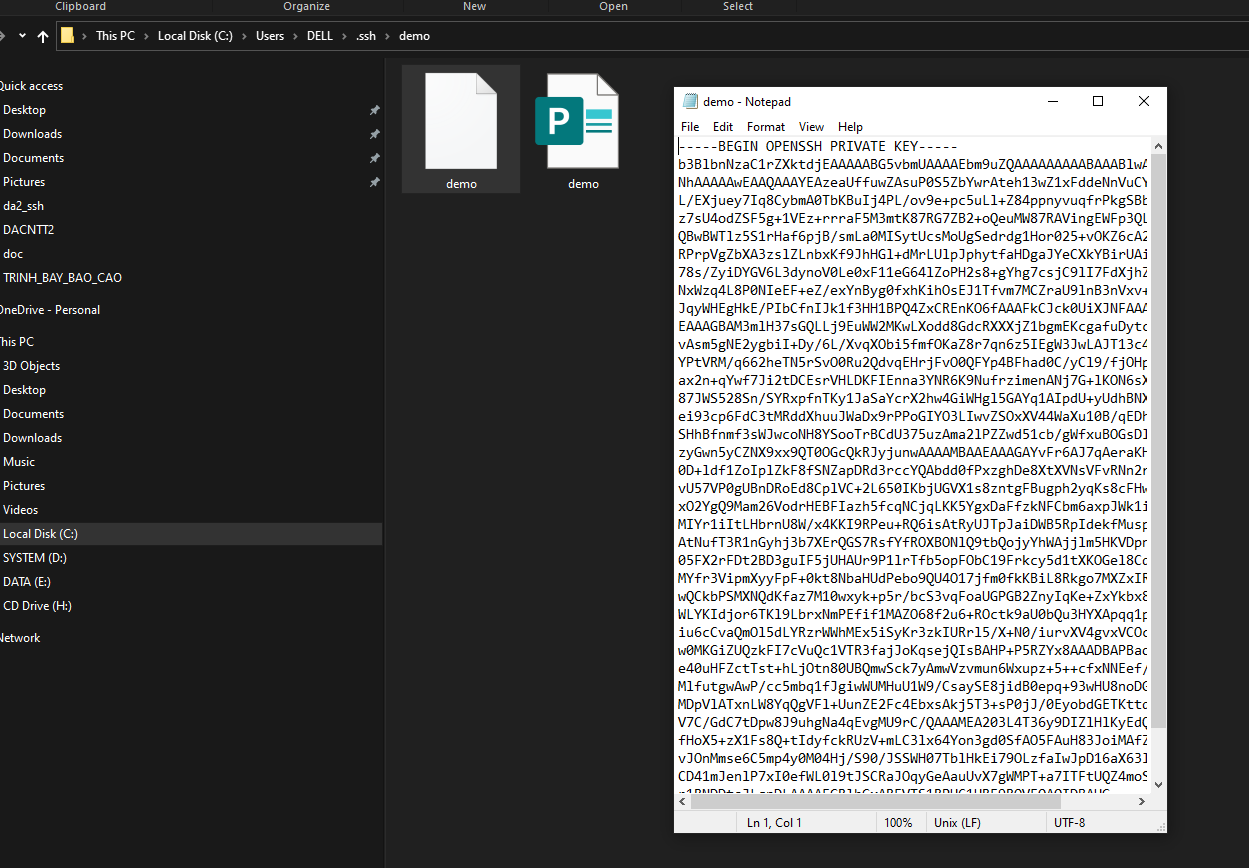
Hình 13 : Thêm ssh key vào máy chủ

* + Khi này ta thực hiện chuyển thư mục bằng cd .ssh và ls để check xem đã có file authorized\_keys hay chưa nên chưa ta sẽ tạo file này bằng sử dụng câu lệnh vừa cd vừa tạo bằng nano “ cd .ssh && sudo nano authorized\_keys “ để chuyển đến thư mục .ssh và có quyền như root sử dụng lệnh viết nano trên tập tin authorized\_keys
  + Ở đây máy chủ đã có tập tin authorized\_keys nên em sẽ dùng vim (vi) để xem nội dung thư mục



Hình 14 : Nội dung tập tin authorized\_keys

* + Nếu là lần đầu thiết lập tập tin này không có gì và ta sẽ thực hiện thao tác nhân nút I trên bàn phím để vào chế độ viết và ta quay lại máy cá nhân thư mục .ssh ta tạo có 2 file public và file thường

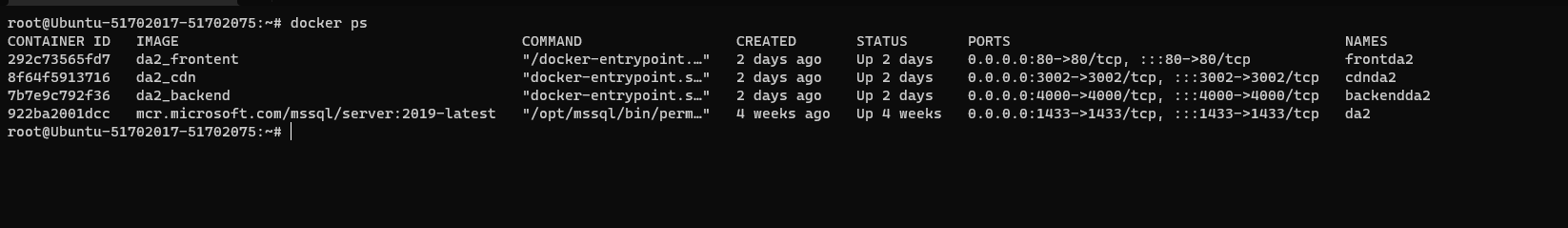


Hình 15 : Ta sẽ copy key trong file private key

* + Và ta sẽ past private key này vào file ở server thì đây là demo sẽ có nội dung khác với key đã được thêm vào server trước đó
  + Sau khi thêm private của máy cá nhân vào server thì ta nhẫn :x để thoát và lưu lại nội dung đã ghi với vim
* Sau khi đã lưu lại private key ta tiến hành update và cài đặt một số thư viện với các dòng lệnh như sau:
  + Sudo apt update : Để Ubuntu sẽ tiến hành việc tải các tài nguyên cần thiết để cấu hình server
  + Sudo apt upgrade : Để nâng cấp phiên bản nếu tài nguyên sau khi update còn quá cũ
  + Sudo apt-get install ca-certificates : Cài đặt tài nguyên này để cho việc chứng thực số cho server này có thể hoạt động trong môi trường mạng
  + Sudo apt-get install curl : Để cài đặt 1 công cụ (tool) có thể cài đặt một số thư viện khác ở đây để tải 1 số thư viện
  + Sudo apt-get install gnupg : Để cài đặt một phần mềm bảo mật cho server
  + Sudo apt-get install lsd-realase : Để cài đặt phiên bản linux tiêu chuẩn nhất giành cho máy chủ nếu phiên bản linux của máy chủ quá cũ
  + Ta sử dụng 1 câu lệnh như sau :

**Curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg –dearmor - /usr/share/keyrings/docker-archive-keyring.gpg** Để thêm khóa mã hóa cho docker (GPG – GNU Private Guard)

* + sudo apt-get install docker-ce docker-ce-cli containerd.io : Để cài đặt các tài nguyên của Docker
* Để sử dụng database mà cụ thể là SQL SERVER (2019) bằng docker thì Microsoft đã có images của sql ở docker hub và ta sẽ thực hiện lệnh docker pull để lấy images sql đó để chạy ở server ở trang chủ của docker : https://hub.docker.com/\_/microsoft-mssql-server
* Ta sẽ thực hiện các lệnh như sau :
* sudo docker pull mcr.microsoft.com/mssql/server:phiên bản-latest thì ở đây máy chủ này đang sử dụng phiên bản 2019 thì sẽ thay phiên bản bằng 2019
* Tiếp theo ta sẽ chạy 1 lệnh nữa để chạy cái images đó thành 1 container trên docker
  + docker run -e “ACCEPT\_EULA =Y” -e “SA\_PASSWORD = MatKhau” – p 1433:1433 –name ten --hostname ten -d https://mcr.microsoft.com/mssql/server:2019-latest
* Để kiểm tra container có hoạt động không ta gõ lệnh docker ps để xem máy chủ có những container nào đang hoạt động



Hình 16 : Hình ảnh thể hiện máy chủ có các container đang hoạt động

**TÀI LIỆU THAM KHẢO**

**Tiếng Việt**

1. Quách Ngọc Ân (1992), “Nhìn lại hai năm phát triển lúa lai”, *Di tuyền học ứng dụng*, 98(1), tr. 10-16.
2. Bộ nông nghiệp & PTNT (1996), *Báo cáo tổng kết 5 năm (1992-1996) phát triển lúa lai,* Hà Nội.
3. Nguyễn Hữu Đống, Đào Thanh Bằng, Lâm Quang Dụ, Phan Đức Trực (1997), *Đột biến –* *Cơ sở lý luận và ứng dụng,* Nhà xuất bản nông nghiệp, Viện khoa học kỹ thuật nông nghiệp Việt Nam, Hà Nội.
4. Nguyễn Thị Gấm (1996), *Phát hiện và đánh giá một số dòng bất dục đực cảm ứng nhiệt* *độ,* Luận văn thạc sĩ khoa học nông nghiệp, Viện khoa học kỹ thuật nông nghiệp Việt Nam, Hà Nội.

……….

1. Võ Thị Kim Huệ (2000), *Nghiên cứu chẩn đoán và điều trị bệnh…,* Luận án Tiến sĩ y khoa, Trường đại học y Hà Nội, Hà Nội.

**Tiếng Anh**

1. Anderson J.E. (1985), The Relative Inefficiency of Quota, The Cheese Case, *American* *Economic Review*, 75(1), pp. 178-90.
2. Borkakati R. P.,Virmani S. S. (1997), Genetics of thermosensitive genic male sterility in Rice, *Euphytica* 88, pp. 1-7.
3. Boulding K.E. (1955), *Economics Analysis*, Hamish Hamilton, London.
4. Burton G. W. (1988), “Cytoplasmic male-sterility in pearl millet (penni-setum glaucum L.)”, *Agronomic Journal* 50, pp. 230-231.
5. Central Statistical Oraganisation (1995), *Statistical Year Book*, Beijing.
6. FAO (1971), *Agricultural Commodity Projections (1970-1980)*, Vol. II. Rome.
7. Institute of Economics (1988), *Analysis of Expenditure Pattern of Urban Households in* *Vietnam,* Departement pf Economics, Economic Research Report, Hanoi.

**PHỤ LỤC**

Phần này bao gồm những nội dung cần thiết nhằm minh họa hoặc hỗ trợ cho nội dung luận văn như số liệu, biểu mẫu, tranh ảnh. . . . nếu sử dụng những câu trả lời cho một *bảng câu hỏi thì bảng câu hỏi mẫu này phải được đưa vào phần Phụ lục ở dạng nguyên bản* đã dùng để điều tra, thăm dò ý kiến; **không được tóm tắt hoặc sửa đổi**. Các tính toán mẫu trình bày tóm tắt trong các biểu mẫu cũng cần nêu trong Phụ lục của luận văn. Phụ lục không được dày hơn phần chính của luận văn