**BÀI BÁO CÁO THỰC TẬP**

GIỚI THIỆU BẢN THÂN

HỌ và TÊN : Lưu Hoàng Lộc

MSSV : 019101007

GIỚI TÍNH : Nam

NGÀY SINH: 08/08/2001

# 

MỤC LỤC

[BÀI BÁO CÁO 3](#_Toc106801713)

[PHẦN1:GIỚI THIỆU VỀ GIT: 3](#_Toc106801714)

[1.1) GIT là gì: 3](#_Toc106801715)

[1.2) Công Dung Của GIT: 4](#_Toc106801716)

[1.3) GIT hoạt động như thế nào: 4](#_Toc106801717)

[1.4) Những Ưu điểm của GIT là: 5](#_Toc106801718)

[1.5) Những Nhược điểm của GIT là: 5](#_Toc106801719)

[1.6) Các thuật ngữ GIT quan trọng là: 5](#_Toc106801720)

[1.7) Các lệnh GIT cơ bản là: 5](#_Toc106801721)

[1.8) So sánh GIT và các phần mềm khác: 8](#_Toc106801722)

[1.9) Khi sử dụng GIT: 8](#_Toc106801723)

[2.0) Khi không sử dụng GIT: 8](#_Toc106801724)

[2.1) Cách sử dụng GIT: 9](#_Toc106801725)

[PHẦN2:GIỚI Thiệu Về Microservices: 20](#_Toc106801726)

[1.1) Microservices là gi: 20](#_Toc106801727)

[1.2) Kiến trúc của Microservices là gì: 20](#_Toc106801728)

[**a Kiến trúc:** 20](#_Toc106801729)

[**b Tính chất khối Monolithic ảnh hưởng đến cấu trúc Microservices:** 21](#_Toc106801730)

[1.3) Các ưu điểm và nhược điểm nổi bật của Microservices là gì: 21](#_Toc106801731)

[**a) Ưu điểm nổi trội của Microservices là gì:** 21](#_Toc106801732)

[**b) Nhược điểm còn tồn tại:** 22](#_Toc106801733)

[1.4) Các lợi ích mang lại của Microservices là gì: 22](#_Toc106801734)

[1.5) Bạn nên sử dụng kiến trúc Microservices khi nào là hợp lý: 23](#_Toc106801735)

[1.6) Các vấn đề nên lưu ý khi thiết kế Microservices: 23](#_Toc106801736)

[**Hiểu sai về Microservices:** 23](#_Toc106801737)

[**Những điều cần phải tuân thủ:** 24](#_Toc106801738)

[1.7) So sánh giữa kiến trúc Microservice và Monolithic: 24](#_Toc106801739)

[PHẦN 3 : SPRING BOOT: 27](#_Toc106801740)

[**1.1)Spring Boot là gì:** 27](#_Toc106801741)

[**1.2)Những đặc điểm nổi bật của Spring Boot:** 27](#_Toc106801742)

[**1.3)Vì sao nên học Spring Boot:** 28](#_Toc106801743)

[**1.4)Tại sao spring boot áp dung trong Microservices:** 29](#_Toc106801744)

[PHẦN 4 : JHipster 29](#_Toc106801745)

[1)JHipster là gì: 29](#_Toc106801746)

[1.1)Server side: 30](#_Toc106801747)

[1.2)Client side: 30](#_Toc106801748)

[1.3)Deployment: 30](#_Toc106801749)

[2)Cách khởi tạo Monolithic và Microservices 30](#_Toc106801750)

[a) yêu cầu cài đặt: 30](#_Toc106801751)

[b) kiểm tra xem đã cài java, node.js, Git,npm : 30](#_Toc106801752)

[3)Tại sao lựa chọn JHipster: 36](#_Toc106801753)

[4)Tác dụng JHipster: 36](#_Toc106801754)

# BÀI BÁO CÁO

## PHẦN1:GIỚI THIỆU VỀ GIT:

### 1.1) GIT là gì:

Là một hệ thống quản lý phiên bản phân tán(Distributed Version Control System-DVCS)ra đời vào năm 2005 và hiện được dùng rất phổ biến.So với các hệ thống quản lý phiên bản tập trung khi tất cả các mã nguồn và lịch sử thay đổi chỉ được lưu một nơi là máy chủ thì trong hệ thống phân tán, các máy không chỉ "check out" phiên bản mới nhất của các tập tin mà là sao chép(mirror)toàn bộ kho mã nguồn(repository).Như vậy, nếu như máy chủ ngừng hoạt động , thì bạn hoàn toàn có thể lấy kho chứa từ bắt kì máy khách nào để sao chép ngược trở lại máy chủ để khôi phục lại toàn bộ hệ thống.Mỗi checkout thực sự là một bản sao đầy đủ của tất cả dữ liệu của kho chưá từ máy chủ.

## 1.2) Công Dung Của GIT:

a. Lưu lại lịch sử các version của bất kỳ thay đổi nào của dự án. Giúp xem lại các sự thay đổi hoặc khôi phục (revert) lại sau này.

b. Việc chia sẻ code trở nên dễ dàng hơn, lập trình viên có thể để public cho bất kỳ ai, hoặc private chỉ cho một số người có thẩm quyền có thể truy cập và lấy code về.

c. Vốn là một VCS nên Git cũng ghi nhớ lại toàn bộ lịch sử thay đổi của source code trong dự án. Lập trình sửa file, thêm dòng code tại đâu, xóa dòng code ở hàng nào…đều được Git ghi nhận và lưu trữ lại.

## 1.3) GIT hoạt động như thế nào:

Sự khác biệt chính giữa Git và bất kỳ VCS nào khác (bao gồm Subversion…) là cách Git nghĩ về dữ liệu của nó.

Về mặt khái niệm, hầu hết các hệ thống khác đều lưu trữ thông tin dưới dạng danh sách các thay đổi dựa trên file. Các hệ thống này (CVS, Subversion, Perforce, Bazaar, v.v.) coi thông tin chúng lưu giữ dưới dạng một tập hợp các file và những thay đổi được thực hiện đối với mỗi file theo thời gian.

[](https://user-images.githubusercontent.com/107382675/173315605-66111018-514a-482e-ba2a-d06e81973cad.png)

## 1.4) Những Ưu điểm của GIT là:

* Dễ sử dụng, thao tác nhanh, gọn, lẹ và rất an toàn.
* Sẽ dàng kết hợp các phân nhánh (branch), có thể giúp quy trình làm việc code theo nhóm đơn giản hơn rất nhiều.
* Chỉ cần clone mã nguồn từ kho chứa hoặc clone một phiên bản thay đổi nào đó từ kho chứa, hoặc một nhánh nào đó từ kho chứa là bạn có thể làm việc ở mọi lúc mọi nơi.
* Deployment sản phẩm của bạn một cách không thể nào dễ dàng hơn.

## 1.5) Những Nhược điểm của GIT là:

* Thuật toán SHA1 sự va chạm giá trị băm làm cho các pc thông thường làm hư hỏng một kho git.
* Sử dụng GIT trên hệ điều hành Microsoft Windows hơi phức tạp.
* Các tập tin không liên quan mà luôn luôn bị thay đổi, Git có thể chịu thiệt thòi hơn các hệ thống khác bởi vì các tập tin không được giữ dấu viết từng cái riêng lẻ.

## 1.6) Các thuật ngữ GIT quan trọng là:

* Branch - Commit - Checkout - Fetch - Fork - Head - Index - Master - Merge - Origin - Pull - Push - Rebase - Remote - Repository - Stash - Tags - Upstream

## 1.7) Các lệnh GIT cơ bản là:

* a)git config

Tác dụng : Để set user name và email của bạn trong main configuration file.

Cách xài : Để kiểm tra tên và kiểu email trong cấu hình dùng git config -- global user.name và git config -- global user.email. Để set email hoặc tên mới git config -- global user.name = “Hải Nguyễn” và git config -- global user.email = “[hainguyen@gmail.com](mailto:hainguyen@gmail.com)”

* b)git init

Tác dụng : Khởi tạo 1 git repository 1 project mới hoặc đã có.

Cách xài: git init trong thư mục gốc của dự án.

* c)git clone

Tác dụng: Copy 1 git repository từ remote source.

Cách xài: git clone <:clone git url:>

* d)git status

Tác dụng: Để check trạng thái của những file bạn đã thay đổi trong thư mục làm việc. VD: Tất cả các thay đổi cuối cùng từ lần commit cuối cùng.

Cách xài: git status trong thư mục làm việc.

* e)git add

Tác dụng: Thêm thay đổi đến stage/index trong thư mục làm việc.

Cách xài: git add

* f)git commit

Tác dụng: commit nghĩa là một action để Git lưu lại một snapshot của các sự thay đổi trong thư mục làm việc. Và các tập tin, thư mục được thay đổi đã phải nằm trong Staging Area. Mỗi lần commit nó sẽ được lưu lại lịch sử chỉnh sửa của code kèm theo tên và địa chỉ email của người commit. Ngoài ra trong Git bạn cũng có thể khôi phục lại tập tin trong lịch sử commit của nó để chia cho một branch khác, vì vậy bạn sẽ dễ dàng khôi phục lại các thay đổi trước đó.

Cách dùng: git commit -m ”Đây là message, bạn dùng để note những thay đổi để sau này dễ dò lại”

* g)git push/ git pull

Tác dụng: Push hoặc Pull các thay đổi đến remote. Nếu bạn đã added và committed các thay đổi và bạn muốn đẩy nó lên hoặc remote của bạn đã update và bạn apply tất cả thay đổi đó trên code của mình.

Cách dùng: git pull <:remote:> <:branch:> and git push <:remote:> <:branch:>

* h)git branch

Tác dụng: liệt kê tất cả các branch (nhánh).

Cách dùng: git branch hoặc git branch -a

* i)git checkout

Tác dụng: Chuyển sang branch khác

Cách dùng: git checkout <: branch:> hoặc \*\* \_ git checkout -b <: branch:> nếu bạn muốn tạo và chuyển sang một chi nhánh mới.

* k)git stash

Tác dụng: Lưu thay đổi mà bạn không muốn commit ngay lập tức.

Cách dùng: git stash trong thư mục làm việc của bạn.

* l)git merge

Tác dụng: Merge 2 branch lại với nahu.

Cách dùng: Chuyển tới branch bạn muốn merge rồi dùng git merge <:branch\_ban\_muon\_merge:>

* m)git reset

Tác dụng: Bạn đã đưa một tập tin nào đó vào Staging Area nhưng bây giờ bạn muốn loại bỏ nó ra khỏi đây để không phải bị commit theo.

Cách dùng: git reset HEAD tên\_file

* n)git remote

Tác dụng: Để check remote/source bạn có hoặc add thêm remote

Cách dùng: git remote để kiểm tra và liệt kê. Và git remote add <: remote\_url:> để thêm.

* o) git add

Tác dụng: Để đưa một tập tin vào Staging Area

Cách dùng: git add tên\_file hoặc muốn thêm hết file của thư mục thì git add all

## 1.8) So sánh GIT và các phần mềm khác:

Đây là điểm khác biệt quan trọng giữa Git và gần như tất cả các VCS khác. Nó khiến Git phải xem xét lại hầu hết mọi khía cạnh của kiểm soát phiên bản mà hầu hết các hệ thống khác đã sao chép từ thế hệ trước. Điều này làm cho Git giống như một hệ thống tệp nhỏ với một số công cụ cực kỳ mạnh mẽ được xây dựng trên nó, thay vì chỉ đơn giản là một VCS.

## 1.9) Khi sử dụng GIT:

* Lưu lại được các phiên bản khác nhau của mã nguồn dự án phần mềm.
* Khôi phục lại mã nguồn từ 1 phiên bản bất kỳ.
* Dễ dàng so sánh của các phiên bản.
* Phát hiện được ai đã sửa phần nào làm phát sinh lỗi.
* Khôi phục lại tập tin bị mất.
* Dễ dàng thử nghiệm, mở rộng tính năng của dự án mà không làm ảnh hưởng đến phiên bản chính (master branch).
* Giúp phối hợp thực hiện dự án trong một nhóm 1 cách hiệu quả.

## 2.0) Khi không sử dụng GIT:

* Không khôi phục được mã code lỡ xóa gây ra lỗi.
* Không quản lý được những người đã sửa code làm phát sinh lỗi
* Không khôi phục được tập tin đã bị mất.
* Khả năng phối hợp dự án trong nhóm bị hạn chế.

## 2.1) Cách sử dụng GIT:

a. Tạo tài khoản GitHub

[](https://user-images.githubusercontent.com/107382675/173311849-7d286b02-3547-407d-a44a-b53d2495219f.png)

Để tạo tài khoản của bạn, bạn cần truy cập trang web của GitHub và điền vào mẫu đăng ký.

b. Cài đặt Git

Bây giờ chúng ta cần cài đặt các công cụ của Git trên máy tính. Chúng ta sẽ sử dụng CLI để liên lạc với GitHub.

Đối với Ubuntu:

Đầu tiên, cập nhật các gói của bạn.

* sudo apt update

Tiếp theo, cài đặt Git và GitHub với apt-get

* sudo apt-get install git

Cuối cùng, xác minh rằng Git đã được cài đặt đúng

* git –version

Chạy các lệnh sau với thông tin của bạn để đặt tên người dùng và email mặc định khi bạn sẽ lưu công việc của mình.

* git config –global user.name “MV Thanoshan”
* git config –global user.email “[example@mail.com](mailto:example@mail.com)”

c. Sử dụng Git theo hai cách

\_Cách thứ 1: Tạo kho lưu trữ, sao chép nó vào pc của bạn và làm việc với nó.

Tạo một kho lưu trữ mới bằng cách nhấp vào nút Kho lưu trữ mới của Wikipedia trên trang web GitHub.

[](https://user-images.githubusercontent.com/107382675/173312121-da4ea5a0-24ff-43d0-9941-954f247d446d.png)

Chọn tên cho kho lưu trữ đầu tiên của bạn, thêm một mô tả nhỏ, đánh dấu vào ô ‘Khởi tạo kho lưu trữ này với README’ và nhấp vào nút Tạo kho lưu trữ.

[](https://user-images.githubusercontent.com/107382675/173312279-72bcd828-7ad6-4e1d-a339-b8bf801a04d0.png)

Kho GitHub đầu tiên của bạn được tạo.

Nhiệm vụ đầu tiên của bạn là lấy một bản sao của kho lưu trữ vào máy tính của mình. Để làm được điều đó, bạn cần phải sao chép lại kho lưu trữ trên máy tính của bạn.

Để sao chép một kho lưu trữ có nghĩa là bạn đang lấy một kho lưu trữ trên máy chủ và sao chép nó vào máy tính của bạn – giống như tải xuống. Trên trang kho lưu trữ, bạn cần lấy địa chỉ HTTPS.

[](https://user-images.githubusercontent.com/107382675/173312470-1204e5b4-324c-4c3c-9177-e0bb07166734.png)

Khi bạn có địa chỉ của kho lưu trữ, bạn cần sử dụng terminal của mình. Sử dụng lệnh sau trên terminal của bạn. Khi bạn đã sẵn sàng, bạn có thể nhập lệnh này:

* git clone [HTTPS ADDRESS]

Lệnh này sẽ tạo một bản sao cục bộ của kho lưu trữ tại địa chỉ đã cho.

[](https://user-images.githubusercontent.com/107382675/173314285-4c3091bc-4070-41d4-95ef-7be64fd44910.png)

Thông báo đầu ra của lệnh git clone bản sao

Bây giờ, kho lưu trữ của bạn là trên máy tính của bạn. Bạn cần di chuyển trong nó bằng lệnh sau.

* cd [NAME OF REPOSITORY]

Như bạn có thể thấy trong hình trên, tên kho lưu trữ của tôi là tên My- GitHub-Project, và lệnh này đã đưa tôi đến thư mục cụ thể đó.

\*LƯU Ý: Khi bạn sao chép, Git sẽ tạo một kho lưu trữ trên máy tính của bạn. Nếu bạn muốn, bạn có thể truy cập dự án của mình bằng giao diện người dùng máy tính thay vì sử dụng lệnh ‘cd’ trên terminal.

Bây giờ, trong thư mục đó, chúng ta có thể tạo các tệp, làm việc với chúng và lưu chúng cục bộ. Để lưu chúng ở một nơi xa xôi – như GitHub – chúng ta đã thực hiện một quy trình gọi là commit. Để làm điều này, quay trở lại terminal của bạn. Nếu bạn đã đóng nó, như tôi đã nói trước đây, hãy sử dụng lệnh ‘cd’.

* cd [NAME OF REPOSITORY]

Bây giờ, trong terminal, bạn đang ở trong thư mục kho lưu trữ của bạn. Có 4 bước trong một commit: ‘status’, ‘add’, ‘commit’ và ‘push’. Tất cả các bước sau đây phải được thực hiện trong dự án của bạn. Chúng ta hãy đi qua từng cái một.

‘status’: Điều đầu tiên bạn cần làm là kiểm tra các tập tin bạn đã sửa đổi. Để làm điều này, bạn có thể gõ lệnh sau để làm cho một danh sách các thay đổi xuất hiện.

* git status

[](https://user-images.githubusercontent.com/107382675/173314394-3f9665ca-eaa4-4a36-b723-1434985bc2f7.png)

‘add’: Với sự trợ giúp của danh sách thay đổi, bạn có thể thêm tất cả các tệp bạn muốn tải lên bằng lệnh sau,

* git add [FILENAME] [FILENAME] […]

Trong trường hợp của chúng ta, chúng ta sẽ thêm một tệp HTML đơn giản.

* git add sample.html

‘commit’: Bây giờ chúng ta đã thêm các tệp mà chúng ta chọn, chúng ta cần viết một thông điệp để giải thích những gì chúng ta đã làm. Thông báo này có thể hữu ích sau này nếu chúng ta muốn kiểm tra lịch sử thay đổi. Dưới đây là một ví dụ về những gì chúng ta có thể đặt trong trường hợp của chúng ta.

* git commit -m “Added sample HTML file that contain basic syntax”

[](https://user-images.githubusercontent.com/107382675/173314466-a25831b8-6ffd-4ac6-b2f4-c281cd16c8dc.png)

‘push’: Để làm điều đó, chúng ta phải ‘đẩy’ các tệp của mình lên Remote. Remote là một bản sao trùng lặp của kho lưu trữ ở một nơi khác trên máy chủ từ xa. Để làm điều này, chúng ta phải biết tên của Remote (Chủ yếu là từ xa được đặt tên gốc). Để tìm ra tên đó, gõ lệnh sau.

* git remote

Như bạn có thể thấy trong hình trên, nó nói rằng tên từ là origin. Bây giờ chúng ta có thể ‘đẩy’ công việc của mình một cách an toàn bằng lệnh sau.

* git push origin master

Bây giờ, nếu chúng ta truy cập kho lưu trữ của mình trên trang web GitHub, chúng ta có thể thấy tệp sample.html mà chúng ta đã đẩy đến từ xa – GitHub!

[](https://user-images.githubusercontent.com/107382675/173312470-1204e5b4-324c-4c3c-9177-e0bb07166734.png)

\*LƯU Ý : Đôi khi, khi bạn đang sử dụng các lệnh Git trong terminal, nó có thể dẫn bạn đến trình soạn thảo văn bản VIM (trình soạn thảo văn bản dựa trên CLI). Vì vậy, để thoát khỏi nó, bạn phải gõ và ENTER.

Mô tả cách pull và push làm việc:

[](https://user-images.githubusercontent.com/107382675/173314720-36352134-acfd-4d8d-bb88-a9086a1a4474.png)

Pull là hành động nhận từ GitHub.

Push là hành động gửi đến GitHub.

\_Cách thứ 2: Làm việc trên dự án của bạn cục bộ sau đó tạo kho lưu trữ trên github và đẩy nó vào remote.

Loại 2 cho phép bạn tạo một kho lưu trữ mới từ một thư mục hiện có trên máy tính và gửi nó đến GitHub. Trong rất nhiều trường hợp, bạn có thể đã thực sự tạo ra một cái gì đó trên máy tính mà bạn muốn đột nhiên biến thành một kho lưu trữ trên GitHub.

khi thực hiện bất kỳ lệnh Git nào, chúng ta phải đảm bảo rằng chúng ta đang ở đúng thư mục trong terminal.

Theo mặc định, bất kỳ thư mục nào trên máy tính không phải là kho lưu trữ Git – nhưng chúng ta có thể biến nó thành kho lưu trữ Git bằng cách thực hiện lệnh sau trong terminal.

* git init

[](https://user-images.githubusercontent.com/107382675/173314840-f64fce8b-75cc-42bb-bd9f-3e40889e85bd.png)

Sau khi chuyển đổi thư mục của chúng ta sang kho lưu trữ Git, điều đầu tiên chúng ta cần làm là kiểm tra các tệp chúng ta có bằng cách sử dụng lệnh sau.

* git status

[](https://user-images.githubusercontent.com/107382675/173314953-09ded956-8592-4059-b9f2-1e3d3c79e9f0.png)

Vì vậy, có hai tập tin trong thư mục đó mà chúng ta cần để thêm vào Repo của chúng ta.

* git add [FILENAME] [FILENAME] […]

\*LƯU Ý: Để thêm vào tất cả các tệp trong Kho lưu trữ của chúng ta, chúng ta có thể sử dụng lệnh sau:

* git add .

Sau khi khu vực tổ chức (quá trình thêm) hoàn tất, chúng ta có thể kiểm tra xem các tệp có được thêm thành công hay không bằng cách thực hiện git status

Nếu những tệp cụ thể đó có màu xanh lục như hình dưới đây, bạn đã hoàn thành công việc của mình!

[](https://user-images.githubusercontent.com/107382675/173315080-cc952a08-b46f-497c-9d76-614af3f94fab.png)

Sau đó, chúng ta phải commit với một mô tả trong đó.

* git commit -m “Adding web Survey form”

Nếu kho lưu trữ của tôi bắt đầu trên GitHub và tôi đã đưa nó xuống máy tính của mình, một Remote đã được gắn vào nó (Loại 1). Nhưng nếu tôi đang khởi động kho lưu trữ của mình trên máy tính thì nó không có Remote, vì vậy tôi cần thêm Remote đó (Loại 2).

Vì vậy, để thêm Remote, chúng ta phải vào GitHub trước. Tạo một kho lưu trữ mới và đặt tên cho nó bất cứ điều gì bạn muốn lưu trữ trong GitHub. Sau đó nhấp vào nút Tạo kho lưu trữ.

\*LƯU Ý: Trong Loại 2, Vui lòng không khởi tạo kho lưu trữ với tệp README khi tạo kho lưu trữ mới trên trang web GitHub.

[](https://user-images.githubusercontent.com/107382675/173312279-72bcd828-7ad6-4e1d-a339-b8bf801a04d0.png)

Sau khi nhấp vào nút Tạo kho lưu trữ, bạn sẽ tìm thấy dạng trang web.

[](https://user-images.githubusercontent.com/107382675/173313680-0926ac3f-ab44-43cd-a21c-83769318aefb.png)

Sao chép địa chỉ HTTPS. Bây giờ chúng ta sẽ tạo Remote cho kho lưu trữ của mình.

* git remote add origin [HTTPS ADDRESS]

Sau khi thực hiện lệnh này, chúng ta có thể kiểm tra xem chúng ta đã thêm thành công Remote hay chưa bằng lệnh sau

* git remote

Và nếu nó xuất ra Origin thì bạn đã thêm Remote vào dự án của mình.

\*LƯU Ý: Chỉ cần nhớ rằng chúng ta có thể nêu bất kỳ tên nào cho Remote bằng cách thay đổi tên xuất xứ. Ví dụ:

* git remote add [REMOTE NAME] [HTTPS ADDRESS]

Bây giờ, chúng ta có thể đẩy dự án của mình lên GitHub mà không gặp vấn đề gì!

* git push origin master

Sau khi hoàn thành từng bước một, nếu bạn truy cập GitHub, bạn có thể tìm thấy kho lưu trữ của mình với các tệp!

[](https://user-images.githubusercontent.com/107382675/173312470-1204e5b4-324c-4c3c-9177-e0bb07166734.png)

**PHẦN2:GIỚI Thiệu Về Microservices:**

**1.1) Microservices là gi:**

* Microservices là tên gọi của các dịch vụ nhỏ thuộc dạng tách biệt đại diện cho 1 phần nhỏ tương ứng bên trong các Business domain của lập trình viên. Với kiến thức Monolithic thì bạn sẽ sở hữu một server lớn với khả năng chịu mọi trách nhiệm giải quyết hầu hết các requests. Và việc này sẽ gây ra khá nhiều khó khăn trên các phương tiện đối với tất cả requests
* Chính vì vậy, Microservices được xem như giải pháp có thể cân bằng được tất cả các traffic dựa theo yêu cầu của doanh nghiệp. Và nếu như bạn đang nhận một lượng lớn các thanh toán thì hầu hết bạn sẽ có thể scale up thiết bị thanh toán và giữ cho các dịch vụ nằm ở mức sử dụng 1 lượng nhỏ hơn so với các services

**1.2) Kiến trúc của Microservices là gì:**

**a Kiến trúc:**

Bên trong kiến trúc của Microservices thì các services sẽ tồn tại độc lập nhau về xử lý, lưu trữ và cả request. Và cấu trúc cụ thể của nó sẽ như hình sau:

Ví dụ như sau: nếu như bạn cần xây dựng một hệ thống để bán hàng dựa trên Microservices thì giả giả sử đơn hàng sẽ cần tối thiểu 4 service với database độc lập như sau:

* Employee service (sử dụng table tc-employee)
* Store service (sử dụng table tc-store)
* Inventory service (sử dụng table tc-warehouse)
* Order service (sử dụng table tc-order)

Và trong số đó, thì table tc-order sẽ chỉ chứa khóa ngoại chính là ID của nhân viên tại cửa hàng và ID của kho theo đúng tinh thần thiết kế chuẩn hóa database.

**b Tính chất khối Monolithic ảnh hưởng đến cấu trúc Microservices:**

* Được thiết kế, phát triển và triển khai dựa theo một khối duy nhất.
* Các ứng dụng monolithic rất phức tạp và nó sẽ gây ra nhiều khó khăn cho quá trình nâng cấp, bảo trì hoặc thêm các tính năng mới.
* Thường rất khó để có thể áp dụng triển khai dựa theo kiểu agile.
* Bạn cần phải triển khai lại toàn bộ một hệ thống mặc dù chỉ phải cập nhật hoặc nâng cấp một phần duy nhất.
* Mở rộng được các khối ứng dụng nếu như gặp khó khăn thì sẽ có các yêu cầu về những tài nguyên khác nhau.
* Một service thường không có độ ổn định nên có thể làm sập cả hệ thống.
* Khó đổi mới: Bởi vì ứng dụng monolithic cần phải sử dụng chung một công nghệ vì vậy nó rất khó có thể thay đổi hoặc áp dụng thêm các công nghệ mới.

Các tính chất giới hạn này của kiến trúc Monolithic dẫn đến sự phát triển của kiến trúc Microservices.

*Microservice có vai trò quan trọng với một Developer*

**1.3) Các ưu điểm và nhược điểm nổi bật của Microservices là gì:**

**a) Ưu điểm nổi trội của Microservices là gì:**

* Microservices cho phép dễ dàng continuous delivery và deployment các ứng dụng lớn và phức tạp hơn.
* Có thể cải thiện khả năng bảo trì: bởi vì các service tương đối nhỏ nên dễ hiểu và dễ thay đổi hơn.
* Có khả năng testing dễ dàng: nhờ các services nhỏ.
* Có thể triển khai tốt hơn: các services thường rất dễ cho việc triển khai độc lập.
* Cho phép các services được phát triển nhanh chóng bởi những team khác nhau. Khi đó, mỗi team đều sẽ được phát triển và thử nghiệm để triển khai cũng như mở rộng được quy mô của dịch vụ của mình một cách độc lập nhất với tất cả các team.
* Nếu như có lỗi xảy ra trong một service thì chỉ có service đó bị ảnh hưởng và các service khác sẽ thực hiện xử lý các yêu cầu cần thiết. Trong khi đó, thì mỗi một thành phần nếu như hoạt động sai của kiến trúc một khối thì nó sẽ làm ảnh hưởng đến toàn bộ hệ thống.
* Lập trình viên có thể thay đổi dễ dàng bằng cách sử dụng công nghệ mới khi triển khai các service. Tương tự như khi có thay đổi lớn thì các service đều có thể thực hiện và bạn dễ dàng thay đổi được công nghệ hơn.

**b) Nhược điểm còn tồn tại:**

* Bởi vì các nhà phát triển thường xuyên phải đối phó với sự phức tạp khi tạo ra một hệ thống phân tán.
* Cần phải implement việc communication giữa các inter-services.
* Handle partial failure rất phức tạp bởi vì luồng xử lý cần phải đi qua nhiều service khác nhau.
* Khi thực hiện các requests trải rộng trên nhiều service khó khăn thì điều này cần đòi hỏi sự phối hợp giữa các team.
* Thường rất khó khăn trong việc đảm bảo toàn vẹn cho CSDL nếu như triển khai theo các cấu trúc cơ sở dữ liệu dạng phân vùng.
* Việc triển khai và quản lý microservices nếu như làm thủ công theo cách làm với ứng dụng thì sẽ rất phức tạp.
* Lập trình viên cần phải xử lý các sự cố kết nối chậm, lỗi nếu như thông điệp không được gửi hoặc nếu như thông điệp được gửi đến nhiều đích đến vào các thời điểm khác nhau.

*Microservice sở hữu nhiều ưu điểm*

**1.4) Các lợi ích mang lại của Microservices là gì:**

Microservices được sử dụng ngày càng phổ biến nhờ mang lại vô số lợi ích như:

* Source code rất tinh gọn: Bởi vì hệ thống được cấu thành từ các dự án nhỏ, và mỗi dự án đều rất đơn giản cũng như tập trung vào 1 hoặc 1 vài nghiệp vụ chính. Vì vậy, các code base và độ phức tạp của chúng đều không cao. Nhờ vậy, nó sẽ giúp mang lại tính gọn nhẹ, dễ bảo trì cũng như mở rộng hơn.
* Bảo mật tối ưu cho source code: Khi nhân viên làm việc ở các dự án thì chỉ truy cập được vào một source code của dự án đó.
* Được tồn tại độc lập: Bởi vì đây là 4 dự án khác nhau và chúng có thể có cách deploy riêng biệt và một service nào đó chết thì các service khác vẫn sẽ hoạt động một cách bình thường.
* Scale hoàn toàn độc lập: Tùy thuộc vào nhu cầu sử dụng của hệ thống mà bạn có thể scale riêng cho service đó. Có thể như service đơn hàng mà sử dụng thường xuyên nên chạy từ 2 đến 3 server để gia tăng performance.

**1.5) Bạn nên sử dụng kiến trúc Microservices khi nào là hợp lý:**

Với những thách thức đối với nhu cầu sử dụng Microservices thì bạn nên dùng cấu trúc này khi: Khi phát triển những phiên bản đầu tiên cho một ứng dụng, khi đó bạn thường không phải gặp những vấn đề mà Microservices cần phải giải quyết. Hơn nữa, việc sử dụng một kiến trúc phân tán hoặc phức tạp sẽ làm chậm đi quá trình phát triển của ứng dụng. Đây là một trong những vấn đề lớn đối với các start-up bởi vì họ buộc phải phát triển nhanh mô hình kinh doanh của mình cũng như ứng dụng kèm theo. Chính vì vậy, trừ khi bạn đã có một hệ thống phức tạp để quản lý bằng Monolithic hoặc bạn đã xác định được tương lai của ứng dụng sẽ ra sao; thì có thể dùng Microservices.

**1.6) Các vấn đề nên lưu ý khi thiết kế Microservices:**

Sau khi hiểu rõ Microservices là gì thì ngay sau đây, ITNavi sẽ đưa ra các lưu ý khi thiết kế Microservices như sau:

**Hiểu sai về Microservices:**

* Một số dòng code/kích cỡ của một đội lập trình thường là chỉ số tồi.
* Mico là một từ khóa dễ gây hiểu nhầm và bạn nghĩ rằng nên tạo ra services nhỏ hết mức thì đó là cách hiểu hoàn toàn sai.
* Services trở thành các cục monolithic với nhiều hàm, chức năng khác được hỗ trợ nhau. Chính vì thế, khi phát triển services kiểu SOA rồi dán nhãn Microservices hoàn toàn bị đánh lạc hướng và không mang lại bất kỳ lợi ích nào.

**Những điều cần phải tuân thủ:**

* Một Service có phạm vi và chức năng giới hạn thì việc tập trung vào một nhiệm vụ sẽ giúp cho quá trình phát triển cũng như triển khai dịch vụ trở nên nhanh chóng hơn.
* Khi thiết kế, bạn nên xác định và giới hạn cho các service dựa theo chức năng nghiệp vụ thực tế.
* Hãy đảm bảo microservices có thể phát triển cũng như được triển khai độc lập.
* Mục tiêu thiết kế là đưa ra phạm vi cho một microservices phục cụ cho một nghiệp vụ chứ không đơn giản là làm những dịch vụ nhỏ hơn. Khi đó, kích thước hợp lý của một services đó chính là kích thước đủ để đáp ứng cho các yêu cầu của một chức năng bên trong hệ thống.
* Khác biệt so với services trong SOA thì một microservice không nên có quá nhiều hàm hoặc chức năng hỗ trợ xung quanh cũng như định dạng thông báo gửi hoặc gửi tin đơn giản.

**1.7) So sánh giữa kiến trúc Microservice và Monolithic:**

a. Tổng quan

* *Về kiến trúc Monolithic*

Như tên gọi Monolithic – nguyên khối – đây là ứng dụng mà mọi dịch vụ đều nằm trên một hệ thống và dùng chung 1 database. Người dùng cứ thẳng đuột mà thao tác với giao diện, giao diện gọi đến chức năng, chức năng lại tìm đến dữ liệu trong database. Đó chính là kiến trúc Monolithic – Nguyên khối.

* *Về kiến trúc Microservce*

Do có cái nhìn tổng quan nhờ Monolithic thì ta cũng mườn tượng ra kiến trúc Microservice như nào rồi. Đó là kiểu kiến trúc thay vì các chức năng được đặt cùng nhau trong hệ thống thì nó sẽ được tách riêng lẻ và có thể với mỗi chức năng đó sẽ tương tác đến các database khác nhau.

b. So sánh

**b.1 Độ phức tạp:**

Về độ phức tạp ta sẽ phải xem xét về 2 vấn đề:

* Độ phức tạp của quá trình phát triển.
* Độ phức tạp của việc triển khai phần mềm.

Đối với sự phức tạp của việc phát triển, kích thước của codebase có thể nhanh chóng phát triển khi xây dựng phần mềm dựa trên microservice. Nhiều mã nguồn có liên quan, sử dụng các framework, ngôn ngữ khác nhau và các phiên bản thư viện ở từng dịch vụ có thể sẽ 0khác nhau. Đối với khía cạnh điều hành và giám sát, số lượng dịch vụ bị ảnh hưởng có liên quan nhiều. Kiến trúc nguyên khối chỉ giao tiếp trong phạm vi nhất định, đó có nghĩa là mỗi chức năng cụ thể sẽ luôn có một quy trình xử lý của mình. Bên cạnh đó, một chức năng trong kiến trúc microservice có thể ảnh hưởng đến các dịch vụ khác nhau.

Với việc triển khai hệ thống microservices có thể dùng nhiều công cụ khác nhau, lấy ví dụ như Kubernetes là một công cụ phổ biến thì việc triển khai cũng khá phức tạp, mặc dù Kubernetes cho phép các khả năng như tự động mở rộng quy mô, nhưng nó không phải là một hệ thống dễ quản lý. Bên cạnh đó, để triển khai một hệ thống sử dụng kiến trúc nguyên khối, chỉ cần một thao tác sao chép đơn giản là đủ. Để bắt đầu hoặc dừng hệ thống nguyên khối thường chỉ cần một lệnh đơn giản là đủ. Các dữ liệu giữa các dịch vụ cũng làm tăng thêm sự phức tạp khi chạy ở kiến trúc microservice, so với một khối nguyên khối. Ngoài ra việc đồng bộ hóa cũng là 1 thử thách ở kiến trúc microservice. Ví dụ: việc xử lý các requests không tốt có thể gây ra lỗi thanh toán hai lần. Các kiến trúc microservice có thể quản lý điều này bằng cách sử dụng các kỹ thuật như bộ điều phối trung tâm. Tuy nhiên, trong một kiến trúc nguyên khối, các giao dịch rất dễ xử lý hoặc thậm chí minh bạch đối với nhà phát triển.

Kiến trúc nguyên khối thắng về độ phức tạp

**b.2 Độ tin cậy:**

Khi thiết kế hệ thống ta phải giả định rẳng hệ thống sẽ gặp sự cố tại một thời điểm nào đó. Với hệ thống sử dụng kiến trúc Microservcies, một dịch vụ trong hệ thống bị lỗi, người quản lý chỉ cần khoanh vùng và đưa ra giải pháp để xử lý dịch vụ đó khiến cho hệ thống sẽ vẫn hoạt động bình thường với các chức năng đang hoạt động. Kiến trúc Microservices đã cung cấp một số các giải pháp để giải quyết vấn đề này như Chaos Monkey (Netflix).

Bên cạnh đó với kiến trúc nguyên khối việc xảy ra sự cố sẽ gây tê liệt toàn bộ hệ thống dẫn đén thiệt hại khá lớn

VD: Nếu xảy ra sự cố sập máy chủ, hệ thống với các dịch vụ được phân tán khắp nơi sẽ được đảm bảo an toàn hơn.

Microservices giành chiến thắng về độ tin cậy

**b.3 Tài nguyên sử dụng:**

Với một hệ thống nhỏ việc gọi đến một request thì kiến trúc Microservice sẽ sử dụng nhiều tài nguyên hơn (trong trường hợp thuật toán giống nhau). Một số chi phí phát sinh có thể do docker, máy ảo, việc ghi nhật ký, giám sát tiến trình …

Tuy nhiên, microservices cho phép chúng ta sử dụng tài nguyên thông minh hơn rất nhiều. Vì người quản lý có thể phân bổ tài nguyên khi cần thiết nên mức sử dụng tài nguyên thực tế có thể thấp hơn nhiều.

Do việc so sánh 2 kiến trúc thường sẽ thực hiện trên các hệ thống có quy mô lớn nên đây có thể coi là điểm tốt hơn của microservices so với monolithic.

Microservices giành chiến thắng về tài nguyên sử dụng với hệ thống lớn.

**b.4 Khả năng mở rộng:**

Nhờ tính linh hoạt khi kết hợp các dịch vụ trong hệ thống nên việc mở rộng đối với kiến trúc Microservices sẽ dễ dàng hơn. Với kiến trúc Monolithic việc thêm các tính năng sẽ càng làm cho codebase của dự án lớn lên, các công nghệ có thể xung đột khi thêm một chức năng mới. Ngoài ra chi phí cũng là một điểm cần lưu ý do tính linh hoạt khi tính phí theo từng khoảng thời của các dịch vụ riêng lẻ trong hệ thống sử dụng kiến trúc

Microservices, điều này sẽ giúp tiết kiệm hơn so với việc chạy liên tục của hệ thống sử dụng kiến trúc nguyên khối.

Để mở rộng quy mô một cách chính xác và sử dụng tài nguyên tốt hơn thì kiến trúc Microservices sẽ mang lại nhiều lợi ích hơn

Microservices giành chiến thắng về khả năng mở rộng.

**b.5 Hiệu suất truyền tải:**

Trong các khối lượng công việc không thể chạy đồng thời trên toàn mạng, kiến trúc nguyên khối có thể mang lại hiệu suất tốt hơn. Dữ liệu cần được gửi giữa các dịch vụ và cũng như tất cả cơ sở hạ tầng gây ra một chi phí nhất định. Nếu khối lượng công việc không thể được chia tỷ lệ cho nhiều trường hợp, một hệ thống sử dụng kiến trúc nguyên khối có thể mang lại hiệu suất truyền tải cao hơn.

Với khối lượng công việc được chạy trong cùng local và không có chi phí do container, giám sát container hoặc dịch vụ.

Kiến trúc nguyên khối giành chiến thắng về hiệu suất truyền tải

**b.6 Tổng kết:**

|  |  |
| --- | --- |
| **Kiến Trúc Microservices** | **Kiến Trúc Nguyên Khối** |
| Dễ dàng phát triển mở rộng | It phức tạp hơn |
| Tài nguyên sử dụng sẽ ít hơn | Hiệu suất truyền tải dữ liệu tốt hơn |
| Đảm bảo an toàn khi gặp sự cố |  |

Với mỗi từng kiến trúc sẽ có ưu nhược điểm riêng nên việc lựa chọn kiến trúc sẽ tùy thuộc vào dự án mà bạn sẽ triển khai.

**PHẦN 3 : SPRING BOOT:**

**1.1)Spring Boot là gì:**

Spring Boot là một Java framework được phát triển bởi Pivital Team dựa trên Java framework mã nguồn mở để tạo ra các microservice, nhằm mục đích xây dựng các ứng dụng Spring độc lập một cách nhanh chóng và có khả năng thực thi ngay.

Sau khi sử dụng Spring Boot để lập trình ứng dụng Spring, bạn có thể chạy ngay ứng dụng của mình mà không cần phải config (cấu hình) quá nhiều.

**1.2)Những đặc điểm nổi bật của Spring Boot:**

* Spring Boot được phát triển tối ưu sao cho việc cấu hình XML trở nên đơn giản nhất trong Spring.
* Spring Boot được phát triển sao cho việc lập trình trở nên nhanh chóng và dễ dàng.
* Gia tăng năng suất trong lập trình.
* Giảm thời gian lập trình xuống tối thiểu.
* Spring Boot được phát triển nhằm giúp người không có nhiều kiến thức lập trình vẫn có thể xây dựng ứng dụng.

TinoHost có thể kể đến 3 tính năng rất quan trọng tạo nên sự “dễ chịu” của Spring Boot như:

* SpringApplication: bạn lập trình xong và chỉ muốn chạy thử nghiệm luôn ứng dụng của mình? Vậy bạn chỉ cần gọi run() là được. Vì Spring Boot được thiết kế theo dạng “just run” nhằm giúp lập trình viên chỉ cần cấu hình ít Spring nhất có thể, còn lại Spring Boot sẽ tự động lo liệu.
* Externalized Configuration: bạn muốn tạo một ứng dụng có thể chạy trên nhiều môi trường khác nhau? Vậy, Spring Boot sẽ giúp bạn config cấu hình từ bên ngoài và ứng dụng của bạn sẽ có thể thoải mái chạy.
* Profiles: Khi bạn có nhiều config khác nhau, bạn có thể sử dụng Profile để phân chia từng loại config cho từng môi trường để dễ dàng quản lý hơn.

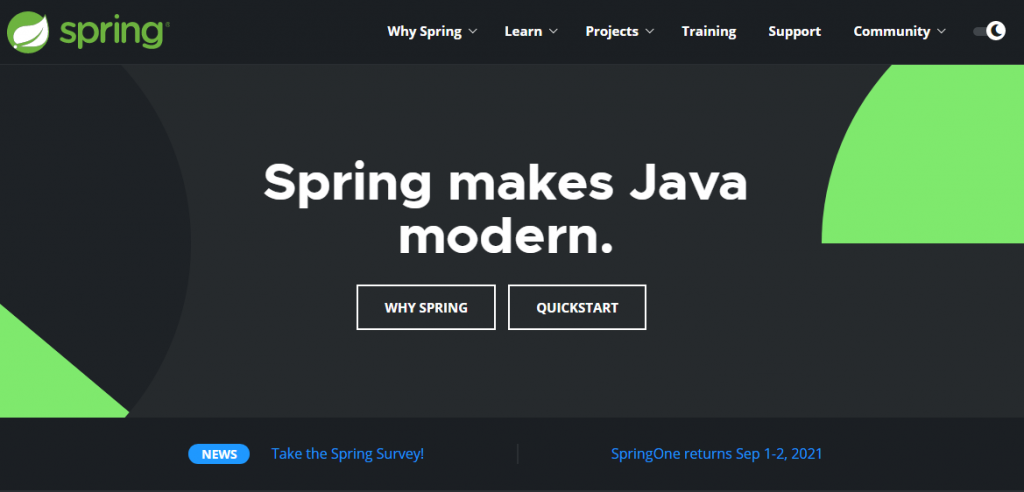
Không chỉ dừng lại ở những tính năng đó, với Spring Boot, bạn còn có thể phát triển web Application, Caching, gửi Email, Validation, làm việc với công nghệ SQL,…

[](https://camo.githubusercontent.com/a82b5c852840c66c992b52c98ad9afa9300dad51136a23f21cf8340d12d3743f/68747470733a2f2f77696b692e74696e6f2e6f72672f77702d636f6e74656e742f75706c6f6164732f323032312f30372f776f72642d696d6167652d313534322e706e67)

**1.3)Vì sao nên học Spring Boot:**

Để xây dựng một ứng dụng Java, bạn sẽ có vô số công nghệ để có thể sử dụng. Theo TinoHost, đây là 5 tính năng và lợi ích vô cùng thuyết phục vì sao bạn nên chọn Spring Boot:

* Bạn ngán ngẩm việc phải cấu hình từng chút một cho ứng dụng của mình? Spring Boot sẽ tự động cấu hình gần như mọi thứ giúp bạn tập trung vào việc code.
* Với một ứng dụng lớn, việc quản lý từng Microservice sẽ vô cùng phức tạp. Spring Boot sẽ giúp bạn đơn giản hóa việc này.
* Annotation Based Configuration là tính năng đắc lực sẽ hỗ trợ bạn tạo lập bean thay vì XML.
* Servlet được nhúng sẵn trong Spring Boot, vì thế bạn có thể bật và chạy Server Tomcat một cách dễ dàng.
* Spring Boot rất linh hoạt để bạn có thể cấu hình Java Beans, XML và cả Database Transaction (giao dịch/ trao đổi) .



**1.4)Tại sao spring boot áp dung trong Microservices:**

* Microservices được sử dung rất rông rãi trong các project java Web bây giờ, bản chất Microservices đó là chia để trị tức là một khối project java Web lớn chúng ta có thể chia thành các project Java web nhỏ nhờ đó dễ dàng quản lý thời gian,lỗi,member,tài nguyên trong dự án
* Mỗi project Java web nhỏ là một Microservices và thường sử dụng spring boot.

# PHẦN 4 : JHipster

## 1)JHipster là gì:

JHipster is a development platform to generate, develop and deploy Spring Boot + Angular/React Web applications and Spring microservices.

Nói một cách đơn giản, JHipster( viết tắt của Java Hipster) là phương pháp đơn giản để chúng ta tạo ra một project xung quanh những công nghệ được ưa thích nhất với Spring technologies và Angular/React.Khi chúng ta bắt đầu dự án chúng ta sẽ quan tâm đến 3 khía cạnh:Server side stack sẽ trông như thế nào?Client side stack sẽ trông như thế nào?Làm sao để chúng ta khả năng deploy project của chúng ta?

### 1.1)Server side:

Khi chúng ta bắt đầu build phần backend có những câu hỏi mà chúng ta quan tâm đó là:

Ngôn ngữ chúng ta lựa chọn là gì?Tầng dữ liệu sẽ như thế nào?Hệ thống sẽ bảo mật ra sao?Khả năng bảo trì và mở rộng hệ thống?Cách cung cấp API document?Kiểm thử ứng dụng thế nào?Câu trả lời sẽ có khi bạn nhìn vào danh sách công nghệ mà JHipster cung cấp :

### 1.2)Client side:

Với những framework frontend mạnh mẽ

### 1.3)Deployment:

Deply dự án đơn giản

## 2)Cách khởi tạo Monolithic và Microservices

### a) yêu cầu cài đặt:

#### 1. Cài đặt Java 11. Chúng tôi khuyên bạn nên sử dụng các bản dựng AdoptOpenJDK , vì chúng là mã nguồn mở và miễn phí.

#### 2. Cài đặt Node.js từ trang web Node.js (vui lòng sử dụng phiên bản LTS 64 bit, các phiên bản không phải LTS không được hỗ trợ)

#### 3. Cài đặt JHipster:npm install -g generator-jhipster

#### 4. (tùy chọn) Nếu bạn muốn sử dụng một mô-đun hoặc một bản thiết kế (ví dụ: từ JHipster Marketplace ), hãy cài đặt Yeoman :npm install -g yo

### b) kiểm tra xem đã cài java, node.js, Git,npm :

#### B1 để mở cmd :

Windows -> Run -> cmd

#### B2 kiểm tra xem máy ban đã có java, node.js, Git,npm:

#### .Kiểm tra java:

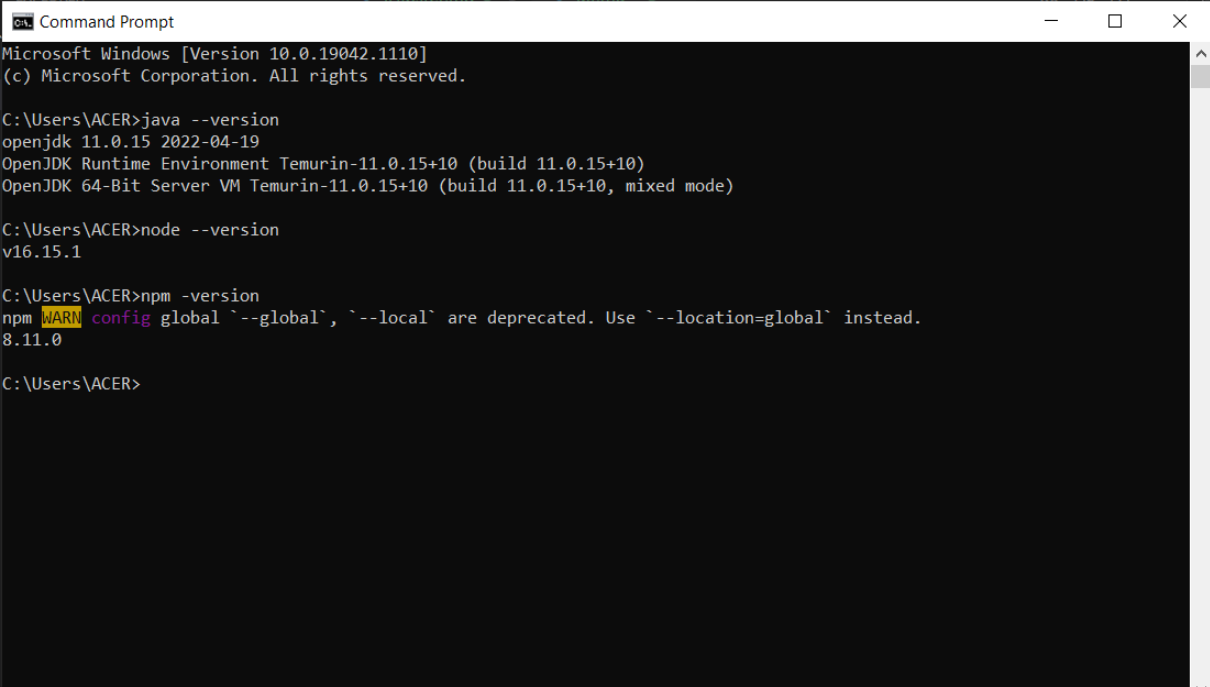
java --version

#### .Kiểm tra node js:

node --version

#### .Lệnh kiểm tra cài đặt npm:

npm -version

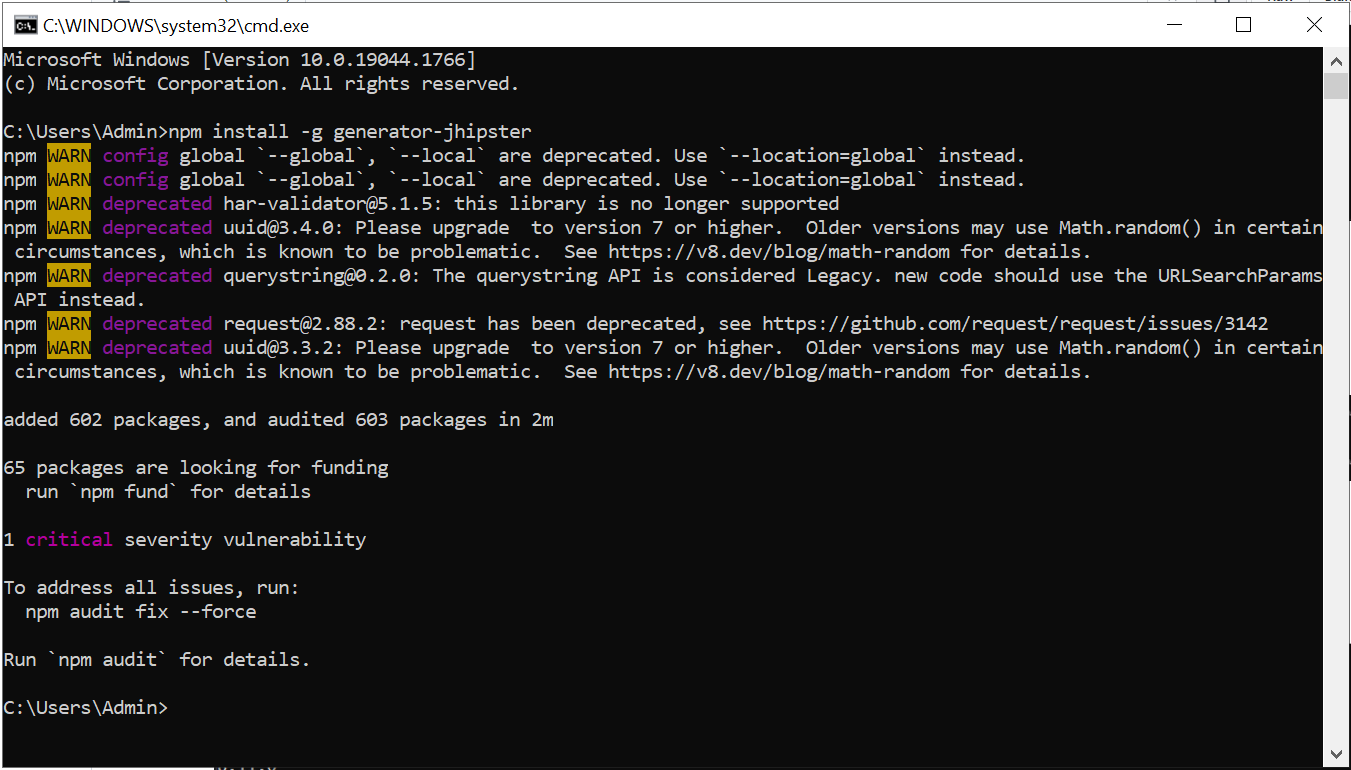
[](https://user-images.githubusercontent.com/107382675/174943769-3a5d9f66-5a1f-4cd2-9e04-523673840e09.png)

#### B3 cày đăt npm và Jhipster:

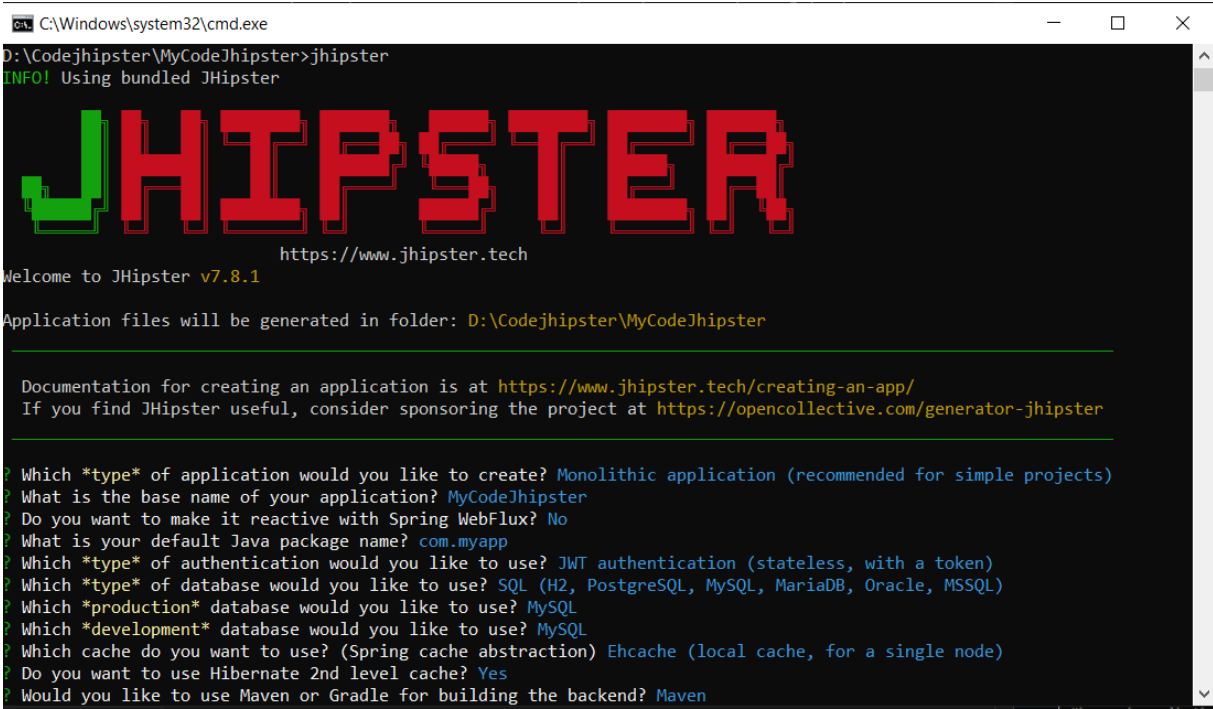
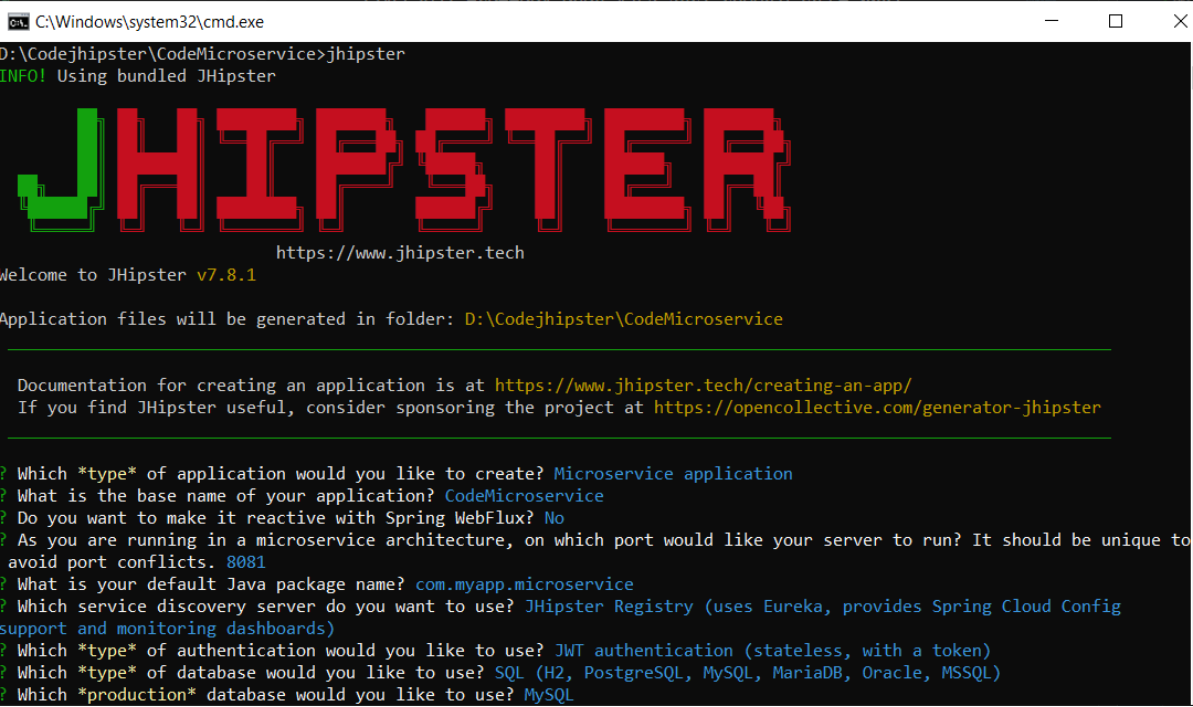
##### a)Cài đặt Jhipster:

Lệnh cài đặt java, node.js, git:

###### .npm install -g generator-jhipster

[](https://user-images.githubusercontent.com/107382675/174944299-4e84eb9a-0f7e-42b0-9a6b-889e08f559ef.png)

#### B4 Khởi tạo Monolithic và Microservices:

Khởi tạo Monolithic [](https://user-images.githubusercontent.com/107382675/174944729-95eace9c-64a5-448f-bb8b-0a82319ea10e.png) Khởi tạo Microservices [](https://user-images.githubusercontent.com/107382675/174944859-6c592711-2a66-468c-b004-773923512c11.png)

###### a)Tạo microservices :

Microservices là một loại ứng dụng JHipster, không có giao diện người dùng (giao diện người dùng Angular phải được tạo trên một cổng) và hoạt động với JHipster Registry để được định cấu hình, phát hiện và quản lý.

###### b)Các thực thể trong kiến ​​trúc microservices:

Thực thể Người dùng không được tạo trong một microservice khi sử dụng JWT hoặc DTO. Khi OAuth 2.0 được sử dụng, có một cơ chế trong microservice để trích xuất dữ liệu người dùng từ mã thông báo và lưu nó vào cơ sở dữ liệu của microservice. Vì vậy, đối với JWT và DTO, bạn không thể sử dụng và / hoặc xác định mối quan hệ với thực thể Người dùng vì nó chỉ tồn tại trong cơ sở dữ liệu của cổng.

Mối quan hệ giữa các thực thể từ các dịch vụ vi mô khác nhau không được hỗ trợ.

###### c)Tạo thực thể:

Đối với các thực thể trong một microfrontend, hãy xem Microfrontends .

Việc sử dụng trình tạo phụ thực thể hoạt động hơi khác một chút trong kiến ​​trúc microservices, vì mã front-end và back-end không nằm trong cùng một ứng dụng.

Đầu tiên, tạo các thực thể trong ứng dụng microservice: điều này hoạt động như bình thường và bạn cũng có thể sử dụng JHipster UML hoặc JDL Studio để giúp bạn tạo các thực thể và mối quan hệ phức tạp. Vì microservices không có giao diện người dùng nên sẽ không có mã giao diện người dùng nào được tạo.

Sau đó, trên (các) cổng, hãy chạy lại trình tạo phụ thực thể. Một câu hỏi mới sẽ xuất hiện ở phần đầu, dành riêng cho các cổng:

###### .Bạn sẽ có lựa chọn để tạo một thực thể mới thông thường (một cổng cũng là một ứng dụng JHipster tiêu chuẩn, vì vậy điều này sẽ hoạt động giống như một ứng dụng nguyên khối) hoặc sử dụng cấu hình JHipster hiện có từ một microservice.

###### .Nếu bạn chọn tạo thực thể từ một microservice, bạn sẽ cần nhập đường dẫn đến microservice này trên máy tính cục bộ của mình và sau đó JHipster sẽ tạo mã giao diện người dùng trên cổng.

###### d)Microfrontends:

Hỗ trợ microfrontend đang được tiến hành. Việc triển khai có thể thay đổi và khác nhau giữa các khuôn khổ. Tham khảo Hỗ trợ Microfrontend để biết trạng thái mới nhất.

Việc triển khai microfrontends của JHipster sử dụng Liên kết mô-đun Webpack và cho phép việc triển khai các thực thể phía trước được đặt trong microservice thay vì trong cổng.

Một cổng vẫn được yêu cầu để chạy trong quá trình phát triển do quá trình xác thực.

Bạn có thể tìm thêm thông tin về cách bắt đầu microfrontends bằng cách sử dụng cổng hoặc một mình trong README đã tạo.

###### e)Bộ nhớ đệm phân tán với Hazelcast

Nếu ứng dụng của bạn sử dụng cơ sở dữ liệu SQL, JHipster đề xuất giải pháp bộ nhớ đệm cấp 2 khác với microservices:

###### .Giải pháp bộ nhớ đệm mặc định của JHipster với microservices là Hazelcast

###### .Bạn vẫn có thể chọn Ehcache (giải pháp mặc định với các ứng dụng nguyên khối) hoặc Caffeine hoặc chọn hoàn toàn không sử dụng bộ đệm

Giải pháp này là giải pháp mặc định với microservices, vì trong kiến ​​trúc này, ý tưởng là bạn sẽ mở rộng các dịch vụ của mình:

###### .với bộ nhớ cache cục bộ, các phiên bản dịch vụ của bạn sẽ không có bộ nhớ cache được đồng bộ hóa, dẫn đến dữ liệu không chính xác

###### .mà không có bất kỳ bộ nhớ cache nào, gánh nặng của việc mở rộng quy mô sẽ được chuyển sang cơ sở dữ liệu, điều này sẽ không tốt lắm (trừ khi bạn sử dụng tùy chọn Cassandra của chúng tôi)

Sử dụng Hazelcast với microservices sẽ dẫn đến một cấu hình cụ thể:

###### .Khi khởi động, ứng dụng của bạn sẽ kết nối với JHipster Registry để tìm xem các phiên bản khác của cùng một dịch vụ có đang chạy hay không

###### .Với devhồ sơ, JHipster sẽ tạo một cụm các trường hợp đó trên localhost ( 127.0.0.1), sử dụng một cổng khác nhau cho mỗi trường hợp. Theo mặc định, cổng Hazelcast là your application's port + 5701(vì vậy nếu cổng của ứng dụng của bạn là cổng 8081, thì Hazelcast sẽ sử dụng cổng 13782)

###### .Với prodcấu hình, JHipster sẽ tạo một cụm với tất cả các nút khác mà nó tìm thấy, sử dụng cổng Hazelcast mặc định ( 5701)

###### f)Microservices không có cơ sở dữ liệu:

Chỉ có thể tạo các ứng dụng microservices mà không có cơ sở dữ liệu. Điều này là do microservices nhỏ và không có mã quản lý người dùng.

Một microservice không có cơ sở dữ liệu là rất nhỏ và có thể được sử dụng để kết nối với một back-end cụ thể như một hệ thống kế thừa.

## 3)Tại sao lựa chọn JHipster:

Chúng ta sẽ dễ dàng có được một project đủ mạnh mẽ đầy đủ những thứ cơ bản để bắt đầu với thời gian nhanh nhất.Và nếu bạn đang tập tành với Spring để trở thành 1 Java Web developer thì bạn nên quan tâm đến công nghệ này.

## 4)Tác dụng JHipster:

##### . Tao một ứng dụng

##### . Tạo một thực thể

##### . Tạo một Sring Controller

##### . Tạo một Sring Service

##### . Tạo DTO

##### . Quản Lý Các mối quan hệ

##### . Quốc Tế hóa

##### . Nâng Cấp Ưng Dụng