# **BÁO CÁO**

# **Chương 1: Git**

## **1. Khái niệm của Git**

GIT 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 người dùng 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.

## **2. Tác dụng của Git**

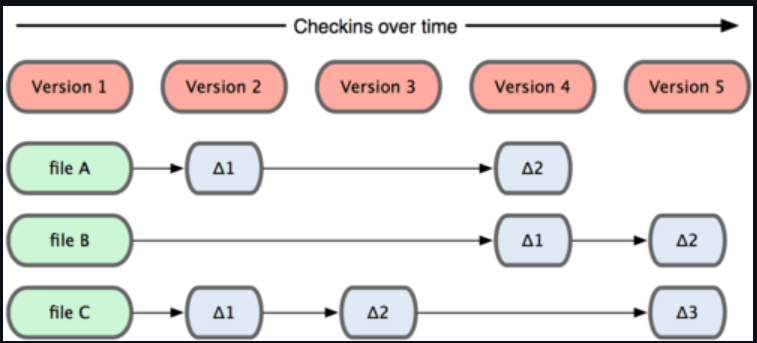
* Lưu lại lịch sử các version của bất kỳ thay đổi nào của dự án. Giúp xem lại các sự thay đổi hoặc khôi phục (revert) lại sau này.
* Việc chia sẻ code trở nên dễ dàng hơn, lập trình viên có thể để public cho bất kỳ ai, hoặc private chỉ cho một số người có thẩm quyền có thể truy cập và lấy code về.

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.

## **3. Cách Git hoạt động**

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)

## **4. Ưu điểm và hạn chế của GIT**

#### **4.1. Ưu điểm của GIT**

* 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 người dùng có thể làm việc ở mọi lúc mọi nơi.
* Deployment sản phẩm của người dùng một cách không thể nào dễ dàng hơn.

#### **4.2. Hạn chế của GIT**

* 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.

## **5. Các thuật ngữ GIT**

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

## **6. Các lệnh GIT cơ bản**

1. git config

Tác dụng : Để set user name và email của người dùng 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)”

1. 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.

1. git clone

Tác dụng: Copy 1 git repository từ remote source.

Cách xài: git clone <:clone git url:>

1. git status

Tác dụng: Để check trạng thái của những file người dùng 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.

1. git add

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

Cách xài: git add

1. 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 người dùng cũng có thể khôi phục lại tập tin trong lịch sử commit của nó để chia cho một branch khác, vì vậy người dùng sẽ dễ dàng khôi phục lại các thay đổi trước đó.

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

1. git push/ git pull

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

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

1. git branch

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

Cách dùng: git branch hoặc git branch -a

1. git checkout

Tác dụng: Chuyển sang branch khác

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

1. git stash

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

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

1. git merge

Tác dụng: Merge 2 branch lại với nahu.

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

1. git reset

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

Cách dùng: git reset HEAD tên\_file

1. git remote

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

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

1. 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

## **7. Sự khác biệt của GIT và các phần mềm khác**

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.

## **8. Khi sử dụng GIT**

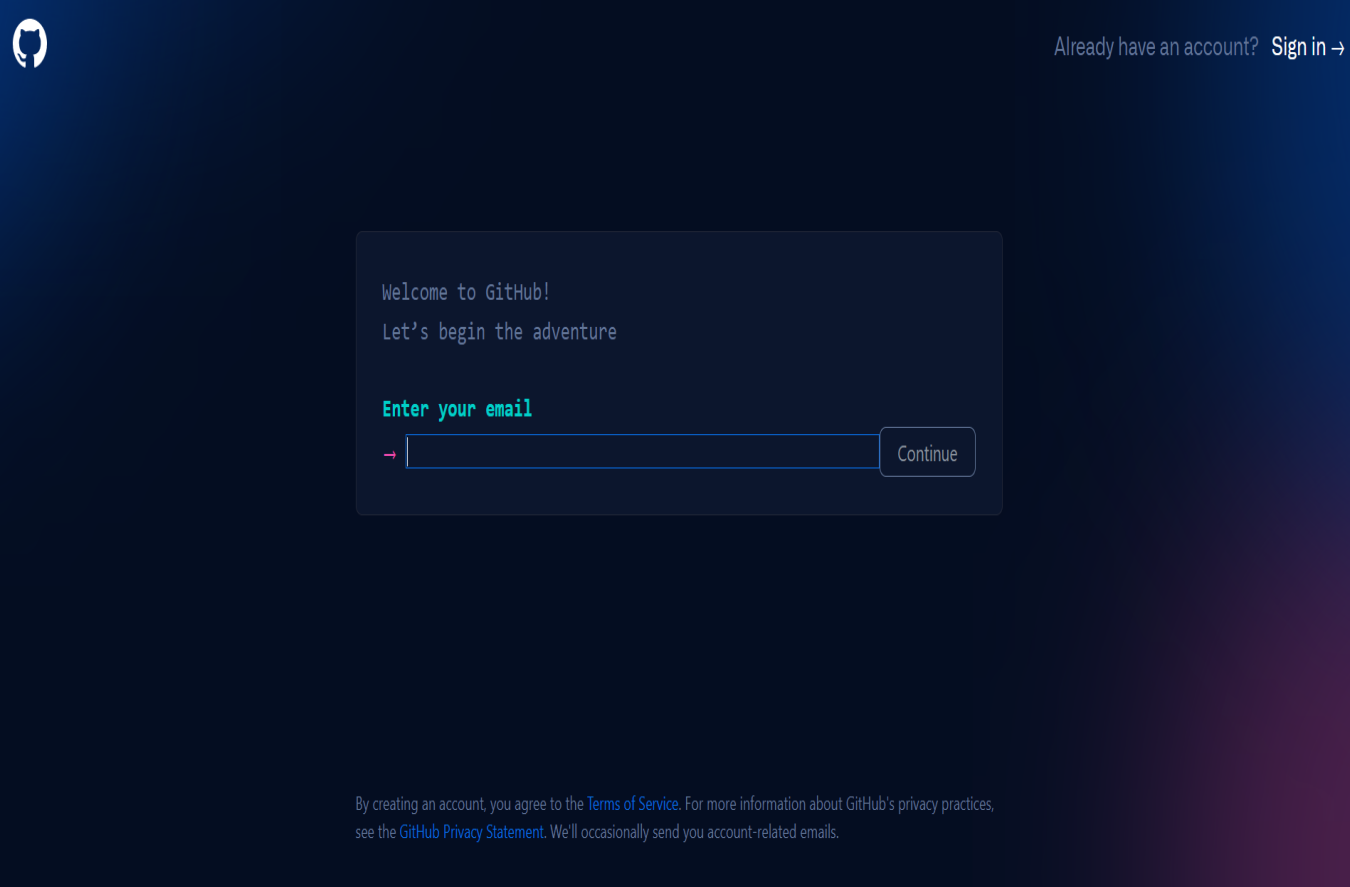
* Lưu lại được các phiên bản khác nhau của mã nguồn dự án phần mềm.
* Khôi phục lại mã nguồn từ 1 phiên bản bất kỳ.
* Dễ dàng so sánh của các phiên bản.
* Phát hiện được ai đã sửa phần nào làm phát sinh lỗi.
* Khôi phục lại tập tin bị mất.
* Dễ dàng thử nghiệm, mở rộng tính năng của dự án mà không làm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

## **9. 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.

## **10. Cách sử dụng GIT**

#### **10.1. B1: Tạo tài khoản GitHub**

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

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

#### **10.2. B2: 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 người dùng.

* 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

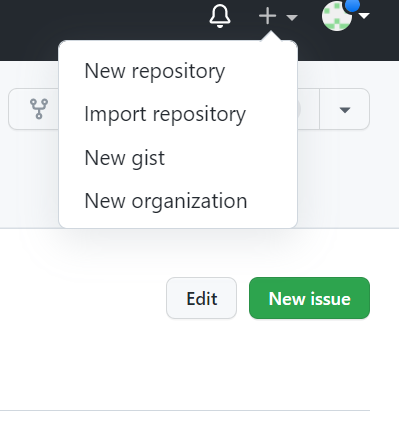
Chạy các lệnh sau với thông tin của người dùng để đặt tên người dùng và email mặc định khi người dùng sẽ lưu công việc của mình.

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

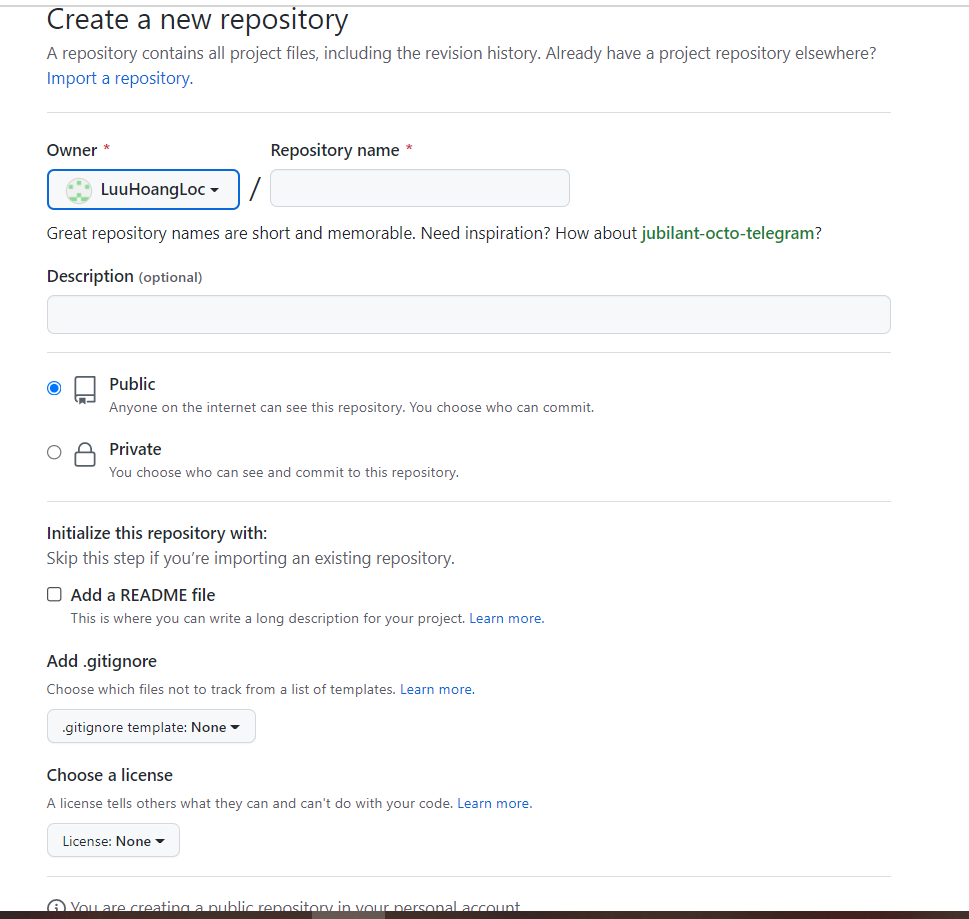
#### **10.3. B3: Cách sử dụng Git**

1. Cách thứ 1: Tạo kho lưu trữ, sao chép nó vào pc của người dùng 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 người dùng, 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 người dùng được tạo.

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

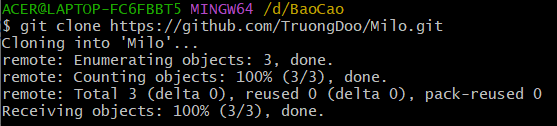
Để sao chép một kho lưu trữ có nghĩa là người dùng đang lấy một kho lưu trữ trên máy chủ và sao chép nó vào máy tính của người dùng – giống như tải xuống. Trên trang kho lưu trữ, người dùng cần lấy địa chỉ HTTPS.

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

Khi người dùng có địa chỉ của kho lưu trữ, người dùng cần sử dụng terminal của mình. Sử dụng lệnh sau trên terminal của người dùng. Khi người dùng đã sẵn sàng, người dùng 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/107389856/173522116-dd2c1d13-6691-430a-b504-1a8a36a80247.png)

Thông báo đầu ra của lệnh git clone bản sao

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

- cd [NAME OF REPOSITORY]

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

\*LƯU Ý: Khi người dùng sao chép, Git sẽ tạo một kho lưu trữ trên máy tính của người dùng. Nếu người dùng muốn, người dùng 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 người dùng. Nếu người dùng đã đóng nó, như người dùng đã nói trước đây, hãy sử dụng lệnh ‘cd’.

- cd [NAME OF REPOSITORY]

Bây giờ, trong terminal, người dùng đang ở trong thư mục kho lưu trữ của người dùng. 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 người dùng. Chúng ta hãy đi qua từng cái một.

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

- git status

[](https://user-images.githubusercontent.com/107389856/173522836-8503e5e4-ba38-4f27-902a-0bf982c40b22.png)

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

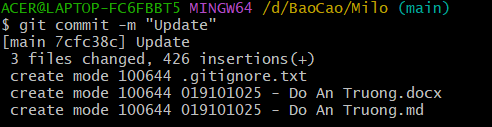
- git add [FILENAME] [FILENAME] […]

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/107389856/173523002-618b4b71-cdd0-4c99-a4b7-fd28e6d7c355.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 người dùng 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!

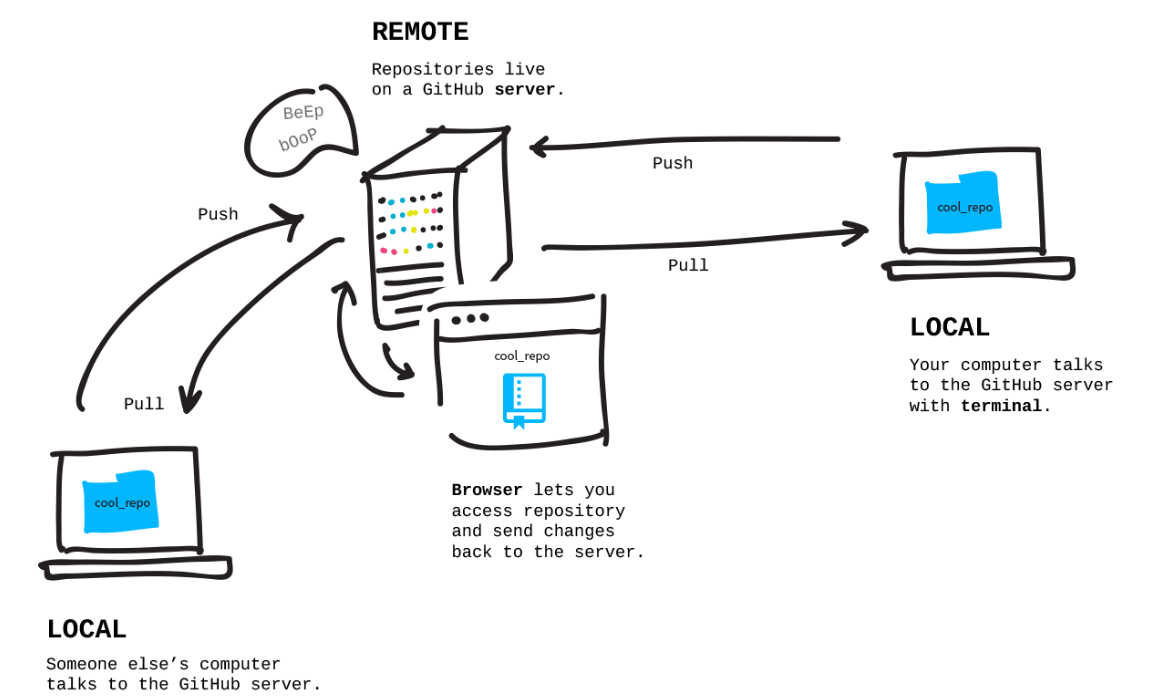
[](https://user-images.githubusercontent.com/107382675/173312470-1204e5b4-324c-4c3c-9177-e0bb07166734.png)

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

- :q

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.

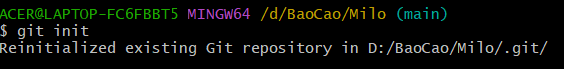
1. Cách thứ 2: Làm việc trên dự án của người dùng cục bộ sau đó tạo kho lưu trữ trên github và đẩy nó vào remote.

Loại 2 cho phép người dùng 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, người dùng có thể đã thực sự tạo ra một cái gì đó trên máy tính mà người dùng 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/107389856/173523170-b961588b-30c8-4489-9315-2d4386f522f6.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/107389856/173522836-8503e5e4-ba38-4f27-902a-0bf982c40b22.png)

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

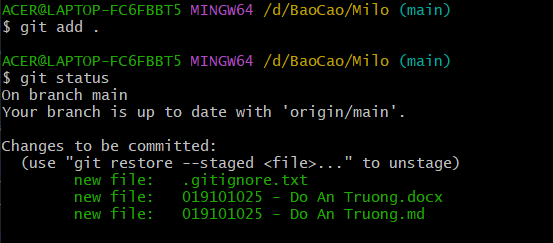
- git add [FILENAME] [FILENAME] […]

\*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, người dùng đã hoàn thành công việc của mình!

[](https://user-images.githubusercontent.com/107389856/173523453-865e14b2-6dce-4d38-a259-59e005960ddb.png)

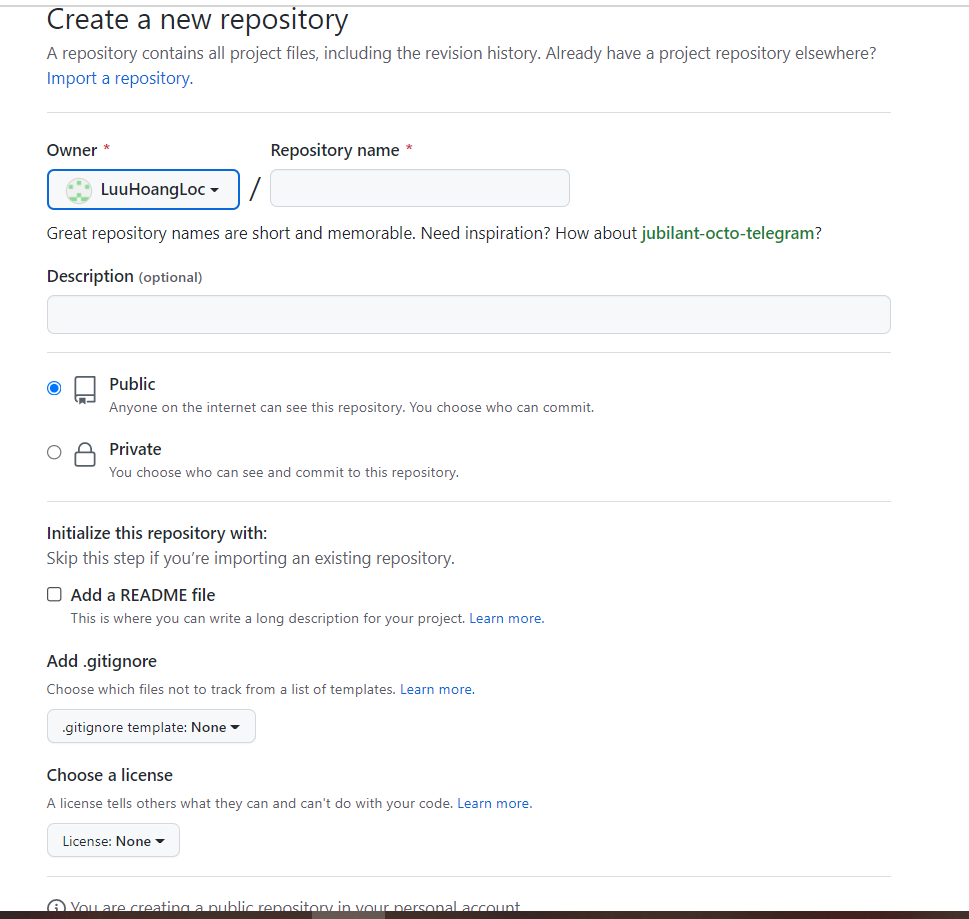
Sau đó, chúng ta phải commit với một mô tả trong đó.

- git commit -m “Adding web Survey form”

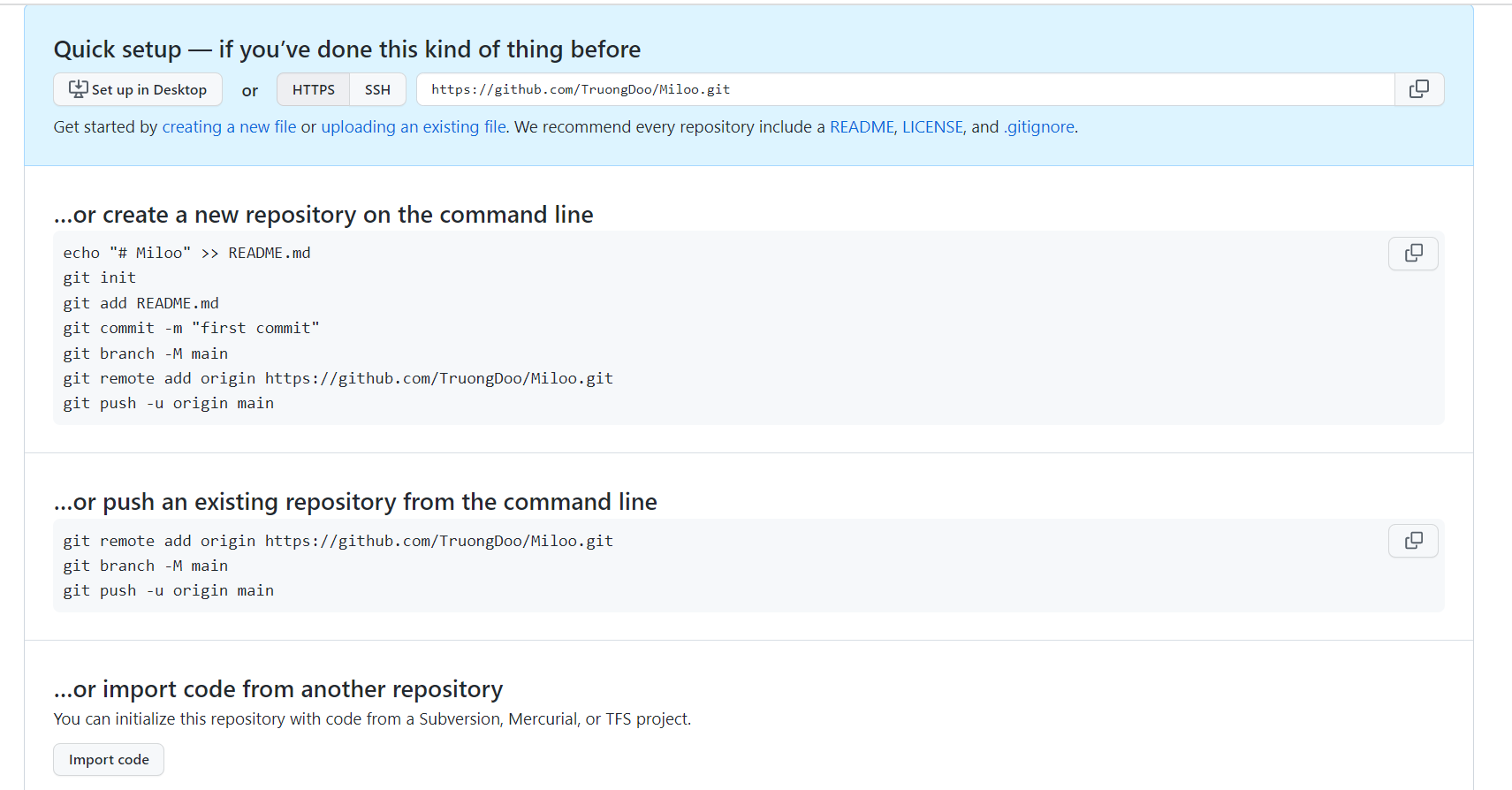
Nếu kho lưu trữ của người dùng bắt đầu trên GitHub và người dùng đã đưa nó xuống máy tính của mình, một Remote đã được gắn vào nó (Loại 1). Nhưng nếu người dùng đang khởi động kho lưu trữ của mình trên máy tính thì nó không có Remote, vì vậy người dùng cần thêm Remote đó (Loại 2).

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 người dùng 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, người dùng 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 người dùng 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 người dùng truy cập GitHub, người dùng 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)

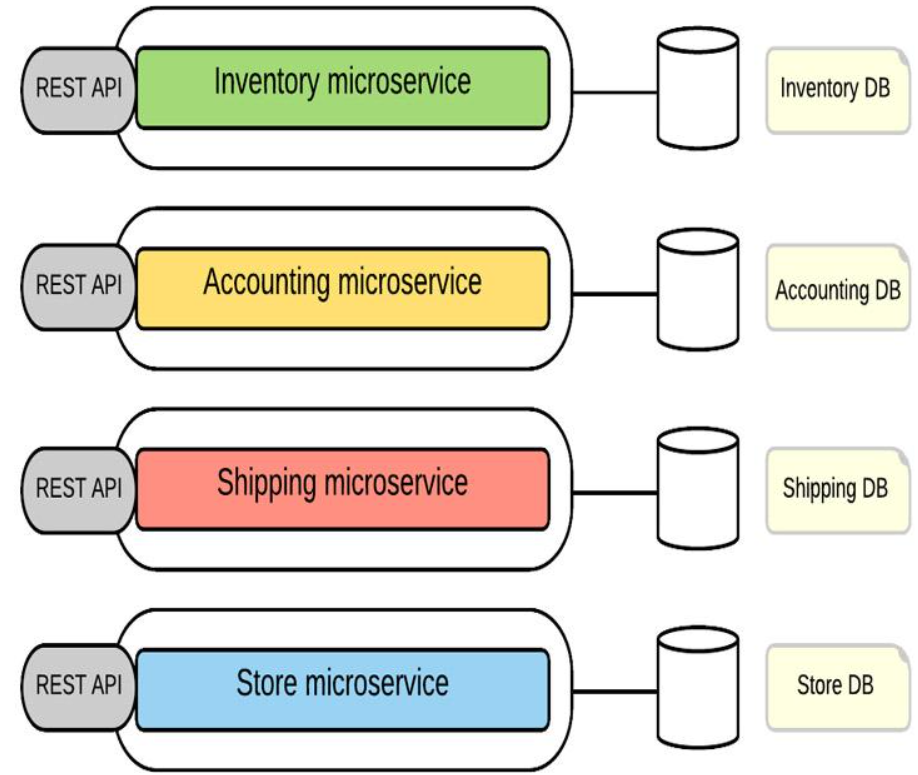
# **Chương 2: Microservices**

## **1. Khái niệm của Microservices**

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.

## **2. Kiến trúc của microservices**

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:

[](https://user-images.githubusercontent.com/107389856/173475379-a9a370fc-aac8-45fb-b170-4dd6dc637998.png)

Ví dụ như sau: nếu như người dùng 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)

## **3. Ưu điểm và hạn chế của microservices**

#### **3.1. Ưu điểm của microservices**

* Khi cần thay đổi một thành phần, thì chỉ cần sửa đổi, cập nhật và triển khai lại thành phần đó chứ không cần triển khai lại toàn bộ hệ thống.
* Dễ dàng mở rộng và tích hợp với các dịch vụ của bên thứ ba.
* Các microservice khởi động nhanh giúp quá trình phát triển, kiểm thử cũng nhanh hơn.
* Một microservice có thể được phát triển bởi một team nhỏ. Do vậy việc quản lý sẽ dễ dàng hơn.
* Dễ dàng thực hiện tự động tích hợp và tự động triển khai (CI-CD) bằng cách sử dụng một số công cụ như Jenkins, Hudson …
* Cho phép lập trình viên linh động hơn trong việc lựa chọn ngôn ngữ, công cụ và nền tảng để phát triển và triển khai các microservice (tuy nhiên trong một hệ thống, việc lựa chọn các ngôn ngữ khác nhau để phát triển các microservice không được khuyến khích)
* Mỗi microservice có kích thước nhỏ, giúp cho các lập trình viên dễ tiếp cận, đọc hiểu source code. Do vậy các thành viên mới tham gia team sẽ hòa nhập và đóng góp cho team nhanh hơn.
* Các microservice khởi động nhanh giúp quá trình phát triển, kiểm thử cũng nhanh hơn.
* Cô lập lỗi tốt hơn, khi một microservice bị lỗi và ngừng hoạt động thì các microservice khác vẫn có thể hoạt động bình thường. Với mô hình nguyên khối, một lỗi nhỏ có thể làm cả hệ thống ngừng hoạt động.

#### **3.2. Hạn chế của microservices**

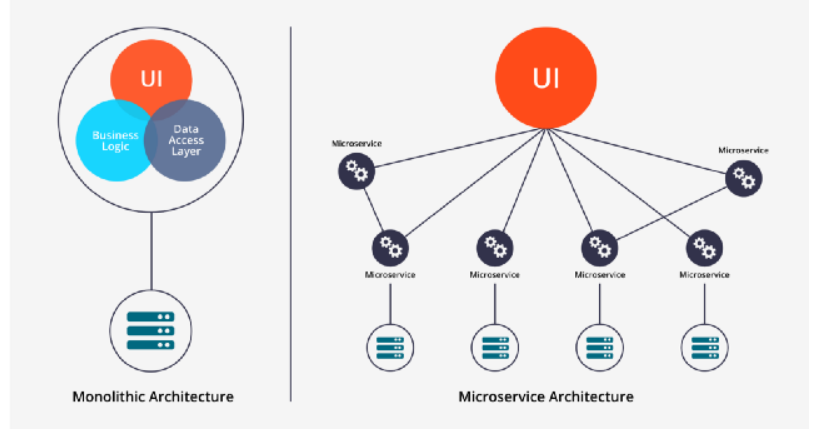
* Nhược điểm của kiến trúc microservice đến từ bản chất của hệ thống phân tán.
* Việc triển khai hệ thống microservice phức tạp hơn nhiều so với việc triển khai hệ thống nguyên khối.
* Các lập trình viên phải tốn nhiều công sức hơn để thực hiện phần giao tiếp giữa các microservice, với kiến trúc nguyên khối có khi họ chỉ cần gọi hàm để thực hiện việc này.
* Các microservice thường (nên) được triển khai bên trong docker container và giao tiếp với nhau qua REST API. Việc này làm hiệu năng của toàn bộ chương trình ứng dụng giảm xuống đáng kể do giới hạn tốc độ truyền tải của các giao thức và tốc độ mạng. Hơn nữa việc giao tiếp giữa các microservice có thể bị lỗi khi các kết nối bị lỗi.
* Cần tính toán kích cỡ của một microservice. Nếu một microservice quá lớn, bản thân nó trở thành một ứng dụng theo kiến trúc nguyên khối. Nếu một microservice quá nhỏ thì độ phức tạp của hệ thống tăng lên rất nhiều, làm cho hệ thống trở lên khó hiểu, lúc này việc quản lý giám sát và triển khai hệ thống sẽ khó khăn hơn.
* Khi ứng dụng ngày càng lớn lên, số lượng microservice ngày càng nhiều, các lập trình viên thường có xu hướng sử dụng sự hỗ trợ từ các công cụ mã nguồn mở, hoặc của bên thứ 3, việc sử dụng, tích hợp các công cụ này làm cho hệ thống khó kiểm soát và có thể bị dính các mã độc làm cho hệ thống kém an toàn.

## **4. Lợi ích của việc dùng microservices**

* 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 người dùng 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.

## **5. Sự khác biệt của kiến trúc microservices đối với các kiến trúc khác**

#### **5.1. Sự khác biệt giữa Microservices và Monolithic**

[](https://user-images.githubusercontent.com/107389856/173479161-4e645c2e-8dc4-4c1e-bfc7-d42783903312.png)

Biểu đồ ở trên giải thích sự khác biệt giữa kiến trúc Monolithic và kiến trúc Microservice rất rất đơn giản. Với kiến trúc Monolithic, bạn sẽ có 1 server lớn chịu trách nhiệm giải quyết tất cả các requests. Việc này sẽ gây khó khăn rất nhiều trên phương diện scale. Tuy nhiên, Microservices có thể cân bằng traffic theo nhu cầu của doanh nghiệp. Nếu đang nhận được 1 lượng lớn thanh toán, người dùng có thể scale up thiết bị thanh toán và giữ các dịch vụ khác ở mức sử dụng 1 lượng nhỏ hơn các services. Đây được gọi là scaling theo chiều ngang.

#### **5.2. Sự khác biệt giữa Microservices và API**

* API là tập hợp các phương thức giao tiếp, truyền thông được xác định một cách rõ ràng các thành phần khác nhau.
* Microservices là một kiến trúc có khả năng phân tách các thành phần của một ứng dụng (nguyên khối) thành các dịch vụ nhỏ hơn và có khả năng tự vận hành.

# **Chương 3: Spring Boot**

## **1. Khái niệm của Spring framework**

Spring là một framework được ra đời để giúp các nhà phát triển có thể xây dựng hệ thống và chạy ứng dụng trên JVM một cách thuận tiện, đơn giản và nhanh chóng. Đây là một mã nguồn mở được phát triển và rất nhiều người sử dụng.

Spring framework là tập hợp gồm rất nhiều các dự án nhỏ khác nhau như: Spring MVC (sử dụng để xây dựng các ứng dụng trên nền tảng web), Spring Data, Spring Boot,…

## **2. Khái niệm của Spring Boot**

Spring Boot là một dự án phát triển bởi JAV (ngôn ngữ java) trong hệ sinh thái Spring framework. Nó giúp cho các lập trình viên chúng ta đơn giản hóa quá trình lập trình một ứng dụng với Spring, chỉ tập trung vào việc phát triển business cho ứng dụng.

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.

[](https://user-images.githubusercontent.com/107389856/173720707-c20bff87-2276-48f4-9c96-d64a829e72a7.png)

## **3. Ưu điểm và hạn chế của Spring Boot**

#### **3.1. Ưu điểm của Spring Boot**

* Spring Boot được phát triển nhằm giúp người không có nhiều kiến thức lập trình vẫn có thể xây dựng ứng dụng.
* Giảm thời gian lập trình xuống tối thiểu.
* Gia tăng năng suất trong lập trình.
* 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.
* Tạo ứng dụng một cách độc lập, có thể chạy trên cả nền tảng Java Web.
* Ngoài ra còn có nhiều plugins để phát triển nhanh chóng bằng các công cụ như Build như Maven hoặc Gradle.
* Cung cấp nhiều plugin.

#### **3.2. Hạn chế của Spring Boot**

* Thiếu kiểm soát. Do style cố định, Spring Boot tạo ra nhiều phụ thuộc không được sử dụng dẫn đến kích thước tệp triển khai lớn.
* Quá trình chuyển đổi dự án Spring cũ hoặc hiện có thành các ứng dụng Spring Boot nhiều khó khăn và tốn thời gian.
* Không thích hợp cho các dự án quy mô lớn. Hoạt động liên tục với các microservices, theo nhiều nhà phát triển, Spring Boot không phù hợp để xây dựng các ứng dụng nguyên khối.

## **4. Tác dụng của Spring Boot**

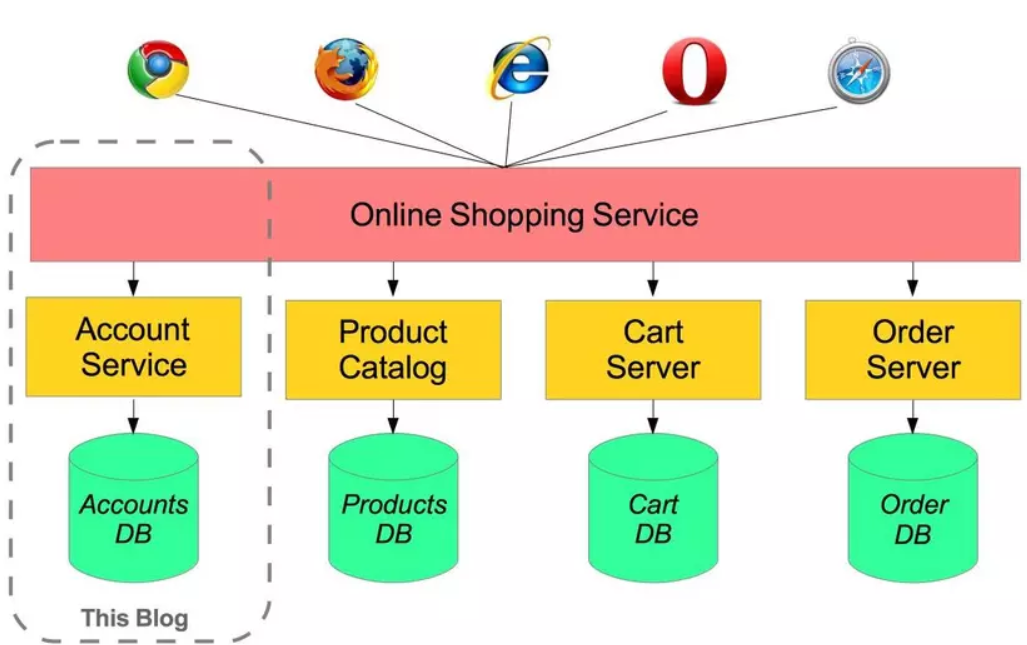
* Tự động cấu hình, có máy chủ nhúng, độc lập.
* Có nhiều tính năng vượt trội hơn các phần mềm khác.
* Code đơn giản, dễ học, dễ sử dụng.
* Dễ dàng tích hợp các mô-đun liên quan như Sping-MVC, Spring Data, Spring Sercurity, Spring Cloud,v.v…

## **5. Cách hoạt động Spring Boot trong microservices**

Spring Boot cho phép xây dựng các ứng dụng sẵn sàng cho sản xuất một cách nhanh chóng và cung cấp các tính năng phi chức năng:

* Máy chủ nhúng dễ dàng triển khai với các vùng chứa.
* Nó giúp theo dõi các thành phần bội số.
* Nó giúp định cấu hình các thành phần bên ngoài.

Spring Boot đã trở thành một yếu tố không thể thiếu của hệ sinh thái Java, cung cấp một bộ công cụ hiệu quả và có thể mở rộng để xây dựng các ứng dụng Spring với kiến trúc microservices. Do thiết lập mặc định cho các bài kiểm tra đơn vị và tích hợp, nó cho phép các nhà phát triển tận hưởng quá trình triển khai và phát triển được tăng tốc. Hơn nữa, Spring Boot giúp các nhà phát triển xây dựng các ứng dụng mạnh với cấu hình rõ ràng và an toàn mà không tốn nhiều thời gian và công sức để có thêm kiến thức về Spring.

[](https://user-images.githubusercontent.com/107389856/173720969-f84d595c-967a-4fa6-9f6a-4847598240e5.png)

# **Chương 4: jHipster**

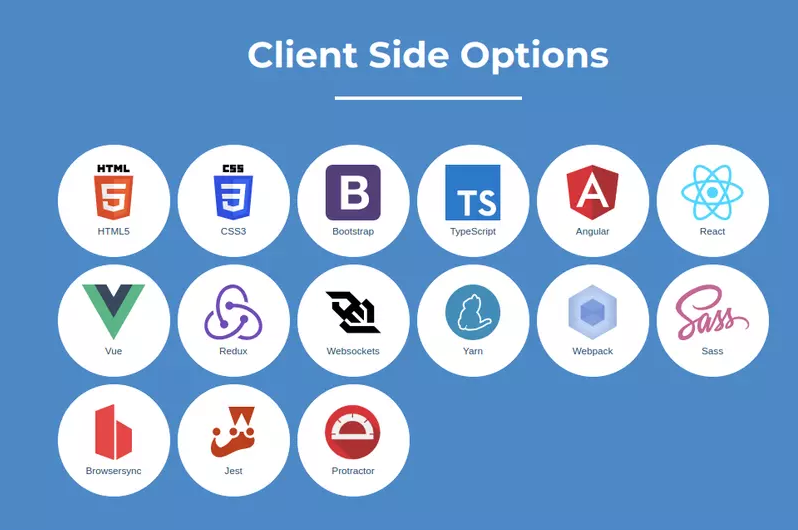
## **1. Khái niệm jHipster**

JHipster( viết tắt của Java Hipster ) là một phương pháp đơn giản để tạo ra một project xung quanh những công nghệ được ưa thích nhất với Spring technology, Angular/React và các microservices Spring.

* Server side xây dựng backend với công nghệ Spring Boot + Spring security.

[](https://user-images.githubusercontent.com/107389856/174565126-e0d02950-83e3-4da5-af70-2f5c01b75c6d.png)

* Client side với những Framework mạnh mẽ.

[](https://user-images.githubusercontent.com/107389856/174565258-ec967162-1958-4309-a6f0-73d80dd771d2.png)

* Deployment dự án dễ dàng.

[](https://user-images.githubusercontent.com/107389856/174565379-b7a25904-d79a-4f53-a832-bc86e0853f7b.png)

## **2. Cách khởi tạo Monolithic và Microservices**

#### **B1: Cài đặt java, node.js, Git**

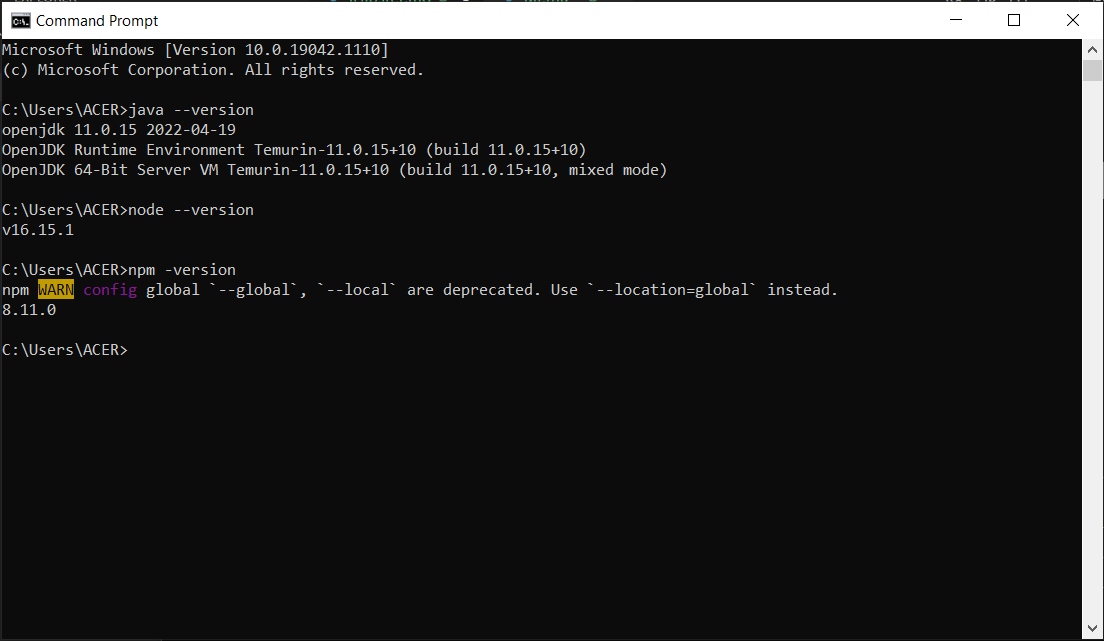
Windows -> Run -> cmd

Kiểm tra java:

* java --version

Kiểm tra node js:

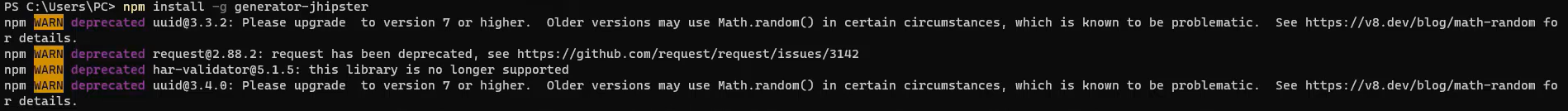
* node --version

[](https://user-images.githubusercontent.com/107389856/174553288-207f80c8-53bd-47d2-b4f3-c295b09a31d3.png)

#### **B2: Cài đặt Jhipster**

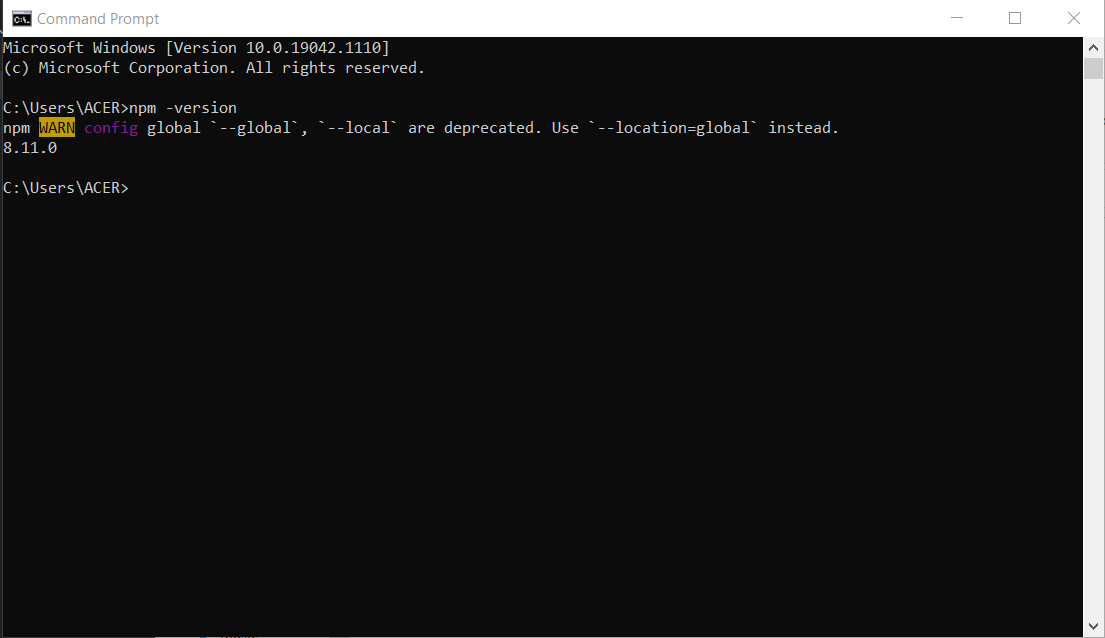
Lệnh cài đặt sau khi cài java, node.js, git:

* npm install -g generator-jhipster

[](https://user-images.githubusercontent.com/107389856/174553818-c8ba5351-ed98-4b32-b2b4-e1fd358a11cf.png)

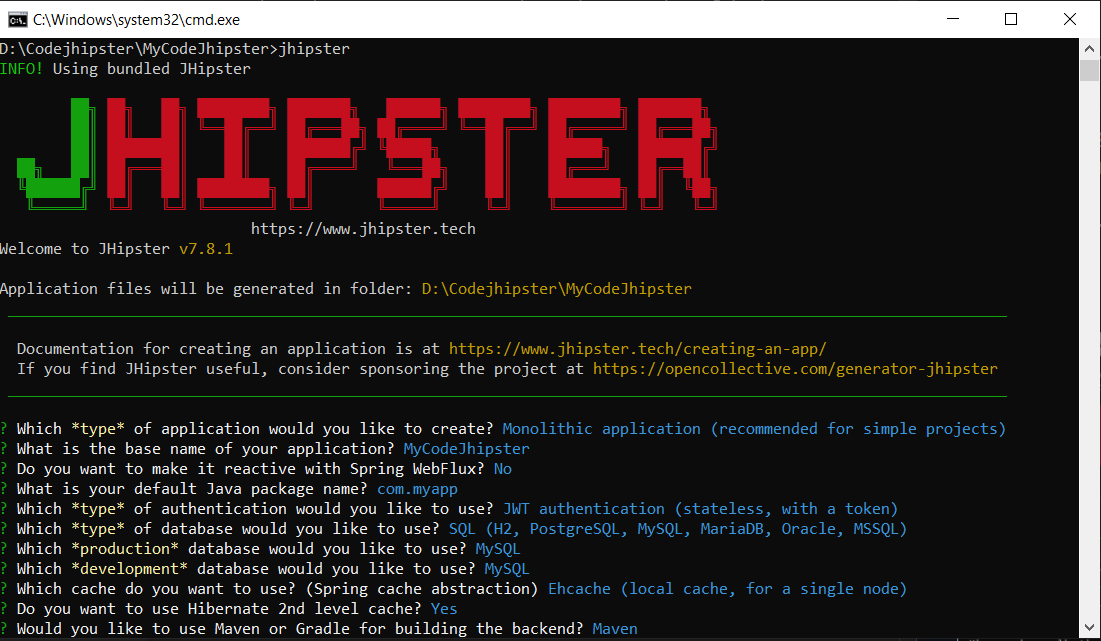
Lệnh kiểm tra cài đặt npm:

* npm -version

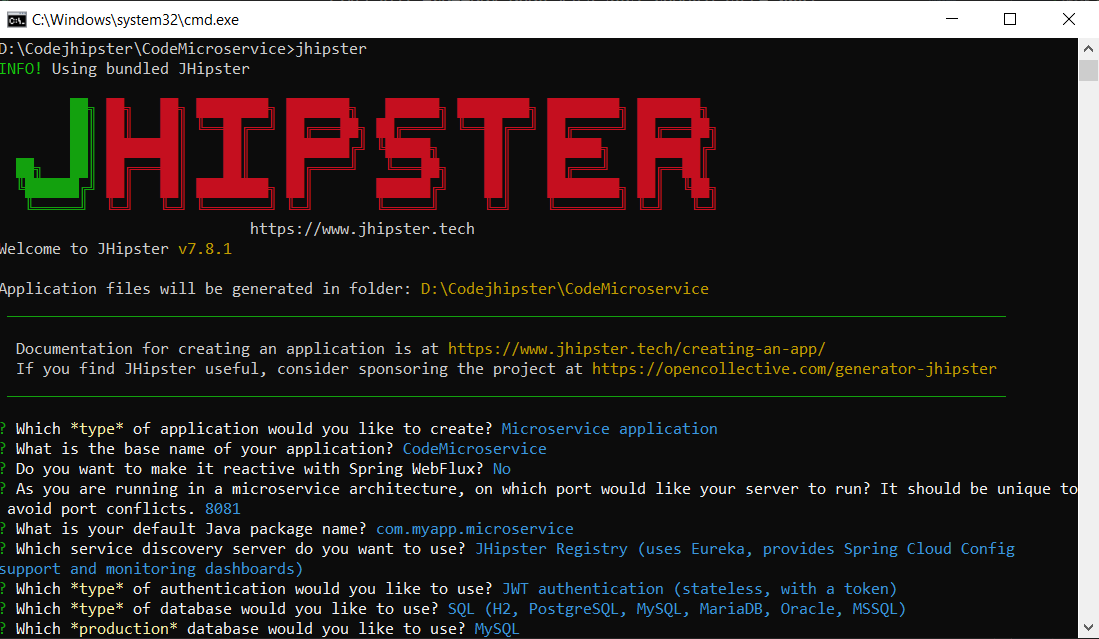
[](https://user-images.githubusercontent.com/107389856/174556415-dca1d2ae-9acc-48c7-b127-0b38aaedc58f.png)

#### **B3: Khởi tạo kiến trúc**

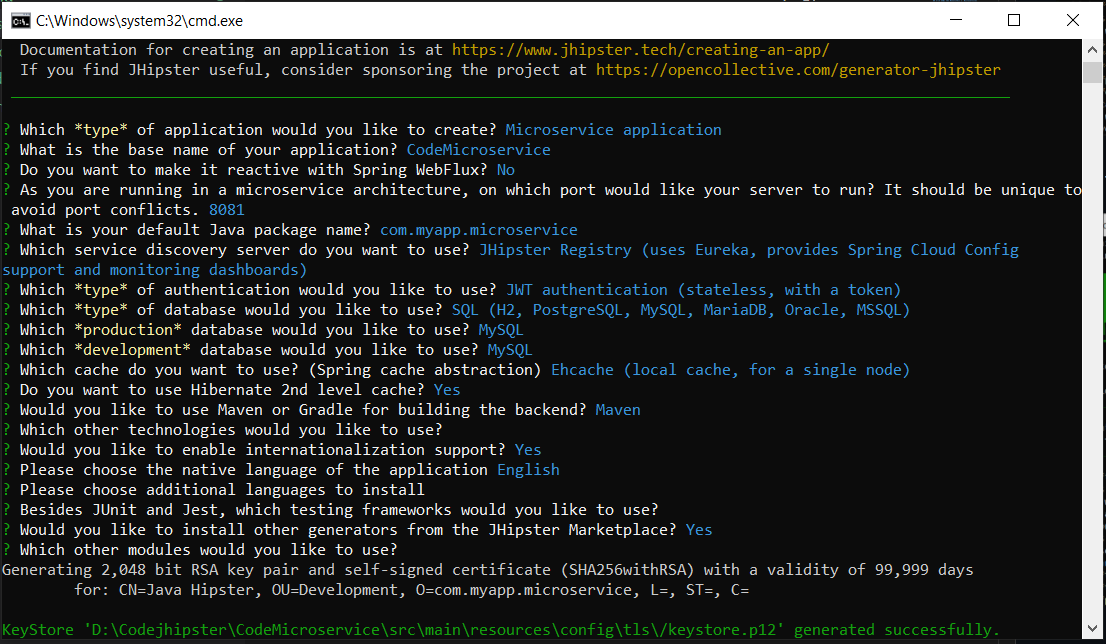
Cách tạo Monolithic

[](https://user-images.githubusercontent.com/107389856/174552971-fc0ff004-080d-435a-adc1-c899e157a55f.png)

Cách tạo Microservices

[](https://user-images.githubusercontent.com/107389856/174554904-23cf10ae-2c80-42d0-acd4-000808fa0e65.png)

## **3. Những câu hỏi cơ bản để khởi tạo project trong jhipster**

[](https://user-images.githubusercontent.com/107389856/174557813-7ffc1f0f-6605-403e-be69-b152a75a51bb.png)

* Lựa chọn kiến trúc: Microservices application

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.

* Tên project: CodeMicroservice
* Chọn port: 8081
* tên package: com.myapp.microservice
* Chọn máy chủ khám phá dịch vụ: Jhipster Registry

JHipster Registry là một ứng dụng thời gian chạy, sử dụng cấu trúc JHipster thông thường, trên đó tất cả các ứng dụng đăng ký và nhận cấu hình của chúng.

* Chọn xác thực: JWT authentication

JSON Web token (JWT) là một chuẩn mở (RFC 7519) định nghĩa một cách nhỏ gọn và khép kín để truyền một cách an toàn thông tin giữa các bên dưới dạng đối tượng JSON. Thông tin này có thể được xác minh và đáng tin cậy vì nó có chứa chữ ký số. JWTs có thể được ký bằng một thuật toán bí mật (với thuật toán HMAC) hoặc một public / private key sử dụng mã hoá RSA.

* Chọn database liên kết: MySQL
* Triển khai database: MySQL

MySQL là một hệ thống quản trị cơ sở dữ liệu mã nguồn mở (gọi tắt là RDBMS) hoạt động theo mô hình client-server. Với RDBMS là viết tắt của Relational Database Management System. MySQL được tích hợp apache, PHP. MySQL quản lý dữ liệu thông qua các cơ sở dữ liệu. Mỗi cơ sở dữ liệu có thể có nhiều bảng quan hệ chứa dữ liệu. MySQL cũng có cùng một cách truy xuất và mã lệnh tương tự với ngôn ngữ SQL.

* Chọn bộ nhớ đệm: Ehcache

Ehcache là một bộ đệm dựa trên tiêu chuẩn, mã nguồn mở để tăng hiệu suất, đơn giản hóa khả năng mở rộng và giảm tải cơ sở dữ liệu của người dùng. ehcache được sử dụng để cải thiện hiệu suất bằng cách giảm tải cho các tài nguyên bên dưới. nó cũng có thể được sử dụng để lưu vào bộ nhớ đệm máy chủ hoạt động ổn định, tính liên tục của ứng dụng và bộ nhớ đệm phân tán.

* Chọn thư viện maven hoặc gradle: maven

Maven là công cụ quản lý và thiết lập tự động 1 dự án phần mềm. Chủ yếu dùng cho các lập trình viên java, nhưng nó cũng có thể được dùng để xây dựng và quản lý các dự án dùng C#, Ruby, Scala hay ngôn ngữ khác.

* Chọn công nghệ: Angular

Angular là một javascript framework do google phát triển để xây dựng các Single Page Application (SPA) bằng JavaScript , HTML và TypeScript . Angular cung cấp các tính năng tích hợp cho animation , http service và có các tính năng như auto-complete , navigation , toolbar , menus ,… Code được viết bằng TypeScript , biên dịch thành JavaScript và hiển thị tương tự trong trình duyệt.

Project được tạo ra trên hình theo các công nghệ:

* Chọn ngôn ngữ App: English
* Chọn ngôn ngữ tải về: Vietnamese

Project được tạo ra trên hình theo các công nghệ:

* Backend: Spring Boot + Spring security
* Database:

- MySQL(production)

- MySQL(development)

* Frontend: Bootstrap + SASS + Angular

## **4. Tác dụng của việc lựa chọn**

Các lựa chọn trên giúp người dùng lựa chọn và sử dụng các công nghệ hiện đại theo ý người dùng để phát triển.

* Tạo ra một project nhanh chóng dễ sử dụng.
* Tạo Microservices Với JHipster.
* Tạo cổng API.
* Tạo thực thể.
* Thêm Logic kinh doanh.
* Thực hiện các cải tiến về giao diện người dùng.
* Cho phép HTML.
* Tạo một dịch vụ nhỏ.
* Tạo thực thể sản phẩm.
* Triển khai lên đám mây.

## **5. Tác dụng của công nghệ jHipster**

* Một công cụ rất hữu ích giúp các người dùng có thể tạo ra 1 project java web một cách nhanh chóng và đầy đủ các chức năng cần thiết.
* Phía Backend người dùng thể dùng nhiều công nghệ như Spring boot, Spring Sercurity, Maven, Grandle,..
* Phía Frontend người dùng có thể dùng các framework như React, Angular, VueJs, ...
* Người dùng cũng có thể sử dụng nhiều loại cơ sở dữ liệu khác nhau cả Sql và NoSql như MySql, Cassandra, MongoDb,...
* Sau khi generate code người dùng còn có thể tùy chọn việc deloy code của người dùng lên server, Jhipster hỗ trợ nhiều cách khác nhau như: Docker, Aws, HeroKu, Google Cloud Flatform,...

# **Sơ Đồ Dự Án**

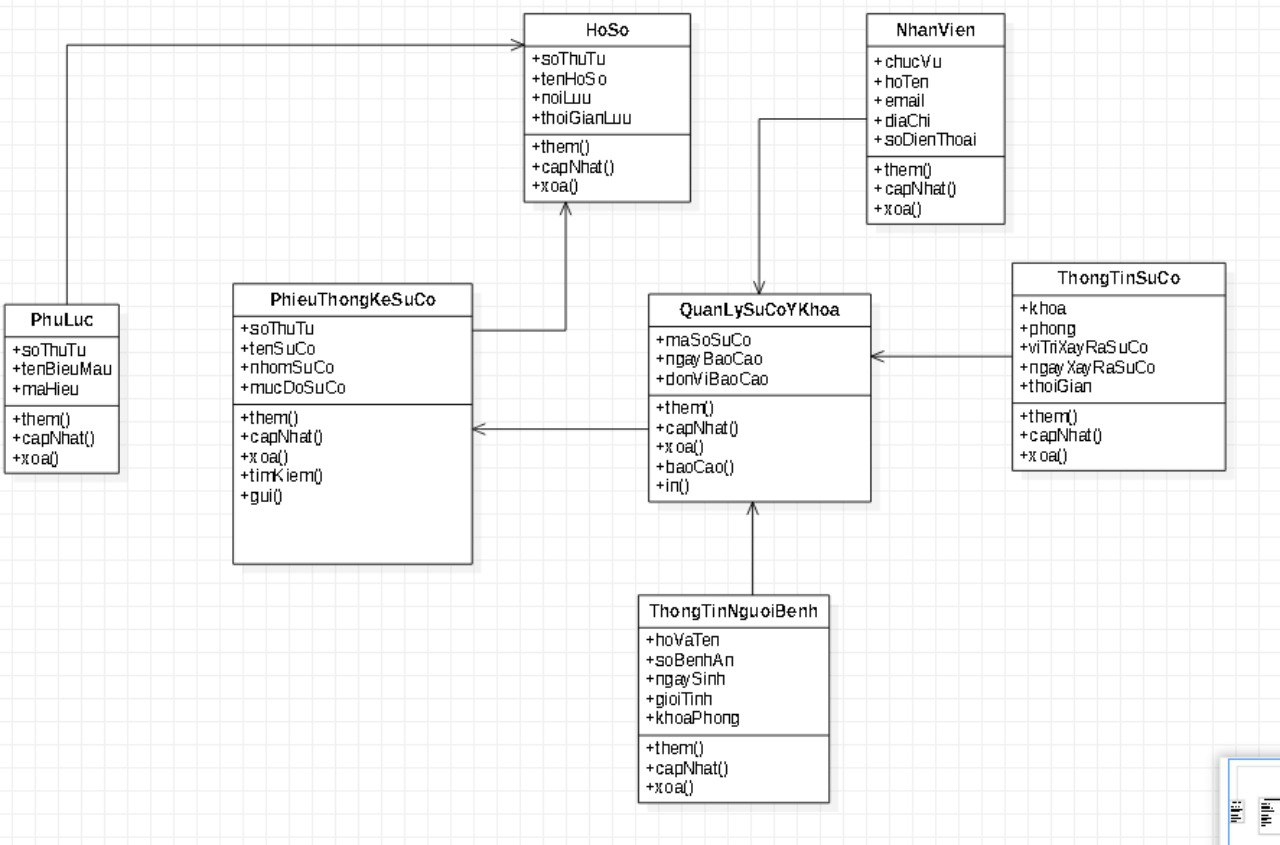
## **1. Sơ đồ Usecase**

QUY TRÌNH GIẢI QUYẾT VÀ BÁO CÁO SỰ CỐ Y KHOA QT.17.HT

[](https://user-images.githubusercontent.com/107389856/176847715-29765978-8392-4ff7-944f-e5d6adc60879.png)

## **2. Sơ đồ Class**

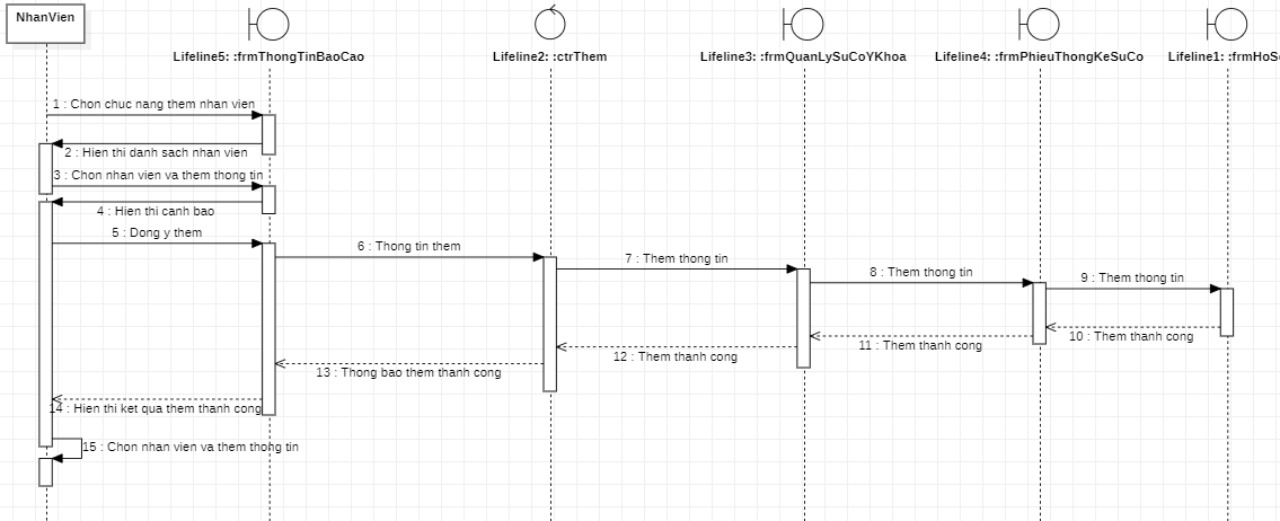
QUY TRÌNH GIẢI QUYẾT VÀ BÁO CÁO SỰ CỐ Y KHOA QT.17.HT

[](https://user-images.githubusercontent.com/107389856/176847790-3ca0c036-62f3-4755-b6b5-c24c26109cd4.png)

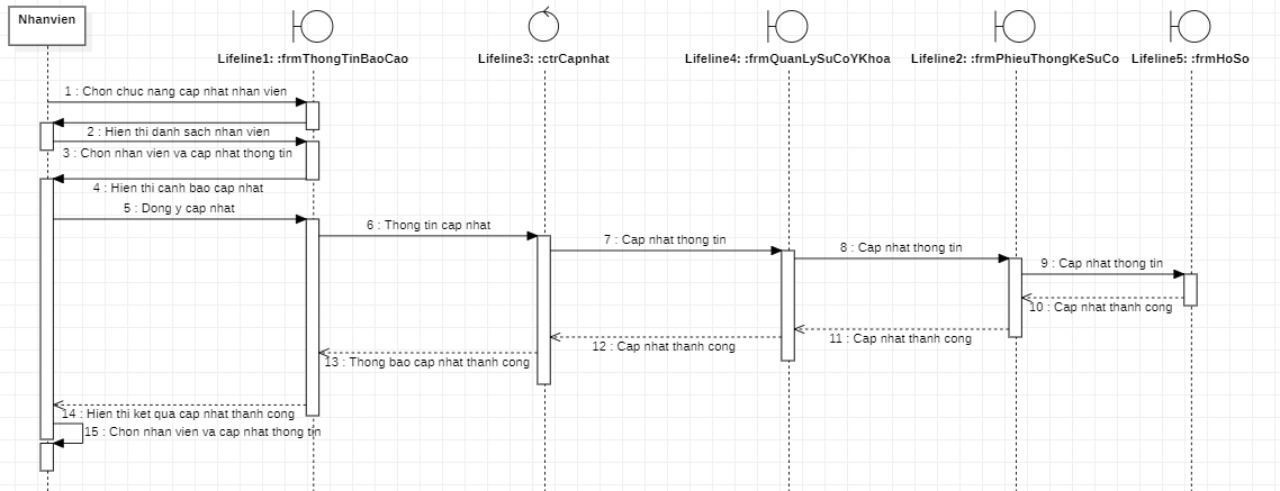
## **3. Sơ đồ tuần tự**

QUY TRÌNH GIẢI QUYẾT VÀ BÁO CÁO SỰ CỐ Y KHOA QT.17.HT

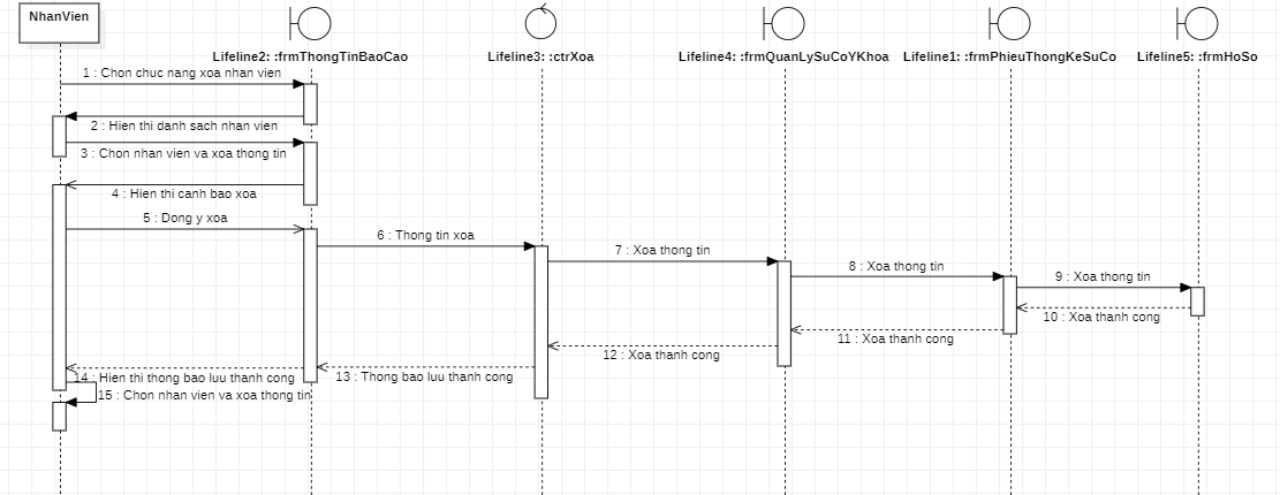
#### **3.1. Thêm thông tin**

[](https://user-images.githubusercontent.com/107389856/176848011-de7a3d65-d400-44cc-bcea-5d8f137246bc.png)

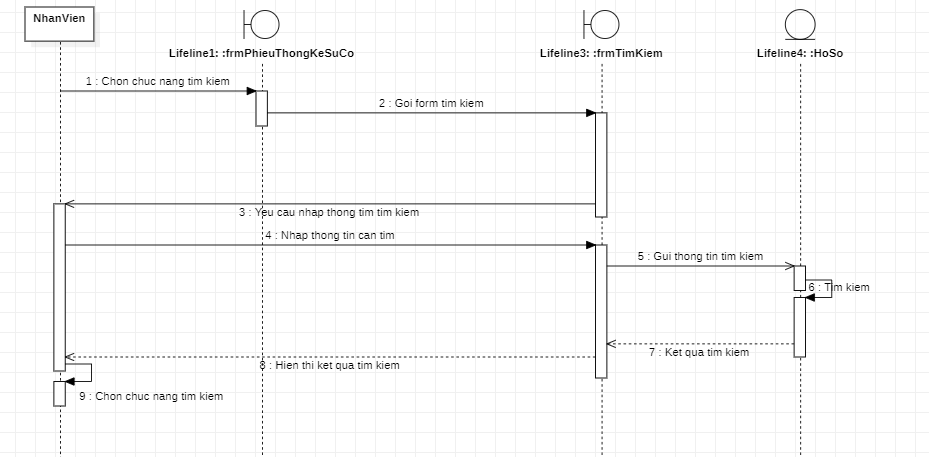
#### **3.2. Cập nhật thông tin**

[](https://user-images.githubusercontent.com/107389856/176848086-851f7a87-0294-4aba-a594-af14001238b2.png)

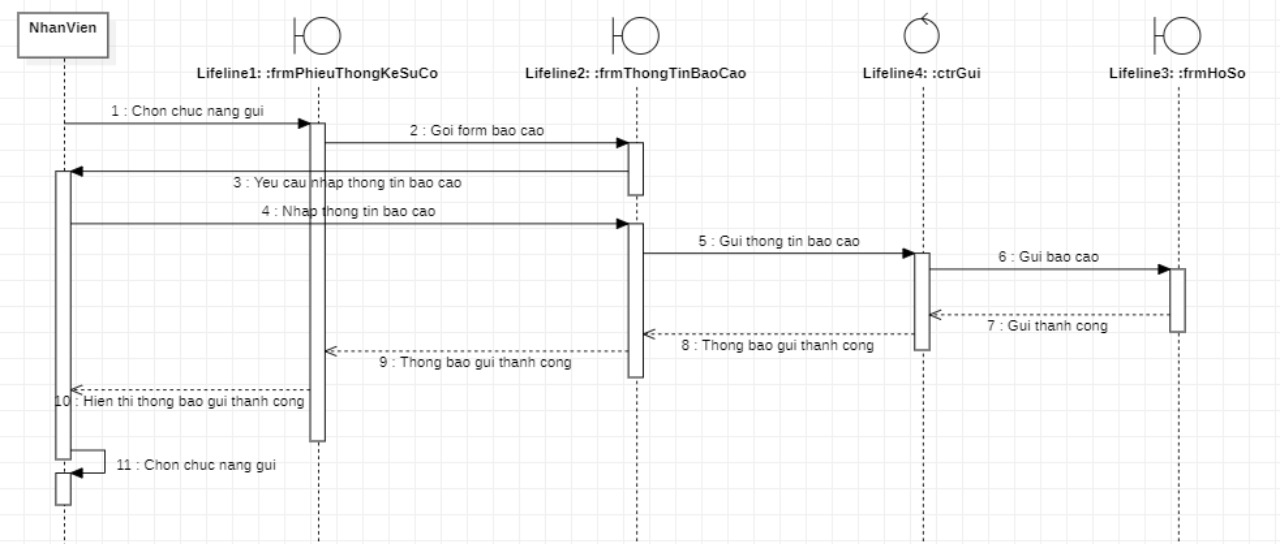
#### **3.3. Xóa thông tin**

[](https://user-images.githubusercontent.com/107389856/176848150-b7da1e44-1c63-48b0-8587-50acc6ebb674.png)

#### **3.4. Tìm kiếm thông tin**

[](https://user-images.githubusercontent.com/107389856/176848642-cc4174e3-c38f-47a2-84ea-0370b0888308.png)

#### **3.5. Gửi thông tin báo cáo**

[](https://user-images.githubusercontent.com/107389856/176848549-d6cf038b-f6fe-472b-9abc-7b53beecc94b.png)

# **Chương 5: JDL**

## **1. Khái niệm JDL**

JDL (JHipster Domain Language) là ngôn ngữ miền dành riêng cho JHipster, nơi người dùng có thể mô tả tất cả các ứng dụng, triển khai, thực thể và các mối quan hệ của chúng trong một tệp duy nhất (hoặc nhiều hơn một) với cú pháp thân thiện với người dùng.

## **2. Tổng quát về JDL**

Người dùng có thể sử dụng JDL-Studio trực tuyến của chúng tôi hoặc một trong các plugin tiện ích mở rộng JHipster IDE, có sẵn cho:

* Eclipse
* Visual studio code

Để tạo tệp JDL và trực quan hóa UML của nó. Người dùng cũng có thể tạo và xuất hoặc chia sẻ URL của mô hình JDL của người dùng.

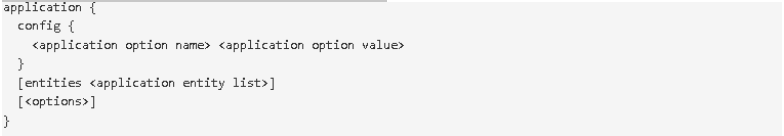
Điều này có thể được sử dụng để thay thế cho việc sử dụng trình tạo phụ thực thể và là cách tiếp cận được khuyến nghị. Ý tưởng là việc quản lý các mối quan hệ bằng cách sử dụng công cụ trực quan sẽ dễ dàng hơn nhiều so với các câu hỏi và câu trả lời cổ điển của Yeoman.

Dự án JDL có sẵn trên GitHub, nó là một dự án Mã nguồn mở giống như JHipster (Giấy phép Apache 2.0). Nó cũng có thể được sử dụng như một thư viện nút để phân tích cú pháp JDL.

## **3. Các ứng dụng**

#### **3.1. Cú pháp:**

Việc khai báo đơn được thực hiện như sau:

[](https://user-images.githubusercontent.com/107389856/177963471-2c6de657-4529-444f-a770-bcbc3b1a0d06.png)

* Các khóa / giá trị cấu hình ứng dụng được chỉ định bên dưới config(phải ở bên trong application)
* Có thể có 0, 1 hoặc bất kỳ tùy chọn ứng dụng nào người dùng muốn (miễn là chúng hợp lệ)
* Các thực thể sẽ được tạo bên trong ứng dụng được liệt kê thông qua entities, đây là cách được khuyến nghị để tạo các thực thể trong ứng dụng.
* Điều này có thể được bỏ qua nhưng việc tạo các thực thể bên trong ứng dụng sẽ yêu cầu thực hiện điều đó:từ một tệp JDL khác bên trong ứng dụng hoặc với CLI.
* Từ entities khóa là tùy chọn: người dùng có thể bỏ qua nó, nhưng mọi thực thể trong tệp JDL sẽ được tạo bên trong ứng dụng.

#### **3.2. Các tùy chọn trong ứng dụng**

Khai báo tùy chọn ( dto,, v.v. service) skipServerđược hỗ trợ trong các ứng dụng JDL, nhưng với một số quy tắc.

Giả sử chúng ta có tệp JDL này :

[](https://user-images.githubusercontent.com/107389856/177963784-1cbb28d4-28ea-420e-819e-1158fe1e69fe.png)

Trong mẫu này, chúng ta có thể thấy một số điều: A, B, C, D, E and F có 6 thực thể được khai báo trong tệp JDL: chúng tôi có 3 ứng dụng:app1, app2 and app3 app1 sử dụng A, B and C app2 sử dụng C and D app3 sử dụng E( \* except A, B, C, D, F) Mỗi ứng dụng này khai báo các tùy chọn và một tùy chọn chung cũng được khai báo. app1 sử dụng dto cho A, B and C app2 sử dụng paginate cho C(vì có một ngoại lệ) app3 sử dụng service cho E Toàn cầu cũng sử dụng pagination(cho mọi thực thể). Đây là cách tệp được tạo:

app1 A: sẽ sử dụng paginate with infinite-scroll (tùy chọn chung không bị ghi đè bởi tùy chọn cục bộ) và dto with mapstruct B: sẽ sử dụng các tùy chọn tương tự C: cũng sẽ sử dụng các tùy chọn tương tự app2: C: sẽ sử dụng paginate with pagination (và không infinite-scroll, vì cái cục bộ được ưu tiên hơn) D: sẽ sử dụng paginate with infinite-scroll như tùy chọn trước đó không bao gồm D app3: E: sẽ paginate with infinite-scrollvàservice E with serviceClass Ví dụ này minh họa nguyên tắc đổ bóng . Các tùy chọn chung được hỗ trợ và sẽ được sử dụng bởi mọi ứng dụng đã khai báo trừ khi các tùy chọn cũng được khai báo trong các ứng dụng.

Cũng lưu ý đoạn mã này được lấy từ mẫu trước đó trong app3:

entities \* except A, B, C, D, F service \* with serviceClass

Về cơ bản, điều này có nghĩa là app3 sẽ chỉ sử dụng E và các thực thể của ứng dụng sẽ sử dụng service tùy chọn, có nghĩa là E và không A to F.

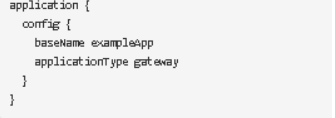
Cuối cùng, F thực thể không có trong bất kỳ ứng dụng nào và thực thể này sẽ không được tạo vì điều đó.

#### **3.3. Các ví dụ**

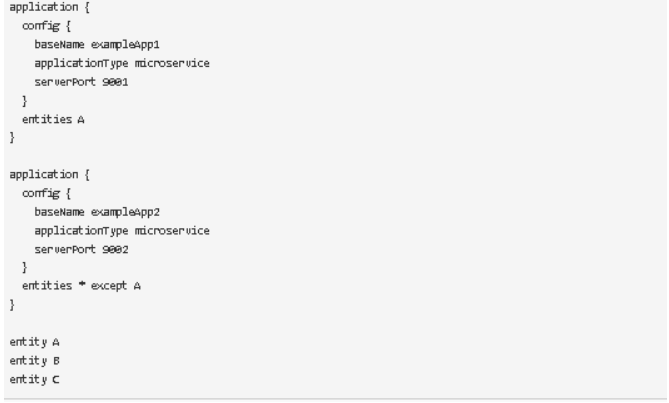
Ví dụ cơ bản Permalink to "Basic example"

[](https://user-images.githubusercontent.com/107389856/177964499-85829a6e-134c-4630-9ed4-9e4bff1f9239.png)

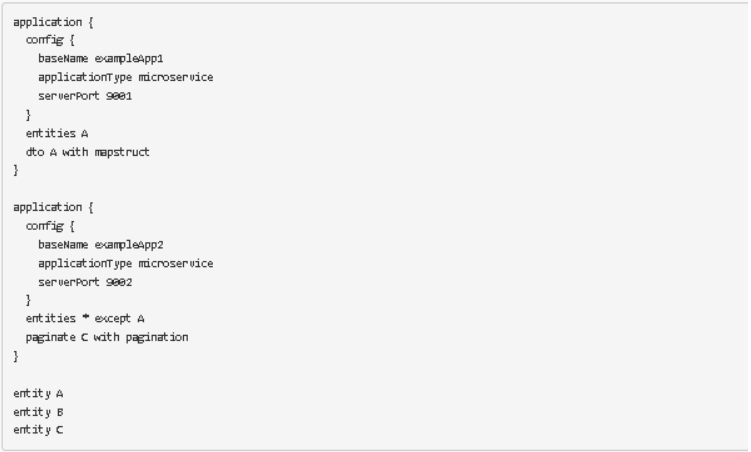
Nhiều ứng dụng

[](https://user-images.githubusercontent.com/107389856/177964551-a9d6f1a3-a152-4163-b53e-83a040ad1615.png)

Với các thực thể

[](https://user-images.githubusercontent.com/107389856/177964666-fb93ead7-3279-4bfb-a76a-e3fac4b5b92e.png)

Với các tùy chọn

[](https://user-images.githubusercontent.com/107389856/177964761-2476e592-3887-4b5c-91b2-46faba05891d.png)

#### **3.4. Toàn bộ bảng phân tích ví dụ**

[](https://user-images.githubusercontent.com/107389856/177964871-5ab0c887-4e21-42be-812f-4e115f70c185.png)

Bây giờ, một số điều sẽ xảy ra khi tạo các ứng dụng và thư mục này:

Bốn ứng dụng sẽ được tạo:

myMonolith trong ./myMonolith, với cổng máy chủ 8080 myGateway trong ./myGateway, với cổng máy chủ 9042 microserviceA trong ./microserviceA, với cổng máy chủ 8081

Mặc dù chúng tôi không chỉ định cổng máy chủ, JHipster đặt một cổng theo mặc định.

Đối với microservices, mặc định là 8081.

Đối với cổng và đá nguyên khối, nó 8080 microserviceB trong ./microservice B với cổng máy chủ 8082.

Bốn thực thể sẽ được tạo

A và Btrong nguyên khối

C và Dcả trong cửa ngõ

C trong microservice đầu tiên

D trong microservice thứ hai

Tùy microservice chọn này ngầm hiểu cho C và D

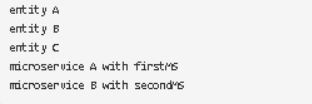
Vì chúng được tạo trên hai microservices nên tùy chọn này sẽ được đặt theo mặc định.

Các tùy chọn hoạt động giống như trước đây

Lưu ý rằng trình tạo đặt các giá trị mặc định nếu chúng không có (như databaseType). JHipster Core làm những điều tương tự cho người dùng.

Đối phó với microservices là một công việc gần như phức tạp, nhưng JDL cung cấp cho người dùng một số tùy chọn để xử lý các thực thể của người dùng khi người dùng thấy phù hợp. Với microservice with <MICROSERVICE\_APP\_NAME>người dùng, người dùng có thể chỉ định thực thể nào được tạo trong microservice nào.

Lấy ví dụ như thiết lập này:

[](https://user-images.githubusercontent.com/107389856/177965374-52dc9505-dc6c-4849-adc1-4070625f2f9f.png)

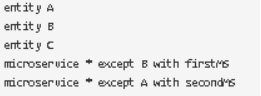
Với hai ứng dụng JHipster ('firstMS' và 'secondMS'), đây là những gì người dùng sẽ nhận được nếu nhập tệp JDL trong hai ứng dụng:

Trong 'firstMS', các thực thể Avà Csẽ được tạo. Trong 'secondMS', các thực thể Bvà Csẽ được tạo. Cđược tạo cả hai vì nếu không có tùy chọn microservice chỉ định nơi thực thể này được tạo, nó sẽ được tạo ở mọi nơi.

Nếu người dùng quyết định nhập JDL này trong một ứng dụng nguyên khối, mọi thực thể sẽ được tạo (nguyên khối không có tùy chọn hạn chế trong JDL).

Lưu ý: nếu người dùng muốn tạo cùng một thực thể trong hai dịch vụ nhỏ khác nhau, người dùng có thể ghi hai tệp JDL thay vì cập nhật tệp JDL. Mỗi lần.

Ví dụ trước không thể được viết như thế này:

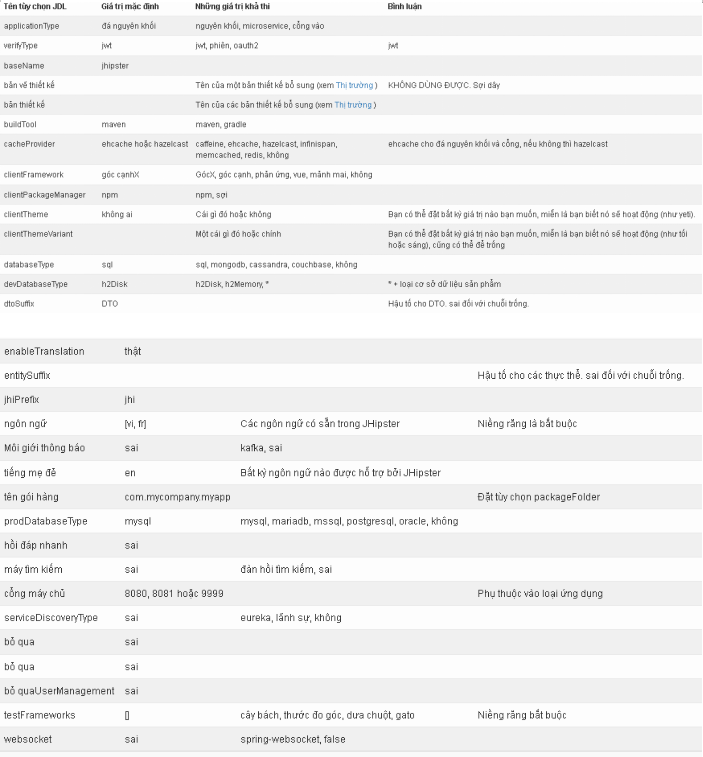
[](https://user-images.githubusercontent.com/107389856/177965516-beae6444-f282-4486-9c14-fedc4656f9bd.png)

Đây là kết quả:

Trong 'firstMS', chỉ thực thể Csẽ được tạo Trong 'secondMS', các thực thể Bvà Csẽ được tạo. Đó là bởi vì, tại thời điểm phân tích cú pháp, nếu một tùy chọn trùng lặp với một tùy chọn khác, thì tùy chọn sau sẽ được ưu tiên hơn. người dùng cũng có thể tạo toàn bộ ngăn xếp microservice bằng JDL, hãy xem bài đăng trên blog này chẳng hạn.

#### **3.5. Các tùy chọn cấu hình ứng dụng có sẵn**

Dưới đây là các tùy chọn ứng dụng được hỗ trợ trong JDL:

[](https://user-images.githubusercontent.com/107389856/177965690-e0116dcc-c1d3-4e1f-82f0-d2b40780e7b2.png)

## **4. Entity**

#### **4.1. Cú pháp**

Việc khai báo thực thể được thực hiện như sau:

[](https://user-images.githubusercontent.com/107389856/177965877-5b08e6f0-4880-4ed1-9169-2a0f746e1af4.png)

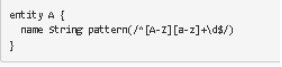
Tên của thực thể, tên của một trường của thực thể, loại trường được hỗ trợ JHipster, và như một tùy chọn: tài liệu của thực thể, các tùy chọn cho thực thể (xem Tùy chọn để có danh sách đầy đủ các tùy chọn có sẵn),

tên bảng cơ sở dữ liệu (nếu người dùng muốn chỉ định một cái gì đó khác mà tên được tính tự động từ tên thực thể), tài liệu của lĩnh vực này, các tùy chọn cho trường, các xác nhận cho trường.

Tuyên bố Blob JHipster cung cấp một sự lựa chọn tuyệt vời vì người ta có thể chọn giữa một loại hình ảnh hoặc bất kỳ loại nhị phân nào. JDL cho phép người dùng làm điều tương tự. Tạo một loại tùy chỉnh (xem DataType) bằng trình chỉnh sửa, đặt tên cho nó theo các quy ước sau:

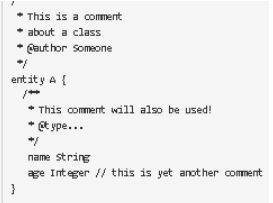
AnyBlob hoặc Blob để tạo một trường thuộc loại nhị phân "bất kỳ"; ImageBlob để tạo một trường có nghĩa là một hình ảnh. TextBlob để tạo trường cho CLOB (văn bản dài). Và người dùng có thể tạo bao nhiêu DataTypes tùy thích.

Biểu thức chính quy Đây là một xác thực nhất định (chỉ có sẵn cho các loại Chuỗi) và cú pháp của nó là:

[](https://user-images.githubusercontent.com/107389856/177966101-07d3d806-d077-4b2e-ab36-46b24c35cb16.png)

Hãy chia nhỏ nó ra:

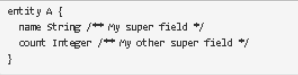
pattern là từ khóa để khai báo xác thực regex (với dấu ngoặc đơn bình thường) /.../mẫu được khai báo bên trong hai dấu gạch chéo \chống chém không cần thiết phải thoát Bình luận Có thể nhận xét trong JDL cho các thực thể và trường, và sẽ tạo ra tài liệu (Javadoc hoặc JSDoc, tùy thuộc vào phần phụ trợ).

[](https://user-images.githubusercontent.com/107389856/177966195-1e9c04ae-d668-49c4-a73b-9b87a7d5d3ca.png)

Những bình luận này sau đó sẽ được thêm vào dưới dạng bình luận Javadoc của JHipster. JDL sở hữu loại nhận xét riêng của nó:

// một bình luận bị bỏ qua / \*\* không phải là một bình luận bị bỏ qua \* / Do đó, bất kỳ thứ gì bắt đầu bằng //đều được coi là nhận xét nội bộ cho JDL và sẽ không được tính là Javadoc. Xin lưu ý rằng các lệnh JDL Studio bắt đầu bằng #sẽ bị bỏ qua trong quá trình phân tích cú pháp.

Một dạng nhận xét khác là các nhận xét sau:

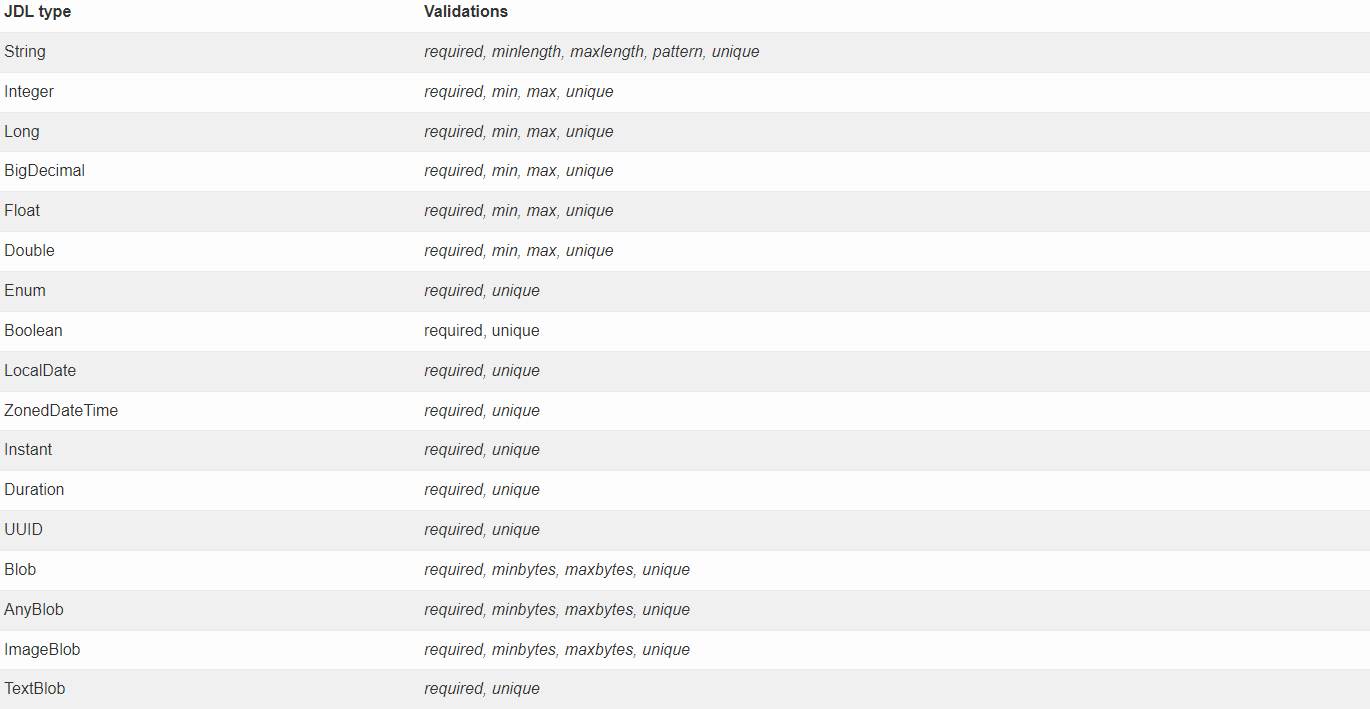
[](https://user-images.githubusercontent.com/107389856/177966276-527e94e2-c4e8-406c-a3b3-dd407bf5dfb3.png)

Ở đây tên của A sẽ được nhận xét với My super field, B với My other super field.

Có, dấu phẩy không phải là bắt buộc nhưng sẽ khôn ngoan hơn nếu có chúng để không mắc lỗi trong mã. Nếu bạn muốn kết hợp dấu phẩy và nhận xét sau, hãy cẩn thận!

[](https://user-images.githubusercontent.com/107389856/177966375-bde5bb31-ec90-4e0c-937c-e0bac1ba9339.png)

#### **4.2. Các loại trường và xác thực**

[](https://user-images.githubusercontent.com/107389856/177966613-60aa541d-4859-4b14-8617-06a7b95b8d76.png)

## **5. EnumsPermalink to " JHipster Domain Language (JDL) - Enums"**

#### **5.1. Cú pháp**

Việc khai báo điều tra được thực hiện như sau:

[IMG_272](https://user-images.githubusercontent.com/107389856/177966773-9e81bffa-32c8-44f7-a6c4-b8be191df0a7.png)

Giá trị mục nhập liệt kê là bắt buộc Và các phím chữ hoa phải được sử dụng Giá trị mục nhập liệt kê là tùy chọn và phải được đặt bên trong dấu ngoặc đơn

## **6. Mối quan hệ**

#### **6.1. Các kiểu quan hệ**

Được đề cập sau relationship từ khóa.

Có bốn kiểu quan hệ:

* OneToOne
* OneToMany
* ManyToOne
* ManyToMany

Để biết thêm về các mối quan hệ và những gì có thể đạt được, bạn có thể truy cập trang dành riêng.

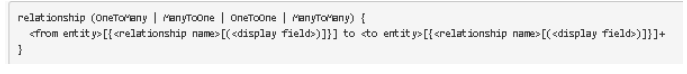
#### **6.2. Các phương pháp quan hệ**

Cú pháp này thực sự hữu ích khi:

Bạn có nhiều mối quan hệ cùng loại.

Bạn không muốn mất thời gian tìm kiếm chúng trong (các) tệp JDL của mình.

#### **6.3. Cú pháp**

[](https://user-images.githubusercontent.com/107389856/177967160-e7f565c5-6444-4f67-9333-c19b8712fd2b.png)

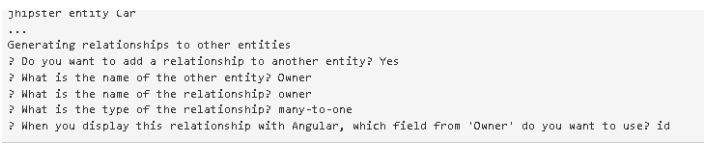
(OneToMany | ManyToOne| OneToOne | ManyToMany)là kiểu mối quan hệ của bạn.

* là tên của chủ sở hữu thực thể của mối quan hệ: nguồn.
* là tên của thực thể mà mối quan hệ đi đến: đích.
* là tên của trường có đầu kia là loại.
* là tên của trường sẽ hiển thị trong các hộp chọn (mặc định id:).

required trường được tiêm có được yêu cầu hay không. with jpaDerivedIdentifier có @MapsId được sử dụng cho liên kết hay không (chỉ áp dụng cho một đối một)

OneToMany

Tạo Car:

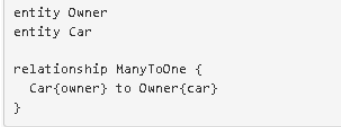
[](https://user-images.githubusercontent.com/107389856/177967400-ad0805aa-c238-4561-865d-199448b94d63.png)

Điều tương tự cũng có thể đạt được bằng cách sử dụng JDL bên dưới:

[](https://user-images.githubusercontent.com/107389856/177967458-a2d32da4-a59d-47b2-b3e4-80b722af1338.png)

ManyToOne

Điều này tương đương với mối quan hệ một-nhiều hai chiều sau khi đảo các cạnh trong tệp JDL:

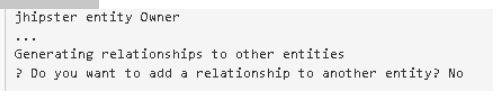
[](https://user-images.githubusercontent.com/107389856/177967531-aaaaa617-0596-4b33-8252-4f3e163f6848.png)

Thực hiện mối quan hệ đó vì hai lý do:

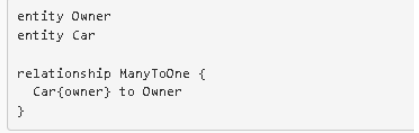
Theo quan điểm kinh doanh, bạn chỉ sử dụng các thực thể của mình theo cách này. Vì vậy, bạn không muốn có một API cho phép các nhà phát triển làm điều gì đó không có ý nghĩa.

Bạn có một mức tăng hiệu suất nhỏ khi sử dụng Owner thực thể.

Tạo Owner:

[](https://user-images.githubusercontent.com/107389856/177967653-a8491177-5a22-459f-9a1f-d282c238e59f.png)

Trên giao diện người dùng ứng dụng Angular / React được tạo, bạn sẽ có một menu thả xuống Car để chọn Owner. Đây là JDL tương ứng:

[](https://user-images.githubusercontent.com/107389856/177967725-4fc97853-d9bd-4a31-a5d1-88f7593c9f40.png)

OneToMany

Mối quan hệ một chiều một-nhiều có nghĩa là Ownercá thể có thể nhận được bộ sưu tập xe của nó, nhưng không phải ngược lại. Nó ngược lại với ví dụ trước.

[IMG_279](https://user-images.githubusercontent.com/107389856/177967816-c5382e61-8cbb-43b1-855d-040c5ae96c53.png)

Loại mối quan hệ này không được cung cấp theo mặc định trong JHipster vào lúc này, hãy xem # 1569 để biết thêm thông tin. Bạn có hai giải pháp cho việc này:

Thực hiện ánh xạ hai chiều và sử dụng nó mà không cần sửa đổi: đây là cách tiếp cận được đề xuất của chúng tôi, vì nó đơn giản hơn nhiều

Thực hiện ánh xạ hai chiều, sau đó sửa đổi nó để biến nó thành ánh xạ một hướng:

Xóa thuộc tính “mappedBy” trên @OneToManychú thích của bạn Tạo bảng tham gia bắt buộc: bạn có thể thực hiện mvn liquibase:diffđể tạo bảng đó, xem tài liệu về cách sử dụng Liquibase diff

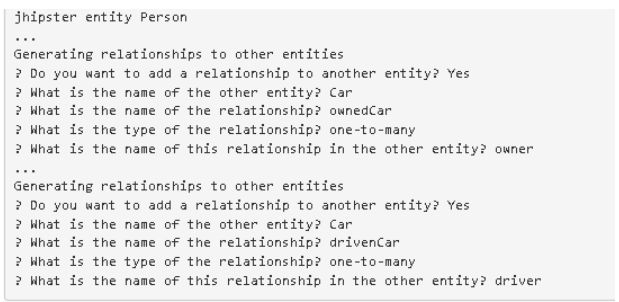
Điều này không được hỗ trợ với JDL vì nó không có trong JHipster.

TwoOneToMany

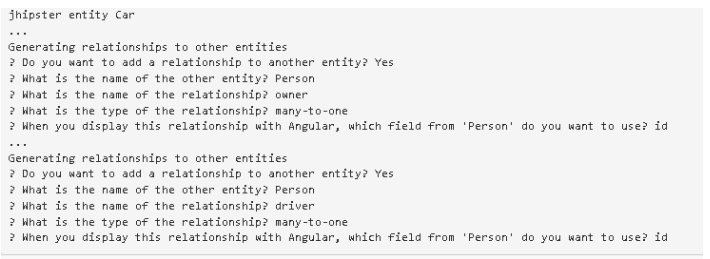
Đối với ví dụ này, a Person có thể là chủ sở hữu của nhiều ô tô và cũng có thể là người điều khiển nhiều ô tô:

[IMG_280](https://user-images.githubusercontent.com/107389856/177967930-60f45cb5-4a1a-4448-a6e6-1aea745012b7.png)

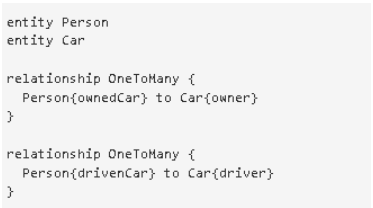
Tạo Person thực thể, có hai mối quan hệ một-nhiều với Carthực thể:

[](https://user-images.githubusercontent.com/107389856/177967959-d7f12393-1c07-4618-a461-adaaafc21b65.png)

Tạo Carthực thể, sử dụng cùng một tên quan hệ đã được định cấu hình trong Person thực thể:

[](https://user-images.githubusercontent.com/107389856/177967988-bcdef774-4ab7-4225-bda6-9e5b70b449a8.png)

Điều tương tự cũng có thể đạt được bằng cách sử dụng JDL bên dưới:

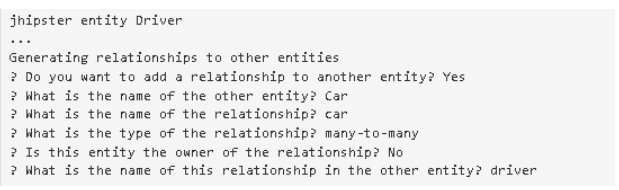
[](https://user-images.githubusercontent.com/107389856/177968223-1ae301b6-84b4-4b4d-8573-bf6dce4bdaaa.png)

ManyToMany

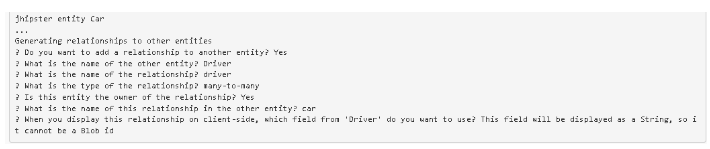
A Drivercó thể lái nhiều ô tô, nhưng một Carcan cũng có nhiều tài xế.

[IMG_284](https://user-images.githubusercontent.com/107389856/177968239-ab961fe7-b630-4438-9481-7b765f492532.png)

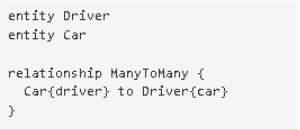
Tạo ra mặt không sở hữu của mối quan hệ, Drivervới mối quan hệ nhiều-nhiều:

[](https://user-images.githubusercontent.com/107389856/177968255-d9d081d6-e176-44f8-b8b1-47b0a3d5f873.png)

Tạo Car, với mặt riêng của mối quan hệ nhiều-nhiều:

[](https://user-images.githubusercontent.com/107389856/177968278-e6a95c14-cb12-47c0-aa0e-9cd6935c1dc3.png)

Điều tương tự cũng có thể đạt được bằng cách sử dụng JDL bên dưới

[](https://user-images.githubusercontent.com/107389856/177968303-26d46546-8e7d-4557-a280-6bdb2a0de71f.png)

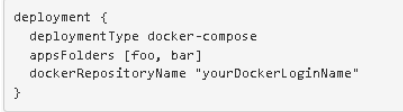
## **7. Triển khai**

#### **7.1. Cú pháp**

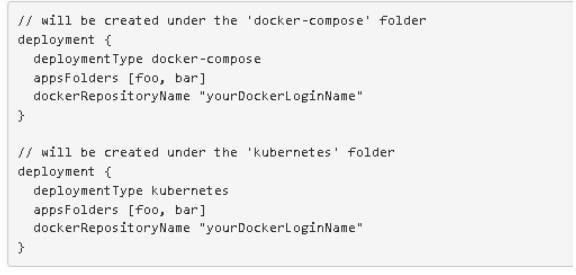
Khai báo triển khai được thực hiện như sau:

[](https://user-images.githubusercontent.com/107389856/177968334-5bf9b8c5-04fa-4fe1-987f-46e0f8112170.png)

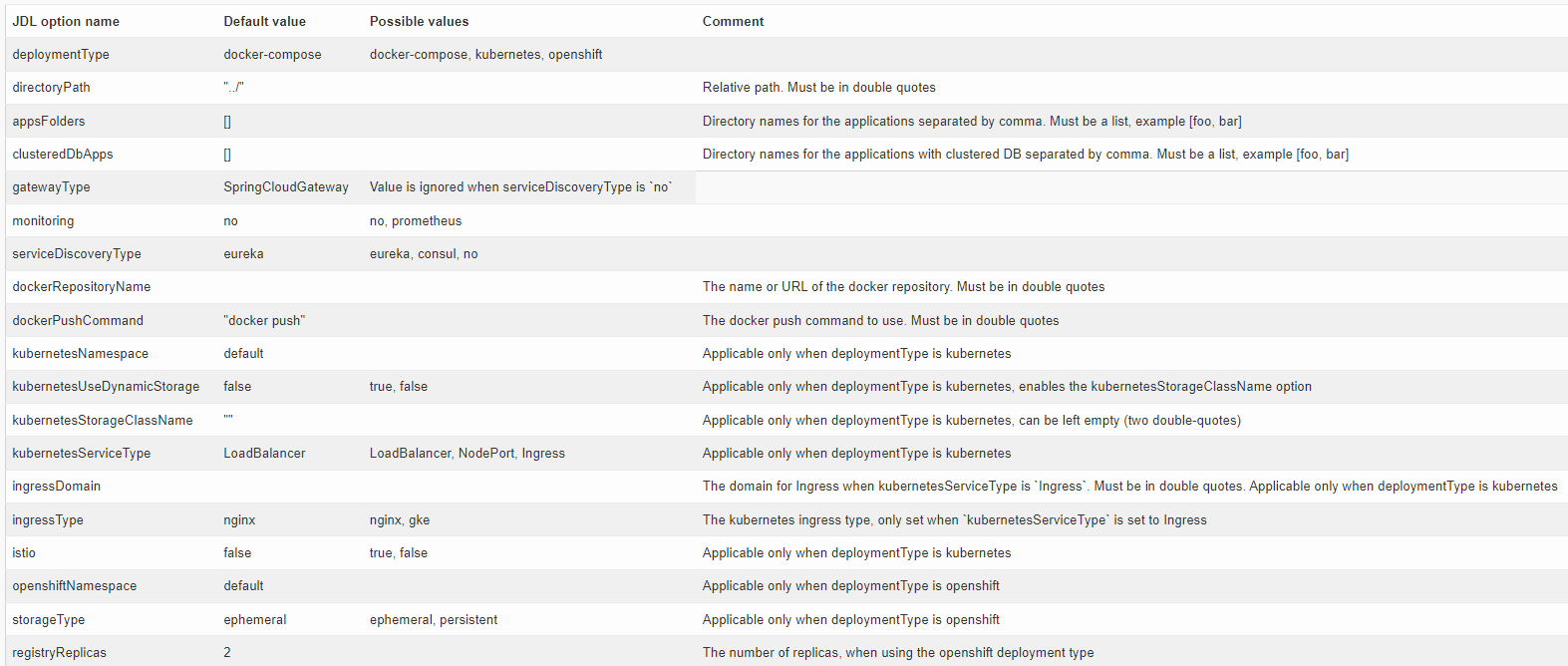
Các ví dụPermalink to "Examples"

[](https://user-images.githubusercontent.com/107389856/177968382-67dbef97-0564-446c-b206-18925bffd54e.png)

Nhiều triển khai

[](https://user-images.githubusercontent.com/107389856/177968420-6cd44343-8ca2-450e-b855-8d17be1ef3db.png)

Các tùy chọn triển khai có sẵnPermalink to "Available deployment options"

[](https://user-images.githubusercontent.com/107389856/177968606-ac0ce4a1-92d7-4667-a6e8-12a98792aced.png)

Xử lý sự cố

Chúng tôi đã cố gắng giữ cho cú pháp thân thiện nhất có thể đối với các nhà phát triển. Bạn có thể làm những điều này với nó:

Khai báo các ứng dụng với các tùy chọn và thực thể của chúng,

Khai báo các thực thể với các thuộc tính của chúng,

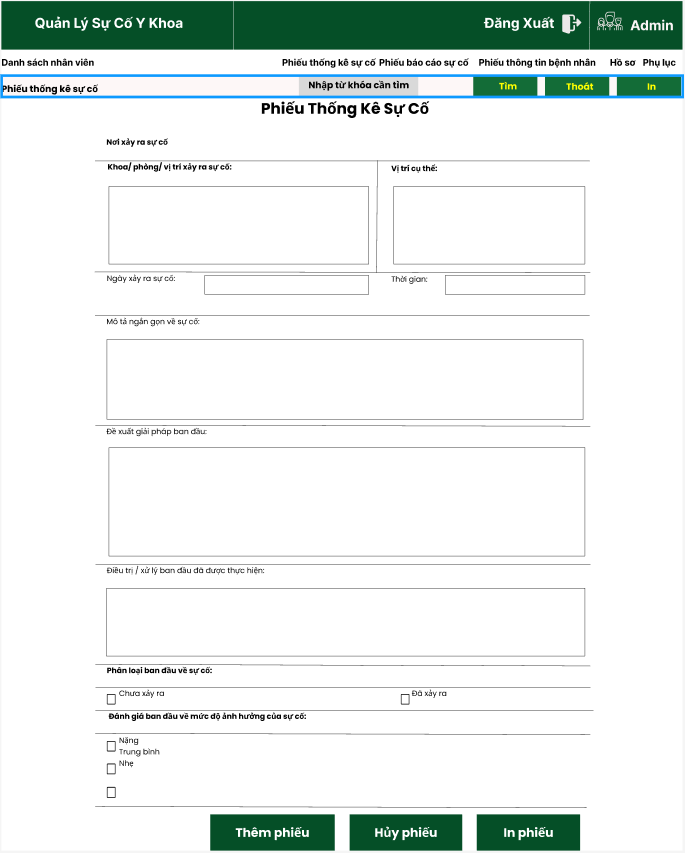
Khai báo các mối quan hệ giữa chúng,

Và khai báo một số tùy chọn cụ thể của JHipster.

# **Chương 6: Thiết Kế Giao Diện**

## **1. Quản Lý Sự Cố Y Khoa**

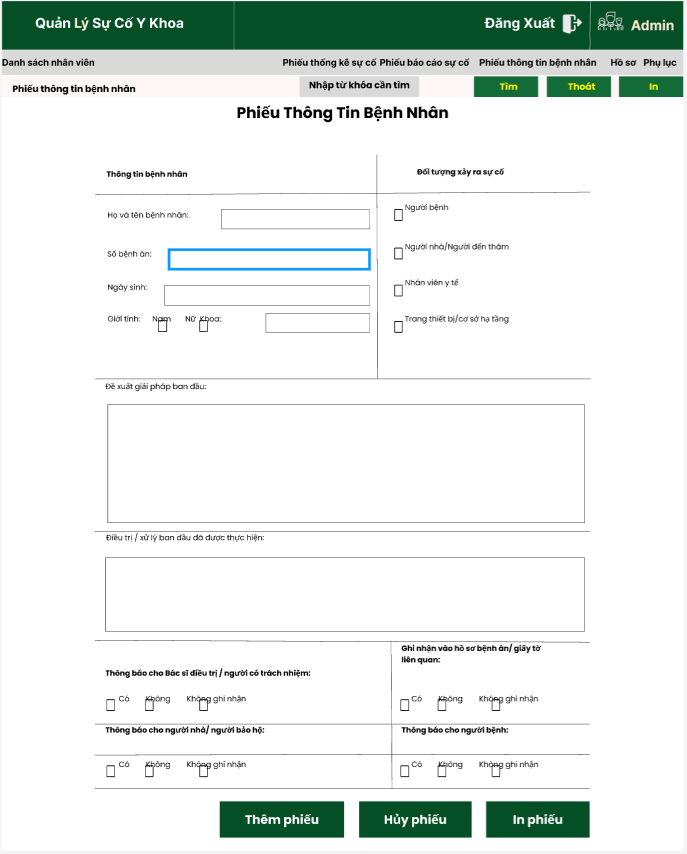
#### **1.1. Phiếu thống kê sự cố**



**1.2. Phiếu báo cáo sự cố**



**1.3. Phiếu thông tin bệnh nhân**



## **2. Quản Lý Nhân Viên**

