**I.GIỚI THIỆU GIT**

**1.Định nghĩa 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 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.

**2.Chức năng 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 thức hoạt động của GIT**

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 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.

**6.Hạn chế 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.

**7.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

**8.Các lệnh GIT cơ bản là:**

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 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.

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 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”

1. 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 người dùng 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:>

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 bạn.

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 bạn 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 bạn 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

**9.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.

**10.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.

**11.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.

**12.Cách sử dụng GIT**

1. 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.

1. 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 người dùng 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)”

1. 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 người dùng 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, người dùng 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 người dùng. 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

* :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.

\_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)

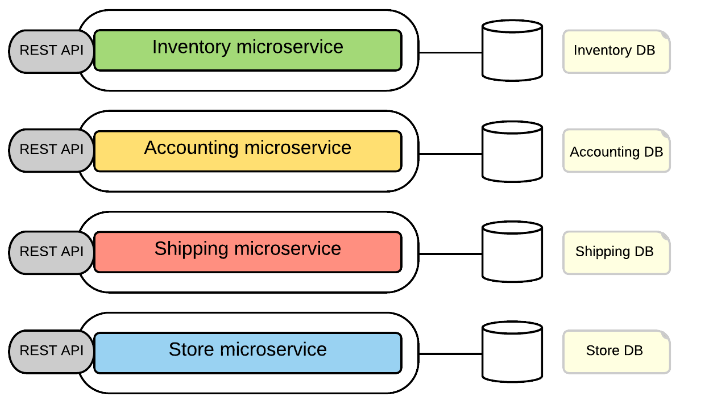
**II.GIỚI THIỆU Microservices**

**1.Định nghĩa 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. 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.

**2.Mục đích sử dụng Microservices**

1.Trong kiến trúc Microservices, các service (dự án) tồn tại độc lập nhau về xử lý (process), lưu trữ (database) và request (client/server model)

[](https://user-images.githubusercontent.com/107390350/173478197-407c1a20-bab3-4203-aeb6-c0d4c6dee01b.png)

2.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). Trong đó table tc\_order chỉ chứa khóa ngoại – tức ID của Nhân viên, ID của cửa hàng, ID của nhà kho theo đúng tinh thần thiết kế chuẩn hóa database.

**3.Ưu điểm khi dùng Microservices**

1.Microservices cho phép dễ dàng continuous delivery và deployment các ứng dụng lớn và phức tạp hơn.

2.Có thể cải thiện khả năng bảo trì: bởi vì các service tương đối nhỏ nên dễ hiểu và dễ thay đổi hơn.

3.Có khả năng testing dễ dàng: nhờ các services nhỏ

4.Có thể triển khai tốt hơn: các services thường rất dễ cho việc triển khai độc lập.

5.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.

6.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.

7.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.

**4.Nhược điểm khi dùng Microservices**

1.Bởi vì các nhà phát triển thường xuyên phải đối phó với sự phức tạp khi tạo ra một hệ thống phân tán.

2.Cần phải implement việc communication giữa các inter-services.

3.Handle partial failure rất phức tạp bởi vì luồng xử lý cần phải đi qua nhiều service khác nhau.

4.Khi thực hiện các requests trải rộng trên nhiều service khó khăn thì điều này cần đòi hỏi sự phối hợp giữa các team.

5.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.

6.Việc triển khai và quản lý microservices nếu như làm thủ công theo cách làm với ứng dụng thì sẽ rất phức tạp.

7.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.

**5.Sự khác biệt giữa kiến trúc Microservices với kiến trúc Monolithic**

Monolithic có những tính chất và ưu nhược điểm như:

Tính chất : Kiến trúc một khối là mẫu thiết kế được dùng nhiều nhất trong giới lập trình web hiện nay bởi tính đơn giản của nó khi phát triển và khi deploy. Mặc dù mỗi ứng dụng sẽ được triển khai theo những cách khác nhau, nhưng nhìn chung thì khi ứng dụng được lập trình theo kiến trúc một khối, kết quả thường đạt được như sau:

1.Nó có thể hỗ trợ nhiều loại client như browser hay các app native trên cả desktop và mobile.

2.Nó có thể expose API cho bên thứ ba.

3.Nó có thể tích hợp với các ứng dụng khác thông qua REST/SOAP web service hoặc các message queue.

4.Nó có thể xử lý các request dạng HTTP, thực hiện logic nghiệp vụ, truy cập cơ sở dữ liệu và có thể trao đổi dữ liệu với các hệ thống khác.

5.Nó có thể chạy trên các container như Tomcat, JBoss,..

6.Nó có thể scale theo chiều ngang (vertical scalability) bằng cách tăng sức mạnh của phần cứng hoặc scale theo chiều dọc (horizontal scalability) bằng cách load balancers.

Về ưu điểm thì Monolithic có những ưu điểm sau:

1.Dễ phát triển vì các stack công nghệ thống nhất ở tất cả các layer.

2.Dễ test do toàn bộ project được đóng gói trong một package nên dễ dàng chạy test integrantion và test end-to-end.

3.Deploy đơn giản và nhanh chóng nếu bạn chỉ có một package để bận tâm.

4.Dễ scale vì chúng ta có thể có nhiều instance cho load banlancer.

5.Yêu cầu team size nhỏ cho việc maintain app.

6.Team member có thể chia sẻ ít nhiều về skill.

7.Tech stack đơn giản và đa số là dễ học.

8.Phát triển ban đầu nhanh hơn do đó có thể đem sale hoặc marketing nhanh hơn.

9.Yêu cầu cơ sở hạ tầng đơn giản. Thậm chí một container đơn giản cũng đủ để chạy ứng dụng.

Về ưu điểm thì Monolithic có những nhược điểm sau:

1.Các component được liên kết chặt chẽ với nhau dẫn đến side effect không mong muốn như khi thay đổi một component ảnh hưởng đến một component khác.

2.Theo thời gian thì project trở nên phức tạp và lớn dần. Các tính năng mới sẽ mất nhiều thời gian hơn để phát triển và tái cấu trúc các tính năng hiện có sẽ nhiều khó khăn hơn. Toàn bộ ứng dụng cần được triển khai lại cho bất kỳ thay đổi nào.

3.Không hề dễ để hiểu project do các module liên quan chặt chẽ lẫn nhau. Một issue nhỏ cũng có thể làm chết toàn bộ ứng dụng.

4.Áp dụng công nghệ mới khó khăn vì toàn bộ ứng dụng phải thay đổi. Do đó nhiều ứng dụng một khối thường phụ thuộc một công nghệ cũ và lỗi thời.

5.Các service quan trọng không thể scale riêng dẫn đến lãng phí tài nguyên vì toàn bộ ứng dụng phải scale theo.

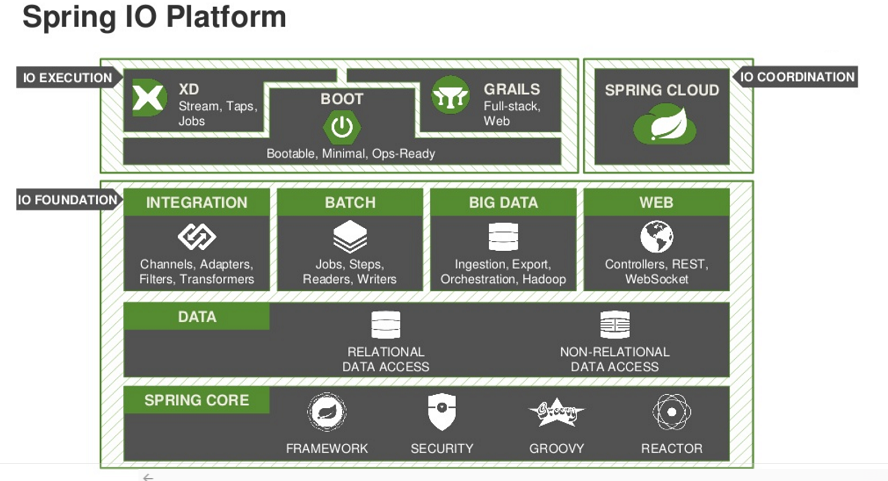
6.Các ứng dụng một khối lớn sẽ có thời gian khởi động lâu và tốn tài nguyên CPU cũng như bộ nhớ.

7.Các team tham gia vào dự án phải phụ thuộc lẫn nhau và tất khó để mở rộng quy mô team.

Kết luận sự khác biệt giữa kiến trúc Microservices với kiến trúc Monolithic là

Microservice là tốt, nhưng không phải cho tất cả các loại ứng dụng. Mẫu này hoạt động tuyệt vời để phát triển các ứng dụng và hệ thống phức tạp. Cân nhắc chọn kiến trúc microservice khi có nhiều nhóm có kinh nghiệm và khi ứng dụng đủ phức tạp để chia nó thành các dịch vụ. Khi ứng dụng lớn và cần phải linh hoạt và có thể mở rộng, microservice có lợi.

**III.Spring Boot**

[](https://user-images.githubusercontent.com/107390350/173716162-24ab7e6a-7093-4872-893e-cb4b1cd78b66.png)

**1.Đinh nghĩa 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.

Để phát triển một ứng dụng web cơ bản HelloWorld sử dụng Spring framework người dùng sẽ cần ít nhất 5 công đoạn sau;

Tạo một project sử dụng Maven với các dependency cần thiết của Spring MVC và Servlet API. Một tập tin web.xml để khai báo DispatcherServlet của Spring MVC. Một tập tin cấu hình của Spring MVC. Một class Controller trả về một trang “Hello World” khi có request đến. Cuối cùng là phải có một web server dùng để triển khai ứng dụng lên chạy. Trong các công đoạn này, chỉ có công đoạn tạo một class Controller thì có thể khác cho các ứng dụng khác nhau vì mỗi ứng dụng có một yêu cầu khác nhau. Còn các công đoạn khác thì như nhau.Giờ đây với Spring Boot chúng ta có thể tạo dự án Spring một cách nhanh chóng và cấu hình cũng đơn giản dùng Sublime Text để phát triển luôn khỏi cần cài đặt eclipse hay netbean nặng.

**2.Ưu điểm của Spring Boot**

1.Tạo các ứng dụng Spring độc lập.

2.Nhúng trực tiếp Tomcat, Jetty hoặc Undertow (không cần phải deploy ra file WAR)

3.Các starter dependency giúp việc cấu hình Maven đơn giản hơn.

4.Tự động cấu hình Spring khi cần thiết.

5.Không sinh code cấu hình và không yêu cầu phải cấu hình bằng XML …

6.Phát triển các ứng dụng dựa trên Spring một cách tiết kiệm thời gian và dễ dàng.

7.Tự động cấu hình tất cả các components cho một ứng dụng Spring cấp sản xuất.

8.Các máy chủ nhúng được tạo sẵn (Tomcat, Jetty và Undertow), dẫn đến việc triển khai ứng dụng được tăng tốc và hiệu quả hơn.

9.Điểm cuối HTTP, cho phép nhập các tính năng bên trong ứng dụng như chỉ số, tình trạng sức khỏe, v.v. Không có cấu hình XML.

10.Nhiều lựa chọn bổ sung, hỗ trợ nhà phát triển làm việc với cơ sở dữ liệu được nhúng và trong bộ nhớ.

11.Dễ dàng truy cập cơ sở dữ liệu và các dịch vụ hàng đợi như MySQL, Oracle, MongoDB, Redis, ActiveMQ và các dịch vụ khác.

12.Tích hợp trơn tru với hệ sinh thái Spring.

13.Cộng đồng lớn và rất nhiều hướng dẫn, tạo điều kiện cho giai đoạn làm quen.

14.Giảm thiểu thời gian phát triển code, tăng hiệu suất phát triển chung của cả dự án. Dễ dàng tích hợp các mô-đun liên quan như Sping-MVC, Spring Data, Spring Sercurity, Spring Cloud,v.v…

15.Nó cung cấp các HTTPs servers như Tomcat, Jety,.. để phát triển, kiểm thử, deploy một cách dễ dàng.

11.Cung cấp công cụ CLI(Command Line Interface) cho việc phát triển và test ứng dụng nhanh chóng từ command line

12.Ngoài ra còn có nhiều plugins để phát triển nhanh chóng bằng các công cụ như Build như Maven hoặc Gradle.

**3.Nhược điểm của Spring Boot**

1.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.

2.Quá trình chuyển đổi dự án Spring cũ hoặc hiện có thành các ứng dụng Spring Boot nhiều khó khăn và tốn thời gian.

3.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.Lợi ích khi dùng spring boot**

Tại vì những lợi ích mà nó mang lại :

1.Giúp tạo được ứng dụng độc lập dựa trên Spring, có thể tự chạy được java – jar.

2.Có ít cấu hình, có khả năng tự động cấu hình lại Spring khi cần, từ đó giúp các thành viên có thể tiết kiệm thời gian viết code và tăng thêm năng suất.

3.Giữ đầy đủ các tính năng của Spring Framework.

4.Spring boot không yêu cầu cấu hình XML và không sinh code cấu hình.

5.Không cần phải triển khai file WAR mà thực hiện nhúng trực tiếp các ứng dụng server.

6.Giúp cung cấp nhiều plugin.

**5.Spring boot được áp dụng trong microservices .**

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.

# JHipster

### JHipster

JHipster là nền tảng phát triển để nhanh chóng tạo ra, phát triển và triển khai các ứng dụng web hiện đại và dịch vụ kiến ​​trúc vi mô. Chúng tôi hỗ trợ nhiều người dùng giao diện nghệ thuật, bao gồm Angular, React và Vue. Chúng tôi thậm chí còn hỗ trợ di động ứng dụng cho Ionic và React Native! Về phần phụ trợ, chúng tôi hỗ trợ Spring Boot (với Java hoặc Kotlin), Micronaut, Quarkus, Node.js và .NET. Để triển khai, chúng tôi áp dụng các đám mây nguyên tắc với Docker và Kubernetes. Hỗ trợ triển khai tồn tại cho AWS, Azure, Cloud Foundry, Google Cloud Platform, Heroku và OpenShift.

## Lựa chọn JHHipster trong trình cài đặt

JHipster (viết tắt của Java Hipster) là cách đơn giản để chúng ta tạo ra một dự án xung quanh những công nghệ được ưa chuộng nhất với các công nghệ Spring và Angular / React. Khi chúng ta bắt đầu dự án, chúng ta sẽ quan tâm đến 3 cạnh cạnh:

* Phía máy chủ.
* Phía khách hàng: Với những khung frontend mạnh mẽ
* Triển khai: Deply dự án dễ dàng.

Chúng ta sẽ dễ dàng có được một dự án đủ mạnh mẽ và đầy đủ các cơ bản để bắt đầu với thời gian nhanh nhất.

## Tạo dự án đầu tiên với JHipster.

To start create a first project with JHipster:

* Install Java 8 Oracle website
* Install Node.js Node.js website
* Install Yeoman: npm install -g yo
* Install JHipster: npm install -gvator-jhipster.

Tạo dự án:

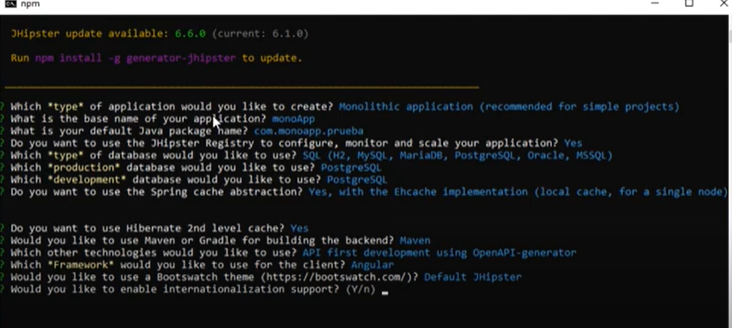
Thực hiện trên Terminal / cmd:

* Tạo 1 trống thư mục là nơi chứa dự án. mkdir myapplication.
* Move to the directory create cd myapplication / Generate application: jhipster.
* Lựa chọn những phù hợp với dự án của bạn.
* Phần phụ trợ: Spring Boot + Spring Security Database:
* Mysql (sản xuất)
* H2 với dựa trên đĩa (phát triển)
* Frontend: Bootstrap + SASS + Angular 7 (hoặc thấp hơn tùy chọn JHipster phiên bản mà bạn cài đặt ở trên) - Sau khi chạy xong bạn sẽ nhận được kết quả như bên dưới. Đó cũng là hướng dẫn để bạn xây dựng dự án của mình trên cục bộ.

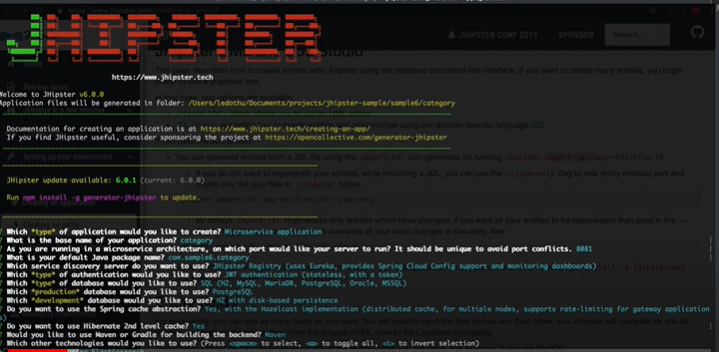
## JHipster JHipster application

* Tạo Microservices Với JHipster.
* Create port API.
* Cannot Create.
* Thêm Logic kinh doanh.
* Thực hiện các cải tiến về người dùng giao diện.
* HTML cho phép.
* Create a small service.
* Tạo sản phẩm.
* Triển khai lên đám mây.

### Khởi tạo nguyên khối bằng JHipster.

[](https://user-images.githubusercontent.com/107390350/174571673-e78cdf03-6796-4b25-849d-947f2179d80e.png)

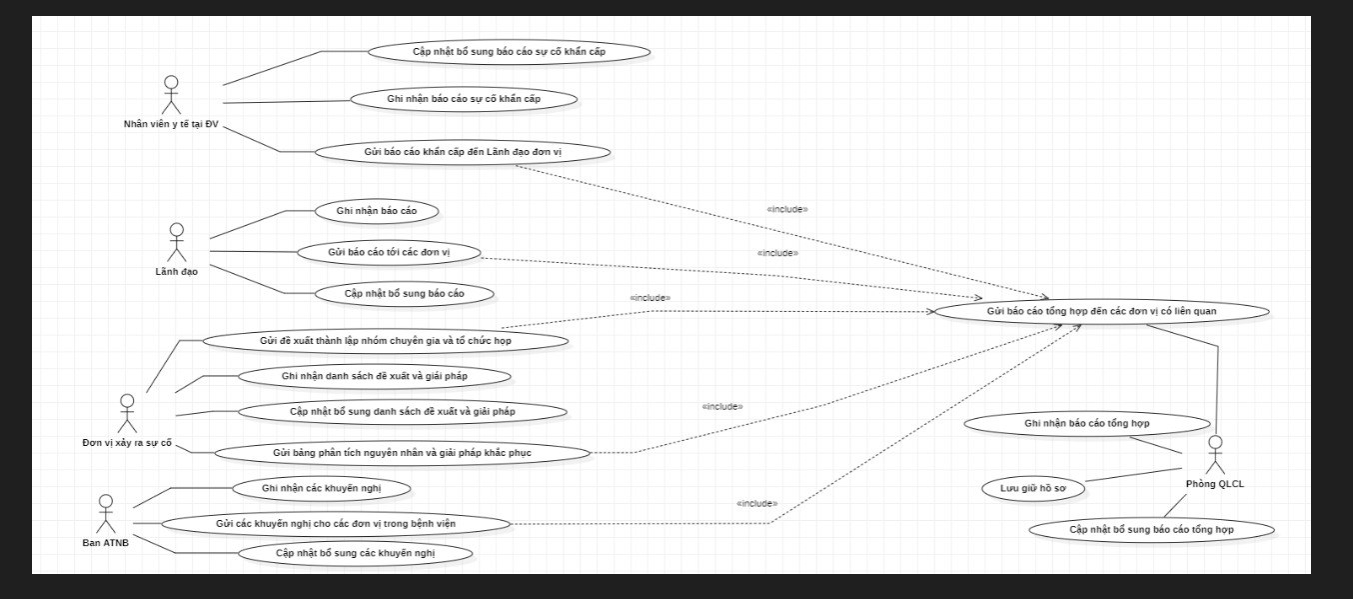
### Khởi tạo micorservices bằng JHipster.

[](https://user-images.githubusercontent.com/107390350/174572417-9a88baeb-77db-4717-81ed-4c8f45e2f370.png)

# Sơ Đồ Dư Án

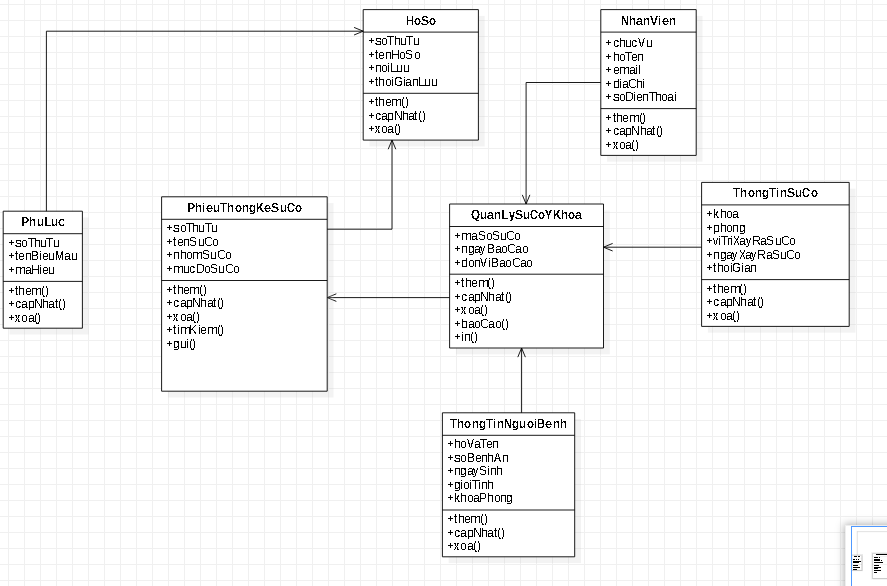
## 1.Sơ đồ Usecase:

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

[](https://user-images.githubusercontent.com/107390350/176072943-08065e38-8c1b-47b2-936a-9417531f4afd.png)

## 2.Sơ đồ Class:

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

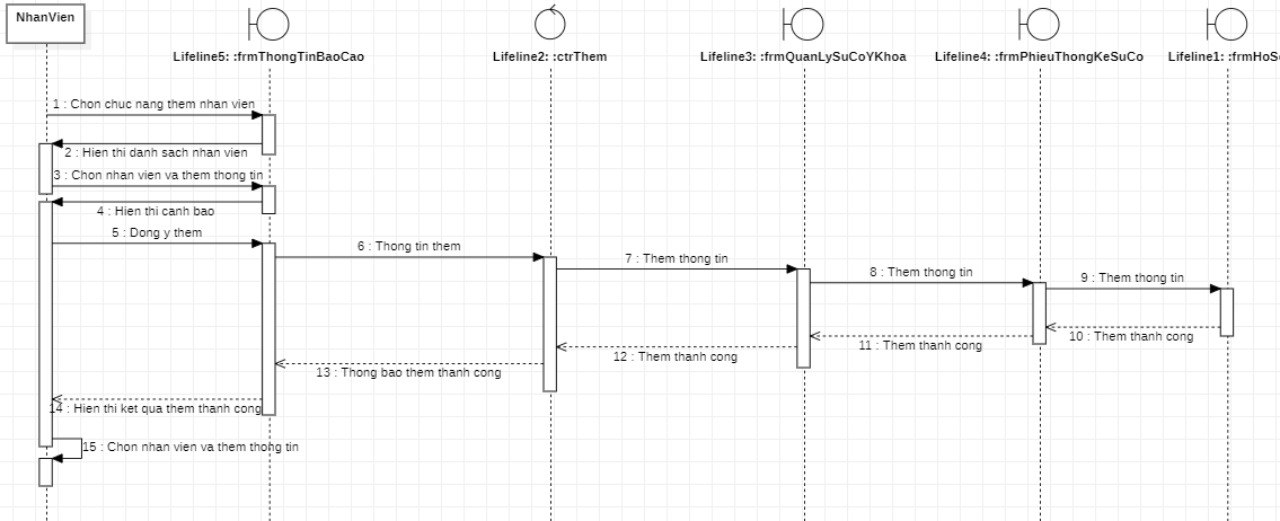
[](https://user-images.githubusercontent.com/107390350/176847681-3651c27e-45bb-4dbe-9424-35a71e12fbed.png)

## 3.Sơ đồ tuần tự:

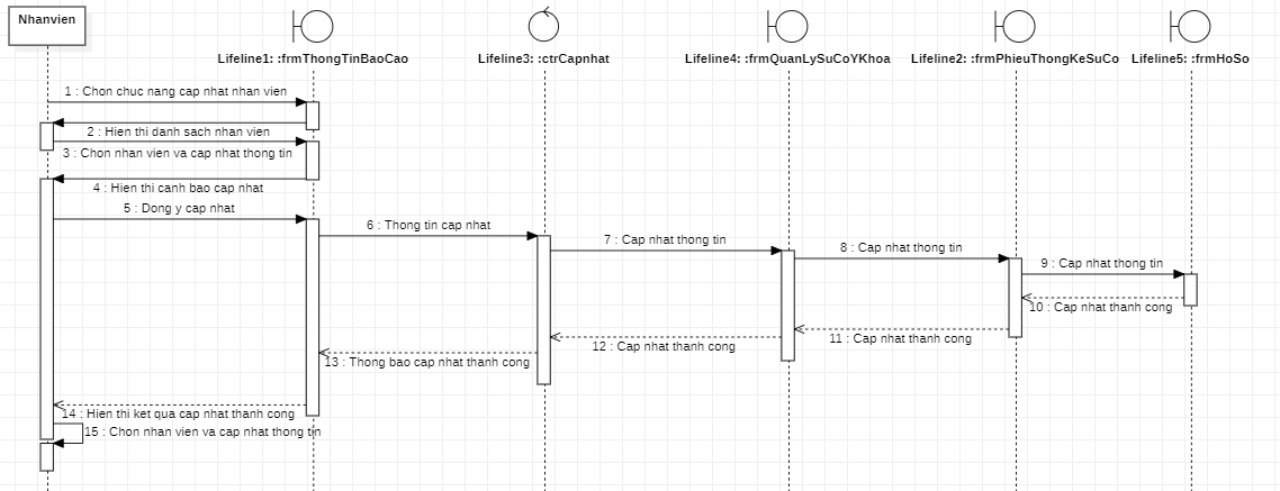
QUY TRÌNH GIẢI QUYẾT VÀ BÁO CÁO SỰ CỐ Y KHOA QT.17.HT

### Nhân viên

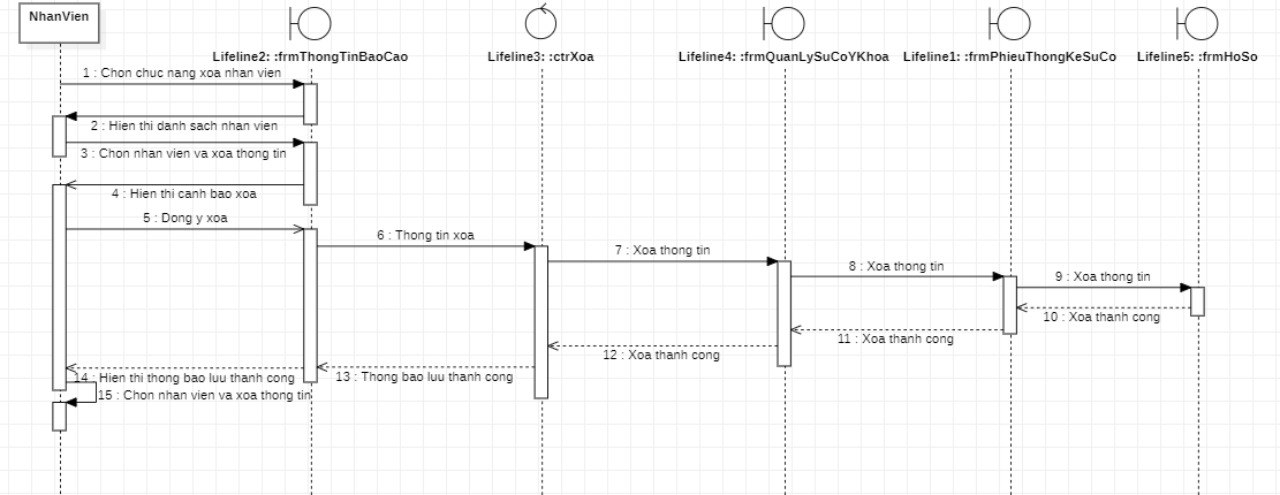
1.Thêm thông tin:

[](https://user-images.githubusercontent.com/107390350/176848218-2e767d01-69bc-4248-b647-de633d9b751d.png)

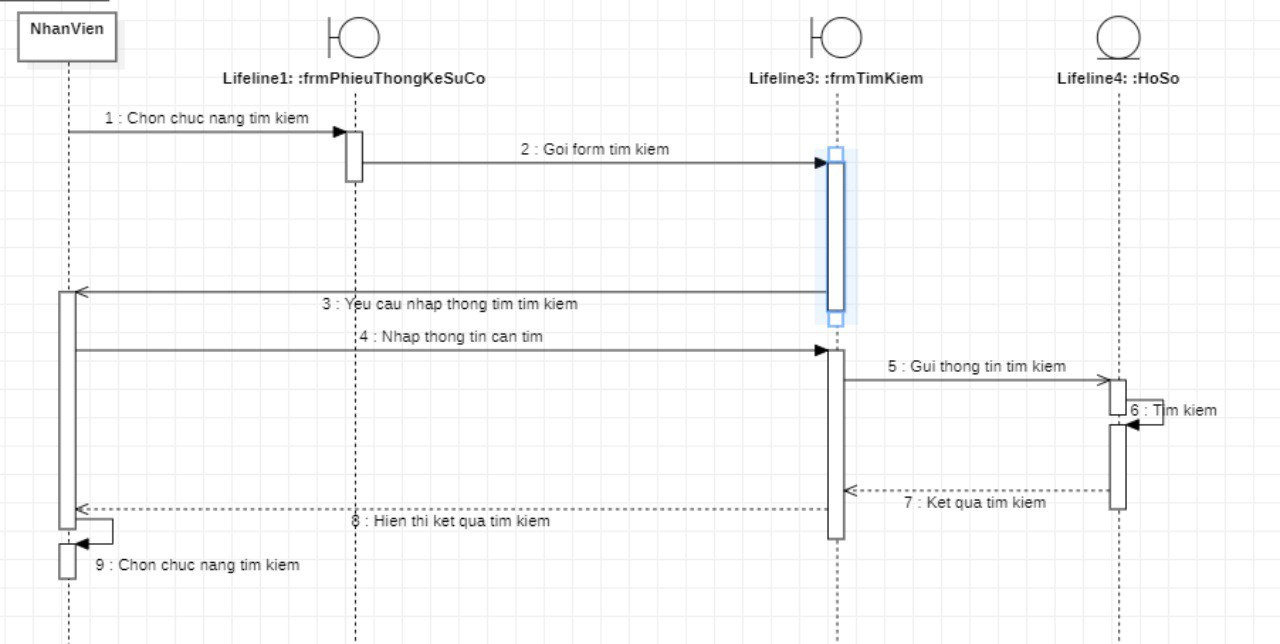
2.Cập nhật thông tin:

[](https://user-images.githubusercontent.com/107390350/176848327-02cd5665-849e-4cdf-a2bf-04aa7c6e2c3f.png)

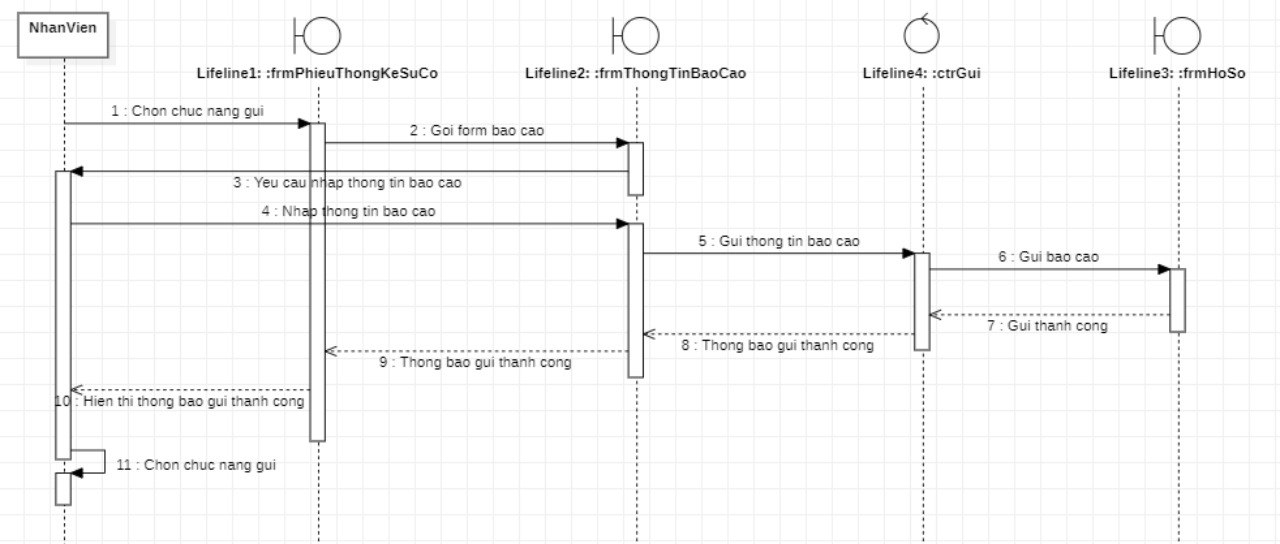
3.Xóa thông tin:

[](https://user-images.githubusercontent.com/107390350/176848414-6dc2f222-834c-42aa-992d-102f35dc7c78.png)

4.Tìm kiếm thông tin:

[](https://user-images.githubusercontent.com/107390350/176848504-c03bac5c-4635-4e8f-a3a4-4ded06974628.png)

5.Gửi thông tin:

[](https://user-images.githubusercontent.com/107390350/176848603-cc4ed8d6-42f3-4cc2-99f2-663f5e5fed0d.png)

# JDL-JHipster.

## Tổng quan:

### Ngôn ngữ miền JHipster (JDL)

JDL là ngôn ngữ miền dành riêng cho JHipster, nơi bạn 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.

Bạn 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:

Nhật thực , Mã VS để tạo tệp JDL và trực quan hóa UML của nó. Bạn cũng có thể tạo và xuất hoặc chia sẻ URL của mô hình JDL của mình.

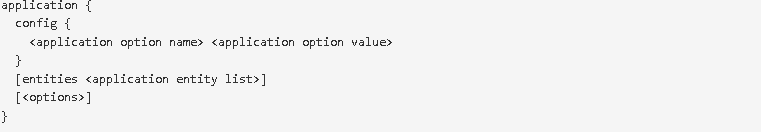
Điều này có thể được sử dụng để thay thế cho việc sử dụng trình tạo phụ thực thể và là cách tiếp cận được khuyến nghị. Ý tưởng là việc quản lý các mối quan hệ bằng cách sử dụng công cụ trực quan sẽ dễ dàng hơn nhiều so với các câu hỏi và câu trả lời cổ điển của người Yeoman.

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.

## Các ứng dụng:

### Cú pháp:

Việc khai báo đơn được thực hiện như sau:

[](https://user-images.githubusercontent.com/107390350/177943319-5792e8b5-fea7-457d-a195-f1ce29bc9057.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 bạn 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 entitieskhóa là tùy chọn: bạn có thể bỏ qua nó, nhưng mọi thực thể trong tệp JDL sẽ được tạo bên trong ứng dụng

-Các ứng dụng có thể có các tùy chọn thông thường (như dtohoặc service), thông tin thêm trong phần tiếp theo .

### 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/107390350/177943872-ee4399f7-eaaa-4587-ae02-e1b02a9d0ffc.png)

Trong mẫu này, chúng ta có thể thấy một số điều:

A, B, C, D, E and FCó 6 thực thể được khai báo trong tệp JDL:. Chúng tôi có 3 ứng dụng:app1, app2 and app3 app1sử dụngA, B and C app2sử dụngC and D app3sử 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. app1sử dụng dtochoA, B and C app2sử dụng paginatecho C(vì có một ngoại lệ) app3sử dụng servicechoE 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-scrollnhư tùy chọn trước đó không bao gồmD 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.

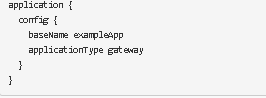
Cũng lưu ý đoạn mã này được lấy từ mẫu trước đó trong app3:

entities \* except A, B, C, D, F service \* with serviceClass Về cơ bản, điều này có nghĩa là app3sẽ chỉ sử dụng Evà các thực thể của ứng dụng sẽ sử dụng servicetùy chọn, có nghĩa là Evà không A to F.

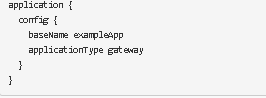
Cuối cùng, Fthực thể không có trong bất kỳ ứng dụng nào và thực thể này sẽ không được tạo vì điều đó.

### Các ví dụ

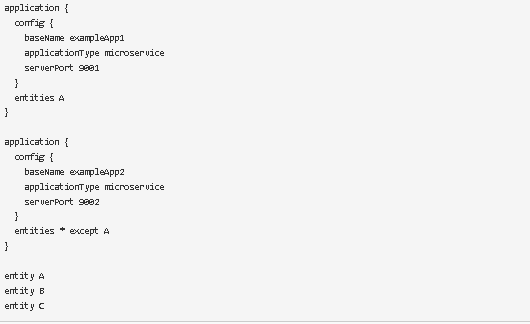
Ví dụ cơ bảnPermalink to "Basic example"

[](https://user-images.githubusercontent.com/107390350/177944177-0f2a42e3-3afd-4939-8c92-25abefe77ed6.png)

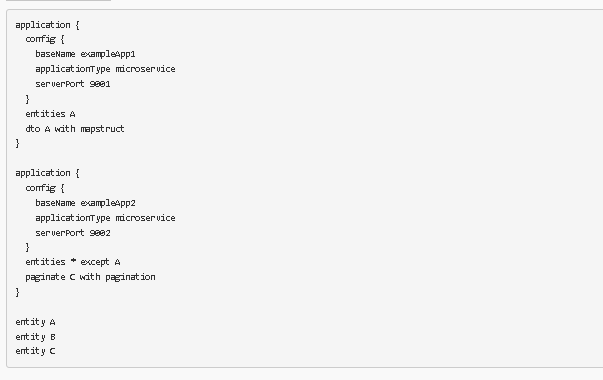
Nhiều ứng dụng

[](https://user-images.githubusercontent.com/107390350/177944177-0f2a42e3-3afd-4939-8c92-25abefe77ed6.png)

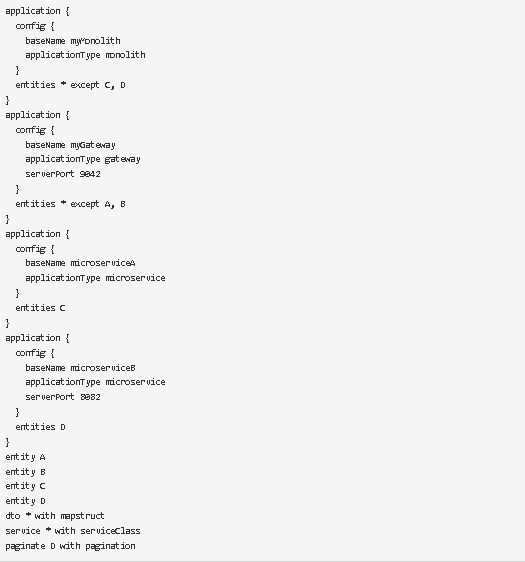
Với các thực thể

[](https://user-images.githubusercontent.com/107390350/177944475-89ca2c8d-640f-4677-a437-c3600e4c5020.png)

Với các tùy chọn

[](https://user-images.githubusercontent.com/107390350/177944714-99b47fe8-810f-4e98-89ce-c5385bc7a5d5.png)

### Toàn bộ bảng phân tích ví dụ

[](https://user-images.githubusercontent.com/107390350/177944873-b1bedb70-64a8-4355-9de7-271bc6d456b6.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 chủ8080 myGateway trong ./myGateway, với cổng máy chủ9042 microserviceA trong ./microserviceA, với cổng máy chủ8081

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

Đối với microservices, mặc định là8081

Đối với cổng và đá nguyên khối, nó8080 microserviceB trong ./microserviceBvới cổng máy chủ8082

Bốn thực thể sẽ được tạo

Avà Btrong nguyên khối

Cvà Dcả trong cửa ngõ

Ctrong microservice đầu tiên

Dtrong microservice thứ hai

Tùy microservicechọn này ngầm hiểu cho CvàD

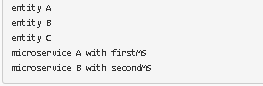
Vì chúng được tạo trên hai microservices nên tùy chọn này sẽ được đặt theo mặc định.

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 bạn.

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

Lấy ví dụ như thiết lập này:

[](https://user-images.githubusercontent.com/107390350/177945309-1236cffd-4757-4b93-96ad-9e3536b440d0.png)

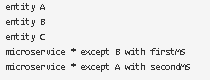
Với hai ứng dụng JHipster ('firstMS' và 'secondMS'), đây là những gì bạn sẽ nhận được nếu nhập tệp JDL trong hai ứng dụng:

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 bạn 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 bạn muốn tạo cùng một thực thể trong hai dịch vụ nhỏ khác nhau, bạn có thể ghi hai tệp JDL thay vì cập nhật tệp JDL. Mỗi lần.

Ví dụ trước không thể được viết như thế này:

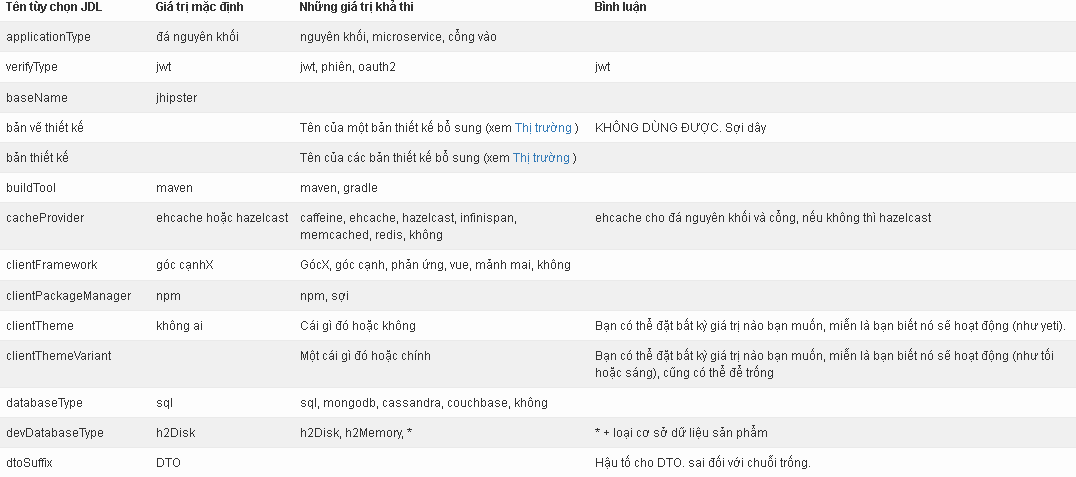
[](https://user-images.githubusercontent.com/107390350/177945459-2cf9e2fe-0f79-4bab-a1bf-746bd2a95b09.png)

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. Bạn cũng có thể tạo toàn bộ ngăn xếp microservice bằng JDL, hãy xem bài đăng trên blog này chẳng hạn

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

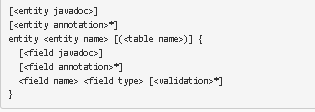
Dưới đây là các tùy chọn ứng dụng được hỗ trợ trong JDL:

[](https://user-images.githubusercontent.com/107390350/177946047-056ecfc4-15bc-4772-b3ea-9d4e15ff7acd.png) [](https://user-images.githubusercontent.com/107390350/177946194-bc7f29ef-8e36-435b-ada0-ec10ba36a5f5.png)

## Thực thể:

### Cú pháp

Việc khai báo thực thể được thực hiện như sau:

[](https://user-images.githubusercontent.com/107390350/177946425-44006ac6-269b-484d-8a09-d5eb3417c475.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 bạn 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 bạn làm điều tương tự. Tạo một loại tùy chỉnh (xem DataType) bằng trình chỉnh sửa, đặt tên cho nó theo các quy ước sau:

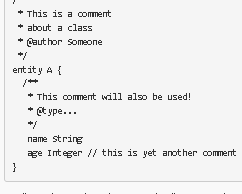
AnyBlobhoặc Blobđể tạo một trường thuộc loại nhị phân "bất kỳ"; ImageBlobđể tạo một trường có nghĩa là một hình ảnh. TextBlobđể tạo trường cho CLOB (văn bản dài). Và bạn có thể tạo bao nhiêu DataTypes tùy thích.

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

[image](https://user-images.githubusercontent.com/107390350/177946664-dd9efec5-c3b4-43d4-8a0c-3792f006562f.png)

Hãy chia nhỏ nó ra:

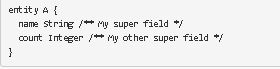
patternlà từ khóa để khai báo xác thực regex (với dấu ngoặc đơn bình thường) /.../mẫu được khai báo bên trong hai dấu gạch chéo \chống chém không cần thiết phải thoát Bình luận Có thể nhận xét trong JDL cho các thực thể và trường, và sẽ tạo ra tài liệu (Javadoc hoặc JSDoc, tùy thuộc vào phần phụ trợ).

[](https://user-images.githubusercontent.com/107390350/177946816-19578aab-0858-4e22-ba55-6bfa76e8ea32.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:

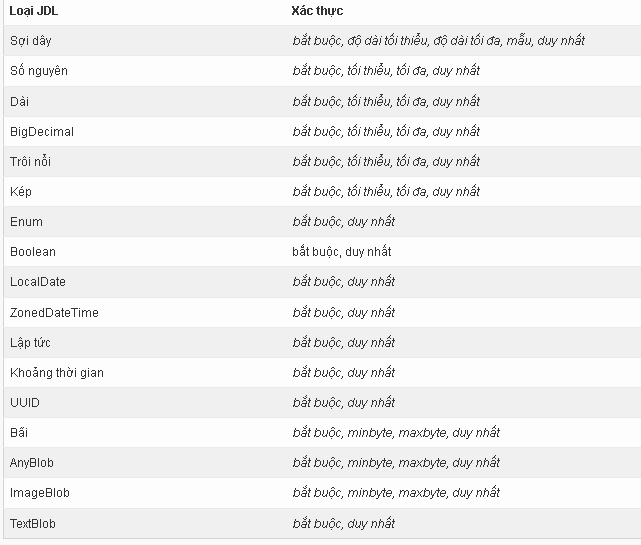
[](https://user-images.githubusercontent.com/107390350/177947133-be3100a2-2841-450c-98a0-fa02ba6096bf.png)

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/107390350/177947260-09bd4fda-4860-4cc3-a7e2-291e424876f7.png)

### Các loại trường và xác thực

[](https://user-images.githubusercontent.com/107390350/177947649-921bf2bd-d722-46bc-897d-74124e002042.png)

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

### Cú pháp

Việc khai báo điều tra được thực hiện như sau:

[image](https://user-images.githubusercontent.com/107390350/177947961-e20e9e4e-4c56-48a5-a249-c29d49538764.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

## Mối quan hệ:

### Các kiểu quan hệ

Được đề cập sau relationshiptừ 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 .

### Các phương pháp quan hệ

Cú pháp này thực sự hữu ích khi:

Bạn có nhiều mối quan hệ cùng loại,

Bạn muốn biết các mối quan hệ là gì,

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

### Cú pháp

[image](https://user-images.githubusercontent.com/107390350/177948556-9a06b5d0-b6d2-47d7-acdd-4d72e6f29776.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,

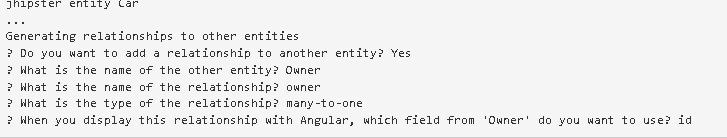
là tên của trường có đầu kia là loại,

là tên của trường sẽ hiển thị trong các hộp chọn (mặc định id:),

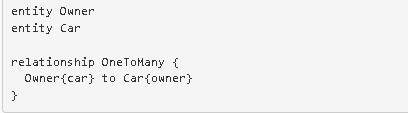
requiredtrường được tiêm có được yêu cầu hay không. with jpaDerivedIdentifiercó @MapsIdđược sử dụng cho liên kết hay không (chỉ áp dụng cho một đối một)

#### OneToMany

Tạo Car:

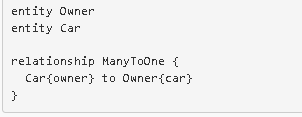
[](https://user-images.githubusercontent.com/107390350/177953533-b61023ca-6a97-4fb8-b09b-2afc712e615d.png)

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

[](https://user-images.githubusercontent.com/107390350/177953629-e2d26ead-ae7f-4ccf-9959-ce30bd69174e.png)

#### ManyToOne

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

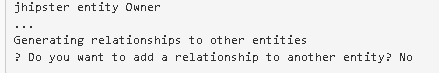
[](https://user-images.githubusercontent.com/107390350/177953974-10cab4a9-0be3-452f-8830-312d05612e7d.png)

Thực hiện mối quan hệ đó vì hai lý do:

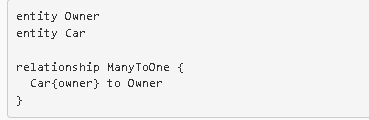
Theo quan điểm kinh doanh, bạn chỉ sử dụng các thực thể của mình theo cách này. Vì vậy, bạn không muốn có một API cho phép các nhà phát triển làm điều gì đó không có ý nghĩa.

Bạn có một mức tăng hiệu suất nhỏ khi sử dụng Ownerthực thể

Tạo Owner:

[](https://user-images.githubusercontent.com/107390350/177954405-368d9bd9-8250-4d28-840e-e4b58333e830.png)

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

[](https://user-images.githubusercontent.com/107390350/177954645-686a8dca-df5b-4a45-bd4d-6aebfd3bebad.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

[image](https://user-images.githubusercontent.com/107390350/177954926-465a5cf7-b21b-4ddd-ad51-2ad56a7e2fea.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

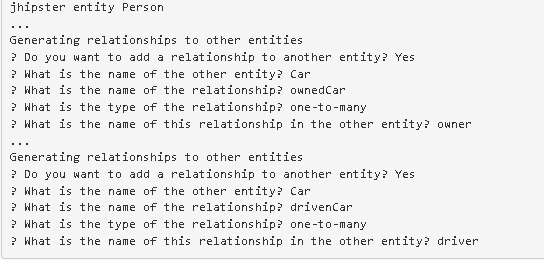
Điều này không được hỗ trợ với JDL vì nó không có trong JHipster.

#### TwoOneToMany

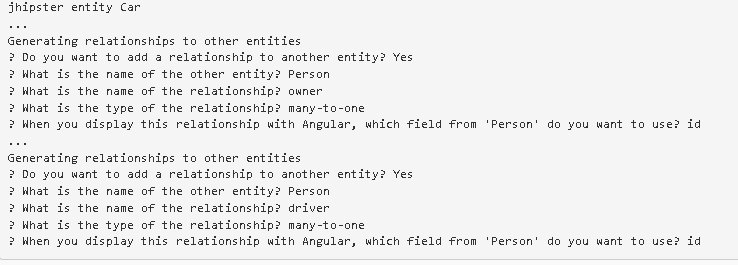
Đối với ví dụ này, a Personcó thể là chủ sở hữu của nhiều ô tô và cũng có thể là người điều khiển nhiều ô tô:

[image](https://user-images.githubusercontent.com/107390350/177955267-11a12a7e-54de-4995-a9eb-ce43609f7d5d.png)

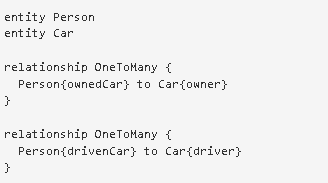
Tạo Personthực thể, có hai mối quan hệ một-nhiều với Carthực thể:

[](https://user-images.githubusercontent.com/107390350/177955400-e039f78e-8bb4-4559-b1a0-7a69adf1e37b.png)

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

[](https://user-images.githubusercontent.com/107390350/177955493-ef006a24-99bf-4ab7-b99d-97de1537aadf.png)

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

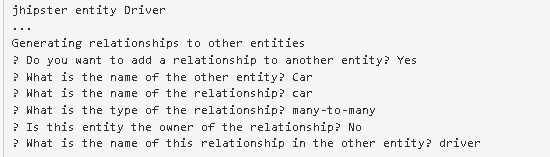
[](https://user-images.githubusercontent.com/107390350/177955577-ca6f7b34-bae0-4f20-80c6-b80b8e93f3af.png)

#### ManyToMany

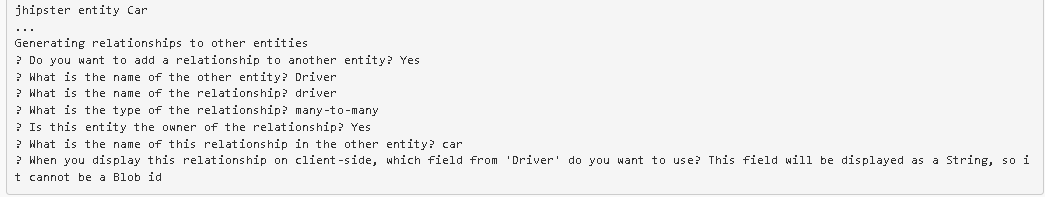
A Drivercó thể lái nhiều ô tô, nhưng một Carcan cũng có nhiều tài xế.

[image](https://user-images.githubusercontent.com/107390350/177955853-afd99ace-6a23-4638-995e-a3ad7f98cb07.png)

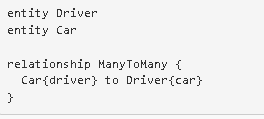
Tạo ra mặt không sở hữu của mối quan hệ, Drivervới mối quan hệ nhiều-nhiều:

[](https://user-images.githubusercontent.com/107390350/177956038-7d663640-75e2-4303-a69f-4ade13a60900.png)

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

[](https://user-images.githubusercontent.com/107390350/177956171-5d1c4618-f068-4caf-acf7-cedc7f1e596d.png)

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

[](https://user-images.githubusercontent.com/107390350/177956297-b00afbb8-057b-438f-8fab-e8207a30c7b6.png)

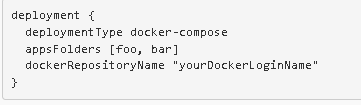
### Triển khai

#### Cú pháp

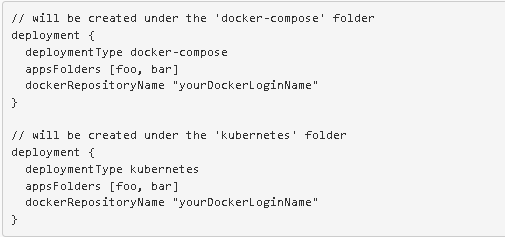
Khai báo triển khai được thực hiện như sau:

[image](https://user-images.githubusercontent.com/107390350/177956892-7358fe09-0a61-4c17-9318-0307382d4bdf.png)

#### Các ví dụPermalink to "Examples"

[](https://user-images.githubusercontent.com/107390350/177957088-1446e921-05d1-4fdc-a946-65ead849285a.png)

Nhiều triển khai

[](https://user-images.githubusercontent.com/107390350/177957219-2514cbd4-c922-4979-8b6f-71f3e594a3ff.png)

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

[](https://user-images.githubusercontent.com/107390350/177957514-e81beaef-3858-4cc9-ba28-d1715c328629.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,

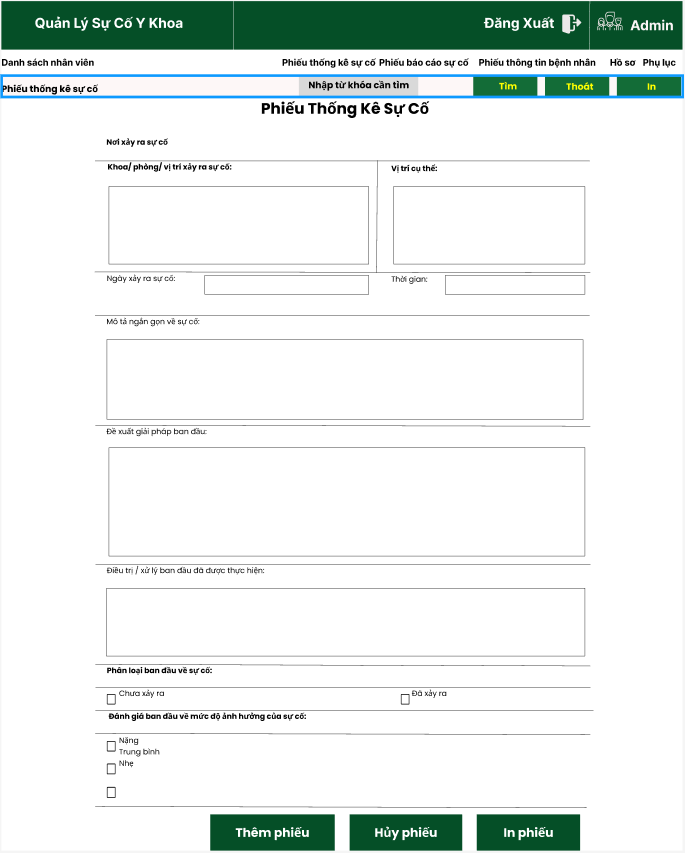
Khai báo các mối quan hệ giữa chúng,

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

# **Chương 6: Thiết Kế Giao Diện**

## **1. Quản Lý Sự Cố Y Khoa**

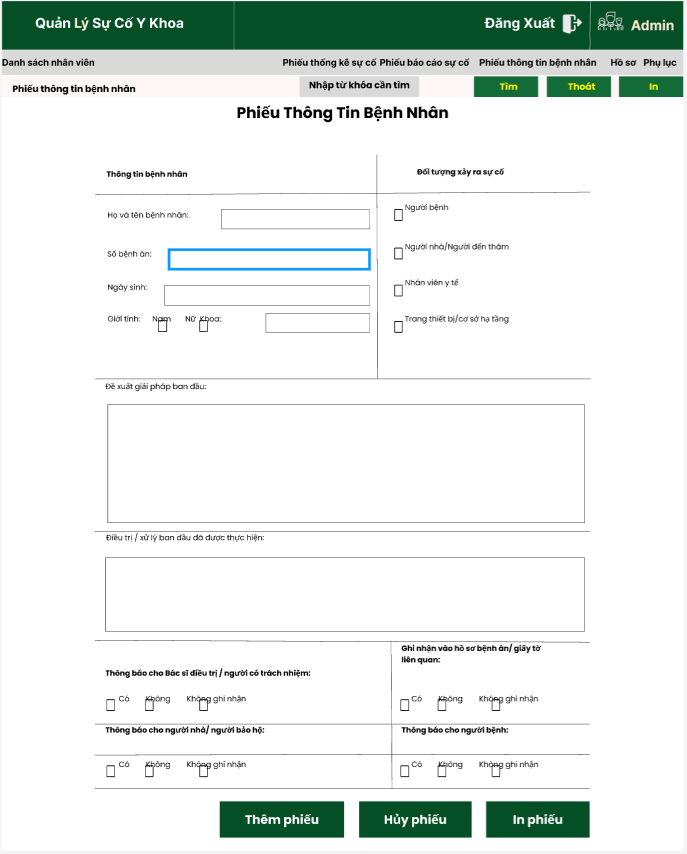
#### **1.1. Phiếu thống kê sự cố**



**1.2. Phiếu báo cáo sự cố**



**1.3. Phiếu thông tin bệnh nhân**



## **2. Quản Lý Nhân Viên**

