**TABLE OF CONTENT**

[1: WEEK 1 3](#_Toc26909239)

[**1.1** **OOP in java** 3](#_Toc26909240)

[**1.2** **Coding Process** 4](#_Toc26909241)

[**1.3** **Unit Testing Process** 6](#_Toc26909242)

[2: WEEK 2 9](#_Toc26909243)

[**2.1** **Đa luồng** 9](#_Toc26909244)

[**2.2** **Thread** 10](#_Toc26909245)

[**2.3** **Thread Pool** 11](#_Toc26909246)

[**2.4** **ExecutorService** 12](#_Toc26909247)

[3: WEEK 3 13](#_Toc26909248)

[**3.1** **Array List** 13](#_Toc26909249)

[**3.2** **Map** 14](#_Toc26909250)

[**3.3** **Set** 15](#_Toc26909251)

[**3.4** **SortMap** 16](#_Toc26909252)

[**3.5** **SortSet** 16](#_Toc26909253)

[**3.6** **TreeSet** 17](#_Toc26909254)

[4: WEEK 4 17](#_Toc26909255)

[**4.1** **Navigation** 17](#_Toc26909256)

[**4.2** **Redirection** 18](#_Toc26909257)

[**4.3** **Permisstion** 18](#_Toc26909258)

[**4.4** **Process** 19](#_Toc26909259)

[5: WEEK 5 20](#_Toc26909260)

[**5.1** **Singleton** 20](#_Toc26909265)

[**5.2** **Factory** 22](#_Toc26909270)

[**5.3** **Astract Factory** 22](#_Toc26909276)

[**5.4** **Decorator** 23](#_Toc26909283)

[**5.5** **Obsever** 24](#_Toc26909291)

[**5.6** **Adapter** 26](#_Toc26909300)

[6: WEEK 6 29](#_Toc26909301)

[**6.1** **Git** 29](#_Toc26909302)

[**6.2** **Maven** 33](#_Toc26909303)

[**6.3** **Spring Framework** 34](#_Toc26909304)

[7: WEEK 7 37](#_Toc26909305)

[**7.1** **Spring Boot** 37](#_Toc26909306)

[**7.2** **Spring data jpa** 38](#_Toc26909307)

[**7.3** **RESTFULL API** 42](#_Toc26909308)

[8: WEEK 8 44](#_Toc26909309)

[**8.1** **HTML** 44](#_Toc26909310)

[**8.2** **CSS** 47](#_Toc26909311)

[**8.3** **JavaScript** 50](#_Toc26909312)

[**8.4** **Bootstrap** 51](#_Toc26909313)

[9: WEEK 9 53](#_Toc26909314)

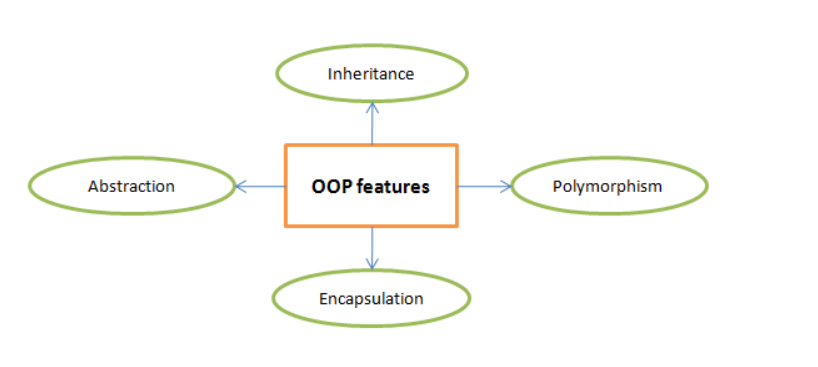
[10: WEEK 10,11 55](#_Toc26909315)

# 1: WEEK 1

* 1. **OOP in java**

**Lập trình hướng đối tượng (OOP-Object-Oriented Programming)** là một phương pháp hay mô hình giúp tăng năng suất, đơn giản hóa việc bảo trì, dễ dàng mở rộng trong thiết kế phần mềm.

**Lập trình hướng đối tượng có 4 tính chất:**

**Kế thừa (Inheritance):**

Khi một đối tượng được truyền lại tất cả các thuộc tính và phương thức của đối tượng cha được gọi là kế thừa. Kế thừa giúp tái sử dụng lại mã nguồn. Nó được sử dụng cho đa hình lúc runtime.

**Đa hình (Polymorphism):**

Khi một nhiệm vụ được thực hiện bởi nhiều cách khác nhau, tính chất này được gọi là đa hình. Ví dụ có nhiều các để thuyết phục các khách hàng khác nhau, để vẽ một cái gì đó như hình tròn, hình chữ nhật, ...

Trong java, để áp dụng tính đa hình chúng ta sử dụng phương thức orverloading hoặc overriding

**Trừu tượng (Abstraction):**

Trừu tượng là sự ẩn đi những chi tiết bên trong và hiển thị ra các chức năng, tính chất này gọi là trừu tượng. Ví dụ: khi gọi điện thoại chúng ta không biết xử lý nội bộ thế nào, khi đi xe máy cũng vậy, mà chúng ta chỉ biết đến các chức năng thông qua giao tiếp bên ngoài.

Trong java, chúng ta áp dụng tính chất trừu tượng bằng cách sử dụng abstract class và interface

**Đóng gói (Encapsulation):**

Việc ràng buộc giữa code và data với nhau tạo thành một khối duy nhất được biết đến là đóng gói. Ví dụ: viên thuốc con nhộng được đóng gói với nhiều loại thuốc bên trong.

Một class trong java là một ví dụ về đóng gói. Java bean là một lớp được đóng gói hoàn toàn vì tất cả các dữ liệu thành viên là private.

**Thế mạnh của OOPs so với ngôn ngữ lập trình hướng thủ tục:**

1) Lập trình hướng đối tượng giúp việc phát triển và bảo trì dễ dàng hơn. Trong khi phương pháp lập trình hướng thủ tục là không dẽ dàng quản lý khi code lớn.

2) Lập trình hướng đối tượng có tính năng ẩn dấu thông tin, trong khi hướng thủ tục có thể truy cập dữ liệu toàn cục ở bất kỳ nơi nào

3) Lập trình hướng đối tượng cung cấp khả năng mô phỏng sự kiện thực tế hiệu quả hơn.

* 1. **Coding Process**

Tiêu chuẩn coding (Coding Standards) là một bộ quy tắc quy định cách viết code của một chương trình mà lập trình viên phải tuân theo khi tham gia vào dự án phát triển chương trình đó. Tùy theo mỗi dự án mà sẽ có những tiêu chuẩn khác nhau, bộ quy tắc đó bao gồm:

* Đặt tên lớp, interface, tên biến, phương thức, …
* Khoảng trắng, tab
* Khai báo và sử dụng biến
* Comment mã nguồn: tên người tạo, phiên bản, ngày tạo file, lớp, phương thức, người thay đổi, nội dung thay đổi, …
* Độ dài tối đa mỗi dòng code, mỗi file, …
* …..

**Tầm quan trọng**

* Dễ bảo trì, sửa lỗi
* Để người khác hiểu được mã nguồn của mình.
* Thống nhất code giữa các thành viên trong nhóm.

**Chuẩn hình thức và chuẩn ngữ nghĩa**

**Chuẩn hình thức:**

Là những quy định liên quan đến sự định dạng của mã nguồn:

* Thụt đầu dòng
* Sử dụng khoảng trắng
* Đóng ngoặc, mở ngoặc
* Đặt tên lớp, thuộc tính, phương thức

**Chuẩn ngữ nghĩa:**

Là những quy định liên quan đến sự thực thi của mã nguồn:

* Biểu thức so sánh
* Cấu trúc điều khiển : if, for, while
* Khai báo và sử dụng biến
* Cài đặt phương thức

**Quy ước đặt tên**

* **Quy tắc viết hoa**

**Pascal case**

* Các chữ cái đầu mỗi từ được viết hoa.
* Các chữ còn lại được viết thường.
* Ví dụ: **M**y**P**rovider, **S**tring**B**uilder

**Camel case**

* Giống với Pascal case nhưng chữ cái đầu của từ đầu tiên viết thường.
* Ví dụ: **m**y**P**rovider, **s**tring**B**uilder
* **Đặt tên class,  interface, abstract class**

Sử dụng danh từ hay cụm danh từ : SinhVien, FormSinhVien,…

Dùng **Pascal case** : SinhVien, FormSinhVien,…

Hạn chế viết tắt gây khó hiểu :

* Sai: Form**SV**
* Đúng:FormSinhVien

Không dùng tiền tố khi đặt tên lớp:

* Sai : **I**SinhVien
* Đúng: SinhVien
* **Phương thức**

Sử dụng **Camel case** để đặt tên phương thức. Ví dụ: xepLoai.

Tên phương thức thể hiện được chức năng của phương thức đó. tinhDiemTrungBinh.Tránh đặt tên gây cảm giác mơ hồ, không rõ nghĩa. Ví dụ: hienThi, tinh.Không phân biệt tên các phương thức bằng số. Ví dụ: tinhDiem1, tinhDiem2.

* **Biến**

Sử dụng **Camel case** để đặt tên biến. Ví dụ: int diemTrungBinh, String hoTen

Không dùng tiền tố. Ví dụ:

* Đúng: String address
* Sai: String strAddress

Tên biến gợi nhớ, tránh viết tắt gây khó hiểu. Ví dụ:

* Đúng: String address
* Sai: String addr

Không đặt tên biến chỉ bằng 1 chữ cái như x, y , z,… trừ trường hợp các biến đếm i, j, k.

Không nên đặt tên biến quá dài, hay quá ngắn vì có thể làm rối chương trình hoặc cũng dẫn đến ý nghĩa biến mơ hồ(quá ngắn).

* **Biến static,enum**

Tất cả các từ được viết hoa và phân cách bằng dấu gạch dưới (\_).

Ví dụ:

* static float PI = 3.14f
* static int MIN**\_**WIDTH = 4
* **Biến final**

Đối với biến final toàn cục: đặt tên biết giống như biến static. Tất cả các từ được viết hoa và phân cách bằng dấu gạch dưới (\_).

Đối với biến fianl cục bộ: đặt tên biến giống như biến thông thường.

* **Đặt tên package**

Tên package: tất cả đều là chữ thường.

Ví dụ:

* Đúng: com.example.deepspace
* Sai: com.example.deepSpace hoặc com.example.deep\_space
* **Viết một phương thức hiệu quả**

Khi một đoạn code xuất hiện ở nhiều nơi trong chương trình ta gom các đoạn code đó thành một phương thức: Tiết kiệm thời gian bảo trì, sửa lỗi.

Khi trong một phương thức có các đoạn code xử lý phức tạp thì ta nên tách đoạn code phức tạp đó ra thành một phương thức riêng biệt: Dễ dàng theo dõi, debug.

Khai báo tham số truyền vào vừa đủ, tránh tình trạng khai báo tham số truyền vào nhưng không sử dụng.

Mỗi phương thức chỉ thực hiện một chức năng.

Kích thước của một phương thức: Nhiều thí nghiệm cho thấy một phương thức có khoảng từ 50 đến 150 dòng code là hợp lý. (Trích Steve McConnell, Chapter 7.4 – Code Complete, Second Edition. 2004).

* 1. **Unit Testing Process**
* **Khái niệm:**

Nhắc lại một chút khái niệm về Unit Testing ở bài trước: **Unit Testing (UT)** là một mức kiểm thử phần mềm với mục đích để xác nhận từng unit của phần mềm được phát triển đúng như được thiết kế. UT là mức test nhỏ nhất trong bất kỳ phần mềm nào. UT bản thân nó là cái gì đó khá trừu tượng vì tùy dự án, chúng ta có thể quy định “unit” ở mức độ nào. Thông thường, “unit” sẽ được quy định giới hạn trong một hàm (method) hay một class. Trong thực tế, tùy vào kinh nghiệm và kĩ năng, developer sẽ đưa ra quyết định viết các UT như thế nào cho phù hợp, có thể test đầu vào đầu ra của hàm, hay kiểm tra một phần hoặc toàn bộ class.

**Unit test** là test do developer viết, được chạy để kiểm tra các hàm do developer viết ra có sai hay không. UT thường được chạy mỗi khi build để đảm bảo các hàm đều chạy đúng sau khi ta sửa code.

UT là các đoạn code có cấu trúc giống như các đối tượng được xây dựng để kiểm tra từng bộ phận trong hệ thống. Mỗi UT sẽ gửi một số yêu cầu đầu vào và kiểm tra kết quả đầu ra có đúng hay không, bao gồm:

* Các kết quả trả về mong muốn.
* Các lỗi ngoại lệ mong muốn.

Các đoạn mã UT hoạt động liên tục hoặc định kỳ để thăm dò và phát hiện các lỗi kỹ thuật trong suốt quá trình phát triển, do đó UT còn được gọi là kỹ thuật kiểm nghiệm tự động.

UT có các đặc điểm sau:

* Đóng vai trò như những người sử dụng đầu tiên của hệ thống.
* Chỉ có giá trị khi chúng có thể phát hiện các vấn đề tiềm ẩn hoặc lỗi kỹ thuật.
* **Vòng đời của UT:**

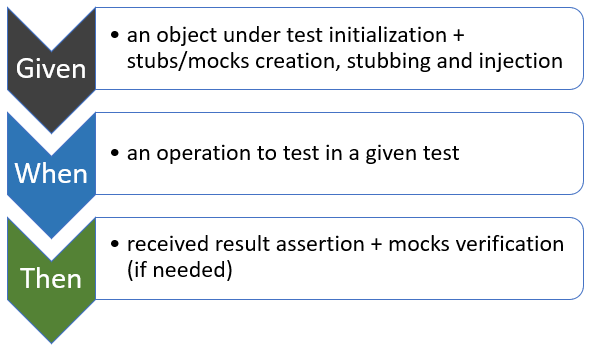
UT có 3 trạng thái cơ bản:

* Fail (trạng thái lỗi).
* Ignore (tạm ngừng thực hiện).
* Pass (trạng thái làm việc).

Toàn bộ UT được vận hành trong một hệ thống tách biệt. Có rất nhiều phần mềm hỗ trợ thực thi UT với giao diện trực quan. Thông thường, trạng thái của UT được biểu hiện bằng các màu khác nhau: màu xanh (pass), màu vàng (ignore) và màu đỏ (fail).

UT chỉ thực sự đem lại hiệu quả khi:

* Được vận hành lặp lại nhiều lần.
* Tự động hoàn toàn.
* Độc lập với các UT khác.
* **Thiết kế UT:**



Mỗi UT đều được tiết kế theo trình tự sau:

* **Given**: Thiết lập các điều kiện cần thiết: khởi tạo các đối tượng, xác định tài nguyên cần thiết, xây dựng các dữ liệu giả, …
* **When**: Triệu gọi các phương thức cần kiểm tra.
* **Then**: Kiểm tra sự hoạt động đúng đắn của các phương thức.
* Dọn dẹp tài nguyên sau khi kết thúc kiểm tra (nếu có).
* **Ứng dụng**

Kiểm tra mọi đơn vị nhỏ nhất là các thuộc tính, sự kiện, thủ tục và hàm.

Kiểm tra các trạng thái và ràng buộc của đối tượng ở các mức sâu hơn mà thông thường chúng ta không thể truy cập được.

Kiểm tra các quy trình (process) và mở rộng hơn là các khung làm việc (workflow – tập hợp của nhiều quy trình).

* **Chiến lược viết mã hiệu quả với UT**
* Phân tích các tình huống có thể xảy ra đối với mã. Đừng bỏ qua các tình huống tồi tệ nhất có thể xảy ra, thí dụ dữ liệu nhập làm một kết nối cơ sở dữ liệu thất bại, ứng dụng bị treo vì một phép toán chia cho không, các thủ tục đưa ra lỗi ngoại lệ sai có thể phá hỏng ứng dụng một cách bí ẩn, …
* Mọi UT phải bắt đầu với trạng thái “fail” và chuyển trạng thái “pass” sau một số thay đổi hợp lý đối với mã chính.
* Mỗi khi viết một đoạn mã quan trọng, hãy viết các UT tương ứng cho đến khi bạn không thể nghĩ thêm tình huống nào nữa.
* Nhập một số lượng đủ lớn các giá trị đầu vào để phát hiện điểm yếu của mã theo nguyên tắc:
  + Nếu nhập giá trị đầu vào hợp lệ thì kết quả trả về cũng phải hợp lệ.
  + Nếu nhập giá trị đầu vào không hợp lệ thì kết quả trả về phải không hợp lệ.
* Sớm nhận biết các đoạn mã không ổn định và có nguy cơ gây lỗi cao, viết UT tương ứng để khống chế.
* Ứng với mỗi đối tượng nghiệp vụ (business object) hoặc đối tượng truy cập dữ liệu (data access object), nên tạo ra một lớp kiểm tra riêng vì những lỗi nghiêm trọng có thể phát sinh từ các đối tượng này.
* Để ngăn chặn các lỗi có thể phát sinh trở lại thực thi tự động tất cả UT mỗi khi có một sự thay đổi quan trọng, hãy làm công việc này mỗi ngày. Các UT lỗi cho chúng ta biết thay đổi nào là nguyên nhân gây lỗi.
* Để tăng hiệu quả và giảm rủi ro khi viết các UT, cần sử dụng nhiều phương thức kiểm tra khác nhau. Hãy viết càng đơn giản càng tốt.
* Cuối cùng, viết UT cũng đòi hỏi sự nỗ lực, kinh nghiệm và sự sáng tạo như viết phần mềm.

# 2: WEEK 2

* 1. **Đa luồng**

**Multitasking**: Là khả năng chạy đồng thời một hoặc nhiều chương trình cùng một lúc trên một hệ điều hành. Hệ điều hành quản lý việc này và sắp xếp lịch phù hợp cho các chương trình đó. Ví dụ, trên hệ điều hành Windows chúng ta có làm việc đồng thời với các chương trình khác nhau như: Microsoft Word, Excel, Media Player, …

Chúng ta sử dụng đa nhiệm để tận dụng tính năng của CPU.

Đa nhiệm có thể đạt được bằng hai cách:

1. Đa nhiệm dựa trên tiến trình (Process) – Đa tiến trình (Multiprocessing).
   * Mỗi tiến trình có địa chỉ riêng trong bộ nhớ, tức là mỗi tiến trình phân bổ vùng nhớ riêng biệt.
   * Tiến trình là nặng.
   * Sự giao tiếp giữa các tiến trình có chi phí cao.
   * Chuyển đổi từ tiến trình này sang tiến trình khác đòi hỏi thời gian để đăng ký việc lưu và tải các bản đồ bộ nhớ, các danh sách cập nhật, …

* Đa nhiệm dựa trên luồng (Thread) – Đa luồng (MultiThreading).
  + Các luồng chia sẻ không gian địa chỉ ô nhớ giống nhau.
  + Luồng là nhẹ.
  + Sự giao tiếp giữa các luồng có chi phí thấp.

Đa tiến trình (multiprocessing) và đa luồng (multithreading) cả hai được sử dụng để tạo ra hệ thống đa nhiệm (multitasking). Nhưng chúng ta sử dụng đa luồng nhiều hơn đa tiến trình bởi vì các luồng chia sẻ một vùng bộ nhớ chung. Chúng không phân bổ vùng bộ nhớ riêng biệt để tiết kiệm bộ nhớ, và chuyển đổi ngữ cảnh giữa các luồng mất ít thời gian hơn tiến trình.

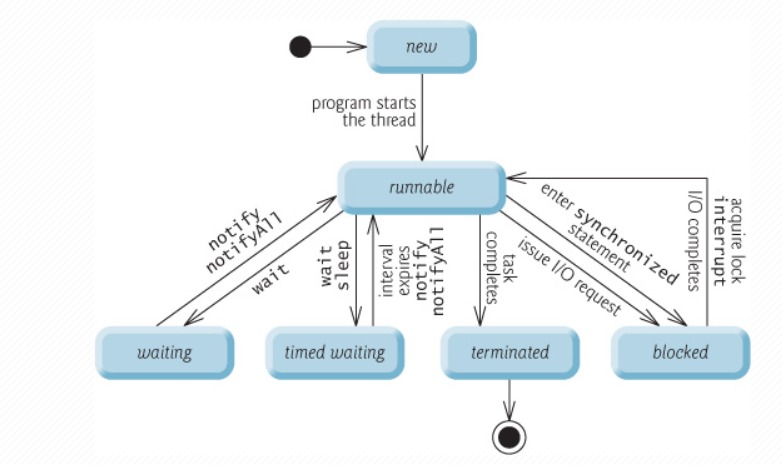
* **Ưu điểm**

Nó không chặn người sử dụng vì các luồng là độc lập và bạn có thể thực hiện nhiều công việc cùng một lúc.Mỗi luồng có thể dùng chung và chia sẻ nguồn tài nguyên trong quá trình chạy, nhưng có thể thực hiện một cách độc lập.Luồng là độc lập vì vậy nó không ảnh hưởng đến luồng khác nếu ngoại lệ xảy ra trong một luồng duy nhất.Có thể thực hiện nhiều hoạt động với nhau để tiết kiệm thời gian. Ví dụ một ứng dụng có thể được tách thành : luồng chính chạy giao diện người dùng và các luồng phụ nhiệm gửi kết quả xử lý đến luồng chính.

* **Nhược điểm**

Càng nhiều luồng thì xử lý càng phức tạp.Xử lý vấn đề về tranh chấp bộ nhớ, đồng bộ dữ liệu khá phức tạp.Cần phát hiện tránh các luồng chết (dead lock), luồng chạy mà không làm gì trong ứng dụng cả.Deadlock (Khoá chết) là gì? Deadlock xảy ra khi 2 tiến trình đợi nhau hoàn thành, trước khi chạy. Kết quả của quá trình là cả 2 tiến trình không bao giờ kết thúc.

* 1. **Thread**
* **Vòng đời của thread**



Vòng đời của thread trong java được kiểm soát bởi JVM. Java định nghĩa các trạng thái của luồng trong các thuộc tính static của lớp **Thread.State**:

* **NEW**: Đây là trạng thái khi luồng vừa được khởi tạo bằng phương thức khởi tạo của lớp Thread nhưng chưa được start(). Ở trạng thái này, luồng được tạo ra nhưng chưa được cấp phát tài nguyên và cũng chưa chạy. Nếu luồng đang ở trạng thái này mà ta gọi các phương thức ép buộc stop,resume,suspend … sẽ là nguyên nhân sảy ra ngoại lệ IllegalThreadStateException .
* **RUNNABLE**: Sau khi gọi phương thức start() thì luồng test đã được cấp phát tài nguyên và các lịch điều phối CPU cho luồng test cũng bắt đầu có hiệu lực. Ở đây, chúng ta dùng trạng thái là Runnable chứ không phải Running, vì luồng không thực sự luôn chạy mà tùy vào hệ thống mà có sự điều phối CPU khác nhau.
* **WAITING** : Thread chờ không giới hạn cho đến khi một luồng khác đánh thức nó.
* **TIMED\_WAITING** : Thread chờ trong một thời gian nhất định, hoặc là có một luồng khác đánh thức nó.
* **BLOCKED**: Đây là 1 dạng của trạng thái “Not Runnable”, là trạng thái khi Thread vẫn còn sống, nhưng hiện tại không được chọn để chạy. Thread chờ một monitor để unlock một đối tượng mà nó cần.
* **TERMINATED**: Một thread ở trong trạng thái terminated hoặc dead khi phương thức run() của nó bị thoát.
* **Cách tạo thread**

Trong java ta có thể tạo ra một luồng bằng một trong hai cách sau: tạo 1 đối tượng của lớp được **extend** từ **class Thread** hoặc **implements** từ **interface Runnable**.

Để tạo luồng bằng cách tạo lớp kế thừa từ lớp Thread, ta phải làm các công việc sau :

1. Khai báo 1 lớp mới kế thừa từ lớp Thread
2. Override lại phương thức run ở lớp này, những gì trong phương thức run sẽ được thực thi khi luồng bắt đầu chạy. Sau khi luồng chạy xong tất cả các câu lệnh trong phương thức run thì luồng cũng tự hủy.
3. Tạo 1 thể hiện (hay 1 đối tượng) của lớp ta vừa khai báo.
4. Sau đó gọi phương thức start() của đối tượng này để bắt đầu thực thi luồng.

Để tạo luồng bằng cách hiện thực từ Interface Runnable, ta phải làm các công việc sau:

1. Khai báo 1 lớp mới implements từ Interface Runnable
2. Hiện thực phương thức run() ở lớp này, những gì trong phương thức run() sẽ được thực thi khi luồng bắt đầu chạy. Sau khi luồng chạy xong tất cả các câu lệnh trong phương thức run thì luồng cũng tự hủy.
3. Tạo 1 thể hiện (hay 1 đối tượng) của lớp ta vừa khai báo. (VD : Tên đối tượng là r1)
4. Tạo 1 thể hiện của lớp Thread bằng phương thức khởi tạo : Thread(Runnable target)
   * Runnable target: Là 1 đối tượng thuốc lớp được implements từ giao diện Runnable.
   * Ví dụ: Thread t1 = new Thread(r1);

* Gọi phương thức start() của đối tượng t1.
  1. **Thread Pool**

**ThreaPool** trong java rất hữu ích khi cần giới hạn số lượng Thread được chạy bên trong ứng dụng của bạn cùng 1 thời điểm. Sẽ có vấn đề về hiệu suất khi tạo mới 1 Thread (new Thread) và mỗi thread được tạo ra cũng được phân bổ bộ nhớ cho việc sử dụng.

Thay vì phải tạo mới thread cho mỗi task (nhiệm vụ) được thực hiện đồng thời, các nhiệm vụ cần đưa vào 1 thread pool (hồ bơi dành cho luồng). Và ngay sau khi trong hồ bơi có bất kì luồng nào đang nhàn rỗi (no task) các nhiệm vụ sẽ được gán vào 1 trong số chúng và được thực thi.

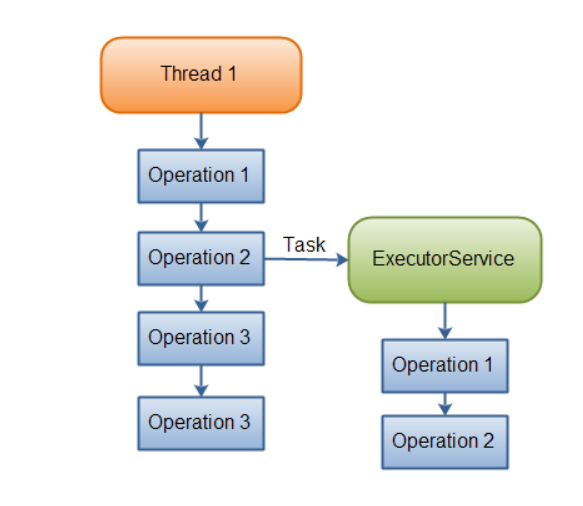
Bên trong ThreadPool các nhiệm vụ được chèn vào một **Blocking Queue** (hàng đợi được khóa) nơi mà các Thread sẽ lấy chúng ra và thực thi lần lượt. Mỗi khi có 1 task mới được thêm vào Queue, sau đó chỉ có 1 thread nhàn rỗi sẽ vào queue và lấy chúng ra, các thread nhàn rỗi còn lại phải chờ sau khi thread trước đó lấy nhiệm vụ ra thành công.

Thread Pool thường xuyên được sử dụng trong các máy chủ thực hiện đa luồng. Mỗi kết nối đến với máy chủ thông qua mạng được bao bọc như là 1 nhiệm vụ (task) và đưa vào thread pool. Các thread trong thread pool sẽ xử lý các yêu cầu về kết nối đồng thời.

* 1. **ExecutorService**

Giao diện (interface) java.util.concurrent.ExecutorService đại diện cho cơ chế thực thi bất đồng bộ có khả năng thực thi các nhiệm vụ trong background (nền). **ExecutorService** là tương tự như một Thread Pool. Trong thực tế, việc thực hiện ExecutorService trong gói là một triển khai thread pool.

* Task Delegation (Sự ủy thác nhiệm vụ)

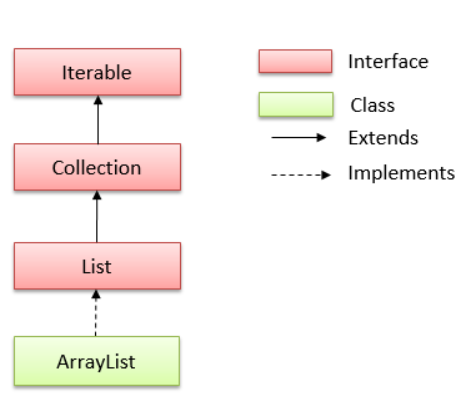


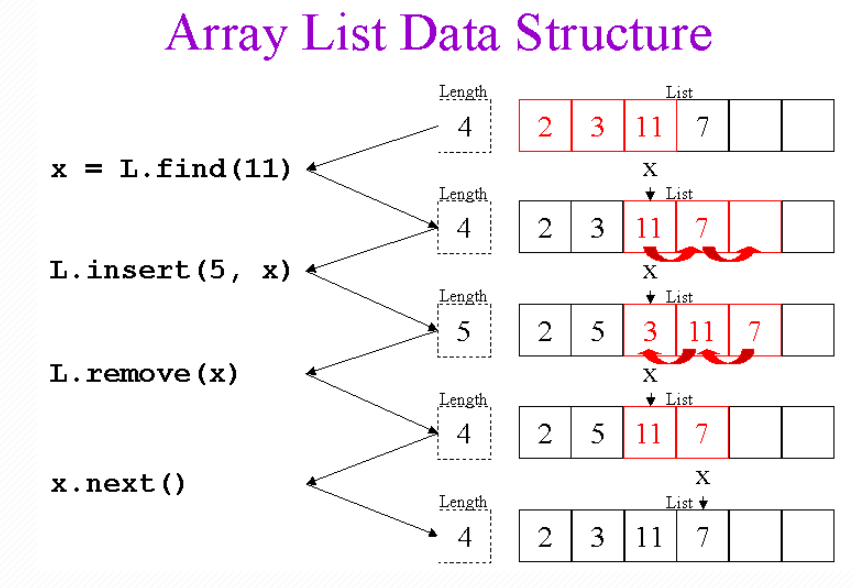
Một khi Thread giao nhiệm vụ cho ExecutorService, nó sẽ tiếp tục thực hiện công việc riêng của nó độc lập với việc thực thi nhiệm vụ của ExecutorService.

# 3: WEEK 3

* 1. **Array List**

**Lớp ArrayList trong java** được sử dụng như một mảng động để lưu trữ các phần tử. Nó kế thừa lớp AbstractList và impliments giao tiếp List.





Các phương thức khởi tạo (constructor) của lớp ArrayList

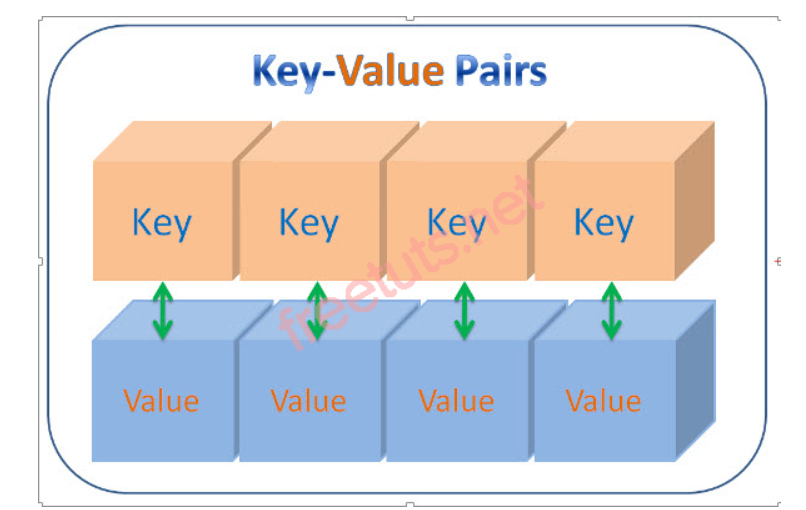
**ArrayList()**: khởi tạo một danh sách mảng trống.

**ArrayList(Collection c)**: khởi tạo một danh sách mảng được khởi tạo với các phần tử của collection c.

**ArrayList(int capacity)**: khởi tạo một danh sách mảng mà có sức chứa (compacity) ban đầu được chỉ định. Nếu không chỉ định, mặc định là **10**. Mỗi lần thêm một phần tử vào danh sách, nếu vượt quá sức chứa cho phép thì danh sách sẽ tự động tăng thêm **50%** kích thước hiện có.

* 1. **Map**

**Map là một tập các cặp khóa - giá trị (key - value)**. Giá trị của các phần tử trong Map có thể giống nhau, nhưng **khóa thì không được giống nhau** (vì thế chúng ta có thể tạo ra 1 Set có các phần tử là khóa của Map). Dựa vào khóa, chúng ta có thể xác định được các giá trị value tương ứng với khóa đó. Dưới đây là hình ảnh minh họa mối quan hệ giữa key và value trong Map:



Map được sử dụng trong trường hợp chúng ta muốn truy xuất, cập nhật hoặc tìm kiếm phần tử thông qua khóa của phần tử đó. Ví dụ:

* Một Map bao gồm thông tin của người quản lý và nhân viên trong một công ty. Mỗi một người quản lý (*key*) sẽ liên kết với danh sách các nhân viên mà người đó quản lý (*value*).
* Một Map bao gồm thông tin của một lớp học và các sinh viên có trong lớp đó. Mỗi một lớp (*key*) sẽ liên kết với danh sách các sinh viên của lớp đó (*value*).

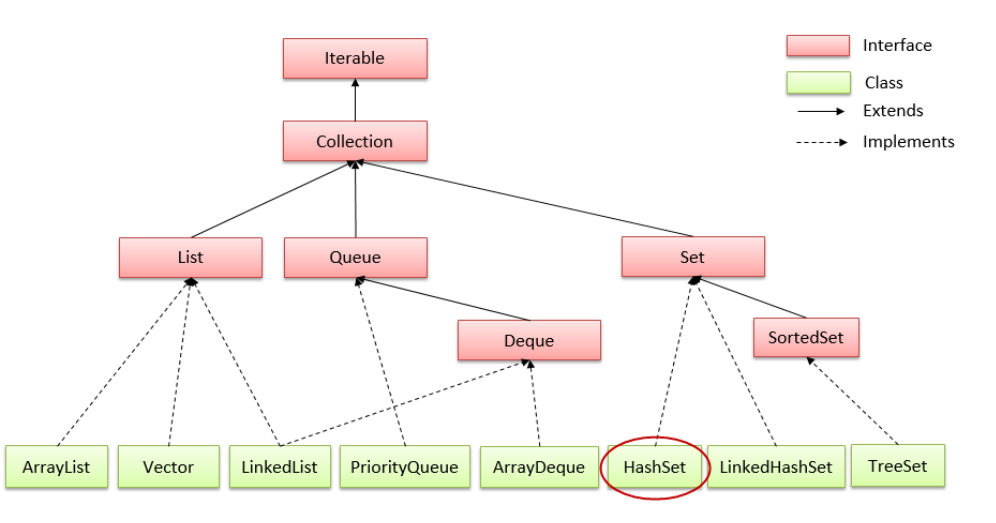
Trong bài Tổng quan, tôi có trình bày những thành phần của Collections Framework, trong đó tôi có đề cập đến **Implementations**là sự triển khai các Interface (ví dụ như các Class), **vì vậy để khai báo một Map chúng ta cần phải dùng đến các Class để triển khai nó**, trong phần này chúng ta sẽ sử dụng 3 loại phổ biến nhất là HashMap, LinkedHashMap và TreeMap. Đối với Map Interface có Class triển khai là HashMap thì thứ tự các phần tử không dựa theo thứ tự lúc thêm vào, đối với Map Interface có Class triển khai là LinkedHashMap thì thứ tự các phần tử dựa theo thứ tự lúc thêm vào, còn đối với Map Interface có Class triển khai là TreeMap thì thứ tự các phần tử được sắp xếp theo chiều tăng dần của khóa. Ví dụ dưới đây sẽ cho các bạn thấy sự khác nhau khi sử dụng HashMap, LinkedHashMap và TreeMap để khai báo Map trong Java:

* 1. **Set**

Set Interface là một loại Interface Collection. Khác với List, các phần tử trong List có thể giống nhau, còn đối với Set, các phần tử trong Set là duy nhất (nghĩa là giá trị của các phần tử này không được giống nhau).

Vậy Set được sử dụng trong trường hợp nào? Chúng ta sẽ sử dụng Set khi chúng ta muốn lưu trữ một danh sách các phần tử không có sự trùng lặp hoặc khi chúng ta không quan tâm đến thứ tự của các phần tử trong danh sách đó.

Đối với Set Interface có Class triển khai là HashSet thì các phần tử không được sắp xếp theo bất kỳ thứ tự nào, còn đối với Set Interface có Class triển khai là TreeSet thì thứ tự các phần tử trong Set được sắp xếp tăng dần. Ví dụ dưới đây sẽ cho các bạn thấy sự khác nhau khi sử dụng HashSet và TreeSet để khai báo Set trong Java:



* 1. **SortMap**

SortedMap Interface là 1 dạng riêng của Map Interface nên nó có những đặc điểm của Map đó là SortedMap cũng bao gồm một tập các cặp khóa - giá trị (key - value). Giá trị của các phần tử trong SortedMap có thể giống nhau, nhưng khóa thì không được giống nhau,vàdựa vào khóa chúng ta có thể xác định được các giá trị value tương ứng với khóa đó. Ngoài ra,SortedMap có điểm vượt trội hơn so với Map là các entry có trong SortedMap được sắp xếp tăng dần theo khóa**.**

Vì SortedMap là 1 dạng riêng của Map nên những phương thức của nó sẽ tương tự như những phương thức có trong Map. Trong bài này, tôi sẽ không trình bày lại những phương thức đó mà tôi sẽ giới thiệu những phương thức riêng của SortedMap để cho các bạn tìm hiểu.

* 1. **SortSet**

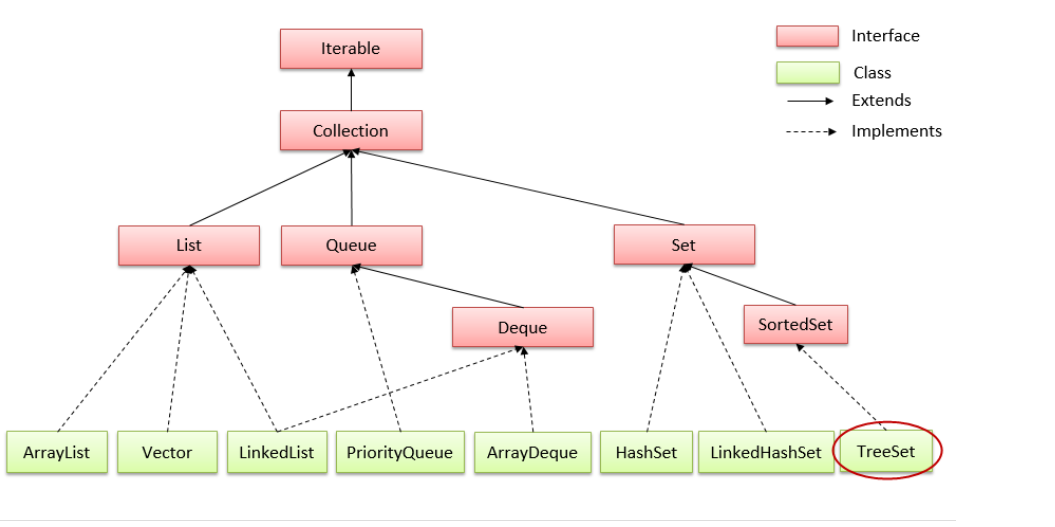
SortedSet Interface là 1 dạng riêng của Set Interface nên nó có những đặc điểm của Set đó là các phần tử trong SortedSet là duy nhất (nghĩa là giá trị của các phần tử này không được giống nhau) và SortedSet được sử dụng khi chúng ta muốn lưu trữ một danh sách các phần tử không có sự trùng lặp. Ngoài ra,**SortedSet có điểm vượt trội hơn so với Set là thứ tự các phần tử trong Set được sắp xếp tăng dần hoặc giảm dần** (***mặc định là tăng dần***).

* 1. **TreeSet**

Lớp **TreeSet trong java** implements giao diện Set sử dụng cấu trúc cây để lưu trữ các phần tử. Nó kế thừa lớp AbstractSet và implements giao diện NavigableSet. Các đối tượng của lớp TreeSet được lưu trữ theo thứ tự tăng dần.

Các điểm quan trọng về lớp TreeSet trong java là:

* Chỉ chứa các phần tử duy nhất giống như HashSet.
* Thời gian truy xuất nhanh.
* Duy trì thứ tự tăng dần.



# 4: WEEK 4

* + - 1. **Navigation**

# hiển thị directory hiện tại

pwd

# Output

/home/nguyen.viet.hung

# liệt kê content của một directory

ls

Desktop Documents Downloads Music Pictures

# chuyển vị trí directory hiện tại có thể sử dụng đường dẫn tuyệt đối, tương đối

cd [absolute\_path|relative\_path]

# chuyển đến vị trí directory ngay trước vị trí hiện tại `n` lần (chỉ để - sẽ là vị trí ngay trước đó)

cd -[n]

# Example

➜ ~ cd /etc/nginx

➜ nginx pwd

/etc/nginx

➜ nginx cd -

~

➜ ~ pwd

/home/nguyen.viet.hung

# Chuyển đến home directory của một user bất kỳ

cd ~[user\_name]

* + - 1. **Redirection**

Các chương trình trong Linux sẽ tự động được kết nối với 3 luồng dữ liệu khi chúng được thực thi:

* **stdin** (standard input): đây là luồng sẽ đưa dữ liệu vào chương trình để xử lý.
* **stdout** (standard output): luồng này dùng để xuất dữ liệu ra màn hình hiển thị sau khi quá trình thực thi hoàn tất mà không gặp lỗi.
* **stderr** (standard error): luồng này có chức năng tương tự **stdout**, tuy nhiên nó chỉ dùng để in các thông báo lỗi và đồng thời khi đó tín hiệu lỗi cũng được gửi tới hệ điều hành. 

Ngoài ra, tùy theo chương trình mà luồng **stdout** sẽ được thay thế bằng tệp tin hoặc máy in .... Việc liên kết các chương trình sẽ là việc đưa dữ liệu đầu ra của chương trình trước đến thẳng đầu vào của chương trình sau mà không để dữ liệu được in ra màn hình hiển thị hoặc file.

* + - 1. **Permisstion**

Trong Linux có 3 nhóm phân quyền chính, không bao nhau:

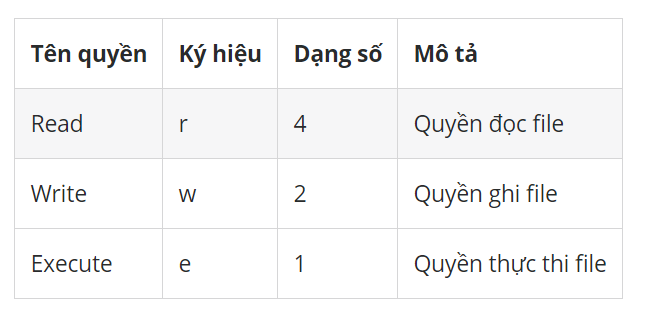
* Owner: chỉ cấp quyền cho chủ sở hữu của file.
* Group: chỉ cấp quyền cho nhóm sở hữu của file.
* Other: cấp quyền cho những người dùng và nhóm không thuộc 2 nhóm trên.

Tuy nhiên vẫn có 1 ngoại lệ, đó là người dùng root (siêu người dùng). Người dùng này có mọi quyền hạn trên mọi file trong hệ thống, không bị ràng buộc bởi bất cứ sự phân quyền nào. Như mình đã nói trong phần mở đầu, hệ thống phân quyền sẽ rất nguy hiểm nếu bạn không hiểu về nó. VD câu lệnh sau sẽ xóa tất cả file trên hệ thống nếu người thực thi là root

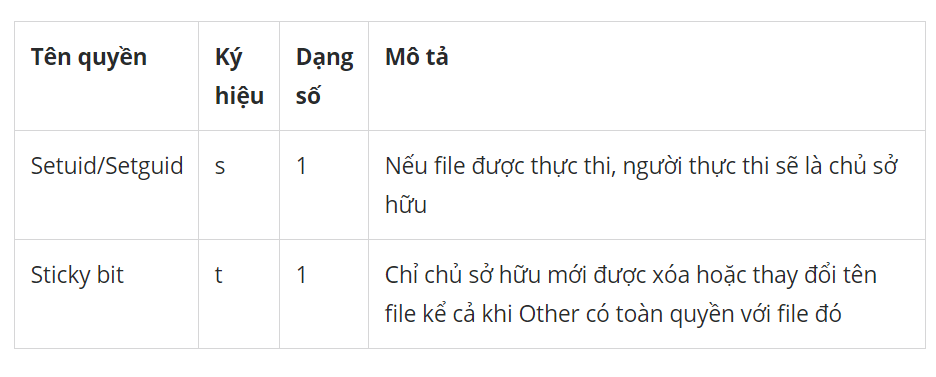
Do vậy với bất cứ một tác vụ gì phải dùng tới quyền hạn của người dùng root ta nên chắc chắn sẽ không gây hại cho hệ thống thì mới nên sử dụng, hoặc tìm một phương án thay thế mà không cần tới quyền hạn nguy hiểm này.

* **Loại phân quyền**

Với một file, có 3 loại phân quyền cơ bản như trong bảng sau:

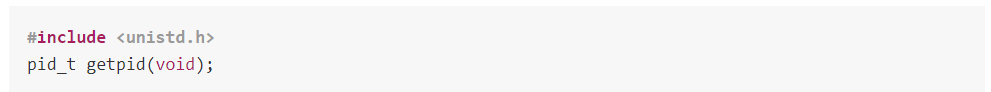


Ngoài ra có một vài phân quyền đặc biệt như sau:



* + - 1. **Process**

Mỗi tiến trình có một ID (PID), đó là một số nguyên dương, dùng để định danh duy nhất tiến trình đó trong hệ thống. Process ID được sử dụng trong rất nhiều các system call. Ví dụ system call kill() cho phép một process gửi tín hiệu tới một process khác thông qua PID của process nhận. PID cũng rất hữu ích khi ta cần định danh cho process, ví dụ như dùng PID để tạo tên cho file... System call getpid() sẽ trả về PID của process gọi nó:

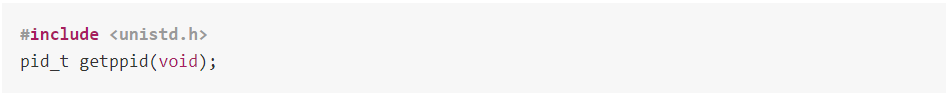


Hàm này luôn trả về thành công.  
Ngoại trừ process init luôn có PID là 1, PID của một process được tạo ra hầu như không liên quan gì đến program (chương trình) tạo ra process đó.

Linux kernel cho phép PID nhỏ hơn hoặc bằng 32,767 (hệ 32bit). Khi một process mới được tạo thì nó được cấp PID một cách tuần tự nếu số PID còn available. Khi có một PID chạm tới ngưỡng 32,767 thì kernel sẽ reset giá trị cấp tiếp theo bắt đầu từ 300.

Giá trị 32,767 này ở từ Kernel 2.6 có thể thay đổi trong file **/proc/sys/kernel/pid\_max**.

Mỗi process luôn có cha của nó. Ta có thể lấy được PID của process cha bằng systemcall:

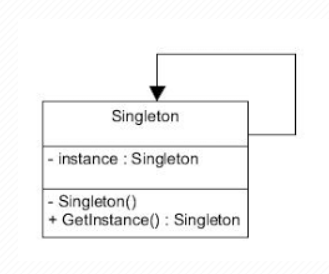


# 5: WEEK 5

2. 1. 1. 1. **Singleton**

**Singleton** là 1 trong 5 design pattern của nhóm **Creational Design Pattern**.

**Singleton** đảm bảo chỉ duy nhất **một thể hiện (instance)** được tạo ra và nó sẽ cung cấp cho bạn một method để có thể truy xuất được thể hiện duy nhất đó mọi lúc mọi nơi trong chương trình.



Sử dụng Singleton khi chúng ta muốn:

* Đảm bảo rằng chỉ có một instance của lớp.
* Việc quản lý việc truy cập tốt hơn vì chỉ có một thể hiện duy nhất.
* Có thể quản lý số lượng thể hiện của một lớp trong giớn hạn chỉ định.
* **Eager initialization**

**Singleton** Class được khởi tạo ngay khi được gọi đến. Đây là cách dễ nhất nhưng nó có một nhược điểm mặc dù instance đã được khởi tạo mà có thể sẽ không dùng tới.

* **Static block initialization**

Cách làm tương tự như **Eager initialization** chỉ khác phần **static block** cung cấp thêm lựa chọn cho việc handle exception hay các xử lý khác.

* **Lazy Initialization**

Là một cách làm mang tính mở rộng hơn so với 2 cách làm trên và hoạt động tốt trong môi trường đơn luồng (single-thread).

Cách này đã khắc phục được nhược điểm của cách **Eager initialization**, chỉ khi nào **getInstance()** được gọi thì instance mới được khởi tạo. Tuy nhiên, cách này chỉ sử dụng tốt trong trường hợp đơn luồng (single-thread), trường hợp nếu có nhiều luồng (multi-thread) cùng chạy và cùng gọi hàm getInstance() tại cùng một thời điểm thì có thể có nhiều hơn 1 thể hiện của instance. Để khắc phục nhược điểm này chúng ta sử dụng **Thread Safe Singleton**.

Một nhược điểm nữa của **Lazy Initialization** cần quan tâm là: đối với thao tác create instance quá chậm thì người dùng có phải chờ lâu cho lần sử dụng đầu tiên.

* **Thread Safe Singleton**

Cách đơn giản nhất là chúng ta gọi phương thức **synchronized** của hàm **getInstance()** và như vậy hệ thống đảm bảo rằng tại cùng một thời điểm chỉ có thể có 1 luồng có thể truy cập vào hàm getInstance() và đảm bảo rằng chỉ có duy nhất 1 thể hiện của class.

Cách này có nhược điểm là một phương thức **synchronized** sẽ chạy rất chậm và tốn hiệu năng, bất kỳ Thread nào gọi đến đều phải chờ nếu có một Thread khác đang sử dụng. Có những tác vụ xử lý trước và sau khi tạo thể hiện không cần thiết phải block. Vì vậy chúng ta cần cải tiến nó đi 1 chút với **Double Check Locking Singleton**.

* **Double Check Locking Singleton**

Để implement theo cách này, chúng ta sẽ kiểm tra sự tồn tại thể hiện của lớp, với sự hổ trợ của đồng bộ hóa, hai lần trước khi khởi tạo. Phải khai báo **volatile** cho instance để tránh lớp làm việc không chính xác do quá trình tối ưu hóa của trình biên dịch.

* **Bill Pugh Singleton Implementation**

Với cách làm này bạn sẽ tạo ra **static nested class** với vai trò 1 Helper khi muốn tách biệt chức năng cho 1 class function rõ ràng hơn. Đây là cách thường hay được sử dụng và có hiệu suất tốt

Khi Singleton được tải vào bộ nhớ thì SingletonHelper chưa được tải vào. Nó chỉ được tải khi và chỉ khi phương thức getInstance() được gọi. Với cách này tránh được lỗi cơ chế khởi tạo instance của Singleton trong Multi-Thread, performance cao do tách biệt được quá trình xử lý. Do đó, cách làm này được đánh giá là cách triển khai Singleton nhanh và hiệu quả nhất.

Bill Pugh Singleton là cách tiếp cận tốt, tuy nhiên, nó dễ dàng bị phá vỡ bởi Reflection.

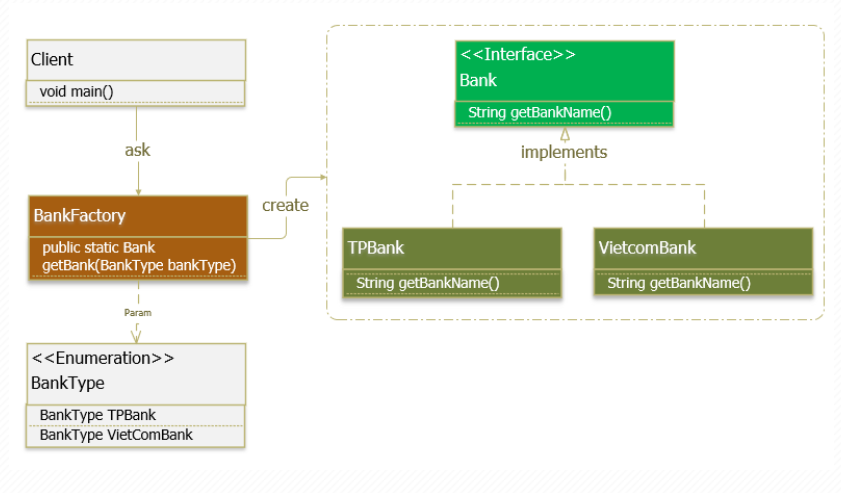
2. 2. 1. **Factory**

**Factory Method Design Pattern**hay gọi ngắn gọn là **Factory Pattern** là một trong những Pattern thuộc nhóm **Creational Design Pattern**. Nhiệm vụ của Factory Pattern là quản lý và trả về các đối tượng theo yêu cầu, giúp cho việc khởi tạo đổi tượng một cách linh hoạt hơn.**Factory Pattern** đúng nghĩa là một **nhà máy**, và nhà máy này sẽ “**sản xuất**” các đối tượng theo yêu cầu của chúng ta.

Trong Factory Pattern, chúng ta tạo đối tượng mà không để lộ logic tạo đối tượng ở phía người dùng và tham chiếu đến đối tượng mới được tạo ra bằng cách sử dụng một interface chung.

Factory Pattern được sử dụng khi có một class cha (super-class) với nhiều class con (sub-class), dựa trên đầu vào và phải trả về 1 trong những class con đó.

Hệ thống được minh họa như sau:



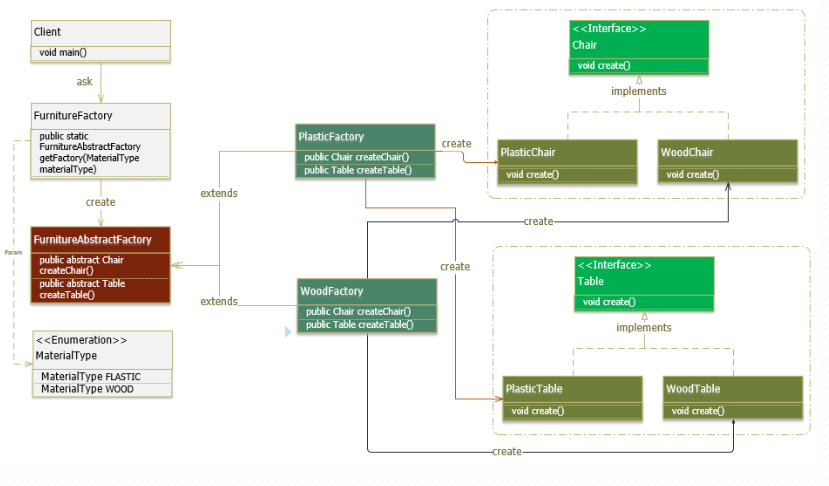
2. 1. 3. **Astract Factory**

**Abstract Factory pattern** là một trong những **Creational pattern**. Nó là phương pháp tạo ra một Super-factory dùng để tạo ra các Factory khác. Hay còn được gọi là Factory của các Factory. Abstract Factory Pattern là một Pattern cấp cao hơn so với Factory Method Pattern.

Trong Abstract Factory pattern, một interface có nhiệm vụ tạo ra một Factory của các object có liên quan tới nhau mà không cần phải chỉ ra trực tiếp các class của object. Mỗi Factory được tạo ra có thể tạo ra các object bằng phương pháp giống như Factory pattern.

Hãy tưởng tượng, Abstract factory như là một nhà máy lớn chứa nhiều nhà máy nhỏ, trong các nhà máy đó có những xưởng sản xuất, các xưởng đó tạo ra những sản phẩm khác nhau.

Hệ thống được minh họa như sau:



2. 1. 4. **Decorator**

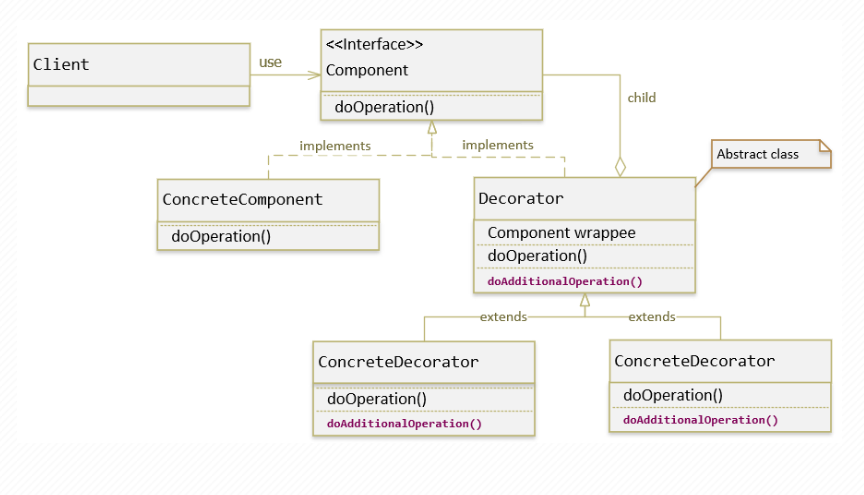
**Decorator pattern** là một trong những Pattern thuộc nhóm cấu trúc (Structural Pattern). Nó cho phép người dùng thêm chức năng mới vào đối tượng hiện tại mà không muốn ảnh hưởng đến các đối tượng khác. Kiểu thiết kế này có cấu trúc hoạt động như một lớp bao bọc (wrap) cho lớp hiện có. Mỗi khi cần thêm tính năng mới, đối tượng hiện có được wrap trong một đối tượng mới (decorator class).

Decorator pattern sử dụng [composition](https://gpcoder.com/4483-huong-dan-java-design-pattern-adapter/#Cai_dat_Adapter_Pattern_nhu_the_nao) thay vì inheritance (thừa kế) để mở rộng đối tượng. Decorator pattern còn được gọi là **Wrapper**hay **Smart Proxy**.

Decorator pattern hoạt động dựa trên một đối tượng đặc biệt, được gọi là decorator (hay wrapper). Nó có cùng một interface như một đối tượng mà nó cần bao bọc (wrap), vì vậy phía client sẽ không nhận thấy khi bạn đưa cho nó một wrapper thay vì đối tượng gốc.

Tất cả các wrapper có một trường để lưu trữ một giá trị của một đối tượng gốc. Hầu hết các wrapper khởi tạo trường đó với một đối tượng được truyền vào constructor của chúng.

Vậy làm thế nào để có thể thay đổi hành vi của đối tượng? Như đã đề cập, wrapper có cùng interface với các đối tượng đích. Khi bạn gọi một phương thức decorator, nó thực hiện cùng một phương thức trong một đối tượng được wrap và sau đó thêm một cái gì đó (tính năng mới) vào kết quả, công việc này tùy thuộc vào logic nghiệp vụ.



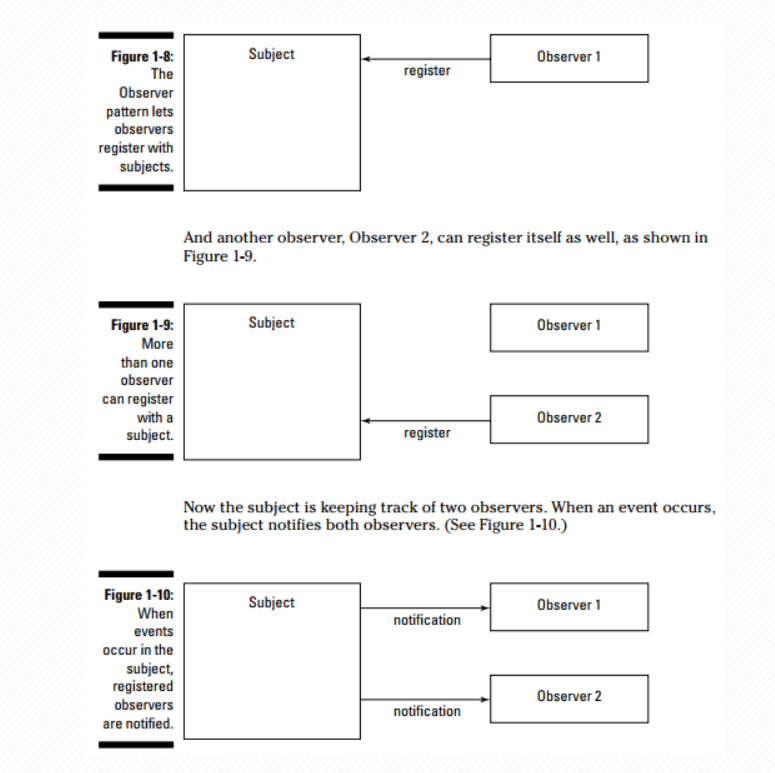
Các thành phần trong mẫu thiết kế Decorator:

* **Component**: là một interface quy định các method chung cần phải có cho tất cả các thành phần tham gia vào mẫu này.
* **ConcreteComponent**: là lớp hiện thực (implements) các phương thức của Component.
* **Decorator**: là một abstract class dùng để duy trì một tham chiếu của đối tượng Component và đồng thời cài đặt các phương thức của Component  interface.
* **ConcreteDecorator**: là lớp hiện thực (implements) các phương thức của Decorator, nó cài đặt thêm các tính năng mới cho Component.
* **Client** : đối tượng sử dụng Component.

2. 1. 5. **Obsever**

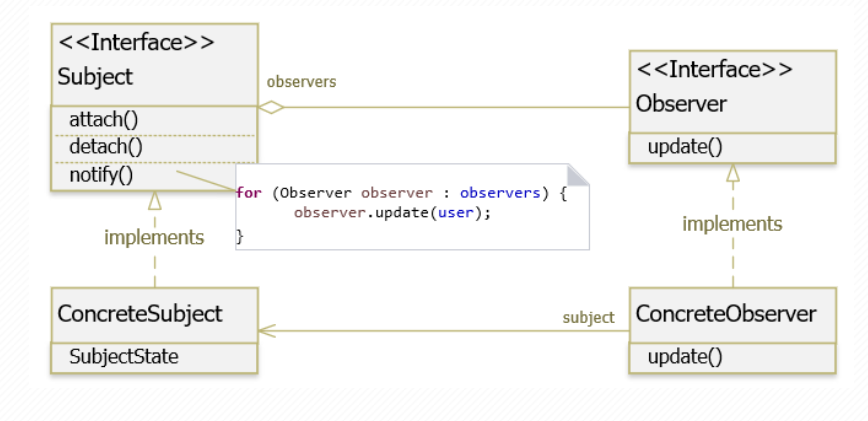
**Observer Pattern** là một trong những Pattern thuộc nhóm hành vi (Behavior Pattern). Nó định nghĩa mối phụ thuộc **một** – **nhiều** giữa các đối tượng để khi mà một đối tượng có sự thay đổi trạng thái, tất các thành phần phụ thuộc của nó sẽ được thông báo và cập nhật một cách tự động.

Observer có thể đăng ký với hệ thống. Khi hệ thống có sự thay đổi, hệ thống sẽ thông báo cho Observer biết. Khi không cần nữa, mẫu Observer sẽ được gỡ khỏi hệ thống.



* Hình 1-8, cho phép observer thứ 1 đăng ký với hệ thống.
* Hình 1-9, cho phép observer thứ 2 đăng ký với hệ thống.
* Hiện tại hệ thống đang liên lạc với 2 observer: Observer 1 và Observer 2. Khi hệ thống phát sinh một sự kiện cụ thể nào đó, nó sẽ thông báo (notification) với cả 2 observer như hình số 1-10.

Observer Pattern còn gọi là **Dependents**, **Publish**/**Subcribe** hoặc **Source**/**Listener**.



Các thành phần tham gia Observer Pattern:

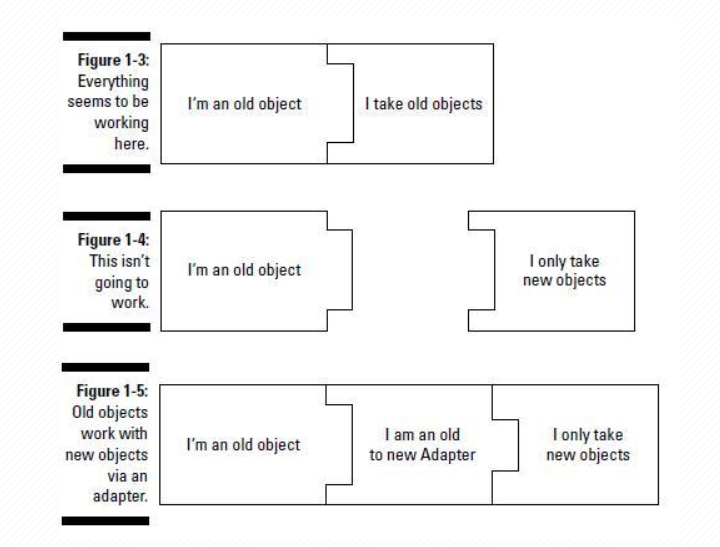
* **Subject** : chứa danh sách các observer,  cung cấp phương thức để có thể thêm và loại bỏ observer.
* **Observer** : định nghĩa một phương thức update() cho các đối tượng sẽ được subject thông báo đến khi có sự thay đổi trạng thái.
* **ConcreteSubject** : cài đặt các phương thức của Subject, lưu trữ trạng thái danh sách các ConcreateObserver, gửi thông báo đến các observer của nó khi có sự thay đổi trạng thái.
* **ConcreteObserver** : cài đặt các phương thức của Observer, lưu trữ trạng thái của subject, thực thi việc cập nhật để giữ cho trạng thái đồng nhất với subject gửi thông báo đến.

Sự tương tác giữa subject và các observer như sau: mỗi khi subject có sự thay đổi trạng thái, nó sẽ duyệt qua danh sách các observer của nó và gọi phương thức cập nhật trạng thái ở từng observer, có thể truyền chính nó vào phương thức để các observer có thể lấy ra trạng thái của nó và xử lý.

2. 1. 6. **Adapter**

Adapter Pattern (Người chuyển đổi) là một trong những Pattern thuộc nhóm cấu trúc (Structural Pattern). Adapter Pattern cho phép các inteface (giao diện) không liên quan tới nhau có thể làm việc cùng nhau. Đối tượng giúp kết nối các interface gọi là Adapter.

Adapter Pattern giữ vai trò trung gian giữa hai lớp, chuyển đổi interface của một hay nhiều lớp có sẵn thành một interface khác, thích hợp cho lớp đang viết. Điều này cho phép các lớp có các interface khác nhau có thể dễ dàng giao tiếp tốt với nhau thông qua interface trung gian, không cần thay đổi code của lớp có sẵn cũng như lớp đang viết.



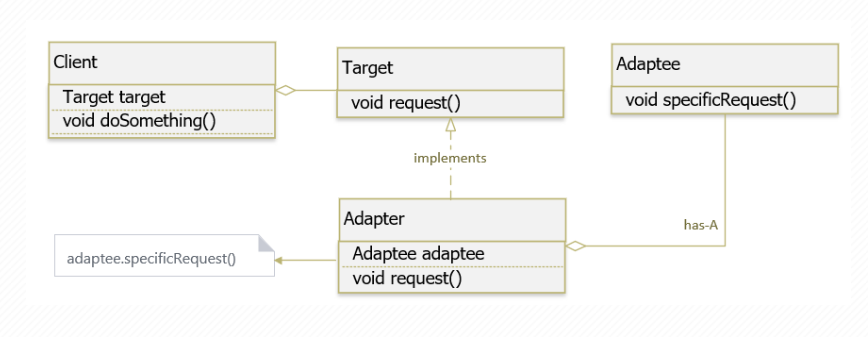
Adapter Pattern còn gọi là **Wrapper Pattern** do cung cấp một interface “bọc ngoài” tương thích cho một hệ thống có sẵn, có dữ liệu và hành vi phù hợp nhưng có interface không tương thích với lớp đang viết.

Một Adapter Pattern bao gồm các thành phần cơ bản sau:

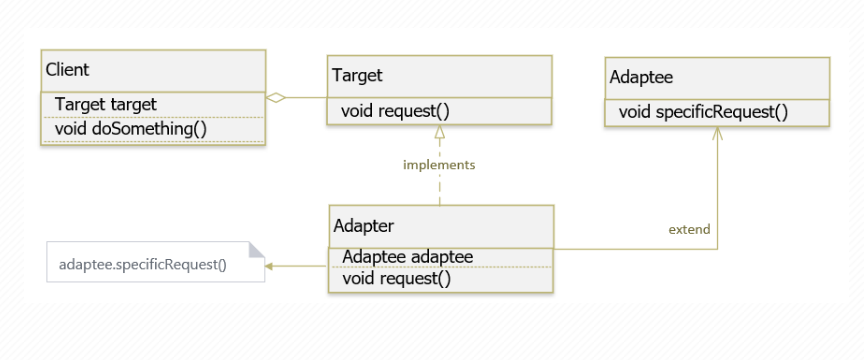
* **Adaptee**: định nghĩa interface không tương thích, cần được tích hợp vào.
* **Adapter**: lớp tích hợp, giúp interface không tương thích tích hợp được với interface đang làm việc. Thực hiện việc chuyển đổi interface cho Adaptee và kết nối Adaptee với Client.
* **Target**: một interface chứa các chức năng được sử dụng bởi Client (domain specific).
* **Client**: lớp sử dụng các đối tượng có interface Target.

Có hai cách để thực hiện Adapter Pattern dựa theo cách cài đặt (implement) của chúng:

* **Object Adapter – Composition**(Tổng hợp): trong mô hình này, một lớp mới (Adapter) sẽ tham chiếu đến một (hoặc nhiều) đối tượng của lớp có sẵn với interface không tương thích (Adaptee), đồng thời cài đặt interface mà người dùng mong muốn (Target). Trong lớp mới này, khi cài đặt các phương thức của interface người dùng mong muốn, sẽ gọi phương thức cần thiết thông qua đối tượng thuộc lớp có interface không tương thích.



**Class Adapter – Inheritance**(Kế thừa) : trong mô hình này, một lớp mới (Adapter) sẽ kế thừa lớp có sẵn với interface không tương thích (Adaptee), đồng thời cài đặt interface mà người dùng mong muốn (Target). Trong lớp mới, khi cài đặt các phương thức của interface người dùng mong muốn, phương thức này sẽ gọi các phương thức cần thiết mà nó thừa kế được từ lớp có interface không tương thích.



So sánh Class Adapter với Object Adapter:

* Sự khác biệt chính là Class Adapter sử dụng Inheritance (kế thừa) để kết nối Adapter và Adaptee trong khi Object Adapter sử dụng Composition (tổng hợp) để kết nối Adapter và Adaptee.
* Trong cách tiếp cận Class Adapter, nếu một Adaptee là một class và không phải là một interface thì Adapter sẽ là một lớp con của Adaptee. Do đó, nó sẽ không phục vụ tất cả các lớp con khác theo cùng một cách vì Adapter là một lớp phụ cụ thể của Adapter.

# 6: WEEK 6

* 1. **Git**

Git là một trong những Hệ thống Quản lý Phiên bản Phân tán, vốn được phát triển nhằm quản lý mã nguồn (source code) của Linux.

Trên Git, ta có thể lưu trạng thái của file dưới dạng lịch sử cập nhật. Vì thế, có thể đưa file đã chỉnh sửa một lần về trạng thái cũ hay có thể biết được file đã được chỉnh sửa chỗ nào.



Thêm nữa, khi định ghi đè (overwrite) lên file mới nhất đã chỉnh sửa của người khác bằng file đã chỉnh sửa dựa trên file cũ, thì khi upload lên server sẽ hiện ra cảnh cáo. Vì thế, sẽ không xảy ra lỗi khi ghi đè lên nội dung chỉnh sửa của người khác mà không hề hay biết.

Git sử dụng mô hình phân tán, ngược lại so với SVN hoặc CSV. Mỗi nơi lưu source sẽ đc gọi là repositories, không cần lưu trữ tập trung một nơi, mà mỗi thành viên trong team sẽ có một repository ở máy của riêng mình. Điều đó có nghĩa là nếu có 3 người A,B,C cùng làm việc trong 1 project. Thì bản thân repo trên máy của người A, người B, và người C có thể kết nối được với nhau.

Khi quyết định thay đổi chỗ nào đó lên server ta chỉ cần một thao tác “push” nó lên server. Chúng ta vẫn có thể share thay đổi của chúng ta cho thành viên khác, bằng cách commit hoặc update trực tiếp từ máy của họ mà không phải thông qua repositories gốc trên server (thông qua share ssh cho nhau).

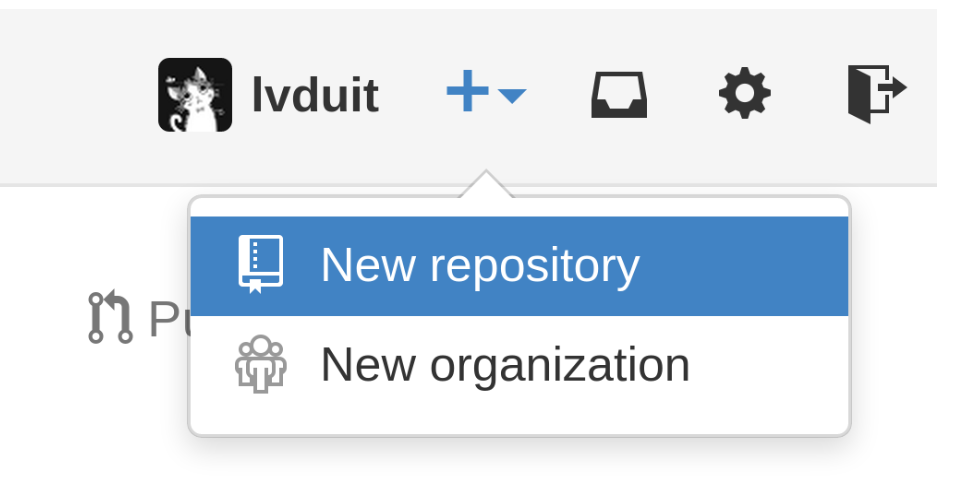
Lợi ích

* An toàn hơn (vì mỗi bản copy của thành viên đều là full copy từ repository gốc, khi server bị down).
* Các thành viên vẫn có thể làm việc offline, họ vẫn có thể commit và update trên local của họ hoặc thậm chí với nhau mà không cần thông qua server.
* Khi server hoạt động trở lại, họ có thể cập nhật tất cả lên lại server.

Tiếp đây mình xin giới thiệu về các khái niệm mà bạn cần nắm trong git:

**- Kho Repo (Repository)**

Repository hay được gọi tắt là Repo, đơn giản là nơi chứa tất cả những thông tin cần thiết để duy trì và quản lý các sửa đổi và lịch sử của toàn bộ project. Trong Repo có 2 cấu trúc dữ liệu chính là Object Store và Index. Tất cả dữ liệu của Repo đèu được chứa trong thư mục bạn đang làm việc dưới dạng folder ẩn có tên là .git

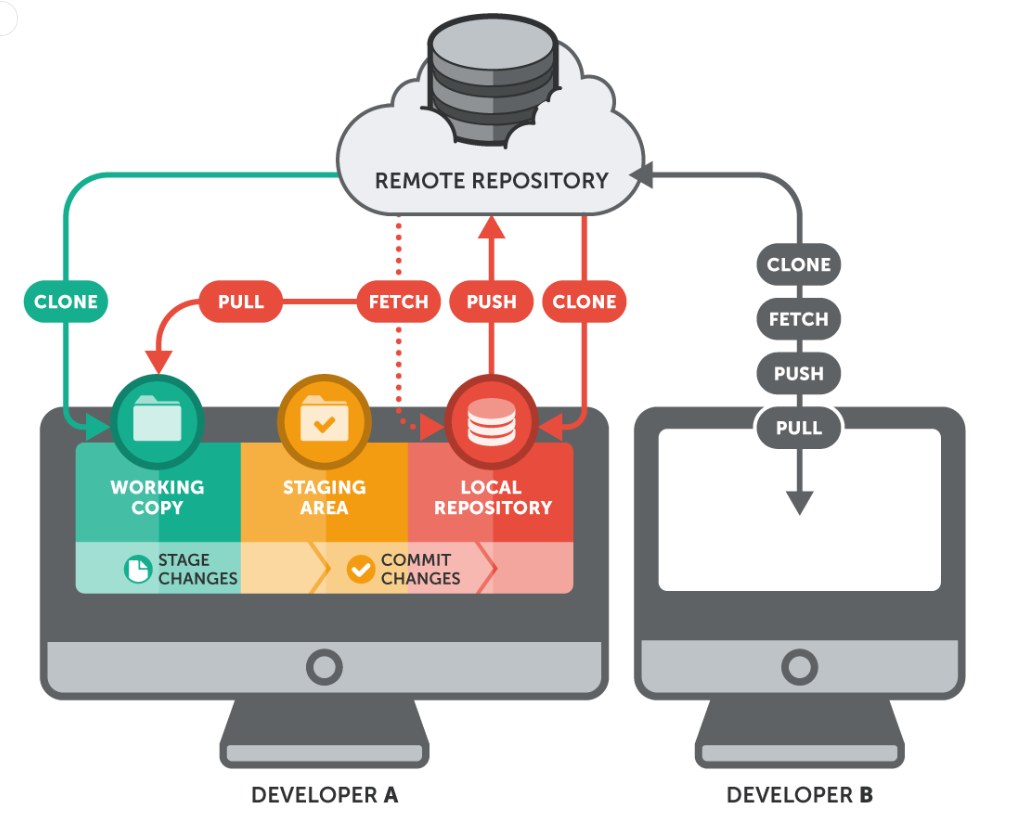


**-** **Remote repository và local repository**

Đầu tiên, repository của Git được phân thành 2 loại là remote repository và local repository.

* Remote repository: Là repository để chia sẻ giữa nhiều người và bố trí trên server chuyên dụng.
* Local repository: Là repository bố trí trên máy của bản thân mình, dành cho một người dùng sử dụng.

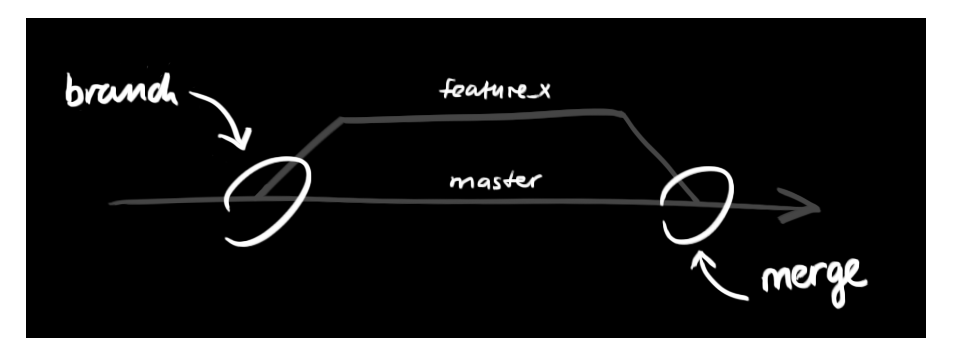
Do repository phân thành 2 loại là local và remote nên với những công việc bình thường thì có thể sử dụng local repository. Khi muốn public nội dung công việc mà mình đã làm trên local repository, thì ta sẽ upload lên remote repository rồi public. Thêm nữa, thông qua remote repository bạn cũng có thể lấy về nội dung thay đổi của người khác.

**- Nhánh (brand)**

Đây là một trong những thế mạnh của git là nhánh. Với git, việc quản lý nhánh rất dễ dàng. Mỗi nhánh trong Git gần giống như một workspace. Việc nhảy vào một nhánh để làm việc trong đó tương tự việc chuyển qua ngữ cảnh làm việc mới, và sau đó có thể nhanh chóng quay lại ngữ cảnh cũ.

Nhánh (branch) được dùng để phát triển tính năng mới mà không làm ảnh hưởng đến code hiện tại.

Nhánh master là nhánh “mặc định” khi bạn tạo một repository. Nhánh master thông thường là nhánh chính của ứng dụng. Ví dụ bạn thử nghiệm một tính năng mới và muốn không ảnh hưởng đến code chính bạn có thể tạo một nhánh mới và sau khi xong sẽ hợp nhất lại với nhánh master. Việc hợp nhất 2 nhánh lại được gọi là merge.

**- Trộn (Merge)**

Trộn source từ một nhánh khác vào nhánh hiện tại. Bạn sẽ hiểu sâu hơn về nó trong các ví dụ ở các bài sau. Chú ý:

* Kiểm tra branch hiện đang làm việc trước khi merge
* Phải đẩy tất cả những thay đổi dưới máy local lên Git trước khi merge
* Trước khi merge phải lấy hết những thay đổi mới nhất của các branch khác, hay ít nhất là branch cần merge về máy
* Merge thành công thì nên đẩy source lên lại lên server
* Nên merge bằng GUI tool.

**- Xung đột (Conflict)**

Conflic là trường hợp có 2 sự thay đổi trong một dòng code và máy tính không thể tự quyết định dòng code nào là “đúng”. Đúng ở đây có nghĩa là “ý đồ của lập trình viên”.

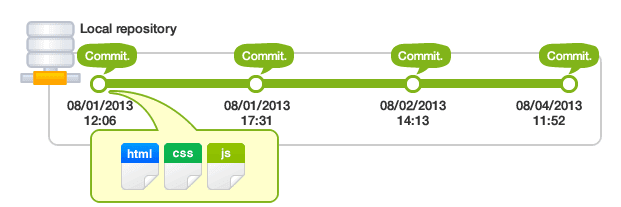
Để giải quyết mâu thuẫn bạn phải dùng “tay không” để sữa các xung đột này. Bạn chỉ việc nhìn vào file bị conflict và tự quyết định dòng code nào giữ lại, dòng nào xóa bỏ.

**- Commit**

Để ghi lại việc thêm/thay đổi file hay thư mục vào repository thì sẽ thực hiện thao tác gọi là Commit.

Khi thực hiện commit, trong repository sẽ tạo ra commit (hoặc revision) đã ghi lại sự khác biệt từ trạng thái đã commit lần trước với trạng thái hiện tại.

Commit này đang được chứa tại repository, các commit nối tiếp với nhau theo thứ tự thời gian. Bằng việc lần theo commit này từ trạng thái mới nhất thì có thể biết được lịch sử thay đổi trong quá khứ hoặc nội dung thay đổi đó.



Các commit này, được đặt tên bởi 40 ký tự alphabet (mã md5 thì phải) không trùng nhau được băm từ thông tin commit. Bằng việc đặt tên cho commit, có thể chỉ định commit nào từ trong repository.

Mỗi commit đều có yêu cầu phải có commit message, để giải thích commit này là bạn đã làm gì trong này.

* 1. **Maven**



Apache maven khởi đầu như một cố gắng để đơn giản hóa quy trình build cho dự án Apache Jakarta Alexandria bây giờ đã không còn tồn tại và sau đó Maven trở thành một tiêu chí trong các dự án Apache Turbine, khi đó nó dùng để thay thế các hệ thống yếu được xây dựng bởi Apache ANT.

Do tiềm năng to lớn và hiệu quả của Maven trong việc giải quyết những thách thức chính ngày nay chúng ta phải đối mặt. nó chở nên rất phổ biến và được sử dụng rộng rãi không chỉ bởi những developer mà còn những người khác bao gồm Scum master, Product owners và Project manager.

Trong những năm gần đây, Maven nổi lên như một phần nhân lực quan trọng cho việc phát trieenr phần mềm.

Trong website chính thức <http://maven.apache.org/> . Các tiêu chí của Apache maven được liệt kê như sau:

* Tạo ra một quy trình build dễ dàng.
* Cung cấp một hệ thống build chuẩn.
* Cung cấp thông tin project chất lượng.
* Cung cấp hướng dẫn tốt nhất cho việc thực hiện phát triển.
* Cho phép chuyển đổi tới những tính năng mới.

Trong bài giới thiệu lần này mình sẽ cơ bản đi qua các mục như sau:

* Cài đặt maven.
* Xác nhận việc cài đặt Maven.
* Tạo mới một dự án Maven.
* Compiling và Testing một project.
* Hiểu về Project Object Model.
* Hiểu về vòng đời Build.
* Hiểu về build profile.

**- Cài đặt**

1. Vào trang chủ của Maven và vào mục download, sau đó chọn bản Maven tương ứng với hệ điều hành (Window, Linux). Bài viết này sẽ hướng dẫn cách cài đặt Maven trên Window, bạn hãy down file .zip về sau đó giải nén ra như thư mục dưới đây

2. Đặt biến môi trường trỏ đến thư mục và ta vừa extract ở trên

3. Để biết được đã cài đặt thành công hay chưa, ta vào command line của window và gõ mvn -version. Nếu đã cài đặt thành công thì sẽ như dưới đây

* 1. **Spring Framework**

**- Giới thiệu**

Spring là một Framework phát triển các ứng dụng Java được sử dụng bởi hàng triệu lập trình viên. Nó giúp tạo các ứng dụng có hiệu năng cao, dễ kiểm thử, sử dụng lại code…

Spring nhẹ và trong suốt (nhẹ: kích thước nhỏ, version cơ bản chỉ khoảng 2MB; trong suốt: hoạt động một cách trong suốt với lập trình viên)

Spring là một mã nguồn mở, được phát triển, chia sẻ và có cộng đồng người dùng rất lơn.

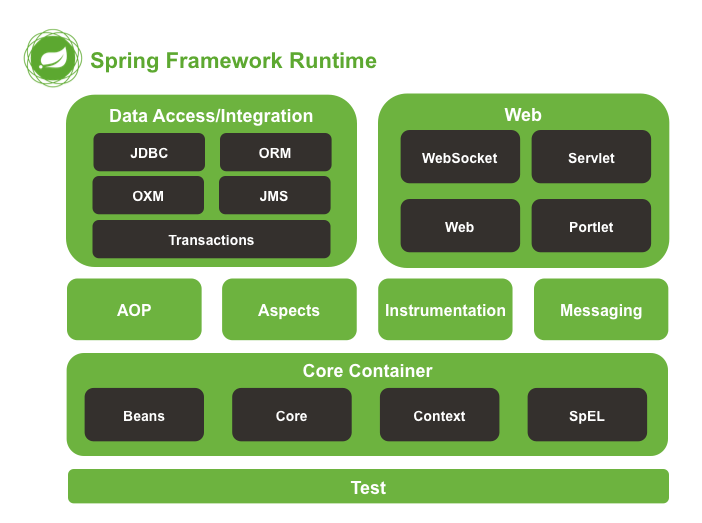
Spring Framework được xây dựng dựa trên 2 nguyên tắc design chính là: Dependency Injection và Aspect Oriented Programming.

Những tính năng core (cốt lõi) của Spring có thể được sử dụng để phát triển Java Desktop, ứng dụng mobile, Java Web. Mục tiêu chính của Spring là giúp phát triển các ứng dụng J2EE một cách dễ dàng hơn dựa trên mô hình sử dụng POJO (Plain Old Java Object)

**- Kiến trúc**

Spring được chia làm nhiều module khác nhau, tùy theo mục đích phát triển ứng dụng mà ta dùng 1 trong các module đó.

Dưới đây là kiến trúc tổng thể của Spring Framework.



**- Test**

Tầng này cung cấp khả năng hỗ trợ kiểm thử với JUnit và TestNG.

**- Spring core container**

Bao gồm các module spring core, beans, context và expression languate (EL)

* Spring core, bean cung cấp tính năng IOC và Dependency Injection.
* Spring Context hỗ trợ đa ngôn ngữ (internationalization), các tính năng Java EE như EJB, JMX.
* Expression Language được mở rộng từ Expresion Language trong JSP. Nó cung cấp hỗ trợ việc setting/getting giá trị, các method cải tiến cho phép truy cập collections, index, các toán tử logic…

**-** **AOP, Aspects and Instrumentation**

Những module này hỗ trợ cài đặt lập trình hướng khía cạnh (Aspect Oriented Programming), hỗ trợ tích hợp với AspectJ.

**-** **Data Access / Integration**

Nhóm này bao gồm JDBC, ORM, OXM, JMS và module Transaction. Những module này cung cấp khả năng giao tiếp với database

**-** **Web**

Hay còn gọi là Spring MVC Nhóm này gồm Web, Web-Servlet… hỗ trợ việc tạo ứng dụng web.

**-**  **Lợi ích**

Spring cho phép lập trình viên sử dụng POJOs. Việc sử dụng POJOs giúp bạn không phải làm việc với EJB, ứng dụng, các luồng chạy, cấu hình… đơn giản hơn rất nhiều.

* Spring được tổ chức theo kiểu mô đun. Số lượng các gói và các lớp khá nhiều, nhưng bạn chỉ cần quan tâm đến những gì bạn cần và không cần quan tâm đến phần còn lại.
* Spring hỗ trợ sử dụng khá nhiều công nghệ như ORM Framework, các logging framework, JEE, các thư viện tạo lịch trình (Quartz và JDK timer)…
* Module Web của Spring được thiết kế theo mô hình MVC nên nó cung cấp đầy đủ các tính năng giúp thay thế các web framework khác như Struts.
* …

# 7: WEEK 7

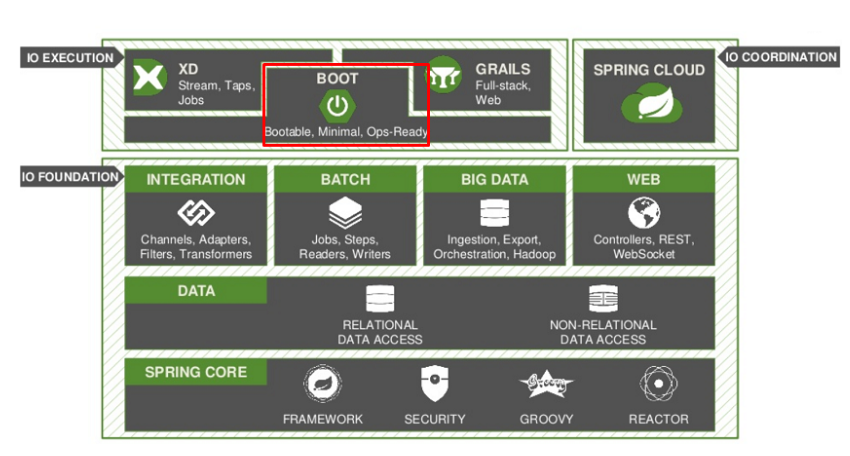
* 1. **Spring Boot**

Mặc dù Spring là một Framework tuyệt vời để phát triển phần mềm nhưng nó vẫn tồn tại một số nhược điểm / hạn chế (Quá nhiều cấu hình…)

Spring Boot được xây dựng để khắc phục những nhược điểm đó, đồng thời cũng cung cấp một hướng phát triển phần mềm mới trong tương lai.

Spring Boot là một module của Spring Framework, cung cấp tính năng RAD (Rapid Application Development) – Phát triển ứng dụng nhanh.

* Spring Boot được dùng để tạo các ứng dụng độc lập dựa trên Spring.
* Spring Boot không yêu cầu cấu hình XML
* Nó là một chuẩn cho cấu hình thiết kế phần mềm, tăng cao năng suất cho developer

****

**-**  **Ưu điểm**

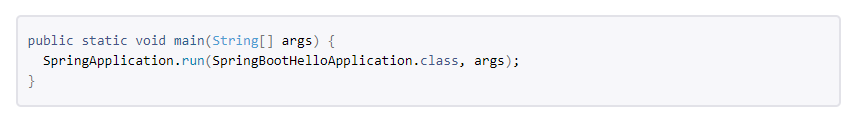
Có các tính năng của Spring Framework.

* Tạo ứng dụng độc lập, có thể chạy bằng java -jar (cho cả java web)
* Nhúng trực tiếp các ứng dụng server (Tomcat, Jetty…) do đó không cần phải triển khai file WAR
* Cấu hình ít, tự động cậu hình bất kì khi nào có thể (Giảm thời gian viết code, tăng năng suất)
* Không yêu cầu XML config…
* Cung cấp nhiều plugin
* Chuẩn cho Microservices (Cloud support; giảm việc setup, config; các thư viện hỗ trợ…)
* …

**-**  **Tính năng**

**SpringApplication**: là một class cung cấp cách thuận tiện để khởi chạy ứng dụng từ hàm main().Để start ứng dụng, chỉ cần gọi method run().

Ví dụ:



* 1. **Spring data jpa**

**-**  **Khái niệm JPA**

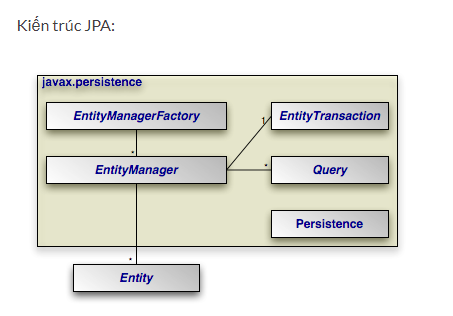
JPA (Java Persistence API) là 1 giao diện lập trình ứng dụng Java, nó mô tả cách quản lý các mối quan hệ dữ liệu  trong ứng dụng sử dụng Java Platform.

JPA cung cấp một mô hình POJO persistence cho phép ánh xạ các table/các mối quan hệ giữa các table trong database sang các class/mối quan hệ giữa các object.

Ví dụ: table Users với các column (Id, name, age…) sẽ tương ứng với class Users.java với các field Id, name, age… từ đó mỗi khi truy vấn table hay các column ta sẽ truy vấn trực tiếp trên các class, các field của class mà không cần quan tâm tới việc đang dùng loại database nào, dữ liệu database ra sao…

Entity: Entity là các đối tượng thể hiện tương ứng 1 table trong cơ sở dữ liệu. Khi lập trình, entity thường là các class POJO đơn giản, chỉ gồm các method getter, setter.

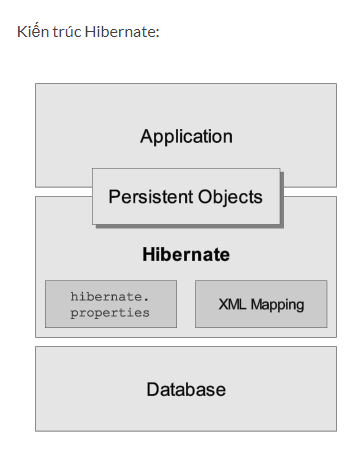
* EntityManager: EntityManager là một giao diện (interface) cung cấp các API cho việc tương tác với các Entity như Persist (lưu một đối tượng mới), merge (cập nhật một đối tượng), remove (xóa 1 đối tượng).
* EntityManagerFactory: EntityManagerFactory được dùng để tạo ra một thể hiện của EntityManager.

****

**-**  **Khái niệm Hibernate**

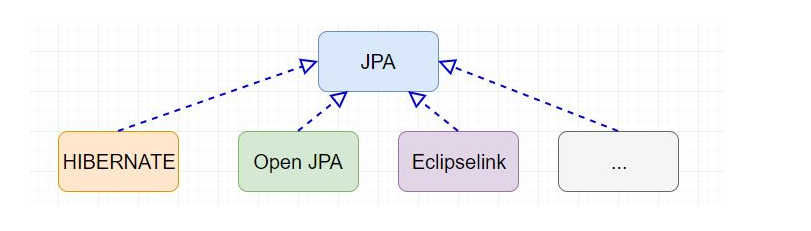
Là 1 ORM (Object Relational Mapping) framework cho phép người lập trình thao tác với database một cách hoàn toàn tự nhiên thông qua các đối tượng. Lập trình viên hoàn toàn không cần quan tâm đến loại database sử dụng, SQL…

Hay nói cách khác, Hibernate chính là cài đặt của JPA (JPA là 1 tập các interface, còn Hibernate implements các interface ấy 1 cách chi tiết).

****

**-**  **Một số Framwork khác**

Ngoài Hibernate ra có 1 số framework khác như Open JPA, Eclipselink cũng thực hiện implements JPA nhưng Hibernate được sử dụng phổ biến hơn cả.

****

**-**  **Sự khác nhau giữa JPA và hibernate**

JPA là tập các giao diện (interfaces), qui chuẩn, định nghĩa. còn Hibernate thực hiện cài đặt (implements) các giao diện đó, định nghĩa đó.

**-**  **Spring Boot JPA**

Spring Boot JPA là một phần trong hệ sinh thái Spring Data, nó tạo ra một layer ở giữa tầng service và database, giúp chúng ta thao tác với database một cách dễ dàng hơn, tự động config và giảm thiểu code thừa thãi.

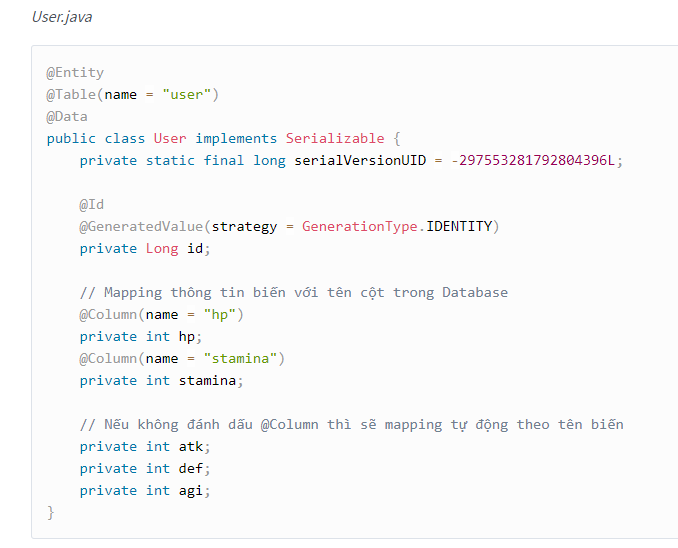
Spring Boot JPA đã wrapper Hibernate và tạo ra một interface mạnh mẽ. Nếu như bạn gặp khó khăn khi làm việc với Hibernate thì đừng lo, bạn hãy để Spring JPA làm hộ.

**-** **Query Creation**

Trong Spring JPA, có một cơ chế giúp chúng ta tạo ra các câu Query mà không cần viết thêm code.

Cơ chế này xây dựng Query từ tên của method.

Ví dụ:

****

Khi chúng ta đặt tên method là: findByAtk(int atk)

Thì Spring JPA sẽ tự định nghĩa câu Query cho method này, bằng cách xử lý tên method. Vậy là chúng ta đã có thể truy vấn dữ liệu mà chỉ mất thêm 1 dòng code.

Cơ chế xây dựng Query từ tên method này giúp chúng ta tiết kiệm thời gian với những query có logic đơn giản, và cũng đặc biệt hữu ích là nó giống ngôn ngữ con người thường nói hơn là SQL. (human-readable)

**-** **Quy tắc đặt tên**

Trong Spring JPA, cơ chế xây dựng truy vấn thông qua tên của method được quy định bởi các tiền tố sau:

find…By, read…By, query…By, count…By, và get…By.

phần còn lại sẽ được parse theo tên của thuộc tính (viết hoa chữ cái đầu). Nếu chúng ta có nhiều điều kiện, thì các thuộc tính có thể kết hợp với nhau bằng chữ And hoặc Or.

* 1. **RESTFULL API**

**-**  **Khái niệm**

REST (REpresenttational State Transfer) lần đầu tiên được giới thiệu vào năm 2000 trong luận văn tiến sĩ của Roy Thomas Fielding (đồng sáng lập giao thức HTTP). Trong luận văn ông giới thiệu khá chi tiết về các ràng buộc, quy ước cũng như cách thức thực hiện với hệ thống để có được một hệ thống REST. Hiểu một cách đơn giản, REST là một hệ thống các ràng buộc (constraints), chỉ cần đảm bảo những điều đó hệ thống của bạn có thể được gọi là RESTfull.

REST định nghĩa các quy tắc kiến trúc để bạn thiết kế Web services, chú trọng vào tài nguyên hệ thống, bao gồm các trạng thái tài nguyên được định dạng như thế nào và được truyền tải qua HTTP, và được viết bởi nhiều ngôn ngữ khác nhau.

REST có kiến trúc đơn giản, định rõ các ràng buộc nhằm tạo ra ứng dụng Web service đạt được những tính chất mong muốn về hiệu suất, khả năng mở rộng, khả năng điều chỉnh v.v..

REST hướng tới việc xây dựng ứng dụng Web service có khả năng làm việc tốt nhất trên môi trường WWW.

Dữ liệu và các tính năng được coi như tài nguyên và được truy suất thông qua các URI (Uniform Resource Identifier)

REST sử dụng 4 phương thức chính của HTTP là POST, GET, PUT và DELETE để thực hiện các hành động CRUD đối với các tài nguyên

Restfull API là một tiêu chuẩn dùng trong việc thiết kế các thiết kế API. Các web service thường được áp dụng các cấu trúc REST vào để xây dựng.

Các cấu trúc cơ bản của cấu trúc REST:

· Sử dụng các phương thức HTTP một cách rõ ràng

· Phi trạng thái

· Hiển thị cấu trúc thư mục như các URLs

· Truyền tải JavaScript Object Notation(JSON), XML hoặc cả hai

**-**  **Giới thiệu POSTMAN**

· POSTMAN là một App Extensions, cho phép làm việc với các API, điển hình là REST.

· Hỗ trợ tất cả các phương thức HTTP (GET, POST, PUT, DELETE, OPTIONS, HEAD …)

**-**  **Sử dụng POSTMAN**

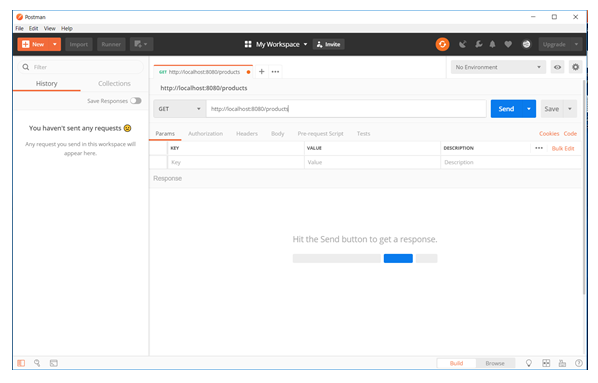
· Bước 1: Nhập URL của web service vào thanh địa chỉ của POSTMAN.

· Bước 2: Chọn phương thức của web service.

· Bước 3: Sửa, thêm các thông tin params phụ thuộc vào web service.

· Bước 4: Nhấn SEND và theo dõi kết quả web service trả ra.

Đây là giao diện chính của POSTMAN:

****

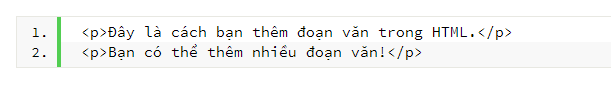
# 8: WEEK 8

* 1. **HTML**

HTMLl à chữ viết tắt của Hypertext Markup Language. Nó giúp người dùng tạo và cấu trúc các thành phần trong trang web hoặc ứng dụng, phân chia các đoạn văn, heading, links, blockquotes, vâng vâng.

HTML không phải là ngôn ngữ lập trình, đồng nghĩa với việc nó không thể tạo ra các chức năng “động” được. Nó chỉ giống như Microsoft Word, dùng để bố cục và định dạng trang web.

Khi làm việc với HTML, chúng ta sẽ sử dụng cấu trúc code đơn giản (tags và attributes) để đánh dấu lên trang web. Ví dụ, chúng ta có thể tạo một đoạn văn bằng cách đặt văn bản vào trong cặp tag mở và đóng văn bản <p> và </p>

****

**-**  **Lịch sử HTML**

HTML được sáng tạo bởi Tim Berners-Lee, nhà vật lý học của trung tâm nghiên cứu CERN ở Thụy Sĩ. Anh ta đã nghĩ ra được ý tưởng cho hệ thống hypertext trên nền Internet.

Hypertext có nghĩa là văn bản chứa links, nơi người xem có thể truy cập ngay lập tức. Anh xuất bản phiên bản đầu tiên của HTML trong năm 1991 bao gồm 18 tag HTML. Từ đó, mỗi phiên bản mới của HTML đều có thêm tag mới và attributes mới.

Theo Mozilla Developer Network: HTML Element Reference, hiện tại có hơn 140 HTML tags, mặc dù một vài trong số chúng đã bị tạm ngưng (không hỗ trợ bởi các trình duyệt hiện đại).

Nhanh chóng phổ biến ở mức độ chóng mặt, HTML được xem như là chuẩn mật của một website. Các thiết lập và cấu trúc HTML được vận hành và phát triển bởi World Wide Web Consortium (W3C). Bạn có thể kiểm tra tình trạng mới nhất của ngôn ngữ này bất kỳ lúc nào trên trang W3C’s website.

Nâng cấp mới nhất gần đây là vào năm 2014, khi ra mắt chuẩn HTML5. Nó thêm vài tags vào markup, để xác định rõ nội dung thuộc loại là gì, như là <article>, <header>, và <footer>.

**-**  **Cách hoạt động của HTML**

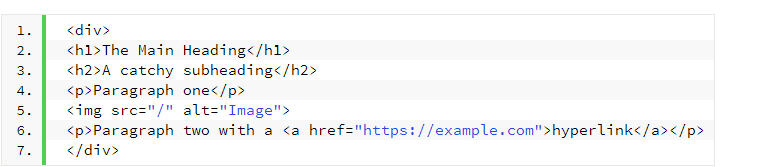
HTML documents là files kết thúc với đuôi .html hay .htm. Bạn có thể xem chúng bằng cách sử dụng bất kỳ trình duyệt web nào (như Google Chrome, Safari, hay Mozilla Firefox). Trình duyệt đọc các files HTML này và xuất bản nội dung lên internet sao cho người đọc có thể xem được nó.

Thông thường, trung bình một web chứa nhiều trang web HTML, ví dụ như: trang chủ, trang about, trang liên hệ, tất cả đều cần các trang HTML riêng.

Mỗi trang HTML chứa một bộ các tag (cũng được gọi là elements), bạn có thể xem như là việc xây dựng từng khối của một trang web. Nó tạo thành cấu trúc cây thư mục bao gồm section, paragraph, heading, và những khối nội dung khác.

Hầu hết các HTML elements đều có tag mở và tag đóng với cấu trúc như <tag></tag>.

Bên dưới, bạn có thể xem code ví dụ của một trang HTML được cấu trúc như thế nào:

****

Element ngoài cùng là bộ tag division (<div></div>), dùng để mark up cho phần nội dung lớn.Nó bao gồm một tag tiêu đề (<h1></h1>), một tag subheading (<h2></h2>), 2 văn bản (<p></p>), và một bước hình (<img>).Đoạn văn thứ 2 chứa tag chứa link (<a></a>) với attribute href chứa địa chỉ URL đích.Tag hình ảnh cũng có 2 attribute: src cho ảnh, và alt cho mô tả của hình.

**-**  **Tổng quan các tag thông dụng của HTML**

HTML tags có 2 loại chính: block-level và inline tags.

1. Elements Block-level sẽ sử dụng toàn không gian trang web và luôn bắt đầu dòng mới của trang web. Headings và paragraph là những ví dụ chính của block tags.
2. Inline elements chỉ chiếm phần nhỏ không gian web và không bắt đầu dòng mới của trang web. Chúng thường dùng để định dạng nội dung bên trong của block level elements. Links và những tag nhấn mạnh là những tag inlines phổ biến.

**-**  **Sự phát triển của html**

Từ những ngày đầu tiên, HTML đã là một phát minh vượt thời đại. W3C liên tục xuất bản các bản cập nhật và nâng cấp.

HTML4 (cũng thường được gọi là “HTML”) được phát hành năm 1999, trong khi phiên bản mới nhất, lớn nhất là HTML5 thì phát hành năm 2014. Phiên bản này đã ra mắt rất nhiều tính năng quan trọng mới.

Một trong số các tính năng hấp dẫn nhất của HTML5 là khả năng hỗ trợ audio và video. Thay vì sử dụng Flash Player, chúng ta giờ đây chỉ cần nhúng videos và audio files vào trong trang web bằng tag <audio></audio> hay <video></video>. Nó cũng hỗ trợ những tính năng mặc định như hỗ trợ scalable vector graphics (SVG) và MathML cho các công thức toán học và phương trình khoa học.

HTML5 có rất nhiều cải thiện về mặt ngôn ngữ. Các tag mới nhằm giới thiệu cho trình duyệt biết mục đích chính của nội dung trong website là gì, việc này có lợi cho cả người đọc và cả search engine.

Các semantic tags đó là <article></article>, <section></section>, <aside></aside>, <header></header>, và <footer></footer>.

**-**  **Ưu và nhược điểm của html**

Như hầu hết mọi chuyện, HTML có ưu và nhược điểm của riêng nó.

Ưu điểm:

* Ngôn ngữ được sử dụng rộng lớn này có rất nhiều nguồn tài nguyên hỗ trợ và cộng đồng sử dụng cực lớn.
  + Sử dụng mượt mà trên hầu hết mọi trình duyệt.
  + Có quá trình học đơn giản và trực tiếp.
  + Mã nguồn mở và hoàn toàn miễn phí.
  + Markup gọn gàng và đồng nhất.
  + Chuẩn chính của web được vận hành bởi World Wide Web Consortium (W3C).
  + Dễ dàng tích hợp với các ngôn ngữ backend như [PHP](https://www.hostinger.vn/huong-dan/php/) và Node.js.

Khuyết điểm:

* Được dùng chủ yếu cho web tĩnh. Đối với các tính năng động, bạn cần sử dụng JavaScript hoặc ngôn ngữ backend bên thứ 3 như PHP.
  + Nó có thể thực thi một số logic nhất định cho người dùng. Vì vậy, hầu hết các trang đều cần được tạo riêng biệt, kể cả khi nó sử dụng cùng các yếu tố, như là headers hay footers.
  + Một số trình duyệt chậm hỗ trợ tính năng mới.
  + Khó kiểm soát cảnh thực thi của trình duyệt (ví dụ, những trình duyệt cũ không render được tag mới)
  1. **CSS**

CSS là ngôn ngữ tạo phong cách cho trang web – Cascading Style Sheet language. Nó dùng để tạo phong cách và định kiểu cho những yếu tố được viết dưới dạng ngôn ngữ đánh dấu, như là HTML. Nó có thể điều khiển định dạng của nhiều trang web cùng lúc để tiết kiệm công sức cho người viết web. Nó phân biệt cách hiển thị của trang web với nội dung chính của trang bằng cách điều khiển bố cục, màu sắc, và font chữ.

CSS được phát triển bởi W3C (World Wide Web Consortium) vào năm 1996, vì một lý do đơn giản. HTML không được thiết kế để gắn tag để giúp định dạng trang web. Bạn chỉ có thể dùng nó để “đánh dấu” lên site.

Những tag như <font> được ra mắt trong HTML phiên bản 3.2, nó gây rất nhiều rắc rối cho lập trình viên. Vì website có nhiều font khác nhau, màu nền và phong cách khác nhau. Để viết lại code cho trang web là cả một quá trình dài, cực nhọc. Vì vậy, CSS được tạo bởi W3C là để giải quyết vấn đề này.

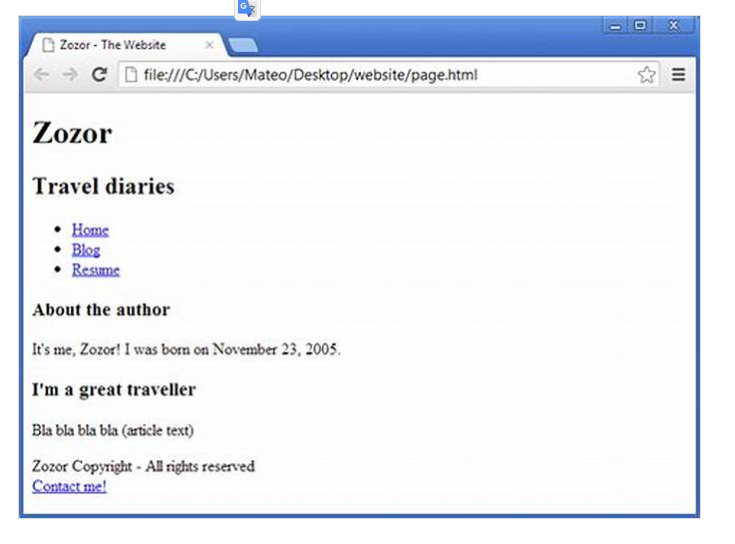
Mối tương quan giữa HTML và CSS rất mật thiết. HTML là ngôn ngữ markup (nền tảng của site) và CSS định hình phong cách (tất cả những gì tạo nên giao diện website), chúng là không thể tách rời.

CSS về lý thuyết không có cũng được, nhưng khi đó website sẽ không chỉ là một trang chứa văn bản mà không có gì khác.

**-**  **Ưu điểm của css**

Sự khác biệt giữa site có CSS và không có CSSrất dễ nhận biết.

Chắc bạn đã thấy rồi, khi website không load được đầy đủ và chỉ có nền trắng và chữ thì chỉ có màu đen và xanh. Như sau:

****

Điều này có nghĩa là thành phần CSS của bạn không tải lên được hay nó không tồn tại.

Đó một website không có CSS, vậy bạn có muốn website của mình trông như thế không? Hẵn là không rồi phải không.

Trước khi sử dụng CSS, tất cả những phong cách của CSS cần được đính kèm vào trong HTML markup. Có nghĩa là bạn cần tách ra để xác định các thành phần như background, font colors, canh hàng, vâng vâng.

CSS giúp bnạ định kiểu mọi thứ trên một file khác, bạn có thể tạo phong cách trước rồi sau đó tích hợp file CSS lên trên cùng của file HTML. Việc này giúp HTML markup rõ ràng và dễ quản lý hơn nhiều.

Tóm lại,với CSS bạn không cần lặp lại các mô tả cho từng thanh phần. Nó tiết kiệm thời gian, làm code ngắn lại để bạn có thể kiểm soát lỗi dễ dàng hơn.

CSS giúp bạn có nhiều styles trên một trang web HTML, vì vậy, khả năng điều chỉnh trang gần như vô hạn

**-**  **Cách hoạt động của CSS**

CSS sử dụng cấu trúc tiếng Anh đơn giản để tạo ra một bộ các quy tắc bạn có thể tận dụng. Như đã nói ở trên, HTML không được dùng để tạo phong cách cho các yếu tố, nó chỉ đánh dấu từng phần để biết được yếu tố đó là gì thôi. Ví dụ: <p>Đây là văn bản.</p>.

Còn làm thế nào để tạo ra phong cách cho văn bản đó? Syntax của CSS rất đơn giản. Nó có phần block chọn và block khai báo. Bạn chọn một yếu tố và khai báo làm gì với nó. Rất đơn giản phải không?

Tuy nhiên, cũng có nhiều quy tắc cần ghi nhớ.

Selector sẽ trỏ về yếu tố HTML bạn cần muốn tạo phong cách. Block khai báo sẽ bao gồm một hay nhiều khai báo cách nhau bởi dấu chấm phẩy,

Mỗi khai báo bao gồm một tên CSS và giá trị, cách nhấu bởi dấu 2 chấm. Khai báo CSS luôn kết thúc bằng dấu chấm phẩn, và block khai báo được đặt trong dấu ngoặc nhọn.

**-** **Internal, External và Inline CSS Styles**

Chúng tôi sẽ nói ngắn gọn và từng loại style css, ở dưới cùng là bài viết chi tiết hơn cho những ai muốn tìm hiểu sâu hơn.

Style CSS Internal là style được tải lên mỗi khi trang web được refresh. Vì vậy nó tăng thời gian tải trang. Ngoài ra, bạn sẽ không dùng một style CSS cho nhiều trang vì nó chỉ áp dụng cho từng trang một. Tuy nhiên, lợi ích của style CSS Internal là khi mọi thứ đã đặt trong một trang thì nó dễ chia sẽ trang để xem trước hơn.

Phương pháp dùng Style External là thuận tiên nhất. Mọi thứ được lưu trong file .css. Có nghĩa là bạn có thể tạo phong cách ở file khác áp dụng CSS vào trang bạn muốn. External style sẽ cải thiện thời gian tải trang rất nhiều.

Cuối cùng, chúng ta sẽ nói về style CSS Inline. Inline hoạt động với một yếu tố nhất định có tag <style>. Mỗi thành phần đều cần được tạo phong cách riêng, vì vậy đây không hẵn là cách tốt nhất và dễ nhất để xử lý CSS. Nhưng có thể khá tiện lợi, vì nếu bạn muốn thay đổi chỉ một yếu tố, nhanh chóng xem trước thay đổi, bạn không cần truy cập trực tiếp vào file CSS để chỉnh sửa mà sử dụng Inline CSS.

* 1. **JavaScript**

**-**  **Lịch sử**

JavaScript được tạo trong mười ngày bởi Brandan Eich, một nhân viên của Netscape, vào tháng 9 năm 1995. Được đặt tên đầu tiên là Mocha, tên của nó được đổi thành Mona rồi LiveScript trước khi thật sự trở thành JavaScript nổi tiếng như bây giờ. Phiên bản đầu tiên của ngôn ngữ này bị giới hạn độc quyền bởi Netscape và chỉ có các tính năng hạn chế, nhưng nó tiếp tục phát triển theo thời gian, nhờ một phần vào cộng đồng các lập trình viên đã liên tục làm việc với nó.

Trong năm 1996, JavaScript được chính thức đặt tên là ECMAScript. ECMAScript 2 phát hành năm 1998 và ECMAScript 3 tiếp tục ra mắt vào năm 1999. Nó liên tục phát triển thành JavaScript ngày nay, giờ đã hoạt động trên khắp mọi trình duyệt và trên khắp các thiết bị từ di động đến máy tính bàn.

JavaScript liên tục phát triển kể từ đó, có lục đạt đến 92% website đang sử dụng JavaScript vào năm 2016. Chỉ trong 20 năm, nó từ một ngôn ngữ lập trình riêng trở thành công cụ quan trọng nhất trên bộ công cụ của các chuyên viên lập trình web. Nếu bạn đang dùng internet, vậy chắc chắn bạn đã từng sử dụng JavaScript rồi.

**-**  **Cách hoạt động của JavaScript**

JavaScript thường được nhúng trực tiếp vào một trang web hoặc được tham chiếu qua file .js riêng. Nó là ngôn ngữ phía client, tức là script được tải về máy của khách truy cập và được xử lý tại đó thay vì phía server là xử lý trên server rồi mới đưa kết quả tới khách truy cập.

Hãy lưu ý là các trình duyệt web phổ biến cũng hỗ trợ việc người dùng có muốn tắt JavaScript hay không. Đó là lý do bạn nên biết trang web sẽ hoạt động như thế nào torng trường hợp không có JavaScript.

**-**  **Ưu điểm của JavaScript**

JavaScript có rất nhiều ưu điểm khiến nó vượt trội hơn so với các đối thủ, đặc biệt trong các trường hợp thực tế. Sau đây chỉ là một số lợi ích của JavaScript:

1. Bạn không cần một compiler vì web browser có thể biên dịch nó bằng HTML;
2. Nó dễ học hơn các ngôn ngữ lập trình khác;
3. Lỗi dễ phát hiện hơn và vì vậy dễ sửa hơn;
4. Nó có thể được gắn trên một số element của trang web hoặc event của trang web như là thông qua click chuột hoặc di chuột tới;
5. JS hoạt động trên nhiều trình duyệt, nền tảng, vâng vâng;
6. Bạn có thể sử dụng JavaScript để kiểm tra input và giảm thiểu việc kiểm tra thủ công khi truy xuất qua database;
7. Nó giúp website tương tác tốt hơn với khách truy cập;
8. Nó nhanh hơn và nhẹ hơn các ngôn ngữ lập trình khác.

**-**  **Khuyết điểm của JavaScript**

Mọi ngôn ngữ lập trình đều có các khuyết điểm. Một phần là vì ngôn ngữ đó khi phát triển đến một mức độ như JavaScript, nó cũng sẽ thu hút lượng lớn hacker, scammer, và những người có ác tâm luôn tìm kiếm những lỗ hổng và các lỗi bảo mật để lợi dụng nó. Một số khuyết điểm có thể kể đến là:

1. Dễ bị khai thác;
2. Có thể được dùng để thực thi mã độc trên máy tính của người dùng;
3. Nhiều khi không được hỗ trợ trên mọi trình duyệt;
4. JS code snippets lớn;
5. Có thể bị triển khai khác nhau tùy từng thiết bị dẫn đến việc không đồng nhất.
   1. **Bootstrap**

Bootstrap là một framework cho phép thiết kế website reponsive nhanh hơn và dễ dàng hơn  
Bootstrap là bao gồm các HTML templates, CSS templates và Javascript tao ra những cái cơ bản có sẵn như: typography, forms, buttons, tables, navigation, modals, image carousels và nhiều thứ khác. Trong bootstrap có thêm các plugin Javascript trong nó. Giúp cho việc thiết kế reponsive của bạn dễ dàng hơn và nhanh chóng hơn.  
Lịch sử Bootstrap  
Bootstrap là dược phát triển bởi Mark Otto và Jacob Thornton tại Twitter. Nó được xuất bản như là một mã nguồn mở vào tháng 8 năm 2011 trên GitHub. Tính ra đến thời điểm mình viết bài viết này nó cũng đã phát triển được 3 năm rồi. Bản bootstrap mới nhất bây giờ là bootstrap 4.



**-**  **Ưu điểm của Bootstrap**

Những điểm thuận lợi khi bản sử dụng bootstrap:

Rất dễ để sử dụng: Nó đơn giản vì nó được base trên HTML, CSS và Javascript chỉ cẩn có kiến thức cơ bản về 3 cái đó là có thể sử dụng bootstrap tốt.  
Tính năng Responsive: Bootstrap’s xây dựng sẵn reponsive css trên các thiết bị phones, tablets, và desktops  
Mobile: Trong Bootstrap 3 mobile-first styles là một phần của core framework  
Tương thích với trình duyệt: Nó tương thích với tất cả các trình duyệt (Chrome, Firefox, Internet Explorer, Safari, and Opera) nhưng lưu ý vì em IE vẫn rất hãm với IE phiên bản cũ vì thế việc IE9 hay IE8 đổ xuống không support là chuyện bình thường. Các bạn có dùng nên luy ý điểm này theo như mình dùng làm dự án thì nó support tuyệt vời trên IE10 đổ lên nhé, thấp nhất có lẽ là IE9 thôi còn IE8 thì không nên xài nhé.

**-**  **Lịch sử**

Bootstrap, ban đầu có tên là Twitter Blueprint, được phát triển bởi Mark Otto và Jacob Thornton tại Twitter như một khuôn khổ để khuyến khích sự nhất quán trên các công cụ nội bộ. Trước khi Bootstrap, các thư viện khác nhau được sử dụng để phát triển giao diện, dẫn đến sự thiếu nhất quán và gánh nặng bảo trì cao.

Sau một vài tháng phát triển bởi một nhóm nhỏ, nhiều nhà phát triển tại Twitter đã bắt đầu đóng góp cho dự án như một phần của Tuần lễ Hack, một tuần theo phong cách hackathon dành cho nhóm phát triển Twitter. Nó được đổi tên từ Twitter Blueprint thành Bootstrap, và được phát hành như một dự án nguồn mở vào ngày 19 tháng 8 năm 2011. Nó tiếp tục được duy trì bởi Mark Otto, Jacob Thornton và một nhóm nhỏ các nhà phát triển cốt lõi, cũng như một cộng đồng lớn người dùng.

# 9: WEEK 9

**-**  **Bài toán:**

Công ty có rất nhiều server để phục vụ cho công việc, một server gồm có tên địa chỉ ip máy server ảo, tên host của server đó, tên chủ server, password của server, trạng thái server đó ….

Do có rất nhiều server và nhân sự quản lý mỗi server thay đổi liên tục nên vấn để kiểm soát server nào không có người dùng đã lâu trở nên rất là khó khăn. Bình thường phải đăng nhập vào từng server kiểm tra xem server này có ai đang dùng hay không, nếu không có ai đang dùng sẽ cấp cho người khác, nhưng do số lượng lên đến vài trăm server thì việc kiểm tra từng server trở nên rất khó khăn và vất vả

Yêu cầu đặt ra là viết một phần mềm chạy trên nền web và chạy trong môi trường server chủ để cùng một lúc có thể kiểm tra tình trạng của các server và thông báo đến người dùng server nào không có người sử dụng hoặc sever bị lỗi để người sử dụng kịp thời xử lý

Chương trình sẽ tạo kết nối ssh đến các server có trong danh sách được yêu cầu kiểm tra và trả về kết qảu thông qua giao diện web.

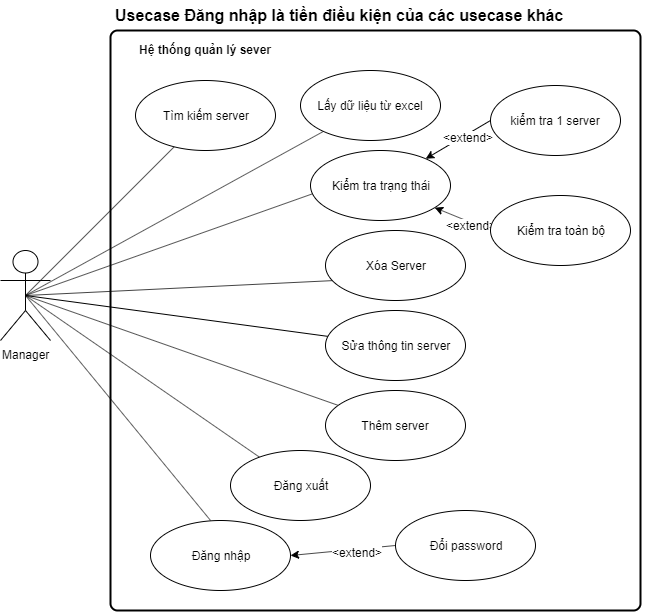
Chương trình gồm các chức năng cơ bản như sau

* Đăng nhập: Chỉ nhửng người được cấp tài khoản mới được sử dụng phần mềm
* Đăng xuất :
* Đổi password: Người sử dụng có thể đổi password mặc định khi được cấp tài khoản
* Thêm một server : Người dùng có thể thêm một server bằng cách điền thông tin của server đó để kiểm tra
* Xóa một server : Người dùng có thể xóa server đã thêm bất kỳ
* Sửa thông tin một server : Người dùng có thể sửa thông tin bất kỳ server nào khi có sự thay đổi xảy ra, hệ thống sẽ tiến hành check server đó lại
* Kiểm tra 1 server : Người dùng có thể kiểm tra trạng thái của bất kỳ server nào theo nhu cầu
* Kiểm tra toàn bộ server : Người dùng có thể kiểm tra toàn bộ server đã thêm
* Thêm danh sách server bằng file excel : Người dùng có thể thêm danh sách server bằng một file excel, với điều kiện file excel đó có định dạng theo quy định của chương trình

Chức năng phi hệ thống

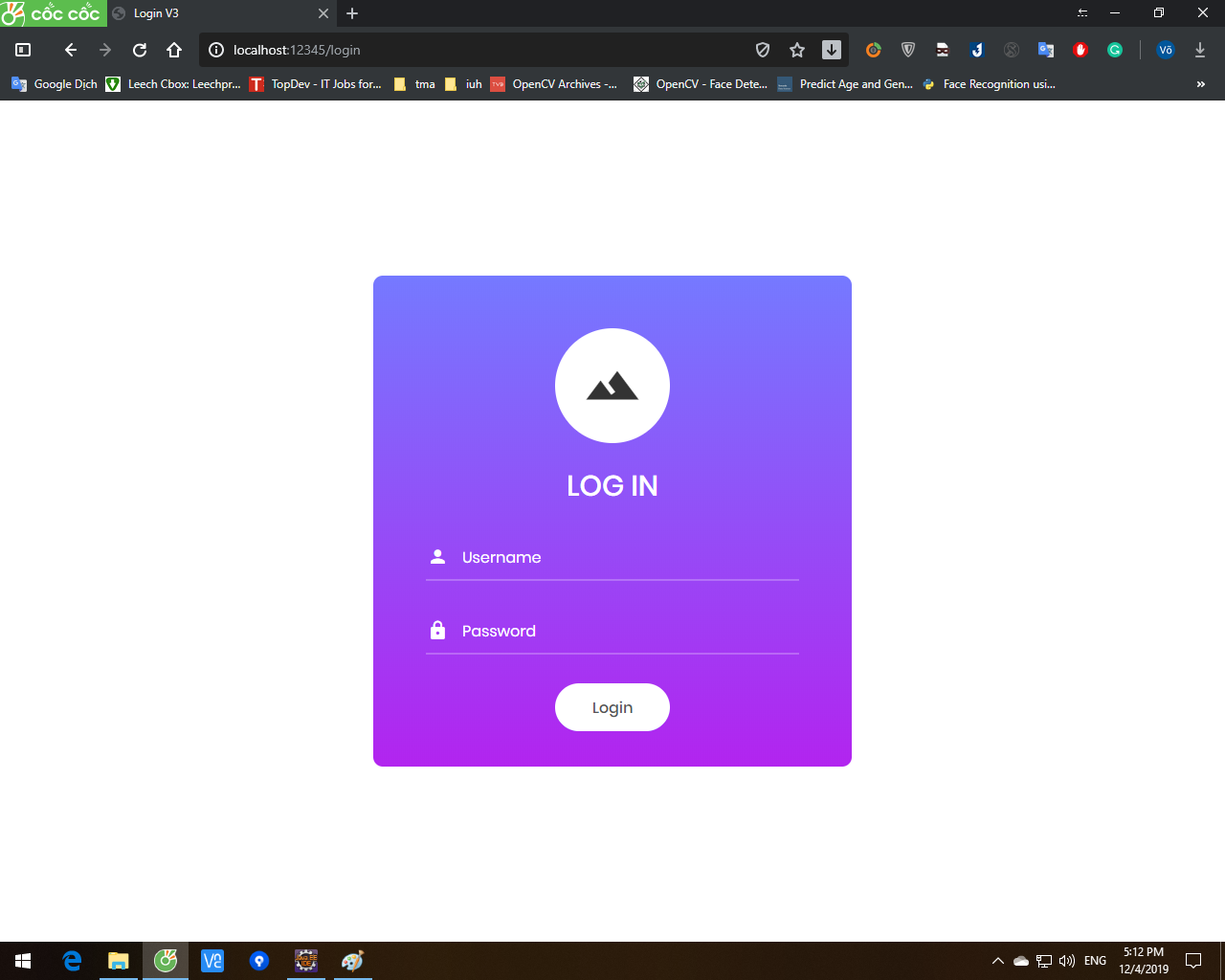
* Thời gian check một server không được quả 30 giây
* Thời gian check 200 server trở xuống không được quá 1 phút
* Giao diện thân thiện với người dùng
* Có thể chạy được trên nhiều hệ điều hành khác nhau

**-**  **Use Case:**

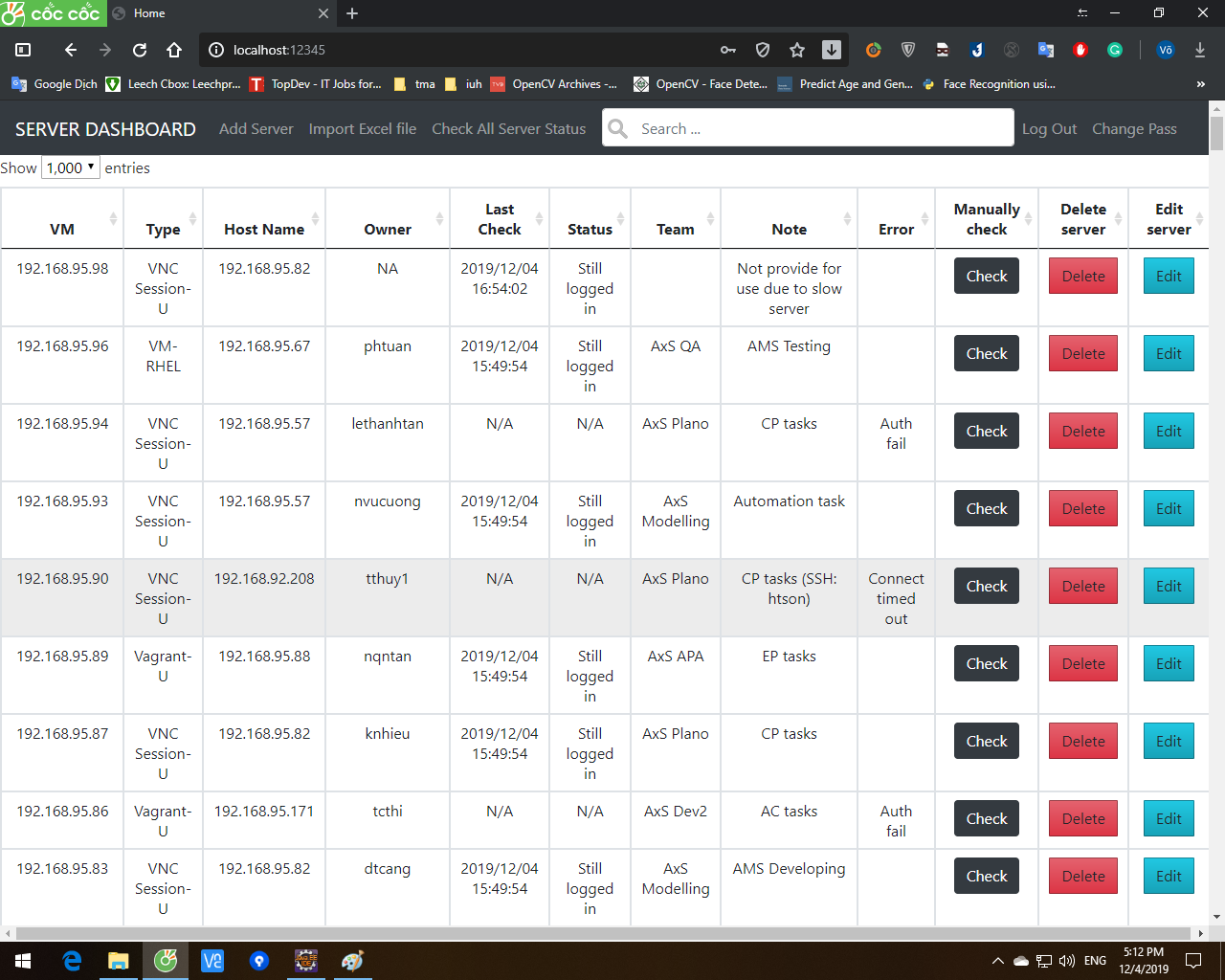
****

# 10: WEEK 10,11

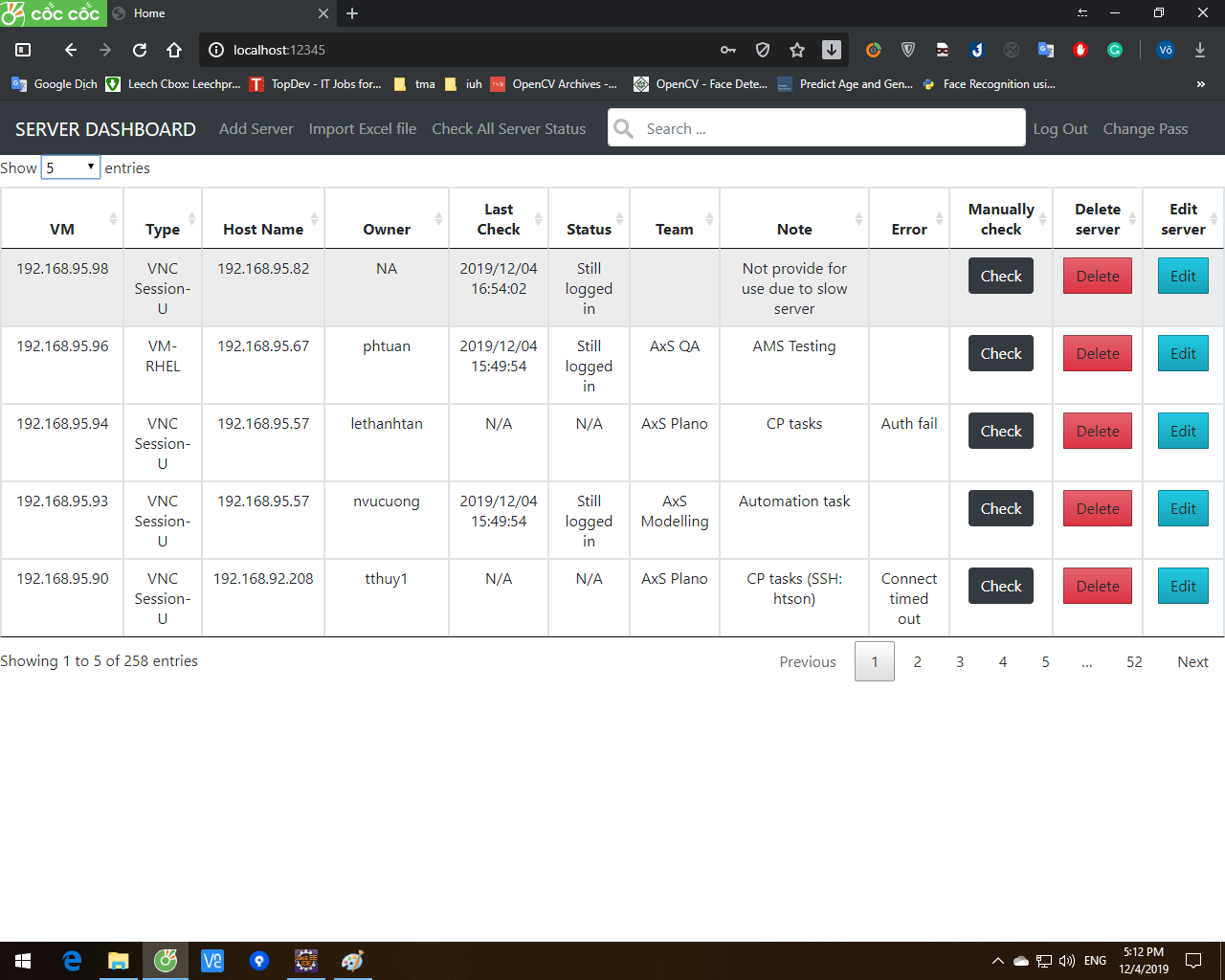
**-**  **Đăng nhập:**

****

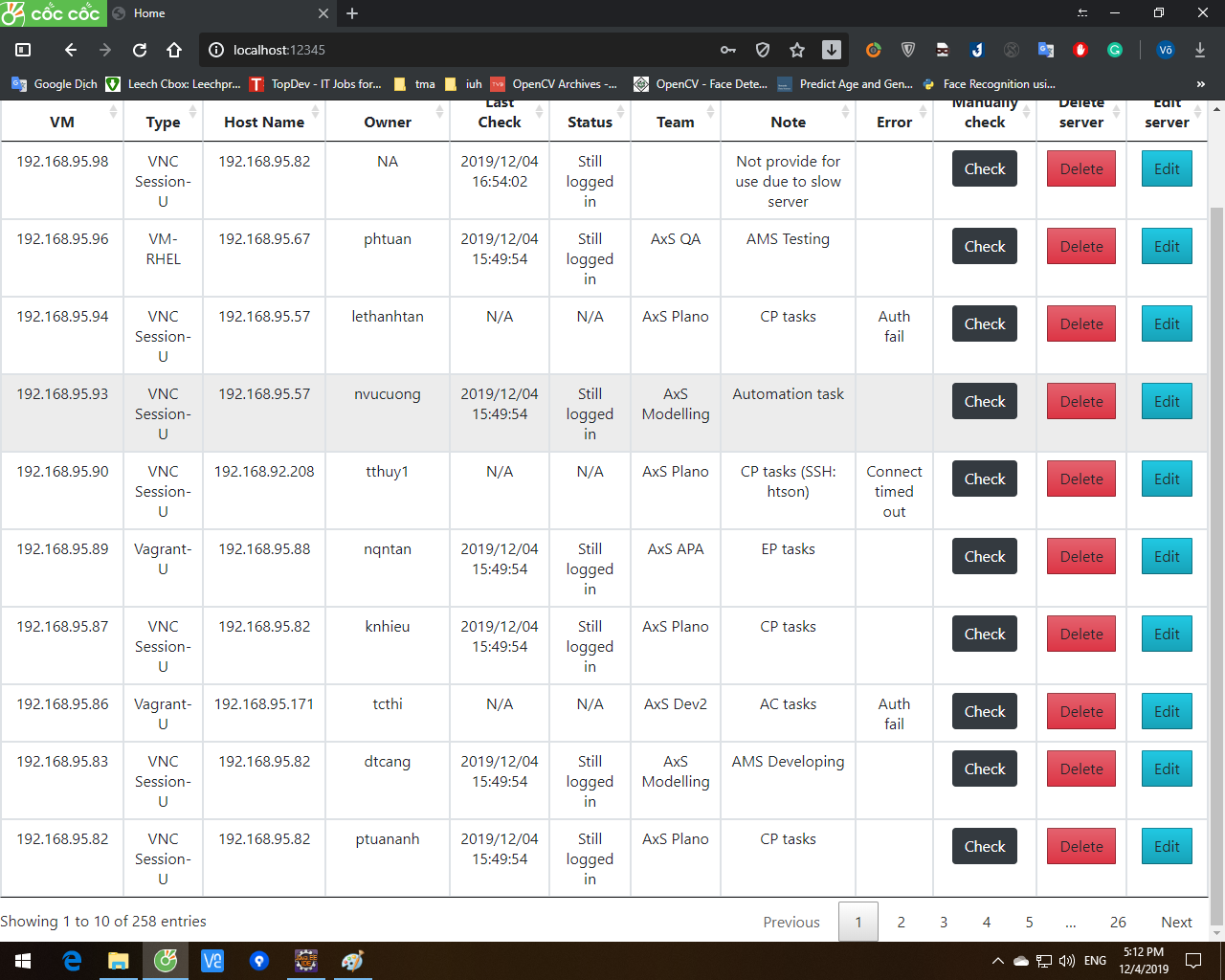
**-**  **Giao diện chính:**

****

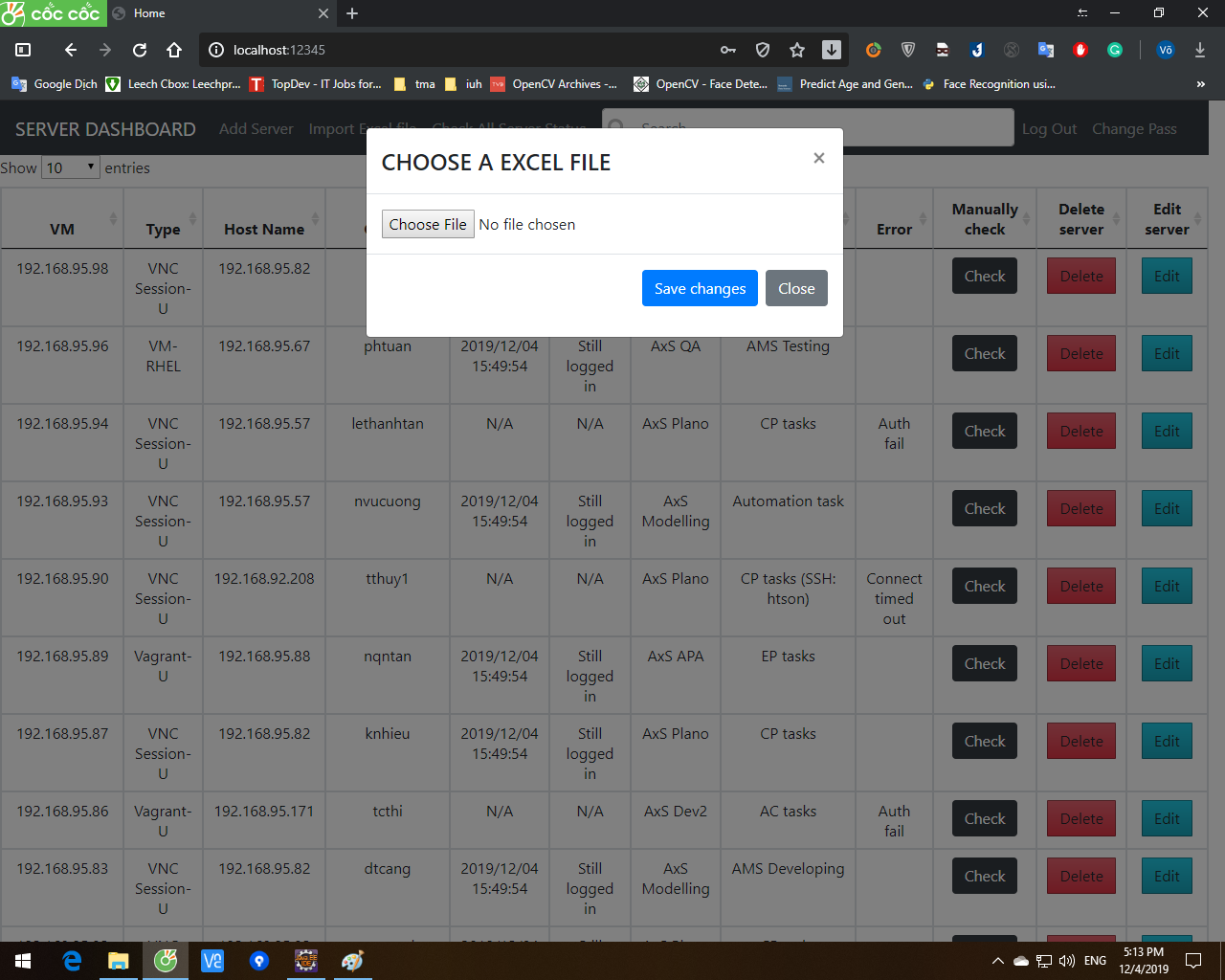
**-**  **Giao diện chính thu gọn:**

****

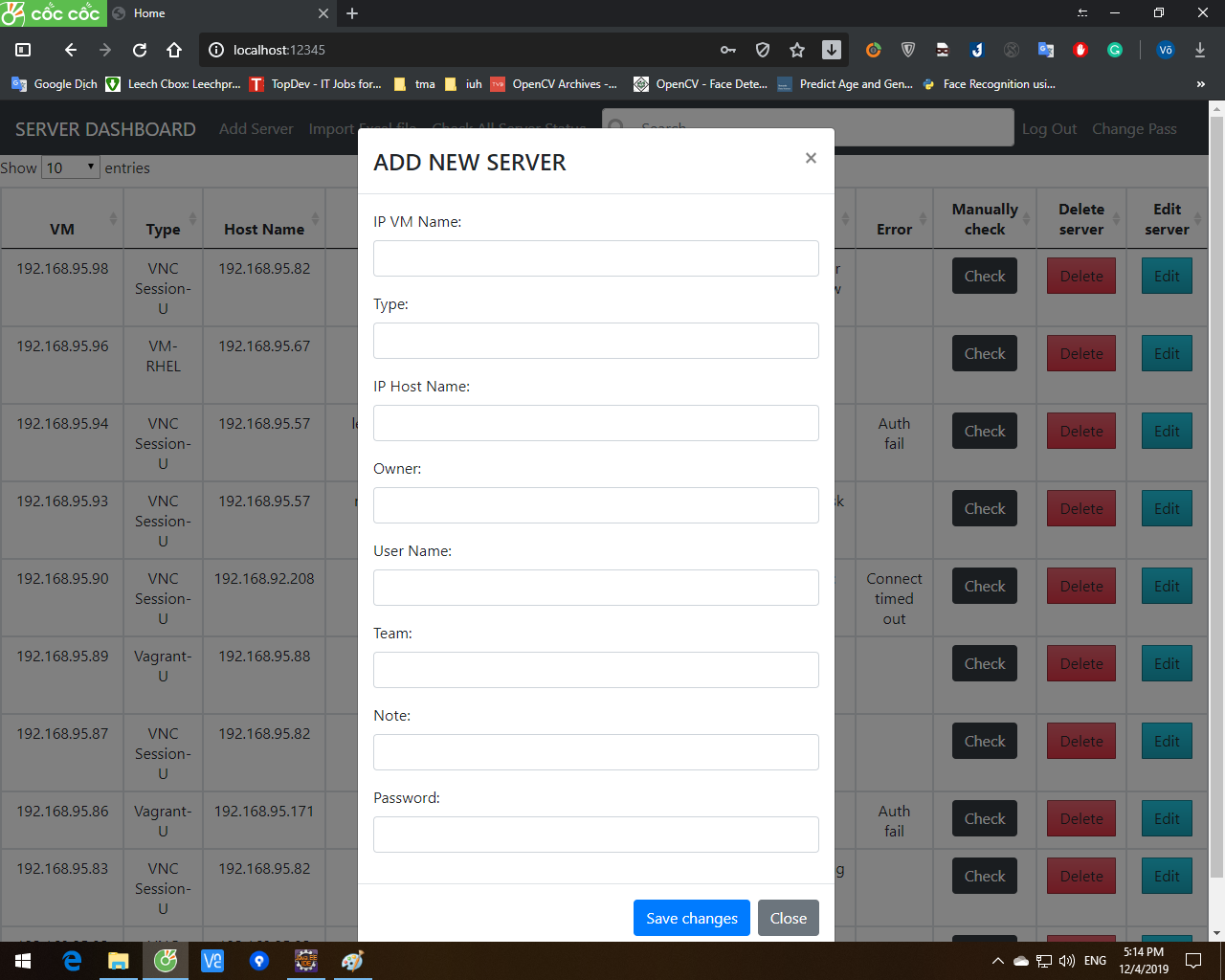
**-**  **Thông tin của các server:**

****

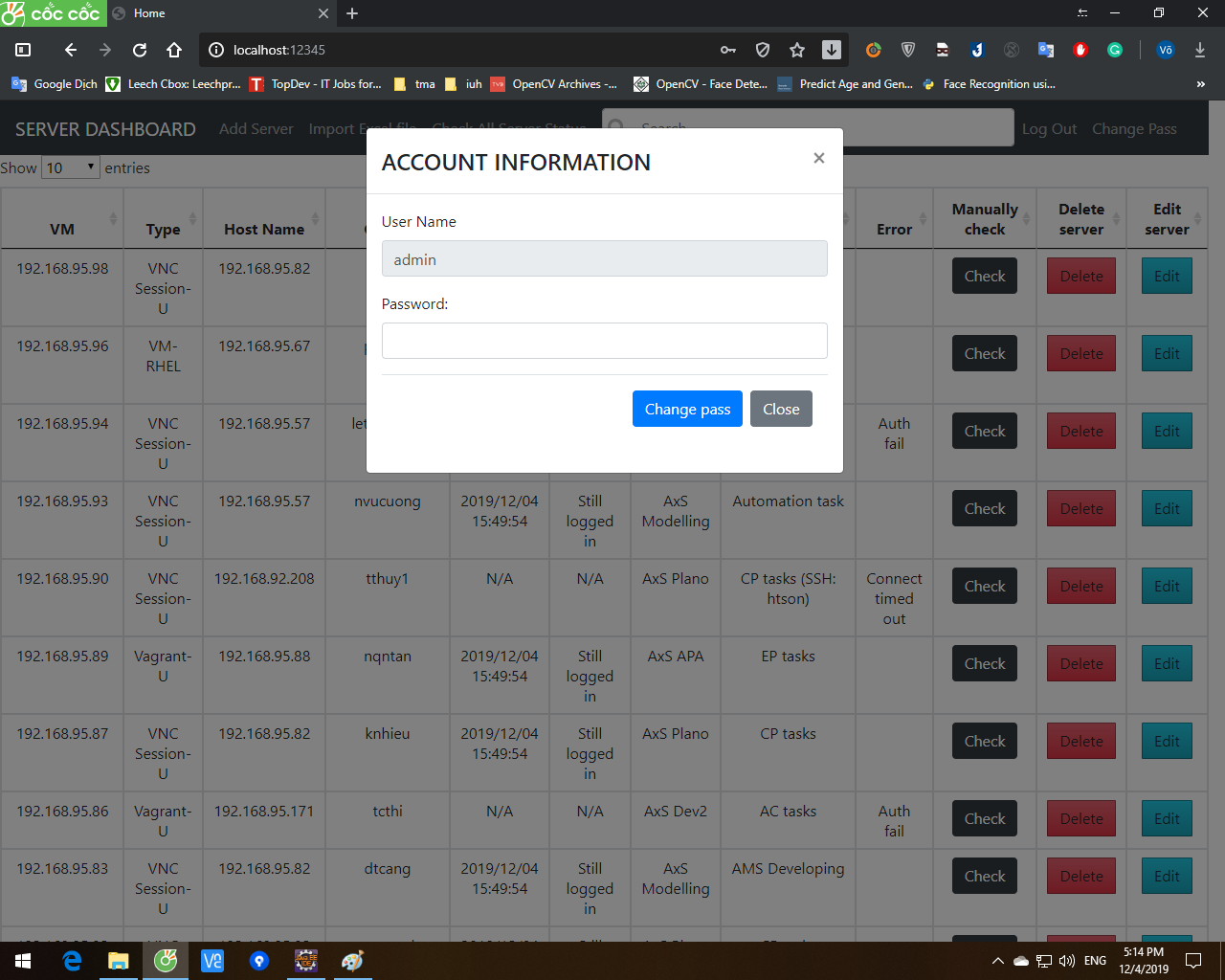
**-**  **Chọn file Excel:**

****

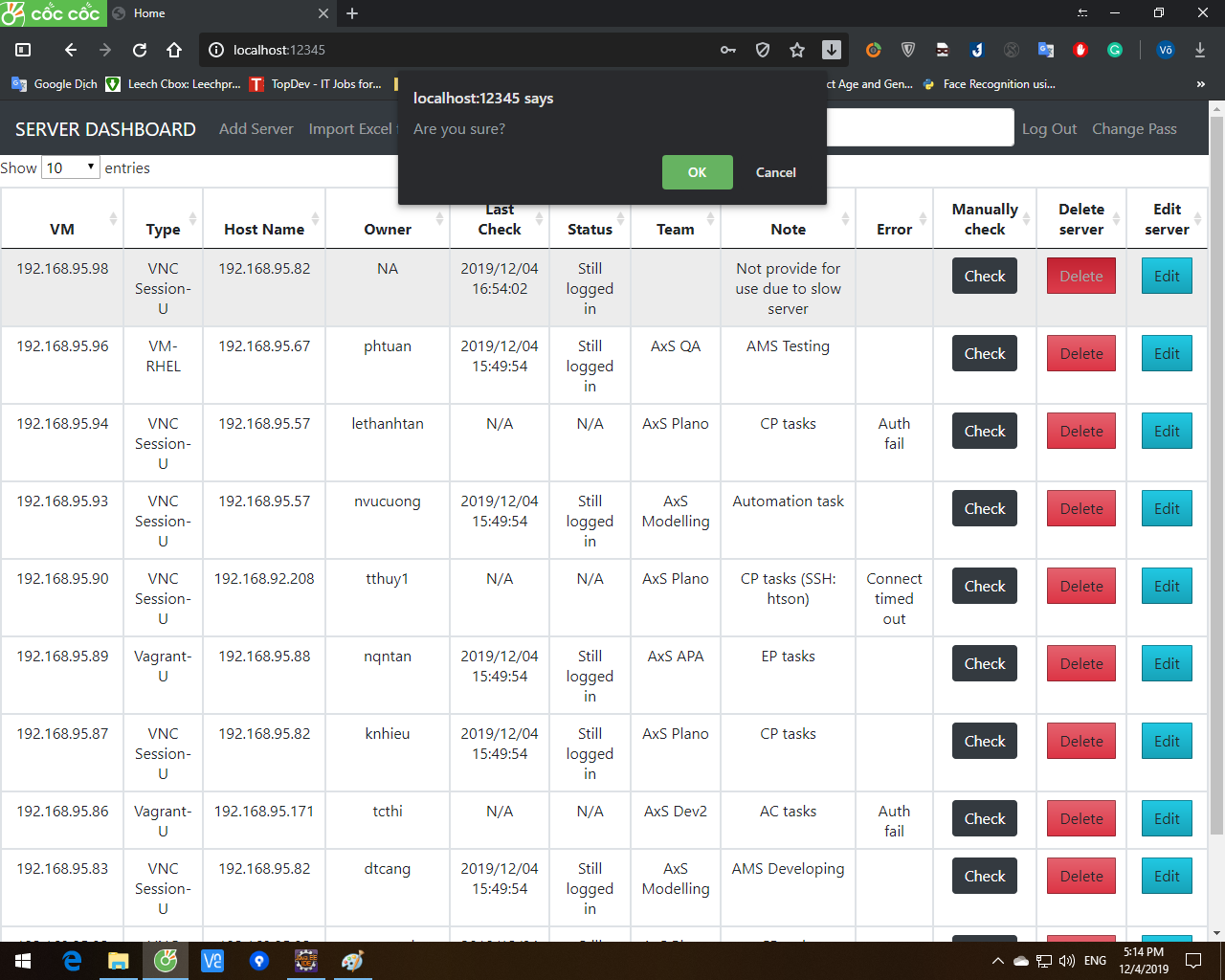
**-**  **Thêm một server:**

****

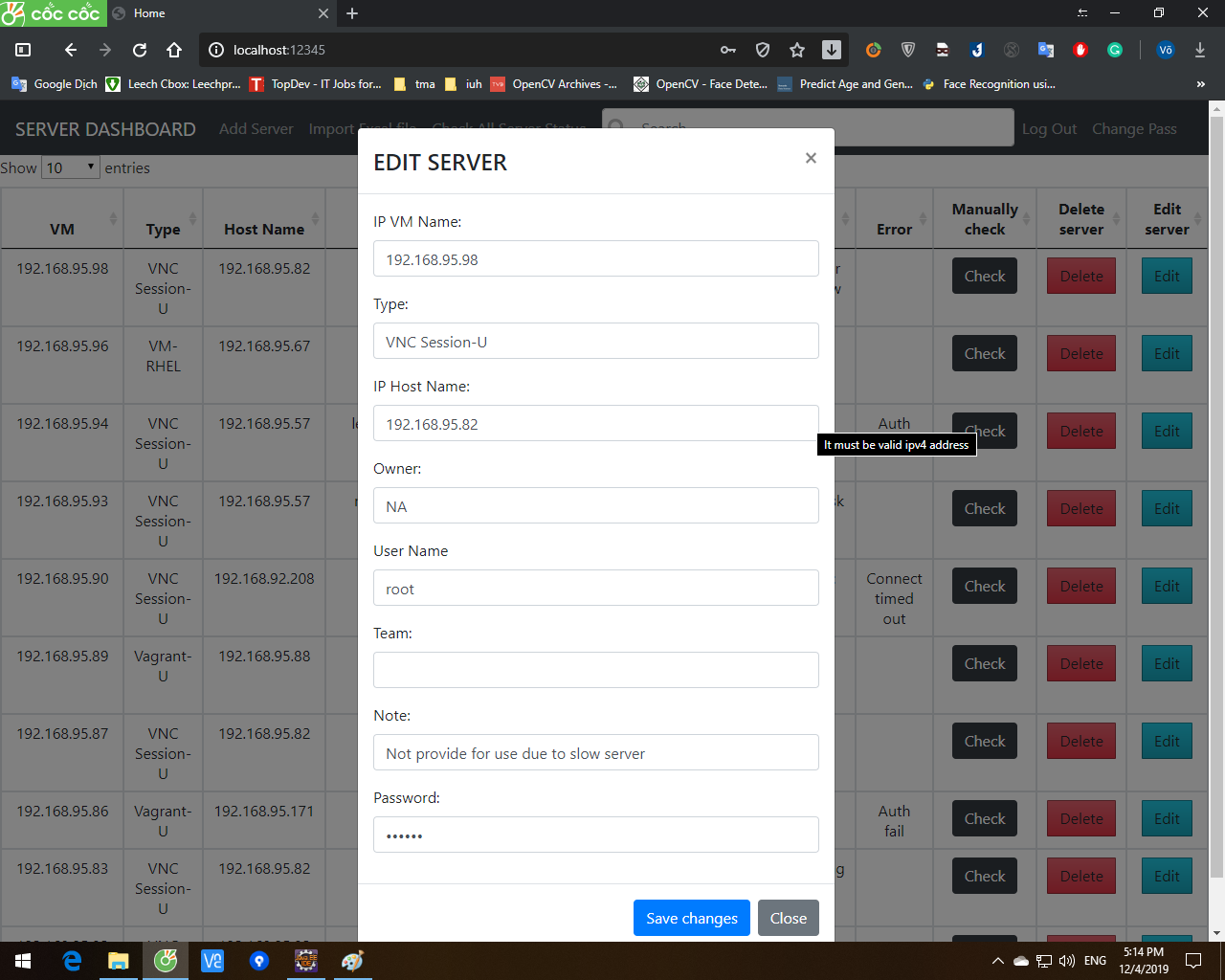
**-**  **Đổi password:**

****

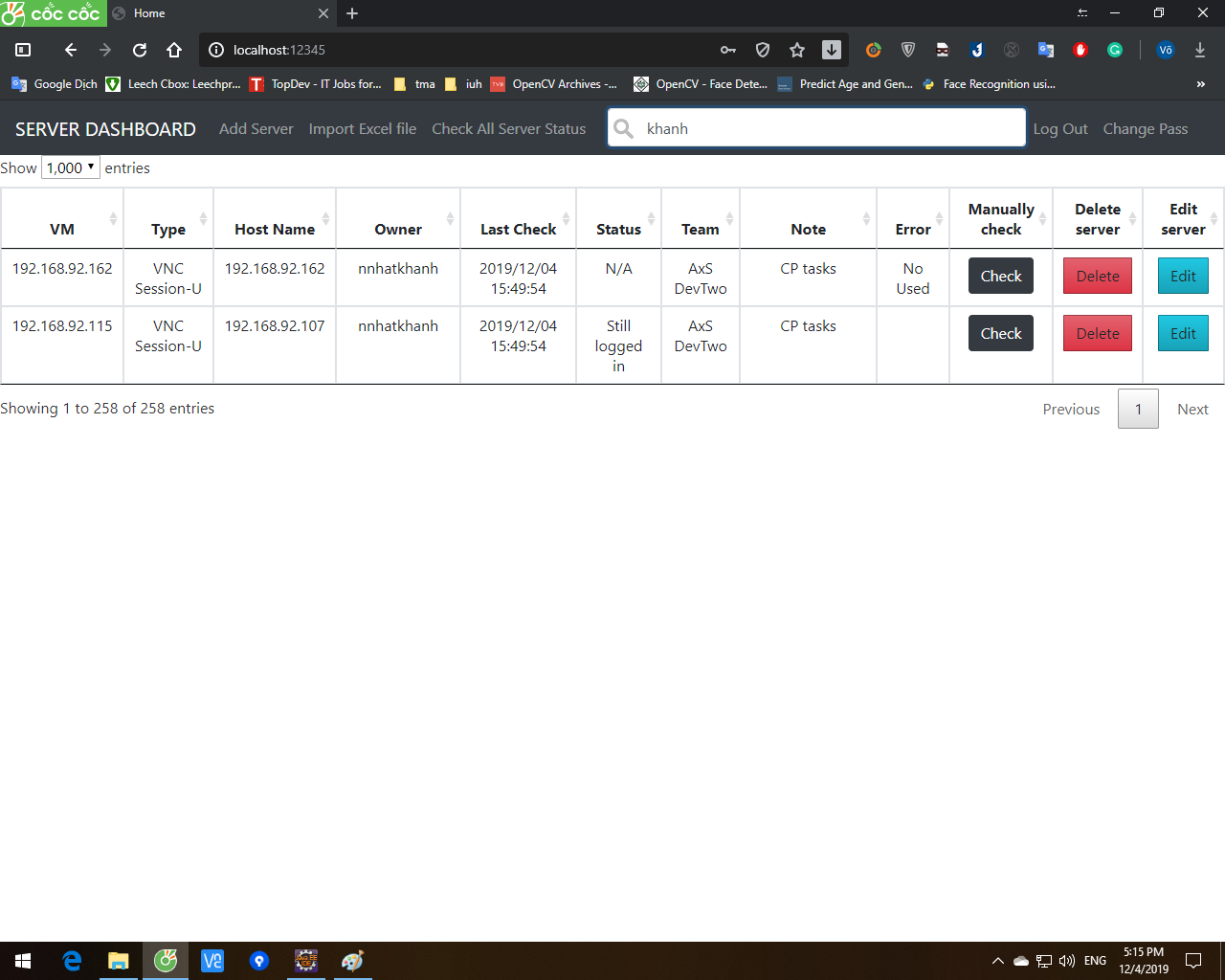
**-**  **Xóa server:**

****

**-**  **Sửa server:**

****

**-**  **Tìm server:**

****