**Tài liệu hướng dẫn sử dụng Java Backend Template - v1.0.0**

*SD2 R&D Elcom 2020*

**Tổng quan**: SpringBoot version 2.2.7

**Tên project**: elcomProjectService

**Mã nguồn:** https://gitlab.elcom.com.vn/xxxxxxxxxxxxx

**Nội dung tài liệu:**

1. Project configuration
2. Authenticate
3. Restful Webservices & Hibernate
4. Redis cache
5. L2-cache/Eh-Cache
6. Schedulers
7. Rabbit-MQ
8. Typesafe configuration
9. Swagger

**1. Project configuration**

*1.1.Cấu hình:*File: /config/application.properties

**-** Chứa các cấu hình của spring-boot project: redis connection, spring-upload, database connection, thông số Jpa & Hibernate, Rabbit-MQ connection, server-port, cấu hình SSL, thư mục file logging, thư mục file cấu hình ehCache.

*1.2. Library version:*

- spring-boot-2.2.7

- config-1.4.0

- mysql-connector-java-8.0.20

- hibernate-core-5.4.15

- ehcache-core-2.5.2

- ehcache-jgroupsreplication-1.7

- jgroups-3.6.6

- amqp-client-5.7.3

- spring-multirabbit-lib-2.2.0

- jedis-3.3.0

- spring-data-redis-2.2.7

- quartz-2.3.2

- tomcat-embed-core-9.0.34

**2. Authenticate** *(Xác thực người dùng)*

- Xác định người dùng có hợp lệ (Authenticate) hay không và người dùng có quyền truy cập những tài nguyên gì(Authorization) sau khi được xác thực thành công.

**- pom.xml:**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

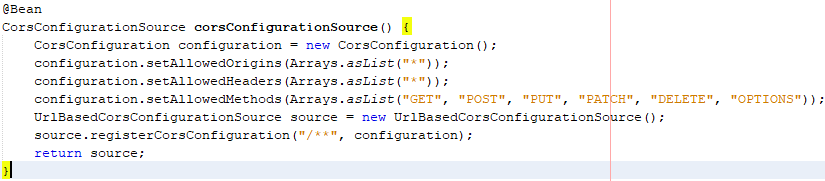
<version>0.9.1</version>

</dependency>

***2.1. Routing các api*:** **com.elcom.project.config.SecurityConfig.java**



***2.2. Giới hạn các kết nối từ client:* com.elcom.project.config.SecurityConfig.java**



***2.3. Xác thực một người dùng : đăng nhập :***

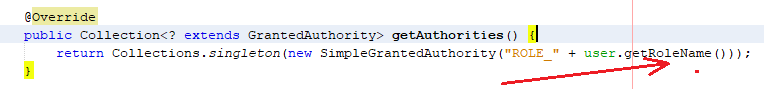
**com.elcom.project.controller.AuthController.java**



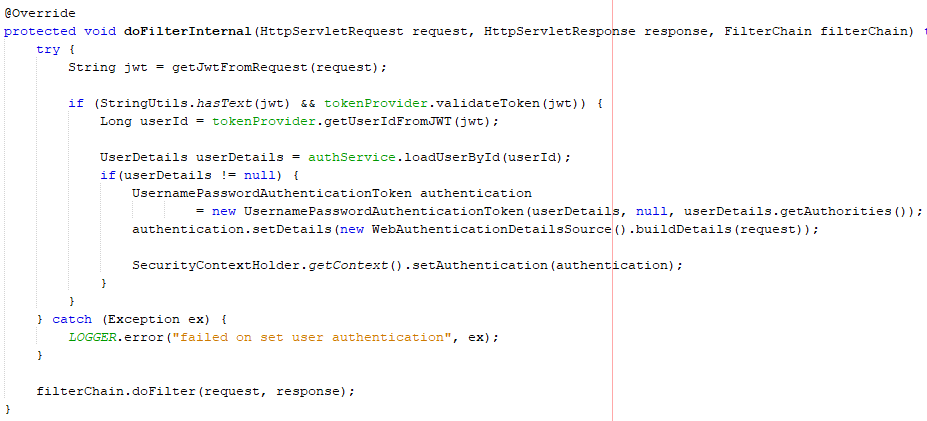
**com.elcom.project.auth.CustomUserDetails.java :** set quyền

- “user.getRoleName()” : là giá trị trong DB cột role\_name

- Mặc định xử lý SpringBoot Authen để prefix “ROLE\_”



***2.4. Xác thực một người dùng : các request truy cập:* com.elcom.project.auth.jwt.JwtAuthenticationFilter.java**



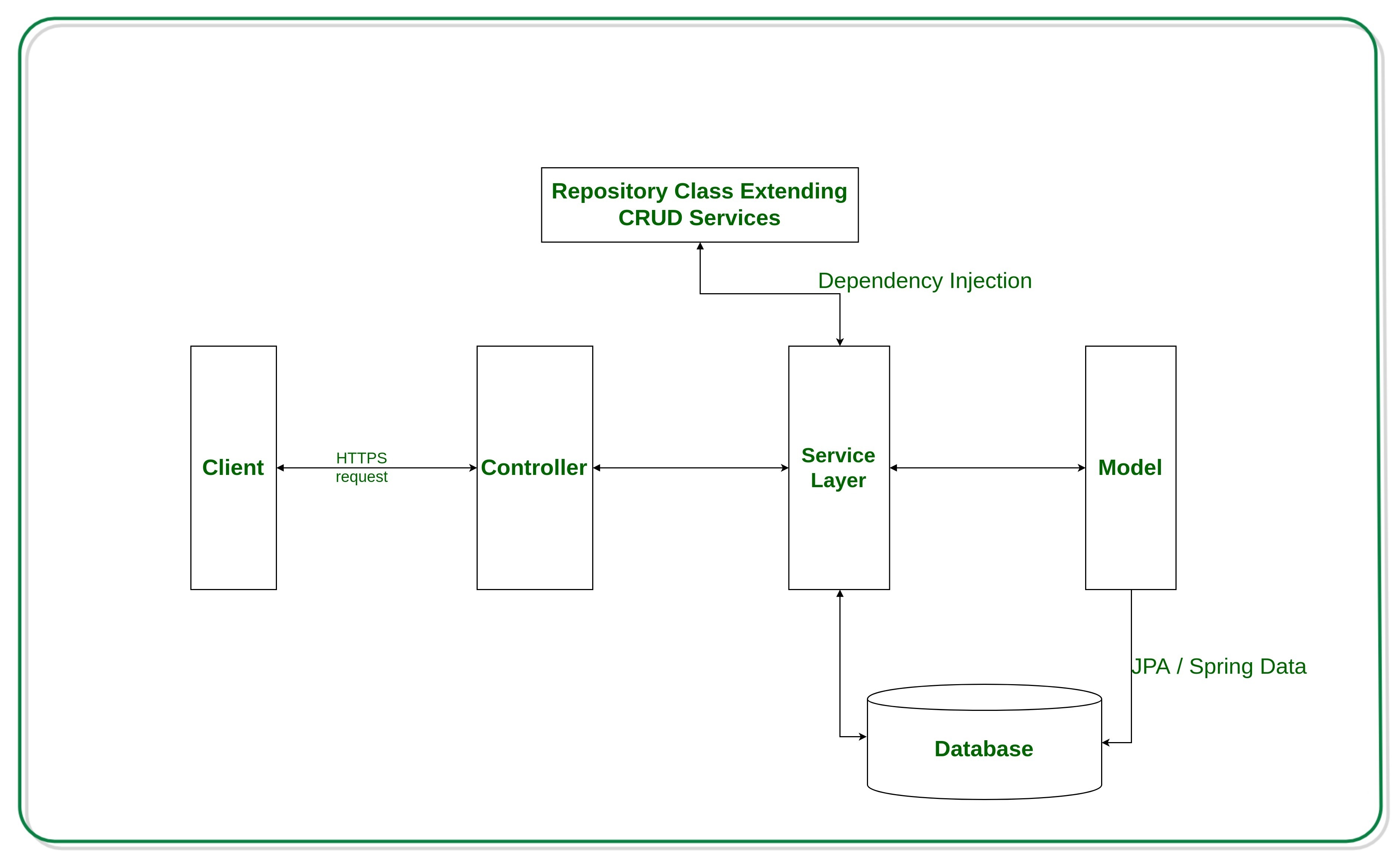
**3. Restful Webservices & Hibernate:**

*- Định nghĩa các quy tắc kiến trúc để thiết kế Web-services qua giao thức HTTP.*

***3.1. Quy tắc thiết kế các API:***

* [GET] /user Lấy danh sách user
* [GET] /user/{id} Lấy chi tiết 1 user cụ thể
* [POST] /user Tạo mới user
* [PUT] /user/{id} Sửa 1 user cụ thể
* [PATCH] /user/{id} Update 1 vài field của 1 user cụ thể
* [DELETE] /user/{id} Xóa user

***3.2. Flow:***



***3.3. Source code:***

**- Pom.xml:**

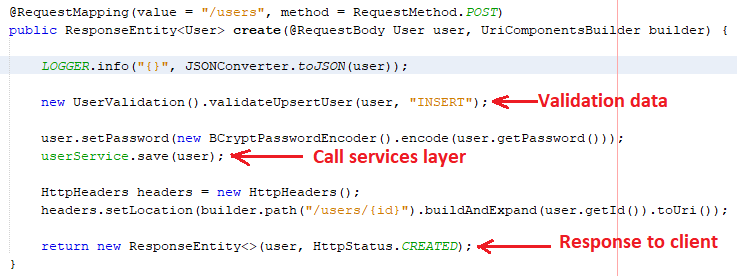
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

*- Tầng Controller:* **com.elcom.project.controller.UserManagerController.java**



*- Tầng Service:* **com.elcom.project.service.impl.UserServiceImpl.java** (implements interface UserService)

*@Override*

public void **save**(User user) {

userRepository.save(user);

}

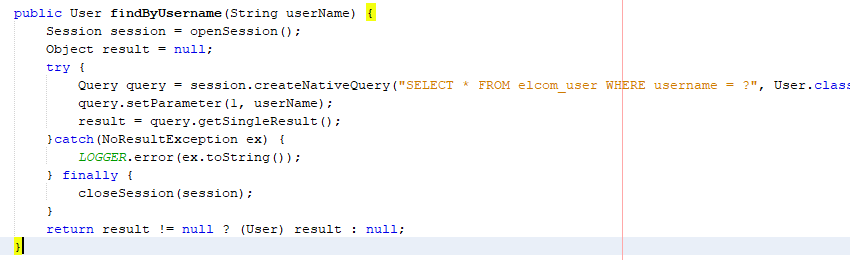
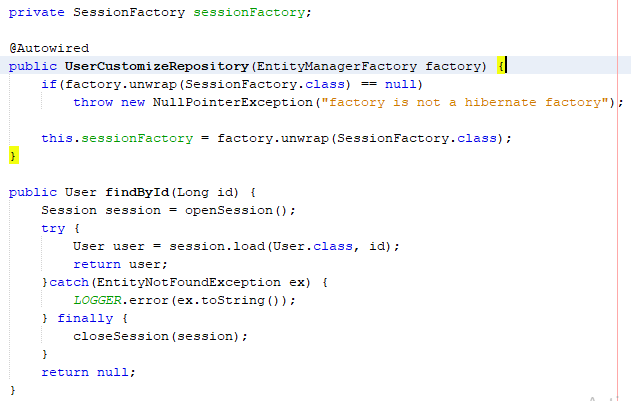
*- Tầng Repository:* **com.elcom.project.** **repository.UserRepository.java**

*@Repository*

public interface **UserRepository** extends PagingAndSortingRepository<User, Long> {

}

*- Tùy biến lại repository:* **com.elcom.project.repository.UserCustomizeRepository.java**



**4. Redis*:*** *(cache)*

*-* Redis được dùng để lưu trữ dữ liệu có cấu trúc, có thể sử dụng như một Database, bộ nhớ Cache hoặc một message-broker, là 1 hệ thống lưu trữ key-value in-memory ổn định và phổ biến.

**- *Cấu hình redis:* com.elcom.project.redis.RedisConfig.java**

**- pom.xml**

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

</dependency>

<dependency>

<groupId>io.lettuce</groupId>

<artifactId>lettuce-core</artifactId>

<version>5.1.3.RELEASE</version>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>3.3.0</version>

<type>jar</type>

</dependency>

**- *Sử dụng:* com.elcom.project.controller.UserManagerController.java**

*@Autowired*

private RedisTemplate redisTemplate;



**5. L2-Cache/Eh-Cache** *(Second level cache & clustered cache)*

***5.1 Tổng quan:***

- Mặc định khi sử dụng hibernate là cơ chế First Level Cache (L1-cache), phạm vi data được cache lại chỉ nằm trong Session/phiên kết nối xử lý đó.

- Khi active L2-cache thì phạm vi data được cache sẽ nằm ở mức cao hơn: thuộc SessionFactory quản lý, ví dụ khi app chạy, sẽ có nhiều session kết nối tới để lấy data của bảng user về thì data chỉ được lấy ở DB từ session đầu tiên, các session sau sẽ được lấy từ SessionFactory chứ không lấy trực tiếp từ DB, các cơ chế update, delete tác động tới entity đó sẽ được hành xử riêng để đảm bảo tính toàn vẹn dữ liệu.

- L2-cache **chỉ hoạt động khi lấy đối tượng theo ID** (session.load(), session.get(), session.find())

- L2-cache cần 1 CacheProvider(nơi lưu trữ data được cache, sửa/xóa/đồng bộ lại cache data khi bản ghi thay đổi) để hoạt động, dự án sử dụng EhCache làm CacheProvider cho Hibernate L2-cache.

- Áp dụng cơ chế Cluster Cache cho L2-cache hibernate, sử dụng giao thức UDP multicast hoặc giao thức TCP để đồng bộ cache data giữa các node.

***5.2 Cấu hình:***

**- pom.xml:**

<!-- L2 cache -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-cache</artifactId>

</dependency>

<!-- EhCache -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-ehcache</artifactId>

<exclusions>

<exclusion>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- Cluster ehCache using Jgroups -->

<dependency>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache-jgroupsreplication</artifactId>

<version>1.7</version>

</dependency>

<dependency>

<groupId>org.jgroups</groupId>

<artifactId>jgroups</artifactId>

<version>3.6.6.Final</version>

</dependency>

- Active L2cache & cache provider: **File: /config/application.properties**

spring.jpa.properties.hibernate.cache.use\_second\_level\_cache=true

spring.jpa.properties.hibernate.cache.region.factory\_class=org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory

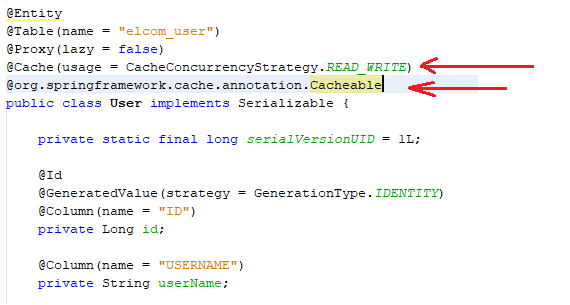
**- com.elcom.project.ElcomServiceApplication.java**



- Cấu hình các thông số cho EhCache**: File: /config/ehcache.xml**

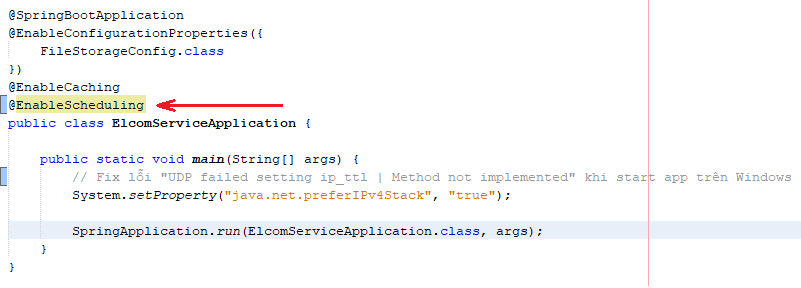
***5.3 Sử dụng:***

**-** Chỉ định cache cho Entity cụ thể : **com.elcom.project.model.** **User.java**



**6. Schedulers**

***6.1. Cấu hình:*****com.elcom.project.ElcomServiceApplication.java**



**- pom.xml:**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-quartz</artifactId>

</dependency>

***6.2 Sử dụng:*****com.elcom.project.schedulers.Schedulers.java**



***7.* Rabbit-MQ**

***7.1. Cài đặt:***

***7.1.1. Cài đặt Rabbit MQ***

Rabbit MQ sử dụng nền tảng Erlang để chạy nên cần cài đặt Erlang trước khi cài đặt RabbitMQ server

+ Windows :

Link hướng dẫn : <https://www.onlinetutorialspoint.com/windows/how-to-install-rabbitmq-on-windows-10.html>

* Khởi động Rabbit MQ Service :

cd C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.8.3\sbin

rabbitmqctl.bat stop (dừng service)

rabbitmqctl.bat start\_app (bật service)

rabbitmq-plugins enable rabbitmq\_management (cài đặt plugin để quản lý trên web)

* Tài khoản đăng nhập mặc định ở localhost là guest / guest ( sau khi đăng nhập có thể tạo them admin) hoặc sử dụng lệnh để tạo admin

rabbitmqctl add\_user admin [adminpassword]

rabbitmqctl set\_user\_tags admin administrator

rabbitmqctl set\_permissions -p / admin ".\*" ".\*" ".\*"

+ Linux :

Link hướng dẫn : <https://123host.vn/tailieu/kb/vps/huong-dan-cai-dat-rabbitmq-tren-centos-7.html>

* Khởi động Rabbit MQ Service :

systemctl start rabbitmq-server.service (bật)

systemctl enable rabbitmq-server.service (tắt)

rabbitmq-plugins enable rabbitmq\_management (cài đặt plugin để quản lý trên web)

* Tài khoản đăng nhập :

rabbitmqctl add\_user admin [adminpassword]

rabbitmqctl set\_user\_tags admin administrator

rabbitmqctl set\_permissions -p / admin ".\*" ".\*" ".\*"

+ Link quản trị : <http://[ip-address]:15672/>

***7.1.2. Cài đặt cluster Rabbit MQ:***

Điều kiện để thiết lập clustering:

* Tất cả các node phải cùng erlang version và rabbitmq version
* Các node liên kết qua LAN network
* Tất cả các node chia sẻ cùng một erlang cookie

+ Trong mô hình cluster, các node sử dụng phương thức trao đổi giữa của erlang. Khi sử dụng phương thức này, hai erlang node chỉ nói chuyện được với nhau khi có cùng erlang cookie. Erlang cookie chỉ là một chuỗi ký tự. Khi startup một rabbitmq server lần đầu tiên, mặc định một erlang cookie ngẫu nhiên được sinh ra nằm trong /var/lib/rabbitmq/.erlang.cookie (Linux) và C:/Windows/System32/config/systemprofile/.erlang.cookie (Windows)

+ Cài đặt xong RabbitMQ, stop hoặc khoan start RabbitMQ, copy cokkie từ node chính qua node phụ. Vì node name của rabbitmq có dạng rabbit@hostname nên bạn cần chuẩn bị sẵn hostname cho mỗi node

+ Đặt hotsname, chỉnh resolve và reboot server. Sau đó đưa vào /etc/hosts (Linux) hoặc C:\Windows\System32\drivers\etc\hosts (Windows) của cả ba node:

192.168.3.241 hostname1 ### gọi là node 1

192.168.3.242 hostname2 ### gọi là node 2

192.168.3.252 hostname3 ### gọi là node 3

+ Khởi động rabbitmq-server trên mỗi node:

service rabbitmq-server start

+ On app trên rabbit 1 trước và check status cluster :

rabbitmqctl start\_app

rabbitmqctl cluster\_status

+ Node phụ thứ nhất :

rabbitmqctl stop\_app

rabbitmqctl join\_cluster rabbit@hostname1

rabbitmqctl start\_app

Check lại cluster status : rabbitmqctl cluster\_status

+ Node phụ thứ hai :

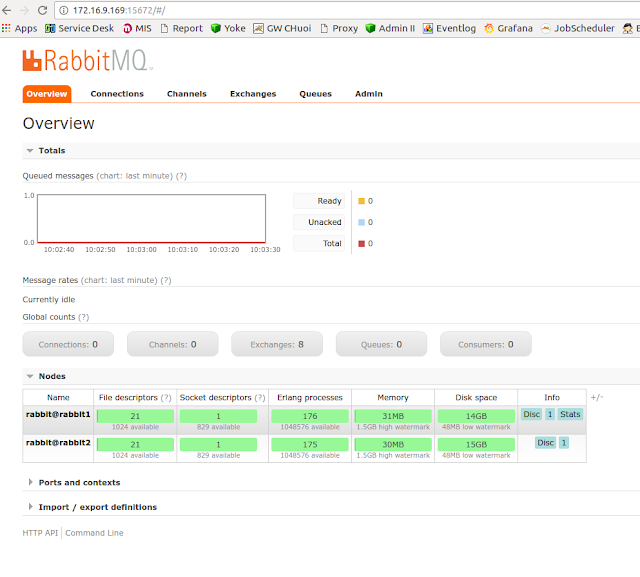
rabbitmqctl stop\_app

rabbitmqctl join\_cluster rabbit@hostname2

rabbitmqctl start\_app

Check lại cluster status : rabbitmqctl cluster\_status

Cài đặt cluster ok, ở màn hình overview quản lý trên web của mỗi node sẽ có d/s các node trong cluster



***7.1.3. Cài đặt HA mode cho các node:***

Link tham khảo : <https://tech.bizflycloud.vn/highly-available-queues-va-clustering-cho-rabbitmq-439.htm>

<https://www.rabbitmq.com/ha.html>

* Mặc định các queue trong RabbitMQ cluster chỉ nằm trên một node duy nhất mà chúng được khai báo trước tiên. Không giống như exchange và binding luôn luôn nằm trên mọi node. Tuy nhiên, queue cũng có thể cấu hình để mirror trên nhiều node. Mỗi "mirrored queue" gồm 1 master và nhiều slave, và slave đầu tiên lên làm master mới nếu master cũ bị mất.
* Message được publish tới queue sẽ được replicate đến tất cả các slave. Slave sẽ drop message mà nó nhận được. Do đó Queue mirroring chỉ nâng cao tính sẵn sàng chứ không phân bố tải trên các node.

**+ Cấu hình Mirroring**

* Cấu hình Queue mirroring được thực hiện thông qua policy. Policie có thể thay đổi mọi thời điểm, đồng nghĩa với việc có thể tạo một non-mirrored queue sau đó biến chúng thành mirrored queue hoặc ngược lại.
* Policy match những queue với tên bắt đầu bằng "ha." sẽ mirror sang tất các node trong cluster:

rabbitmqctl set\_policy ha-all "^ha\." '{"ha-mode":"all"}'

* Policy match những queue với tên bắt đầu bằng "two." sẽ mirror sang 2 node bất kỳ trong cluster với option automatic synchronization:

rabbitmqctl set\_policy ha-two "^two\." '{"ha-mode":"exactly","ha-params":2,"ha-sync-mode":"automatic"}'

* Policy match những queue với tên bắt đầu bằng "nodes." sẽ mirror các node được chỉ định cụ thể trong cluster:

rabbitmqctl set\_policy ha-nodes "^nodes\." '{"ha-mode":"nodes","ha-params":["nodeA", "nodeB"]}'

***7.2. Sử dụng:***

***7.2.1. Ngữ cảnh sử dụng:***

*-* Rabbit MQ là một ứng dụng quản lý hàng đợi, trung gian giữa việc nhận và gửi tin nhắn

- Phù hợp trong các ngữ cảnh

- Webserver cần response lại nhanh cho người dùng : Webserver sẽ tạm trả về kết quả ok, đẩy message vào rabbitmq, rabbitmq sẽ lần lượt đẩy về các worker xử lý ở background job (ví dụ các bài toán về upload file, gửi mail, …)

- Trong các bài toán cần sử dụng queue (như trong xử lý tin nhắn sms), thay vì sử dụng queue tự viết trên memory, sử dụng rabbit mq là 1 service riêng tránh bị mất dữ liệu khi chương trình bị khởi động hoặc treo…

- Trong các bài toán về multi thread, các công việc cần xử lý sẽ được queue lại rồi đẩy về lần lượt các consumer xử lý

- Trong các mô hình microservices.

***7.2.2. Sử dụng Rabbit-MQ:***

* Khai báo pom :

<!--Rabbit MQ-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.amqp</groupId>

<artifactId>spring-rabbit-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.amqp</groupId>

<artifactId>spring-amqp</artifactId>

<version>2.2.6.RELEASE</version>

</dependency>

* Khai báo cấu hình RabbitMQ Server trong application.properties:
  + Khai báo dùng 1 node :

spring.rabbitmq.host=103.21.151.185

spring.rabbitmq.port=5672

spring.rabbitmq.password=test

spring.rabbitmq.username=test

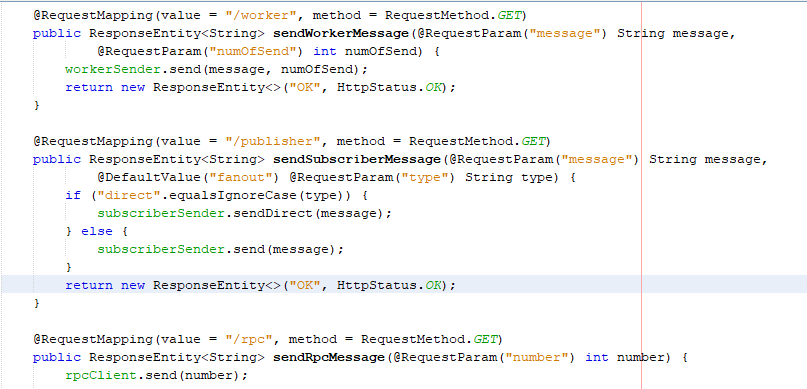
* + Khai báo dùng cluster nhiều node

spring.rabbitmq.password=test

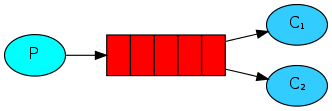
spring.rabbitmq.username=test

spring.rabbitmq.address=DESKTOP-43RI0J4:5672,DESKTOP-NO9CT29:5672

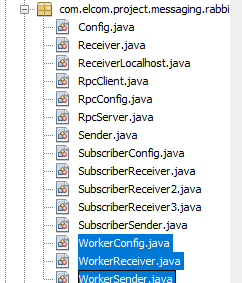
* + Lưu ý : Khi sử dụng cluster cần cài đặt HA mode
  + Các node hoặc cluster cài đặt plugin quản lý trên web sẽ có link quản lý tương ứng trên web (http:// <IP or host>:15672/)
* Sử dụng RabbitmqController để tạo các path test các case tương ứng : /rabbitmq/worker 🡪 test worker, /rabbitmq/publisher 🡪 test pub/sub theo cơ chế broadcast hoặc router, /rabbitmq/rpc 🡪 test rpc



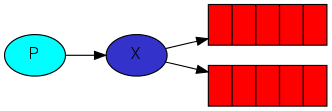
* Sử dụng Rabbit MQ Worker
  + RabbitMQ Worker dùng trong các bài toán cần để phản hồi nhanh người dùng trên web giúp trong suốt trải nghiệm người dùng hoặc các background job tốn thời gian xử lý 🡺 chia cho các worker xử lý đồng thời



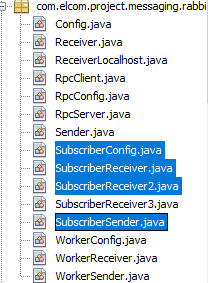
* + Các file trong project

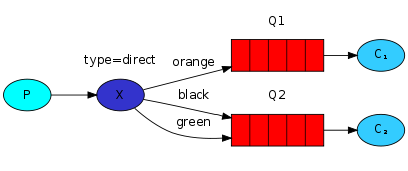


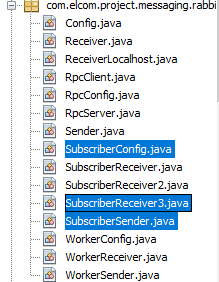
* + File WorkerConfig.java: Cấu hình khai báo tên queue\_name, Sender gửi tin nhắn và các Worker nhận tin nhắn
  + File WorkerSender.java : Khai báo 2 đối tượng queue và rabbitTemplate để gửi tin nhắn lên queue, method send() để xử lý gửi tin nhắn lên queue
    - Trong WorkerSender có khai báo thêm cách gửi tin nhắn theo dạng Persistence Mode (Mặc định của spring amqp) hoặc Non Persistence Mode 🡺 Có comment cách dùng
    - Persistence Mode : Tin nhắn trong queue sẽ được giữ nguyên ngay cả khi reset rabbit và ngược lại non – persistence sẽ hủy hết tin nhắn nếu reset rabbit
  + File WorkerReceive.java : Khai báo anotation @RabbitListener(queues = "xxx") để nhận các tin nhắn từ queue xxx trả về. Trong class đó có method receiver khai báo anotation @RabbitHandler để thực hiện việc xử lý tin nhắn
* Sử dụng RabbitMQ Pub/Sub
  + **Broadcast all** : Fanout exchange - Sử dụng trong các bài toán broadcast all tin nhắn tới các subscriber



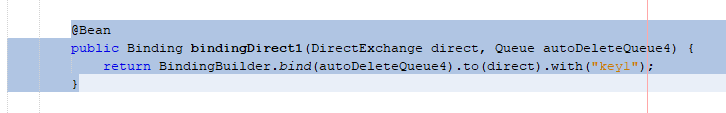
* + Sử dụng 3 file : SubsciberConfig, SubsciberSender, SubscriberReceiver, SubscriberReceiver2



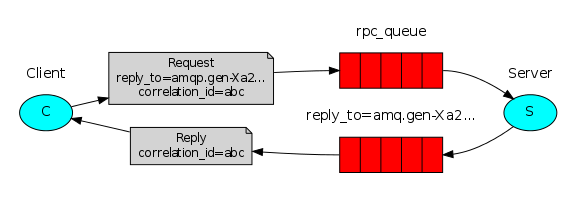
* + - SubsciberConfig: Khai báo 1 đối tượng FanoutExchange để gửi tin lên exchange, đồng thời bind exchange đó với các queue tự sinh
    - SubscriberSender: Khai báo đối tượng FanoutExchange được khởi tạo bên config, và đối tương RabbitTemplate để gửi tin
      * Hàm send để gửi qua fanout exchange
    - SubscriberReceiver và SubscriberReceiver2 : 2 subsciber nhận tin thông qua khai báo @RabbitListener(queues = "#{autoDeleteQueue1.name}"), @RabbitListener(queues = "#{autoDeleteQueue2.name}")…
  + **Router** : DirectExchange – gửi và nhận tin broadcast được filter theo key
  + Sử dụng 3 file : SubsciberConfig, SubsciberSender, SubscriberReceiver3



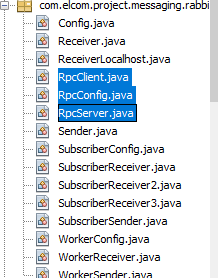
* + - SubsciberConfig: Khai báo 1 đối tượng DirectExchange để gửi tin lên exchange, đồng thời bind exchange đó với các queue tự sinh, kèm theo key (“key1”) để chỉ những subsciber nhận khai báo “key1” mới nhận được



* + - SubscriberSender: Khai báo đối tượng DirectExchange được khởi tạo bên config, và đối tương DirectExchange để gửi tin
      * Hàm sendDirect để gửi qua direct exchange
    - SubscriberReceiver3 : Subsciber nhận tin thông qua khai báo @RabbitListener(queues = "#{autoDeleteQueue4.name}")
* Sử dụng RabbitMQ RPC
  + Dùng trong các bài toán : bên gửi tin muốn nhận kết quả trả về thay vì sử dụng kiểu dạng rest api, phải trả về ngay lập tức, rabbit RPC sẽ đẩy các request vào queue và xử lý dần dần rồi trả lại cho bên client



* + Sử dụng 3 file : RpcConfig, RpcClient, RpcServer



* + RpcConfig :
    - Khai báo 1 DirectExchange client với tên “rpc\_exchange” cho client gửi request lên
    - Khai báo 1 DirectExchange server với tên “rpc\_exchange”, 1 queue name “rpc\_requests”, sau đó bind exchange và queue với key “rpc”
  + RpcClient :
    - Client gửi tin qua đối tượng RabbitTemplate, sử dụng method convertSendAndReceive() lên exchange “rpc\_exchange” thông qua key “rpc”, đồng thời nhận kết quả trả về
  + RpcServer :

Khai báo annotation @RabbitListener(queues = "rpc\_requests"), để nhận các yêu cầu qua queue “rpc\_requests”

**8. Typesafe configuration**

* Thư viện Typesafe config để khai báo các hằng số sử dụng bên ngoài spring boot như các trường hợp (URL payment bên thứ 3, các hằng số đối tác cấp hoặc các hằng số trong chương trình chạy…)
* Khai báo pom :

<!-- Config load external resources -->

<dependency>

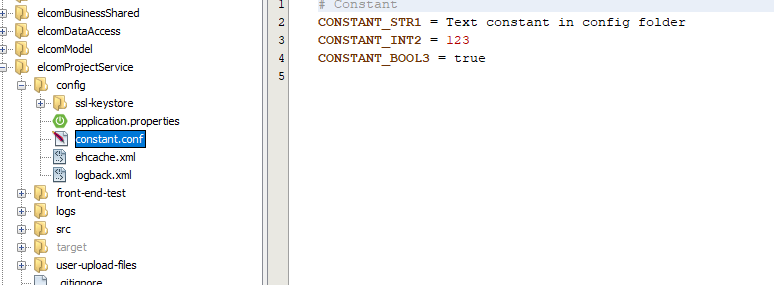
<groupId>com.typesafe</groupId>

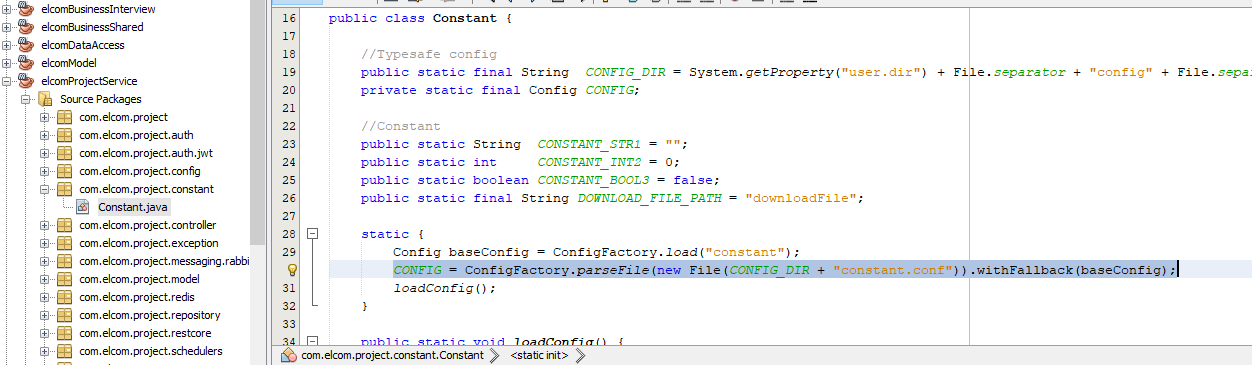
<artifactId>config</artifactId>

<version>1.4.0</version>

</dependency>

* File cấu hình trong thư mục config/constant.conf





* Sử dụng trong project : Đối tượng Config (thư viện TypeSafe) sẽ đọc file cấu hình được khai báo ở config/constant.conf
  + Khai báo các biến static tương ứng trong file constant và load ra
  + Các mục của chương trình sẽ sử dụng các biến static này

Khi cần sửa lại chỉ cần sửa file cấu hình và restart lại chương trình.

**9. Swagger**

- Thông thường, tất cả các API sau khi được đưa lên sẽ phải đi kèm với document mô tả, để bất kì ai sử dụng đến thì có thể tra cứu, **Swagger** giúp mô tả tài liệu dự án một cách nhanh chóng bằng các annotation.

***Cấu hình:***

**- pom.xml:**

*<dependency>*

*<groupId>io.springfox</groupId>*

*<artifactId>springfox-swagger2</artifactId>*

*<version>2.7.0</version>*

*</dependency>*

*<dependency>*

*<groupId>io.springfox</groupId>*

*<artifactId>springfox-swagger-ui</artifactId>*

*<version>2.7.0</version>*

*</dependency>*

*<!-- Repository cho Swagger -->*

*<repositories>*

*<repository>*

*<id>jcenter-snapshots</id>*

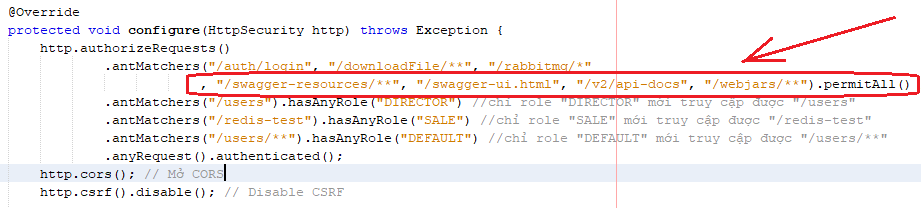
*<name>jcenter</name>*

*<url>https://jcenter.bintray.com/</url>*

*</repository>*

*</repositories>*

**com.elcom.project.config.SecurityConfig.java:**



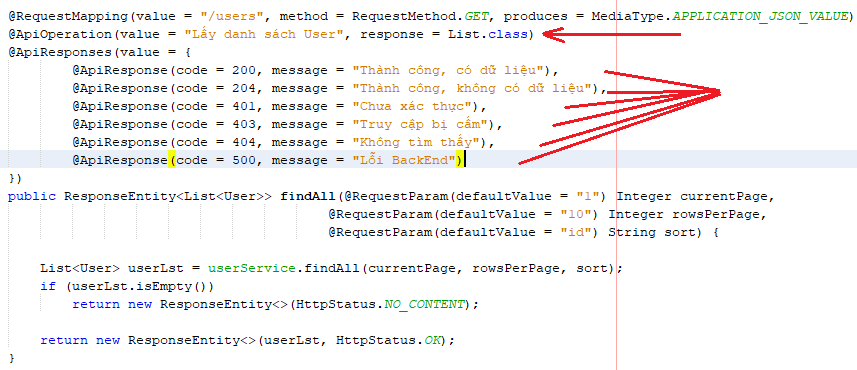
**com.elcom.project.config.SwaggerConfig.java:**



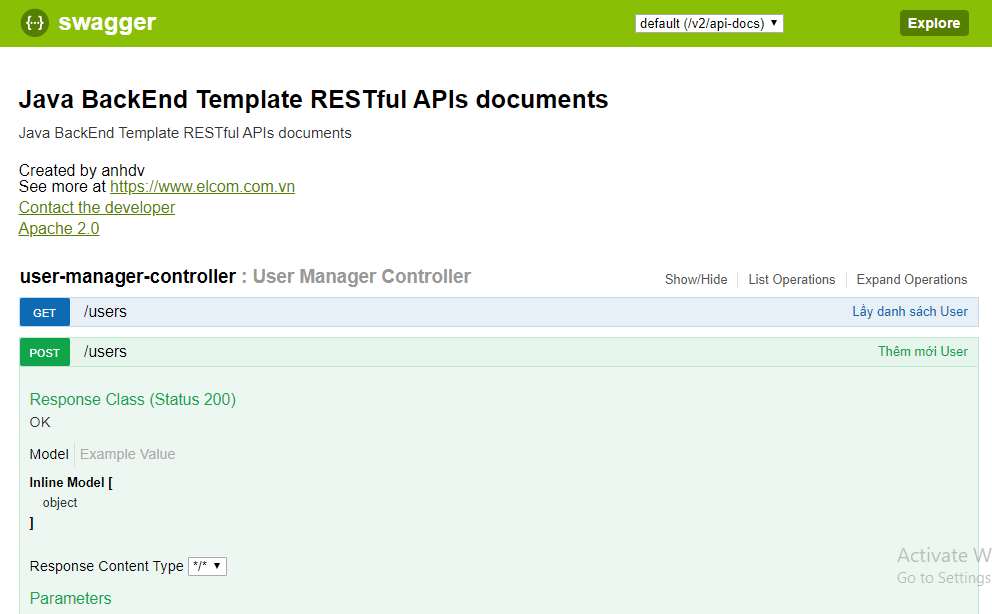
*RequestHandlerSelectors.any()*: quét để docs hết tất cả các controller trong dự án.

*PathSelectors.any()*: quét để docs hết tất cả các api\_path của dự án.

**Hiệu chỉnh document:**



**Giao diện APIs document:** *https://<HOST>:<PORT>/swagger-ui.html*



- - - - - - - - - - - - - - - - - - - - - - -