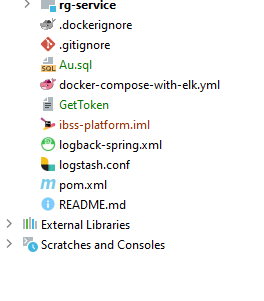
# **HƯỚNG DẪN PHÁT TRIỂN**

**File Sql tạo DB cho authen service:**

**File: Au.sql**



**Get token:**

curl --location --request POST 'http://localhost:8080/ibss/uaa/oauth/token' \  
--header 'Content-Type: application/x-www-form-urlencoded' \  
--header 'Authorization: Basic bW9iaWxlOnBpbg==' \  
--data-urlencode 'grant\_type=password' \  
--data-urlencode 'username=tranngoc897' \  
--data-urlencode 'password=123456' \  
--data-urlencode 'client-id=mobile' \  
--data-urlencode 'client-secret=pin'

## Tham khảo:

[An Illustrated Guide to OAuth and OpenID Connect | Okta Developer](https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc)

https://blog.marcosbarbero.com/centralized-authorization-jwt-spring-boot2/

https://sultanov.dev/blog/custom-jwt-claims-in-spring-security-oauth/

[java - Difference between AuthorizationServerConfigurerAdapter vs WebSecurityConfigurerAdapter - Stack Overflow](https://stackoverflow.com/questions/50919268/difference-between-authorizationserverconfigureradapter-vs-websecurityconfigurer/50938657)

https://www.javacodegeeks.com/2019/01/expression-based-access-control.html

[Microservices for Java Developers: Managing Security and Secrets | Java Code Geeks - 2021](https://www.javacodegeeks.com/2019/01/microservices-for-java-developers-managing-security-and-secrets.html)

[Log aggregation with Spring Boot, Elastic Stack and Docker | cassiomolin](https://cassiomolin.com/2019/06/30/log-aggregation-with-spring-boot-elastic-stack-and-docker/)

[Spring Boot Apps with Kong API Gateway using OpenAPI](https://blog.codecentric.de/en/2020/11/spring-boot-kong/)

https://suraj-batuwana.medium.com/managing-spring-boot-micro-services-with-kong-part-1-install-kong-c652eec67a35

https://suraj-batuwana.medium.com/managing-spring-boot-micro-services-with-kong-part-2-create-springboot-api-ef8af12e6522

https://www.mitrais.com/news-updates/asynchronous-communication-between-microservices-with-apache-kafka/

https://www.javacodegeeks.com/2019/08/microservices-log-management.html

https://walkingtree.tech/logs-monitoring-microservices-using-elk

https://dzone.com/articles/the-future-of-spring-cloud-microservices-after-net

[Spring Boot Microservices and Spring Cloud | Udemy](https://www.udemy.com/course/spring-boot-microservices-and-spring-cloud/?start=0)

**Một số thư viện đặc biệt đang dùng:**

**Lombok**: <https://projectlombok.org/>

<https://huongdanjava.com/vi/gioi-thieu-ve-project-lombok>

Hướng dẫn cài plugin Lombok trên IntelliJ : <https://projectlombok.org/setup/intellij>

**Maven**: Project được tổ chức theo mô hình Maven Multi module. Ibss-platform là root module.

**common** là module được dùng chung - nơi chứa code được sử dụng nhiều lần ở nhiều module,

**Thứ tự chạy các module : cf-service (config) -> rg-servive (registry) -> gw-service(gateway) -> au-service (authentication)-> các service còn lại.**

# **PHÁT TRIỂN**

Tham khảo thêm Open-source project:

<https://github.com/vaquarkhan/PiggyMetrics-microservice-poc>

**Module cf-service:**

Là nơi chứa file config của các service.

Các file config được chia làm 2 folder:

**shared-native**: Là thư mục dùng ở local

**shared-prod**: là thư mục dùng trên cloud.

Trong mỗi thư mục có 1 file là **application.yml** . Đây là file config root của của từng thư mục.

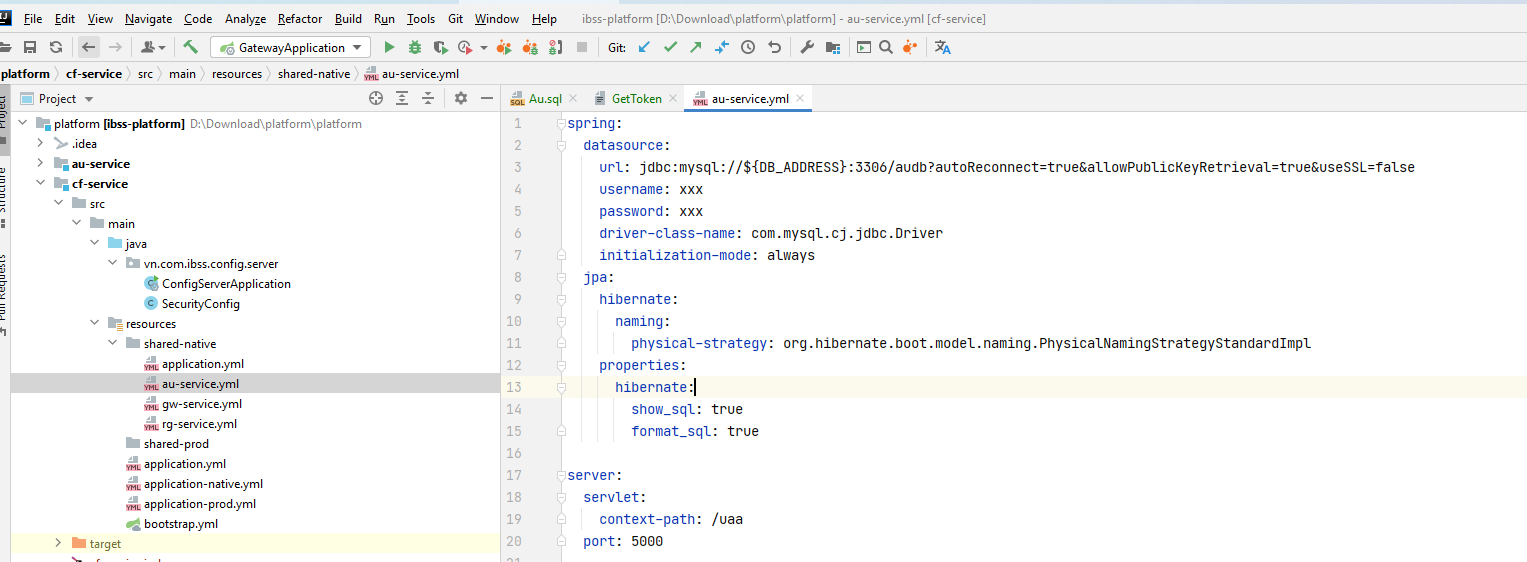
Tất cả các file config khác đều kế thừa tham số từ file này.

Nó được dùng để cấu hình những thuộc tính dùng chung cho cả folder.

Ví dụ Lấy giá trị 1 biến HOST ADDRESS trong file **application.yml**:

**Cú pháp: ${HOST\_ADDRESS}**

**Thay đổi thông tin DB au-service:**



**Cách Thêm 1 module:**

**Bước 1: Tao mới module bằng IntelliJ**

**Vào File →Project Structure → Modules → Click vào + (Add)**

**Rồi chọn New Module → Chọn Spring Initializr**

**Ví dụ tạo new Service**

**Chọn next để add các thư viện cần thiết nếu cần, rồi finish.**

**Bước tiếp theo là vào thư mục resources của new-service để tạo cac file cần thiết**

**như application.yml, bootstrap.yml**

**File application.yml**

spring:  
 profiles:  
 active: @activatedProfile@

activatedProfile lưu ở file pom.xml của root project.

**File bootstrap.yml**

spring:  
 application:  
 name: new-service  
 cloud:  
 config:  
 *uri:* [*http://localhost:8888*](http://localhost:8888/) *# uri:* [*http://172.105.118.95:8888*](http://172.105.118.95:8888/)fail-fast: true  
 user: user  
 password: password

(Xem application.yml trong cf-service )

**User và password ở là để truy cập vào cf-service**

**lưu tên service và config server để lấy thông tin config lúc khởi chạy service.**

**Mặc đinh nếu ko điền URI của config server thì sẽ là** [*http://localhost:8888*](http://localhost:8888/)

**Tiếp theo vào file pom của project new service dể chỉnh root mudule**

**Xóa thẻ parent đang có và thay bằng**

<parent>  
 <artifactId>ibss-platform</artifactId>  
 <groupId>vn.com.ibss</groupId>  
 <version>0.0.1-SNAPSHOT</version>  
 </parent>

đồng thời add common

<dependency>  
 <groupId>vn.com.ibss.common</groupId>  
 <artifactId>common</artifactId>  
 <version>1.0</version>  
 </dependency>

**giống như cac service khác**

**Trong file** NewServiceApplication,java phải thêm các Anotation cần thiết

Ví dụ

@SpringBootApplication  
 @EnableResourceServer  
 @EnableDiscoveryClient  
 @RibbonClient(name = "new-service")  
 @EnableSwagger2

**nếu add được thì thêm thư viện có thể copy thư việc ở file pom.xml trong service khác để thêm.**

**Tham khảo pom.xml trong py service**

**Tiếp theo là thêm routing config mới cho new -service ở gateway trong cf-service**

**Ở thư mục shared-native**

**thêm các thông tin về id, url, … trong path: zuul → routes**

new-service:  
 serviceId: new-service  
 url: [http://localhost:5009](http://localhost:5009/) path: /new/\*\*  
 stripPrefix: false

spring:  
 profiles:  
 active: @activatedProfile@

**Vì sử dụng config server nên config của new service cũng đăt ở trong cf-service**

**Tạo mới 1 file new-service.yml thư mục shared-native rồi copy nội dung từ py-service sang**

Cần điều chỉnh lại thông tin context-path và port cho new service để gateway có thể gọi đến

Ví dụ   
server:  
 servlet:  
 context-path: /new  
 port: 5009

giống với config ở file gw-service

Cuối cùng là các thông tin khác nếu cần.

Lưu ý biến ${DB\_ADDRESS} , ${HOST\_ADDRESS}

là để tham số hóa tập trung , nó sẽ lấy giá trị ở file application.yml trong **thư mục shared-native**

Tiếp :   
eureka:  
 instance:  
 prefer-ip-address: true  
 client:  
 service-url:  
 *# default-zone: http://user:password@${HOST\_ADDRESS}:8761/eureka*  
default-zone: http://${HOST\_ADDRESS}:8761/eureka  
 fetchRegistry: true  
   
là config cho việc service registry (rg-service)

Cuối cùng là add new-service module vào root-module

Trong pom.xml add <module>new-service</module> trong thẻ <modules>

<modules>  
 <module>cf-service</module>  
 <module>rg-service</module>  
 <module>au-service</module>  
 <module>common</module>  
 <module>new-service</module>  
   
</modules>

# Cách cấu hình:

**Chung:**

Project hiện tại chia làm 2 profile:

**native** là profile cho dev ở local.

**prod**: là profile ở trên cloud.

Cấu hình profile ở thuộc tính **activatedProfile** trong file pom.xml ở root module.

Ví dụ:

<**properties**>

<**java.version**>1.8</**java.version**>

<**spring-cloud.version**>Greenwich.RELEASE</**spring-cloud.version**>

<**configUriProd**>localhost</**configUriProd**>

<**activatedProfile**>native</**activatedProfile**>

</**properties**>

**Riêng cho từng Project:**

Trước khi start module: name của service và URI của config server được đặt ở file bootstrap.yml.

Thông số cấu hình sẽ được đặt ở file theo tên module trong config server.

Active của module đặt ở file application.yml.

**spring**:

**profiles**:

**active**: @activatedProfile@

Khi start module thì **@activeProfile@** sẽ đc thay bằng giá trị trong file pom.xml ở root module.