

SPRING BOOT – WEB SERVICE - MICROSERVICE

Author: Đỗ Thị Hồng Ngát

Mục lục

A. Web Service	2
I. Giới thiệu Web Service.....	2
1. Web Service là gì	2
2. Sự khác nhau giữa RESFUL và SOAP	3
II. Tạo Web Service CRUD cơ bản	4
1. Tạo dự án spring boot	4
1.1 Giới thiệu về Spring boot	4
1.2 Tạo project.....	4
1.3 Cấu trúc của một project Spring Boot.....	5
1.4 Cách run một project Spring Boot	6
2. RESTFUL.....	6
2.1 User Model	6
2.2 User Service	6
2.3 User Controller.....	7
2.4 Testing	8
3. SOAP	10
2.1 Xml schema.....	10
2.2 UserService	12
2.3 Endpoint	12
2.4 Config.....	13
2.5 Testing	14
III. Rest Template	15
1. Giới thiệu	15
2. Gửi request với method GET	15
2.1 Get JSON	15
2.2 Get Object.....	16
3. Gửi request với method POST	16
3.1 postForObject API.....	16
3.2 Submit form data	16
4. Gửi request với Options	17
5. Gửi request với method PUT	17
6. Gửi request với method DELETE.....	17
B. Microservice.....	17
I. Giới thiệu Microservice	17
1. Microservice và cái công trình phụ	18
2. Microservice ưu và nhược.....	20
2.1. Ưu điểm	20
2.2. Nhược điểm	21
II. Bài toán ví dụ	22
III. Tạo dự án Microservice với Spring Boot	23
1. Tạo project Spring Boot cho 3 microservice application.....	23
2. Xây dựng movie-catalog-service.....	23
3. Xây dựng movie-info-service	24
4. Xây dựng ratings-data-service	25

5.	Cải tiến Movie Catalog Service.....	26
5.1.	Sử dụng Bean annotation	28
5.2.	Sử dụng Discovery Server	29
6.	Xây dựng Discovery Server	30
6.1.	Tạo dự án	30
6.2.	Đăng kí service với Discovery Server	33
6.3.	Cải tiến Movie Catalog Service lần 2	35
7.	Gateway - Zuul.....	36
8.	Xác thực user bằng JWT, Spring Security	40
8.1	Authentication work flows.....	40
8.2.	Json Web Token (JWT).....	40
8.3.	Gateway	41
8.4.	AuthService	44
8.5.	Testing	48

A. Web Service

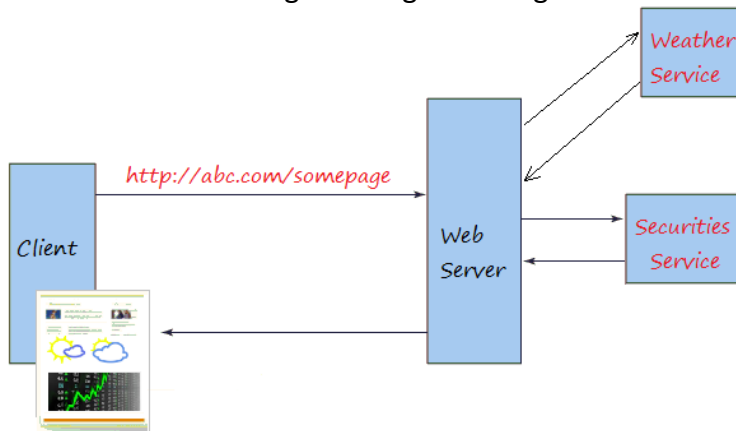
I. Giới thiệu Web Service

1. Web Service là gì

Khi bạn gõ vào một URL vào trình duyệt và bạn nhận được một trang web. Đây là một nội dung mà thông thường bạn có thể đọc được, nó là nội dung dành cho người dùng ở đầu cuối.



Web Service là một dịch vụ web, nó là một khái niệm rộng hơn so với khái niệm web thông thường, nó cung cấp các thông tin thô, và khó hiểu với đa số người dùng, chính vì vậy nó được sử dụng bởi các ứng dụng. Các ứng dụng này sẽ chế biến các dữ liệu thô trước khi trả về cho người dùng cuối cùng.

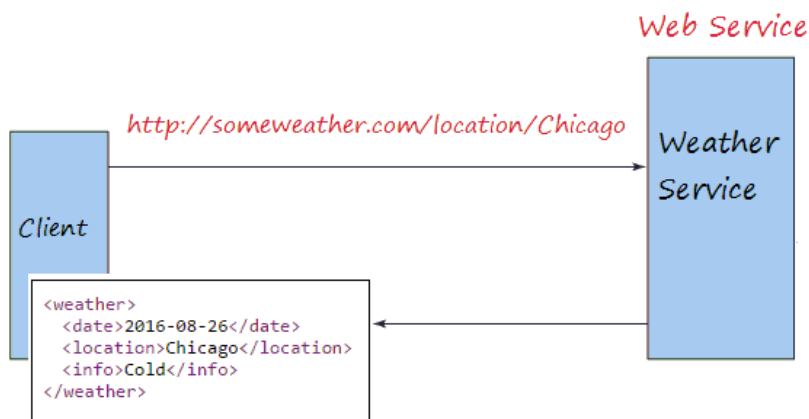


Ví dụ bạn vào một trang web ABC nào đó để xem thông tin về thời tiết và chứng khoán. Trang web đó sẽ hiển thị cho bạn các thông tin bạn muốn.

Để có được các dữ liệu về thời tiết ứng dụng ABC cần phải lấy thông tin từ một nguồn nào đó, nó có thể là một dịch vụ web chuyên cung cấp các số liệu thời tiết ứng với các vùng miền khác nhau.

Tương tự như vậy để có các số liệu về chứng khoán ứng dụng ABC cũng cần phải liên hệ với dịch vụ cung cấp các số liệu này. Các dữ liệu sẽ được chế biến trước khi trả về cho bạn là một trang web hoàn chỉnh.

Các Web Service thường cung cấp các dữ liệu thô mà nó khó hiểu đối với đa số người dùng thông thường, chúng thường được trả về dưới dạng XML hoặc JSON.



2. Sự khác nhau giữa RESFUL và SOAP

REST (REpresentational State Transfer) là một kiểu cấu trúc (architectural style) cung cấp API thông qua internet để xử lý các hoạt động CRUD trên dữ liệu. REST tập trung vào việc truy cập các tài nguyên được đặt tên thông qua một giao diện duy nhất. Những lý do khiến bạn thích thú với REST:

- REST sử dụng chuẩn HTTP nên nó đơn giản hơn nhiều so với trước đây. Tạo clients, phát triển các API, tài liệu dễ hiểu hơn và không có nhiều thứ mà REST không làm được. Về cơ bản điều này thực sự tốt hơn SOAP
- REST cho phép nhiều định dạng dữ liệu khác nhau trong khi SOAP chỉ cho phép XML. Mặc dù điều này có vẻ như làm tăng thêm sự phức tạp cho REST vì bạn cần phải xử lý nhiều định dạng. Nhưng theo kinh nghiệm của tôi, nó lại thực sự có lợi. JSON phù hợp hơn cho dữ liệu và phân tích cú pháp nhanh hơn. REST cho phép hỗ trợ tốt hơn cho browser client do nó có hỗ trợ cho JSON.
- REST có hiệu suất tốt hơn và khả năng mở rộng. Những lần đọc của REST có thể cached lại được còn SOAP thì không.
- Có một điều thú vị là REST hoàn toàn có thể sử dụng SOAP web services để thực hiện.
- **SOAP** (Simple Object Access Protocol) là giao thức sử dụng XML để định nghĩa dữ liệu dạng thuần văn bản (plain text) thông qua HTTP. SOAP là cách mà Web Service sử dụng để truyền tải dữ liệu. Vì dựa trên XML nên SOAP là một giao thức không phụ thuộc platform cũng như bất kỳ ngôn ngữ lập trình nào.

Dưới đây là một số lý do có thể khiến bạn muốn lựa chọn SOAP cho dịch vụ web của mình.

- **WS-Security** SOAP không chỉ hỗ trợ SSL (giống như REST) mà còn hỗ trợ WS-Security, bổ sung thêm một số tính năng enterprise security. Hỗ trợ nhận dạng thông qua các trung gian, không chỉ là point-to-point như SSL. Nó được dùng khi muốn xây dựng những web service đảm bảo và tin cậy. Web Service Security đảm bảo cho tính an toàn, sự toàn vẹn thông điệp và tính tin cậy của thông điệp.

- *WS-AtomicTransaction* Khi muốn có các giao dịch ACID qua một dịch vụ, bạn sẽ phải cần SOAP. Mặc dù REST có hỗ trợ các transactions, nhưng nó không toàn diện và cũng không phù hợp với ACID. REST bị hạn chế bởi HTTP nên không thể cung cấp cam kết hai pha trên các tài nguyên giao dịch phân tán, nhưng SOAP lại có thể làm được điều này. Thật may mắn các giao dịch ACID gần như không có ý nghĩa nhiều đối với các dịch vụ internet thông thường. Nhưng đôi khi các ứng dụng doanh nghiệp lại cần mức độ tin cậy giao dịch này.
- *WS-ReliableMessaging* REST không có hệ thống báo lỗi chuẩn và mong muốn khách hàng giải quyết các lỗi communicate bằng cách retry và ... retry... (yaoming). SOAP đã thành công trong việc xử lý những tình huống này và cung cấp end-to-end một cách tin cậy thông qua các trung gian SOAP

SOAP rõ ràng là hữu ích và quan trọng. Ví dụ, Nếu bạn viết một ứng dụng để giao tiếp với ngân hàng chắc chắn bạn sẽ cần phải sử dụng SOAP. Tất cả ba tính năng trên là bắt buộc đối với các giao dịch ngân hàng. Ví dụ: nếu tôi chuyển tiền từ tài khoản này sang tài khoản khác, tôi cần phải chắc chắn rằng nó đã hoàn tất. Việc cứ cố gắng retry thực sự là quá kinh dị nếu giao dịch thành công lần đầu tiên nhưng thông báo tôi nhận được lại là thất bại

II. Tạo Web Service CRUD cơ bản

1. Tạo dự án spring boot

1.1 Giới thiệu về Spring boot

Spring Boot là một dự án nổi bật trong hệ sinh thái Spring Framework. Nếu như trước đây, công đoạn khởi tạo một dự án Spring khá vất vả từ việc khai báo các dependency trong file pom.xml cho đến cấu hình bằng XML hoặc annotation phức tạp, thì giờ đây với Spring Boot, chúng ta có thể tạo các ứng dụng Spring một cách nhanh chóng và cấu hình cũng đơn giản hơn.

Dưới đây là một số tính năng nổi bật của Spring Boot:

- Đóng gói ứng dụng Spring dưới dạng một file JAR (stand-alone application). Chúng ta có thể dễ dàng start ứng dụng Spring chỉ với câu lệnh quen thuộc `java -jar`.
- Tối ưu công đoạn cấu hình cho ứng dụng Spring, không sinh code cấu hình và không yêu cầu phải cấu hình bằng XML.
- Cung cấp một loạt các tính năng phi chức năng phổ biến cho các dự án lớn như nhúng trực tiếp web server như Tomcat, Jetty, ... vào ứng dụng, bảo mật, health check...

Hiện tại, Spring Boot có 2 version chính là: v1.5.x, v2.x

1.2 Tạo project

Tạo project bằng trang <https://start.spring.io/>

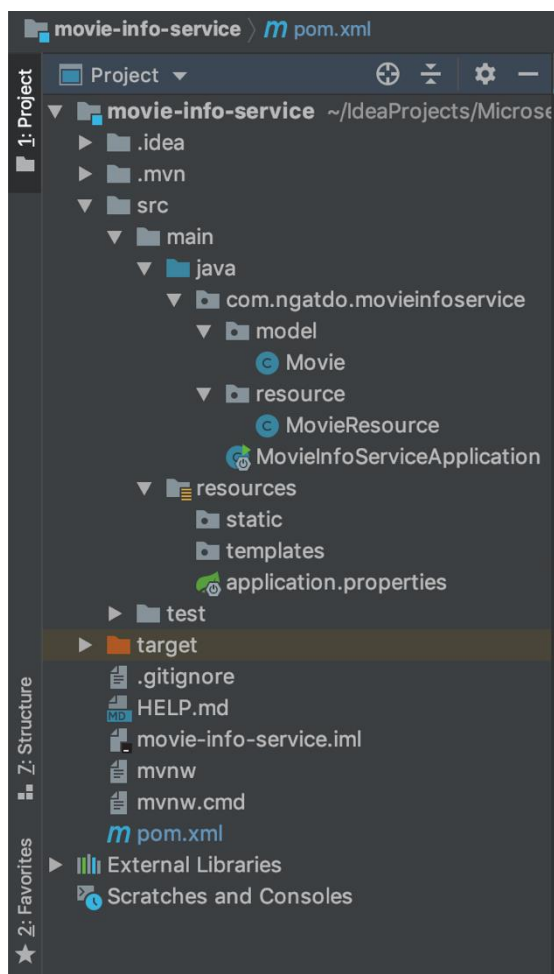
Điền các thông tin:

- Chọn build project bằng Maven, Java 8
- Group: tên tác giả, nhóm tác giả
- Artifact: Tên dự án

- Mục dependencies: Thêm các thư viện cho project, ở ví dụ này thêm Web dependency
Ví dụ:

The screenshot shows the Spring Initializr web interface. On the left, there's a sidebar with sections: Project, Language, Spring Boot, Project Metadata, and Dependencies. The main area is divided into two tabs: 'Maven Project' (selected) and 'Gradle Project'. Under 'Maven Project', there are three language tabs: 'Java' (selected), 'Kotlin', and 'Groovy'. Below these, the 'Spring Boot' version is set to '2.1.3'. The 'Project Metadata' section shows the 'Group' as 'com.ngatdo' and the 'Artifact' as 'movie-info-service'. A 'More options' button is visible. The 'Dependencies' section has a search bar with the text 'Web, Security, JPA, Actuator, Devtools...' and a list of 'Dependencies selected' including 'Web [Web]' with the description 'Servlet web application with Spring MVC'. At the bottom, there's a green button labeled 'Generate Project' with a download icon.

1.3 Cấu trúc của một project Spring Boot



- các file model sẽ nằm trong phần src/java/model
- các file controller sẽ nằm trong phần src/java/resource
- pom.xml file quản lý các library, plugin...
- application.properties file config (name, port, routes...)
- thư mục src/java/resources/static chứa các file static như css
- thư mục src/java/resources/static chứa các file rft để làm giao diện cho dự án
- Lúc này toàn bộ cấu hình của dự án sẽ gói gọn trong 1 file là MovieInforApplication

1.4 Cách run một project Spring Boot

IDE: Ta sẽ chạy hàm main của dự án như các chương trình bình thường
commandline, lệnh maven: Tại thư mục ngoài cùng của dự án

+ Build: **mvn clean install**

+ Run: **mvn spring-boot:run**

* Nếu muốn bỏ test, ta cần thêm tham số **-Dmaven.test.skip=true**

* Có thể vừa build vừa run cùng một lúc

Ví dụ: **mvn clean install spring-boot:run -Dmaven.test.skip=true**

2. RESTFUL

Link Project: <https://github.com/ngatdo3003/Web-Service>

2.1 User Model

Class User gồm 3 thuộc tính, name, empId, salary

```
public class User {
    protected String name;
    protected int empId;
    protected double salary;

    public User() {}

    public User(String name, int empId, double salary) {
        this.name = name;
        this.empId = empId;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getEmpId() {
        return empId;
    }

    public void setEmpId(int empId) {
        this.empId = empId;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

2.2 User Service

Muốn có data khi truy vấn, để đơn giản đối với phần này ta tạo ra một Map gồm một số user có sẵn như bên dưới.

Class **UserService** dùng annotation **@Service** để khai báo đây là một Bean với tên là `restUserService` gồm các phương thức CRUD và **@Postconstruct** là phương thức khởi tạo của một service

```
@Service("restUserService")
public class UserService {
    private static final Map<String, User> users = new HashMap<>();

    @PostConstruct
    public void initialize() {

        User peter = new User("Peter", 1111, 12000);
        User sam = new User("Sam", 1112, 32000);
        User ryan = new User("Ryan", 1113, 16000);

        users.put(peter.getName(), peter);
        users.put(sam.getName(), sam);
        users.put(ryan.getName(), ryan);
    }

    public User getUsers(String name) {
        return users.get(name);
    }

    public User[] getAllUsers()
    {
        Set<String> list = users.keySet();
        User[] u = new User[list.size()];
        int i=0;
        for(String name : list){
            u[i] = users.get(name);
            i++;
        }
        return u;
    }
    public User addUser(User user)
    {
        users.put(user.getName(), user);
        return user;
    }
    public User updateUser(User user)
    {
        users.put(user.getName(), user);
        return user;
    }
    public User deleteUser(String name)
    {
        User user = users.get(name);
        users.remove(name);
        return user;
    }
}
```

2.3 User Controller

- Annotation **@RestController** để thể hiện đây là một Restful controller
- Khai báo `userService` bằng annotation **@Autowired**
- Các method GET, PUT, POST, DELETE tương ứng **@GetMapping**, **@PutMapping**, **@PostMapping**, **@DeleteMapping**, thuộc tính `value` là đường dẫn tới service
- **@PathVariable** dùng để lấy biến trên đường dẫn mà người dùng truyền vào


```

@RestController
public class UserController {
    @Autowired
    private UserService userService;

    @GetMapping(value = "/all")
    public User[] getUser() {
        return userService.getAllUsers();
    }

    @GetMapping(value = "/user/{name}")
    public User getUser(@PathVariable("name") String name) {
        return userService.getUser(name);
    }

    @PostMapping(value = "/add")
    public User addUser(@RequestBody User user) {
        return userService.addUser(user);
    }

    @PutMapping(value = "/edit")
    public User updateUser(@RequestBody User user) {
        return userService.updateUser(user);
    }

    @RequestMapping(value = "/delete/{name}")

    public User deleteUser(@PathVariable("name") String name) {
        return userService.deleteUser(name);
    }
}

```

2.4 Testing

- Test phương thức GET

GET localhost:8080/all Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (3) Test Results Status: 200 OK Time: 20 ms Size: 269 B Save

Pretty Raw Preview Visualize JSON

```

1 {
2   {
3     "name": "Ryan",
4     "empId": 1113,
5     "salary": 16000.0
6   },
7   {
8     "name": "Peter",
9     "empId": 1111,
10    "salary": 12000.0
11  },
12  {
13    "name": "Sam",
14    "empId": 1112,
15    "salary": 32000.0
16  }
17 }

```

- Test phương thức POST

POST localhost:8080/add Send

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "name": "Ngat",
3   "empId": 1999,
4   "salary": 100000
5 }
```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 456 ms Size: 176 B Sav

Pretty Raw Preview Visualize JSON

```
1 {
2   "name": "Ngat",
3   "empId": 1999,
4   "salary": 100000.0
5 }
```

- Test phương thức PUT

PUT localhost:8080/edit Send

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "name": "Ngat",
3   "empId": 1999,
4   "salary": 131231
5 }
```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 30 ms Size: 176 B Sav

Pretty Raw Preview Visualize JSON

```
1 {
2   "name": "Ngat",
3   "empId": 1999,
4   "salary": 131231.0
5 }
```

- Test phương thức DELETE

DELETE localhost:8080/delete/Ngat Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (3) Test Results Status: 200 OK Time: 29 ms Size: 176 B Sav

Pretty Raw Preview Visualize JSON

```

1 {
2   "name": "Ngat",
3   "empId": 1999,
4   "salary": 131231.0
5 }

```

3. SOAP

2.1 Xml schema

Xmlns là xml namespace dùng để viết tắt thay vì viết link(ví dụ viết xs thay vì viết link http:w3...)

Để tạo một class: bắt đầu bằng tag **<complexType>**

<sequence>

<element> để khai báo các thuộc tính của class

...

```

<xs:complexType name="getUserRequest">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
  </xs:sequence>
</xs:complexType>

```

Class getUserRequest nhận vào name:String và class getUserResponse có trách nhiệm trả về user có tên là name.

Ta phải định nghĩa thêm class user có 3 thuộc tính: name, empId, salary

File users.xds được tạo trong src/main/resources

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://ngatdo.com/soapws"
  targetNamespace="http://ngatdo.com/soapws"
  elementFormDefault="qualified">

  <xs:element name="getUserRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="addUserRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="user" type="tns:user" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="deleteUserRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="updateUserRequest">
    <xs:complexType>

```

```

<xs:sequence>
  <xs:element name="user" type="tns:user" />
</xs:sequence>
</xs:complexType>
</xs:element>

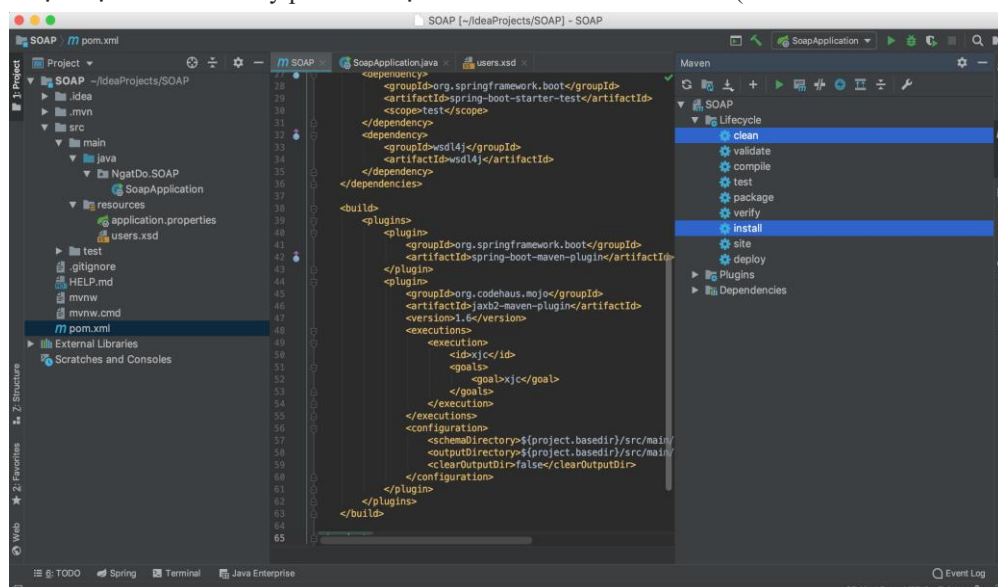
<xs:element name="getUserResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="user" type="tns:user" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="user">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="empId" type="xs:int" />
    <xs:element name="salary" type="xs:double" />
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

Generate java class dùng maven clean

Chọn mục maven bên tay phải rồi chọn clean và install rồi bấm run (mũi tên run xanh ở dưới)



Ta được các class như hình, xmlns tns chỉ package mà các class sẽ được generate, đường dẫn tới package configure trong pom.xml

```

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>jaxb2-maven-plugin</artifactId>
  <version>1.6</version>
  <executions>
    <execution>
      <id>xjc</id>
      <goals>
        <goal>xjc</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <schemaDirectory>${project.basedir}/src/main/resources/</schemaDirectory>
    <outputDirectory>${project.basedir}/src/main/java</outputDirectory>
  </configuration>
</plugin>

```

```
<clearOutputDir>false</clearOutputDir>
</configuration>
</plugin>
```

Sau khi tạo ta sẽ được các file: User, AddUserRequest, UpdateUserRequest, DeleteUserRequest, GetUserRequest, GetUserResponse, ObjectFactory, package-info

2.2 UserService

Tạo UserService tương tự với Restful service

```
@Service("soapUserService")
public class UserService {

    private static final Map<String, User> users = new HashMap<>();

    @PostConstruct
    public void initialize() {

        User peter = new User("Peter", 1111, 12000);
        User sam = new User("Sam", 1112, 32000);
        User ryan = new User("Ryan", 1113, 16000);

        users.put(peter.getName(), peter);
        users.put(sam.getName(), sam);
        users.put(ryan.getName(), ryan);
    }

    public User getUsers(String name) {
        return users.get(name);
    }

    public User[] getAllUsers()
    {
        Set<String> list = users.keySet();
        User[] u = new User[list.size()];
        int i=0;
        for(String name : list){
            u[i] = users.get(name);
            i++;
        }
        return u;
    }

    public User addUser(User user)
    {
        users.put(user.getName(), user);
        return user;
    }

    public User updateUser(User user)
    {
        users.put(user.getName(), user);
        return user;
    }

    public User deleteUser(String name)
    {
        User user = users.get(name);
        users.remove(name);
        return user;
    }
}
```

2.3 Endpoint

Đóng vai trò như **Controller** bên restful service, với mỗi phương thức ta cần khai báo **namespace**: là tên của các service và **localPart** tương ứng với các request đã khai báo ở bên trên

```
@Endpoint
public class UserEndpoint {

    @Autowired
    private UserService userService;

    @PayloadRoot(namespace = "http://ngatdo.com/soapws",
        localPart = "getUserRequest")
    @ResponsePayload
    public GetUserResponse getUserRequest(@RequestPayload GetUserRequest request) {
        GetUserResponse response = new GetUserResponse();
        response.setUser(userService.getUsers(request.getName()));
        return response;
    }

    @PayloadRoot(namespace = "http://ngatdo.com/soapws",
        localPart = "addUserRequest")
    @ResponsePayload
    public GetUserResponse addUserRequest(@RequestPayload AddUserRequest request) {
        GetUserResponse response = new GetUserResponse();
        response.setUser(userService.addUser(request.getUser()));
        return response;
    }

    @PayloadRoot(namespace = "http://ngatdo.com/soapws",
        localPart = "updateUserRequest")
    @ResponsePayload
    public GetUserResponse updateUserRequest(@RequestPayload UpdateUserRequest
request) {
        GetUserResponse response = new GetUserResponse();
        response.setUser(userService.updateUser(request.getUser()));
        return response;
    }

    @PayloadRoot(namespace = "http://ngatdo.com/soapws",
        localPart = "deleteUserRequest")
    @ResponsePayload
    public GetUserResponse deleteUserRequest(@RequestPayload DeleteUserRequest
request) {
        GetUserResponse response = new GetUserResponse();
        response.setUser(userService.deleteUser(request.getName()));
        return response;
    }
}
```

2.4 Config

Config path cho service

```
@EnableWs
@Configuration
public class SoapWebServiceConfig extends WsConfigurerAdapter {

    @Bean
    public ServletRegistrationBean
messageDispatcherServlet(ApplicationContext context) {
        MessageDispatcherServlet servlet = new
MessageDispatcherServlet();
        servlet.setApplicationContext(context);
        servlet.setTransformWsdLocations(true);
        return new ServletRegistrationBean(servlet, "/soapWS/*");
    }
}
```

```

@Bean
public XsdSchema userSchema() {
    return new SimpleXsdSchema(new ClassPathResource("users.xsd"));
}

@Bean
public DefaultWsdll11Definition defaultWsdll11Definition(XsdSchema
userSchema) {

    DefaultWsdll11Definition definition = new
DefaultWsdll11Definition();
    definition.setSchema(userSchema);
    definition.setLocationUri("/soapWS");
    definition.setPortTypeName("UserServicePort");
    definition.setTargetNamespace("http://ngatdo.com/soapws");
    return definition;
}
}

```

2.5 Testing

Viết xml file cho các request

- GET:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:us="http://ngatdo.com/soapws">
    <soapenv:Header/>
    <soapenv:Body>
        <us:getUserRequest>
            <us:name>Sam</us:name>
        </us:getUserRequest>
    </soapenv:Body>
</soapenv:Envelope>

```

- PUT:

```

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:updateUserRequest xmlns:ns2="http://ngatdo.com/soapws">
            <ns2:user>
                <ns2:name>Ngat</ns2:name>
                <ns2:empId>40</ns2:empId>
                <ns2:salary>32</ns2:salary>
            </ns2:user>
        </ns2:updateUserRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

- POST:

```

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:addUserRequest xmlns:ns2="http://ngatdo.com/soapws">
            <ns2:user>
                <ns2:name>Ngat</ns2:name>

```

```

        <ns2:empId>30</ns2:empId>
        <ns2:salary>320300.0</ns2:salary>
    </ns2:user>
</ns2:addUserRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

- DELETE:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:us="http://ngatdo.com/soapws">
    <soapenv:Header/>
    <soapenv:Body>
        <us:deleteUserRequest>
            <us:name>Sam</us:name>
        </us:deleteUserRequest>
    </soapenv:Body>
</soapenv:Envelope>

```

Test tương tự Resful service

The screenshot shows the Postman interface for a SOAP POST request. The URL is `http://localhost:8091/soapWS`. The request body is XML, and the response status is 200 OK.

Request Body (XML):

```

1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2   xmlns:us="http://ngatdo.com/soapws">
3   <soapenv:Header/>
4   <soapenv:Body>
5       <us:getUserRequest>
6           <us:name>Ryan</us:name>
7       </us:getUserRequest>
8   </soapenv:Body>
9 </soapenv:Envelope>

```

Response Body (XML):

```

1 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
2   <SOAP-ENV:Header/>
3   <SOAP-ENV:Body>
4       <ns2:getUserResponse xmlns:ns2="http://ngatdo.com/soapws">
5           <ns2:user>
6               <ns2:name>Ryan</ns2:name>
7               <ns2:empId>1113</ns2:empId>
8               <ns2:salary>16000.0</ns2:salary>
9           </ns2:user>
10          </ns2:getUserResponse>
11      </SOAP-ENV:Body>
12 </SOAP-ENV:Envelope>

```

III. Rest Template

1. Giới thiệu

RestTemplate cung cấp các API giúp bạn tạo ra một Restful client

2. Gửi request với method GET

2.1 Get JSON

```

System.out.println("GET JSON");
RestTemplate restTemplate = new RestTemplate();
String url = "http://localhost:8080/all";

```



```

ResponseBody<String> response
    = restTemplate.getForEntity(url , String.class);
if(response.getStatusCode() == HttpStatus.OK){
    System.out.println(response);
}
else {
    System.out.println("GET JSON error");
}

```

2.2 Get Object

```

System.out.println("GET OBJECT");
RestTemplate restTemplate = new RestTemplate();
String url = "http://localhost:8080/user/Ryan";
User user = restTemplate.getForObject(url , User.class);
if(user != null){
    System.out.println("Name: " + user.getName());
    System.out.println("EmpID: " + user.getEmpId());
    System.out.println("Salary: " + user.getSalary());
}
else {
    System.out.println("GET Object error");
}

```

Test:

```

GET JSON
21:14:11.709 [main] DEBUG org.springframework.web.client.RestTemplate - Created GET request for "http://localhost:8080/all"
21:14:11.722 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept header to [text/plain, application/json, application/*+json, /*]
21:14:11.839 [main] DEBUG org.springframework.web.client.RestTemplate - GET request for "http://localhost:8080/all" resulted in 200 (null)
21:14:11.841 [main] DEBUG org.springframework.web.client.RestTemplate - Reading [java.lang.String] as "application/json; charset=UTF-8" using [org.springframework.http.converter.S
<200, [{"name": "Ryan", "empId": 1113, "salary": 16000.0}, {"name": "Ngat", "empId": 1999, "salary": 100123.0}, {"name": "Peter", "empId": 1111, "salary": 12000.0}, {"name": "Sam", "empId": 1112, "sala
GET OBJECT
21:14:11.882 [main] DEBUG org.springframework.web.client.RestTemplate - Created GET request for "http://localhost:8080/user/Ryan"
21:14:12.009 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept header to [application/json, application/*+json]
21:14:12.015 [main] DEBUG org.springframework.web.client.RestTemplate - GET request for "http://localhost:8080/user/Ryan" resulted in 200 (null)
21:14:12.016 [main] DEBUG org.springframework.web.client.RestTemplate - Reading [class com.ngatdo.rest.model.User] as "application/json; charset=UTF-8" using [org.springframework.h
Name: Ryan
EmpID: 1113
Salary: 16000.0

```

3. Gửi request với method POST

3.1 postForObject API

```

System.out.println("POST OBJECT");
RestTemplate restTemplate = new RestTemplate();
String url = "http://localhost:8080/add";
HttpEntity<User> request = new HttpEntity<>(new User("Ngat", 1999, 100123));
User user = restTemplate.postForObject(url, request, User.class);
if (user != null){
    System.out.println("Name: " + user.getName());
    System.out.println("EmpID: " + user.getEmpId());
    System.out.println("Salary: " + user.getSalary());
} else {
    System.out.println("POST Object error");
}

```

Test:

```

POST OBJECT
21:14:12.075 [main] DEBUG org.springframework.web.client.RestTemplate - Created POST request for "http://localhost:8080/add"
21:14:12.077 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept header to [application/json, application/*+json]
21:14:12.170 [main] DEBUG org.springframework.web.client.RestTemplate - Writing [com.ngatdo.rest.model.User@438a68] using [org.springframework.http.converter.json.MappingJackson2H
21:14:12.266 [main] DEBUG org.springframework.web.client.RestTemplate - POST request for "http://localhost:8080/add" resulted in 200 (null)
21:14:12.266 [main] DEBUG org.springframework.web.client.RestTemplate - Reading [class com.ngatdo.rest.model.User] as "application/json; charset=UTF-8" using [org.springframework.h
Name: Ngat
EmpID: 1999
Salary: 100123.0

```

3.2 Submit form data

```

public void submitFormData() {
    System.out.println("SUBMIT FORM DATA");
    RestTemplate restTemplate = new RestTemplate();
    String url = "http://localhost:8080/form";
    //1. Set content-Type to form urlencoded

```

```

HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
//2. Add all key-value paired to a map
MultiValueMap<String, String> map= new LinkedMultiValueMap<>();
map.add("id", "1");
//3. Put map to header of request
HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map,
headers);
//4. use postForEntity to call service
ResponseEntity<String> response = restTemplate.postForEntity(
    url, request, String.class);
}

```

4. Gửi request với Options

Options request sẽ cho phép gửi method theo tùy chọn trong danh sách supportedMethod

```

Set<HttpMethod> optionsForAllow = restTemplate.optionsForAllow(url);
HttpMethod[] supportedMethods
    = {HttpMethod.GET, HttpMethod.POST, HttpMethod.PUT,
HttpMethod.DELETE};

```

5. Gửi request với method PUT

```

public void putObject() {
    System.out.println("PUT OBJECT");
    RestTemplate restTemplate = new RestTemplate();
    User user = new User("Ngat", 32, 100123);
    String url = "http://localhost:8080/edit";
    restTemplate.put(url, user);
}

```

6. Gửi request với method DELETE

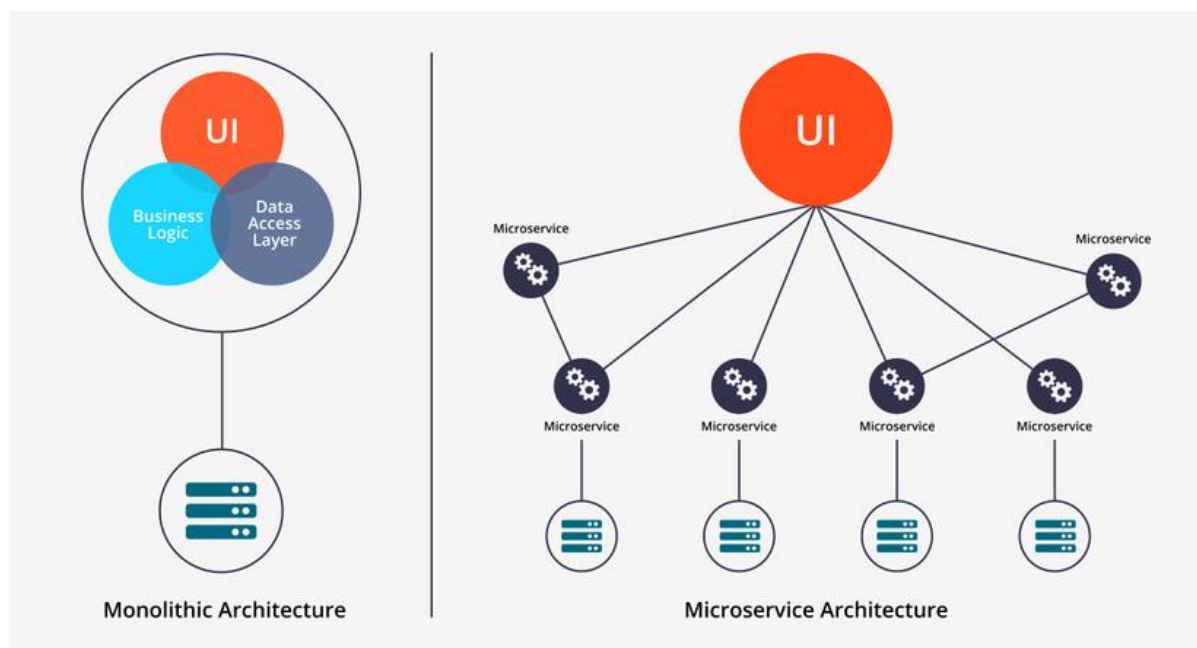
```

public void deleteObject() {
    System.out.println("DELETE OBJECT");
    RestTemplate restTemplate = new RestTemplate();
    String url = "http://localhost:8080/delete/{name}";
    Map<String, String> params = new HashMap<String, String>();
    params.put("name", "Ngat");
    restTemplate.delete(url, params);
}

```

B. Microservice

I. Giới thiệu Microservice



Dạo gần đây, cái dự án đang làm càng ngày càng phình ra, tính năng chồng chéo, muốn update cái này lại dính đến cái kia, nhiều lúc chả biết phải làm thế nào. Hỏi ý kiến các huynh đệ thì được xui tìm hiểu Microservice, nào là dễ quản lý, dễ deploy, phát triển nhanh, vận vận và mây mây. Ô-kê được tư vấn đến thế thì cũng phải tìm hiểu tí xem nó thế nào.

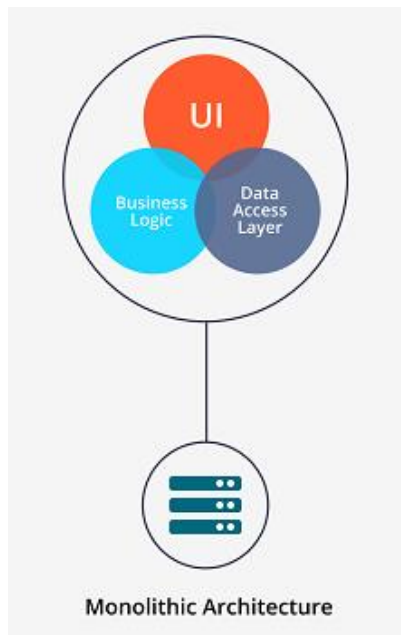
1. Microservice và cái công trình phụ

Ơ hay, Microservice thì liên quan gì đến công trình phụ. Bình tĩnh bạn hiền, từ từ để tôi phân tích đã nào.

Từ cái công trình phụ và kiến trúc Monolithic



Đầu tiên, khi bạn nhìn vào hình trên bạn thấy gì ? Đúng vậy, chỉ là một cái nhà vệ sinh bình thường thôi đúng không. Nhưng hãy để ý tới đặc điểm của nó, cái phòng này có những chức năng gì? Bồn tắm, bồn rửa mặt, bồn vệ sinh, treo đồ... nói tóm lại là all-in-one. Thấy quen không, chúng ta thiết kế phần mềm cũng vậy đấy, gom tất tần tã các tính năng của tất cả các loại user vào cùng 1 project lớn - cách thiết kế này gọi là **kiến trúc nguyên khối (Monolithic Architecture)**.



Kiến trúc này hoạt động tốt và phù hợp với hầu hết các project bởi sự đơn giản và tiện lợi khi tất cả đều ở chung 1 chỗ. Tuy nhiên nó sẽ dần bộc lộ nhược điểm do các module dính liền với nhau thành 1 cục .

Ứng dụng của chúng ta phát triển liên tục, kéo theo đó yêu cầu tính năng mới tăng, dữ liệu tăng, logic phức tạp hơn, giao tiếp với hệ thống khác tăng, và hàng trăm thứ khác dẫn đến một kết quả là ứng dụng phình to ra một cách khủng khiếp. Sau mỗi sprint, hàng loạt tính năng mới được thêm vào, thêm code, thêm bảng, thêm logic... Chỉ sau một thời gian, ứng dụng đơn giản sẽ trở nên kênh càng như một con quái vật. Và số lượng effort bỏ ra để phát triển cũng như bảo trì con quái vật này sẽ không hề nhỏ.

Khi ứng dụng phình quá to, mọi nỗ lực tối ưu đều không còn hiệu quả. Chỉ một chỉnh sửa nhỏ, sẽ phải xem xét sự ảnh hưởng của nó lên toàn bộ hệ thống. Nó quá khó để cho một lập trình viên có thể nắm và hiểu toàn bộ code hệ thống. Và như một hệ quả tất yếu, việc fix bug, hay thêm tính năng mới trở nên khó hơn và tốn nhiều thời gian hơn.

Ngoài ra rất khó để scale ứng dụng do có nhiều module khác nhau với nhu cầu về tài nguyên khác nhau. Nhưng lại nằm chung 1 khối nên chúng ta sẽ phải tính toán tài nguyên để phù hợp với tất cả module gây ảnh hưởng không nhỏ đến chi phí bỏ ra.

Thêm vào đó, ứng dụng nguyên khối sử dụng một ngôn ngữ nên khi muốn update công nghệ để đáp ứng nhu cầu của 1 module thì kéo theo là phải update toàn bộ hệ thống mặc dù những phần khác chưa chắc đã cần.

Cuối cùng, do tất cả các module đang chạy trên cùng 1 process nên tệ nhất là một module bị lỗi có thể làm ngưng toàn hệ thống. Ví dụ như cái phòng kia, giả sử mà cái bồn cầu bị tắc thôi là khởi vào rửa mặt hay tắm luôn đi

Sau những hạn chế của monolithic thì **Kiến trúc hướng dịch vụ (Service Oriented Architecture - SOA)** được sinh ra để giải quyết một phần của vấn đề bằng cách giới thiệu khái niệm "service". Một dịch vụ là một nhóm tổng hợp các tính năng tương tự trong một ứng dụng. Do đó trong SOA, ứng dụng phần mềm được thiết kế như một tổ hợp của các dịch vụ. Tuy nhiên, với SOA, giới hạn hay phạm vi của một dịch vụ khá là rộng và được định nghĩa khá "thô" (coarse-

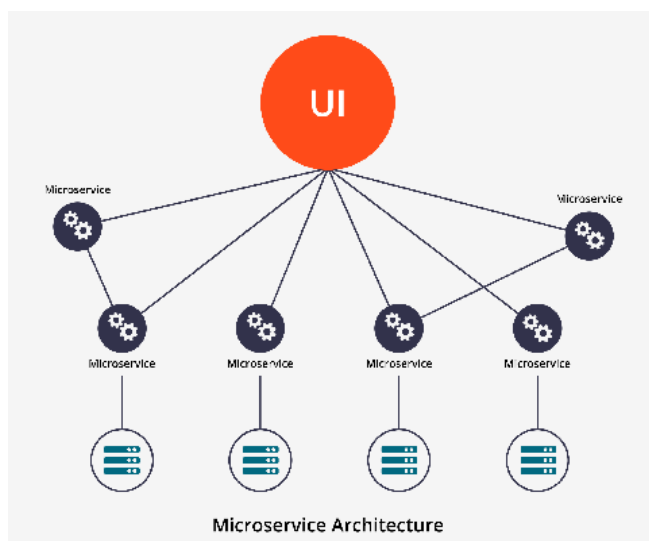
grained). Việc này khiến các services cũng có thể trở nên quá to và phức tạp. Tương tự như ứng dụng monolithic, những dịch vụ này to và phức tạp lên theo thời gian vì thường xuyên thêm các tính năng. Và thế là những dịch vụ này lại trở thành một mớ các dịch vụ monolithic, cũng không còn khác mấy so với kiến trúc một khối thông thường.

Cho đến phần mềm và microservice

Quay trở lại câu chuyện về cái công trình phụ, chúng ta hãy cùng nhau ngắm qua một bức ảnh khác



Chúng ta có thể thấy rằng đây vẫn là cái nhà vệ sinh nhưng trong căn phòng này từng chức năng phòng tắm, bồn rửa mặt, bồn cầu... được xây dựng độc lập để có thể sử dụng riêng biệt mà không hề ảnh hưởng đến nhau. Cách thiết kế này gọi là **Kiến trúc dịch vụ nhỏ (Microservice Architecture)**,



Thiết kế phần mềm cũng vậy, thay vì gom tất cả lại thành một khối, chúng ta chia ứng dụng thành các service nhỏ hơn, chạy riêng biệt, có khả năng phát triển và triển khai độc lập.

2. Microservice ưu và nhược

Microservice sinh ra để giải quyết những nhược điểm của Monolithic, thế nhưng nó không phải là viên đạn bạc toàn năng, nó vẫn có những ưu và nhược điểm riêng.

2.1. Ưu điểm

Đầu tiên, microservices giúp giảm thiểu quá trình phức tạp hóa trong các hệ thống lớn. Với tổng số chức năng không đổi, kiến trúc microservices chia nhỏ hệ thống công kênh ra làm nhiều dịch vụ nhỏ lẻ dễ dàng quản lý và triển khai từng phần so với kiến trúc monolithic.

Trong microservice, các dịch vụ giao tiếp với nhau thông qua Remote Procedure Call (RPC) hay Message-driven API. Ngoài ra, kiến trúc microservices thúc đẩy việc phân tách rạch ròi các module, việc khó có thể làm nếu xây dựng theo kiến trúc monolithic.

Và quan trọng hơn cả, với mỗi dịch vụ nhỏ, chúng ta sẽ có thời gian phát triển nhanh hơn, dễ nắm bắt cũng như bảo trì hơn.

Thứ hai, kiến trúc này cho phép mỗi service được phát triển độc lập bởi một team tập trung cho service đó. Developer có thể tự do lựa chọn công nghệ cho mỗi dịch vụ mình phát triển miễn là nó phù hợp. Sự tự do này không phải là tạo ra một mớ công nghệ hổ lốn, mà có nghĩa là các developer không còn phải bắt buộc phải sử dụng các công nghệ lỗi thời còn tồn tại từ khi bắt đầu dự án. Khi đó, khi bắt tay vào viết một service mới, họ có cơ hội sử dụng công nghệ và môi trường tối ưu nhất với chức năng của service đấy.

Thứ ba, microservices cho phép mỗi service được đóng gói và triển khai độc lập với nhau. (VD: Mỗi service có thể được đóng gói vào một docker container độc lập, giúp giảm tối đa thời gian deploy). Bên cạnh đó, microservices rất phù hợp để áp dụng continuous deployment.

Thứ tư, microservices cho phép mỗi service có thể được scale một cách độc lập với nhau. Việc scale có thể được thực hiện dễ dàng bằng cách tăng số instance cho mỗi service rồi phân tải bằng load balancer. Ngoài ra, chúng ta còn có thể triển khai mỗi service lên server có resource thích hợp để tối ưu hóa chi phí vận hành (việc mà không thể làm được trong kiến trúc monolithic).

2.2. Nhược điểm

Nhược điểm đầu tiên của Microservices cũng chính từ tên gọi của nó. **Microservice nhấn mạnh kích thước nhỏ gọn của dịch vụ.** Một số developer đề xuất dịch vụ siêu nhỏ cỡ dưới 100 dòng code. Service nhỏ là tốt, nhưng nó không phải mục tiêu chính của Microservice. Mục tiêu của Microservice là phân tích đầy đủ ứng dụng để tạo điều kiện phát triển và triển khai ứng dụng nhanh chóng.

Nhược điểm thứ hai của Microservices đến từ đặc điểm hệ thống phân tán (distributed system). Developer cần phải lựa chọn phát triển mỗi dịch vụ nhỏ giao tiếp với các dịch vụ khác bằng cách nào messaging hay là RPC. Hơn thế nữa, họ phải xử lý sự cố khi kết nối chậm, lỗi khi thông điệp không gửi được. Việc này phức tạp hơn nhiều so với ứng dụng nguyên khối nơi các module gọi nhau thông qua các method/procedure cấp ngôn ngữ.

Thứ ba, **phải đảm bảo giao dịch phân tán (distributed transaction)** cập nhật dữ liệu đúng đắn (all or none) vào nhiều dịch vụ nhỏ khác nhau khó hơn rất nhiều, đôi khi là không thể so với đảm bảo giao dịch cập nhật vào nhiều bảng trong một cơ sở dữ liệu trung tâm. Theo nguyên tắc CAP (CAP theorem) thì giao dịch phân tán sẽ không thể thỏa mãn cả 3 điều kiện:

- **consistency:** dữ liệu ở điểm khác nhau trong mạng phải giống nhau
- **availability** yêu cầu gửi đi phải có phúc đáp
- **partition tolerance** hệ thống vẫn hoạt động được ngay cả khi mạng bị lỗi

Những công nghệ cơ sở dữ liệu phi quan hệ (NoSQL) hay message broker tốt nhất hiện nay cũng chưa vượt qua nguyên tắc CAP.

Thứ tư, **testing một dịch vụ trong kiến trúc microservices đôi khi yêu cầu phải chạy cả các dịch vụ nhỏ khác mà nó phụ thuộc**. Do đó khi phân rã ứng dụng một khối thành microservices cần luôn kiểm tra mức độ ràng buộc giữa các dịch vụ. Nếu các dịch vụ nhỏ thiết kế phụ thuộc vào nhau theo chuỗi. A gọi B, B gọi C, C gọi D. Nếu một mắt xích có giao tiếp API thay đổi, liệu các mắt xích khác có phải thay đổi theo không? Nếu có thì việc bảo trì, kiểm thử sẽ phức tạp tương tự ứng dụng một khối. Thiết kế dịch vụ tốt sẽ giảm tối đa ảnh hưởng lan truyền đến các dịch vụ khác.

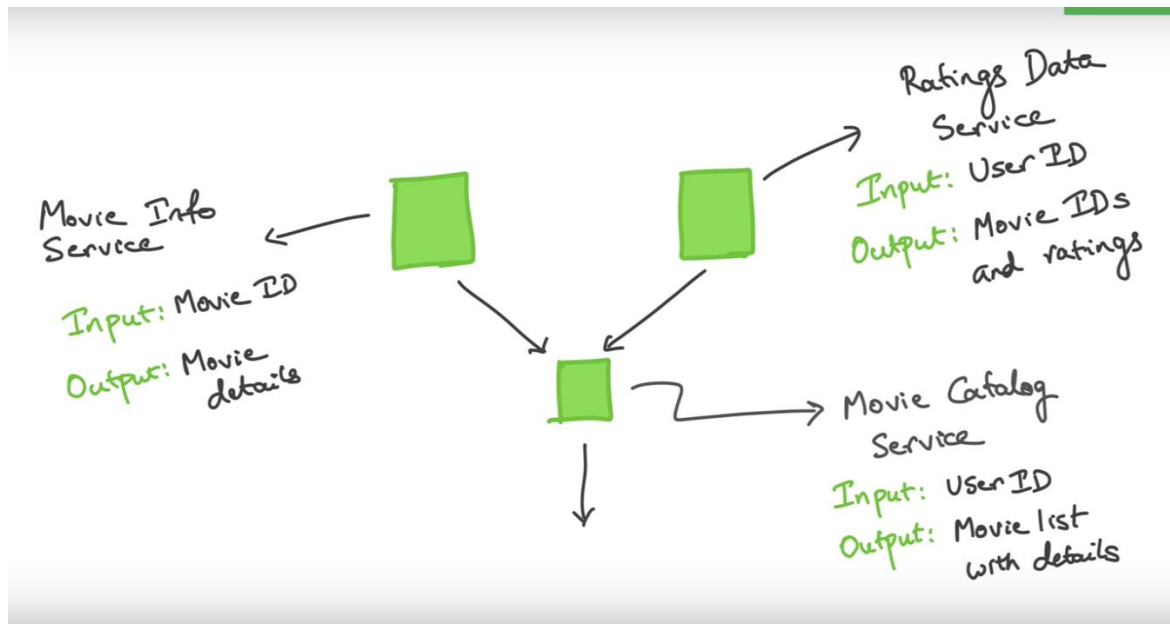
Cuối cùng, **việc triển khai microservices phức tạp hơn rất nhiều nếu làm thủ công theo cách đã làm với ứng dụng monolithic**. Theo Adrian Crockcroft, Hailo có 160 dịch vụ, Netflix có hơn 600 dịch vụ. Với cloud, các máy ảo, docker container có thể linh động bật tắt, dịch chuyển. Vậy cần thiết phải có một cơ chế service discovery để cập nhật tự động địa chỉ IP và cổng, mô tả, phiên bản của mỗi dịch vụ,...

II. Bài toán ví dụ

Giả sử ta đang xây dựng một hệ thống movie recommendation, cần thu thập dữ liệu về rating của người xem đối với các movie, từ đó tính toán và gợi ý cho người dùng một danh sách các phim phù hợp nhất

Ta xây dựng 3 microservice:

1. Movie Info Service: Nhận vào movieID và trả về thông tin của movie đó
2. Rating Data Service: Nhận vào userID, trả về danh sách các rating và movieID tương ứng mà user đã đánh giá
3. Movie Catalog Service: Nhận vào userID, trả về danh sách thông tin các phim mà người đó đánh giá
 - ⇒ Ở service này, ta cần tổng hợp kết quả từ 2 service trên để ra kết quả cuối cùng
 - ⇒ Đầu tiên là từ userID ta lấy được list<movieID, rating> bằng cách gọi Rating Data Service
 - ⇒ Tiếp theo, với mỗi movieID vừa lấy được, gọi Movie Info Service để lấy ra thông tin phim



III. Tạo dự án Microservice với Spring Boot

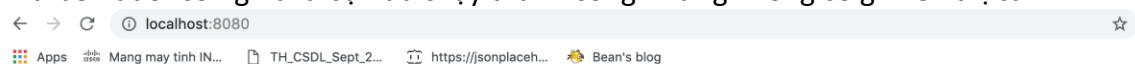
Link full project: <https://github.com/ngatdo3003/Microservice>

1. Tạo project Spring Boot cho 3 microservice application

Tạo project tương tự hướng dẫn cách tạo một application spring boot ở trên

2. Xây dựng movie-catalog-service

- Import project vào IntelliJ và run project
- Cổng mặc định cho 1 project là 8080, vào trang <http://localhost:8080/> ta sẽ thấy trang như bên dưới có nghĩa là bạn đã chạy thành công nhưng không có gì hiển thị cả



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Mar 11 10:20:57 ICT 2019

There was an unexpected error (type=Not Found, status=404).

No message available

- Tại package **model**, tạo class **CatalogItem** gồm 3 thuộc tính: name, desc, rating


```
public class CatalogItem{
    private String name;
    private String desc;
    private int rating;
    // constructor && getter && setter
}
```

- Tại package **resources**, tạo class **MovieCatalogResource** như là một **RestController**, mỗi khi client gửi request thì Spring Boot sẽ tìm các service tương ứng ở đây
- Hiện tại ta chưa check userId, tức là mình nhập userId gì cũng trả về một kết quả giống nhau

```
@RestController
@RequestMapping("/catalog")
public class MovieCatalogResource
{
    @RequestMapping("/{userId}")
    public List<CatalogItem> getCatalog(@PathVariable("userId") String userId)
    {
        return Collections.singletonList(new CatalogItem("Transformer","desc",4));
    }
}
```

Sau khi run và test thì ta thấy response như sau

```
[{"name": "Transformer", "desc": "desc", "rating": 4}]
```

3. Xây dựng movie-info-service

Tương tự ta có class **Movie** và **MovieResource**

```
public class Movie{
    private String movieId;
    private String name;
    //getter & setter
}
```

```
@RestController
@RequestMapping("/movies")
```

```
public class MovieResource{
    @RequestMapping("/{movieId}")
    public Movie getMovieInfor(@PathVariable("movieId") String movieId){
        return new Movie(movieId,"Test name");
    }
}
```

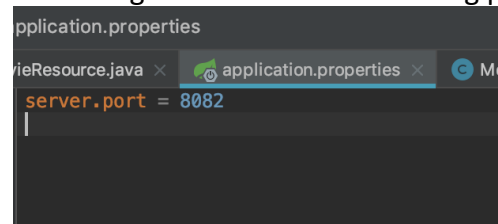
Khi run ta sẽ thấy xuất hiện lỗi

The Tomcat connector configured to listen on port 8080 failed to start. The port may already be in use or the connector may be misconfigured.

Vì movie-catalog-service đã sử dụng port này rồi

- ⇒ Ta đổi port cho cả 3 application và không nên dùng port mặc định 8080
- ⇒ Movie-catalog-service: 8081
- ⇒ Movie-info-service: 8082
- ⇒ Rating-data-service: 8083

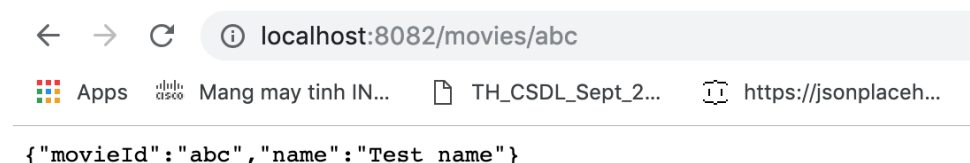
Đổi port bằng cách tìm file application.properties trong resources và thêm config port:



```
application.properties
server.port = 8082
```

server.port=<port>

Sau khi run ta thấy kết quả



```
{ "movieId": "abc", "name": "Test name" }
```

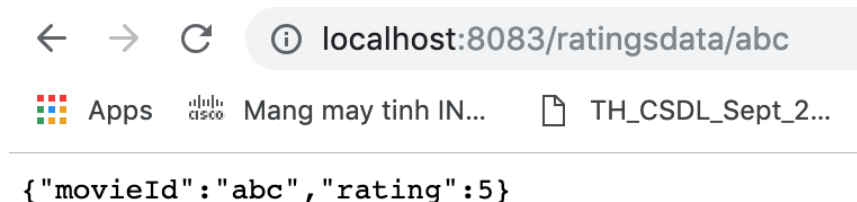
4. Xây dựng ratings-data-service

```
public class Rating{
    private String movieId;
    private int rating;
    //getter & setter
}
```

```
@RestController
@RequestMapping("/ratingsdata")
public class RatingsDataResource{
```

```
@RequestMapping("/{movieId}")
public Rating getRating(@PathVariable("movieId") String movieId){
    return new Rating(movieId, 5);
}
}
```

Sau khi run ta thấy như sau



```
{ "movieId": "abc", "rating": 5 }
```

5. Cải tiến Movie Catalog Service

- Kết thúc việc xây dựng 3 service như 3 application riêng biệt.
- Có một điểm cần cải tiến cho movie-catalog-service
- + Input: userId
- + Output: danh sách các phim người đó đã đánh giá
- 1. Từ userId trả về danh sách movie đã đánh giá(ratings-data-service)
- 2. Từ movieId trả về name (movie-info-service)

⇒ Sử dụng RestTemplate để gọi service khác
 Ví dụ muốn lấy ra movie có movieId là abc ta gọi như sau:

```
Movie movie = restTemplate.getForObject("http://localhost:8082/movies/abc", Movie.class);
```

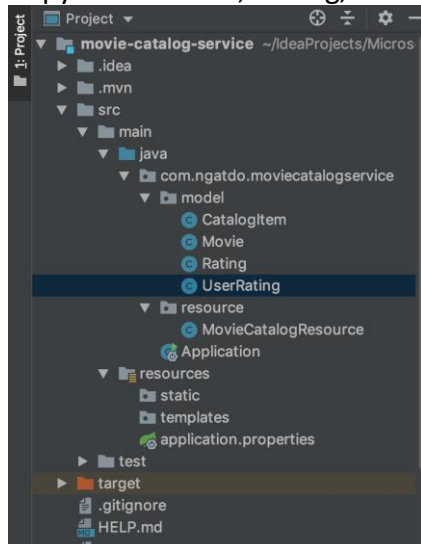
- Đầu tiên ta thêm một web service cho **ratings-data-service**, nhận userId, trả về danh sách phim đã đánh giá.
- Ở đây ta hoàn toàn có thể trả về dưới dạng List<Rating> nhưng lát nữa khi restTemplate gọi đến API này sẽ không có bất cứ một Object cụ thể nào có thể trả về, vậy nên ta viết thêm một class UserRating gồm List<Rating>

```
@RequestMapping("users/{userId}")
public UserRating getUsersRating(@PathVariable("userId") String userId){
    List<Rating> ratings = Arrays.asList(
        new Rating("1234", 4),
        new Rating("5678", 5)
    );
    UserRating userRating = new UserRating(ratings);
    return userRating;
}
```

Tại package model, tạo class **UserRating**:

```
public class UserRating{
    private List<Rating> userRating;
    //getter & setter
}
```

Copy class Movie, Rating, UserRating từ 2 service kia vào phần model của catalog-service



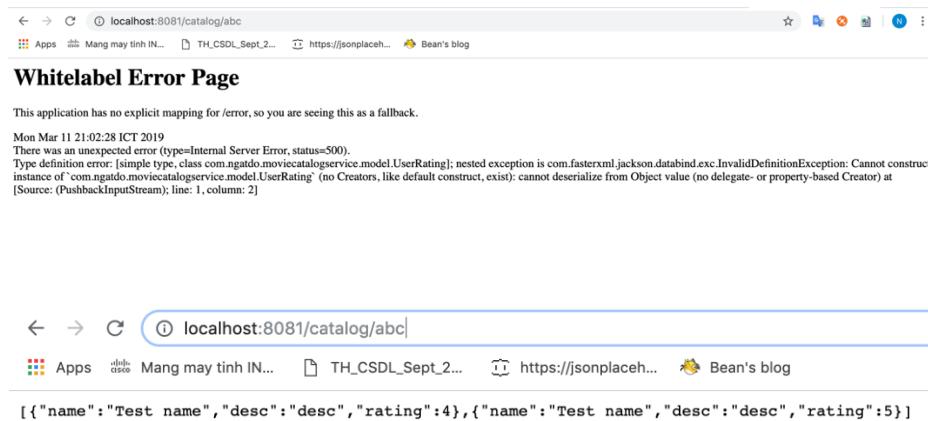
- Lợi ích của việc copy chứ không phải dùng chung 1 class là vì đây là microservice, nhiều khi class trên bị thay đổi nhưng không cần thiết đối với service hiện tại, nên ta nên dùng bản copy thay vì dùng chung 1 file

- Dùng RestTemplate để gọi 2 service còn lại cho catalog-service:

```
@RestController
@RequestMapping("/catalog")
public class MovieCatalogResource{
    @RequestMapping("/{userId}")
    public List<CatalogItem> getCatalog(@PathVariable("userId") String userId) {
        RestTemplate restTemplate = new RestTemplate();
        //get all rated movieId
        UserRating userRating = restTemplate.getForObject("http://localhost:8083/ratingsdata/users/" +
        + userId, UserRating.class);

        //for each movieId, call movie-info-service and get details(name)
        return userRating.getUserRating().stream().map(rating -> {
            Movie movie = restTemplate.getForObject("http://localhost:8082/movies/" +
            rating.getMovieId(), Movie.class);
            return new CatalogItem(movie.getName(), "desc", rating.getRating());
        }).collect(Collectors.toList());
    }
}
```

Khi run nếu thấy báo lỗi này là do thiếu constructor rỗng trong class UserRating, thêm vào ta sẽ thấy hết lỗi



Vậy là ta đã hoàn thành xong, nhưng bạn có nhìn thấy điểm gì có vấn đề?

5.1. Sử dụng Bean annotation

Mỗi khi ta gọi service catalog (hoặc đơn giản refresh lại trang) một đối tượng restTemplate được sinh ra, và điều này rất là lãng phí, chỉ nên có 1 instance và dùng nó cho nhiều lần, dùng chung cho tất cả các class.

Sử dụng **@Bean** trong Spring vì nó mặc định là singleton. Có thể khai báo Bean ở bất cứ đâu trong project, Khi dùng thêm annotation **@Autowired** thì Spring sẽ tự tìm kiếm trong project và map với một @Bean nào có cùng kiểu dữ liệu để trả về

```
@Bean
public RestTemplate getRestTemplate() {
    return new RestTemplate();
}
```

Ở class **controller**:

```
@RestController
@RequestMapping("/catalog")
public class MovieCatalogResource
{
    @Autowired
    RestTemplate restTemplate;

    @RequestMapping("/{userId}")
    public List<CatalogItem> getCatalog(@PathVariable("userId") String userId){
        //code here
    }
}
```

5.2. Sử dụng Discovery Server

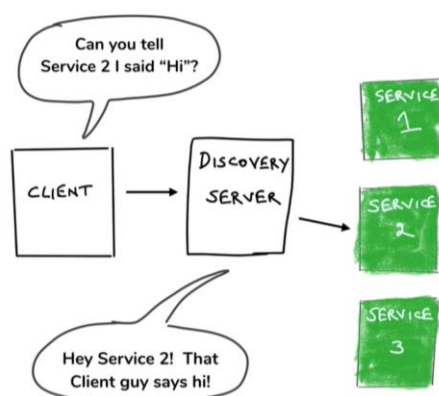
URL khi gọi đến 2 service còn lại có điều gì bất thường?

⇒ **Hard coded URLs**

- Chuyện gì sẽ xảy ra nếu chúng ta sửa lại port của một service?
- Hay là ta có dynamic URLs
- Load balancing
- Multiple Environment

⇒ Dùng Discovery server

Khi các service giao tiếp với nhau, nhiệm vụ của server này là kết nối các service khác với nhau



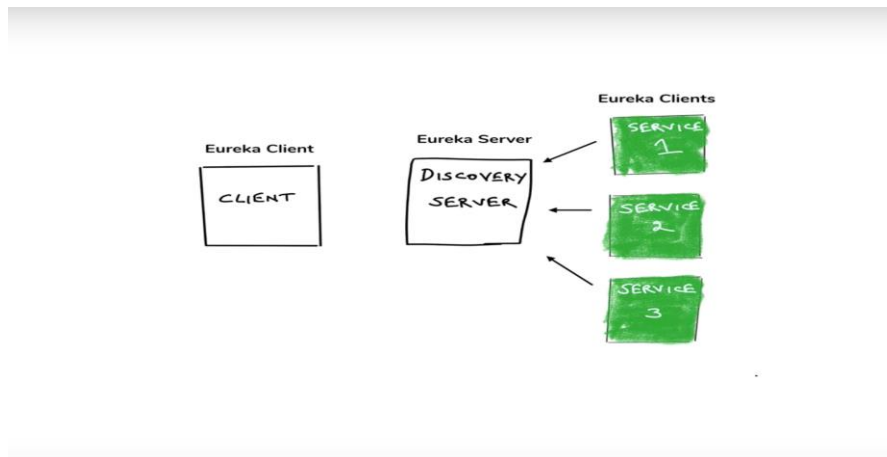
Giải pháp là dùng Eureka Server

Đây là một máy chủ dùng để quản lý, đặt tên cho các service, hay còn gọi là service registry. Nhưng tại sao chúng ta lại cần một server để đặt tên cho mỗi service?

Lý do:

- Chúng ta không muốn hardcode địa chỉ IP của mỗi microservice. Bạn chẳng bao giờ dùng địa chỉ 64.233.181.99 để truy cập vào trang google.com, đúng chứ?
- Khi mà các service của chúng ta sử dụng IP động, nó sẽ tự động cập nhật, chúng ta không cần thay đổi code.

Vậy là mỗi service sẽ được đăng ký với Eureka và sẽ ping cho Eureka để đảm bảo chúng vẫn hoạt động. Nếu Eureka server không nhận được bất kỳ thông báo nào từ service thì service đó sẽ bị gỡ khỏi Eureka một cách tự động.



6. Xây dựng Discovery Server

6.1. Tạo dự án

Tạo project Discovery Server, như 3 project trên, ở đây thêm dependency: **Eureka Server**

Thêm `@EnableEurekaServer` tại class main để Spring biết đây là Eureka Server

```
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServerApplication.class, args);
    }

}
```

Thêm dependency trong pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
```

```

    <version>2.1.3.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.ngatdo</groupId>
<artifactId>discovery-server</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>discovery-server</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>11</java.version>
    <spring-cloud.version>Greenwich.RELEASE</spring-cloud.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
    <dependency>
        <groupId>javax.xml.bind</groupId>
        <artifactId>jaxb-api</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>com.sun.xml.bind</groupId>
        <artifactId>jaxb-impl</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jaxb</groupId>
        <artifactId>jaxb-runtime</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>javax.activation</groupId>
        <artifactId>activation</artifactId>
        <version>1.1.1</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>

```



```

    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-dependencies</artifactId>
    <version>${spring-cloud.version}</version>
    <type>pom</type>
    <scope>import</scope>
  </dependency>
</dependencies>
</dependencyManagement>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
<repositories>
  <repository>
    <id>spring-milestones</id>
    <name>Spring Milestones</name>
    <url>https://repo.spring.io/milestone</url>
  </repository>
</repositories>
</project>

```

Port mặc định của Eureka là 8761, nếu không run với port này thì vào application.properties đổi lại port.

Ngoài ra vì Eureka server nhiều lúc có thể dùng như là một Eureka Client, khi chạy nó sẽ đăng kí cả chính nó nên ta cần bỏ việc này bằng cách thêm 2 dòng dưới

```

server.port = 8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

```

Sau khi run ta được kết quả:

```

Thread-12] o.s.c.n.e.server.EurekaServerBootstrap : isAws returned false
Thread-12] o.s.c.n.e.server.EurekaServerBootstrap : Initialized server context
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8761 (http) with context path ''
main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8761
main] c.n.d.DiscoveryServerApplication : Started DiscoveryServerApplication in 5.184 seconds (JVM

```

The screenshot shows the Spring Eureka web interface. At the top, there's a navigation bar with 'HOME' and 'LAST 1000 SINCE STARTUP'. Below this, the 'System Status' section contains two tables. The first table shows 'Environment' as 'test' and 'Data center' as 'default'. The second table shows 'Current time' as '2019-03-11T22:19:03 +0700', 'Uptime' as '00:00', 'Lease expiration enabled' as 'false', 'Renews threshold' as '0', and 'Renews (last min)' as '0'. Below this is the 'DS Replicas' section, which shows 'localhost'. The 'Instances currently registered with Eureka' section shows a table with columns 'Application', 'AMIs', 'Availability Zones', and 'Status', but it states 'No instances available'. Finally, the 'General Info' section shows a table with 'Name' and 'Value', listing 'total-avail-memory' as '308mh'.

6.2. Đăng kí service với Discovery Server

Trong project **Movie-catalog-service**

1. Pom.xml thêm dependency Eureka client

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

2. Thêm `@EnableEurekaClient` vào class main để Spring biết đây là Eureka client

```
@SpringBootApplication
@EnableEurekaClient
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

3. Run `movie-catalog-service` project và refresh lại trang eureka server

<http://localhost:8761/>

Phần application có thêm một app tên là UNKNOWN, địa chỉ là 192.168.1.109:8081

⇒ Đây là app `movie-catalog-service` vừa đăng kí bên trên

localhost:8761

System Status

Environment	test	Current time	2019-03-11T23:10:26 +0700
Data center	default	Uptime	00:06
		Lease expiration enabled	false
		Renews threshold	3
		Renews (last min)	0

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
UNKNOWN	n/a (1)	(1)	UP (1) - 192.168.1.109:8081

General Info

Name	Value
total-avail-memory	308mb
environment	test

Đặt tên cho service bằng cách thêm vào file application.properties như sau

```
spring.application.name = movie-catalog-service
server.port = 8081
```

Sau khi run lại ta được như sau:

localhost:8761

System Status

Environment	test	Current time	2019-03-11T23:19:29 +0700
Data center	default	Uptime	00:15
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	4

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MOVIE-CATALOG-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.109:movie-catalog-service:8081

General Info

Name	Value
------	-------

Làm tương tự với 2 service còn lại ta được

localhost:8761

Apps Mang máy tính IN... TH_CSDL_Sept_2... https://jsonplaceh... Bean's blog

System Status

Environment	test	Current time	2019-03-11T23:24:25 +0700
Data center	default	Uptime	00:20
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	4

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MOVIE-CATALOG-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.109:movie-catalog-service:8081
MOVIE-INFO-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.109:movie-info-service:8082
RATINGS-DATA-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.109:ratings-data-service:8083

General Info

6.3. Cải tiến Movie Caltalog Service lần 2

Thêm `@LoadBalanced` vào phần khai báo bean của Restemplate

⇒ Thông báo với restTemplate rằng không đi thẳng tới service trực tiếp bằng URL nhận được, đây chỉ là một hint để tìm ra service

⇒ restTemplate gọi Eureka xem có service nào tên như vậy không? Lúc này Eureka chỉ biết 3 tên service chứ không biết localhost:8082... là gì

```
@Bean
@LoadBalanced
public RestTemplate getRestTemplate()
{
    return new RestTemplate();
}
```

Đổi URL service

```
@RestController
@RequestMapping("/catalog")
public class MovieCatalogResource{
    @Autowired
    RestTemplate restTemplate;

    @RequestMapping("/{userId}")
    public List<CatalogItem> getCatalog(@PathVariable("userId") String userId){

        //get all rated movieID

        UserRating userRating = restTemplate.getForObject("http://ratings-data-
```

```

service/ratingsdata/users/" + userId, UserRating.class);

//for each movieId, call movie-info-service and get details(name)
return userRating.getUserRating().stream().map(rating -> {
    Movie movie = restTemplate.getForObject("http://movie-info-service/movies/" +
rating.getMovieId() , Movie.class);
    return new CatalogItem(movie.getName(),"desc",rating.getRating());
}).collect(Collectors.toList());

}
}

```

Sau khi run ta được kết quả y như ban đầu



← → ↻ ⓘ localhost:8081/catalog/abc

Apps Mang may tinh IN... TH_CSDL_Sept_2... https://jsonplaceh... Bean's blog

[{"name":"Test name","desc":"desc","rating":4},{ "name":"Test name","desc":"desc","rating":5}]

Kết luận: Vì tên application thường không thay đổi nên cách này sẽ tốt hơn nhiều so với việc dùng hard coded URLs

7. Gateway - Zuul

Khi chúng ta gọi đến bất kỳ service nào từ browser, chúng ta không thể gọi trực tiếp bằng tên của chúng như ở trên đã làm bởi vì những tên service như vậy phải được bí mật và chỉ sử dụng trong nội bộ các service với nhau.

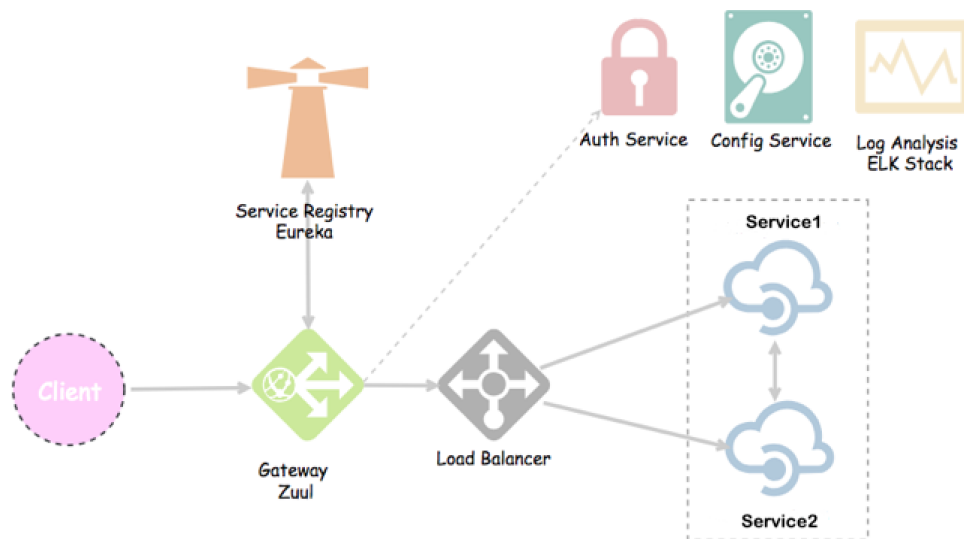
Nếu chúng ta có nhiều instance của một service, mỗi instance lại sử dụng một port khác nhau. Vậy làm thế nào chúng ta có thể gọi tất cả các service từ browser và phân tán những request đến các instance đó thông qua các cổng khác nhau? Câu trả lời là sử dụng một Gateway.

Một gateway là một entry point đơn trong hệ thống, được dùng để handle các request bằng cách định tuyến chúng đến các service tương ứng. Nó cũng có thể được dùng để xác thực, giám sát và làm nhiều việc khác.

Zuul là gì?

Nó là một proxy, gateway và một lớp trung gian giữa user và các service của bạn. Eureka server đã giải quyết vấn đề đặt tên cho từng service thay vì dùng địa chỉ IP của chúng. Tuy nhiên một service vẫn có thể có nhiều instance và chúng sẽ chạy trên các cổng khác nhau nên nhiệm vụ của Zuul sẽ là:

4. Map giữa một prefix path (/catalog/**) và một service (movie-calatog-service). Nó sử dụng Eureka server để định tuyến các service được request.
5. Nó giúp cân bằng tải giữa các instance của một service.
6. Còn gì nữa? Chúng ta có thể dùng nó để filter request, thêm xác thực,...



Điều đáng nói là Zuul hoạt động như một Eureka client. Vì vậy chúng ta có thể đặt tên cho nó, chỉ định port và đường dẫn đến Eureka server như các client trước đó.

⇒ Tạo project như các project trên

Trong pom.xml, thêm Eureka Client, Zuul Proxy

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

Trong class main, thêm @EnableEurekaClient @EnableZuulProxy

```

@SpringBootApplication
@EnableEurekaClient
@EnableZuulProxy
public class GatewayZuulApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewayZuulApplication.class, args);
    }

}

```

Trong application.properties thêm các đường dẫn tới các service

```

server.port= 8762
spring.application.name=gateway-zuul

# A prefix that can added to beginning of all requests.

#zuul.prefix=/api

# Disable accessing services using service name (i.e. catalog-service).

# They should be only accessed through the path defined below.

zuul.ignored-services=*

# Map paths to services

zuul.routes.movie-catalog-service.path=/catalog-service/**
zuul.routes.movie-catalog-service.service-id=movie-catalog-service

zuul.routes.movie-info-service.path=/info-service/**
zuul.routes.movie-info-service.service-id=movie-info-service


zuul.routes.ratings-data-service.path=/ratings-data-service/**
zuul.routes.ratings-data-service.service-id=ratings-data-service

```

Sau khi run tất cả các project, mở Eureka server ta thấy kết quả

← → ↻ ⓘ localhost:8761 ☆ 📄 📄 📄 📄 📄

Apps 📄 Mang may tinh IN... 📄 TH_CSDL_Sept_2... 📄 https://jsonplaceh... 🐼 Bean's blog

 HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2019-03-12T09:31:59 +0700
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	8
		Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY-ZUUL	n/a (1)	(1)	UP (1) - 10.20.1.54:gateway-zuul:8762
MOVIE-CATALOG-SERVICE	n/a (1)	(1)	UP (1) - 10.20.1.54:movie-catalog-service:8081
MOVIE-INFO-SERVICE	n/a (1)	(1)	UP (1) - 10.20.1.54:movie-info-service:8082
RATINGS-DATA-SERVICE	n/a (1)	(1)	UP (1) - 10.20.1.54:ratings-data-service:8083

<http://localhost:8762/catalog-service/catalog/abc>

← → ↻ ⓘ localhost:8762/catalog-service/catalog/abc

Apps 📄 Mang may tinh IN... 📄 TH_CSDL_Sept_2... 📄 https://jsonplaceh... 🐼 Bean's blog

```
[{"name": "Test name", "desc": "desc", "rating": 4}, {"name": "Test name", "desc": "desc", "rating": 5}]
```

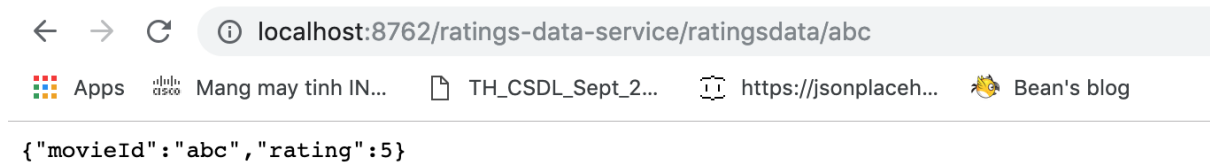
<http://localhost:8762/info-service/movies/abc>

← → ↻ ⓘ localhost:8762/info-service/movies/abc

Apps 📄 Mang may tinh IN... 📄 TH_CSDL_Sept_2... 📄 https://jsonplaceh... 🐼 Bean's blog

```
{"movieId": "abc", "name": "Test name"}
```

<http://localhost:8762/ratings-data-service/ratingsdata/abc>



Kết thúc việc dùng gateway để định tuyến các service tại đây.

8. Xác thực user bằng JWT, Spring Security

8.1 Authentication work flows

Chúng ta sẽ xem xét những bước xác thực diễn ra như thế nào trong một hệ thống bình thường:

1. Người dùng gửi một request để lấy một token thông qua một hệ thống đã nhập.
2. Server sẽ xác định danh tính của người dùng và gửi trả về một token.
3. Với mỗi request sau đó, người dùng sẽ cung cấp token nhận được để server xác nhận.

Chúng ta sẽ sử dụng một service mới là auth service để xác nhận danh tính của user và cung cấp token. Vậy còn xác nhận token thì ai đảm nhận? Thực ra chúng ta có thể triển khai việc này trong auth service và cổng gateway của chúng ta sẽ gọi đến service này để xác nhận token trước khi cho phép request gọi đến bất kỳ service nào. Tuy nhiên trong bài viết này, chúng ta sẽ thực hiện việc xác nhận token ở tầng gateway luôn, và nhiệm vụ của auth service chỉ là xác nhận danh tính user sau đó cung cấp một token tương ứng. Dù sử dụng cách nào thì việc chúng ta làm đều có mục đích là chặn các request chưa được xác thực gọi đến các service.

8.2. Json Web Token (JWT)

Một token là một chuỗi String được mã hóa, được gen từ ứng dụng của chúng ta (sau khi xác thực thành công) và được gửi bởi người dùng trong mỗi request để cung cấp quyền truy cập vào các tài nguyên cho phép của ứng dụng. JSON Based Token (JWT) là một chuẩn JSON để tạo token. Nó bao gồm ba phần: header, payload và chữ ký (signature).

Phần header sẽ chứa thuật toán hash:

```
{type: "JWT", hash: "HS256"}
```

Phần payload sẽ chứa các thuộc tính (username, email, ...) và giá trị của nó:

```
{username: "Omar", email: "omar@example.com", admin: true }
```

Phần chữ ký là giá trị hash của Header + "." + Payload + Secret key.

Ví dụ một mã token:

Bearer

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0dWJlYW4iLCJhdXRob3JpdGllcyI6WyJST0xFX1VTRVliXSwaWF0IjoxNTUyNjY1NTEyLCJleHAiOiJlNTI3NTE5MTJ9.AFqQgfp4tHBVzjt2mzgl30biaaXLWbBF_VGfNqjGJZUasWbjtloqmladgEmzlZQgeGutdC6bAde33q9uq6mNgA

8.3. Gateway

Ở trong gateway, chúng ta sẽ cần thêm hai phần: một là xác nhận token cho mỗi request và hai là chặn tất cả các request không được xác thực.

Thêm dependency spring security, JWT, jaxb

Pom.xml:

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.0</version>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.0</version>
</dependency>
```

Trong file **application.properties** chúng ta khai báo đường dẫn path đến auth service (sẽ tạo sau):

```
zuul.routes.auth-service.path=/auth/**
zuul.routes.auth-service.service-id=auth-service
```

Để kích hoạt **Spring Security**, trước tiên ta cần phải viết một lớp kế thừa **abstract class WebSecurityConfigurerAdapter**: (đọc cmt trong code để hiểu hơn)

```
@EnableWebSecurity // Enable security config. This annotation denotes config for spring security.
public class SecurityTokenConfig extends WebSecurityConfigurerAdapter {
    @Autowired
```

```

private JwtConfig jwtConfig;

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        // make sure we use stateless session; session won't be used to store user's state.
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
        // handle an authorized attempts
        .exceptionHandling().authenticationEntryPoint((req, rsp, e) ->
            rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED))
        .and()
        // Add a filter to validate the tokens with every request
        .addFilterAfter(new JwtTokenAuthenticationFilter(jwtConfig), UsernamePasswordAuthenticationFilter.class)
        // authorization requests config
        .authorizeRequests()
        // allow all who are accessing "auth" service
        .antMatchers(HttpMethod.POST, jwtConfig.getUri()).permitAll()
        // must be an admin if trying to access admin area (authentication is also required here)
        .antMatchers("/catalog-service/catalog/{userId}/**").hasRole("ADMIN")
        // Any other request must be authenticated
        .anyRequest().authenticated();
}

@Bean
public JwtConfig jwtConfig() {
    return new JwtConfig();
}
}

```

Trong class **JwtConfig** chúng ta khai báo các biến để cấu hình JWT:

```

public class JwtConfig {
    @Value("${security.jwt.uri:/auth/**}")
    private String uri;

    @Value("${security.jwt.header:Authorization}")
    private String header;

    @Value("${security.jwt.prefix:Bearer }")
    private String prefix;

    @Value("${security.jwt.expiration:#{24*60*60}}")
    private int expiration;

    @Value("${security.jwt.secret:JwtSecretKey}")
    private String secret;
}

```

Spring có các lớp filter cho phép thực thi trong life-cycle của một request. Để khởi động và sử dụng các filter này, chúng ta cần extend chúng. Ở chế độ mặc định thì spring sẽ định nghĩa riêng

cho mỗi filter lúc nào thì nên sử dụng. Tuy nhiên thì bạn có thể sửa lại thứ tự một filter sẽ hoạt động trước hoặc sau filter nào.

Công việc tiếp theo là chúng ta cần triển khai một filter dùng để xác nhận token.

Filter **OncePerRequestFilter** là một filter được kích hoạt cho mỗi một lần request đến nên chúng ta sẽ dùng nó để tạo một filter của chúng ta:

```
public class JwtTokenAuthenticationFilter extends OncePerRequestFilter {

    private final JwtConfig jwtConfig;

    public JwtTokenAuthenticationFilter(JwtConfig jwtConfig) {
        this.jwtConfig = jwtConfig;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
        throws ServletException, IOException {

        // 1. get the authentication header. Tokens are supposed to be passed in the authentication header
        String header = request.getHeader(jwtConfig.getHeader());

        // 2. validate the header and check the prefix
        if(header == null || !header.startsWith(jwtConfig.getPrefix())) {
            chain.doFilter(request, response);    // If not valid, go to the next filter.
            return;
        }

        // If there is no token provided and hence the user won't be authenticated.
        // It's Ok. Maybe the user accessing a public path or asking for a token.

        // All secured paths that needs a token are already defined and secured in config class.
        // And If user tried to access without access token, then he won't be authenticated and an exception will be
        // thrown.

        // 3. Get the token
        String token = header.replace(jwtConfig.getPrefix(), "");

        try { // exceptions might be thrown in creating the claims if for example the token is expired

            // 4. Validate the token
            Claims claims = Jwts.parser()
                .setSigningKey(jwtConfig.getSecret().getBytes())
                .parseClaimsJws(token)
                .getBody();

            String username = claims.getSubject();
            if(username != null) {
                @SuppressWarnings("unchecked")
                List<String> authorities = (List<String>) claims.get("authorities");

                // 5. Create auth object
                // UsernamePasswordAuthenticationToken: A built-in object, used by spring to represent the current
                // authenticated / being authenticated user.
                // It needs a list of authorities, which has type of GrantedAuthority interface, where
```

```

SimpleGrantedAuthority is an implementation of that interface
    UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(
        username, null,
authorities.stream().map(SimpleGrantedAuthority::new).collect(Collectors.toList()));

    // 6. Authenticate the user
    // Now, user is authenticated
    SecurityContextHolder.getContext().setAuthentication(auth);
}

} catch (Exception e) {
    // In case of failure. Make sure it's clear; so guarantee user won't be authenticated
    SecurityContextHolder.clearContext();
}

// go to the next filter in the filter chain
chain.doFilter(request, response);
}
}

```

8.4. AuthService

Trong auth service, chúng ta sẽ làm hai việc: một là xác thực định danh mà người dùng cung cấp và hai là gen ra một token trong trường hợp xác thực hợp lệ hoặc trả về một exception nếu nó không hợp lệ.

Trong file pom.xml chúng ta cần các dependencies sau: Web, Eureka Client, Spring Security và JWT.

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>

```

Giống như cấu hình cổng Gateway, chúng ta cần tạo một lớp extends từ WebSecurityConfigurerAdapter và đánh anotation @EnableWebSecurity cho nó:

```

@EnableWebSecurity // Enable security config. This annotation denotes config for spring security.
public class SecurityCredentialsConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private JwtConfig jwtConfig;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            // make sure we use stateless session; session won't be used to store user's state.
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            // handle an authorized attempts
            .exceptionHandling().authenticationEntryPoint((req, rsp, e) ->
                rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED))
            .and()
            // Add a filter to validate user credentials and add token in the response header

            // What's the authenticationManager()?
            // An object provided by WebSecurityConfigurerAdapter, used to authenticate the user passing user's
            // credentials
            // The filter needs this auth manager to authenticate the user.
            .addFilter(new JwtUsernameAndPasswordAuthenticationFilter(authenticationManager(), jwtConfig))
            .authorizeRequests()
            // allow all POST requests
            .antMatchers(HttpMethod.POST, jwtConfig.getUri()).permitAll()
            // any other requests must be authenticated
            .anyRequest().authenticated();
    }

    // Spring has UserDetailsService interface, which can be overridden to provide our implementation for fetching
    // user from database (or any other source).
    // The UserDetailsService object is used by the auth manager to load the user from database.
    // In addition, we need to define the password encoder also. So, auth manager can compare and verify
    // passwords.

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    @Bean
    public JwtConfig jwtConfig() {
        return new JwtConfig();
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

```

Trong đoạn code trên, chúng ta sử dụng interface `UserDetailsService` nên chúng ta cần implement nó:

```
@Service // It has to be annotated with @Service.
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private BCryptPasswordEncoder encoder;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        // hard coding the users. All passwords must be encoded.
        final List<AppUser> users = Arrays.asList(
            new AppUser(1, "ngatdo", encoder.encode("12345"), "USER"),
            new AppUser(2, "admin", encoder.encode("12345"), "ADMIN")
        );

        for(AppUser appUser: users) {
            if(appUser.getUsername().equals(username)) {

                // Remember that Spring needs roles to be in this format: "ROLE_" + userRole (i.e. "ROLE_ADMIN")
                // So, we need to set it to that format, so we can verify and compare roles (i.e. hasRole("ADMIN")).
                List<GrantedAuthority> grantedAuthorities = AuthorityUtils
                    .commaSeparatedStringToAuthorityList("ROLE_" + appUser.getRole());

                // The "User" class is provided by Spring and represents a model class for user to be returned by
                // UserDetailsService
                // And used by auth manager to verify and check user authentication.
                return new User(appUser.getUsername(), appUser.getPassword(), grantedAuthorities);
            }
        }

        // If user not found. Throw this exception.
        throw new UsernameNotFoundException("Username: " + username + " not found");
    }

    // A (temporary) class represent the user saved in the database.
    private static class AppUser {
        private Integer id;
        private String username, password;
        private String role;

        public AppUser(Integer id, String username, String password, String role) {
            this.id = id;
            this.username = username;
            this.password = password;
            this.role = role;
        }

        //getter & setter
    }
}
```

Lớp **UserDetailsServiceImpl** sẽ phải ghi đè phương thức **loadByUsername**(String username) của **UserDetailsService**.

Tham số **username** ở đây ta cần phải hiểu là email mà người dùng nhập vào. Mình sẽ tìm kiếm trong CSDL user thỏa mãn username này. Nếu như không tìm thấy, sẽ throw new **UsernameNotFoundException**("User not found");

Phương thức này sẽ trả về một **org.springframework.security.core.userdetails.User** - là một implementation của **UserDetails**. Hàm khởi tạo của lớp này sẽ nhận 3 tham số: username, password và grantedAuthorities của user. Cách tìm **grantedAuthorities** các bạn có thể xem trong đoạn code trên.

Đây là class hoạt động như một nơi cung cấp cho người dùng, nghĩa là nó tải các thông tin chi tiết của người dùng từ database lên và sử dụng chúng.

Bước tiếp theo cũng là bước cuối cùng, chúng ta cần một filter. Ở đây chúng ta sử dụng **JwtUsernameAndPasswordAuthenticationFilter** để xác thực định danh người dùng và tạo token. Các thông tin về username hay password phải được gửi dưới dạng POST request.

```
public class JwtUsernameAndPasswordAuthenticationFilter extends UsernamePasswordAuthenticationFilter {

    // We use auth manager to validate the user credentials
    private AuthenticationManager authManager;

    private final JwtConfig jwtConfig;

    public JwtUsernameAndPasswordAuthenticationFilter(AuthenticationManager authManager, JwtConfig
jwtConfig) {
        this.authManager = authManager;
        this.jwtConfig = jwtConfig;

        // By default, UsernamePasswordAuthenticationFilter listens to "/login" path.
        // In our case, we use "/auth". So, we need to override the defaults.
        this.setRequiresAuthenticationRequestMatcher(new AntPathRequestMatcher(jwtConfig.getUri(), "POST"));
    }

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
        throws AuthenticationException {

        try {

            // 1. Get credentials from request
            UserCredentials creds = new ObjectMapper().readValue(request.getInputStream(), UserCredentials.class);

            // 2. Create auth object (contains credentials) which will be used by auth manager
            UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                creds.getUsername(), creds.getPassword(), Collections.emptyList());

            // 3. Authentication manager authenticate the user, and use
            // UserDetailsServiceImpl::loadUserByUsername() method to load the user.
            return authManager.authenticate(authToken);
        }
    }
}
```



```

    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}

// Upon successful authentication, generate a token.
// The 'auth' passed to successfulAuthentication() is the current authenticated user.
@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
FilterChain chain,
Authentication auth) throws IOException, ServletException {

    Long now = System.currentTimeMillis();
    String token = Jwts.builder()
        .setSubject(auth.getName())
        // Convert to list of strings.
        // This is important because it affects the way we get them back in the Gateway.
        .claim("authorities", auth.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority).collect(Collectors.toList()))
        .setIssuedAt(new Date(now))
        .setExpiration(new Date(now + jwtConfig.getExpiration() * 1000)) // in milliseconds
        .signWith(SignatureAlgorithm.HS512, jwtConfig.getSecret().getBytes())
        .compact();

    // Add token to header
    response.addHeader(jwtConfig.getHeader(), jwtConfig.getPrefix() + token);
}

// A (temporary) class just to represent the user credentials
private static class UserCredentials {
    private String username, password;
    //getter & setter
}
}

```

Đừng quên class *JwtConfig* như bên Gateway!

8.5. Testing

Chúng ta sẽ chạy lần lượt Eureka Server, các service info, rating, catalog, auth và gateway.

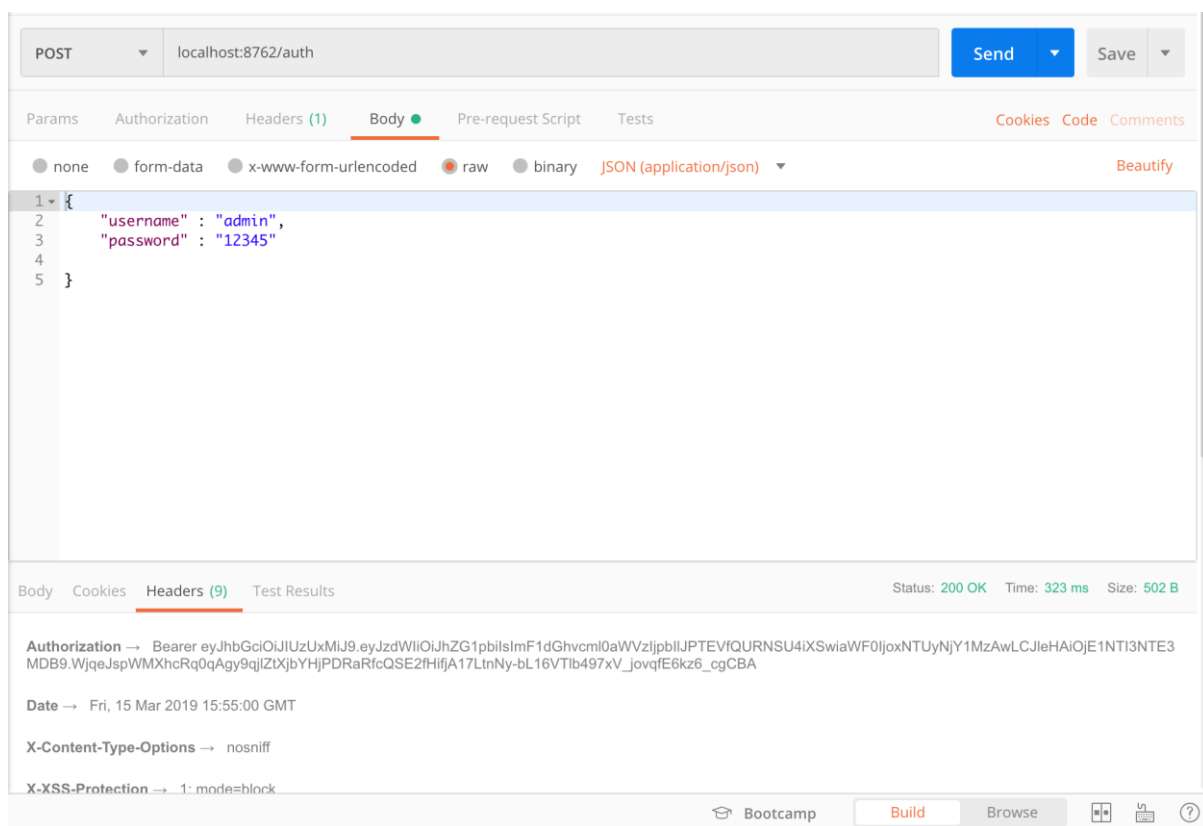
Đầu tiên chúng ta thử truy cập vào gallery service thông qua đường dẫn localhost:8762/catalog-service/catalog/abc mà không có token. Chúng ta sẽ nhận về một lỗi 401 như sau:

```

1 {
2   "timestamp": "2019-03-15T16:22:21.870+0000",
3   "status": 401,
4   "error": "Unauthorized",
5   "message": "No message available",
6   "path": "/catalog-service/catalog/abc"
7 }

```

Để nhận token, chúng ta gửi định danh của chúng ta đến địa chỉ localhost:8762/auth như sau:



Bây giờ gọi lại đến catalog service kèm theo một token trong header:

GET localhost:8762/catalog-service/catalog/abc

Authorization: Bearer Token

Token: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWUiOiJhZG1pbGlzImF1dGhvcml0aWVz...

Status: 200 OK Time: 52 ms Size: 405 B

```
1 [
2   {
3     "name": "Test name",
4     "desc": "desc",
5     "rating": 4
6   },
7   {
8     "name": "Test name",
9     "desc": "desc",
10    "rating": 5
11  }
12 ]
```

Thử dùng user không phải quyền ADMIN, sẽ nhận được lỗi sau:

GET localhost:8762/catalog-service/catalog/abc

Authorization: Bearer Token

Token: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWUiOiJhZG1pbGlzImF1dGhvcml0aWVz...

Status: 403 Forbidden Time: 57 ms Size: 456 B

```
1 {
2   "timestamp": "2019-03-15T15:59:06.163+0000",
3   "status": 403,
4   "error": "Forbidden",
5   "message": "Forbidden",
6   "path": "/catalog-service/catalog/abc"
7 }
```

Nguồn:
Webservice:

<https://viblo.asia/p/web-service-ban-se-chon-rest-hay-soap-ByEZkWyAZQ0>

<https://viblo.asia/p/xay-dung-restful-web-service-trong-java-voi-dropwizard-framework-bJzKm0xX59N>

<http://viettuts.vn/web-service/web-service-la-gi>

https://www.youtube.com/watch?v=mr_2-AWYCoc

Microservice:

<https://viblo.asia/p/microservice-tu-ban-giay-den-trien-khai-phan-1-microservice-la-cai-gi-Az45bx6QZxY>

<https://www.youtube.com/playlist?list=PLqq-6Pq4lTTZSKAFG6aCDVDP86Qx4lNas>

<https://tubean.github.io/2018/12/microservice-springboot-eureka/>

<https://kipalog.com/posts/Huong-dan-lap-trinh-Spring-Security>

-----HẾT-----