Laborator 14: RESTful Webservices

1. Porniți aplicația lab14 și testați documentația swagger:

```
http://localhost:8080/v2/api-docs

http://localhost:8080/swagger-ui.html
```

2. Modificați documentația swagger generată astfel încât info.description să conțină textul “Api lab 14 documentation”, info.tile: “Api lab 14”. Adăugați de asemenea infomații de contact. În clasa SwaggerConfig adăugați constantele:

```
private static final Collection<VendorExtension> VENDOR_EXTENSION
= new LinkedList();

private static final Contact CONTACT = new
Contact("apbdoo","http://apbdoo","lab14@apbdoo");

public static final ApiInfo API_INFO =
     new ApiInfo("API Lab 14 documentation",
          "API lab 14", "1.0", "urn:tos",
          CONTACT, "Apache 2.0",
          "http://www.apache.org/licenses/LICENSE-2.0",
          VENDOR_EXTENSION );

@Bean
public Docket api(){
       return new Docket(DocumentationType.SWAGGER_2)
                 .apiInfo(API_INFO);
}
```

3. Adăugați dependența:

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.8</version>
</dependency>
```

4. În postman testați requestul pentru a obține lista tuturor utilizatorilor. Adăugți un header pentru a specifica tipul acceptat de răspuns, xml.

```
http://localhost:8080/users
[{"key":"Accept","value":"application/xml","description":""}]
```

5. Adăugați în SwaggerConfig tipul xml ca tip de răspuns acceptat:

```
private static final List<String> ACCEPT_TYPES =
            Arrays.asList("application/xml", "application/json");
    private static final Set<String> PRODUCES =
            new HashSet<String>(ACCEPT_TYPES);

return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(API_INFO)
                .produces(PRODUCES);
```

6. Dați o descrierea suplimentară definiției modelului User. Adăugați informații despre validări:

```
@ApiModel("User details")
public class User {
…
    @ApiModelProperty(notes= "max 20 ch")
    @Size(max=20, message = "max 20 ch.")
    private String name;
```

Microservices
-- REST Services
-- Deployable units
-- Clound Enabled: Pentru fiecare microserviciu pot exista mai multe instanțe create dinamic.

Arhitectură scalabilă dinamic care se compune din mai multe (micro)servicii executate independent, fiecare de către un proces, și care comunică (interacționează) de regulă prin http.
Se pot folosi tehnologii diferite în cadrul aceleași arhitecturi, spre exemplu fiecare microserviciu poate fi implementat într-un alt limbaj de programare.

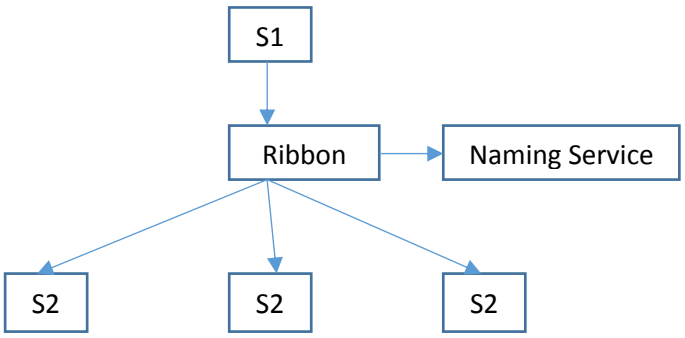
Context: clarificare funcționalități
Configurare: simplificarea configurări instnțe multiple
Dynamic Scaling instanțe up/down, dynamic load balancing, distribute load
Visibility: monitorizare servicii/servere.
Dependencies: fault tolerance

Spring Cloud
https://spring.io/projects/spring-cloud

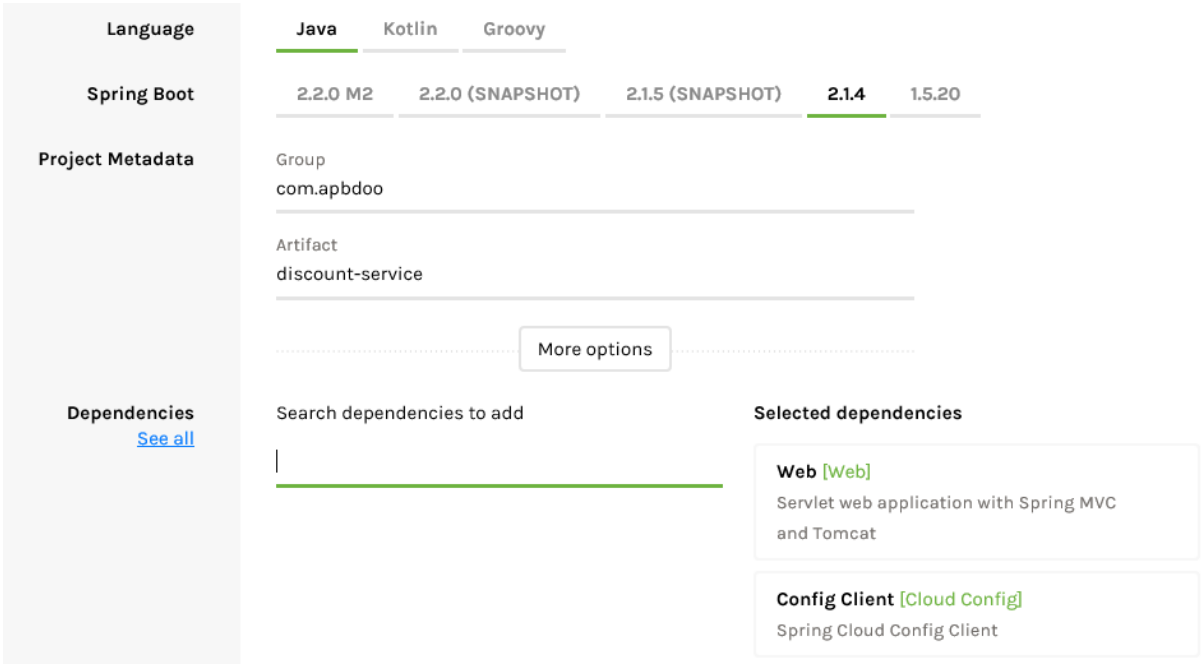
Grupează proiecte care pun la dispoziție instrumente pentru aplicații distribuite.

Configurare: Spring Cloud Config Server**,** configurare centralizată
Dynamic Scale: Naming Server **Eureka** utilizat pentru *înregistrarea microserviciilor* și *service-discovery*.
Ribbon client side Load Balancing
Feign (REST Clients)

Visibility: monitorizare servicii/servere **Zipkin Distributed Tracing Server**.
Dependencies: fault tolerance **Hystrix** defaul behaviour in case of failure.



7. Folosind start.spring.io creați un proiect cu Artifact Id: discount-service. Adăugați dependențele: Web și ConfigClient



8. Adăugți două proprități în fișierul application.properties și configurați numele aplicației:

```
spring.application.name=discount-service

discount-service.month=10
discount-service.year=20
```

9. Creați în pachetul config, clasa PropertiesConfiguration:

```
@Component
@ConfigurationProperties("discount-service")
@Getter
@Setter
public class PropertiesConfiguration {
    private int month;
    private int year;
}
```

10. În pachetul model creați clasa Discount:

```
@Setter
@Getter
@AllArgsConstructor
public class Discount {
    private int month;
    private int year;
}
```

11. În pachetul controllers adăugați clasa Controller

```
@RestController
public class Controller {
    @Autowired
    private PropertiesConfiguration configuration;

    @GetMapping("/discount")
    public Discount getDiscount(){
        return new
Discount(configuration.getMonth(),configuration.getYear());
    }
}
```

12. Folosind start.spring.io creați un proiect cu Artifact Id: config-server. Adăugați dependența ConfigServer. Completați în fișierul application.properties numele aplicației și portul pe care va rula aceasta.

```
spring.application.name=config-server
server.port = 8800
```

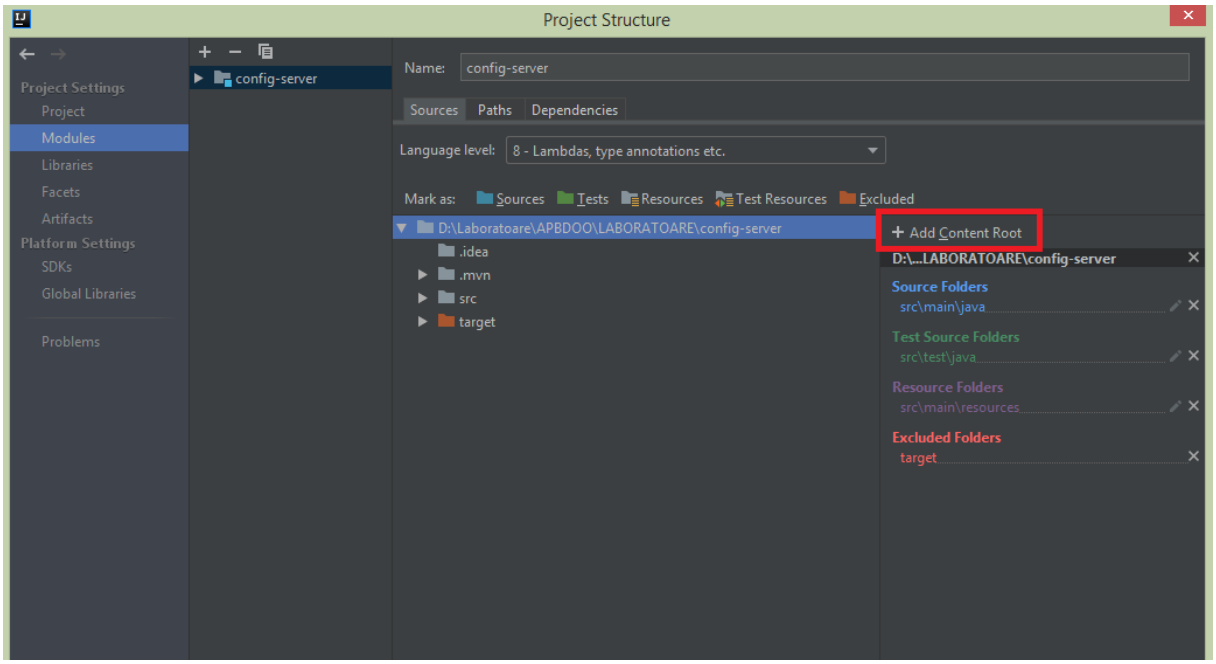
13. Instalați git:

```
https://git-scm.com/downloads
```

14. Creați un repository local.

```
>>D:\Laboratoare\APBDOO\GIT>mkdir local-rep
>>D:\Laboratoare\APBDOO\GIT>cd local-rep
>>D:\Laboratoare\APBDOO\GIT\local-rep>git init
```

15. În File - Project Structure adăugați directorul git:



16. Adăugați în directorul git fișierul discount-service.properties:

```
spring.application.name=discount-service

discount-service.month=30
discount-service.year=50
```

```
>>D:\Laboratoare\APBDOO\GIT\local-rep>git add -A
>>D:\Laboratoare\APBDOO\GIT\local-rep>git config --global user.username
tiulia
>>D:\Laboratoare\APBDOO\GIT\local-rep>git commit -m "config file"
[master (root-commit) e331f2c] config file
 1 file changed, 5 insertions(+)
 create mode 100644 discount-service.properties
```

17. Adnotați clasa ConfigServerApplication:

```
@EnableConfigServer
@SpringBootApplication
public class ConfigServerApplication {

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

18. Testați configurarea serviciului discount-service:

```
http://localhost:8800/discount-service/default
```

19. Redenumiți fișierul application.properties al proiectului discount-service (bootstrap.properties):

```
spring.application.name=discount-service
spring.cloud.config.uri=http://localhost:8800
spring.profiles.active=default

http://localhost:8080/discount
```