Universiteti i Shkodrës "Luigj Gurakuqi"
Bul. Shk. Ser. Nat., 2019. Nr. 69: 20 - 30

Projektimi dhe zhvillimi i një microservice në Spring MVC

Sidita Duli[1], Fatjona Kroni[1]

[1]Universiteti i Shkodrës "Luigj Gurakuqi", Departamenti i Matematikës

PËRMBLEDHJE

Në vitet e fundit, arkitektura e microservice është bërë një komponent kryesor për zhvillimin e aplikacioneve të vendosura në Cloud. Migrimi i aplikacioneve web ekzistuese në sistem microservice ofron mjaft avantazhe dhe lehtësi.
Ky artikull përfshin një pasqyrë të përgjithshme të Spring MVC. Do të paraqesim se si një microservice u mundëson bizneseve të shkëmbejnë të dhëna mes platformave të ndryshme. Nëpërmjet një implementimi konkret do të tregohet se si të dhënat e ruajtura në format JSON, në kod të hapur, ndihmojnë në rritjen e performancës të shërbimeve të bizneseve që do t'i përdorin këto të dhëna.

Fjalë kyçe: microservice, Spring MVC, JSON, Java

Design and development of a Spring MVC microservice

ABSTRACT

Recently, the architecture of the microservices has become an important factor in building web applications deployed in the Cloud. Migrating the existing web applications to microservices offers many advantages. This article includes a general overview of Spring MVC. It presents how the microservice enables businesses to exchange data between different platforms. Through a concrete implementation, it will be shown how data stored in JSON form, in open source, help increase the performance of business services that will use this data.

Key words: microservice, Spring MVC, JSON, Java

Hyrje

Një microservice është një shërbim elementar, autonom, i cili funksionon në bashkëpunim me shërbime të tjera. Një microservice mund të programohet si një shërbim i vetëm në një Platform as a Service (PAAS), ose mund të jetë si një proces i pavarur. Microservice mund të ndryshohen pavarësisht nga shërbimet e tjera me të cilët bashkëpunon (NEWMAN 2015). Microservice bën të mundur që sisteme software komplekse të ndërtohen në bazë të komponentëve bashkëpunues. Ata zakonisht paketohen në kontenier (container) (MERKEL 2014). Një kontenier përfshin microservice bashkë me libraritë, bazën e të dhënave etj., në një entitet unik, i cili mund të shpërndahet në çdo platformë që mbështet teknologjinë e kontenierit. Portabiliteti i ofruar nga kontenieri e bën më të thjeshtë rivendosjen e microservice në platforma të ndryshme. Arkitektura e microservice është ideale për shkallëzimin horizontal të sistemeve, sepse microservice-t mund të rivendosen në hostime të reja (DRAGONI et al. 2017). Në (SHARMA & GONZALEZ, 2017) dhe (SELMADJI et al. 2018) përmenden këto karakteristika kryesore të microservice:

- Fokusimi në një funksionalitet të vetëm: një microservice është përgjegjës vetëm për një funksionalitet të thjeshtë.
- Autonomia: microservice janë entitete të ndara. Ata mund të komunikojnë me njëri-tjetrin nëpërmjet thirrjeve në rrjet. Secili prej microservice administron bazën e vet të të dhënave.
- Teknologji neutrale: një sistem i përbërë nga disa microservice mund të jetë heterogjen, përsa i përket teknologjisë që microservice përdorin.
- Shpërndarje automatike: në rastin kur numri i microservice në sistem është rritur mjaftueshëm, shpërndarja e tyre duhet të bëhet në mënyrë automatike.

Në vijimësi të aplikacionit të analizuar (DULI, 2018), do të bëhet kalim në teknologjinë e Spring MVC, i një Web Service. Projekti i implementuar është një microservice mbi të dhënat meteorologjike, nga matjet e shërbimit meteorologjik të Universitetit të Shkodrës. Ky shërbim shpërndahet në platformë Cloud.

Metoda

Implementimi i një microservice i cili do ofrojë të dhënat në format JSON, për shërbimin meteorologjik pranë Universitetit të Shkodrës mund të realizohet me bazë teknologjinë Spring MVC (Model View Controller), në gjuhën e programimit Java. Aplikacionet më të fundit në Java duhet të përdorin Java 8, me të gjitha cilësitë që ajo ofron, si lambdas, veprimet paralele dhe rrjedhën e të dhënave. Mund të krijohen microservice

funksionale edhe me libraritë e mëparshme në Java, por natyrisht rekomandohet që microservice-t të ndërtohen në bazë të librarive më të fundit në Java 8 (STANLEY, 2016).
Një nga metodologjitë e përdorura për krijimin e microservice është REpresentational State Transfer (REST). REST është një arkitekturë e bazuar në Web (NEWMAN 2015). REST përfshin mjaft teknika, por do të fokusohemi në mjetet e integrimit të microservice. Vetë REST nuk fokusohet në përcaktimin e protokolleve mbi të cilat ajo bazohet, por më së shumti përdor protokollin HTTP. Kjo metodë implementimi e shërbimeve në Web përdoret nga shumë portale të njohura, si Google, Yahoo (RUBIO et al. 2014).
Në këtë punim, aplikacioni në Spring do përdorë REST, për të ofruar dhe për të marrë akses ndaj të dhënave të disponueshme me kod të hapur të ruajtur në format JSON.
Klasa në Java 8 do jetë MappingJackson2JsonView, e cila formon pjesën e Spring, për publikimin e përmbajtjes së të dhënave në JSON.
Nëse aplikacioni në Spring përfshin instruksione në Ajax, atëherë microservice në REST do të ofrojë të dhënat detyrimisht në formatin JSON. Ky kusht ndodh për shkak të mundësive përpunuese të limituara të shfletuesve Web. Edhe pse shfletuesit mund të procesojnë dhe të marrin të dhënat në XML nga microservice në REST, kjo metodë nuk është shumë eficiente. Përkundrazi, formati JSON bazohet në një gjuhë të cilën shfletuesit e kanë të përfshirë interpretuesin në JavaScript. Kjo është arsyeja pse procesimi dhe marrja e të dhënave në formatin JSON është më eficient (RUBIO et al. 2014).
Platforma Spring MVC, si shumë platforma të tjera MVC, bazohet në kërkesa që i drejtohen një servlet qendror i quajtur DispatcherServlet. Ky servlet ia përcjell këto kërkesa për tek kontrollerat, duke i ofruar kështu funksionalitete që thjeshtojnë zhvillimin e aplikacioneve web.

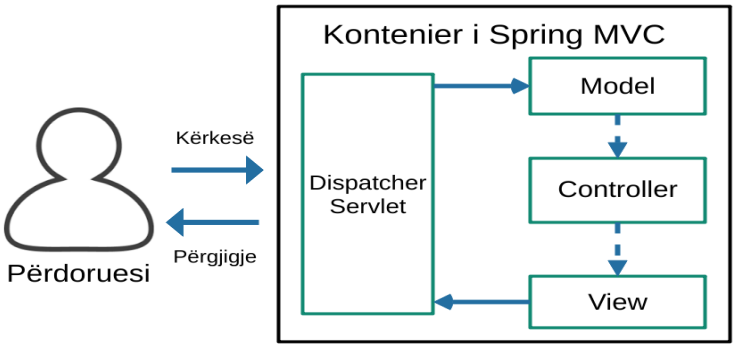

Figura 2: Skema e Microservice në Spring MVC.

DispatcherServlet implementon një nga kontrollerat model të JavaEE, i quajtur FrontController. DispatcherServlet ka rolin e një kontrolleri frontal të Spring MVC. Çdo kërkesë web kalon nëpër këtë kontroller, i cili gjithashtu administron procesin e trajtimit të kërkesave web (LAYKA 2014).

Rezultate dhe diskutim

DispatcherServlet është një klasë e cila ofron funksionalitete të marrjes të kërkesave nga përdoruesi dhe kalimin e tyre për tek kontenieri Spring MVC. Ky funksionalitet duhet të pasqyrohet edhe tek skedari web.xml. Detyra kryesore e këtij Servlet është kalimi i kërkesës tek kontrolleri i duhur dhe kthimi i përgjigjes, pasi pjesa View e modulit Model-View-Controller ka plotësuar faqen web për përgjigje.
Skedari pom.xml do përmbajë varësitë e spring-web, spring-webmvc, servlet-api, jsp-api dhe jstl.
Ky skedar implementohet si më poshtë:

```
<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>

      <groupId>com.siditaduli.spring.mvc</groupId>

      <artifactId>shembull-artikull-FSHN</artifactId>

      <version>0.0.1-SNAPSHOT</version>

      <packaging>war</packaging>

      <name>Shembull Buletini FSHN UNISHK</name>

      <description>Shembull Demostrativ</description>

      <dependencies>

            <dependency>

                  <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-webmvc</artifactId>
        <version>4.3.9.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>4.3.9.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
```

```
            </dependency>
      </dependencies>
      <build>
            <plugins>
                  <plugin>
                        <artifactId>maven-compiler-
plugin</artifactId>
                        <version>3.6.1</version>
                        <configuration>
                              <source>1.8</source>
                              <target>1.8</target>
                        </configuration>
                  </plugin>
                  <plugin>
                        <artifactId>maven-war-
plugin</artifactId>
                        <version>3.0.0</version>
                        <configuration>
<warSourceDirectory>WebContent</warSourceDirectory>
                        </configuration>
                  </plugin>
            </plugins>
            <finalName>${project.artifactId}</finalName>
      </build>
```

```
</project>
```

Kodi 1: Implementimi i pom.xml

Project Object Model (POM) është një skedar XML i rëndësishëm i një projekti në Maven. Ky skedar është i vendosur në direktorinë e projektit, nën emërtimin pom.xml. Skedari pom.xml përmban informacion rreth projektit dhe konfigurimeve të Maven. Skedari pom.xml gjithashtu përmban konfigurime të plug-ins. Sapo nis të ekzekutohet aplikacioni, do të lexohet skedari pom.xml dhe do të interpretohen konfigurimet e tij. Në këtë skedar ndodhen edhe informacione, të tilla si: versioni i projektit, përshkrimi i projektit dhe zhvilluesi i projektit. Pjesë e këtij projekti është implementimi i Spring Bean, me skedarin konfigurues spring-servlet.xml. Ky implementim është si më poshtë:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans:beans
xmlns="http://www.springframework.org/schema/mvc"

      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:beans="http://www.springframework.org/schema/bean
s"

      xmlns:context="http://www.springframework.org/sch
ema/context"

      xsi:schemaLocation="http://www.springframework.or
g/schema/mvc
http://www.springframework.org/schema/mvc/spring-
mvc.xsd

           http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd


      http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-
context.xsd">
```

```
	<annotation-driven />

	<context:component-scan
base-package="com.siditaduli.spring" />

	<beans:bean
class="org.springframework.web.servlet.view.InternalRes
ourceViewResolver">

<beans:property name="prefix" value="/WEB-INF/views/" />

	<beans:property name="suffix" value=".jsp" />

	</beans:bean>

</beans:beans>
```

Kodi 2: Implementimi i spring-servlet.xml

Skedari spring-servlet.xml krijohet si skedar i llojit Bean, duke mbivendosur përcaktimet dhe emërtimet nga çdo skedar tjetër global Bean me të njëtin emërtim. Tag-u <context:component-scan...> përdoret për të aktivizuar shënimin e Spring MVC, e cila lejon të shënohen @Controller dhe @RequestMapping, si dhe shënime tjera të kësaj platforme.
View e importuar InternalResourceViewResolver përmban rregulla të emërtimit të view të projektit. Në këtë projekt, një view e SpringMVC është e vendosur në rrugëkalimin /WEB-INF/jsp/shembullViewFshn.jsp .

Në vijim implementohet DispatcherServlet, i cili është një front Controller.
Disa nga funksionalitetet e DispatcherServlet janë:

- merr të gjitha kërkesat nga Fron Controller dhe mundëson një pikë hyrëse për tek aplikacioni web,

- drejton kërkesat e përdoruesit për tek kontrolleri i duhur në Spring MVC,
- konsultohet me ViewResolver për të gjetë view të duhur,
- ia përcjell kërkesën drejt view,
- i kthen përgjigjen klientit,
- krijon një faqe web si përgjigje, duke përmbledhur të dhënat nga bean, kontrolleri dhe view.

Në këtë projekt, DispatcherServlet është implementuar si më poshtë:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">

  <display-name>shembull-springmvc-fshn-
unishk</display-name>

     <servlet>

        <servlet-name>spring</servlet-name>

        <servlet-class>


org.springframework.web.servlet.DispatcherServlet

        </servlet-class>

        <init-param>

                <param-
name>contextConfigLocation</param-name>

                <param-value>/WEB-INF/spring-
servlet.xml</param-value>

           </init-param>

        <load-on-startup>1</load-on-startup>

    </servlet>

    <servlet-mapping>

        <servlet-name>spring</servlet-name>
```

```
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Kodi 3 : Implementimi i DispatcherServlet

Në një version të ardhshëm, funksionaliteti duhet të ndahet në disa microservice, të cilët lidhen bashkë, duke formuar kështu një aplikacion të unifikuar. Në (SURYOTRISONGKO et al. 2017) analizohet një shërbim i tillë, i cili ka si avantazh faktin se një sistemi microservice mund t'i shtojmë funksionalitete të tjera në formën e microservice, pa ndikuar në sistemin ekzistues. Programuesi mund të zhvillojë funksionalitete të reja, në mënyrë të pavarur nga sistemi aktual, gjë që kursen mjaft kohë nga programimi nga e para e sistemit.

Përfundime

Teknologjia e Microservice preferohet për lehtësitë që ofron, ku falë klasave të paracaktuara të Spring, mund të ndërtohen sisteme microservice të përshkallëzueshëm. Bazuar në zhvillimin e aplikacionit të realizuar në këtë punim, mund të arrijmë në konkluzionin se arkitektura e microservice duke përdorë teknologjinë Spring MVC, mund të realizojë shërbimin e komunikimit të të dhënave në format JSON, të implementuar me të dhëna të shërbimit meteorologjik pranë Universitetit "Luigj Gurakuqi".

Referencat

DRAGONI, N., LANESE, I., LARSEN, S. T., MAZZARA, M., MUSTAFIN, R. & SAFINA, L. 2017: Microservices: How To Make Your Application Scale

DULI, S. 2018: Projektimi dhe zhvillimi në RESTful API i shërbimit Web që ofron të dhëna në kod të hapur, Buletini Shkencor i Universitetit të Shkodrës "Luigj Gurakuqi", Nr. 68 /Seria e Shkencave të Natyrës

LAYKA, V. 2014: Learn Java for Web Development, Apress

MERKEL,D. 2014: Docker: lightweight Linux containers for consistent development and deployment. Linux J.

NEWMAN,S. 2015: *Building Microservices* (1st ed.), O'Reilly Media, Inc

RUBIO, D., LONG, J., MAK, G. & DEINUM,M. 2014: Spring Recipes, A Problem-Solution Approach, Apress

SELMADJI, A., SERIAI, A.D., BOUZIANE, H.L., DONY, C. & MAHAMANE, R. O., 2018: Re-architecting OO Software into Microservices: A Quality-Centred Approach. ESOCC: European Conference on Service-Oriented and Cloud Computing

SHARMA, S. & GONZALEZ, D. 2017: Microservices: Building scalable software

SURYOTRISONGKO, H., JAYANTO, D.P. & TJAHYANTO, A. 2017: Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot, Procedia Computer Science, Volume 124

STANLEY, K., SCHNABEL, E. & HOFMANN, M. 2016: Microservices Best Practices for Java, IBM