Ministerul Educației al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Inginerie Software Automatică

Raport

Lucrarea de laborator nr. 3
Disciplina: Programarea aplicațiilor distribuite
Tema: Protocolul HTTP: mijloc transport date distribuite

A realizat:
St. gr. TI-142 D. Gorduz

A verificat:
Lect. asist. M. Pecari

Chișinău 2017

Scopul lucrării

Studiul protocolului HTTP în contextul distribuirii datelor. Utilizarea metodelor HTTP în implementarea interacțiunii dintre client și un server de aplicații.

Obiectivele

Aplicarea protocolului HTTP în transmisiuni de date;
Elaborarea unui sistem cu procesare concurentă a cererilor HTTP;
Utilizarea posibilităților oferite de HTTP pentru implementarea unor funcționalități mai avansate a sistemului.

Linkul spre repositoriul GIT:

Mersul lucrării de laborator

Am efectuat această lucrare de laborator în limbajul de programare Java cu implementarea frameworkului SPRING.

Aceste serviciu va deține cereri de GET pentru /person, și POST pentru a introduce date în system. Cererea GET va returna un răspuns 200 OK, cu o JSON structură ce reprezintă perosana. Va arata în felul următor:

```
{
    "id": 1,
    "name": "Dorin"
}
```

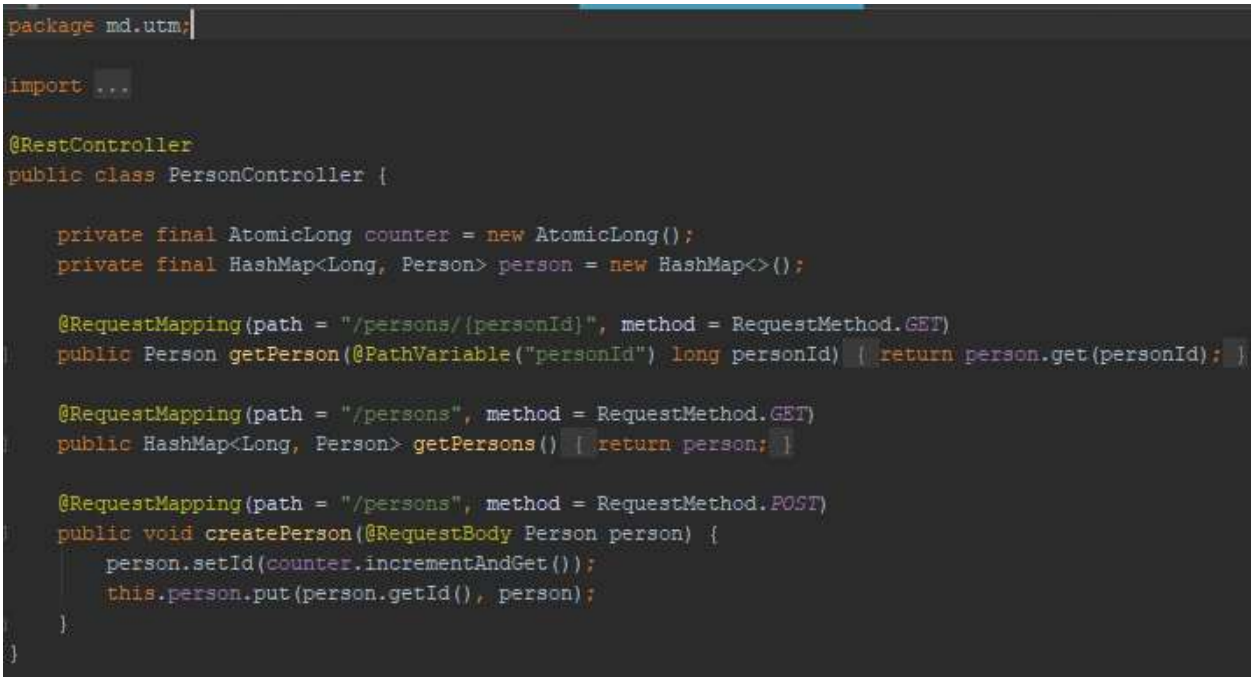
Câmpul id este identificatorul unic pentru *person* și *name* este reprezentarea textuală a greeting-ului.

Pentru a modela reprezentarea a *person,* vom crea resursa reprezentantă a clasei, vezi figura 1. Aici se includ contructorii, get-terii și set-terii pentru câmpurile *id* și *name.*

```
package md.utm;

public class Person {

    private long id;
    private String name;

    public Person() {

    }

    public Person(long id, String name) {
        this.id = id;
        this.name = name;
    }

    public long getId() { return id; }

    public String getContent() { return name; }

    public void setId(long id) { this.id = id; }

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

Figura 1 – Clasa Person

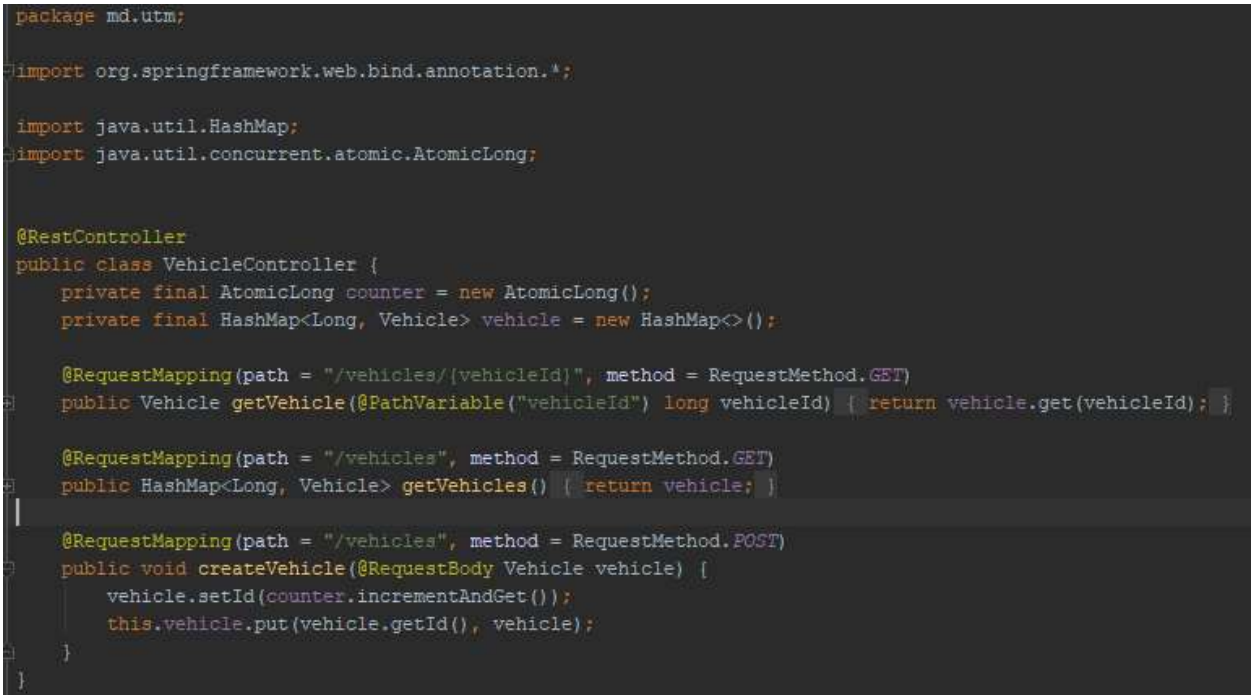
Aici este reprezentat controlul de resurse ce v ava servi aceste date. În Sring pentru a crea RESTful web servicii, HTTP cereri sunt controlate de un controler special. Aceste componente sunt usor indentificate după anotațiile: @RestController și mai jos va fi reprezntat structura codului a PersonController.java, figura 2, ce procesează cererile GET și POST prin returnarea unei nou instnațe a clasei Person.



```java
package md.utm;

import ...

@RestController
public class PersonController {

    private final AtomicLong counter = new AtomicLong();
    private final HashMap<Long, Person> person = new HashMap<>();

    @RequestMapping(path = "/persons/{personId}", method = RequestMethod.GET)
    public Person getPerson(@PathVariable("personId") long personId) { return person.get(personId); }

    @RequestMapping(path = "/persons", method = RequestMethod.GET)
    public HashMap<Long, Person> getPersons() { return person; }

    @RequestMapping(path = "/persons", method = RequestMethod.POST)
    public void createPerson(@RequestBody Person person) {
        person.setId(counter.incrementAndGet());
        this.person.put(person.getId(), person);
    }
}
```

Figura 2 – Clasa PersonController

@RequestMapping este o adnotare pentru maparea cererilor web pe anumite clase de handler și / sau metode handler. Într-un mod similar a fost și efectuată pentru clasa și controller-ul Vehicle.



```java
package md.utm;

import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.concurrent.atomic.AtomicLong;


@RestController
public class VehicleController {
    private final AtomicLong counter = new AtomicLong();
    private final HashMap<Long, Vehicle> vehicle = new HashMap<>();

    @RequestMapping(path = "/vehicles/{vehicleId}", method = RequestMethod.GET)
    public Vehicle getVehicle(@PathVariable("vehicleId") long vehicleId) { return vehicle.get(vehicleId); }

    @RequestMapping(path = "/vehicles", method = RequestMethod.GET)
    public HashMap<Long, Vehicle> getVehicles() { return vehicle; }

    @RequestMapping(path = "/vehicles", method = RequestMethod.POST)
    public void createVehicle(@RequestBody Vehicle vehicle) {
        vehicle.setId(counter.incrementAndGet());
        this.vehicle.put(vehicle.getId(), vehicle);
    }
}
```

Figura 3 – Clasa VehicleController

În figura 4 avem clasa Application care este și Main al nostru. Am utilizat suportul Spring pentru încorporarea containerului de servlet Tomcat în runtime a HTTP, în loc să fie deplasată la o instanță externă.

@SpringBootApplication este o adnotare comodă care adaugă toate următoarele:

@Configuration etichetează clasa ca o sursă de definiții de fasole pentru contextul aplicației.

@EnableAutoConfiguration spune Spring Boot pentru a începe adăugarea beans pe baza setărilor de clasă, a altor bean-uri și a diferitelor setări de proprietăți.

În mod normal, ați adăuga @EnableWebMvc pentru o aplicație MVC de primăvară, dar Boot-ul de primăvară îl adaugă automat atunci când vede primăvara-webmvc pe clasă. Aceasta semnalizează aplicația ca aplicație web și activează comportamente cheie, cum ar fi configurarea unui DispatcherServlet.

@ComponentScan spune Spring să caute alte componente, configurații și servicii în pachetul hello, permițându-i să găsească controlorii.

```
package md.utm;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

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

Figura 4 – Clasa Application

Astfel datele sunt salvate pe serverul local la adresa http://localhost:8080

Ca interfață a clientului nostru am folosit extensia Advanced REST Client din Google Chrome. Aceasta ne permite să accesăm serverul local și să introducem datele prin POST și să afișăm acestea prin GET.

Introducerea în baza de date are loc prin selectarea metodei POST, inserarea a Request URL, ce este în cazul nostru http://localhost:8080/persons șl în body adăugăm câmpurile necesare în structura JSON, figura 5.

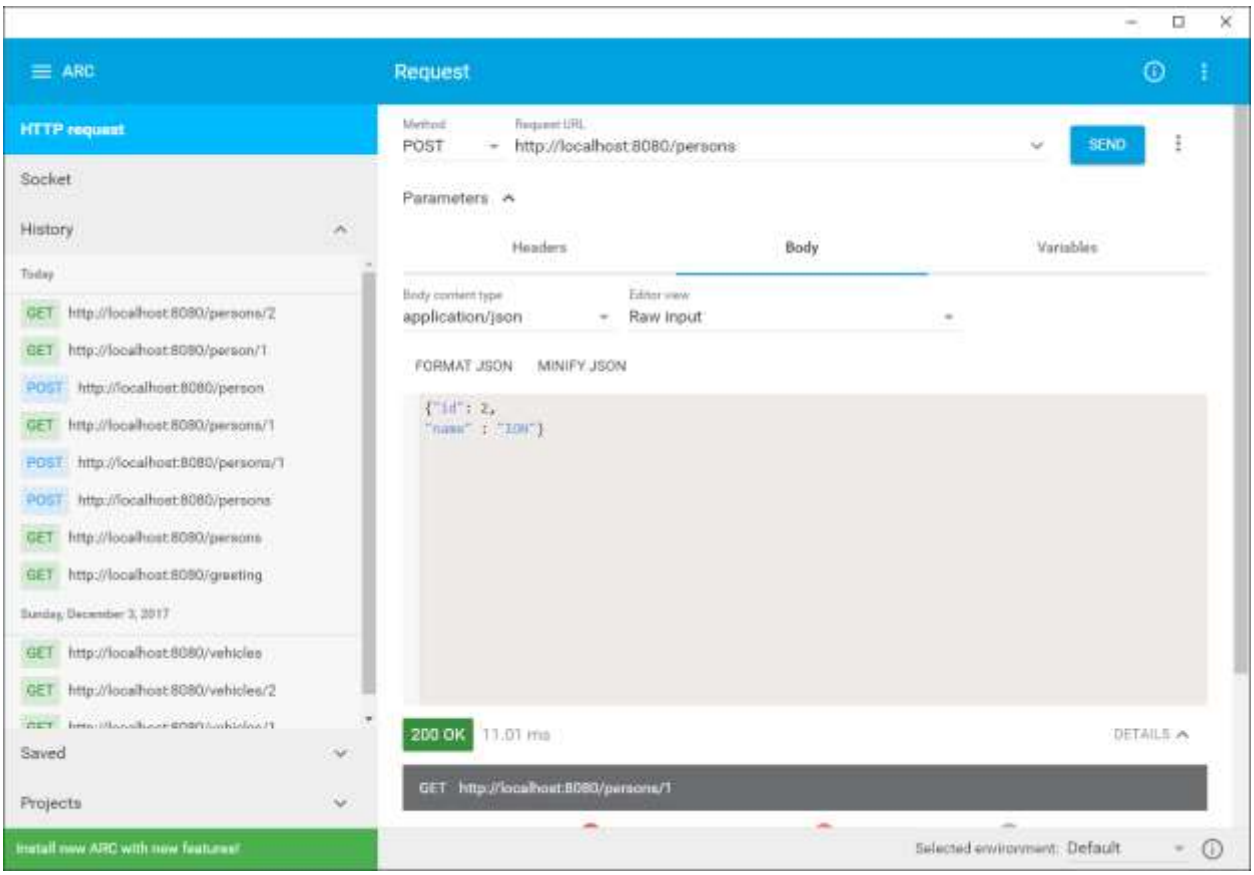


Figura 5 – inserarea datelor

Pentru extragerea acestora vom folosi selecta metoda GET, vom introduce request-ul URL: http://localhost:8080/persons/2 unde la sfârșit indicăm id-ul care dorim să îl afișăm, figura 6.

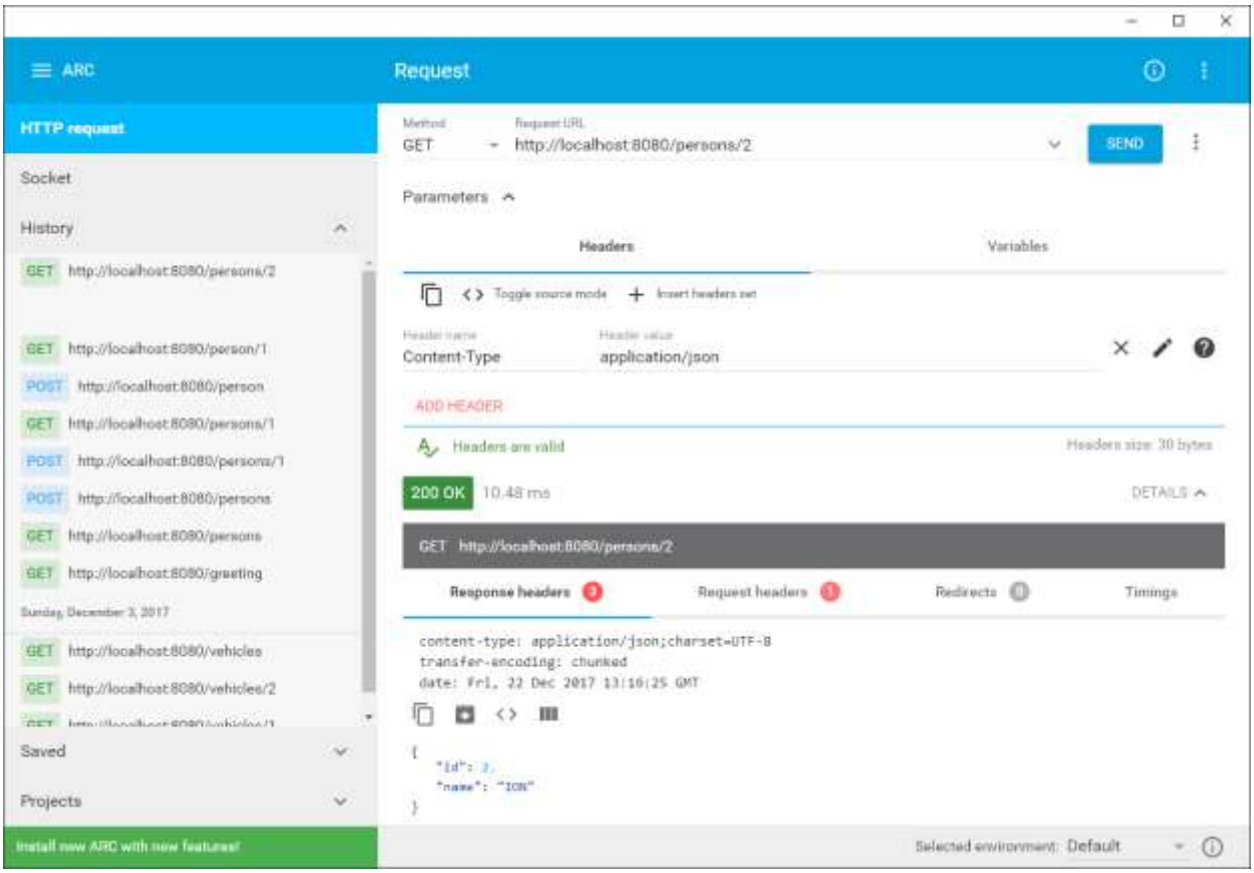


Figura 6 – Extragerea din baza de date a unei persoane

Concluzie

În această lucrare de laborator am studiat protocolul HTTP în contextul distribuirii datelor. Am utilizat metodele HTPP – GET, POST, pentru a obține date în format JSON. Prin utilizarea acestora, am creat un RestFull API capabil să interacționeze cu alte aplicații și care poate handle errors.