Лабораторная №10

Spring Framework.

REST.

**Цель**: Разработать учебную подсистему ввода и модификации данных о клиентах условного сервиса используя метод взаимодействия компонентов распределённого приложения REST (Representational State Transfer — «передача репрезентативного состояния»).

**Среда**. OS - произвольно. Рекомендуемый язык программирования Java EE. Рекомендуемые базы данных - MySQL, MS SQL (одна любая, на выбор).

Теоретическая часть

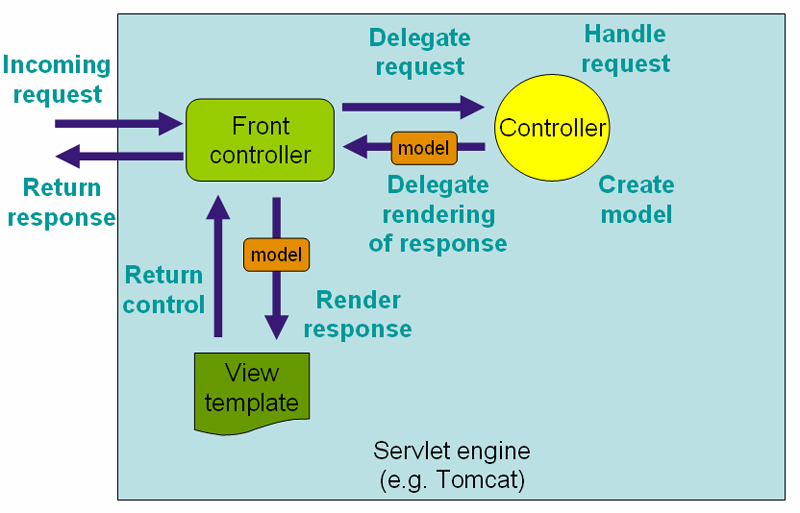
Так что такое REST?

"Передача состояния представления (Representational State Transfer (REST)) — это стиль архитектуры программного обеспечения для распределенных гипермедиа систем, подобных Всемирной паутине."  
В основе REST подхода лежит ресурс, обладающий уникальным адресом (URI). С этим ресурсом мы работаем при помощи HTTP протокола.  
HTTP протокол поддерживает 11 операций над ресурсами. REST использует освные из них для CRUD операций: POST, GET, PUT, DELETE. POST - добавить ресурс (create), GET - получить ресурс (read), PUT - обновить ресурс (update) и DELETE - удалить ресурс (delete).

RESTful веб-сервис - это веб-сервис реализованный с использованием HTTP и принципов REST. Он представляет из себя набор ресурсов с тремя определенными аспектами (википедия):

* базовый URI для веб-службы, например http://example.com/resources/
* тип содержимого Интернет для данных, поддерживаемых веб-службой. Часто это JSON, XML или YAML, но можно использовать любой другой действительный тип содержимого Интернет.
* множество операций, поддерживаемых веб-службой, используя основные механизмы протокола (то есть, POST, GET, PUT или DELETE).

"  
Тип содержимого, который хотим получить, мы указываем в HTTP запросе.  
Это лишь краткое описание данной архитектуры.



Разница в конфигурации бинов через xml и java классов:

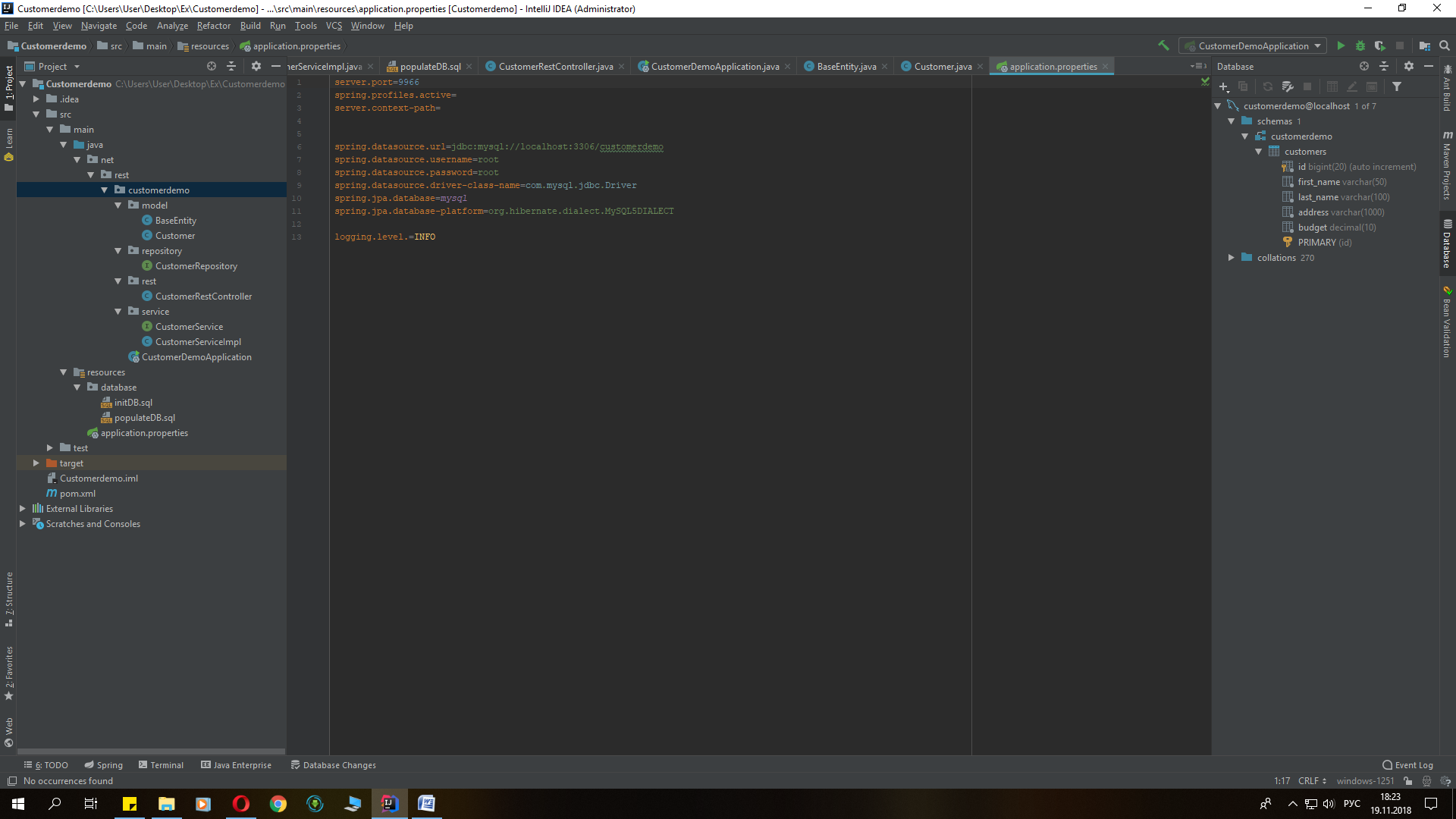
XML

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">  
  
 <bean id="helloBean" class="com.devcolibri.hello.impl.HelloWorldImpl">  
  
</beans>

Класс

package com.devcolibri.config;  
  
import com.devcolibri.hello.impl.HelloWorldImpl;  
import com.devcolibri.hello.\*;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
 @Configuration  
 public class BeenConfig {  
  
 @Bean(name="helloBean")  
 public HelloWorld helloWorld() {  
 return new HelloWorldImpl();  
 }  
  
 }

Создадим проект:



Конфигурация проекта будет основана на Spring Boot.

Так же в проекте присутствует Lombok, который позволяет не конфигурировать свойства в POJO классах.

Создадим нашу модель из двух классов.

Класс BaseEntity

package net.rest.customerdemo.model;  
  
import lombok.Getter;  
import lombok.Setter;  
import lombok.ToString;  
  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.MappedSuperclass;  
  
*/\*\*  
 \** ***@author*** *User  
 \* класс для свойств  
 \*/*@MappedSuperclass  
@Getter  
@Setter  
@ToString  
public class BaseEntity {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private long id;  
}

Класс BaseEntity

package net.rest.customerdemo.model;  
  
import lombok.Getter;  
import lombok.Setter;  
import lombok.ToString;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Table;  
import java.math.BigDecimal;  
  
*/\*\*  
 \** ***@author*** *User  
 \* класс для заказчика  
 \*/*@Entity  
@Table(name = "customers")  
@Getter  
@Setter  
@ToString  
  
public class Customer extends BaseEntity {  
  
 @Column(name = "first\_name")  
 private String firstName;  
  
 @Column(name = "last\_name")  
 private String lastName;  
  
 @Column(name = "address")  
 private String address;  
  
 @Column(name = "budget")  
 private BigDecimal budget;  
}

Интерфейс репозитрий CustomerRepository

package net.rest.customerdemo.repository;  
  
import net.rest.customerdemo.model.Customer;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
*/\*\*  
 \** ***@author*** *User  
 \*/*public interface CustomerRepository extends JpaRepository<Customer,Long> {  
}

Интерфейс CustomerService

package net.rest.customerdemo.service;  
  
import net.rest.customerdemo.model.Customer;  
  
import java.util.List;  
  
*/\*\*  
 \** ***@author*** *User  
 \*/*public interface CustomerService {  
  
 Customer getById(Long id);  
  
 void save(Customer customer);  
  
 void delete(Long id);  
  
 List<Customer> getAll();  
}

Класс CustomerServiceImpl

package net.rest.customerdemo.service;  
  
import lombok.extern.slf4j.Slf4j;  
import net.rest.customerdemo.model.Customer;  
import net.rest.customerdemo.repository.CustomerRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.logging.Logger;  
  
*/\*\*  
 \** ***@author*** *User  
 \*/*@Slf4j  
@Service  
public class CustomerServiceImpl implements CustomerService {  
  
  
  
 @Autowired  
 CustomerRepository customerRepository;  
  
 @Override  
 public Customer getById(Long id) {  
 // log.info("IN CustomerServiceImpl getById {}");  
 return customerRepository.findOne(id);  
 }  
  
 @Override  
 public void save(Customer customer) {  
 // log.info("IN CustomerServiceImpl save {}");  
 customerRepository.save(customer);  
 }  
  
 @Override  
 public void delete(Long id) {  
 // log.info("IN CustomerServiceImpl delete {}");  
 customerRepository.delete(id);  
 }  
  
 @Override  
 public List<Customer> getAll() {  
 // log.info("IN CustomerServiceImpl getAll");  
 return customerRepository.findAll();  
 }  
}

Класс CustomerRestController

package net.rest.customerdemo.rest;  
  
import net.rest.customerdemo.model.Customer;  
import net.rest.customerdemo.service.CustomerService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpHeaders;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.util.UriComponentsBuilder;  
  
import javax.validation.Valid;  
import java.util.List;  
  
*/\*\*  
 \** ***@author*** *User  
 \*/*@RestController  
@RequestMapping("/api/customers")  
public class CustomerRestController {  
  
 @Autowired  
 private CustomerService customerService;  
  
 @RequestMapping(value = "{is}", method = RequestMethod.*GET*, produces = MediaType.*APPLICATION\_JSON\_UTF8\_VALUE*)  
 public ResponseEntity<Customer> getCustomer(@PathVariable("id") Long customerId) {  
  
 if (customerId == null) {  
 return new ResponseEntity<>(HttpStatus.*BAD\_REQUEST*);  
 }  
  
 Customer customer = this.customerService.getById(customerId);  
  
 if (customer == null) {  
 return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 }  
  
 return new ResponseEntity<>(customer, HttpStatus.*OK*);  
 }  
  
 @RequestMapping(value = "", method = RequestMethod.*POST*, produces = MediaType.*APPLICATION\_JSON\_UTF8\_VALUE*)  
 public ResponseEntity<Customer> saveCustomer(@RequestBody @Valid Customer customer) {  
 HttpHeaders headers = new HttpHeaders();  
  
 if (customer == null) {  
 return new ResponseEntity<>(HttpStatus.*BAD\_REQUEST*);  
 }  
  
 this.customerService.save(customer);  
 return new ResponseEntity<>(customer, headers, HttpStatus.*CREATED*);  
 }  
  
 @RequestMapping(value = "", method = RequestMethod.*PUT*, produces = MediaType.*APPLICATION\_JSON\_UTF8\_VALUE*)  
 public ResponseEntity<Customer> updateCustomer(@RequestBody @Valid Customer customer, UriComponentsBuilder builder) {  
 HttpHeaders headers = new HttpHeaders();  
  
 if (customer == null) {  
 return new ResponseEntity<>(HttpStatus.*BAD\_REQUEST*);  
 }  
  
 this.customerService.save(customer);  
  
 return new ResponseEntity<>(customer, headers, HttpStatus.*OK*);  
 }  
  
 @RequestMapping(value = "{id}", method = RequestMethod.*DELETE*, produces = MediaType.*APPLICATION\_JSON\_UTF8\_VALUE*)  
 public ResponseEntity<Customer> deleteCustomer(@PathVariable("id") Long id) {  
 Customer customer = this.customerService.getById(id);  
  
 if (customer == null) {  
 return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 }  
  
 this.customerService.delete(id);  
  
 return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
  
 @RequestMapping(value = "", method = RequestMethod.*GET*, produces = MediaType.*APPLICATION\_JSON\_UTF8\_VALUE*)  
 public ResponseEntity<List<Customer>> getAllCustomers() {  
 List<Customer> customers = this.customerService.getAll();  
  
 if (customers.isEmpty()) {  
 return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 }  
  
 return new ResponseEntity<>(customers, HttpStatus.*OK*);  
 }  
}

Main класс

package net.rest.customerdemo;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
*/\*\*  
 \** ***@author*** *User  
 \*/*@SpringBootApplication  
public class CustomerDemoApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(CustomerDemoApplication.class);  
 }  
}

application.propperties

server.port=9966  
spring.profiles.active=  
server.context-path=  
  
  
spring.datasource.url=jdbc:mysql://localhost:3306/customerdemo  
spring.datasource.username=root  
spring.datasource.password=root  
spring.datasource.driver-class-name=com.mysql.jdbc.Driver  
spring.jpa.database=*mysql*spring.jpa.database-platform=org.hibernate.dialect.MySQL5DIALECT  
  
logging.level.=INFO

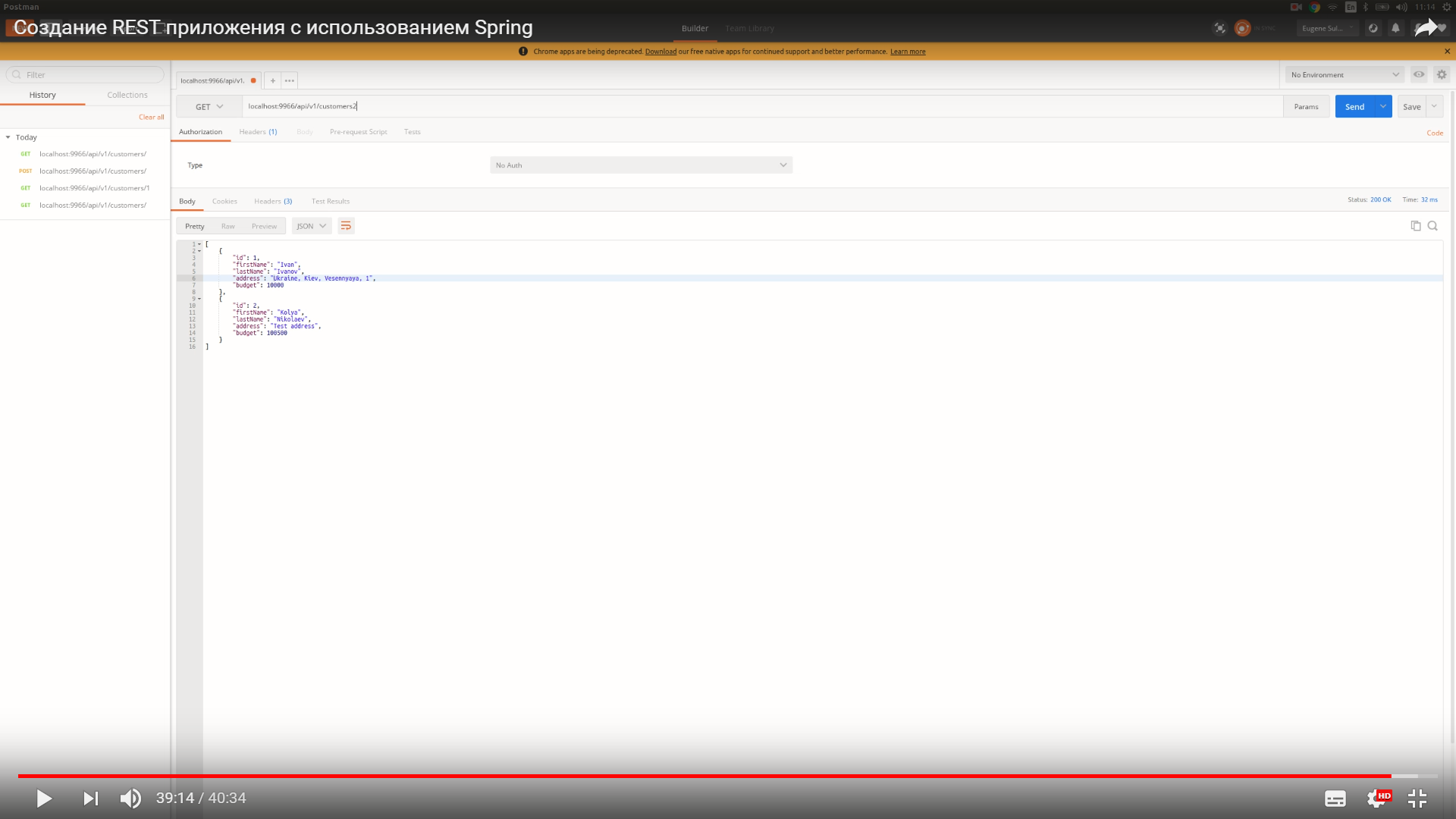
pom.xml

<?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>  
  
 <groupId>net.spring.rest</groupId>  
 <artifactId>Customer demo</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <name>CustomerDemo</name>  
 <description>REST\_Example</description>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.2.RELEASE</version>  
 <relativePath/>  
 </parent>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <lombok.version>1.16.18</lombok.version>  
 </properties>  
  
 <dependencies>  
 <!-- Spring -->  
 <!-- для работы с ORM -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <!-- для REST контроллера -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-rest</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-tomcat</artifactId>  
 </dependency>  
 <!-- MySQL -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.46</version>  
 </dependency>  
 <!-- Lombok -->  
 <!-- для работы с get и set -->  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>${lombok.version}</version>  
 <scope>provided</scope>  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

БД

**CREATE TABLE IF NOT EXISTS** customers (  
 id **BIGINT AUTO\_INCREMENT PRIMARY KEY**,  
 first\_name **VARCHAR**(50) **NOT NULL**,  
 last\_name **VARCHAR**(100) **NOT NULL**,  
 address **VARCHAR**(1000) **NOT NULL**,  
 budget **DECIMAL NOT NULL**);

Используем POSTMAN для отправки и получения запросов.



Практическая часть

1. Выбрать поля базы данных для ввода и хранения информации о клиентах для выполнения лабораторной работы согласно номеру варианта.

**Варианты индивидуального задания.**

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование поля | Тип поля | Обязательное (да, нет) | **Вариант N** |
| Фамилия | Текстовый | Да | Любой |
| Имя | Текстовый | Да | Любой |
| Отчество | Текстовый | Да | Любой |
| Дата рождения | Дата | Да | Любой |
| Пол | Boolean | Да | **Четный** |
| Серия паспорта | Текстовый | Да | Любой |
| № паспорта | Текстовый (с маской) | Да | Любой |
| Идент. номер | Текстовый (с маской) | Нет | **Четный** |
| Город проживания | **Список** (от 5 городов) | Да | Любой |
| Адрес факт.проживания | Текстовый | Да | Любой |
| Телефон дом | Текстовый (с маской) | Нет | **Четный** |
| Телефон моб | Текстовый (с маской) | Нет | Любой |
| E-mail | Текстовый | Нет | **Нечетный** |
| Трудостроен | Boolean | Нет | **Нечетный** |
| Должность | Текстовый | Нет | **Нечетный** |
| Город прописки | **Список** (от 5 городов) | Да | **1,4,7... (N%3==1)** |
| Адрес прописки | Текстовый | Да | **2,5,8... (N%3==2)** |
| Гражданство | **Список** | Да | Любой |
| Военнообязанный | Boolean | Да | **3,6,9... (N%3==0)** |

1. Реализовать API REST на базе Spring Framework для добавления, чтения, обновления и удаления (CRUD) клиентов в СУБД условного сервиса по адресу http://host:port/rest/user/userID, где userID=0 для операции создания пользователя, и id для остальных операций. Ответ сервера содержит данные в формате:

- **json** (для вариантов **1..6, 13..18**);

- **xml** (для вариантов **7..12, 19..24**).

5. На **хорошую** оценку дополнительно реализовать API REST добавления, чтения, обновления и удаления (CRUD) группы клиентов в СУБД условного сервиса по адресу http://host:port/rest/users.

6. На **отличную** оценку предусмотреть **серверную** проверку корректности введенных значений в каждом из полей возвратом кода **400** – Bad request в случае ошибки или незаполнения обязательного поля.

7. Добавить в базу данных через разработанный программный интерфейс не менее 5 клиентов, включая себя.

8. На защите продемонстрировать четыре операции (CRUD).

**Рекомендации**.

1. При проектировании внутренних Java-интерфейсов рекомендуется использовать паттерн Repository (см. http://habrahabr.ru/post/263033/ ).

2. Для проверки REST можно использовать расширение Postman или заранее заготовленные html-формы с вписанными по умолчанию данными.

3. Для возможности смены базы данных на более производительное решение подключение к базе данных удобнее выполнить через JPA или аналог.

4. При подготовке работы на отличную оценку поля нужно проверить регулярным выражением, по смыслу соответствующим типу (или маске) поля, а поля справочников проверить на допустимость указанных в запросе значений.