**Spring Rest Controllers**

**Теория**

**Controllers.html –** введение в контроллеры, описание RestController

**Request Annotaions.html** – описание аннотаций в REST котроллерах

**Jackson Annotation.html –** описание аннотаций для настройки сущностей при передачи по сети в виде JSON

<https://www.getpostman.com/docs/> – документация по Postman

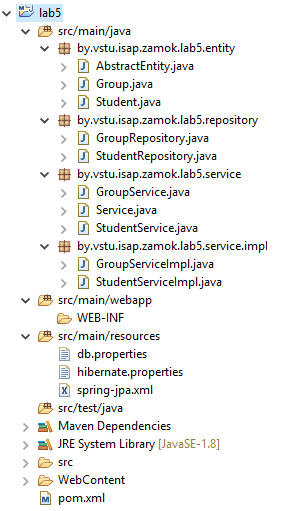
**Практическая часть**

# Создание проекта

Создать проект **maven-archetype-webapp** и сконфигурировать его. Удалить web.xml, указать новую версию **Dynamic Web Module** в Project Facets (Лабораторная работа 3).

# Добавить к проекту Spring Data

Скопировать пакеты сущностей, repository, сервисов, а также содержимое папке resources из лабораторной работы 4.



# Сконфигурировать pom.xml для работы с Jackson

Добавить представленные зависимости и плагины:

<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/maven-v4\_0\_0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>by.vstu.isap.zamok</groupId>

<artifactId>lab5</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>lab5 Maven Webapp</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

<spring.framework.version>4.3.10.RELEASE</spring.framework.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.framework.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-jpa</artifactId>

<version>1.11.7.RELEASE</version>

</dependency>

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.2.11.Final</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.framework.version}</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.8.6</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.8.6</version>

</dependency>

</dependencies>

<build>

<finalName>REST</finalName>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.2</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

# Настроить сущности для передачи по сети

**Примечание**: Так как Jackson переводит в json все поля объекта, из за рекурсивности ссылок (в группе лежит список студентов, в каждом студенте лежит ссылка на группу), требуется настроить Jackson таким образом, чтобы при передаче на сервер студента группа сохранялась, но при передачи на клиент не возникало рекурсивной ссылки. Для полного игнорирования поля указывается аннотация @JsonIgnore, однако в этом случае поле будет недоступно для записи.

В классе студента добавить к полю group аннотацию @JsonProperty(access = Access.***WRITE\_ONLY***):

@ManyToOne(fetch = FetchType.***LAZY***)

@JoinColumn(name = "group\_id", nullable = **false**)

@JsonProperty(access = Access.***WRITE\_ONLY***)

**private** Group group;

**public** String getName() {

**return** name;

}

# Настроить веб-приложение при помощи Java Configuration

Создать пакет **config** и внутри него следующие классы:

Конфигурацию сервлета:

**public** **class** WebAppServletInitializer **extends** AbstractAnnotationConfigDispatcherServletInitializer {

@Override

**protected** Class<?>[] getRootConfigClasses() {

**return** **new** Class[] { WebAppConfig.**class** };

}

@Override

**protected** Class<?>[] getServletConfigClasses() {

**return** **null**;

}

@Override

**protected** String[] getServletMappings() {

**return** **new** String[] { "/" };

}

}

Конфигурацию контекста

@Configuration

@EnableWebMvc

@ImportResource("classpath:spring-jpa.xml")

@ComponentScan("by.vstu.isap.zamok.lab5")

**public** **class** WebAppConfig **extends** WebMvcConfigurerAdapter {

}

# Создание REST-контроллеров

Создать пакет **controller** и внутри него следующие классы:

Абстрактный контроллер, содержащий типовую реализацию обработки REST–запросов.

**public** **abstract** **class** AbstractController<T **extends** AbstractEntity> {

**protected** HttpHeaders headers;

@PostConstruct

**private** **void** init() {

headers = **new** HttpHeaders();

headers.setContentType(MediaType.***APPLICATION\_JSON***);

}

@GetMapping

**public** ResponseEntity<List<T>> get() {

List<T> entities = getService().read();

**if** (entities.isEmpty()) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(entities, headers, HttpStatus.***OK***);

}

@GetMapping("/{id}")

**public** ResponseEntity<T> getById(@PathVariable **long** id) {

T entity = getService().read(id);

**if** (entity == **null**) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(entity, headers, HttpStatus.***OK***);

}

@PutMapping

**public** ResponseEntity<String> put(@RequestBody T entity) {

getService().save(entity);

**return** **new** ResponseEntity<>(HttpStatus.***CREATED***);

}

@PostMapping

**public** ResponseEntity<String> post(@RequestBody T entity) {

getService().save(entity);

**return** **new** ResponseEntity<>(HttpStatus.***OK***);

}

@DeleteMapping("/{id}")

**public** ResponseEntity<String> delete(@PathVariable **long** id) {

getService().delete(id);

**return** **new** ResponseEntity<>(HttpStatus.***OK***);

}

**public** **abstract** Service<T> getService();

}

**Примечание**: в случае, если для всех контроллеров типовые действия не могут быть выделены, абстрактный класс не создается, а методы описываются в каждом из классов.

Контроллер группы:

@RestController

@RequestMapping("api/group")

**public** **class** GroupController **extends** AbstractController<Group> {

@Autowired

**private** GroupService service;

@Override

**public** GroupService getService() {

**return** service;

}

@GetMapping("/name/{name}")

**public** ResponseEntity<Group> getStudentsBySurname(@PathVariable String name) {

Group group = service.readByName(name);

**if** (group == **null**) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(group, headers, HttpStatus.***OK***);

}

}

Контроллер студента:

@RestController

@RequestMapping("api/student")

**public** **class** StudentController **extends** AbstractController<Student> {

@Autowired

**private** StudentService service;

@Override

**public** StudentService getService() {

**return** service;

}

@GetMapping("/group/{id}")

**public** ResponseEntity<List<Student>> getStudentsByGroup(@PathVariable **long** id) {

List<Student> students = service.readByGroupId(id);

**if** (students.isEmpty()) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(students, headers, HttpStatus.***OK***);

}

@GetMapping("/surname/{surname}")

**public** ResponseEntity<List<Student>> getStudentsBySurname(@PathVariable String surname) {

List<Student> students = service.readBySurname(surname);

**if** (students.isEmpty()) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

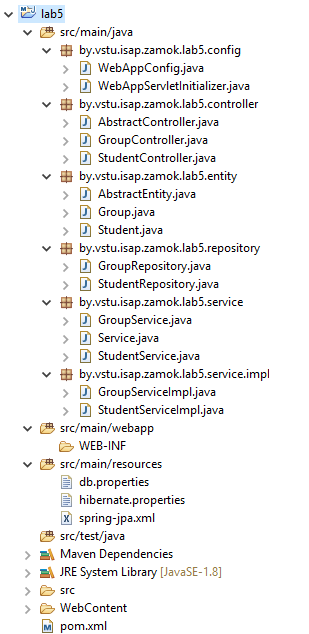
**return** **new** ResponseEntity<>(students, headers, HttpStatus.***OK***);

}

}

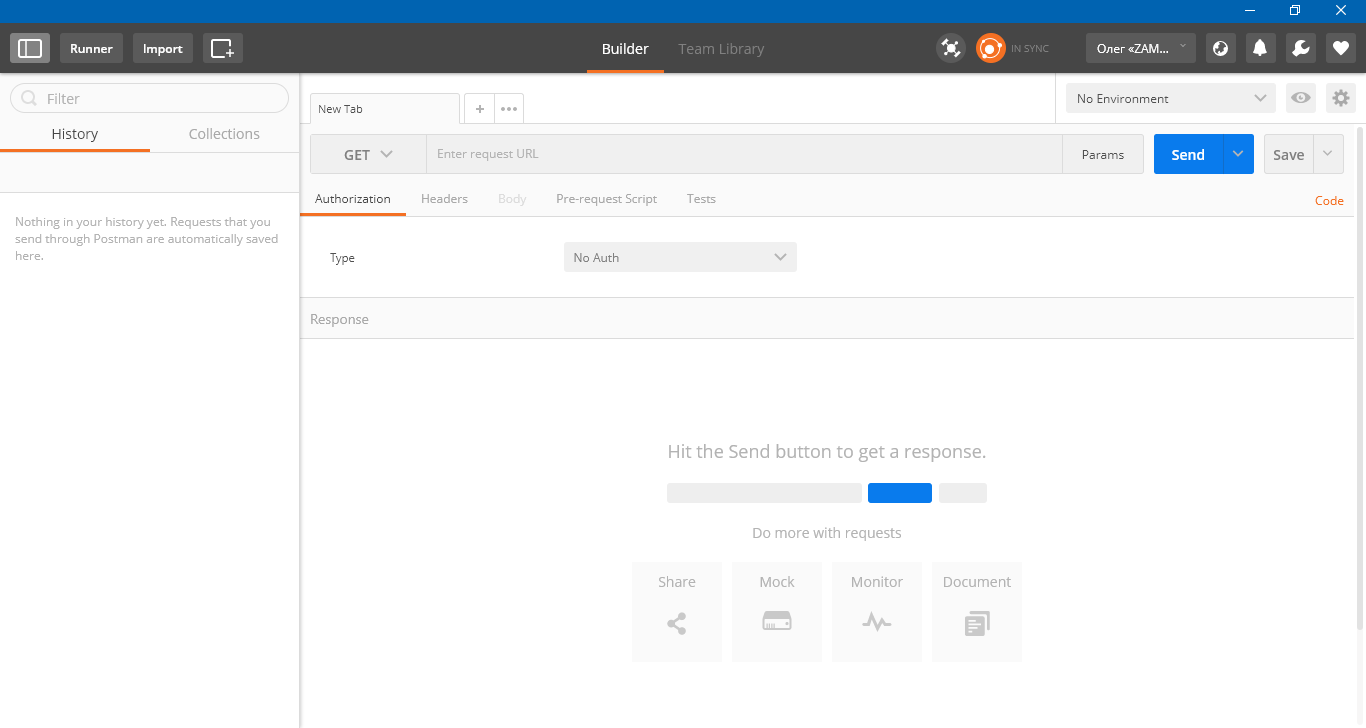
В аннотации маппинга (GetMapping, PostMapping и т.д.) указывается путь, при котором данный метод будет отрабатывать. Переменные могут быть переданы в теле запроса (тогда перед параметром метода указывается @RequestBody), либо в адресе (тогда перед параметром указывается @PathVariable, а название параметра должно совпадать с параметром в маппинге. Например, @GetMapping("/name/{name}") **public** ResponseEntity getByName(@PathVariable String name)) В ответ клиенту придёт то, что было возвращено из метода.

В результате получится следующая архитектура проекта:

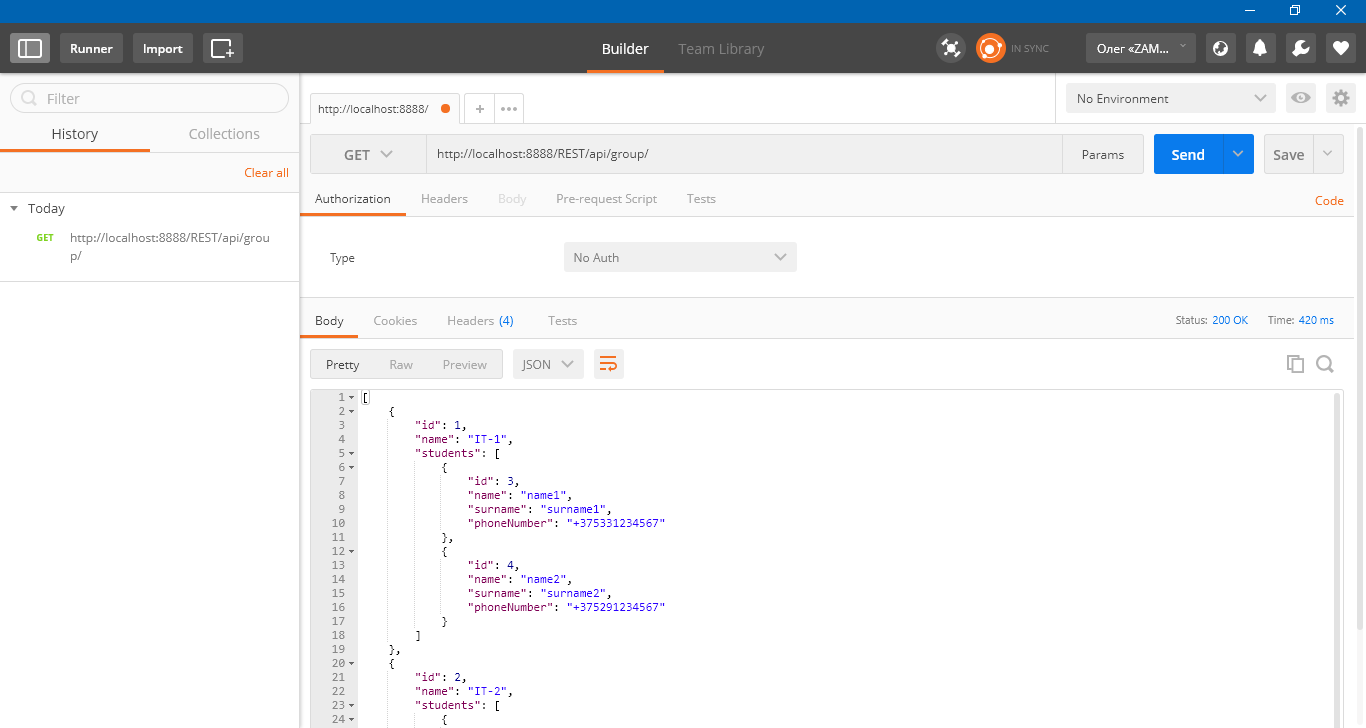


# Тестирование работоспособности REST-сервера

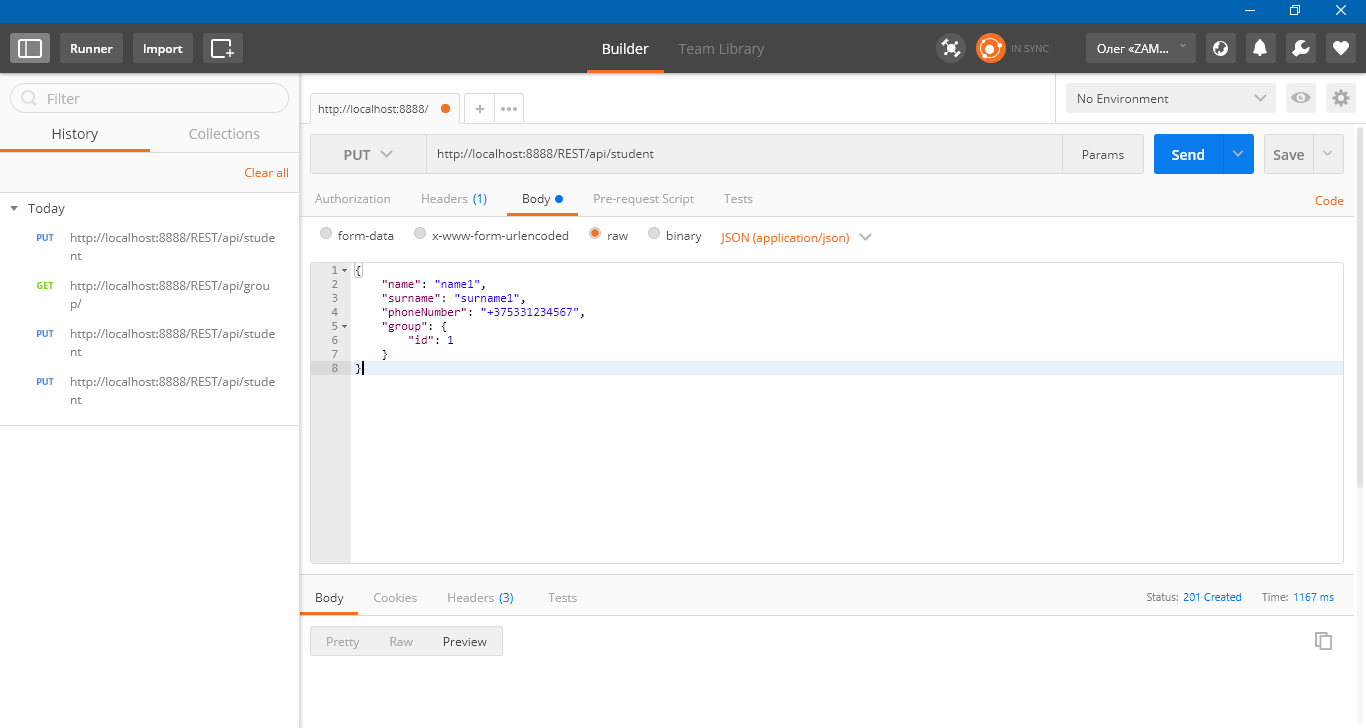
По умолчанию Tomcat сконфигурирован на порт 8080



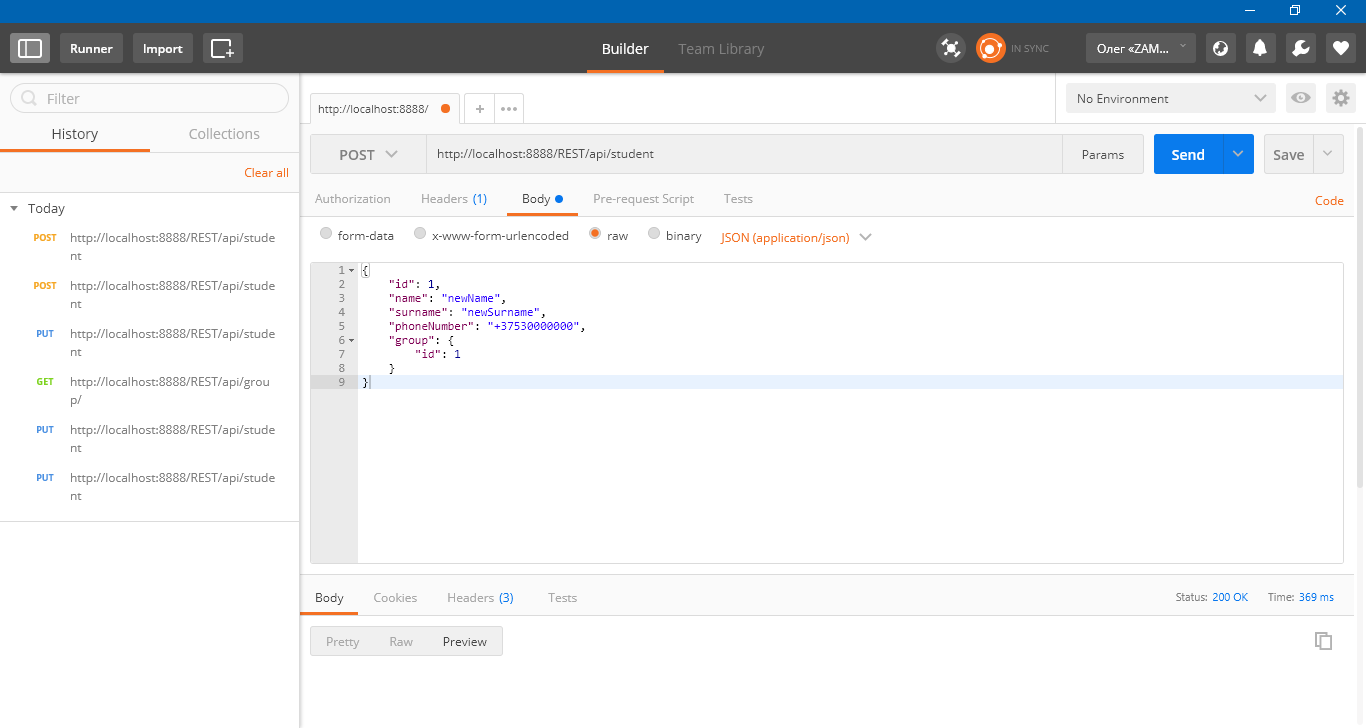
Для запроса сущности требуется указать адрес и тип запроса GET:



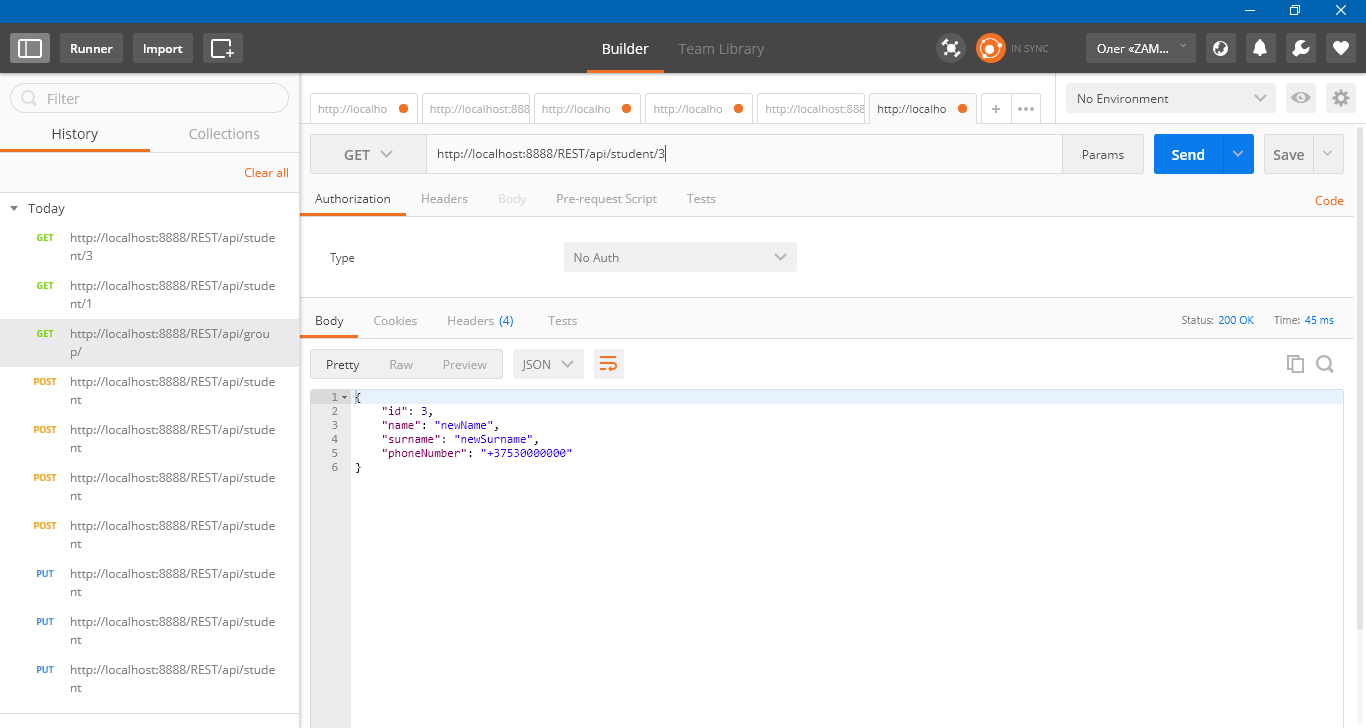
Для добавления новой сущности PUT и добавить заголовок Content-Type:application/json



Для редактирования указывается id и метод POST:



Для запроса либо удаления одной сущности требуется указать ее идентификатор в пути:

 **Индивидуальное задание**: добавить контроллер в соответствии с сущностью, созданной в лабораторной работе 4.

Для сдачи лабораторной требуется продемонстрировать работу методов CRUD в веб-сервисе (create, read, update, delete).