



**СПбГЭТУ «ЛЭТИ»**  
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

# Создание микросервиса на основе REST с помощью Spring Boot


# Инструменты для создания микросервисов



- Liberica JDK 20  
<https://bell-sw.com/pages/downloads/>
- IntelliJ IDEA  
<https://www.jetbrains.com/idea/download/?section=windows>
- Служба Spring Initializr  
<https://start.spring.io/>
- Postman  
<https://www.postman.com/downloads/>

# Создание проекта с помощью Spring Initializr





**Project**  
☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ **Java** ☐ Kotlin ☐ Groovy  
☒ **Maven**

**Language**  
☒ **Java** ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.2.0 (SNAPSHOT) ☐ 3.2.0 (M2) ☐ 3.1.1 (SNAPSHOT) ☒ **3.1.3**  
☐ 3.0.11 (SNAPSHOT) ☐ 3.0.10 ☐ 2.7.16 (SNAPSHOT) ☐ 2.7.15

**Project Metadata**  

Group

Artifact

Name

Description

Package name


Packaging ☒ **Jar** ☐ War


Java ☒ **20** ☐ 17 ☐ 11 ☐ 8

**Dependencies** ADD DEPENDENCIES... CTRL + B

**Spring Web** **WEB**  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Boot Actuator** **OPS**  
Supports built-in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

 fractures.zip

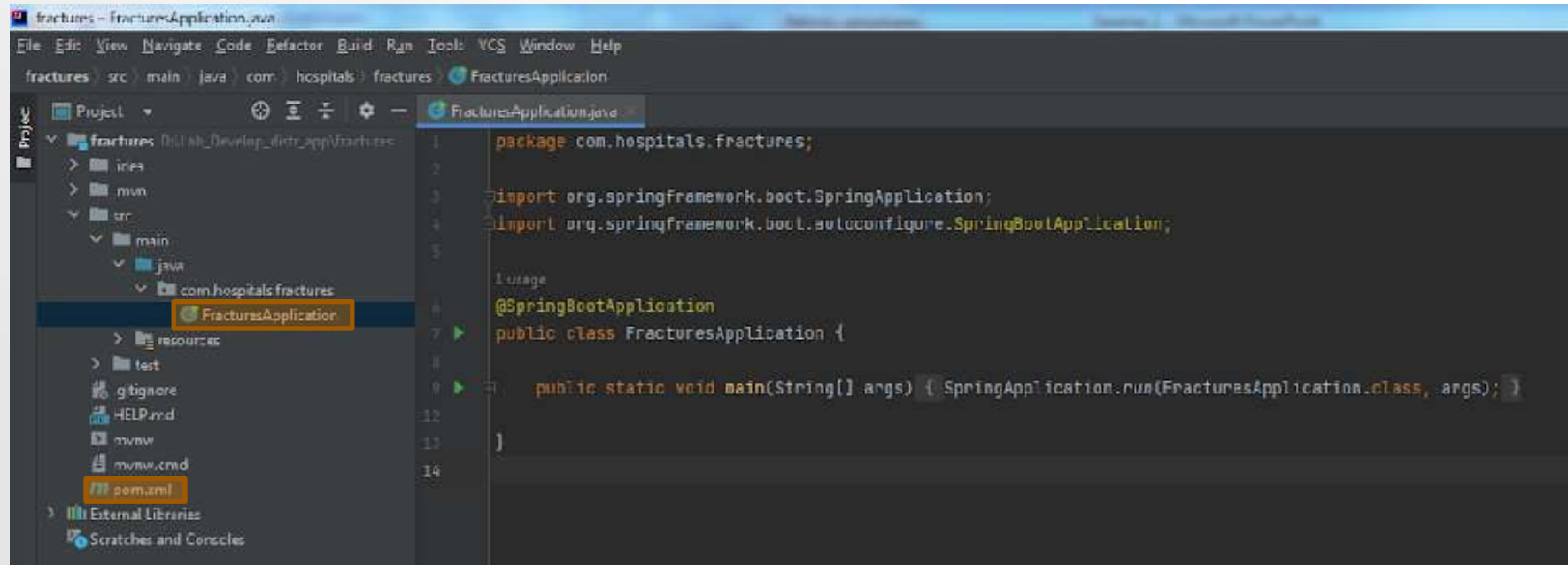


**GENERATE** CTRL + G

**EXPLORE** CTRL + SPACE

**SHARE...**

# Структура проекта в IntelliJ IDEA



# Apache Maven и файл pom.xml



**Apache Maven** – фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (англ. Project Object Model), являющемся подмножеством XML.

**pom.xml** – XML-файл, в котором содержится информация для сборки проекта, поддерживаемого Apache Maven.

Минимальная конфигурация включает версию конфигурационного файла, имя проекта, его автора и версию.

Зависимости от других проектов, индивидуальные фазы процесса построения проекта, список плагинов, реализующих порядок сборки конфигурируются с помощью pom.xml.

# Содержимое файла pom.xml



Подключение зависимостей набора инструментов Spring Boot Starter (Maven должен загрузить версию 3.1.3 фреймворка Spring Boot)

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring boot starter parent</artifactId>
  <version>3.1.3</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Версия Java

```
<properties>
  <java.version>20</java.version>
</properties>
```

# Содержимое файла pom.xml



Подключение зависимостей Spring Boot Actuator и Spring Web

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Подключение плагинов для сборки и развертывания приложений Spring Boot (Maven должен установить последнюю версию плагина Spring Boot для Maven)

```
<plugins>
  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
</plugins>
```

# Класс инициализации FracturesApplication.java



```
fractures - FracturesApplication.java
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
fractures / src / main / java / com / hospitals / fractures / FracturesApplication

Project
  fractures
    src
      main
        java
          com.hospitals.fractures
            FracturesApplication
          resources
          test
          gitignore
          HELP.md
          mvnw
          mvnw.cmd
          pom.xml
    External Libraries
    Scratches and Console

1 package com.hospitals.fractures;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class FracturesApplication {
8
9     public static void main(String[] args) { SpringApplication.run(FracturesApplication.class, args); }
10
11 }
12
13
14
```



**Аннотация** — это специальная форма синтаксических метаданных (например, в Java), которая может быть добавлена в исходный код с целью предоставления данных о программе, не являющихся частью самой программы.

Пакеты, классы, методы, переменные и параметры могут быть аннотируемы.

Внешний вид:                    @ТипАннотации

                                 @ТипАннотации (элемент = значение, ...)

Аннотация размещается в коде перед определением переменной, параметра, метода, класса, пакета.

- Информация для компилятора. Аннотации могут использоваться компилятором для обнаружения ошибок или игнорирования предупреждений.
- Обработка во время компиляции и развертывания. Программные средства могут обрабатывать информацию из аннотаций для генерации кода, файлов XML и т. д.
- Обработка во время выполнения. Некоторые аннотации доступны для использования во время выполнения.

# Аннотация @SpringBootApplication



@SpringBootApplication уведомляет Spring Boot, что класс FracturesApplication является классом инициализации проекта (иначе, классом начальной загрузки).

```
@SpringBootApplication
public class FracturesApplication {

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

}
```

@SpringBootApplication является мета-аннотацией, т.е. инкапсулирует в себе несколько аннотаций:

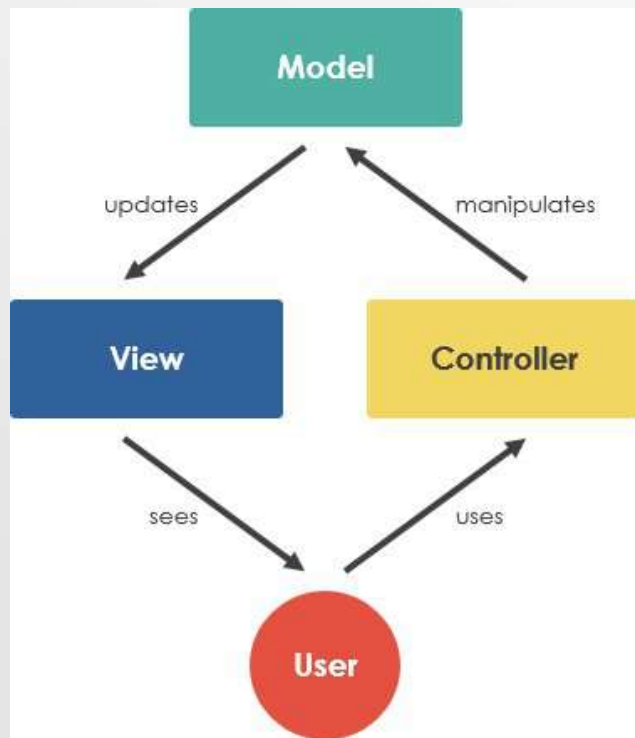
- @ComponentScan;
- @EnableAutoConfiguration;
- @SpringBootConfiguration.

Таким образом @SpringBootApplication включает сканирование компонентов, автоконфигурацию и показывает разным компонентам Spring (например, интеграционным тестам), что это Spring Boot приложение.

# Паттерн Model-View-Controller



Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») – это шаблон проектирования, который предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.



Модель (Model) предоставляет данные предметной области представлению и реагирует на команды контроллера, изменяя свое состояние.

Представление (View) отвечает за отображение данных предметной области (модели) пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

# Создание класса контроллера



Класс контроллера в Spring Boot экспортирует конечные точки службы и на основе данных из входящего HTTP-запроса выбирает метод Java, который будет обрабатывать данный запрос.

Создание класса `FracturesController` производится следующим образом:

- щелкнуть ПКМ на пакете `com.hospitals.fractures` и создать новый пакет `controller`;
- щелкнуть ПКМ на пакете `controller` и создать новый класс `FracturesController`.

Класс `FracturesController` будет экспортировать четыре конечные точки HTTP, которые соответствуют HTTP-запросам `POST`, `GET`, `PUT`, `DELETE`.

# Код класса контроллера



```
package com.hospitals.fractures.controller;
```

```
import org.springframework.web.bind.annotation.RestController;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@RestController
```

```
@RequestMapping(value="hospitals/{hospitalName}/fractures")
```

```
public class FracturesController {
```

```
}
```

# Аннотации @RestController и @RequestMapping



@RestController – это аннотация Java уровня класса, которая сообщает контейнеру Spring, что класс, к которому она применяется, будет использоваться в роли службы REST. Аннотация включает автоматическое преобразование данных, передаваемых в службу, в формат JSON или XML (JSON по умолчанию).

@RequestMapping – это аннотация Java уровня класса и метода, которая сообщает контейнеру Spring, какую конечную точку HTTP служба будет экспортировать.



# Применение @RequestMapping к классу



Определение корневого пути URL ко всем конечным точкам, экспортируемым классом контроллера FracturesController:

```
@RequestMapping(value="hospitals/{hospitalName}/fractures")
```

Все конечные точки службы начинаются с `hospitals/{hospitalName}/fractures`. `{hospitalName}` – это заполнитель, который указывает, что URL будет параметризован значением `hospitalName`, передаваемым в каждом вызове. Он позволяет различать клиентов, которые используют службу.

# Создание класса модели



Создание класса Fractures производится следующим образом:

- щелкнуть ПКМ на пакете `com.hospitals.fractures` и создать новый пакет `model`;
- щелкнуть ПКМ на пакете `model` и создать новый класс `Fractures`.

# Код класса модели



```
package com.hospitals.fractures.model;
```

```
import lombok.Getter;
```

```
import lombok.Setter;
```

```
import lombok.ToString;
```

```
@Getter
```

```
@Setter
```

```
@ToString
```

```
public class Fractures {
```

```
    private int id;
```

```
    private String hospitalName;
```

```
    private String boneType;
```

```
    private String segmentBone;
```

```
    private String fractureType;
```

```
    private boolean infection;
```

```
    private int patientsNumber;
```

```
}
```

**Lombok** – это библиотека для сокращения кода в классах и расширения функциональности языка Java. Подключается к среде разработки или инструменту сборки приложений Maven в качестве плагина.

Одним из недостатков Java является необходимость написания шаблонного кода для конструкторов классов, методов доступа к данным (getter, settter), метода toString() и т.д. Lombok основана на использовании аннотаций.

## Преимущества Lombok:

- многократно сокращает шаблонный код;
- экономит время разработчиков;
- улучшает читаемость кода.



- `@Getter` и `@Setter` – это аннотации, которые предоставляют геттеры и сеттеры для поля. Они используются как на уровне поля, так и на уровне класса.
- `@ToString` – это аннотация, которая переопределяет метод `toString()` и создает для него реализацию по умолчанию. Она выводит имена класса и полей по порядку, разделяя их запятыми.

# Установка Lombok в IntelliJ IDEA. Шаг 1



Добавить зависимость в файл pom.xml

```
<dependency>  
    <groupId>org.projectlombok</groupId>  
    <artifactId>lombok</artifactId>  
    <version>1.18.28</version>  
    <scope>provided</scope>  
</dependency>
```

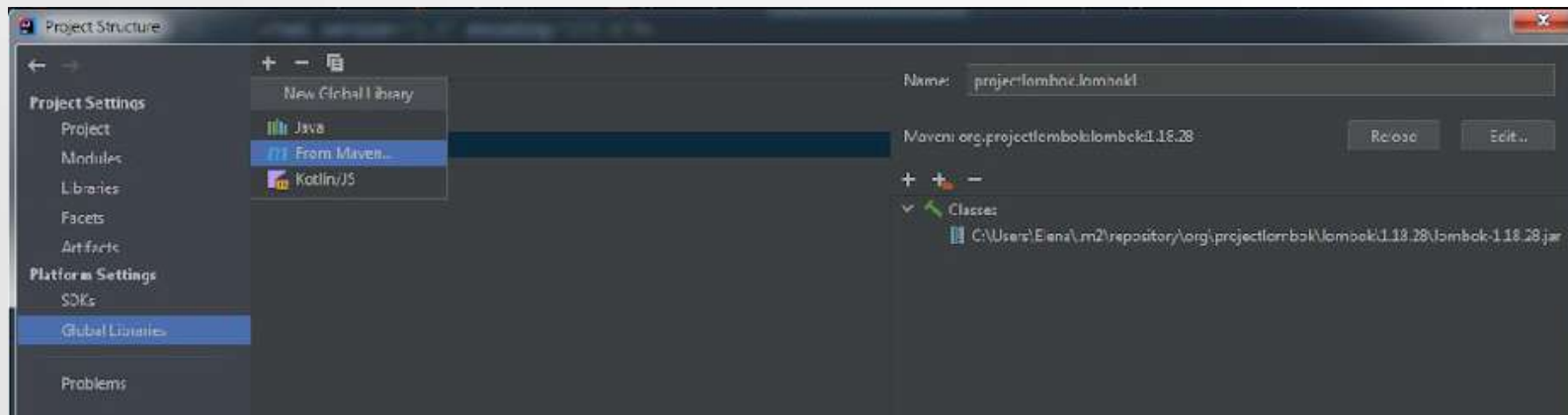
Текущая версия Lombok 1.18.28 (<https://projectlombok.org/changelog>)

# Установка Lombok в IntelliJ IDEA. Шаг 2



Добавить библиотеку `org.projectlombok:lombok:1.18.28` через пункт меню **File -> Project Structure -> Global Libraries**.

После загрузки библиотека появится в ветке проекта **External Libraries**.



# Создание класса службы



Создание класса `FracturesService` производится следующим образом:

- щелкнуть ПКМ на пакете `com.hospitals.fractures` и создать новый пакет `service`;
- щелкнуть ПКМ на пакете `model` и создать новый класс `FracturesService`.

Данный класс используется для разработки логики различных микросервисов в классе контроллера.



# Код класса службы



```
package com.hospitals.fractures.service;
```

```
import java.util.Random;
```

```
import org.springframework.stereotype.Service;
```

```
import com.hospitals.fractures.model.Fractures;
```

```
@Service
```

```
public class FracturesService {
```

```
    public Fractures getFractures(String hospitalName, String boneType, int patientsNumber){
```

```
        Fractures fractures = new Fractures();
```

```
        fractures.setId(new Random().nextInt(1000));
```

```
        fractures.setHospitalName(hospitalName);
```

```
        fractures.setBoneType(boneType);
```

```
        fractures.setSegmentBone("diaphyseal");
```

```
        fractures.setFractureType("closed");
```

```
        fractures.setInfection(false);
```

```
        fractures.setPatientsNumber(patientsNumber);
```

```
        return fractures;
```

```
    }
```

@Service - это аннотация, объявляющая, что класс представляет собой сервис для реализации бизнес-логики.

# Код класса службы



```
public String createFractures(Fractures fractures, String hospitalName){
    String responseMessage = null;
    if(fractures != null) {
        fractures.setHospitalName(hospitalName);
        responseMessage = String.format("This is the post and the object is: %s", fractures.toString());
    }

    return responseMessage;
}
}
```

# Обновление кода класса контроллера



```
package com.hospitals.fractures.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import com.hospitals.fractures.model.Fractures;
import com.hospitals.fractures.service.FracturesService;
```

```
@RestController
@RequestMapping(value="hospitals/{hospitalName}/fractures")
public class FracturesController {
```

```
    @Autowired
    private FracturesService fracturesService;
```

```
    @GetMapping(value="/{boneType}/{patientsNumber}")
    public ResponseEntity<Fractures> getFractures(
        @PathVariable("hospitalName") String hospitalName,
        @PathVariable("boneType") String boneType,
        @PathVariable("patientsNumber") int patientsNumber) {
```

```
        Fractures fractures = fracturesService.getFractures(hospitalName, boneType, patientsNumber);
        return ResponseEntity.ok(fractures);
    }
```

```
}
```

**@GetMapping** - это аннотация, которая используется для сопоставления HTTP GET запросов с определенными методами контроллера. Альтернативно может быть использована аннотация `@RequestMapping(value="/{boneType}/{patientsNumber}", method = RequestMethod.GET)`.

## Создание конечной точки:

1. `@RequestMapping(value="hospitals/{hospitalName}/fractures")` - аннотация на уровне класса с корневым путем;
2. `@GetMapping(value="/{boneType}/{patientsNumber}")` - аннотация на уровне метода с расширением данного пути.

**Конечная точка:** `hospitals/{hospitalName}/fractures/{boneType}/{patientsNumber}`  
Все HTTP-запросы с данным URL будут передаваться контроллеру.

# Аннотации @PathVariable и @Autowired



**@PathVariable** - это аннотация, которая отображает значения параметров из URL (например, {hospitalName}) в параметры метода.

```
@PathVariable("hospitalName") String hospitalName,  
@PathVariable("boneType") String boneType,  
@PathVariable("patientsNumber") int patientsNumber) {
```

**@Autowired** - это аннотация, которая применяется к полям, методам и конструкторам и необходима для автоматического обнаружения компонентов и внедрения взаимодействующих компонентов в наш компонент.

**ResponseEntity** – класс для возврата ответов. Объект ResponseEntity представляет весь HTTP-ответ, включая код состояния, заголовки и тело.

```
return ResponseEntity.ok(fractures);
```

В коде выше возвращаем объект и код 200 (Http-статус ответа – OK).

# Обновление кода класса контроллера



```
package com.hospitals.fractures.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.DeleteMapping;

import com.hospitals.fractures.model.Fractures;
import com.hospitals.fractures.service.FracturesService;

@RestController
@RequestMapping(value="hospitals/{hospitalName}/fractures")
public class FracturesController {

    @Autowired
    private FracturesService fracturesService;

    @GetMapping(value="/{boneType}/{patientsNumber}")
    public ResponseEntity<Fractures> getFractures(
        @PathVariable("hospitalName") String hospitalName,
        @PathVariable("boneType") String boneType,
        @PathVariable("patientsNumber") int patientsNumber) {

        Fractures fractures = fracturesService.getFractures(hospitalName, boneType, patientsNumber);
        return ResponseEntity.ok(fractures);
    }
}
```

# Обновление кода класса FracturesController



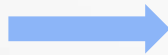
**@PostMapping**

```
public ResponseEntity<String> createFractures(  
    @PathVariable("hospitalName") String hospitalName,  
    @RequestBody Fractures request) {  
    return ResponseEntity.ok(fracturesService.createFractures(request, hospitalName));  
}
```

```
}
```

**@RequestBody** - это аннотация, которая отображает тело HTTP-запроса в объект (в примере, в объект Fractures).

```
Terminal Local Local(2) + v
Microsoft Windows [Version 6.1.7601]
(С) корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.
D:\Lab_Develop_dist_app\fractures>mvnw spring-boot:run
```

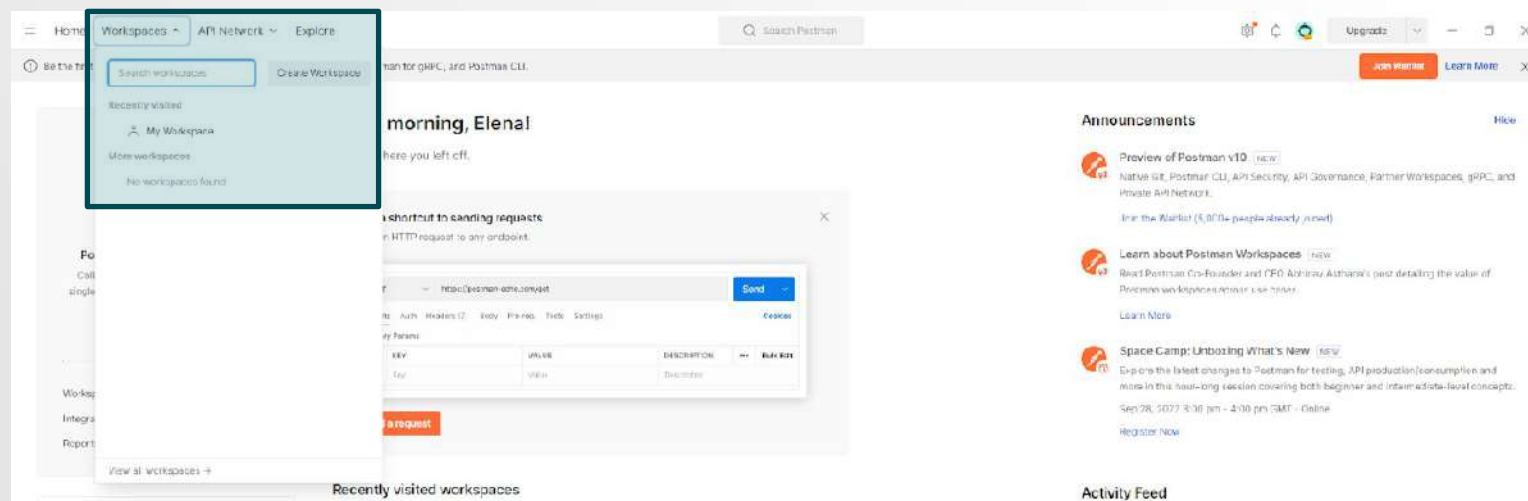
[illegible]



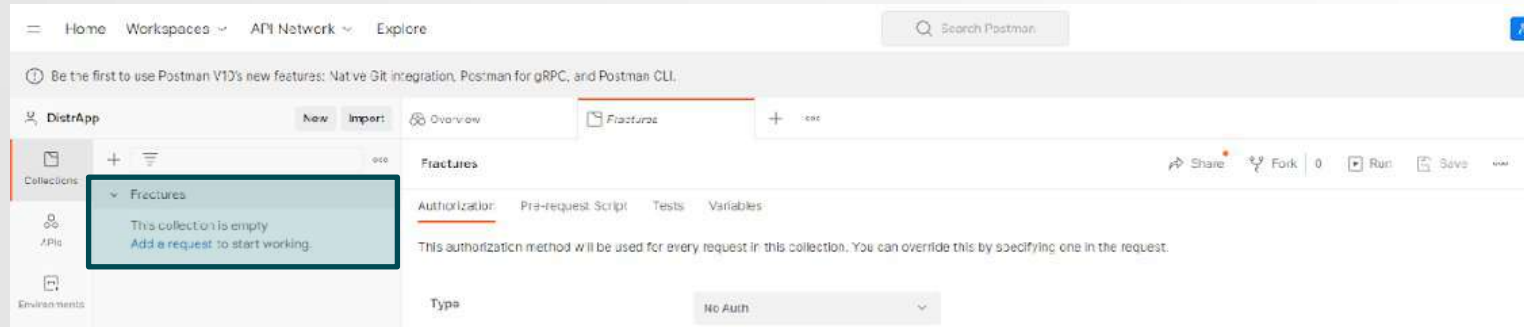
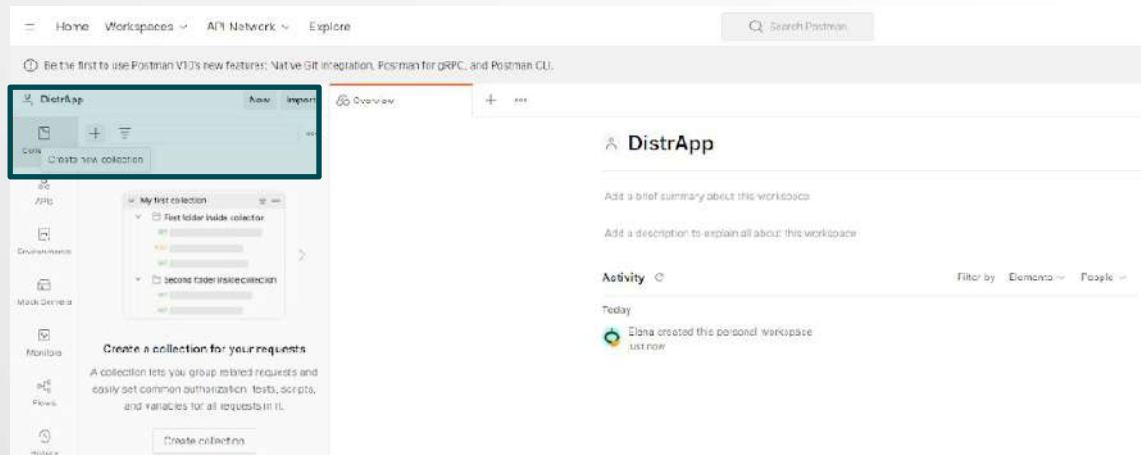
# Postman

**Postman** — это программа, которая представляет собой HTTP-клиент для тестирования API. HTTP-клиент тестирует отправку запросов с клиента на сервер и получение ответа от сервера.

Скачать <https://www.postman.com/downloads/>



# Создание коллекции



# Вызов конечной точки GET через URL



Postman interface showing a GET request configuration and response.

**1** GET `http://localhost:8083/hospitals/Marinsky/fractures/tibia/1532`

**2** Save

**3** Body

**4** Send

Query Params

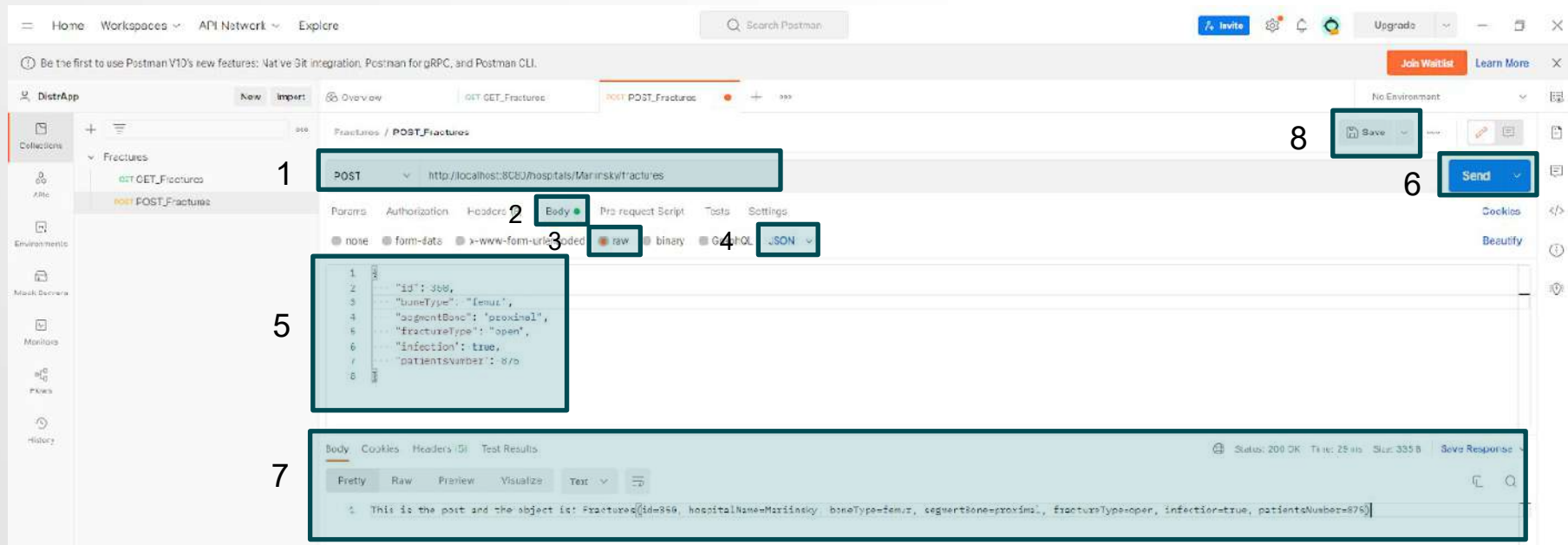
KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

```
{
  "id": 848,
  "hospitalName": "Marinsky",
  "boneType": "tibia",
  "segmentBone": "distal",
  "fractureType": "closed",
  "intention": false,
  "patientsNumber": 1532
}
```

Status: 200 OK Time: 281 ms Size: 311 B Save Response

# Вызов конечной точки POST через URL



The screenshot displays the Postman interface with a POST request configured for the endpoint `http://localhost:8080/hospitals/Marinsky/fractures`. The request body is set to raw JSON. The response status is 200 OK, and the response body contains a JSON object representing a fracture record.

Numbered annotations (1-8) highlight the following elements:

1. The `POST` method and the request URL.
2. The `Body` tab in the request configuration.
3. The `raw` radio button for the body type.
4. The `JSON` format dropdown.
5. The raw JSON body text: 

```
{
  "id": 356,
  "boneType": "femur",
  "segmentBase": "proximal",
  "fractureType": "open",
  "infection": true,
  "patientsNumber": 876
}
```
6. The `Send` button to execute the request.
7. The response body tab showing the JSON response: 

```
{
  "id": 356,
  "hospitalName": "Marinsky",
  "boneType": "femur",
  "segmentBase": "proximal",
  "fractureType": "open",
  "infection": true,
  "patientsNumber": 876
}
```
8. The `Save` button to save the request.

1. Провести анализ предметной области на предмет выявления её характеристик (найти три научных статьи с высоким индексом цитируемости на <https://scholar.google.com/>).
2. Реализовать классы контроллера, модели и службы.
  - Количество полей в классе модели – не менее шести.
  - Класс службы должен содержать четыре метода (создание, чтение, модификация, удаление).
  - Класс контроллера должен содержать четыре метода, которые реализуют HTTP-запросы POST, GET, PUT, DELETE соответственно.
3. Выполнить запуск приложения.
4. Выполнить обращение к конечным точкам (запросы POST, GET, PUT, DELETE) с использованием Postman.

# Предметная область гр. 9307 (бригада - 2 человека)



Вариант	Предметная область
1	Промышленные роботы
2	Носимые медицинские датчики
3	Биометрические терминалы распознавания лиц
4	Автономные датчики движения
5	Умный дом
6	Интеллектуальные парковочные системы
7	Умная упаковка
8	Интеллектуальные системы управления трафиком

# Предметная область гр. 9308 (бригада - 2 человека)



Вариант	Предметная область
1	Сахарный диабет
2	Аритмия
3	Отит
4	Пиелонефрит
5	Кишечная инфекция
6	Пищевая аллергия
7	Конъюнктивит
8	Радикулит
9	Пневмония
10	Панкреатит
11	Артрит
12	Холецистит
13	Фарингит

# Предметная область гр. 9387 (бригада - 2 человека)



Вариант	Предметная область
1	Беспилотные летательные аппараты
2	Истребители
3	Железнодорожный пассажирский транспорт
4	Почтовые услуги
5	Строительство мостов
6	Ледоколы
7	Автомобильный грузовой транспорт
8	Гражданские авиалайнеры
9	Строительство автомобильных дорог
10	Аэропорты