МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»

(БГТУ им. В. Г. Шухова)

Курсовая работа

Дисциплина «Интерфейсы вычислительных систем»

На тему: «Реализация клиент-серверного приложения с использованием Spring Framework»

Выполнила: студентка группы

ВТ-41 Сидорова А.С.

Проверил: Торопчин Д. А.

г. Белгород

2020 г.

**Оглавление**

[Введение 3](#_Toc61473417)

[Глава 1. Теоретические сведения 4](#_Toc61473418)

[1.1 Java 4](#_Toc61473419)

[1.2 Spring Framework 6](#_Toc61473420)

[1.3 Hibernate 9](#_Toc61473421)

[1.4 Apache Maven 11](#_Toc61473422)

[Глава 2. Разработка приложения 13](#_Toc61473423)

[2.1 Исследование предметной области 13](#_Toc61473424)

[2.2 Реализация моделей 15](#_Toc61473425)

[2.3 Репозитории и контроллеры 15](#_Toc61473426)

[Глава 3. Тестирование ПО 16](#_Toc61473427)

[Заключение 20](#_Toc61473428)

[Список литературы 21](#_Toc61473429)

[Приложения 22](#_Toc61473430)

[Приложение А. Содержимое файла Comment.java: 22](#_Toc61473431)

[Приложение Б. Содержимое файла PetService.java: 23](#_Toc61473432)

[Приложение В. Содержимое файла CommentController.java: 24](#_Toc61473433)

[Приложение Г. Содержимое файла PetServiceController.java: 25](#_Toc61473434)

[Приложение Д. Содержимое файла PetServiceRepository.java: 26](#_Toc61473435)

[Приложение Д. Содержимое файла CommentRepository.java: 27](#_Toc61473436)

# **Введение**

На данный момент, на рынке услуг существует довольно большое количество салонов по уходу за животными. Люди стали более активно следить за своей внешностью своих питомцев, и услуги салонов для животных стали более востребованными. Многие владельцы бизнесов осознали, что присутствие сайтов является обязательной составляющей результативной маркетинговой стратегии развития бизнеса, поэтому разработка сайтов актуальна и развивается очень активно.

Записаться на стрижку, посмотреть каталог мастеров, выбрать удобное

для себя время– все это можно и удобно сделать в режиме online.

Целью данной курсовой работы является разработка и реализация клиент-серверного приложения салона по уходу за домашними животными «Луиза», в котором в качестве серверной части будет выступать приложение, реализованное на языке Java с использованием Spring Framework, а в качестве клиентской части – одностраничное приложение, реализованное на языке JavaScript с помощью фреймворка React.js.

# **Глава 1.** **Теоретические сведения**

## **Java**

**Java** — мультифункциональный объектно-ориентированный язык со строгой типизацией.

С **мультифункциональностью** всё достаточно просто: на Java—можно разрабатывать десктопные приложения, приложения под Android, заниматься веб-разработкой.

**Строгая (сильная) типизация** не позволяет смешивать в выражениях разные типы и не выполняет автоматически неявные преобразования. Это добавляет мороки: какие-то части приходится прописывать самому, а свободы у вас меньше, — зато в обмен на это вы получаете надёжность.

**Объектно-ориентированный** язык — это язык, созданный по модели объектно-ориентированного программирования. В ней существуют классы и объекты. **Классы —** это типы данных, а **объекты —** представители классов. Мы создаём их сами, даём названия и присваиваем им свойства и операции, которые с ними можно выполнять. Это как конструктор, который позволяет построить то, что мы хотите. Именно с помощью этой системы объектов в основном программируют на Java.

У всех качеств Java, будь то строгая типизация или объектная ориентированность, есть свои плюсы и минусы, а ещё они есть у самой Java как у языка.

#### Плюсы

* Независимость — ваш код будет работать на любой платформе, которая поддерживает Java.
* Надёжность — в немалой мере достигается благодаря строгой статической типизации.
* Мультифункциональность.
* Сравнительно простой синтаксис.
* Java — основной язык для Android-разработки.
* Объектно-ориентированное программирование (ООП) тоже приносит много выгод:

1. параллельная разработка;
2. гибкость;
3. одни и те же классы можно использовать много раз;
4. код хорошо организован, и его легче поддерживать.

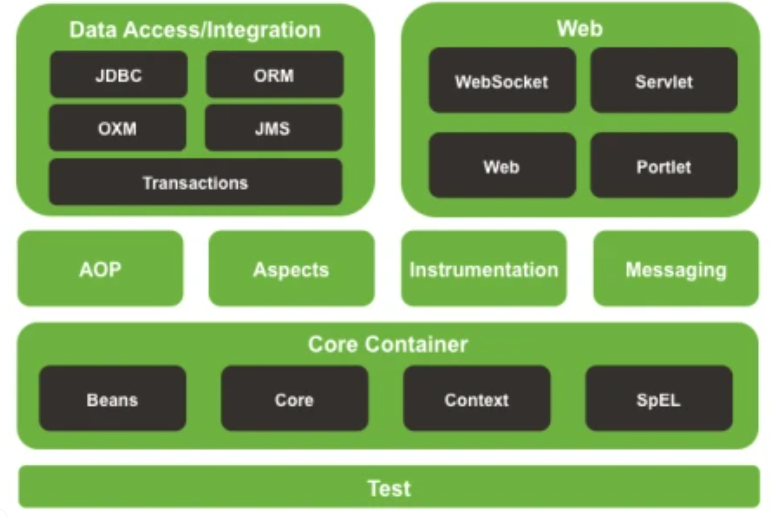
#### Минусы

* Низкая скорость (по сравнению с С и С++).
* Требует много памяти.
* Нет поддержки низкоуровневого программирования (Java — высокоуровневый язык). Например, у неё нет указателей.
* С 2019 года обновления для бизнеса и коммерческого использования стали платными.
* Для ООП нужен опыт, а планирование новой программы занимает много времени.

## **Spring Framework**

**Spring Framework**, или просто Spring — один из самых популярных фреймворков для создания веб-приложений на Java.

**Фреймворк** — это что-то похожее на библиотеку, но есть один момент. Грубо говоря, используя библиотеку, мы просто создаем объекты классов, которые в ней есть, вызываем нужные нам методы, и таким образом получаем нужный нам результат.



Как видно, у спринга модульная структура. Это позволяет подключать только те модули, что нам нужны для нашего приложения и не подключать те, которыми мы заведомо не будем пользоваться. Этот подход и помог спрингу обойти своего конкурента в то время (EJB) и захватить лидерство.

На изображении видно, что спринг фреймворк состоит как-бы из нескольких модулей:

* data access;
* web;
* core;
* и других.

Модуль data access содержит в себе средства для работы с данными (в основном, с базами данных), web — для работы в сети (в том числе и для создания веб-приложений).

Вот некоторые наиболее важные особенности Spring Framework:

* Предопределенные шаблоны
* облегченный
* Быстрое развитие
* Мощная абстракция
* Предлагает множество ресурсов
* Декларативная поддержка
* Предлагает комплексные инструменты

Преимущества Spring Framework

* Spring позволяет разработчикам разрабатывать приложения корпоративного класса с помощью POJO.
* Предлагает шаблоны для Hibernate, JDBC, Hibernate, JPA и т. Д., Чтобы уменьшить объем написанного кода.
* Предоставляет абстракцию для Java Enterprise Edition (JEE).
* Вы можете организовать по модульному принципу. Так что, если количество пакетов и классов является существенным, вам нужно только об этом и игнорировать все остальное.
* Он предлагает декларативную поддержку транзакций, форматирования, проверки, кэширования и т. Д.
* Приложение, разработанное с использованием Spring, является простым, поскольку код, зависящий от среды, перемещен в эту среду.

**Spring Web MVC Framework** предлагает архитектуру модель-представление-контроллер и предлагает компоненты, которые помогают вам быть гибкими и слабо связанными веб-приложениями.

Шаблон MVC позволяет разделять различные аспекты приложения, предлагая слабую связь между этими элементами. Spring MVC также помогает вам создавать гибкие и слабо связанные веб-приложения.

Дизайн MVC также позволяет разделить бизнес-логику, логику представления и логику навигации. Он также предлагает элегантное решение для использования MVC в Spring Framework с помощью DispatcherServlet.

Работа MVC:

* DispatcherServlet получает запрос.
* После этого DispatcherServlet связывается с HandlerMapping. Он также отзывает контроллер, связанный с этим конкретным запросом.
* Контроллер обрабатывает этот запрос, вызывая методы службы и объект ModelAndView, возвращаемый DispatcherServlet.
* Имя представления отправляется ViewResolver для поиска фактического представления для вызова.
* После этого DispatcherServlet передается в View для отображения результата.
* Используя данные модели, представление рендерится и отправляет результат обратно пользователю.

## **Hibernate**

**Hibernate** - одна из наиболее популярных реализаций ORM-модели. Объектно-реляционная модель описывает отношения между программными объектами и записями в БД.

Позволяет сократить объёмы низкоуровневого программирования при работе с реляционными базами данных; может использоваться как процессе проектирования системы классов и таблиц «с нуля», так и для работы с уже существующей базой.

Библиотека не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных SQL), но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода. Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора данных и преобразования объектов, максимально облегчая перенос (портирование) приложения на любые базы данных SQL.

Hibernate обеспечивает прозрачную поддержку сохранности данных (persistence) для «POJO» (то есть для стандартных Java-объектов); единственное строгое требование для сохраняемого класса — наличие конструктора по умолчанию (без параметров). Для корректного поведения в некоторых приложениях требуется также уделить внимание методам equals() и hashCode().

Hibernate может использоваться как в самостоятельных приложениях Java, так и в программах Java EE, выполняемых на сервере (например, сервлет или компоненты EJB). Также он может включаться как дополнительная возможность к другим языкам программирования. Например, Adobe интегрировал Hibernate в девятую версию ColdFusion (запускаемый на серверах с поддержкой приложений J2EE) с уровнем абстракции новых функций и синтаксиса, приложенных к CFML.

**Отображение** (mapping, сопоставление, проецирование) Java-классов с таблицами базы данных осуществляется с помощью конфигурационных XML-файлов или Java-аннотаций. При использовании файла XML Hibernate может генерировать скелет исходного кода для классов длительного хранения. В этом нет необходимости, если используется аннотация. Hibernate может использовать файл XML или аннотации для поддержки схемы базы данных.

Обеспечиваются возможности по организации отношения между классами «один-ко-многим» и «многие-ко-многим». В дополнение к управлению связями между объектами Hibernate также может управлять рефлексивными отношениями, где объект имеет связь «один-ко-многим» с другими экземплярами своего собственного типа данных.

Hibernate поддерживает отображение пользовательских типов значений. Это делает возможными такие сценарии:

* переопределение типа по умолчанию SQL, Hibernate выбирает при отображении столбца свойства;
* отображение перечисляемого типа Java на поле базы данных, будто они являются обычными свойствами;
* отображение одного свойства в несколько столбцов.

## **Apache Maven**

**Maven** — инструмент для управления и сборки проектов.

Он облегчает жизнь девелоперу на всех стадиях работы: от создания структуры проекта и подключения необходимых библиотек до развертывания продукта на сервере. При работе с любым фреймворком придется использовать Maven.

**Apache Maven** — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM.

Жизненный цикл **maven** проекта – это чётко определённая последовательность фаз. Когда *maven* начинает сборку проекта, он проходит через определённую последовательность фаз, выполняя задачи, указанные в каждой из фаз. *Maven* имеет 3 стандартных жизненных цикла:

* clean — жизненный цикл для очистки проекта;
* default — основной жизненный цикл;
* site — жизненный цикл генерации проектной документации.

Каждый из этих циклов имеет фазы pre и post. Они могут быть использованы для регистрации задач, которые должны быть запущены перед и после указанной фазы.

**Объектная модель проекта (POM)** предоставляет всю конфигурацию для одного проекта. Общая конфигурация охватывает имя проекта, его владельца и его зависимости от других проектов. Также можно настроить отдельные фазы процесса сборки, которые реализованы в виде плагинов. Например, можно настроить подключаемый модуль компилятора для использования Java версии 1.5 для компиляции или указать упаковку проекта, даже если некоторые модульные тесты завершились неудачно.

Maven базируется на plugin-архитектуре, которая позволяет использовать плагины для различных задач (test, compile, build, deploy и т.п). Иными словами, *maven* запускает определенные плагины, которые выполняют всю работу. То есть, если мы хотим научить *maven* особенным сборкам проекта, то необходимо добавить в *pom.xml* указание на запуск нужного плагина в нужную фазу и с нужными параметрами. Это возможно за счет того, что информация поступает плагину через стандартный вход, а результаты пишутся в его стандартный выход.

Количество доступных плагинов очень велико и включает разнотипные плагины, позволяющие непосредственно из *maven* запускать web-приложение для тестирования его в браузере, генерировать Web Services. Главной задачей разработчика в этой ситуации является найти и применить наиболее подходящий набор плагинов.

В простейшем случае запустить плагин просто - для этого необходимо выполнить команду в определенном формате.

# **Глава 2. Разработка приложения**

## **2.1 Исследование предметной области**

В качестве предметной области был выбран сайт по уходу за домашними животными. Данный сайт нацелен на то, чтобы клиентам салона было удобнее ориентироваться в графике работы салона, записи на разные виды услуг, визуализации качества выполнения ухода за питомцами и обратной связи с салоном.

На первом этапе разработки приложения проведем анализ предметной области, выделим сущности и связи между ними; составим диаграмму классов.

Выделим сущности в данной предметной области и обозначим их свойства:

Тэг

* название тэга
* id тэга

Услуга

* название услуги
* описание услуги
* цена за услугу
* id услуги

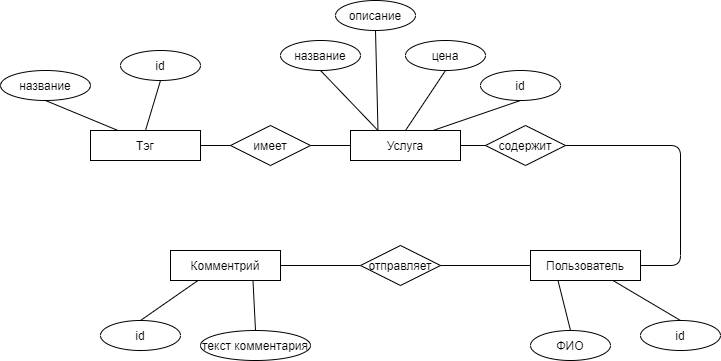
Пользователь

* ФИО
* id пользователя

Комментарий

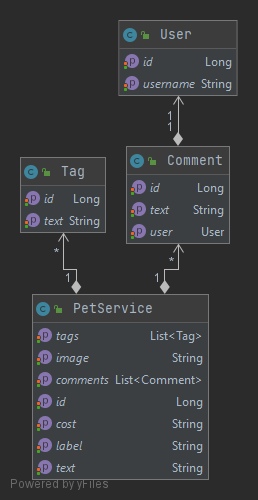
* текст комментария
* id комментария

Составим диаграмму «Сущность-связь»:



*Рис. 2.1.1 ER-диаграмма*

Чтобы реализовать данную структуру сущностей и отношений, построим uml-диаграмму моделей, в которой укажем необходимые поля:



## **2.2 Реализация приложения**

Для предоставления доступа к данным через REST API необходимо спроектировать и реализовать контроллеры, было создано 3 пакета: models, controllers, repositories. Запуск приложения выполняется через файл petsApplication.java:

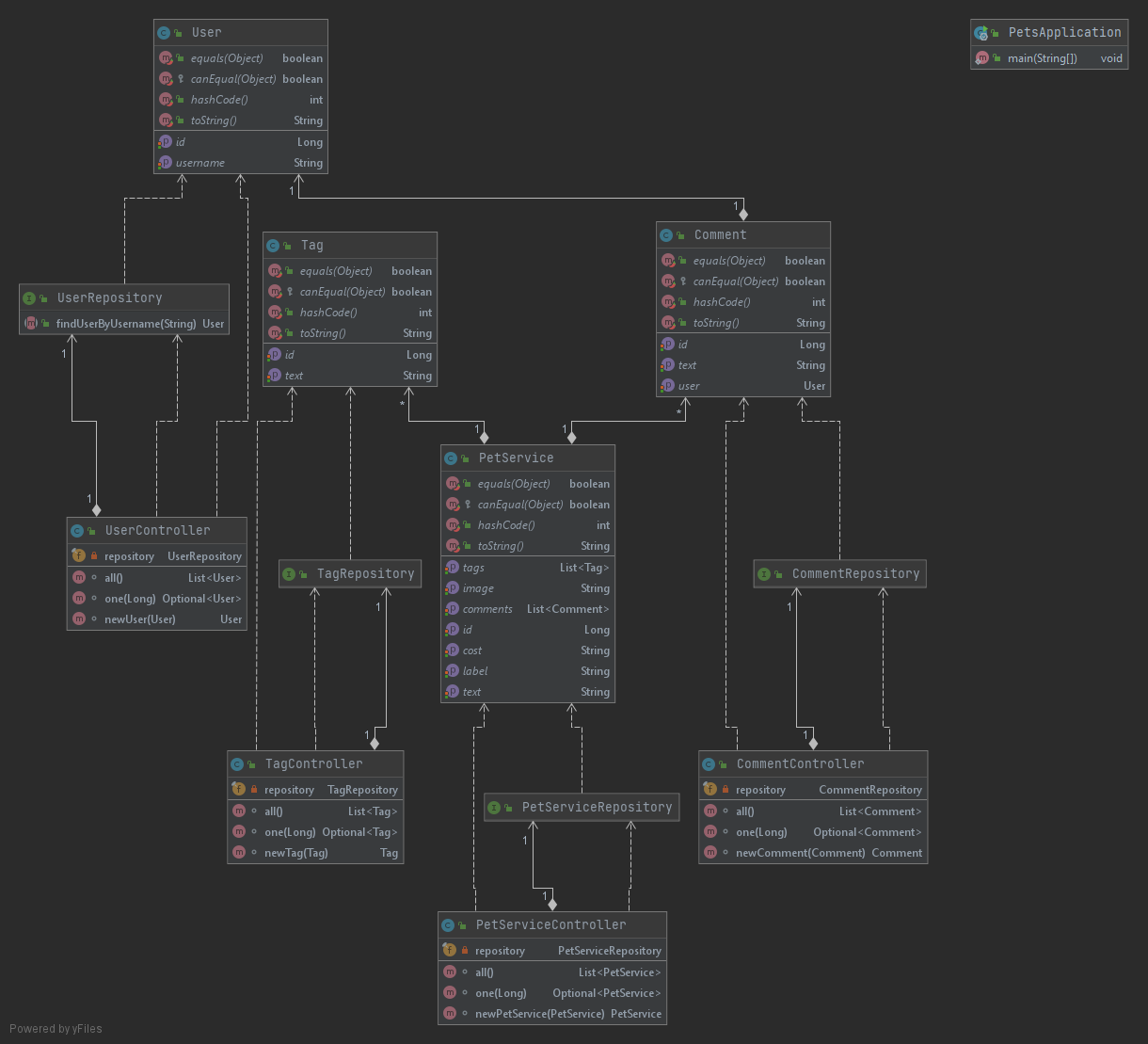
|  |
| --- |
| package ru.bstu.iitus.povtas.vt41.pets;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  @SpringBootApplication @EnableJpaRepositories public class PetsApplication {   public static void main(String[] args) {  SpringApplication.*run*(PetsApplication.class, args);  } } |

В пакете models были созданы модели, реализующие сущности базы данных. Примеры приведены в приложениях А, Б.

Для хранения и поддержки данных для каждой сущности были репозитории, родительским классом которых является JpaRepository.

REST-контроллеры были реализованы в пакете controllers. В них были сконфигурированы те url, по которым клиент может отправлять POST и GET запросы.

В конечном итоге, получим такую структуру приложения:



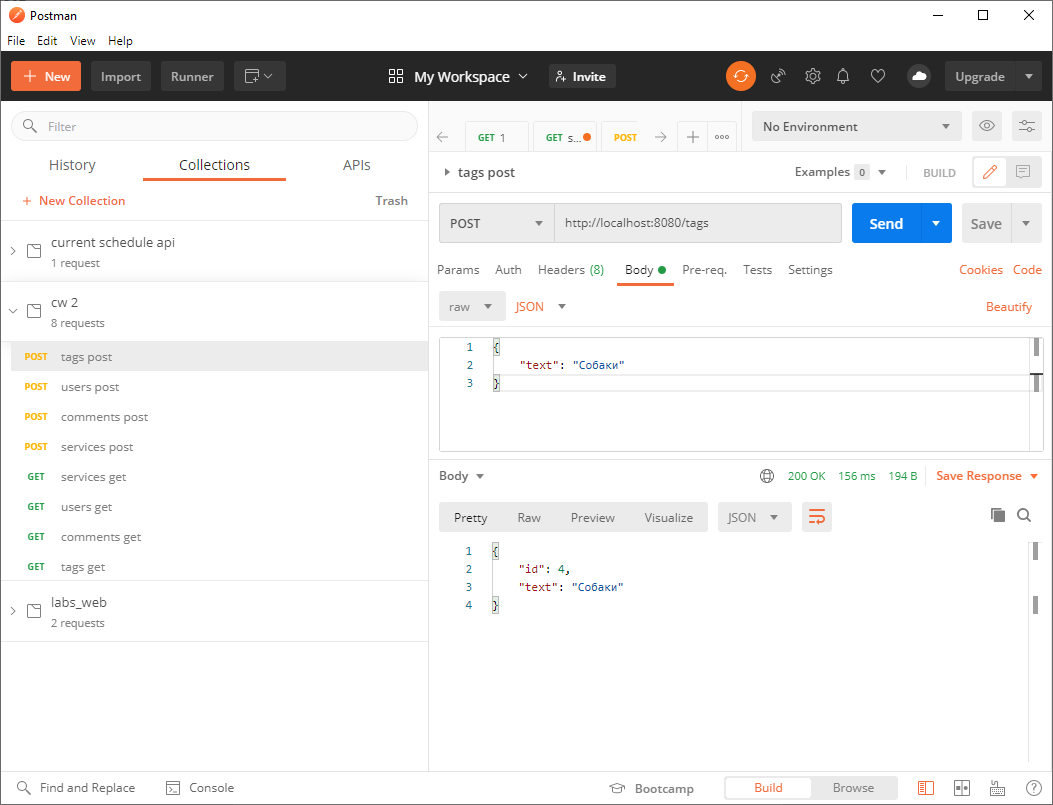
*Рис. 2.2.1 UML-диаграмма классов*

# **Глава 3. Тестирование ПО**

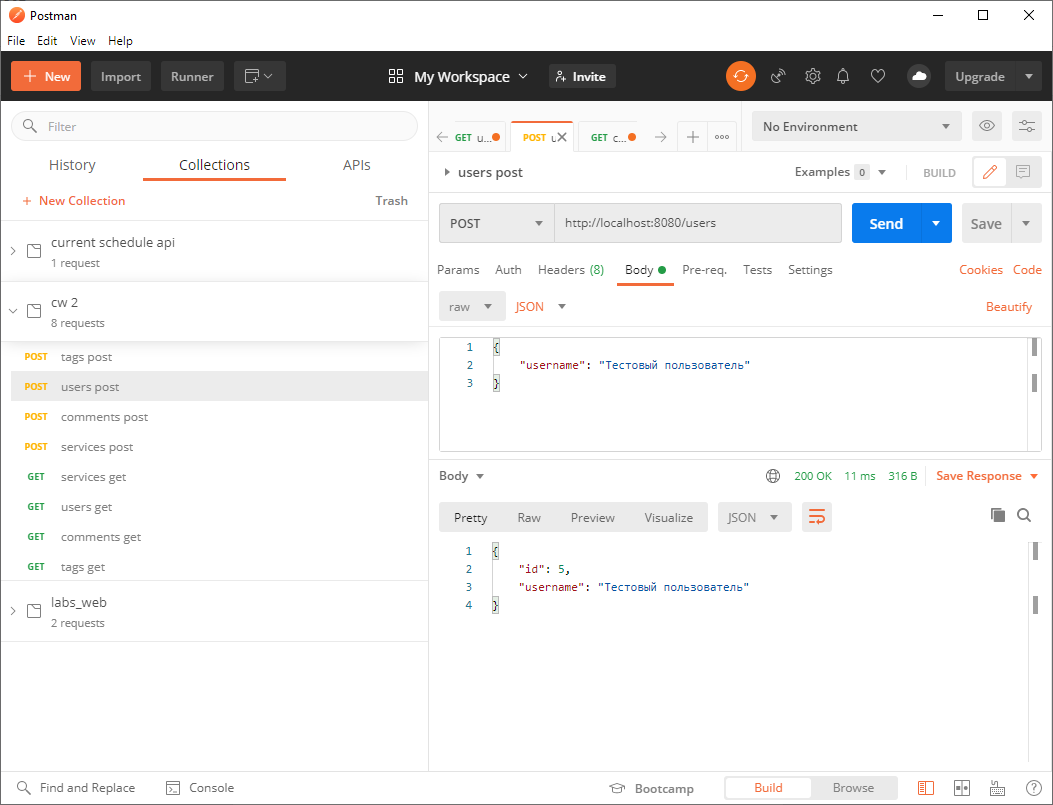
Для тестирования разработанного API выполним POST и GET запросы со следующим содержимым:

|  |  |
| --- | --- |
| Тестируемый URL | Содержимое POST-запроса |
| localhost/tags | {      "text": "Собаки"  } |
| localhost/users | {      "username": "Тестовый пользователь"  } |
| localhost/comments | {      "user": {          "id": 2      },      "text": "Отзыв об услуге"  } |
| localhost/services | {      "label": "Название услуги",      "text": "Описание услуги",      "image": "https://dinozoopasaule.lv/ru/getimage/uploads/news/ikkXHS-2AesJFyftXGg2mEg82YscHLdd.png?w=600&h=400&fit=crop",      "price": "1000 рублей",      "comments": [          {              "id": 3          }      ],      "tags": [          {              "id": 1          }      ]  } |

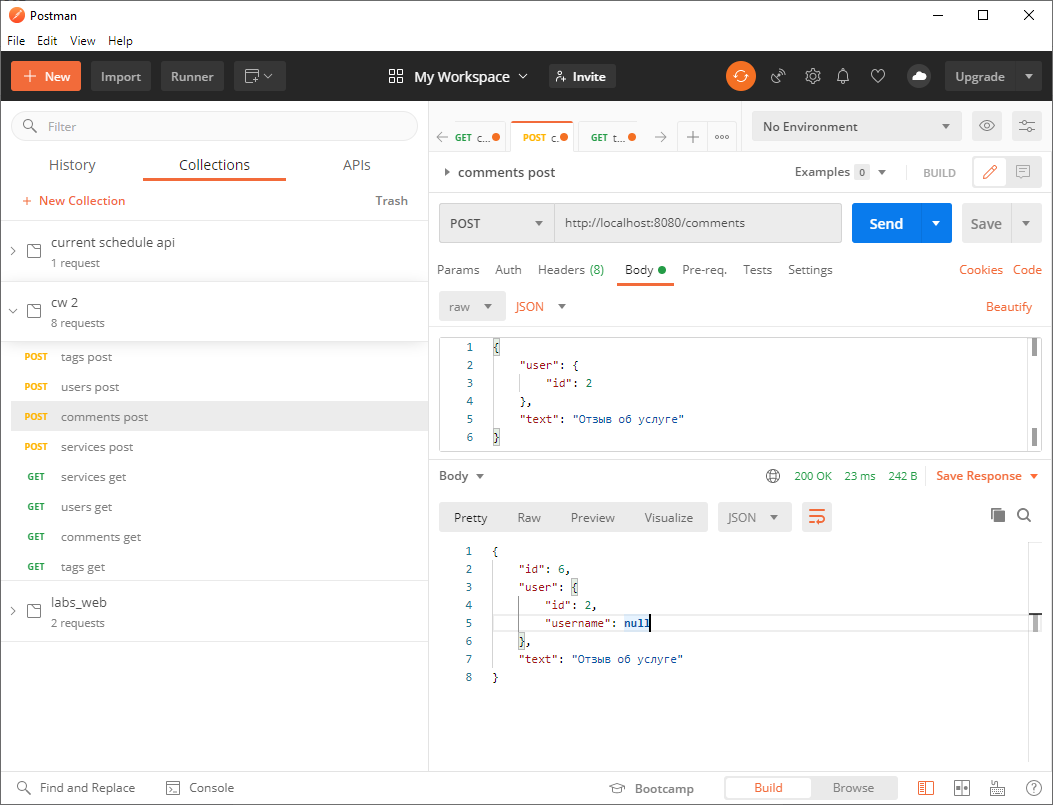
Протестируем разработанное приложение с помощью приложения postman [9]. Для этого в каждый из 4 url отправим запросы с тестовыми данными и проанализируем результат:

**

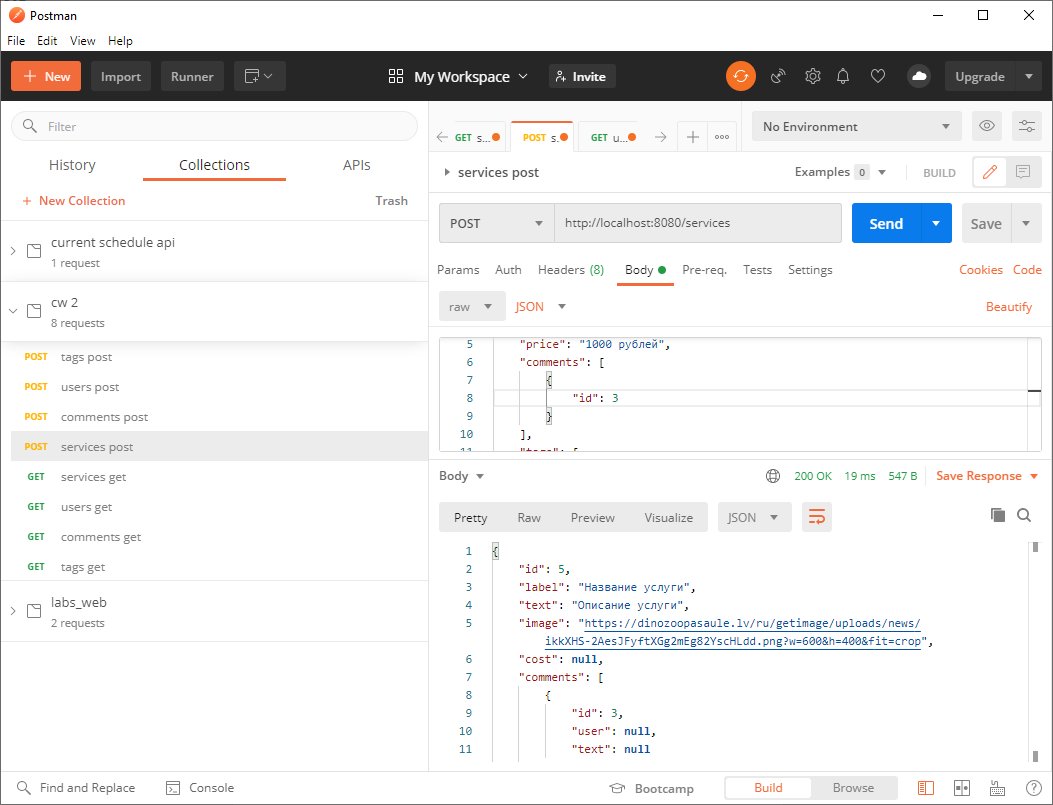
*рис 3.1 Результат POST-запроса №1*

**

*рис 3.2 Результат POST-запроса №2*

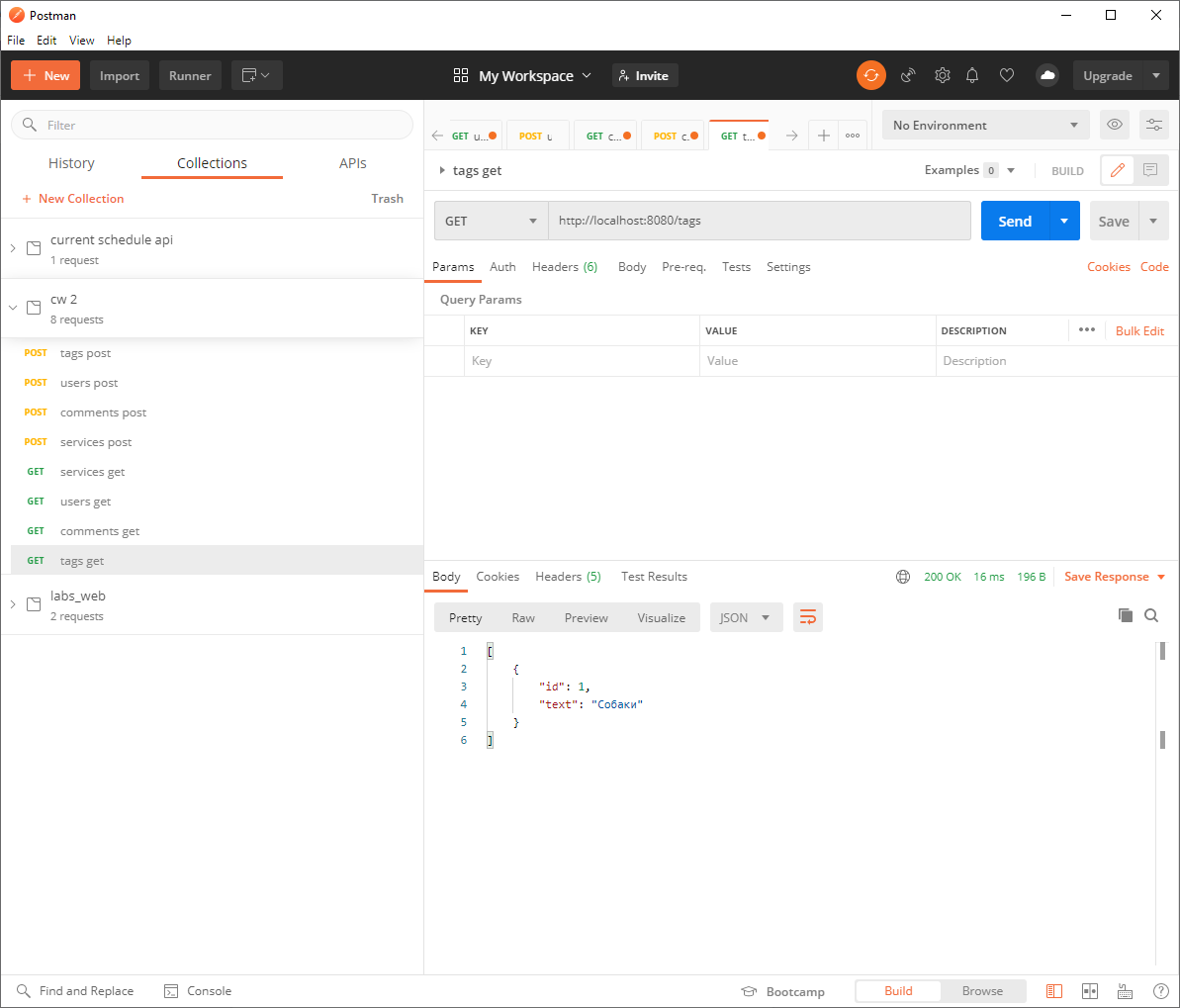
**

*Рис 3.3. Результат POST-запроса №3*

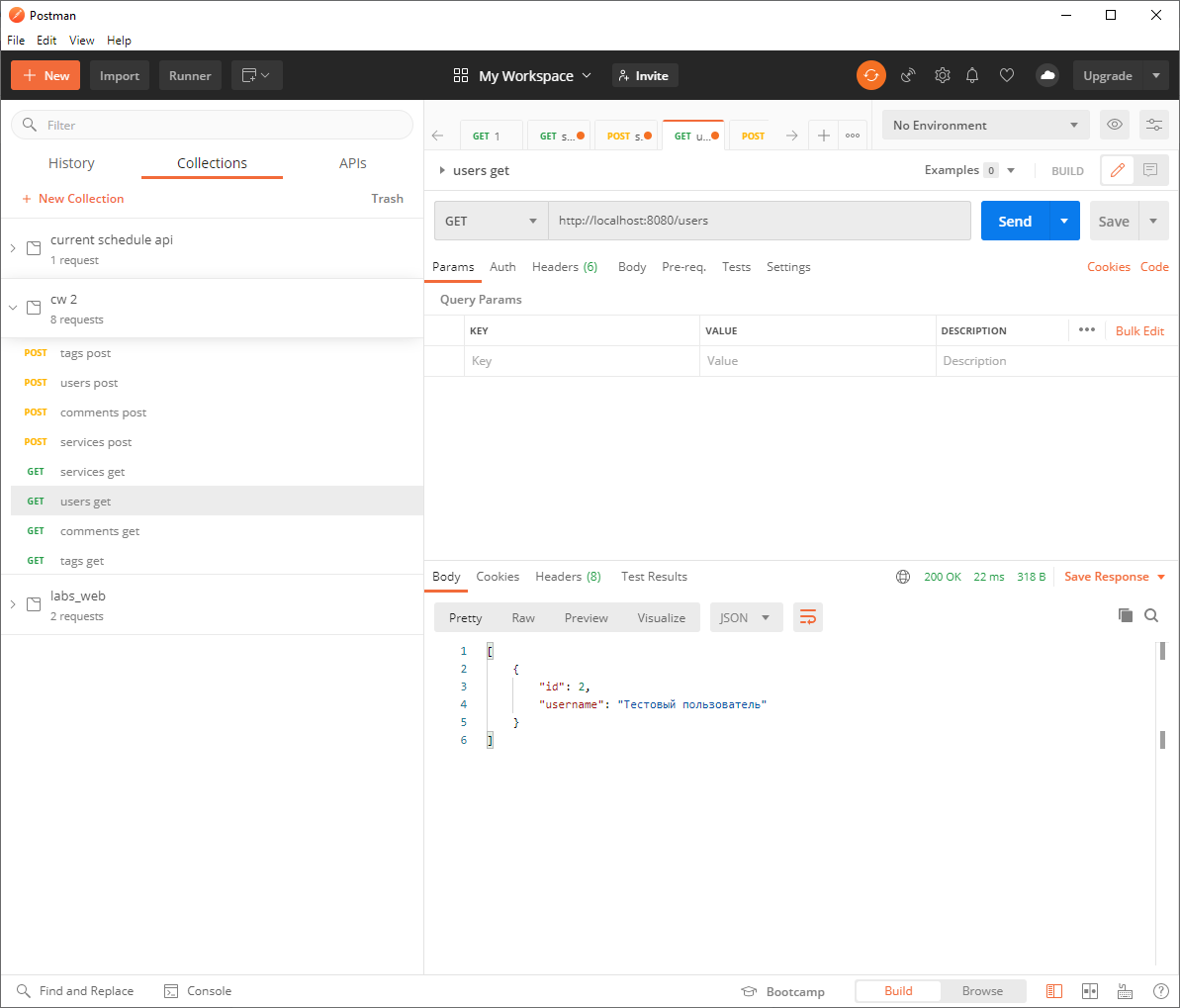
**

*Рис 3.4. Результат POST-запроса №4*

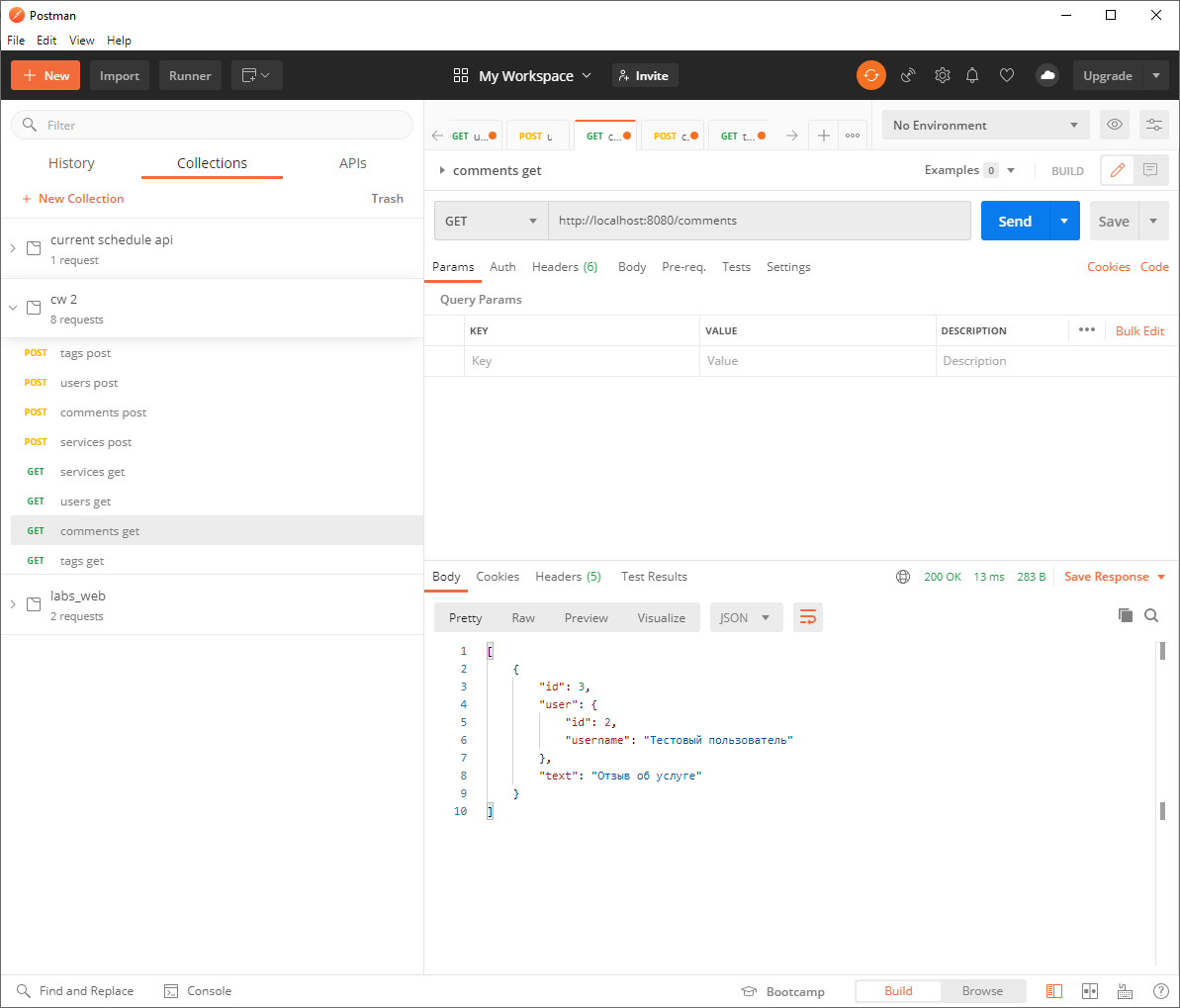
На все запросы от сервера был получен ответ 200 ОК, что свидетельствует об успешном их выполнении. Для проверки правильности загрузки данных выполним GET-запросы по тем же URL:

**

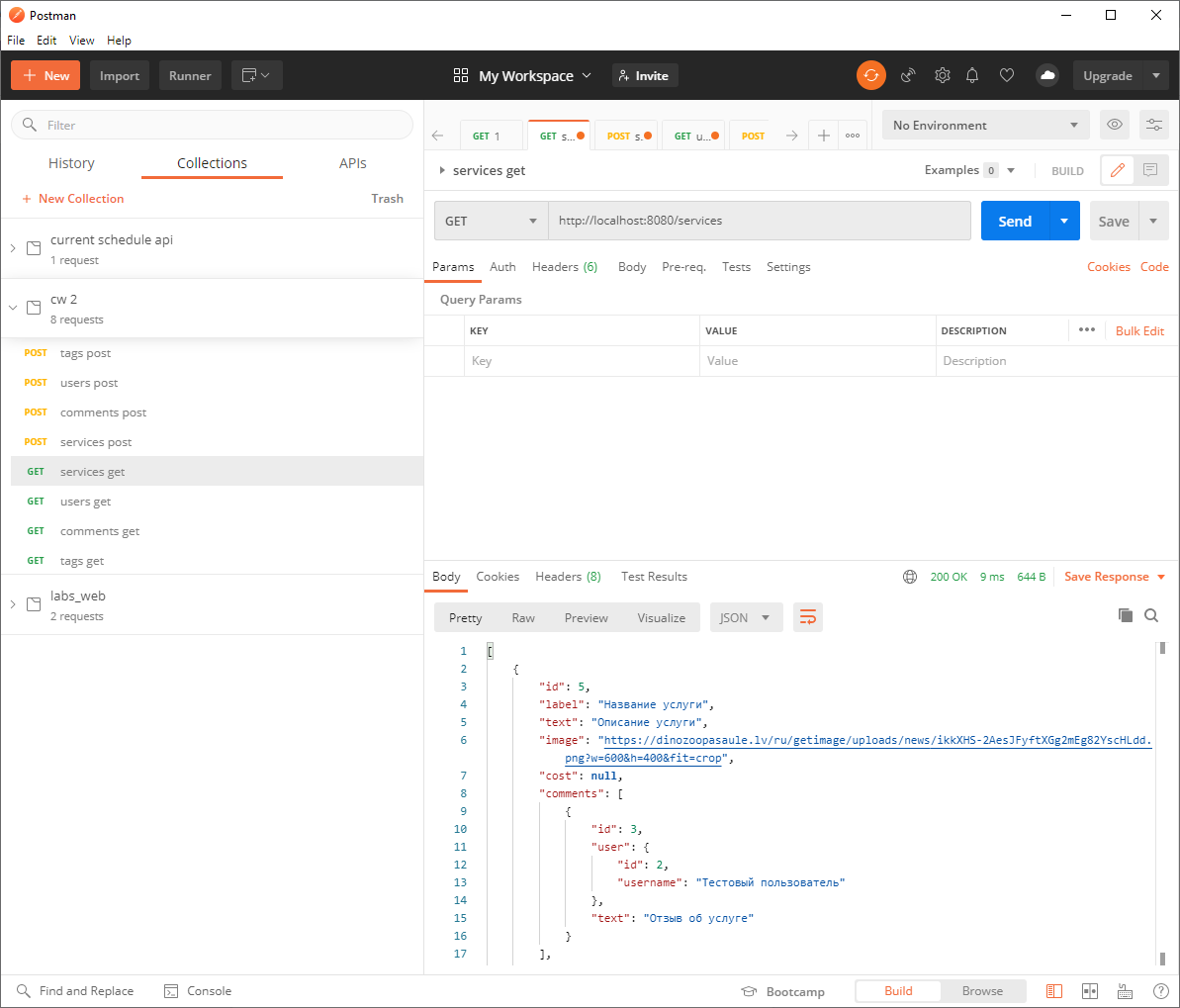
*Рис. 3.5. Результат GET-запроса №1*

**

*Рис. 3.6. Результат GET-запроса №2*

**

*Рис. 3.7. Результат GET-запроса №3*

**

*Рис. 3.8. Результат GET-запроса №4*

Были получены корректные данные, следовательно, API работает верно.

# **Заключение**

В ходе выполнения курсовой работы была выполнена разработка и реализация клиент-серверного приложения салона по уходу за домашними животными «Луиза», в котором за серверную часть будет выступает приложение, реализованное на языке Java с использованием Spring Framework.

Была разработана структура базы данных для хранения информации о объектах предметной области, спроектировано API для клиентского приложения, реализующее поставленные задачи, а именно: возможность получения информации о исполнителях, альбомах, а также поиск по ним.

Полученное приложение было успешно протестировано.

В данном проекте были использованы такие технологии, как Spring Framework, Maven, Hibernate, Project Lombok.

# **Список литературы**

1. JavaRush – Статьи - [Электронный ресурс]. – URL: https://javarush.ru/groups/posts
2. Apache Maven - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Apache\_Maven
3. Tproger – Статьи - [Электронный ресурс]. – URL: https://tproger.ru/articles/spring-modules-overview/
4. Javahelp – Основы Java - [Электронный ресурс]. – URL: https://javahelp.online/osnovy/java-spring
5. Skillbox – Статьи – [Электронный ресурс]. – URL: https://skillbox.ru/media/code/vernyem\_veb\_razrabotke\_byloe\_velichie/
6. Javaonline – Статьи – [Электронный ресурс]. – URL: http://java-online.ru/maven-pom.xhtml
7. Hibernate (библиотека) - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Hibernate\_(библиотека)
8. Project Lombok: Clean, Concise Java Code. – [Электронный ресурс]. – URL: https://www.oracle.com/corporate/features/project-lombok.html
9. Postman | The collaboration platform for API developers [Электронный ресурс]. – URL: https://www.postman.com/

# **Приложения**

## **Приложение А. Содержимое файла Comment.java:**

package ru.bstu.iitus.povtas.vt41.pets.models;  
  
import lombok.Data;  
  
import javax.persistence.\*;  
  
@Data  
@Entity  
public class Comment {  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private Long id;  
 @OneToOne  
 private User user;  
 private String text;  
 public Comment() {}  
}

## **Приложение Б. Содержимое файла PetService.java:**

package ru.bstu.iitus.povtas.vt41.pets.models;  
import lombok.\*;  
import javax.persistence.\*;  
import java.util.List;  
  
@Data  
@Entity  
public class PetService {  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private Long id;  
 private String label;  
 private String text;  
 private String image;  
 private String cost;  
 @OneToMany  
 private List<Comment> comments;  
 @OneToMany  
 private List<Tag> tags;  
 public PetService() {  
  
 }  
}

## **Приложение В. Содержимое файла CommentController.java:**

package ru.bstu.iitus.povtas.vt41.pets.controllers;  
  
import org.springframework.web.bind.annotation.\*;  
import ru.bstu.iitus.povtas.vt41.pets.models.Comment;  
import ru.bstu.iitus.povtas.vt41.pets.repositories.CommentRepository;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
public class CommentController {  
 private final CommentRepository repository;  
  
 CommentController(CommentRepository repository) {  
 this.repository = repository;  
 }  
  
 @GetMapping("/comments")  
 List<Comment> all() {  
 return repository.findAll();  
 }  
  
 @GetMapping("/comments/{id}")  
 Optional<Comment> one(@PathVariable Long id) {  
 return repository.findById(id);  
 }  
  
 @PostMapping("/comments")  
 Comment newComment(@RequestBody Comment comment) {  
 return repository.save(comment);  
 }  
}

## **Приложение Г. Содержимое файла PetServiceController.java:**

package ru.bstu.iitus.povtas.vt41.pets.controllers;  
  
import org.springframework.web.bind.annotation.\*;  
import ru.bstu.iitus.povtas.vt41.pets.models.PetService;  
import ru.bstu.iitus.povtas.vt41.pets.repositories.PetServiceRepository;  
  
import java.util.List;  
import java.util.Optional;  
  
@CrossOrigin(origins = "\*")  
@RestController  
public class PetServiceController {  
 private final PetServiceRepository repository;  
  
 PetServiceController(PetServiceRepository repository) {  
 this.repository = repository;  
 }  
  
 @GetMapping("/services")  
 List<PetService> all() {  
 return repository.findAll();  
 }  
  
 @GetMapping("/services/{id}")  
 Optional<PetService> one(@PathVariable Long id) {  
 return repository.findById(id);  
 }  
  
 @PostMapping("/services")  
 PetService newPetService(@RequestBody PetService service) {  
 return repository.save(service);  
 }  
}

## **Приложение Д. Содержимое файла PetServiceRepository.java:**

package ru.bstu.iitus.povtas.vt41.pets.repositories;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import ru.bstu.iitus.povtas.vt41.pets.models.PetService;  
  
@Repository  
public interface PetServiceRepository extends JpaRepository<PetService, Long> {  
  
}

## **Приложение Д. Содержимое файла CommentRepository.java:**

package ru.bstu.iitus.povtas.vt41.pets.repositories;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import ru.bstu.iitus.povtas.vt41.pets.models.Comment;  
  
@Repository  
public interface CommentRepository extends JpaRepository<Comment, Long> {  
  
}