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: «Разработка одностраничного приложения с использованием Spring Framework»

Выполнил: студент группы ВТ-42 Воскобойников И. С.
Проверил: Торопчин Д. А.

г. Белгород
2021 г.

Оглавление

Введение

Java является универсальным языком программирования. Этот язык может работать на любой процессорной архитектуре. Он широко используется в разработке десктопных, мобильных и веб-приложений, в банковской, научной и промышленной областях. Фреймворк Spring, предназначенный для Java, позволяет программисту ускорить разработку приложений, в том числе, предназначенных для веб. В настоящее время такие приложения, как правило, выполнены по технологии Single Page Application (SPA). SPA – одностраничное веб-приложение – это популярный способ создания кроссплатформенных приложений с клиент-серверной архитектурой. Клиентская часть приложения представляет собой веб-страницу, работающую в интернет-браузере. Клиент взаимодействует с сервером по API, основанному, например, на архитектуре REST API. Такой подход позволяет с легкостью создавать интерактивные веб-страницы и позволять пользователю взаимодействовать с ними, не ожидая перезагрузки страниц. Java в данном случае может использоваться в качестве языка программирования на сервере.

Цель курсового проекта – реализация одностраничного приложения, бэкенд которого выполнен с использованием языка Java и фреймворка Spring. Бэкенд должен предоставлять REST API для использования клиентским приложением.

В качестве предметной области курсового проекта выбран сервис-агрегатор предложений о продаже автомобилей. Сервис предоставляет возможность просмотреть информацию о ценах автомобиля у различных дилеров. Автомобили можно выбирать исходя из марки, модели, комплектации, цвета, и т. д. Для помощи в выборе автомобиля имеется система отзывов.

Глава 1. Теоретические сведения

1.1 SPA

Single Page Application (SPA), одностраничное приложение – это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript документы, обычно, посредством AJAX (Asynchronous JavaScript and XML), то есть, фоновом обмене данных браузера с веб-сервером. Данный подход выгодно отличается тем, что при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

Одностраничные приложения напоминают «нативные» приложения, но исполняются в рамках браузера, а не в собственном процессе операционной системы.

1.2 REST API

REST API – это подход к построению API (Application Programming Interface, программный интерфейс приложения)[3]. REST происходит от англ. REpresentational State Transfer – передача состояния представления. REST представляет собой архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно GET или POST; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве параметров запроса.

Для веб-служб, построенных с учётом REST (то есть не нарушающих накладываемых им ограничений), применяют термин «RESTful»[4].

REST приложения должны быть построены с учетом следующих обязательных ограничений [5]:

1. Модель клиент-сервер
2. Отсутствие состояния
3. Кэширование
4. Единообразие интерфейса
5. Архитектура, поддерживающая слои
6. Код по требованию.

Считается, что приложение, построенное с использованием архитектуры REST, обладает следующими преимуществами [6]:

- Надёжность
- Производительность
- Масштабируемость
- Прозрачность системы взаимодействия
- Простота интерфейсов
- Портативность компонентов

1.3 Java

Java — строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Язык и основные реализующие его технологии распространяются по лицензии GPL [7].

Java выпущена компанией Sun Microsystems в 1995 г. Java отличается быстротой, высоким уровнем защиты и надежностью. Сегодня этот язык является основой практически для всех типов сетевых приложений и всеобщим стандартом для разработки и распространения встроенных и мобильных приложений, игр, веб-контента и корпоративного программного обеспечения. В мире насчитывается более 9 миллионов специалистов, разрабатывающих приложения на Java, которая позволяет эффективно разрабатывать, внедрять и использовать превосходные приложения и услуги [8].

В состав языка Java входят следующие компоненты:

1. JVM (Java Virtual Machine) – виртуальная машина Java, которая исполняет скомпилированную компилятором javac в байт-код программу.
2. JRE (Java Runtime Environment) – минимальная среда, необходимая для исполнения Java-приложений. JRE состоит из Java Virtual Machine (JVM), базовых классов платформы Java и вспомогательных библиотек платформы Java.
3. JDK (Java Development Kit) – комплект разработчика приложений на языке Java. Он включает в себя компилятор, стандартные библиотеки классов Java, документацию, исполнительную систему JRE и прочее.

1.4 Spring Framework

Spring – это свободно распространяемый фреймворк, созданный Родом Джонсоном (Rod Johnson) и описанный в его книге «Expert One-on-One: J2EE Design and Development». Spring был создан с целью устранить сложности разработки корпоративных приложений и сделать возможным использование простых компонентов JavaBean для достижения всего того, что ранее было возможным только с использованием EJB (Enterprise Java Beans) [9].

Spring представляет собой контейнер внедрения зависимостей с несколькими слоями, что позволяет быстрее и удобнее создавать Java-приложения.

Фреймворк Spring может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании [10]. Эти фреймворки делятся на структурные элементы типовых комплексных приложений:

1. Inversion of Control-контейнер
2. Фреймворк аспектно-ориентированного программирования
3. Фреймворк доступа к данным
4. Фреймворк управления транзакциями
5. Фреймворк MVC
6. Фреймворк удалённого доступа
7. Фреймворк аутентификации и авторизации
8. Фреймворк удалённого управления
9. Фреймворк работы с сообщениями
10. Тестирование

1.5 Spring Boot

Из-за громоздкой конфигурации зависимостей настройка Spring для корпоративных приложений превратилась в весьма утомительное и подверженное ошибкам занятие. Особенно это относится к приложениям, которые используют также несколько сторонних библиотек. Проект Spring Boot предоставляет разработчикам утилиты, которые автоматизируют процедуру настройки и ускоряют процесс создания и развертывания Spring-приложений [11]. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода.

Возможности Spring Boot следующие:

- предоставляет гибкий способ настройки Java Beans, конфигураций XML и транзакций базы данных.
- обеспечивает мощную пакетную обработку и управляет конечными точками REST.
- в Spring Boot все настраивается автоматически; ручные настройки не требуются.
- облегчает управление зависимостями
- включает встроенный контейнер сервлетов

1.6 MVC

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо [12].

- Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
- Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Концепция MVC была описана Трюгве Реенскаугом в 1978 году [13]. Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи:

К одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели.

Не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных) — для этого достаточно использовать другой контроллер [12].

1.7 Apache Maven

Apache Maven — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (англ. Project Object Model), являющемся подмножеством XML. Проект Maven издаётся сообществом Apache Software Foundation [14].

Maven обеспечивает декларативную сборку проекта. В файлах описания проекта содержится его спецификация, а не отдельные команды выполнения. Он может использоваться для построения и управления проектами, написанными на Java, C#, Ruby, Scala, и других языках. В настоящее время актуальная ветка проекта – 3.x.

Минимальная конфигурация проекта включает в себя версию конфигурационного файла, имя проекта, его автора и версию. С помощью файла pom.xml конфигурируются зависимости от других проектов, индивидуальные фазы процесса построения проекта, список плагинов и так далее [15].

Глава 2. Разработка приложения

2.1 Исследование предметной области

Проведем анализ предметной области, выделим присутствующие сущности.

Предметная область в данной работе – сервис-агрегатор предложений о продаже автомобилей. Сервис предоставляет возможность просмотреть информацию о ценах автомобиля у различных дилеров. Автомобили можно выбирать исходя из марки, модели, комплектации, цвета, и т. д. Для помощи в выборе автомобиля имеется система отзывов.

Опишем сущности и их свойства:

1. Автомобиль

   Может иметь следующие свойства:

   - Модель
   - Цвет
   - Комплектация
   - Тип

   Каждый автомобиль имеет марку (т.е., производителя); к автомобилю прилагаются отзывы, оставленные пользователями, и автомобиль продается у дилера по определенной цене.
2. Марка (производитель). Выделим одно свойство: название.
3. Дилер (продавец автомобилей).

   Выделим следующие свойства:

   - Название
   - Адрес

   Дилер продает автомобили по определенной цене. Стоит заметить, что связь между автомобилями и дилерами имеет соотношение M..N, т.к. один и тот же автомобиль может быть в наличии у нескольких дилеров; а дилер может продавать несколько различных автомобилей.

4. Отзыв. Имеет следующие свойства:
   - Автор
   - Содержание

   Автомобиль может иметь несколько отзывов, но отзыв не может распространяться на несколько автомобилей.

С учетом сказанного, определим связи между сущностями. Удобно отобразить связи в виде ER-диаграммы «Сущность-связь».

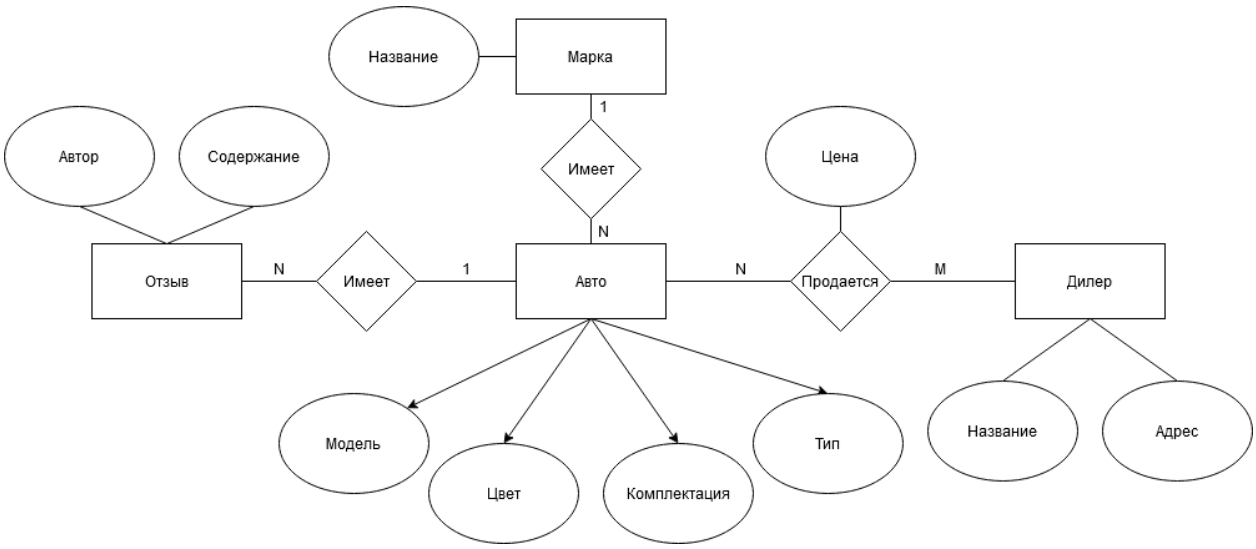

Рис. 2.1 ER-диаграмма

2.2 Реализация моделей

Описанные сущности в приложении Spring реализуются в виде классов внутри пакета Models. При создании класса используем аннотации, предоставляемые Spring. Поле id, использующееся как первичный ключ (PK, primary key), снабжено аннотациями @Id, @GeneratedValue, @Column, которые указывают на: роль поля как первичного ключа; добавление правила автоматической генерации значений id; необходимость создания столбца с таким именем.

Связи между моделями определяются с помощью аннотаций @ManyToOne, @ManyToMany и @OneToOne.

При связывании моделей с помощью @ManyToOne с помощью аннотации @JoinColumn указывается, как необходимо соединить таблицы для получения объекта element. Аргументы JoinColumn указывают на имя таблицы и имя столбца, по которому выполняется операция JOIN.

Связывание с помощью @ManyToMany предполагает создание дополнительной промежуточной таблицы, описывающейся с помощью аннотации @JoinTable. Параметры аннотации задают имя промежуточной таблицы и имена столбцов, по которым происходит склеивание.

Листинг с описаниями моделей можно найти в приложениях.

Приведем описание созданных моделей в виде UML-диаграммы:

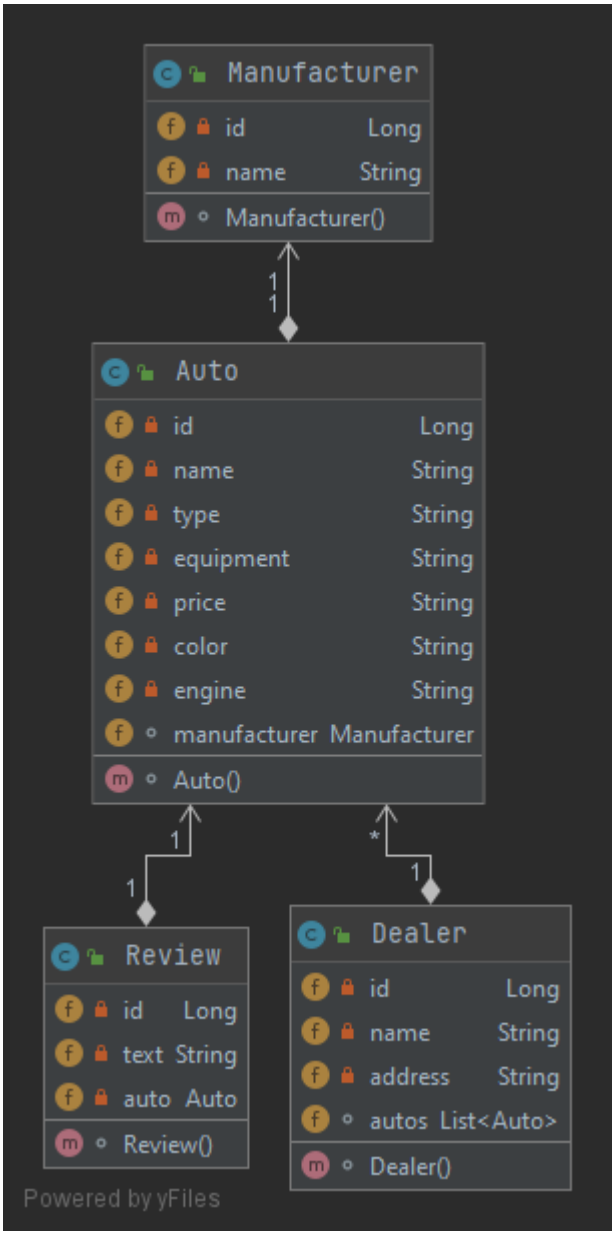

Рис. 2.2 UML-диаграмма моделей

2.3 Репозитории и контроллеры

Для удобного доступа к данным из БД, описанным в виде моделей, необходимо создать классы-репозитории. При создании репозитория будем учитывать потребности приложения в фильтрации объектов. В качестве примера приведем фрагмент кода одного из репозиториев:

```
@Repository
Public interface AutoRepository extends JpaRepository<Auto, Long>{
      List<Auto> findAutoByName(String name);
}
```

Листинг 2.1. Фрагмент кода репозитория AutoRepository

В репозитории описаны сигнатуры методов, предназначенных для поиска объектов согласно передаваемым параметрам. Реализация объявленного интерфейса генерируется автоматически, благодаря чему мы избавляемся от необходимости написания большого количества однотипного кода.

Опишем контроллеры. Контроллеры, в соответствии с принципом MVC, обрабатывают запрос пользователя. Для контроллера необходимо задать точки доступа к API, находящиеся в зоне ответственности его методов, и описать действия, которые необходимо предпринять для обработки запроса.

При описании контроллеров используются следующие аннотации: @CrossOrigin описывает, от каких адресов контроллер будет принимать запросы. В данном случае указали значение “*”, что позволит обрабатывать запросы, отправленные от любого хоста. В целях безопасности имеет смысл ограничить данное значение, добавив только доверенные хосты.

Аннотации @GetMapping и @PostMapping определяют HTTP-метод и URL-адрес, к которым будет привязан относящийся к ним метод. Аннотации работают, соответственно, для методов GET и POST.

Аннотация @RequestParams позволяет получить из GET запроса параметры вида “?name=audi”, называемые также Query parameters. В случае отсутствия данных параметров, значение переменной будет равно null. Это позволяет определить факт наличия или отсутствия параметра в запросе.

Аннотация @PathVariable получает параметр, находящийся в составе URL адреса, такой, как “/reviews/1/”, где “1” – это id элемента.

Приведем фрагмент исходного кода, описывающий один из контроллеров:

```
@RestController
public class ReviewController {
    private ReviewRepository repo;

    ReviewController(ReviewRepository repo) {
        this.repo = repo;
    }
    @GetMapping("/reviews")
    List<Review> getAll() {
        return repo.findAll();
    }
    @GetMapping("/reviews/{id}")
    Optional<Review> getOne(@PathVariable Long id) {
        return repo.findById(id);
    }
    @PostMapping("/reviews")
    Review newReview(@RequestBody Review review) {
        return repo.save(review);
    }
}
```

Листинг 2.4. Фрагмент кода контроллера ReviewController

Приведем UML-диаграмму, описывающую отношения между моделями, репозиториями и контроллерами:

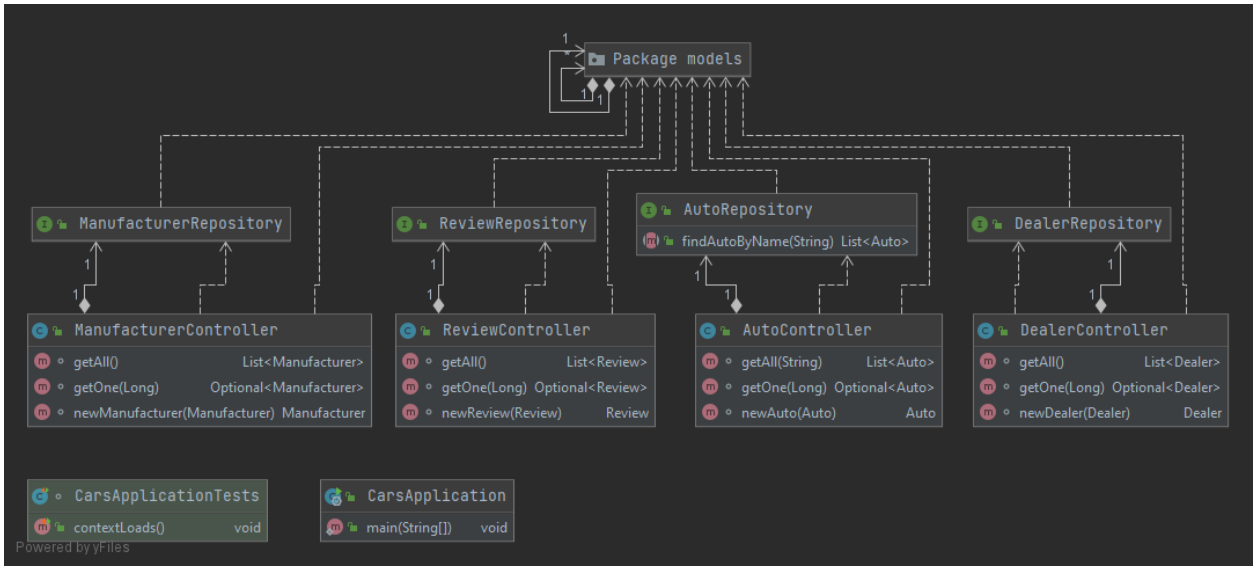

Рис. 2.3 UML-диаграмма приложения

Листинги кода контроллеров можно найти в приложениях Д-Е.

Глава 3. Тестирование приложения

3.1 Тестирование REST API

Протестируем приложение с помощью ПО для отправки запросов к API. В данном случае таким ПО выступит приложение Postman [15]. Протестируем отправку запросов с методами POST и GET. POST-запросы позволят добавить в базу данных новые объекты, а GET – проконтролировать их корректное получение. На момент начала тестирования база данных не содержит записей, относящихся к тестируемому приложению.

Используемый набор тестовых данных:

Тестируемый URL	Содержимое json-файла

localhost:8080/dealers	{
	"name": "Дилер №1",
	"address": "Белгород, ул. Пушкина, д. Колотушкина"
	}

localhost:8080/manufacturers	{
	"name": "Renault"
	}

localhost:8080/autos	{
	"name": "Logan",
	"type": "Седан",
	"equipment": "Максимальная комплектация",
	"price": "600 000 рублей",
	"color": "Красный",
	"engine": "1.6",
	"manufacturer": {
	"id": 2
	}
	}

localhost:8080/reviews	{
	"text": "Очень хорошая машина! Есть колеса,
	на ней можно кататься. Иногда можно не кататься. 9 из 10",
	"auto": {
	"id": 3
	}
	}

Таблица 3.1. Тестовые данные

Выполним запросы с помощью приложения Postman и проанализируем результат их выполнения.

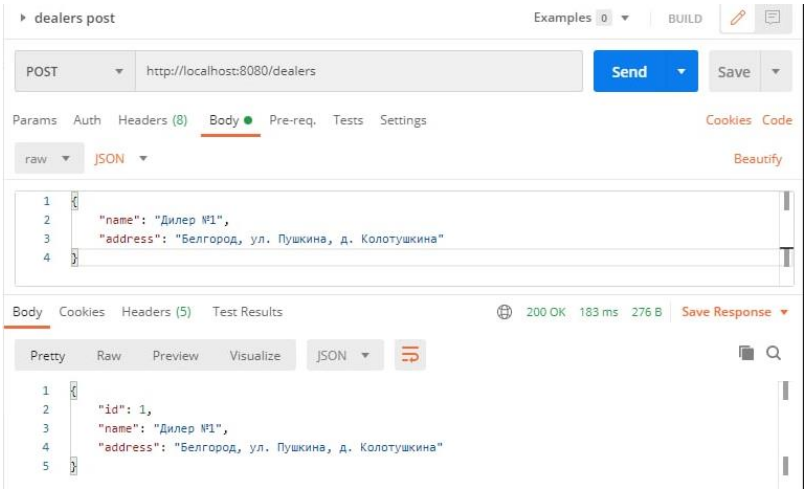

Рис. 3.1. Результат POST-запроса №1

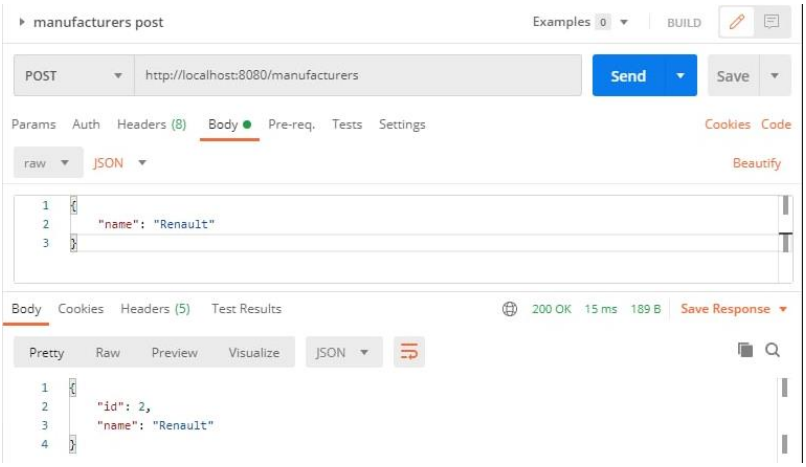

Рис. 3.2. Результат POST-запроса №2

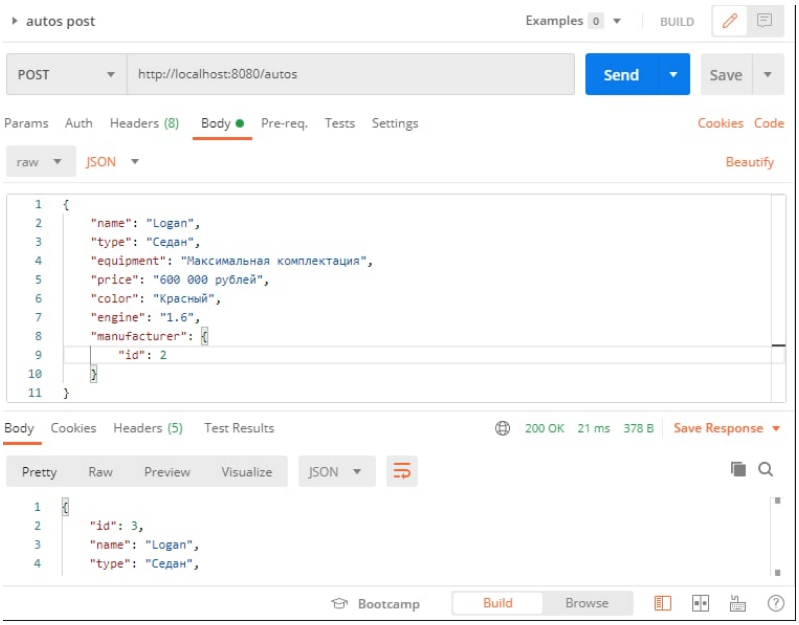

Рис. 3.3. Результат POST-запроса №3

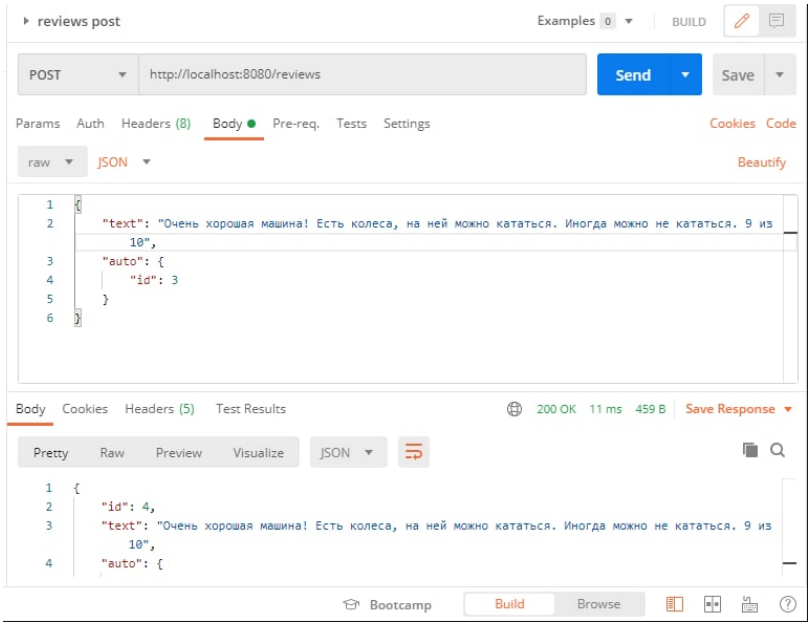

Рис. 3.4. Результат POST-запроса №4

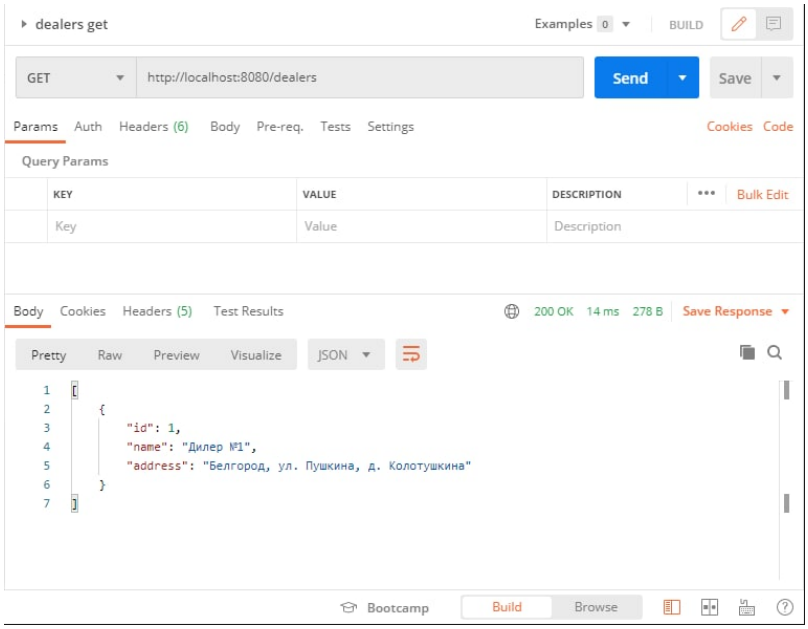

Рис. 3.5. Результат GET-запроса №1

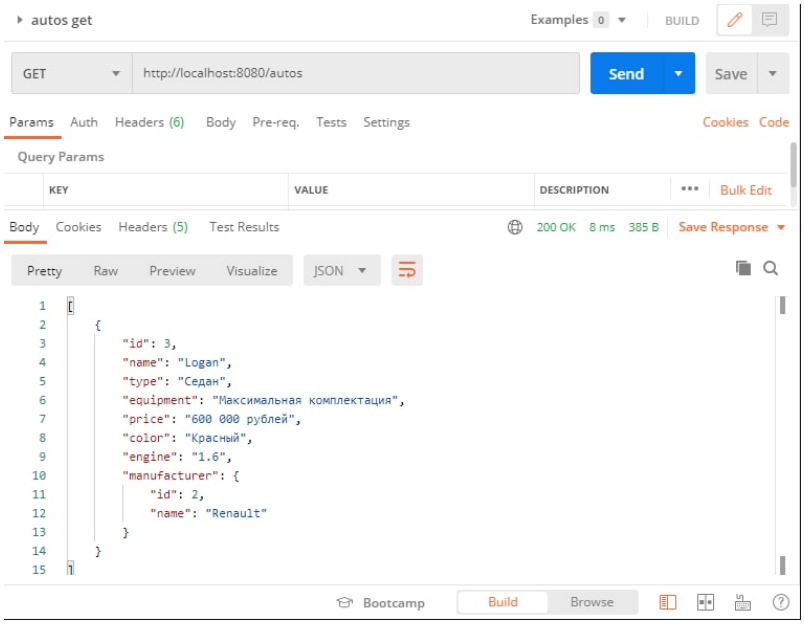

Рис. 3.6. Результат GET-запроса №2

Полученные результаты говорят о том, что приложение работает корректно: в результате запросов с методом POST в базу данных успешно добавлены новые объекты; а запросы с методом GET успешно возвращают объекты, находящиеся в базе данных, в соответствии с ожидаемыми ответами. Вывод: разработанное API функционирует в соответствии с предъявленными требованиями.

Заключение

В ходе выполнения курсового проекта были изучены возможности языка Java и фреймворка Spring (в комплекте с Spring Boot) для разработки веб-приложений. Изучены современные подходы к построению одностраничных веб-приложений: концепция AJAX, архитектуры REST API и MVC.

На примере создания одностраничного приложения – сервиса-агрегатора предложений о продаже автомобилей – рассмотрены основные этапы разработки веб-приложений с использованием REST API.

Рассмотрена предметная область, выделены сущности, составлена ER-диаграмма их отношений. На основе полученных данных реализованы модели ORM, преобразующихся в структуру в базе данных. В серверном приложении спроектированы и реализованы такие компоненты, как репозитории и контроллеры.

Полученное приложение протестировано с помощью инструмента выполнения HTTP запросов Postman. В результате проведения тестирования сделали вывод о правильном функционировании REST API, реализуемого приложением.

Список литературы

1. Одностраничное приложение – Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Одностраничное_приложение
2. AJAX – Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/AJAX
3. API – Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/API
4. Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST) – [Электронный ресурс]. – URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
5. Erik Wilde, Cesare Pautasso. REST: From Research to Practice. — Springer Science & Business Media, 2011. — 528 p.
6. Java – Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Java
7. Java – официальный сайт. – [Электронный ресурс]. – URL: https://www.java.com/
8. Уоллс К. Spring в действии. – М.: ДМК Пресс, 2013. – 752 с.: ил.
9. Spring Framework - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Spring_Framework
10. Лонг Джош, Бастани Кеннет – Java в облаке. Spring Boot, Spring Cloud, Cloud Foundry. – СПб.: Питер, 2019 – 524 с.: ил.
11. Model-View-Controller – Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Model-View-Controller
12. Обобщенный Model-View-Controller – RSDN.org– [Электронный ресурс]. – URL: http://rsdn.org/article/patterns/generic-mvc.xml
13. Apache Maven - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Apache_Maven
14. Maven – Welcome to Apache Maven. – [Электронный ресурс]. – URL: https://maven.apache.org/
15. Postman – [Электронный ресурс]. – URL: https://www.postman.com/

Приложения

Приложение А. Содержимое файла Auto.java:

```
package ru.bstu.iitus.povtas.vt41.cars.models;

import lombok.Data;

import javax.persistence.*;

@Data
@Entity
public class Auto {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String type;
    private String equipment;
    private String price;
    private String color;
    private String engine;

    @OneToOne
    Manufacturer manufacturer;

    Auto () {}

}
```

Приложение Б. Содержимое файла Review.java:

```
package ru.bstu.iitus.povtas.vt41.cars.models;

import lombok.Data;

import javax.persistence.*;

@Data
@Entity
public class Review {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String text;

    @OneToOne
    private Auto auto;

    Review () {}
}
```

Приложение В. Содержимое файла AutoRepository.java:

```
package ru.bstu.iitus.povtas.vt41.cars.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.bstu.iitus.povtas.vt41.cars.models.Auto;

import java.util.List;

@Repository
public interface AutoRepository extends JpaRepository<Auto, Long> {
    List<Auto> findAutoByName(String name);
}
```

Приложение Г. Содержимое файла ReviewRepository.java:

```
package ru.bstu.iitus.povtas.vt41.cars.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.bstu.iitus.povtas.vt41.cars.models.Review;

@Repository
public interface ReviewRepository extends JpaRepository<Review, Long>
{

}
```

Приложение Д. Содержимое файла AutoController.java:

```
package ru.bstu.iitus.povtas.vt41.cars.controllers;

import org.springframework.web.bind.annotation.*;
import ru.bstu.iitus.povtas.vt41.cars.models.Auto;
import ru.bstu.iitus.povtas.vt41.cars.repositories.AutoRepository;

import java.util.List;
import java.util.Optional;

@RestController
public class AutoController {
    private final AutoRepository repo;

    public AutoController(AutoRepository repo) {
        this.repo = repo;
    }
    @GetMapping("/autos")
    List<Auto> getAll(@RequestParam(required = false) String name) {
        if (name == null){
            return repo.findAll();
        }
        return repo.findAutoByName(name);
    }

    @GetMapping("/autos/{id}")
    Optional<Auto> getOne(@PathVariable Long id) {
        return repo.findById(id);
    }
    @PostMapping("/autos")
    Auto newAuto(@RequestBody Auto auto) {
        return repo.save(auto);
    }
}
```

Приложение Е. Содержимое файла ReviewController.java:

```
package ru.bstu.iitus.povtas.vt41.cars.controllers;

import org.springframework.web.bind.annotation.*;
import ru.bstu.iitus.povtas.vt41.cars.models.Manufacturer;
import ru.bstu.iitus.povtas.vt41.cars.models.Review;
import ru.bstu.iitus.povtas.vt41.cars.repositories.ReviewRepository;

import java.util.List;
import java.util.Optional;

@RestController
public class ReviewController {
    private ReviewRepository repo;

    ReviewController(ReviewRepository repo) {
        this.repo = repo;
    }
    @GetMapping("/reviews")
    List<Review> getAll() {
        return repo.findAll();
    }
    @GetMapping("/reviews/{id}")
    Optional<Review> getOne(@PathVariable Long id) {
        return repo.findById(id);
    }
    @PostMapping("/reviews")
    Review newReview(@RequestBody Review review) {
        return repo.save(review);
    }
}
```