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»

Выполнил: студент группы ВТ-41 Ковалёв И. Д.

Проверил: Торопчин Д. А.

г. Белгород

2020 г.

**Оглавление**

[Введение 3](#_Toc60194625)

[Глава 1. Теоретические сведения 4](#_Toc60194626)

[1.1 Java 4](#_Toc60194627)

[1.2 Maven 5](#_Toc60194628)

[1.3 Spring Framework 6](#_Toc60194629)

[1.4 Hibernate 8](#_Toc60194630)

[Глава 2. Разработка приложения 9](#_Toc60194631)

[Глава 3. Тестирование ПО 14](#_Toc60194632)

[Заключение 19](#_Toc60194633)

[Список литературы 20](#_Toc60194634)

[Приложения 21](#_Toc60194635)

[Приложение А. Содержимое файла Album.java 21](#_Toc60194636)

[Приложение Б. Содержимое файла Band.java 22](#_Toc60194637)

[Приложение В. Содержимое файла Genre.java 23](#_Toc60194638)

[Приложение Г. Содержимое файла Track.java 24](#_Toc60194639)

[Приложение Д. Содержимое файла AlbumController.java 25](#_Toc60194640)

[Приложение Е. Содержимое файла BandController.java 26](#_Toc60194641)

[Приложение Ж. Содержимое файла AlbumRepository.java 27](#_Toc60194642)

[Приложение Ж. Содержимое файла BandRepository.java 28](#_Toc60194643)

# **Введение**

Целью курсового проекта является реализация клиент-серверного приложения, 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 .

В качестве предметной области для курсового проекта был выбран сервис для поиска музыки. Приложение должно предоставлять возможность просмотра и поиска музыки по исполнителям или альбомам.

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

## **1.1 Java**

Java — строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process; язык и основные реализующие его технологии распространяются по лицензии GPL [1].

Программы, реализованные на языке программирования Java транслируются в байт-код, который выполняется виртуальной машиной. Главным преимуществом данного подхода является полная независимость кода от операционной системы и оборудования, на котором будет запущена программа.

В Java присутствуют такие компоненты:

1. JVM (Java Virtual Machine) – виртуальная машина Java, которая исполняет байт-код, созданный из исходного кода программы компилятором javac.
2. JRE (Java Runtime Environment) – минимальная реализация виртуальной машины, необходимая для исполнения Java-приложений. В ее состав не входит компилятор и другие средства разработки.
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.

Следует отметить, что недостатком применения виртуальной машины является снижение производительности приложений.

## **1.2 Maven**

Maven – фреймворк, применяемый для сборки проектов на основе описания их структуры в файлах на языке РОМ (Project Object Model). Проект Maven издается сообществом Apache Software Foundation [2].

Maven обеспечивает декларативную сборку проекта. В файлах описания проекта содержится его спецификация, а не отдельные команды выполнения. Все задачи по обработке файлов, описанные в спецификации, выполняются посредством из обработки последовательностью встроенных и внешних плагинов.

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.

Жизненный цикл maven-проекта представляет собой список поименованных фаз, определяющий порядок действия при его построении. Имеет три независимых порядка выполнения:

1. Clean – жизненный цикл для очистки проекта;
2. Default – Основной жизненный цикл, включающий в себя такие этапы, как validate, compile, test, package, install, deploy и прочие.
3. Site – жизненный цикл генерации проектной документации.

## **Spring Framework**

Spring Framework — универсальный фреймворк с открытым исходным кодом для Java-платформы. Фреймворк был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года. Первая стабильная версия 1.0 была выпущена в марте 2004. Spring 2.0 был выпущен в октябре 2006, Spring 2.5 — в ноябре 2007, Spring 3.0 в декабре 2009, и Spring 3.1 в декабре 2011. Текущая версия — 5.2.1 [3].

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

Данный фреймворк можно рассмотреть как коллекцию меньших фреймворков, большая часть которых может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании. Вышеуказанные 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

Рассмотрим подробнее фреймворк MVC. Spring MVC является веб-средой Spring. Это позволяет создавать все, что связанно с сетью, от небольших веб-сайтов до сложных веб-сервисов.

MVC состоит из таких частей:

1. Модель – содержит данные, которые необходимо отобразить, представляют собой Java-объекты.
2. Представление – отвечает за вывод данных пользователю.
3. Контроллер – отвечает за обработку запросов пользователей и передачу данных модулю представления для обработки.

В основе Spring MVC лежит DispatchServlet, задача которого заключается в обработке всех HTTP запросов и ответов.

Spring Boot представляет собой проект, целью которого является упрощения создания приложений на основе Spring. Он позволяет наиболее простым способом создать веб-приложение, требуя от разработчиков минимум усилий по настройке и написанию кода.

Среди особенностей Spring Boot можно отметить такие, как простота управления зависимостями, автоматическая конфигурация проекта и встроенная поддержка сервера приложений.

## **1.4 Hibernate**

Hibernate — библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения, самая популярная реализация спецификации JPA. Распространяется свободно на условиях GNU Lesser General Public License [4].

Главным достоинством данной библиотеки является возможность сократить объемы низкоуровневого программирования при работе с реляционными базами данных.

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая виртуальную объектную базу данных.

Данная библиотека позволяет решить задачу связи классов Java и таблиц данных в реляционной базе данных, а также типов данных Java и базы данных.

Помимо этого, предоставляются возможности по автоматической генерации и обновлению набора таблиц, построения запросов к базе данных, обработки полученных результатов, что приводит к значительному уменьшению времени разработки.

Отображение 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.

# **Глава 2. Разработка приложения**

Процесс разработки приложения следует разделить по этапам, первым из которых будет анализ предметной области, выделение сущностей и связей между ними, и составление диаграммы классов.

В качестве предметной области был выбран сервис для поиска музыки.

К сущностям данной предметной области можно отнести:

1. Исполнитель, информация о нем должна содержать:
   * + Название
     + Описание
     + Дата создания/рождения
     + Рейтинг
     + Логотип
2. Альбом
   * + Название
     + Дата выхода
     + Обложка
     + Рейтинг
3. Песня
   * + Название
     + Длительность
     + Рейтинг
4. Жанр
   * + Название

При этом связи между этими сущностями можно определить таким образом:

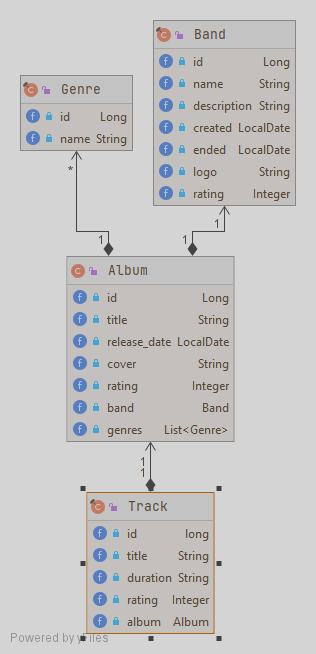
У одного исполнителя может быть несколько альбомов, в свою очередь альбом может содержать несколько песен. Сам альбом может быть отнесен к нескольким жанрам.

Составим схему «Сущность-связь»:

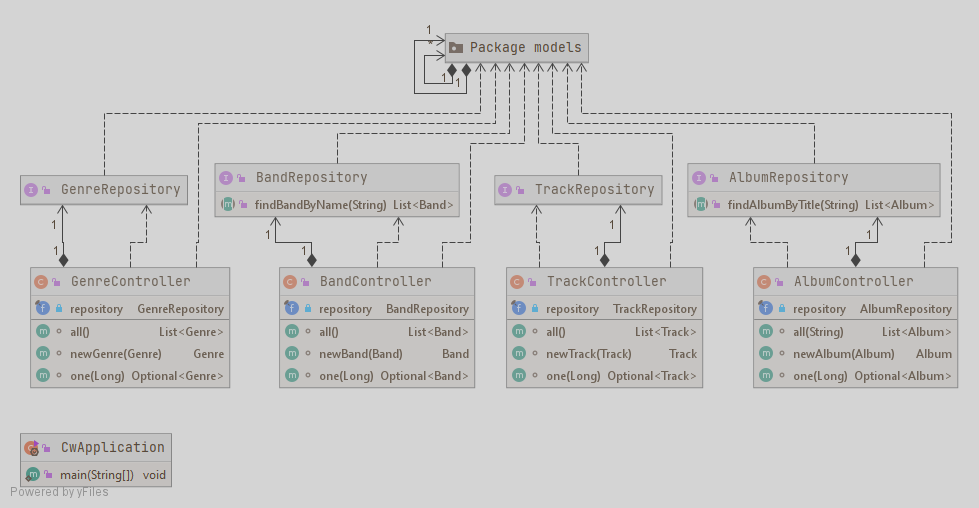


*Рис. 2.1 ER-диаграмма*

Построим UML-диаграммы классов разрабатываемого приложения.



*Рис. 2.2 Диаграмма классов пакета Models*



*Рис. 2.3 Диаграмма классов приложения, включая пакеты Controllers и Repositories*

Реализуем данные схемы. Создадим модели Band, Track, Genre и Album.

Исходный код модели Album:

|  |
| --- |
| @Data public class Album {  @Id  @GeneratedValue(strategy = GenerationType.AUTO)  @Column(name = "album\_id")  private Long id;  private String title;  private LocalDate release\_date;  private String cover;  private Integer rating;  @ManyToOne  @JoinColumn(name = "band", referencedColumnName = "id")  private Band band;   @ManyToMany  @JoinTable(  name = "album\_genres",  joinColumns = @JoinColumn(name="album\_id"),  inverseJoinColumns = @JoinColumn(name="genre\_id")  )  private List<Genre> genres;  } |

Рассмотрим данную модель. В качестве primary key используется поле Long id с аннотациями @Id, @GeneratedValue, @Column. В частности, применение второй аннотации позволяет совершать POST-запросы к серверу, не вкладывая в JSON поле id, ведь оно будет сгенерировано автоматически.

Связи один ко многим и многие ко многим реализуются с помощью аннотаций @OneToMany и @ManyToMany соответственно. В данном случае объекта класса Band является контейнером, который указывает на исполнителя, которому принадлежит альбом. С помощью аргументов name и referencedColumnName мы указываем, какое поле будет использоваться в качетсве foreign key. Со связью многие ко многим ситуация немного сложнее: здесь необходимо условно описать промежуточную таблицу для хранения отношений между двумя объектами. В данном случае описание содержит название таблицы, столбцы, содержащий внешние ключи альбомов и жанров.

Исходный код остальных моделей приведен в приложениях Б – Г соответстенно.

Были созданы репозитории для каждого из классов. Ниже приведен исходный код для репозитория альбомов:

|  |
| --- |
| @Repository public interface AlbumRepository extends JpaRepository<Album, Long> {  List<Album> findAlbumByTitle(String name);  } |

Здесь была описана сигнатура метода findAlbumByTitle, которая возвращает объект(ы) класса Album с названием name. Реализация данного метода создается автоматически, что заметно экономит время, особенно в случаях, когда таких методов надо сделать несколько и для нескольких классов.

Далее опишем контроллеры. В качестве примера возьмем BandController:

|  |
| --- |
| @RestController  @CrossOrigin(origins = "\*") public class BandController {  private final BandRepository repository;  BandController(BandRepository repository) {  this.repository = repository;  }  @GetMapping("/bands")  List<Band> all() {  return (List<Band>) repository.findAll();  }  @PostMapping("/bands")  Band newBand(@RequestBody Band newBand) {  return repository.save(newBand);  }  @GetMapping("/bands/{id}")  Optional<Band> one(@PathVariable Long id){  return repository.findById(id);  } } |

По результатам, для реализуемого API мы получаем URL hostname:8080/bands.

Были описаны методы GET и POST, при этом благодаря методу newBand клиент может получать JSON, содержащий данные об одном экземпляре класса Band, primary key которого был указан в URL. Остальные контроллеры принципиально ничем не отличаются от приведенного выше, кроме контроллера AlbumController. Здесь был реализован поиск:

|  |
| --- |
| @CrossOrigin(origins = "\*") @GetMapping("/albums") List<Album> all(@RequestParam(required = false) String title) {  if (title == null) {  return repository.findAll();  }  return repository.findAlbumByTitle(title); } |

Здесь и пригодился метод findAlbumByTitle. В качестве параметров запроса был описан параметр title, который является необязательным. Теперь, чтобы совершить поиск по названию, достаточно воспользоваться URL hostname:8080/bands/?title= «Некоторое название».

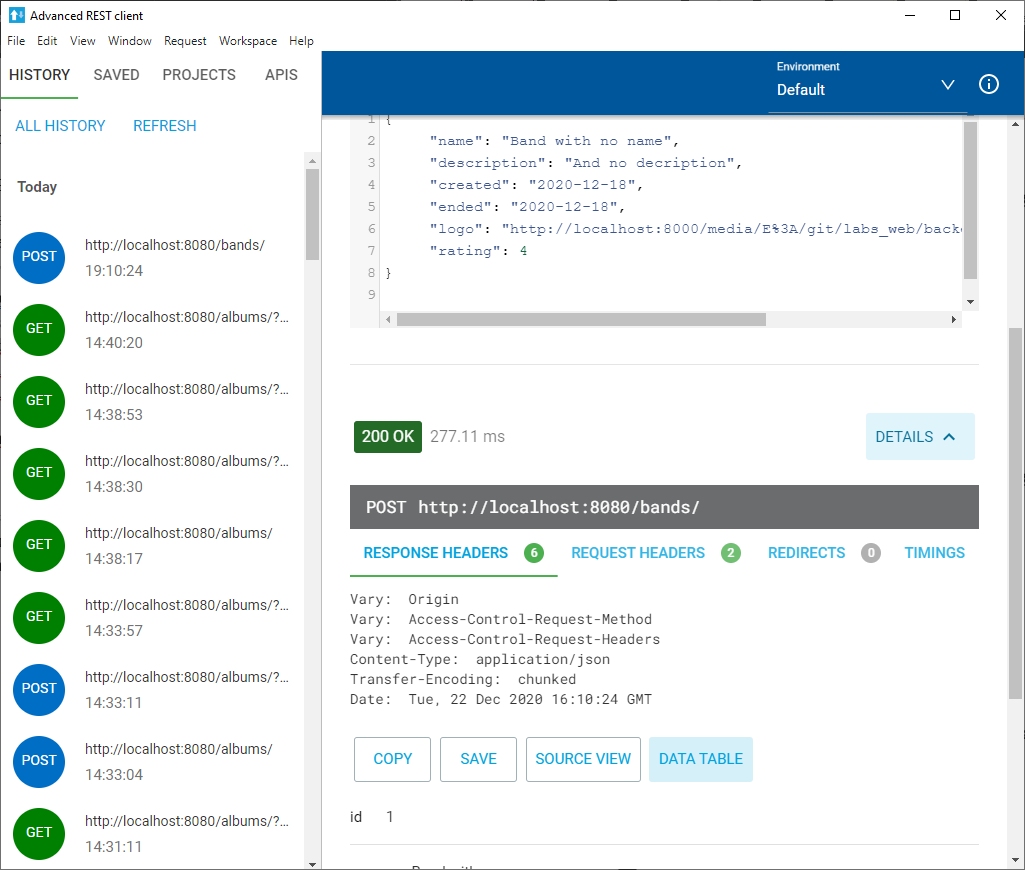
# **Глава 3. Тестирование ПО**

Проведем тестирование реализованного API с помощью приложения Advanced REST Client[5]. Попробуем добавить некоторые тестовые данные, чтобы выявить неполадки, если такие присутствуют.

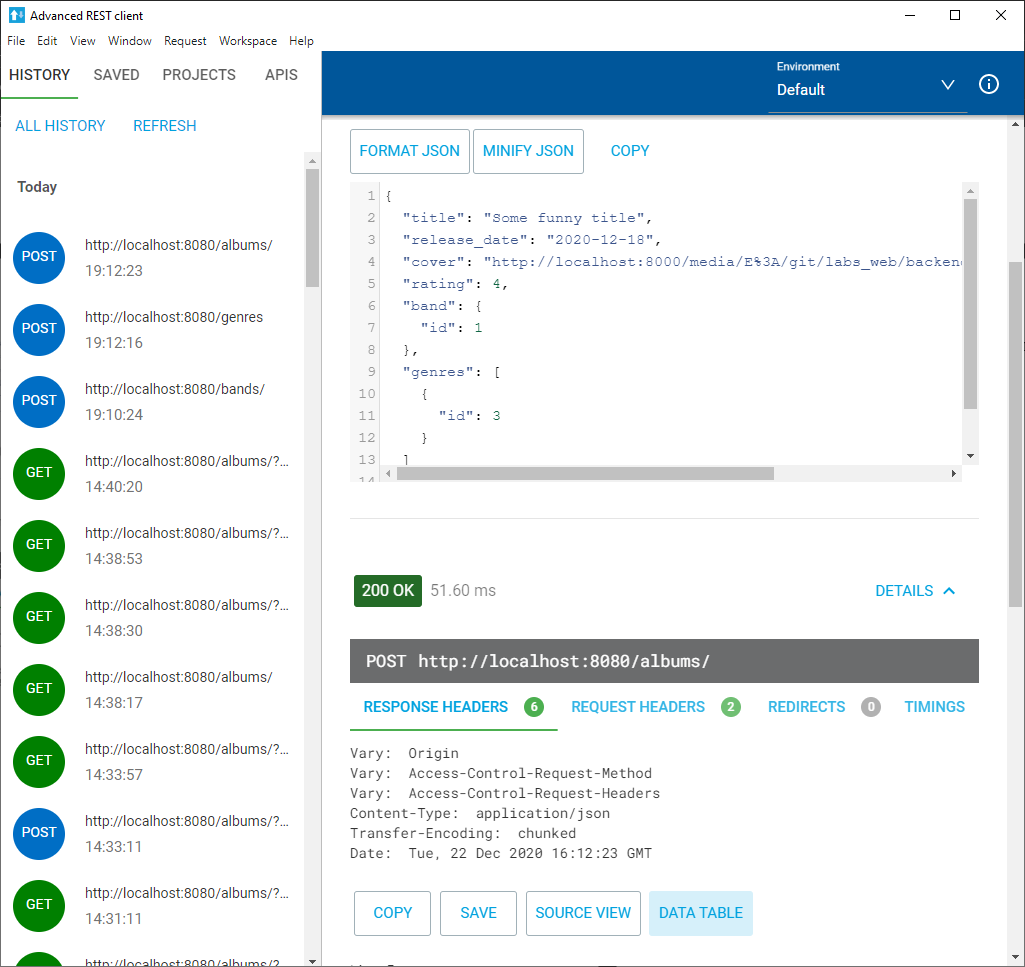
Был выбран такой набор тестовых данных:

|  |  |
| --- | --- |
| Тестируемый URL | Содержимое POST-запроса |
| [hostname/bands/](http://localhost:8080/bands/) | {  "name": "Band with no name",  "description": "And no decription",  "created": "2020-12-18",  "ended": "2020-12-18",  "logo": "http://hostname/media/ /Front\_eZs9usl.jpg",  "rating": 4  } |
| [hostname/albums/](http://hostname:8080/albums/) | {  "title": "Some funny title",  "release\_date": "2020-12-18",  "cover": "http://hostname/ /media/Front\_lNLPXFi.jpg",  "rating": 4,  "band": {  "id": 1  },  "genres": [  {  "id": 2  }  ]  } |
| [hostname/genres](http://localhost:8080/genres)/ | {  "name": "Жанр 1"  } |

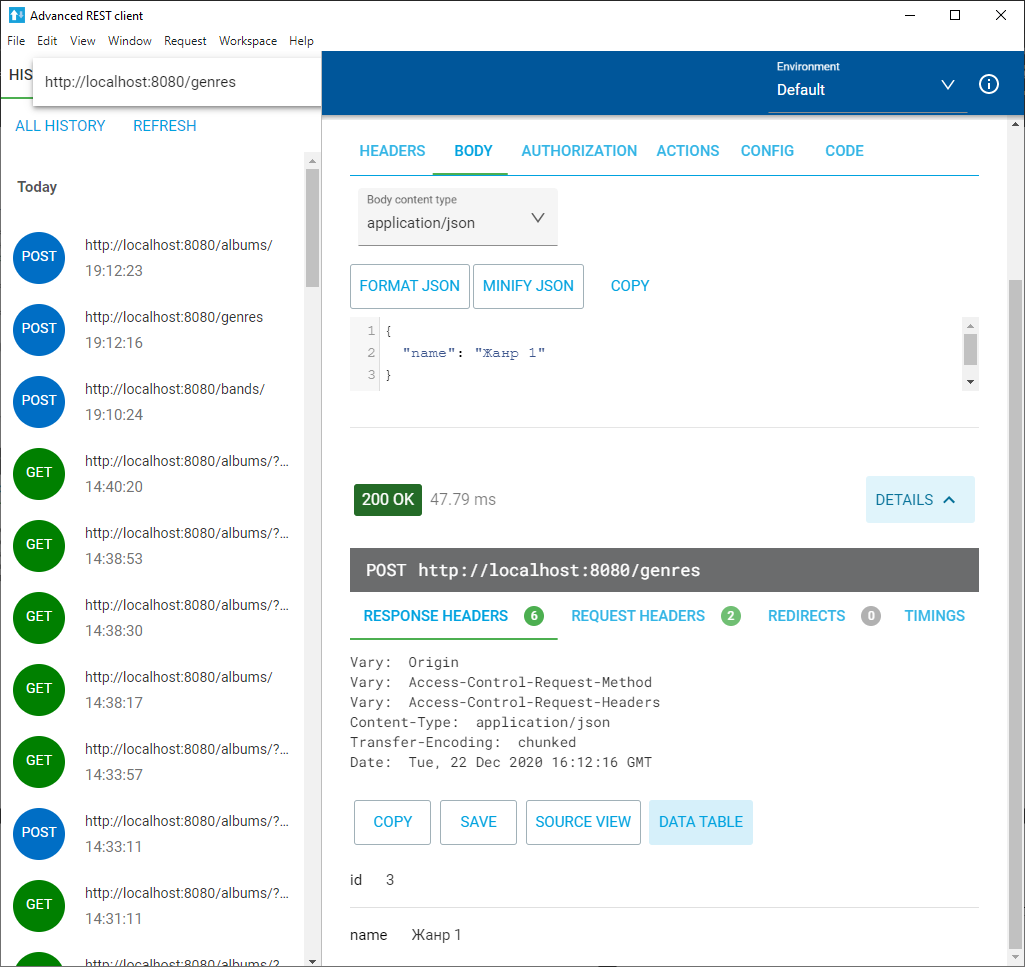
Выполним POST-запросы и проанализируем полученный результат.



*Рис. 4.1. Результат POST-запроса №1*



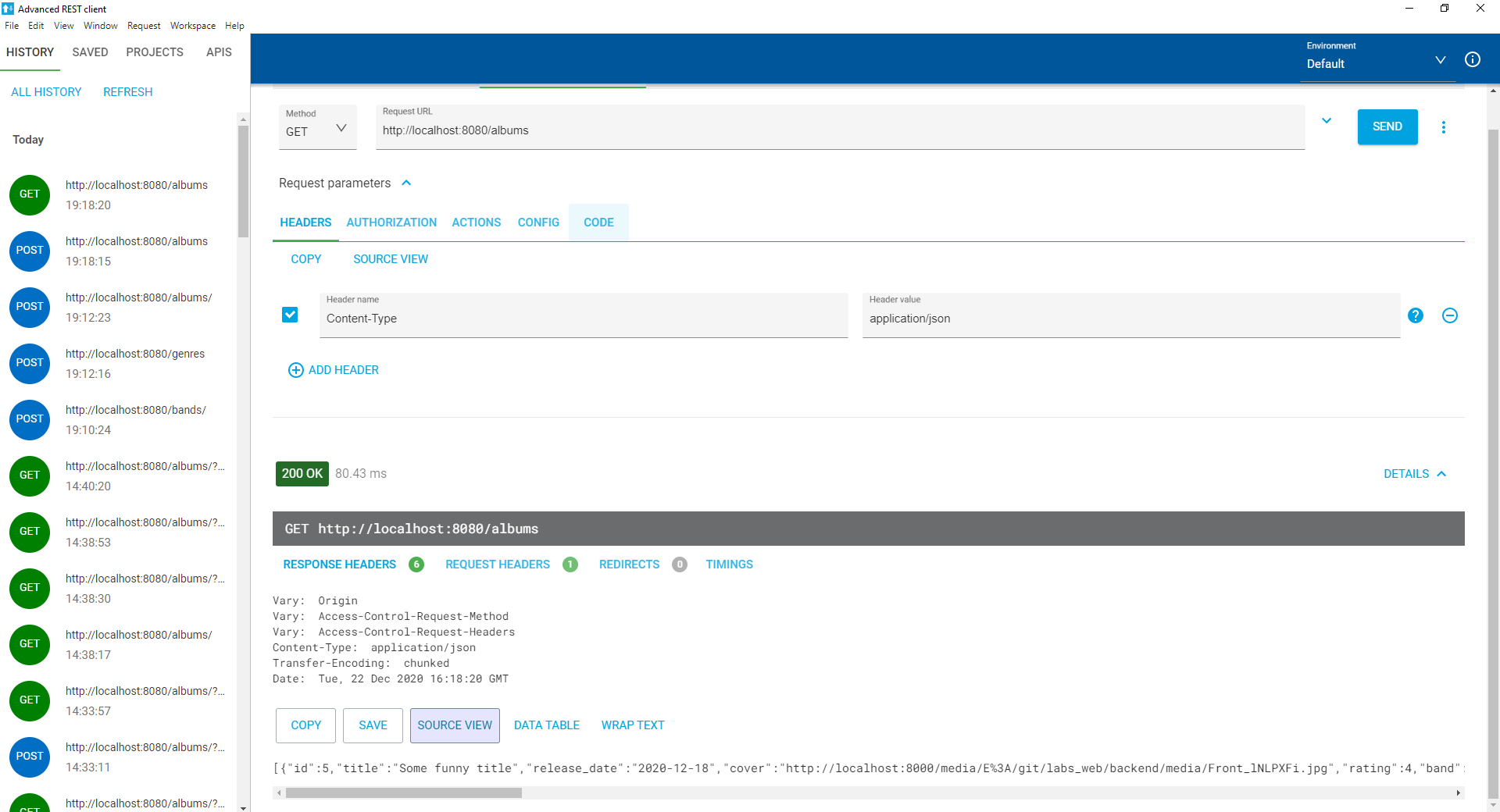
*Рис. 4.2. Результат POST-запроса №2*



*Рис. 4.3. Результат POST-запроса №3*

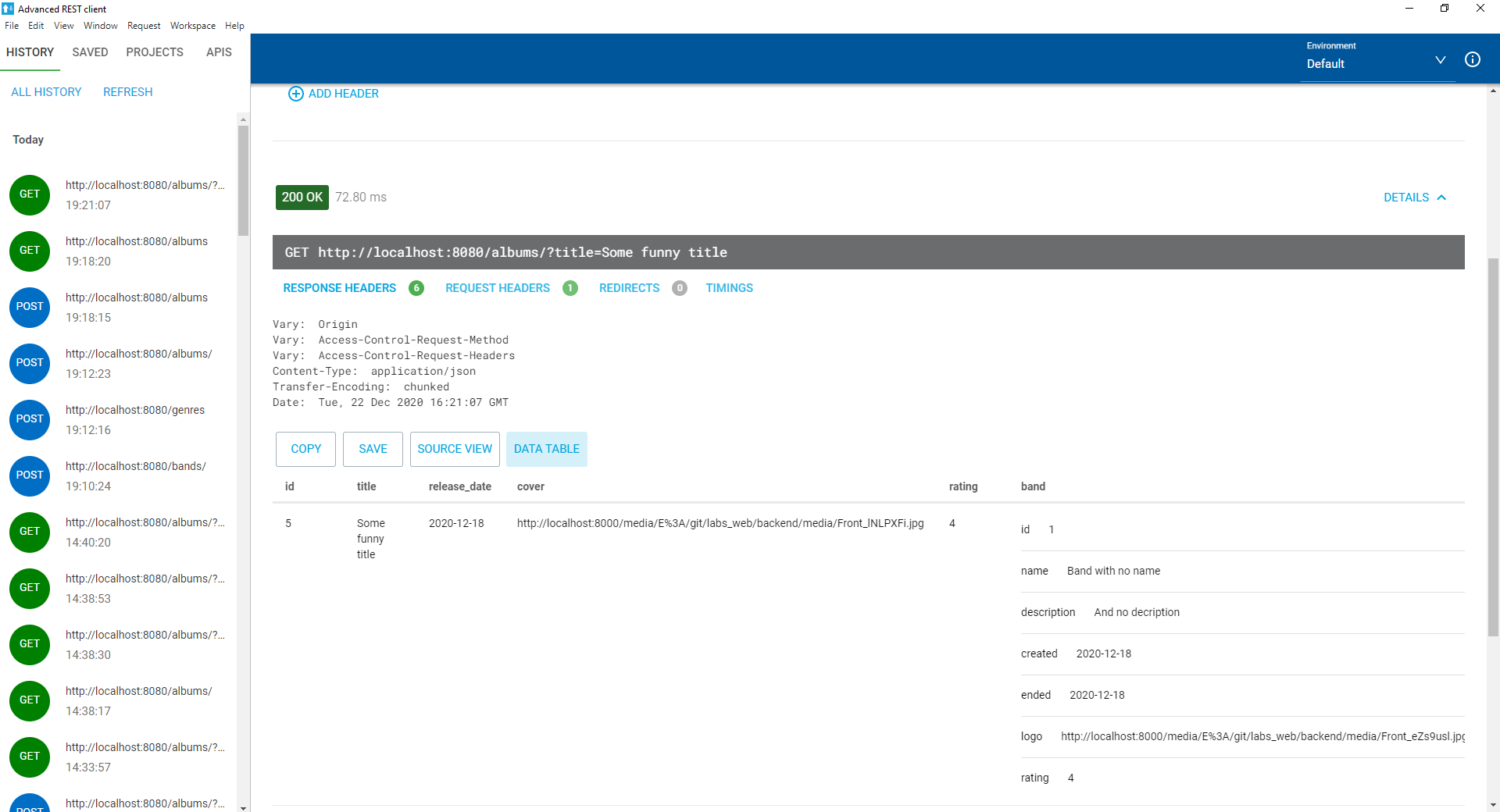
На каждый запрос от сервера был получен ответ 200 OK, что свидетельствует о том, что все запросы были выполнены успешно.

Поскольку теперь в базе данных уже содержатся некоторые данные, выполним GET-запрос по <http://localhost:8080/albums/>



*Рис. 4.4 Результат GET-запроса*

Был получен ответ 200 ОК, API вернула JSON с данными об альбомах. Добавим еще несколько альбомов и протестируем поиск по альбомам

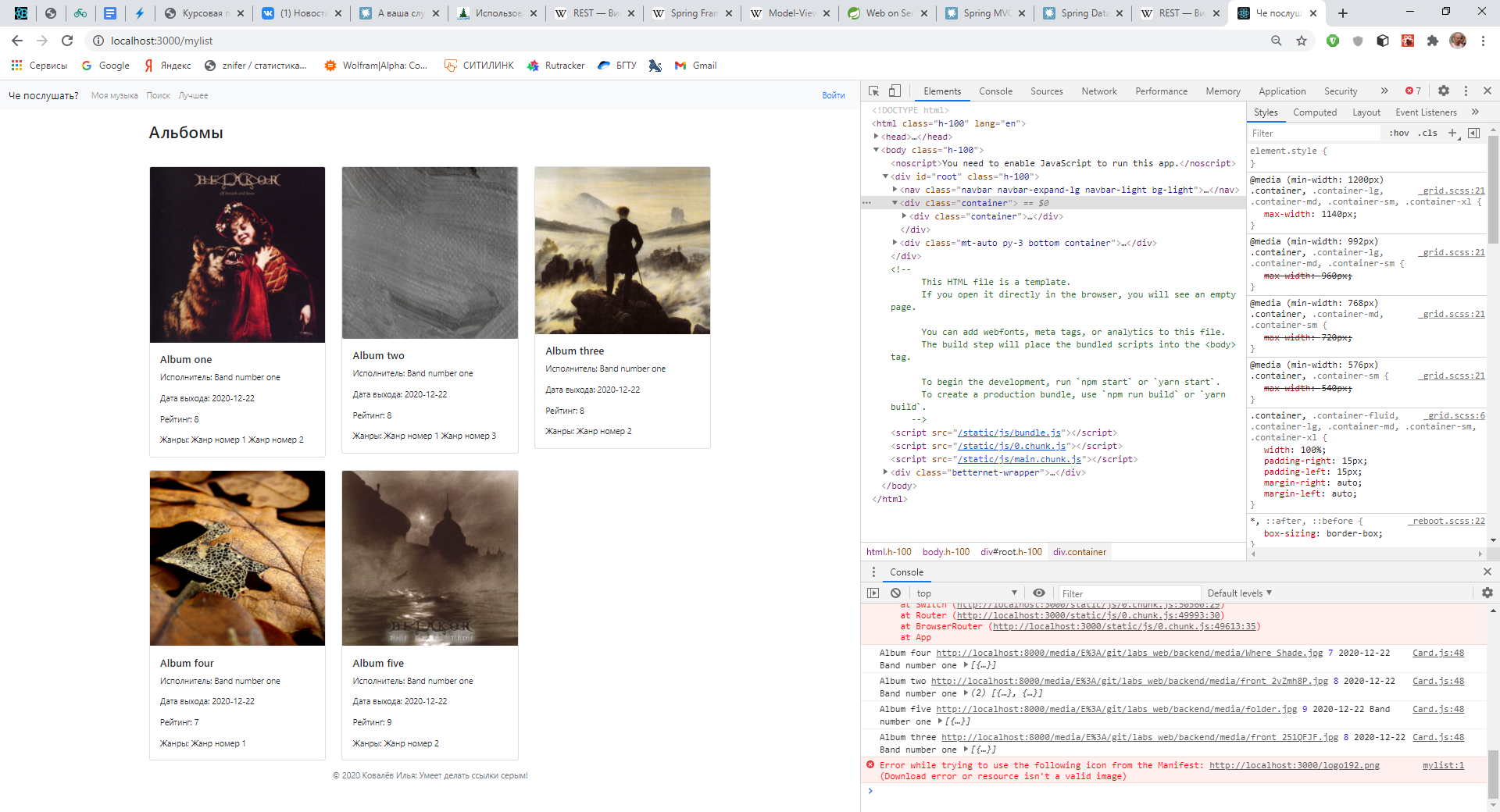


*Рис. 4.5 Результат GET-запроса с параметром title*

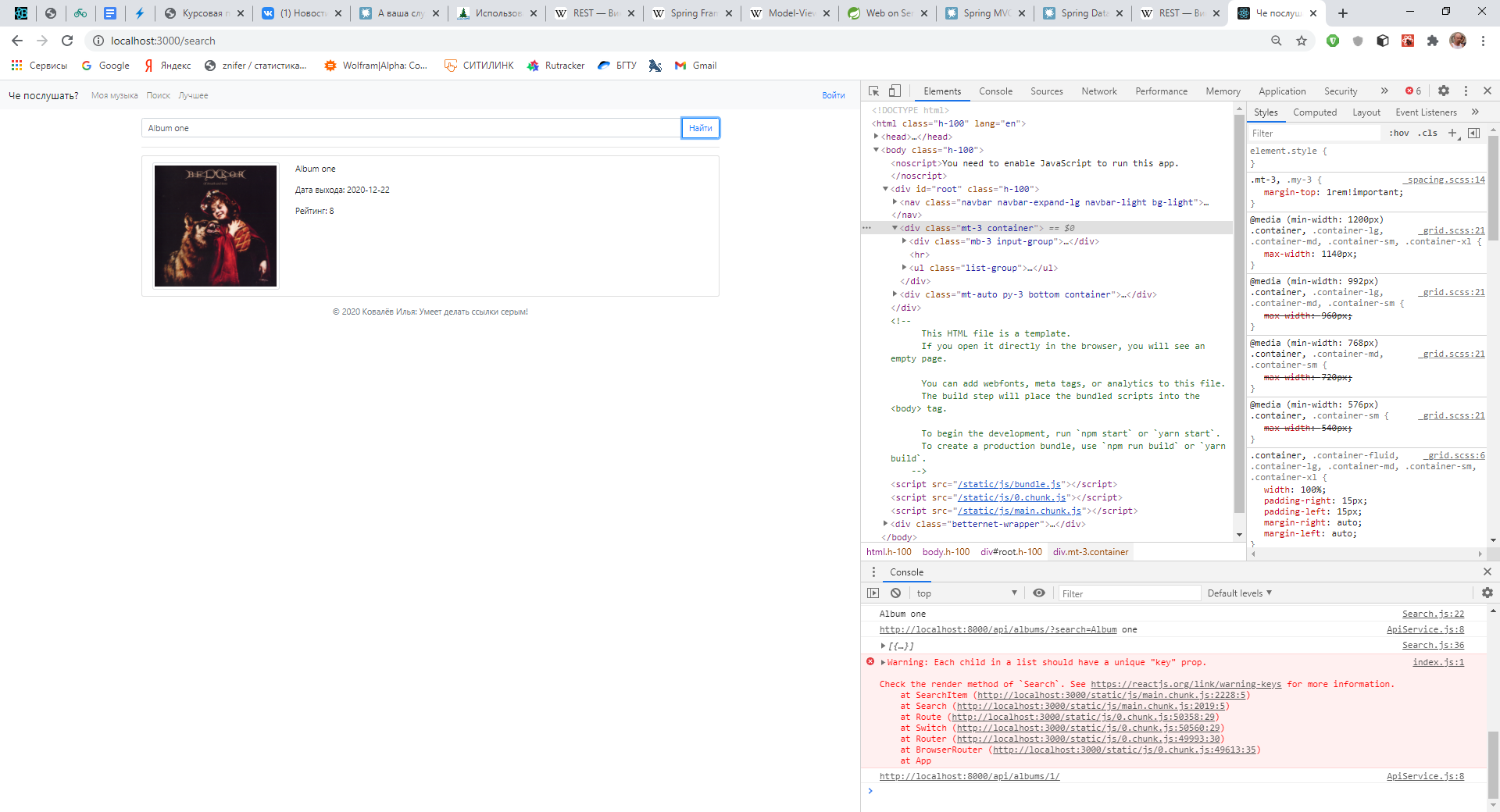
Как и до этого, были получены корректные данные, из чего следует сделать вывод, что API функционирует верно.

В дальнейшем разработанное API будет использоваться frontend-приложением, реализованном на языке JavaScript с использованием библиотеки ReactJs.

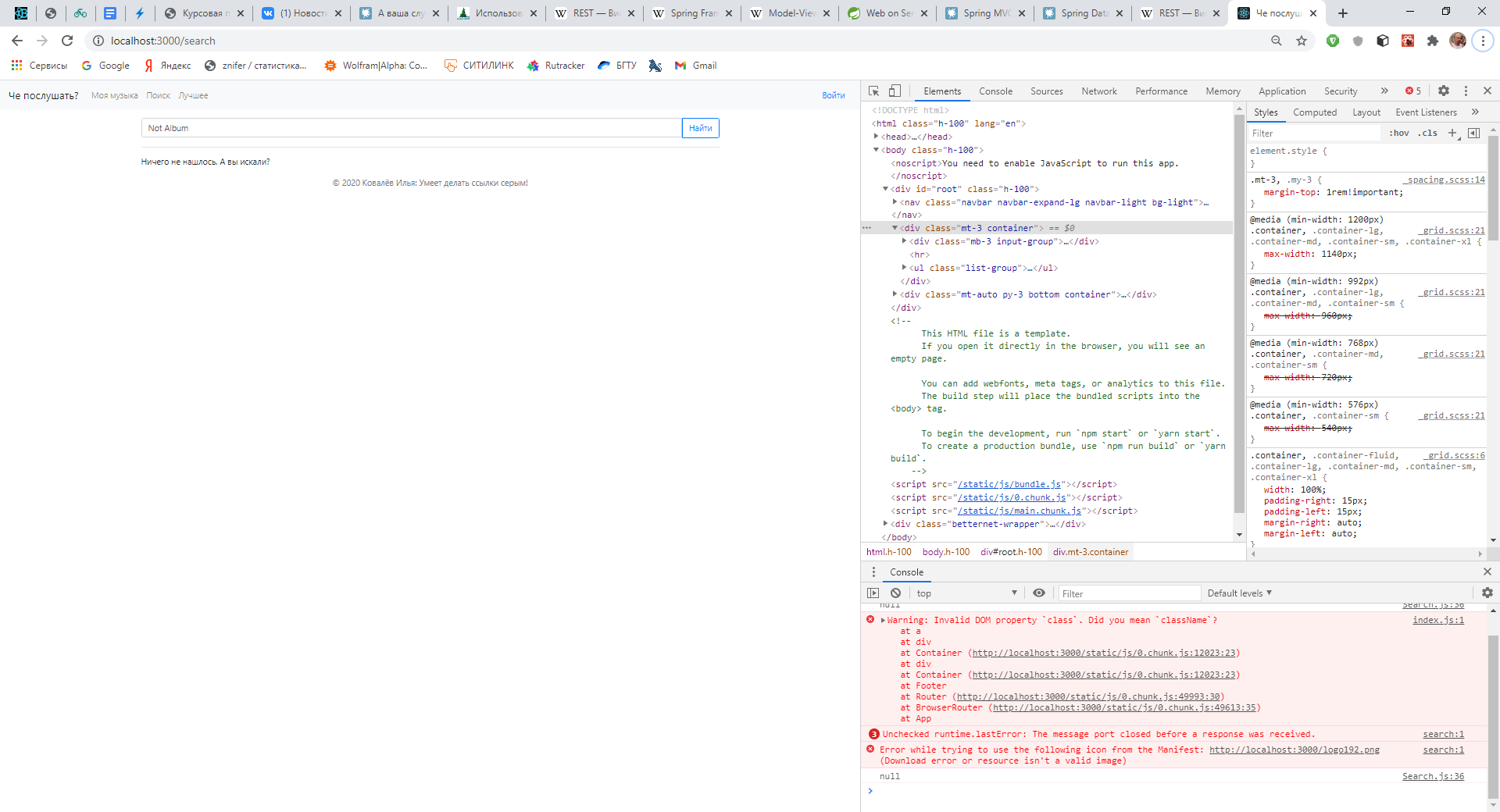
Ниже приведены скриншоты frontend-приложения, использующее разработанную API.



*Рис. 4.5 Главная страница*



*Рис 4.6 Страница поиска в случае, если в строку поиска было введено название существующего альбома*



*Рис 4.7 Страница поиска в случае, если было введено название несуществующего альбома.*

# **Заключение**

В ходе выполнения курсового проекта были получены навыки в разработке клиент-серверного приложения в котором в качестве серверной части будет выступать 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. Java - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Java
2. Apache Maven - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Apache\_Maven
3. Spring Framework - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Spring\_Framework
4. Hibernate (библиотека) - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Hibernate\_(Библиотека)
5. Advanced REST client – [Электронный ресурс]. – URL: https://advancedrestclient.com/
6. Машнин Тимур Сергеевич. Технология Web-сервисов платформы Java. — БХВ-Петербург, 2012. — С. 115. — 560 с.
7. Spring Framework Documentation. – [Электронный ресурс]. – URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/>
8. Spring Boot Reference Documentation. – [Электронный ресурс]. – URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
9. API Help (Lombok). – [Электронный ресурс]. – URL: <https://projectlombok.org/api/help-doc.html>
10. Maven – Maven documentation. – [Электронный ресурс]. – URL: <https://maven.apache.org/guides/index.html>

# **Приложения**

## **Приложение А. Содержимое файла Album.java**

package ru.bstu.iitus.vt41.kid.cw.models;  
  
import lombok.\*;  
  
import javax.persistence.\*;  
import java.time.LocalDate;  
import java.util.List;  
  
@Data  
public class Album {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "album\_id")  
 private Long id;  
 private String title;  
 private LocalDate release\_date;  
 private String cover;  
 private Integer rating;  
 @ManyToOne  
 @JoinColumn(name = "band", referencedColumnName = "id")  
 private Band band;  
  
 @ManyToMany  
 @JoinTable(  
 name = "album\_genres",  
 joinColumns = @JoinColumn(name="album\_id"),  
 inverseJoinColumns = @JoinColumn(name="genre\_id")  
 )  
 private List<Genre> genres;  
}

## **Приложение Б. Содержимое файла Band.java**

package ru.bstu.iitus.vt41.kid.cw.models;  
  
import lombok.\*;  
  
import javax.persistence.\*;  
import java.time.LocalDate;  
import java.util.List;  
  
@Data  
public final class Band {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String name;  
 private String description;  
 private LocalDate created;  
 private LocalDate ended;  
 private String logo;  
 private Integer rating;  
}

## **Приложение В. Содержимое файла Genre.java**

package ru.bstu.iitus.vt41.kid.cw.models;  
  
import lombok.\*;  
  
import javax.persistence.\*;  
  
@Data  
public final class Genre {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "genre\_id")  
 private Long id;  
 private String name;  
  
}

## **Приложение Г. Содержимое файла Track.java**

package ru.bstu.iitus.vt41.kid.cw.models;  
  
import lombok.\*;  
  
import javax.persistence.\*;  
  
@Data  
public final class Track {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "track\_id")  
 private long id;  
 private String title;  
 private String duration;  
 private Integer rating;  
  
 @ManyToOne  
 @JoinColumn(name="album", referencedColumnName = "album\_id")  
 private Album album;  
  
 public Track() {  
  
 }  
}

## **Приложение Д. Содержимое файла AlbumController.java**

package ru.bstu.iitus.vt41.kid.cw.controllers;  
  
import org.springframework.web.bind.annotation.\*;  
import ru.bstu.iitus.vt41.kid.cw.models.Album;  
import ru.bstu.iitus.vt41.kid.cw.repositories.AlbumRepository;  
  
import java.util.List;  
import java.util.Optional;

@CrossOrigin(origins = "\*")  
@RestController  
public class AlbumController {  
 private final AlbumRepository repository;  
  
 AlbumController(AlbumRepository repository) {  
 this.repository = repository;  
 }  
  
 @GetMapping("/albums")  
 List<Album> all(@RequestParam(required = false) String title) {  
 if (title == null) {  
 return repository.findAll();  
 }  
 return repository.findAlbumByTitle(title);  
 }  
  
 @PostMapping("/albums")  
 Album newAlbum(@RequestBody Album newAlbum){  
 return repository.save(newAlbum);  
 }  
  
 @GetMapping("/albums/{id}")  
 Optional<Album> one(@PathVariable Long id){  
 return repository.findById(id);  
 }  
}

## **Приложение Е. Содержимое файла BandController.java**

package ru.bstu.iitus.vt41.kid.cw.controllers;  
  
import org.springframework.web.bind.annotation.\*;  
import ru.bstu.iitus.vt41.kid.cw.models.Band;  
import ru.bstu.iitus.vt41.kid.cw.repositories.BandRepository;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController

@CrossOrigin(origins = "\*")  
public class BandController {  
  
 private final BandRepository repository;  
  
 BandController(BandRepository repository) {  
 this.repository = repository;  
 }  
  
 @GetMapping("/bands")  
 List<Band> all() {  
 return (List<Band>) repository.findAll();  
 }  
  
 @PostMapping("/bands")  
 Band newBand(@RequestBody Band newBand) {  
 return repository.save(newBand);  
 }  
  
 @GetMapping("/bands/{id}")  
 Optional<Band> one(@PathVariable Long id){  
 return repository.findById(id);  
 }  
}

## **Приложение Ж. Содержимое файла AlbumRepository.java**

package ru.bstu.iitus.vt41.kid.cw.repositories;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import ru.bstu.iitus.vt41.kid.cw.models.Album;  
  
import java.util.List;  
  
@Repository  
public interface AlbumRepository extends JpaRepository<Album, Long> {  
  
 List<Album> findAlbumByTitle(String name);  
  
}

## **Приложение Ж. Содержимое файла BandRepository.java**

package ru.bstu.iitus.vt41.kid.cw.repositories;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import ru.bstu.iitus.vt41.kid.cw.models.Band;  
  
import java.util.List;

@Repository  
public interface BandRepository extends JpaRepository<Band, Long> {  
  
 List<Band> findBandByName(String name);  
  
}

**Приложение З. Содержимое файла CwApplication.java**

package ru.bstu.iitus.vt41.kid.cw;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  
  
@SpringBootApplication  
@ComponentScan("ru.bstu.iitus.vt41.kid.cw")  
@EnableJpaRepositories  
public class CwApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(CwApplication.class, args);  
 }  
  
}