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

по дисциплине «Интерфейсы ВС»

Тема: “ Создание клиент-серверного RESTful приложения, с использованием Spring Framework”

Автор работы _____ Шевченко К.К.
(подпись) ВТ-41

Руководитель проекта _____ ст. пр.
(подпись) Торопчин Д.А.

Оценка _____

Белгород
2020 г.

Оглавление

Введение

Целью курсового проекта является создание 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 – приложение, реализованное на языке JavaScript.

В качестве предметной области для курсового проекта был выбран веб-сервис “Автовокзал”. Приложение должно предоставлять возможность просмотра билетов для поездки из пункта отправления в пункт назначения в заданный день.

Выбор данной предметной области был обусловлен тем, что в городе, в котором я родился, никогда не было адекватного веб-сервиса для просмотра и покупки билетов. Не то, что бы очень сильно хотелось что-то исправить, просто было не очень удобно ехать на другую часть города в определенное время для покупки билетов на автовокзале.

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. Права на торговую марку принадлежат корпорации Oracle [1].

Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Иногда к недостаткам концепции виртуальной машины относят снижение производительности. Однако все это в прошлом: ряд усовершенствований увеличил скорость выполнения программ на Java:

- применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде,
- обширное использование платформенно-ориентированного кода (native-код) в стандартных библиотеках,
- аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами архитектуры ARM).

На данный момент, конструкция Java состоит из трех основных компонентов:

- JVM (Java Virtual Machine) – виртуальная машина Java, которая исполняет байт-код, созданный из исходного кода программы компилятором javac.

- JRE (Java Runtime Environment) – минимальная реализация виртуальной машины, необходимая для исполнения Java-приложений. В ее состав не входит компилятор и другие средства разработки.
- 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.

1.2. Spring Framework

Spring Framework - универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET [2].

Фреймворк был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года. Первая стабильная версия 1.0 была выпущена в марте 2004.

Несмотря на то, что Spring не обеспечивал какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет большую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring не всецело связан с платформой Java Enterprise, несмотря на его масштабную интеграцию с ней, что является важной причиной его популярности и признания разработчиков.

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 (иначе говоря, модули) типовых комплексных приложений:

- Inversion of Control-контейнер
- Фреймворк аспектно-ориентированного программирования
- Фреймворк доступа к данным
- Фреймворк управления транзакциями
- Фреймворк MVC

- Фреймворк удалённого доступа
- Фреймворк аутентификации и авторизации
- Фреймворк удалённого управления
- Фреймворк работы с сообщениями
- 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 расшифровывается как Модель-Представление-Контроллер (Model View Controller), что как раз-таки означает три основные части данного фреймворка. Рассмотрим их поподробнее:

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

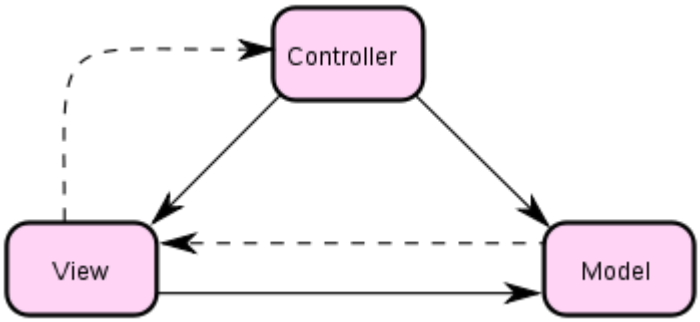

Рис. 1.2.1.: схема MVC фреймворка.

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.

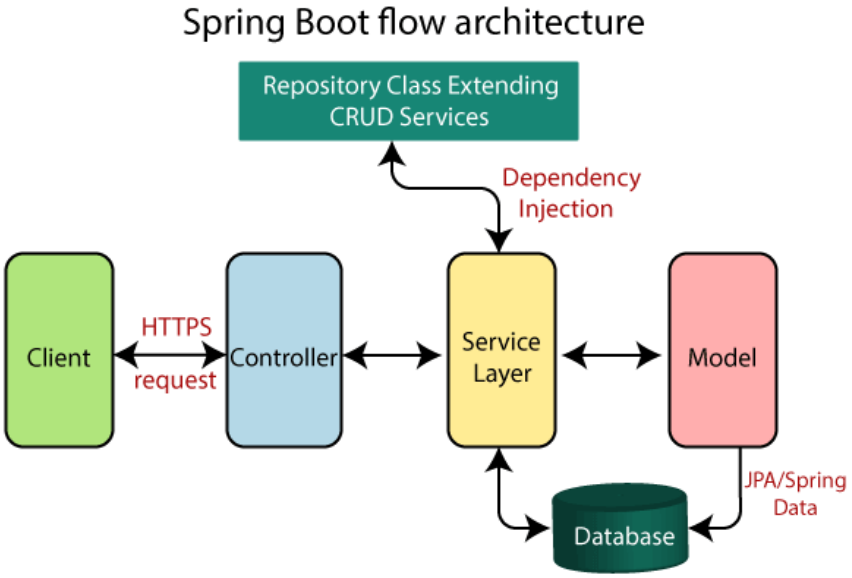

Рис. 1.2.2.: архитектура Spring Boot.

1.3. Maven

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 [3].

Maven обеспечивает декларативную сборку проекта. Это значит, что разработчику не нужно уделять внимание каждому аспекту сборки — все необходимые параметры настроены по умолчанию. Изменения нужно вносить лишь в том объёме, в котором программист хочет отклониться от стандартных настроек. В файлах описания проекта содержится его спецификация, а не отдельные команды выполнения. Все задачи по обработке файлов, описанные в спецификации, Maven выполняет посредством их обработки последовательностью встроенных и внешних плагинов. В основном, Maven используется для построения и управления проектами, написанными на Java, но есть и плагины для интеграции с C/C++, Ruby, Scala, PHP и другими языками.

Конечно же, нельзя не отметить, что Maven ценят за декларативную сборку проекта. Но есть еще одно огромное достоинство проекта — гибкое управление зависимостями. Maven умеет подгружать в свой локальный репозиторий сторонние библиотеки, выбирать необходимую версию пакета, обрабатывать транзитивные зависимости. Разработчики также подчёркивают независимость фреймворка от ОС. При работе из командной строки параметры зависят от платформы, но Maven позволяет не обращать внимания на этот аспект.

1.4. Java Persistence API (JPA)

Java Persistence API (JPA) — спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных [4].

Сама Java не содержит реализации JPA, однако существует множество реализаций данной спецификации от разных компаний (открытых и нет). JPA реализует концепцию ORM.

ORM (Object-Relational Mapping или же объектно-реляционное отображение) – 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.

ORM — это по сути концепция о том, что Java объект можно представить как данные в БД (и наоборот). Она нашла воплощение в виде спецификации JPA — Java Persistence API. Спецификация — это уже описание Java API, которое выражает эту концепцию. Спецификация рассказывает, какими средствами мы должны быть обеспечены (т.е. через какие интерфейсы мы сможем работать), чтобы работать по концепции ORM. И как использовать эти средства. Реализацию средств спецификация не описывает. Это даёт возможность использовать для одной спецификации разные реализации. Можно упростить и сказать, что спецификация — это описание API. Следовательно, чтобы использоать JPA нам требуется некоторая реализацию, при помощи которой мы будем пользоваться технологией. Одна из самых популярных реализаций JPA является 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 [5].

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

Глава 2. Проектирование и разработка приложения

Процесс разработки приложения следует разделить по этапам, первым из которых будет проектирование приложения, реализация клиентской (frontend) составляющей приложения, а затем уже разработка серверной (backend) составляющей приложения.

2.1. Проектирование БД

Первым делом, стоит определить структуру базы данных. Сделаем это вручную, нарисовав таблицу связи данных. Так как проектировать мы будем в MySQL [6] (которая является реляционной СУБД), то имеет смысл сразу начать с реляционной схемы БД.

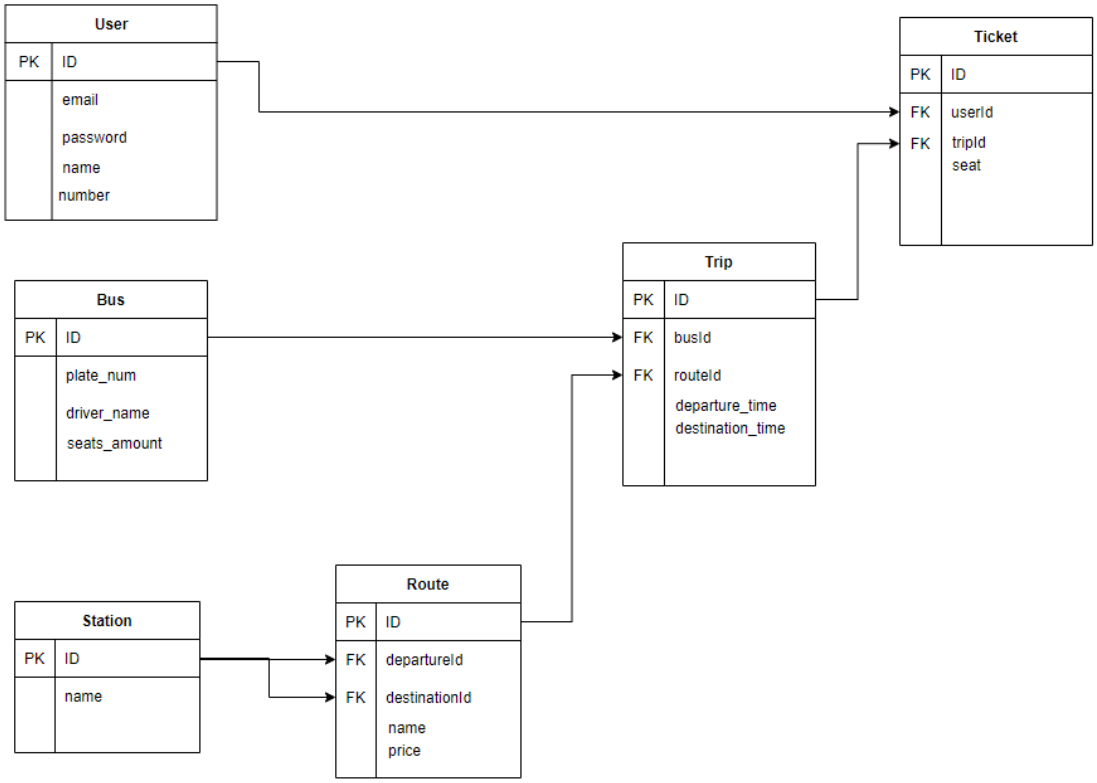

Рис 2.1.1. Реляционная схема БД автовокзала

Разберем каждую сущность по отдельности:

1. User (Пассажир/пользователь) – сущность, которая хранит в себе информацию о пассажире. Например, у пассажира есть такие атрибуты, как:
   - Email – электронная почта. Необязательный атрибут. Через данный атрибут можно сделать авторизацию, но в моем случае, я буду это делать через number.
   - Password – пароль. Обязательный атрибут. Представляет собой 32-х битную строку, содержащую хэш код пароля пользователя. Требуется для авторизации.

- Name – ФИО пользователя. Обязательный атрибут. Требуется для покупки билета, идентификации личности.
- Number – контактный телефон. Обязательный атрибут. Выступает как логин для пользователя.

2. Bus (Автобус) – сущность, которая хранит в себе информацию об автобусе, перевозящем пассажиров. У автобуса есть следующие атрибуты:
    - Plate_num – гос. номер транспортного средства. Обязательный атрибут. Строка.
    - Driver_name – ФИО водителя. Обязательный атрибут. Строка.
    - Seats_amount – кол-во мест в автобусе (всего). Обязательный атрибут. Число.

3. Station (Населенный пункт) – сущность, которая хранит в себе только имя населенного пункта. Комментарии не требуются.

4. Route (Маршрут) – сущность, которая хранит в себе два внешних ключа: id пункта отправления (departureId) и id пункта назначения (destinationId). Также имеет атрибут name – это названия двух станций через тире, и атрибут price – цена маршрута.

5. Trip (Поездка) – сущность, имеющая два внешних ключа: id маршрута (routeId) и id автобуса (busId), а также два атрибута: departure_time и destination_time – время отправления и время прибытия соответственно.

6. Ticket (Билет) – сущность с двумя внешними ключами: userId – id пользователя и tripId – id поездки, к которой билет действителен. Также здесь имеется атрибут seat – номер места в автобусе, которое может занять пассажир в этом автобусе.

Как уже было сказано ранее, Spring Boot имеет фреймворк Spring MVC, поэтому необходимо реализовать модели, соответствующие сущностям нашей БД.

Пример: исходный код модели User

```
import lombok.Data;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Data
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String phone;
    private String email;
    private String password;

    public User(String phone, String password) {
        this.phone = phone;
        this.password = password;
    }
}
```

2.2. Разработка клиентской части приложения

Клиентская часть приложения была разработана с помощью фреймворка Vue JS [7]. В функционале реализованы такие базовые функции, как: авторизация/регистрация, получение списка поездок из пункта отправления в пункт назначения по нужной дате, просмотр свободных билетов, а также покупка билетов.

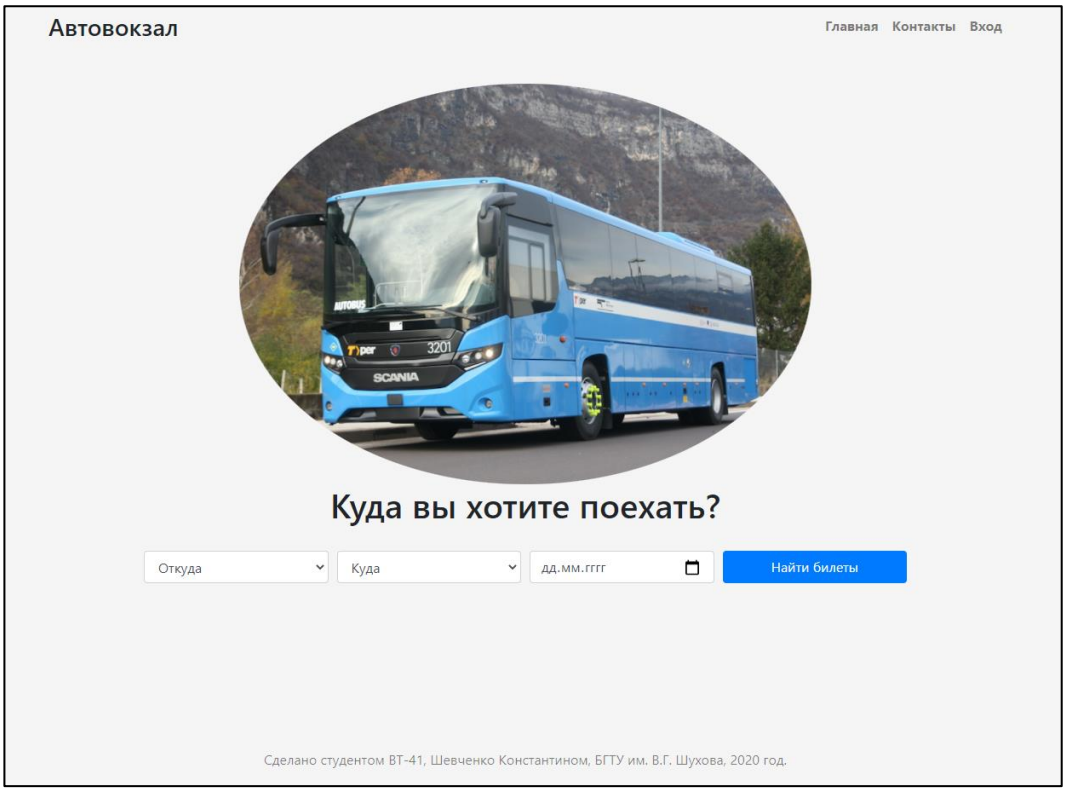

Рис. 2.2.1. Главная страница

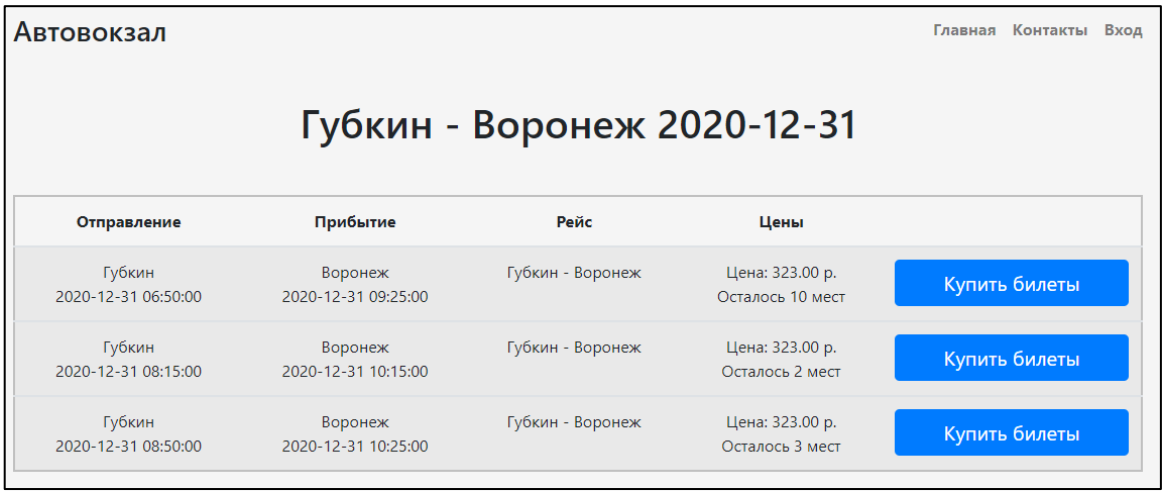

Рис. 2.2.2. Выдача билетов по нужному маршруту

(в данном случае маршрут: Губкин – Воронеж, 31-12-2020)

Пример vue страницы находится в приложении.

2.3. Разработка серверной части приложения

Разработка backend-приложения началась с создания шаблона проекта на Spring Boot с помощью фирменной утилиты Spring Initializer. По итогу был создан файл зависимостей Maven – Pom.xml, основной класс BusStationApplication и файл конфигурации application.property.

Далее для работы с базой данных и Lombok нам потребуется отредактировать файл зависимостей Pom.xml, добавив нужные зависимости соответственно.

В процессе проектирования БД был упомянут фрагмент для создания моделей. Модели являются лишь представлением записей таблиц БД, в дальнейшем их необходимо дополнить функционалом, с помощью которого можно будет получать и заполнять данные базы данных.

Чтобы дополнить функционалом таблицы, мы реализуем интерфейсы к разработанным моделям. Пример модели находится в приложении.

Доступ к функционалу серверной части осуществляется через REST API, которое реализуется контроллерами. Аннотация @Restontroller сообщает фреймворку, что данный класс является REST контроллером и в данном классе будет реализована логика обработки клиентских запросов. @Autowired означает, что для атрибута необходимо внедрить зависимости. Аннотации @GetMapping и @PostMapping реализуют соответственно GET и POST запросы, при этом параметры GET запроса передаются через URL, а POST запроса – в теле запроса.

Нужно понимать, что контроллер не реализует непосредственно логику обработки запроса, а вызывает методы класса-сервиса для соответствующей модели. То есть, можно сказать, что контроллер передает всю работу сервису, а сервис уже реализует логику и вызывает методы, определённые в интерфейсе модели.

Документация к контроллерам REST API выглядит примерно так:

1. Контроллер станций StationsController
   URL: /stations

   Тип запроса: GET

Название/url	Описание	Параметры		
		параметр	тип	обязательный
actionStation	Возвращает id населенного пункта	id	Integer	Да
actionAll_departures	Возвращает пункт отправления	Нет		
actionAll_destinations	Возвращает пункт назначения	Нет		

2. Контроллер билетов TicketsController
   URL: /tickets

   Тип запроса: GET

Название/url	Описание	Параметры		
		параметр	тип	обязательный
actionUser_tickets	Возвращает билеты пользователя по его id	userId	Integer	Да
actionTrip_tickets	Возвращает билеты рейса по его id	tripId	Integer	Да
actionTrip_empty_seats	Возвращает все пустые билеты	tripId	Integer	Да
actionTickets	Возвращает все билеты	Нет		

3. Контроллер пользователей UsersController
   URL: /users

   Тип запроса: GET

Название/url	Описание	Параметры		
		параметр	тип	обязательный
actionLogin	Выполняет авторизацию пользователя по его number и password. Возвращает соответствующего пользователя или Exception.	number password	Integer String(32)	Да Да

   Тип запроса: POST

Название/url	Описание	Данные
actionRegister	Создаёт пользователя с указанным number и password, если такого пользователя не существует, иначе throw Exception	{number, password}

4. Контроллер поездок TripsController
   URL: /trips

   Тип запроса: GET

Название/url	Описание	Параметры		
		параметр	тип	обязательный
actionTrips	Возвращает поездки на дату departureTime, со станции departure и на станцию destination	id	Integer, Integer, data	Да
actionTrip_from_id	Возвращает поездку по его id	id	Integer	Да

2.4. Тестирование

Для тестирования реализованного API воспользуемся Postman [8]. Попробуем написать запрос для получения пунктов назначения.

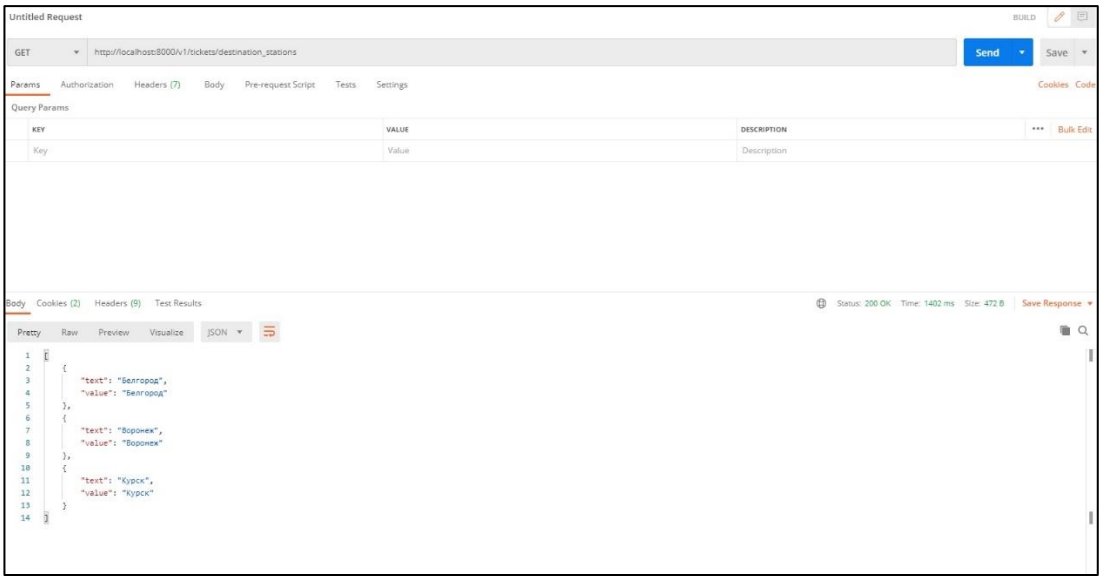

Рис 2.4.1. Запрос через Postman. Как видно на фрагменте экрана, результат успешен

Также проверим нашу БД на работоспособность. Чтобы это сделать, напишем еще один запрос для БД. На этот раз попробуем получить список непроданных билетов (это такие билеты, у которых userId = ‘1’. Идея реализации была такова, чтобы под данным id создать служебного пользователя, который будет иметь при себе все пустые билеты.)

```
SELECT * FROM `tickets` WHERE userId='1'
```

id	userId	tripId	seat	createdAt	updatedAt
1	1	1	1	2020-12-27 15:27:09	2020-12-27 15:27:09
13	1	1	3	2020-12-27 15:28:45	2020-12-27 15:28:45
14	1	2	1	2020-12-27 15:29:08	2020-12-27 15:29:08
15	1	2	2	2020-12-27 15:29:27	2020-12-27 15:29:27
16	1	2	3	2020-12-27 15:30:16	2020-12-27 15:30:16
17	1	3	1	2020-12-27 15:30:28	2020-12-27 15:30:28

Рис 2.4.2. Запрос к БД. Как видно на фрагменте экрана, результат успешен

Теперь попробуем купить билет через клиентскую часть. Например, попробуем купить билет №6. Посмотрим, что вышло.

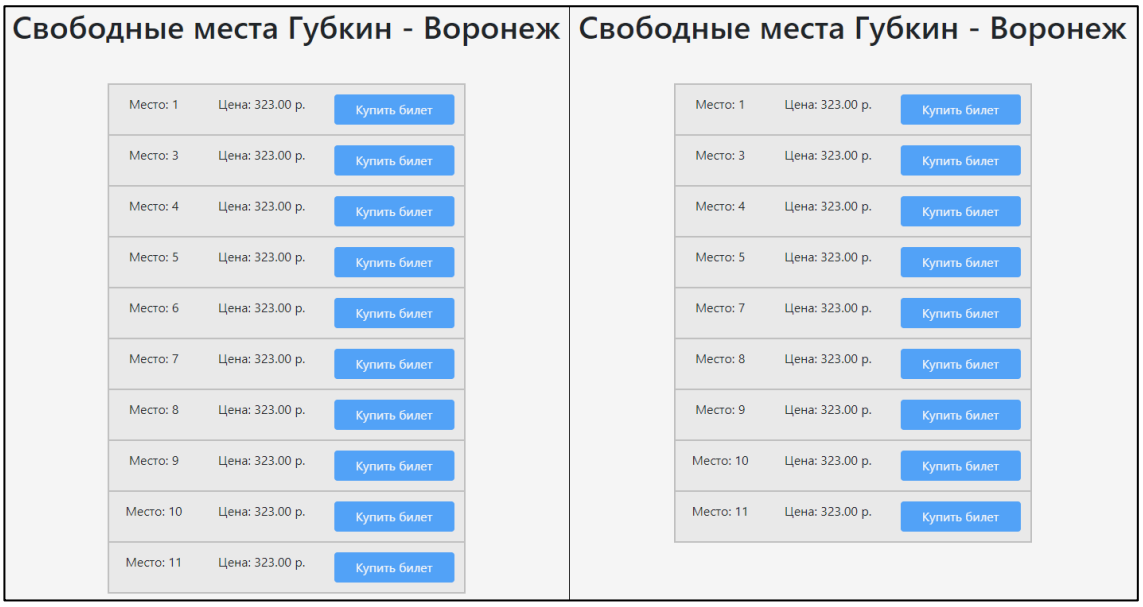

Рис 2.4.2. Покупка билета №6. Как видно на фрагменте экрана, результат успешен

Заключение

В ходе выполнения курсового проекта были получены практические навыки в разработке клиент-серверного RESTful приложения с использованием Spring Framework. В настоящее время популярность информационных технологий растет, а вместе с ними растет и популярность веб-сервисов: люди все чаще ищут различную информации, пользуются интернет-услугами и т.п. И конечно же, не стоят на месте и технологии веб-разработки. С использованием клиент-серверной структуры можно в кратчайшие сроки создать удобное, красивое и эффективное веб-приложение для различных потребностей.

Список литературы

1. Java - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Java
2. Spring Framework - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Spring_Framework
3. Apache Maven - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Apache_Maven
4. Java Persistence_API - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Java_Persistence_API
5. Hibernate (библиотека) - Википедия. – [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Hibernate_(Библиотека)
6. MySQL (СУБД) - Документация – [Электронный ресурс]. – URL: https://dev.mysql.com/doc/
7. Vue JS - Документация – [Электронный ресурс]. – URL: https://ru.vuejs.org/v2/guide/index.html
8. Postman - Документация – [Электронный ресурс]. – URL: https://learning.postman.com/docs/publishing-your-api/documenting-your-api/

Приложение

Содержимое файла Login.vue

```
<template>
  <form class="form-login" @submit="login">
    <h1 class="h3 mb-3 font-weight-normal">Авторизация</h1>
    <label for="inputPhone" class="sr-only">Номер телефона</label>
    <input type="text" id="inputPhone" class="form-control" v-
bind:class="edits_status" placeholder="Номер телефона" required autofocus v-
model="phone" v-on:change="fieldChanged()">
    <label for="inputPassword" class="sr-only">Пароль</label>
    <input type="password" id="inputPassword" class="form-control" v-
bind:class="edits_status" placeholder="Пароль" required v-model="password" v-
on:change="fieldChanged()">
    <button class="btn btn-lg btn-primary btn-block"
type="submit">Войти</button>
    <a class="btn btn-lg btn-success btn-block" href="register">Регистрация</a>
    <div class="mt-3" v-bind:class="edits_status" v-if="edits_status !=
''">Пожалуйста, проверьте правильность написания логина и пароля.</div>
  </form>
</template>

<script>
import User from "@/components/User";
import router from "@/router";
import MD5 from "crypto-js/md5";
export default {
  name: "login",
  data() {
    return {
      email: '',
      password: '',
      edits_status: ""
    }
  },
  methods: {
    signIn(event){
      this.edits_status = ""
      this.$http.get('/user/login/' + this.email + '/' +
MD5(this.password).toString())
          .then((response) => {this.responseProcessing(response.data)})
          .catch((errors) => {console.log(errors)})
      event.preventDefault()
    },
    fieldChanged(){
      this.edits_status = "";
    },
    responseProcessing(user){
      if (user.length === 0)
        this.edits_status = "border-danger text-danger"
      else {
        User.login(user[0])
        router.push({name: 'Home'})
      }
    }
  }
}
</script>
```

Содержимое файла User.java

```
package ru.bstu.iitus.vt41.skk.BusStation.models;
import lombok.Data;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Data
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String phone;
    private String email;
    private String password;

    public User(String email, String password) {
    this.email = email;
    this.password = password;
}
}
```

Содержимое файла TripsController.java

```
package ru.bstu.iitus.vt41.skk.BusStation.models;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import ru.bstu.iitus.vt41.skk.BusStation.models.Trip;
import ru.bstu.iitus.vt41.skk.BusStation.repo.TripRepository;
import ru.bstu.iitus.vt41.skk.BusStation.services.TripService;

import java.util.List;

@RestController
public class TripsController {
    @Autowired
    private TripService service_trip;

    @GetMapping("trip/trips/{departureId}/{destinationId}/{departureDate}")
    private List<Trip> TripsByOptions(@PathVariable("departureId") Long
departureId,
    @PathVariable("destinationId") Long destinationId,
    @PathVariable("departureDate") String departureDate){
    return service_trip.Trips(departureId, destinationId, departureDate);
}
```

Содержимое файла UserService.java

```
package ru.bstu.iitus.vt41.skk.BusStation.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.bstu.iitus.vt41.skk.BusStation.models.User;
import ru.bstu.iitus.vt41.skk.BusStation.repo.UserRepository;
```

```java
import java.util.ArrayList;
import java.util.List;

@Service
public class UserService {
    @Autowired
    UserRepository rep_user;
    public List<User> Login(String number, String password){
    List<User> res = rep_user.findByNumber(number);
    if (res.size() == 0)
    return res;
    String pas = res.get(0).getPassword();
    if (!pas.equals(password))
    res.remove(0);
    return res;
}
public List<User> Register(String number, String password){
    List<User> res = rep_user.findByNumber(number);
    if (res.size() != 0) {
        res.remove(0);
        return res;
    }
    User user = new User(number, password);
    rep_user.save(user);
    res.add(user);
    return res;
}

}
```