



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Шаблоны программных платформ языка Джава

по профилю: _____

направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Приложение «Сервис подбора билетов для путешествий»

Студент: Хитров Никита Сергеевич

Группа: ИКБО-20-21

Работа представлена к защите _____ (дата) _____ / _____ /
(подпись и ф.и.о. студента)

Руководитель: старший преподаватель Зорина Наталья Валентиновна

Работа допущена к защите _____ (дата) _____ /Зорина Н.В./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____
_____ / _____ /



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ на выполнение курсовой работы

по дисциплине: Шаблоны программных платформ языка Джава
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Хитров Никита Сергеевич

Группа: ИКБО-20-21

Срок представления к защите: 17.05.2023

Руководитель: старший преподаватель Зорина Наталья Валентиновна

Тема: Приложение «Сервис подбора билетов для путешествий»

Исходные данные: Индивидуальное задание на разработку; Разработка серверного программного приложения должна быть выполнена с использованием языка Java, а также фреймворком Spring, в частности Spring MVC, Spring REST, Spring Data JPA, Spring Security. Программное приложение должно поддерживать авторизацию пользователей с использованием Cookies или JWT. Нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Провести анализ предметной области и формирование основных требований к приложению, 2. Обосновать выбор средств ведения разработки. 3. Разработать приложение с использованием фреймворка Spring, выбранной технологии и инструментария. 4. Провести тестирование приложения. 5. Оформить пояснительную записку по курсовой работе 6. Провести анализ текста на антиплагиат 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: _____ / Болбаков Р. Г. /, «_____» _____ 2023 г.

Задание на КР выдал: _____ / Зорина Н. В. /, «_____» _____ 2023 г.

Задание на КР получил: _____ / Хитров Н. С. /, «_____» _____ 2023 г.

Хитров Н.С. Приложение «Сервис подбора билетов для путешествий»/ Курсовая работа по дисциплине «Шаблоны программных платформ языка Джава» профиля «Разработка программных продуктов и проектирование информационных систем» направления профессиональной подготовки бакалавриата 09.03.04 «Программная инженерия» (2-ый семестр) / руководитель старший преподаватель Н.В. Зорина / кафедра ИиППО Института ИТ МИРЭА – с. 42, табл. 0, ист. 10.

Целью работы является создание серверного программного приложения на тему «Барбершоп».

В рамках работы осуществлен краткий анализ аналогов веб-приложения по выбранной тематики, разработано приложение с использованием фреймворка Spring, произведено тестирование приложения и проверка на антиплагиат.

Khitrov N.S. Application "Tickets for travel searching service"/ Coursework on the discipline "Templates of program platforms Java" profile "Software Development and Information Systems Engineering" under the bachelor degree 09.03.04 "Software Engineering" (2nd Semester) / supervisor Senior Lecturer N.V. Zorina / Department of IT and Software Engineering of the Institute of IT MIREA - p. 42, tabl. 0, source 10.

The purpose of the work is to create a server software application on the theme "Barber Shop".

As part of the work, a brief analysis of web-application analogues on the selected topic, developed the application using the Spring framework, tested the application and checked for anti-plagiarism.

М. МИРЭА. Ин-т ИИ. Каф. ИиППО. 2023 г. Хитров Н.С.

Аннотация

В курсовой работе описывалось создание интернет-ресурса, на тему «Сервис подбора билетов для путешествий». Работа содержит анализ предметной области разрабатываемого интернет-ресурса, создание веб-страниц интернет-ресурса с использованием технологий Spring Framework и тестирование разработанного приложения.

В введении обосновывается актуальность выбранной темы, определяется цель работы и задачи, подлежащие решению для её достижения, описываются объект и предмет исследования, используемые методы и информационная база исследования, а также кратко характеризуется структура КР по разделам.

В основной части содержится материал, необходимый для достижения цели КР. Основная часть включает в себя общие сведения (в частности, наименование интернет-ресурса, перечисление прикладного программного обеспечения, необходимого для разработки и функционирования интернет-ресурса, а также названия языков и технологий, с помощью которых реализован интернет-ресурс), описание функционального назначения интернет-ресурса и его логической структуры, описание разработки и функций программного продукта, тестирование работы приложения.

В заключении последовательно излагаются теоретические выводы, которые были сформулированы в результате выполнения данной курсовой работы.

Курсовая работа на 42 листах, содержит 16 рисунков, 10 использованных источников, 9 листингов.

The course work described the creation of an Internet resource on the topic "Planetarium". The work contains the analysis of the subject area of the developed Internet resource, the creation of web pages of the Internet resource using the Spring

Framework technologies and testing the developed application.

The introduction substantiates the relevance of the selected topic, defines the purpose of the work and the tasks to be solved to achieve it, describes the object and subject of the research, the methods used and the information base of the research, and also briefly characterizes the structure of the CD by sections.

The main part contains the material needed to achieve the CD goal. The main part includes general information (in particular, the name of the Internet resource, a listing of the application software necessary for the development and operation of the Internet resource, as well as the names of languages and technologies with which the Internet resource is implemented), a description of the functional purpose of the Internet resource and its logical structure, description of the development and functions of the software product, testing the application.

In the conclusion, the theoretical conclusions that were formulated as a result of the implementation of this course work are consistently presented.

Course work on 42 sheets, contains 16 figures, 10 sources used, 9 listings.

СОДЕРЖАНИЕ

Аннотация	6
СОДЕРЖАНИЕ	8
Обозначения и сокращения.....	9
ВВЕДЕНИЕ.....	10
1 СБОР И АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ 12	
2 РАЗРАБОТКА ПРОГРАММНОГО ПРИЛОЖЕНИЯ.....	14
2.1 Проектирование программного продукта.....	14
2.2 Выбор средств и технологий разработки	16
2.3 Структура программного приложения.....	18
2.4 Создание приложения	21
2.4.1 Создание моделей данных	21
2.4.2 Создание JPA репозиторий для работы с базой данных	23
2.4.3 Настройка DTO классов	24
2.4.4 Создание сервисов для реализации логики работы приложения.....	24
2.4.5 Создание контроллеров для обработки REST запросов	30
3 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО API.....	35
ЗАКЛЮЧЕНИЕ	43
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	44

Обозначения и сокращения

AOP	– Aspect-Oriented Programming (Аспектно-ориентированное программирование)
API	– Application Programming Interface (прикладной программный интерфейс)
CRUD	– Create, Read, Update, Delete (Создавать, читать, обновлять, удалять)
DTO	– Data Transfer Object (Объект передачи данных)
DI	– Dependency Injection (Внедрение зависимости)
IoC	– Inversion of Control (Инверсия управления)
IDE	– Integrated Development Environment (Интегрированная среда разработки)
JPA	– Java Persistence API (Java API хранения данных)
JSON	– JavaScript Object Notation (Нотация объектов JavaScript)
ORM	– Object-Relational Mapping (Отображение объектно-ориентированных структур на реляционные данные)
REST	– Representational State Transfer (Архитектурный стиль передачи данных через HTTP)
XML	– eXtensible Markup Language (Расширяемый язык разметки)
IATA	– International Air Transport Association

ВВЕДЕНИЕ

Разработка приложений является невероятно востребованной, однако в то же время и очень сложной и многогранной задачей. Она требует хорошего знания предметной области, а так же технологий и инструментов разработки. Более того, для создания качественного приложения необходима правильно продуманная архитектура. Благодаря использованию фреймворков, таких как Java Spring разработчик имеет возможность решать эти задачи.

Spring Framework - это мощный и популярный фреймворк для разработки приложений на языке Java, который предоставляет широкий набор инструментов и функций для создания высокопроизводительных и масштабируемых приложений. Он основан на принципах инверсии управления и внедрения зависимостей, что позволяет разработчикам создавать гибкие и легко поддерживаемые приложения. Spring Framework также обладает богатым экосистемой расширений и плагинов, которые помогают упростить разработку и ускорить процесс развертывания приложений. Благодаря своей популярности и активному сообществу разработчиков, Spring Framework является одним из наиболее используемых фреймворков для разработки приложений на языке Java.

Одной из ключевых функций Spring является поддержка RESTful архитектуры веб-сервисов, которые позволяют создавать API для взаимодействия с приложением. Для этого определяются контроллеры и маршруты запросов. Spring также предоставляет возможность автоматической сериализации и десериализации данных в формате JSON или XML с помощью библиотеки Jackson. Это позволяет упростить процесс создания RESTful веб-сервисов и обеспечить эффективное взаимодействие между клиентом и сервером.

В данной курсовой работе рассматривается использование Spring Framework для создания RESTful веб-сервиса, применение основных принципов REST, таких как ресурсы, методы и статусы HTTP.

Более того, в работе был рассмотрен процесс взаимодействия с базами данных в Spring, используя Spring Data и JPA (Java Persistence API)[5], создание модели данных, настройка соединения с базой данных и выполнение CRUD

(Create, Read, Update, Delete) операции. Также были рассмотрены слоистая архитектура приложения, реализуемая при помощи модуля Spring MVC, DTO (Data Transfer Object)[6] и механизмы обработки исключений.

Целью курсовой работы является создание интернет-ресурса на тему «Сервис подбора билетов для путешествий». Объектом исследования является интернет-ресурс и люди. Предметом исследования является создание интернет-ресурса, с использованием технологий Spring Framework, JDK и IntelliJIDEA.

Основная часть курсовой работы состоит из следующих разделов:

1. Анализ тестируемого программного продукта;
2. Разработка требований к программному продукту;
3. Разработка программного продукта.
4. Тестирование программного продукта.

Выполнение курсовой работы должно опираться на положения СМК О Инструкцией по организации и проведению курсового проектирования (СМК О МИРЭА 7.5.1/04.И.05-18

1 СБОР И АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

Темой разработанного приложения было выбрано приложение «Сервис подбора билетов для путешествий». Разработанное приложение получило название – «PickTicket».

Для его реализации необходимо провести анализ приложений-аналогов, чтобы выявить основные требования к функционалу сервиса. В качестве объектов анализа были выбраны одни из наиболее популярных агрегаторов билетов для путешествий: OneTwoTrip, TuTu.ru, Kupibilet. В результате анализа были сформированы следующие достаточные функциональные требования к программному приложению.

Функционал для всех пользователей:

- Просмотр подобранных билетов и информации о соответствующих поездках/перелетах
- Фильтрация подбора билетов по:
 - Дате отправления
 - Дате возвращения
 - Видам транспорта
- Вход в аккаунт, регистрация нового аккаунта

Дополнительный функционал для авторизованных пользователей:

- Просмотр истории поисков пользователя
- При доступности (по дате) поиска – быстрый переход к поиску из истории.

Выводы к разделу 1

Исходя из вышеперечисленного можно сделать следующий вывод: web-Сервис должен предоставлять пользователям возможности по поиску билетов между wybranymi населенными пунктами с фильтрацией по датам отправления и возвращения и видам транспорта. Так же должна быть реализована возможность авторизации пользователя. Авторизованный пользователь должен иметь в качестве дополнительной возможности по

отношению к неавторизованным пользователям: просмотр ранее искомых маршрутов и быстрый переход к ним при их доступности по дате.

2 РАЗРАБОТКА ПРОГРАММНОГО ПРИЛОЖЕНИЯ

2.1 Проектирование программного продукта

Для удобного хранения данных была использована СУБД PostgreSQL.

Далее была разработана небольшая БД (Рисунок 2.1.1). В БД входит 4 таблицы:

1. Таблица "person" (таблица пользователей) содержит следующие колонки:
 - id - идентификатор пользователя
 - username – имя пользователя
 - email – адрес электронной почты пользователя
 - password – пароль пользователя закодированный в sha256
 - role – роль пользователя
2. Таблица "flight_request" (таблица истории запросов воздушных маршрутов пользователей) содержит следующие колонки:
 - id - идентификатор запроса
 - origin – место отправления в запросе
 - destination – место прибытия
 - departure_at – дата отправления в запросе
 - return_at – дата возвращения
 - created_at – дата создания запроса пользователем
 - person_id – внешний ключ, связывающий с таблицей «person» по идентификатору пользователя
3. Таблица "train_request" (таблица истории запросов железнодорожных маршрутов пользователей) содержит следующие колонки:
 - id - идентификатор маршрута
 - from – место отправления
 - to – место прибытия
 - departure_at – дата отправления
 - created_at – дата создания запроса пользователем

2.2 Выбор средств и технологий разработки

При выборе технологий для создания приложения были учтены его особенности и функциональные требования.

Для реализации серверной части приложения был выбран фреймворк Spring Boot, который обладает множеством преимуществ. Во-первых, он позволяет создавать сложные приложения с легкостью благодаря своей модульной структуре и встроенным инструментам. Во-вторых, Spring Boot гарантирует высокую безопасность и производительность приложения, что позволяет разработчикам сосредоточиться на бизнес-логике, а не на технических деталях. В-третьих, использование Dependency Injection (DI) / Inversion of Control (IoC) упрощает организацию кода приложения и упрощает его тестирование. Наконец, Spring Boot имеет большой выбор инструментов для работы с базами данных и активное сообщество разработчиков.

В качестве базы данных была выбрана PostgreSQL, которая является мощной системой управления базами данных с открытым исходным кодом. PostgreSQL поддерживает широкий диапазон функций и возможностей, включая многопоточность, транзакционность и поддержку различных типов данных. Кроме того, PostgreSQL обладает отличной производительностью и масштабируемостью, что делает ее оптимальным выбором для создания приложений любой сложности.

Для взаимодействия между серверной и клиентской частями приложения использовался протокол HTTP и формат передачи данных JSON. HTTP является стандартным протоколом для обмена данными в Интернете, что делает его удобным выбором для создания веб-приложений. Формат JSON был выбран из-за его простоты и удобства в использовании, а также из-за широкой поддержки этого формата в различных языках программирования.

Для ORM был выбран фреймворк Hibernate, который упрощает работу с базами данных, скрывая от разработчика детали взаимодействия с БД и позволяет использовать различные стратегии отображения данных. Hibernate также обеспечивает высокую производительность приложения благодаря

своей оптимизации запросов к БД и кэшированию данных.

Для уменьшения количества кода использовалась библиотека Lombok, которая генерирует методы автоматически во время компиляции кода. Это позволяет сократить количество повторяющегося кода и упростить его чтение и понимание, что в свою очередь ускоряет процесс разработки приложения.

В качестве технологий разработки клиентской части сервиса было принято решение использовать:

- HTML5 – наиболее удобная и популярная технология описания разметки веб-страниц
- CSS – наиболее удобная и популярная технология описания стилей элементов веб-страниц
- React Framework в связке с Redux state-manager – конкретно React был выбран исходя из личных интересов конкретно к этому фреймворку, в целом, фреймворк был выбран для использования его большого функционала в отличие от нативного js, state-manager Redux был выбран, так как на данный момент является самой оптимизированной технологией стейт-менеджмента, так же его популярность на данный момент самая высокая среди подобных технологий и, как следствие, на рынке труда кандидаты со знанием его более востребованы

Для настройки межстраничной навигации была использована библиотека react-router-dom, позволяющая делать веб-приложение одностраничным, с перерисовкой необходимых компонентов, в отличие от чистого HTML где маршрутизация по страницам вызывает полную подгрузку соответствующих файлов .html

Для обеспечения безопасности в каждом запросе к серверу в заголовках запроса отправляется JSON Web Token, сохраняемый в cookie браузера (при помощи библиотеки universal-cookie) и получаемый от сервера при входе или регистрации пользователя

Отправка запросов к серверу осуществляется при помощи асинхронного вызова методов библиотеки axios, предоставляющей удобство

работы с запросами.

2.3 Структура программного приложения

Корневой папкой программного продукта является `ru.coursework.flightSearchSystem` (Рисунок 2.3.1). Она содержит следующие папки:

1. `config` - содержит классы конфигурации. Классы конфигурации служат для настройки `SpringSecurity`.
2. `controllers` - содержит классы контроллеров, предоставляющие данные пользователю.
3. `dto` - содержит классы, которые используются для передачи данных между различными слоями приложения и между приложением и клиентом.
4. `entities` - пакет, содержащий классы сущностей.
5. `repositories` – хранит интерфейсы, которые взаимодействуют с БД.
6. `security` - Пакет `Security` содержит классы, отвечающие за авторизацию на основе JWT (JSON Web Token) токенов.
7. `services` – пакет, который хранит классы сервисов – оболочек репозитория.
8. `util` - Пакет `mappers` содержит классы, которые выполняют преобразование объектов одного типа в объекты другого типа.

В папке `resources` находятся файлы для работы библиотеки `liquibase` позволяющей создавать и настраивать таблицы базы данных при запуске приложения, `liquibase.properties` для настройки `liquibase` и ее связи с базой данных. Также есть `application.properties` для связывания приложения с базой данных. В подпапке `static` находятся файлы `json` содержащие информацию о населенных пунктах, аэропортах, станциях жд и автобусных станциях для отправки запросов к другим API. `pom.xml` - основной файл системы сборки проекта Maven, папка `test` содержит классы модульного тестирования приложения (Рисунок 2.3.2).

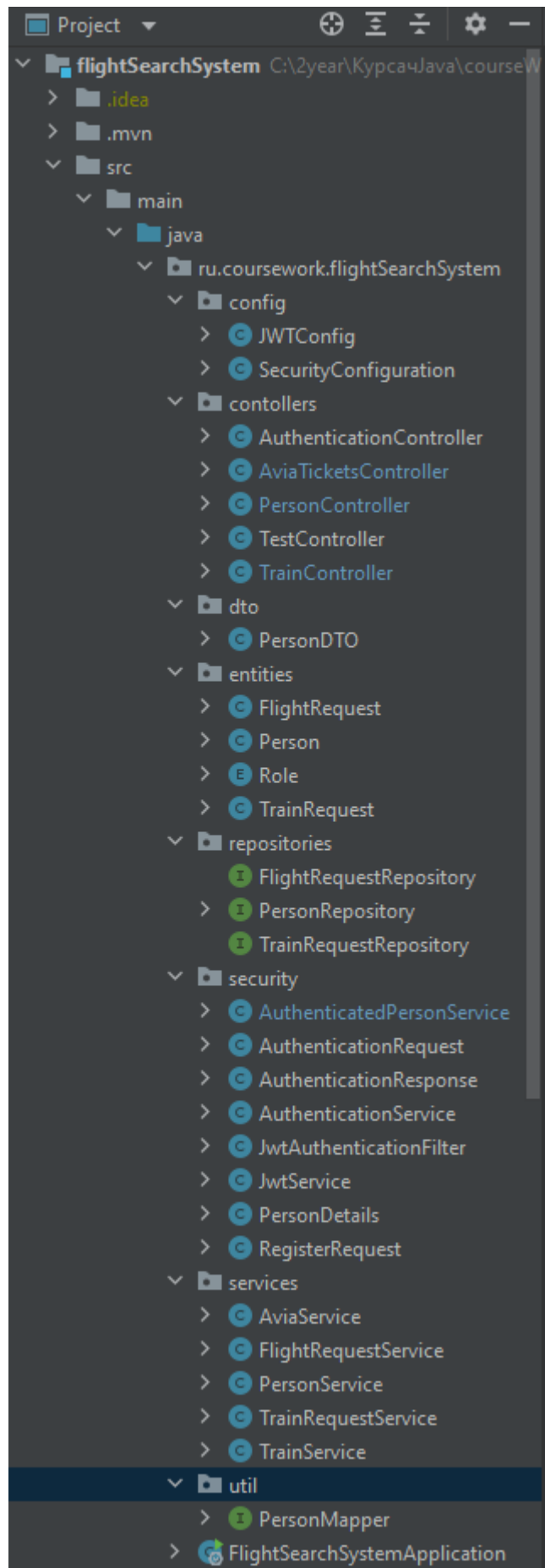


Рисунок 2.3.1 – Содержимое корневого пакета

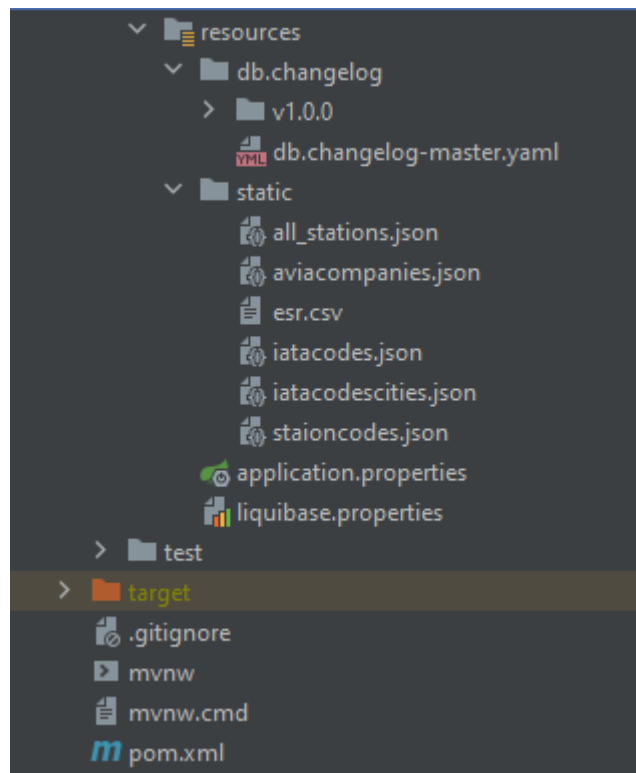


Рисунок 2.3.2 – важные файлы проекта

2.4 Создание приложения

2.4.1 Создание моделей данных

Для разработки сервиса было необходимо создать модели данных описывающие основные сущности приложения. Среди них – модель запроса поиска авиабилетов (FlightRequest), модель запроса поиска билетов на поезд (TrainRequest), модель роли пользователя для целей безопасности (Role), модель пользователя (Person). Каждая из этих моделей имеет свои поля и методы, которые описывают ее поведение в системе.

Для создания моделей данных использовались такие аннотации как @Entity, которая указывает, что класс является сущностью базы данных., @Data - генерирует геттеры, сеттеры, методы toString(), equals() и hashCode(),

Класс "Person" - модель пользователя в системе. Он содержит поля "id", "email", "password", "username", "role". Поле "id" – уникальный идентификатор пользователя, который генерируется автоматически при создании нового пользователя. Поле "email" хранит адрес электронной почты пользователя, а поле "password" - пароль пользователя. Поле "username" - имя пользователя. Поле "role" указывает на роль пользователя в системе, например, администратор или обычный пользователь. Поле "id" имеет аннотации "@Id" и "@GeneratedValue(strategy = GenerationType.IDENTITY)", а поле "role" имеет аннотацию "@Enumerated(value = EnumType.STRING)".

Модель Person представляет собой важный элемент системы, позволяющий управлять пользователями приложения.

Класс "FlightRequest" представляет модель запроса поиска авиабилетов в системе, которые я сохраняю для отображения истории поиска пользователя, и содержит поля: "id", "origin", "destination", "departure_at", "return_at", "created_at", "person_id". Поле "id" является уникальным идентификатором поискового запроса, а поля "origin" и "destination" хранят аэропорт вылета и прибытия соответственно. Поля "departure_at" и "return_at" хранят дату вылета и возврата соответственно, "person_id" – уникальный идентификатор пользователя, совершившего запрос. Код модели представлен на листинге

2.4.1.1.

Листинг 2.4.1.1 – код модели запроса поиска авиабилетов

```
@FieldDefaults(level = AccessLevel.PRIVATE)
@Data
@AllArgsConstructor
@RequiredArgsConstructor
@Entity
@Table(name = "flight_request")
public class FlightRequest {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    long id;

    String origin;
    String destination;
    LocalDate departure_at;
    LocalDate return_at;
    LocalDate created_at;

    long person_id;
}
```

Класс "TrainRequest" содержит поля: "id", "from" и "to", "departure_at" и "created_at". Поле "id" является уникальным идентификатором запроса на поиск билетов на поезд, которое генерируется автоматически при создании нового запроса. Поле "from" хранит станцию отправления. Поле "to" хранит станцию прибытия. Дата отправления хранится в поле "departure_at", дата создания запроса хранится в поле "created_at". Код модели представлен на листинге 2.4.1.2.

Листинг 2.4.1.2 – код модели запроса поиска билетов на поезд

```
@FieldDefaults(level = AccessLevel.PRIVATE)
@Data
@AllArgsConstructor
@RequiredArgsConstructor
@Entity
@Table(name = "train_request")
public class TrainRequest {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    long id;

    @Column(name = "fromm")
    String from;
```

Продолжение листинга 2.4.1.2

```
@Column(name = "too")
String to;
@Column(name = "departure_at")
LocalDate departureAt;
@Column(name = "created_at")
LocalDate createdAt;

long person_id;
}
```

2.4.2 Создание JPA репозиториев для работы с базой данных

Репозитории в Java Spring являются частью слоя доступа к данным (Data Access Layer)[9] и предназначены для управления объектами, хранимыми в базе данных. Они предоставляют высокоуровневый API для выполнения операций CRUD (Create, Read, Update, Delete) и запросов к базе данных, что упрощает работу с данными.

Репозитории позволяют абстрагироваться от конкретных технологий хранения данных и скрыть сложность работы с базой данных за простым интерфейсом. Это позволяет существенно упростить разработку приложений и снизить затраты на поддержку и обновление кода.

Важным аспектом репозиториев является возможность использования Spring Data JPA[10], который позволяет создавать репозитории с помощью аннотаций и автоматически генерировать SQL-запросы на основе сигнатур методов. Это позволяет повысить производительность, уменьшить количество кода и сократить время на разработку приложения.

Еще одним важным аспектом репозиториев является возможность использования транзакций. Транзакции обеспечивают целостность данных, позволяют избежать ошибок в случае нескольких одновременных обращений к базе данных и гарантируют, что все изменения будут либо применены, либо откатываются.

Для автоматического определения репозиториев, они были помечены аннотацией `@Repository`.

Для всех сущностей использовались стандартные методы интерфейса репозитория JpaRepository. На листинге 2.4.2 представлен пример кода репозитория.

Листинг 2.4.2 – код репозитория запросов поиска авиабилетов

```
@Repository
public interface PersonRepository extends JpaRepository<Person,
Long> {
    Optional<Person> findByEmail(String email);
}
```

2.4.3 Настройка DTO классов

Классы DTO (Data Transfer Object) в Java Spring используются для передачи данных между различными слоями приложения. Они представляют собой простые POJO-классы, которые содержат только необходимые поля и методы для передачи данных. DTO-классы могут использоваться для сериализации и десериализации данных, а также для уменьшения количества запросов к базе данных. Они являются частью паттерна проектирования "Data Access Object" (DAO) и помогают упростить разработку приложений.

В разработке был использован один DTO класс – PersonDTO, для преобразования необходимых данных из таблицы. Реализация класса представлена на листинге 2.4.3

Листинг 2.4.3 – Код класса PersonDTO

```
@Getter
@Setter
public class PersonDTO {
    private String username;
    private String email;
}
```

2.4.4 Создание сервисов для реализации логики работы приложения

Сервисы в Spring Framework – это компоненты приложения, которые выполняют бизнес-логику и обрабатывают запросы от других компонентов. Они предназначены для выделения бизнес-логики в отдельный слой приложения.

Сервисы в Spring могут быть созданы с помощью аннотации `@Service`. Эта аннотация позволяет Spring IoC контейнеру управлять жизненным циклом[11] сервиса и предоставлять его для инъекции в другие компоненты приложения.

Сервисы могут использовать механизм внедрения зависимостей (Dependency Injection)[12], который позволяет управлять зависимостями между компонентами приложения. Использование DI позволяет упростить код и уменьшить количество дублирующегося кода.

Сервисы могут использовать аннотацию `@Transactional`, чтобы управлять транзакциями в приложении. Это позволяет убедиться в том, что при выполнении операций над данными будет использоваться транзакционный контекст, что обеспечивает целостность данных и предотвращает их потерю или порчу.

Были разработаны классы-сервисы отвечающие за реализацию бизнес-логики каждой отдельной сущности. На листинге 2.4.4.1 изображен пример кода класса-сервиса.

Листинг 2.4.4.1 – код сервиса PersonService

```
@Service
@RequiredArgsConstructor
public class PersonService {

    private final PersonRepository personRepository;
    private final PersonMapper personMapper =
PersonMapper.INSTANCE;

    public List<Person> findAll() {
        return personRepository.findAll();
    }
    public void delete(long id) {
        personRepository.delete(getById(id));
    }
    public Person getById(long id) {
        Optional<Person> person =
personRepository.findById(id);
        return person.get();
    }
    public Person update(long id, Person person) {
        return personRepository.save(person);
    }
}
```

Класс `PersonService` – сервис отвечающий за обработку запросов, связанных с пользователями. Он содержит методы для обновления, удаления и поиска пользователей в базе данных через объект `PersonRepository`.

Конструктор класса использует аннотацию `@RequiredArgsConstructor` из библиотеки `Lombok`, что позволяет сгенерировать автоматический конструктор со всеми требуемые полями, которые будут инъектироваться другими компонентами приложения.

Метод `findAll` возвращает список всех пользователей, найденных в базе данных через репозиторий `PersonRepository`.

Метод `findById(Long id)` находит пользователя по его идентификатору `id` и возвращает объект `Person`. Если пользователь не найден, возвращает `null`.

Метод `update(Long id, Person person)` находит пользователя по его идентификатору `id`, обновляет его данные на основе данных из объекта `Person`, сохраняет изменения в базе данных через репозиторий `PersonRepository` и возвращает объект `Person`. Если пользователь не найден, возвращает `null`.

Метод `delete(Long id)` удаляет пользователя из базы данных по его идентификатору `id`.

`AviaService` – сервис отвечающий за реализацию бизнес-логики связанной с работой с API данных об авиабилетах. Сервис содержит методы получения IATA кода аэропорта по названию города, в котором расположен аэропорт, получения имени авиакомпании по ее IATA коду, конвертации кода авиакомпании в ее название, добавления IATA кода авиакомпании в ответный JSON.

Метод `findIATACode(String city)` принимает на вход название города и возвращает соответствующий ему IATA-код, используя файл с JSON-объектами, содержащими информацию об аэропортах.

Метод `getCompanyNameByCode(String code)` принимает на вход код авиакомпании и возвращает ее название, используя файл с JSON-объектами, содержащими информацию об авиакомпаниях.

Метод `convertCompanyCodeToName(JsonNode dataNode)` принимает на

вход узел JSON-дерева и заменяет код авиакомпании на ее название, используя метод `getCompanyNameByCode`.

Метод `addIATACodeOfAirline(JsonNode dataNode)` принимает на вход узел JSON-дерева и добавляет новое поле "IATA", содержащее IATA-код авиакомпании, используя метод `getCompanyNameByCode`.

Класс `TrainService` отвечает за работу с информацией о железнодорожных станциях.

Метод `deleteBusStops()` удаляет информацию о автобусных остановках из файла `src/main/resources/static/staioncodes.json`. Для этого метод читает содержимое файла, проходит по всем станциям и удаляет те, у которых значение поля `station_type` равно `"bus_stop"`.

Метод `findCodeByName(String name)` принимает на вход название станции и возвращает соответствующий ей код Yandex. Метод читает содержимое файла `staioncodes.json`, проходит по всем станциям и возвращает код той, у которой значение поля `title` равно заданному названию. Если станция с таким названием не найдена, метод возвращает `null`

Класс `FlightRequestService` представляет собой сервис, отвечающий за обработку запросов на авиарейсы. Он содержит два метода для сохранения и получения списка запросов на рейсы.

Метод `saveRequest` принимает объект типа `FlightRequest` и сохраняет его в репозитории `flightRequestRepository`.

Метод `findAll` возвращает список всех запросов на рейсы, хранящихся в репозитории `flightRequestRepository`.

Класс `AuthenticationService` реализует методы аутентификации и регистрации пользователей. Он содержит зависимости `PersonRepository`, `PasswordEncoder`, `JwtService` и `AuthenticationManager`, которые используются для выполнения логики.

Метод `register(RegisterRequest request)` принимает на вход объект `RegisterRequest`, создает нового пользователя, кодирует его пароль с помощью `PasswordEncoder`, устанавливает роль `USER` и сохраняет его в репозитории.

Затем генерируется JWT-токен с помощью `JwtService`, содержащий информацию о пользователе, и возвращается объект `AuthenticationResponse` с токеном.

Метод `authenticate(AuthenticationRequest request)` принимает на вход объект `AuthenticationRequest`, проверяет подлинность учетных данных пользователя с помощью `AuthenticationManager`, находит соответствующего пользователя в репозитории и генерирует JWT-токен с помощью `JwtService`. Затем возвращается объект `AuthenticationResponse` с токеном. Данный класс представлен на листинге 2.4.4.2.

Листинг 2.4.4.2 – код сервиса `AuthenticationService`

```
@Service
@RequiredArgsConstructor
public class AuthenticationService {
    private final PersonRepository repository;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private final AuthenticationManager authenticationManager;

    public AuthenticationResponse register(RegisterRequest request) {
        var user = Person.builder()
            .username(request.getUsername())
            .email(request.getEmail())

            .password(passwordEncoder.encode(request.getPassword()))
            .build();
        user.setRole(Role
            .USER);

        var savedUser = repository.save(user);
        var jwtToken = jwtService.generateToken(new
        PersonDetails(user));
        return AuthenticationResponse.builder()
            .token(jwtToken)
            .build();
    }

    public AuthenticationResponse
    authenticate(AuthenticationRequest request) {
        authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                request.getEmail(),
                request.getPassword()
            )
        )
    }
}
```

Продолжение листинга 2.4.4.2

```
        );  
        var user = repository.findByEmail(request.getEmail())  
                                .orElseThrow();  
        var jwtToken = jwtService.generateToken(new  
PersonDetails(user));  
        return AuthenticationResponse.builder()  
                                .token(jwtToken)  
                                .build();  
    }  
}
```

Класс `JwtService` предоставляет методы для генерации, проверки и извлечения данных из JSON Web Token (JWT).

Метод `extractUsername` извлекает имя пользователя из токена.

Метод `extractClaim` извлекает произвольный клейм из токена, используя переданную функцию для определения нужного клейма.

Метод `generateToken` генерирует токен на основе переданных данных пользователя и дополнительных клеймов.

Метод `isTokenValid` проверяет, является ли переданный токен действительным для указанного пользователя.

Приватные методы `isTokenExpired`, `extractExpiration` и `extractAllClaims` извлекают соответствующие данные из токена.

Приватный метод `getSignInKey` возвращает ключ для подписи токена на основе секретного ключа.

Класс `AuthenticatedPersonService` предоставляет метод `getAuthenticatedPerson()`, который возвращает аутентифицированного пользователя из контекста безопасности.

Метод `getAuthenticatedPerson()` извлекает информацию об аутентифицированном пользователе из контекста безопасности, используя класс `PersonDetails`, который содержит объект пользователя и его авторизационные данные.

Метод использует методы Spring Security, такие как `SecurityContextHolder.getContext()` и `getAuthentication()`, чтобы получить доступ к аутентификационным данным пользователя в контексте безопасности.

Данный класс позволяет упростить получение информации об аутентифицированном пользователе в приложении и использовать эту информацию для выполнения необходимых действий.

2.4.5 Создание контроллеров для обработки REST запросов

RESTful контроллеры в Spring[13] используются для создания веб-сервисов, которые предоставляют клиентам API для доступа к данным и функциональности на сервере. Они позволяют обрабатывать запросы через HTTP-протокол и возвращать данные в формате JSON/XML[14].

Spring Framework предоставляет ряд аннотаций для работы с RESTful контроллерами. Одной из наиболее часто используемых является `@RestController`, которая указывает, что класс является RESTful контроллером и будет обрабатывать входящие HTTP-запросы[15]. Другие полезные аннотации включают `@GetMapping`, `@PostMapping`, `@PutMapping` и `@DeleteMapping`, которые указывают тип запроса, который будет обработан методом контроллера, а также `@RequestBody`, который используется для чтения данных запроса HTTP и преобразования их в объект Java. `@PathVariable` используется для получения параметров URL, а `@RequestParam` – для чтения параметров запроса HTTP. Другие аннотации, такие как `@ResponseStatus`, `@CrossOrigin`, `@ModelAttribute`, `@Valid` и `@ExceptionHandler`, также могут использоваться для управления поведением RESTful контроллеров в Spring. Каждая аннотация имеет свою особенность и может использоваться в зависимости от конкретной задачи.

Для реализации управления приложением были реализованы следующие классы контроллеры: `AviaTicketsController`, `TrainController`, `PersonController`, `AuthenticationController`. На листинге 2.4.5 представлен пример кода класса-контроллера.

Листинг 2.4.5 – код контроллера запросов поиска авиабилетов

```
@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequiredArgsConstructor
public class AviaTicketsController {

    private final AviaService aviaService;
    private final AuthenticatedPersonService
authenticatedPersonService;
    private final FlightRequestService flightRequestService;

    @PostMapping("/getFlights")
    public JsonNode getFlights(@RequestBody FlightRequest
flightRequest) throws IOException {

        String origin =
aviaService.findIATACode(flightRequest.getOrigin());
        String destination =
aviaService.findIATACode(flightRequest.getDestination());

        String urlToApi =
"https://api.travelpayouts.com/aviasales/v3/prices_for_dates?"
+
        "origin=" + origin + "&destination=" +
destination +
        "&departure_at=" +
flightRequest.getDeparture_at() +
        "&return_at=" + flightRequest.getReturn_at() +
        "&sorting=price&direct=true&limit=10&" +
        "token=15f19213084d9b861001e4d44ffe7d08";

        RestTemplate restTemplate = new RestTemplate();
        String result = restTemplate.getForObject(urlToApi,
String.class);

        ObjectMapper mapper = new ObjectMapper();
        JsonNode jsonNode = mapper.readTree(result);

        JsonNode dataNode = jsonNode.get("data");

        aviaService.addIATACodeOfAirline(dataNode);
        aviaService.convertCompanyCodeToName(dataNode);

        flightRequest.setCreated_at(LocalDate.now());

        flightRequest.setPerson_id(authenticatedPersonService.getAuthen
ticatedPerson().getId());

        flightRequestService.saveRequest(flightRequest);
    }
}
```

Продолжение листинга 2.4.5

```
        return dataNode;
    }

    @GetMapping("/get_flight_search_history")
    public List<FlightRequest> getSearchHistory() {
        long personId =
authenticatedPersonService.getAuthenticatedPerson().getId();

        List<FlightRequest> flightRequests = new ArrayList<>();

        for(FlightRequest flight :
flightRequestService.findAll()) {
            if (flight.getPerson_id() == personId)
                flightRequests.add(flight);
        }
        return flightRequests;
    }
}
```

Класс `PersonController` - это контроллер, который обрабатывает запросы, связанные с сущностью `Person`. Он содержит методы для получения списка всех пользователей и текущего пользователя.

Метод `findAll()` обрабатывает GET-запрос по адресу `/people` и возвращает список всех пользователей. Внутри метода используется `personService` для получения списка пользователей, затем каждый пользователь маппится в DTO с помощью `personMapper` и добавляется в результирующий список.

Метод `getPerson()` обрабатывает GET-запрос по адресу `/get_person` и возвращает текущего пользователя. Внутри метода используется `authenticatedPersonService` для получения текущего пользователя через `Spring Security` и возвращает его объект типа `Person`.

Оба метода имеют `JavaDoc`, которые описывают их работу и возвращаемые значения.

Класс `AviaTicketsController` представляет собой REST контроллер для поиска авиабилетов и получения истории поиска. Контроллер содержит два метода: `getFlights` и `getSearchHistory`.

Метод `getFlights` принимает запрос в формате JSON, содержащий информацию о городе вылета, городе прибытия, дате вылета и дате возвращения. Метод отправляет запрос на API сервис для получения списка авиабилетов, соответствующих параметрам запроса. Далее, метод преобразует полученные данные в формат JSON и сохраняет данные запроса в базу данных. Метод возвращает список авиабилетов, соответствующих параметрам запроса в формате JSON.

Метод `getSearchHistory` возвращает список запросов на поиск авиабилетов, сделанных текущим пользователем. Метод получает `id` текущего пользователя, и затем получает все сохраненные запросы на поиск авиабилетов. Метод возвращает список запросов на поиск авиабилетов пользователя в формате JSON.

Класс `TrainController` является REST контроллером, который обрабатывает HTTP запросы, связанные с поиском поездов между двумя станциями в конкретную дату. Данный контроллер взаимодействует с другими сервисами: `TrainService`, `TrainRequestService`, `AuthenticatedPersonService`, `EntityManager`.

Метод `getTrains()` является обработчиком HTTP POST запроса на адрес `"/getTrains"`. Он получает на вход запрос на поиск поезда между двумя станциями в конкретную дату и возвращает JSON объект, содержащий информацию о найденном поезде. Для поиска используется сервис `TrainService`, который определяет коды станций, между которыми ищется поезд, а затем происходит обращение к API Яндекс.Расписания через HTTP запрос с помощью класса `RestTemplate`. Полученный ответ парсится с помощью `ObjectMapper` в объект `JsonNode`, из которого извлекается необходимая информация и формируется ответ в виде JSON.

Класс `AuthenticationController` является контроллером Spring, который обрабатывает HTTP-запросы для аутентификации пользователей. Он содержит два метода POST-запроса: `register` и `authenticate`, которые обрабатывают запросы на регистрацию новых пользователей и аутентификацию существующих пользователей соответственно. Класс использует аннотации `@CrossOrigin`, `@RestController` и `@RequiredArgsConstructor` для работы веб-приложения и внедрения сервиса аутентификации.

Метод `register` принимает POST-запрос, который содержит данные нового пользователя в формате JSON и передает их сервису аутентификации для создания новой учетной записи пользователя. Метод возвращает JWT-токен в виде JSON-объекта.

Метод `authenticate` принимает POST-запрос, который содержит данные пользователя, такие как имя пользователя и пароль, в формате JSON, и передает их сервису аутентификации для проверки подлинности пользователя. Метод возвращает JWT-токен в виде JSON-объекта.

3 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО API

На рисунке 3.1 изображено тестирование регистрации. Отправляется запрос с почтой, именем пользователя и паролем, при этом возвращается JWT токен.

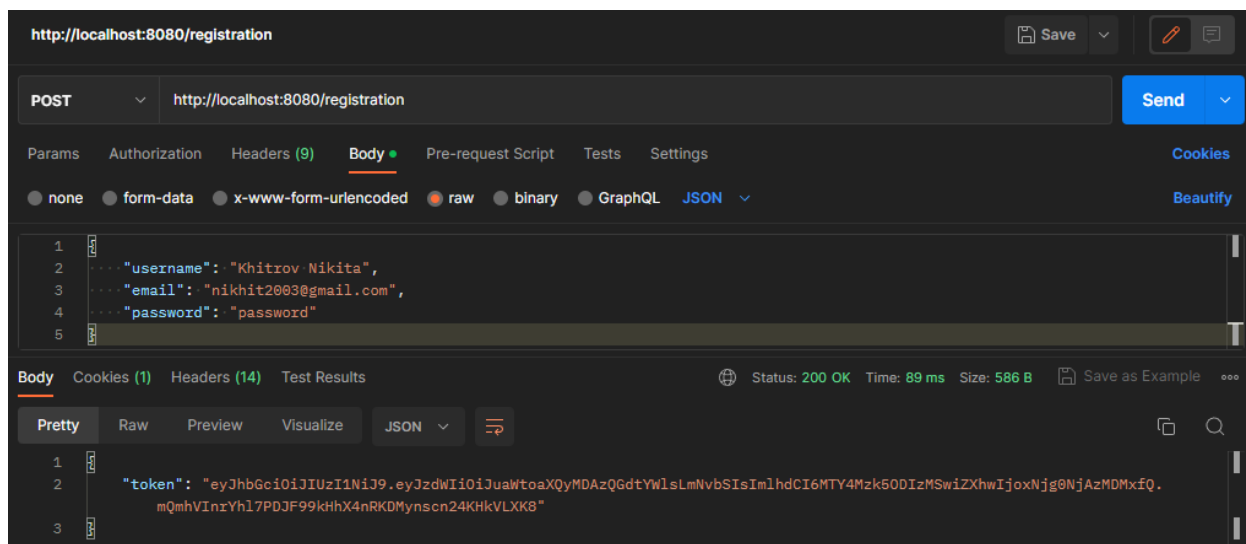


Рисунок 3.1 - Тестирование регистрации

На рисунке 3.2 изображено тестирование авторизации. Отправляется запрос с почтой и паролем, при этом возвращается JWT токен.

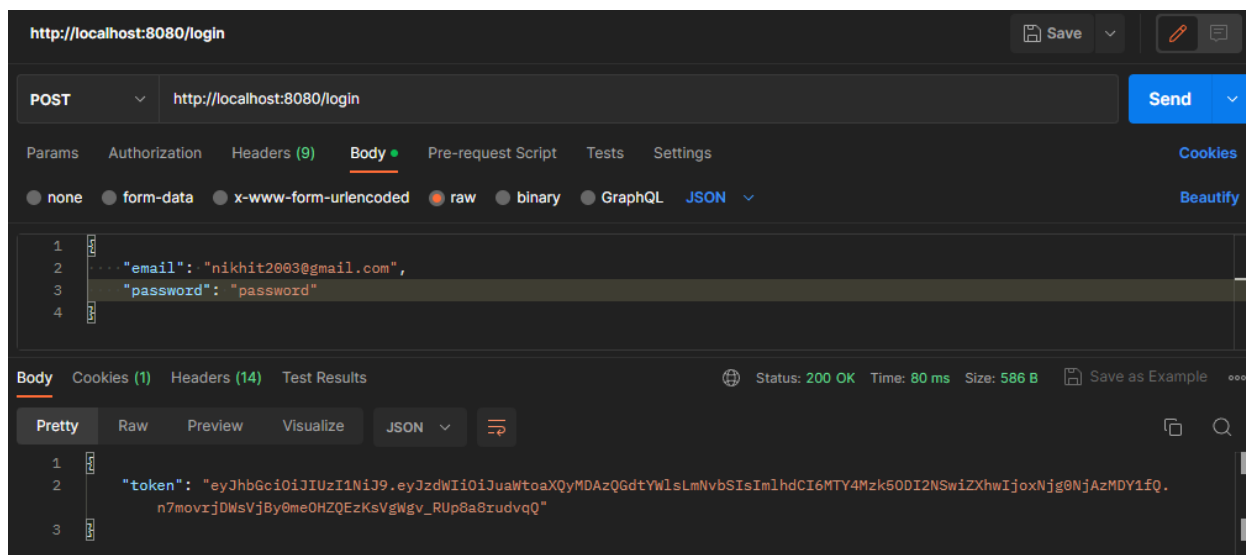


Рисунок 3.2 - Тестирование авторизации часть 1

На рисунке 3.3 изображено тестирование авторизации. Отправляется запрос с почтой и паролем, при этом возвращается статус ошибки 403, так как такого пользователя не существует.

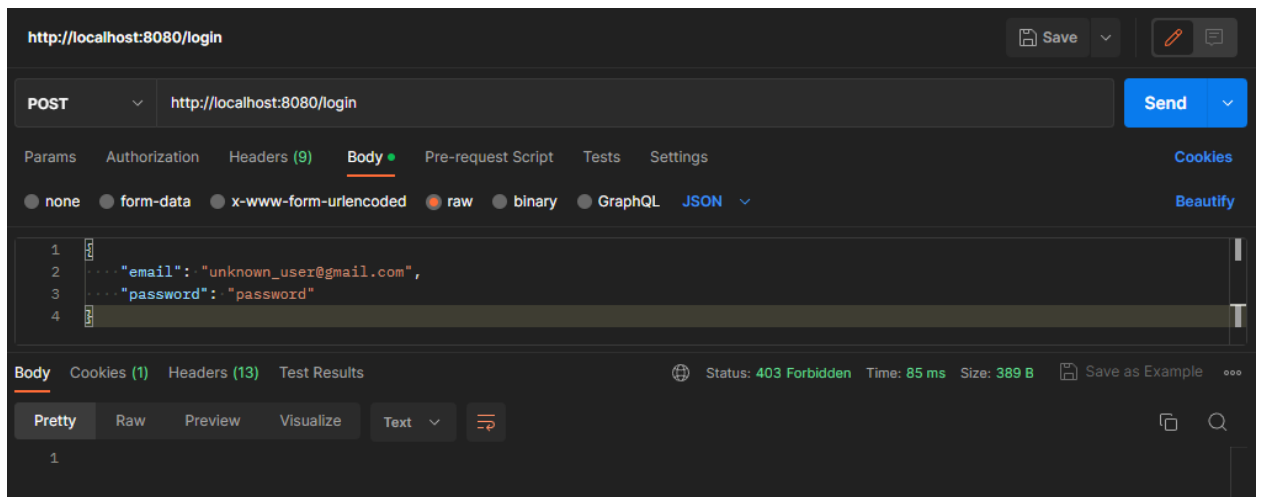


Рисунок 3.3 - Тестирование авторизации часть 2

На рисунке 3.4 изображено отправление запроса получения списка отправлений поездов на указанную дату в указанном направлении. В ответе получен код успешного выполнения запроса 200 и детальная информация об отправлениях.

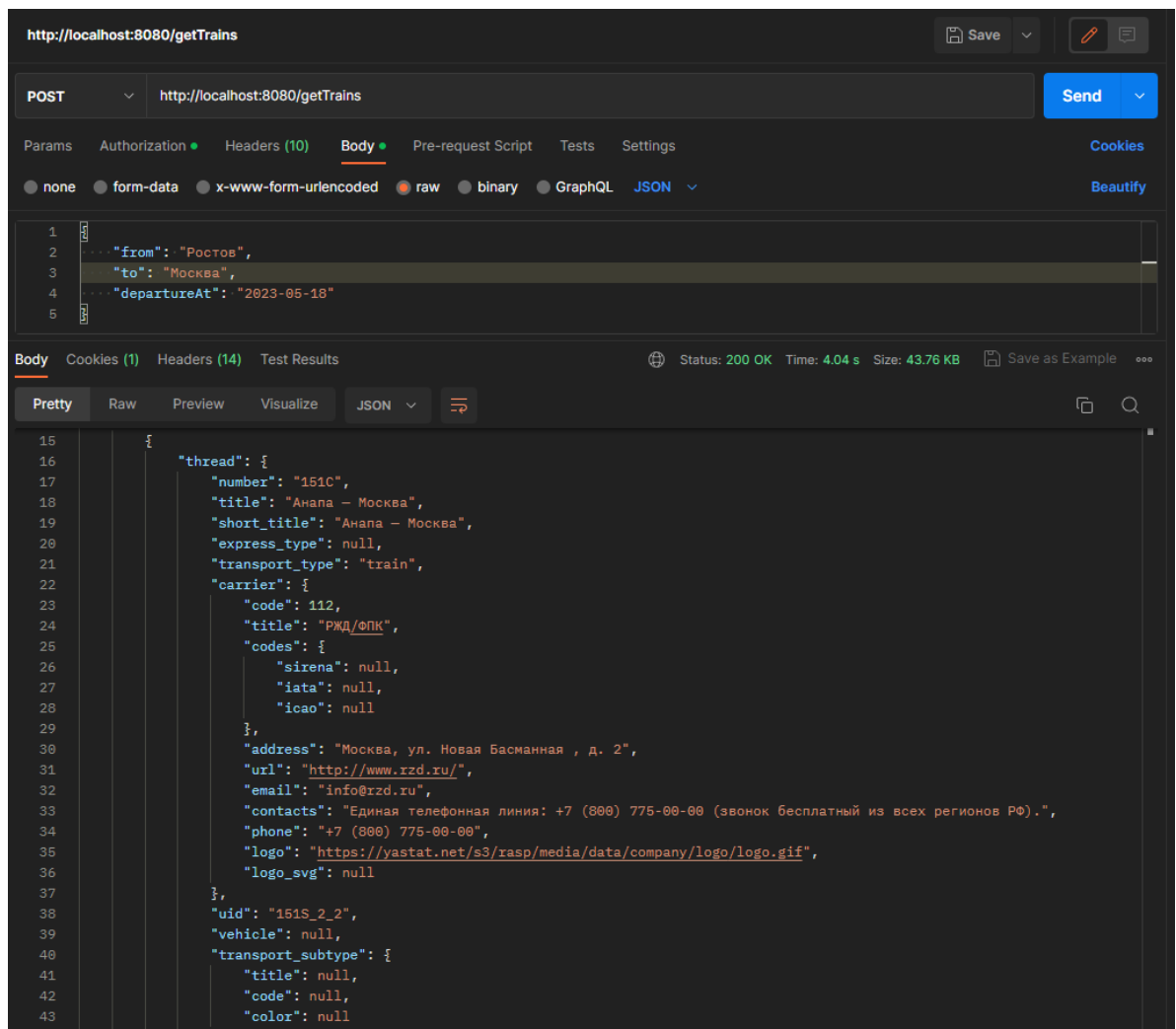


Рисунок 3.4 - Тестирование получения списка отправлений поездов

На рисунке 3.5 изображено отправление запроса на поиск авиабилетов на определенные даты и в определенном направлении. В ответе получен код успешного выполнения запроса 200 и детальная информация об отправлениях.

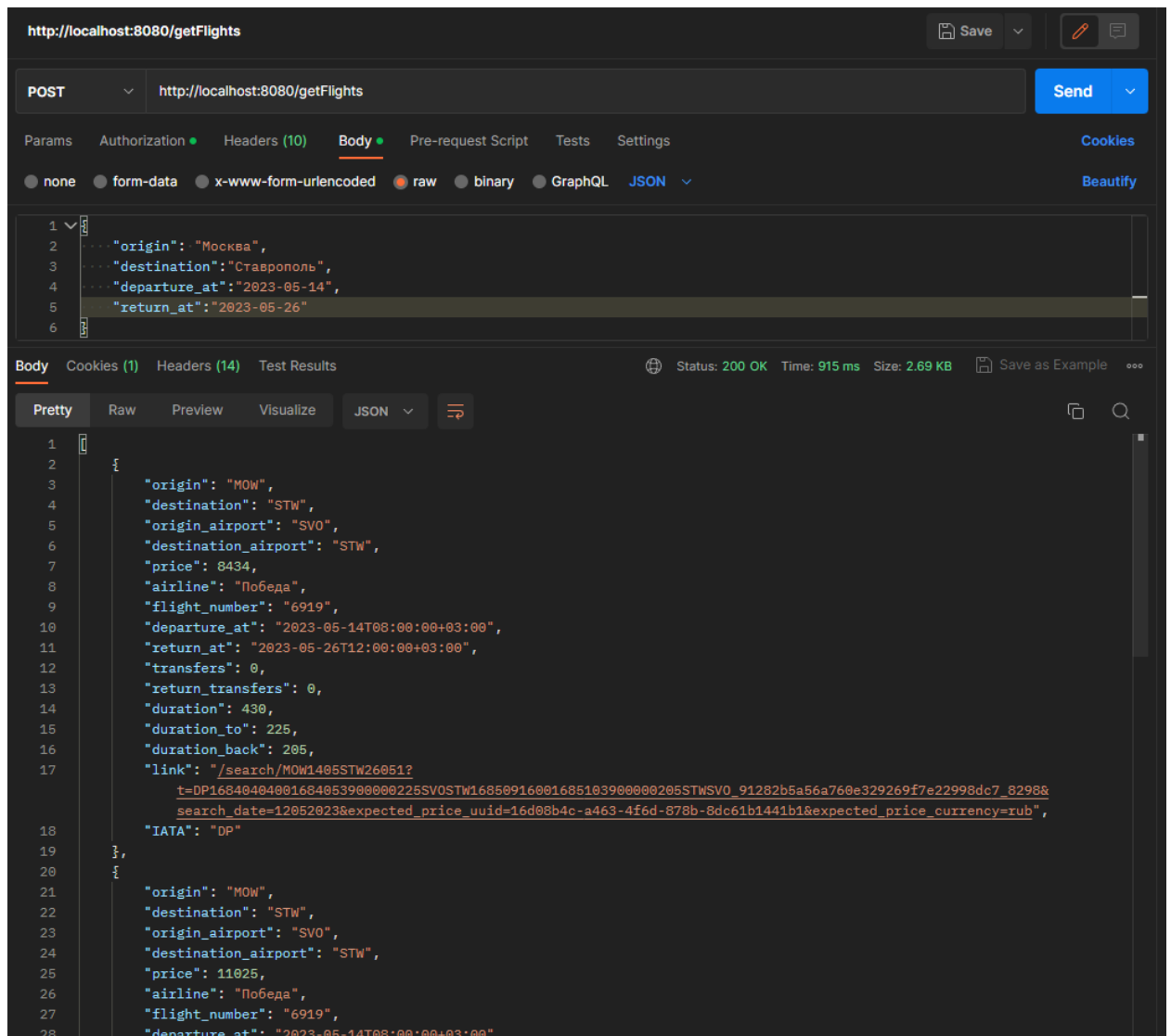


Рисунок 3.5 - Тестирование получения списка авиабилетов

На рисунке 3.6 изображено отправление запроса на получение списка истории поиска авиабилетов для текущего авторизованного пользователя.

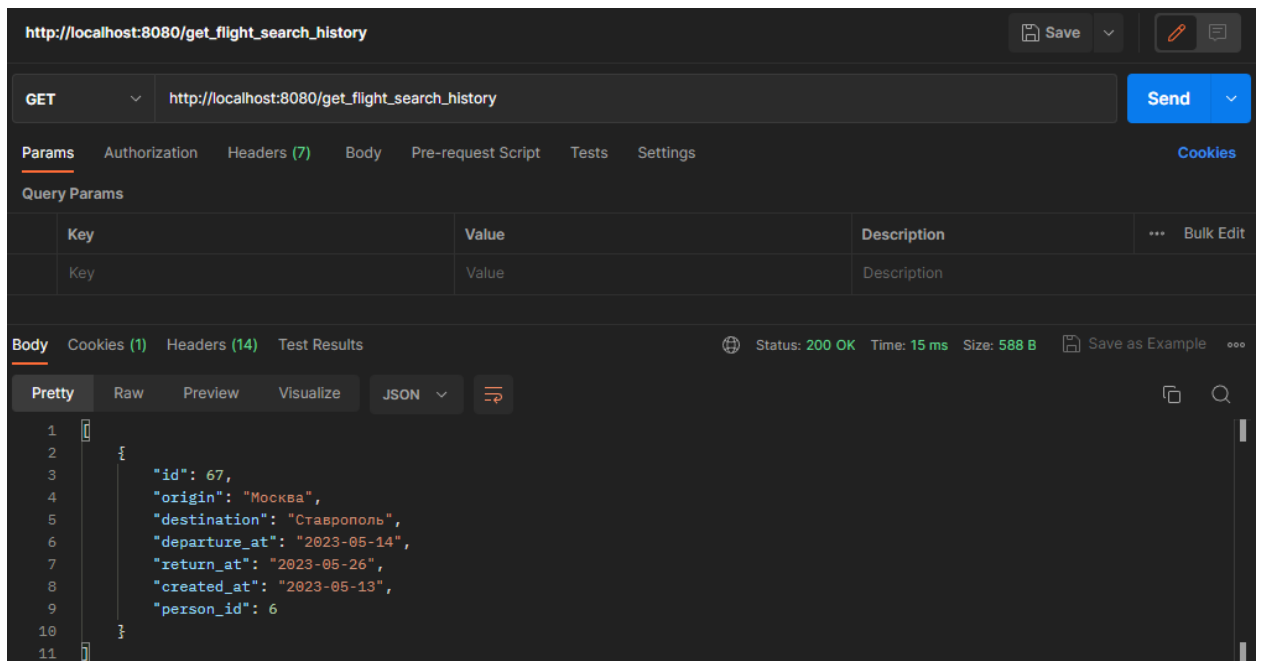


Рисунок 3.6 – Тестирование получения истории

На рисунке 3.7 изображено отправление запроса на получение списка истории поиска отправлений поездов для текущего авторизованного пользователя.

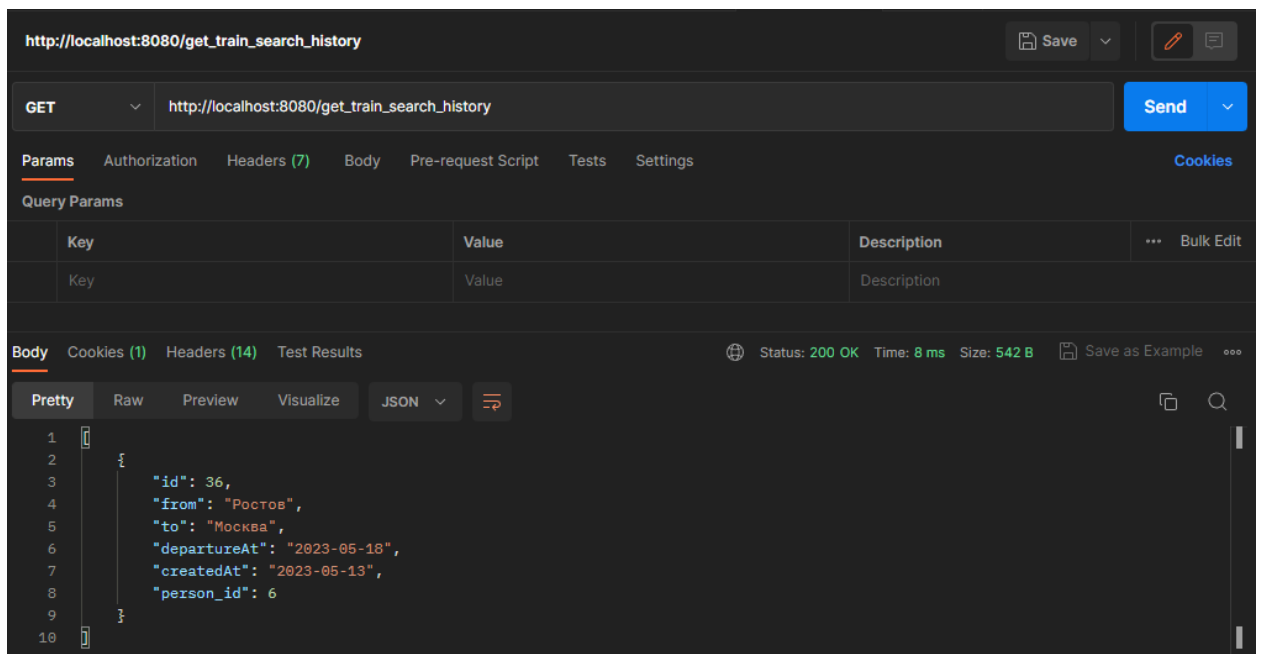


Рисунок 3.7 – Тестирование получения истории

На рисунке 3.8 изображено отправление запроса на получения списка зарегистрированных пользователей.

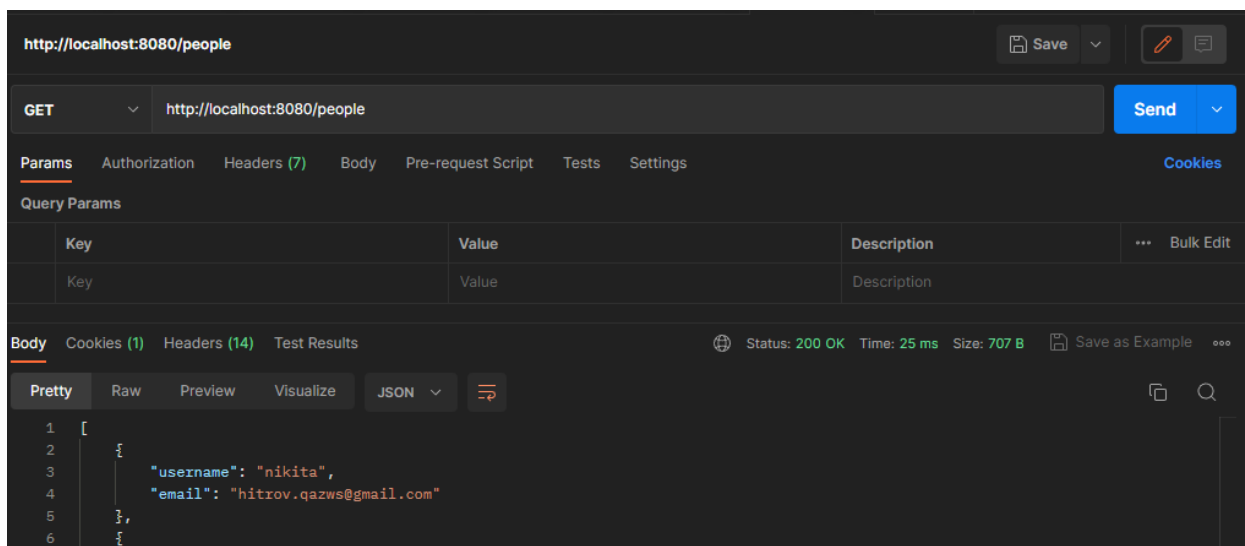


Рисунок 3.8 – Тестирование получения списка пользователей

На рисунке 3.9 изображено получение полной информации о текущем пользователе (пароль закодирован).

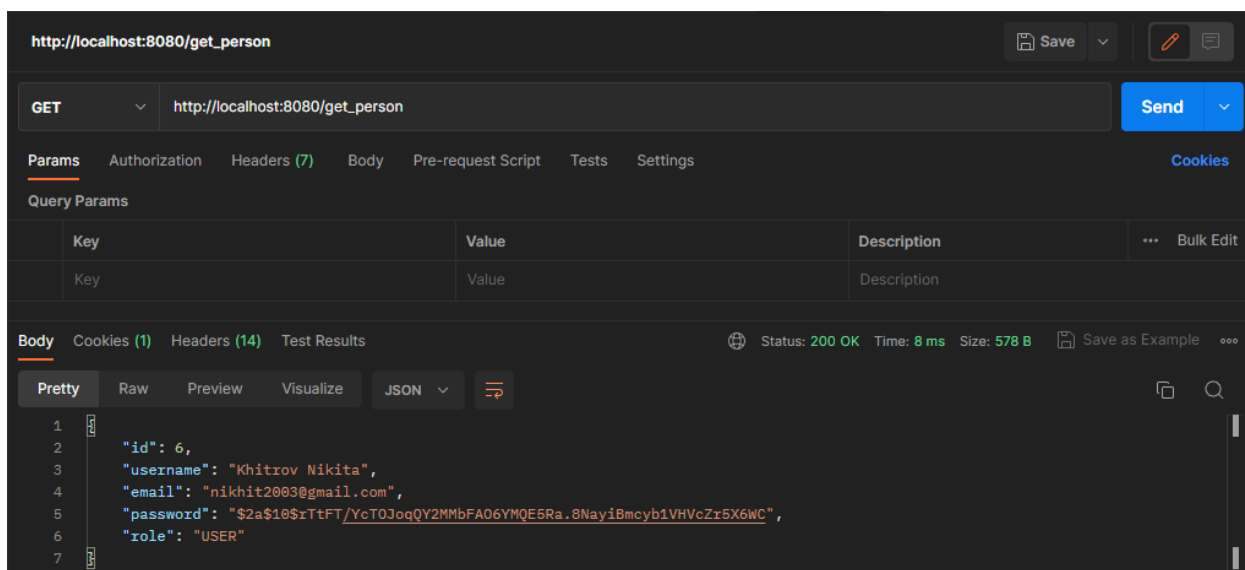
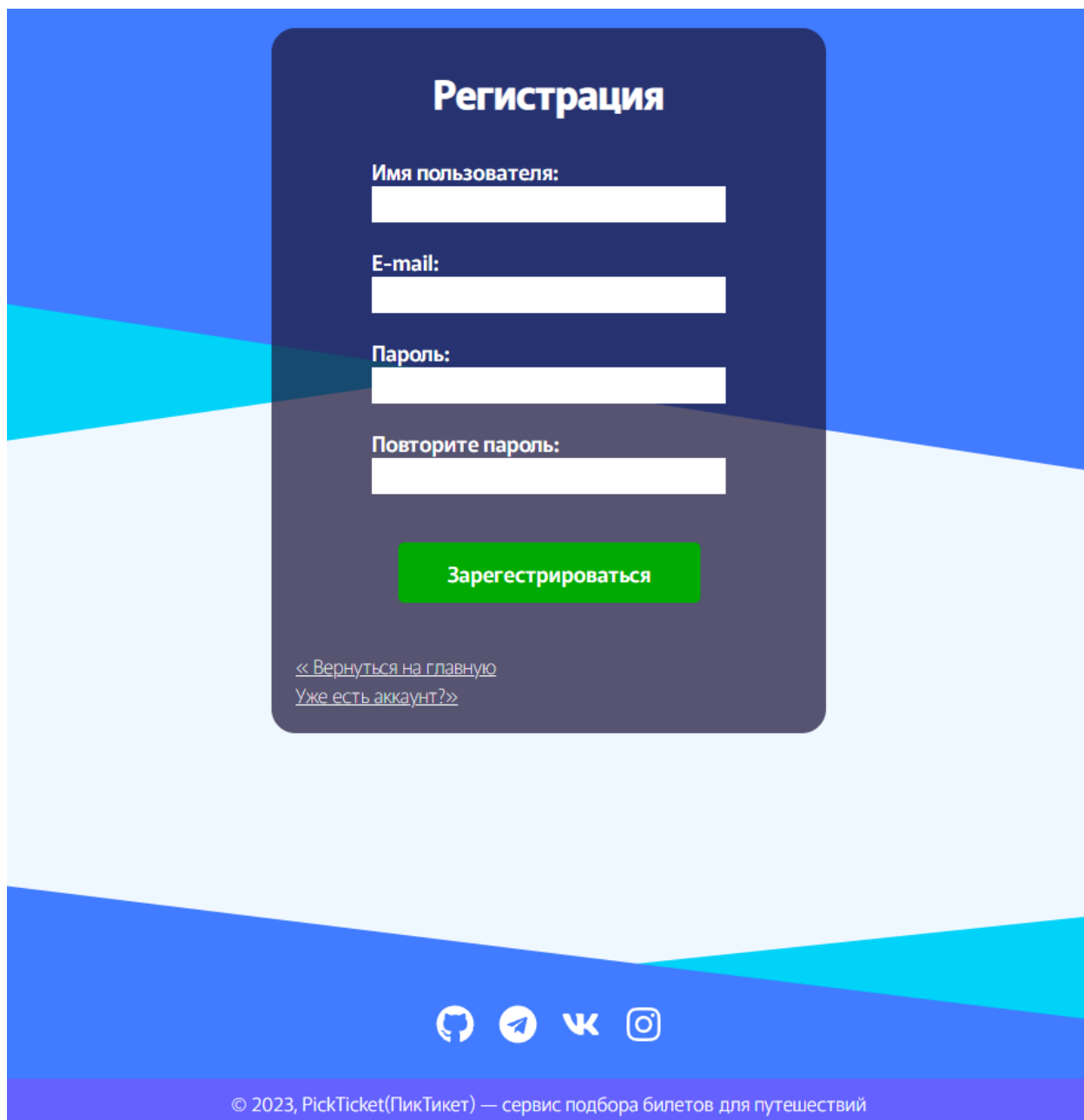


Рисунок 3.9 – Тестирование получения информации о текущем пользователе

На рисунке 3.10 представлено поле регистрации. Вводимые данные сохраняются в состоянии и при правильном вводе отправляются на сервер, при успешной аутентификации пользователь переходит на главную страницу а полученный токен пользователя записывается в куки. При необходимости так же есть ссылки на главную страницу и страницу регистрации внизу формы.



The image shows a registration form titled "Регистрация" (Registration) on a dark blue background. The form is centered and contains the following fields and elements:

- Имя пользователя:** (Username) with a white input field.
- E-mail:** with a white input field.
- Пароль:** (Password) with a white input field.
- Повторите пароль:** (Repeat password) with a white input field.
- Зарегистрироваться** (Register) button in green.
- Links: [« Вернуться на главную](#) and [Уже есть аккаунт?»](#) (Already have an account?).
- Social media icons: GitHub, Telegram, VK, and Instagram.
- Footer: © 2023, PickTicket(ПикТикет) — сервис подбора билетов для путешествий

Рисунок 3.10 – Тестирование регистрации пользователя

На рисунке 3.11 показано получение списка билетов на самолет в определенном направлении в указанные даты. Вводимые в форму данные сохраняются в глобальное состояние и содержащий их запрос отправляется на сервер, получаемый ответ от сервера распределяет данные по найденным маршрутам в карточки маршрутов.

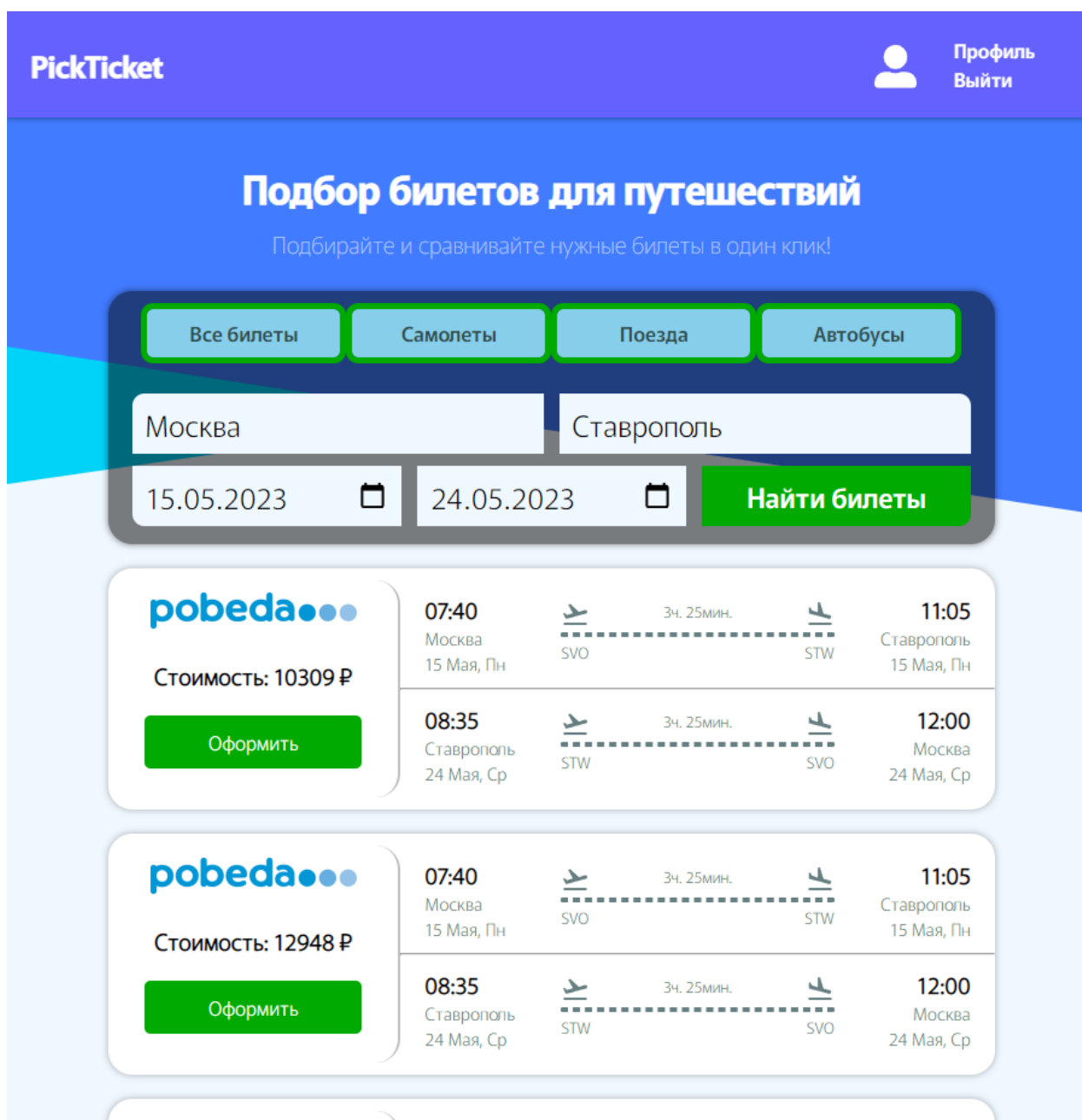


Рисунок 3.11 – Тестирование получения списка авиабилетов

На рисунке 3.12 изображено тестирование личного кабинета реализованного в клиентской части приложения, данная страница предоставляет пользователю возможность обратиться к истории своих поисков.

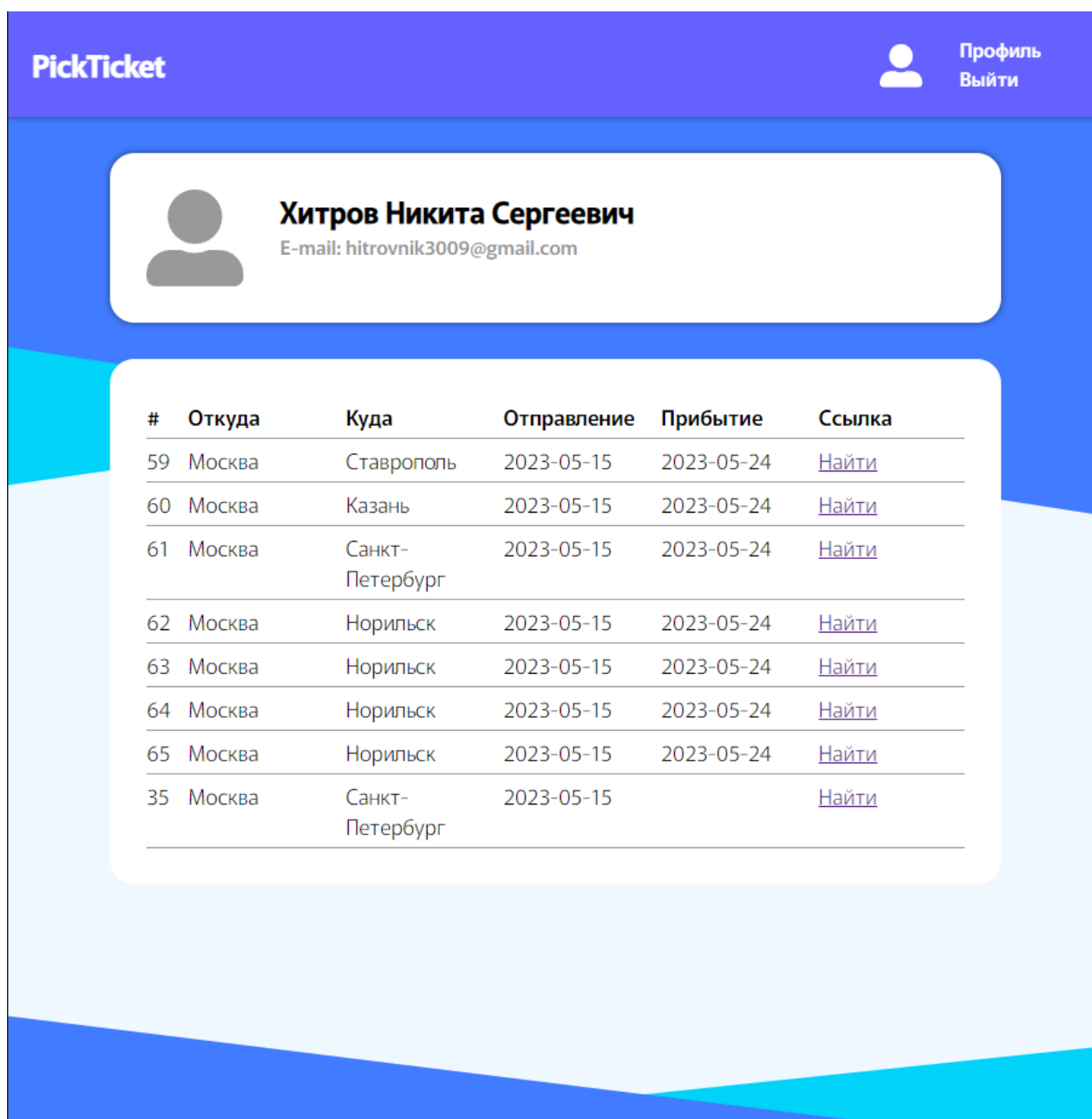


Рисунок 3.12 – Тестирование личного кабинета пользователя

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была освоена компетенция «Шаблоны программных платформ языка Джава» в степени соответствующей рабочей программной дисциплины. Как результат – выполненное приложение на тему «Сервис подбора билетов для путешествий».

В процессе разработки курсовой работы был проведен анализ предметной области, сформированы и реализованы функциональные требования приложения. Было создано 4 таблицы БД, 3 сущности, 5 контроллеров, 4 сервиса, 3 репозитория, 2 DTO-класса и настроены классы Spring Security с технологией JWT токенов. Весь код и, соответственно, все файлы были написаны с помощью таких технологий как JDK, Spring Framework, Spring Security, Spring Boot, Lombok, Postman. Также в ходе создания проекта была полностью изучена среда разработки IntelliJIDEA.

Все сформированные задачи для достижения цели были выполнены. Следовательно, выполнение курсовой работы можно считать успешным.

Ссылка на git-репозиторий с кодом выполненной курсовой работы:
https://github.com/BIGmindede/Course2_2Work

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная документация Spring Boot: <https://spring.io/projects/spring-boot>
2. Книга "Spring Boot in Action" от Craig Walls: <https://www.manning.com/books/spring-boot-in-action>
3. Курс "Building RESTful Web Services with Spring" на сайте Pluralsight: <https://www.pluralsight.com/courses/building-restful-web-services-spring>
4. Статьи на сайте Baeldung, посвященные Spring Boot и RESTful приложениям: <https://www.baeldung.com/>
5. Курс "Spring & Hibernate for Beginners" на сайте Udemy: <https://www.udemy.com/course/spring-hibernate-tutorial/>
6. Видеоуроки на канале YouTube "Java Brains" по теме Spring Boot и RESTful приложений: <https://www.youtube.com/playlist?list=PLqq-6Pq4lTTZSKAFG6aCDVDP86Qx4lNas>
7. Статьи на сайте DZone, посвященные Spring Boot и RESTful приложениям: <https://dzone.com/articles/spring-boot-tutorial-series-1>
8. Курс "RESTful Web Services with Spring Framework - A quick start" на сайте Udemy: <https://www.udemy.com/course/restful-web-services-with-spring-framework-a-quick-start/>
9. Книга "Pro Spring Boot 2" от Felipe Gutierrez: <https://www.apress.com/gp/book/9781484236758>
10. Видеокурсы на сайте Udemy, посвященные Spring Boot и RESTful приложениям: <https://www.udemy.com/topic/spring-boot/>