Geekbrains

Создание веб-приложения для взаимодействия фотографов и клиентов.

Программа: Разработчик

Специализация: Веб-разработка на Java

ФИО: Гаврилов Валентин Сергеевич

Москва

2025

Содержание

[Введение 3](#_Toc198909805)

[Глава 1. Веб-приложение 5](#_Toc198909806)

[1.1 Архитектура веб-приложения 5](#_Toc198909807)

[Глава 2. Анализ существующих решений и постановка задач разработки 16](#_Toc198909808)

[2.1. Обзор существующих платформ для фотографов 16](#_Toc198909809)

[2.2. Выявление потребностей целевой аудитории 16](#_Toc198909810)

[2.3. Постановка задач разработки 17](#_Toc198909811)

[2.4. Технические требования к системе 17](#_Toc198909812)

[2.5. Выбор технологий разработки 18](#_Toc198909813)

[Глава 3. Проектирование веб-сервиса 19](#_Toc198909814)

[Глава 4. Практическая часть 22](#_Toc198909815)

[4.1 Структура сервиса: 22](#_Toc198909816)

[4.1.1 api – основные объекты приложения 22](#_Toc198909817)

[4.1.2 controllers 23](#_Toc198909818)

[4.1.3. Services 25](#_Toc198909819)

[4.1.4 Config 28](#_Toc198909820)

[4.1.5 Jwt 30](#_Toc198909821)

[4.2 Отладка приложения 32](#_Toc198909822)

[4.3 Результат разработки сервиса 33](#_Toc198909823)

[5. Заключение 34](#_Toc198909824)

[6. Список используемой литературы 35](#_Toc198909825)

[7. Приложение 1. 36](#_Toc198909826)

[8. Приложение 2. 52](#_Toc198909827)

## Введение

Современный рынок услуг фотографов требует более удобных и эффективных решений для взаимодействия клиентов и исполнителей. Веб-приложение, разработанное с использованием Java, будет служить платформой для предоставления услуг, автоматизации процессов и повышения удобства как для фотографов, так и для клиентов.   
 Веб-технологии в последнее время стали неотъемлемой частью повседневной жизни. Наблюдается стремительный рост интернет-рынка, что обуславливает необходимость создания современных, функциональных и удобных веб-приложений. Количество пользователей интернета постоянно растет, что создаёт огромные возможности и вызовы для разработчиков. В условиях такой динамичной среды ключевым аспектом успешной веб-разработки является использование эффективных инструментов и технологий.  
Java — один из самых распространенных языков программирования, обладающий множеством преимуществ, таких как кроссплатформенность, высокая производительность и широкие возможности для разработки серверных приложений. Широкая экосистема Java включает в себя фреймворки, такие как Spring и Hibernate, которые значительно упрощают процесс разработки, обеспечивают гибкость и дают возможность легко интегрироваться с другими системами. Эти технологии позволяют создавать современные веб-приложения, которые могут обрабатывать большое количество запросов и выполнять сложные бизнес-логики.  
Цель данной дипломной работы заключается в разработке полноценного веб-приложения с использованием языка Java и соответствующих фреймворков. В рамках выполнения работы будут решены следующие задачи:  
- Проектирование архитектуры веб-приложения. В этом разделе будет определена структура приложения, его основные компоненты и совокупность взаимодействий между ними.  
  
- Реализация функционала. В процессе разработки будут использованы подходы, удобные для создания RESTful API, а также интеграция с базами данных с помощью ORM.

- Документирование проекта. Будет использоваться OpenAPI 3.0 для структур передачи данных и для контроллеров.  
  
 В работе будет рассмотрен вопрос безопасности веб-приложений, который является критически важным в условиях растущих угроз кибербезопасности. Уделим внимание таким аспектам, как аутентификация, авторизация и шифрование данных.  
Таким образом, данная дипломная работа будет полезна как с точки зрения практической реализации, так и с точки зрения углубленного понимания архитектуры и технологий веб-разработки. Ожидается, что результаты исследования смогут служить основой для будущих профессиональных проектов в области веб-разработки.

Весь проект планирую реализовать сам. Клиентскую часть в рамках данного проекта реализовывать не планирую.

## Глава 1. Веб-приложение

### 1.1 Архитектура веб-приложения

Архитектура веб-приложения включает в себя организацию компонентов, их взаимодействие и технологию, используемую для создания системы. Основные слои архитектуры веб-приложения можно разделить на несколько категорий:

1.1.1 Клиентская часть (Frontend)

Frontend — это клиентская часть веб-приложения, которая отвечает за взаимодействие пользователя с приложением через браузер.

Основные технологии  
- HTML (HyperText Markup Language): Язык разметки, который используется для структуры веб-страниц. Определяет различные элементы, такие как заголовки, параграфы, ссылки, изображения и т.д.  
  
- CSS (Cascading Style Sheets): Используется для оформления HTML-документов. Позволяет изменять шрифты, цвета, отступы, размеры и другие визуальные аспекты элементов.  
  
- JavaScript: Язык программирования, который добавляет интерактивность на веб-страницы. Позволяет реагировать на действия пользователя, такие как клики, наведение мыши и отправка форм.

1.1.2. Серверная часть (Backend)

Отвечает за обработку логики приложения и управление данными. Основные аспекты серверной части:  
- Язык программирования: Java, Python, Node.js, Ruby и другие.  
- Фреймворки: Spring Boot для Java, Django для Python, Express.js для Node.js и другие.

Spring Boot — это популярный фреймворк для разработки веб-приложений на Java, который упрощает создание, развертывание и управление приложениями Spring. Он создан для того, чтобы минимизировать количество конфигураций и позволить разработчикам сосредоточиться на бизнес-логике, а не на рутинных задачах настройки.  
Основные особенности Spring Boot  
- Автоконфигурация: Spring Boot использует умные настройки для автоматической конфигурации приложения на основе добавленных зависимостей. Это означает, что если вы импортируете библиотеку, например, для работы с базой данных, Spring Boot автоматически настроит соединение с базой данных.  
  
- Организация проекта: Spring Boot создает стандартную структуру проекта, что позволяет легко организовать код. Он использует принципы модульности, инверсии управления и аспектно-ориентированного программирования (AOP).  
  
- Встроенные серверы: Spring Boot позволяет вам запускать приложение как самостоятельное Java-приложение, включая встроенный сервер, такой как Tomcat или Jetty. Это убирает необходимость установки сервера отдельно.  
  
- Легкость в развертывании: Приложения можно упаковывать в исполняемые JAR-файлы, которые можно развертывать на любом сервере с установленной Java.  
  
- Шаблоны проекта: Spring Boot предоставляет множество начальных шаблонов (Starters), которые позволяют быстро подключать необходимые зависимости и библиотеки.  
  
- Зависимости и управление версиями: Spring Boot использует BOM (Bill of Materials) для управления версиями зависимостей, что упрощает обновление библиотек.  
  
Основные аннотации  
- @SpringBootApplication: Основная аннотация для приложения Spring Boot, которая объединяет три другие аннотации: @Configuration, @EnableAutoConfiguration и @ComponentScan.  
  
- @RestController: Аннотация для определения контроллера, который обрабатывает HTTP-запросы и возвращает данные в формате JSON.  
  
- @RequestMapping: Используется для сопоставления HTTP-запросов с методами контроллера. Можно задавать путь, метод (GET, POST и т.д.) и другие параметры.  
  
- @Autowired: Позволяет автоматически внедрять зависимости в классы, что помогает управлять компонентами приложения.  
  
- @Entity: Обозначает сущность JPA, которая будет отображаться на таблицу в базе данных.  
  
Рбота с базой данных  
Spring Boot поддерживает работу с разными базами данных через Spring Data JPA. Можно легко подключаться к реляционным и NoSQL базам данных. Основные шаги включают:  
- Добавление зависимостей: Включение необходимых библиотеки в файл pom.xml (для Maven) или build.gradle (для Gradle).  
  
- Создание сущностей: Определяются классы, помеченные аннотацией @Entity, которые будут представлять таблицы в базе данных.  
  
- Создание репозиториев: Создаются интерфейсы, которые расширяют JpaRepository или CrudRepository для выполнения операций CRUD.  
  
- Конфигурация подключения: параметры подключения к базе данных настраиваются в [application.properties](https://web.telegram.org/a/application.properties) или application.yml.  
  
Поддержка микросервисов  
Spring Boot часто используется для создания микросервисов. В сочетании с другими проектами Spring (такими как Spring Cloud) он позволяет разрабатывать распределенные системы с помощью подхода микросервисной архитектуры.

- API: RESTful или GraphQL интерфейсы для взаимодействия с клиентом.

1.1.3. База данных

Используется для хранения и управления данными приложения. Может быть:  
- Реляционная: MySQL, PostgreSQL, Oracle и др.  
- NoSQL: MongoDB, Cassandra, Firebase и др.

1.1.4. Архитектурные стили

Наиболее распространенные архитектурные подходы:  
- Монолитная архитектура: Все компоненты приложения объединены в одном коде и развертываются как единое целое.  
- Микросервисная архитектура: Приложение разбивается на небольшие, независимые сервисы, каждый из которых выполняет определенную функцию.

1.1.5. Протоколы и технологии

- HTTP/HTTPS: Протоколы передачи данных в Интернете.  
- WebSocket: Протокол для обмена данными в реальном времени.  
- REST/GraphQL: Стандарты для создания API.

1.1.5.1 RESTful API — это интерфейс, который позволяет взаимодействовать с веб-приложениями и сервисами через стандартные HTTP-запросы. REST (Representational State Transfer) представляет собой архитектурный стиль для работы с распределенными системами, такими как веб-приложения. Он призван упростить создание и поддержку сетевых API.  
Основные принципы REST  
- Статус без сохранения состояния: Взаимодействия между клиентом и сервером не содержат информации о предыдущих запросах. Каждое взаимодействие является независимым, и клиент должен отправлять всю необходимую информацию с каждым запросом.  
- Клиент-серверная архитектура: Клиент и сервер разделены, что позволяет каждому из них развиваться независимо. Клиент отвечает за пользовательский интерфейс, а сервер — за логику обработки данных.  
- Унифицированный интерфейс: REST использует стандартные HTTP-методы (GET, POST, PUT, DELETE) для выполнения операций над ресурсами:  
- GET: Получение данных от сервера.  
- POST: Создание нового ресурса.  
- PUT: Обновление существующего ресурса.  
- DELETE: Удаление ресурса.  
  
- Идентификация ресурсов: Ресурсы (например, пользователи, статьи) идентифицируются с помощью уникальных URI (Uniform Resource Identifier).

- База данных:  
- Реляционные базы данных (например, PostgreSQL или MySQL), что позволит эффективно хранить и управлять данными о пользователях, фотографиях и заказах.

1.1.6. Безопасность

Включает механизмы аутентификации (например, OAuth, JWT) и защиты от атак (например, CSRF, XSS).

1.1.7 Документирование

Документация — это технический паспорт программного продукта, который объясняет, как работает тот или иной элемент кода. Это не просто набор комментариев, а структурированная информация, помогающая другим разработчикам (и вам в будущем) понимать архитектуру и логику программы.

Как работает документирование

В Java существует несколько уровней документации:

* Javadoc — официальный инструмент, встроенный в JDK
* IDE-подсказки — всплывающие подсказки в среде разработки
* Внутренние комментарии — пояснения непосредственно в коде

Базовые элементы документации

Комментарии бывают трёх видов:

// Однострочный комментарий

/\* Многострочный комментарий \*/

/\*\* Документационный комментарий \*/

Структура документационного комментария

/\*\*

\* Краткое описание метода или класса

\*

\* Подробное описание с примерами использования

\*

\* @param параметр1 Описание первого параметра

\* @param параметр2 Описание второго параметра

\* @return что возвращает метод

\* @throws Исключение Описание возможных ошибок

\* @author Автор

\* @version Версия

\*/

Автоматизация процесса

Генерация документации может быть автоматизирована:

* Через командную строку
* С помощью IDE
* Через системы сборки (Maven, Gradle)

Важные моменты

* Консистентность — используйте единый стиль оформления
* Актуальность — документация должна соответствовать коду
* Полезность — информация должна быть применимой
* Доступность — документация должна быть легко найдена

Инструменты для работы

* Javadoc — базовый инструмент для создания HTML-документации
* Dokka — современный инструмент с поддержкой Markdown
* Swagger — для документирования API
* GitHub Wiki — для хранения документации проекта

Рекомендации по внедрению

* Стандарты — создайте и придерживайтесь корпоративного стиля
* Обучение — обучите команду правилам документирования
* Контроль — внедрите проверку документации в процесс разработки
* Автоматизация — используйте инструменты для генерации и проверки

1.1.8 Тестирование

Что такое тестирование ПО

Тестирование ПО — это процесс проверки программного обеспечения на соответствие заданным требованиям и выявление ошибок.

Основные виды тестирования

* Unit-тестирование — проверка отдельных модулей кода
* Интеграционное тестирование — проверка взаимодействия между компонентами
* Системное тестирование — проверка всей системы в целом
* Приемочное тестирование — проверка соответствия требованиям заказчика

Unit-тестирование в Java

Unit-тестирование — это метод проверки отдельных модулей исходного кода на корректность работы. Основные цели:

* Выявление ошибок на ранних стадиях разработки
* Снижение затрат на исправление багов
* Обеспечение надежности кода
* Улучшение дизайна кода

Инструменты тестирования

* JUnit — популярный фреймворк для unit-тестирования
* Mockito — библиотека для мокирования объектов
* TestNG — альтернативный фреймворк для тестирования

Основные аннотации JUnit

* @Test — указывает на тестовый метод
* @Before — метод, выполняемый перед каждым тестом
* @After — метод, выполняемый после каждого теста
* @BeforeClass — метод, выполняемый один раз перед всеми тестами
* @AfterClass — метод, выполняемый один раз после всех тестов

Лучшие практики тестирования

* TDD подход — написание тестов до кода
* Независимость тестов — каждый тест должен быть изолированным
* Мокирование зависимостей — использование Mockito для изоляции
* Проверка граничных случаев — тестирование крайних значений
* Простота тестов — тесты должны быть понятными
* Регулярный запуск — автоматизация запуска тестов
* Использование утверждений — assertEquals, assertTrue, assertFalse
* Документирование — добавление комментариев к тестам

Пример простого теста

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CalculatorTest {

@Test

public void testAddition() {

Calculator calculator = new Calculator();

int result = calculator.add(2, 3);

assertEquals(5, result);

}

}

Рекомендации по написанию тестов

* Покрытие кода — стремитесь к высокому проценту покрытия
* Параметризованные тесты — используйте для тестирования разных данных
* Валидация ошибок — проверяйте обработку исключительных ситуаций
* Тестирование производительности — проверяйте время выполнения
* Документация тестов — добавляйте описание проверяемых сценариев

Автоматизация тестирования

* CI/CD системы — Jenkins, GitHub Actions
* Контейнеризация — использование Docker для изоляции окружения
* Отчетность — автоматизированная генерация отчетов по тестам

Преимущества тестирования

* Качество кода — повышение надежности приложения
* Сокращение затрат — раннее выявление ошибок
* Документация — тесты как живая документация
* Поддержка — упрощение внесения изменений
* Доверие — повышение доверия к продукту

Типичные ошибки при тестировании

* Недостаточное покрытие — отсутствие тестов для критических участков
* Сложные тесты — трудности в поддержке
* Зависимые тесты — влияние одного теста на другой
* Редкий запуск — отсутствие автоматизации
* Формальное тестирование — написание тестов ради покрытия

Качественное тестирование — это важная часть разработки, которая помогает создавать надежное и стабильное программное обеспечение.

## Глава 2. Анализ существующих решений и постановка задач разработки

### 2.1. Обзор существующих платформ для фотографов

**Конкурентный анализ** показывает, что на рынке представлены следующие типы платформ:

Универсальные площадки для фрилансеров

Специализированные фотосервисы

Нишевые платформы для конкретных видов фотосъемок

Личные сайты фотографов

**Основные преимущества** существующих решений:

Широкий функционал

Большая база пользователей

Налаженная система платежей

Развитая система отзывов

**Недостатки** текущих платформ:

Сложность использования

Высокая комиссия

Ограниченный функционал для фотографов

Проблемы с безопасностью данных

### 2.2. Выявление потребностей целевой аудитории

**Целевая аудитория** включает:

Профессиональные фотографы

Начинающие специалисты

Клиенты, ищущие фотоуслуги

Организаторы мероприятий

**Основные потребности** пользователей:

Удобный интерфейс

Быстрый поиск специалистов

Надежная система оплаты

Прозрачная система рейтингов

Качественное портфолио

### 2.3. Постановка задач разработки

**Основные задачи** проекта:

Создание удобного интерфейса для фотографов

Разработка системы поиска специалистов

Внедрение системы обратной связи

Реализация системы оплаты

Обеспечение безопасности данных

### 2.4. Технические требования к системе

**Функциональные требования**:

Регистрация и аутентификация пользователей

Управление профилем

Создание и редактирование портфолио

Система поиска

Механизм обратной связи

Система оплаты

### 2.5. Выбор технологий разработки

**Технологическая stack**:

Backend: Spring Boot

База данных: PostgreSQL

Система контроля версий: Git

Инструменты разработки: Docker

## Глава 3. Проектирование веб-сервиса

При проектировании веб-приложения важно учитывать архитектуру клиент-сервер. В нашем случае:  
- Клиентская часть: в рамках дипломной работы не будет проводиться разработка интерфейса пользователя, функционал будет заключаться в ответах на запросы к серверной части.  
- Серверная часть:  
- Реализация на Java с использованием Spring Boot для создания RESTful API.  
- Важно использовать MVC (Model-View-Controller) паттерн для разделения логики приложения.

Основные функции приложения  
Приложение должно включать следующие ключевые функции:  
- Регистрация и аутентификация: Удобный процесс создания учетных записей для клиентов и фотографов с использованием OAuth2 или JWT.

- Профили пользователей: Возможность фотографов загружать свой портфолио, устанавливать расценки и описания услуг, вести свой календарь фотосессий, доступный всем пользователям, для планирования даты съемок(при заказе фотосессии).

- Поиск и фильтрация: Функциональность, позволяющая клиентам находить фотографов по различным критериям (стиль, цена, местоположение, дата съемки).  
  
- Создание заказов: Клиенты могут оформлять заказы с указанием всех деталей, а фотографы – подтверждать или отклонять их.  
  
- Отзывы и рейтинги: Система обратной связи, позволяющая клиентам оставлять отзывы о работе фотографов, тем самым формируя их репутацию.  
  
Безопасность веб-приложения  
Безопасность является критически важной частью любого веб-приложения. Вам следует учесть:  
- Аутентификация: Реализация безопасной аутентификации с помощью JWT и шифрование паролей (например, с использованием bcrypt).  
  
- Шифрование данных: Обеспечение передачи данных по HTTPS для защиты информации пользователей.  
  
- Защита от атак: Введение дополнительных мер для защиты от SQL-инъекций, XSS и CSRF атак.

## Глава 4. Практическая часть

### 4.1 Структура сервиса:

Все объекты проекта разбиты на логические части(package):

**api** – основные объекты проекта

**controller** – контроллеры для взаимодействия с пользователем

**service** – реализация бизнес-логики

**repository** – механизмы сохранения данных

**config** – настройки Security и Web socket

**dto** – структуры данных для передачи информации между клиентом и сервером

**jwt** – механизмы работы аутентификации

### 4.1.1 api – основные объекты приложения

- Album – группировка файлов пользователей, принадлежит определенному пользователю.

- Calendar – группировка дат(отрезков времени), принадлежит определенному пользователю.

- CalendarUnit – дата календаря(отрезок времени), принадлежит определенному календарю(который в свою очередь привязан к определенному пользователю), может быть «занят» другим пользователем(клиентом)

- Chat – чаты пользователей, могут принадлежать нескольким пользователям, группируют сообщения чатов.

- Message – сообщение чата, принадлежит определенному пользователю и определенному чату, непосредственно сообщение.

- Photosession – фотосессия, принадлежит двум пользователям – фотографу и клиенту. Группирует комментарии, группирует файлы фотографий.

- Comment – комментарии, принадлежат Албомам, Файлам, Фотосессиям

- Role – роли пользователей

- User – пользователь – личная информация, логин, пароль, почта для регистрации, основной объект.   
 - связан с Role многий ко многим,

- связан с Photosession один ко многим, как фотограф и один ко многим как клиент,

- связан с Calendar один к одному

- FileEntity – файл, описывает кто владелец, в какой альбом входит файл, к какой фотосессии относится, в каких комментах упомянут.

### 4.1.2 controllers

Контроллеры для реализации взаимодействия с пользователем.

- AuthController

/api/auth/login – post запрос, получение авторизации для пользователя передается пара логин/пароль, при успешной аутентификации выдается токен для работы с API без дальнейшего ввода данных для входа

/api/auth/refresh – post запрос, для обновления пары токенов (токен доступа, токен для обновления), в случае когда время действия токена доступа истекает.

- ChatController

/api/chats – создание нового чата, передается коллекция id пользователей, которые будут включены в чат при создании. Возвращается информация о вновь созданном чате.

/api/chats/{chatId}/users/{userId} – добавление еще одного пользователя в чат. Передается id чата и id добавляемого пользователя.

/api/chats /{chatId}/messages – передача сообщения в чат. Передается id чата, id пользователя передающего сообщение, содержимое сообщения.

/api/chats/{chatId}/messages – получение всех сообщений чата. Передается id чата, в ответ коллекция сообщений чата.

/api/chats/{chatId}/messages/latest – получение нескольких последних сообщений из чата. Передается id чата, и, опционально, количество сообщений которые необходимо получить. В ответ коллекция сообщений.

- FileController

/files – post запрос, в теле запроса передается набор двоичных данных файла, параметром передается id пользователя владельца. Происходит валидация файла – непустой, определенного типа(изображение).

/files/{id} – get запрос, получение файла по id.

/files – get запрос – получение списка всех доступных файлов, со всеми их свойствами.

- UserController

/api/users/register – post запрос, создание нового пользователя, в теле запроса json структура с параметрами

/api/users/{id} – get запрос, получение данных пользователя по id.

/api/users – get запрос, получение списка с данными всех пользователей.

/api/users/{id} – put запрос, обновление данных пользователя

/api/users/{id} – delete запрос, удаление пользователя по id.

### 4.1.3. Services

Общая архитектура

Все сервисы взаимодействуют с соответствующими репозиториями для работы с данными. Архитектура построена по принципу разделения ответственности (SoA), где каждый сервис отвечает за свою область функциональности.

Основные принципы реализации:

* Использование Spring Framework
* Внедрение зависимостей
* Обработка исключений
* Валидация данных
* Безопасность

Каждый сервис имеет четкую структуру и отвечает за определенный набор операций, что обеспечивает модульность и удобство поддержки приложения.

- AuthService (Сервис аутентификации)

Назначение сервиса:  
Сервис отвечает за аутентификацию пользователей, генерацию JWT-токенов и управление доступом.

Основные функции:

* Авторизация пользователя
* Генерация access и refresh токенов
* Обновление токенов
* Валидация токенов
* Хранение refresh токенов в памяти

Ключевые методы:

* login() - авторизация пользователя
* getAccessToken() - получение access токена
* refresh() - обновление токенов
* getAuthInfo() - получение информации об аутентификации

- ChatService (Сервис чата)

Назначение сервиса:  
Управление чатами, сообщениями и участниками.

Основные функции:

* Создание чатов
* Управление участниками
* Отправка сообщений
* Получение сообщений

Ключевые методы:

* createChat() - создание чата
* addUserToChat() - добавление пользователя в чат
* sendMessage() - отправка сообщения
* getAllMessages() - получение всех сообщений
* getLatestMessages() - получение последних сообщений

- FileService (Сервис файлов)

Назначение сервиса:  
Работа с файлами, их загрузка и хранение.

Основные функции:

* Загрузка файлов
* Хранение файлов
* Получение файлов
* Управление файлами

Ключевые методы:

* saveFile() - сохранение файла
* loadFileAsResource() - загрузка файла
* getAllFileInfo() - получение информации о файлах

- UserDetailsService (Сервис пользовательских данных)

Назначение сервиса:  
Предоставляет информацию о пользователе для Spring Security.

Основные функции:

* Загрузка пользователя по имени
* Формирование UserDetails объекта
* Проверка прав доступа

Ключевые методы:

* loadUserByUsername() - загрузка пользователя

- UserService (Сервис пользователей)

Назначение сервиса:  
Управление пользователями и их данными.

Основные функции:

* Регистрация пользователей
* Получение информации о пользователях
* Обновление данных
* Удаление пользователей

Ключевые методы:

* createUser() - создание пользователя
* registerUser() - регистрация
* getUserById() - получение пользователя по ID
* updateUser() - обновление данных
* deleteUser() - удаление пользователя

### 4.1.4 Config

Модуль SecurityConfig

Назначение модуля:  
Модуль конфигурации безопасности отвечает за настройку системы аутентификации и авторизации в приложении. Он обеспечивает защиту ресурсов и управление доступом пользователей.

Основные компоненты:

* SecurityFilterChain - цепочка фильтров безопасности
* JwtFilter - фильтр для работы с JWT-токенами
* UserDetailsService - сервис для работы с пользовательскими данными
* DaoAuthenticationProvider - провайдер аутентификации

Функциональные возможности:

* Настройка политики сессий (STATELESS)
* Настройка точек входа для аутентификации
* Управление доступом к ресурсам
* Обработка исключений аутентификации
* Настройка фильтров безопасности

Ключевые особенности:

* Отключение CSRF-защиты
* Настройка анонимного доступа к определенным эндпоинтам
* Поддержка JWT-аутентификации
* Настройка обработки ошибок аутентификации
* Управление правами доступа к ресурсам

Защищаемые эндпоинты:

* /api/auth/ - открытый доступ
* /api/users/register - открытый доступ
* /v3/ - открытый доступ
* /swagger-ui/ - открытый доступ
* Все остальные запросы требуют аутентификации

### 4.1.5 Jwt

- Модуль JwtAuthentication

Назначение модуля:  
Модуль реализует пользовательский объект аутентификации на основе JWT-токенов. Он предоставляет расширенную информацию о пользователе и его ролях для системы безопасности.

Основные компоненты:

* JwtAuthentication - класс, реализующий интерфейс Authentication
* Lombok аннотации (@Getter, @Setter) - для автоматической генерации геттеров и сеттеров
* Spring Security - интеграция с системой безопасности

Функциональные возможности:

* Хранение информации о пользователе (имя, роли)
* Управление статусом аутентификации
* Получение списка полномочий
* Получение основной информации о пользователе

Ключевые атрибуты:

* authenticated - флаг аутентификации
* username - логин пользователя
* firstName - имя пользователя
* roles - набор ролей пользователя

- Модуль JwtFilter

Назначение модуля:  
Модуль представляет собой фильтр для обработки JWT-токенов в HTTP-запросах. Он отвечает за валидацию токенов и установку информации о пользователе в SecurityContext.

Основные компоненты:

* JwtFilter - реализация GenericFilterBean
* JwtProvider - сервис для работы с JWT
* SecurityContextHolder - управление контекстом безопасности

Функциональные возможности:

* Извлечение токена из HTTP-заголовка
* Валидация JWT-токена
* Парсинг информации из токена
* Установка аутентификации в SecurityContext

Ключевые методы:

* doFilter - основной метод фильтрации запросов
* getTokenFromRequest - метод извлечения токена из запроса
* validateAccessToken - валидация токена доступа

- Модуль JwtProvider

Назначение модуля:  
Модуль отвечает за генерацию и валидацию JWT-токенов. Он обеспечивает криптографическую защиту токенов и управление их сроком действия.

Основные компоненты:

* JwtProvider - основной класс для работы с JWT
* SecretKey - секретные ключи для подписи токенов
* Claims - структура для хранения данных в токене

Функциональные возможности:

* Генерация токенов доступа и обновления
* Валидация токенов
* Парсинг информации из токенов
* Управление сроком действия токенов

Ключевые методы:

* generateAccessToken - создание токена доступа
* generateRefreshToken - создание токена обновления
* validateAccessToken - валидация токена доступа
* getAccessClaims - получение данных из токена

- Модуль JwtUtils

Назначение модуля:  
Модуль содержит служебные методы для работы с JWT-токенами. Он обеспечивает преобразование данных из токенов в объекты приложения.

Основные компоненты:

* JwtUtils - утилитный класс
* Claims - данные из JWT-токена
* Role - перечисление ролей пользователя

Функциональные возможности:

* Генерация объекта аутентификации из токена
* Преобразование ролей из строкового представления
* Обработка данных, полученных из токена

Ключевые методы:

* generate - создание объекта аутентификации
* getRoles - преобразование ролей из токена

Взаимодействие модулей

* JwtFilter использует JwtProvider для валидации токенов
* JwtProvider создает токены, которые обрабатываются JwtFilter
* JwtUtils помогает преобразовать данные из токенов в объекты JwtAuthentication

### 4.2 Отладка приложения

В ходе разработки были выполнены тесты функциональности каждого ендпонинта.

Использовалось приложение Postman/

**4.2.1. Что такое Postman**

**Postman** — это универсальная платформа для работы с API, которая объединяет инструменты разработки, тестирования и документирования интерфейсов. Изначально созданное как простое расширение для браузера, сегодня это полноценная экосистема, поддерживающая современные протоколы: REST, SOAP, GraphQL и WebSockets.

**Основные преимущества**

* **Комплексный подход** к работе с API в единой среде
* **Автоматизация тестирования** через встроенный JavaScript-движок
* **Автоматическое создание документации** на основе коллекций

Результаты тестовых запросов представлены в виде скриншотов в приложении 1.

### 4.3 Результат разработки сервиса

Файлы проекта находятся на сайте GitHub(см. приложение 2).

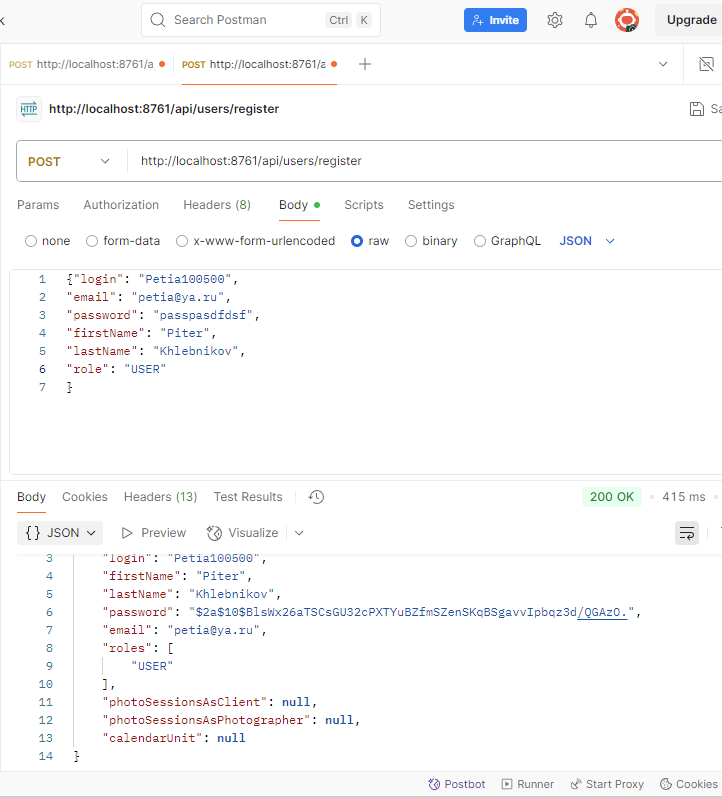
## 5. Заключение

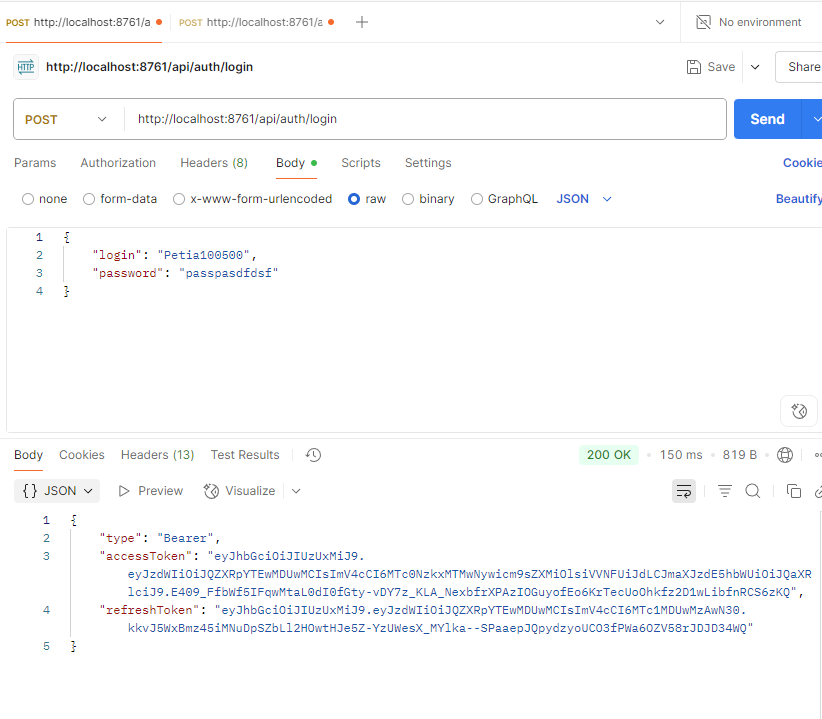
Разработка веб-приложения для взаимодействия фотографов и клиентов на Java имеет значительный потенциал для улучшения качества обслуживаемых клиентов и расширения возможностей фотографов. При разработке приложения были затронуты основные аспекты разработки на JAVA, использован популярный фреймворк Spring, написаны тесты, обеспечена безопасность пользователей приложения, реализовано хранение информации в базе данных, данные, вводимые пользователем, проходят валидацию.   
 Так же в ходе работы было обнаружено, что первоначальное ТЗ, при разработке серверной части может претерпевать значительные изменения в связи с особенностями технологий разработки. Так же при составлении ТЗ необходим разработчик со стороны фронтэнда, потому что без схемы работы пользователя сложно проектировать взаимодействие между сущностями приложения. В ходе разработки вышеперечисленные факторы привели к тому, что была реализована базовая функциональность приложения. Взаимодействие между объектами в рамках идеи веб-сервиса для фотографов и клиентов требует первоначальной проработки frontend составляющей проекта, для определения необходимых endpoint и архитектуры взаимодействия между ними.

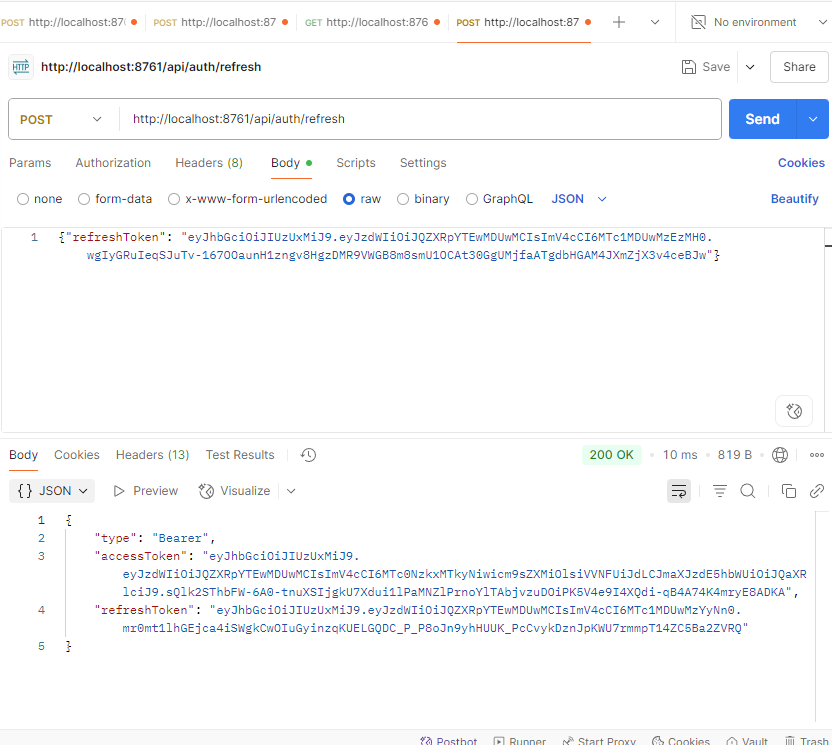
## 6. Список используемой литературы

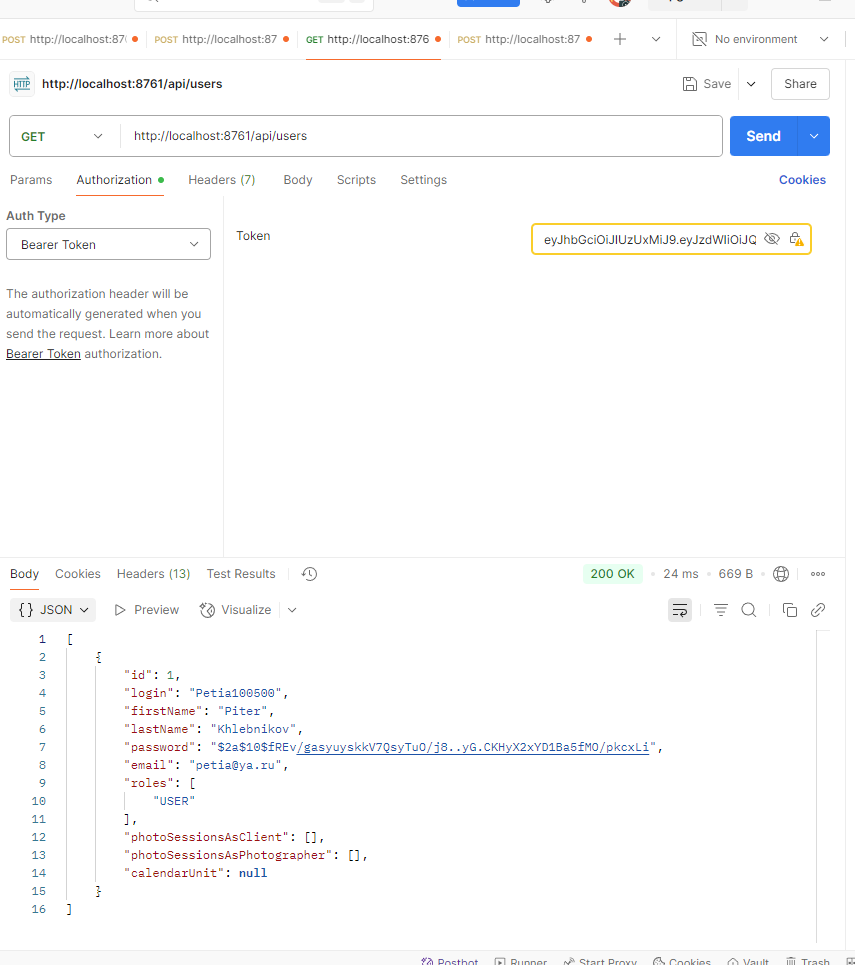
1. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования.
2. **Блох Дж.** Эффективная Java.
3. **Фаулер М.** Архитектура корпоративных программных приложений.
4. **Кент Б.** Паттерны объектно-ориентированного проектирования.

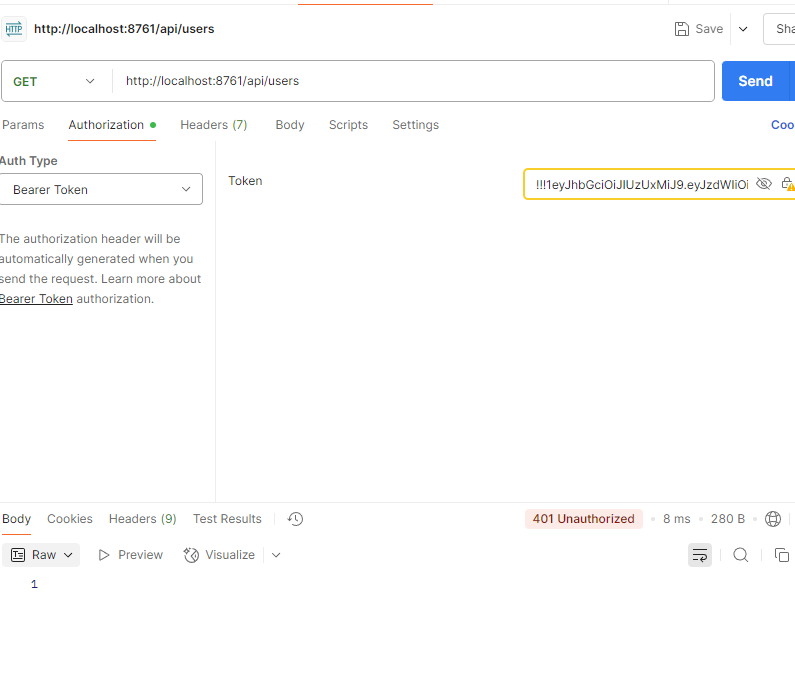
## 7. Приложение 1.

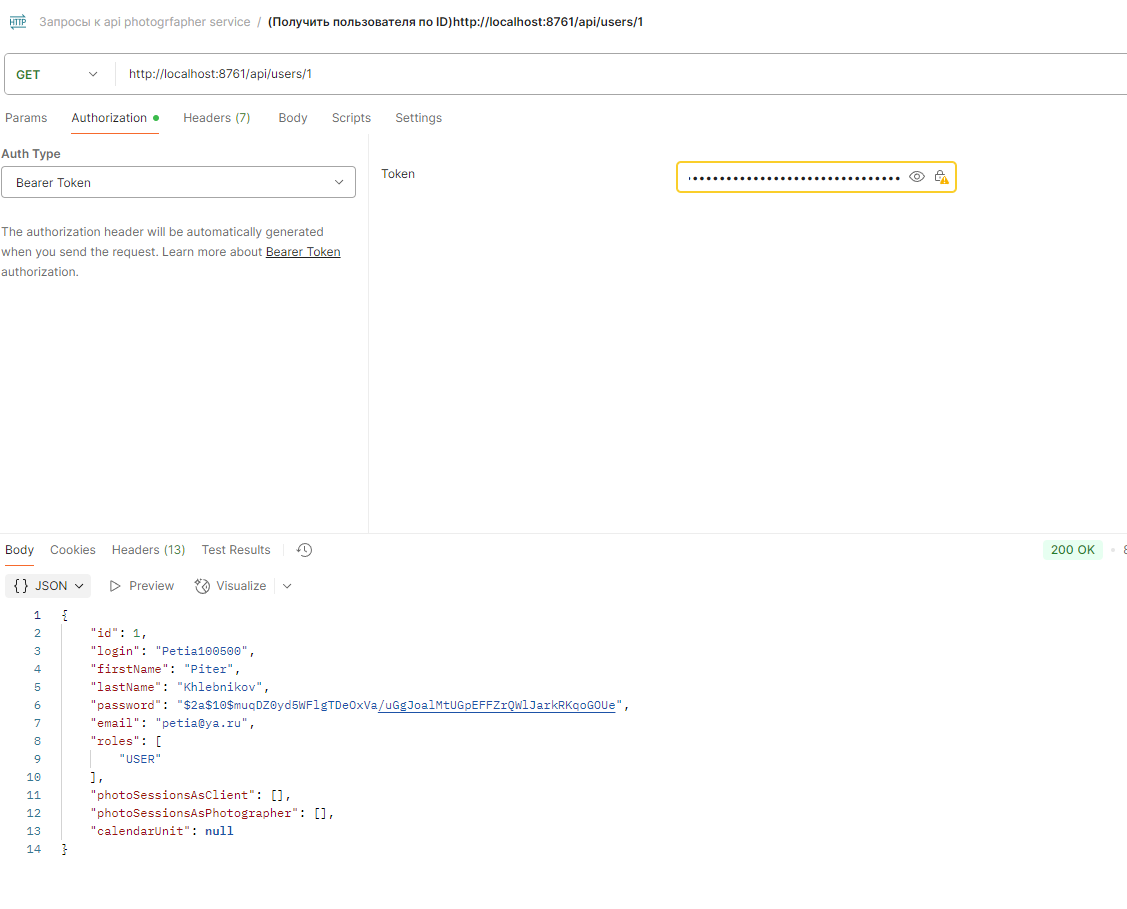


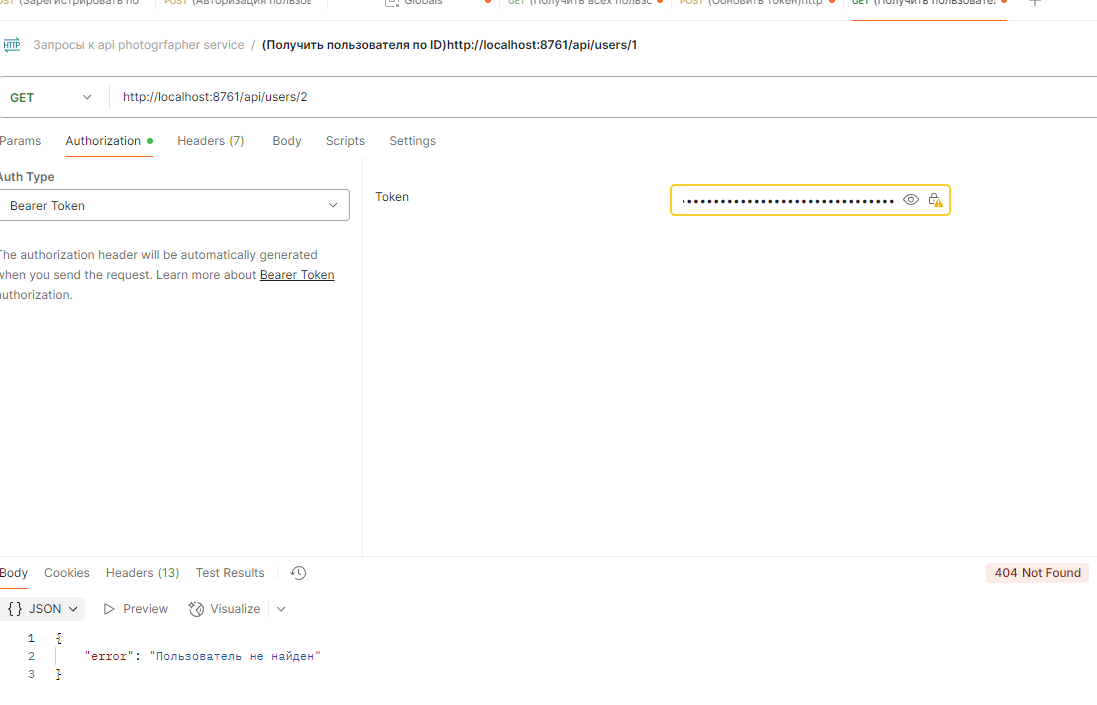


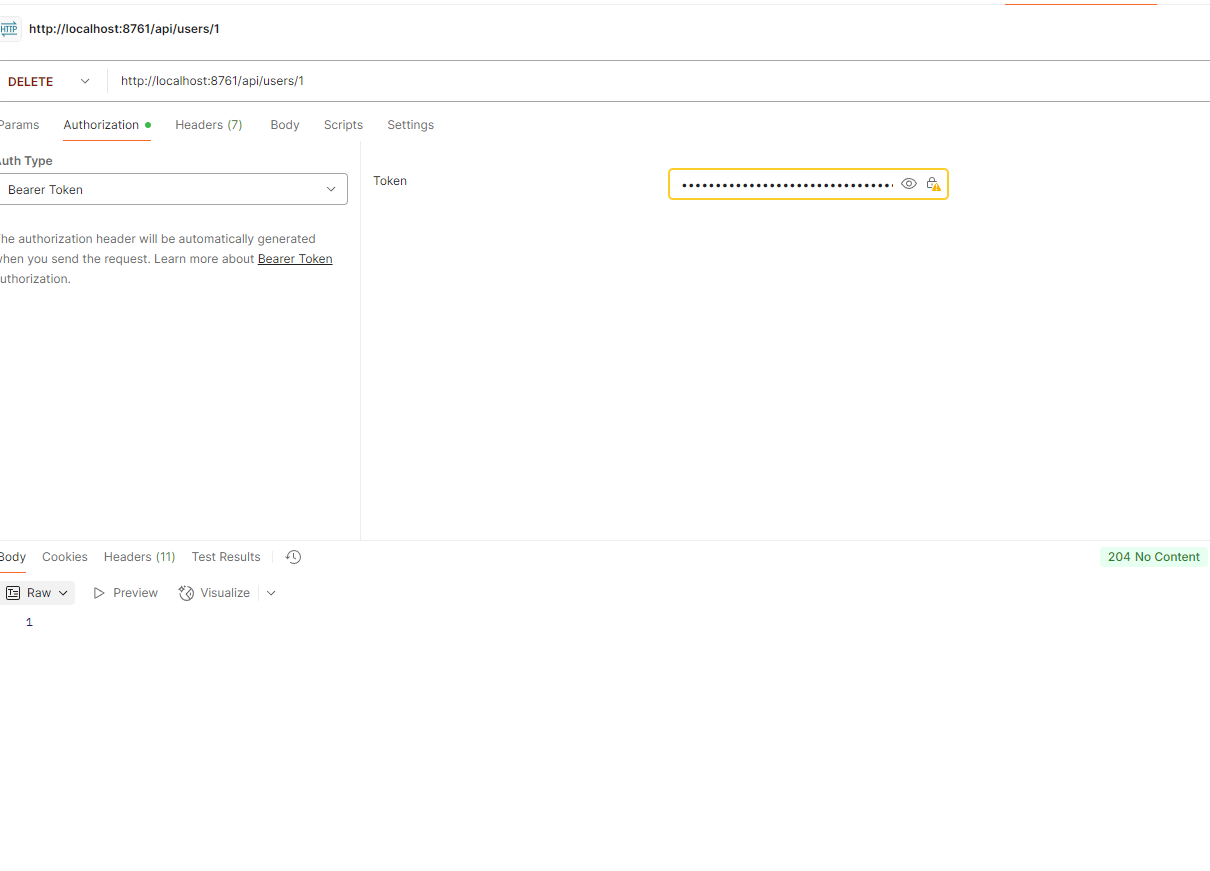


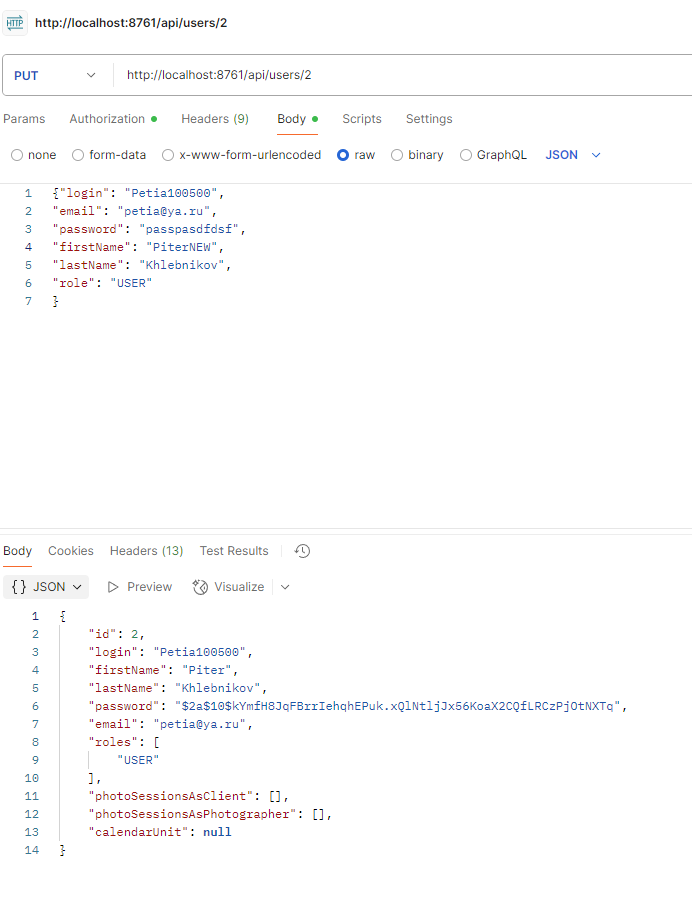


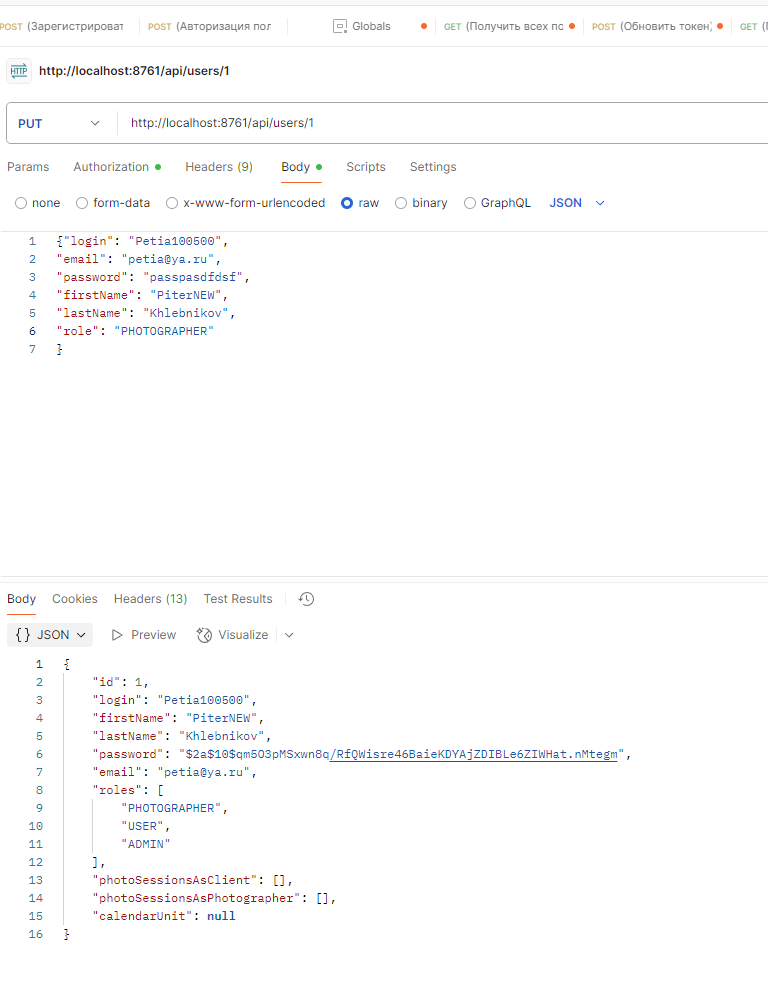


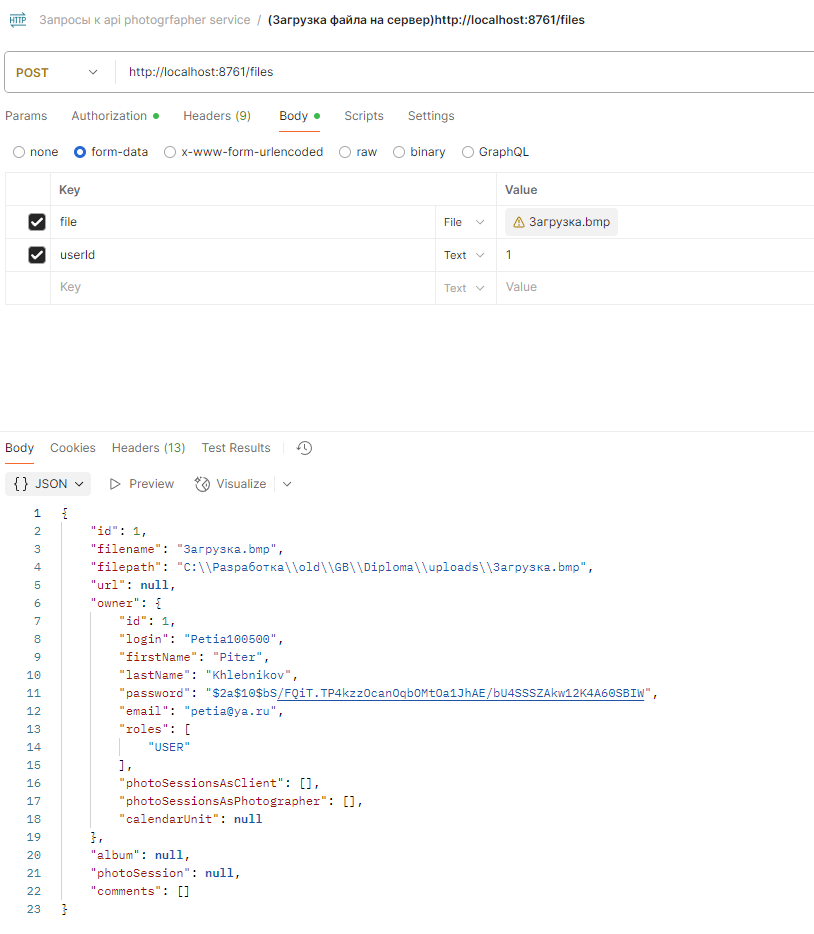


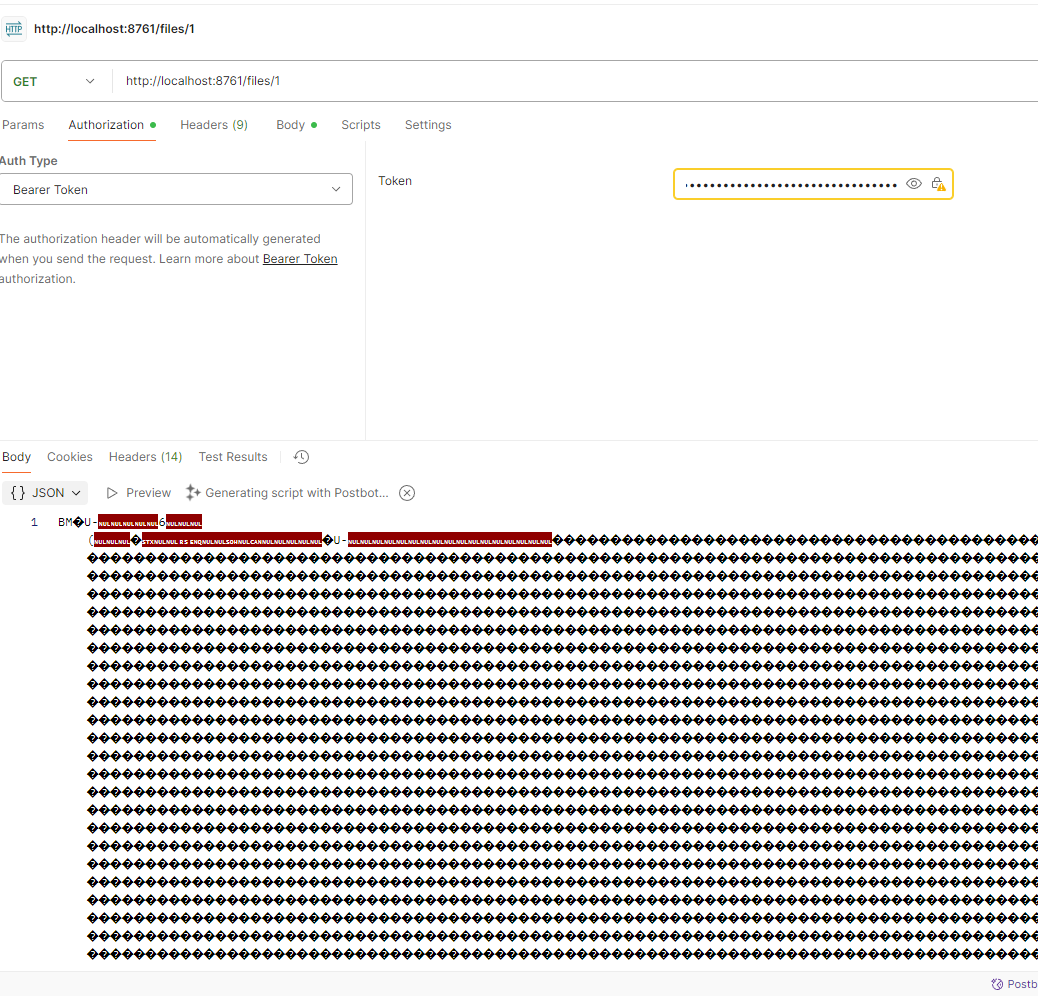


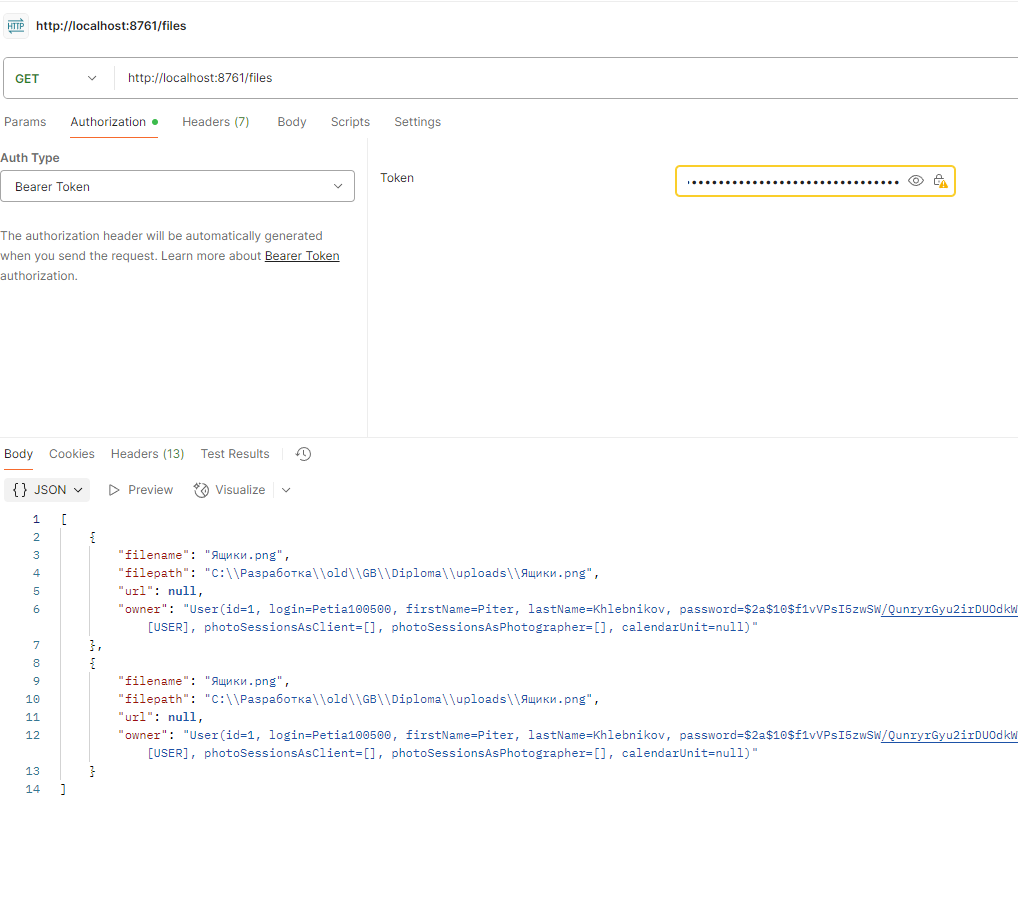


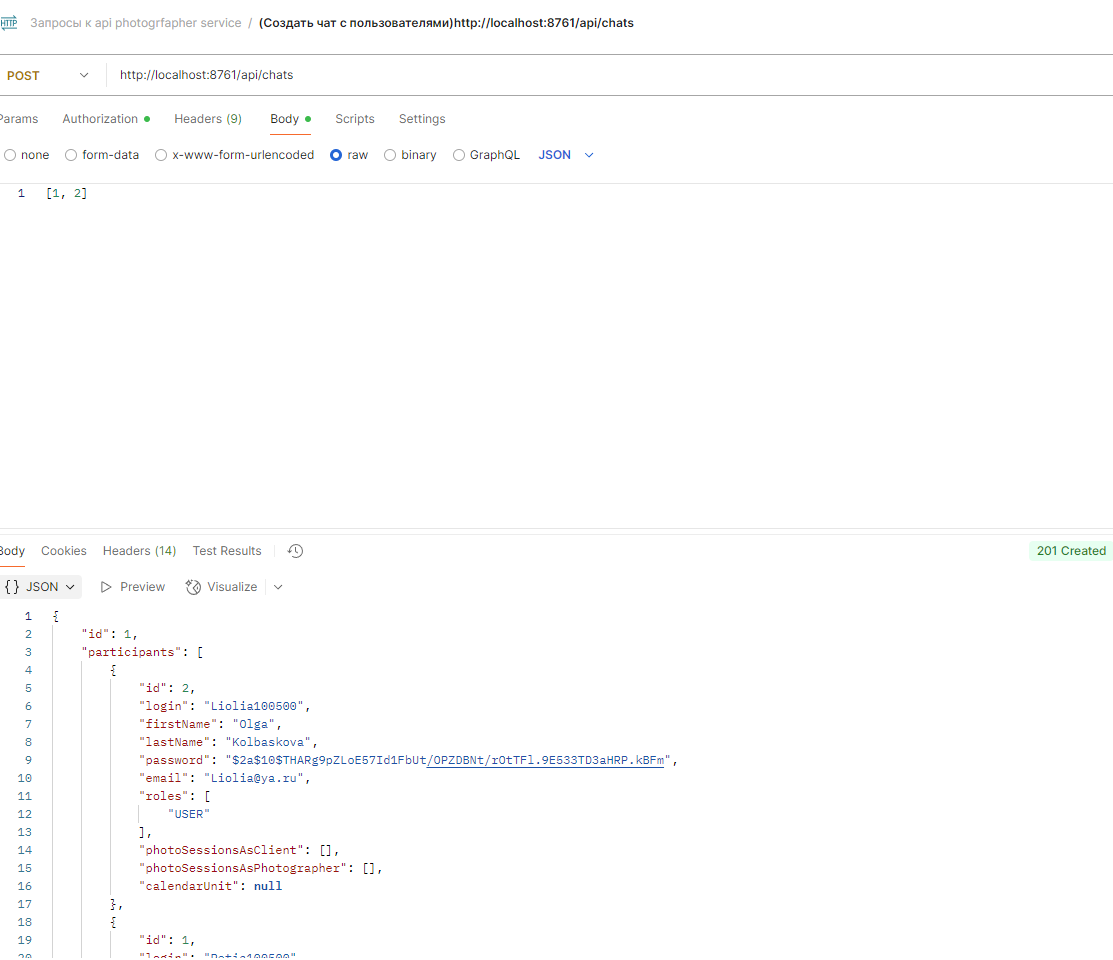


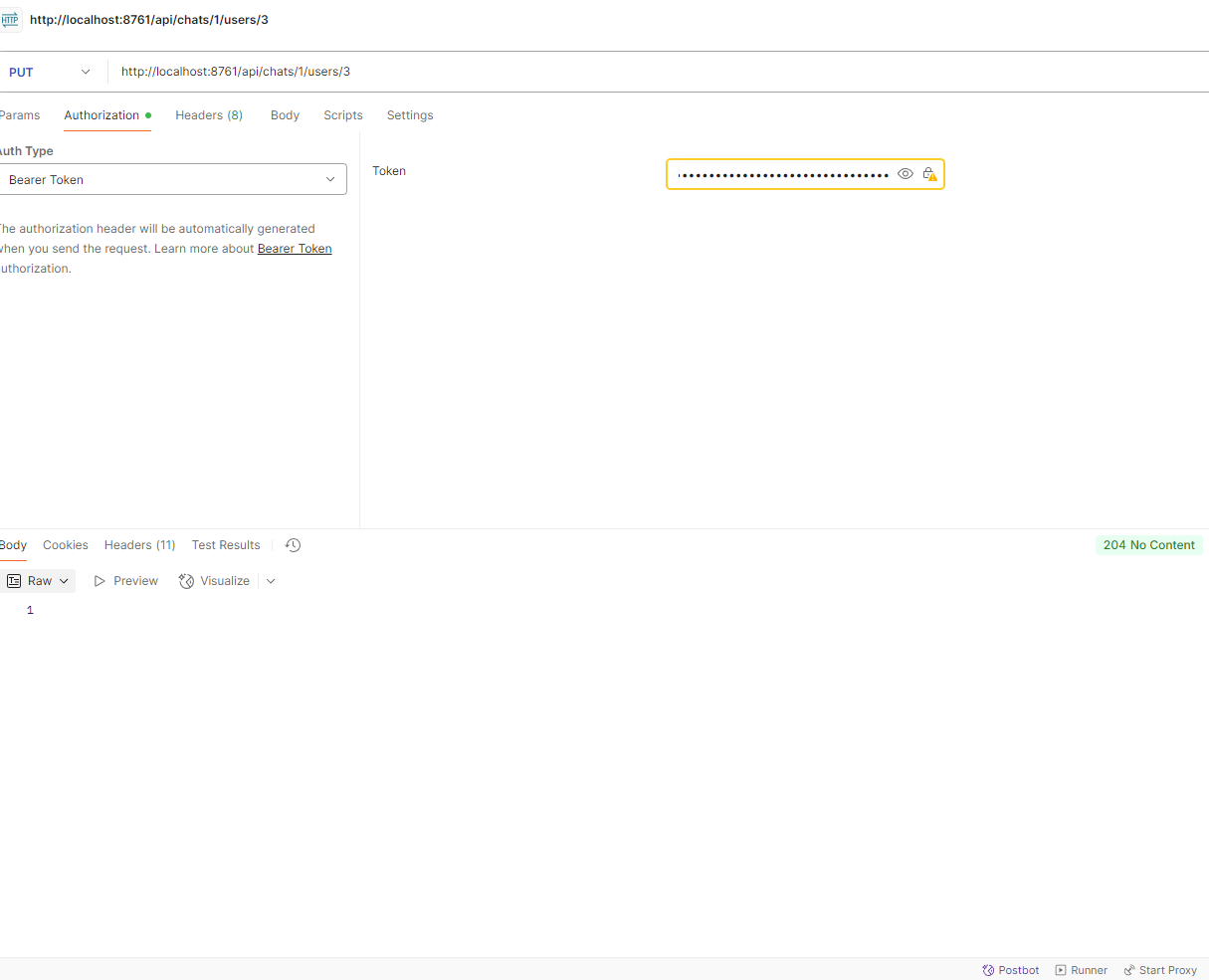


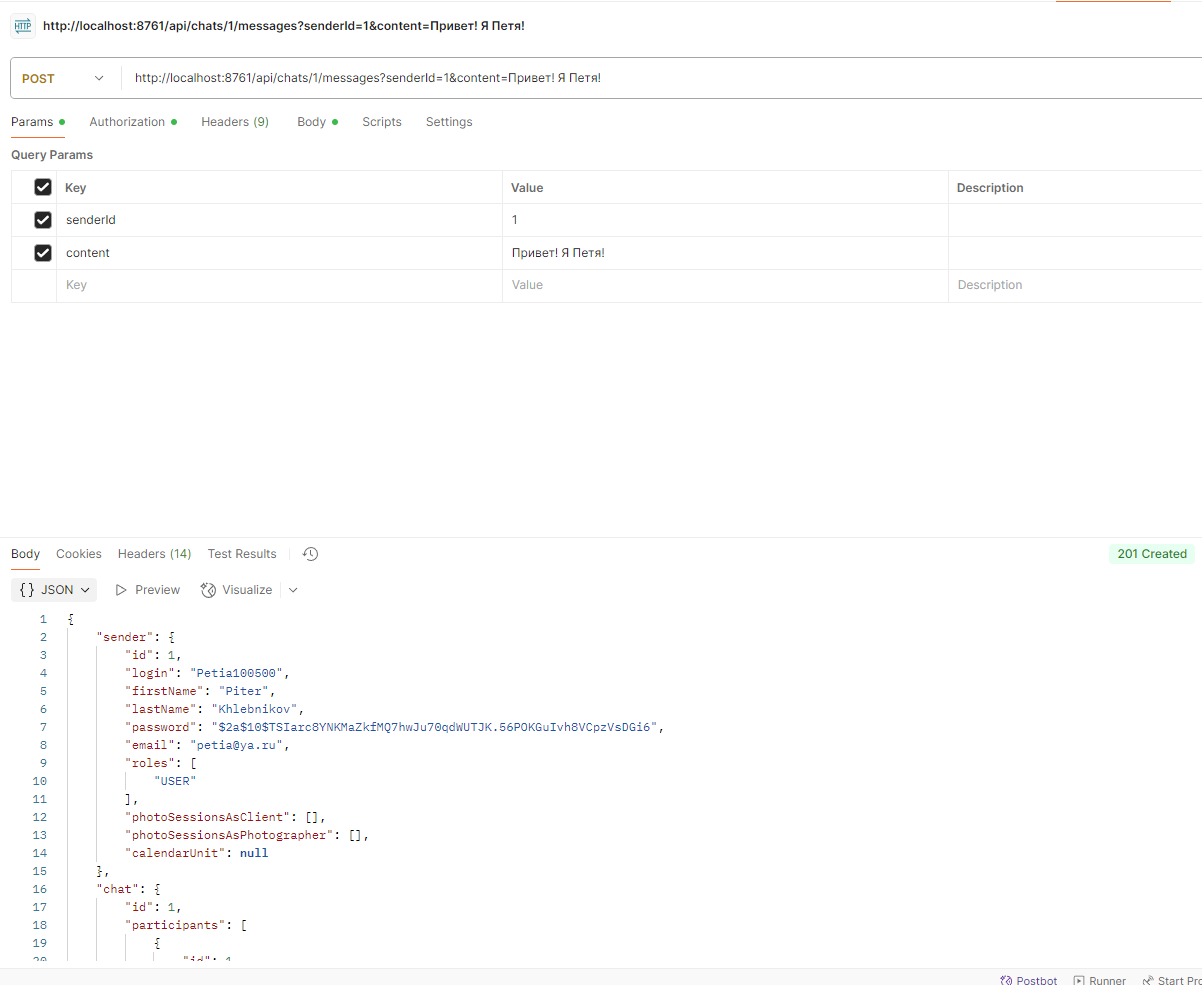












## 8. Приложение 2.

Готовый проект дипломного веб-приложения размещен на платформе GitHub.

Ссылка: <https://github.com/ValSGav/Diploma.git>