Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное

образовательное учреждение высшего образования

«Самарский национальный исследовательский университет

имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №3

Дисциплина: «Технологии сетевого программирования»

Тема «Авторизация»

Выполнили: Жиляев М.И.

Прохоров С.А.

Группа: 6301-010302D

Самара 2025

1. Задание на лабораторную работу

Цель работы: Реализовать систему авторизации пользователей, используя JWT-токены, и обеспечить безопасность API.

1. **Реализация механизма аутентификации:**
   * Регистрация нового пользователя;
   * Вход в систему и получение JWT-токена.
2. **Настройка middleware для защиты API:**
   * Проверка валидности токена;
   * Ограничение доступа к определенным эндпоинтам.
3. **Реализация управления пользователями:**
   * Изменение пароля;
   * Выход из системы (отзыв токена - по желанию реализовать blacklist токенов).
4. **Тестирование API:**
   * Проверка работы аутентификации с помощью Postman/Curl;
   * Проверка статусов HTTP-ответов (200, 400, 401, 403 и т. д.).
5. **Оформление отчета:**
   * Описание модели пользователя и JWT-аутентификации;
   * Примеры запросов, ответов и их статусов;
   * Презентация с результатами работы.
6. Выполнение работы
   1. Реализация механизма аутентификации:

В ходе лабораторной работы был разработан механизм регистрации и аутентификации пользователей. Пользователь вводит свои данные в форму регистрации, после чего они передаются на сервер через HTTP-запрос (POST /register). Серверная часть проверяет уникальность предоставленной информации, шифрует пароль с использованием алгоритма Bcrypt и сохраняет логин с зашифрованным паролем в базе данных. После успешной регистрации пользователь получает возможность авторизоваться в системе.  
При входе сервер проводит сравнение введённого пароля с сохранённым в базе зашифрованным значением. При совпадении данных генерируется JWT-токен, который передаётся пользователю. Этот токен, создаваемый индивидуально для каждого аккаунта, предоставляет доступ к закрытым разделам системы. Сервер проверяет валидность токена при каждом обращении к защищённым ресурсам, анализируя его подлинность и срок действия путём декодирования.

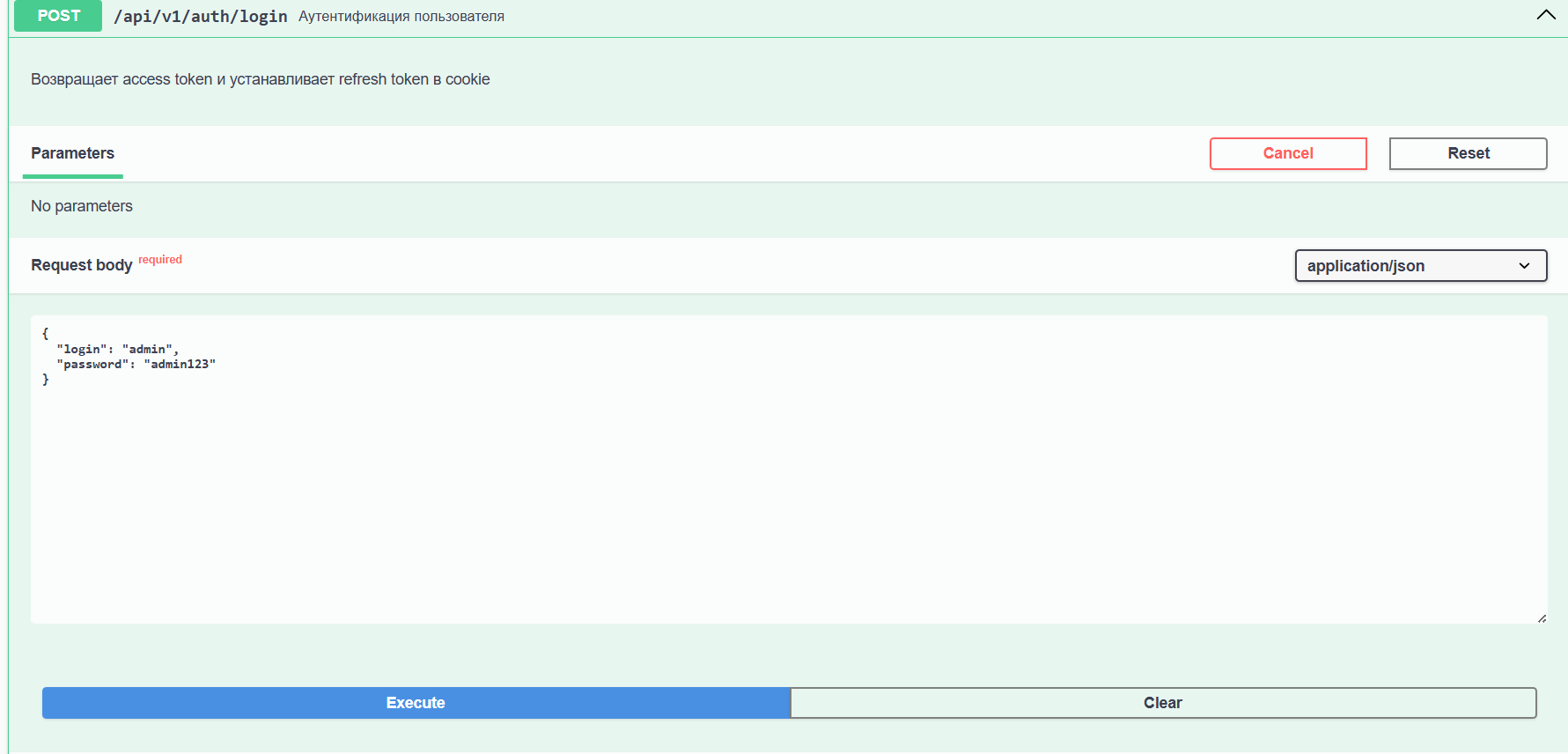


Рисунок 1 – Входные данные для запроса входа в Swagger

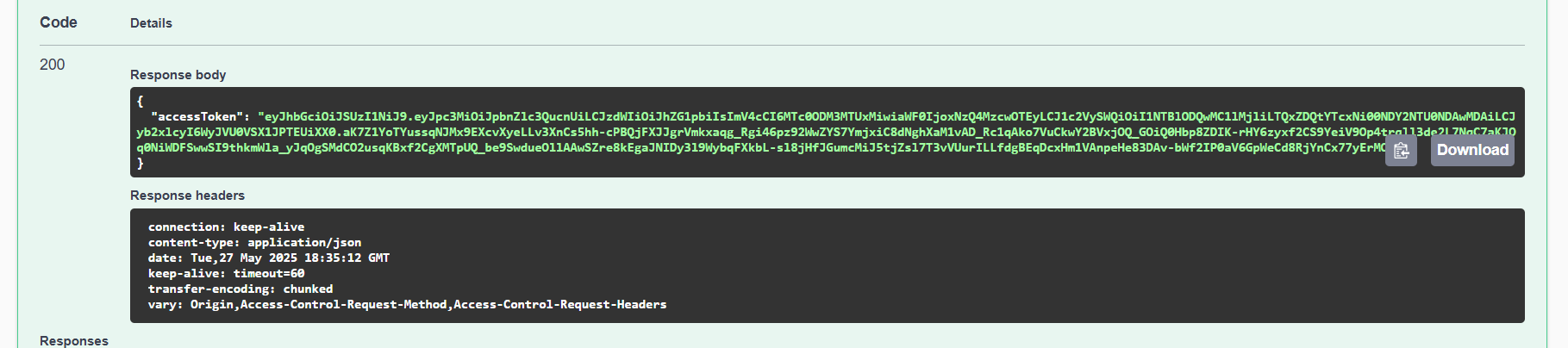


Рисунок 2 – Access токен, выданный по запросу через Swagger

* 1. Настройка middleware для защиты API:

Для реализации защиты API с проверкой JWT-токенов и ограничением доступа в Spring Boot требовалось настроить конфигурацию SecurityFilterChain, которая представляет собой набор фильтров безопасности в рамках Spring Security. Мы произвели настройку сразу для двух форматов - тестирования самого веб-приложения и тестирования системы через Postman.

В нашем приложении не хранится состояние сессии, поэтому мы отключили защиту от CSRF(Cross-Site Request Forgery) с помощью csrf(AbstractHttpConfigurer::disable), а также отключили базовую HTTP-аутентификацию с помощью httpBasic(AbstractHttpConfigurer::disable).

На следующем шаге мы создали бин фильтра JwtFilter, который отвечает за проверку JWT-токена в запросах, а также реализовали сам JWT-фильтр в классе JwtFilter. JWT-токен содержит информацию о имени пользователя, дате создания самого токена, и находится в заголовке авторизации запроса. Класс JwtService позволяет генерировать токены и производить их валидацию.

Код фильтра:

package dev.invest.config.filter;

import dev.invest.service.JwtService;

import dev.invest.service.UserService;

import io.github.resilience4j.core.lang.NonNull;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

@Slf4j

@Component

@RequiredArgsConstructor

public class JwtAuthFilter extends OncePerRequestFilter {

private final JwtService jwtService;

private final UserService userDetailsService;

@Override

protected void doFilterInternal(

@NonNull HttpServletRequest request,

@NonNull HttpServletResponse response,

@NonNull FilterChain filterChain

) throws ServletException, IOException {

final String authHeader = request.getHeader("Authorization");

if (authHeader == null || !authHeader.startsWith("Bearer ")) {

filterChain.doFilter(request, response); // Пропускаем, если нет токена (если эндпоинт не требует авторизации)

return;

}

final String jwt = authHeader.substring(7); // Убираем "Bearer "

String username = jwtService.extractUsername(jwt); // Извлекаем username из токена

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

if (jwtService.isTokenValid(jwt, userDetails.getUsername())) {

// Создаем аутентифицированный объект и сохраняем в SecurityContext

UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(

userDetails,

null,

userDetails.getAuthorities()

);

authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

filterChain.doFilter(request, response);

}

}

* 1. Реализация управления пользователями:

В сервисе предусмотрена возможность изменения пароля через специальную форму. Для обработки запросов на обновление пароля реализована конечная точка (POST "/api/v1/user/password"). В теле запроса передаются текущий и новый пароли. Серверная часть извлекает данные аутентифицированного пользователя из контекста безопасности, проверяет совпадение текущего пароля с хранящимся в базе данных с помощью метода PasswordEncoder.matches(), после чего обновляет учетные данные при успешной проверке.

Для обеспечения безопасности система использует stateless JWT-токены. Механизм отзыва токенов реализован через черный список (blacklist), функционирующий на базе сервиса TokenStoreService. Этот сервис сохраняет недействительные токены и проверяет их статус при каждом обращении к защищённым ресурсам. При выходе пользователя из системы (POST "/api/v1/logout") токен извлекается из заголовка Authorization и добавляется в черный список, блокируя его дальнейшее использование для доступа к закрытым разделам приложения.

* 1. Тестирование API:

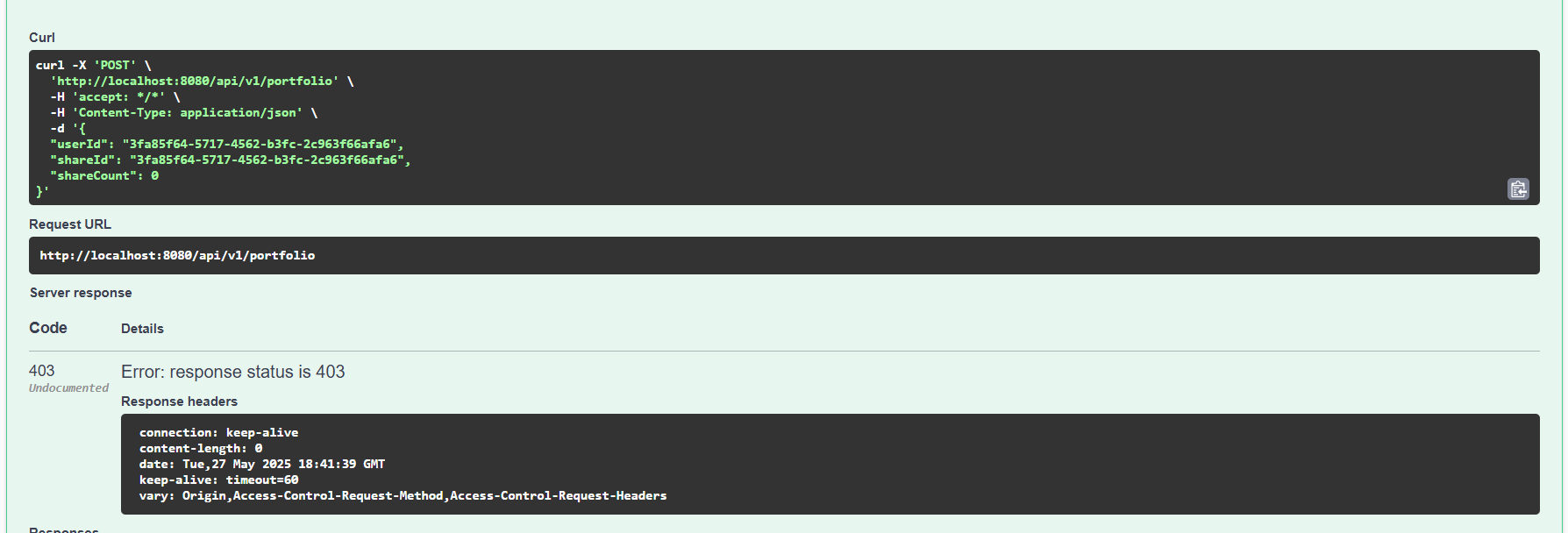


Рисунок 3 – Возвращение кода 403 если запрос не прошел аутентификацию