МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики и кибернетики
Кафедра технической кибернетики

ОТЧЕТ

по лабораторной работе №3
Дисциплина «Технологии сетевого программирования»

Выполнили студенты
группы 6301-010302D
Жданова В.И.
Русяева С.П.

САМАРА 2025

Реализована система аутентификации и авторизации на основе JWT токенов для веб-приложения.

Реализована регистрация нового пользователя:

POST /api/auth/register

```
  @PostMapping("/register")
   public ResponseEntity<?> register(@RequestBody UsersDto userDto) {
      return ResponseEntity.ok(authService.save(userDto));
   }
  public Users save(UsersDto userDto) {
      Users user = mapper.toUsers(userDto);
      user.setPassword(passwordEncoder.encode(user.getPassword()));
      if (usersRepository.findByEmail(userDto.getEmail()).isPresent())
         throw new DuplicateException(String.format("email %s already exists", userDto.
getEmail()));
      Users newUser = usersRepository.save(user);
      cartService.save(user.getUserId());
      return newUser;
   }
```

Реализован вход пользователя в систему:

POST /api/auth/login

```
  @PostMapping("/login")
   public ResponseEntity<JwtResponseDto> login(@RequestBody AuthDto authDto) {
      return ResponseEntity.ok(authService.authenticate(authDto));
   }
  public JwtResponseDto authenticate(AuthDto authDto) {
      Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(authDto.getEmail(), authDto.getPassword())
      );
```

```
        SecurityContextHolder.getContext().setAuthentication(authentication);
        UserDetails userDetails = (UserDetails) authentication.getPrincipal();
        Users user = usersRepository.findByEmail(authDto.getEmail())
                .orElseThrow(() -> new ObjectNotFoundException("User not found"));

        String accessToken = jwtProvider.generateAccessToken(userDetails, user.getUserId());
        String refreshToken = jwtProvider.generateRefreshToken(userDetails.getUsername());
        refreshTokenRepository.save(new RefreshToken(null, refreshToken, user,
Instant.now().plusMillis(refreshExpiration)));

        return new JwtResponseDto(accessToken, refreshToken);
    }
```

Содержимое токена:

- Subject (email пользователя)
- User ID
- Роль (USER/ADMIN)
- Время выдачи и expiration

Реализована через проверка валидности токена через JwtFilter. Проверяется:

- Наличие токена в заголовке Authorization
- Валидность подписи
- Срок действия
- Наличие в черном списке (для logout)

Реализовано ограничение доступа к эндпоинтам. Для доступа необходима авторизация (кроме /api/auth/**). Также некоторые эндпоинты доступны только пользователям с определенными ролями:

```
@PreAuthorize("hasRole('USER')")
@PostMapping("/{productId}")
public ResponseEntity<CartItem> createCartItem(@PathVariable Long productId) {
```

```
  return
ResponseEntity.status(HttpStatus.CREATED).body(productService.addProductToCart(productId));

}

@PreAuthorize("hasRole('ADMIN')")

@PutMapping("/{productId}")

public ResponseEntity<Product> updateProduct(@Valid @RequestBody ProductUpdateDto
updateDto, @PathVariable Long productId) {

  return ResponseEntity.ok(productService.update(updateDto, productId));

}
```

Изменение пароля:

POST /api/shop/user/me/edit/password

```
@PreAuthorize("hasRole('USER')")

@PatchMapping("/me/edit/password")

public ResponseEntity<?> updatePassword(@RequestBody PasswordDto passwordDto) {

  usersService.updatePassword(passwordDto);

  return ResponseEntity.ok().build();

}

  public void updatePassword(PasswordDto passwordDto) {

      Users user = usersRepository.findById(jwtProvider.getCurrentUserId())

            .orElseThrow(() -> new ObjectNotFoundException("User not found"));

      if(user.getPassword().equals(passwordEncoder.encode(passwordDto.getOldPass())))

         user.setPassword(passwordEncoder.encode(passwordDto.getNewPass()));

      usersRepository.save(user);

  }
```

Выход из системы (отзыв токена):

POST /api/auth/logout

```
@PostMapping("/logout")

   public ResponseEntity<?> logout(@RequestBody RefreshTokenDto refreshTokenDto,
@RequestHeader("Authorization") String authHeader) {
```

```
        refreshTokenService.findByToken(refreshTokenDto.getRefreshToken())
                .ifPresent(refreshTokenService::delete);
        String accessToken = authHeader.substring(7);
        blackListService.addToBlacklistToken(accessToken,
jwtProvider.extractExpiration(accessToken));
        return ResponseEntity.ok().build();
    }
```

Обновление access токена:

POST /api/auth/refresh

```
@PostMapping("/refresh")
    public ResponseEntity<JwtResponseDto> refreshToken(@RequestBody RefreshTokenDto
refreshTokenDto) {
        return ResponseEntity.ok(refreshTokenService.refresh(refreshTokenDto));
    }
   public JwtResponseDto refresh(RefreshTokenDto refreshTokenDto) {
        String requestRefreshToken = refreshTokenDto.getRefreshToken();
        return findByToken(requestRefreshToken)
                .map(this::verifyExpiration)
                .map(existingToken -> {
                    delete(existingToken);
                    UserDetails userDetails =
userDetailsService.loadUserByUsername(existingToken.getUser().getEmail());
                    String refreshToken =
jwtProvider.generateRefreshToken(existingToken.getUser().getEmail());
                    String accessToken = jwtProvider.generateAccessToken(userDetails,
existingToken.getUser().getUserId());
                    return new JwtResponseDto(accessToken, refreshToken);
                })
                .orElseThrow(() -> new ObjectNotFoundException("Refresh token not found"));
    }
```