Алгоритмы работы с пользователями

Обзор системы пользователей

Система управления пользователями обеспечивает регистрацию, аутентификацию, управление ролями и доступ к пользовательской информации с использованием JWT токенов и ролевой модели безопасности.

Управление ролями пользователей

Запрос статуса администратора (`requestAdmin`)

Система позволяет обычным пользователям получать права администратора через секретный код.

Алгоритм повышения роли

1. **Извлечение данных аутентификации**: Получение текущего пользователя из `SecurityContext`
2. **Валидация существования пользователя**: Проверка наличия пользователя в базе данных
3. **Проверка секретного кода**:
   - Сравнение переданного секрета с хешированным значением из конфигурации
   - Использование `BCryptEncoder.matches()` для безопасного сравнения
4. **Обновление роли в базе**: Изменение роли пользователя на `User.Role.ADMIN`
5. **Генерация нового JWT**: Создание токена с обновленной ролью через `changeRoleInJwt()`
6. **Возврат обновленного токена**: Клиент получает новый JWT с правами администратора

Безопасность операции

- **Хеширование секрета**: Секретный код хранится в хешированном виде в конфигурации
- **Транзакционность**: Операция выполняется в рамках транзакции
- **Валидация токена**: Проверка аутентификации перед выполнением операции
- **Обработка ошибок**: `AccessDeniedException` при неверном секрете

Получение информации о пользователях

Информация о текущем пользователе (`getMe`)

1. **Извлечение аутентификации**:
   Получение `Authentication` из `SecurityContextHolder`
2. **Извлечение ID**: Преобразование `auth.getName()` в `UUID`
3. **Поиск пользователя**: Запрос данных из базы по ID
4. **Формирование ответа**: Создание `UserInfoResponse` с базовой информацией

Структура ответа

- **ID пользователя**: Уникальный идентификатор UUID
- **Логин**: Имя пользователя для входа
- **Роль**: Текущая роль (USER/ADMIN)

Список всех пользователей (`getAll`)

Доступно только администраторам для управления системой.

Алгоритм получения списка

1. **Проверка прав доступа**: `@PreAuthorize("hasRole('ROLE_ADMIN')")`
2. **Создание пагинации**: `PageRequest.of(page, size)`
3. **Запрос к базе данных**: Получение страницы пользователей
4. **Маппинг результатов**: Преобразование в список `UserInfoResponse`
5. **Возврат данных**: Список с базовой информацией о пользователях

Параметры пагинации

- **page**: Номер страницы (по умолчанию 0)
- **size**: Размер страницы (по умолчанию 3)

Поиск и валидация пользователей

Поиск по логину (`findByLogin`)

1. **Валидация логина**: Проверка через `User.validateLogin()`
2. **Запрос к базе данных**: Поиск пользователя по уникальному логину
3. **Обработка отсутствия**: `NotFoundException` если пользователь не найден
4. **Возврат результата**: Объект `User` или исключение

Проверка уникальности логина (`checkIfNotExistsByLogin`)

1. **Валидация формата логина**: Соответствие требованиям системы
2. **Проверка существования**: Запрос `existsByLogin()` к репозиторию
3. **Обработка дубликата**: `BadRequestException` если логин уже занят
4. **Успешное завершение**: Логин доступен для регистрации

Создание пользователя (`save`)

1. **Валидация данных пользователя**: Проверка через `User.validateUser()`
2. **Транзакционное сохранение**: `@Transactional` для целостности данных
3. **Сохранение в базе**: Использование `userRepository.save()`
4. **Возврат результата**: Сохраненный объект пользователя с ID

Система аутентификации

Извлечение данных аутентификации (`getAuthData`)

1. **Получение контекста**: `SecurityContextHolder.getContext().getAuthentication()`
2. **Проверка наличия**: Валидация на `null`
3. **Проверка аутентификации**: `auth.isAuthenticated()`
4. **Обработка ошибок**: `UnauthorizedException` при отсутствии аутентификации

Использование в системе

- **Извлечение ID пользователя**: `UUID.fromString(auth.getName())`
- **Получение JWT токена**: `auth.getCredentials().toString()`
- **Проверка ролей**: Через `auth.getAuthorities()`

API эндпойнты и документация

Swagger документация

Все эндпойнты документированы с использованием OpenAPI 3.0 аннотаций:

`/api/user/admin` (PATCH)

- **Описание**: Запрос роли администратора
- **Авторизация**: Требуется JWT токен
- **Тело запроса**: `AdminRequest` с секретным кодом

- **Ответ**: Обновленный JWT токен с правами ADMIN

`/api/user/me` (GET)

- **Описание**: Информация о текущем пользователе
- **Авторизация**: Требуется аутентификация
- **Ответ**: `UserInfoResponse` с данными пользователя

`/api/user` (GET)

- **Описание**: Список пользователей с пагинацией
- **Авторизация**: Только администраторы
- **Параметры**: page (default: 0), size (default: 3)
- **Ответ**: Массив `UserInfoResponse`

Коды ответов

- **200**: Успешное выполнение операции
- **401**: Ошибка аутентификации или неверные данные
- **403**: Недостаточно прав доступа
- **404**: Пользователь не найден
- **400**: Некорректные входные данные

Обработка ошибок и безопасность

Типы исключений

- `NotFoundException` : Пользователь не найден по ID или логину
- `BadRequestException` : Логин уже существует или некорректные данные
- `UnauthorizedException` : Проблемы с аутентификацией
- `AccessDeniedException` : Неверный секрет администратора

Безопасность системы

- **Хеширование паролей**: Секретные данные хранятся в хешированном виде
- **Валидация входных данных**: Проверка всех пользовательских данных
- **Ролевая модель**: Строгое разделение прав USER и ADMIN
- **JWT токены**: Безопасная передача информации о ролях

Транзакционная целостность

- `@Transactional` : Все операции изменения данных
- **Атомарность**: Откат изменений при ошибках
- **Консистентность**: Целостность данных при обновлении ролей