Geekbrains

**Разработка веб-приложения "Управление планером" с использованием Java и Spring Framework**

IT-специалист:

Веб-разработка на Java

Лихачёва В.М.

Москва

2025

**Дипломный проект**

**«Разработка веб-приложения "Управление планером" с использованием Java и Spring Framework»**

**Содержание**

|  |  |
| --- | --- |
| Введение……………………………………………………………………. | 3 |
| Глава 1. Планирование проекта………………………………………… | 5 |
| 1.1 Планирование web-приложения...…………………………………… | 5 |
| 1.2 Выбор технологического стека ………….…………………………… | 6 |
| 1.3 Среда разработки и архитектура проекта….………………………… | 7 |
| Глава 2. Проектирование приложения …..……………………………… | 9 |
| 2.1 Создание проекта и выбор необходимых зависимостей…………… | 9 |
| 2.2 Разработка приложения…………..…………………….. | 11 |
| 2.3 Проектирование базы данных…………………………………………  2.3.1 Структура таблиц и их взаимосвязи………………………………...  2.3.2 Связи между сущностями и персистентностями объектов………..  2.3.3 Настройки параметров приложения………………………………...  2.3.4 Обработка данных и взаимодействие с репозиторием…………….  2.4 Реализация «Thymeleaf-контроллеров»……………………………… | 17  17  20  23  25  38 |
| 2.5 Разработка конфигурации безопасности web-приложения ………... | 44 |
|  |  |
| Заключение………………………………………………………………… | 51 |
| Список использованной литературы…………………………………….. | 52 |

**Введение**

Дипломный проект нацелен на создание веб-приложения для управления планером с использованием современных технологий программирования, основанных на Java и Spring Framework. Проект включает функционал аутентификации пользователей, управления планером с возможностью добавления задач, ролей и уведомлений. В ходе работы над проектом также уделено внимание оптимизации кода, безопасности данных и эффективному взаимодействию между компонентами приложения.

Тема представляет собой комплексный подход к разработке веб-приложения, интегрирующего функционал управления планером, что позволит продемонстрировать знание Java, Spring Framework, а также навыки проектирования и разработки программного обеспечения в соответствии с современными стандартами и требованиями к качеству.

Данная тема объединяет аспекты разработки программного обеспечения, базовых технологий веб-приложений и лучших практик в области программирования на JavaSpring, что делает ее актуальной и значимой для текущего IT-сектора.

Актуальность данной работы определяется необходимостью разработки современного веб-приложения, способного эффективно управлять планером и повышать продуктивность пользователей. В современном мире управление временем и задачами играет ключевую роль в повседневной деятельности людей, и создание такого приложения станет значимым вкладом в область информационных технологий.

Целью данного проекта является разработка веб-приложения "Управление планером", используя Java и Spring Framework, с целью обеспечить удобное и функциональное средство планирования задач и управления временем. Приложение должно быть интуитивно понятным, безопасным и масштабируемым для удовлетворения потребностей пользователей.

Основные задачи проекта включают:

1. Разработку пользовательского интерфейса, обеспечивающего удобство использования приложения для добавления, редактирования и удаления задач.

2. Создание системы аутентификации и авторизации пользователей для обеспечения безопасности данных.

3. Интеграцию функциональности управления планером, позволяющую пользователям эффективно организовывать свои задачи.

4. Оптимизацию кода и архитектуры приложения для обеспечения его производительности и масштабируемости.

Данный проект имеет ключевое значение для повышения эффективности управления задачами и временем пользователей, а также демонстрации навыков разработки веб-приложений с применением современных технологий программирования.

**Глава 1. Планирование проекта**

**1.1 Планирование web-приложения**

В ходе разработки проекта "Управление планером" было проведено планирование и определение необходимых шагов для успешной реализации приложения. Стадия планирования включала в себя:

1. Планирование проекта:

- Определение функциональных и нефункциональных требований к приложению.

- Проектирование архитектуры приложения и выбор используемых технологий.

2. Настройка окружения разработки:

- Установка и конфигурация необходимых инструментов разработки, включая Java Development Kit (JDK), Apache Maven для сборки проекта и Spring Boot для создания и запуска приложения.

3. Создание проекта:

- Использование Spring Initializr для создания нового проекта.

- Выбор необходимых зависимостей, таких как Spring Web, Spring Data JPA, Lombok, Spring Security, Thymeleaf, для обеспечения необходимой функциональности приложения.

Эти этапы представляют собой основу для успешного старта проекта "Управление планером" и обеспечивают правильное направление для последующих этапов разработки приложения. Планирование и настройка окружения разработки являются основополагающими этапами, определяющими дальнейшее развитие проекта и его успешное завершение.

**1.2 Выбор технологического стека для приложения "Управление планером"**

**При разработке приложения "Управление планером" были сделаны стратегические решения по выбору технологий, обеспечивающих эффективное функционирование и простоту в поддержке. Вот основные аспекты выбранного технологического стека:**

**1. Язык программирования:**

**Для разработки серверной части приложения выбран Java, так как он обладает высокой производительностью и надежностью.**

**2. Фреймворк для веб-разработки:**

**Использование Spring Boot с модулями Spring MVC и Spring Security обеспечивает удобство в разработке и обеспечивает безопасность приложения.**

**3. База данных:**

**Для хранения данных пользователей и другой информации в приложении "Управление планером" была использована MySQL - надежная и широко используемая реляционная база данных.**

**4. Хранение данных о планере:**

**Данные о планере и управлении им эффективно хранятся в MySQL, обеспечивая надежность и целостность информации.**

**5. Шаблонизатор:**

**Для генерации HTML-страниц на сервере в приложении "Управление планером" был использован Thymeleaf, что позволяет удобно работать с данными из Java кода.**

**6. Аутентификация и авторизация: Spring Security обеспечивает безопасность приложения, включая механизмы аутентификации пользователей и управления доступом к ресурсам.**

**Технологический стек, описанный выше, способствует устойчивости и эффективности приложения "Управление планером", а также обеспечивает гибкость в будущем развитии проекта. Несмотря на изменения в условиях и требованиях, выбранные технологии позволят обеспечить стабильную работу и удовлетворение потребностей пользователей.**

**1.3 Среда разработки и архитектура проекта "Управление планером"**

**При разработке дипломного проекта "Управление планером" использовалась среда разработки IntelliJ IDEA для создания серверной части приложения. Этот инструментарий обеспечивает высокий уровень функциональности, производительности и безопасности, необходимый для успешной реализации проекта.**

**Архитектура приложения, ориентированная на паттерн проектирования MVC (Model-View-Controller), представляет собой систему, состоящую из следующих основных компонентов:**

**1. Модель (Model):**

**В рамках проекта "Управление планером" модель представлена несколькими сущностями, такими как Note, Planner, Role и User, а также двумя перечислениями (enums): NoteStatus и RoleName. Каждая сущность и перечисление описывают определенные аспекты данных и их взаимосвязи в приложении.**

**2. Представление (View):**

**Отображение данных для пользователя в веб-приложении реализуется с помощью шаблонизаторов, таких как Thymeleaf или JSP. Представления обеспечивают визуализацию информации для пользователей, улучшая пользовательский опыт.**

**3. Контроллер (Controller):**

**Контроллеры, включающие NoteController и UserController, обрабатывают запросы от пользователей и взаимодействуют с моделью и представлением. Они содержат методы для обработки различных типов запросов и управления бизнес-логикой.**

**4. Репозиторий (Repository):**

**Репозитории, такие как NoteRepository, PlannerRepository, RoleRepository и UserRepository, обеспечивают доступ к данным из базы данных MySQL. Они содержат методы для получения, сохранения, обновления и удаления данных.**

**5. Сервис (Service):**

**Сервисы, такие как NoteService, NotificationService, PlannerService и UserService, содержат бизнес-логику приложения. Интерфейсы сервисов и их реализации обеспечивают выполнение операций с данными и управление функциональностью приложения.**

**Дополнительно, был создан CustomUserDetailsService для обеспечения дополнительной функциональности при работе с пользователями в приложении "Управление планером". Эта архитектурная концепция обеспечивает модульность, гибкость и масштабируемость проекта, позволяя эффективно управлять его разработкой и поддержкой.**

**Глава 2. Проектная стадия**

**2.1. Создание проекта и выбор необходимых зависимостей**

При создании проекта "Управление планером" с использованием Spring Initializr были выбраны следующие ключевые зависимости:

1. **\*spring-boot-starter-data-jpa:**

Эта зависимость содержит необходимые библиотеки для работы с JPA (Java Persistence API). Она облегчает взаимодействие с базой данных, предоставляя удобные инструменты для создания объектно-реляционной модели.

2. **\*spring-boot-starter-jdbc:**

Зависимость включает инструменты для работы с JDBC (Java Database Connectivity), что обеспечивает доступ к базе данных и выполнение SQL-запросов. Это позволяет эффективно управлять данными приложения.

3. **\*spring-boot-starter-thymeleaf:**

Данная зависимость предоставляет интеграцию с Thymeleaf - шаблонизатором для генерации динамических HTML-страниц на сервере. Thymeleaf облегчает разработку пользовательского интерфейса и взаимодействие с данными.

4. **\*spring-boot-starter-web:**

Включает инструменты для разработки веб-приложений на Spring Boot. Зависимость автоматически добавляет компоненты, необходимые для создания веб-сервисов, обработки HTTP-запросов и взаимодействия с клиентами.

5. **\*h2 (runtime):**

Встраиваемая база данных H2 используется в режиме runtime для хранения данных при разработке и тестировании приложения. H2 предоставляет простой и легковесный способ работы с базой данных во время разработки.

6. **\*spring-boot-starter-test:**

Зависимость включает инструментарий для написания и выполнения тестов в приложении. Это помогает обеспечить высокое качество программного обеспечения путем тестирования основных функциональностей.

7. **\*spring-boot-starter-security:**

Данная зависимость предоставляет инструменты для обеспечения безопасности приложения. Она включает механизмы аутентификации, авторизации и защиты ресурсов, обеспечивая безопасность данных и операций.

8. **\*spring-boot-starter-actuator:**

Зависимость включает инструменты мониторинга и управления приложением. Актуаторы обеспечивают доступ к метрикам, healthchecks и другим полезным показателям работы приложения.

9. **\*spring-integration-file и \*spring-integration-core:**

Зависимости используются для интеграции файловой системы в приложение. Они предоставляют инструменты для обработки файлов и данных, что полезно при взаимодействии с файловыми ресурсами.

10. **\*lombok:**

Библиотека Lombok упрощает написание кода путем автоматической генерации методов, таких как геттеры и сеттеры. Это способствует уменьшению объема шаблонного кода и повышает производительность разработки.

11. **\*jakarta.validation-api:**

Данная зависимость предоставляет инструменты для валидации данных и использования аннотаций валидации. Она облегчает проверку корректности входных данных и обеспечивает целостность данных в приложении.

12. **\*mysql-connector-java:**

MySQL Connector/J - это официальный драйвер JDBC для интеграции Java-приложений с базой данных MySQL. Обеспечивает эффективную работу приложений, поддерживает основные возможности MySQL, такие как транзакция и хранимые процедуры, манипуляцию данными и надежное взаимодействие с базой данных.

Выбор этих зависимостей обеспечивает не только необходимый функционал для разработки проекта "Управление планером", но также упрощает процесс разработки, улучшает безопасность, обеспечивает мониторинг и управление приложением, обеспечивает тестирование и интеграцию с базой данных и файловой системой.

**2.2 Разработка приложения**

Приложение "Управление планером" предоставляет широкий спектр функциональности, основанный на различных моделях данных, которые охватывают ключевые аспекты управления записями, планерами, пользователями и их ролями. В рамках разработки интерфейса приложения, важно рассмотреть подробности каждой из этих моделей и функциональности, которую они предоставляют:

1. **Модель "Note" (рис.1):**

- Управление записями:

Модель Note предоставляет возможность создания, редактирования и удаления записей. Пользователи могут добавлять заголовок, текст и указывать статус каждой записи.

- Временные отметки:

Дата и время создания записи автоматически фиксируются при добавлении записи, что обеспечивает контроль хронологии событий.

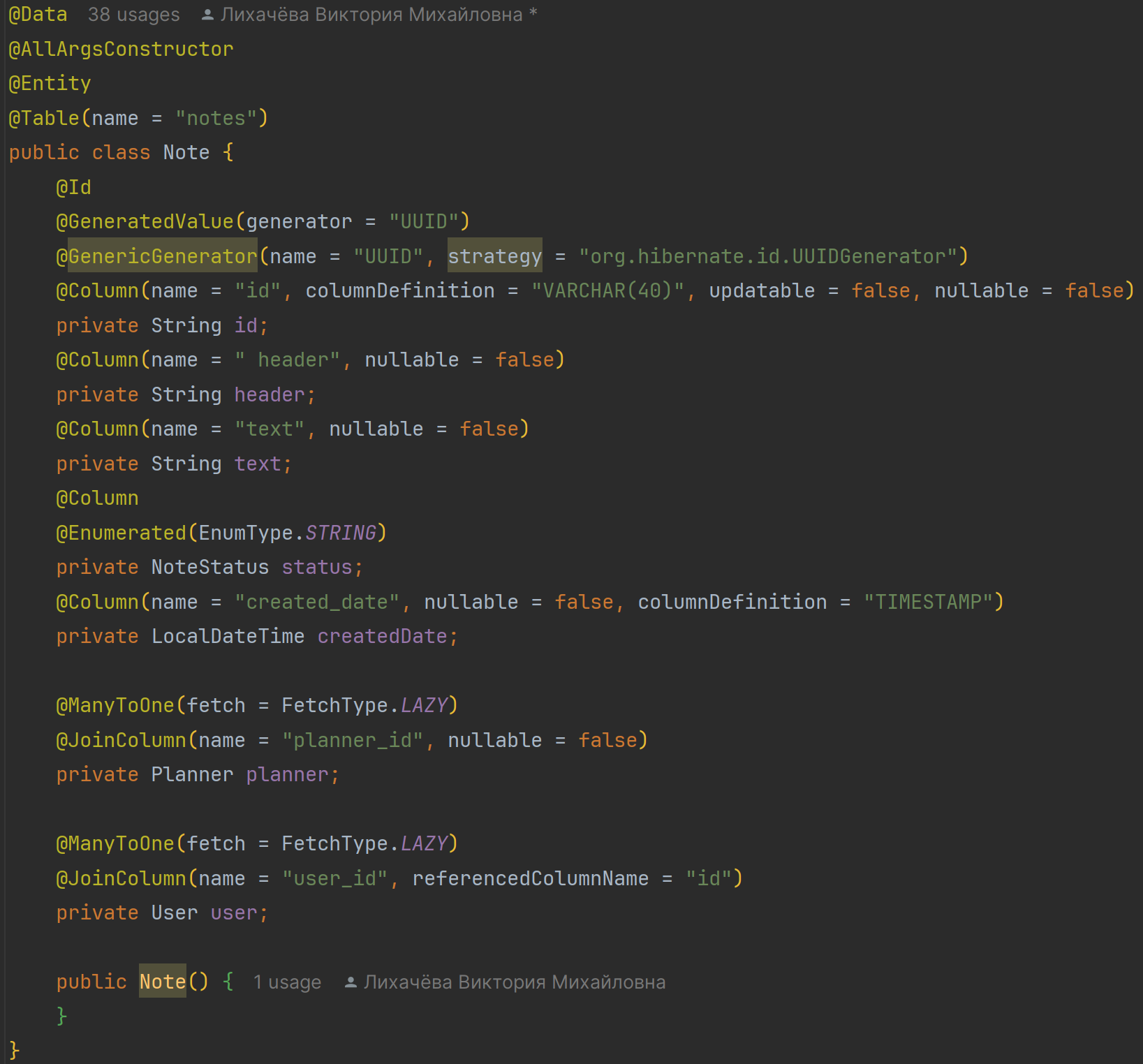


Рис.1 Модель "Note"

- Связь с Planner и User:

Каждая запись привязана к определенному планеру и пользователю, что обеспечивает структурированность и организацию данных.

2. **Модель "Planner" (рис.2):**

- Управление планерами:

Модель Planner позволяет пользователям создавать новые планеры, добавлять им заголовок и описание. Планеры могут содержать список записей, а также связаны с определенным пользователем-владельцем.

- Связь с Notes:

Каждый планер содержит список записей, что обеспечивает удобство организации и группировки данных внутри планера.

- Планирование и активация:

Предполагается реализация функционала активности планера, который может отражать его текущее состояние – активный или неактивный.

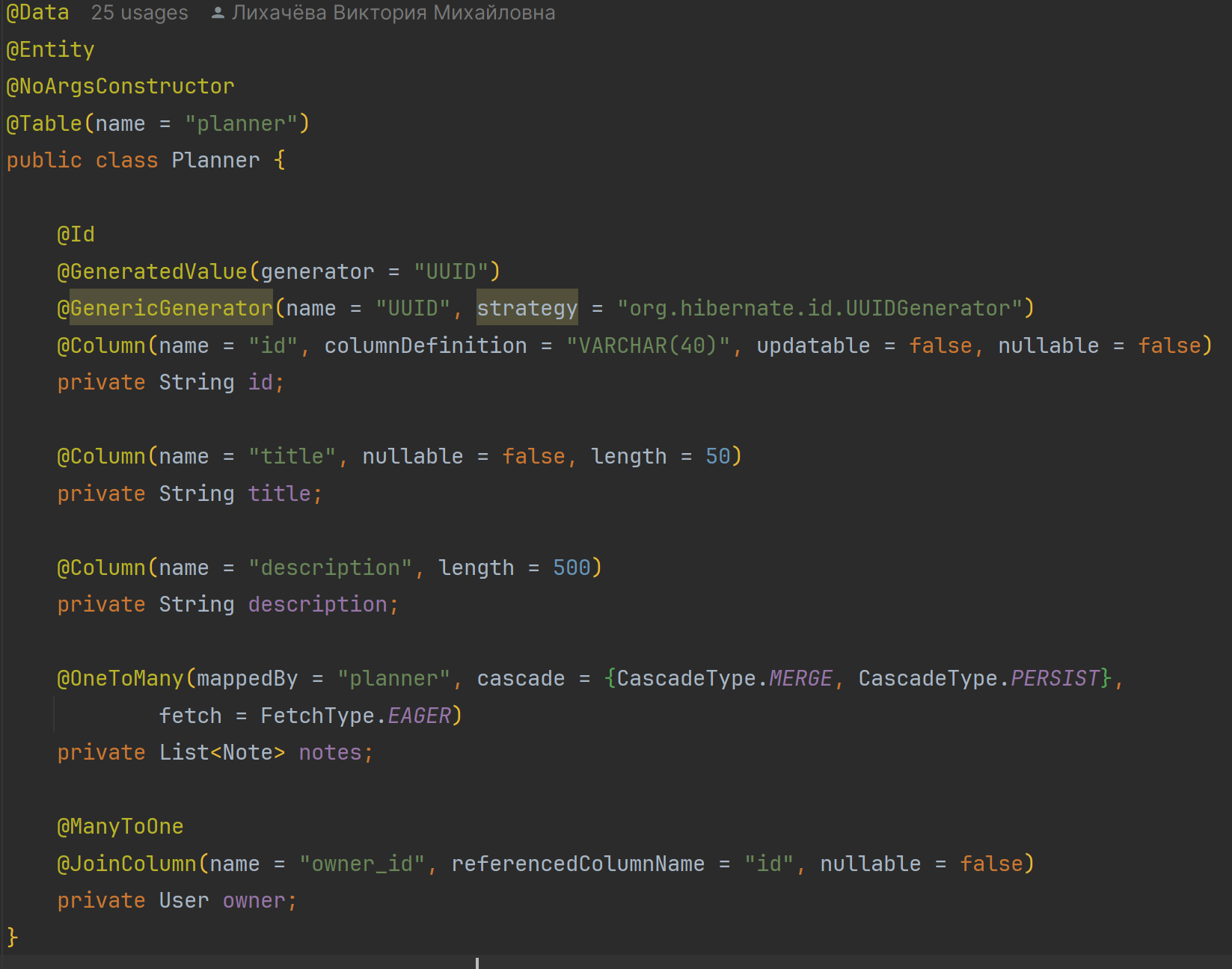


Рис.2 Модель "Planner"

3. **Модель "Role" (рис.3):**

- Управление ролями:

Модель Role представляет различные роли пользователей, такие как USER и ADMIN. Роли определяют уровень доступа и права пользователя в системе.

- Связь с User:

Каждая роль связана с определенным пользователем, что позволяет назначать конкретные роли для управления доступом к функциональности.



Рис.3 Модель "Role"

4. **Модель "User" (рис.4):**

- Управление пользователями:

Модель User содержит основные данные пользователя, включая имя, фамилию, адрес электронной почты и пароль. Пользователи могут управлять своими личными данными и авторизационной информацией.

- Связь с Notes и Planner:

Пользователи могут создавать записи и планеры, которые автоматически связываются с соответствующим пользователем.

- Назначение роли и планера:

Для каждого пользователя задается роль с соответствующими правами доступа и назначается планер, которым он управляет.

Благодаря функциональностям моделей приложения "Управление планером" пользователи могут эффективно организовывать свои задачи, делиться информацией, контролировать доступ к данным и управлять своими аккаунтами в системе. Эти модели обеспечивают стройную архитектуру данных, на основе которой будет реализован интуитивно понятный и удобный пользовательский интерфейс.

****

****

Рис.4 Модель "User"

**2.3. Проектирование базы данных**

**2.3.1 Структура таблиц и их взаимосвязи**

При анализе структуры таблиц и их взаимосвязей в приложении "Управление планером" рассматриваются основные компоненты данных, такие как сущности Note, Planner, Role и User, а также определяется организация колонок каждой таблицы, первичные и внешние ключи для установления связей между сущностями.

В данной главе представлен детальный анализ проектирования базы данных для приложения "Управление планером" на базе сущностей Note, Planner, Role и User. Рассмотрим структуру таблиц и их взаимосвязи:

1. **Таблица "notes":**

- Колонки (@Column):

- id: String первичный ключ;

- header: строка, заголовок записи;

- text: текст записи;

- status: перечисление NoteStatus;

- createdDate: дата и время создания записи;

- planner: внешний ключ на таблицу "planner";

- user: внешний ключ на таблицу "user".

2. **Таблица "planner":**

- Колонки (@Column):

- id: String первичный ключ;

- title: строка, заголовок планера;

- description: описание планера;

- notes: список связанных записей;

- owner: внешний ключ на пользователя из таблицы "user".

3. **Таблица "roles":**

- Колонки (@Column):

- id: String первичный ключ;

- roleName: перечисление RoleName;

- user: связь с таблицей "user" через роль пользователя.

4. **Таблица "users":**

- Колонки (@Column):

- id: String первичный ключ;

- firstName: строка, имя пользователя;

- lastName: строка, фамилия пользователя;

- email: строка, адрес электронной почты;

- password: строка, пароль пользователя;

- notes: список связанных записей;

- role: связь с таблицей "roles" через роль пользователя;

- planner: связь с таблицей "planner".

Для идентификаторов (id) моделей были использованы следующие аннотации:

@GeneratedValue(generator = "UUID") указывает, что значение идентификатора (ID) для модели будет генерироваться с использованием UUID (Универсального уникального идентификатора);

@GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator") определяет генератор с именем "UUID" и стратегию, основанную на использовании UUID;

@Column(name = "id", updatable = false, nullable = false) указывает, что поле "id" является столбцом базы данных с обязательным значением и не подлежит обновлению.

Выбор связей @ManyToMany и @JoinColumn в сущностях определен в соответствии с логикой приложения. Например:

- Использование FetchType.LAZY для полей, связанных с другими сущностями позволяет отложить загрузку данных, что полезно при работе с крупными объемами информации.

- Применение @JoinColumn в связке с @ManyToOne и @OneToOne гарантирует корректное сопоставление таблиц, задавая внешние ключи и соответствующие столбцы для связи.

В таблице "notes":

- Использование FetchType.LAZY для связи сущности Note с сущностью Planner позволяет отложить загрузку данных о планере до момента реальной необходимости, что способствует увеличению производительности приложения.

- Использование @JoinColumn с @ManyToOne определяет столбец "planner\_id" в таблице "notes" в качестве внешнего ключа к таблице "planner", обеспечивая корректное соединение между этими двумя сущностями.

В таблице "planner":

- Использование FetchType.EAGER для связи сущности Planner с сущностью Note позволяет немедленно загрузить список записей планера, что оправдано в случаях, когда вся информация о планере необходима при доступе к ней.

- Применение @OneToOne и @JoinColumn с указанием столбца "owner\_id" в качестве внешнего ключа к таблице "user" устанавливает связь между владельцем планера и пользователем, обеспечивая связь один к одному между сущностями.

В таблице "users":

- Использование FetchType.EAGER для связи сущности User с сущностью Note позволяет немедленно загрузить список записей пользователя, что удобно при работе с данными о пользователях.

- Применение @OneToOne и @JoinColumn с указанием столбца "role\_id" в качестве внешнего ключа к таблице "roles" определяет связь между пользователями и их ролями, обеспечивая уникальность и целостность данных в системе.

- Применение @OneToOne и @JoinColumn с указанием столбца "owner" в качестве внешнего ключа к таблице "planner" определяет связь между пользователем и планером, указывая на принадлежность планера к данному пользователю.

Такие решения по связям и использование различных типов FetchType и @JoinColumn способствуют эффективной организации взаимосвязей между таблицами в базе данных, учитывая особенности логики приложения и обеспечивая оптимальное управление данными в рамках функционала приложения "Управление планером".

В данном разделе подробно рассматривается установление связей между сущностями, используя аннотации @ManyToOne, @OneToOne, @ManyToMany и @JoinColumn для правильного маппинга данных между таблицами базы данных. Особое внимание уделяется выбору типов FetchType (EAGER, LAZY) для оптимизации загрузки данных в соответствии с потребностями приложения.

**2.3.2 Связи между сущностями и персистентностями объектов**

В представленных моделях приложения "Управление планером" используются различные аннотации, которые играют важную роль в определении структуры данных, связей между сущностями и персистентности объектов при работе с базой данных. Вот краткое описание наиболее часто встречающихся аннотаций:

1. @Entity:

Аннотация, которая помечает класс как сущность JPA (Java Persistence API), что позволяет использовать объекты класса для сохранения и извлечения данных из базы данных.

2. @Table:

Аннотация, позволяющая задать информацию о таблице, с которой связана сущность. Здесь можно указать имя таблицы, а также другие параметры, такие как имя схемы, к которой относится таблица.

3. @Id:

Аннотация, указывающая, что поле класса является первичным ключом в базе данных.

4. @GeneratedValue:

Аннотация, которая определяет способ генерации значений для первичного ключа. В данном случае используется generator = "UUID", что означает автоматическую генерацию значения ключа.

5. @Column:

Аннотация для задания свойств столбца в таблице базы данных. Здесь указываются имя столбца, его обязательность (nullable), длина строки и другие параметры.

6. @Enumerated:

Аннотация для указания, что поле класса отображается в столбец базы данных, содержащий перечислимое значение. EnumType.STRING используется для хранения строки с именем перечисления.

7. @ManyToOne и @OneToOne:

Аннотации, определяющие отношения между сущностями в базе данных (многие к одному и один к одному соответственно). fetch = FetchType.LAZY позволяет оптимизировать загрузку связанных сущностей в ленивом режиме.

8. @OneToMany:

Аннотация, указывающая на отношение один ко многим между сущностями. mappedBy указывает имя поля в объекте-владельце связи.

9. @JoinColumn:

Аннотация для указания столбца, который используется для задания внешнего ключа и связывания двух сущностей.

10. @NotEmpty, @Email:

Аннотации использованы для валидации данных, обеспечивая, что поля не будут пустыми и соответствуют определенному формату (например, адрес электронной почты).

11. @NoArgsConstructor, @AllArgsConstructor:

Аннотации Lombok, автоматически создающие конструкторы без параметров и с параметрами соответственно, уменьшая шаблонный код.

Использование этих аннотаций помогает определить структуру базы данных, отношения между сущностями, валидацию данных и упрощает разработку, обеспечивая эффективное взаимодействие с хранилищем данных и обеспечивая целостность информации.

**2.3.3 Настройки параметров приложения**

В этом разделе описываются настройки параметров приложения, связанные с базой данных, включая определение уникальных ограничений, правил каскадного обновления и удаления, а также другие аспекты, направленные на обеспечение целостности данных и эффективного управления информацией.

В рамках дипломного исследования представлен файл конфигурации application.yml (рис.5), который используется для настройки параметров приложения «Управление планером» на основе Spring Boot. Данный файл представляет собой удобный формат для определения настроек, влияющих на работу приложения. Вот подробное описание содержимого файла настроек:

1. `server: port: 8080`

- Указывает порт (8080), на котором будет запущен сервер.

2. `spring: datasource: driver-class-name: com.mysql.cj.jdbc.Driver`

- Задает название класса драйвера (com.mysql.cj.jdbc.Driver) для подключения к базе данных MySQL.

3. `spring: datasource: url: jdbc:mysql://localhost:3306/MyDBforDiploma`

- Указывает URL для подключения к базе данных MySQL (jdbc:mysql://localhost:3306/MyDBforDiploma), где localhost - хост, 3306 - порт и MyDBforDiploma - название базы данных.

4. `spring: datasource: username: root`

- Устанавливает имя пользователя (root) для доступа к базе данных.

5. `spring: datasource: password: "пароль"`

- Устанавливает пароль для доступа к базе данных.

6. `spring: jpa: generate-ddl: true`

- Включает генерацию DDL скриптов (create, update) при старте приложения.

7. `spring: jpa: hibernate: ddl-auto: update`

- Устанавливает стратегию обновления схемы базы данных в соответствии с моделью данных (update) при запуске приложения Hibernate.

8. `spring: jpa: open-in-view: false`

- Отключает режим открытия сессии для просмотра при работе с JPA, чтобы улучшить производительность и избежать проблем с ленивой загрузкой.

Файл application.yml является важной частью конфигурации проекта, где определены параметры сервера, настройки Spring Boot приложения, спецификации JPA и Hibernate для работы с базой данных. Правильная настройка этих параметров позволяет обеспечить стабильную работу приложения, обеспечивая эффективное взаимодействие с данными и оптимальную производительность системы.

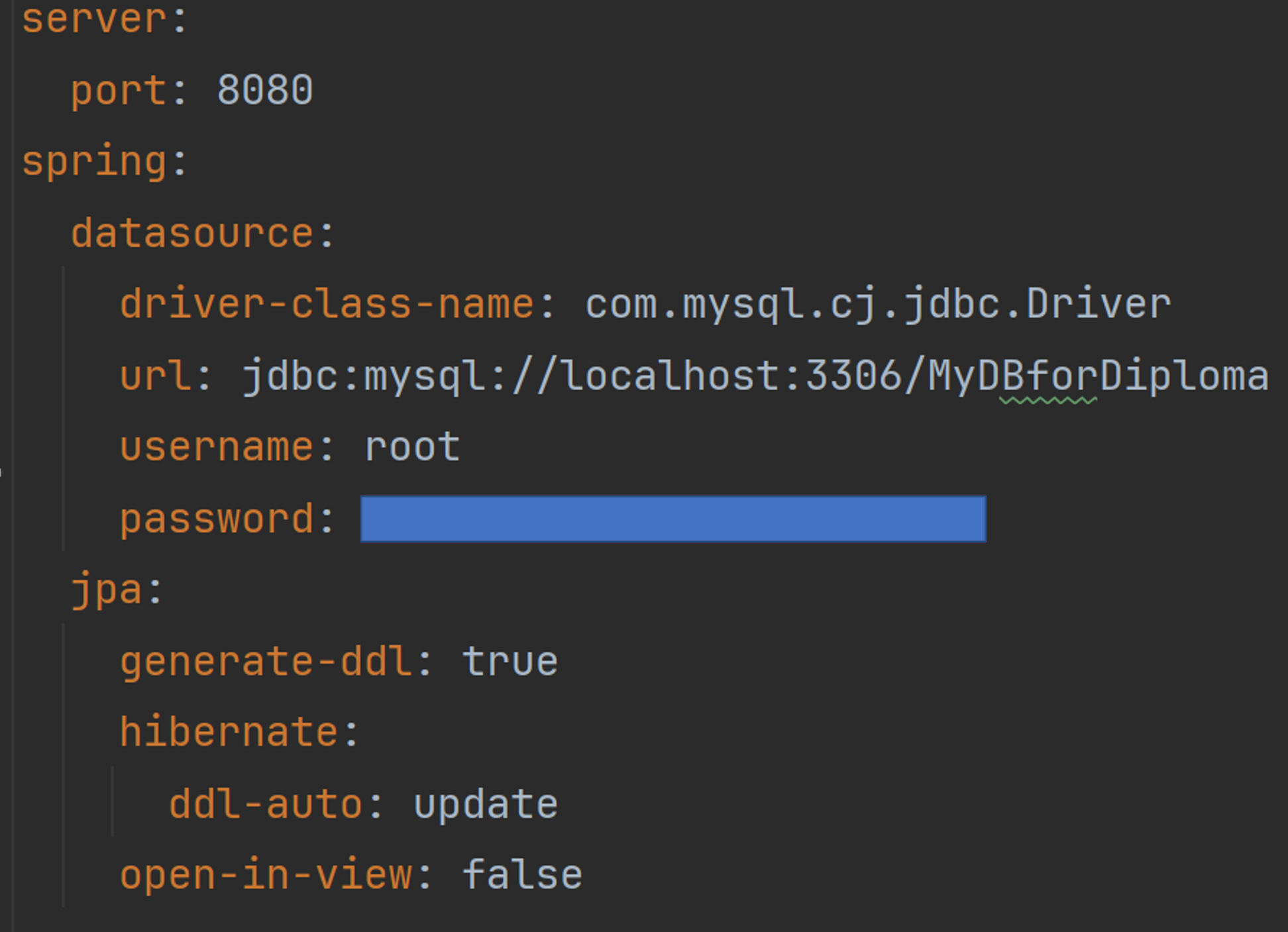


Рис.5 Файл application.yml

**2.3.4 Обработка данных и взаимодействие с репозиторием**

Данный раздел фокусируется на методах обработки информации, взаимодействии с базой данных через репозитории и сервисы, а также настройке каскадных операций для эффективного управления данными. Здесь освещаются аспекты выбора правильных стратегий загрузки данных и обработки запросов для обеспечения оптимальной производительности приложения.

В рамках организации базы данных в контексте проектирования приложения могут быть включены репозитории и сервисы, играющие ключевую роль в обработке данных и взаимодействии с хранилищем. Репозитории и сервисы интегрируются в архитектуру проекта следующим образом:

1. **Репозитории:**

Репозитории представляют собой классы, ответственные за доступ к данным из базы данных. Они содержат методы для выполнения CRUD-операций (Create, Read, Update, Delete) над объектами моделей.

Репозитории обычно аннотированы как интерфейсы или классы с аннотацией @Repository.

Репозитории, как основа доступа к данным из базы, содержат методы для осуществления операций создания, чтения, обновления и удаления объектов моделей. Аннотация @Repository над классами репозиториев позволяет Spring Framework автоматически создавать бины и управлять транзакциями, обеспечивая надежность и безопасность операций с данными.

В репозиториях определяются методы для поиска и фильтрации данных, сортировки результатов и других операций, связанных с работой с базой данных.

1. NoteRepository (рис.6):

- Репозиторий для работы с сущностью Note.

- Методы:

- `findAllByStatus(NoteStatus status)`: Возвращает список записей по указанному статусу.

- `findByUser(User user)`: Возвращает список записей, принадлежащих определенному пользователю.

- `findById(String id)`: Находит запись по уникальному идентификатору.

- `deleteById(String id)`: Удаляет запись по ее уникальному идентификатору.

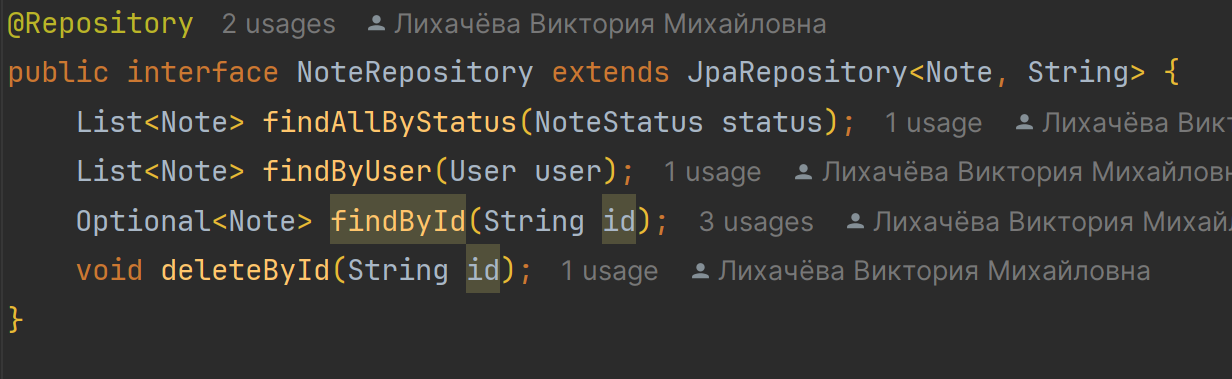


Рис.6 Интерфейс NoteRepository

2. PlannerRepository (рис.7):

- Репозиторий для работы с сущностью Planner.

- Методы:

- `findById(String id)`: Находит планер по уникальному идентификатору.

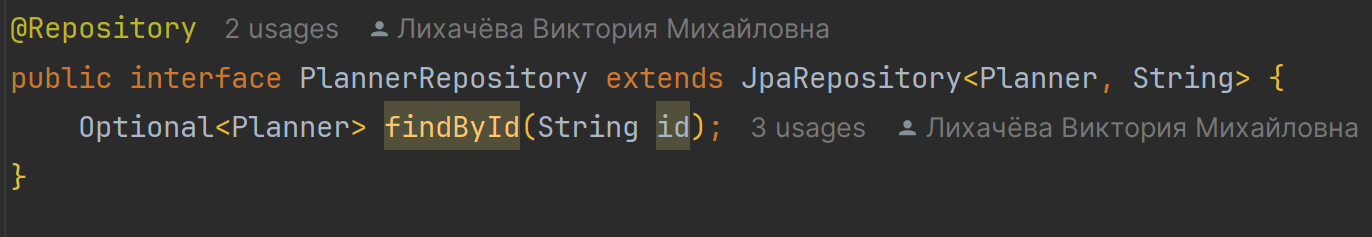


Рис.7 Интерфейс PlannerRepository

3. RoleRepository (рис.8):

- Репозиторий для работы с сущностью Role.

- Методы:

- `findByRoleName(RoleName roleName)`: Находит роль по указанному имени роли.

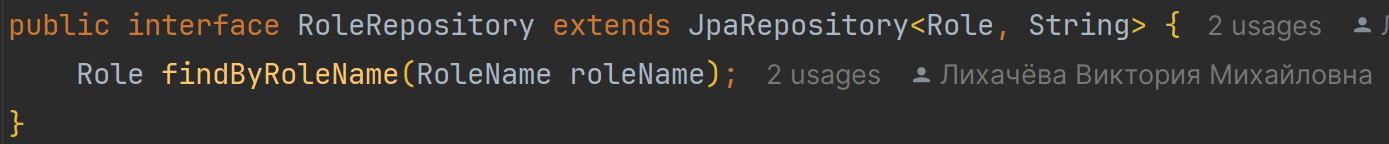


Рис.8 Интерфейс RoleRepository

4. UserRepository (рис.9):

- Репозиторий для работы с сущностью User.

- Методы:

- `findByEmail(String email)`: Находит пользователя по адресу электронной почты.

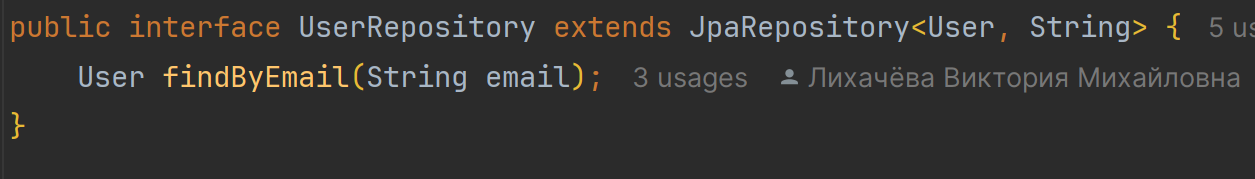


Рис.9 Интерфейс UserRepository

Каждый репозиторий расширяет JpaRepository, что обеспечивает стандартные методы доступа к данным, такие как сохранение, удаление, поиск и т. д. Использование интерфейсов репозиториев с JpaRepository в Spring Data JPA позволяет автоматически создавать реализации для операций CRUD (Create, Read, Update, Delete) без необходимости явного написания SQL-запросов. Такой подход значительно упрощает работу с базой данных и обеспечивает более высокую производительность разработки.

2. **Сервисы:**

Сервисы являются классами, в которых содержится бизнес-логика приложения. Они обеспечивают обработку данных, взаимодействие с репозиториями и другими компонентами системы.

Сервисы обычно аннотированы как @Service и представляют собой промежуточный уровень между контроллерами и репозиториями, где реализована основная логика приложения.

В сервисах определяются методы для выполнения сложных операций, агрегации данных, взаимодействия с внешними сервисами и других задач, связанных с обработкой информации.

Сервисы - это ключевые компоненты приложения, которые обеспечивают реализацию бизнес-логики и управляют взаимодействием с другими слоями приложения. Они часто организованы в виде интерфейсов и их соответствующих имплементаций, где интерфейсы объявляют контракты методов, а имплементации предоставляют конкретные реализации этих методов.

1. NoteService (рис.10, 11):

- Обеспечивает операции с записями (Note) в системе:

- Получение всех записей, получение записи по идентификатору, обновление записи, создание новой записи, удаление записи.

- Получение списка задач по статусу, получение записей для конкретного пользователя.

- Поиск пользователя по адресу электронной почты, инициализация, создание пользователя, сохранение пользователя, получение списка пользователей, аутентификация пользователя.

- `getAllNotes()`: Метод возвращает полный список всех записей в системе. Используется для отображения всех имеющихся записей.

- `getNoteById(String id)`: Данный метод позволяет получить конкретную запись по уникальному идентификатору. Обычно используется для получения информации о конкретной записи по ее идентификатору.

- `updateNote(String id, Note noteDetails)`: Метод обновляет информацию о конкретной записи на основе переданных данных. Позволяет вносить изменения в существующую запись.

- `createNote(Note note)`: Создает новую запись на основе переданных данных. Часто используется при добавлении новой записи в систему.

- `deleteNote(String id)`: Удаляет запись по указанному идентификатору из системы. Используется для удаления неактуальных записей.

- `getTasksByStatus(NoteStatus status)`: Возвращает список задач с указанным статусом. Полезен для фильтрации задач по их текущему статусу.

- `getNotesByUser(User user)`: Возвращает список записей, принадлежащих конкретному пользователю. Позволяет получить все записи пользователя для отображения или обработки.

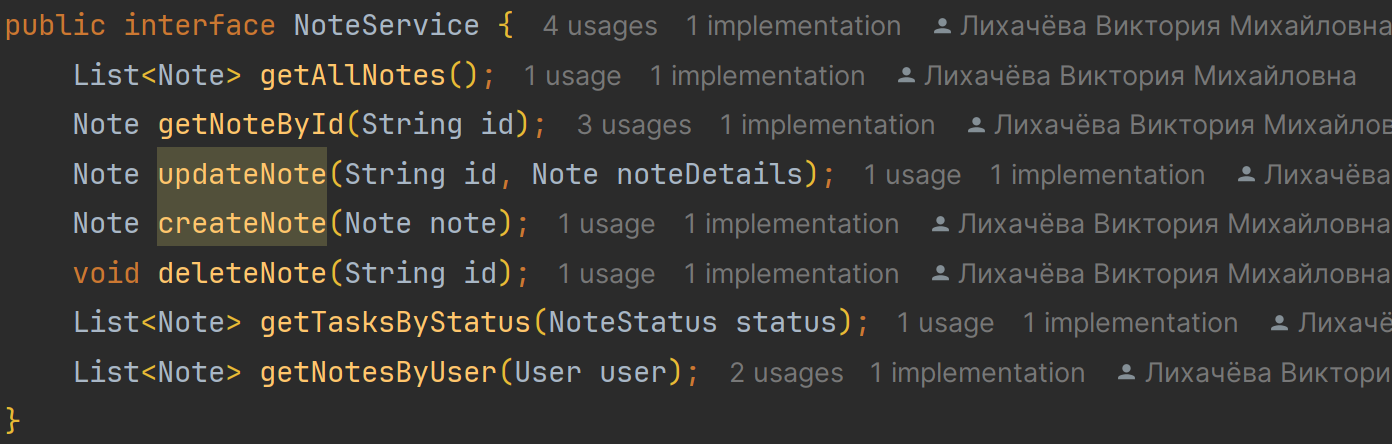


Рис.10 Интерфейс NoteService



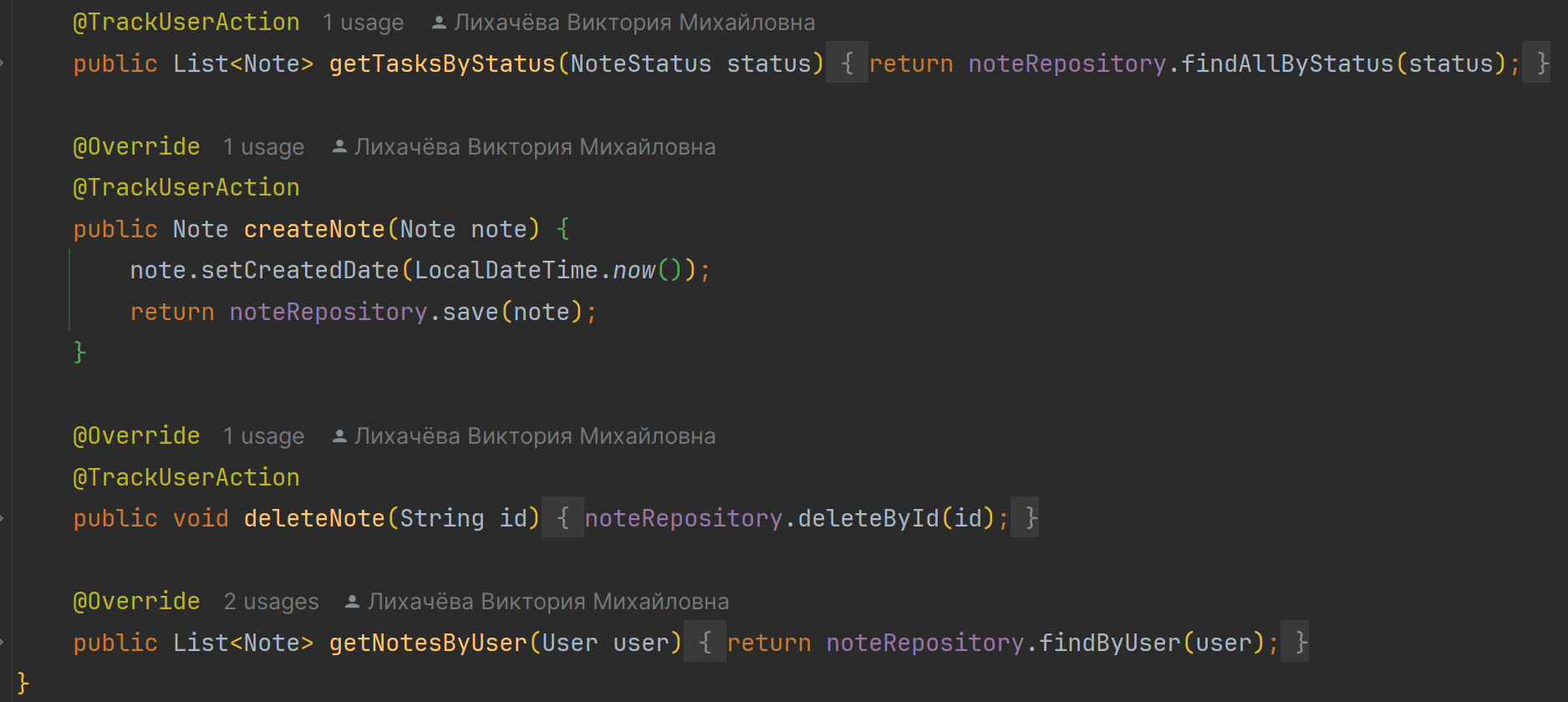


Рис.11 Сервис NoteServiceImpl

2. NotificationService (рис.12, 13):

- Отвечает за отправку уведомлений пользователям:

- Уведомление конкретного пользователя, отправка уведомления с заданным сообщением.

- `notifyUser(User user)`: Метод оповещает конкретного пользователя о событии или действии. Часто используется для отправки уведомлений пользователю.

- `sendNotification(String msg)`: Отправляет уведомление с указанным сообщением. Может быть использован для отправки различных уведомлений с заданным текстом.

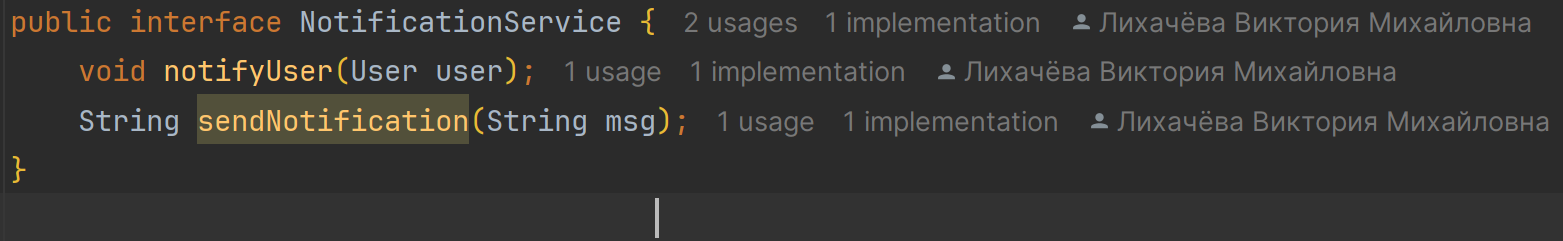


Рис.12 Интерфейс NotificationService

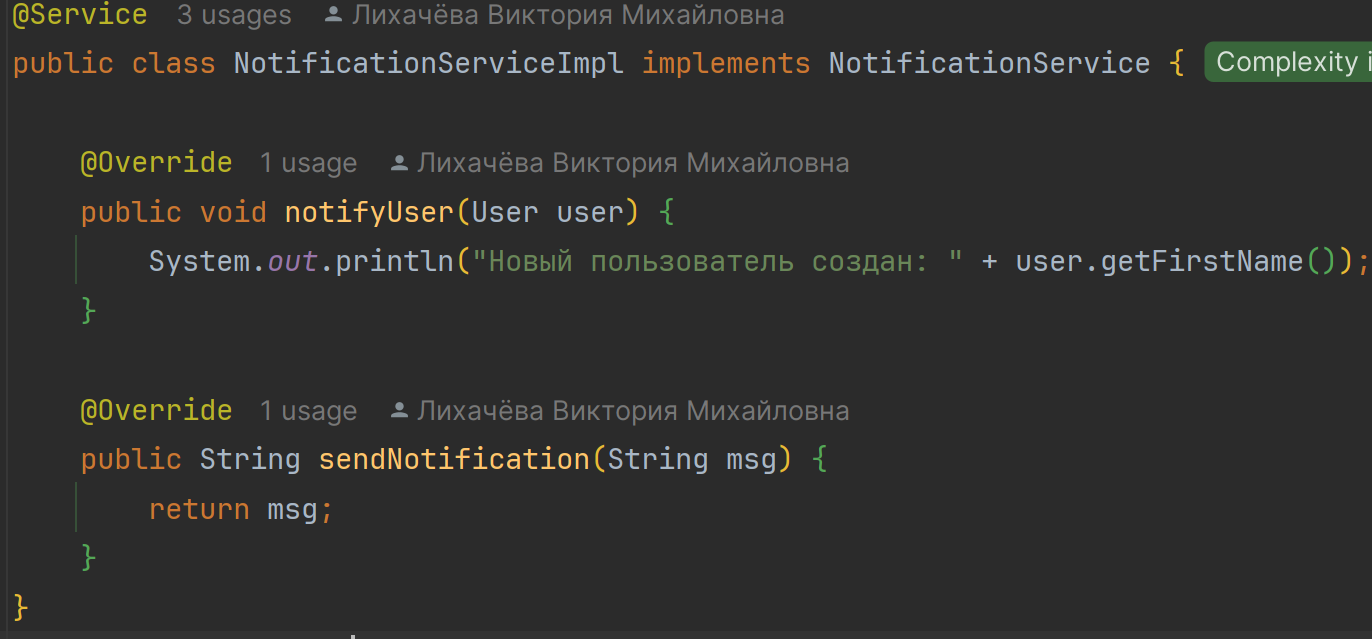


Рис.13 Сервис NotificationServiceImpl

3. PlannerService (рис.14, 15):

- Осуществляет управление календарем планирования (Planner) в приложении:

- Создание нового планировщика, получение всех планировщиков, получение планировщика по идентификатору, обновление информации о планировщике, удаление планировщика.

- `createPlanner(Planner planner)`: Создает новый планировщик на основе переданных данных. Используется для внесения новых планов в систему.

- `getAllPlanners()`: Возвращает список всех доступных планировщиков. Полезен для отображения всех имеющихся планировщиков.

- `getPlannerById(String id)`: Получает информацию о конкретном планировщике по его идентификатору. Используется для получения подробностей о конкретном планировщике.

- `updatePlanner(Planner planner)`: Обновляет информацию о планировщике на основе переданных данных. Позволяет вносить изменения в существующий планировщик.

- `deletePlanner(String id)`: Удаляет планировщик по указанному идентификатору. Используется для удаления ненужных планировщиков из системы.

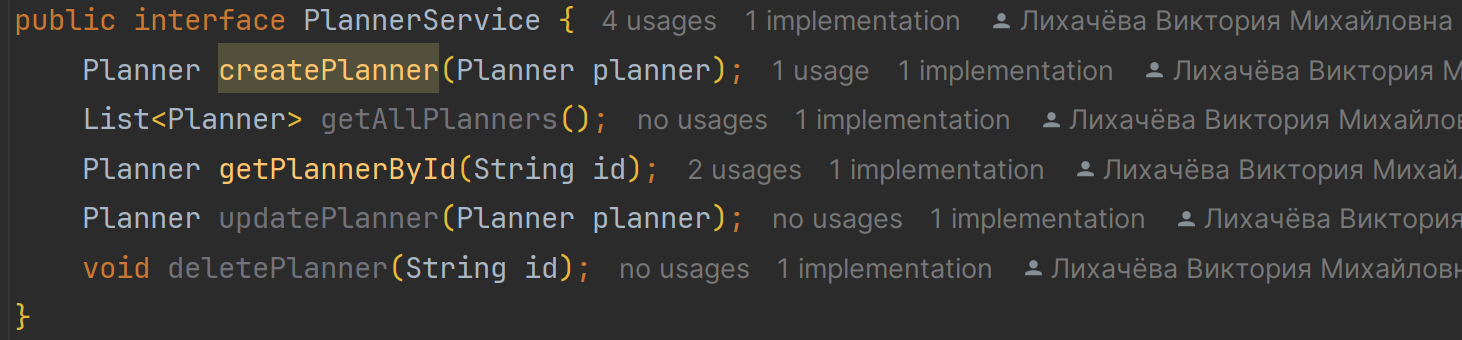


Рис.14 Интерфейс PlannerService

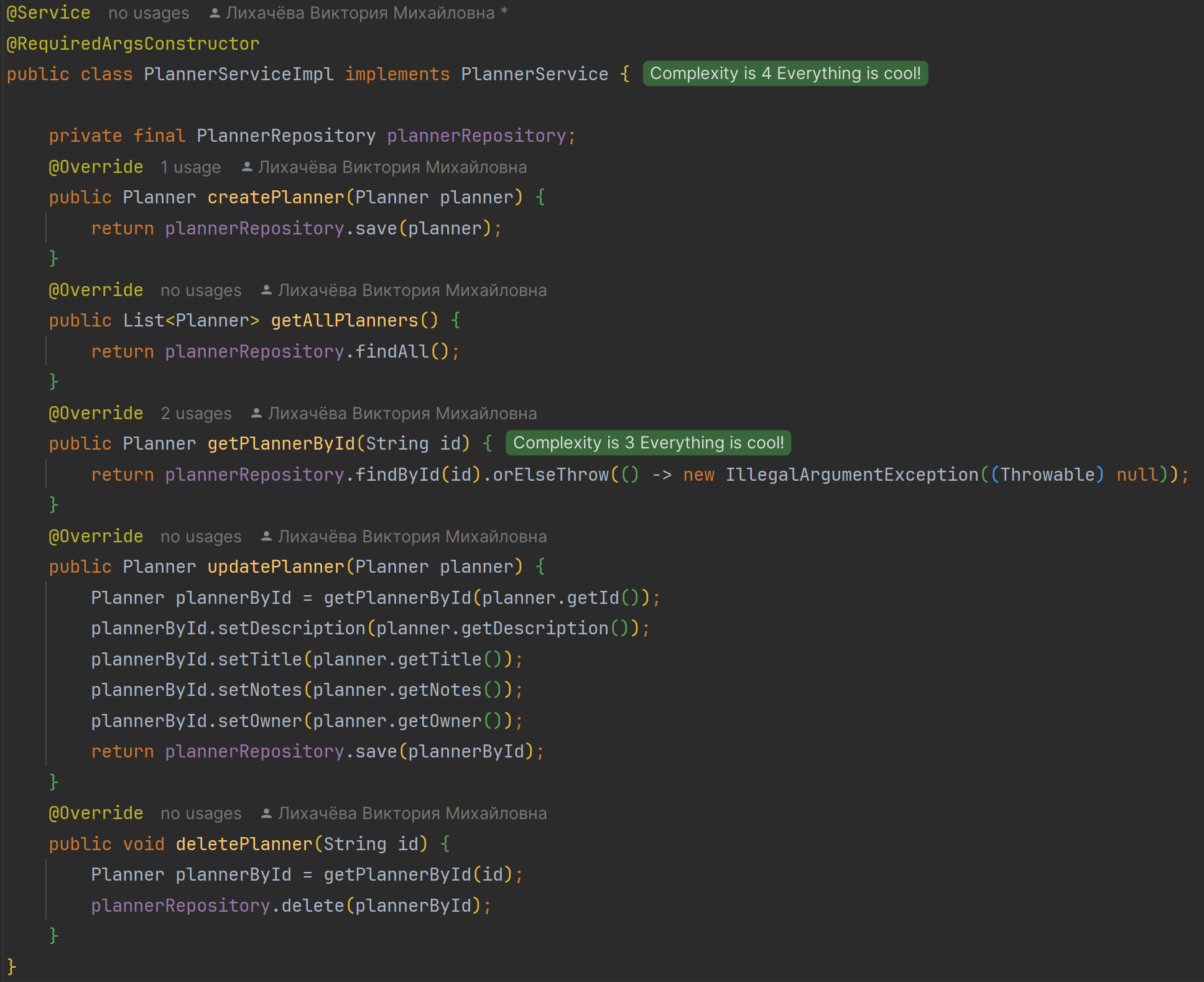


Рис.13 Сервис PlannerServiceImpl

4. UserService (рис.16, 17):

- Обеспечивает операции с пользователями:

- Поиск пользователя по адресу электронной почты, инициализация, создание пользователя, сохранение пользователя, получение списка пользователей, аутентификация пользователя.

- `findByEmail(String email)`: Поиск пользователя по адресу электронной почты. Используется для получения информации о пользователе по его email.

- `init()`: Метод инициализации для определенных настроек или данных в системе. Часто используется при инициализации приложения.

- `createUser(User user)`: Создает нового пользователя на основе переданных данных. Используется для регистрации новых пользователей.

- `saveUser(User user)`: Сохраняет информацию о пользователе. Часто используется для обновления данных пользователя.

- `getUsersList()`: Возвращает список всех пользователей в системе. Полезен для отображения списка всех пользователей.

- `authenticateUser(String email, String password)`: Проверка аутентификации пользователя по email и паролю. Используется для проверки подлинности учетных данных пользователя.

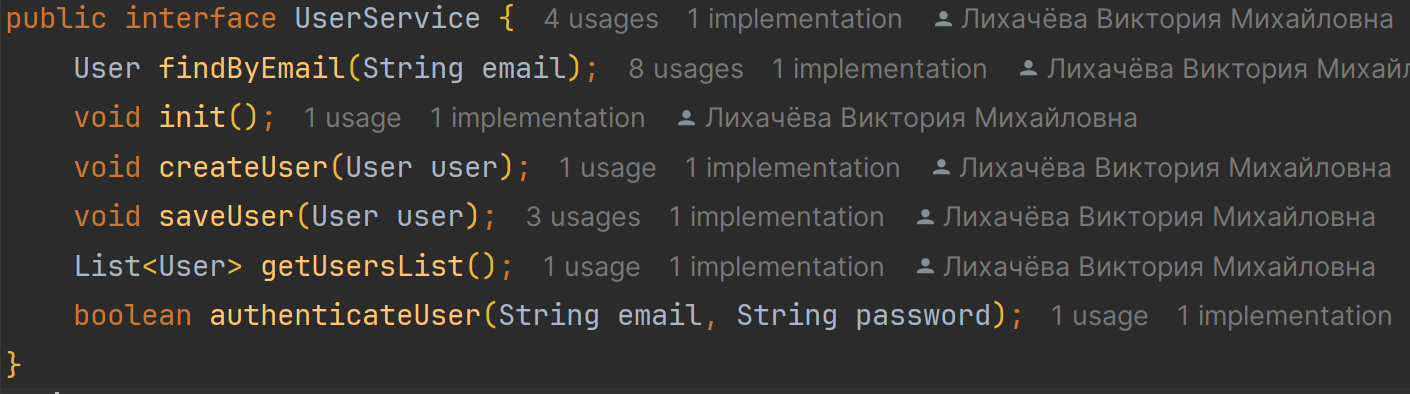
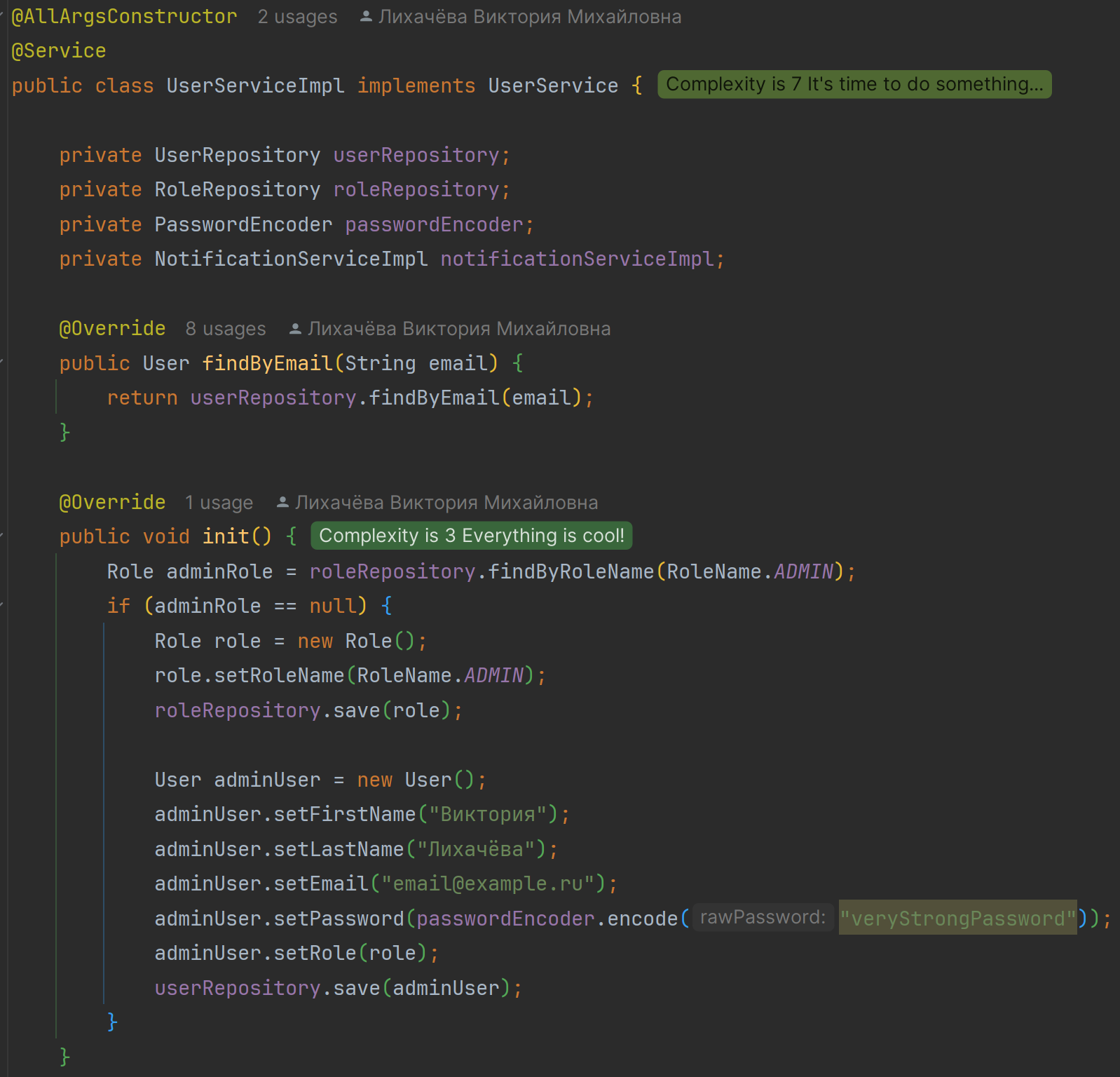


Рис.16 Интерфейс UserService







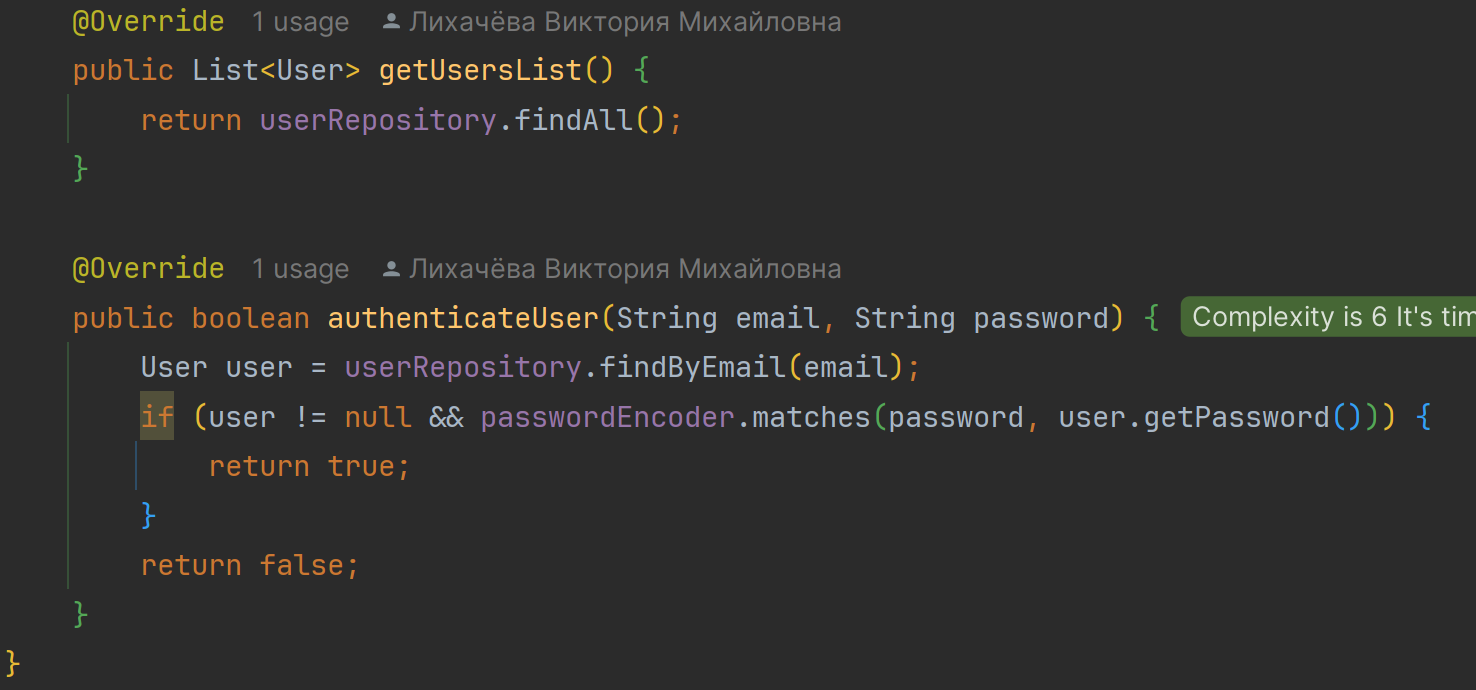


Рис.17 Сервис UserServiceImpl

Имплементации соответствующих интерфейсов предоставляют конкретную реализацию каждого метода, позволяя сервисам выполнять свои функции. Например, CustomUserDetailsService реализует интерфейс UserDetailsService для загрузки данных пользователя при аутентификации, а NoteServiceImpl предоставляет операции над записями с помощью методов, специфичных для работы с репозиторием NoteRepository.

Также стоит отметить, что каждая имплементация сервиса обычно аннотируется как @Service, что позволяет Spring Framework автоматически создавать экземпляры сервисов и внедрять их в другие компоненты. Разделение логики на интерфейсы и их имплементации помогает структурировать приложение, изоляции бизнес-логики и улучшению его расширяемости и тестируемости.

Интеграция репозиториев и сервисов в проектирование базы данных обеспечивает структурированную архитектуру приложения, где разделение задач и ответственностей позволяет эффективно управлять данными, обеспечивать безопасность и обеспечивать модульность кода.

Оптимальное взаимодействие компонентов приложения формирует основу для расширения и развития системы с соблюдением принципов надежности и масштабируемости.

**2.4 Реализация «Thymeleaf-контроллеров»**

Контроллер UserController (рис.18) представляет собой составную часть дипломного проекта, ответственную за управление пользователями в веб-приложении. Данный контроллер содержит набор методов, предназначенных для обработки запросов, связанных с аутентификацией, регистрацией и администрированием пользователей. Задачей контроллера является обеспечение безопасного взаимодействия между пользователями и системой, управление процессом аутентификации и регистрации пользователей, а также обработка запросов на отображение списка зарегистрированных пользователей и инициализацию системы.

Метод home() контроллера UserController предназначен для обработки GET-запроса на корневой URL. При вызове этого метода возвращается представление "home", позволяя пользователю увидеть главную страницу приложения.

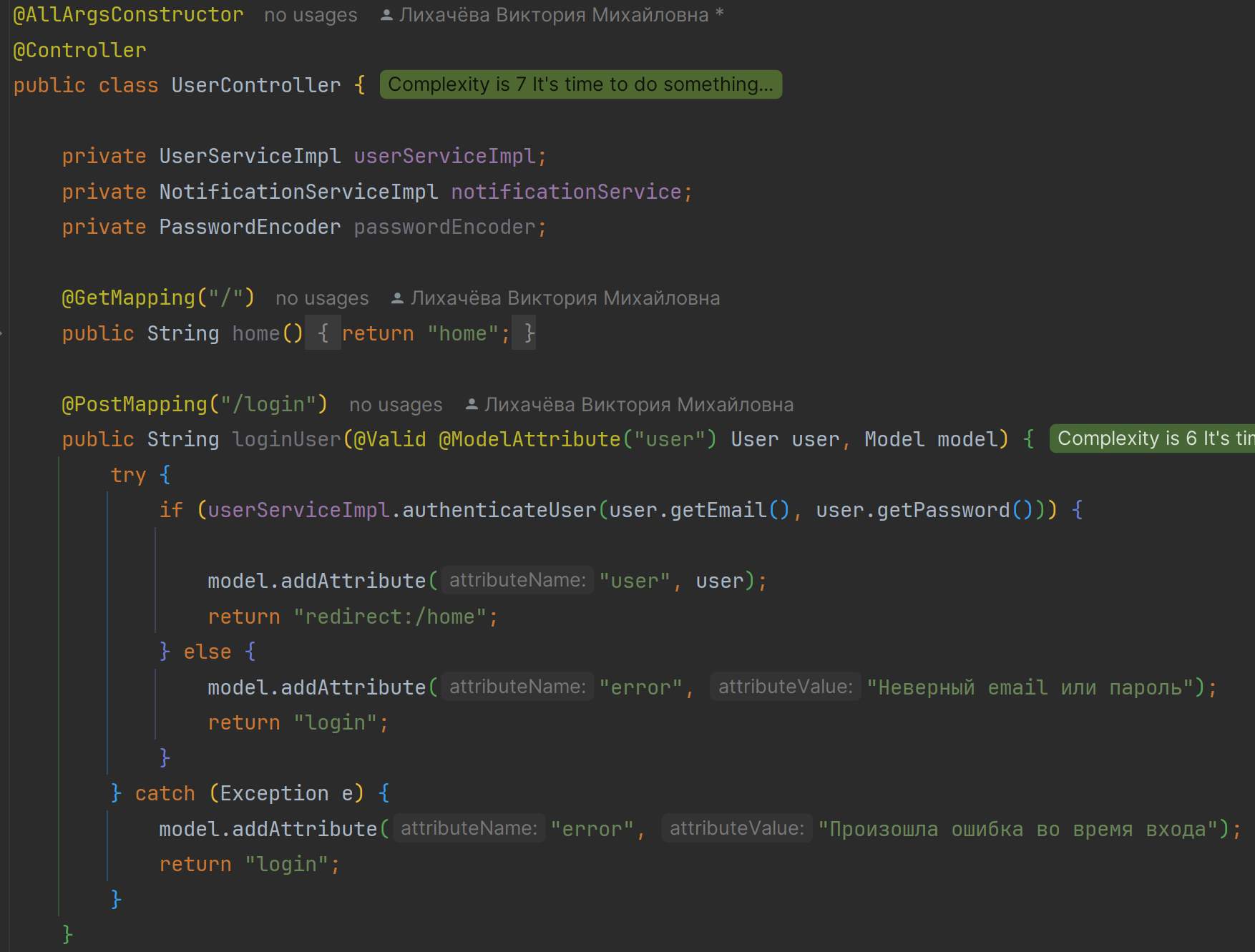
Метод loginUser() обрабатывает POST-запрос на URL '/login'. В процессе выполнения данного метода происходит аутентификация пользователя на основе предоставленных учетных данных. При успешной аутентификации пользователь перенаправляется на страницу '/home', в противном случае ему выводится сообщение о неверном вводе email или пароля, либо об иной ошибке при выполнении операции входа.

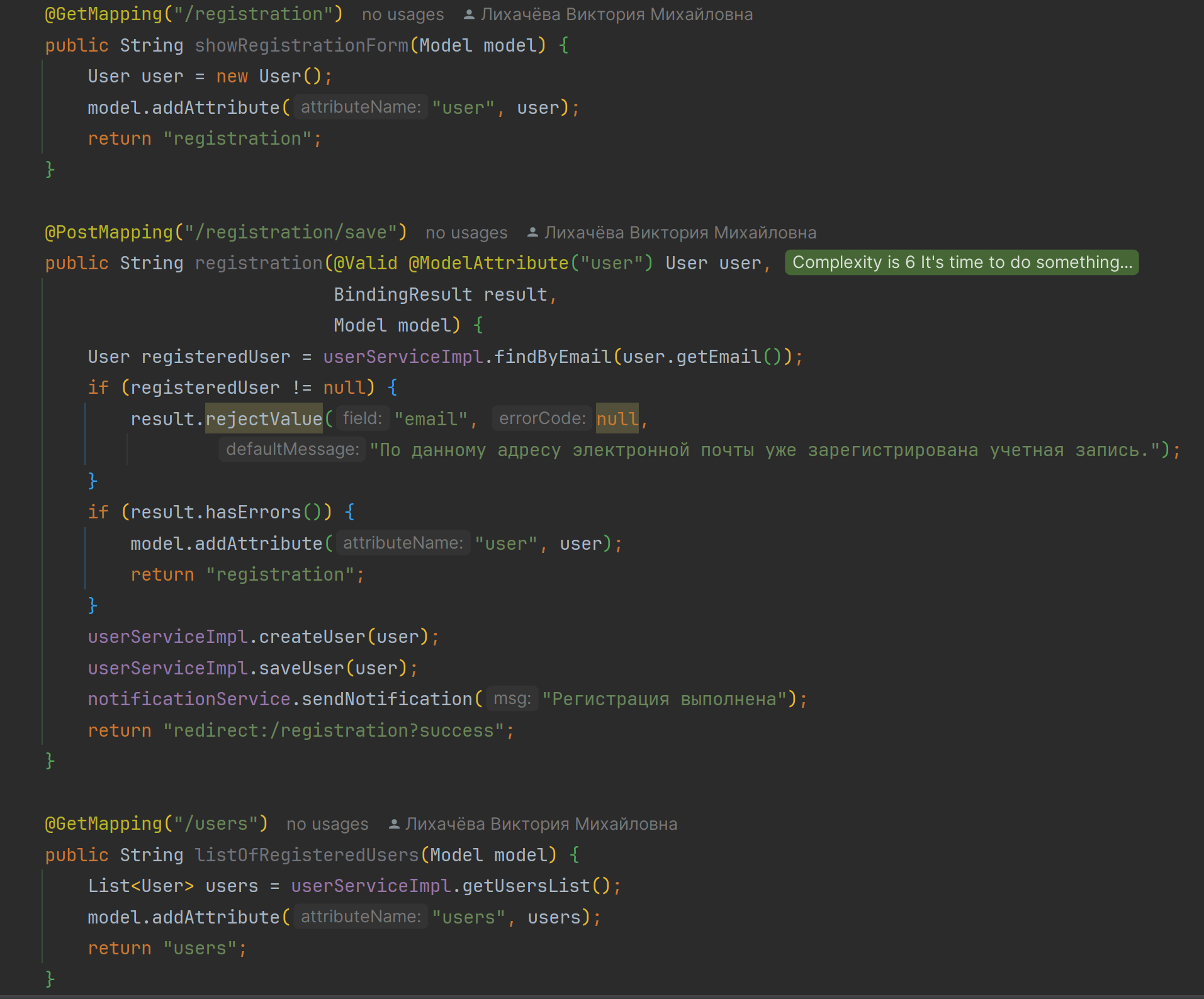
Метод showRegistrationForm() контроллера UserController используется для отображения формы регистрации нового пользователя при обработке GET-запроса на '/registration'. Данный метод предоставляет возможность пользователям заполнить необходимую информацию для регистрации в системе.

Метод registration() обеспечивает сохранение данных нового пользователя в системе при обработке POST-запроса на '/registration/save'. В ходе выполнения метода осуществляется проверка наличия пользователя с указанным email в системе. При выявлении ошибок валидации вводимых данных происходит отображение формы регистрации с сообщениями об ошибках. В случае успешной регистрации нового пользователя генерируется уведомление о завершении операции.

Метод listOfRegisteredUsers() предназначен для формирования списка зарегистрированных пользователей на странице '/users' при обработке GET-запроса. На данной странице отображаются данные о пользователях, предоставляя администратору системы возможность просмотра информации о зарегистрированных пользователях.

Метод \*\*init()\*\* контроллера \*\*UserController\*\* позволяет выполнить инициализацию системы путем создания пользователя с ролью ADMIN. При запросе POST на '/init' данный метод инициирует процесс добавления нового пользователя в качестве администратора системы, после чего возвращается сообщение об успешном создании учетной записи. Этот метод является важным для первоначальной настройки системы и обеспечивает наличие административного аккаунта для управления приложением.





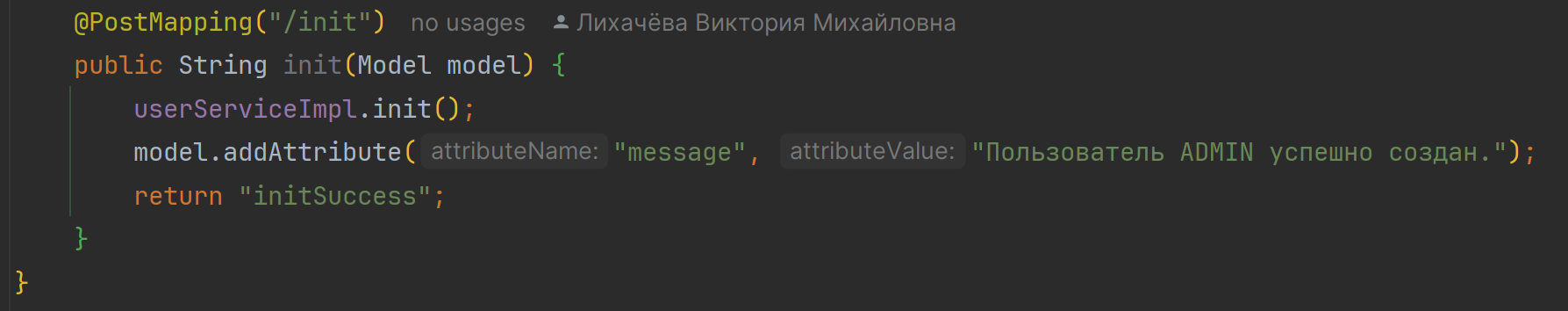


Рис.18. Контролер UserController

Контроллер NoteController (рис.19) является ключевым компонентом в структуре дипломного проекта, ответственным за управление записями пользователей и планировщиками в веб-приложении. Он содержит ряд методов, предназначенных для обработки запросов, связанных с созданием, отображением и управлением записями пользователей, а также планированием задач. Контроллер NoteController обеспечивает эффективное взаимодействие пользователя с системой, предоставляя функционал для создания новых записей, просмотра списка заметок и управления планировщиком.

Метод home() контроллера NoteController отвечает за обработку GET-запроса на URL '/home'. При вызове данного метода возвращается представление "home", позволяя пользователю перейти на главную страницу приложения и получить общий обзор доступных функциональных возможностей.

Метод getAllNotesByUser() обрабатывает GET-запрос на '/notes'. В процессе выполнения данного метода происходит получение всех заметок для текущего пользователя, аутентификация которого осуществляется с использованием контекста безопасности. Полученные заметки передаются в представление "notesList" для отображения пользователю.

Метод createPlanner() является обработчиком POST-запроса на '/createPlanner'. При вызове данного метода создается новый планировщик для текущего пользователя, что позволяет ему эффективно планировать свои задачи и обязанности. После создания планировщика происходит сохранение информации о нем и возврат пользователю соответствующего представления.

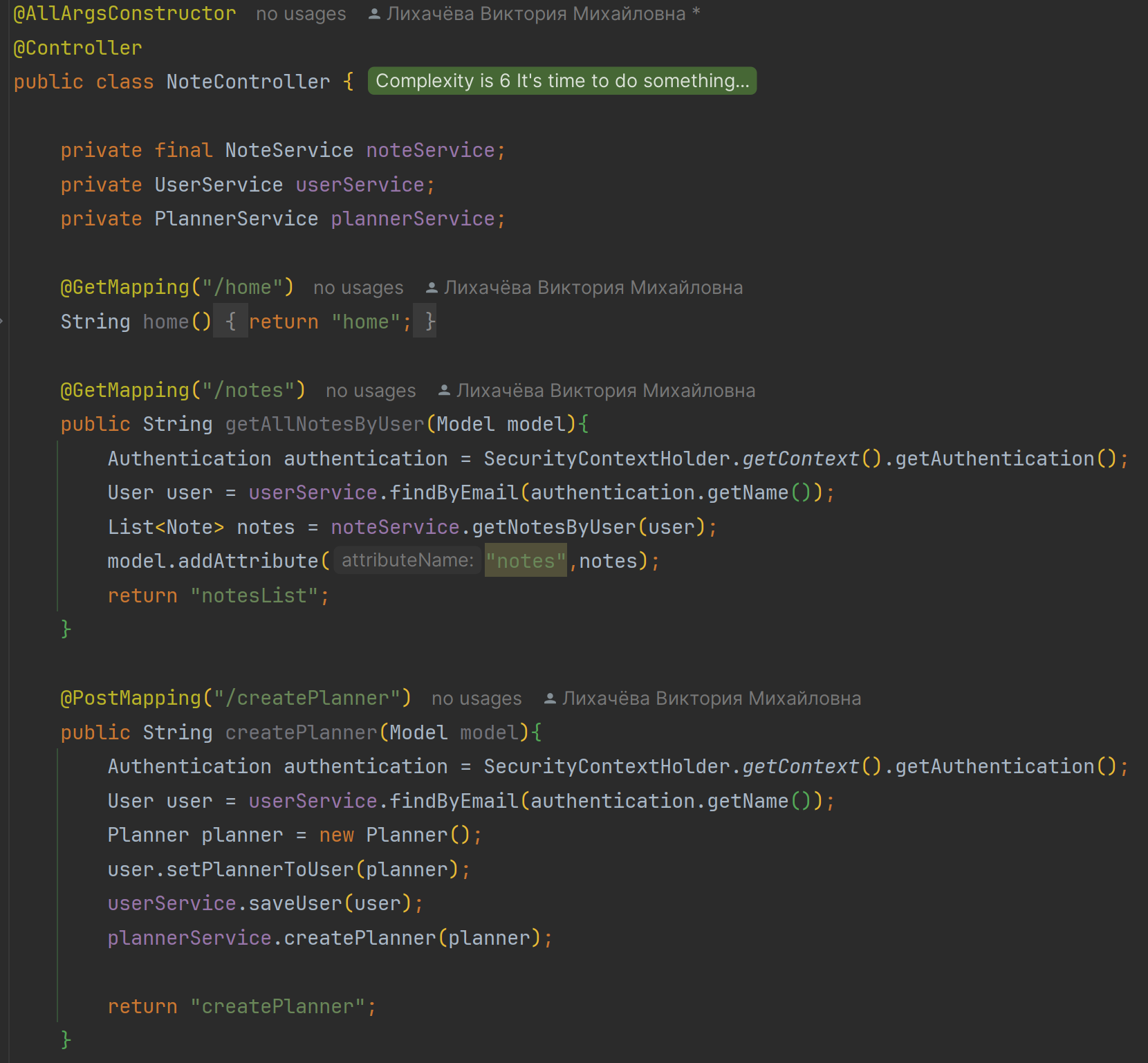
Метод createNote() предназначен для обработки POST-запроса на '/createNote'. При выполнении этого метода создается новая запись пользователя с заданными данными. После создания записи информация о ней сохраняется в базе данных для дальнейшего доступа и управления пользовательскими записями.

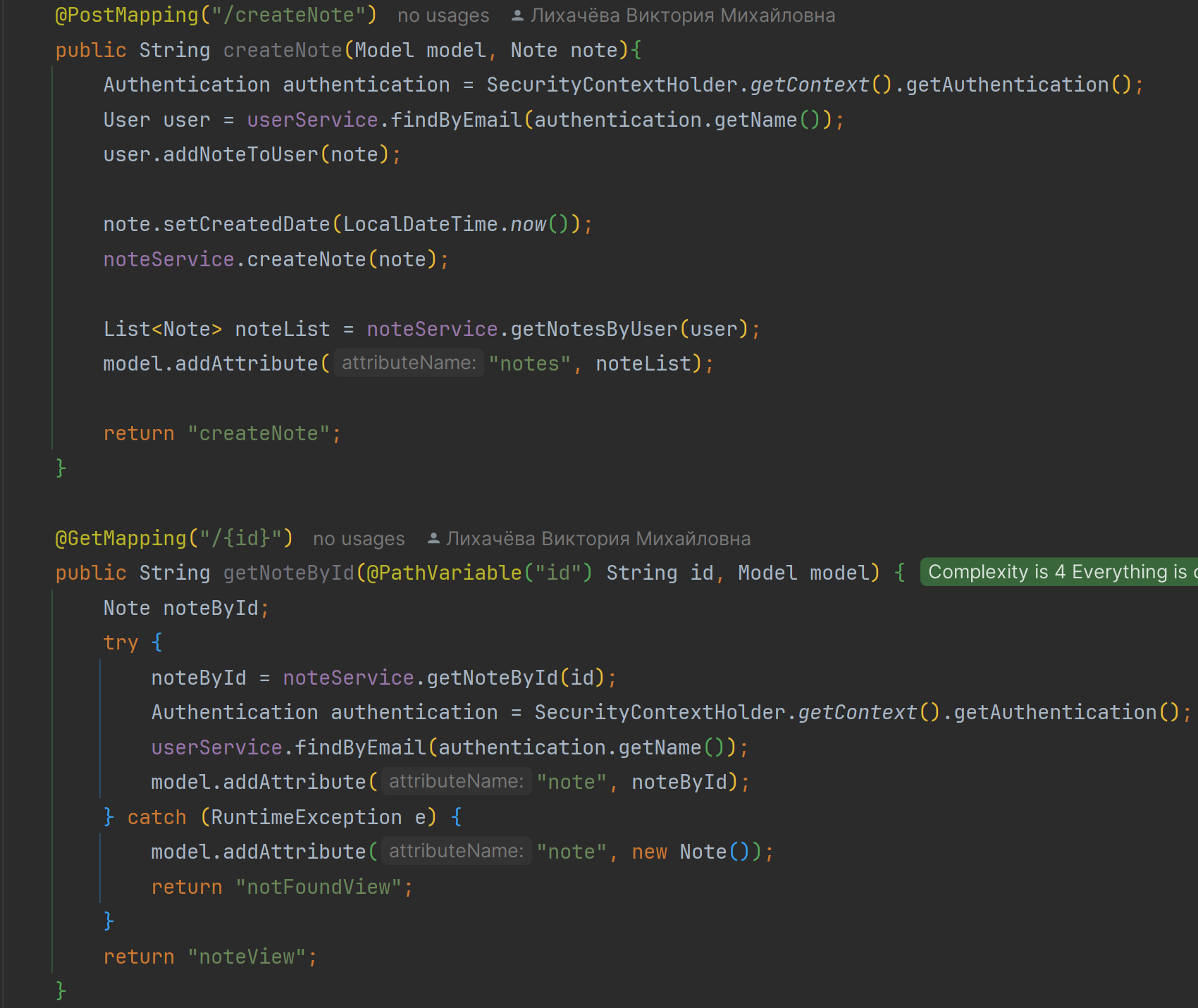
Метод getNoteById() контроллера NoteController обрабатывает GET-запрос на '/{id}'. В ходе выполнения данного метода осуществляется получение конкретной заметки по её идентификатору. Метод взаимодействует с сервисом заметок для извлечения нужной записи, а также проводит аутентификацию пользователя через контекст безопасности. Если заметка успешно найдена, она добавляется в модель для отображения на странице "noteView". В случае возникновения исключения метод возвращает страницу "notFoundView".

Метод updateNote() контроллера NoteController обрабатывает PUT-запрос на '/{id}'. При вызове данного метода происходит обновление существующей заметки. Метод принимает в теле запроса обновленную информацию о заметке и идентификатор необходимой заметки. После проверки наличия заметки и аутентификации пользователя метод осуществляет обновление данных о заметке в базе данных с использованием соответствующего сервиса. После успешного обновления заметки происходит перенаправление на страницу "noteView".

Метод deleteNote() контроллера NoteController обрабатывает DELETE-запрос на '/{id}'. При вызове данного метода происходит удаление конкретной заметки по её идентификатору. Заметка извлекается по идентификатору с помощью сервиса заметок, а затем удаляется из списка заметок пользователя. После удаления заметки происходит сохранение обновленных данных о пользователе и заметках, а затем перенаправление на страницу "noteView" с обновленным списком заметок.

Метод getTasksByStatus() контроллера NoteController обрабатывает GET-запрос на '/status/{status}'. Данный метод предназначен для получения списка заметок с заданным статусом. При выполнении метода извлекаются заметки с указанным статусом с использованием сервиса заметок. Аутентификация пользователя осуществляется через контекст безопасности, после чего список заметок по указанному статусу добавляется в модель для отображения на странице "statusNotesView".





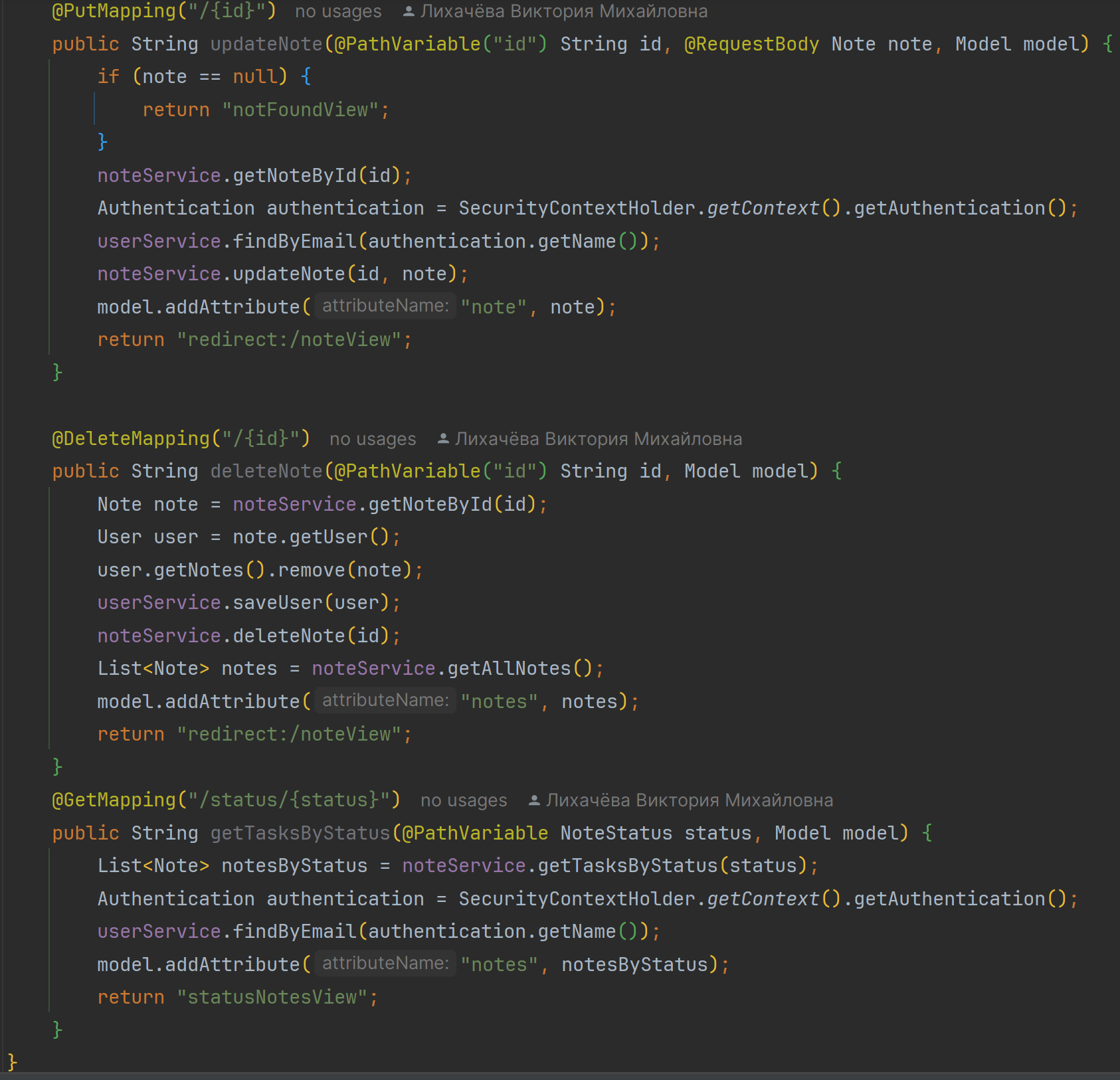


Рис.19 Контролер NoteController

**2.5 Разработка конфигурации безопасности web-приложения**

Класс CustomUserDetailsService (рис.18) является сервисом, реализующим интерфейс UserDetailsService в рамках Spring Security для аутентификации и авторизации пользователей. Он ответственен за загрузку информации о пользователе по его имени пользователя (email), а также преобразует роли пользователя в соответствующие права доступа.

1. Метод loadUserByUsername(String username):

- Метод загружает данные пользователя по его имени пользователя (email) из UserRepository.

- Параметры:

- `String username`: Имя пользователя (email), по которому происходит поиск пользователя.

- Возвращаемое значение: UserDetails, содержащий информацию о пользователе и его правах доступа.

- Исключение: UsernameNotFoundException выбрасывается, если пользователь с указанным email не найден в системе.

2. Приватный метод private Collection mapRolesToAuthorities(Collection roles):\*\*

- Преобразует роли пользователя в объекты SimpleGrantedAuthority, представляющие разрешения доступа.

- Параметры:

- `Collection roles`: Список ролей пользователя.

- Возвращаемое значение: Коллекция SimpleGrantedAuthority, содержащая права доступа на основе ролей пользователя.

3. Конструктор CustomUserDetailsService(UserRepository userRepository):

- Описание: Конструктор класса, принимающий UserRepository в качестве зависимости для доступа к данным пользователей.

- Параметры:

- `UserRepository userRepository`: Репозиторий, через который осуществляется доступ к информации о пользователях.

- Действие: Инициализирует поле userRepository класса CustomUserDetailsService.

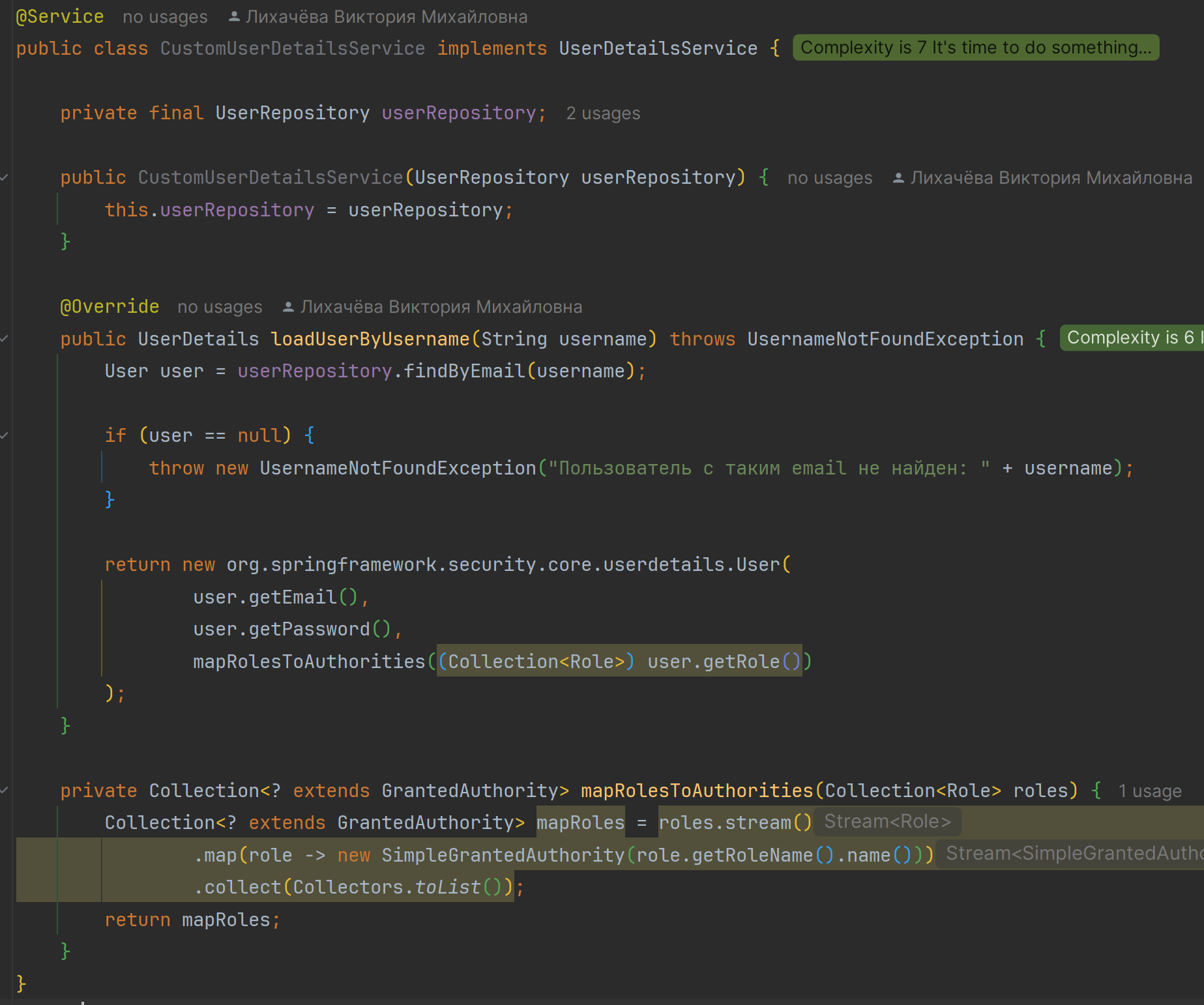


Рис.18 Сервис CustomUserDetailsService

Класс CustomUserDetailsService играет важную роль в процессе аутентификации пользователей в приложении, загружая данные о пользователе из репозитория, создавая объект UserDetails для аутентификации и авторизации. Метод mapRolesToAuthorities обеспечивает преобразование ролей пользователя в разрешения доступа, необходимые для управления безопасностью в системе. Данный класс является существенной частью разработки безопасного и защищенного приложения, обеспечивая аутентификацию пользователей и управление доступом к ресурсам.

Класс SecurityConfiguration (рис.19) играет ключевую роль в определении параметров безопасности приложения, включая управление доступом, аутентификацию и авторизацию пользователей. Этот класс взаимодействует с CustomUserDetailsService, ответственным за загрузку информации о пользователях и их ролях для процесса аутентификации.

1. Метод configure(AuthenticationManagerBuilder auth):

- Данный метод позволяет настраивать аутентификацию пользователей и выбирать источники для проверки подлинности.

- В методе применяется объект userDetailsService, который является интерфейсом для загрузки информации о пользователях, а также passwordEncoder для шифрования паролей.

- Выбор конфигурации:

Использование userDetailsService связывает SecurityConfiguration с CustomUserDetailsService, обеспечивая доступ к данным пользователей, в то время как использование passwordEncoder гарантирует безопасное хеширование паролей.

2. Метод passwordEncoder():

- Этот метод создает и возвращает экземпляр PasswordEncoder для шифрования паролей пользователей.

- Здесь используется BCryptPasswordEncoder, предоставляющий сильное и безопасное хеширование паролей.

- Выбор конфигурации:

Использование BCryptPasswordEncoder обеспечивает высокий уровень безопасности при хранении паролей в системе.

3. Метод securityFilterChain(HttpSecurity builder):

- Данный метод определяет конфигурацию безопасности для HTTP-запросов и URL-адресов.

- Здесь прописываются правила доступа к различным URL-адресам, настройки формы входа и выхода из системы.

- Выбор конфигурации:

Указание различных URL-адресов и их доступности, настройки формы входа и выхода позволяют регулировать доступ пользователей и обеспечить безопасность приложения.

Связь с CustomUserDetailsService:

- В SecurityConfiguration используется объект userDetailsService из CustomUserDetailsService, обеспечивая интеграцию классов для обеспечения аутентификации и авторизации пользователей.

- UserDetailsService в SecurityConfiguration внедряется через механизм @Autowired, обеспечивая доступ к данным пользователей.

- В методе configure используется userDetailsService для определения правил аутентификации пользователей.

- PasswordEncoder из CustomUserDetailsService используется для безопасного шифрования паролей пользователей, обеспечивая надежность хранения учетных данных.

В SecurityConfiguration установлены следующие правила доступа:

- URL `/registration/\*\*` доступен всем пользователям без ограничений.

- URL `/index` открыт для всех без требований к аутентификации.

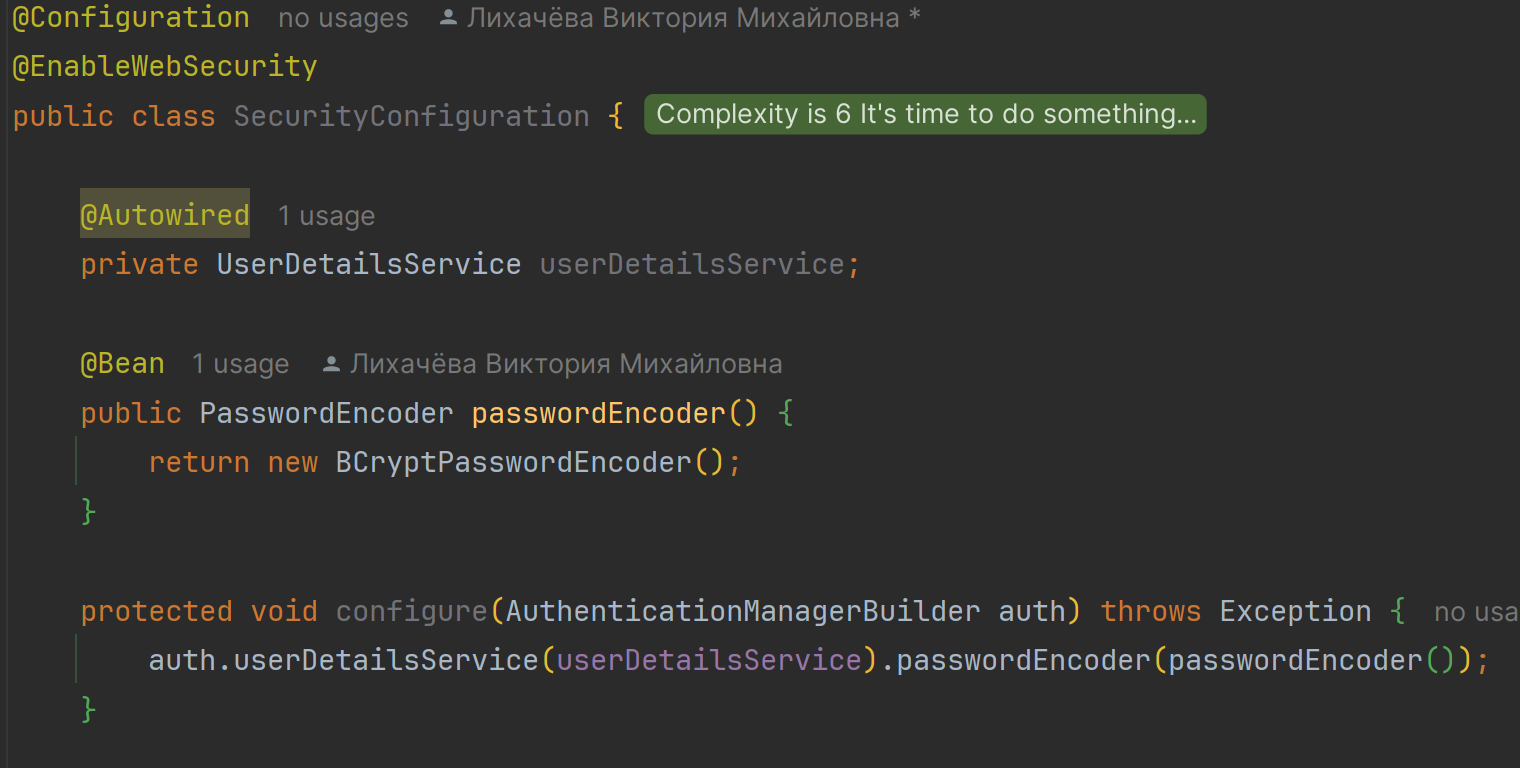
- URL `/users` и `/\*\*` доступны только пользователям с ролью "ADMIN".

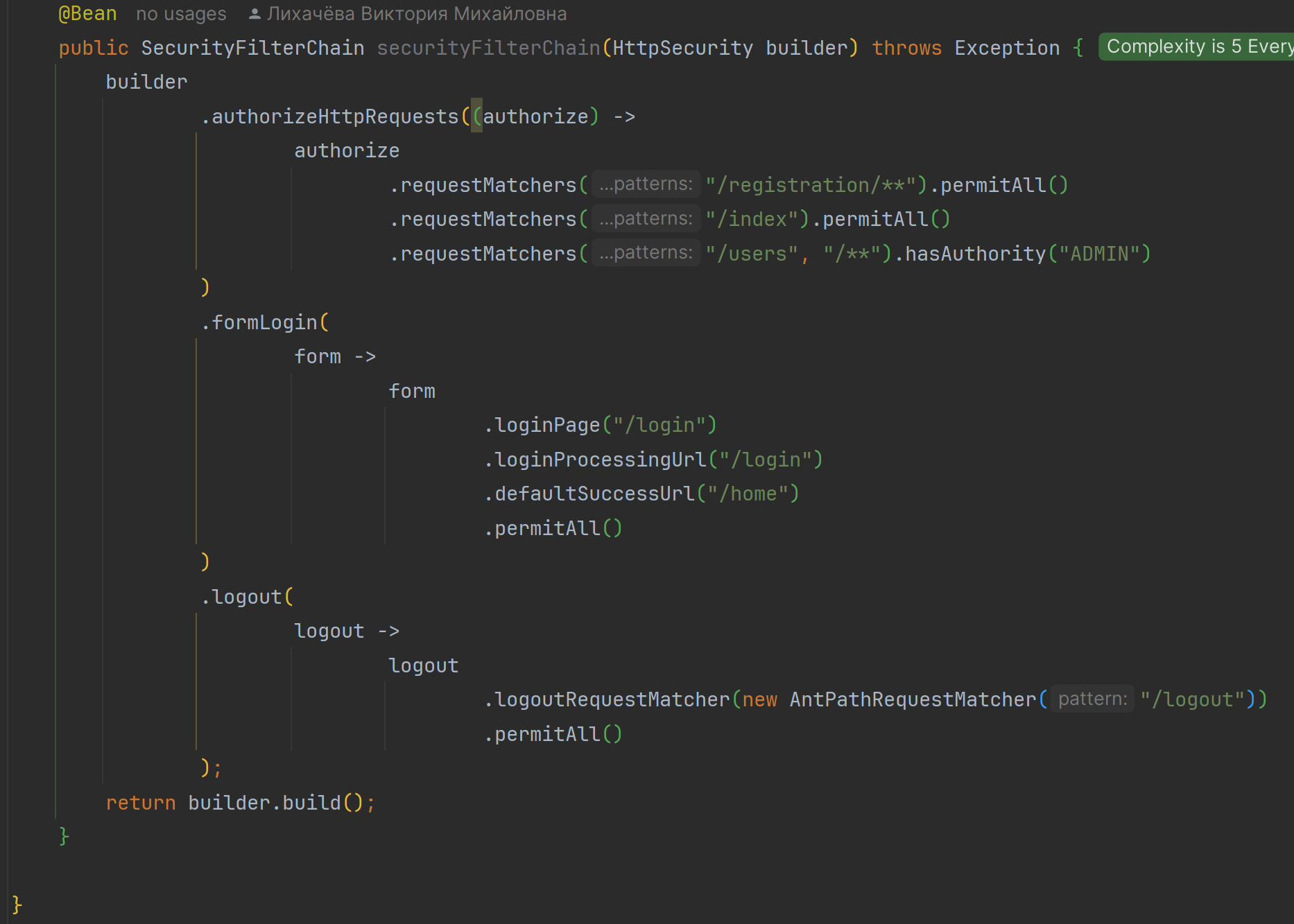
Система входа и выхода настроена следующим образом:

- Форма входа (/login) предоставляет возможность ввода учетных данных и прохождения аутентификации.

- При успешной аутентификации пользователь перенаправляется на страницу `/home`.

- Функционал выхода из системы (/logout) позволяет пользователям завершить сеанс работы.





**Рис.18 Класс SecurityConfiguration**

**Итак, учитывая детальные конфигурации методов в \*\*SecurityConfiguration\*\* с использованием \*\*CustomUserDetailsService\*\*, гарантируется надежная настройка безопасности приложения, защита данных, контроль доступа пользователей и обеспечение безопасности взаимодействия пользователя с приложением.**

**Заключение**

**Список литературы**

1. **Spring в действии,** Крейг Уоллс (6-е издание, 2022 год).
2. Учебный материал Гикбрейнс.
3. Статьи и видео-материалы из сети интернет.