Geekbrains

**Разработка высокоэффективного бекенд-сервиса для управления конспектами с использованием языка программирования Java и фреймворка Spring.**

Программа:

Разработчик Веб

разработка на Java

Чертоновский В.О.

Барнаул

2024

Содержание

Введение

Глава 1 Что такое Веб-приложение

1.1 Что такое веб-приложение его особенности

1.2 Отличие веб-приложения и веб-сайта.

1.3 Виды веб-приложений

1.4 Из чего состоит веб-приложение

1.5 Из чего состоит серверная часть веб-приложения(Backend)

Глава 2 Описание используемых программных средств

2.1 Язык программирования Java

2.2 Фреймворк.

2.3 База данных

2.4 Архитектура серверной части приложения

2.5 Spring Framework

2.6 Spring Container

2.7 Spring Aspect-oriented Programming (AOP)

2.8 Spring MVC

2.9 Spring Security

2.10 Spring Data Access

2.11 Spring Boot

2.12 Swagger REST

2.13 JUnit и Mockito

2.14 Project Lombok

2.15 Maven

Глава 3. Предпроектная стадия

3.1 Порядок проведения работ

3.2 Определение функционала и архитектуры приложения

Глава 4 Разработка серверной части

4.1 Создание проекта

4.2 Создание основных объектов Entity

4.3 Создание интерфейса репозиториев Spring Repository

4.4 Создание сервис-слоя

4.5 Создание сущностей DTO

4.6 Создание маппера

4.7 Создание контроллера

4.8 Регистрация и авторизация с помощью Spring Security

4.8.1 Реализация слоя доступа к данным и сервисного слоя

4.8.2 Добавление контроллеров

4.8.3 Настройки безопасности

4.9 Документирование SpringBoot API с помощью Swagger

Заключение

Список используемой литературы

**Введение**

В связи с тем, что в многие занимаются обучением и само обучение каким либо новым для себя делом, я решили создать веб приложение, в котором можно будет записывать свои конспекты как в книгу конспектов.

Приложение "Дневник конспектов" представляет собой удобный инструмент для записи и редактирования конспектов, сортировки их по темам. С его помощью пользователь может записывать свои конспекты, который состоит: Названия, даты создания, основных тезисов и вопросов, Подробного объяснения понятий, кратких выводов или самых важных мыслей по конспекту.

Основная функциональность приложения включает в себя:

**Записи конспектов**: Пользователь может создавать конспекты.

**Редактировать**: Пользователь может вносить изменения в конспект и сохранять их.

**Сортировка по темам**: Конспекты могут сортироваться по темам.

**Конфиденциальность и безопасность:** Для защиты личных данных пользователей приложение обеспечивает возможность установки пароля.

Я думаю, что ведение конспектов помогает в познание новой темы и лучшем запоминании, что позволяет дневнику конспектов быть отличным помощником при изучении нового.

**Глава 1 Что такое Веб-приложение**

**1.1 Что такое веб-приложение его особенности**

Веб-приложение – это программа с определенными функциями, которая работает через браузер. Его можно использовать как во внутренних бизнес-процессах компании, так и в работе с клиентами. Во втором случае оно позволяет пользователям заказывать товары, оплачивать покупки, бронировать номера, оставлять отзывы и др.

К основным преимуществам веб-приложений можно отнести следующие пункты:

* Взаимодействие с любой операционной системой и браузером;
* Не требуют одобрения сторонней платформы для выхода на рынок, как в случае с мобильными приложениями;
* Не требуют установки на устройство — достаточно знать URL-адрес приложения, чтобы взаимодействовать с ним;
* Нет необходимости в подписке или покупки лицензии, что очень выгодно для компаний.

В принцип работы веб-приложения состоит в том, что пользователь формирует запрос, данный запрос отправляется на сервер, где хранятся все необходимые данные, при получении запроса сервером, запрос обрабатывается на сервере после чего дает ответ пользователю, пользователь получает ответ и видит результаты своего запроса.

**1.2 Отличие веб-приложения и веб-сайта.**

Веб-сайт это одна или несколько веб страниц, которые предоставляют пользователю информационный контент, которые как правило статичны, пользователь не может взаимодействовать с ними, изменять данные страницы. Обычно данные сайты используются для демонстрации информации пользователям, чаще всего данные сайты не требуют аутентификацию, в следствии чего вся информация находится в общем доступе.

Веб-приложение это веб-сайт с расширенным функционалом, который включает в себя интерактивные элементы, которые решают определенные задачи пользователя. Пользователь может взаимодействовать с контентом и проводить разные манипуляции, в пределах доступного функционала вложенного в данные сайты. Важным моментом является авторизация пользователя, для хранения информации о взаимодействии пользователя с сайтом.

**1.3 Виды веб-приложений**

SPA (Single Page Application) — одностраничное интерактивное приложение, которое не требует перезагрузки основной страницы во время своей работы. В отличие от традиционных веб-приложений, они работают быстрее, так как вся логика приложения реализуется в браузере, а не на сервере.

MPA (Multi Page Application) — многостраничное веб-приложение. Оно позволяет пользователю переходить между отдельными страницами при совершении запроса. У таких ресурсов, как правило, более сложный интерфейс и большое количество дополнительных ссылок. При каждом новом запросе, например, при оплате продукта, у пользователя откроется новая страница.

PWA (Progressive Web Application) — прогрессивное веб-приложение. Разрабатывается с помощью специальных технологий и способно работать без подключения к интернету. Такие приложения доступны на всех устройствах и адаптируются под — любую ширину экрана телефона или компьютера. Их можно устанавливать на главный экран своего смартфона, отправлять уведомления и пользоваться им, как полноценным приложением.

**1.4 Из чего состоит веб-приложение**

Веб-приложение состоит:

* Клиентской части (Frontend) реализует пользовательский интерфейс, создает запросы к серверу и обрабатывает ответы от него.
* Серверной части (Backend) принимает запрос от клиента, производит обработку, после, формирует и отправляет страницу клиенту с использованием протокола HTTP.

База данных программное обеспечение на сервере, занимающееся хранением данных и их выдачей в нужный момент. Серверная часть веб-приложения обращается к базе данных, извлекая данные, которые необходимы для формирования страницы, запрошенной пользователем.

**1.5 Из чего состоит серверная часть веб-приложения(Backend)**

Для разработки веб-приложений можно использовать практически любые современные языки программирования:

• PHP, Perl

• Ruby

• Java

• платформа .NET (языки VB.NET, C# и другие, поддерживаемые .NET)

• С/C++

Внутренние компоненты включают серверы, базы данных и логику приложения, каждый из которых играет определенную роль. Серверы отвечают за размещение веб-приложений и управление клиентскими запросами, а базы данных хранят и управляют данными, необходимыми для различных процессов веб-приложений. С другой стороны, логика приложения состоит из функций и процедур, которые определяют поток данных и управление ими, включая выполнение различных задач в ответ на ввод пользователя. По сути, серверная часть выступает в качестве основы, на которой работают внешние компоненты, снабжая веб-приложения необходимыми ресурсами и помогая поддерживать их общую эффективность и производительность.

Хорошо спроектированный бэкэнд должен быть масштабируемым, надежным и безопасным. Масштабируемость гарантирует, что веб-приложение сможет адаптироваться к возросшим нагрузкам и приспособиться к растущей базе пользователей, а надежность гарантирует стабильную и бесперебойную работу пользователей. Безопасность имеет решающее значение для защиты конфиденциальных данных и снижения потенциальных угроз веб-приложению.

Подводя итог, можно сказать, что серверная часть — это неотъемлемый аспект архитектуры веб-приложения, который обеспечивает бесперебойную работу элементов внешнего интерфейса и служит базовым механизмом, обеспечивающим различные функции приложения.

Backend отвечает за:

* Обработку пользовательских запросов и передачу соответствующих данных в Frontend.
* Управление базами данных и хранение информации, необходимой для работы приложения.
* Обеспечение безопасности данных и защиту от угроз.
* Взаимодействие с API и сторонними сервисами для расширения функциональности приложения.
* Аутентификация пользователя

## Функции Backend на сайте:

* Обработка пользовательских запросов, поступающих от пользователей через Frontend. Это может быть отправка данных, получение информации из базы данных, выполнение вычислений или любые другие действия, необходимые для предоставления ответа пользователю.
* Безопасность и аутентификация, включает в себя реализацию механизмов безопасности и аутентификации пользователей. Что обеспечивает защиту данных, проверку подлинности пользователей и контроль доступа к различным функциям приложения.
* Управление контентом и базой данных, включая его создание, редактирование и удаление. Также занимается управлением базы данных, обеспечивая её оптимальную работу, резервное копирование и восстановление данных.
* Взаимодействие с API и сторонними сервисами
* Проверка подлинности пользователя по авторизованной информации, осуществляющая доступ в соответствии с ролью;
* Новый пользователь – процедура создания новых пользователей системы, добавление с распределением ролей;
* Существующий пользователь – поиск необходимого пользователя по введенным критериям, как логин, пароль;
* Редактирование данных пользователя – изменение данных о пользователе или удаление его учетной записи из базы данных;
* Осуществляется выход пользователя из системы, после чего необходима повторная аутентификация для продолжения работы.

Преимущества использования Backend разработки:

* Улучшение производительности сайта. Оптимизированный Backend способствует более быстрой загрузке страниц и обработке пользовательских запросов.
* Расширяемость и масштабируемость.
* Разделение обязанностей (Frontend vs. Backend). Backend и Frontend выполняют разные функции и разделяют обязанности, что позволяет создавать более эффективные команды разработчиков. Frontend-разработчики занимаются визуальным представлением и пользовательским интерфейсом, в то время как Backend-разработчики работают с данными и логикой приложения.
* Обеспечение безопасности данных. Backend ответственен за обеспечение безопасности приложения и данных пользователей. Он должен защищать информацию от несанкционированного доступа, предотвращать атаки и обеспечивать конфиденциальность чувствительных данных.

**Глава 2 Описание используемых программных средств**

**2.1 Язык программирования Java**

Язык программирования Java — это строго типизированный объектно-ориентированный язык программирования высокого уровня. Всё взаимодействие в нём происходит через объекты. Программа [в стиле ООП](https://practicum.yandex.ru/blog/obektno-orientirovannoe-programmirovanie/) состоит из отдельных блоков, которые хорошо расширяются и масштабируются. Он был разработан компанией Sun, а позднее приобретён корпорацией Oracle, которой и принадлежит до сих пор. Несмотря на свой почётный возраст, Java продолжает развиваться, оставаясь одним из самых популярных языков программирования. Одним из главных преимуществ этого языка является его кроссплатформенность. Программы на Java компилируются в байт-код, который затем выполняется виртуальной машиной JVM (Java Virtual Machine). Преимущество такого подхода в том, что байт-код не зависит от операционной системы, таким образом, программы могут выполняться на любой платформе, для которой существует виртуальная машина Java. На Java пишется код не под macOS, Windows или Linux, а под Java-машину — а она уже сама адаптирует код к железу и операционной системе. Код на языке Java пишется один раз и запускается на любом устройстве, для которого написана Java-машина. Это позволяет тратить меньше ресурсов на разработку программ. Другое важное преимущество Java — наличие огромного количества библиотек с открытым исходным кодом, которые подходят для решения задачи любой сложности. В частности, в работе будут использованы фреймворки и библиотеки для создания веб-сервисов REST, контроля доступа, авторизации пользователей, взаимодействия с базой данных, внедрения зависимостей, аспектно-ориентированного программирования и мн. др. Кроме того, этот язык достаточно прост в использовании. В нем присутствует автоматическое управление памятью за счет использования механизма сборки мусора, а также строгая типизация, которая делает Java удобным средство для написания крупных проектов. Нельзя не отметить и богатый инструментарий. Интегрированные среды разработки (IDE), написанные на Java, такие, как Eclipse, NetBeans и продукты компании JetBrains, давно вышли за пределы экосистемы языка, став стандартным средством разработки программ на многих современных языках.

Язык программирования Java был тепло встречен мировым сообществом разработчиков программного обеспечения и провайдерами интернет-контента. Пользователи Интернета и всемирной паутины получают доступ к безопасным, независимым от платформы приложениям, которые могут поступать из любой точки мира в Интернете.

Многим язык был известен, прежде всего, как инструмент для создания апплетов для всемирной паутины. Апплет — это мини-приложение, которое работает внутри веб-страницы. Апплет может выполнять задачи и взаимодействовать с пользователями на их страницах браузера без использования ресурсов веб-сервера после загрузки. Некоторые апплеты могут, и конечно, взаимодействуют с сервером.

Язык программирования Java действительно ценен для распределенных сетевых сред, таких как Интернет.

Возможность безопасного выполнения загруженного кода на удаленных хостах имеет решающее значение, это требования многих организаций.

Другие группы используют его как язык программирования общего назначения для проектов, в которых требуется независимость от разворачиваемой среды. Простота программирования и функции безопасности, помогают быстро создавать рабочий код. Некоторые часто встречаемые ошибки программирования никогда не возникают из-за таких функций, как сборка мусора и безопасные ссылки.

Поддержка многопоточности предназначена для современных сетевых приложений и приложений с графическим пользовательским интерфейсом, которые должны заниматься несколькими задачами одновременно, а механизмы обработки исключений облегчают работу с ошибочными состояниями приложения. Хотя встроенные инструменты являются мощными, это простой язык, на котором программисты могут быстро стать опытными.

Язык программирования Java предназначен для максимальной переносимости с минимальной реализацией.

С момента введения объектно-ориентированной концепции многие языки были разработаны с ее использованием. Благодаря Sun Microsystems и ее поддержке приложений с высоким уровнем безопасности Java стала известна во всем мире.

**2.2 Фреймворк.**

Фреймворк – это набор шаблонов, заготовок для разработки приложений, которые позволяют быстро и качественно решить определенный набор задач.

Для разработки веб-приложений есть множество фреймворков, которые можно использовать в Java, но самым популярных из них сейчас является Spring. Он имеет несколько модулей, которые отвечают за определенный пул задач.

Любое нетривиальное приложение состоит из множества компонентов, каждый из которых отвечает за отдельную часть общей функциональности приложения, координируется с другими элементами приложения для выполнения работы. Когда приложение запускается, эти компоненты должны быть каким-то образом созданы и представлены друг другу.

По своей сути Spring предлагает контейнер, часто называемый контекстом приложения Spring, который создает компоненты приложения и управляет ими. Эти компоненты, или бины, связаны друг с другом внутри контекста приложения Spring. Соединений bean-компонентов вместе основано на шаблоне, известном как внедрение зависимостей(DI). Вместо того, чтобы компоненты создавали и поддерживали жизненный цикл других компонентов от которого они зависят, приложение с внедрением зависимостей полагается на отдельный объект(контейнер) для создания и поддержки всех компонентов и внедрения их в bean-компоненты.

Помимо своего основного контейнера, Spring предлагает веб-фреймворк, различные варианты сохранения данных, каркас безопасности, интеграция с другими системами, мониторинг времени выполнения, поддержка микросервисов, модель реактивного программирования и многие другие функции, необходимые для современного приложения.

**2.3 База данных**

Неотъемлемой частью любого приложения является база данных.

База данных представляет собой набор больших объемов данных. Система управления базой данных(СУБД) – это организованный способ управления, извлечения и сохранения данных. Один из действия, которые можно выполнять в базе данных:

* Выполнение сложных расчетов
* Извлекать записи на основе соответствия или сравнение функций
* Массовое обновление записей в экземпляре.
* Поддержка связей в таблицах. Информация в нескольких таблицах может быть связана.

Надежность модели базы данных проверяется с помощью принципов ACID.

* Атомарность означает «все или ничего». Если есть часть сделки, которая остается незавершенной, то вся транзакция считается несостоявшейся.
* Непротиворечивость гарантирует, что база данных до и после любой транзакции стабильна в действительном состоянии.
* Изоляция гарантирует, что несколько транзакций выполняющиеся одновременно не влияют друг на друга.
* Долговечность гарантирует, что после того, как транзакция зафиксирована, она останется в том же состоянии, т.е. хранится постоянно, даже если есть некоторые ошибки, или даже в случае сбоя системы или потери питания.

Когда набор элементов данных организован с помощью формально описанных таблицы, сформированная база данных называется реляционная база данных. К ней можно легко получить доступ, создать и расширить ее. По этой причине это преобладающий выбор хранения данных, по иерархической модели или сетевой модели.

Некоторые из преимуществ базы данных, разработанной в соответствии с реляционной моделью:

* Большая часть информации хранится в базе данных, а не в приложении, поэтому база данных самостоятельно документируются.
* Легко добавлять, обновлять или удалять данные.
* Дает преимущества обобщения данных, поиска и отчетности.
* База данных имеет табличную структуру с сильно связанными таблицами, характер базы данных предсказуемый.
* Кроме того, довольно просто внести любые изменения

Три наиболее широко используемые реляционные базы данных: MySQL, Oracle и PostgreSQL. MySQL более популярен среди сайтов. Это легкая система, которая работает очень быстро, Oracle в основном используется в случае большой базы данных, например, в сфере банкинга, страхования, ERP и финансовых компаний. База данных PostgeSQL почти ничем не отличается от Oracle, она также используется для хранения больших данных и аналитических задач.

**2.4** **Архитектура серверной части приложения**

Серверная часть приложения является очень важной, она содержит всю бизнес-логику приложения и отвечает за весь функционал, который есть в приложении. Важно понимать, что приложение со временем всегда будет развиваться и дополняться новыми функциями, поэтому очень важно разрабатывать серверную часть приложения с возможностью легкого масштабирования, именно поэтому архитектуре серверной части приложения должно быть уделено много времени и внимания.

Архитектура программного обеспечения — это фундаментальная структура программного обеспечения, определяющая технические и эксплуатационные требования. Она отвечает за оптимизацию каждого атрибута приложения, таких как эффективность, управляемость, масштабируемость, надежность, модифицируемость, возможность развертывания и многие другие аспекты. Поэтому выбор подходящей архитектуры очень важен на начальном этапе разработки программного обеспечения. Существует несколько типов программных архитектур, однако монолитная и микросервисная архитектуры относятся к самым популярным.

В последние несколько лет разработчики программного обеспечения успешно использовали монолитную архитектуру. Монолитное приложение представляет собой программное обеспечение, в котором различные компоненты (например, авторизация, бизнес-логика, модуль уведомлений и т.д.) объединены в единую программу с единой платформой. На рисунке ниже представлен пример монолитной архитектуры.

Несмотря на реализацию многих частей всего ПО, приложение развернуто как одна отдельная автономная программа.

Преимущества такой архитектуры:

* Простота разработки
* Простота развертывания

Недостатки монолитной архитектуры:

* Сложность поддержки
* Надежность (одна неисправность может вывести из строя все приложение)
* Доступность (повторное развертывание всего приложения на каждое обновление)
* Трудно масштабировать

Микросервисная архитектура это архитектурный стиль, который структурирует приложение как набор сервисов. Каждый микросервис должен был предоставить один часть бизнес-логики.

Микросервисная архитектура подразумевает разделение приложения на сервисы по его функционалу, то есть каждый сервис ответственен всего лишь за одну функциональность. Это дает свои преимущества при разработке приложения, при его поддержке, а также масштабировании.

Преимущества микросервисной архитектуры:

* Простота обслуживания (легко понять разработчику, каждая основная функциональность является отдельным модулем)
* Надежность (сбой микросервиса влияет только на этот микросервис)
* Доступность (повторное развертывание новой версии микросервиса требует небольшого времени)
* Простота масштабирования

Недостатки микросервисной архитектуры:

* Более сложное развертывание
* Автономность (это положительный момент, однако достичь этого вместе с интеграцией данных довольно сложно)

Обе архитектуры имеют некоторые преимущества и недостатки. Все зависит от задачи, которую мы хотим решить. Микросервисная архитектура более эффективна, если приложению приходится обрабатывать большое количество запросов. В нем много преимуществ, которые позволяют создавать высококачественное программное обеспечение, которое легко масштабируется, более надежное и в долгосрочной перспективе более удобно поддерживать. Несмотря на то, что у него так много преимуществ, монолитная архитектура — это неплохо. Такой подход более эффективен во время небольшой нагрузки, а также при такой архитектуре проще разрабатывать приложение. Мы не получаем больших проблем с интеграцией, развертыванием и настройкой. Выбор правильной архитектуры должен определяться целями, которые нужно достичь, чтобы получить продукт, который оправдает ожидания.

В данной работе при разработке приложения будет использована микросервисная архитектура, учитывая ее популярность и преимущества, в данный момент сложно убедить разработчика или заказчика использовать монолитную архитектуру. Кейс, при котором монолитная архитектура оправдывает себя это, когда нужно максимально быстро получить результат, здесь и сейчас, но это тянет за собой огромное количество недостатков, которые всплывут в самом ближайшем будущем. Поэтому самый оптимальный вариант – использование микросервисной архитектуры при разработке приложения, для дальнейшей возможность его развития и масштабирования.

**2.5 Spring Framework**

Разрабатываемый проект основан на использовании фреймворка с открытым исходным кодом Spring.

Spring Framework является основой для всех других проектов. Spring Boot, Spring Data, Spring Batch — все это построено поверх Spring.

Spring Framework представляет собой [контейнер внедрения зависимостей](https://en.wikipedia.org/wiki/Dependency_injection). Это все позволяет быстрее и удобнее создавать Java-приложения.

**2.6 Spring Container**

Spring Container — это модуль, использующийся для внедрения зависимостей. Контейнер Spring управляет жизненным циклом объектов и связывает их друг с другом. Объекты, которые он контролирует, называются бинами (от слова Bean). Классы бинов могут зависеть друг от друга, и задача контейнера Spring состоит в создании экземпляров этих классов и внедрении их в другие зависимые от них бины. Такой механизм обеспечивает слабую связанность классов, что улучшает структуру кода, упрощает их взаимозаменяемость и тестирование.

**2.7 Spring Aspect-oriented Programming (AOP)**

Spring AOP — это парадигма программирования, основанная на выделении «сквозной функциональности» (cross-cutting concerns) в отдельные объекты. Такая функциональность затрагивает многие компоненты системы, но при этом не относится непосредственно к логике приложения. Примерами сквозной функциональности можно назвать логирование информации и управление транзакциями базы данных.

**2.8 Spring MVC**

Spring MVC — это фреймворк, в основе которого лежит технология Java Servlets. Spring MVC предназначен для разработки веб-приложений, и именно с его помощью разрабатывается REST API сервиса. Этот модуль предоставляет возможность обработки пользовательских запросов, их валидации, генерации HTML-страниц и т. д. Иначе говоря, Spring MVC отвечает за всё, что связано с сетевой частью приложения.

**2.9 Spring Security**

Задача Spring Security заключается в аутентификации и авторизации клиентов приложения. Контроль доступа осуществляется как на уровне HTTP-запросов, так и на уровне вызова отдельных методов. Spring Security также реализует поддержку протокола OAuth, позволяя развернуть и гибко настроить собственный сервер авторизации и сервер ресурсов.

**2.10 Spring Data Access**

Spring Data отвечает за взаимодействие приложения с базой данных. Этот модуль поддерживает интеграцию с другими технологиями работы с базами данных на Java, такими как JDBC и JPA, а также дает возможность работать с нереляционными хранилищами. Кроме того, Spring Data позволяет декларативно управлять транзакциями. Одним из главных преимуществ этого модуля является возможность динамически генерировать классы доступа к базе данных на основе интерфейсов, определенных разработчиком.

**2.11 Spring Boot**

Spring Boot — это проект, направленный на упрощение разработки приложений с помощью фреймворка Spring. Он позволяет в максимально короткие сроки создать полностью функциональное приложение, требуя при этом минимум усилий от разработчика.

Особенности Spring Boot:

1) встроенный веб-сервер (Tomcat, Jetty или Undertow), который автоматически запускается вместе с приложением;

2) набор готовых пакетов зависимостей (starter dependencies);

3) автоматическая конфигурация как самого Spring, так и некоторых сторонних библиотек;

4) возможность следить за состоянием приложения в реальном времени с помощью Spring Boot Actuator;

5) полностью избавляет разработчика от необходимости использовать XML конфигурацию.

**2.12 Swagger REST**

— это один из способов взаимодействия между клиентом и сервером в веб-приложениях. REST — это архитектурный стиль, набор требований к программному интерфейсу. Клиенту же необходимо точно знать, какие данные ожидает получить сервер, в каком формате должны быть эти данные, значение каждого возможного кода состояния и т. д. Таким образом, разрабатываемый API приложения должен быть подробно задокументирован.

Swagger — это проект с открытым исходным кодом, предназначенный для документирования программного интерфейса REST. Он даёт возможность сгенерировать документацию и визуализировать её, используя Swagger UI. Кроме того, с помощью Swagger UI также можно отправлять запросы для проверки и тестирования разработанного API. Спецификация Swagger, также известная как OpenAPI, имеет несколько реализаций. Одной них является библиотека SpringFox, позволяющая интегрировать Swagger с приложениями, разработанными с помощью Spring Framework.

**2.13 JUnit и Mockito**

Есть множество видов тестирования программного обеспечения: функциональное, интеграционное, нагрузочное и т. д. Одним из них является модульное, или юнит-тестирование, которое обычно выполняется программистами. Цель такого тестирования — проверка на корректность работы отдельных компонентов приложения (модулей) в изоляции друг от друга. Для написания юнит-тестов используются фреймворки JUnit и Mockito, которые чаще всего используются в связке друг с другом. JUnit является стандартным фреймворком для автоматизированного модульного тестирования приложений, написанных на Java. Он позволят запускать тестовые методы, группировать их для выполнения, проверять соответствие результатов теста ожидаемым и многое другое. Mockito же необходим для создания объектов-заглушек, которые позволяют тестировать модули изолированно. Они имитируют другие классы, и их поведение предопределено программистом. Кроме того, заглушки позволяют убедиться, что тестируемый объект выполнил требуемые от него действия.

**2.14 Project Lombok**

Зачастую необходимо писать большое количество однообразного кода. Описанные выше классы отображения являются одним из примеров такого кода. Но если необходимость писать подобные классы возникает не так часто, то писать стандартные методы доступа (getters и setters), сравнения (equals), вычисления хеш-кода (hashcode), преобразования объекта в строку (toString) и др. приходится в каждом Java-проекте без исключения. Project Lombok помогает упростить работу программиста, автоматизировать этот процесс. Lombok — это проект для генерации исходного кода, добавляющий дополнительную функциональность в язык Java. Кроме описанных выше стандартных методов объектов, Lombok также позволяет:

1) использовать неявные типы переменных (var и val);

2) проверять аргументы методов на равенство null;

3) создавать конструкторы классов с разным набором параметров;

4) создавать классы-строители;

5) создавать неизменяемые классы;

6) автоматически освобождать ресурсы.

Всё, что нужно, чтобы использовать этот функционал, — это добавить аннотацию к классу или методу, и Lombok автоматически генерирует необходимый код при компиляции программы.

**2.15 Maven**

Maven — это фреймворк с открытым исходным кодом, который существенно упрощает и автоматизирует процесс сборки проектов. Maven позволяет добавлять внешние зависимости в проект, используя для этого центральный репозиторий, предоставляет богатый набор плагинов, расширяющих функционал, позволяет создавать дистрибутивы программ в виде архивов jar/war и многое другое. Кроме того, эта система сборки поддерживается всеми популярными средами разработки, которые позволяют импортировать проект Maven без необходимости его дополнительной конфигурации. Центральным компонентом Maven является специальный XML-файл Project Object Model (pom.xml). POM используется для описания проекта и конфигурирования процесса сборки. Этот файл содержит информацию об имени и версии проекта, его авторе, используемой версии Java, внешних зависимостях, плагинах и т. д. Все проекты, использующие Maven, имеют единую унифицированную структуру каталогов:

1) /src/main/java содержит исходный код программы;

2)/src/main/resources содержит дополнительные ресурсы приложения (SQL-скрипты, файлы конфигурации и т. д.);

3) /src/test/java содержит исходный код тестовых классов;

4) /src/test/resources содержит дополнительные ресурсы приложения, необходимые для тестирования;

5) /target содержит результаты сборки, которые включают в себя дистрибутив программы, скомпилированные классы, результаты тестирования и т. д.

**Глава 3. Предпроектная стадия**

**3.1 Порядок проведения работ**

В ходе работы над проектом были проведены следующие стадии

* Проектирование веб-приложения: определение функционала и архитектуры приложения, выбор используемых технологий
* Подготовительные работы: установка и настройка окружающей среды.
* Разработка серверной части проекта: с использование Java и фреймворка Spring, создание моделей, репозитория, сервисов, контроллера.
* Разработка пользовательского интерфейса: создание HTML шаблонов с использование Thymeleaf.
* Отладка и оптимизация: Тестируем веб-приложение, исправляем выявленные ошибки, выполняем оптимизацию.
* Документирование: создание документации по проекту.

**3.2 Определение функционала и архитектуры приложения**

Для разработки веб-приложения “Книга конспектов” было определено:

Функции приложения:

* Пользователь должен имеет возможность создание записи конспекта, содержащего поля тему, основные понятия, подробное объяснение, вывод
* Приложение должно предоставлять возможность сортировки конспектов по теме
* Пользователь должен иметь возможность просмотра и редактирования конспектов.

**Глава 4 Разработка серверной части**

**4.1 Создание проекта**

Существует несколько способов создания проекта на Spring Boot, был использован наиболее распространенный метод использование веб-сайта Spring Initializer ([https://start.spring.io /](https://start.spring.io/)).

При переходе на веб-сайт Spring Initializer ([https://start.spring.io /](https://start.spring.io/)), были выбраны параметры, которые подходят для создаваемого проекта, такие как тип проекта Maven, язык программирования Java и версия Spring Boot и языка, Метаданные проекта. Зависимости использование в проекте.

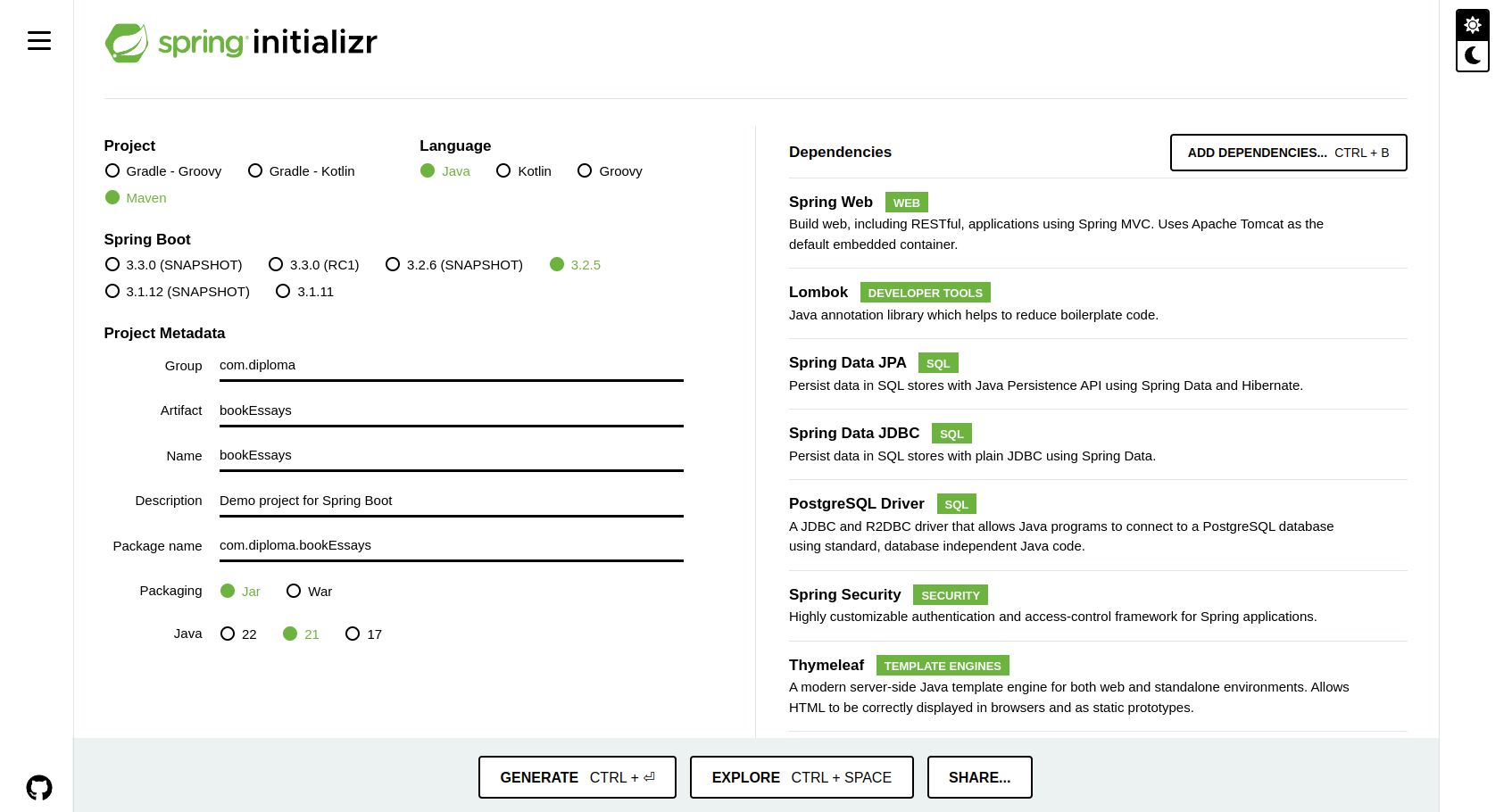


Рисунок 1: Spring Initializer ([https://start.spring.io /](https://start.spring.io/))

**4.2 Создание основных объектов Entity**

* Модель представляет собой структуру данных приложения.
* Включает классы, описывающие сущности приложения

Класс Memo(Конспект) с инициализацией полей, класс представляет собой запись конспекта в книгу конспектов. Каждая запись содержит информацию о индивидуальном индексе(id), дате создания(createDate), тема(topic), основные понятия(notion), подробное объяснение(description), вывод(conclusion).

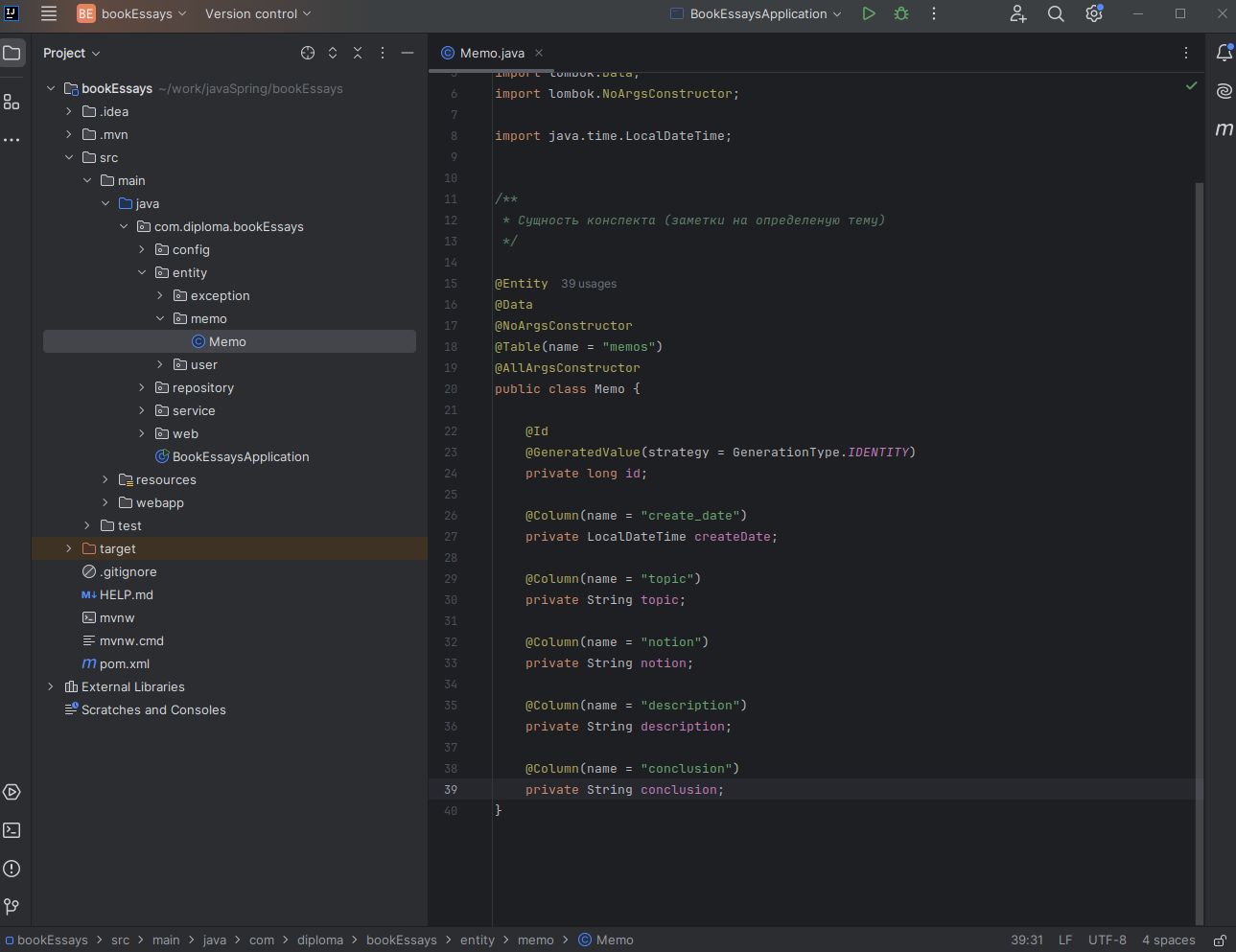


Рисунок 2: Класc Memo

Класс User(Пользователь) с инициализацией полей, класс представляет собой пользователя. Каждый пользователь содержит информацию о индивидуальном индексе(id), имени пользователя(name), роль пользователя(roles) определяющей права пользователя и доступ к информации, индивидуальном логине пользователя(username), пароль пользователя (password), подтверждении пороля (passwordConfirmation), о состоянии пользователя (active), лист зависимости по индивидуальным индексам созданных пользователем конспектов (memos).

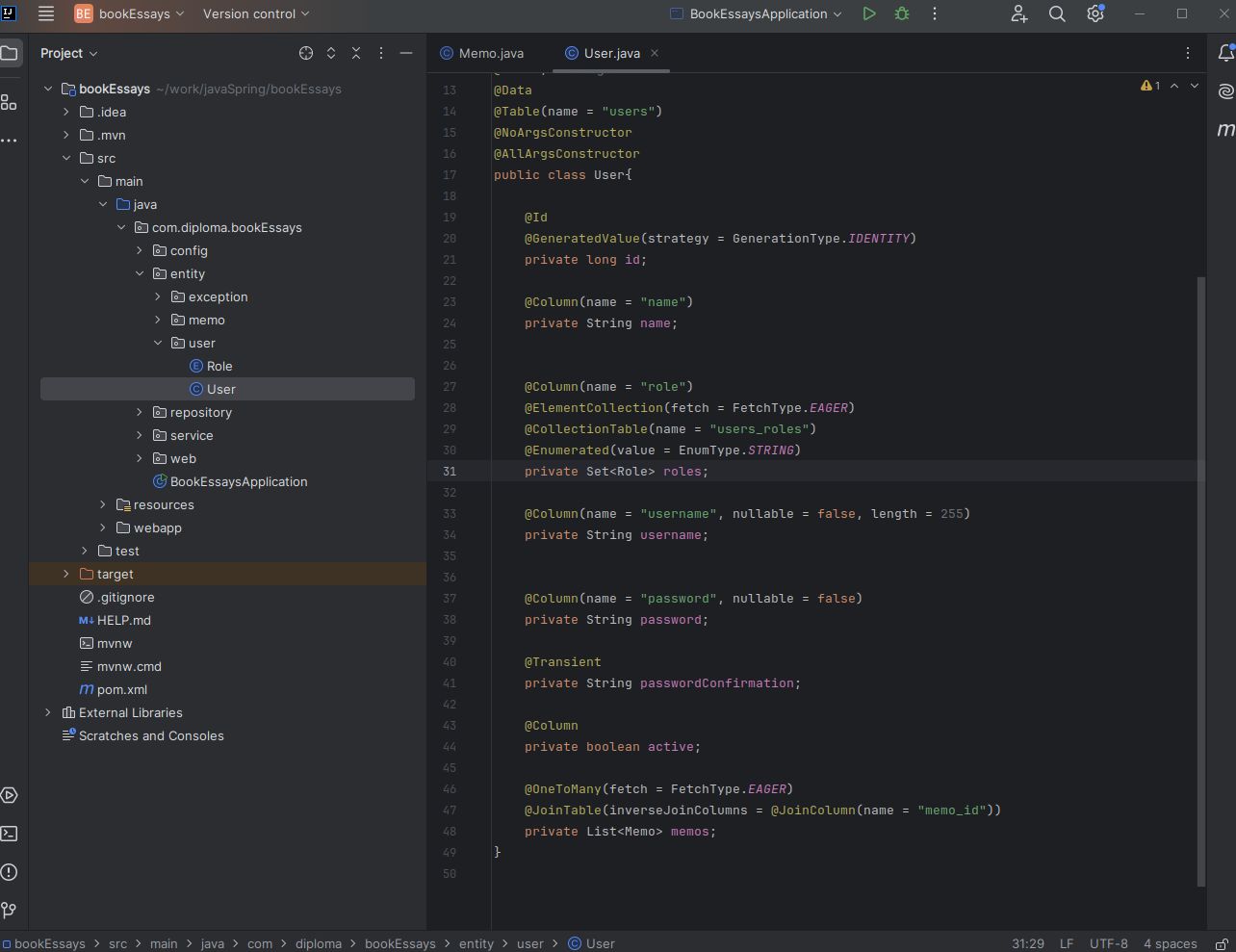


Рисунок 3: Класс User

Класс Role(Роль) представляющий собой класс перечисления ролей пользователя, для определения прав доступа.

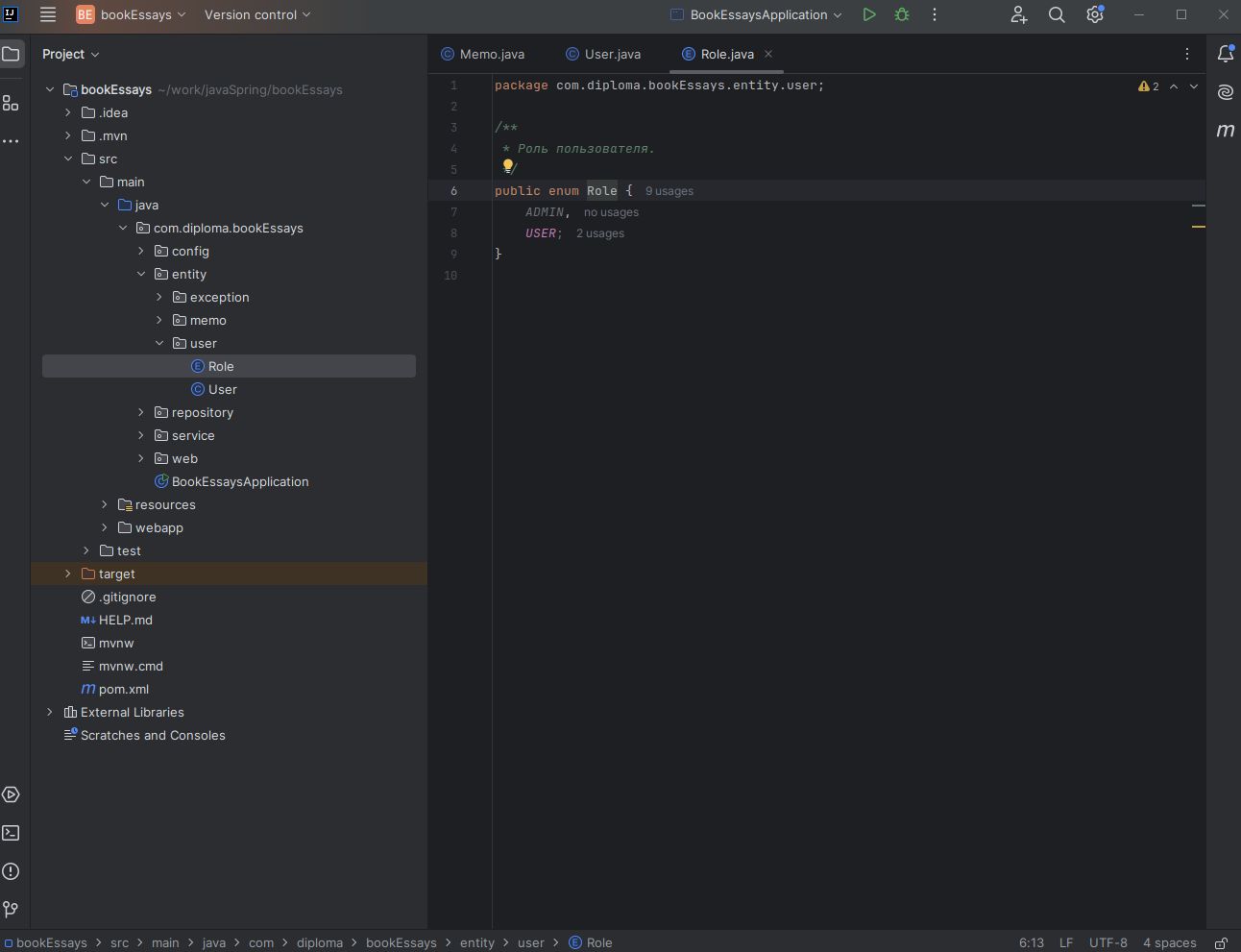


Рисунок 4: enum Role

Классы User и Memo имеют аннотацию:

* @Table указывает на имя таблицы, которая будет отображаться в этой сущности.
* @Entity Указывает, что данный бин (класс) является сущностью.
* @Data – это сокращенная аннотация, сочетающая возможности @ToString , @EqualsAndHashCode , @Getter @Setter и @RequiredArgsConstructor . Так что @Data генерирует весь шаблонный код, вовлеченный в работу с объектами POJO (Plain Old Java Objects).

Поля User и Memo имеют аннотацию:

* @Column — указывает на имя колонки, которая отображается в свойство сущности.
* @Id — id колонки
* @GeneratedValue — указывает, что данное свойство будет создаваться согласно указанной стратегии.
* @Transient — указывает, что свойство не нужно записывать. Значения под этой аннотацией не записываются в базу данных (также не участвуют в сериализации). static и final переменные экземпляра всегда transient.
* @OneToMany — указывает на связь [один ко многим](https://javastudy.ru/hibernate/hibernate-one-to-many/). Применяется с другой стороны от сущности с @ManyToOne
* @JoinTable — указывает на связь с таблицей

**4.3 Создание интерфейса репозиториев Spring Repository**

* Репозиторий отвечает за доступ к данным из базы данных.
* В Java Spring репозитории обычно реализуются с помощью интерфейсов, расширяющих **JpaRepository**, которые предоставляют базовые методы для работы с данными.
* Репозитории используются сервисами для получения, сохранения, обновления и удаления данных

UserRepository и MemoRepository это интерфейсы унаследованные от репозитория JpaRepository который используют Entity для взаимодействия с базами данных, осуществления выборки и сортировки, предоставления данных.

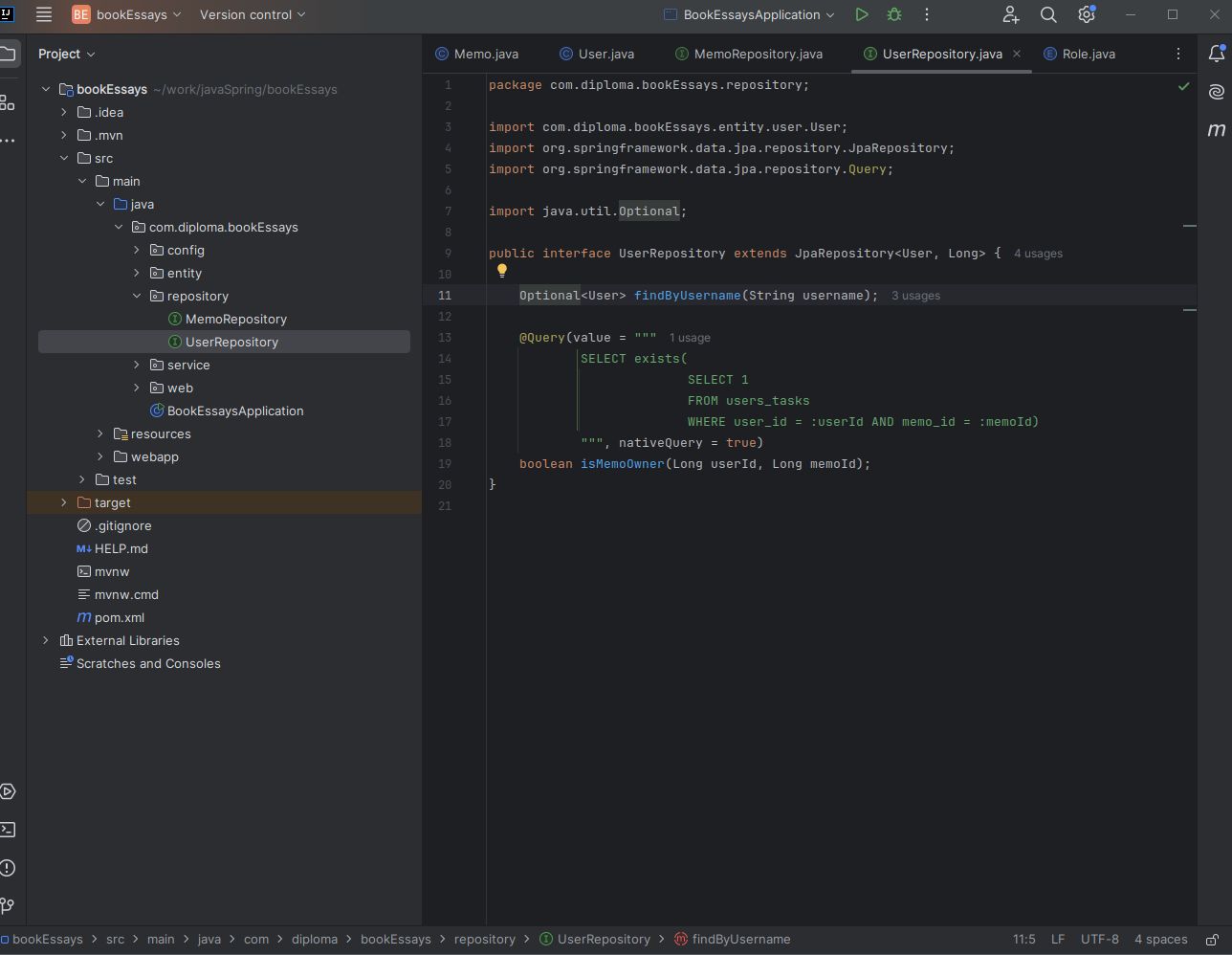


Рисунок 5: UserRepository

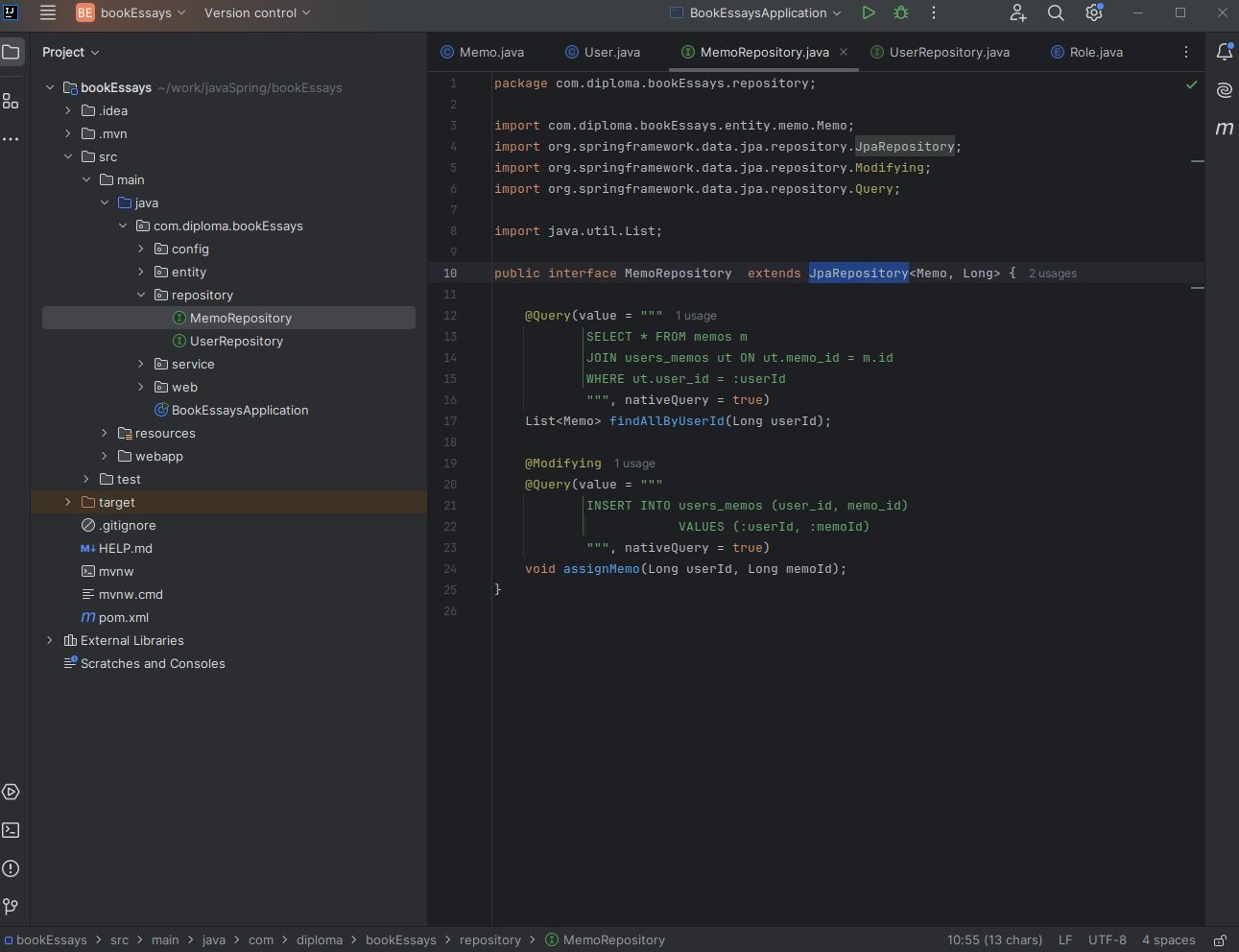


Рисунок 6: MemoRepository

**4.4 Создание сервис-слоя**

* Сервис содержит бизнес-логику приложения.
* Он обычно используется для выполнения операций с данными, координирования действий между различными компонентами приложения и обеспечения единого доступа к функциональности приложения.

Для создания сервис-слоя были созданы сервис-компоненты UserServiceImpl и MemoServiceImpl, они необходимы для того чтобы абстрагировать логику, создавая общий сервис, для отсутствия фактов дублирования кода и обслуживания отдельного класса.

Класс UserService является сервисом, предоставляющим функционал для работы с пользователями:

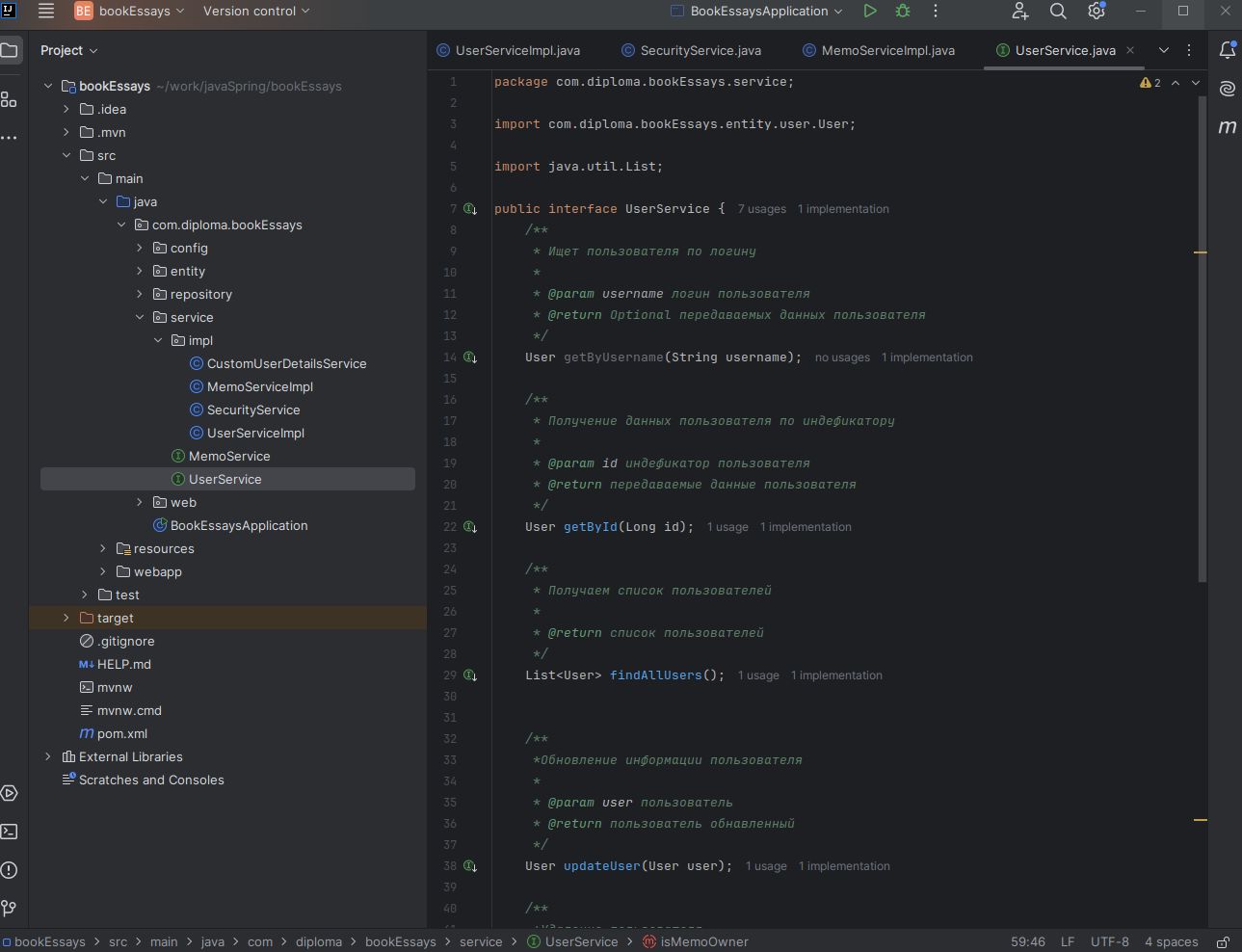


Рисунок 7: interface UserService

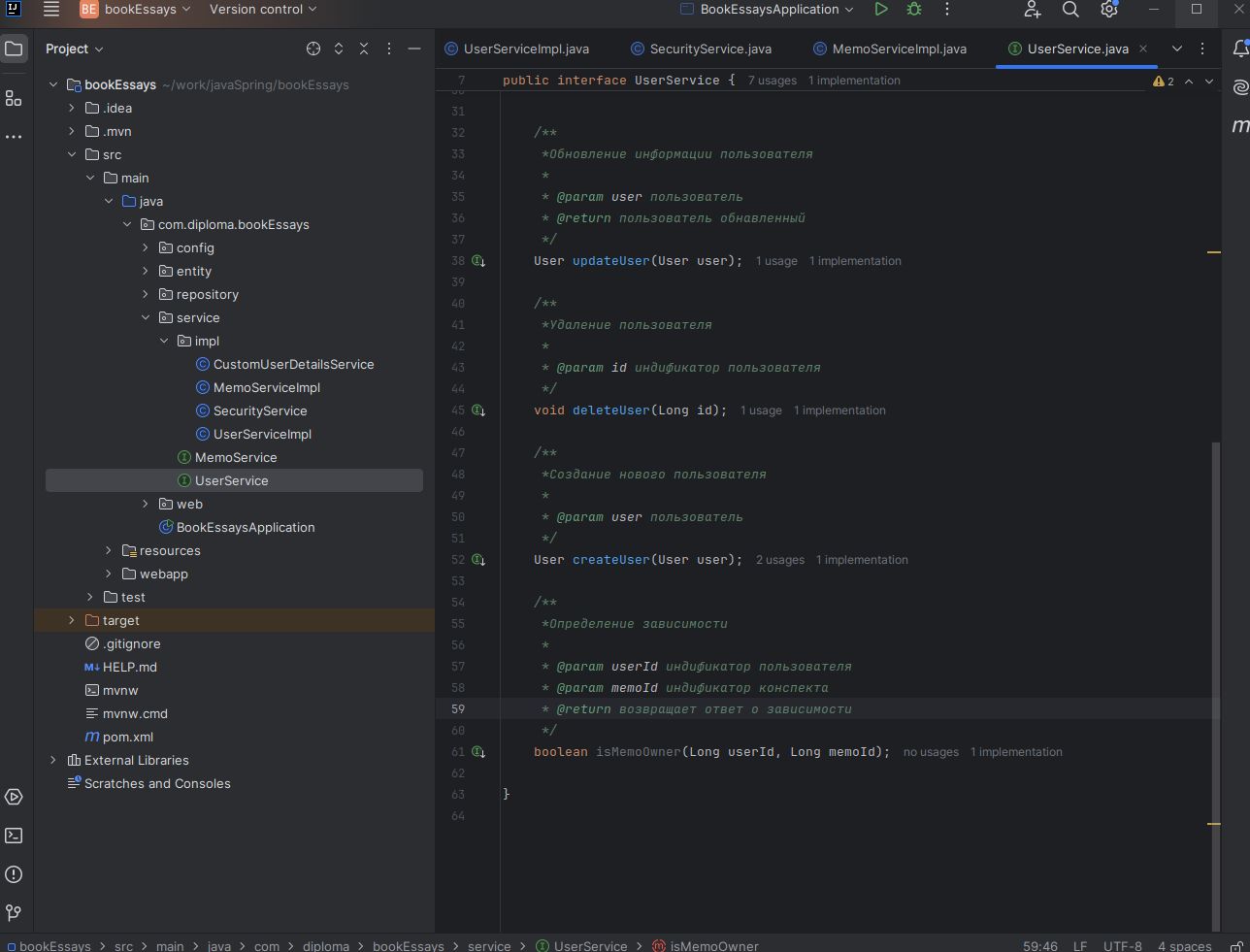


Рисунок 8: продолжение рисунка 7

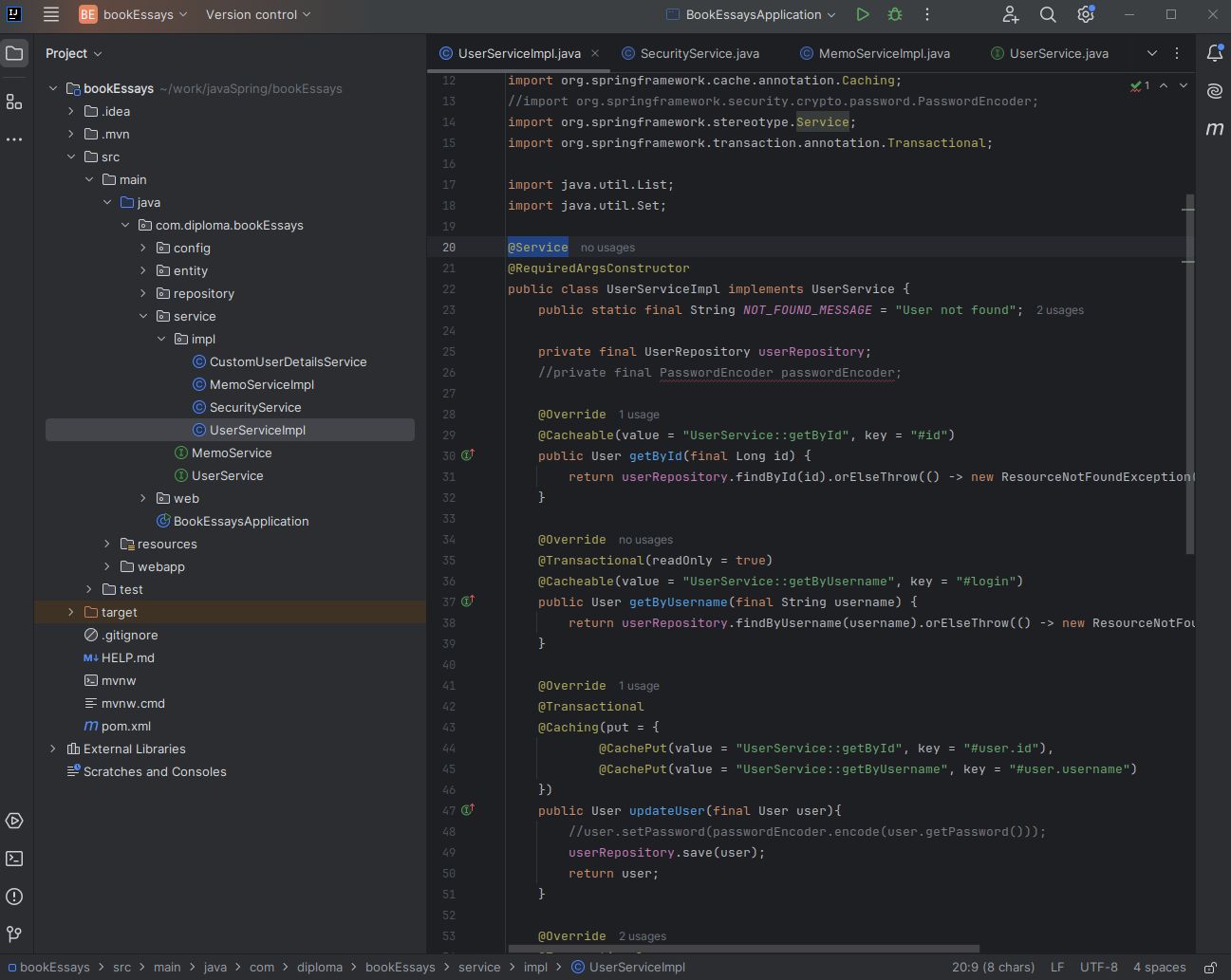


Рисунок 9: UserServiceImpl implements UserService

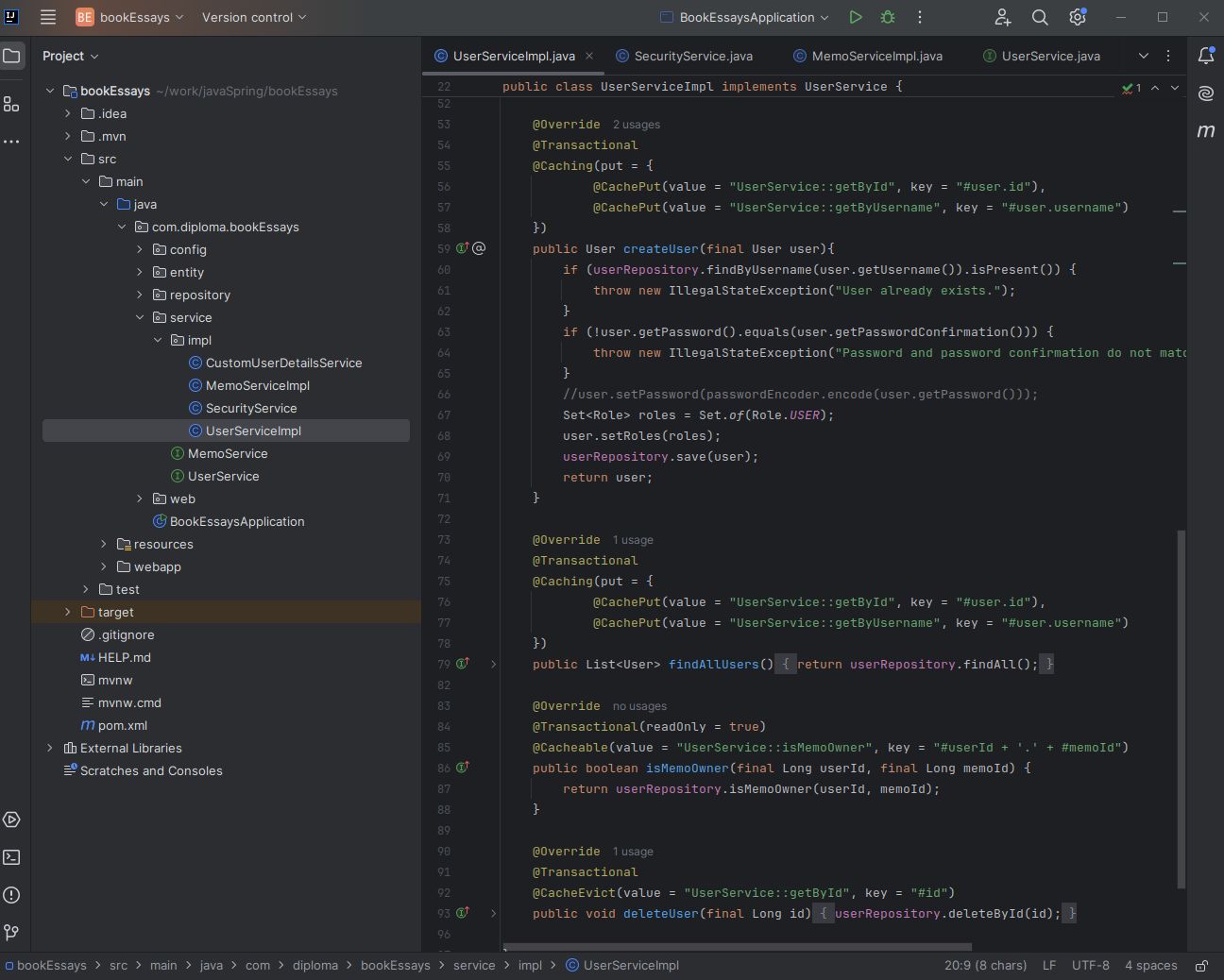


Рисунок 10: продолжение рисунка 9

@Service - (Сервис-слой приложения) Аннотация, объявляющая, что этот класс представляет собой сервис – компонент сервис-слоя. Сервис является подтипом класса @Component.

@RequiredArgsConstructor генерирует конструктор с 1 параметром для каждого поля, которое требует специальной обработки. Все неинициализированные final поля получают параметр, также как все остальные поля, помеченные @NonNull, которые не инициализированы при объявлении.

Для работы с пользователями UserService предоставляет методы:

User getByUsername(String username) ищет пользователя по параметру username, username является логином пользователя, возвращает найденного пользователя, если он имеется в базе данных, в ином случае выбрасывается ResourceNotFoundException(*NOT\_FOUND\_MESSAGE*), User not found.

User getById(Long id) ищет пользователя по идентификатору id, id является идентификатором пользователя, возвращает найденного пользователя, если он имеется в базе данных, в ином случае выбрасывается ResourceNotFoundException(*NOT\_FOUND\_MESSAGE*), User not found.

User updateUser(User user) осуществляет обновление полей пользователя и сохраняет их в базе данных, возвращает пользователя с обновленными полями.

User createUser(User user) Создает нового пользователя, производит проверку о наличии пользователя по username, проверяет пользователя на правильность введенного пароля, и устанавливает роль "USER". Если пользователь с таким именем уже существует, операция добавления завершается неудачей, в случае добавления пользователя возвращает пользователя.

List<User> findAllUsers() возвращает список пользователей, сохраненных в базе данных.

void deleteUser(Long id) удаляет пользователя по идентификатору id.

Класс MemoService является сервисом, предоставляющим функционал для работы с конспектами:

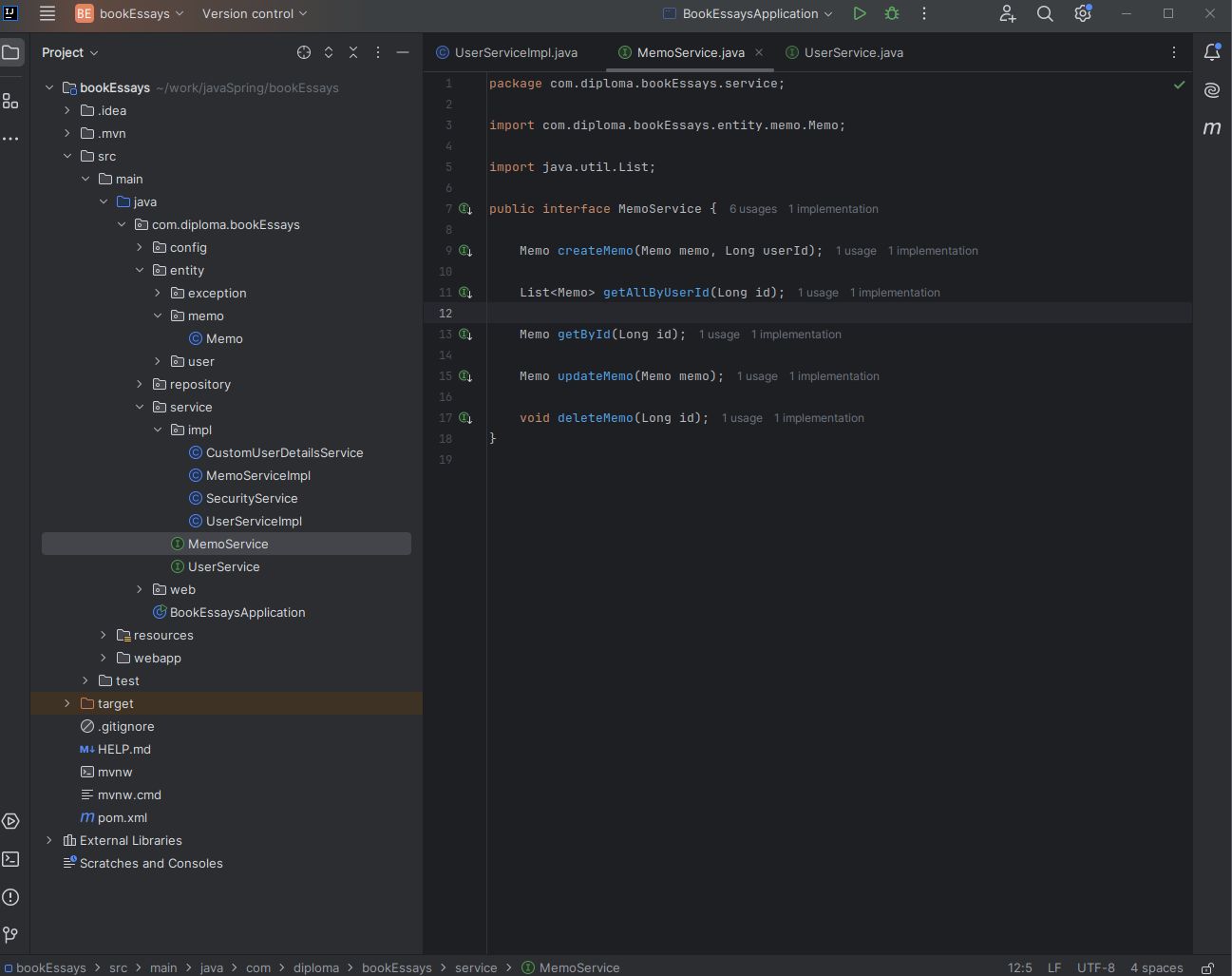


Рисунок 11: MemiService

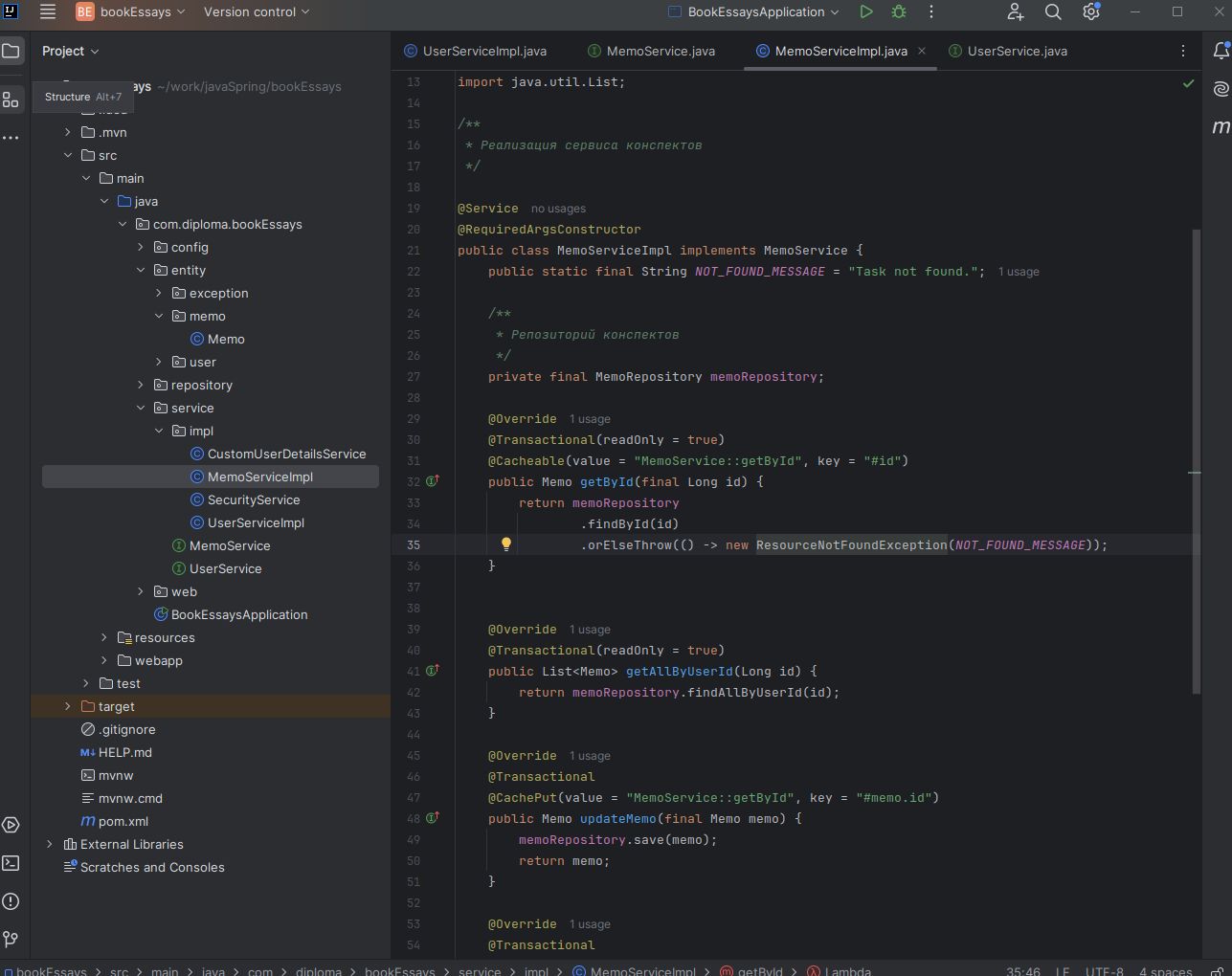


Рисунок 12: MemoServiceImpl implements MemiService

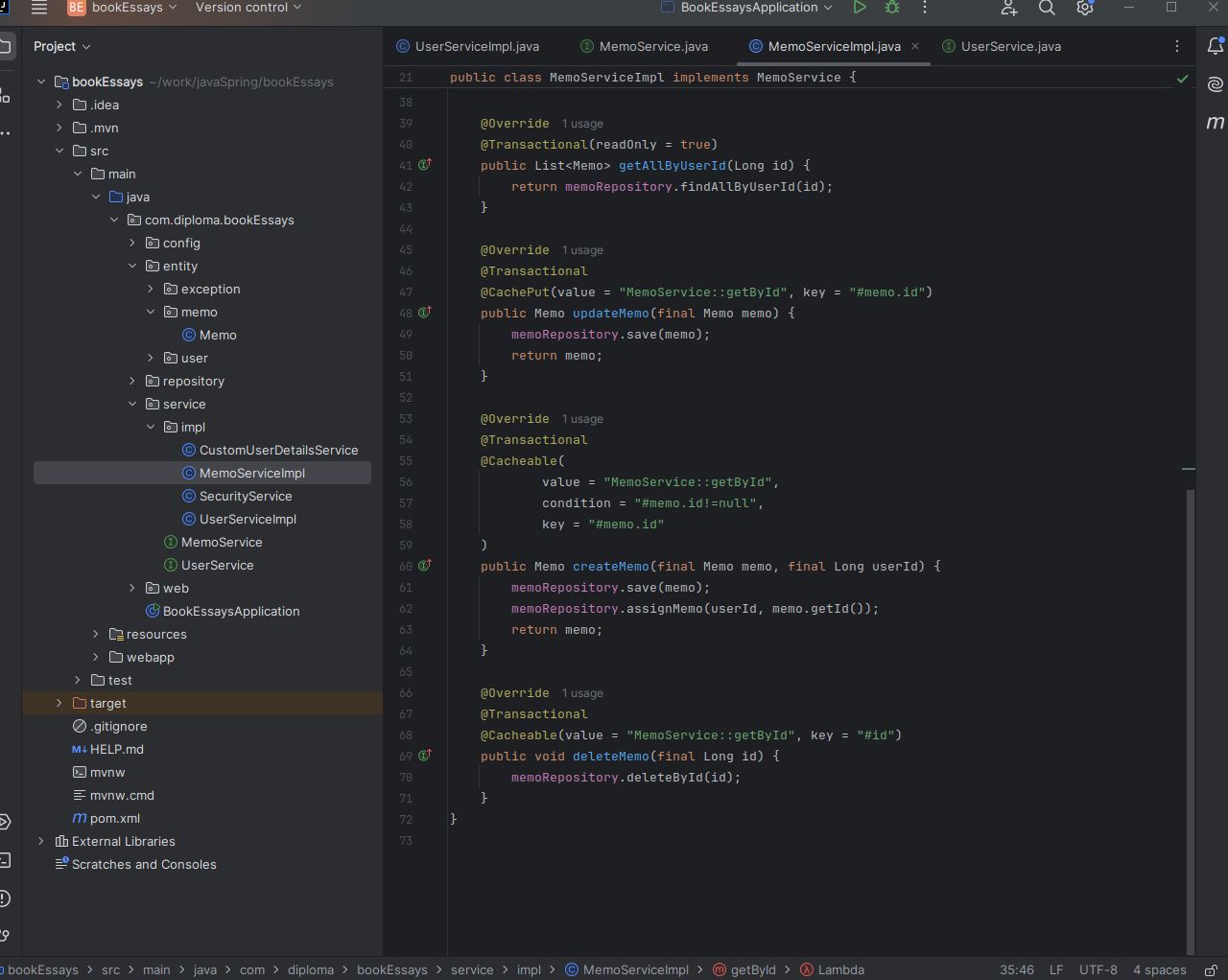


Рисунок 13: продолжение рисунка 12

Для работы с конспектами MemoService предоставляет методы:

Memo createMemo(Memo memo) Создает новую запись конспекта, добавляет зависимость конспекта с пользователем по идентификатору.

List<Memo> findAllByUsersId(Long id) возвращает список конспектов, сохраненных в базе данных, с фильтрацией по пользовательскому индификатору.

Memo getById(Long id) ищет конспект по идентификатору id, id является идентификатором конспекта, возвращает найденный конспект, если он имеется в базе данных, в ином случае выбрасывается ResourceNotFoundException(*NOT\_FOUND\_MESSAGE*), Memo not found.

Memo updateMemo(Memo memo) осуществляет обновление полей конспекта и сохраняет их в базе данных, возвращает конспект с обновленными полями.

void deleteMemo(Long id) удаляет конспект по идентификатору id.

**4.5 Создание сущностей DTO**

**Data Transfer Object (DTO)** — один из [шаблонов проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), используется для передачи данных между подсистемами приложения.

В приложении реализован MemoDto и UserDto.

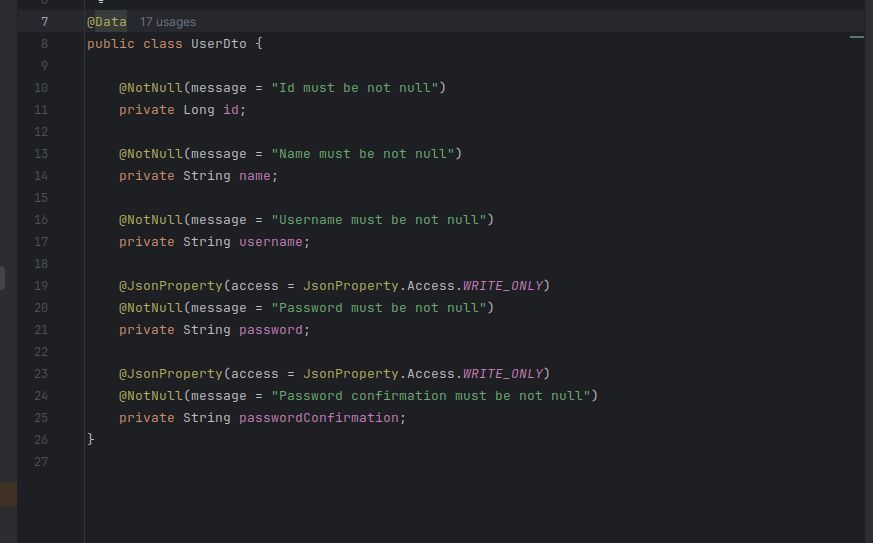


Рисунок 14: UserDto

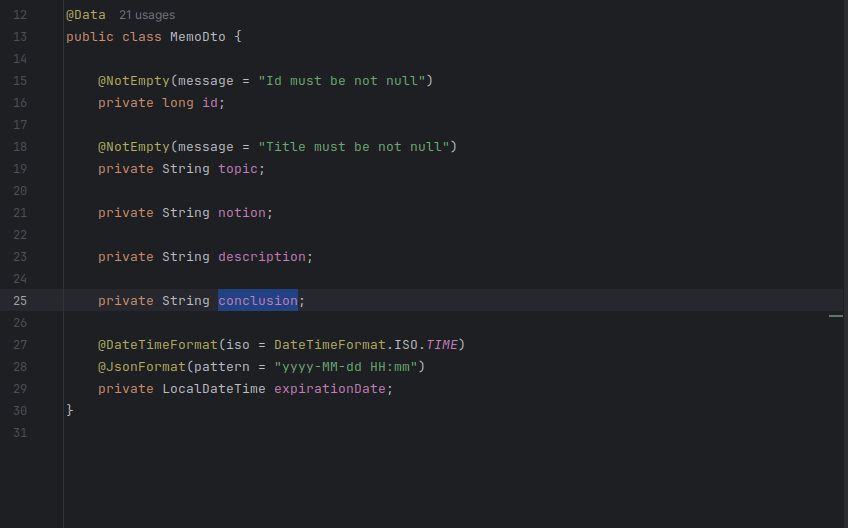


Рисунок 15: MemoDto

**4.6 Создание маппера**

Отвечает за преобразование сущностей в DTO и обратно, @Mapper

позволяет автоматизировать данный процесс и уменьшить количество кода приложения. С его помощью создаются специальные мапперы под каждую сущность. Внутри них определяются правила конвертирования в DTO или из DTO в зависимости от потребностей. Дальше эти мапперы используются в нужных местах, сводя преобразования к одной строчке.

Для этого в приложении реализованы маппер Mappfble<E, D>

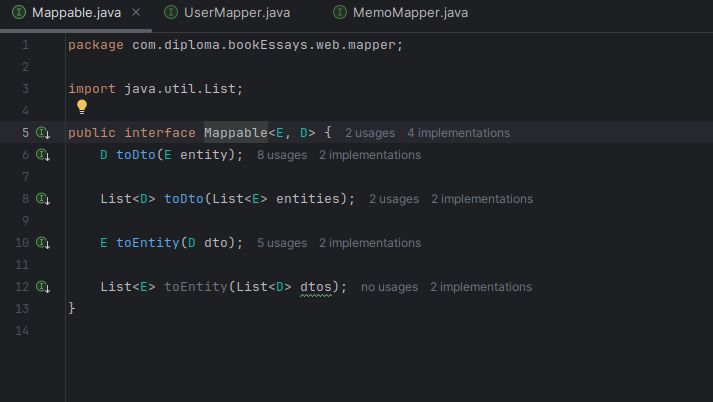


Рисунок 16: Mappfble<E, D>

Для преобразование Memo и User, созданы мапперы UserMapper<User, UserDto> и MemoMapper<Memo, MemoDto>

D toDto (E entity) данный метод преобразует Dto в Entity

List<D> toDto(List<E> entities) данный метод преобразует лист Dto в лист Entity

E toEntity(D dto) данный метод преобразует Entity в Dto

List<E> toEntity(List<D> dto) данный метод преобразует лисе Entity в лист Dto

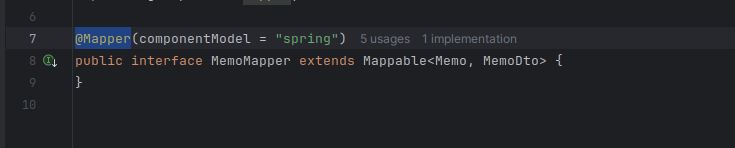


Рисунок 17: MemoMapper<Memo, MemoDto>

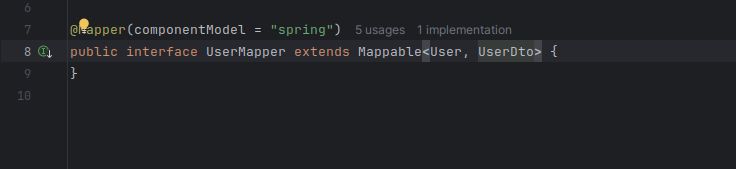


Рисунок 18: UserMapper<User, UserDto>

**4.7 Создание контроллера**

* Контроллер обрабатывает запросы от пользователя и взаимодействует с моделью и представлением.
* В Java Spring контроллеры представлены классами, аннотированными **@Controller** или **@RestController**.
* Контроллеры содержат методы для обработки различных типов запросов (GET, POST, PUT, DELETE) и вызывают соответствующие сервисы для выполнения бизнес-логики.

UserController является компонентом веб-приложения, отвечающим за взаимодействие с пользователем и обработку запросов, связанных с управлением записями конспектов. Контроллер предоставляет различные методы для обработки запросов пользователей и взаимодействия с соответствующими сервисами и использует сервисы UserService и MemoService для выполнения операций с данными.

Методы контроллера позволяют пользователю просматривать, добавлять, изменять и удалять записи в дневнике чувств и справочнике. Кроме того, контроллер обеспечивает перенаправление пользователя на соответствующие страницы для отображения результатов операций.

Методы контроллера обрабатывают различные HTTP-запросы, например, GET и POST, для отображения страниц, получения и обновления данных.



Рисунок 19: UserController

Для взаимодействия с пользователями UserController предоставляет методы:

public UserDto updated(@RequesBody final UserDto dto) данный метод обновляет поля пользователя, принимает @RequestBody который осуществляет преобразование входящего JSON/XML в объект Java UserDto, осуществляет обновление полей после чего возвращает ответ в виде обновленного объекта UserDto.

public UserDto getById(@PathVariable final Long id) данный метод осуществляет поиск пользователя по идентификатору id, для работы с параметрами, передаваемыми через адрес запроса в Spring WebMVC используется аннотация [@PathVariable](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html), осуществляет поиск после чего возвращает ответ в виде объекта UserDto.

public void deleteById(@PathVariable final Long id) данный метод осуществляет удаление пользователя по идентификатору id.

public List<MemoDto> getMemoByUserId(@PathVariable final Long id) данный метод осуществляет фильтрацию по идентификатору пользователя id, возвращает лист конспектов, который относится к данному пользователю.

public MemoDto createMemo(@PathVariable final Long id, @RequesBody final MemoDto dto) данный метод осуществляет создание нового конспекта с привязкой по идентификатору пользователя, к определенному пользователю.

MemoController является компонентом веб-приложения, отвечающим за взаимодействие с конспектами и обработку запросов, связанных с управлением записями конспектов.

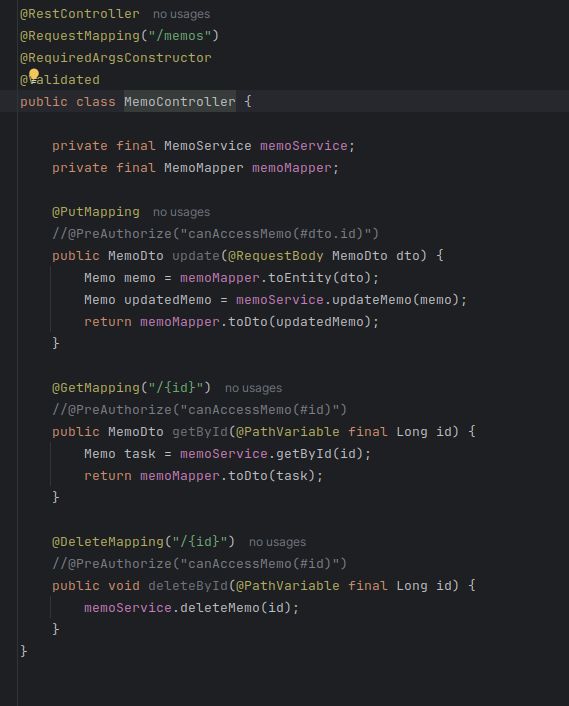


Рисунок 20: MemoController

Для взаимодействия с конспектами MemoController предоставляет методы:

public MemoDto updated(@RequesBody final MemoDto dto) данный метод обновляет поля конспекта, принимает @RequestBody который осуществляет преобразование входящего JSON/XML в объект Java MemoDto, осуществляет обновление полей после чего возвращает ответ в виде обновленного объекта MemoDto.

public MemoDto getById(@PathVariable final Long id) данный метод осуществляет поиск конспекта по идентификатору id, для работы с параметрами, передаваемыми через адрес запроса в Spring WebMVC используется аннотация [@PathVariable](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html), осуществляет поиск после чего возвращает ответ в виде объекта MemoDto.

public void deleteById(@PathVariable final Long id) данный метод осуществляет удаление конспекта по идентификатору id.

**4.8 Регистрация и авторизация с помощью Spring Security**

В приложении будет реализована регистрация новых пользователей и авторизация, ограничение доступа к страницам сайта в зависимости от роли пользователя. Самым фундаментальным объектом является SecurityContextHolder. В нем хранится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе, работающим с приложением. Spring Security использует объект Authentication, пользователя авторизованной сессии. «Пользователь» – это просто Object. В большинстве случаев он может быть приведен к классу UserDetails. UserDetails можно представить, как адаптер между БД пользователей и тем что требуется Spring Security внутри SecurityContextHolder.

Для того, чтобы в дальнейшим использовать класс User в Spring Security, он реализует интерфейс UserDetails. Для этого были переопределены все его методы.

Функции Spring Security:

* Аутентификация: Это процесс проверки подлинности пользователя. Spring Security поддерживает различные механизмы аутентификации, такие как форма, HTTP Basic, OAuth и другие.
* Авторизация: Это процесс определения разрешений пользователя на доступ к определенным ресурсам или выполнение определенных действий. Spring Security использует роли и разрешения для управления доступом.
* Фильтры безопасности: Spring Security использует цепочку фильтров для обработки запросов и применения правил безопасности. Эти фильтры обеспечивают аутентификацию, авторизацию, защиту от CSRF-атак, управление сессиями и другие функции.
* Защита от CSRF-атак: CSRF (межсайтовая подделка запросов) - это атака, при которой злоумышленник заставляет пользователя совершить нежелательное действие на веб-сайте, на который он авторизован. Spring Security предоставляет защиту от таких атак.
* Управление сессиями: Spring Security предоставляет возможности управления сессиями, включая защиту от смены сессии, контроль одновременных сессий и настройку таймаута сессии.
* Шифрование паролей: Spring Security советует хранить пароли в зашифрованном виде, предоставляя возможность использовать алгоритмы шифрования, такие как BCrypt, SCrypt и PBKDF2.
* Интеграция с Spring Framework: Spring Security интегрируется плавно с Spring Framework, что упрощает обеспечение безопасности в приложениях на основе Spring.
* Настройка и расширяемость: Spring Security предоставляет множество возможностей для настройки и расширения. Вы можете настраивать логику аутентификации и авторизации, создавать собственные фильтры безопасности и интегрироваться с сторонними сервисами аутентификации.
* Spring Security обеспечивает полную защиту вашего приложения на Java, делая его популярным выбором для реализации безопасности в проектах, использующих Spring.

**4.8.1 Реализация слоя доступа к данным и сервисного слоя**

Содержит методы для бизнес-логики приложения. Класс СustomUserDetailsService реализует интерфейс UserDetailsService (необходим для Spring Security), в котором был переопределен один метод loadUserByUsername().

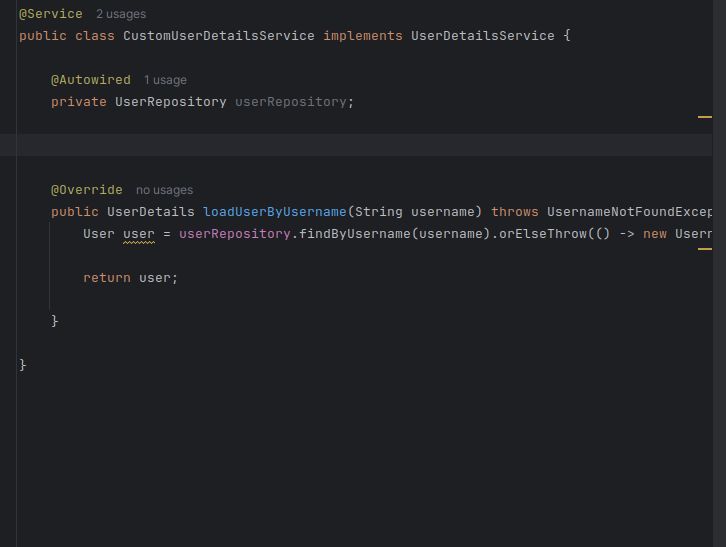


Рисунок 21: СustomUserDetailsService

**4.8.2 Добавление контроллеров**

Для страниц, которые никак не обрабатываются сервером, а просто возвращают страницу, маппинг можно настроить в конфигурации. Страница login обрабатывается Spring Security контроллером по умолчанию, поэтому для неё отдельный контроллер не требуется.

RegistrationController. Отдельный контроллер нужен для страницы регистрации. Для обработки GET запроса используется аннотация @GetMapping("/registration"), для POST – @PostMapping("/registration").

Чтобы что-то добавить или получить со страницы мы обращаемся к model. В GET запросе на страницу добавляется новый пустой объект класса User. Это сделано для того, чтобы при POST запросе не доставать данные из формы регистрации по одному (username, password, passwordConfirmation), а сразу получить заполненный объект userForm.

Метод addUser() в качестве параметров ожидает объект пользователя (userForm), который был добавлен при GET запросе. Аннотация Valid проверяет выполняются ли ограничения, установленные на поля. Если ограничения не были выполнены, то bindingResult будет содержать ошибки.

Если пароль и его подтверждение не совпадают добавляем сообщение на страницу и возвращаем её. В конце пробуем сохранить добавить пользователя в БД.

Метод saveUser() возвращает false, если пользователь с таким именем уже существует и true, если пользователь сохранен в БД. При неудачной попытке сохранения – добавляем сообщение об ошибке и возвращаем страницу. При удачном сохранении пользователя – переходим на главную страницу.

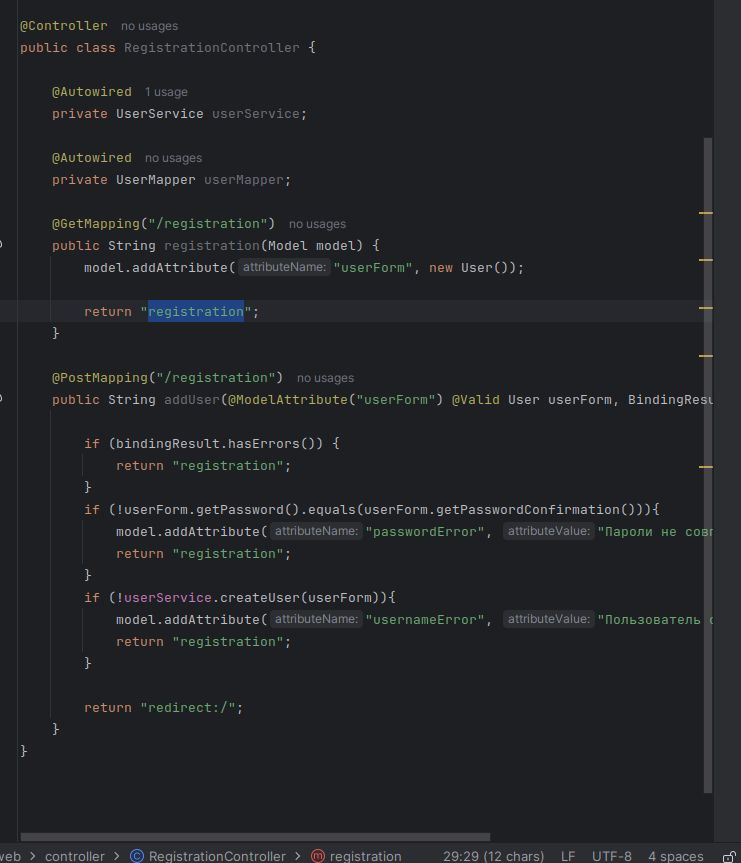


Рисунок 22: RegistrationController

AdminController. Доступ к странице admin имеют только пользователи с ролью администратора. В методе userList() нет ничего нового, он получает данные всех пользователей и добавляет их на страницу.

Метод deleteUser() использует аннотацию RequestParam т.е. в представлении будет форма, которая должна передать два параметра – userId и action. Ссылка будет иметь вид <http://localhost:8080/admin?userId=24&action=delete> при выполнении такого запроса будет удален пользователь с id=24.

Еще один вариант передать параметры в URL – использовать PathVariable. С помощью этой аннотации мы получаем отдельные части URL, для метод getUser() URL будет выглядеть следующим образом: <http://localhost:8080/admin/gt/24>, после перехода выведется список всех пользователей с id>24.

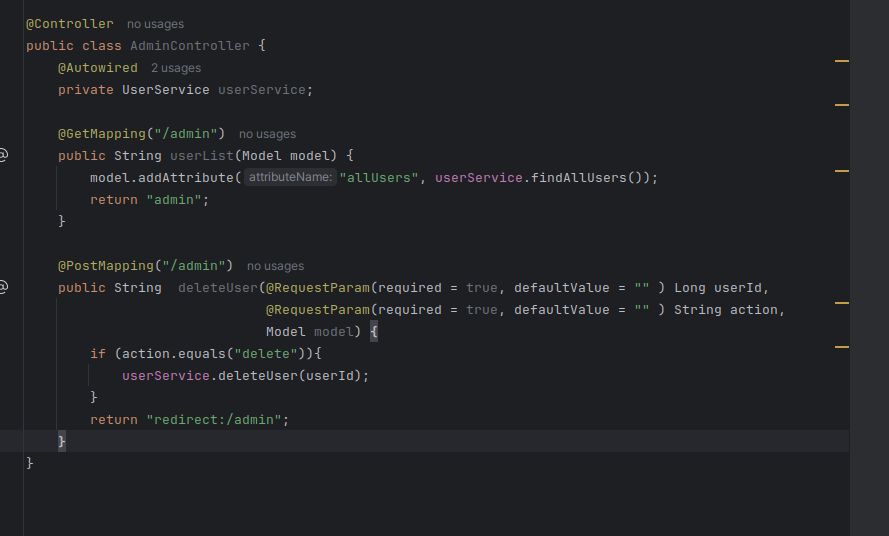


Рисунок 23 AdminController

**4.8.3 Настройки безопасности**

WebSecurityConfig. Содержит 2 бина BCryptPasswordEncoder и AuthenticationManager, которые, уже встречались ранее в классе userService.

Кроме этого в методе configure() настраивается доступ к различным ресурсам сайта. В качестве параметров метода requestMatchers() передаем пути, для которых хотим установить ограничение. Затем указываем, пользователям с какой ролью будет доступна эта страница/страницы.

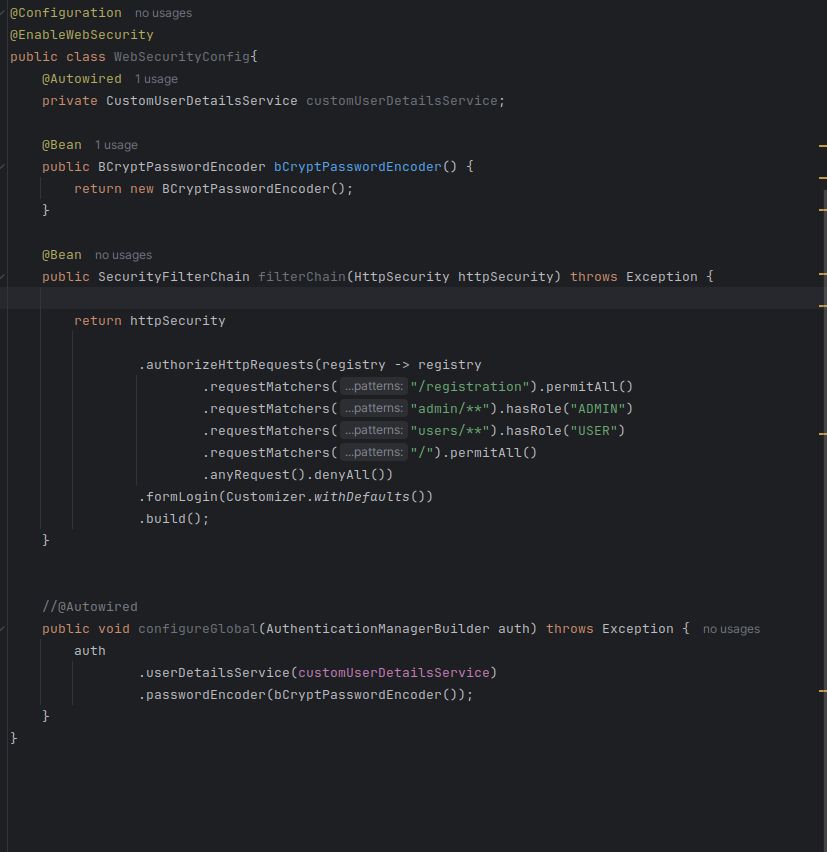


Рисунок 24: WebSecurityConfig

**4.9** **Документирование SpringBoot API с помощью Swagger**

Swagger позволяет разработчикам описывать структуру своих API и генерировать интерактивную документацию, клиентские библиотеки и серверные модули для реализации API на разных языках.

Swagger предоставляет спецификацию для документирования REST API, которая называется OpenAPI Specification (OAS). Эта спецификация предоставляет четкий и лаконичный способ описания эндпойнтов, их параметров, моделей запросов и ответов и других аспектов API.

Существуют библиотеки, которые на основе OAS могут сгенерировать интерактивную документацию для API, которая позволяет отправлять запросы, и получать ответы. В приложении используется библиотека SpringDoc.

Также Swagger позволяет сгенерировать непосредственно код клиента или сервера по имеющейся OAS, для этого нужен генератор кода [Swagger-Codegen](https://swagger.io/tools/swagger-codegen/).

Для добавления Swagger в проект необходимо добавить следующие зависимости в pom.xml:

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.0.2</version>

</dependency>

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webflux-ui</artifactId>

<version>2.0.2</version>

</dependency>

Swagger автоматически находит список всех контроллеров. При нажатии на любой из них будут перечислены допустимые методы HTTP (DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT).

Для каждого метода доступные следующие данные: статус ответа, тип содержимого и список параметров.

Поэтому после добавления зависимостей у нас уже есть документация, доступная по ссылке: [http://localhost:8080/swagger-ui](http://localhost:8080/swagger-ui.html). А также есть OAS, доступный по адресу: <http://localhost:8080/v3/api-docs>.

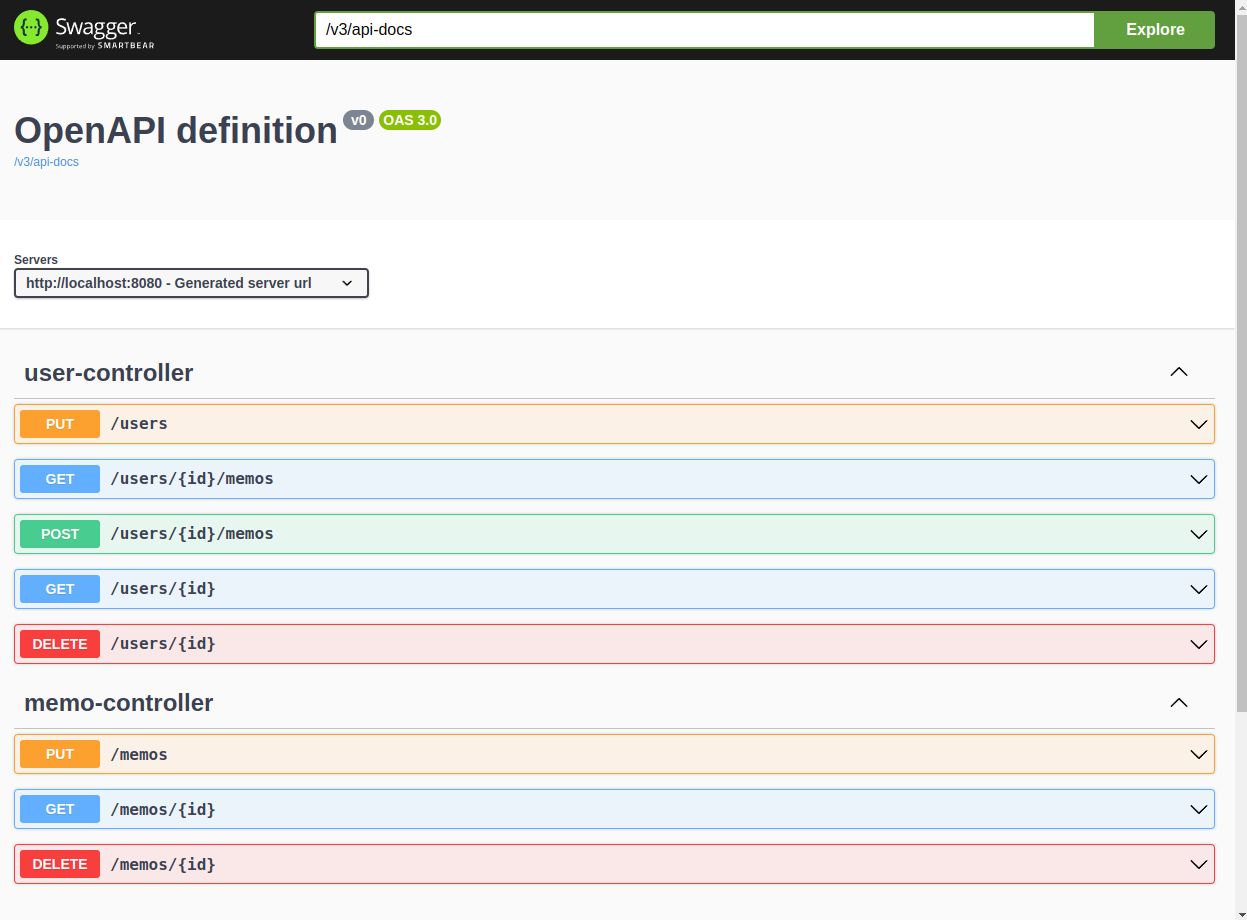


Рисунок 25: [http://localhost:8080/swagger-ui](http://localhost:8080/swagger-ui.html)

**Заключение**

В рамках данного дипломного проекта было разработано серверная часть веб-приложение «Книга конспектов», предназначенное для записи и отслеживания конспектов пользователей. Проект был реализован с использованием современных технологий веб-разработки, таких как Java Spring Framework.

В процессе разработки были достигнуты следующие цели:

* Реализация функционала добавления, редактирования и удаления записей о конспектов.
* Обеспечение безопасности данных пользователей с использованием механизмов аутентификации и авторизации Spring Security.
* Использование принципов SOLID и паттернов проектирования для обеспечения гибкости и расширяемости кода.

Дальнейшее развитие приложения может включать в себя добавление нового функционала, добавление пользовательского интерфейса, средств тестирования приложения и автотестов, также добавление функционала, по сохранению конспекта в документ, добавление изображений в конспект.

В целом, дипломный проект "Книга конспектов" позволил не только показать навыки разработки веб-приложений с использованием современных технологий, но и продемонстрировать понимание принципов безопасности и удобства использования для конечного пользователя.

Список используемой литературы:

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