Geekbrains

**Разработка веб приложения «Дневник эмоций» на JavaSpring.**

IT-специалист:

Разработка веб приложений

Шаповалов М.А.

г.Сочи

2024

**Дипломный проект**

**«Разработка веб приложения «Дневник эмоций» на JavaSpring»**

**Содержание**

|  |  |
| --- | --- |
| Введение……………………………………………………………………. | 3 |
| Глава 1. Предпроектная стадия…………………………………………… | 5 |
| 1.1. Порядок проведения работ…………………………………………… | 5 |
| 1.2. Определение требований к приложению…………………………… | 6 |
| 1.3. Выбор стека технологий……………………………………………… | 7 |
| Глава 2. Проектная стадия………………………………………………… | 10 |
| 2.1. Проектирование общей архитектуры приложения………………… | 10 |
| 2.2. Проектирование web-интерфейса приложения…………………….. | 22 |
| 2.3. Проектирование базы данных……………………………………….. | 29 |
| 2.4. Безопасность приложения……………………………………………. | 37 |
| 2.5. Разворачивание Docker контейнера…………………………………. | 51 |
| Заключение………………………………………………………………… | 52 |
| Список использованной литературы…………………………………….. | 53 |

**Введение**

В связи с тем, что в последнее время все больше людей увлекаются психологией и склонны к самоанализу, было решено создать веб приложение, в котором можно будет записывать свои эмоции как в дневник.

Приложение "Дневник эмоций" представляет собой удобный инструмент для отслеживания и анализа своих эмоциональных состояний. С его помощью пользователь может записывать свои эмоции, настроение и эмоции каждый день, делая пометки о причинах изменений в своем внутреннем мире.

«Дневник эмоций» помогает пользователям лучше понимать свои эмоции, отслеживать изменения своего настроения и делать выводы о факторах, влияющих на их психологическое состояние, а также находить способы эффективного управления своими эмоции.

«Дневник эмоций» предлагает удобный и интуитивно понятный интерфейс для записи ваших эмоций, возможность создания категорий для классификации эмоций.

Основная функциональность приложения включает в себя:

**Ежедневные записи**: Пользователь может создавать записи о своих эмоциях, описывать свои переживания и выражать свои мысли.

**Отслеживание настроения**: Приложение позволяет пользователю оценивать своё настроение каждый день, использовать разные шкалы (например, от 1 до 5 или отлично, хорошо, удовлетворительно, плохо, ужасно).

**Статистика и анализ**: Пользователь может просматривать статистику своего настроения за определенный период времени, а также анализировать связь между своими эмоциями и внешними факторами (какие события или обстоятельства влияют на его настроение).

**Уведомления и напоминания**: Приложение может отправлять пользователю уведомления, напоминая о важности записи своих эмоций и настроения.

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

Я убежден, что умение осознанно управлять своими эмоциями — это ключ к психологическому благополучию и гармонии с собой. Пользуйтесь "Дневником эмоций" для того, чтобы быть ближе к своим эмоциям и лучше понимать себя! Начало формы

**Глава 1. Предпроектная стадия**

**1.1. Порядок проведения работ**

В ходе создания проекта были проведены следующие виды работ:

1. **Планирование проекта**:
   * Определение функциональных и нефункциональных требований к приложению.
   * Проектирование архитектуры приложения и выбор используемых технологий.
2. **Настройка окружения разработки**:
   * Установка и настройка JavaDevelopmentKit (JDK), ApacheMaven (для сборки проекта), и SpringBoot (для создания и запуска приложения).
3. **Создание проекта**:
   * Создан новый проект с использованием SpringInitializr.
   * Выбраны необходимые зависимости, такие как SpringWeb, SpringDataJPA, Lombok, SpringSecurity, Thymeleaf, для поддержки функциональности приложения.
4. **Разработка модели данных, репозитория, сервисов, контроллера.**
5. **Разработка пользовательского интерфейса**:
   * Создание HTML-шаблонов с использованием Thymeleaf (или других шаблонизаторов) для отображения данных и взаимодействия с пользователем.
6. **Отладка и оптимизация**:
   * Отладка кода для исправления ошибок и улучшения производительности.
   * Оптимизация кода и конфигурации приложения для улучшения скорости работы и эффективности использования ресурсов.
7. **Документирование**:
   * Создание документации по проекту, включая описание архитектуры, инструкции по установке и запуску, описание API (если применимо) и другие справочные материалы.

**1.2. Определение требований к приложению**

Для разработки приложения "Дневник эмоций" были определены следующие требования:

1. Функциональные требования:
   * Пользователи должны иметь возможность создавать записи о своих эмоциях.
   * Приложение должно предоставлять возможность категоризации эмоций (например, радость, грусть, страх) для удобного анализа и отслеживания.
   * Пользователи должны иметь возможность просматривать и редактировать свои записи.
   * Приложение должно предоставлять возможность анализа и визуализации данных об эмоциях с течением времени.
   * Возможность получения советов и рекомендаций по управлению эмоциями на основе данных пользователя.
2. Нефункциональные требования:
   * Удобство использования: Интерфейс должен быть интуитивно понятным и привлекательным для пользователя.
   * Безопасность: Данные пользователей должны храниться и передаваться в зашифрованном виде.
   * Производительность: Приложение должно обеспечивать быстрый доступ к данным и отзывчивый интерфейс.
   * Масштабируемость: Возможность расширения функциональности и увеличения числа пользователей без значительного ухудшения производительности.
3. Требования к интерфейсу пользователя:
   * Интуитивно понятный интерфейс, который позволяет пользователям легко создавать и просматривать записи о своих эмоциях.
   * Возможность выбора цветовой схемы или темы интерфейса для создания комфортной атмосферы.
4. Требования к безопасности:
   * Аутентификация и авторизация пользователей.
   * Защита данных пользователей от несанкционированного доступа.
5. Требования к производительности:
   * Быстрый доступ к данным и отзывчивый интерфейс приложения.
6. Требования к масштабируемости:
   * Возможность добавления новых функций и поддержка роста числа пользователей.
7. Требования к поддержке и обновлениям:
   * Поддержка приложения и обновления для улучшения функциональности и безопасности.

**1.3. Выбор стека технологий**

1. **Язык программирования**: Java для разработки серверной части приложения так как этот язык хорошо подходит для создания масштабируемых и надежных веб-приложений.
2. **Фреймворк для веб-разработки**: SpringBoot с модулями Spring MVC и SpringSecurity для быстрой и удобной разработки серверной части приложения. SpringSecurity обеспечит безопасность приложения, а Spring MVC позволит создать удобный интерфейс.
3. **База данных**: PostgreSQL для хранения данных пользователей и их эмоций. Обе системы хорошо масштабируются и обеспечивают надежное хранение данных.
4. **Хранение данных об эмоциях**: Для хранения данных об эмоциях пользователей используется **PostgreSQL.**
5. **Фронтенд: HTML для создания интерфейса приложения. Для упрощения разработки интерфейса можно использовать фреймворк Bootstrap.**
6. **Шаблонизатор**: Thymeleaf для генерации HTML-страниц на сервере. Thymeleaf интегрируется хорошо со Spring и позволяет удобно работать с данными из Java кода.
7. **Аутентификация и авторизация**: SpringSecurity используется для обеспечения безопасности приложения, включая аутентификацию пользователей и управление доступом к ресурсам.
8. **Среда разработки**: IntelliJ IDEA для разработки серверной части приложения. Этот стек технологий обеспечивает разработку приложения "Дневник эмоций" с учетом требований к функциональности, производительности и безопасности.

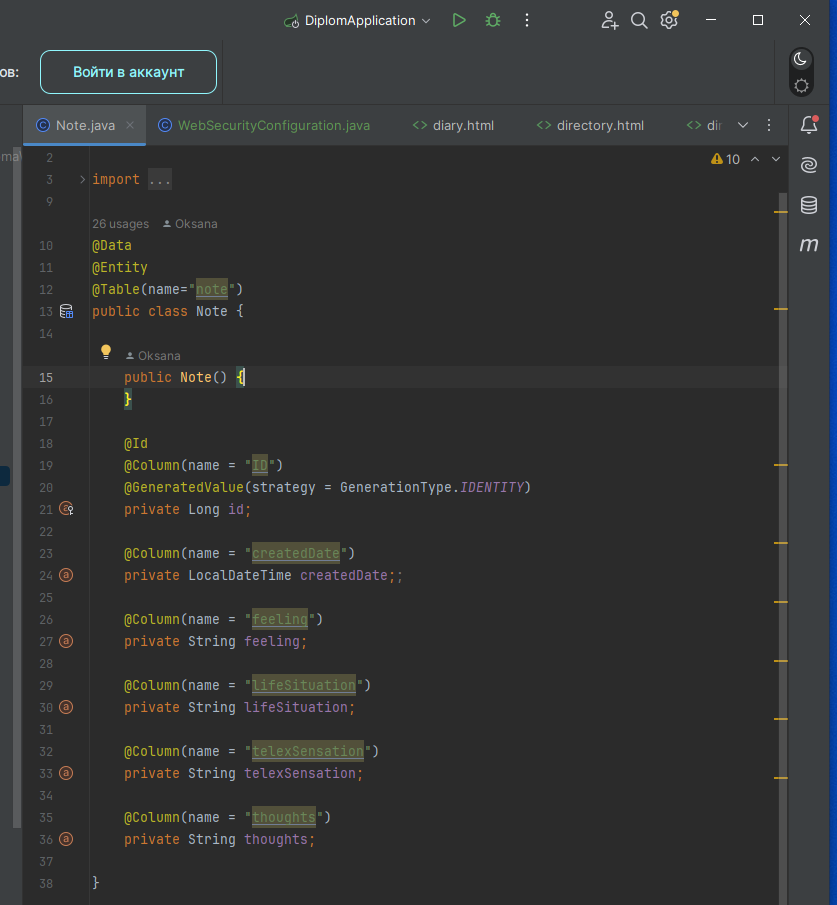
Архитектура приложения в рамках JavaSpring основывается на паттерне проектирования MVC (Model-View-Controller), который позволяет разделить приложение на три основных компонента: модель (Model), представление (View) и контроллер (Controller). Наше приложение состоит из данных составных частей:

1. **Модель (Model)**:
   * Модель представляет собой структуру данных приложения.
   * Включает классы, описывающие сущности приложения ("Пользователь", "Запись эмоций" ).
2. **Представление (View)**:
   * Представление отвечает за отображение данных пользователю.
   * В веб-приложениях на JavaSpring представление часто реализуется с использованием шаблонизаторов, таких как Thymeleaf или JSP, которые позволяют вставлять данные из контроллера непосредственно в HTML-шаблоны.
3. **Контроллер (Controller)**:
   * Контроллер обрабатывает запросы от пользователя и взаимодействует с моделью и представлением.
   * В JavaSpring 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.
4. **Репозиторий (Repository)**:
   * Репозиторий отвечает за доступ к данным из базы данных.
   * В JavaSpringрепозитории 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.
5. **Сервис (Service)**:
   * 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.
6. **Сервер (Server)**:
   * Сервер представляет собой окружение, в котором работает приложение.
   * В JavaSpring приложение обычно запускается на встроенном сервере, таком как Tomcat или Jetty, который обрабатывает HTTP-запросы и управляет жизненным циклом приложения.

Каждый из этих компонентов играет важную роль в построении архитектуры приложения на JavaSpring, обеспечивая его модульность, гибкость и масштабируемость. Они позволяют разделить различные аспекты приложения и обеспечить их независимую разработку и тестирование.

|  |
| --- |
| **Глава 2. Проектная стадия** |
| **2.1. Проектирование общей архитектуры приложения** |

Первое, с чего начиналась работа - это класс с инициализацией полей класса Note. Класс Note представляет собой запись в дневнике эмоций. Каждая запись содержит информацию о дате создания, ощущениях (feeling), жизненной ситуации (lifeSituation), телесных ощущениях (telexSensation) и мыслях (thoughts), которые переживает автор записи.



MyController является компонентом веб-приложения, отвечающим за взаимодействие с пользователем и обработку запросов, связанных с управлением записями дневника эмоций и справочника. Контроллер предоставляет различные методы для обработки запросов пользователей и взаимодействия с соответствующими сервисами и использует сервисы NoteService и NoteDiaryService для выполнения операций с данными.

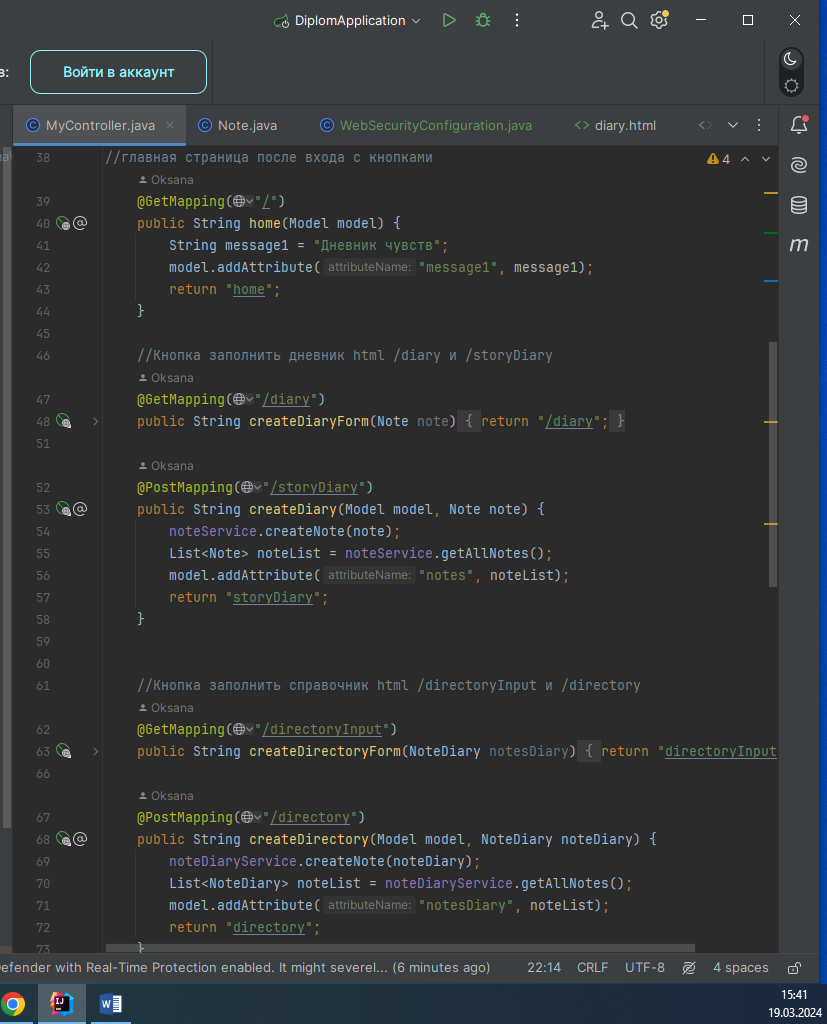
Методы контроллера позволяют пользователю просматривать, добавлять, изменять и удалять записи в дневнике эмоций и справочнике. Кроме того, контроллер обеспечивает перенаправление пользователя на соответствующие страницы для отображения результатов операций.

Для обеспечения безопасности приложения контроллер также взаимодействует с модулем безопасности (например, SpringSecurity), обеспечивая аутентификацию и авторизацию пользователей при доступе к различным функциям приложения.

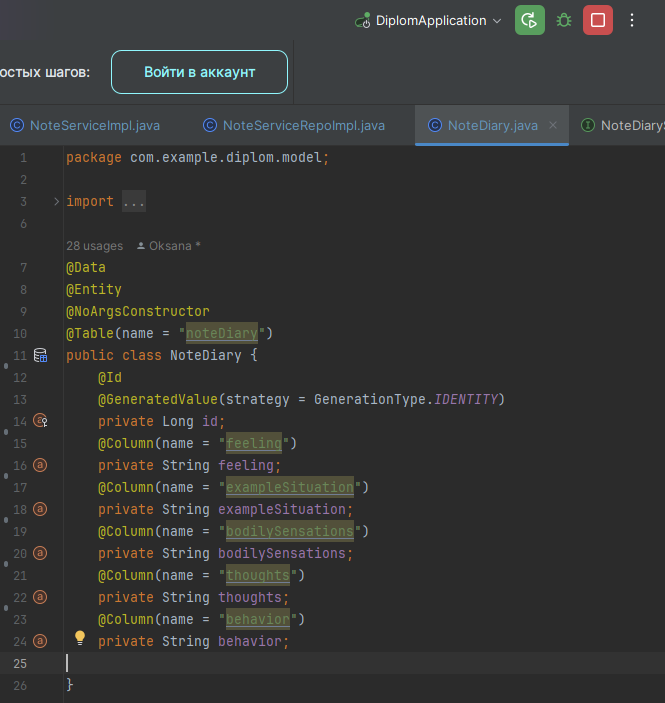
Методы контроллера обрабатывают различные HTTP-запросы, например, GET и POST, для отображения страниц, получения и обновления данных. Например, методы createDiaryForm и createDirectoryForm отображают формы для создания новых записей в дневнике и справочнике соответственно, а методы createDiary и createDirectory обрабатывают отправленные данные и добавляют новые записи в базу данных.

Кроме того, контроллер содержит методы для удаления записей (deleteNoteDiary) и обновления существующих записей (updateNoteDiary). Эти методы позволяют пользователям управлять своими данными и вносить изменения в уже существующие записи.

В целом, MyController является важной частью приложения, обеспечивающей удобный интерфейс для работы с данными дневника эмоций и справочника.



Класс NoteDiary представляет собой запись в справочнике, где пользователь может описывать свои эмоции, мысли, поведение и телесные ощущения в определенной ситуации. Каждая запись содержит уникальный идентификатор (id), тип эмоции (feeling), пример ситуации (exampleSituation), физические ощущения (bodilySensations), мысли (thoughts) и поведение (behavior).

Такая модель данных позволяет пользователям систематизировать и анализировать свои реакции на различные ситуации, что может помочь им лучше понять себя и свои эмоциональные реакции, а также научиться эффективнее управлять своим поведением и эмоциями.

Класс UserService в данном контексте является сервисом, предоставляющим функционал для работы с пользователями. Давайте рассмотрим его подробнее:

**@Service**: Аннотация @Service является частью SpringFramework и используется для пометки классов, которые выполняют бизнес-логику, обычно связанную с обработкой данных.

**@AllArgsConstructor**: Это аннотация из библиотеки Lombok, которая генерирует конструктор, принимающий все поля класса в качестве аргументов. Этот конструктор обычно используется для внедрения зависимостей через конструктор.

**privateUserRepositoryuserRepository**: Поле userRepository является зависимостью класса UserService и предполагается, что Spring внедряет его экземпляр в этот класс через конструктор.

**privatePasswordEncoderpasswordEncoder**: Поле passwordEncoder также является зависимостью класса UserService и используется для кодирования паролей пользователей.

**publicbooleanaddUser(Useruser)**: Метод addUser принимает объект User в качестве аргумента и возвращает булево значение, указывающее на успешность операции добавления пользователя.

**Stringname = user.getUsername()**: Получение имени пользователя из объекта user.

**if (userRepository.findByUsername(name).isPresent()) returnfalse**: Проверка, существует ли уже пользователь с таким именем. Если пользователь уже существует, метод возвращает false, что означает, что операция добавления пользователя не выполнена.

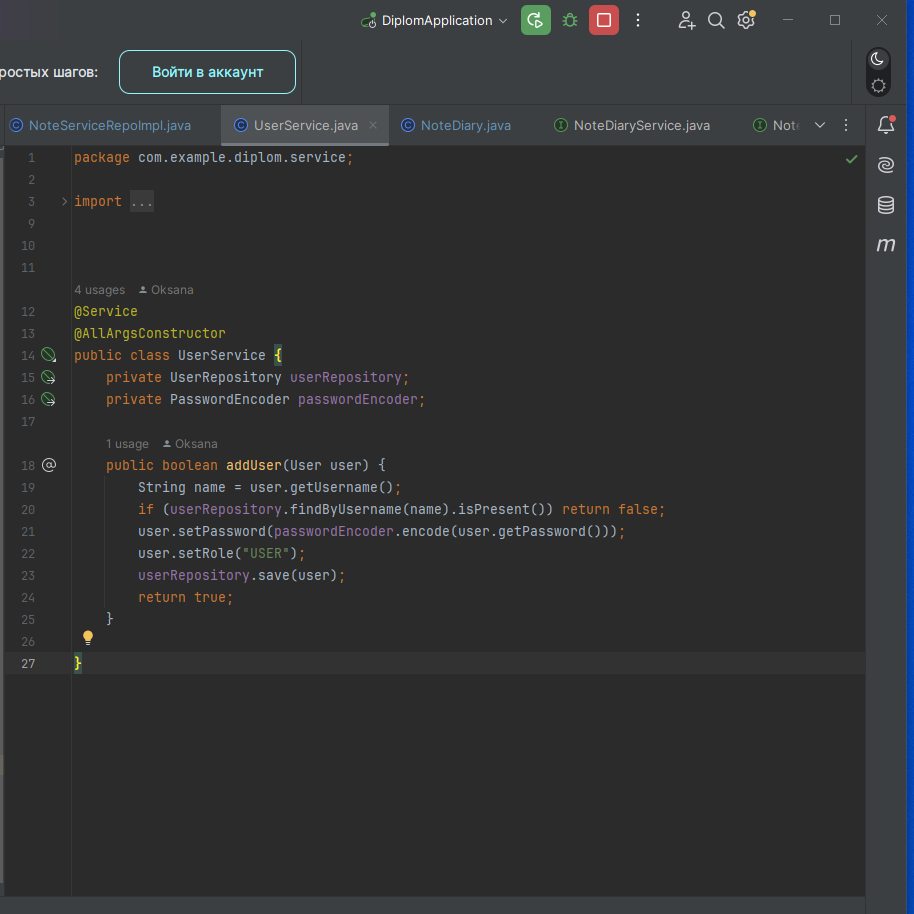
**user.setPassword(passwordEncoder.encode(user.getPassword()))**: Кодирование пароля пользователя с использованием passwordEncoder.

**user.setRole("USER")**: Установка роли пользователя. В данном случае, устанавливается роль "USER".

**userRepository.save(user)**: Сохранение пользователя в репозитории.

**returntrue**: Возвращение true, чтобы указать, что операция добавления пользователя выполнена успешно.

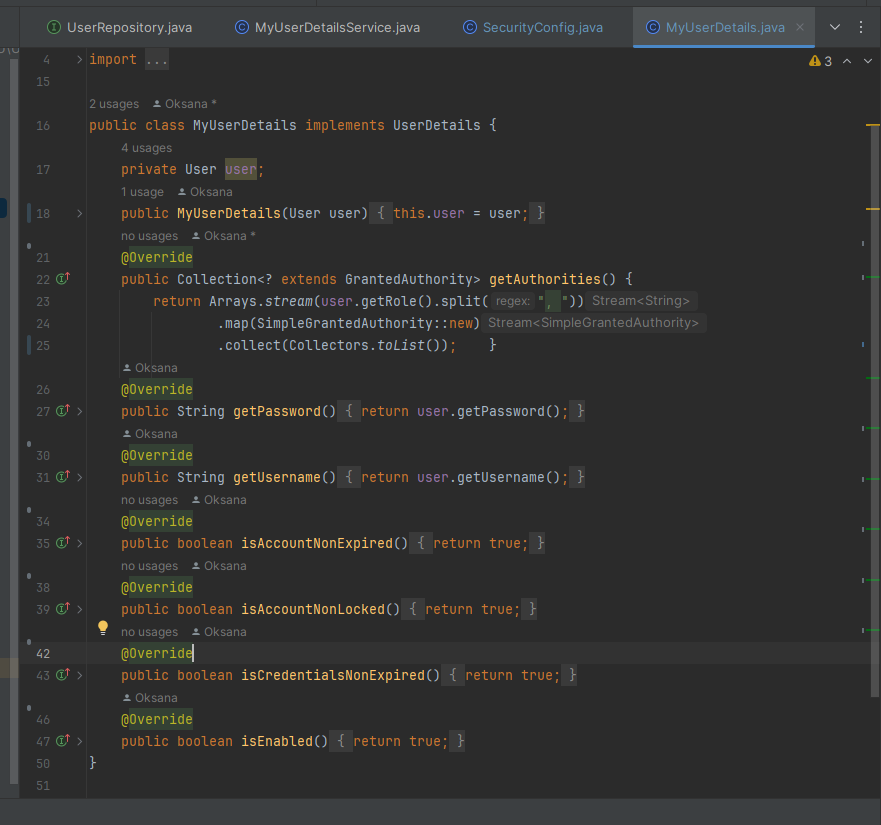
Класс UserService предоставляет метод addUser, который добавляет нового пользователя с уникальным именем, кодирует его пароль и устанавливает роль "USER". Если пользователь с таким именем уже существует, операция добавления завершается неудачей.



КлассMyUserDetails реализует интерфейс UserDetails, который используется SpringSecurity для представления информации о пользователе, необходимой для аутентификации и авторизации. Вот его функции:

1. privateUseruser; - поле, хранящее объект User, который представляет собой модель пользователя вашего приложения.
2. publicMyUserDetails(Useruser){...} - конструктор класса, принимающий объект User и сохраняющий его в поле user.
3. getAuthorities() - возвращает коллекцию ролей (GrantedAuthority), которые имеет пользователь. В данном случае, роли берутся из поля role объекта User и преобразуются в GrantedAuthority.
4. getPassword() и getUsername() - возвращают пароль и имя пользователя соответственно, из объекта User.
5. isAccountNonExpired(), isAccountNonLocked(), isCredentialsNonExpired(), isEnabled() - методы, возвращающие булево значение, указывающее, истек ли срок действия учетной записи, заблокирована ли учетная запись, устарели ли учетные данные и включена ли учетная запись. В данном случае, все они всегда возвращают true, что означает, что учетная запись всегда действительна, не заблокирована, учетные данные не устарели и учетная запись всегда включена.

Этот класс используется вместе с вашим UserDetailsService, чтобы предоставить



Класс NoteServiceImpl является реализацией интерфейса NoteService и предоставляет функционал по работе с заметками. Аннотация @Service говорит о том, что этот класс является сервисным компонентом, который предоставляет бизнес-логику для работы с данными. Аннотация @RequiredArgsConstructor генерирует конструктор с аргументами для всех полей класса, что позволяет Spring автоматически внедрить зависимости.

Метод createNote создает новую заметку, устанавливая текущую дату и время создания, после чего сохраняет заметку в репозитории.

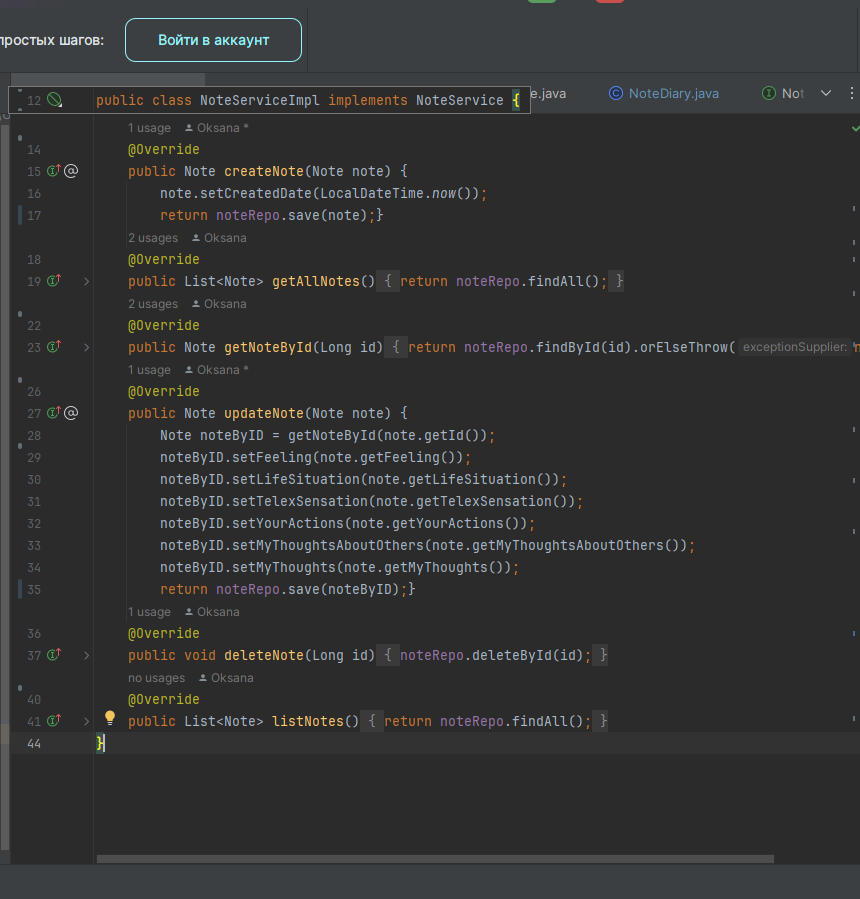
Метод getAllNotes возвращает список всех заметок из репозитория.

Метод getNoteById возвращает заметку по её идентификатору. Если заметка не найдена, выбрасывается исключение.

Метод updateNote обновляет существующую заметку, используя данные переданной заметки. Для этого сначала получается заметка по идентификатору, затем её поля обновляются значениями из переданной заметки и сохраняется обновленная заметка.

Метод deleteNote удаляет заметку по её идентификатору.

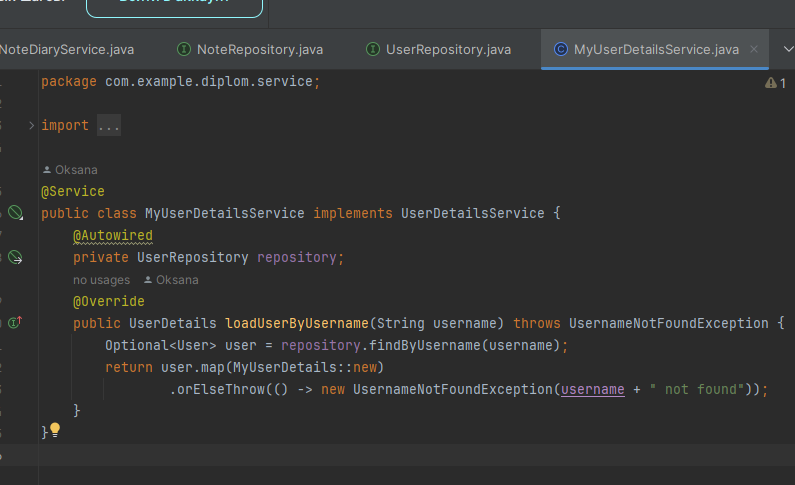
Метод listNotes также возвращает список всех заметок из репозитория.



Аннотация @Autowired в Spring используется для автоматического внедрения зависимостей. Когда Spring встречает это объявление поля или метода, он пытается найти подходящий бин (компонент), который соответствует этому типу, и автоматически внедряет его в это поле или метод.

В нашем случае, поле UserRepositoryrepository помечено аннотацией @Autowired, что означает, что Spring будет искать компонент, реализующий интерфейс UserRepository, и автоматически внедрять его экземпляр в это поле.

Метод loadUserByUsername используется в Spring Security для загрузки пользовательских данных (UserDetails) по имени пользователя. В данном случае, он пытается найти пользователя в репозитории по имени пользователя. Если пользователь найден, создается объект MyUserDetails, который является пользовательской реализацией интерфейса UserDetails, содержащий информацию о пользователе. Если пользователь не найден, выбрасывается исключение UsernameNotFoundException.



Аннотация @Service в Spring обозначает, что класс является сервисом, который содержит бизнес-логику приложения. Она используется для пометки классов, которые предоставляют различные сервисы, такие как обработка данных, выполнение бизнес-логики и управление транзакциями.

В нашем случае, класс NoteServiceRepoImpl помечен аннотацией @Service, что указывает, что этот класс представляет собой сервис для работы с записями (notediary). Аннотация @RequiredArgsConstructor генерирует конструктор с аргументами для всех полей класса, что позволяет Spring автоматически внедрить зависимости.

Класс NoteServiceRepoImpl реализует интерфейс NoteDiaryService и предоставляет следующие методы:

createNote - создает новую запись в дневнике, сохраняя её в репозитории.

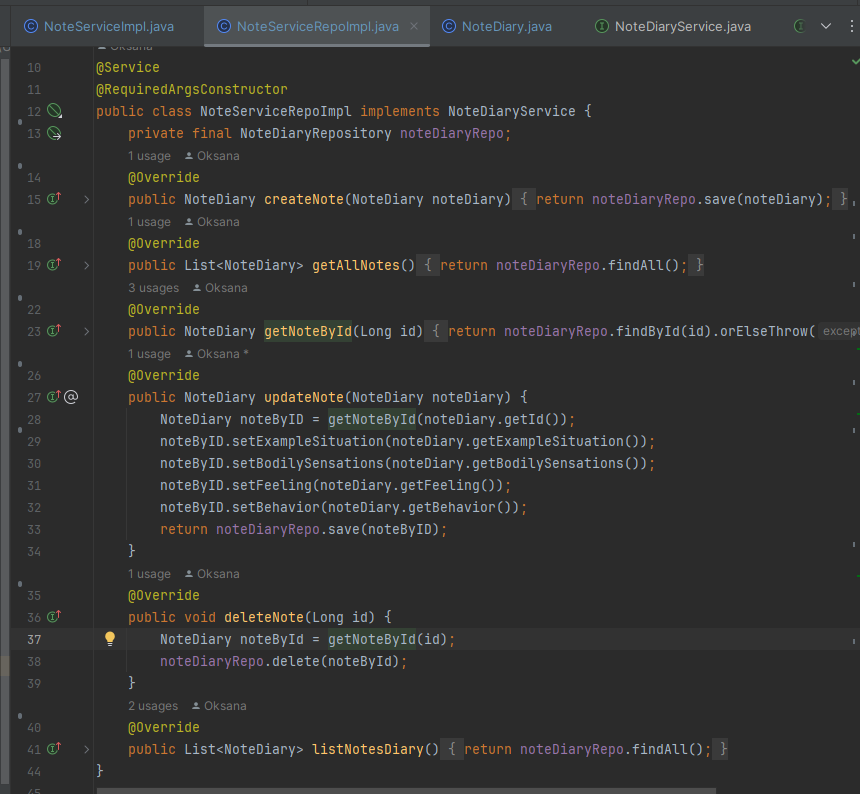
getAllNotes - возвращает список всех записей из дневника.

getNoteById - возвращает запись по её идентификатору. Если запись не найдена, выбрасывается исключение.

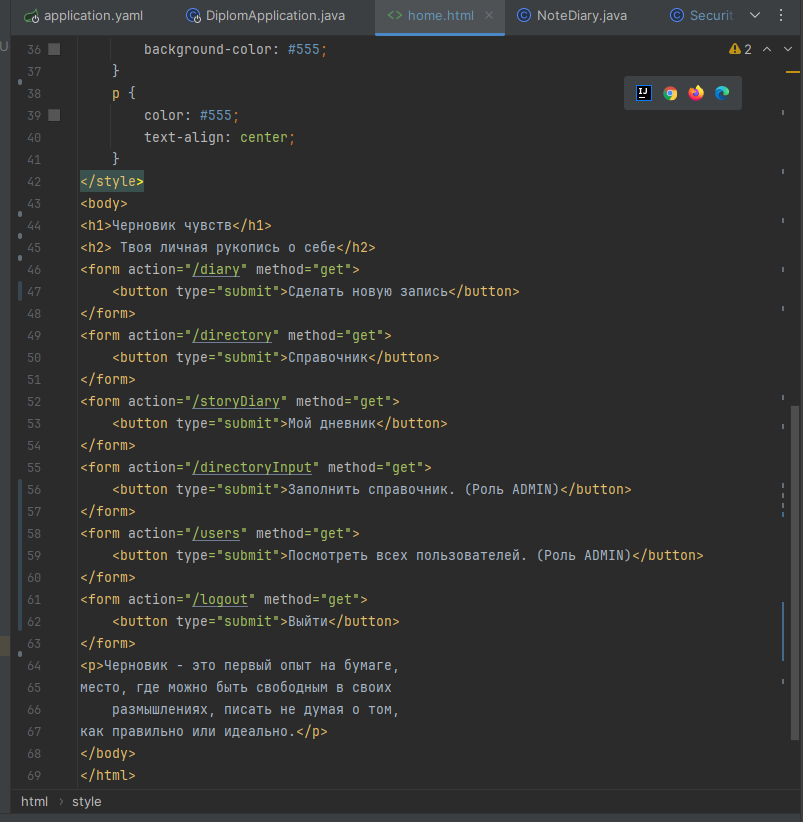
updateNote - обновляет существующую запись в дневнике, используя данные переданной записи. Для этого сначала получается запись по идентификатору, затем её поля обновляются значениями из переданной записи и сохраняется обновленная запись.

deleteNote - удаляет запись по её идентификатору.

listNotesDiary - возвращает список всех записей из дневника.



**2.2. Проектирование WEB – интерфейса приложения**

Для проектирования WEB – интерфейса приложения был использован HTML. После прохождения авторизации пользователь попадает на страницу home.html.

Данный HTML-код представляет собой шаблон для веб-страницы, использующей Thymeleaf в качестве шаблонизатора. Thymeleaf позволяет создавать динамические веб-страницы, вставляя данные из модели Java в HTML-шаблон. В данном случае, предполагается, что на сервере настроены контроллеры для обработки запросов, указанных в атрибуте action форм.

Давайте разберем основные элементы данного шаблона:

<!DOCTYPEhtml>: Определяет тип документа и его версию (HTML5).

<htmlxmlns:th="http://www.thymeleaf.org">: Указывает на использование Thymeleaf в качестве шаблонизатора и определяет пространство имен для Thymeleaf.

<head>: Содержит метаданные страницы, такие как заголовок (<title>).

<style>: Секция стилей для определения внешнего вида элементов страницы.

<body>: Основная часть страницы, содержащая контент, видимый пользователю.

Формы (<form>): Используются для создания кнопок, которые выполняют определенные действия при нажатии. Атрибут action указывает на адрес, куда будет отправлен запрос при отправке формы, а method определяет HTTP-метод (GET или POST).

Кнопки (<button>): Содержатся внутри форм и используются для выполнения определенных действий на сервере при их нажатии.

Текстовый контент (<h1>, <h2>, <p>): Отображают заголовки и текстовую информацию на странице.

Этот шаблон предполагает наличие на сервере контроллеров, обрабатывающих запросы на /diary, /directory, /storyDiary, /directoryInput, /users и /logout, и возвращающих соответствующие HTML-шаблоны или данные.Начало формы

На этой странице предлагается выбор соответствующих действий:

- сделать новую запись;

- открыть справочник;

- открыть мой дневник;

- заполнить справочник (Роль Admin);

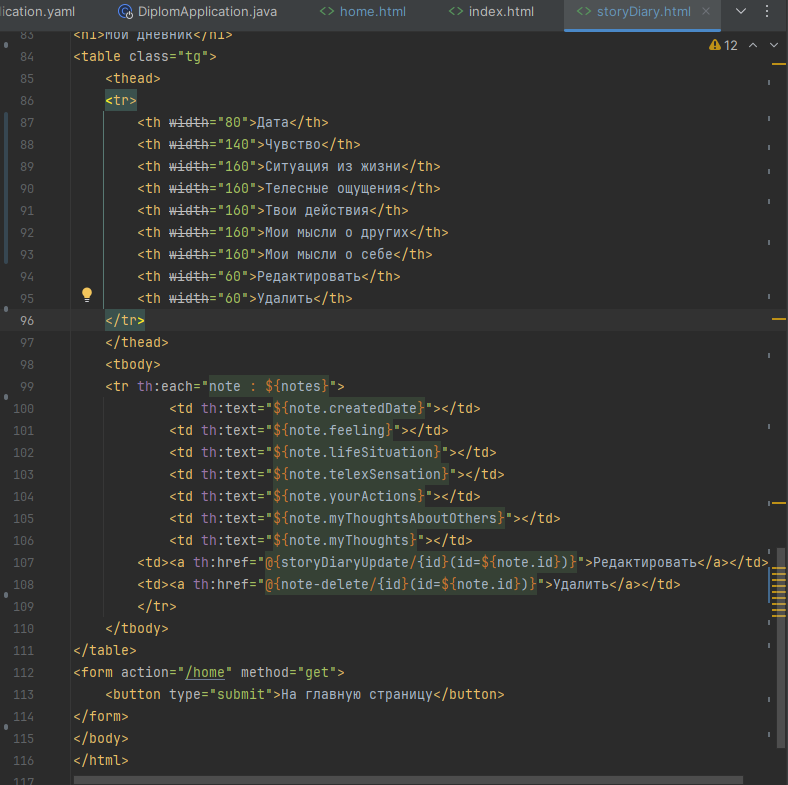
- посмотреть всех пользователей (Роль Admin);

- выйти.

С каждой страницы осуществлен возврат на главную страницу.

При нажатии кнопки «Сделать новую запись» открывается страница «Мои эмоции», где выходят несколько полей, которые нужно заполнить и сохранить.

HTML-код данной страницы заполнения использует Thymeleaf для динамической генерации таблицы с данными записей и кнопки для возврата на главную страницу.



Давай разберем основные элементы данного шаблона:

1. <h1>: Заголовок страницы.
2. <table>: Таблица для отображения данных записей. Каждая запись представлена строкой <tr>, где данные каждой колонки заполняются с помощью Thymeleaf итерации (th:each="note : ${notes}"), где ${notes} представляет список записей. Каждая колонка таблицы соответствует определенному полю записи, такому как дата (${note.createdDate}) или эмоцийо (${note.feeling}).
3. Ссылки <a>: Используются для редактирования и удаления записей. Ссылки генерируются с помощью Thymeleaf, где th:href="@{storyDiaryUpdate/{id}(id=${note.id})}" создает ссылку для редактирования записи с определенным идентификатором, а th:href="@{note-delete/{id}(id=${note.id})}" - для удаления.
4. <form>: Форма для кнопки, возвращающей пользователя на главную страницу. Кнопка отправляет GET-запрос на /home.

Этот шаблон предполагает, что на сервере есть контроллеры, обрабатывающие запросы для обновления и удаления записей дневника, а также возвращения на главную страницу.

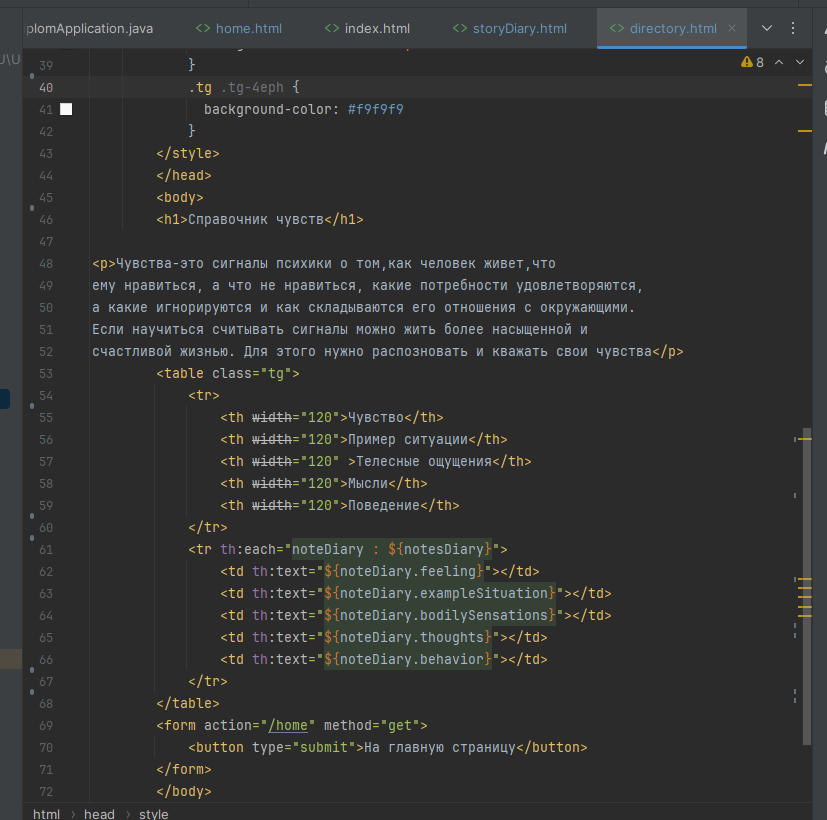
Этот HTML-код представляет собой шаблон страницы для отображения списка записей дневника. Он использует Thymeleaf для динамической генерации таблицы с данными записей и кнопки для возврата на главную страницу.

Давай разберем основные элементы данного шаблона:

1. <h1>: Заголовок страницы.
2. <table>: Таблица для отображения данных записей. Каждая запись представлена строкой <tr>, где данные каждой колонки заполняются с помощью Thymeleaf итерации (th:each="note : ${notes}"), где ${notes} представляет список записей. Каждая колонка таблицы соответствует определенному полю записи, такому как дата (${note.createdDate}) или эмоцийо (${note.feeling}).
3. Ссылки <a>: Используются для редактирования и удаления записей. Ссылки генерируются с помощью Thymeleaf, где th:href="@{storyDiaryUpdate/{id}(id=${note.id})}" создает ссылку для редактирования записи с определенным идентификатором, а th:href="@{note-delete/{id}(id=${note.id})}" - для удаления.
4. <form>: Форма для кнопки, возвращающей пользователя на главную страницу. Кнопка отправляет GET-запрос на /home.

Этот шаблон предполагает, что на сервере есть контроллеры, обрабатывающие запросы для обновления и удаления записей дневника, а также возвращения на главную страницу.

Далее следующий HTML-код представляет собой шаблон страницы для отображения справочника эмоций. Он использует Thymeleaf для динамической генерации таблицы с данными о эмоциих и их проявлениях, а также содержит небольшой текстовый блок с информацией о том, как правильно распознавать и уважать свои эмоции.



Разберем основные элементы данного шаблона:

<h1>: Заголовок страницы.

<p>: Текстовый блок с информацией о эмоциих.

<table>: Таблица для отображения данных о эмоциих. Каждая запись представлена строкой <tr>, где данные каждой колонки заполняются с помощью Thymeleaf итерации (th:each="noteDiary : ${notesDiary}"), где ${notesDiary} представляет список записей. Каждая колонка таблицы соответствует определенному полю записи, такому как эмоцийо (${noteDiary.feeling}) или телесные ощущения (${noteDiary.bodilySensations}).

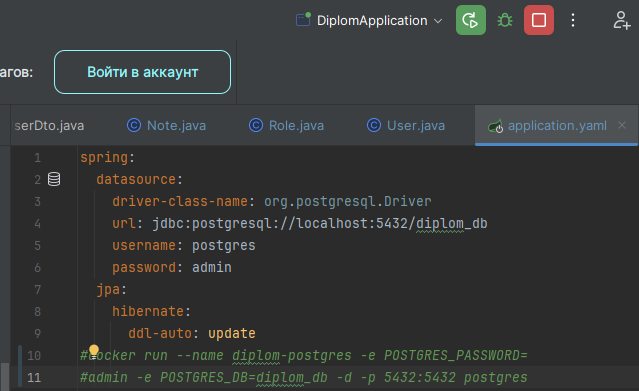
<button>: Кнопка для возврата на главную страницу.

Этот шаблон предполагает, что на сервере есть контроллер для обработки запроса на возврат на главную страницу (/home) и что список записей о эмоциих (${notesDiary}) доступен в модели Thymeleaf.

Следующая кнопка в списке страницы /homeидет кнопка «Мой дневник». Нажав эту кнопку можно сразу попасть на страницу с таблицей эмоций. На ней отображаются дата создания записи, и само описание эмоций. Также на нее сделан переход со страницы заполнения «Сделать новую запись».

**2.3. Проектирование базы данных**

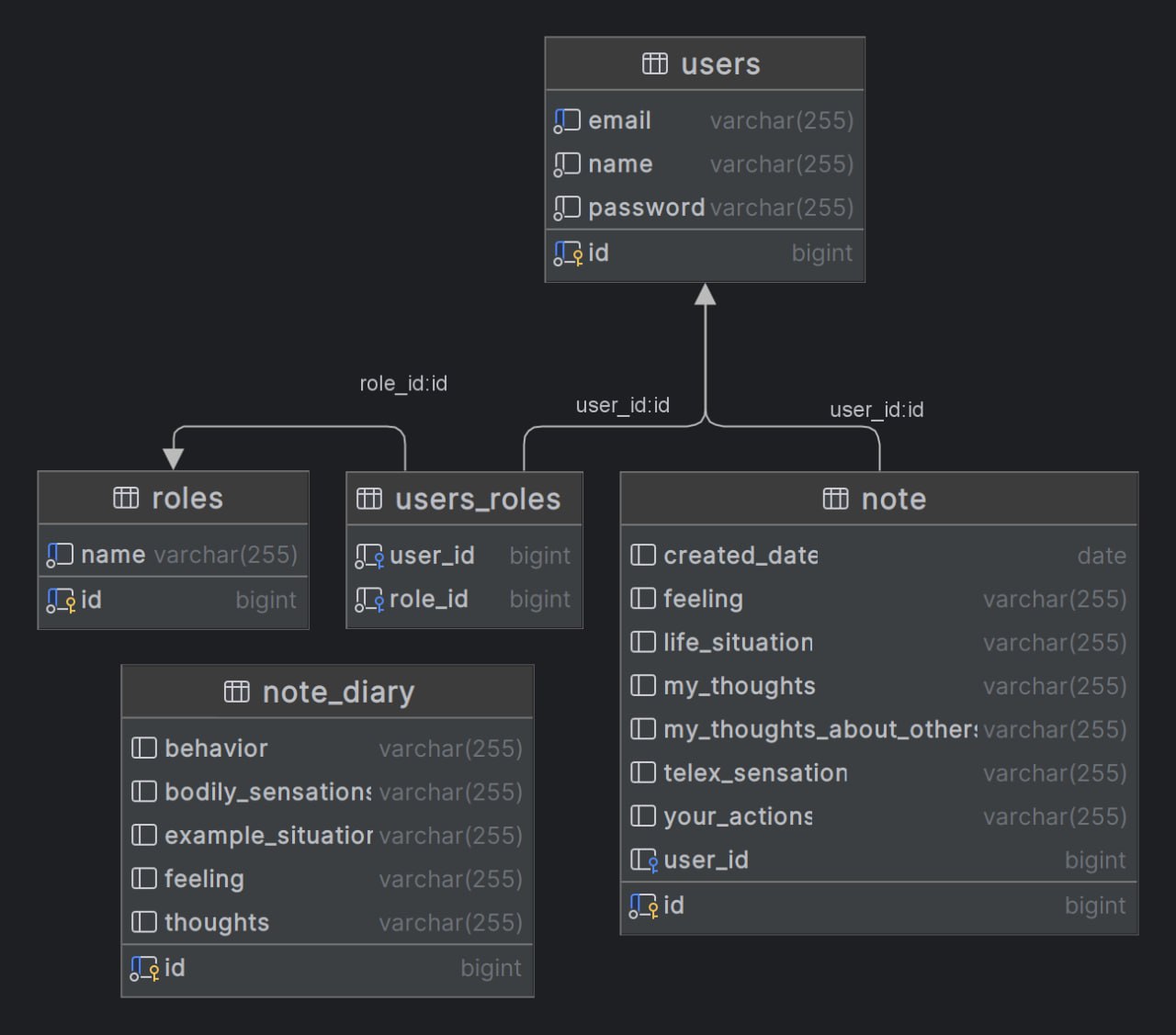
В качестве базы данных для этого проекта был выбран PostgreSQL. В файле application.yamlуказаны следующие настройки.



В данном примере конфигурации SpringBoot для подключения к базе данных PostgreSQL через JPA и Hibernate используется следующая конфигурация:

1. datasource: Содержит настройки для подключения к базе данных. driver-class-name указывает класс драйвера PostgreSQL. url содержит URL-адрес для подключения к базе данных. username и password содержат учетные данные пользователя базы данных.
2. jpa: Содержит настройки JPA и Hibernate. hibernate.ddl-auto определяет стратегию автоматического создания таблиц в базе данных. Значение update указывает Hibernate на автоматическое обновление схемы базы данных при изменении сущностей JPA.

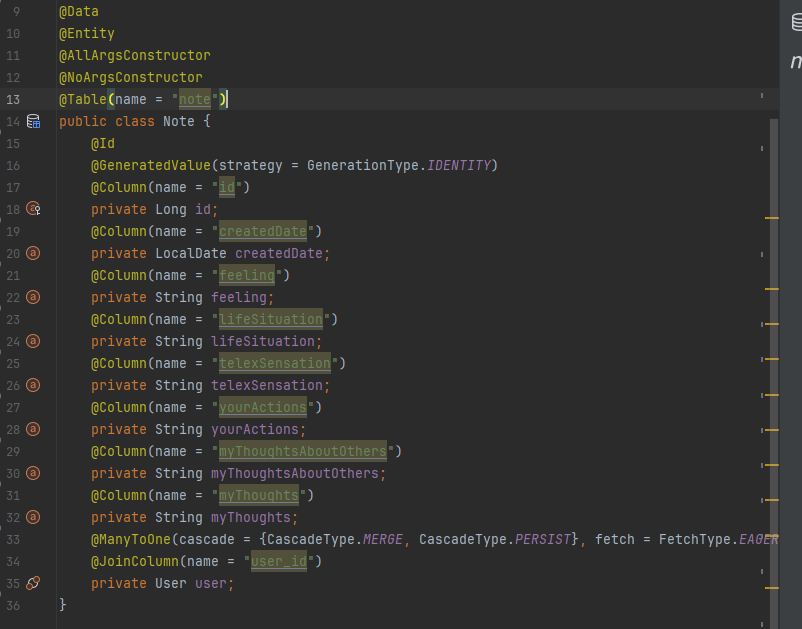
В общем, схема взаимодействия таблиц сделана таким образом:



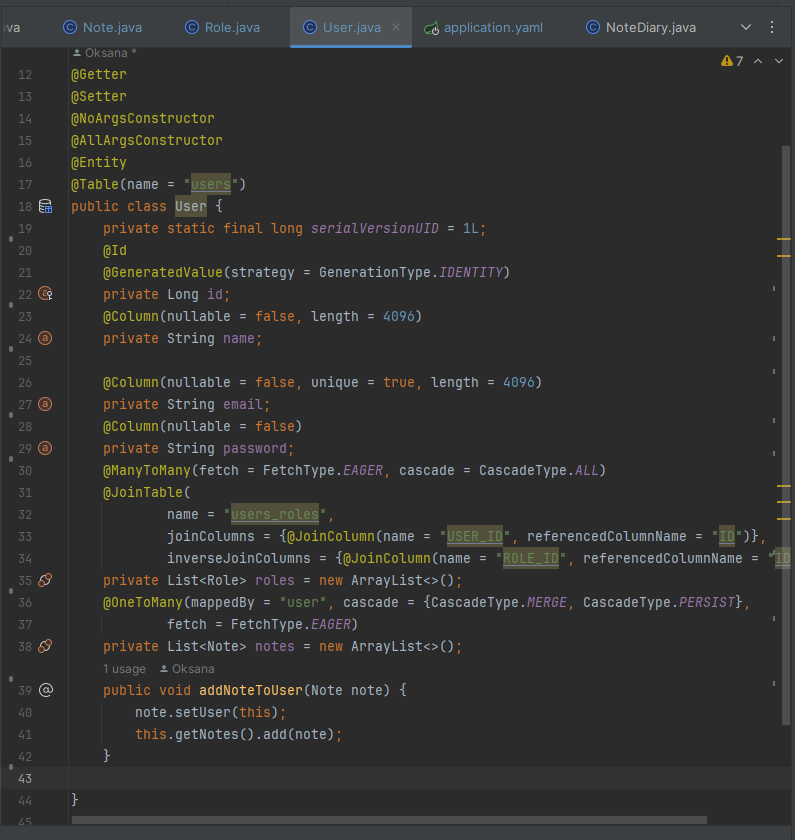
В ходе разработки базы данных в класс Note были добавлены:

1. @ManyToOne: Определяет отношение многие к одному между записью и пользователем (User). Тоестьодинпользовательможетиметьмногозаписей.
2. @JoinColumn(name = "user\_id"): Указывает на столбец в базе данных, который используется для хранения внешнего ключа, связывающего запись с пользователем.
3. privateUseruser;Поле для хранения пользователя, связанного с записью.

Этот класс является частью модели данных вашего приложения и используется для создания таблицы в базе данных и маппинга данных между Java-объектами и таблицами в базе данных.



Далее рассмотрим новый созданный класс User. Этот код представляет собой сущность (Entity) JPA для хранения информации о пользователях (User) в базе данных.



Разберем каждую аннотацию и поле:

@Entity: Обозначает, что класс является сущностью JPA.

@Id: Обозначает поле, которое является первичным ключом в базе данных.

@GeneratedValue: Указывает, как будет генерироваться значение для первичного ключа. В данном случае GenerationType.IDENTITY используется для автоматической генерации значения идентификатора базой данных.

@Column: Определяет атрибуты столбца в базе данных, такие как имя столбца (name), обязательность (nullable) и длину (length).

privateLongid;: Поле для хранения идентификатора пользователя.

privateStringname;: Поле для хранения имени пользователя.

privateStringemail;: Поле для хранения адреса электронной почты пользователя. Уникальность адреса почты обеспечивается атрибутом unique = true.

privateStringpassword;: Поле для хранения пароля пользователя.

@ManyToMany: Определяет отношение многие ко многим между пользователями и ролями (Role). Каждый пользователь может иметь несколько ролей.

@JoinTable: Определяет таблицу, которая используется для хранения связей между пользователями и ролями.

@OneToMany: Определяет отношение один ко многим между пользователем и записями (Note). Каждый пользователь может иметь несколько записей.

privateList<Role>roles = newArrayList<>();: Поле для хранения списка ролей пользователя.

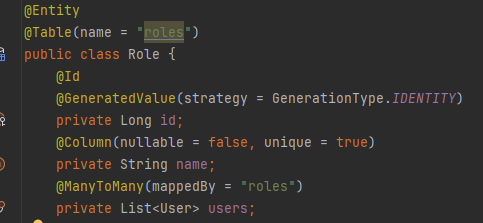
privateList<Note>notes = newArrayList<>();: Поле для хранения списка записей пользователя.

publicvoidaddNoteToUser(Notenote) { ... }: Метод для добавления записи к пользователю. Метод устанавливает пользователя для записи и добавляет запись в список записей пользователя.

Этот класс является частью модели данных нашего приложения и используется для создания таблицы в базе данных и маппинга данных между Java-объектами и таблицами в базе данных.

Также создан новый класс Role. Рассмотрим его подробнее.

Аннотации @Entity и @Table относятся к JPA и используются для обозначения класса как сущности, которая будет отображена в таблицу базы данных. @Entity указывает, что класс является сущностью JPA, а @Table(name = "roles") указывает на имя таблицы, в которую будет отображаться эта сущность.



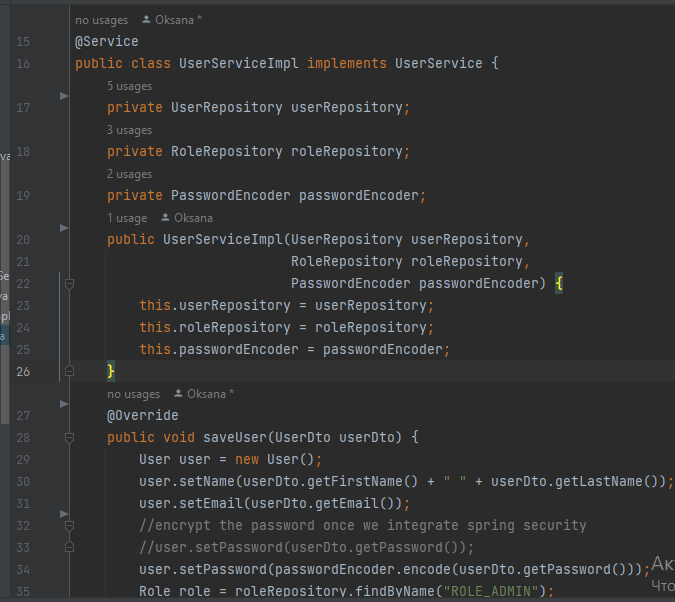
Класс Role представляет собой сущность, представляющую роль пользователя в системе. Он содержит поля для идентификатора роли (id), имени роли (name) и списка пользователей, связанных с этой ролью (users). Отношение многие ко многим между пользователями и ролями обозначается аннотацией @ManyToMany(mappedBy = "roles"), указывающей на поле roles в классе User, которое является владельцем этой связи.

Далее создан класс UserServiceImpl.

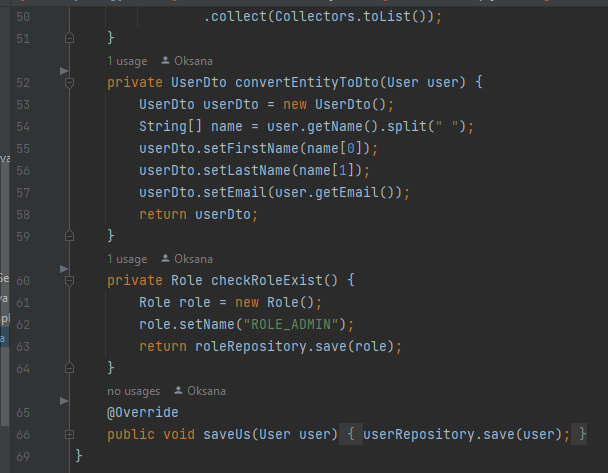
Пример класса UserServiceImpl, помеченного аннотацией @Service, показывает реализацию интерфейса UserService, который, вероятно, определяет операции для работы с пользователями. Конструктор этого класса принимает несколько зависимостей, таких как UserRepository, RoleRepository и PasswordEncoder, которые инжектируются контейнером Spring через внедрение зависимостей (Dependency Injection).

Методы в этом классе реализуют различные операции, такие как сохранение пользователя, поиск пользователя по электронной почте, получение списка всех пользователей и другие. Внутри методов выполняются операции над данными, такие как сохранение, поиск и преобразование данных в DTO (Data Transfer Object).

Таким образом, классы, помеченные аннотацией @Service, являются важной частью приложения на Spring, предоставляя службы и функциональность для обработки бизнес-логики и операций с данными.







**UserServiceImp** представляет службу для работы с пользователями. Этот класс содержит методы для сохранения пользователя, поиска пользователя по электронной почте и преобразования сущностей пользователей в DTO (Data Transfer Object).

В нашем случае **CustomUserDetailsService** также является службой, но реализует интерфейс **UserDetailsService** из Spring Security. Этот интерфейс определяет метод **loadUserByUsername**, который загружает пользователя по его имени пользователя (в вашем случае - по электронной почте) и возвращает объект **UserDetails**, который представляет информацию о пользователе, необходимую для аутентификации и авторизации в Spring Security.

|  |
| --- |
| **2.4. Безопасность приложения** |

Для обеспечения безопасности приложения был использован Spring Security.

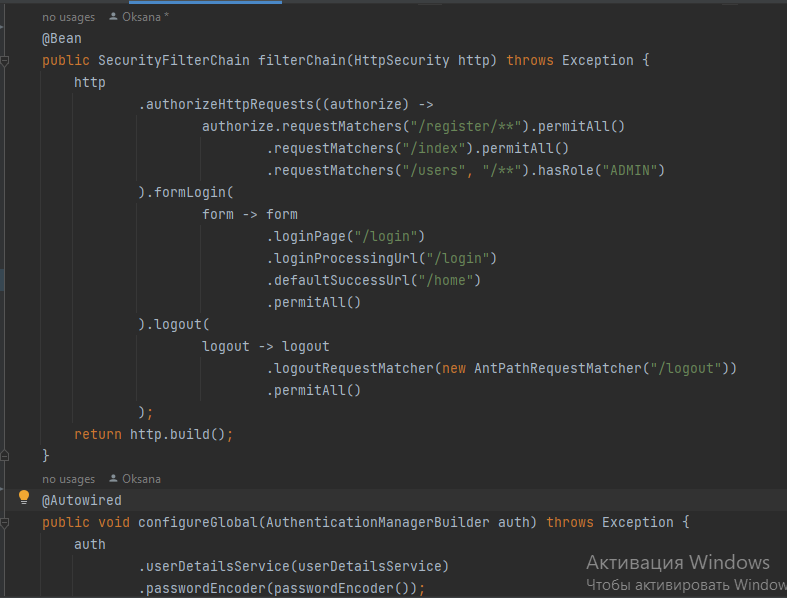
Spring Security - это мощный и гибкий фреймворк аутентификации и авторизации для приложений на платформе Java. Он обеспечивает обширный набор функций для защиты ваших приложений, включая аутентификацию пользователей, управление их правами доступа, защиту от атак, таких как CSRF и сессионные атаки, а также интеграцию с другими инфраструктурными службами, такими как LDAP, OAuth и другими.

Spring Security позволяет легко настраивать правила безопасности с использованием аннотаций или конфигурационных классов Java. Он также интегрируется с другими модулями Spring, такими как Spring MVC, что облегчает создание безопасных и надежных приложений на платформе Java.   
Он предоставляет механизмы аутентификации и авторизации, защиту от атак и другие функции безопасности. Вот основные концепции и функции Spring Security:

1. **Аутентификация**: Это процесс проверки подлинности пользователя. Spring Security поддерживает различные механизмы аутентификации, такие как форма, HTTP Basic, OAuth и другие.
2. **Авторизация**: Это процесс определения разрешений пользователя на доступ к определенным ресурсам или выполнение определенных действий. Spring Security использует роли и разрешения для управления доступом.
3. **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.
4. **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.
5. **Управление сессиями**: Spring Security предоставляет возможности управления сессиями, включая защиту от смены сессии, контроль одновременных сессий и настройку таймаута сессии.
6. **Шифрование паролей**: Spring Security советует хранить пароли в зашифрованном виде, предоставляя возможность использовать алгоритмы шифрования, такие как BCrypt, SCrypt и PBKDF2.
7. **Интеграция с Spring Framework**: Spring Security интегрируется плавно с Spring Framework, что упрощает обеспечение безопасности в приложениях на основе Spring.
8. **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.

Рассмотрим класс SecurityConfig.



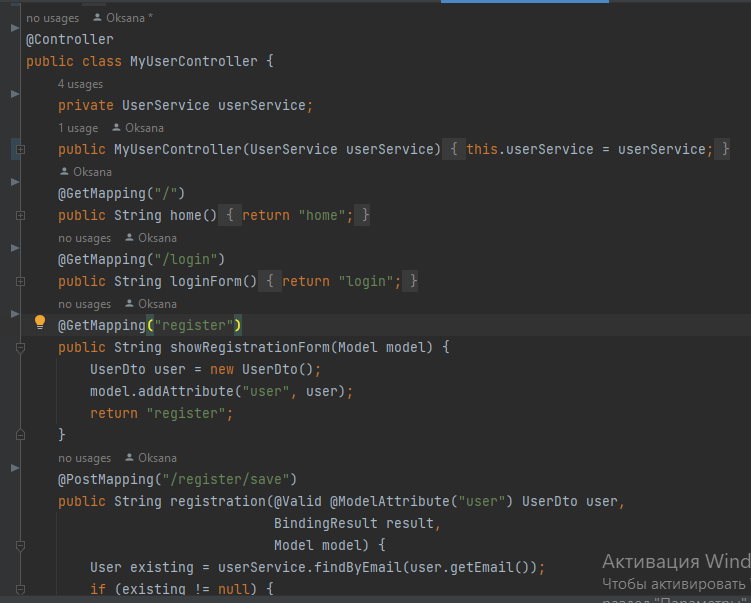
1. **@Configuration** и **@EnableWebSecurity**: Эти аннотации указывают, что этот класс является конфигурационным классом Spring и активирует поддержку веб-безопасности.
2. **SecurityFilterChain**: Этот метод создает фильтр цепочки безопасности, который определяет правила доступа к различным URL-адресам. В данном случае, он позволяет доступ к **/register/\*\*** и **/index** всем пользователям, а к **/users** и всем остальным URL-ам только пользователям с ролью "ADMIN".
3. **formLogin**: Этот метод настраивает аутентификацию через форму. Он указывает, что страница логина находится по адресу **/login**, обработка логина выполняется по адресу **/login**, а после успешной аутентификации пользователь перенаправляется на страницу **/home**.
4. **logout**: Этот метод настраивает логаут, позволяя пользователям разлогиниваться по адресу **/logout**.
5. **PasswordEncoder**: Этот бин создает экземпляр BCryptPasswordEncoder, который используется для хэширования паролей пользователей.
6. **configureGlobal**: Этот метод конфигурирует аутентификацию, указывая, что UserDetailsService используется для загрузки пользовательских данных и BCryptPasswordEncoder для проверки паролей.

Этот код предоставляет базовую конфигурацию безопасности для вашего приложения Spring,

Также в ходе реализации безопасности был создан метод MyUserController.

Этот класс представляет собой контроллер Spring MVC для обработки запросов, связанных с пользователями.

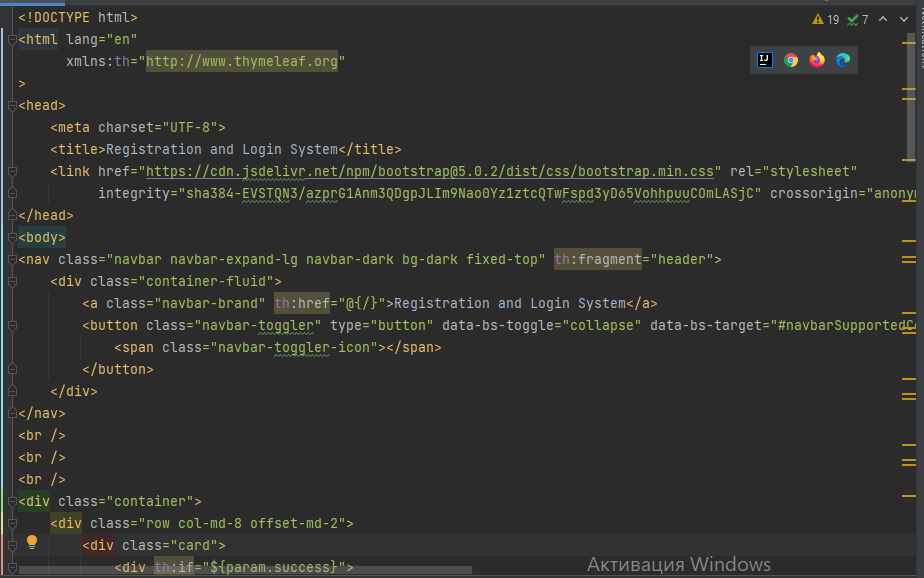
Этот контроллер обеспечивает основную функциональность для регистрации пользователей, отображения списка зарегистрированных пользователей и управления представлением для логина. Он использует объекты UserDto для передачи данных о пользователях между контроллером и представлением.

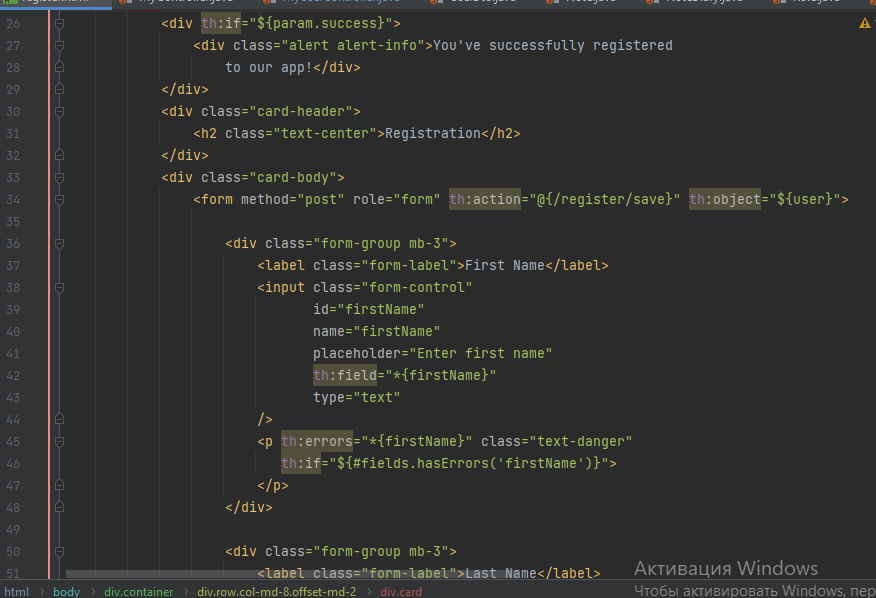


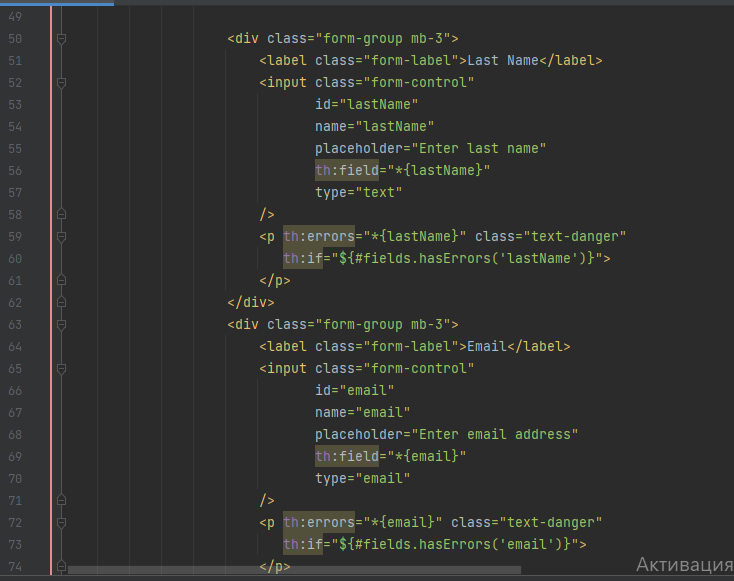


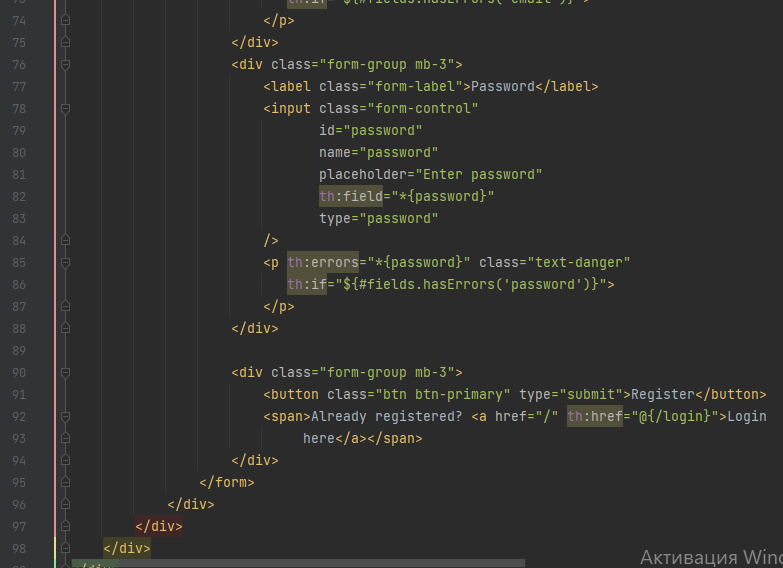
1. **@Controller**: Аннотация, которая указывает, что этот класс является контроллером Spring MVC.
2. **@Autowired UserService userService**: Инъекция зависимости UserService для доступа к функциональности управления пользователями.
3. **@GetMapping** и **@PostMapping**: Аннотации, определяющие, что методы обрабатывают GET и POST запросы соответственно.
4. **home()**: Метод, обрабатывающий запросы к корневому URL. Возвращает имя представления "home".
5. **loginForm()**: Метод, обрабатывающий GET запросы к URL "/login" и возвращающий имя представления "login".
6. **showRegistrationForm(Model model)**: Метод, обрабатывающий GET запросы к URL "/register" и возвращающий имя представления "register", при этом добавляется атрибут "user" в модель для использования в представлении.
7. **registration(@Valid @ModelAttribute("user") UserDto user, BindingResult result, Model model)**: Метод, обрабатывающий POST запросы к URL "/register/save" для регистрации новых пользователей. Аннотация @Valid используется для валидации объекта UserDto. Если есть ошибки валидации, метод возвращает имя представления "register" с атрибутами ошибок. В противном случае пользователь сохраняется через UserService, и пользователь перенаправляется на страницу "/register?success".
8. **listRegisteredUsers(Model model)**: Метод, обрабатывающий GET запросы к URL "/users" для отображения списка зарегистрированных пользователей. Получает список пользователей через UserService и добавляет его в модель для использования в представлении "users".

Далее рассмотрим HTML код (файл register) функции регистрации при открытии приложения.

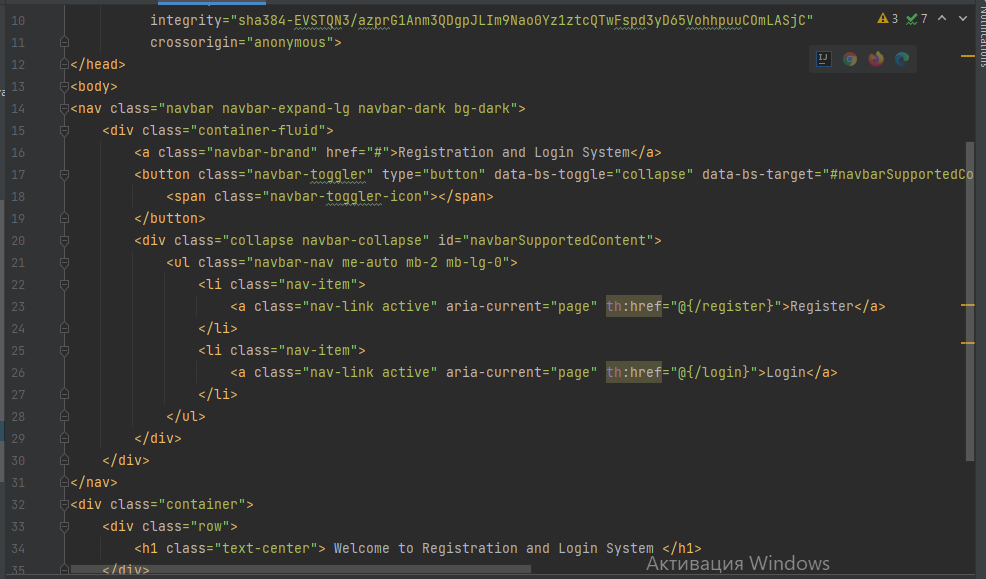






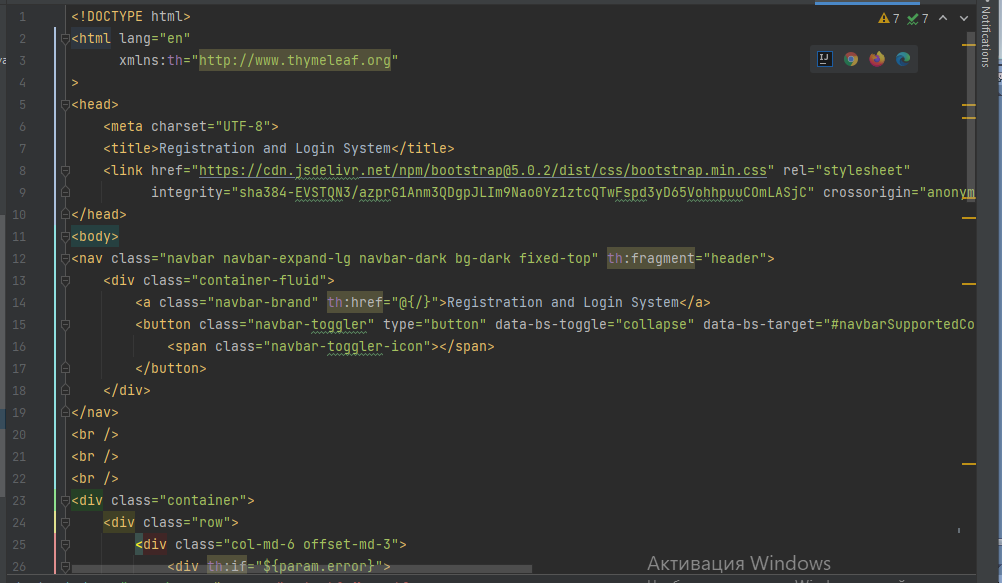


Следующий шаблон страницы веб-приложения users.html, реализующий систему регистрации и входа пользователей.



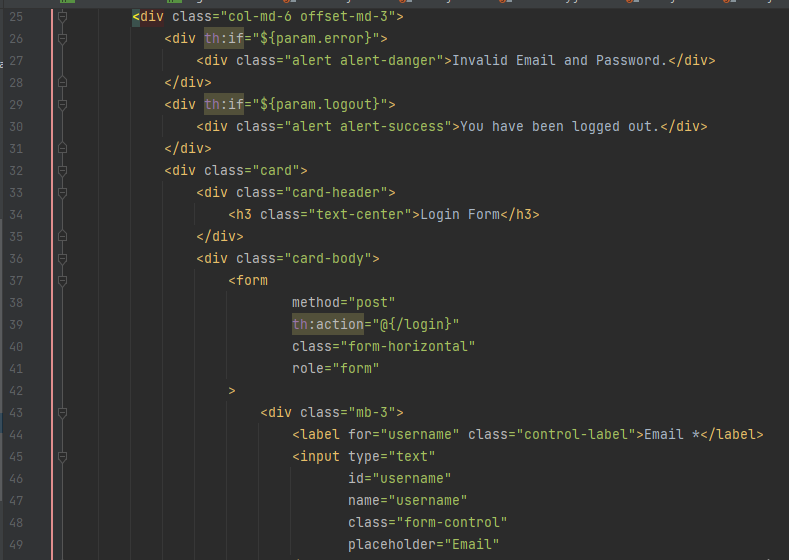
Разберем основные элементы:

1. **<title>Registration and Login System</title>**: Заголовок страницы.
2. Ссылка на стили Bootstrap для оформления страницы.
3. **<body>**: Основное содержимое страницы.
4. **<nav>**: Навигационное меню.
5. **<div class="container">**: Контейнер для размещения содержимого страницы.
6. **<h1> Registered Users </h1>**: Заголовок раздела.
7. **<table>**: Таблица для отображения зарегистрированных пользователей.
8. Использование Thymeleaf для итерации по списку пользователей и вывода их данных в таблицу.
9. Форма для перехода на главную страницу.
10. Следующий HTML-код представляет собой простую форму системы регистрации и входа на сайт с использованием Thymeleaf и Bootstrap.



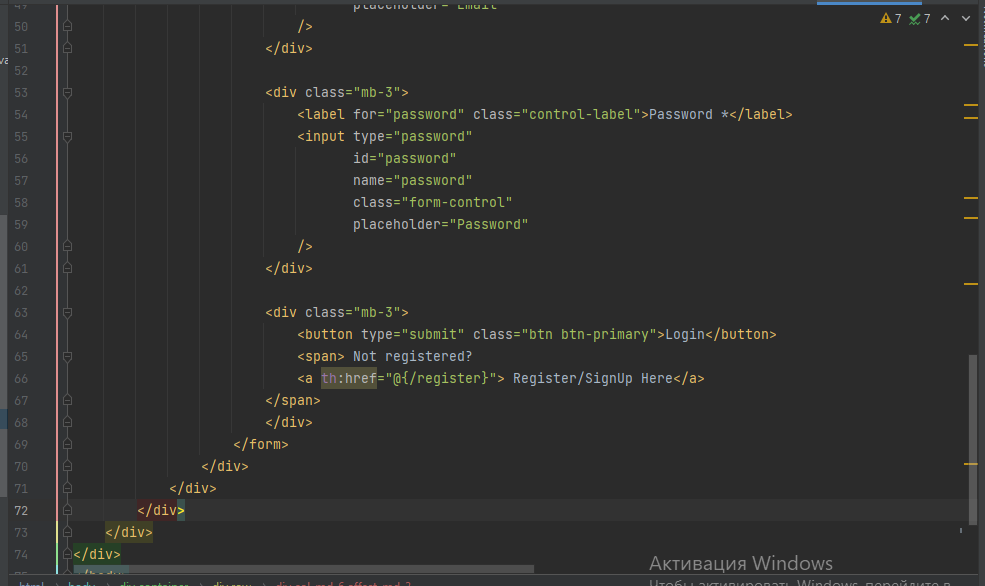
Вот краткое описание основных компонентов:

1. **link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">**: Подключает таблицу стилей Bootstrap из CDN.
2. **<body>**: Содержит основное содержимое веб-страницы.

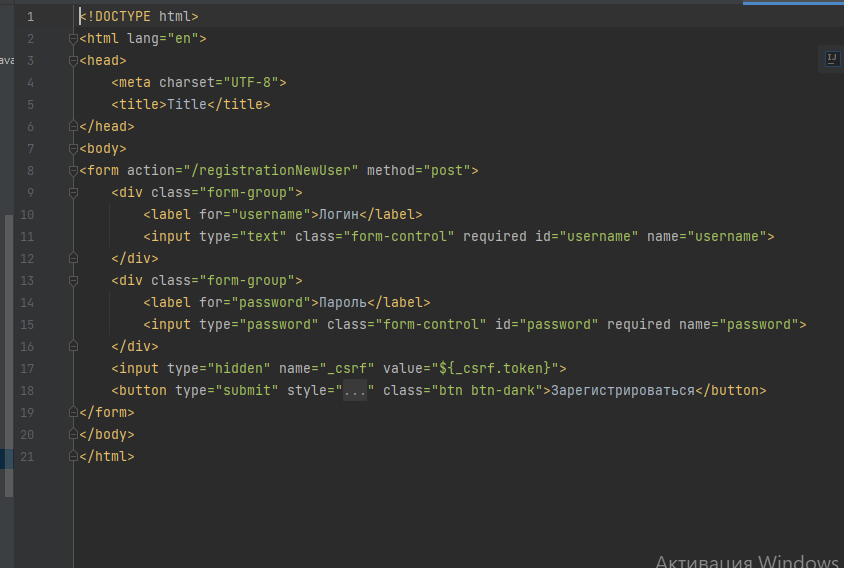


1. **<div th:if="${param.error}">** и **<div th:if="${param.logout}">**: Отображают сообщения об ошибках в случае неверного ввода или успешного выхода из системы.
2. Форма входа: Позволяет пользователю ввести имя пользователя (email) и пароль для входа, а также содержит ссылку на страницу регистрации.

Этот код создает простой, но функциональный интерфейс для системы регистрации и входа на сайт.



И рассмотрим registrationNewUser.html.



1. **<input type="text" class="form-control" required id="username" name="username">**: Создает поле ввода для логина с указанием класса стиля и обязательностью заполнения.
2. **<label for="password">Пароль</label>** и **<input type="password" class="form-control" id="password" required name="password">**: Аналогично создают поле ввода для пароля.
3. **<input type="hidden" name="\_csrf" value="${\_csrf.token}">**: Создает скрытое поле для передачи токена CSRF (Cross-Site Request Forgery) для защиты от подделки межсайтовых запросов.
4. **<button type="submit" style="width: 100%" class="btn btn-dark">Зарегистрироваться</button>**: Создает кнопку для отправки формы с текстом "Зарегистрироваться" и стилями Bootstrap.

Этот код создает простую форму для регистрации нового пользователя на веб-сайте, с обязательным вводом логина и пароля, а также защитой от CSRF-атак.

Начало формы

**2.5. Разворачивание Docker-контейнера**

Разворачивание Docker-контейнеров представляет собой процесс запуска и управления контейнерами на целевом хосте. Для этого используется Docker-демон, который управляет жизненным циклом контейнеров, такими как создание, запуск, остановка и удаление.

Основные шаги разворачивания Docker-контейнеров:

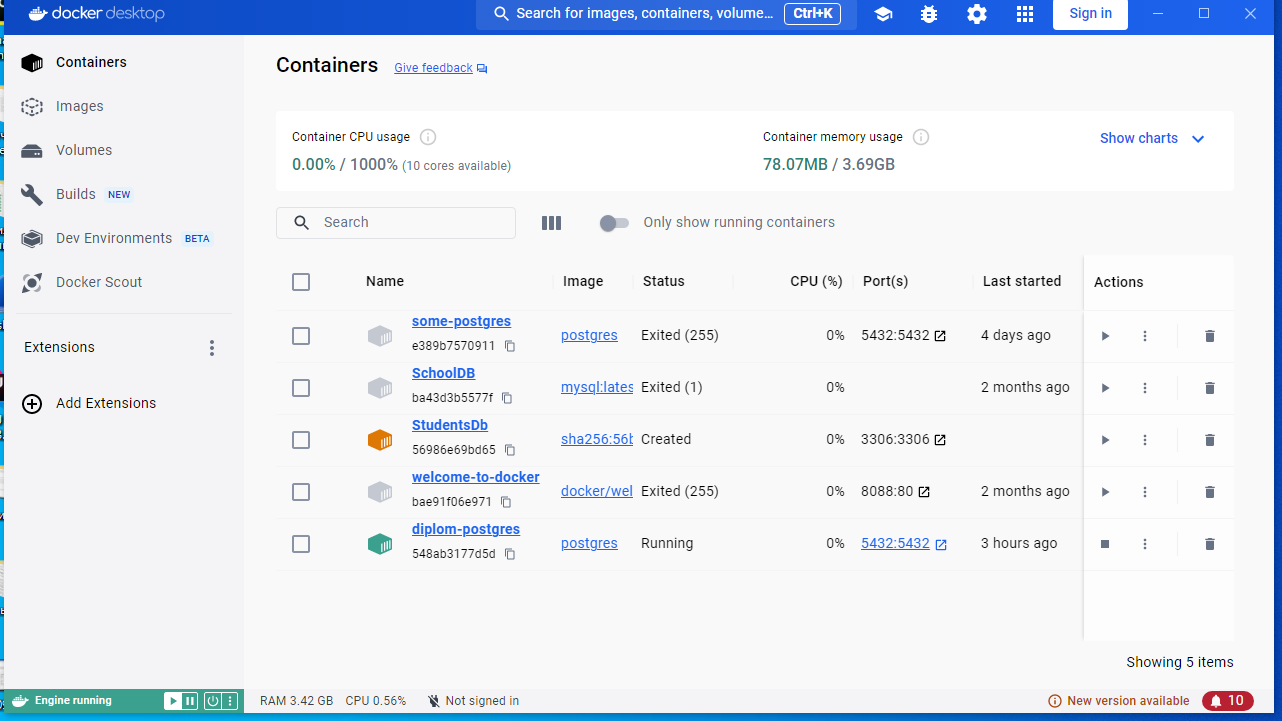
1. **Установка Docker**: Сначала необходимо установить Docker на целевом хосте. Для этого можно воспользоваться официальным сайтом Docker, где доступны инструкции по установке для различных операционных систем.
2. **Создание Docker-образа**: Docker-образ представляет собой шаблон, на основе которого создается контейнер. Образ содержит все необходимые зависимости и конфигурации приложения. Образ можно создать с помощью Dockerfile, который описывает шаги по сборке образа.
3. **Создание контейнера**: После создания образа необходимо создать контейнер на его основе. Это делается с помощью команды docker run. Например, для запуска контейнера на основе созданного образа:

**docker run --name diplom-postgres -e POSTGRES\_PASSWORD=admin -e POSTGRES\_DB=diplom\_db -d -p 5432:5432 postgres**

Команда **docker run** используется для создания и запуска контейнера на основе Docker-образа. Давайте разберем каждый параметр данной команды:

* **--name diplom-postgres**: Устанавливает имя контейнера как **diplom-postgres**. Это удобно для идентификации контейнера при его управлении.
* **-e POSTGRES\_PASSWORD=admin**: Устанавливает переменную окружения **POSTGRES\_PASSWORD** со значением **admin** для задания пароля для пользователя **postgres** в базе данных PostgreSQL.
* **-e POSTGRES\_DB=diplom\_db**: Устанавливает переменную окружения **POSTGRES\_DB** со значением **diplom\_db** для создания базы данных с именем **diplom\_db** в PostgreSQL.
* **-d**: Запускает контейнер в фоновом режиме (detached mode).
* **-p 5432:5432**: Пробрасывает порт 5432 из контейнера на хостовую машину. Первое число (5432) - порт на хостовой машине, второе число (5432) - порт в контейнере, на котором работает PostgreSQL.
* **postgres**: Имя Docker-образа, на основе которого будет создан контейнер. В данном случае, это официальный образ PostgreSQL из Docker Hub.

После выполнения этой команды будет создан и запущен контейнер с именем **diplom-postgres**, в котором будет работать PostgreSQL с базой данных **diplom\_db** и паролем **admin**, пробрасывая порт 5432 для доступа к базе данных извне контейнера.



**Заключение**

В рамках данного дипломного проекта было разработано веб-приложение «Дневник эмоций», предназначенное для записи и отслеживания эмоционального состояния пользователей. Проект был реализован с использованием современных технологий веб-разработки, таких как Java Spring Framework, Thymeleaf, Bootstrap и PostgreSQL.

В процессе разработки были достигнуты следующие цели:

* Создание удобного и интуитивно понятного интерфейса для пользователей приложения.
* Реализация функционала добавления, редактирования и удаления записей о эмоциих и эмоциях.
* 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, такого как статистика эмоционального состояния пользователя, возможность подключения календаря для отображения эмоциональных изменений во времени, а также мобильное приложение для удобного доступа к дневнику эмоций из любой точки.

В целом, дипломный проект "Дневник эмоций" 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 литературы**

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