Geekbrains

**Разработка веб-приложения для комплексной оценки эмоционального состояния «Дневник эмоций».**

Программа: Java-разработчик. Специалист

Специализация: Fullstack разработчик

Дукачова Дарья Валерьевна

Ростов-на-Дону

2025 г.

## Содержание

[Содержание 2](#_Toc187183390)

[Введение 3](#_Toc187183391)

[Глава 1. Основы разработки веб-приложений 6](#_Toc187183392)

[1.1 Что такое веб-приложение, его особенности 6](#_Toc187183393)

[1.2 Кто такой веб-разработчик и что входит в его специальность 9](#_Toc187183394)

[1.3 Этапы создания веб-сайта 11](#_Toc187183395)

[Глава 2. Разработка веб-приложения на Java Spring 15](#_Toc187183396)

[2.1 Что такое Java Spring? 15](#_Toc187183397)

[2.2 Этапы разработки на Java Spring 18](#_Toc187183398)

[Глава 3. Разработка веб-приложения. 24](#_Toc187183399)

[3.1 Выбор стека технологий 24](#_Toc187183400)

[3.2 Проектирование общей архитектуры приложения 26](#_Toc187183401)

[Разработка бэкенда приложения 28](#_Toc187183402)

[3.3 Безопасность приложения 37](#_Toc187183403)

[3.5 Проектирование WEB – интерфейса приложения 47](#_Toc187183404)

[Заключение 52](#_Toc187183405)

[Список источников 54](#_Toc187183406)

## Введение

**Актуальность**

Дневник эмоций — это терапевтический инструмент, который помогает человеку следить за мыслями и чувствами, лучше понимать себя и учиться контролировать эмоциональные всплески. Дневниковые техники изначально зародились для терапии тревоги и депрессии. Сегодня этот метод считается стандартом для работы по отслеживанию эмоций в разных состояниях.

Этот навык важен, чтобы наладить контакт с собой. Например, чтобы отследить, какие события и люди дают вам энергию, а какие отнимают. А ещё ведение дневника эмоций помогает в осознании и принятии своих потребностей.

Например, понимание своих чувств способно облегчить жизнь:

* тем, кто чувствует себя одиноким и несчастным, но не понимает причины;
* тем, кто беспочвенно грубит близким, а потом горько жалеет об этом;
* тем, кто регулярно испытывает тревогу в бытовых ситуациях и не знает, как с ней справиться.

А ещё отслеживание и анализ своих состояний поможет людям, которые привыкли сдерживать эмоции и не разрешать себе злиться, завидовать или грустить. Такая отстранённость от себя чревата: трудно чувствовать радость или удовольствие, если другие, негативные эмоции под запретом.

По данным ВОЗ[[1]](#footnote-1), от депрессивного расстройства страдают 5% взрослых во всем мире. При этом в некоторых странах депрессия гораздо более распространена, чем в других. Что частично объясняется особенностями сбора статистических данных и тем, насколько среди жителей распространена практика обращения за психиатрической помощью.

Также ученые наблюдают тенденцию: ментальное здоровье людей становится все хуже. Так, исследования показали: распространенность депрессии в США выросла с 7,3% в 2015 году до 9,2% в 2019 году. Аналогичную ситуацию наблюдают специалисты по всему миру[[2]](#footnote-2).

Особенно сильно тенденция к ухудшению психологического состояния видна среди молодых людей. Исследование американского Центра по контролю и профилактике заболеваний показало: количество грустных людей, чувствующих подавленность, увеличилось с 26% в 2009 году до 44% в 2021 году. Стало больше и тех, кто страдает от расстройств пищевого поведения и проявляет суицидальное поведение[[3]](#footnote-3).

Статистика, скорее всего, не до конца отражает реальность и настоящие цифры выше. Тема ментального здоровья все еще стигматизирована. Одни боятся обращаться к психиатрам, думая, что этим поставят под угрозу свою карьеру. Другие убеждены, что в помощи специалиста нуждаются только странные и маргинальные люди, а они таковыми не являются.

Таким образом, разработка приложения "Дневник эмоций" является актуальной в современной социальной и технологической среде. Оно не только отвечает глобальным вызовам, связанным с психоэмоциональной перегрузкой, но и способствует повышению уровня осознанности, саморефлексии и гармонизации внутреннего состояния пользователей.

В этом заключается актуальность работы.

**Какую проблему решает:** предоставление ресурса помощи с ежедневным стрессом, тревогой, пониманием причин своего поведения, улучшения эмоционального состояния.

**Цель** – изучить возможности Spring и разработать веб-приложение «Дневник эмоций».

Для достижения поставленной цели необходимо решение следующих **задач:**

* Изучить литературу, касающуюся темы исследования.
* Изучить фреймворк Spring.
* Разработать бэкенд приложения.
* Разработать фронтенд приложения.

**Инструменты:** IntelliJ IDEA, Git, Microsoft 365 Word, Postman, GitHub.

**Структура работы:** работа состоит из введения, трёх глав, заключения, списка источников и приложений.

**Состав команды:** Дукачова Дарья Валерьевна (разработчик).

## Глава 1. Основы разработки веб-приложений

### 1.1 Что такое веб-приложение, его особенности

Веб-приложение представляет собой программное обеспечение, которое функционирует через веб-браузер и предоставляет пользователям доступ к своим функциям и данным посредством сети Интернет. В отличие от традиционных десктопных программ, которые требуют установки на компьютер, веб-приложения работают в режиме онлайн и не зависят от конкретного устройства или операционной системы. Главная особенность веб-приложений заключается в их доступности – пользователю достаточно иметь интернет-соединение и браузер для использования всех возможностей такого приложения без необходимости скачивания дополнительных компонентов.

Одной из ключевых характеристик веб-приложений является их универсальность и кроссплатформенность. Они разрабатываются с использованием веб-технологий, таких как HTML, CSS и JavaScript, а также серверных языков программирования, что позволяет им работать одинаково на любых устройствах, будь то компьютер, планшет или смартфон. Это делает их удобными как для пользователей, так и для разработчиков, так как нет необходимости адаптировать их под разные операционные системы или устройства.

Еще одной важной особенностью веб-приложений является их централизованность. Данные и логика работы таких приложений хранятся на сервере, а не на устройстве пользователя. Это предоставляет пользователям доступ к своим профилям, данным или настройкам из любой точки мира, где есть подключение к Интернету. Например, такие популярные сервисы, как Google Docs или Gmail, являются примерами успешных веб-приложений, которые предоставляют широкий функционал через веб-браузер, включая хранение данных в облаке и возможность совместной работы в реальном времени.

Веб-приложения также просты в обновлении и поддержке. Поскольку все изменения происходят на сервере, пользователи получают самую актуальную версию программы при каждом входе в систему, не утруждаясь в скачивании обновлений. Это выгодно отличает их от традиционных программ, где обновления требуют участия пользователя и могут вызвать технические сбои.

Особенности, которые делают веб-приложения особенно привлекательными для бизнеса, включают их масштабируемость, улучшенный пользовательский опыт и сниженные затраты на разработку. Возможность масштабировать функции и пользовательскую базу практически неограниченна, так как новые пользователи могут быть добавлены через серверные технологии без перенастройки клиентов. Для больших компаний это открывает доступ к массовой, а иногда и глобальной аудитории, без необходимости значительных вложений в инфраструктуру.

Безусловно, веб-приложения имеют и свои ограничения. Их функциональность напрямую зависит от качества интернет-соединения и мощности сервера. При медленном подключении или перегрузке серверов производительность приложения может снизиться, что негативно скажется на работе пользователей. Вдобавок вопросы безопасности веб-приложений требуют особого внимания, так как они хранят данные на внешних серверах, что делает их уязвимыми для кибератак и попыток несанкционированного доступа. Однако современные технологии, такие как шифрование данных и двухфакторная аутентификация, позволяют минимизировать эти риски.

Веб-приложение состоит из двух основных частей: фронтенда (клиентской части) и бэкенда (серверной части), каждая из которых требует использования различных языков программирования и фреймворков. Фронтенд представляет собой ту часть приложения, с которой взаимодействует пользователь через браузер. Для его создания разработчики используют такие базовые языки, как HTML для построения структуры страниц, CSS для оформления внешнего вида и JavaScript для добавления интерактивности и динамических элементов. Современные динамичные интерфейсы разрабатываются с применением фреймворков, таких как React.js, Vue.js и Angular, каждый из которых предоставляет инструменты для создания удобных и гибких пользовательских интерфейсов. Для упрощения стилизации широко используются CSS-фреймворки вроде Bootstrap или Tailwind CSS, которые помогают быстро создавать адаптивные и эстетичные приложения.

Бэкенд, в свою очередь, отвечает за серверную часть приложения, где обрабатывается логика, хранятся и обрабатываются данные, а также выполняются запросы от клиента. На этой стороне используются языки, такие как JavaScript (с помощью среды Node.js), Python, PHP, Java, Ruby, C# и Go. Любой из этих языков может быть использован в зависимости от потребностей проекта. Для упрощения работы с сервером разработчики применяют фреймворки. Express.js используется совместно с Node.js для быстрой разработки серверной части. Django и Flask являются популярными выбором для Python благодаря своим широким возможностям и гибкости. Ruby on Rails хорошо подходит для быстрого создания масштабируемых приложений, а Java в сочетании с Spring Boot применяется для крупных и корпоративных систем. Для C# разработчики используют ASP.NET Core, который обеспечивает высокую производительность.

Одной из важных составляющих веб-приложений являются базы данных. Они позволяют хранить, организовывать и управлять данными приложения. На практике используются как реляционные базы данных, например, MySQL, PostgreSQL и SQLite, так и нереляционные базы данных, такие как MongoDB и Redis. Выбор типа базы данных зависит от характера данных и особенностей проекта. Взаимодействие между фронтендом и бэкендом осуществляется с помощью API, которые строятся на REST или GraphQL, что даёт возможность клиентской части получать и отправлять данные к серверу. Для внедрения функций реального времени, таких как чаты или уведомления, используется технология WebSocket.

Дополнительно современные веб-приложения нередко зависят от таких технологий, как Docker, который помогает упаковывать приложение в контейнер для упрощения развёртывания, а также инструментов автоматизации деплоя, таких как Jenkins или GitHub Actions. Таким образом, создание веб-приложений предполагает использование широкого спектра языков программирования и фреймворков, выбор которых зависит от целей, задач и специфики проекта.

Таким образом, веб-приложения являются важным звеном в современном цифровом мире. Они объединяют удобство, универсальность и постоянный доступ, что делает их востребованными как для пользователей, так и для разработчиков. Быстрые темпы развития технологий продолжают расширять возможности веб-приложений, интегрируя в них искусственный интеллект, машинное обучение и облачные вычисления, что позволяет создавать более адаптивные и полезные инструменты для решения самых разных задач.

### 1.2 Кто такой веб-разработчик и что входит в его специальность

Веб-разработчик — это специалист, который занимается созданием, модификацией и поддержкой веб-сайтов, веб-приложений и других цифровых продуктов, работающих через интернет. Его основная задача заключается в разработке функциональных и визуально привлекательных веб-ресурсов, обеспечивающих удобство использования и выполнение задач, поставленных клиентом или бизнесом. Веб-разработка охватывает широкий спектр деятельности, включая программирование, работу с базами данных, интеграцию интерфейсов и обеспечение безопасности веб-ресурсов.

Ключевая особенность профессии веб-разработчика заключается в её междисциплинарном характере. Специалисту необходимо не только обладать техническими навыками программирования, но и понимать основы дизайна, юзабилити (удобства использования) и работы веб-технологий. Веб-разработка подразделяется на несколько направлений, каждое из которых имеет свои особенности.

Первое направление — фронтенд-разработка. Фронтенд-разработчик отвечает за внешний вид и поведение сайта с точки зрения конечного пользователя. Основными инструментами фронтенд-разработчика являются HTML (для структуры страницы), CSS (для оформления и визуального стиля) и JavaScript (для добавления интерактивных элементов). Например, именно фронтенд-разработчик делает так, чтобы кнопки меняли цвет при наведении, меню раскладывались по клику, а веб-страница адаптировалась под размер экрана. Также специалистами в этой области активно используются такие современные технологии и фреймворки, как React, Angular или Vue.js, которые помогают ускорить процесс разработки и сделать веб-приложения более динамичными.

Второе направление — бэкенд-разработка. Бэкенд-разработчик отвечает за серверную часть сайта или веб-приложения. Это “скрытая” часть веб-ресурса, где происходят основные вычисления, обработка данных и их хранение. Бэкенд-разработчики используют такие языки, как Python, PHP, Ruby, Java, или Node.js, а также работают с базами данных (например, MySQL, PostgreSQL или MongoDB). Их задача — настроить логику работы веб-ресурса: будь то регистрация пользователя, обработка заказов в интернет-магазине или взаимодействие с внешними API. Важно, чтобы бэкенд был надёжным, быстрым и безопасным.

Еще одно популярное направление — фулстек-разработка. Фулстек-разработчик объединяет сразу две сферы: фронтенд и бэкенд. Такие специалисты умеют создавать как пользовательский интерфейс, так и серверную логику, что делает их универсальными и высоко востребованными. Однако путь к овладению навыками фулстек-разработки требует больше времени и усилий, так как необходимо изучить широкий спектр технологий.

Кроме технической работы, в обязанности веб-разработчика входит понимание задач клиента, анализ требований проекта и обеспечение его успешной реализации. На начальных этапах веб-разработчик часто взаимодействует с дизайнерами (UI/UX-дизайнерами), которые создают макеты сайта. Затем он приступает к их воплощению в реальную, работающую веб-страницу. На более поздних этапах он может заниматься тестированием своего кода, выявлением ошибок и их исправлением, а также внедрением новых функций.

Не менее важным аспектом работы веб-разработчика является постоянное изучение новых технологий. Веб-разработка — это быстро развивающаяся область. Инструменты, технологии и подходы, которые были актуальны несколько лет назад, могут устаревать, а на их место приходят новые концепции. Успешный веб-разработчик — это специалист, готовый к постоянному обучению и адаптации к изменениям.

Также в круг обязанностей веб-разработчика входит обеспечение безопасности веб-ресурсов. Это особенно актуально в условиях роста числа кибератак и утечек данных. Например, разработчик должен уметь работать с протоколами шифрования, защитой от SQL-инъекций, настройкой прав доступа и другими аспектами информационной безопасности.

Веб-разработчики востребованы в различных сферах, включая бизнес, медицину, образование, маркетинг, финансы и общественные услуги. Интернет-сайты и приложения стали неотъемлемой частью почти любой компании, что делает данную профессию крайне популярной. Сегодня веб-разработчики могут работать в крупных IT-компаниях, заниматься аутсорсингом или быть фрилансерами, выполняя заказы для клиентов по всему миру.

### 1.3 Этапы создания веб-сайта

Создание веб-сайта — это сложный и многоступенчатый процесс, который включает в себя различные этапы, начиная от планирования и заканчивая публикацией ресурса в сети. Каждый этап требует взаимодействия разных специалистов, таких как дизайнеры, разработчики, тестировщики и контент-менеджеры. Основной целью является создание функционального, удобного и привлекательного сайта, который отвечает требованиям пользователей и клиента.

Первым этапом является определение целей и задач проекта. На этом этапе проводится анализ требований клиента и целевой аудитории. Здесь важно понять, зачем создаётся сайт, какими функциями он должен обладать и каких пользователей он будет привлекать. Отождествляется модель его будущего использования: будет ли это сайт-визитка, интернет-магазин, блог или большой портал. Параллельно разрабатывается план проекта, который включает описание структуры сайта, ключевых разделов, пользовательских сценариев и общих функций.

После определения целей наступает этап проектирования и разработки структуры. На этом этапе создаётся каркас сайта, который демонстрирует расположение элементов на страницах: шапки, основного контента, меню, виджетов и подвала. Часто используется прототипирование с помощью специальных инструментов, таких как Figma или Adobe XD, чтобы визуализировать структуру будущего сайта. Это позволяет заранее представить, как будет работать пользовательский интерфейс, и учесть возможные исправления ещё до написания кода.

Следующий шаг — создание дизайна. На основе утверждённой структуры разрабатывается визуальное оформление сайта, которое должно соответствовать целям проекта и быть интуитивно понятным для пользователя. Дизайнеры создают макеты страниц, выбирают цветовые схемы, шрифты, расположение фотографий и графических элементов. Важно, чтобы дизайн был не только оригинальным, но и адаптивным, то есть одинаково хорошо выглядел на экранах различных устройств — от компьютеров до смартфонов.

После утверждения дизайна начинается разработка сайта. Этот этап делится на два аспекта: фронтенд и бэкенд. Фронтенд-разработчик создаёт клиентскую часть сайта — ту, которую видит пользователь, основываясь на созданных дизайн-макетах. Он использует технологии HTML для структуры, CSS для оформления и JavaScript для добавления интерактивности. Параллельно бэкенд-разработчики работают над серверной частью, отвечающей за обработку данных, логику работы сайта и его взаимодействие с базами данных. На этом же этапе могут быть использованы фреймворки (например, React или Django) для ускорения и стандартизации процесса разработки.

Когда разработка завершена, проводится тестирование сайта. Тестировщики проверяют, как сайт функционирует в различных браузерах, на устройствах с разными размерами экранов и операционными системами. Проверяется работоспособность всех функций: форм обратной связи, поиска, оплаты и других интерактивных элементов. Особое внимание уделяется скорости загрузки страниц, ведь медленный сайт значительно ухудшает пользовательский опыт. Также тестирование направлено на поиск багов и недоработок, которые на этом этапе исправляются.

Далее сайт размещается в сети. Для этого он переносится на хостинг (площадку для хранения файлов сайта в Интернете) и настраивается доменное имя (читаемое имя сайта, которое вводится в строку браузера). После настройки серверной среды проводится финальная проверка на работоспособность. На этом этапе также проводится первичная индексация сайта для поисковых систем (например, Google и Яндекс), чтобы ресурс стал доступен пользователям.

После публикации начинается этап поддержки и развития сайта. В процессе эксплуатации могут выявляться незначительные недочёты, которые устраняются по мере их обнаружения. Также сайт требует регулярных обновлений: это может быть добавление нового контента, модификация функционала или улучшение дизайна. Для повышения трафика используется поисковая оптимизация (SEO), которая помогает сайту занимать высокие позиции в поисковых системах, а также стратегии интернет-маркетинга, направленные на привлечение новых пользователей.

Каждый этап создания веб-сайта играет ключевую роль в его успехе. Планирование обеспечивает чёткое понимание целей, дизайн привлекает пользователей, разработка воплощает идеи в жизнь, а тестирование и поддержка гарантируют стабильную и качественную работу ресурса. Только при чётком соблюдении всех этапов можно создать сайт, который будет одновременно функциональным, удобным и эффективным.

## Глава 2. Разработка веб-приложения на Java Spring

### 2.1 Что такое Java Spring?

Java Spring — это один из самых популярных и широко используемых фреймворков для разработки приложений на языке Java. Его успех объясняется универсальностью, гибкостью и мощным набором инструментов, которые позволяют создавать как простые приложения, так и сложные корпоративные системы. Spring предоставляет разработчикам удобные решения для работы с различными аспектами программного обеспечения, включая управление зависимостями, безопасность, взаимодействие с базами данных, создание веб-интерфейсов и многое другое. Благодаря своей привлекательности и богатым возможностям Spring заслуженно стал выбором многих компаний и разработчиков по всему миру.

История Java Spring начинается в начале 2000-х годов, когда возникла потребность в облегчении традиционных процессов разработки приложений на Java. Созданный как альтернатива громоздкой спецификации Java Enterprise Edition (JEE), Spring изначально был предназначен для упрощения работы над объектно-ориентированными приложениями и управления их зависимостями. Основная идея Spring заключалась в использовании принципа IoC (Inversion of Control, инверсия управления), который изменил подход к взаимодействию между компонентами в приложениях. Вместо того чтобы объекты создавали свои зависимости самостоятельно, контроль передавался контейнеру Spring, который связывал и управлял всеми компонентами. Это революционное изменение сделало код более структурированным, тестируемым и удобным для сопровождения.

Ключевым преимуществом Java Spring является его модульная архитектура. Фреймворк состоит из множества небольших и взаимосвязанных модулей. Разработчик может использовать только те модули, которые нужны для решения конкретной задачи, что делает фреймворк лёгким, а приложение — производительным. Среди таких модулей наиболее популярны Spring Core, Spring Data, Spring Security и Spring Web. Благодаря этому подходу Spring подходит для самых разных случаев использования: от создания REST API до реализации сложных микросервисов.

Одной из самых удобных частей Spring является Spring Boot — специальное расширение, направленное на упрощение разработки приложений. Он практически исключает необходимость в сложной конфигурации, автоматически предоставляет настройки и интеграции для большинства распространённых случаев. Это делает Spring Boot идеальным выбором для быстрого прототипирования и реализации производственных систем. Достаточно добавить подходящие зависимости, и разработка может начаться без длительной подготовки.

Ещё одной важной возможностью Spring является его поддержка работы с веб-технологиями. Модуль Spring MVC предоставляет полноценную реализацию архитектуры Model-View-Controller, которая позволяет эффективно разделить коды для обработки данных, бизнес-логики и представления. Это делает приложения, созданные на Spring, более структурированными и легкими в развитии. С другой стороны, популяризация REST API значительно увеличила спрос на использование Spring для создания веб-сервисов. Spring позволяет легко создавать и поддерживать RESTful-приложения с использованием таких технологий, как JSON, а также интегрироваться с технологиями, необходимыми для микросервисов, такими как Spring Cloud.

Значительное внимание в Spring уделяется безопасности. Spring Security предоставляет мощный инструмент для защиты приложений, реализуя такие функции, как аутентификация, авторизация и защита от атак. Особенно он полезен в корпоративной среде, где приложения требуют защиты данных пользователей. С большим количеством встроенных функций безопасность можно настроить за короткое время, а гибкость модуля позволяет разрабатывать как простые формы входа, так и сложные схемы на основе ролей или OAuth2.

Также важно отметить поддержку баз данных, которую обеспечивает модуль Spring Data. Он предлагает удобные средства для работы с реляционными (например, MySQL, PostgreSQL) и нереляционными базами данных (MongoDB, Cassandra и т.д.). Благодаря этому модулю разработчикам не нужно писать сложные SQL-запросы вручную, а можно воспользоваться концепцией ORM (Object-Relational Mapping) для представления базы данных в виде Java-объектов. Это ускоряет процесс разработки и снижает вероятность ошибок при взаимодействии с базами данных.

Spring также активно используется в разработке микросервисов — масштабируемых и изолированных приложений, которые взаимодействуют друг с другом. С помощью Spring Cloud можно быстро настроить коммуникацию между микросервисами, управление конфигурациями, обработку отказов и балансировку нагрузки. Это особенно важно для крупномасштабных систем, которые требуют высокой производительности и устойчивости.

Одним из дополнительных преимуществ Spring является его активное сообщество и развитая экосистема. Тысячи разработчиков ежедневно совершенствуют фреймворк, делятся своим опытом и предлагают новые идеи. Существуют обширные официальные документы и примеры, которые помогают новичкам быстро освоить Spring, а профессионалам — решать более сложные задачи. Кроме того, благодаря популярности Spring множество вакансий и проектов на рынке связаны именно с этим фреймворком, что делает его изучение перспективным для карьеры.

Таким образом, Java Spring представляет собой мощный, гибкий и удобный инструмент для разработки. Благодаря своей модульной архитектуре, поддержке современных технологий, безопасности и активной экосистеме он используется для решения широкого спектра задач: от создания простых веб-приложений до разработки сложных корпоративных систем. Способность адаптироваться к новым требованиям и вызовам делает Spring не только настоящим лидером в мире Java, но и незаменимым помощником в создании современного программного обеспечения.

### 2.2 Этапы разработки на Java Spring

Разработка приложений с использованием Java Spring включает в себя последовательность этапов, которые позволяют создать масштабируемое, функциональное и безопасное программное обеспечение. Благодаря модульной структуре фреймворка Spring и удобным инструментам, таким как Spring Boot, процесс разработки значительно упрощается, становится логичным и структурированным. Рассмотрим основные этапы создания приложения на Java Spring и их значения для достижения конечной цели.

Любое программное приложение начинается с определения задач и понимания требований. На этом этапе разработчик или команда должны ответить на ключевые вопросы: кто целевая аудитория приложения, какие функции оно должно выполнять, с какими внешними сервисами или данными оно будет взаимодействовать. Составление спецификации задач помогает определить, какую архитектуру лучше выбрать — будет ли это монолитное приложение или микросервисная архитектура. Например, приложение для управления пользовательскими данными может использовать REST API для взаимодействия с клиентами.

Важным шагом на этой стадии является проектирование структуры приложения, где определяется логическая разбивка системы на модули, слои и компоненты. При использовании Spring чаще всего применяется архитектурный паттерн MVC (Model-View-Controller), который чётко разделяет данные, их обработку и представление.

После прояснения требований осуществляется создание проекта с использованием Spring Boot. Spring Boot является упрощённым решением для разработки приложений на Java Spring и позволяет избежать сложности ручной конфигурации.

Для быстрого старта можно воспользоваться утилитой Spring Initializr. Она позволяет выбрать необходимые зависимости (модули), такие как:

- Spring Web для разработки REST API или веб-интерфейсов.

- Spring Data JPA для работы с базами данных.

- Spring Security для реализации механизмов авторизации и аутентификации.

- Thymeleaf для серверного рендеринга страниц HTML.

- Другие вспомогательные компоненты.

После выбора параметров генерируется базовая структура проекта, которая будет использоваться для последующей разработки.

Следующий этап — это проектирование моделей данных. Здесь решается, какие сущности будут использоваться в приложении, их характеристики и взаимосвязи. Например, для веб-приложения, управляющего списком задач, сущностями могут быть `Task` (задача) и `User` (пользователь). Каждая сущность преобразуется в Java-класс, который аннотируется с использованием JPA (Java Persistence API) для связи с реляционными базами данных.

Пример модели:

import jakarta.persistence.\*;

@Entity

public class Task {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String description;

private boolean completed;

public Task() {}

// Геттеры и сеттеры

}

Эти модели становятся основой для взаимодействия приложения с базой данных.

На уровне доступа к данным используются репозитории. Spring Data JPA предлагает готовые интерфейсы, которые значительно упрощают работу с базой данных. Помимо базовых операций (сохранение, удаление, поиск по ID), репозитории позволяют добавлять пользовательские методы запросов.

Пример:

import org.springframework.data.jpa.repository.JpaRepository;

public interface TaskRepository extends JpaRepository<Task, Long> {

// Встроенные методы: save, findById, deleteById и др.

}

Репозитории инкапсулируют операции с базой данных, предоставляя удобный способ взаимодействия с хранилищем.

Сервисный слой отвечает за обработку бизнес-логики приложения. На этом этапе пишутся методы, которые используют репозитории для выполнения конкретных операций. Например, можно создать метод для получения списка задач у конкретного пользователя или для изменения статуса задачи.

Пример сервиса:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class TaskService {

@Autowired

private TaskRepository taskRepository;

public List<Task> getAllTasks() {

return taskRepository.findAll();

}

public Task saveTask(Task task) {

return taskRepository.save(task);

}

}

Сервисы изолируют бизнес-логику от контроллеров, что улучшает структуру приложения и делает код более поддерживаемым.

Контроллеры работают на уровне взаимодействия с клиентами. Они принимают HTTP-запросы, вызывают методы сервисного слоя и возвращают ответы. С помощью аннотации `@RestController` или `@Controller` Spring позволяет легко интегрировать серверную часть с клиентскими приложениями.

Пример контроллера:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/tasks")

public class TaskController {

@Autowired

private TaskService taskService;

@GetMapping

public List<Task> getTasks() {

return taskService.getAllTasks();

}

@PostMapping

public Task createTask(@RequestBody Task task) {

return taskService.saveTask(task);

}

}

Контроллеры позволяют получать данные из внешних запросов, обрабатывать их и возвращать клиенту результат.

На этом этапе подключается база данных к проекту. Для небольших приложений может использоваться H2 (встраиваемая база), а для крупных проектов обычно применяются MySQL, PostgreSQL и другие системы управления базами данных (СУБД). Настройки указываются в файле `application.properties` или `application.yml`.

Пример настроек для MySQL:

spring.datasource.url=jdbc:mysql://localhost:3306/tasksdb

spring.datasource.username=root

spring.datasource.password=pass1234

spring.jpa.hibernate.ddl-auto=update

После реализации всех ключевых компонентов проводится тестирование. Spring предоставляет мощные инструменты для разработки модульных и интеграционных тестов. Такие инструменты, как JUnit и Mockito, позволяют проверить функциональность каждого компонента приложения.

Пример простого теста:

import static org.assertj.core.api.Assertions.assertThat;

@Test

void testGetTasks() {

List<Task> tasks = taskService.getAllTasks();

assertThat(tasks).isNotEmpty();

Когда разработка завершена, приложение разворачивается в рабочей среде. Spring Boot позволяет легко упаковывать приложение в автономный JAR-файл с встроенным сервером (например, Tomcat), что делает развёртывание максимально удобным. Также часто используются контейнеры Docker для развёртывания в облачных средах.

Этапы разработки на Java Spring обеспечивают эффективное создание веб-приложений, начиная от предварительного планирования до их развёртывания. Структурированный подход позволяет сосредоточиться на каждой части системы, обеспечивая модульность и лёгкую поддерживаемость. Благодаря инструментам Spring фреймворк остаётся ведущим выбором для разработки современных приложений.

## Глава 3. Разработка веб-приложения.

### 3.1 Выбор стека технологий

Выбор стека технологий для веб-приложения на Java Spring — это важный этап разработки, который во многом определяет производительность, масштабируемость, удобство поддержки и функциональные возможности приложения. Spring Framework часто выступает ядром серверной части веб-приложения благодаря своей гибкости и мощной экосистеме. Однако для создания полноценного веб-приложения требуется больше, чем только фреймворк для серверной логики. Необходимо выбрать дополнительные технологии для работы с базой данных, пользовательским интерфейсом, безопасностью, хостингом и интеграцией с другими сервисами.

Для разработки серверной части приложения будем использовать Java, так как этот язык хорошо подходит для создания масштабируемых и надежных веб-приложений.

Центральным элементом технологии для серверной части является Spring Framework. Он предоставляет мощное ядро для реализации веб-приложений любой сложности, начиная от небольших сайтов до крупных корпоративных систем.

Spring Boot — используется для упрощения конфигурации и быстрого старта проекта. Он подходит как для разработки REST API, так и для создания монолитных приложений.

Spring MVC — реализует архитектурный паттерн Model-View-Controller и управляет маршрутизацией запросов, взаимодействием с данными и шаблонизацией.

Spring Data JPA — используется для взаимодействия с базами данных. Оно позволяет отказаться от написания большого количества SQL-запросов, предоставляя удобный инструментарий ORM (Object-Relational Mapping).

Spring Security — отвечает за безопасность приложения. Он используется для аутентификации, авторизации и защиты данных от внешних угроз.

Spring является ядром серверной логики, так как предоставляет всё необходимое для работы с HTTP-запросами, базами данных и обработки бизнес-логики.

База данных — это фундамент для хранения информации в любом веб-приложении. Выбор зависит от спецификации задачи и объёма данных, с которыми будет работать приложение. Для приложения будет использоваться MySQL. Это свободная реляционная система управления базами данных (СУБД). Под словом «свободная» подразумевается ее бесплатность, под «реляционная» – работа с базами данных, основанных на двумерных таблицах.

Для создания интерфейса приложения используется HTML. Для упрощения разработки интерфейса можно использовать фреймворк Bootstrap — это открытый и бесплатный HTML-, CSS- и JS-фреймворк, который используют веб-разработчики для быстрой верстки адаптивных дизайнов сайтов и веб-приложений. Включает в себя CSS- и HTML-шаблоны оформления для веб-форм, меток, типографики, кнопок, блоков навигации и других компонентов веб-интерфейса. Также используется для генерации HTML-страниц на сервере шаблонизатор Thymeleaf. Thymeleaf интегрируется хорошо со Spring и позволяет удобно работать с данными из Java кода.

Современные приложения часто разворачиваются в облачных средах, обеспечивая доступность и масштабируемость. Для приложения будет использоваться Docker. Контейнеры упрощают развёртывание приложения независимо от операционной системы.

Для повышения производительности процесса разработки и автоматизации развертывания будет использоваться:

Maven — инструмент для управления зависимостями и сборки проекта.

Git — система контроля версий.

GitHub — для настройки автоматического развёртывания и тестирования приложения.

В качестве среды разработки будет использоваться IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java.

### 3.2 Проектирование общей архитектуры приложения

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:

Model (Модель) инкапсулирует (объединяет) данные приложения. Включает классы, описывающие сущности приложения ("MyUser"(пользователь), "Post"(пост в дневник эмоций), "Tracker"(запись в дневник эмоций)).

View (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.

Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид. 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. В нашем случае у нас будут AuthController и RegistrationController для обработки регистрации и аутентификации, WebController для обработки страниц регистрации. PostContoller для работы со страницами дневника чувств.

Репозиторий (Repository) 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. (MyUserRepository, PostRepository, TrackerRepositiry).

Сервис (Service) 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. В приложении будут такие сервисы – JwtService, MyUserDetailService, MyUserService.

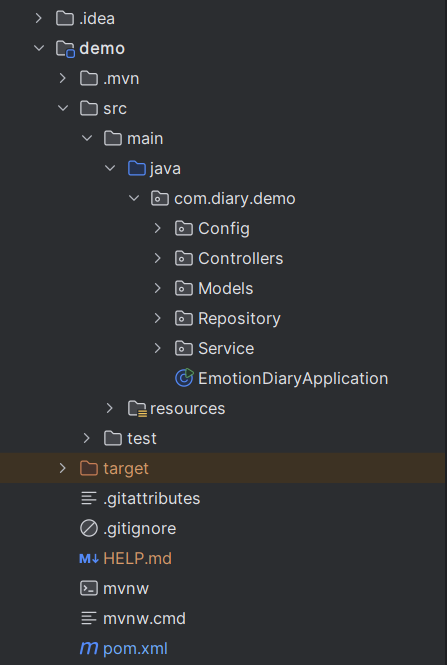
Конфигурация (Config) используется для настройки параметров конфигурации веб-сайта.Соответственно, он может оказать значительное влияние на производительность или поведение приложения.

Ресурсы (resourses) включаются в себя компоненты для фронтеда. Html-страницы, Thymleaf-блоки и страницы для отображения ошибок.

Файл application.properties – служит для хранения параметров конфигурации приложения, таких как строка подключения к базе данных, пути к файлам и других.

Файл pom.xml содержит информацию о проекте и различных деталях конфигурации, которые используются Maven.

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.



### Разработка бэкенда приложения

Первое, что предстоит сделать – это создать проект с использованием Spring Initializr. Выбраны необходимые зависимости, такие как Spring Web, Spring Data JPA, Spring Security, Thymeleaf, MySQL Driver, JDBC API для поддержки функциональности приложения.

Далее был создан класс MyUser. Этот класс представляет сущность пользователя в приложении. Этот класс используется для хранения информации о пользователях, таких как имя, пароль и роль.

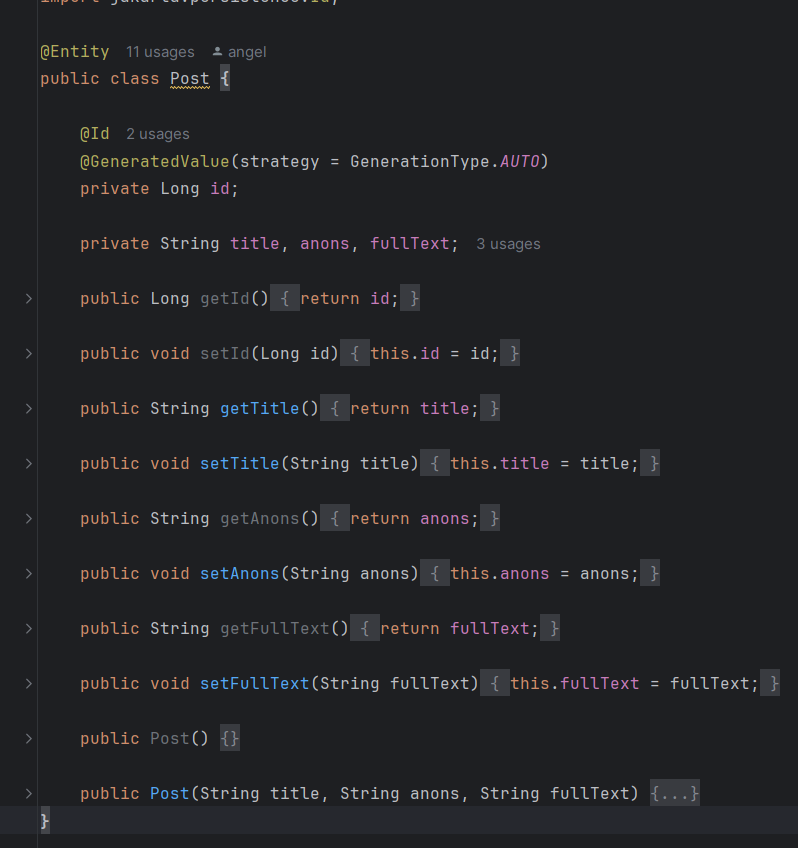
Аннотация @Entity указывает, что этот класс является сущностью для базы данных.

@Id: Обозначает поле как первичный ключ в таблице базы данных.

@GeneratedValue(strategy = GenerationType.AUTO) - автоматически генерирует значения для первичного ключа. Стратегия AUTO позволяет Hibernate выбрать вариант генерации, подходящий для конкретной базы данных.



Класс Post следует такой же логике и представляет из себя модель поста в приложении. Этот класс используется для хранения информации о посте, такие как заголовок, краткое описание и полный текст.



Последний класс в модели – это модель Tracker. Класс является моделью записи ежедневных чувств в приложении. В нем хранится информация о дате и эмоции.



Следующими были созданы конроллеры. Класс PostController обрабатывает HTTP-запросы, которые поступают от клиента (браузера, приложения или другого веб-клиента), и взаимодействует с базой данных для выполнения CRUD-операций (Create, Read, Update, Delete) над объектами Post и Tracker. Затем он возвращает соответствующее представление (HTML-страницу) или перенаправляет пользователя на другую веб-страницу.

Аннотация @Autowired в Spring Framework используется для автоматического связывания компонентов бина между собой. Она позволяет автоматически настраивать свойства бина и методы, упрощая тем самым процесс инъекции зависимостей. С помощью @Autowired подключаются два репозитория:

PostRepository для работы с постами (Post) — создание, редактирование, удаление и получение данных.

TrackerRepository для работы с трекером эмоций (Tracker) — добавление и удаление записей о настроениях пользователей.

Аннотация @RequestMapping используется для отображения запросов на методы контроллеров. Она имеет различные атрибуты для сопоставления по URL-адресу, HTTP-методу, параметрам запроса, заголовкам и типам среды передачи данных.

@PostMapping — это аннотациz, которая используется для сопоставления HTTP-запросов POST с определенными методами обработчика . Это сокращение для аннотации @RequestMapping с методом = RequestMethod. Атрибут POST.

@RequestParam - это аннотация, которая перехватывает параметр запроса. Это то, что передается после вопросительного знака в строке запроса.

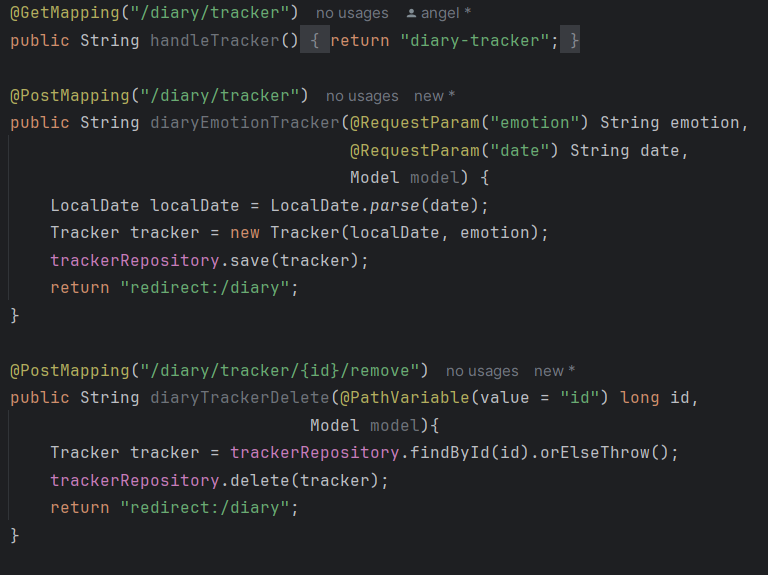
Логика обработки запросов заключается прежде всего в чтении данных (R - Read) Метод handlePost - обрабатывает GET-запрос на адрес /diary, загружает все записи из базы данных, вызвав метод findAll() репозиториев PostRepository и TrackerRepository, добавляет список записей (posts и trackers) в модель Model, чтобы их можно было отобразить в представлении diary и возвращает имя HTML-шаблона "diary", который будет отображать эти данные на странице.

Создание новой записи (C - Create) содержит в себе методы handleUserHome и diaryPostAdd. Первый возвращает страницу для добавления нового поста (diary-add), которая отображает форму для ввода данных. Второй обрабатывает POST-запрос на /diary/add, получает данные формы через параметры запроса: title, anons, fullText, создаёт новый объект типа Post, сохраняет новый пост в базе данных с помощью метода postRepository.save(post) и перенаправляет пользователя на страницу /diary, где можно увидеть добавленный пост.

Редактирование записи включает два метода - postEdit и diaryPostUpdate. Метод postEdit  обрабатывает GET-запрос по пути /diary/{id}/edit, где {id} — идентификатор редактируемого поста, проверяет, существует ли пост с указанным ID (через postRepository.existsById(id)). Если поста с таким идентификатором нет, перенаправляет пользователя на /diary, а если пост найден, извлекает его данные и передаёт их в модель, чтобы они появились в форме редактирования. Затем возвращает HTML-шаблон diary-edit, где отображается форма редактирования. Метод diaryPostUpdate обрабатывает POST-запрос на путь /diary/{id}/edit, извлекает запись из базы данных по ID, обновляет её свойства (заголовок, анонс, полный текст) на основе данных, полученных из формы, сохраняет изменения в базе данных и перенаправляет пользователя обратно на страницу /diary.

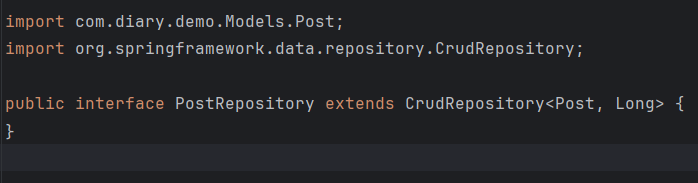
Удаление записи (D - Delete) включает методы diaryPostDelete (для постов) и diaryTrackerDelete (для записей трекера).Они обрабатывают POST-запрос на соответствующие URL (/diary/{id}/remove для постов и /diary/tracker/{id}/remove для трекеров), извлекают объект по ID из базы данных, удаляют объект через delete метод репозитория и перенаправляют пользователя на страницу /diary для просмотра оставшихся элементов.

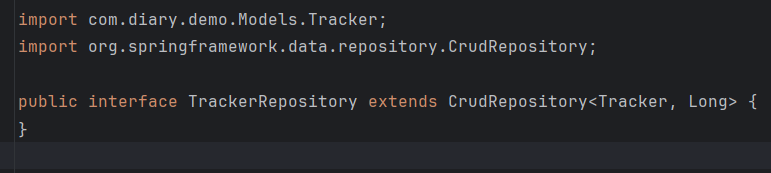
 

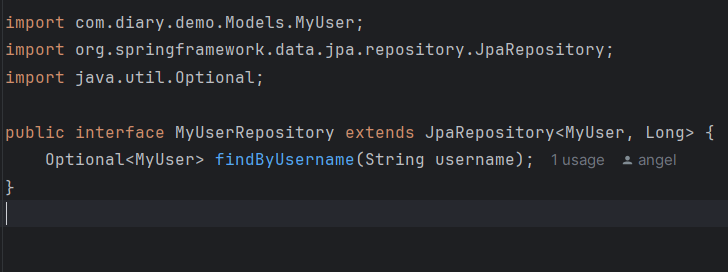


Были упомянуты интерфейсы MyUserRpository, TrackerRepository и PostRepository. Они расширяют JpaRepository и предоставляет стандартные методы для работы с базой данных, такие как save, findById, findAll, delete.

Метод findByUsername(String username) позволяет находить пользователя по полю username. Spring Data JPA автоматически генерирует SQL-запрос на основе имени метода (SELECT \* FROM my\_user WHERE username = ?).







### 3.3 Безопасность приложения

Для обеспечения безопасности приложения был использовать фреймворк Spring Security.

Spring Security — это мощный модуль Spring Framework, предназначенный для реализации безопасности в приложениях Java. Он предоставляет готовые инструменты для выполнения задач аутентификации (проверка личности пользователя) и авторизации (контроль доступа), а также для защиты приложений от угроз, таких как CSRF, XSS и другие. Spring Security позволяет разработчику легко интегрировать проверку логина и пароля в приложение, задавать права доступа на основе ролей, использовать защиты HTTP-заголовков, шифровать пароли и подключать внешние системы аутентификации, такие как OAuth2 или LDAP.

Логика работы Spring Security строится на уровне фильтров: каждый запрос, поступающий в приложение, проходит через цепочку фильтров, где проверяется аутентификация пользователя и его права доступа. Например, если пользователь пытается зайти на защищённый ресурс, система проверяет, прошёл ли он аутентификацию (введён логин и пароль) и обладает ли необходимыми правами (ролью).

Типичные задачи безопасности можно легко настроить через Java-код с использованием классов Spring Security, таких как WebSecurityConfigurerAdapter, а также через аннотации, например, @EnableWebSecurity и @PreAuthorize. Spring Security автоматически добавляет стандартные механизмы, такие как редирект на страницу входа при неаутентифицированном запросе, обработка выхода из аккаунта, а также защита от CSRF-атак. Это решение применяется для построения любых современных приложений — от простых до корпоративных — и предоставляет гибкость в настройке, что позволяет адаптировать его под потребности конкретного проекта, обеспечивая безопасность данных и защищая приложение от многих уязвимостей.

Рассмотрим класс SecurityConfiguration — это конфигурационный класс, который настраивает безопасность приложения с помощью Spring Security. Он аннотирован с помощью @Configuration (указывает, что это класс конфигурации Spring) и @EnableWebSecurity (включает поддержку безопасности Spring Security).

Основная задача класса — настроить аутентификацию, авторизацию, фильтрацию запросов и управление поведением приложения в части безопасности.

Прежде всего Инжектируется MyUserDetailService, который реализует интерфейс UserDetailsService. Этот сервис отвечает за получение данных о пользователях из базы данных или другого источника. Используется для аутентификации пользователей.

Метод securityFilterChain настраивает всю безопасность приложения. Этот метод конфигурирует цепочку фильтров на базе объекта HttpSecurity. В методе происходит обработка основных объектов.

Сначала отключается CSRF (Cross-Site Request Forgery), так как используется JWT для авторизации (JWT делает CSRF менее актуальным, поскольку токены передаются в заголовке запроса).

Затем происходит авторизация запросов, определяются правила доступа для URL:

/home, /register, /authenticate, /article: доступны всем без аутентификации.

/admin/\*\*: доступны только пользователям с ролью ADMIN.

/user/\*\*: доступны только пользователям с ролью USER.

Все остальные запросы требуют аутентификации.

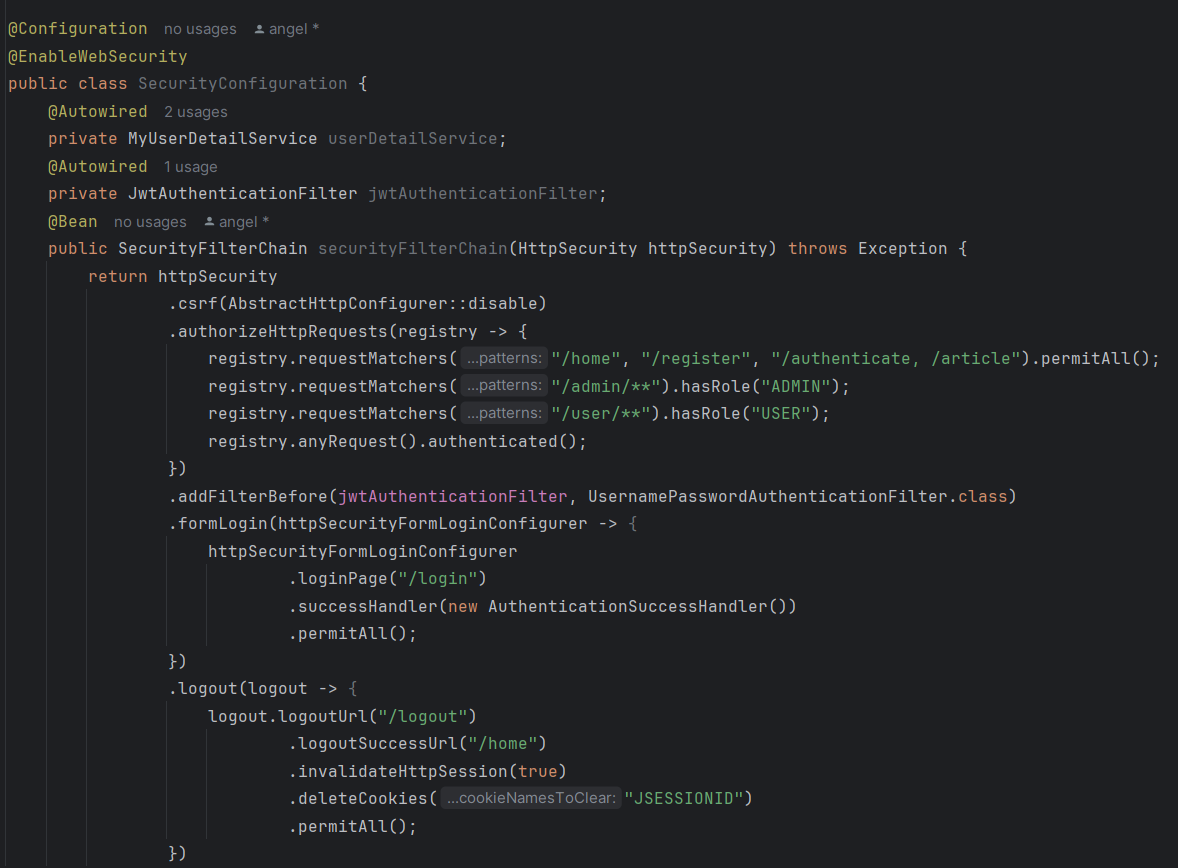
Пользовательский jwtAuthenticationFilter добавляется перед стандартным фильтром UsernamePasswordAuthenticationFilter, чтобы сначала обрабатывать JWT-токены.

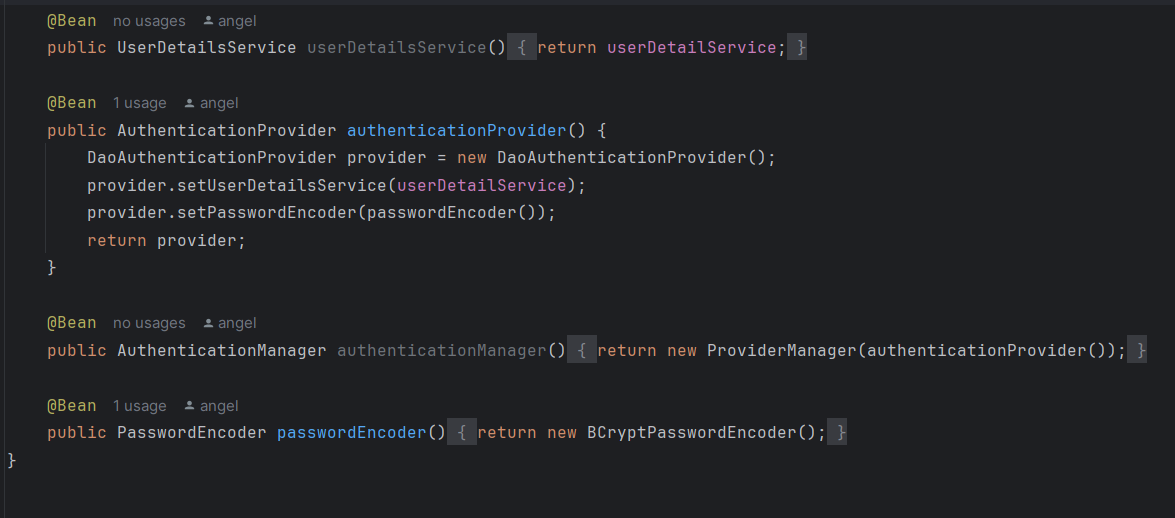
Указывается кастомная страница логина (/login). После успешного входа применяется обработчик AuthenticationSuccessHandler, который позволяет настроить поведение после аутентификации. Доступ к форме логина разрешён всем.

При выходе срабатывает такая логика: сессия пользователя уничтожается, удаляются cookie (JSESSIONID) и пользователь перенаправляется на /home.

Регистрируется MyUserDetailService как имплементация интерфейса UserDetailsService. Это будет использоваться для получения данных пользователя (логин, пароль, роли) при аутентификации.

Для шифрования паролей используется BCryptPasswordEncoder, что обеспечивает безопасность при хранении пароля в базе данных.





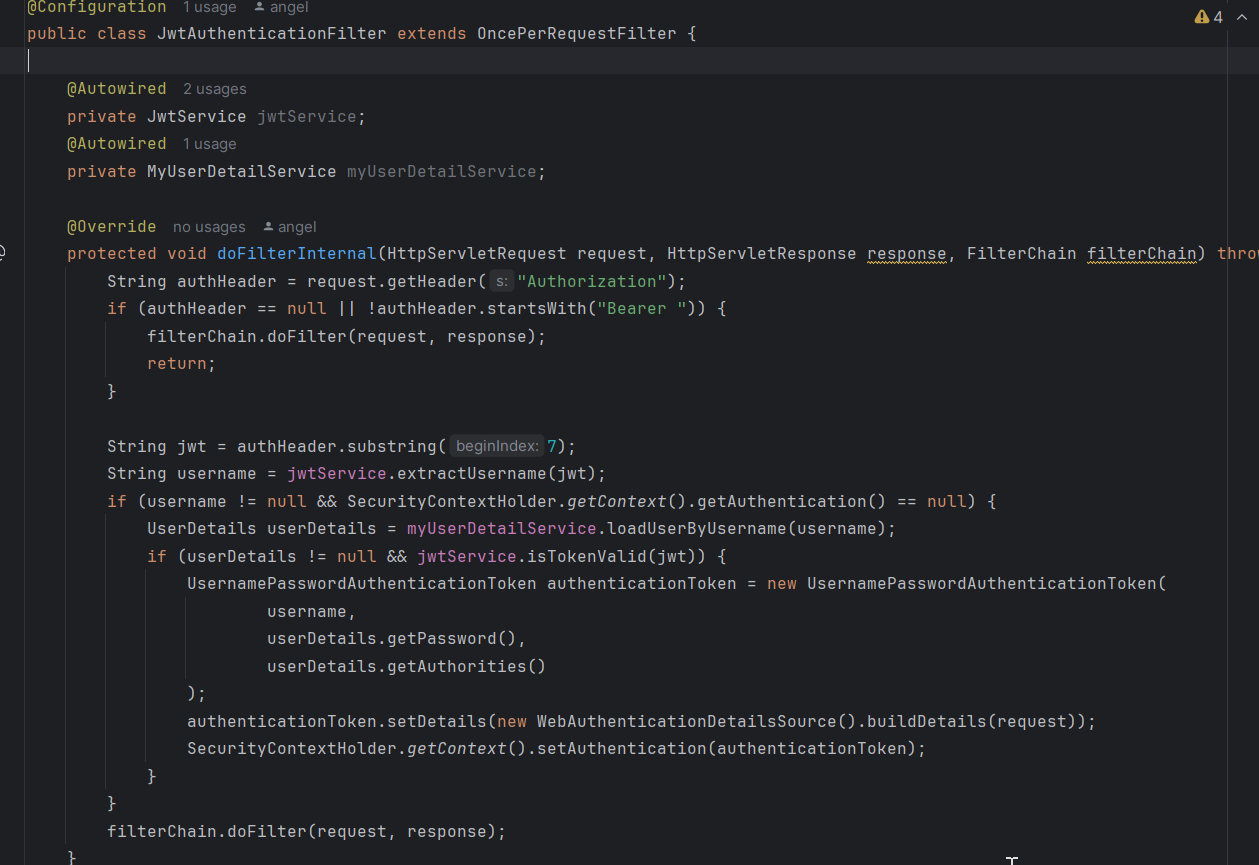
Класс JwtAuthenticationFilter — это фильтр, который выполняется один раз для каждого запроса (так как он расширяет класс OncePerRequestFilter) и отвечает за проверку JWT-токенов в входящих HTTP-запросах. Он проверяет, есть ли JWT-токен в заголовке Authorization, валиден ли токен, и устанавливает аутентифицированного пользователя в контекст безопасности (SecurityContextHolder).

Аннотация @Configuration обозначает, что данная конфигурация будет зарегистрирована как компонент в контексте Spring.

Фильтр обрабатывает каждый входящий HTTP-запрос. Его основной метод doFilterInternal действует по следующей логике: получение заголовка Authorization, заголовок проверяется для извлечения JWT-токена, JWT-токены обычно передаются в этом заголовке в формате: Bearer <token> затем происходит проверка наличия и формата токена. Если заголовок отсутствует или не начинается с Bearer (что означает, что он не содержит JWT-токен), фильтр пропускает запрос дальше по цепочке (filterChain.doFilter(...)) без выполнения дополнительной логики. Потом происходит извлечение токена и имени пользователя, из заголовка извлекается токен, отбрасывая часть Bearer. Если имя пользователя успешно извлечено и в текущем контексте безопасности (SecurityContextHolder) отсутствует аутентифицированный пользователь, выполняется дальнейшая логика (иначе запрос пропускается). Загружается объект UserDetails (пользователь), который содержит всю информацию о пользователе (логин, пароль, роли и другие атрибуты). Это выполняется с помощью MyUserDetailService, который работает с базой данных.

Если userDetails найден, и токен проходит проверку на валидность (например, не истёк, подписан корректно), пользователь считается аутентифицированным.

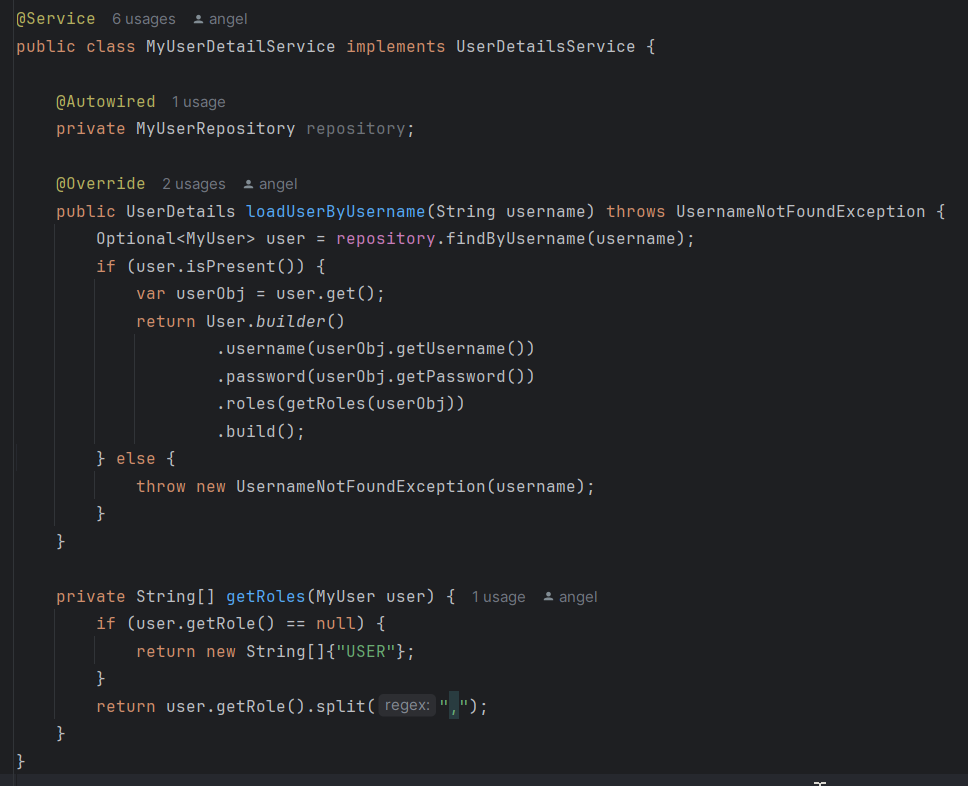
Создаётся объект UsernamePasswordAuthenticationToken, содержащий информацию об имени пользователя, пароле и ролях (авторизация), токен добавляется в SecurityContextHolder, который используется Spring Security для определения аутентифицированного пользователя.Затем запрос передаётся дальше по цепочке фильтров.



Класс MyUserDetailService реализует интерфейс UserDetailsService из Spring Security. Этот класс отвечает за загрузку данных пользователя из источника данных (например, базы данных) и возвращает объект UserDetails — стандартный интерфейс Spring Security для представления пользователя. MyUserDetailService используется в процессе аутентификации Spring Security для проверки учетных данных (например, имени пользователя и пароля) и получения информации о ролях (авторизации).

Spring Security вызывает этот метод при попытке аутентификации пользователя. Аннотация @Override указывает, что метод переопределён из интерфейса UserDetailsService.

Работа метода loadUserByUsername начинается с поиска пользователя в базе данных, используется репозиторий MyUserRepository, чтобы найти пользователя с указанным именем (username). Возвращается объект Optional<MyUser>, который может содержать пользователя, если он найден или может быть пустым, если пользователь с таким именем отсутствует.Если пользователь найден (user.isPresent()), метод продолжает обработку. А если пользователь не найден, выбрасывается исключение UsernameNotFoundException, уведомляющее Spring Security о том, что аутентификация невозможна.



Класс JwtService отвечает за управление созданием, валидацией и декодированием JWT-токенов (JSON Web Tokens). Он предоставляет функционал для генерации токена на основе данных пользователя, проверки токена на валидность и извлечения информации из токена.

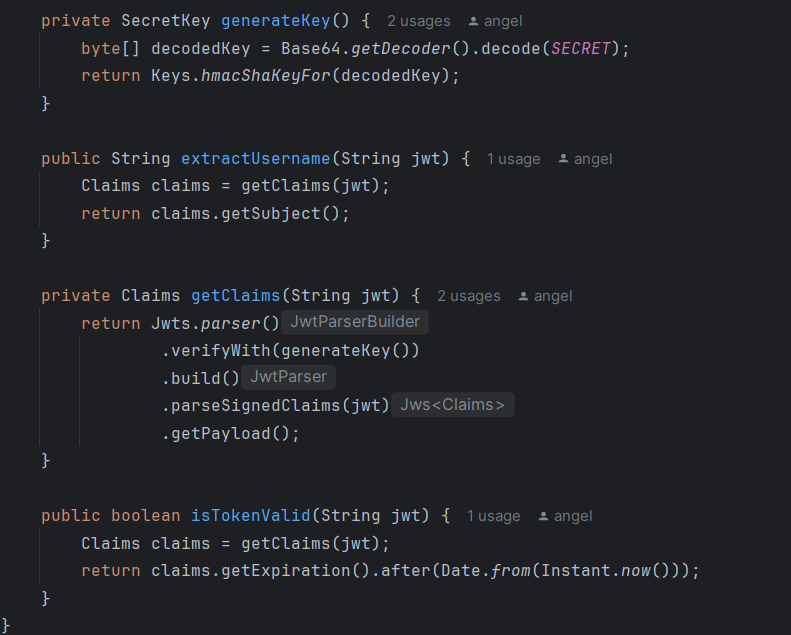
Основными аспектами класса являются секретный ключ (SECRET) ключ, который используется для подписи (signing) токенов. Это гарантия того, что токен, подписанный сервером, невозможно подделать. И время действия токена (VALIDITY) - токен будет действителен 30 минут с момента создания.

Сначала создаётся объект Map для хранения дополнительных заявлений (claims). Здесь добавляется iss (issuer), указывающий, что токен был выпущен <https://secure.emotiondiary.com>, используется билдер Jwts.builder() для создания токена и возвращается созданный JWT в виде строки.

Дальше описана генерация ключа для подписи токена. Секретный ключ декодируется из Base64-строки в массив байтов (byte[]). На основе декодированных данных создаётся ключ для алгоритма подписи HMAC-SHA256 (с использованием метода Keys.hmacShaKeyFor). Этот ключ используется для подписи и проверки JWT.

Извлечение имени пользователя из токена происходит по следующей логике: метод принимает токен (jwt), с помощью приватного метода getClaims извлекаются все claims (заявления) из токена и с помощью метода getSubject() извлекается свойство subject (в нём хранится имя пользователя), затем возвращается имя пользователя.

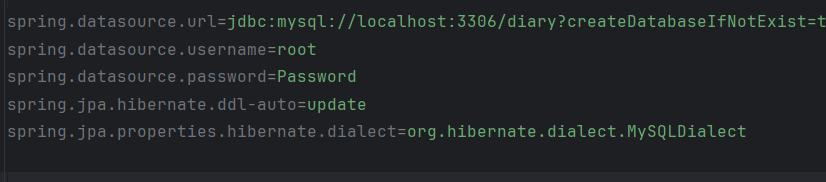




**3.4 Проектирование базы данных**

В качестве базы данных для приложения был выбран MySQL.

Application.properties содержит следующие настройки:



Эти параметры представляют собой конфигурацию подключения к базе данных и настройку Hibernate в приложении Spring Boot.

1. spring.datasource.url=jdbc:mysql://localhost:3306/diary?createDatabaseIfNotExist=true

jdbc:mysql://localhost:3306/diary: Это URL для подключения к базе данных.

jdbc:mysql: Протокол соединения (для работы с MySQL).

localhost: Хост сервера базы данных. Здесь используется локальный хост (машина, где запущено приложение).

3306: Стандартный порт для MySQL.

diary: Имя базы данных, с которой приложение будет работать.

?createDatabaseIfNotExist=true: Указывает MySQL автоматически создать базу данных diary, если она отсутствует.

2. spring.datasource.username=root

root: Имя пользователя для подключения к базе данных. Значение root — это стандартный пользователь, который создаётся в MySQL по умолчанию. В реальных приложениях лучше использовать другого, менее привилегированного пользователя.

3. spring.datasource.password=Password

Password: Пароль для указанного пользователя (в данном случае root).

Это пароль, установленный на сервере MySQL для данного пользователя.

4. spring.jpa.hibernate.ddl-auto=update

Этот параметр определяет, как Hibernate работает с моделями данных (Entity классами) и сущностями в базе данных. Он управляет созданием, обновлением, инициализацией или удалением таблиц в вашей базе данных на основе аннотированных классов. Update автоматически обновляет таблицы в базе данных при каждом запуске приложения. Если в сущностях (Entity) произошли изменения (например, добавлены или удалены столбцы), Hibernate внесёт эти изменения в таблицы базы данны

5. spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

hibernate.dialect: Этот параметр указывает Hibernate, как именно нужно взаимодействовать с конкретной СУБД. Hibernate поддерживает работу с разными базами данных (например, PostgreSQL, Oracle, MySQL), и для каждой из них есть свой диалект. Диалект — это набор правил, определяющих SQL-запросы, которые Hibernate отправляет базе данных.

org.hibernate.dialect.MySQLDialect: Указывает, что приложение использует MySQL в качестве базы данных, и Hibernate должен в соответствии с этим генерировать SQL-запросы.

### 3.5 Проектирование WEB – интерфейса приложения

Для проектирования WEB – интерфейса приложения был использован HTML. После прохождения авторизации пользователь попадает на страницу home\_user.html.Данный HTML-код с использованием Thymeleaf является шаблоном веб-страницы для "Дневника эмоций".

<!DOCTYPE HTML> Указывает тип документа страниц, написанных на HTML. DOCTYPE сообщает браузеру, что документ написан на языке HTML 5. Это помогает браузеру правильно отображать разметку документа.

Тег <html> является корневым тег всего документа HTML. Вся HTML-разметка находится внутри него. xmlns:th – это атрибут пространства имён для Thymeleaf, который используется для определения Thymeleaf-шаблонов. Thymeleaf добавляет динамическую логику к статическому HTML.

Секция <head> содержит метаинформацию о странице, например, её заголовок, стили и внешние ресурсы.

<title>Главная страница</title> - заголовок веб-страницы, который отображается на вкладке браузера или в результатах поиска.

Добавлены встроенные стили:

.container-intro: Текст внутри этой секции будет по центру.

.container-add и .container-emotion: Таблицы определяют отступ внутри боксов и способ расчёта размеров боксов (box-sizing).

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

Указывает кодировку страницы как UTF-8. Это важно для поддержки всех символов (например, русских).

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"> Подключает CSS-фреймворк Bootstrap. Bootstrap помогает создавать адаптивную и стильную разметку, упрощая стилизованный интерфейс.

Тэг </body> содержит основное содержимое (главную структуру) страницы, которая видна пользователю.

 Тег <header th:insert> явяяется Thymeleaf-атрибутом, который вставляет содержимое блока "header" (закладка шаблонного фрагмента) из внешнего шаблона blocks/header.

Основное назначение секции с классом container-intro представляет "Добро пожаловать" или вводный текст.

Элементы этой секции:

<h1>: Заголовок Дневник эмоций — основной заголовок страницы.

<h3>: Подзаголовок, поясняющий назначение дневника эмоций.

<p> + <a>:

Содержит ссылку (<a>) /article, на страницу с подробной информацией о ведении дневника эмоций.

Классы Bootstrap:

mt-5: Устанавливает внешний отступ сверху (margin-top: 5).

mb-5: Устанавливает внешний отступ снизу (margin-bottom: 5).

text-center (явно не указан в исходнике, но может быть полезен для стилизации).

Также в других секциях используются такие классы Bootstrap:

btn: Базовый стиль кнопки.

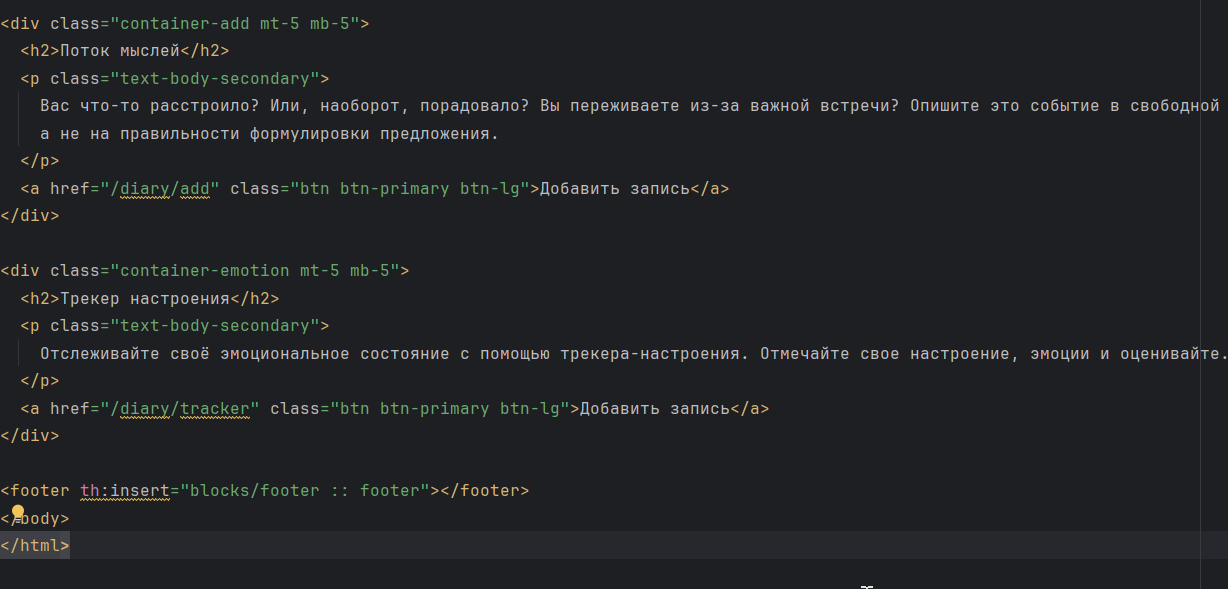
btn-primary: Задаёт основной стиль кнопки (например, синий цвет по умолчанию).

btn-lg: Устанавливает размер кнопки как "большой".

Остальные секции строятся таким же образом.

Тег <footer th:insert> - это Thymeleaf-атрибут, который подключает шаблонную секцию "footer".





* В других файлах html описаны формы, которые работают по похожему принципу. На примере разберем секцию в html документе «diary-add».

Форма <form> — это основной тег, позволяющий отправлять данные пользователя (например, текст) на сервер.

Атрибуты формы:

action="/diary/add" указывает адрес, куда отправляются данные формы при её отправке.

method="post" указывает, каким HTTP-методом выполняется запрос. Зost используется для отправки данных, которые должны быть сохранены или обработаны.В данном случае предполагается, что запись будет добавлена в базу данных.

Поле ввода <input> создаёт текстовую строку, куда пользователь может вводить данные. Оно используется для ввода названия записи.

Атрибуты:

type="text" указывает, что это текстовое поле, предназначенное для ввода текстовой информации.

name="title" указывает имя этого поля (ключ). Это имя будет отправлено на сервер вместе с введённым текстом. Сервер сможет идентифицировать значение по ключу title.

placeholder="Введите название" показывает серый текст внутри поля до тех пор, пока пользователь не начнёт вводить свой собственный текст.

class="form-control" - это класс Bootstrap для стилизации поля ввода. Добавляет к полю оформления (например, рамку, отступы и адаптивность).

Кнопка отправки формы: <button>. Когда пользователь нажимает на неё, данные из формы отправляются на сервер.

Атрибуты:

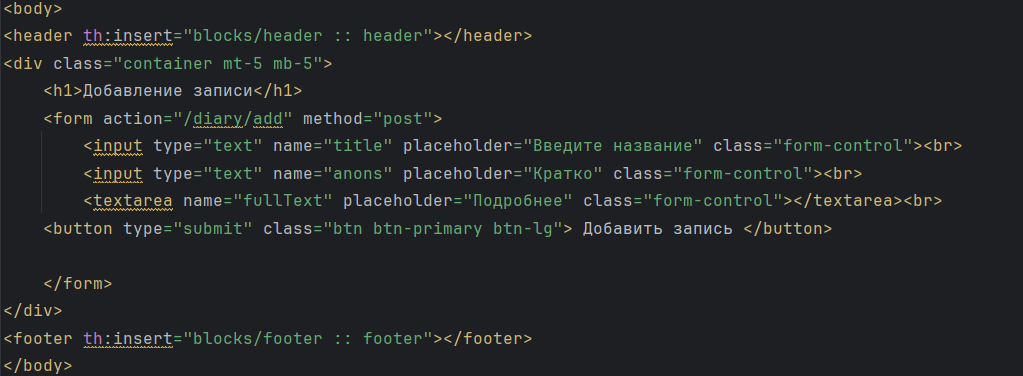
type="submit" указывает на то, что это кнопка команды "отправить". При нажатии форма отправляет данные на указанный адрес (action).

class="btn btn-primary btn-lg" – это классы Bootstrap для стилизации:

btn: Базовый стиль кнопки.

btn-primary указывает, что кнопка должна быть синей (или другого цвета в зависимости от темы).

btn-lg устанавливает размер кнопки как "большой".



## ****Заключение****

Приложение «Дневник чувств», разработанное с использованием Spring Boot, представляет собой функциональное и интуитивно понятное решение, предназначенное для записи и анализа эмоций пользователей. Данный проект продемонстрировал возможности Spring Boot как мощного фреймворка, обеспечивающего удобную разработку современного веб-приложения.

Одним из ключевых аспектов стала работа с базой данных на основе Spring Data JPA, что упростило реализацию операций CRUD для записей и трекеров настроения. MVC-архитектура позволила разделить логику, интерфейсы и данные, обеспечив гибкость и ясность кода. Thymeleaf стал основным инструментом для динамической генерации HTML-страниц, а Bootstrap обеспечил адаптивность и современный интерфейс. Было успешно реализовано взаимодействие между клиентской частью и сервером, а также возможности для авторизации и аутентификации благодаря Spring Security. Форма для добавления записей организована таким образом, чтобы пользователи могли легко и интуитивно взаимодействовать с приложением, что делает его удобным и приятным в использовании.

Проект показал, как эффективно можно интегрировать технологии в современном приложении. Удобство использования и возможность масштабирования стали важными преимуществами. В будущем приложение может быть расширено графическим анализом эмоций, интеграцией с мобильными устройствами и добавлением дополнительных функций, таких как тегирование записей или построение диаграмм эмоционального состояния. Кроме того, возможно развитие API для сторонних сервисов и мобильных приложений.

Данный учебный проект наглядно демонстрирует принципы строительства веб-приложений на Spring Boot, ради чего были объединены передовые подходы программирования с понятными интерфейсами. Приложение является базой для дальнейшего улучшения и предоставляет условия для изучения ключевых аспектов разработки, от работы с базой данных и безопасности до организации пользовательского интерфейса.

### Список источников

Атрощенко, Н. А. Оценка подхода к разработке веб-приложений на Java и JavaScript / Н. А. Атрощенко // Теоретические и прикладные вопросы науки и образования : сборник научных трудов по материалам Международной научно-практической конференции: в 16 частях, Тамбов, 31 января 2015 года. Том Часть 11. – Тамбов: ООО "Консалтинговая компания Юком", 2015. – С. 10-11. – EDN TKDVPB.

Депрессивное расстройство (депрессия) [Электронный ресурс]:Всемирная организация здравоохранения - Режим доступа: <https://www.who.int/ru/news-room/fact-sheets/detail/depression> (дата обращения: 13.12.2024)

Ибраимов, Р. И. Spring Boot и Spring Initializr: как развернуть Spring Boot приложение за 5 минут / Р. И. Ибраимов, А. Б. Джемалетдинов, А. А. Шевченко // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. – 2017. – № 3(17). – С. 105-111. – EDN YMQFYB.

Ибраимов, Р. И. Spring Boot: создание тестов для Spring MVC контроллеров / Р. И. Ибраимов, А. Б. Джемалетдинов, А. А. Шевченко // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. – 2017. – № 4(18). – С. 104-111. – EDN VQLFBC.

Иванов, А. Н. Разработка приложения с использованием веб-технологий Java / А. Н. Иванов, П. С. Власюк // Ученые заметки ТОГУ. – 2020. – Т. 11, № 4. – С. 261-266. – EDN GDHSLM.

Матинова, Е. В. Архитектура веб приложений на основе Java EE технологий / Е. В. Матинова // Научно-исследовательский центр "Вектор развития". – 2021. – № 2. – С. 329-334. – EDN PLESKA.

Свидетельство о государственной регистрации программы для ЭВМ № 2020610538 Российская Федерация. Виртуальный HSM (версия 2.1) - Java-сервлет веб-приложения, реализующего функциональность модуля безопасности : № 2019667624 : заявл. 27.12.2019 : опубл. 15.01.2020 / А. С. Совков, Е. В. Ксенофонтов, С. Г. Новоченко ; заявитель Общество с ограниченной ответственностью "СМАРТ ЛАБ СОЛЮШЕНС". – EDN MDVNBW.

Свидетельство о государственной регистрации программы для ЭВМ № 2020610538 Российская Федерация. Виртуальный HSM (версия 2.1) - Java-сервлет веб-приложения, реализующего функциональность модуля безопасности : № 2019667624 : заявл. 27.12.2019 : опубл. 15.01.2020 / А. С. Совков, Е. В. Ксенофонтов, С. Г. Новоченко ; заявитель Общество с ограниченной ответственностью "СМАРТ ЛАБ СОЛЮШЕНС". – EDN MDVNBW.

COVID-19 pandemic triggers 25% increase in prevalence of anxiety and depression worldwide) [Электронный ресурс]: Всемирная организация здравоохранения - Режим доступа: https://www.who.int/news/item/02-03-2022-covid-19-pandemic-triggers-25-increase-in-prevalence-of-anxiety-and-depression-worldwide(дата обращения: 13.12.2024)

mm7024e1

Yard E, Radhakrishnan L, Ballesteros MF, et al. Emergency department visits for suspected suicide attempts among persons aged 12–25 years before and during the COVID-19 pandemic—United States, January 2019–May 2021. MMWR Morb Mortal Wkly Rep 2021;70:888–94. PMID:34138833 https://doi.org/10.15585/mmwr.

1. Депрессивное расстройство (депрессия) [Электронный ресурс]:Всемирная организация здравоохранения - Режим доступа: <https://www.who.int/ru/news-room/fact-sheets/detail/depression> (дата обращения: 13.12.2024) [↑](#footnote-ref-1)
2. COVID-19 pandemic triggers 25% increase in prevalence of anxiety and depression worldwide) [Электронный ресурс]: Всемирная организация здравоохранения - Режим доступа: https://www.who.int/news/item/02-03-2022-covid-19-pandemic-triggers-25-increase-in-prevalence-of-anxiety-and-depression-worldwide(дата обращения: 13.12.2024) [↑](#footnote-ref-2)
3. Yard E, Radhakrishnan L, Ballesteros MF, et al. Emergency department visits for suspected suicide attempts among persons aged 12–25 years before and during the COVID-19 pandemic—United States, January 2019–May 2021. MMWR Morb Mortal Wkly Rep 2021;70:888–94. PMID:34138833 https://doi.org/10.15585/mmwr.

   mm7024e1 [↑](#footnote-ref-3)