Geekbrains

**Разработка web приложения «Reminder» на JavaSpring.**

IT-специалист:

Разработка web приложений

Прокопенко О.А.

Таганрог

2024

**Дипломный проект**

**«Разработка веб приложения «Reminder» 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…………………….. | 20 |
| 2.3. Проектирование базы данных……………………………………….. | 23 |
| 2.4. Безопасность приложения……………………………………………. | 31 |
| 2.5. Разворачивание Docker контейнера…………………………………. | 36 |
| Заключение………………………………………………………………… | 43 |
| Список использованной литературы…………………………………….. | 44 |

**Введение**

Современный ритм жизни стремительно ускоряется, и все больше людей сталкиваются с проблемой забывчивости и необходимостью эффективно управлять своим временем. Именно поэтому разработка приложения Reminder представляется крайне актуальной задачей.

Reminder – это web приложение, предназначенное для создания напоминаний и управления ими. После авторизации пользователь попадает на главную страницу, где он может увидеть список своих напоминаний, отсортированных по дате и времени. Такой подход позволяет легко отслеживать предстоящие события и задачи, не забывая о важных делах.

Одной из функций Reminder является возможность создавать новые напоминания. Пользователь может задать название напоминания, описание, дату и время. Не менее важной является функция редактирования напоминаний. Жизнь непредсказуема, и часто возникает необходимость вносить изменения в уже созданные напоминания. Приложение должно предоставлять пользователю возможность легко изменять название, описание, дату. Удаление напоминаний также является необходимой функцией. После выполнения задачи или отмены события пользователь должен иметь возможность удалить соответствующее напоминание из списка, чтобы избежать беспорядка. Сортировка напоминаний по дате и времени поможет пользователю легко ориентироваться в списке и планировать свою деятельность. Кроме того, приложение должно предоставлять возможность фильтровать напоминания по различным критериям, таким как дата, время, категория, статус выполнения и т.д. Это позволит пользователю быстро находить необходимые напоминания и сосредоточиться на наиболее важных задачах.

Также при наступлении заданного времени, напоминание отсылается пользователю в Telegram и на электронную почту, которые он указывает при регистрации.

В заключение следует отметить, что разработка приложения Reminder является важным шагом в направлении повышения продуктивности и организованности пользователей. Благодаря широкому набору функций, таких как создание, редактирование, удаление, сортировка и фильтрация напоминаний, уведомление в telegram и на email приложение станет незаменимым помощником в планировании и управлении задачами. Кроме того, привлекательный и интуитивно понятный интерфейс сделает использование приложения максимально удобным и приятным.

Данное приложение было разработано студентом Geekbrains Прокопенко О.А.

**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 Java Development Kit (JDK), Apache Maven (для сборки проекта), и Spring Boot (для создания и запуска приложения).
3. **Создание проекта**:
   * Создан новый проект с использованием SpringInitializer.
   * Выбраны необходимые зависимости, такие как SpringWeb, SpringDataJPA, Lombok, SpringSecurity, Thymeleaf, TelegramBots, SpringBootStarterMail 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 "Reminder" были определены следующие требования:

1. Функциональные требования:
   * Пользователи должны иметь возможность создавать записи напоминаний.
   * Пользователи должны иметь возможность просматривать и редактировать свои записи.
   * Пользователи должны иметь возможность удалять свои записи.
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 напоминаниях.
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 SpringMVC и Spring Security для быстрой и удобной разработки серверной части приложения. Spring Security обеспечит безопасность приложения, а Spring MVC позволит создать удобный интерфейс.
3. **База данных**: PostgreSQL для хранения данных пользователей и их напоминаний. Обе системы хорошо масштабируются и обеспечивают надежное хранение данных.
4. **Фронтенд:** HTML для создания интерфейса приложения. Для упрощения разработки интерфейса можно использовать фреймворк Bootstrap.
5. **Шаблонизатор**: Thymeleaf для генерации HTML-страниц на сервере. Thymeleaf интегрируется хорошо со Spring и позволяет удобно работать с данными из Java кода.
6. **Аутентификация и авторизация**: SpringSecurity используется для обеспечения безопасности приложения, включая аутентификацию пользователей и управление доступом к ресурсам.
7. **Среда разработки**: IntelliJ IDEA для разработки серверной части приложения. Этот стек технологий обеспечивает разработку приложения "Reminder" 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). Приложение состоит из данных составных частей:

1. **Модель (Model)**:
   * Модель представляет собой структуру данных приложения.
   * Включает классы, описывающие сущности приложения ("User", "Reminder").
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 JavaSpring репозитории обычно реализуются с помощью интерфейсов, расширяющих **JpaRepository**, которые предоставляют базовые методы для работы с данными.
   * Репозитории используются сервисами для получения, сохранения, обновления и удаления данных.
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 инициализация полей класса Reminder. Класс Reminder представляет собой записи напоминаний. Каждая запись содержит информацию о названии (title), описании (description), дате наступления (remind), принадлежность пользователю (user) и флаг (isReminded), который при инициализации - false, а после отправки напоминания принимает значение true.

Аннотация **@Entity** в Spring Boot указывает, что класс считается сущностью, связанной с таблицей в базе данных.

Сущности используются для распознавания классов, представляющих модель данных. С их помощью Spring может выполнять обычные действия с базой данных, такие как вставка, удаление, поиск и т. д.

**@Table** — это аннотация в Spring Data JPA, которая используется для указания имени таблицы в конкретной базе данных. Attribute name — имя таблицы (опционально).

**@Data** — это аннотация, которая является частью библиотеки Lombok и предназначена для сокращения шаблонного кода и упрощения проектов на Java.

С помощью @Data Lombok автоматически генерирует следующие методы для класса:

* + геттеры для всех полей;
  + сеттеры для всех нефинальных полей;
  + методы equals(), hashCode() и toString().

Это делает код более читаемым и значительно сокращает время, затрачиваемое на написание повторяющегося кода.

**@Id** — это аннотация в Spring Data JPA, которая помечает поле в классе модели как первичный ключ. Она необходима для идентификации уникального идентификатора каждой persisted сущности в базе данных.

**@GeneratedValue** — это аннотация, которая в Spring Boot используется для указания стратегии генерации значения первичного ключа сущности. Обычно её применяют к полю первичного ключа, чтобы указать, что значение должно быть автоматически сгенерировано.

У аннотации @GeneratedValue есть следующие атрибуты:

strategy — стратегия, используемая для генерации значения первичного ключа. По умолчанию используется IDENTITY, что означает, что база данных сгенерирует значение автоматически. Другие поддерживаемые стратегии: SEQUENCE, TABLE и AUTO.

generator — имя генератора, который нужно использовать. Это используется только когда стратегия установлена в AUTO.

parameters — карта дополнительных параметров, которые нужно передать генератору.

Также можно определить собственную стратегию генерации, реализовав интерфейс org.hibernate.id.IdentifierGenerator. Чтобы использовать собственную стратегию, её нужно зарегистрировать в контексте приложения Spring Boot с помощью аннотации @Bean.

**@Column** — это аннотация в контексте Spring Boot с JPA (Java Persistence API), которая используется для указания того, как атрибут сущности сопоставляется с столбцом в базе данных.

Attributes of the @Column annotation:

name: указывает имя столбца в базе данных. По умолчанию оно совпадает с именем поля в сущности.

nullable: определяет, может ли столбец содержать нулевые значения. По умолчанию — true.

unique: указывает, должны ли значения в столбце быть уникальными. По умолчанию — false.

insertable: указывает, должно ли значение столбца включаться в операторы SQL INSERT. По умолчанию — true.

updatable: указывает, должно ли значение столбца включаться в операторы SQL UPDATE. По умолчанию — true.

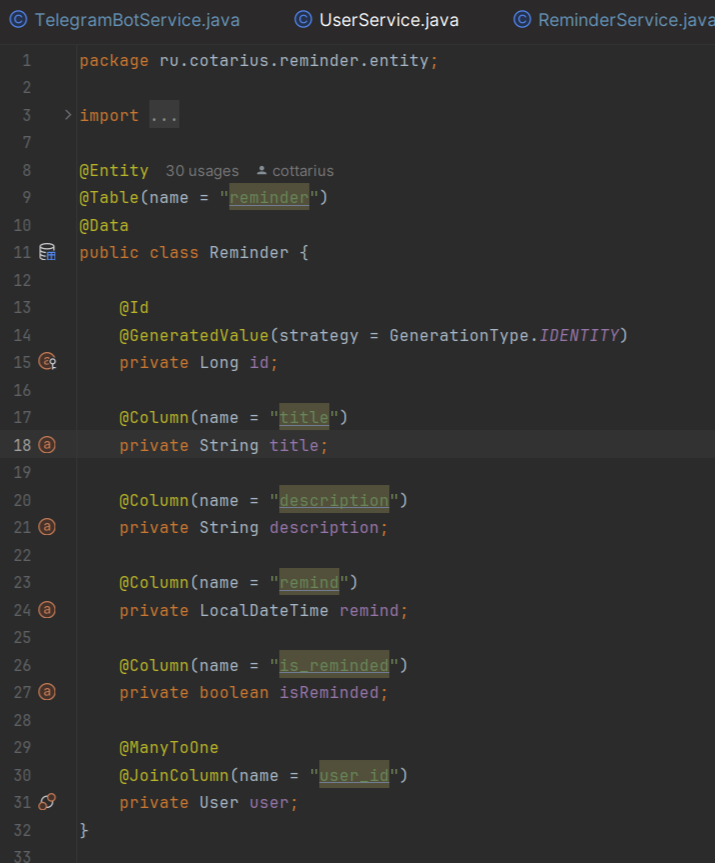
columnDefinition: указывает фрагмент SQL для создания столбца в базе данных (например, VARCHAR(255)). Используется, если нужно более точно определить столбец.

length: если столбец имеет строковый тип, указывает максимальную длину столбца. По умолчанию — 255.

**@ManyToOne** — это аннотация в Spring Data JPA, которая определяет связь «многие к одному» между двумя сущностями.

Наличие аннотации создаёт внешний ключ в базе. При этом сама колонка будет называться по имени поля в классе с добавлением \_id в конце.

@JoinColumn — это аннотация в JPA, которая используется для указания столбца, который отвечает за хранение значения внешнего ключа, устанавливающего связь между двумя сущностями. Она применяется к полю или свойству, которое представляет сторону ассоциации.



Класс ReminderService предоставляет функционал по работе с напоминаниями. Аннотация @Service говорит о том, что этот класс является сервисным компонентом, который предоставляет бизнес-логику для работы с данными.

В классе присутствуют три поля: reminderRepository, telegramBotService, myEmailService, а аннотация @RequiredArgsConstructor генерирует конструктор с аргументами для всех полей класса, что позволяет Spring автоматически внедрить зависимости.

Метод save сохраняет новое напоминание в репозиторий.

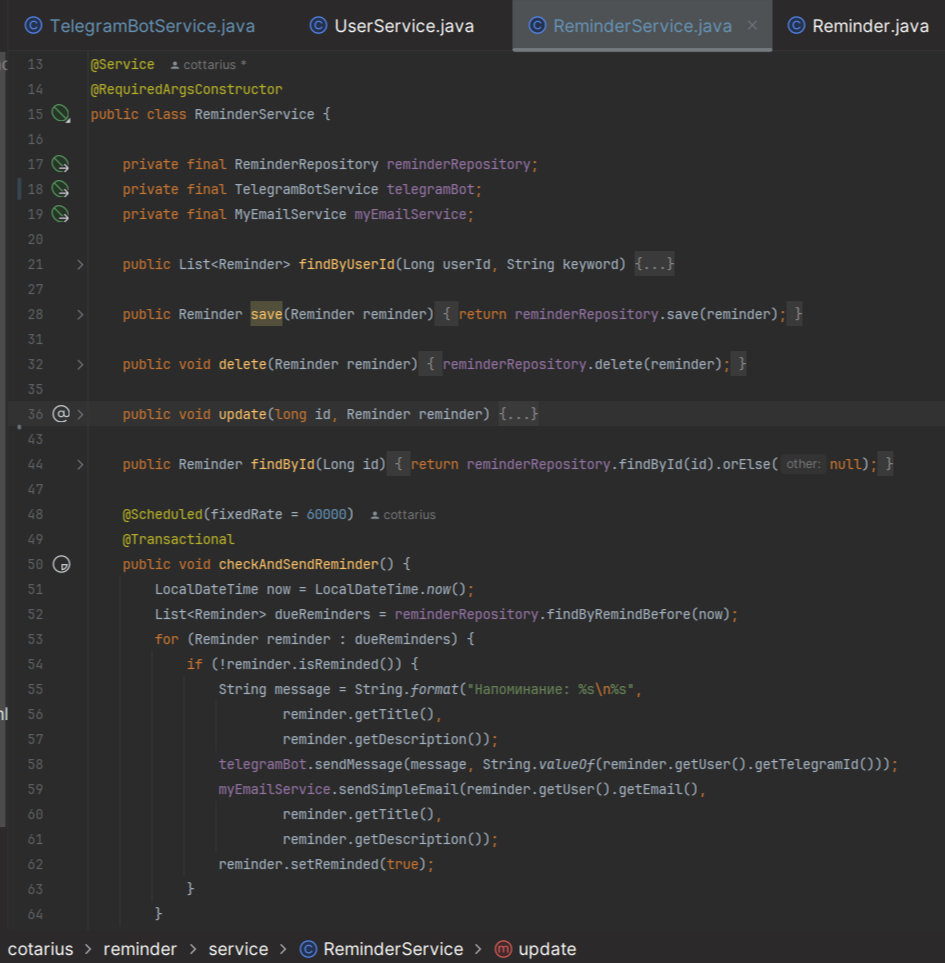
Метод delete удаляет напоминание из репозитория.

Метод update вносит изменение в текущее напоминание и сохраняет его

Метод findById выполняет поиск напоминания по его id.

Метод findByUserId выполняет возвращает все напоминания, принадлежащие авторизованному пользователю.

Метод checkAndSendReminder сканирует таблицу с напоминаниями «reminder» каждые 60 секунд (период можно изменить, выставив другое время в аннотации @Scheduled) и создает список напоминаний, время которых before текущего времени. Далее, пробегая по этому списку, происходит проверка флага isReminded и если он false, то напоминание отправляется пользователю в телеграм и на электронную почту. Флагу присваивается значение true.



ReminderController является компонентом веб-приложения, отвечающим за взаимодействие с пользователем и обработку запросов, связанных с управлением записями напоминаний. Контроллер использует сервисы ReminderService и UserService для выполнения операций с данными.

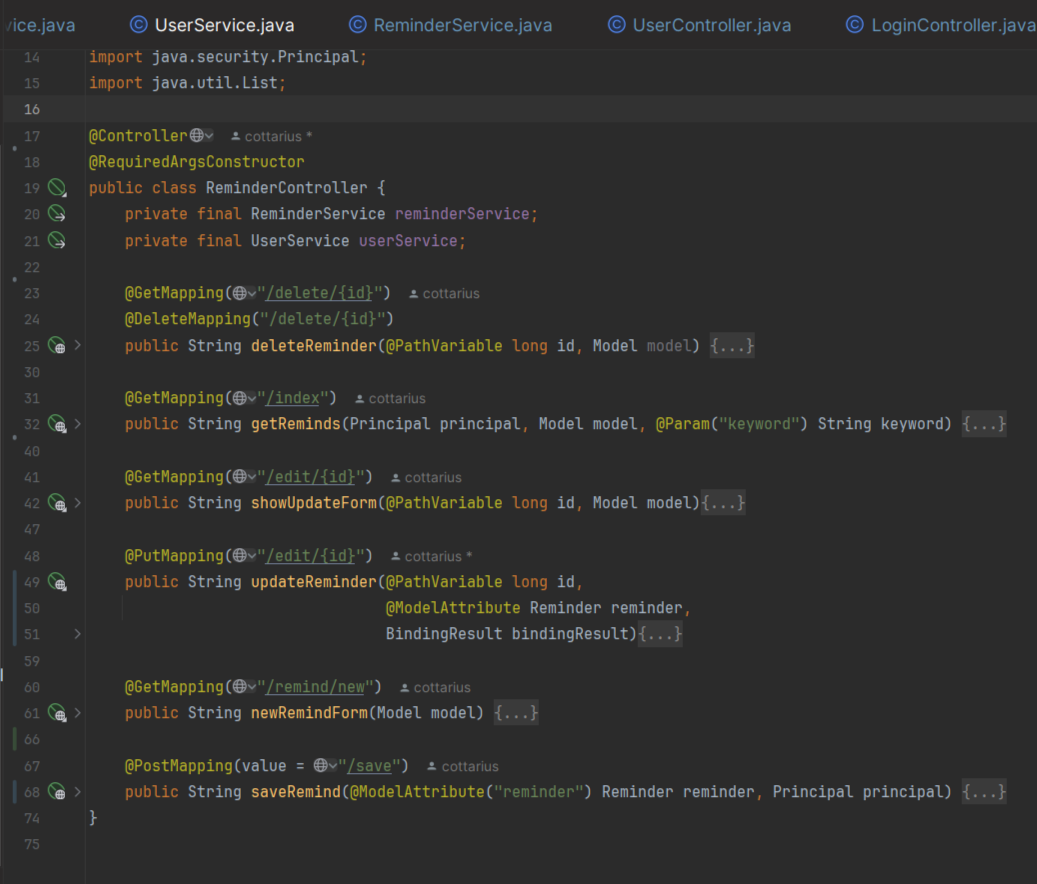
Методы контроллера позволяют пользователю просматривать, добавлять, изменять и удалять записи напоминаний. Кроме того, контроллер обеспечивает перенаправление пользователя на соответствующие страницы для отображения результатов операций.

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, метод newRemindForm отображает форму для создания новых напоминаний, а метод saveRemind обрабатывает отправленные данные и добавляют новые записи в базу данных.

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

В целом, ReminderController является важной частью приложения, обеспечивающей удобный интерфейс для работы с данными.



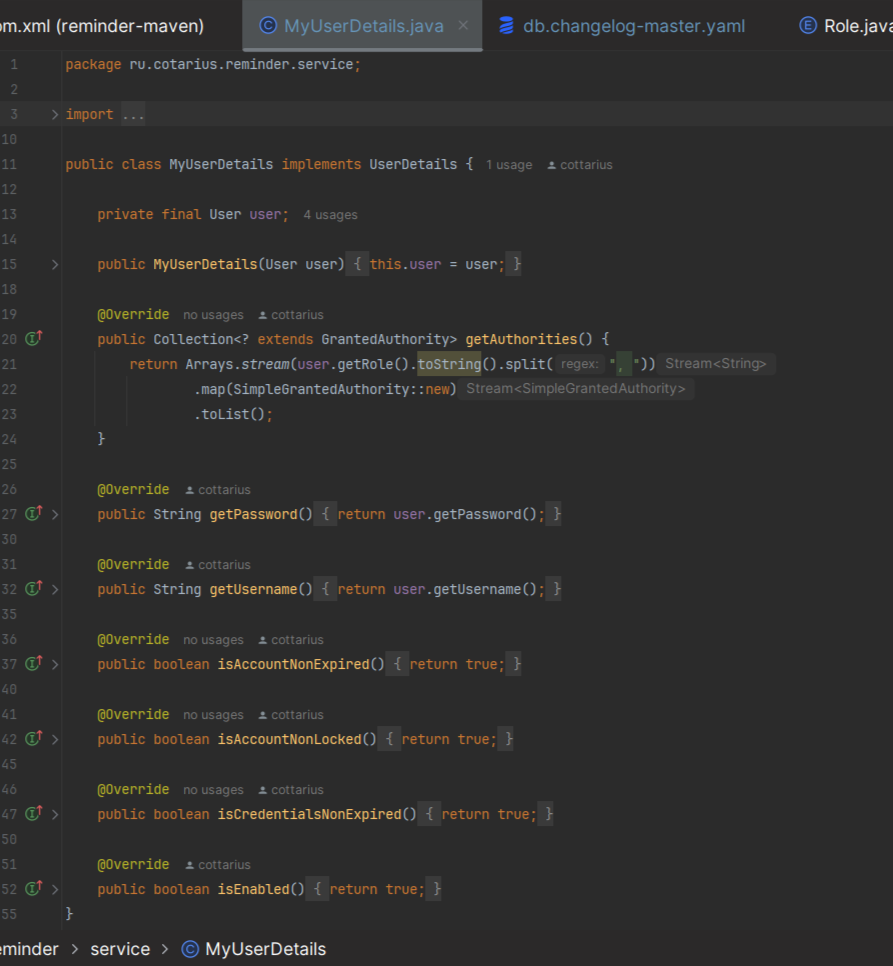
Класс MyUserDetails реализует интерфейс UserDetails, который используется SpringSecurity для представления информации о пользователе, необходимой для аутентификации и авторизации. Вот его функции:

1. privateUseruser; - поле, хранящее объект User, который представляет собой модель пользователя вашего приложения.
2. publicMyUserDetails(User user){...} - 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.

Класс MyUserDetails, который реализует интерфейс UserDetails, используется для предоставления информации о пользователе, необходимой для аутентификации и авторизации в приложении.

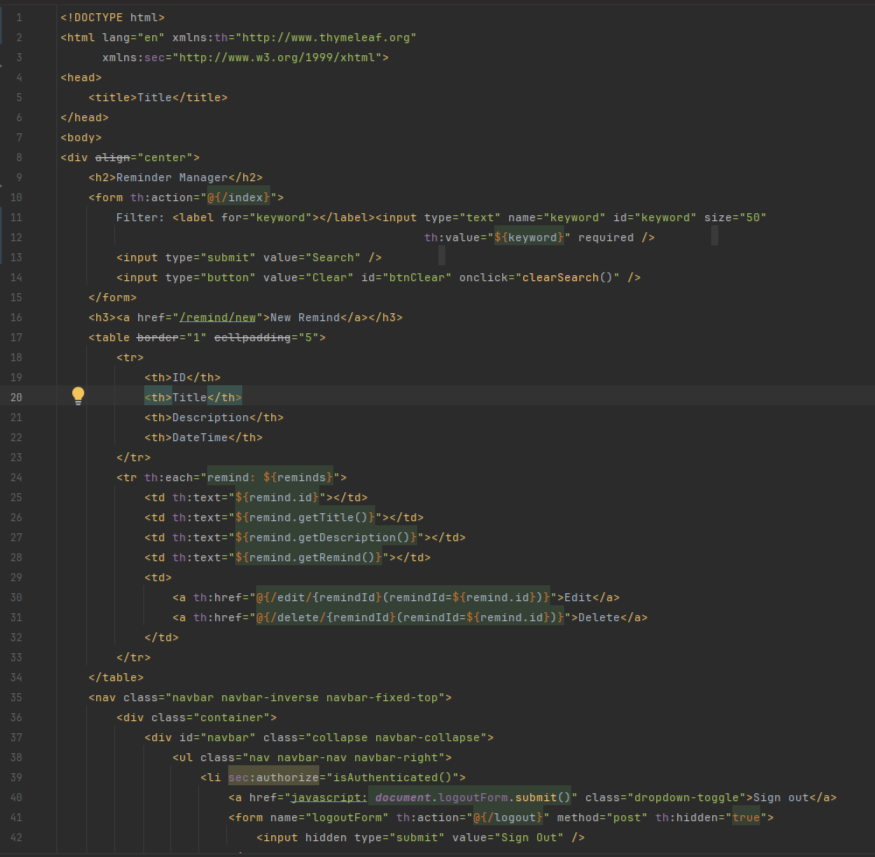
Интерфейс UserDetails является частью Spring Security и предоставляет основную информацию о пользователе, такую как его имя, пароль, роли, разрешения и статус активности.

MyUserDetails при необходимости расширяет интерфейс UserDetails и добавляет свои дополнительные поля и методы для предоставления дополнительной информации о пользователе, которая может быть необходима в конкретном приложении.

****

**2.2. Проектирование WEB – интерфейса приложения**

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

<!DOCTYPE html>: определяет тип документа и его версию (HTML5).

<html lang="en" xmlns: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>).

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

Ссылки <a>: используются для редактирования и удаления записей. Ссылки генерируются с помощью Thymeleaf, где <a th:href="@{/edit/{remindId}(remindId=${remind.id})}">Edit</a> создает ссылку для редактирования записи с определенным идентификатором, а <a th:href="@{/delete/{remindId}(remindId=${remind.id})}">Delete</a> - для удаления.

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.Начало формы

На этой странице предлагается выбор соответствующих действий:

- сделать новую запись;

- изменить запись;

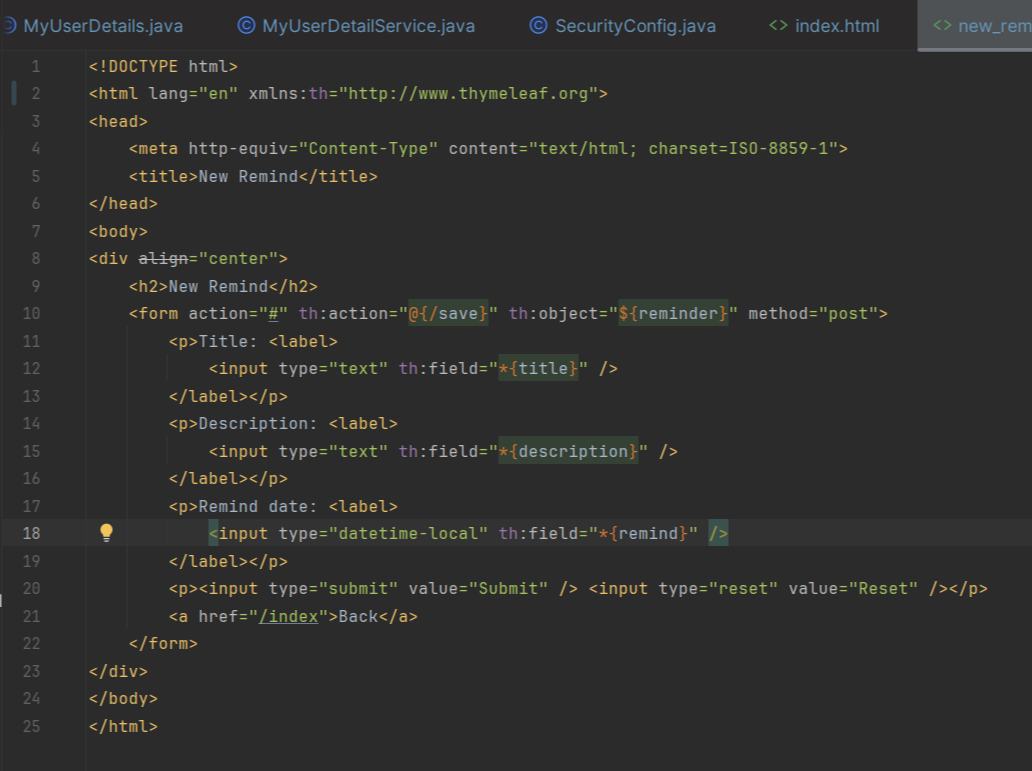
- удалить запись;

- поиск записи по любому словосочетанию или дате:

- выйти.

При нажатии кнопки «New Remind» открывается страница создания новой записи с полями: заголовок, описание, дата напоминания, которые нужно заполнить и сохранить.

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



Здесь в action внедрён метод контроллера /save, в object - внедряется экземпляр класса Reminder, method – post. Вводимые данные на html странице вставляются в соответствующие поля экземпляра класса Reminder

**2.3. Проектирование базы данных**

В качестве базы данных для этого проекта был выбран PostgreSQL.

PostgreSQL — это мощная и популярная система управления базами данных, которая предлагает ряд преимуществ и функций, поэтому многие люди выбирают его для своих проектов. Вот некоторые из основных причин, почему PostgreSQL часто рекомендуется:

1. Бесплатно и с открытым исходным кодом: PostgreSQL распространяется по лицензии с открытым исходным кодом, что означает, что вы можете использовать его бесплатно и изменять исходный код по своему усмотрению. Это делает его доступным для широкой аудитории пользователей и разработчиков.

2. Производительность и масштабируемость: PostgreSQL известен своей высокой производительностью и способностью масштабироваться для обработки больших объемов данных. Он предлагает эффективные индексы, оптимизацию запросов и поддержку параллельной обработки для улучшения производительности вашей базы данных.

3. Расширяемость: PostgreSQL имеет модульную архитектуру, которая позволяет разработчикам создавать и устанавливать свои собственные расширения и функции для удовлетворения уникальных потребностей и требований проектов. Это позволяет создавать более гибкие и мощные базы данных.

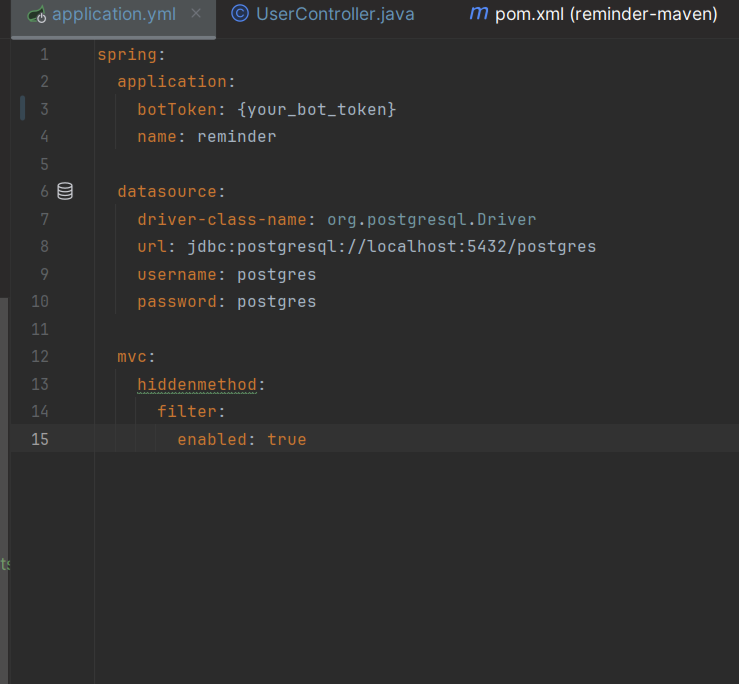
4. Большой набор функций: PostgreSQL предлагает широкий набор функций и возможностей, включая поддержку различных типов данных (таких как географические данные), триггеры, полнотекстовый поиск, репликацию, транзакции, хранимые процедуры и другие.

5. Надежность и устойчивость: PostgreSQL известен своей надежностью и устойчивостью. Он предлагает механизмы восстановления после сбоев, а также поддержку резервного копирования и восстановления данных, чтобы обеспечить сохранность вашей базы данных.

6. Активное сообщество и экосистема разработчиков: PostgreSQL имеет активное сообщество пользователей и разработчиков, что означает, что вы можете получить поддержку и помощь от опытных специалистов. Существует также множество инструментов и библиотек, разработанных в рамках экосистемы PostgreSQL, которые могут существенно упростить разработку и управление базами данных.

В целом, PostgreSQL представляет собой мощную и универсальную систему управления базами данных, которая может быть применена в широком спектре сценариев и проектов. Он сочетает в себе производительность, надежность, гибкость и масштабируемость, делая его хорошим выбором для многих приложений.

В файле application.yml указаны следующие настройки.

****

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

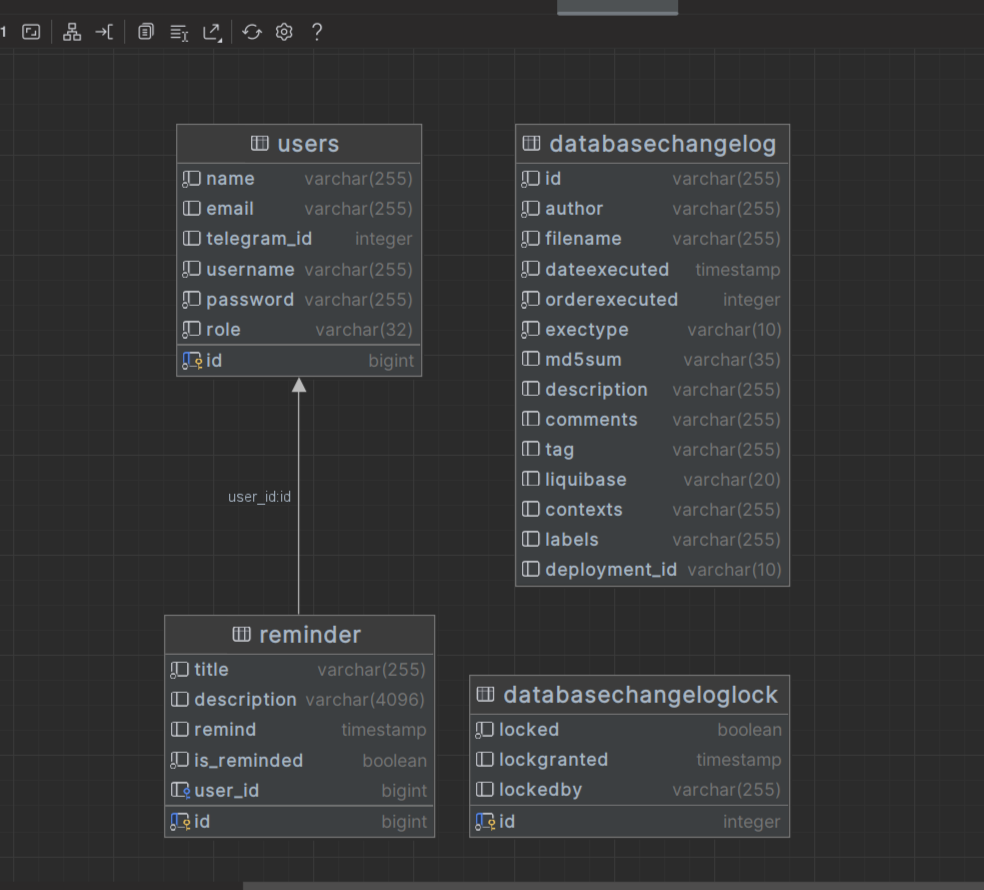
- datasource: содержит настройки для подключения к базе данных.

- 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.

В общем, схема взаимодействия таблиц сделана таким образом:



В базе данных присутствуют четыре таблицы.

Две из них – это users и reminder. Владельцем связи здесь является reminder, так как в этой сущности поставлена аннотация отношения. В данном случае – это @ManyToOne.

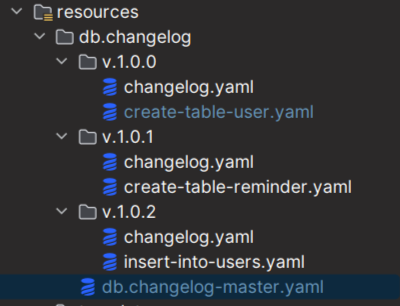
Связь между таблицами осуществляется с помощью поля user\_id, которое является foreign key поля id таблицы users.

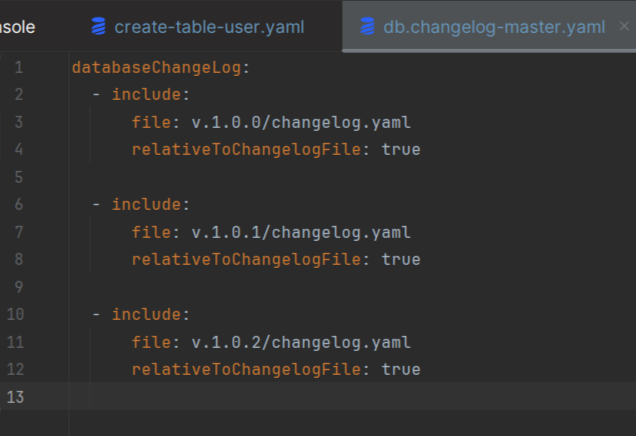
Две других таблицы databasechangelog и databasechangeloglock относятся к Liquibase.

Liquibase представляет из себя систему управления версиями базы данных, в основном это касается структуры и в меньшей степени содержимого базы. При этом описание базы с одной стороны достаточно абстрактно и позволяет использовать на нижнем уровне различные СУБД, и с другой стороны всегда можно перейти на SQL-диалект конкретной СУБД, что достаточно гибко. Liquibase является устоявшимся проектом с открытым исходным кодом и активно используется за пределами своей родной Java среды и не требует глубоких знаний Java для работы. В качестве описания структуры базы и изменений базы исторически использовался XML формат, однако сейчас параллельно поддерживается YAML и JSON. В моем проекте я использую более современный yaml формат.

Изменения структуры базы данных записываются в файлы, которые называются changeLog. Эти файлы могут быть описаны в разных форматах: XML, YAML, JSON или SQL.

Данные хранятся в папке resources/db.changelog. Структура такова: в корне resources/db.changelog находится файл db.changelog-master.yaml.



Как только версия приложения меняется, мы создаем новую папку, например, v.1.0.0 и новый changeLog создаем уже в ней. И все созданные changelog.xml мы включаем в db.changelog-master.xml, путём создания ссылки в основном changelog на локальный changelog.

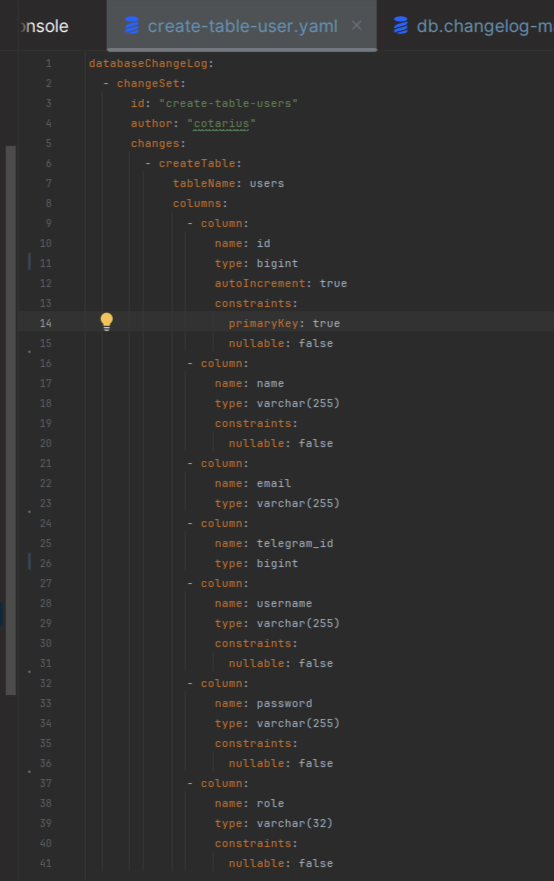
Атрибут relativeToChangelogFile="true" указывает путь относительно той папки, в которой располагается текущий changeLog.

Создание таблицы.

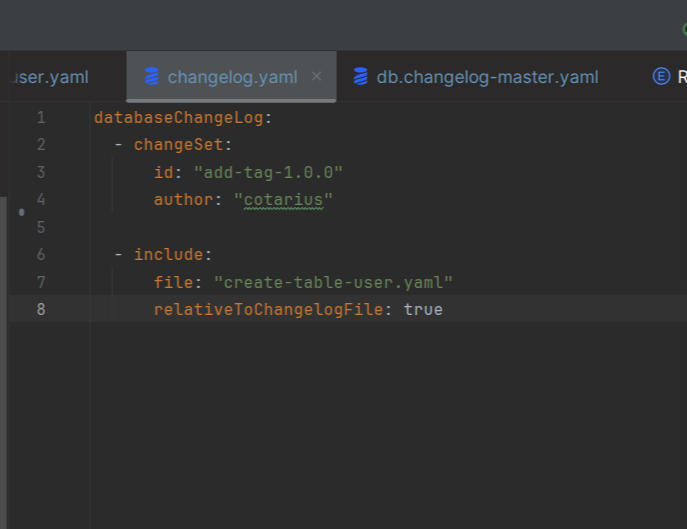
Создадим таблицу users. Для этого в папке v.1.0.0 создается новый файл create-table-user.yaml, который будет содержать следующую миграцию:

Строка changeSet имеет следующие подстроки:

* + Id – уникальный идентификатор текущего changeSet;
  + Author – имя автора changeset
  + Changes – здесь указывается сама логика изменения. В данном случае создается таблица users с полями id (primary key, autoincrement, bigint, nullable - false. При использовании primaryKey параметр nullable не обязателен. Но если вы используете H2 для тестов, то у вас могут возникнуть проблемы из-за его отсутствия), name (varchar), email (varchar), username (varchar), password (varchar), role (varchar).



Теперь в локальном файле changelog.xml необходимо указать тег include, ссылающийся на этот changeLog файл:

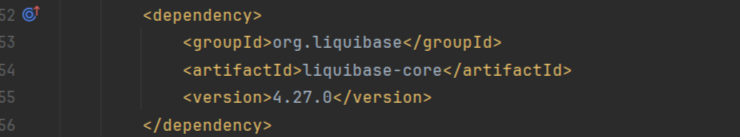


Запуск Luquibase

Liquibase можно запускать несколькими способами, все они используются в зависимости от контекста:

* + Запуск при помощи исполняемого файла
  + Запуск при помощи Docker образа
  + Запуск при старте SpringBoot или Quarkus сервиса

В моём приложении «Reminder» внедрение Liquibase в StringBoot. Для этого в pom.xml добавляется следующая зависимость:



Теперь достаточно запустить SpringBoot приложение. В процессе запуска в лог будут выведены сообщения от Liquibase и таблица users будет создана. Также будут созданы две технические таблицы: databasechangelog и databasechangeloglock. После этого, если внести изменения в таблицу users, например, добавив новую колонку просто изменив старый changeset, мы получим ошибку. То есть, если changeSet уже выполнился, и запись об этом есть в databasechangelog, то нельзя просто изменить changeSet. Это точно также, как нельзя в git изменить уже опубликованный коммит.

В этом случае у есть три пути:

- Создать новый changeSet с изменениями. [Рекомендуемый]

- Выполнить откат средствами Liquibase.

- Удалить запись о выполнении changeSet из databasechangelog. Не рекомендуют этот вариант, если changeSet уже был выполнен на каком-то стенде. Этот вариант удобен при локальной разработке.

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

Для обеспечения безопасности приложения был использован Spring Security.

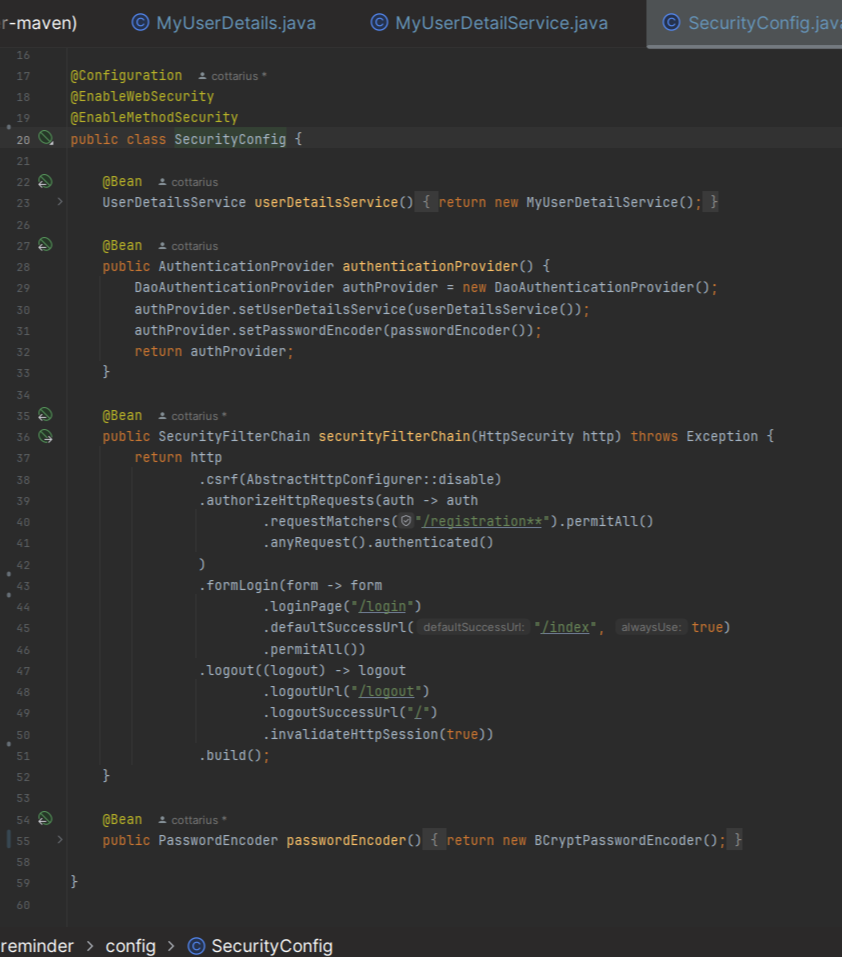
Spring Security — это мощный и гибкий фреймворк аутентификации и авторизации для приложений на платформе Java. Он обеспечивает обширный набор функций для защиты ваших приложений, включая аутентификацию пользователей, управление их правами доступа, защиту от атак, таких как CSRF и сессионные атаки, а 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 SecurityConfig – конфигурационный класс SpringSecurity.



1. **@Configuration** и **@EnableWebSecurity**: Эти аннотации указывают, что этот класс является конфигурационным классом Spring и активирует поддержку веб-безопасности.
2. **SecurityFilterChain**. В свою очередь, в SecurityFilterChain прописаны фильтры, которые разрешают доступ всем пользователям (авторизованным и не авторизованным) к странице регистрации нового пользователя. Ко всем остальным страницам доступ есть только у авторизованных пользователей.
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 **/index**.
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. **AuthenticationProvider**: Этот метод конфигурирует аутентификацию, указывая, что UserDetailsService используется для загрузки пользовательских данных и BCryptPasswordEncoder для проверки паролей.

Этот код предоставляет базовую конфигурацию безопасности для приложения Spring,

Также в ходе реализации безопасности был создан класс UserController.

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.

**2.5. Разворачивание приложения с помощью docker - compose**

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:

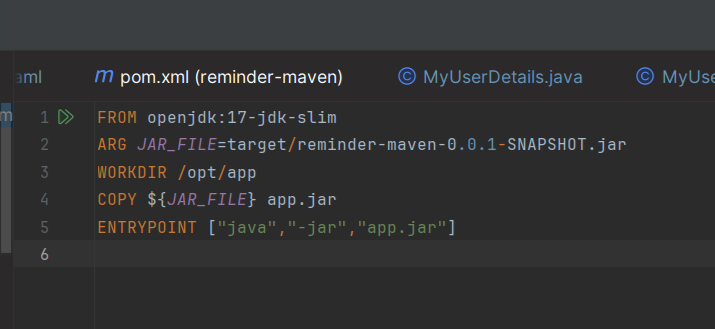
1. **Установка Docker**: сначала необходимо установить Docker на целевом хосте. Для этого можно воспользоваться официальным сайтом Docker, где доступны инструкции по установке для различных операционных систем.
2. **Создание Docker-образа и docker-compose.yml**: Docker-образ представляет собой шаблон, 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.

Но, сперва нужно упаковать приложение в jar файл с помощью команды в терминале, находясь в корневой папке проекта:

**mvn clean package -Dmaven.test.skip**

В результате в папке target появится jar файл проекта.

Теперь создаём файл с именем Dockerfile в корне проекта, который содержит инструкции для сборки образа со следующим текстом:



Вот что происходит, когда мы вводим этот код:

**FROM openjdk:17-jdk-slim** – образ создаётся на основе linux с установленной openjdk17-slim

**ARG JAR\_FILE=target/reminder-maven-0.0.1-SNAPSHOT.jar -** Переменной JAR\_FILE присваивается путь к jar- архиву

**WORKDIR /opt/app** - Назначаем рабочую директорию, в которой будут выполняться дальнейшие команды (перемещаемся в папку app)

**COPY ${JAR\_FILE} app.jar -** Наш jar-файл, указанный в JAR\_FILE, копируется в папку app, и копии задаётся имя app.jar

**ENTRYPOINT ["java","-jar","app.jar"] -** jar-файл запускается, собирается команда java -jar app.jar из заданной рабочей директории

После этого в терминале вводится команда, с помощью которой собирается образ:

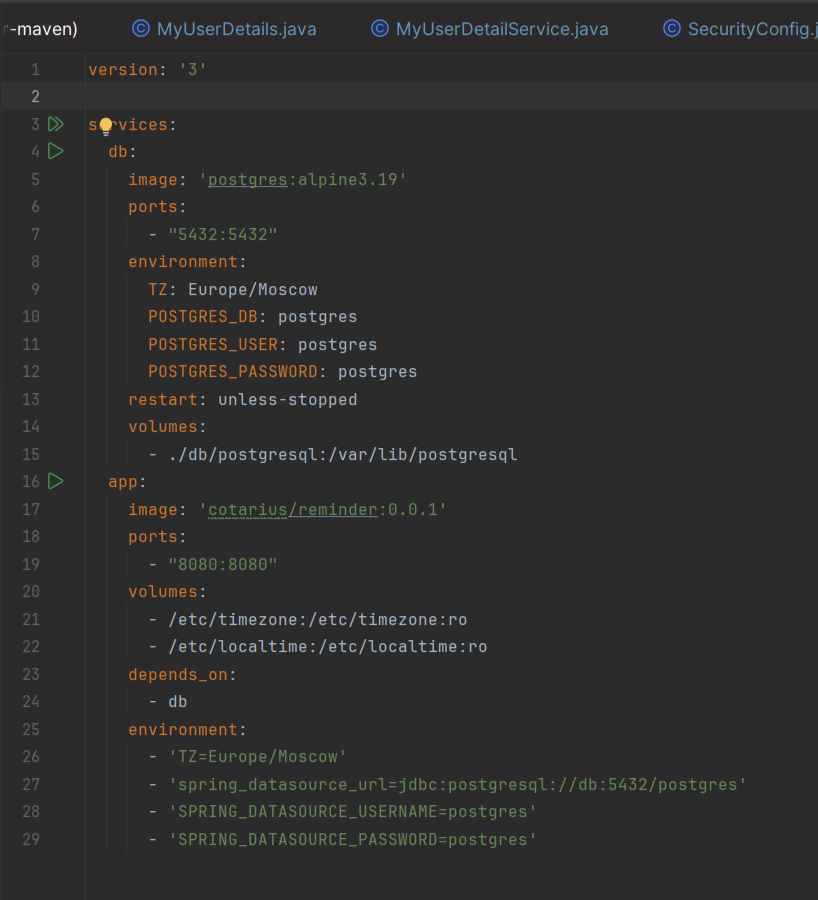
**Docker build -t cotarius/reminder:0.0.1 .**

Точка в конце важна, она указывает на расположение Dockerfile (символ «точка» означает текущую директорию)

Далее регистрируемся/авторизуемся на сайте hub.docker.com и загружаем наш образ в репозиторий командой

**docker push cotarius/reminder:0.0.1**

Теперь проект можно развернуть практически в любой среде (Windows, Linux, MacOs) если загрузить/создать файл docker-compose.yml со следующим содержимым:



Здесь находятся инструкции для создания двух контейнеров. Первый контейнер (db) – база данных PostgreSQL версия alpine:3.19. Второй контейнер (app) – приложение «Reminder».

**image**: сообщает Docker, какое имя он должен дать образу, когда используются функции сборки. В противном случае он ищет этот образ в библиотеке или удаленном реестре.

**ports** - здесь указываем первый порт - это внешний порт, с помощью которого мы можем получить доступ к нашему контейнеру, а второй порт - это внутренний порт контейнера.

**depends\_on: db** - указываем, что наш контейнер должен подняться после того, как поднимется контейнер с базой данных, так как, если контейнер с бэкендом поднимется первым и не будет доступа к базе данных, то он упадет с ошибкой.

**environment -** указываем дополнительные настройки, в том числе url для подключения к базе данных. В этом пути мы уже используем внутреннее имя нашего контейнера с базой данных db**.**

**restart: unless-stopped –** контейнер будет автоматически перезапускаться до тех пор, пока его не остановят «вручную».

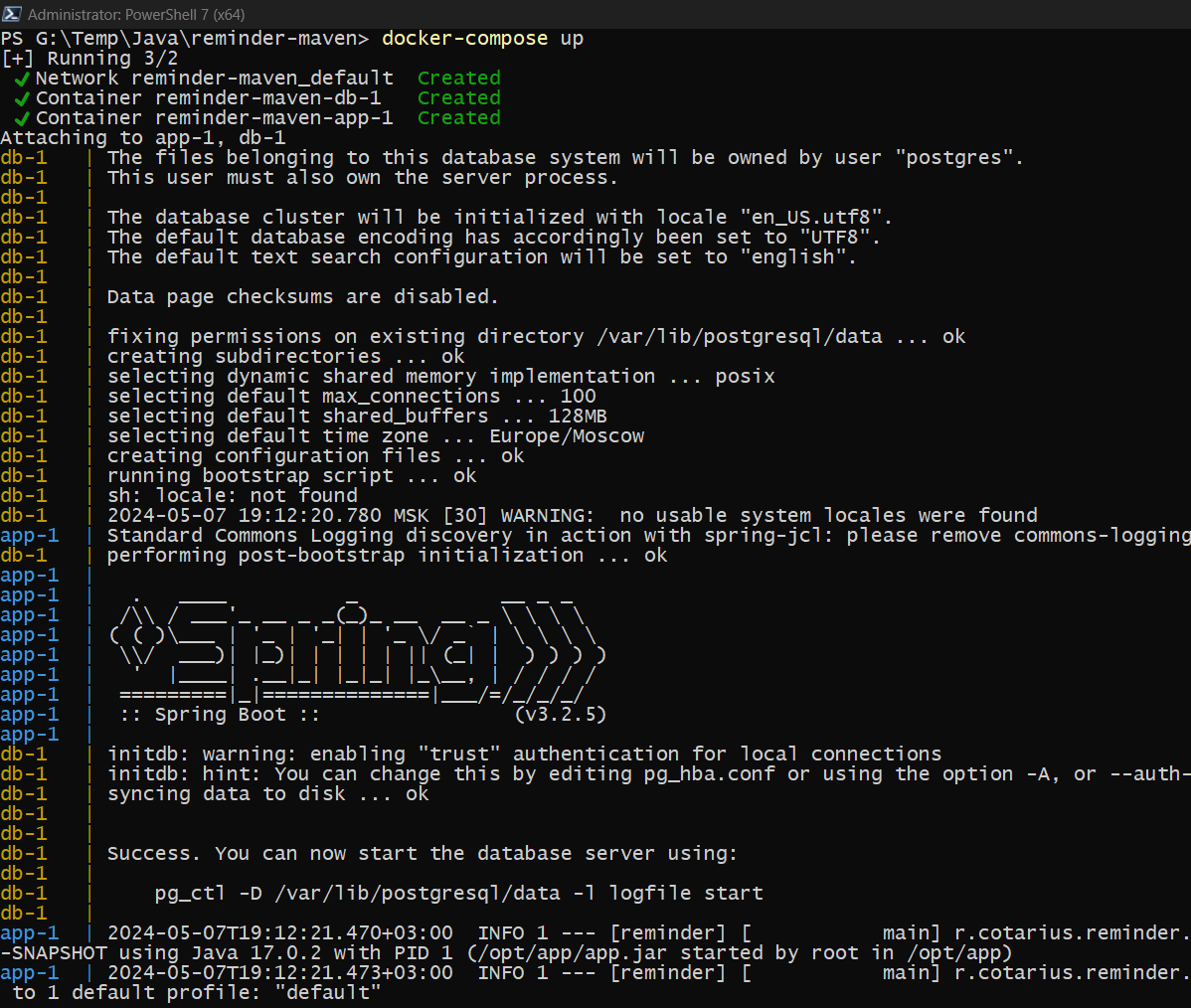
**volumes:** здесь указывается, чтобы данные сохранялись не локально в контейнер, а в директорию, расположенную вне контейнера. Это нужно для того, чтобы при перезапуске образа, если удалить и запустить снова, чтобы все изменения в базе данных не потерялись, а сохранились.

**environment: TZ: Europe/Moscow: -** задает необходимый TimeZone внутри контейнера, чтобы текущее время совпадало с временем внутри контейнера.

**3. Запуск приложения:**

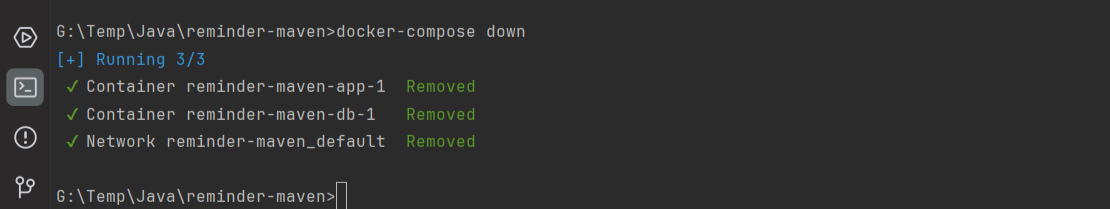
Для запуска приложения необходимо выполнить команду

**docker-compose up**

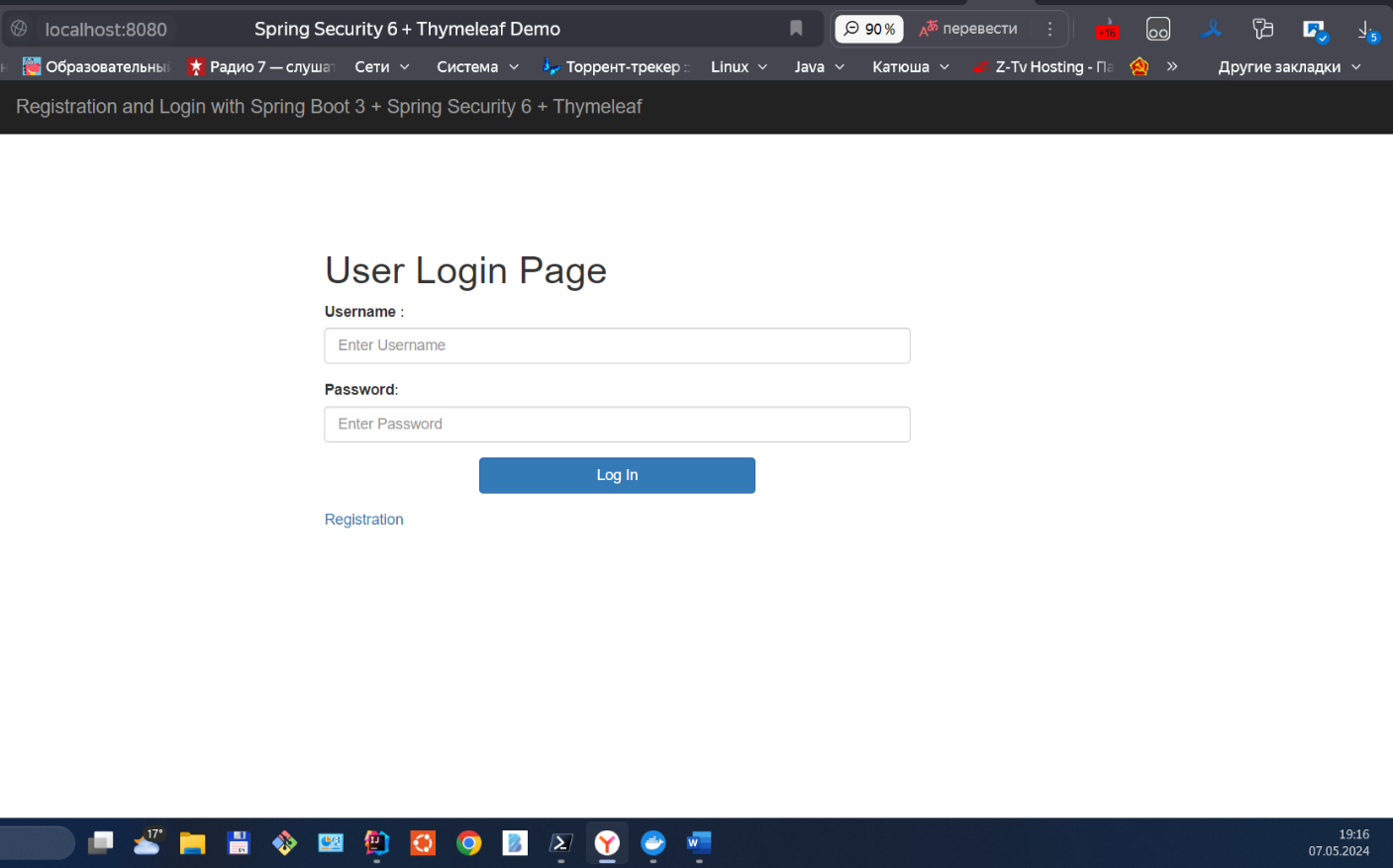
****

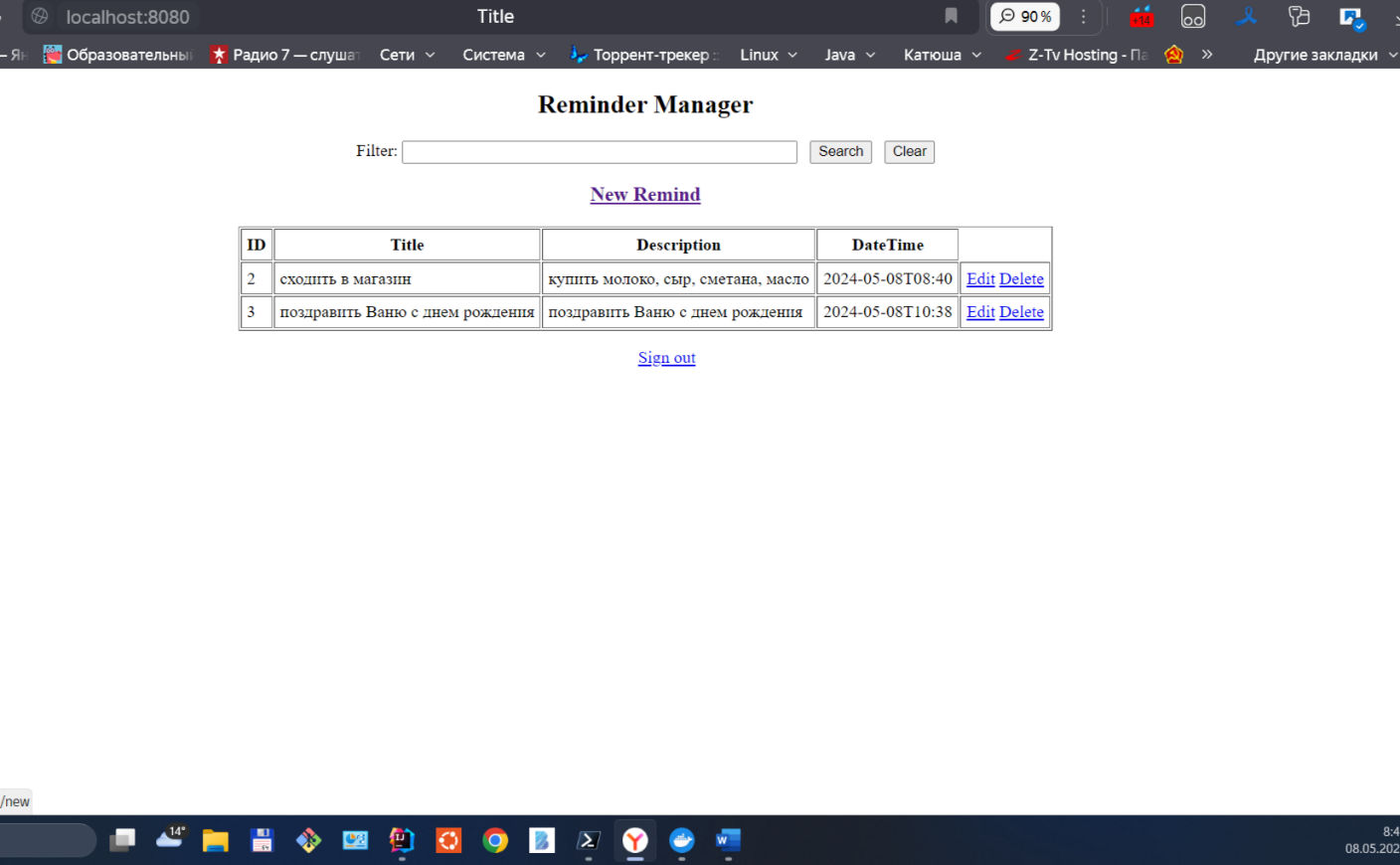
Для остановки приложения необходимо выполнить команду

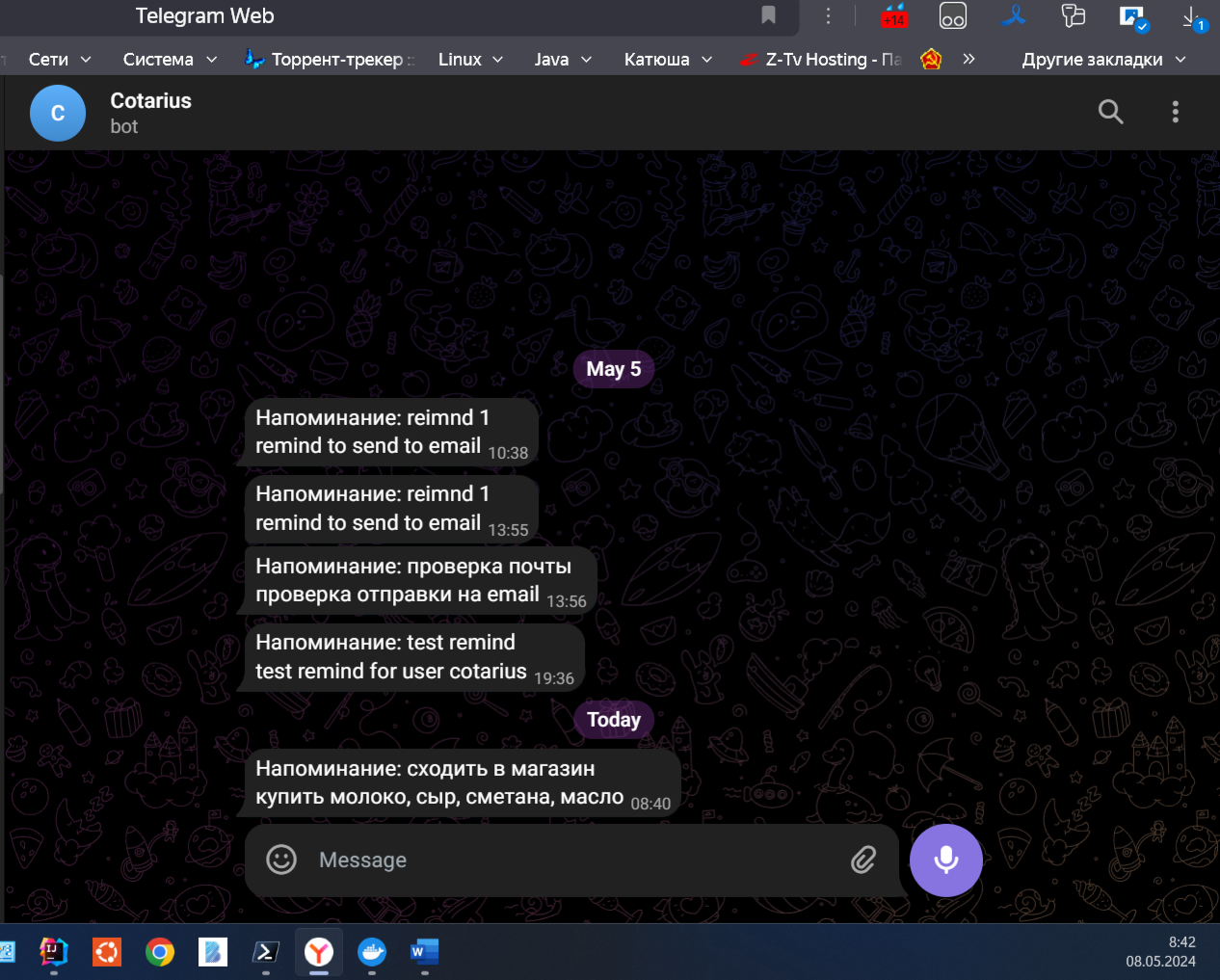
**docker-compose down**

****

Ну и напоследок пара скриншотов работающего приложения:







**Заключение**

В рамках данного дипломного проекта было разработано веб-приложение «Reminder», предназначенное для создания, редактирования, удаления напоминаний, а, также, отправки этих напоминаний пользователям в telegram и email. Проект был реализован с использованием современных технологий веб-разработки, таких как Java Spring Framework, Thymeleaf, Bootstrap, PostgreSQL, Liquibase.

В процессе разработки были достигнуты следующие цели:

* Создание удобного и интуитивно понятного интерфейса для пользователей приложения.
* Реализация функционала добавления, редактирования и удаления напоминаний и отправки их в telegram, email.
* 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 возможность подключения календаря для отображения напоминаний, а также мобильное приложение для удобного доступа к напоминаниям из любой точки.

В целом, дипломный проект "Reminder" 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 in Action" by Craig Walls - 2019**

**2. "Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools" by Iuliana Cosmina, Rob Harrop, Chris Schaefer, and Clarence Ho - 2017**

**3. "Spring Boot in Action" by Craig Walls - 2016**

**4. "Spring Data: Modern Data Access for Enterprise Java" by Mark Pollack, Oliver Gierke, Thomas Risberg, and Jon Brisbin - 2012**

**5. "Liquibase: Continuous Database Evolution for Databases" by Nathan Voxland - 2020**

**6. "PostgreSQL: Up and Running" by Regina Obe and Leo Hsu - 2017**

**7. "The Art of PostgreSQL" by Dimitri Fontaine - 2018**

**8. "PostgreSQL High Performance Cookbook" by Chitij Chauhan - 2017**

**9. "Mastering PostgreSQL 12" by Hans-Jürgen Schönig - 2019**

**10. "SQL Performance Explained" by Markus Winand – 2016**

**11. Учебный материал Geekbrains**

**12.** Статьи и видеоматериалы из сети интернет.