Geekbrains

Создание приложения для заметок с регистрацией пользователей с использованием фреймворка Spring и Spring Security

Программа: Java разработчик
Специализация: java backend разработчик
Васина Анна Вадимовна

Г. Петрозаводск
2025

Введение 2
Глава 1. Теоретическая часть 3
1.1 О языке Java 4
1.2 Что такое фреймворки 4
1.3 Фреймворк спринг 6
1.4 Что такое Maven и зачем он нужен 8
1.5 Архитектура проекта 10
Глава 2. Практическая часть 11
2.1 Структура моего проекта 12
2.2 Диаграммы связей классов с классами 13
2.3 Как запустить программу 14
Заключение 15
Список литературы 16
Приложения 17

Введение

Мы живем в быстроменяющемся мире, когда запомнить все свои задачи и планы на день становится трудновыполнимой задачей; люди прибегают к разным способам фиксации своего расписания: использование блокнота, запись в сообщениях в социальной сети или использование специального приложения, заточенного под создание списка дел и расписания.

Перенос собственных планов из памяти мозга на физический носитель становится необходимостью для каждого, кто работает в мире, где планы меняются быстрее, чем их удается согласовать, и держать всю информацию исключительно в голове становится не только сложным делом, но и появляется риск что-нибудь упустить. В качестве дипломной работы я решила разработать собственное приложение, в котором пользователь сможет создавать заметки и составлять план действий.

Для выполнения этого проекта были поставлены следующие задачи:

1. Изучить фреймворк Spring
2. Изучить дополнительные инструменты, ускоряющие разработку программы, такие как: Spring Security, Lombok, Thymelef
3. Изучить, как работать со сборщиком проектов Maven

Глава 1. Теоретическая часть

1.1 О языке Java

Java — один из самых популярных языков программирования в мире. Он был создан Джеймсом Гослингом и Патриком Ноттоном, сотрудниками компании Sun Microsystems, при поддержке Билла Джоя, сооснователя Sun Microsystems.

Компания Sun официально представила язык Java на конференции SunWorld 23 мая 1995 года. Затем, в 2009 году, компания Oracle купила компанию Sun, что объясняет, почему сейчас язык принадлежит Oracle.

Java описывается как многоцелевой, строго типизированный, объектно-ориентированный язык программирования. По замыслу, Java обладает минимальным количеством зависимостей от конкретных реализаций.

Этот язык программирования позволяет создавать приложения на множестве устройств. Его применение очень широкое, включая разработку программного обеспечения для мобильных устройств, POS-терминалов, банкоматов, Интернета вещей (IoT), а также веб-страниц.

Ключевые особенности языка Java:

- объектно-ориентированный язык программирования
- динамический язык программирования
- надёжность и безопасность
- переносимость и независимость от платформы
- высокая производительность

Благодаря своим отличным характеристикам, Java стала популярным и полезным языком программирования. Компания Sun охарактеризовала его как компилируемый и интерпретируемый

1.2 Что такое фреймворки

Согласно определению из Википедии,

«Фреймворк — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.»

Фреймворк отличается от библиотеки тем, что последняя может быть использована в программном продукте просто как набор подпрограмм похожей функциональности, не влияя на архитектуру программного продукта и не накладывая на неё никаких ограничений. В то время как фреймворк диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию — «каркас», который нужно будет расширять и изменять согласно указанным требованиям

Иными словами и вкратце, можно дать следующее определение фреймворку:

«Фреймворк — готовая модель в IT, заготовка, шаблон для программной платформы, на основе которого можно дописать собственный код.»

У использования фреймворков есть ряд плюсов.

Во-первых, с точки зрения бизнеса разработка на фреймворке экономически эффективнее и качественнее по результату, нежели написание проекта без использования каких-либо платформ.

Во-вторых, приложения на фреймворках значительно проще сопровождать и дорабатывать. Дело в том, что их стандартизированная структура понятна всем разработчикам на этой платформе.

Фреймворки эффективны в проектах со сложной бизнес-логикой и высокими требованиями к скорости работы, надёжности и безопасности. В простых, типовых проектах лучше использовать другие решения.

Существенный минус фреймворков — открытый код большинства из них. Это значит, что нужно отдельно решать вопросы безопасности проекта.

Фреймворки дают стабильность и удобство разработки, но ограничивают программистов своей архитектурой.

Большинство фреймворков основано на шаблоне проектирования MVC (Model-View-Controller или модель-представление-контроллер).

Модель (Model) — это место, где хранятся данные приложения.

Представление или отображение (View) определяет взаимодействие с пользователем через модель. Он отвечает за отображение данных, определяет внешний вид проекта — веб-сайтов или приложений. От него зависят возможные действия пользователя: где печатать текст, какие кнопки нажимать.

Контроллер (Controller) отвечает за реакцию двух предыдущих объектов на действия пользователя. Его код и логика проверяют, как сайт обрабатывает запросы и выдает ли правильный результат.

1.3 Фреймворк спринг

Spring Framework (или коротко **Spring**) — универсальный фреймворк с открытым исходным кодом для Java-платформы

Spring обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring не всецело связан с платформой Java Enterprise, несмотря на свою масштабную интеграцию с ней, что является важной причиной его популярности.

Spring может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании

Дополнительные библиотеки, которые были использованы в проекте:

- Lombok

  Проект Lombok позволяет сильно упростить Java-код, генерируя методы установки значений, генерации значений, hashCode, equals и ещё много чего.

  Добавляется в проект с помощью добавления зависимости в pom файл:

  ```
  <dependency>
          <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    </dependency>
  ```

- Spring Security

  Spring Security это Java/Java EE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для промышленных приложений, созданных с помощью Spring Framework

  Добавляется в проект с помощью добавления зависимости в pom файл:

  ```
  <dependency>
  ```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

- Thymeleaf — это современный серверный шаблонизатор Java для веб— и автономных сред.

  Основная цель Thymeleaf — привнести элегантные естественные шаблоны в рабочий процесс разработки с помощью HTML, которые будут корректно отображаться в браузерах, а также работать как статические прототипы, обеспечивая упрощенное сотрудничество при командной работе.

  Благодаря модулям для фреймворка Spring, множеству интеграций с другими инструментами и возможности подключать собственные функции, Thymeleaf идеально подходит для современной веб—разработки на HTML5 вместе с джава.

  Добавляется в проект с помощью добавления зависимости в pom файл:

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring—boot—starter—thymeleaf</artifactId>
    </dependency>
```

- H2 database

  H2 — открытая кроссплатформенная СУБД, полностью написанная на языке Java.

  Добавляется в проект с помощью добавления зависимости в pom файл:

```
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>2.3.232</version>
    </dependency>
```

1.4 Что такое Maven и зачем он нужен

Maven — это мощный инструмент, который использует структуру проектов на основе XML. Он предоставляет широкий набор плагинов и позволяет разработчикам создавать сложные проекты с множеством зависимостей и настроек. Один из его основных принципов— «соглашение против конфигурации», что означает, что Maven предполагает наличие определенной структуры и настроек в вашем проекте, что упрощает работу с ним.

Помимо компиляции исходного кода, системы сборки выполняют множество дополнительных функций, таких как:

1. Управление зависимостями: Одна из самых важных функций систем сборки — это управление зависимостями между библиотеками и модулями вашего проекта. Maven позволяет автоматически загружать и интегрировать библиотеки из удаленных репозиториев, таких как Maven Central или JCenter.

2. Тестирование: Системы сборки интегрируются с различными фреймворками тестирования, такими как JUnit или TestNG, и автоматически запускают тесты во время сборки проекта. Это обеспечивает постоянное контролирование качества вашего кода.

3. Генерация документации: Maven может автоматически генерировать документацию для кода, используя такие инструменты, как Javadoc. Это значительно облегчает поддержку и развитие проекта на долгосрочной основе.

4. Пакетирование и дистрибуция: Системы сборки создают исполняемые файлы и архивы (например, JAR, WAR или EAR) для развертывания и распространения приложения. Это делает процесс дистрибуции быстрым и непринужденным.

5. Поддержка плагинов: Maven имеет мощную систему плагинов, которая позволяет расширять функциональность системы сборки путем добавления дополнительных задач и интеграции с другими инструментами, такими как системы контроля версий или инструменты для непрерывной интеграции.

- Структура проекта: XML—структура (файл pom.xml)
- Читабельность: менее читабельный из—за XML
- Скорость: стабильно, но медленно
- Гибкость: есть ограничения

Основные понятия Maven:

1. POM (Project Object Model):

POM — это XML—файл, который описывает проект и его настройки. Он является сердцем проекта Maven и включает информацию о зависимостях, плагинах, версиях и других параметрах.

2. Зависимости:

Зависимости — это внешние библиотеки или модули, необходимые для корректной работы вашего приложения. Maven автоматически управляет зависимостями, загружая их из удаленных репозиториев и интегрируя в проект.

3. Плагины:

Плагины — это компоненты, которые расширяют функциональность Maven, добавляя дополнительные задачи и интеграцию с другими инструментами.

4. Жизненный цикл:

Жизненный цикл — это последовательность фаз, определяющих процесс сборки проекта. Основные циклы жизни в Maven включают clean (очистка сборки), default (основная сборка) и site (генерация сайта проекта).

1.5 Архитектура проекта

Архитектура IT проекта — это процесс планирования структуры и взаимодействия компонентов программного обеспечения для достижения целей. Она включает в себя выбор технологий, определение требований, создание диаграмм и планирование тестирования.

Рассмотрим типичную архитектуру java-проекта:

src/main/java: Здесь хранится исходный код приложения.

src/main/resources: Здесь находятся ресурсы приложения, такие как файлы конфигурации, изображения ит. д.

src/test/java: Здесь размещается исходный код тестов приложения.

src/test/resources: Здесь хранятся ресурсы, связанные с тестированием.

pom.xml: Это основной файл конфигурации Maven, содержащий информацию о проекте и его настройках.

Рисунок 1. Структура проекта Maven

Глава 2. Практическая часть

2.1 Структура моего проекта

Структура моего проекта состоит из стандартных src/main/java, src/main/resources, в папке приложения com.example.NoteApp содержатся папки config - для настройки конфигураций, controller - для связи между действиями пользователя и ответом от сервера, подпапки notecontroller и usercontroller для лучшего разграничения классов, repository - для хранения файлов интерфейсов, service - для загрузки специфичных данных для определенных сущностей.

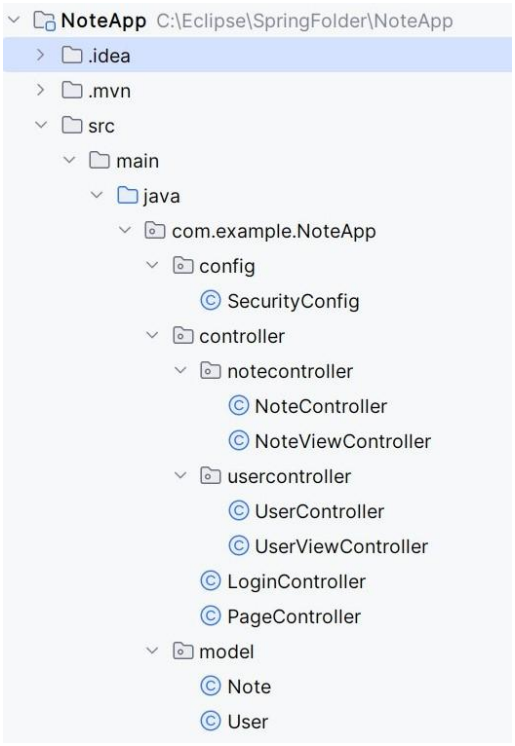

Рисунок 2. Скриншот структуры проекта, часть 1

Рисунок 3. Скриншот структуры проекта, часть 2

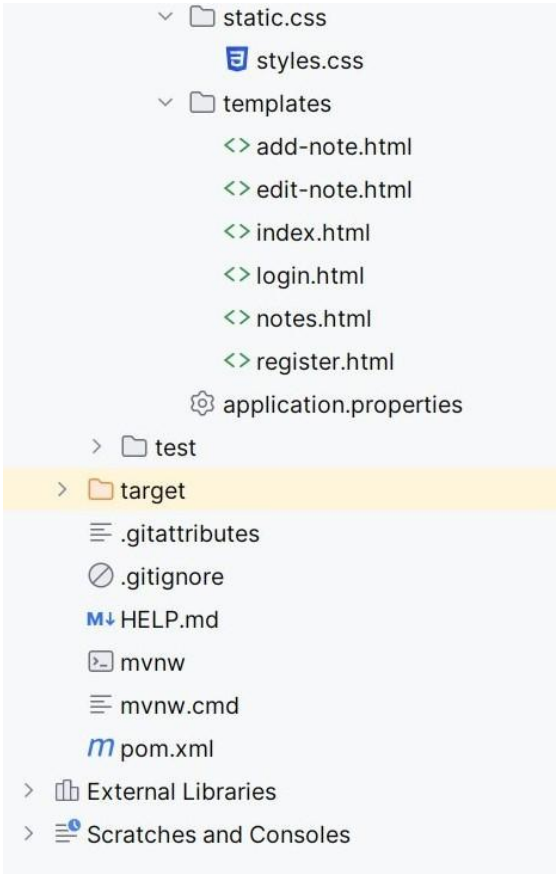

Рисунок 4. Скриншот структуры проекта, часть 3

2.2 Диаграммы связей классов с классами

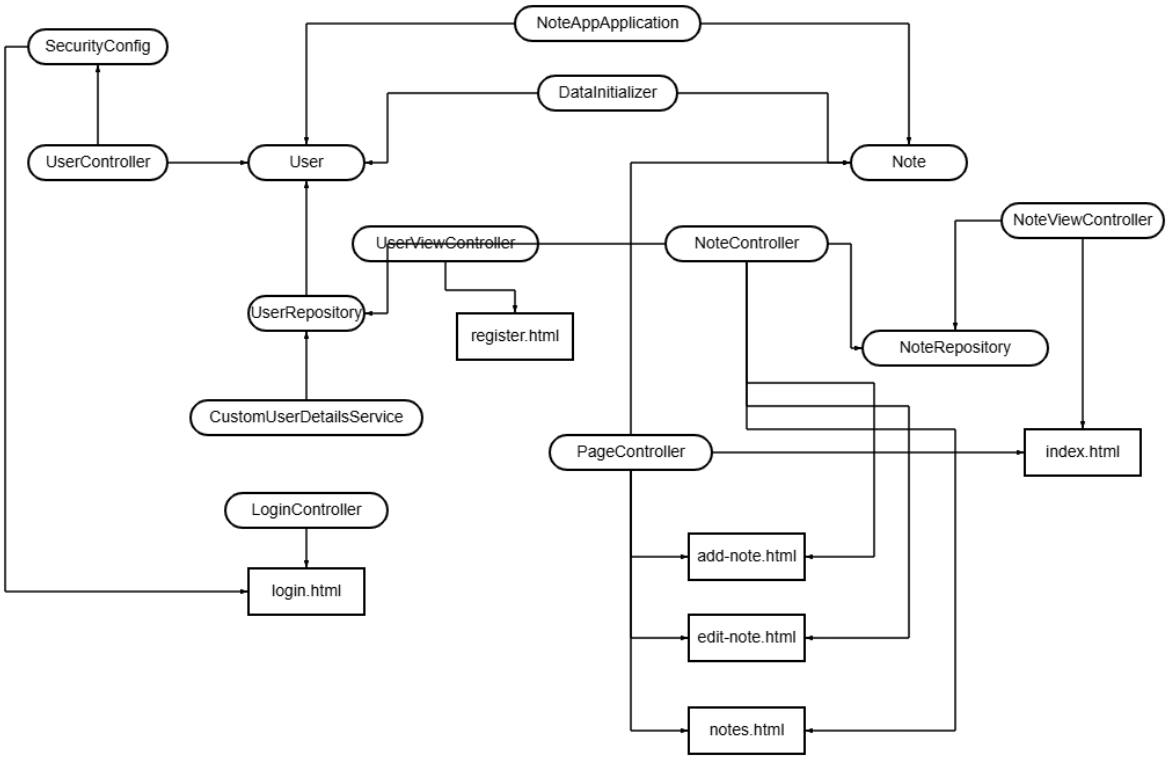

Рисунок 5. Связи между классами

2.3 Как запустить программу

Программа находится в репозитории на GitHub по ссылке https://github.com/KittyMew0/final_final_diplom_project

После запуска программы на компьютере, будет предложено зарегистрировать нового пользователя, или воспользоваться предсозданным пользователем (username: testuser, password: password) с тестовой заметкой для изучения функционала.

Навигация внутри приложения может осуществляться с помощью указания путей:

- Главная страница и список заметок: / ;
- Добавление заметки: /notes/add ;
- Редактирование заметки: /notes/edit/{id}* ;
- Удаление заметки: /notes/delete/{id}* ;

*{id} - указание идентификатора заметки

Или с помощью кнопок, добавленных при использовании html и css.

Заключение

В процессе работы над проектом были изучены разные библиотеки и фреймворки, упрощающие разработку сервисных приложений, среди которых популярный фреймворк Spring и сборщик проектов Maven, задачи, поставленные в начале работы, были выполнены.

В теоретической части дипломной работы представлены ключевые определения и характеристики используемых библиотек, в практической части представлены структура проекта и взаимосвязи между классами внутри проекта вместе с демонстрацией работы программы.

В качестве итогового проекта было создано веб-приложение для создания заметок, которое в дальнейшем может быть выпущено в публичный доступ.

Список литературы

Spring framework // springframework URL:

https://springframework.net/

Spring framework // Wikipedia URL:

https://en.wikipedia.org/wiki/Software_framework

H2 Database Engine // h2database URL:

https://www.h2database.com/html/main.html

Spring Security // Spring.io URL:

https://spring.io/projects/spring-security

Lombok // projectlombok URL:

https://projectlombok.org/

История Java // Baeldung URL:

https://www.baeldung.com/java-history

Приложения

Приложение 1, класс NoteAppApplication

```
package com.example.NoteApp;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class NoteAppApplication {

	public static void main(String[] args) {
		SpringApplication.run(NoteAppApplication.class, args);
	}
}
```

Приложение 2, класс SecurityConfig

```
package com.example.NoteApp.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

/**
 * Класс конфигурации для настройки безопасности
 */
@Configuration
public class SecurityConfig {

    /**
     * Аннотация Bean для кодирования пароля
     *
     * Возвращение пароля с @return
     */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

    /**
     * Настраивает цепочку фильтров безопасности.
     *
```

```
     * @param http объект HttpSecurity для настройки
     * @return настроенная цепочка SecurityFilterChain
     * @throws Exception если во время настройки произошла ошибка
     */
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
                .csrf().disable()
                .authorizeHttpRequests()
                .requestMatchers("/", "/register", "/css/**", "/js/**", "/h2-
console/**").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
                .logout()
                .permitAll();
        return http.build();
    }
}
```

Приложение 3, класс NoteController

```
package com.example.NoteApp.controller.notecontroller;

import com.example.NoteApp.model.Note;
import com.example.NoteApp.model.User;
import com.example.NoteApp.repository.NoteRepository;
import com.example.NoteApp.repository.UserRepository;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
 * Контроллер для управления Заметками
 */

@Controller
@RequestMapping("/notes")
public class NoteController {

    private final NoteRepository noteRepository;
    private final UserRepository userRepository;

        /**
     * Конструктор для NoteController.
     *
     * @param noteRepository репозиторий для заметок
```

```
     * @param userRepository репозиторий для пользователей
     */
    public NoteController(NoteRepository noteRepository, UserRepository
userRepository) {
        this.noteRepository = noteRepository;
        this.userRepository = userRepository;
    }

    @GetMapping
    public String listNotes(org.springframework.ui.Model model) {
        List<Note> notes = noteRepository.findAll();
        model.addAttribute("notes", notes);
        return "notes";
    }

    /**
     * Добавляет новую заметку.
     *
     * @param note добавляемая заметка
     * @return сохраненная заметка
     */
    @GetMapping("/notes/add")
    public String showAddNoteForm(org.springframework.ui.Model model) {
        model.addAttribute("note", new Note());
        return "add-note";
    }

    @PostMapping("/notes")
    public String addNote(@ModelAttribute("note") Note note) {
        String username = ((UserDetails)
```

```
SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getUsername
();
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new RuntimeException("User not found"));

        note.setUser(user);
        noteRepository.save(note);
        return "redirect:/";
    }

    @GetMapping("/notes/edit/{id}")
    public String showEditNoteForm(@PathVariable Long id,
org.springframework.ui.Model model) {
        Note note = noteRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Note not found"));
        model.addAttribute("note", note);
        return "edit-note";
    }
        /**
     * Обновляет заметку по ее ID.
     *
     * @param id ID обновляемой заметки
     * @param updatedNote заметка с обновленной информацией
     * @return обновленная заметка
     */
        @PostMapping("/notes/update/{id}")
        public String updateNote(@PathVariable Long id, @ModelAttribute("note")
Note updatedNote) {
            Note note = noteRepository.findById(id)
                    .orElseThrow(() -> new RuntimeException("Note not found"));
```

```
            note.setTitle(updatedNote.getTitle());
            note.setContent(updatedNote.getContent());
            noteRepository.save(note);
            return "redirect:/";
        }

        /**
     * Удаляет заметку по ее ID.
     *
     * @param id ID удаляемой заметки
     * @return нет содержимого, если заметка удалена, не найдено, если
заметка не существует
     */
        @GetMapping("/notes/delete/{id}")
        public String deleteNote(@PathVariable Long id) {
            if (noteRepository.existsById(id)) {
                noteRepository.deleteById(id);
            }
            return "redirect:/";
        }
}
```

Приложение 4, класс NoteViewController

```
package com.example.NoteApp.controller.notecontroller;

import com.example.NoteApp.model.Note;
import com.example.NoteApp.repository.NoteRepository;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.List;

/**
 * Класс контроллера для обработки представлений заметок.
 */
@Controller
@RequestMapping("/")
public class NoteViewController {

    private final NoteRepository noteRepository;

    /**
     * Конструктор для NoteViewController.
     *
     * @param noteRepository репозиторий для заметок
     */
    public NoteViewController(NoteRepository noteRepository) {
        this.noteRepository = noteRepository;
    }
```

```
    /**
     * Отображает все заметки.
     *
     * @param model модель для добавления атрибутов
     * @return имя представления
     */
    @GetMapping
    public String viewNotes(Model model) {
        List<Note> notes = noteRepository.findAll();
        model.addAttribute("notes", notes);
        return "index";
    }
}
```

Приложение 5, класс UserController

```
package com.example.NoteApp.controller.usercontroller;

import com.example.NoteApp.model.User;
import com.example.NoteApp.repository.UserRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

/**
 * Класс контроллера для управления пользователями.
 */
@RestController
@RequestMapping("/users")
public class UserController {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    /**
     * Конструктор для UserController.
     *
     * @param userRepository  репозиторий для пользователей
     * @param passwordEncoder кодировщик паролей
     */
    public UserController(UserRepository userRepository,
                          PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
```

```
    }

    /**
     * Регистрирует нового пользователя.
     *
     * @param user регистрируемый пользователь
     * @return ответ сущности с результатом регистрации
     */
    @PostMapping("/register")
    public ResponseEntity<String> register(@RequestBody User user) {
        if (userRepository.findByUsername(user.getUsername()).isPresent()) {
            return ResponseEntity.badRequest().body("Username is already taken");
        }
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        return ResponseEntity.ok("User registered successfully");
    }
}
```

Приложение 6, класс UserViewController

```
package com.example.NoteApp.controller.usercontroller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

/**
 * Класс контроллера для обработки представлений пользователей.
 */
@Controller
public class UserViewController {

    /**
     * Отображает страницу регистрации.
     *
     * @return имя представления для страницы регистрации
     */
    @GetMapping("/register")
    public String showRegistrationPage() {
        return "register";
    }
}
```

Приложение 7, класс LoginController

```
package com.example.NoteApp.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

/**
 * Класс контроллера для обработки представлений входа.
 */
@Controller
public class LoginController {

    /**
     * Отображает страницу входа.
     *
     * @return имя представления для страницы входа
     */
    @GetMapping("/login")
    public String showLoginPage() {
        return "login";
    }
}
```

Приложение 8, класс Note

```
package com.example.NoteApp.model;

import jakarta.persistence.*;
import lombok.Data;
import lombok.Getter;
import lombok.Setter;

import java.time.LocalDateTime;

/**
 * Класс сущности, представляющий Заметку.
 */
@Data
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String content;

    @Column(nullable = false, updatable = false)
    private LocalDateTime createdAt;
```

```
@ManyToOne
@JoinColumn(name = "user_id", nullable = false)
private User user;

/**
 * Устанавливает метку времени создания перед сохранением сущности.
 */
@PrePersist
public void prePersist() {
    this.createdAt = LocalDateTime.now();
}

/**
 * Получает заголовок заметки.
 *
 * @return заголовок
 */
public String getTitle() {
    return title;
}

/**
 * Устанавливает заголовок заметки.
 *
 * @param title устанавливаемый заголовок
 */
public void setTitle(String title) {
    this.title = title;
}
```

```
/**
 * Получает содержимое заметки.
 *
 * @return содержимое
 */
public String getContent() {
    return content;
}

/**
 * Устанавливает содержимое заметки.
 *
 * @param content устанавливаемое содержимое
 */
public void setContent(String content) {
    this.content = content;
}

/**
 * Получает пользователя, связанного с заметкой.
 *
 * @return пользователь
 */
public User getUser() {
    return user;
}

/**
 * Устанавливает пользователя, связанного с заметкой.
```

```
     *
     * @param user устанавливаемый пользователь
     */
    public void setUser(User user) {
        this.user = user;
    }

}
```

Приложение 9, класс User

```
package com.example.NoteApp.model;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

/**
 * Класс сущности, представляющий Пользователя.
 */
@Data
@NoArgsConstructor
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval
```

```
= true)
    private List<Note> notes;

    /**
     * Получает имя пользователя.
     *
     * @return имя пользователя
     */
    public String getUsername() {
        return username;
    }

    /**
     * Устанавливает имя пользователя.
     *
     * @param username устанавливаемое имя пользователя
     */
    public void setUsername(String username) {
        this.username = username;
    }

    /**
     * Получает пароль пользователя.
     *
     * @return пароль
     */
    public String getPassword() {
        return password;
    }
```

```
    /**
     * Устанавливает пароль пользователя.
     *
     * @param password устанавливаемый пароль
     */
    public void setPassword(String password) {
        this.password = password;
    }
}
```

Приложение 10, interface NoteRepository

```
package com.example.NoteApp.repository;

import com.example.NoteApp.model.Note;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

/**
 * Интерфейс репозитория для сущностей Заметки.
 */
public interface NoteRepository extends JpaRepository<Note, Long> {

    /**
     * Находит заметку по ее ID.
     *
     * @param id ID заметки
     * @return Optional, содержащий заметку, если она найдена, иначе пустой
     */
    Optional<Note> findById(Long id);
}
```

Приложение 11, interface UserRepository

```
package com.example.NoteApp.repository;

import com.example.NoteApp.model.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

/**
 * Интерфейс репозитория для сущностей Пользователь.
 */
public interface UserRepository extends JpaRepository<User, Long> {

    /**
     * Находит пользователя по его имени пользователя.
     *
     * @param username имя пользователя
     * @return Optional, содержащий пользователя, если он найден, иначе пустой
     */
    Optional<User> findByUsername(String username);
}
```

Приложение 12, класс CustomUserDetailsService

```
package com.example.NoteApp.service;

import com.example.NoteApp.model.User;
import com.example.NoteApp.repository.UserRepository;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.ArrayList;

/**
 * Сервисный класс для загрузки данных, специфичных для пользователя.
 */
@Service
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    /**
     * Конструктор для CustomUserDetailsService.
     *
     * @param userRepository репозиторий для пользователей
     */
    public CustomUserDetailsService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
```

```
    /**
     * Загружает данные, специфичные для пользователя.
     *
     * @param username имя пользователя
     * @return объект UserDetails, содержащий данные, специфичные для
пользователя
     * @throws UsernameNotFoundException если пользователь не найден
     */
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not found"));

        System.out.println("Found user: " + user.getUsername());
        System.out.println("Password: " + user.getPassword());

        return new org.springframework.security.core.userdetails.User(
                user.getUsername(),
                user.getPassword(),
                new ArrayList<>()
        );
    }
}
```

Приложение 13, класс DataInitializer

```
package com.example.NoteApp;

import com.example.NoteApp.model.Note;
import com.example.NoteApp.model.User;
import com.example.NoteApp.repository.NoteRepository;
import com.example.NoteApp.repository.UserRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

/**
 * Компонент для инициализации данных.
 */
@Component
public class DataInitializer {

    /**
     * Инициализирует пользователей в базе данных.
     *
     * @param userRepository репозиторий для пользователей
     * @param passwordEncoder кодировщик паролей
     * @return бин CommandLineRunner для инициализации пользователей
     */
    @Bean
    public CommandLineRunner initUsers(UserRepository userRepository,
                                       NoteRepository noteRepository,
                                       PasswordEncoder passwordEncoder) {
```

```
        return args -> {
            User user = new User();
            user.setUsername("testuser");
            user.setPassword("password");
            userRepository.save(user);

            Note note = new Note();
            note.setTitle("Test Note");
            note.setContent("This is a test note");
            note.setUser(user);
            noteRepository.save(note);
        };
    }
}
```

Приложение 14, файл pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.4.1</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.example</groupId>
	<artifactId>NoteApp</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>NoteApp</name>
	<description>Demo project for Spring Boot</description>
	<url/>
	<licenses>
		<license/>
	</licenses>
	<developers>
		<developer/>
	</developers>
	<scm>
		<connection/>
		<developerConnection/>
		<tag/>
```

```
        <url/>
</scm>
<properties>
        <java.version>17</java.version>
        <spring-security.version>6.0.0</spring-security.version>
</properties>
<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
                <groupId>com.h2database</groupId>
                <artifactId>h2</artifactId>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
                <version>1.18.24</version>
```

```
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

</dependencies>

<build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
            <plugin>
```

```
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Приложение 15, файл application.properties

```
spring.application.name=NoteApp

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.h2.console.path=/h2-console
```

Приложение 16, файл register.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register</title>
    <link rel="stylesheet" th:href="@{/css/styles.css}">
</head>
<body>
<h1>Register</h1>
<form action="/users/register" method="post">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br>
    <button type="submit">Register</button>
</form>
</body>
</html>
```

Приложение 17, класс PageController

```
package com.example.NoteApp.controller;

import com.example.NoteApp.model.Note;
import com.example.NoteApp.model.User;
import com.example.NoteApp.repository.NoteRepository;
import com.example.NoteApp.repository.UserRepository;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;
/**
 * Класс контроллера для обработки просмотра страниц и операций с заметками.
 */
@Controller
@RequestMapping("/index")
public class PageController {

    private final NoteRepository noteRepository;
    private final UserRepository userRepository;

    /**
     * Конструктор для PageController.
     *
     * @param noteRepository репозиторий для заметок
```

```
     * @param userRepository репозиторий для пользователей
     */
    public PageController(NoteRepository noteRepository, UserRepository
userRepository) {
        this.noteRepository = noteRepository;
        this.userRepository = userRepository;
    }

    /**
     * Отображает главную страницу индекса.
     *
     * @return имя представления для страницы индекса
     */
    @GetMapping("/")
    public String index() {
        return "index";
    }

    /**
     * Выводит список всех заметок.
     *
     * @param model модель для добавления атрибутов
     * @return имя представления для списка заметок
     */
    @GetMapping("/index")
    public String listNotes(Model model) {
        List<Note> notes = noteRepository.findAll();
        model.addAttribute("notes", notes);
        return "index";
    }
```

```
    /**
     * Отображает форму для добавления новой заметки.
     *
     * @param model модель для добавления атрибутов
     * @return имя представления для формы добавления заметки
     */
    @GetMapping("/notes/add")
    public String showAddNoteForm(Model model) {
        model.addAttribute("note", new Note());
        return "add-note";
    }

    /**
     * Добавляет новую заметку.
     *
     * @param note добавляемая заметка
     * @return URL перенаправления на список заметок
     */
    @PostMapping("/notes")
    public String addNote(@ModelAttribute("note") Note note) {
        String username = ((UserDetails)
SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getUsername
();
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new RuntimeException("User not found"));

        note.setUser(user);
        noteRepository.save(note);
        return "redirect:/notes";
```

```
    }

    /**
     * Отображает форму для редактирования существующей заметки.
     *
     * @param id ID редактируемой заметки
     * @param model модель для добавления атрибутов
     * @return имя представления для формы редактирования заметки
     */
    @GetMapping("/notes/edit/{id}")
    public String showEditNoteForm(@PathVariable Long id, Model model) {
        Note note = noteRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Note not found"));
        model.addAttribute("note", note);
        return "edit-note";
    }

    /**
     * Обновляет существующую заметку.
     *
     * @param id ID обновляемой заметки
     * @param updatedNote заметка с обновленной информацией
     * @return URL перенаправления на список заметок
     */
    @PostMapping("/notes/update/{id}")
    public String updateNote(@PathVariable Long id, @ModelAttribute("note") Note
updatedNote) {
        Note note = noteRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Note not found"));
```

```
        note.setTitle(updatedNote.getTitle());
        note.setContent(updatedNote.getContent());
        noteRepository.save(note);
        return "redirect:/notes";
    }

    /**
     * Удаляет заметку по ее ID.
     *
     * @param id ID удаляемой заметки
     * @return URL перенаправления на список заметок
     */
    @GetMapping("/delete/{id}")
    public String deleteNote(@PathVariable Long id) {
        if (noteRepository.existsById(id)) {
            noteRepository.deleteById(id);
        }
        return "redirect:/notes";
    }
}
```

Приложение 18, файл login.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <link rel="stylesheet" th:href="@{/css/styles.css}">
</head>
<body>
<h1>Login</h1>
<form action="/login" method="post">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br>
    <button type="submit">Login</button>
</form>
<p>Don't have an account? <a href="/register">Register here</a></p>
</body>
</html>
```

Приложение 19, файл index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Note App</title>
    <link rel="stylesheet" th:href="@{/css/styles.css}">
</head>
<body>
<h1>Welcome to NoteApp!</h1>
<a href="/notes/add">Add New Note</a>
<table border="1" cellspacing="0" cellpadding="10">
    <thead>
    <tr>
        <th>ID</th>
        <th>Title</th>
        <th>Content</th>
        <th>Actions</th>
    </tr>
    </thead>
    <tbody>
    <tr th:if="${#lists.isEmpty(notes)}">
        <td colspan="4">No notes available.</td>
    </tr>
    <tr th:each="note : ${notes}">
        <td th:text="${note.id}"></td>
        <td th:text="${note.title}"></td>
        <td th:text="${note.content}"></td>
        <td>
            <a th:href="@{/notes/edit/{id}(id=${note.id})}" style="color: blue; text-
```

```
decoration: none;">Edit</a>
            <a th:href="@{/notes/delete/{id}(id=${note.id})}" style="color: red; text-
decoration: none; margin-left: 10px;">Delete</a>
        </td>
    </tr>
    </tbody>
</table>
</body>
</html>
```

Приложение 20, файл add-note.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Note</title>
</head>
<body>
<h1>Add a New Note</h1>
<form action="/notes" method="post">
    <label for="title">Title:</label>
    <input type="text" id="title" name="title" required>
    <br>
    <label for="content">Content:</label>
    <textarea id="content" name="content" required></textarea>
    <br>
    <button type="submit">Save</button>
</form>
<a href="/pages/notes">Back to Notes</a>
</body>
</html>
```

Приложение 21, файл edit-note.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Edit Note</title>
</head>
<body>
<h1>Edit Note</h1>
<form th:action="@{/notes/update/{id}(id=${note.id})}" method="post">
    <label for="title">Title:</label>
    <input type="text" id="title" name="title" th:value="${note.title}" required>
    <br>
    <label for="content">Content:</label>
    <textarea id="content" name="content" th:text="${note.content}"
required></textarea>
    <br>
    <button type="submit">Save</button>
</form>
<a href="/pages/notes">Back to Notes</a>
</body>
</html>
```

Приложение 22, файл styles.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    color: #333;
    padding: 0;
    margin: 20px;
}

header {
    background-color: #007BFF;
    color: white;
    padding: 10px 20px;
    text-align: center;
}

main {
    padding: 20px;
}

h1, h2 {
    margin: 0 0 10px;
}

ul {
    list-style-type: none;
    padding: 0;
}
```

```
li {
    background: white;
    margin: 10px;
    padding: 10px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

button, a {
    text-decoration: none;
    background-color: #007BFF;
    color: white;
    padding: 10px;
    border-radius: 5px;
    border: none;
    cursor: pointer;
}

button:hover, a:hover {
    background-color: #0056b3;
}
```