Современная разработка на Spring Boot: REST API с валидацией и базой данных

Конспект по разработке веб-приложений

18 ноября 2025 г.

Содержание

1 Введение

Данный конспект посвящен созданию RESTful API с использованием современного стека технологий Spring Boot. Рассматриваются ключевые компоненты для быстрой разработки веб-приложений с поддержкой базы данных, валидации данных и автоматического конфигурирования.

1.1 Технологический стек

1. **Spring WEB** --- основной модуль для создания веб-приложений и REST API
2. **H2 Database** --- in-memory база данных для разработки и тестирования
3. **Spring Boot DevTools** --- инструменты для ускорения разработки (автоматическая перезагрузка)
4. **Spring Data JPA** --- абстракция для работы с базами данных через JPA
5. **Lombok** --- библиотека для автоматической генерации шаблонного кода
6. **Validation** --- валидация входных данных с помощью Jakarta Validation
7. **Java 17/21** --- современные LTS-версии Java с улучшенными возможностями

2 Модель данных

Создание сущности для работы с заметками. Используется аннотации JPA для маппинга на таблицу базы данных.

2.1 Сущность Note

```
1  package com.example.demo.model;
2
3  import jakarta.persistence.*;
4  import lombok.*;
5
6  @Entity
7  @Table(name = "notes")
8  @Getter
9  @Setter
10 @NoArgsConstructor
11 @AllArgsConstructor
12 @Builder
13 @ToString
14 @EqualsAndHashCode
15 public class Note {
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
```

```
20
21      @Column(nullable = false,length = 100)
22      private String title;
23
24      @Column(nullable = false, columnDefinition = "TEXT")
25      private String content;
26  }
```

Listing 1: Модель данных: класс Note

Пояснения к аннотациям:

1. `@Entity` --- помечает класс как JPA-сущность
2. `@Table(name = "notes")` --- указывает имя таблицы в базе данных
3. `@Id` и `@GeneratedValue` --- определяют первичный ключ и стратегию генерации
4. `@Column` --- настраивает параметры колонки (обязательность, размер, тип)
5. `@Getter`, `@Setter` --- генерируют геттеры и сеттеры (Lombok)
6. `@NoArgsConstructor`, `@AllArgsConstructor` --- генерируют конструкторы без параметров и со всеми параметрами
7. `@Builder` --- создает паттерн Builder для удобного создания объектов
8. `@ToString`, `@EqualsAndHashCode` --- генерируют методы toString(), equals() и hashCode()

3 Репозиторий данных

Spring Data JPA предоставляет готовые CRUD-операции через наследование от JpaRepository.

3.1 NoteRepository

```
1  package com.example.demo.repositories;
2
3  import com.example.demo.model.Note;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface NoteRepository extends JpaRepository<Note, Long> {
```

```
}
```

Listing 2: Репозиторий для работы с заметками

Важное замечание: Spring Boot автоматически сканирует все классы внутри пакета (и подпакетов) на предмет аннотаций, включая `@Repository`. Обнаруженные компоненты регистрируются в ApplicationContext, после чего происходит автоматическая инъекция зависимостей (DI).

4 DTO и валидация

Для передачи данных между клиентом и сервером используются DTO (Data Transfer Objects) с валидацией.

4.1 NoteCreateDto с валидацией

```
record NoteCreateDto(
        @NotEmpty
        @Size(max = 100,
                message = "Слишком длинная строка")
        String title,

        @Size(max = 10000,
                message = "Слишком длинная строка")
        String content
) {

}
```

Listing 3: DTO для создания заметки с валидацией

Аннотации валидации:

1. `@NotNull` --- поле не должно быть null (для объектов)
2. `@NotEmpty` --- поле не null и не пустая коллекция/строка
3. `@NotBlank` --- не null, не пустая и не состоит только из пробелов (лучше всего для строк)
4. `@Size(min=, max=)` --- ограничение на размер строки или коллекции
5. `@Min/@Max` --- ограничение для числовых значений
6. `@Positive/@PositiveOrZero` --- положительные числа
7. `@Past/@Future` --- проверка дат

8. `@Email` --- базовая проверка email-адреса
9. `@Pattern(regexp = "...")` --- проверка по регулярному выражению
10. `@AssertTrue/@AssertFalse` --- кастомная логическая валидация

5 Контроллер REST API

Реализация CRUD-операций для управления заметками.

5.1 NoteController

```java
package com.example.demo.controllers;

import com.example.demo.model.Note;
import com.example.demo.repositories.NoteRepository;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.concurrent.atomic.AtomicReference;

@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class NoteController {

    private final NoteRepository noteRepository;

    @GetMapping
    public List<Note> getAllNotes() {
        return noteRepository.findAll();
    }

    //ResponseEntity для корректных статусов
    @PostMapping
    public ResponseEntity<Note> createNote(@RequestBody @Valid NoteCreateDto
    noteDto) {
        var note = Note.builder().title(noteDto.title()).content(noteDto.
    content()).build();
        var saved = noteRepository.save(note);
        return new ResponseEntity(saved,HttpStatus.CREATED);
    }

```

```
34      @PutMapping("/{id}")
35      public ResponseEntity<Note> updateNote(
36              @PathVariable Long id,                  // или Long, если ещё не
    перешёл на UUID
37              @RequestBody @Valid NoteCreateDto dto) { // @Valid — обязательно
    для валидации
38
39          // Возвращаем готовый ResponseEntity сразу через orElseThrow + map
40          return noteRepository.findById(id)
41                  .map(existingNote -> {
42                      existingNote.setTitle(dto.title());
43                      existingNote.setContent(dto.content());
44                      Note updatedNote = noteRepository.save(existingNote);
45                      return ResponseEntity.ok(updatedNote);          // 200 OK
46                  })
47                  .orElseGet(() -> ResponseEntity.notFound().build());  // 404
    Not Found
48      }
49
50      @DeleteMapping("/{id}")
51      public ResponseEntity<Void> deleteNote(@PathVariable Long id) {
52          noteRepository.deleteById(id);
53          return ResponseEntity.noContent().build();
54      }
55  }
```

Listing 4: REST контроллер для работы с заметками

Ключевые особенности:

1. `@RestController` --- комбинирует @Controller и @ResponseBody
2. `@RequiredArgsConstructor` --- генерирует конструктор для final-полей (Lombok)
3. `ResponseEntity` --- позволяет контролировать HTTP-статусы и заголовки
4. `@Valid` --- активирует валидацию DTO перед обработкой
5. Функциональный подход с `map()` и `orElseGet()` для обработки Optional
6. Корректные HTTP-статусы: 201 Created, 200 OK, 404 Not Found, 204 No Content

6 Глобальная обработка ошибок

Обработка исключений валидации на уровне приложения.

6.1 GlobalExceptionHandler

```java
package com.example.demo.controllers;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>> handleValidationExceptions(
            MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(error ->
                errors.put(error.getField(), error.getDefaultMessage()));

        return ResponseEntity.badRequest().body(errors);
    }
}
```

Listing 5: Глобальный обработчик исключений

Принцип работы:

1. `@RestControllerAdvice` --- применяет обработчики ко всем контроллерам
2. `@ExceptionHandler` --- перехватывает конкретные типы исключений
3. `MethodArgumentNotValidException` --- генерируется при провале валидации
4. Формируется понятный JSON-ответ с ошибками для каждого поля
5. Возвращается HTTP-статус 400 Bad Request

7 Рекомендации по разработке

1. **Валидация** --- всегда используйте `@Valid` для DTO в контроллерах
2. **HTTP-статусы** --- возвращайте соответствующие статусы для каждой операции

3. **DTO** --- никогда не передавайте сущности напрямую в клиент, используйте DTO
4. **Иммутабельность** --- используйте record для DTO вместо классов
5. **Обработка ошибок** --- централизованная обработка исключений улучшает поддерживаемость
6. **Производительность** --- для продакшена замените H2 на PostgreSQL или MySQL

8 Заключение

Представленный стек технологий позволяет быстро создавать надежные REST API с минимальным количеством шаблонного кода. Использование Lombok, Spring Data JPA и валидации значительно ускоряет разработку, а правильная обработка ошибок и HTTP-статусов делает API профессиональным и удобным для клиентов.

Данный подход легко масштабируется и может быть дополнен аутентификацией, авторизацией, кэшированием и другими enterprise-функциями при необходимости.