

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

Конспект по разработке веб-приложений

21 ноября 2025 г.

Содержание

1 Введение	1
1.1 Технологический стек	1
2 Модель данных	2
2.1 Сущность Note	2
3 Репозиторий данных	3
3.1 NoteRepository	3
4 DTO и валидация	3
4.1 NoteCreateDto с валидацией	3
5 Контроллер REST API	4
5.1 NoteController	4
6 Глобальная обработка ошибок	6
6.1 GlobalExceptionHandler	6
7 Рекомендации по разработке	7
8 Заключение	8

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> {
```

```
9 }
```

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

```
1 record NoteCreateDto(
2     @NotEmpty
3     @Size(max = 100,
4             message = "Слишком длинная строка")
5     String title,
6
7     @Size(max = 10000,
8             message = "Слишком длинная строка")
9     String content
10 ) {
11
12 }
```

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

```

1 package com.example.demo.controllers;
2
3 import com.example.demo.model.Note;
4 import com.example.demo.repositories.NoteRepository;
5 import jakarta.validation.Valid;
6 import lombok.RequiredArgsConstructor;
7 import org.springframework.http.HttpStatus;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.web.bind.annotation.*;
10
11 import java.util.List;
12 import java.util.concurrent.atomic.AtomicReference;
13
14 @RestController
15 @RequestMapping("/api")
16 @RequiredArgsConstructor
17 public class NoteController {
18
19     private final NoteRepository noteRepository;
20
21     @GetMapping
22     public List<Note> getAllNotes() {
23         return noteRepository.findAll();
24     }
25
26     // ResponseEntity для корректных статусов
27     @PostMapping
28     public ResponseEntity<Note> createNote(@RequestBody @Valid NoteCreateDto
29     noteDto) {
30         var note = Note.builder().title(noteDto.title()).content(noteDto.
31         content()).build();
32         var saved = noteRepository.save(note);
33         return new ResponseEntity(saved, HttpStatus.CREATED);
34     }
35 }
```

```

34     @PutMapping("/{id}")
35     public ResponseEntity<Note> updateNote(
36         @PathVariable Long id,                                // или Long, если ещё не
37         @RequestBody @Valid NoteCreateDto dto) { // @Valid – обязательно
38         // для валидации
39
39     // Возвращаем готовый ResponseEntity сразу черезorElseThrow + 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
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

```
1 package com.example.demo.controllers;
2
3 import org.springframework.http.ResponseEntity;
4 import org.springframework.web.bind.MethodArgumentNotValidException;
5 import org.springframework.web.bind.annotation.ExceptionHandler;
6 import org.springframework.web.bind.annotation.RestControllerAdvice;
7
8 import java.util.HashMap;
9 import java.util.Map;
10
11 @RestControllerAdvice
12 public class GlobalExceptionHandler {
13
14     @ExceptionHandler(MethodArgumentNotValidException.class)
15     public ResponseEntity<Map<String, String>> handleValidationExceptions(
16         MethodArgumentNotValidException ex) {
17         Map<String, String> errors = new HashMap<>();
18         ex.getBindingResult().getFieldErrors().forEach(error ->
19             errors.put(error.getField(), error.getDefaultMessage()));
20
21         return ResponseEntity.badRequest().body(errors);
22     }
23 }
```

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.