Семинар №6
Проектирование и реализация API для серверного приложения

1. **Инструментарий:**

Урок
Презентация

---

2. **Цели семинара:**

- Получить базовое понимание проектирования и реализации API для серверного приложения.
- Закрепить знания посредством практических заданий

По итогам семинара слушатель должен знать:

- Основные принципы проектирования API.
- Стандарты и практики для реализации API (например, RESTful).
- Особенности безопасности при реализации API

По итогам семинара слушатель должен уметь:

- Создавать документацию для API с использованием инструментов, например Swagger.
- Проектировать и реализовывать базовые API запросы.
- Применять механизмы авторизации и аутентификации в API

---

3. **План Содержание:**

Этап урока	Тайминг, минуты	Формат
Введение, обзор темы	20	Модерирует преподаватель
Задание 1	40	Студенты выполняют, преподаватель помогает в решении проблем

Задание 2	40	Студенты выполняют, преподаватель помогает в решении проблем
Вопросы и обсуждение	20	Модерирует преподаватель
Длительность:	**120**	

4. Блок 1.

Тайминг:
Объяснение правил – 10 минут
Работа – 30 минут

Задание:

Ваша задача - разработать RESTful API для серверного приложения, используя Spring Boot и Spring Web. API будет предназначено для управления сущностью "Продукт" (Product) в интернет-магазине.

Сущность "Продукт" должна содержать следующие поля:
- ID (тип Long и автоинкрементное)
- Название (тип String, не может быть пустым)
- Цена (тип Double, не может быть меньше 0)
- Количество на складе (тип Integer, не может быть меньше 0)

Требуется реализовать следующие эндпоинты:

- GET /products - получение списка всех продуктов.
- GET /products/{id} - получение продукта по ID.
- POST /products - создание нового продукта.
- PUT /products/{id} - обновление продукта по ID.
- DELETE /products/{id} - удаление продукта по ID.

Пример решения:

1. Сущность Product:

```
@Entity
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private Double price;

    @Column(nullable = false)
    private Integer quantityInStock;

    // геттеры, сеттеры, конструкторы, equals, hashCode
}
```

2. Контроллер:

```
@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping
    public List<Product> getAllProducts() {
        return productService.findAll();
    }

    @GetMapping("/{id}")
    public Product getProductById(@PathVariable Long id) {
        return productService.findById(id);
    }

    @PostMapping
    public Product createProduct(@RequestBody Product product) {
        return productService.save(product);
    }

    @PutMapping("/{id}")
    public Product updateProduct(@PathVariable Long id, @RequestBody Product product) {
        return productService.update(id, product);
    }
```

```
    @DeleteMapping("/{id}")
    public void deleteProduct(@PathVariable Long id) {
        productService.delete(id);
    }
}
```

Часто встречающиеся ошибки:

1. Не указание @RequestBody в методах POST и PUT, что приведет к тому, что тело запроса не будет преобразовано в объект Product.
2. Забывают про обработку исключений, например, не найденный продукт по ID, что может привести к непредвиденным ошибкам на стороне клиента.
3. Не проведение валидации входящих данных, что может привести к добавлению некорректных продуктов (например, с отрицательной ценой или количеством).
4. Неправильное или отсутствующее тестирование, что может привести к неработающему или нестабильному API.

5. Блок 2.

Тайминг:
Объяснение правил – 10 минут
Работа в команде – 20 минут

Задание:

Создайте RESTful API на базе Spring Boot и Spring Web для управления библиотекой. Основной сущностью будет "Читатель" (Reader) и "Книга" (Book). Необходимо учитывать, что одна книга может быть взята только одним читателем, но один читатель может взять несколько книг.

Сущность "Читатель":
- ID (тип Long и автоинкрементное)
- Имя (тип String, не может быть пустым)

Сущность "Книга":
- ID (тип Long и автоинкрементное)
- Название (тип String, не может быть пустым)
- Автор (тип String)
- Читатель (ссылка на сущность "Читатель", может быть null)

Требуется реализовать следующие эндпоинты:

- GET /books - получение списка всех книг.
- GET /books/{id} - получение книги по ID.
- POST /books - добавление новой книги.
- PUT /books/{id}/reader/{readerId} - назначение читателя для книги.
- DELETE /books/{id} - удаление книги.
- GET /readers - получение списка всех читателей.
- POST /readers - добавление нового читателя.

Пример решения:

1. Сущность Reader:

```
@Entity
public class Reader {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(nullable = false)
   private String name;

   // геттеры, сеттеры, конструкторы, equals, hashCode
}
```

2. Сущность Book:

```
@Entity
public class Book {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(nullable = false)
   private String title;

   @Column
   private String author;

   @ManyToOne
   @JoinColumn(name = "reader_id")
```

```
    private Reader reader;

    // геттеры, сеттеры, конструкторы, equals, hashCode
}
```

3. Контроллер:

```
@RestController
public class LibraryController {

    @Autowired
    private BookService bookService;

    @Autowired
    private ReaderService readerService;

    @GetMapping("/books")
    public List<Book> getAllBooks() {
        return bookService.findAll();
    }

    @GetMapping("/books/{id}")
    public Book getBookById(@PathVariable Long id) {
        return bookService.findById(id);
    }

    @PostMapping("/books")
    public Book addBook(@RequestBody Book book) {
        return bookService.save(book);
    }

    @PutMapping("/books/{id}/reader/{readerId}")
    public Book assignReaderToBook(@PathVariable Long id, @PathVariable Long readerId) {
        return bookService.assignReader(id, readerId);
    }

    @DeleteMapping("/books/{id}")
    public void deleteBook(@PathVariable Long id) {
        bookService.delete(id);
    }

    @GetMapping("/readers")
```

```
    public List<Reader> getAllReaders() {
        return readerService.findAll();
    }

    @PostMapping("/readers")
    public Reader addReader(@RequestBody Reader reader) {
        return readerService.save(reader);
    }
}
```

Часто встречающиеся ошибки:

1. Отсутствие валидации на уровне API (например, при добавлении книги без названия).
2. Не правильно настроенные отношения между сущностями, что может привести к ошибкам целостности данных.
3. Не обработанные исключения (например, при попытке взять книгу, которая уже у другого читателя).
4. Неправильная или отсутствующая обработка ошибок, что может привести к неинформативным ответам API.

6. Домашнее задание

Домашнее задание:

Условие:

Разработайте небольшое веб-приложение на Spring Boot, которое будет представлять из себя сервис для учета личных заметок. Приложение должно поддерживать следующие функции:

1. Добавление заметки.
2. Просмотр всех заметок.
3. Редактирование заметки.
4. Удаление заметки.

Структура заметки:

- ID (автоинкрементное)
- Заголовок (не может быть пустым)
- Содержимое (не может быть пустым)

- Дата создания (автоматически устанавливается при создании заметки)