Product Operations: Saving, Updating, and Deleting

This documentation provides an overview of how to create APIs for saving, updating, and deleting products in a Spring Boot application.

Prerequisites

Before you start, ensure that you have the following set up:

- Java Development Kit (JDK) 8 or above installed.
- An integrated development environment (IDE) such as IntelliJ IDEA or Eclipse.
- A new Spring Boot project created.

Step 1: Create the Product Model

The Product model represents the data structure for a product. Here's an example implementation:

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private double price;
    private String description;

    // Constructors, getters, and setters
}
```

Step 2: Create the Product Repository

The Product repository allows interaction with the database. Here's an example implementation:

```
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

Step 3: Implement the Product Service

The Product service contains the business logic for saving, updating, and deleting products. Here's an example implementation:

```
@Service
public class ProductService {
    @Autowired
    private ProductRepository productRepository;

    public Product saveProduct(Product product) {
        // Perform validation or business logic if needed
        return productRepository.save(product);
    }
    public Product updateProduct(Long id, Product updatedProduct) {
        Optional<Product> existingProduct = productRepository.findById(id);
        if (existingProduct.isPresent()) {
            Product product = existingProduct.get();
            product.setName(updatedProduct.getName());
            product.setPrice(updatedProduct.getPrice());
            product.setDescription(updatedProduct.getDescription());
            return productRepository.save(product);
        }
        return null; // Handle the case when the product doesn't exist
    }

    public void deleteProduct(Long id) {
        productRepository.deleteById(id);
    }
}
```

Step 4: Create the Product Controller

The Product controller handles HTTP requests related to products. Here's an example implementation:

```
@RestController
@RequestMapping("/api/products")
public class ProductController {
    @Autowired
    private ProductService productService;

    @PostMapping
    public ResponseEntity<Product> createProduct(@RequestBody Product product) {
        Product savedProduct = productService.saveProduct(product);
        return ResponseEntity.status(HttpStatus.CREATED).body(savedProduct);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Product> updateProduct(@PathVariable Long id, @RequestBody
```

```
Product product) {
        Product updatedProduct = productService.updateProduct(id, product);
        if (updatedProduct != null) {
            return ResponseEntity.ok(updatedProduct);
        }
        return ResponseEntity.notFound().build();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
        productService.deleteProduct(id);
        return ResponseEntity.noContent().build();
    }
}
```

Usage:

Once you have implemented the above components, you can use the endpoints mentioned in the ***"E-commerce Application"*** to interact with the Product API. Make sure to replace com.example.demo with your package name. You can test the API endpoints using tools like Postman.