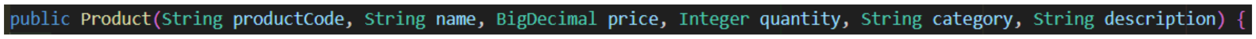
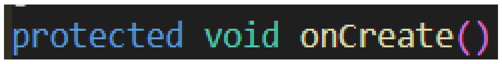
Product.java:

1. Object Creation – A new "Product" is instantiated using the constructor with product details.



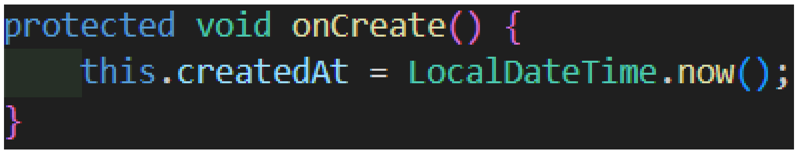
```java
public Product(String productCode, String name, BigDecimal price, Integer quantity, String category, String description) {
```

2. Before Save – When "save()" is called in JPA/Hibernate, the "@PrePersist" method runs and sets "createdAt" to now.



```java
protected void onCreate()
```

3. Insertion – Hibernate generates an SQL "INSERT" into the "products" table using the entity fields.

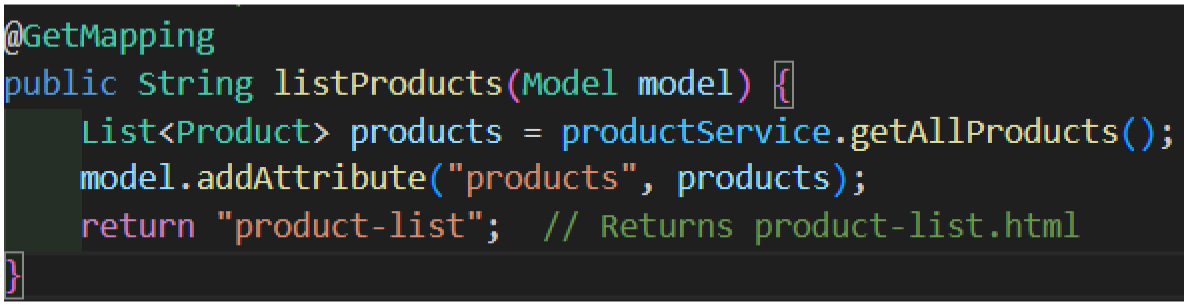


```java
protected void onCreate() {
    this.createdAt = LocalDateTime.now();
}
```

4. ID Handling – The database auto-generates the "id" (because of "GenerationType.IDENTITY") and JPA updates the entity with that ID. (@Entity, @Table, @Column, and all field declarations)
5. Entity Usage – The saved product can now be queried, updated, or displayed using getters, setters, or "toString()". (@Id + @GeneratedValue(strategy = GenerationType.IDENTITY) on id)

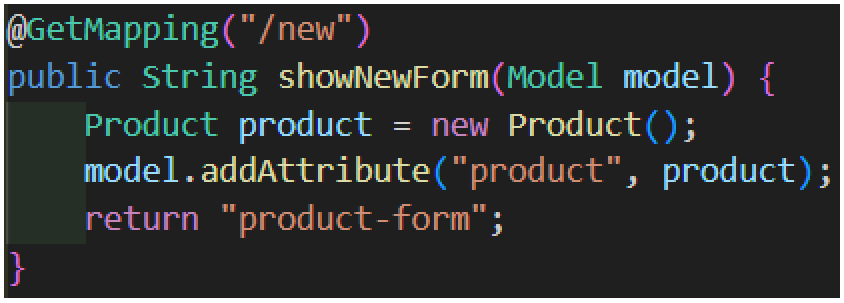
ProductController.java:

1. User requests product list → Controller handles (@GetMapping → listProducts() → calls productService.getAllProducts() → returns "product-list".)

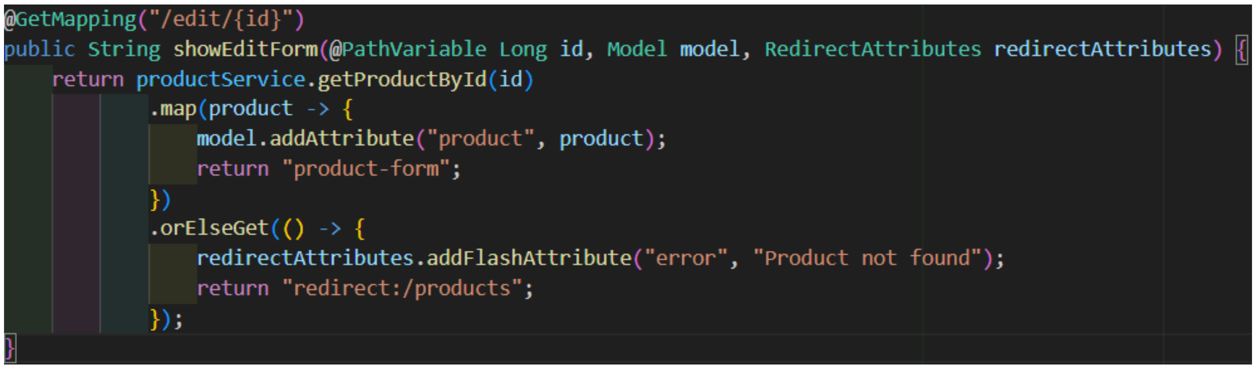


```java
@GetMapping
public String listProducts(Model model) {
    List<Product> products = productService.getAllProducts();
    model.addAttribute("products", products);
    return "product-list";  // Returns product-list.html
}
```

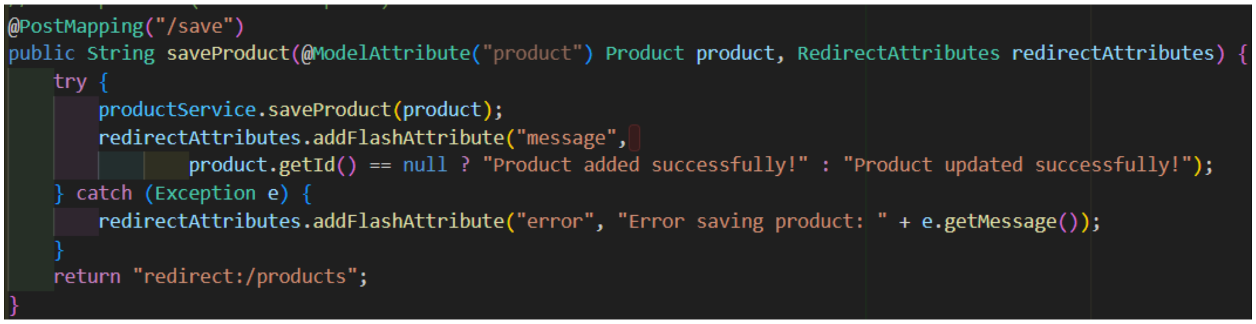
2. User opens create or edit form → Controller prepares model
   - New: @GetMapping("/new") → showNewForm() → adds new Product()


```
@GetMapping("/new")
public String showNewForm(Model model) {
    Product product = new Product();
    model.addAttribute("product", product);
    return "product-form";
}
```

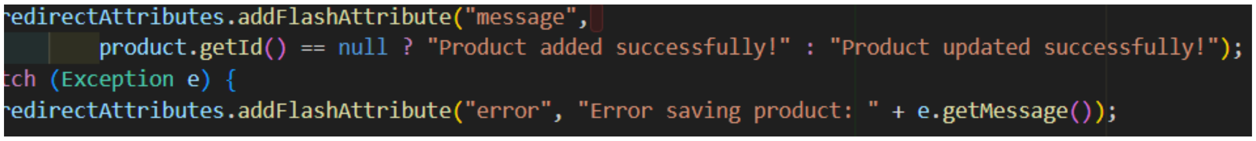
- Edit: @GetMapping("/edit/{id}") → showEditForm() → loads product from productService.getProductById(id)


```
@GetMapping("/edit/{id}")
public String showEditForm(@PathVariable Long id, Model model, RedirectAttributes redirectAttributes) {
    return productService.getProductById(id)
            .map(product -> {
                model.addAttribute("product", product);
                return "product-form";
            })
            .orElseGet(() -> {
                redirectAttributes.addFlashAttribute("error", "Product not found");
                return "redirect:/products";
            });
}
```

3. User submits form → Controller saves product (@PostMapping("/save") → saveProduct() → calls productService.saveProduct(product).)

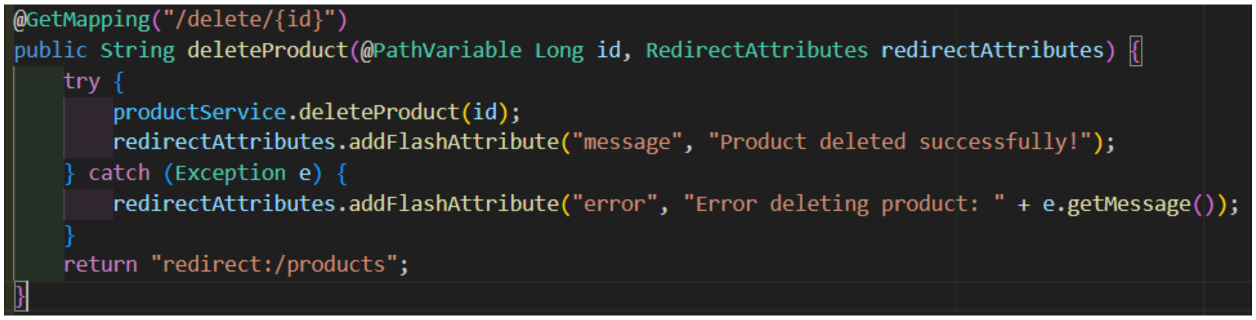

```
@PostMapping("/save")
public String saveProduct(@ModelAttribute("product") Product product, RedirectAttributes redirectAttributes) {
    try {
        productService.saveProduct(product);
        redirectAttributes.addFlashAttribute("message",
                product.getId() == null ? "Product added successfully!" : "Product updated successfully!");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", "Error saving product: " + e.getMessage());
    }
    return "redirect:/products";
}
```

4. After save → Redirect with success/error message (RedirectAttributes.addFlashAttribute(...) → return "redirect:/products".)


```
edirectAttributes.addFlashAttribute("message",
        product.getId() == null ? "Product added successfully!" : "Product updated successfully!");
tch (Exception e) {
edirectAttributes.addFlashAttribute("error", "Error saving product: " + e.getMessage());
```

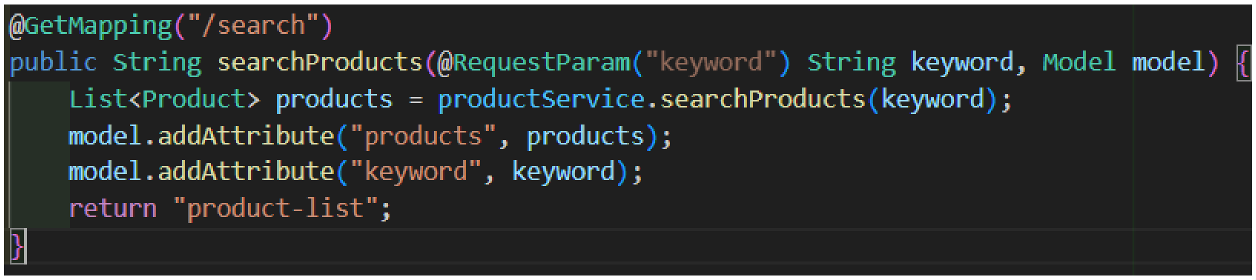
5. Optional actions → Delete or Search

- Delete → @GetMapping("/delete/{id}") → deleteProduct()



```java
@GetMapping("/delete/{id}")
public String deleteProduct(@PathVariable Long id, RedirectAttributes redirectAttributes) {
    try {
        productService.deleteProduct(id);
        redirectAttributes.addFlashAttribute("message", "Product deleted successfully!");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", "Error deleting product: " + e.getMessage());
    }
    return "redirect:/products";
}
```

- Search → @GetMapping("/search") → searchProducts(keyword)



```java
@GetMapping("/search")
public String searchProducts(@RequestParam("keyword") String keyword, Model model) {
    List<Product> products = productService.searchProducts(keyword);
    model.addAttribute("products", products);
    model.addAttribute("keyword", keyword);
    return "product-list";
}
```

ProductRepository.java:

1. Controller or Service requests data
   => Calls inherited JPA methods like "findAll()", "findById(id)", or custom methods.
2. Spring Data JPA auto-generates SQL based on method names
   e.g., "findByNameContaining(String keyword)" → generates LIKE "%keyword%" query automatically.
3. Repository sends the query to the database
   (All methods defined in "ProductRepository" (e.g., "findByCategory", "findByPriceBetween").)
4. Database returns matching rows which Spring maps into "Product" entities
   (Entity class: "Product" bound to table "products".)
5. Service returns results back to controller for displaying in UI
   (Controller methods like "listProducts()", "searchProducts()", etc.)

ProductService.java:

1. Controller calls service methods to handle business logic
   (Methods like "getAllProducts()", "saveProduct(product)" are invoked from "ProductController.")

2. Service implementation interacts with repository
   (Each method in “ProductService” will call equivalent repository methods (e.g., “findAll()”, “save()”).)
3. Business rules are applied before saving or returning data
   (In “saveProduct(Product product)” the implementation may validate fields or check duplicates.)
4. Service receives data from repository
   (Methods like “getProductById(Long id)” return “Optional<Product>”.)
5. Service returns final results to controller for views or redirection
   (“searchProducts(keyword)” → returned to controller → rendered in UI.)

ProductServiceImpl.java:

1. Controller calls service → Service receives request
   (Controller uses methods like “getAllProducts()”, “saveProduct(product)”.)
2. Service forwards request to repository
   (“productRepository.findAll()”, “productRepository.findById(id)”, “productRepository.save(product)”.)
3. Business logic (optional) applied inside service
   (Inside “saveProduct(Product product)” you can add validation before “save()”.)
4. Repository executes SQL via Spring Data JPA
   (All repository calls inside service: “findAll()”, “findById()”, “save()”, “deleteById()”, “findByNameContaining()”.)
5. Service returns result back to controller
   (Methods return “List<Product>”, “Optional<Product>”, or saved “Product”.)