Projet Spring Boot - Inventory Management

Sylvain Wanja Njangang

Contents

Licensed to the Apache Software Foundation (ASF) under one

distributed with this work for additional information

regarding copyright ownership. The ASF licenses this file

to you under the Apache License, Version 2.0 (the

Unless required by applicable law or agreed to in writing,

software distributed under the License is distributed on an

“AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

specific language governing permissions and limitations

under the License.

wrapperVersion=3.3.2 distributionType=only-script distributionUrl=https://repo.maven.apache.org/mave
maven/3.9.9/apache-maven-3.9.9-bin.zip

Read Me First

The following was discovered as part of building this project:

- The original package name ‘com.example.inventory-management’ is invalid and this project uses ‘com.example.inventorymanagement’ instead.

Getting Started

Reference Documentation

For further reference, please consider the following sections:

- Official Apache Maven documentation
- Spring Boot Maven Plugin Reference Guide
- Create an OCI image
- Spring Web
- Spring Data JPA
- Spring Security
- Spring Boot Actuator
- Validation

Guides

The following guides illustrate how to use some features concretely:

- Building a RESTful Web Service
- Serving Web Content with Spring MVC
- Building REST services with Spring
- Accessing Data with JPA
- Securing a Web Application
- Spring Boot and OAuth2
- Authenticating a User with LDAP
- Building a RESTful Web Service with Spring Boot Actuator
- Validation
- Accessing data with MySQL

Maven Parent overrides

Due to Maven's design, elements are inherited from the parent POM to the project POM. While most of the inheritance is fine, it also inherits unwanted elements like `<license>` and `<developers>` from the parent. To prevent this, the project POM contains empty overrides for these elements. If you manually switch to a different parent and actually want the inheritance, you need to remove those overrides.

package com.example.inventorymanagement;

import com.example.inventorymanagement.entity.Category; import com.example.inventorymanagement.en
import com.example.inventorymanagement.repository.CategoryRepository; import
com.example.inventorymanagement.repository.ProductRepository; import com.example.inventorymanager
import com.example.inventorymanagement.service.ProductService; import org.springframework.boot.Com
import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.Sprin
import org.springframework.context.annotation.Bean;

@SpringBootApplication public class InventoryManagementApplication {

```
public static void main(String[] args) {
    SpringApplication.run(InventoryManagementApplication.class, args);
}

@Bean
CommandLineRunner testServices(ProductService productService, CategoryService categorySe
    return args -> {
        // Créer une catégorie
        //Category cat = new Category();
        //cat.setName("Informatique");
        //categoryService.createCategory(cat);

        // Créer un produit
        ////////productService.createProduct(product);

        System.out.println("Produits en base : " + productService.getAllProducts().size(
    };
}
```

}

import com.example.inventorymanagement.entity.AuditLog; import com.example.inventorymanagement.se
import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/api/audit-logs") public class AuditLogController {

```
private final AuditLogService auditLogService;

public AuditLogController(AuditLogService auditLogService) {
    this.auditLogService = auditLogService;
}

@GetMapping
public ResponseEntity<List<AuditLog>> getAllLogs() {
    return ResponseEntity.ok(auditLogService.getAllLogs());
}
```

```
@GetMapping("/{id}")
public ResponseEntity<AuditLog> getLogById(@PathVariable Long id) {
    return ResponseEntity.ok(auditLogService.getLogById(id));
}

// Si on souhaite créer un audit log manuellement
@PostMapping
public ResponseEntity<AuditLog> createAuditLog(@RequestBody AuditLog log) {
    return ResponseEntity.ok(auditLogService.createLog(log));
}

// Souvent, on n'autorise pas la suppression ni la mise à jour d'un audit log
```

}

package com.example.inventorymanagement.controller;

import com.example.inventorymanagement.dto.AuthResponse; import com.example.inventorymanagement
import com.example.inventorymanagement.security.AuthService; import org.springframework.http.Respon
import org.springframework.web.bind.annotation.*;

@RestController @RequestMapping("/auth") public class AuthController {

```
private final AuthService authService;

public AuthController(AuthService authService) {
    this.authService = authService;
}

@PostMapping("/login")
public ResponseEntity<AuthResponse> login(@RequestBody LoginRequest loginRequest) {
    // Utiliser loginRequest.username() et loginRequest.password() car ce sont des recor
    String token = authService.login(loginRequest.username(), loginRequest.password());
    return ResponseEntity.ok(new AuthResponse(token));
}
```

}

package com.example.inventorymanagement.controller;

import com.example.inventorymanagement.entity.Category; import com.example.inventorymanagement.se
import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/api/categories") public class CategoryController {

```
private final CategoryService categoryService;
```

```
public CategoryController(CategoryService categoryService) {
    this.categoryService = categoryService;
}

@GetMapping
public ResponseEntity<List<Category>> getAllCategories() {
    return ResponseEntity.ok(categoryService.getAllCategories());
}

@GetMapping("/{id}")
public ResponseEntity<Category> getCategoryById(@PathVariable Long id) {
    return ResponseEntity.ok(categoryService.getCategoryById(id));
}

@PostMapping
public ResponseEntity<Category> createCategory(@RequestBody Category category) {
    Category created = categoryService.createCategory(category);
    return ResponseEntity.ok(created);
}

@PutMapping("/{id}")
public ResponseEntity<Category> updateCategory(@PathVariable Long id,
                                               @RequestBody Category category) {
    Category updated = categoryService.updateCategory(id, category);
    return ResponseEntity.ok(updated);
}

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

}

import com.example.inventorymanagement.entity.Customer; import com.example.inventorymanagement.se
import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping(“/api/customers”) public class CustomerController {

```
private final CustomerService customerService;

public CustomerController(CustomerService customerService) {
    this.customerService = customerService;
}
```

```
@GetMapping
public ResponseEntity<List<Customer>> getAllCustomers() {
    return ResponseEntity.ok(customerService.getAllCustomers());
}

@GetMapping("/{id}")
public ResponseEntity<Customer> getCustomerById(@PathVariable Long id) {
    return ResponseEntity.ok(customerService.getCustomerById(id));
}

@PostMapping
public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) {
    Customer created = customerService.createCustomer(customer);
    return ResponseEntity.ok(created);
}

@PutMapping("/{id}")
public ResponseEntity<Customer> updateCustomer(@PathVariable Long id,
                                               @RequestBody Customer customer) {
    Customer updated = customerService.updateCustomer(id, customer);
    return ResponseEntity.ok(updated);
}

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

}

import com.example.inventorymanagement.entity.Location; import com.example.inventorymanagement.ser
import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/api/locations") public class LocationController {

```
private final LocationService locationService;

public LocationController(LocationService locationService) {
    this.locationService = locationService;
}

@GetMapping
public ResponseEntity<List<Location>> getAllLocations() {
    return ResponseEntity.ok(locationService.getAllLocations());
}

@GetMapping("/{id}")
public ResponseEntity<Location> getLocationById(@PathVariable Long id) {
    return ResponseEntity.ok(locationService.getLocationById(id));
}

@PostMapping
public ResponseEntity<Location> createLocation(@RequestBody Location location) {
    Location created = locationService.createLocation(location);
    return ResponseEntity.ok(created);
}

@PutMapping("/{id}")
public ResponseEntity<Location> updateLocation(@PathVariable Long id,
                                               @RequestBody Location location) {
    Location updated = locationService.updateLocation(id, location);
    return ResponseEntity.ok(updated);
}

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

}

import com.example.inventorymanagement.entity.Order; import com.example.inventorymanagement.servic
import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/api/orders") public class OrderController {

```
private final OrderService orderService;

public OrderController(OrderService orderService) {
    this.orderService = orderService;
}

@GetMapping
public ResponseEntity<List<Order>> getAllOrders() {
    return ResponseEntity.ok(orderService.getAllOrders());
}
```

```
@GetMapping("/{id}")
public ResponseEntity<Order> getOrderById(@PathVariable Long id) {
    return ResponseEntity.ok(orderService.getOrderById(id));
}

@PostMapping
public ResponseEntity<Order> createOrder(@RequestBody Order order) {
    return ResponseEntity.ok(orderService.createOrder(order));
}

@PutMapping("/{id}")
public ResponseEntity<Order> updateOrder(@PathVariable Long id,
                                         @RequestBody Order order) {
    return ResponseEntity.ok(orderService.updateOrder(id, order));
}

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

// Exemple d'endpoint pour changer le statut d'une commande
@PatchMapping("/{id}/status")
public ResponseEntity<Order> updateOrderStatus(@PathVariable Long id,
                                               @RequestParam String status) {
    return ResponseEntity.ok(orderService.updateOrderStatus(id, status));
}
```

}

package com.example.inventorymanagement.controller;

import com.example.inventorymanagement.entity.Product; import com.example.inventorymanagement.ser
import jakarta.validation.Valid; import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/api/products") // URL de base public class Product-Controller {

```
private final ProductService productService;

// Injection du service
public ProductController(ProductService productService) {
    this.productService = productService;
}
```

```
// GET /api/products
@GetMapping
public ResponseEntity<List<Product>> getAllProducts() {
    List<Product> products = productService.getAllProducts();
    return ResponseEntity.ok(products);
}

// GET /api/products/{id}
@GetMapping("/{id}")
public ResponseEntity<Product> getProductById(@PathVariable Long id) {
    Product product = productService.getProductById(id);
    return ResponseEntity.ok(product);
}

// POST /api/products
@PostMapping
public ResponseEntity<Product> createProduct(@Valid @RequestBody Product product) {
    Product created = productService.createProduct(product);
    return ResponseEntity.ok(created);
}

// PUT /api/products/{id}
@PutMapping("/{id}")
public ResponseEntity<Product> updateProduct(@PathVariable Long id,
                                             @Valid @RequestBody Product product) {
    Product updated = productService.updateProduct(id, product);
    return ResponseEntity.ok(updated);
}

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

}

import com.example.inventorymanagement.entity.Stock; import com.example.inventorymanagement.servic
import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/api/stocks") public class StockController {

```
private final StockService stockService;
```

```
public StockController(StockService stockService) {
    this.stockService = stockService;
}

@GetMapping
public ResponseEntity<List<Stock>> getAllStocks() {
    return ResponseEntity.ok(stockService.getAllStocks());
}

@GetMapping("/{id}")
public ResponseEntity<Stock> getStockById(@PathVariable Long id) {
    return ResponseEntity.ok(stockService.getStockById(id));
}

@PostMapping
public ResponseEntity<Stock> createStock(@RequestBody Stock stock) {
    Stock created = stockService.createStock(stock);
    return ResponseEntity.ok(created);
}

@PutMapping("/{id}")
public ResponseEntity<Stock> updateStock(@PathVariable Long id,
                                         @RequestBody Stock stock) {
    Stock updated = stockService.updateStock(id, stock);
    return ResponseEntity.ok(updated);
}

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

}

import com.example.inventorymanagement.entity.StockMovement; import com.example.inventorymanager
import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/api/movements") public class StockMovementController {

```
private final StockMovementService stockMovementService;

public StockMovementController(StockMovementService stockMovementService) {
    this.stockMovementService = stockMovementService;
}

@GetMapping
public ResponseEntity<List<StockMovement>> getAllMovements() {
    return ResponseEntity.ok(stockMovementService.getAllMovements());
}

@GetMapping("/{id}")
public ResponseEntity<StockMovement> getMovementById(@PathVariable Long id) {
    return ResponseEntity.ok(stockMovementService.getMovementById(id));
}

@PostMapping
public ResponseEntity<StockMovement> createMovement(@RequestBody StockMovement movement)
    StockMovement created = stockMovementService.createMovement(movement);
    return ResponseEntity.ok(created);
}

// Souvent, on ne met pas de updateMovement, car un mouvement historique ne se modifie p
// Mais vous pouvez en ajouter un si nécessaire

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

}

import com.example.inventorymanagement.entity.Supplier; import com.example.inventorymanagement.ser
import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/api/suppliers") public class SupplierController {

```
private final SupplierService supplierService;

public SupplierController(SupplierService supplierService) {
    this.supplierService = supplierService;
}

@GetMapping
public ResponseEntity<List<Supplier>> getAllSuppliers() {
    return ResponseEntity.ok(supplierService.getAllSuppliers());
}
```

```
@GetMapping("/{id}")
public ResponseEntity<Supplier> getSupplierById(@PathVariable Long id) {
    return ResponseEntity.ok(supplierService.getSupplierById(id));
}

@PostMapping
public ResponseEntity<Supplier> createSupplier(@RequestBody Supplier supplier) {
    return ResponseEntity.ok(supplierService.createSupplier(supplier));
}

@PutMapping("/{id}")
public ResponseEntity<Supplier> updateSupplier(@PathVariable Long id,
                                               @RequestBody Supplier supplier) {
    return ResponseEntity.ok(supplierService.updateSupplier(id, supplier));
}

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

}

package com.example.inventorymanagement.dto;

public record AuthResponse(String token) {

}

package com.example.inventorymanagement.dto;

public record LoginRequest(String username, String password) {}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class AppRole {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(unique = true)
private String roleName; // ex: ROLE_ADMIN, ROLE_USER, etc.
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import lombok.*;

import java.util.HashSet; import java.util.Set;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class AppUser {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(unique = true)
private String username;

private String password; // Hashé en pratique (BCrypt)

@ManyToMany(fetch = FetchType.EAGER)
@JoinTable(name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "role_id"))
private Set<com.example.inventorymanagement.entity.AppRole> roles = new HashSet<>();
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import lombok.*;

import java.time.LocalDateTime;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class AuditLog {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

private String action;         // Ex : "CREATE_PRODUCT", "DELETE_STOCK", etc.
private LocalDateTime dateAction;

@ManyToOne
private com.example.inventorymanagement.entity.AppUser doneBy;

private String details;       // Info supplémentaire (ID créé, etc.)
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import jakarta.validation.constraints.NotBlank; import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class Category {

```
@Id
```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@NotBlank
@Column(unique = true)
private String name;   // Nom unique de la catégorie

private String description;
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class Customer {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

private String name;

@Column(unique = true)
private String email;   // Email unique

private String address; // Informations complémentaires
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class Location {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

private String name;        // Nom (ex: Magasin 1, Entrepôt A, Rayon B5)
private String description; // Infos diverses
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import lombok.*;

import java.time.LocalDate;

@Entity @Data @NoArgsConstructor @AllArgsConstructor @Table(name = “orders”) // ‘order’ est un mot réservé dans certaines BD public class Order {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

// Ex: "PURCHASE" ou "SALE"
private String type;

// Relation avec le client (si type == SALE)
@ManyToOne
private com.example.inventorymanagement.entity.Customer customer;

// Relation avec le fournisseur (si type == PURCHASE)
@ManyToOne
private com.example.inventorymanagement.entity.Supplier supplier;

// Info sur le produit commandé
@ManyToOne
private com.example.inventorymanagement.entity.Product product;

private int quantity;

private LocalDate orderDate;
private LocalDate deliveryDate;

private String status; // en cours, livré, annulé, etc.
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import jakarta.validation.constraints.NotBlank; import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class Product {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@NotBlank
private String name;           // Nom du produit

@NotBlank
@Column(unique = true)
private String sku;            // Code unique (SKU)
private String barcode;        // Code-barres
private String qrCode;         // QR code
private String photoUrl;       // Chemin/URL de l'image
private String description;

// Relation avec Category
@ManyToOne
@JoinColumn(name = "category_id")
private com.example.inventorymanagement.entity.Category category;

// Date de création, date de mise à jour, etc. (facultatif : vous pouvez utiliser @Creat
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import jakarta.validation.constraints.Min; import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class Stock {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

// Relation avec Product (un produit peut être dans plusieurs stocks/emplacements)
@ManyToOne
private com.example.inventorymanagement.entity.Product product;

// Relation avec Location
@ManyToOne
private com.example.inventorymanagement.entity.Location location;

@Min(0)
private int quantity;       // Quantité en stock

private Integer minQuantity; // Niveau minimum
private Integer maxQuantity; // Niveau maximum
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import lombok.*;

import java.time.LocalDateTime;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class StockMovement {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
private Long id;

private String type; // IN, OUT, TRANSFER

@ManyToOne
private com.example.inventorymanagement.entity.Product product;

private int quantity;

private LocalDateTime movementDate;

// Si besoin de la localisation source/destination pour un TRANSFER
@ManyToOne
private com.example.inventorymanagement.entity.Location fromLocation;

@ManyToOne
private com.example.inventorymanagement.entity.Location toLocation;

// L'utilisateur qui a effectué le mouvement
@ManyToOne
private AppUser doneBy;
```

}

package com.example.inventorymanagement.entity;

import jakarta.persistence.*; import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class Supplier {

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

private String name;          // Nom du fournisseur
private String contactInfo;   // Coordonnées (email, téléphone, etc.)
```

}

package com.example.inventorymanagement.exception;

public class DuplicateException extends RuntimeException { public DuplicateException(String message) { super(message); } }

package com.example.inventorymanagement.exception;

import java.time.LocalDateTime;

public class ErrorResponse {

```
private LocalDateTime timestamp;
private String message;
private int status;  // code HTTP

public ErrorResponse(LocalDateTime timestamp, String message, int status) {
    this.timestamp = timestamp;
    this.message = message;
    this.status = status;
}

// Getters / Setters
public LocalDateTime getTimestamp() {
    return timestamp;
}

public void setTimestamp(LocalDateTime timestamp) {
    this.timestamp = timestamp;
}

public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}

public int getStatus() {
    return status;
}

public void setStatus(int status) {
    this.status = status;
}
```

}

package com.example.inventorymanagement.exception;

import org.springframework.http.HttpStatus; import org.springframework.web.bind.MethodArgumentNot
import org.springframework.web.bind.annotation.ExceptionHandler; import org.springframework.web.bind
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.time.LocalDateTime;

@RestControllerAdvice public class GlobalExceptionHandler {

```
@ExceptionHandler(ResourceNotFoundException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
```

```
public ErrorResponse handleResourceNotFound(ResourceNotFoundException ex) {
    return new ErrorResponse(LocalDateTime.now(), ex.getMessage(), HttpStatus.NOT_FOUND.
}

@ExceptionHandler(DuplicateException.class)
@ResponseStatus(HttpStatus.CONFLICT)
public ErrorResponse handleDuplicate(DuplicateException ex) {
    return new ErrorResponse(LocalDateTime.now(), ex.getMessage(), HttpStatus.CONFLICT.v
}

// Méthode pour gérer les erreurs de validation
@ExceptionHandler(MethodArgumentNotValidException.class)
@ResponseStatus(HttpStatus.BAD_REQUEST)
public ErrorResponse handleValidation(MethodArgumentNotValidException ex) {
    // Récupère le premier message d'erreur
    String msg = ex.getBindingResult().getAllErrors().get(0).getDefaultMessage();
    return new ErrorResponse(LocalDateTime.now(), msg, HttpStatus.BAD_REQUEST.value());
}

// Handler générique si vous le souhaitez
@ExceptionHandler(Exception.class)
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
public ErrorResponse handleGeneric(Exception ex) {
    return new ErrorResponse(LocalDateTime.now(), ex.getMessage(), HttpStatus.INTERNAL_S
}
```

}

package com.example.inventorymanagement.exception;

public class ResourceNotFoundException extends RuntimeException { public ResourceNotFoundException(String message) { super(message); } }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.AppRole; import org.springframework.data.jpa.repositor
import org.springframework.stereotype.Repository;

@Repository public interface AppRoleRepository extends JpaRepository<AppRole, Long>
{ // Optionnel : findByRoleName(String roleName); }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.AppUser; import org.springframework.data.jpa.repositor
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository public interface AppUserRepository extends JpaRepository<AppUser, Long>
{ Optional findByUsername(String username); }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.AuditLog; import org.springframework.data.jpa.reposito
import org.springframework.stereotype.Repository;

@Repository public interface AuditLogRepository extends JpaRepository<AuditLog, Long>
{ }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.Category; import org.springframework.data.jpa.reposito
import org.springframework.stereotype.Repository;

@Repository public interface CategoryRepository extends JpaRepository<Category, Long>
{ // Optionnel : des méthodes de requête personnalisées // Category findByName(String
name); }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.Customer; import org.springframework.data.jpa.reposito
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository public interface CustomerRepository extends JpaRepository<Customer, Long>
{ Optional findByEmail(String email); }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.Location; import org.springframework.data.jpa.reposito
import org.springframework.stereotype.Repository;

@Repository public interface LocationRepository extends JpaRepository<Location, Long>
{ }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.Order; import org.springframework.data.jpa.repository.J
import org.springframework.stereotype.Repository;

@Repository public interface OrderRepository extends JpaRepository<Order, Long> { //
Par exemple : List findByStatus(String status); }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.Product; import org.springframework.data.jpa.repository
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository public interface ProductRepository extends JpaRepository<Product, Long> {

```
// Exemple de query dérivée
Optional<Product> findBySku(String sku);
```

```
// Autres exemples possibles :
// List<Product> findByNameContainingIgnoreCase(String namePart);
// List<Product> findByCategoryId(Long categoryId);
```

}

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.StockMovement; import org.springframework.data.jpa.re
import org.springframework.stereotype.Repository;

@Repository public interface StockMovementRepository extends JpaRepository<StockMovement, Long> { }

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.Stock; import org.springframework.data.jpa.repository.J
import org.springframework.stereotype.Repository;

@Repository public interface StockRepository extends JpaRepository<Stock, Long> {

```
// Par exemple : trouver un Stock par produit et emplacement
// Optional<Stock> findByProductIdAndLocationId(Long productId, Long locationId);
```

}

package com.example.inventorymanagement.repository;

import com.example.inventorymanagement.entity.Supplier; import org.springframework.data.jpa.repository
import org.springframework.stereotype.Repository;

@Repository public interface SupplierRepository extends JpaRepository<Supplier, Long> {
}

package com.example.inventorymanagement.security;

import io.jsonwebtoken.Claims; import io.jsonwebtoken.Jwts; import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.security.authentication.AuthenticationManager; import org.springframework.security.authentication.UsernamePasswordAuthenticationToken; import org.springframework.security.core.userdetails.UserDetails; import org.springframework.security.core.us
import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.stere

import java.util.ArrayList; import java.util.Date; import java.util.function.Function;

@Service public class AuthService {

```
private final String secretKey = "JwtUtil.JWT_SECRET"; // Clé secrète (à sécuriser en pr
private final AuthenticationManager authenticationManager;
private final UserDetailsService userDetailsService;
private final PasswordEncoder passwordEncoder;

public AuthService(AuthenticationManager authenticationManager,
                   UserDetailsService userDetailsService,
                   PasswordEncoder passwordEncoder) {
    this.authenticationManager = authenticationManager;
    this.userDetailsService = userDetailsService;
    this.passwordEncoder = passwordEncoder;
}

/**
 * Méthode login : authentifie l'utilisateur et renvoie un token JWT.
 */
public String login(String username, String password) {
    // Authentifie l'utilisateur ; une exception est levée en cas d'erreur
    authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username,
    // Charge les détails de l'utilisateur
    UserDetails userDetails = userDetailsService.loadUserByUsername(username);
    // Génère le token à partir des UserDetails
    return generateToken(userDetails);
}

/**
 * Surcharge pour générer un token à partir d'un simple username.
 * Crée un UserDetails minimal et appelle generateToken(UserDetails).
 */
public String generateToken(String username) {
    UserDetails userDetails = new org.springframework.security.core.userdetails.User(use
    return generateToken(userDetails);
}

/**
 * Génère un JWT à partir d'un UserDetails complet.
 */
public String generateToken(UserDetails userDetails) {
    return Jwts.builder()
            .setSubject(userDetails.getUsername())
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60)) // Val
            .signWith(SignatureAlgorithm.HS256, secretKey)
            .compact();
}

public boolean validateToken(String token) {
    try {
        Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token);
        return true;
    } catch (Exception e) {
        return false;
    }
}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
    Claims claims = Jwts.parser()
            .setSigningKey(secretKey)
            .parseClaimsJws(token)
            .getBody();
    return claimsResolver.apply(claims);
}
```

}

import org.springframework.security.core.userdetails.User; import org.springframework.security.core.userde
import org.springframework.security.core.userdetails.UserDetailsService; import org.springframework.secur
import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.secur
import org.springframework.stereotype.Service;

@Service public class CustomUserDetailsService implements UserDetailsService { private final InMemoryUserDetailsManager inMemoryUserDetailsManager; private final PasswordEncoder passwordEncoder;

```
public CustomUserDetailsService(PasswordEncoder passwordEncoder) {
    this.passwordEncoder = passwordEncoder;
    UserDetails user = User.builder()
            .username("user")
            .password(passwordEncoder.encode("password"))
            .roles("USER")
            .build();
    this.inMemoryUserDetailsManager = new InMemoryUserDetailsManager(user);
}

@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
    return inMemoryUserDetailsManager.loadUserByUsername(username);
}
```

}

package com.example.inventorymanagement.security;

import jakarta.servlet.FilterChain; import jakarta.servlet.ServletException; import jakarta.servlet.http.HttpServletRequest; import jakarta.servlet.http.HttpServletResponse;
import org.springframework.context.annotation.Lazy; import org.springframework.security.authentication.
import org.springframework.security.core.context.SecurityContextHolder; import org.springframework.secu
import org.springframework.stereotype.Component; import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException; import java.util.function.Function;

@Component public class JwtAuthFilter extends OncePerRequestFilter {

```
private final AuthService authService;

// On marque l'injection de AuthService comme @Lazy pour briser le cycle
public JwtAuthFilter(@Lazy AuthService authService) {
    this.authService = authService;
}

@Override
protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain filterChain) throws ServletException, IOException {

    String header = request.getHeader("Authorization");
    if (StringUtils.hasText(header) && header.startsWith("Bearer ")) {
        String token = header.substring(7);
        if (authService.validateToken(token)) {
            // Récupération du username depuis le token
            String username = authService.extractClaim(token, claims -> claims.getSubjec

            UsernamePasswordAuthenticationToken authToken =
                    new UsernamePasswordAuthenticationToken(username, null, null);
            authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(reque
            SecurityContextHolder.getContext().setAuthentication(authToken);
        }
    }
    filterChain.doFilter(request, response);
}
```

}

package com.example.inventorymanagement.security;

import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Con
import org.springframework.security.authentication.AuthenticationManager; import
org.springframework.security.config.Customizer; import org.springframework.security.config.annotation.aut
import org.springframework.security.config.annotation.web.builders.HttpSecurity; import
org.springframework.security.config.http.SessionCreationPolicy; import org.springframework.security.core.u
import org.springframework.security.core.userdetails.UserDetails; import org.springframework.security.core
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder; import
org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.security.prov
import org.springframework.security.web.SecurityFilterChain; import org.springframework.security.web.au

@Configuration public class SecurityConfig {

```
private final JwtAuthFilter jwtAuthFilter;
```

```
public SecurityConfig(JwtAuthFilter jwtAuthFilter) {
    this.jwtAuthFilter = jwtAuthFilter;
}

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                    .requestMatchers("/auth/**").permitAll()
                    .anyRequest().authenticated()
            )
            .sessionManagement(sess -> sess.sessionCreationPolicy(SessionCreationPolicy.
            .httpBasic(Customizer.withDefaults());

    // Ajout du filtre JWT
    http.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}

@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration config) t
    return config.getAuthenticationManager();
}
@Bean
public UserDetailsService userDetailsService() {
    UserDetails user = User.withDefaultPasswordEncoder()
            .username("user")
            .password("password")
            .roles("USER")
            .build();
    return new InMemoryUserDetailsManager(user);
}
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

}
```

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.AuditLog; import com.example.inventorymanagement.re
import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transa

import com.example.inventorymanagement.exception.ResourceNotFoundException;

import java.time.LocalDateTime; import java.util.List;

@Service @Transactional public class AuditLogService {

```
private final AuditLogRepository auditLogRepository;

public AuditLogService(AuditLogRepository auditLogRepository) {
    this.auditLogRepository = auditLogRepository;
}

public AuditLog createLog(String action, String details, Long userId) {
    AuditLog log = new AuditLog();
    log.setAction(action);
    log.setDateAction(LocalDateTime.now());
    log.setDetails(details);
    // setDoneBy(...) si vous voulez lier à un utilisateur existant
    return auditLogRepository.save(log);
}

public AuditLog createLog(AuditLog log) {
    // ou version plus brute
    if (log.getDateAction() == null) {
        log.setDateAction(LocalDateTime.now());
    }
    return auditLogRepository.save(log);
}

public List<AuditLog> getAllLogs() {
    return auditLogRepository.findAll();
}

public AuditLog getLogById(Long id) {
    return auditLogRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Log introuvable (ID : " +
}

// Souvent on évite de mettre update ou delete pour un audit log,
// car c'est supposé être un enregistrement immuable.
```

}

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.Category; import com.example.inventorymanagement.re
import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transa
import com.example.inventorymanagement.exception.ResourceNotFoundException;

import java.util.List;

@Service @Transactional public class CategoryService {

```
private final CategoryRepository categoryRepository;

public CategoryService(CategoryRepository categoryRepository) {
    this.categoryRepository = categoryRepository;
}

public Category createCategory(Category category) {
    // vérifier doublon sur name
    // categoryRepository.findByName(category.getName())...
    return categoryRepository.save(category);
}

public List<Category> getAllCategories() {
    return categoryRepository.findAll();
}

public Category getCategoryById(Long id) {
    return categoryRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Catégorie introuvable (ID
}

public Category updateCategory(Long id, Category updatedCategory) {
    Category existing = getCategoryById(id);
    existing.setName(updatedCategory.getName());
    existing.setDescription(updatedCategory.getDescription());
    return categoryRepository.save(existing);
}

public void deleteCategory(Long id) {
    Category category = getCategoryById(id);
    categoryRepository.delete(category);
}
```

}

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.Customer; import com.example.inventorymanagement.re
import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transa
import com.example.inventorymanagement.exception.ResourceNotFoundException;

import java.util.List;

@Service @Transactional public class CustomerService {

```
private final CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {
    this.customerRepository = customerRepository;
}

public Customer createCustomer(Customer customer) {
    // Vérifier email unique
    customerRepository.findByEmail(customer.getEmail())
            .ifPresent(c -> {
                throw new RuntimeException("Email déjà utilisé : " + customer.getEmail()
            });
    return customerRepository.save(customer);
}

public List<Customer> getAllCustomers() {
    return customerRepository.findAll();
}

public Customer getCustomerById(Long id) {
    return customerRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Client introuvable (ID : "
}

public Customer updateCustomer(Long id, Customer updatedCust) {
    Customer existing = getCustomerById(id);

    // Vérifier si on change l'email
    if (!existing.getEmail().equals(updatedCust.getEmail())) {
        customerRepository.findByEmail(updatedCust.getEmail())
                .ifPresent(c -> {
                    throw new RuntimeException("Email déjà utilisé : " + updatedCust.get
                });
    }

    existing.setName(updatedCust.getName());
    existing.setEmail(updatedCust.getEmail());
    existing.setAddress(updatedCust.getAddress());
    return customerRepository.save(existing);
}

public void deleteCustomer(Long id) {
    Customer cust = getCustomerById(id);
    customerRepository.delete(cust);
}
```

}

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.Location; import com.example.inventorymanagement.rep
import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transa
import com.example.inventorymanagement.exception.ResourceNotFoundException;

import java.util.List;

@Service @Transactional public class LocationService {

```
private final LocationRepository locationRepository;

public LocationService(LocationRepository locationRepository) {
    this.locationRepository = locationRepository;
}

public Location createLocation(Location location) {
    // Si vous avez besoin de vérifier l'unicité, vous pourriez :
    // locationRepository.findByName(location.getName()).ifPresent(loc -> {
    //     throw new DuplicateException("Nom d'emplacement déjà utilisé : " + location.g
    // });
    return locationRepository.save(location);
}

public List<Location> getAllLocations() {
    return locationRepository.findAll();
}

public Location getLocationById(Long id) {
    return locationRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Emplacement introuvable (I
}

public Location updateLocation(Long id, Location updatedLoc) {
    Location existing = getLocationById(id);
    existing.setName(updatedLoc.getName());
    existing.setDescription(updatedLoc.getDescription());
    return locationRepository.save(existing);
}

public void deleteLocation(Long id) {
    Location loc = getLocationById(id);
    locationRepository.delete(loc);
}
```

}

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.Order; import com.example.inventorymanagement.repos
import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transa
import com.example.inventorymanagement.exception.ResourceNotFoundException;

import java.util.List;

@Service @Transactional public class OrderService {

```
private final OrderRepository orderRepository;
private final StockService stockService; // si on veut mettre à jour le stock

public OrderService(OrderRepository orderRepository, StockService stockService) {
    this.orderRepository = orderRepository;
    this.stockService = stockService;
}

public Order createOrder(Order order) {
    // Ex: si c'est une commande de type "SALE" (vente), on peut décrémenter le stock
    if ("SALE".equalsIgnoreCase(order.getType())) {
        // stockService.decreaseStock(order.getProduct(), locationId, order.getQuantity(
        // ou un autre mécanisme pour sélectionner l'emplacement
    }
    order.setStatus("EN_COURS");
    return orderRepository.save(order);
}

public List<Order> getAllOrders() {
    return orderRepository.findAll();
}

public Order getOrderById(Long id) {
    return orderRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Commande introuvable (ID :
}

public Order updateOrder(Long id, Order updatedOrder) {
    Order existing = getOrderById(id);
    existing.setType(updatedOrder.getType());
    existing.setQuantity(updatedOrder.getQuantity());
    existing.setStatus(updatedOrder.getStatus());
    existing.setOrderDate(updatedOrder.getOrderDate());
    existing.setDeliveryDate(updatedOrder.getDeliveryDate());
    existing.setCustomer(updatedOrder.getCustomer());
    existing.setSupplier(updatedOrder.getSupplier());
    existing.setProduct(updatedOrder.getProduct());
    return orderRepository.save(existing);
}

public void deleteOrder(Long id) {
    Order ord = getOrderById(id);
    orderRepository.delete(ord);
}

/**
 * Exemple : changer le statut et déclencher maj stock si nécessaire.
 */
public Order updateOrderStatus(Long id, String newStatus) {
    Order existing = getOrderById(id);
    existing.setStatus(newStatus);
    // Si newStatus == "LIVRE" et type == "PURCHASE", on peut incrémenter le stock, etc.
    return orderRepository.save(existing);
}
```

}

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.Product; import com.example.inventorymanagement.rep
import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transa
import com.example.inventorymanagement.exception.ResourceNotFoundException;

import java.util.List;

@Service @Transactional public class ProductService {

```
private final ProductRepository productRepository;

// Constructeur
public ProductService(ProductRepository productRepository) {
    this.productRepository = productRepository;
}

/**
 * Crée un nouveau produit en vérifiant l'unicité du SKU.
 */
public Product createProduct(Product product) {
    // Vérification du SKU
    productRepository.findBySku(product.getSku())
            .ifPresent(existing -> {
                throw new RuntimeException("SKU déjà existant : " + product.getSku());
            });
    return productRepository.save(product);
}
```

```
/**
 * Récupère la liste de tous les produits.
 */
public List<Product> getAllProducts() {
    return productRepository.findAll();
}

/**
 * Récupère un produit par son ID, ou lance une exception si introuvable.
 */
public Product getProductById(Long id) {
    return productRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Produit introuvable (ID :
}

/**
 * Met à jour un produit.
 */
public Product updateProduct(Long id, Product updatedProduct) {
    Product existingProduct = getProductById(id);

    // Si on change le SKU, vérifier l'unicité
    if (!existingProduct.getSku().equals(updatedProduct.getSku())) {
        productRepository.findBySku(updatedProduct.getSku())
                .ifPresent(other -> {
                    throw new RuntimeException("SKU déjà existant : " + updatedProduct.g
                });
    }

    // Mettre à jour les champs
    existingProduct.setName(updatedProduct.getName());
    existingProduct.setSku(updatedProduct.getSku());
    existingProduct.setDescription(updatedProduct.getDescription());
    existingProduct.setBarcode(updatedProduct.getBarcode());
    existingProduct.setQrCode(updatedProduct.getQrCode());
    existingProduct.setPhotoUrl(updatedProduct.getPhotoUrl());
    existingProduct.setCategory(updatedProduct.getCategory());

    return productRepository.save(existingProduct);
}

/**
 * Supprime un produit par ID.
 */
```

```
public void deleteProduct(Long id) {
    Product product = getProductById(id); // lève ResourceNotFoundException si non trouv
    productRepository.delete(product);
}
```

}

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.StockMovement; import com.example.inventorymanagem
import com.example.inventorymanagement.repository.StockMovementRepository; import
org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transactional

import java.time.LocalDateTime; import java.util.List;

@Service @Transactional public class StockMovementService {

```
private final StockMovementRepository stockMovementRepository;
private final StockService stockService; // pour décrémenter/incrémenter le stock

public StockMovementService(StockMovementRepository stockMovementRepository, StockServic
    this.stockMovementRepository = stockMovementRepository;
    this.stockService = stockService;
}

public StockMovement createMovement(StockMovement movement) {
    movement.setMovementDate(LocalDateTime.now());

    // Logique : si type == "IN", on incrémente le stock
    // si type == "OUT", on décrémente
    // si type == "TRANSFER", on décrémente fromLocation et incrémente toLocation
    switch (movement.getType().toUpperCase()) {
        case "IN":
            // stockService.increaseStock(movement.getProduct(), movement.getToLocation(
            break;
        case "OUT":
            // stockService.decreaseStock(movement.getProduct(), movement.getFromLocatio
            break;
        case "TRANSFER":
            // stockService.decreaseStock(movement.getProduct(), movement.getFromLocatio
            // stockService.increaseStock(movement.getProduct(), movement.getToLocation(
            break;
    }

    return stockMovementRepository.save(movement);
}

public List<StockMovement> getAllMovements() {
    return stockMovementRepository.findAll();
}

public StockMovement getMovementById(Long id) {
    return stockMovementRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Mouvement introuvable (ID
}

public void deleteMovement(Long id) {
    StockMovement mov = getMovementById(id);
    stockMovementRepository.delete(mov);
}

// Vous pouvez aussi implémenter "updateMovement" si nécessaire,
// mais souvent, on évite de modifier un mouvement historique.
```

}

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.Product; import com.example.inventorymanagement.ent
import com.example.inventorymanagement.repository.StockRepository; import org.springframework.stere
import org.springframework.transaction.annotation.Transactional; import com.example.inventorymanage

import java.util.List;

@Service @Transactional public class StockService {

```
private final StockRepository stockRepository;

public StockService(StockRepository stockRepository) {
    this.stockRepository = stockRepository;
}

/**
 * Crée un nouvel enregistrement de stock pour un produit et un emplacement donnés.
 */
public Stock createStock(Stock stock) {
    // Exemple : Vérifier si un Stock existe déjà pour (productId, locationId) pour évit
    // stockRepository.findByProductIdAndLocationId(stock.getProduct().getId(), stock.ge
    //         .ifPresent(s -> {
    //             throw new DuplicateException("Un stock pour ce produit et cet emplacem
    //         });
    if (stock.getQuantity() < 0) {
        throw new RuntimeException("La quantité ne peut pas être négative.");
    }
    return stockRepository.save(stock);
}
```

```
/**
 * Retourne la liste de tous les stocks.
 */
public List<Stock> getAllStocks() {
    return stockRepository.findAll();
}

/**
 * Retourne un stock par ID.
 */
public Stock getStockById(Long id) {
    return stockRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Stock introuvable (ID : "
}

/**
 * Met à jour la quantité, le min/max, etc.
 */
public Stock updateStock(Long id, Stock updatedStock) {
    Stock existing = getStockById(id);
    existing.setQuantity(updatedStock.getQuantity());
    existing.setMinQuantity(updatedStock.getMinQuantity());
    existing.setMaxQuantity(updatedStock.getMaxQuantity());
    existing.setLocation(updatedStock.getLocation());
    existing.setProduct(updatedStock.getProduct());
    return stockRepository.save(existing);
}

/**
 * Supprime un stock par ID.
 */
public void deleteStock(Long id) {
    Stock stock = getStockById(id);
    stockRepository.delete(stock);
}

/**
 * Exemple de méthode de logique métier : vérifier et décrémenter le stock.
 */
public void decreaseStock(Product product, Long locationId, int qty) {
    // find stock par (productId, locationId) -> custom query
    // Ex : stockRepository.findByProductIdAndLocationId(product.getId(), locationId)
    // ...
    // Vérifier si quantity >= qty, décrémenter, save
```

}

}

package com.example.inventorymanagement.service;

import com.example.inventorymanagement.entity.Supplier; import com.example.inventorymanagement.rep
import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transa
import com.example.inventorymanagement.exception.ResourceNotFoundException;

import java.util.List;

@Service @Transactional public class SupplierService {

```
private final SupplierRepository supplierRepository;

public SupplierService(SupplierRepository supplierRepository) {
    this.supplierRepository = supplierRepository;
}

public Supplier createSupplier(Supplier supplier) {
    // Exemple : vérifier si le nom existe déjà
    // Optional<Supplier> existing = supplierRepository.findByName(supplier.getName());
    // ...
    return supplierRepository.save(supplier);
}

public List<Supplier> getAllSuppliers() {
    return supplierRepository.findAll();
}

public Supplier getSupplierById(Long id) {
    return supplierRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Fournisseur introuvable (I
}

public Supplier updateSupplier(Long id, Supplier updatedSup) {
    Supplier existing = getSupplierById(id);
    existing.setName(updatedSup.getName());
    existing.setContactInfo(updatedSup.getContactInfo());
    return supplierRepository.save(existing);
}

public void deleteSupplier(Long id) {
    Supplier sup = getSupplierById(id);
    supplierRepository.delete(sup);
}
```

}

import io.jsonwebtoken.SignatureAlgorithm; import io.jsonwebtoken.security.Keys; import javax.crypto.SecretKey;

public class JwtUtil { public static final SecretKey JWT_SECRET = Keys.secretKeyFor(SignatureAlgorith
}

spring.application.name=inventory-management spring.datasource.url=jdbc:mysql://localhost:3306/inven
spring.datasource.username=root spring.datasource.password=nashero

spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true

Permet d'activer la validation (optionnel en Spring Boot 3)

spring.mvc.pathmatch.matching-strategy=ant_path_matcher

package com.example.inventorymanagement;

import org.junit.jupiter.api.Test; import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest class InventoryManagementApplicationTests {

```
@Test
void contextLoads() {
}
```

}

package com.example.inventorymanagement.controller;

import com.example.inventorymanagement.security.AuthService; import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.au
import org.springframework.boot.test.context.SpringBootTest; import org.springframework.http.MediaTyp
import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@SpringBootTest @AutoConfigureMockMvc class AuthControllerTest {

```
@Autowired
private MockMvc mockMvc;

@Autowired
private AuthService authService;

@Test
void shouldReturnTokenForValidLogin() throws Exception {
    String username = "testUser";
    String password = "password";
    String requestBody = """
            {
                "username": "%s",
                "password": "%s"
            }
            """.formatted(username, password);

    mockMvc.perform(post("/auth/login")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(requestBody))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.token").exists());
}

@Test
void shouldRejectInvalidLogin() throws Exception {
    String requestBody = """
            {
                "username": "wrongUser",
                "password": "wrongPassword"
            }
            """;

    mockMvc.perform(post("/auth/login")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(requestBody))
            .andExpect(status().isUnauthorized());
}
```

}

package com.example.inventorymanagement.controller;

import com.example.inventorymanagement.security.AuthService; import org.junit.jupiter.api.Test; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.au import org.springframework.boot.test.context.SpringBootTest; import org.springframework.http.HttpHead import org.springframework.http.MediaType; import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get; import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@SpringBootTest @AutoConfigureMockMvc class ProtectedRoutesTest {

```
@Autowired
private MockMvc mockMvc;

@Autowired
private AuthService authService;
```

```
@Test
void shouldAllowAccessWithValidToken() throws Exception {
    // Utilise generateToken(String username)
    String validToken = authService.generateToken("testUser");

    mockMvc.perform(get("/protected-endpoint")
                    .header(HttpHeaders.AUTHORIZATION, "Bearer " + validToken)
                    .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk());
}

@Test
void shouldRejectAccessWithoutToken() throws Exception {
    mockMvc.perform(get("/protected-endpoint"))
            .andExpect(status().isUnauthorized());
}

@Test
void shouldRejectAccessWithInvalidToken() throws Exception {
    mockMvc.perform(get("/protected-endpoint")
                    .header(HttpHeaders.AUTHORIZATION, "Bearer invalidToken"))
            .andExpect(status().isForbidden());
}
```

}

package com.example.inventorymanagement.security;

import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import org.mockito.Mock; import org.mockito.MockitoAnnotations; import org.springframework.security.authenti
import org.springframework.security.core.userdetails.UserDetailsService; import org.springframework.secur

import static org.junit.jupiter.api.Assertions.assertFalse;

class AuthServiceTest {

```
@Mock
private AuthenticationManager authenticationManager;
@Mock
private UserDetailsService userDetailsService;
@Mock
private PasswordEncoder passwordEncoder;

private AuthService authService;

@BeforeEach
void setUp() {
    // Initialise les mocks
    MockitoAnnotations.openMocks(this);
    // Injecte les mocks dans le constructeur
    authService = new AuthService(authenticationManager, userDetailsService, passwordEnc
}

@Test
void shouldReturnFalseForInvalidToken() {
    assertFalse(authService.validateToken("invalidToken"));
}
```

}

spring.application.name=inventory-management spring.datasource.url=jdbc:mysql://localhost:3306/inven
spring.datasource.username=root spring.datasource.password=nashero

spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true

Permet d'activer la validation (optionnel en Spring Boot 3)

```
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
```