



# Rapport Complet

## *Projet XML*

École : ENSA Tanger

Professeur : LACHKAR Abdelmonaime

Réalisé par : ALLAM ELARBI, AL AZAMI TAREK

# Table des matières

<b>1</b>	<b>Introduction : Contexte et Technologies Utilisées</b>	<b>3</b>
1.1	Projet XML et Choix Technologiques . . . . .	3
<b>2</b>	<b>Présentation Générale du Frontend</b>	<b>4</b>
<b>3</b>	<b>Structure et Organisation du Frontend</b>	<b>5</b>
3.1	Arborescence du Projet (React) . . . . .	5
<b>4</b>	<b>Fonctionnalités Principales (Frontend)</b>	<b>7</b>
4.1	Accueil (Home) et Filtrage des Produits . . . . .	7
4.2	Panier (Cart) . . . . .	8
4.3	Confirmation de Commande (Checkout) . . . . .	9
4.3.1	Interface de confirmation . . . . .	9
4.3.2	Facture générée automatiquement . . . . .	9
4.4	Espace d'Administration (AdminPanel) . . . . .	10
4.5	Export Excel . . . . .	15
4.5.1	Bouton d'export Excel . . . . .	16
4.5.2	Résultat de l'export Excel . . . . .	16
<b>5</b>	<b>Interaction avec le Backend et Méthodes HTTP</b>	<b>17</b>
<b>6</b>	<b>Résumé du Fonctionnement Global (Frontend)</b>	<b>18</b>
6.1	Cycle de Vie Simplifié . . . . .	18
<b>7</b>	<b>Architecture du Projet (Backend)</b>	<b>19</b>
7.1	UML (Diagramme de Classes) . . . . .	19
7.2	Architecture en Couches . . . . .	20
7.2.1	Couche Entity (Entités) . . . . .	20
7.2.2	Couche Util (Utilitaires) . . . . .	21
7.2.3	Couche Repository (Persistance) . . . . .	23
7.2.4	Couche Service (Logique Métier) . . . . .	24
7.2.5	Couche Controller (Gestion des Requêtes) . . . . .	25

<b>8 Manipulation des Données XML</b>	<b>26</b>
8.1 Stockage des Produits en XML . . . . .	26
8.2 Validation des Fichiers XML avec XSD . . . . .	26
8.3 Stockage et Validation des Users en XML / XSD . . . . .	27
8.4 Stockage et Validation des Commandes, Factures, Lignes de Commande, Catégories . . . . .	28
<b>9 Génération de Fichiers depuis XML (HTML, Excel, PDF)</b>	<b>30</b>
9.1 Rapports HTML via XSLT . . . . .	30
9.2 Factures PDF . . . . .	33
9.3 Génération des Fichiers Excel . . . . .	34
9.4 Recherche et Filtrage de Produits via XPath . . . . .	35
9.4.1 Recherche de produits avec XPath . . . . .	36
9.5 Conclusion . . . . .	37

# Chapitre 1

## Introduction : Contexte et Technologies Utilisées

### 1.1 Projet XML et Choix Technologiques

Ce projet vise à développer une application e-commerce en utilisant **Spring Boot** et **XML** comme format principal pour le stockage des données. Contrairement aux solutions classiques basées sur une base de données relationnelle, nous avons choisi d'utiliser des **fichiers XML** pour représenter les informations des produits, clients et commandes. Le projet exploite également :



- **XSD** pour la validation des fichiers XML,
- **XSLT** pour les transformations vers des formats lisibles comme HTML et PDF,
- **XPath** pour la recherche et l'extraction d'informations,
- **Java** et dépendances Spring : intégration avec Spring Boot.

Grâce à cette approche, nous pouvons bénéficier d'une structure de données *flexible* (XML), d'une *validation robuste* (XSD) et de transformations (XSLT) pour une présentation adaptée (HTML, PDF, etc.).

## Chapitre 2

# Présentation Générale du Frontend

Dans le cadre du **Projet XML**, nous avons également développé une application **e-commerce** dont la partie **frontend** est gérée en **React**. Le but de ce rapport (côté frontend) est de présenter de façon **complète et professionnelle** :

- L'**architecture** du code et la **structure** des répertoires côté frontend.
- Les principales **fonctionnalités** (Administration, Ajout au Panier, Confirmation de Commande, Export Excel, etc.).
- L'interaction avec le **backend** via des **endpoints** REST, illustrée avec une coloration et un style adaptés aux méthodes **GET** ..., **POST** ..., **PUT** ..., **DELETE** ....
- Des **captures d'écran** (images) plutôt que du code brut, afin de préserver une présentation claire et concise.

Le **contexte** global est le suivant : nous gérons un site e-commerce proposant un catalogue de produits, un panier, un système de commandes et un module d'administration avec **CRUD** et **exports Excel**.

# Chapitre 3

## Structure et Organisation du Frontend

### 3.1 Arborescence du Projet (React)

Le **frontend** est organisé suivant une arborescence simple (fichiers React, contextes, pages, etc.). Ci-dessous, une représentation illustrée :

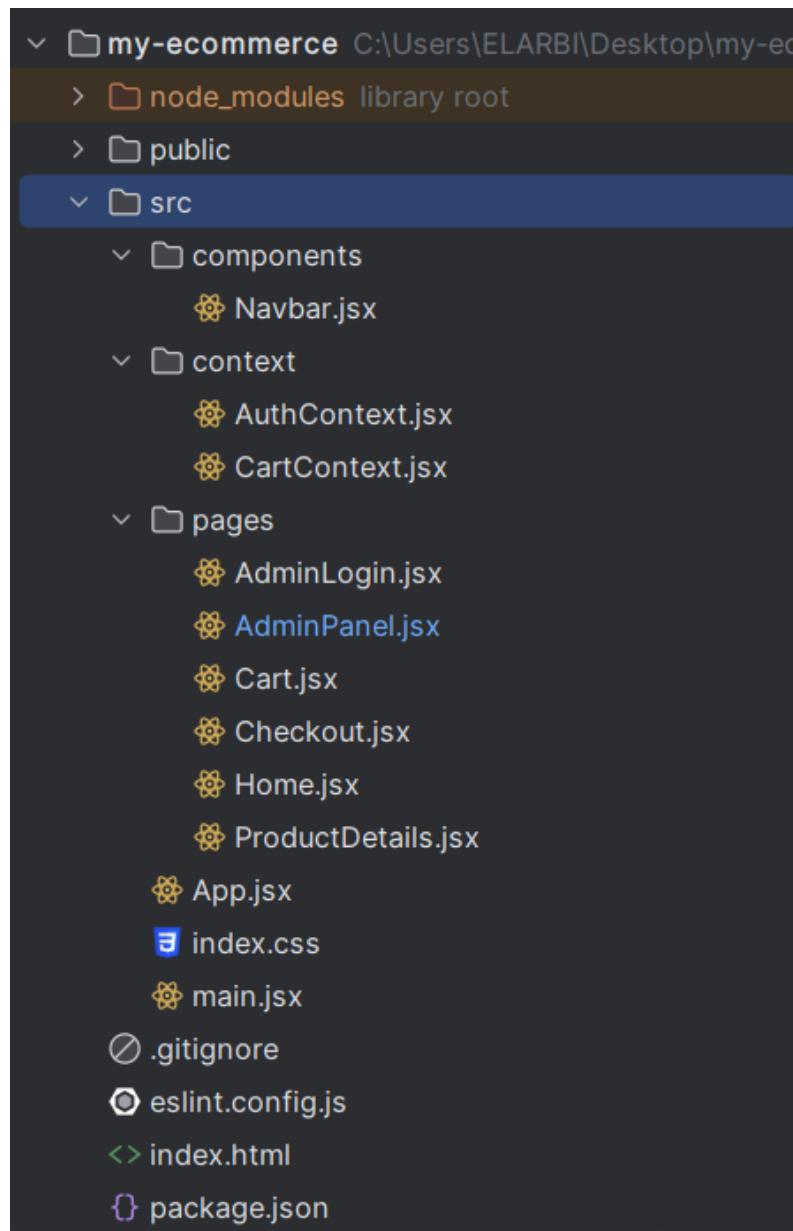


FIGURE 3.1 – Arborescence simplifiée du projet React (Frontend)

Le dossier `src/` contient la logique essentielle de l’application. Plus précisément :

- `public/` : contient les ressources statiques (icônes, images, etc.).
- `src/` : dossier principal avec la logique React (pages, composants, contextes).
  - `components/` : composants réutilisables (`Navbar`, etc.).
  - `context/` : `CartContext` pour le panier, `AuthContext` pour l’authentification Admin.
- `pages/` : `Home`, `Cart`, `Checkout`, `AdminPanel`, etc.
- `App.jsx` : composant racine qui gère les routes et la mise en page globale.
- `main.jsx` : point d’entrée pour `ReactDOM`.
- `.gitignore` : spécifie les fichiers à ignorer par Git.

# Chapitre 4

## Fonctionnalités Principales (Frontend)

### 4.1 Accueil (Home) et Filtrage des Produits

La page Home permet de présenter l'ensemble des produits retournés par `GET /api/products/all`. Elle fournit également un moteur de recherche ainsi que des filtres par catégorie et par prix. Un bouton “*Add to Cart*” permet l'ajout direct d'un produit au panier.

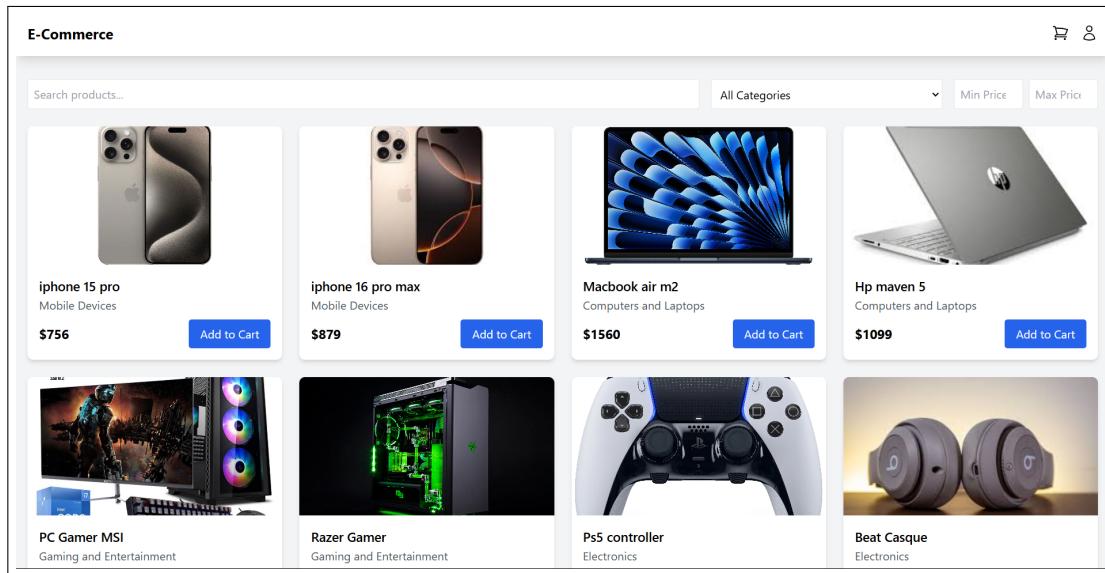


FIGURE 4.1 – Vue générale de la page Home (liste de produits)

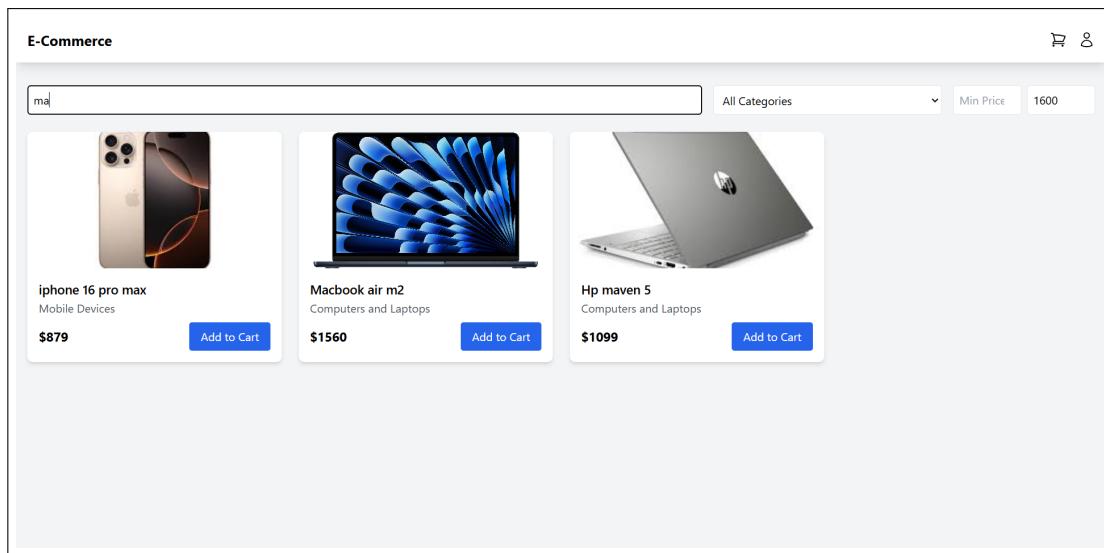


FIGURE 4.2 – Home : moteur de recherche et filtrage

## 4.2 Panier (Cart)

Chaque fois qu'un utilisateur clique sur “Add to Cart”, le `CartContext` enregistre l'item. Sur la page `Cart`, il est possible :

- D'afficher tous les produits ajoutés.
- De modifier la quantité d'un article.
- De supprimer un article.
- De visualiser le **total** du panier.

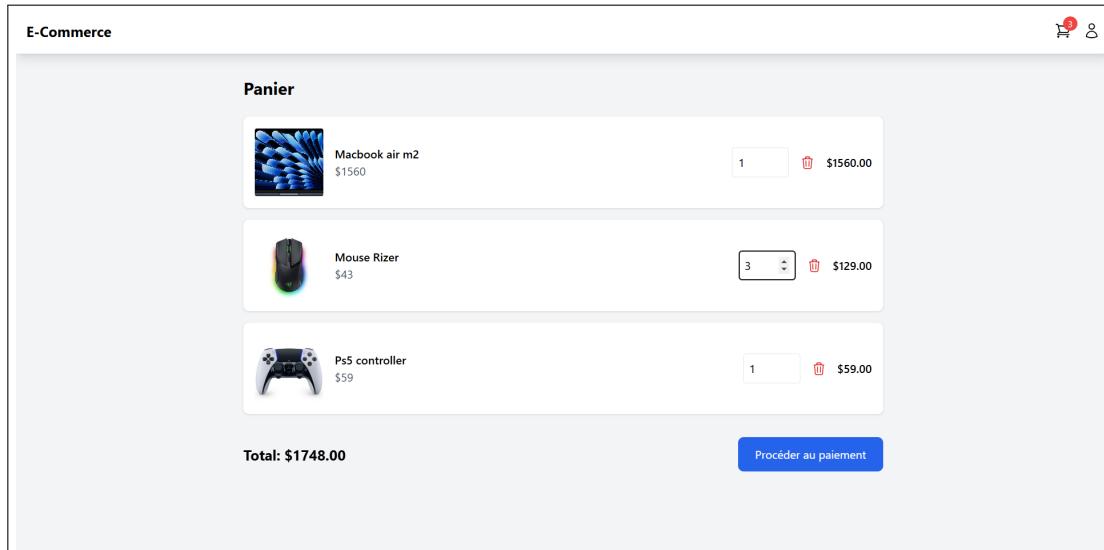


FIGURE 4.3 – Interface du panier avec modification des quantités

## 4.3 Confirmation de Commande (Checkout)

Pour finaliser la commande, l'utilisateur renseigne un formulaire permettant de collecter ses informations (nom, email, adresse, etc.). Un appel **POST** /api/commands/create est alors émis vers le backend.

### 4.3.1 Interface de confirmation

The screenshot shows a web-based e-commerce checkout interface. At the top left is the 'E-Commerce' logo. On the right side, there are icons for a shopping cart with a red notification bubble containing the number '9' and a user profile. The main area is titled 'Checkout'. It contains several input fields for customer information: 'Username' (Houssam\_M), 'Email' (Houssam@gmail.com), 'First Name' (Houssam), 'Last Name' (Marouani), 'Phone Number' (0612356498), and 'Address' (Rabat\_Riyad). To the right of these fields is an 'Order Summary' table:

	Total
Macbook air m2 x 1	\$1560.00
Mouse Rizer x 3	\$129.00
Ps5 controller x 1	\$59.00
<b>Total</b>	<b>\$1748.00</b>

At the bottom center is a large blue button labeled 'Place Order'.

FIGURE 4.4 – Formulaire Checkout (informations client)

### 4.3.2 Facture générée automatiquement

Une fois l'appel **POST** /api/commands/create validé par le serveur, un PDF est retourné (récapitulatif de la commande). Le client le télécharge automatiquement.

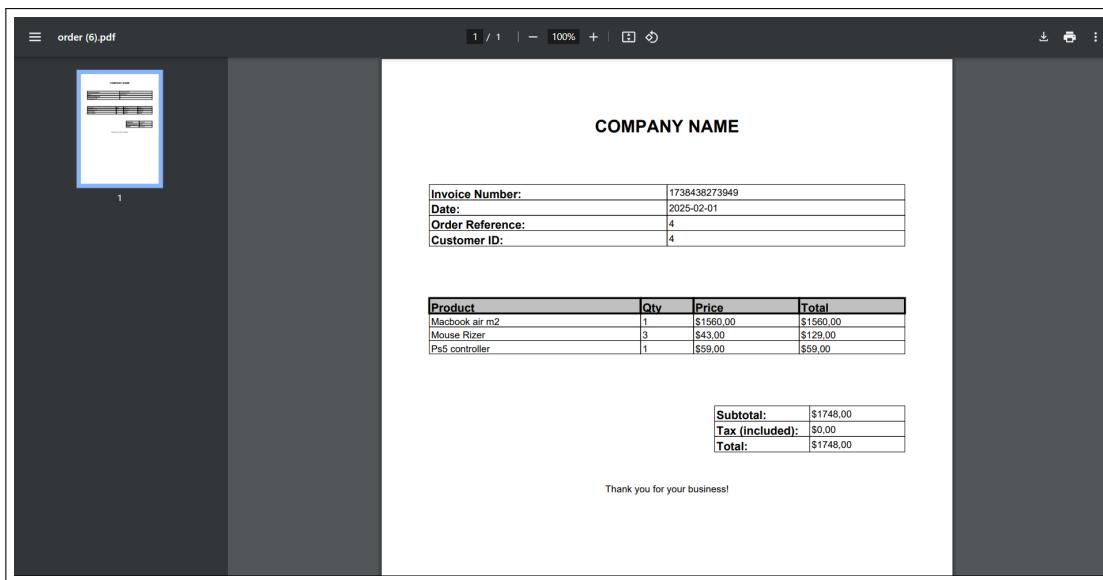


FIGURE 4.5 – Exemple de facture PDF générée

## 4.4 Espace d'Administration (AdminPanel)

Seuls les administrateurs connectés (via AdminLogin) peuvent accéder à l'AdminPanel.  
Cet espace leur donne des options de gestion :

- **Produits (CRUD)** : Création, Modification, Suppression, etc.
- **Commandes** : Liste, changement de statut (*Pending*, *Shipped*, *Delivered*, etc.).
- **Utilisateurs** : Affichage, modification, suppression.
- **Rapports** : Visualisation des statistiques, exports, etc.

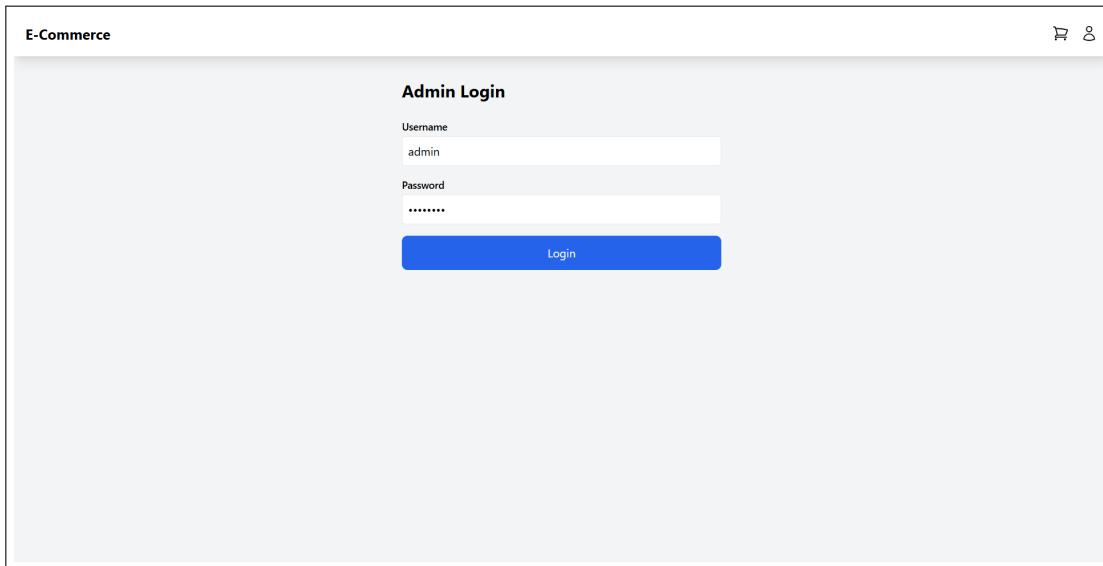


FIGURE 4.6 – Page d'authentification admin

E-Commerce

Admin Panel

Products Orders Users Rapport

Add New Product

Title	Category
	Select a category
Price	Stock Quantity
Description	
Image URL	
<input type="button" value="Add Product"/>	

FIGURE 4.7 – AdminPanel : gestion des produits

Products List

Export Excel

Products List

	<b>iphone 15 pro</b> \$756 - 8 in stock - Mobile Devices 1To , Toute les couleurs	<a href="#">Edit</a> <a href="#">Delete</a>
	<b>iphone 16 pro max</b> \$879 - 21 in stock - Mobile Devices 512Go , Golden	<a href="#">Edit</a> <a href="#">Delete</a>
	<b>Macbook air m2</b> \$1560 - 6 in stock - Computers and Laptops laptop high performance	<a href="#">Edit</a> <a href="#">Delete</a>
	<b>Hp maven 5</b> \$1099 - 15 in stock - Computers and Laptops Laptop Hp 1To nvem , 16Go ram , Nvidia GTX 1060Ti	<a href="#">Edit</a> <a href="#">Delete</a>
	<b>PC Gamer MSI</b> \$1399 - 8 in stock - Gaming and Entertainment PC complet	<a href="#">Edit</a> <a href="#">Delete</a>

FIGURE 4.8 – Ajout / modification de produit (CRUD)

**Admin Panel**

Products Orders Users Rapport

Orders List Export Excel

All Pending Shipped Delivered Cancelled

**Order #1**  
Date: 2025-02-01 | Status: Shipped  
Shipped Show Details Show User

**Order #2**  
Date: 2025-02-01 | Status: Delivered  
Delivered Show Details Show User

**Order #3**  
Date: 2025-02-01 | Status: Cancelled  
Cancelled Show Details Show User

**Order #4**  
Date: 2025-02-01 | Status: Pending  
Pending Show Details Show User

FIGURE 4.9 – Liste des commandes dans l'espace Admin

Products Orders Users Rapport

Orders List Export Excel

All Pending Shipped Delivered Cancelled

**Order #1**  
Date: 2025-02-01 | Status: Shipped  
Shipped Hide Details Hide User

**User Information:**  
Username: Hamid\_EL  
Email: Hamid@gmail.com  
Name: Hamid Elawarchani  
Phone: 0654987321  
Address: Tanger\_Dradeb

**Order Details:**

Product #3  
Macbook air m2  
laptop high performance  
\$1560 Computers and Laptops

FIGURE 4.10 – Détails d'une commande et changement de statut

**Add New User**

Username	Email
Halima_Ab	Halima@gmail.com
First Name	Last Name
Halima	Aboudi
Phone Number	
0612356479	
Address	
Casablanca_Sidi-Maarouf	
<input type="button" value="Add User"/>	

FIGURE 4.11 – Gestion des utilisateurs (AdminPanel)

**Users List**

		Export Excel
<b>Hamid Elawarchani</b> (Hamid_EL)	Hamid@gmail.com	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
0654987321		
Tanger_Dradeb		
<b>Saad Kharoub</b> (Saad_02)	Saad@gmail.com	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
0632659874		
Marrakech_daoudiat		
<b>Halima Aboudi</b> (Halima_Ab)	Halima@gmail.com	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
0612356479		
Casablanca_Sidi-Maarouf		
<b>Houssam Marouani</b> (Houssam_M)	Houssam@gmail.com	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
0612356498		
Dakar_Diour		

FIGURE 4.12 – Édition ou suppression d'un utilisateur

The screenshot shows a web-based reporting interface. At the top, there are four navigation tabs: 'Products', 'Orders', 'Users', and 'Rapport' (Report). The 'Rapport' tab is highlighted with a blue background. Below the tabs, the title 'Monthly Report' is displayed in blue, along with a 'Generate Report' button. A 'Close Report' link is also present. Under the report title, there is a section titled 'Summary' containing three statistics: 'Total Products:10', 'Total Items in Stock:152', and 'Low Stock Items (Quantity ≤ 10):4'. Below the summary, there is a section titled 'Product Inventory' with a table. The table has columns for 'ID', 'Title', 'Category', 'Price (\$)', and 'Stock'. The data is as follows:

ID	Title	Category	Price (\$)	Stock
9	Clavier RGB	Computer Components and Accessories	29.00	16
10	Mouse Rizer	Computer Components and Accessories	43.00	24
3	Macbook air m2	Computers and Laptops	1,560.00	6
4	Hp maven 5	Computers and Laptops	1,099.00	15
7	Ps5 controller	Electronics	59.00	35

FIGURE 4.13 – Rapports : vue globale des indicateurs

Summary				
Total Products:10 Total Items in Stock:152 Low Stock Items (Quantity ≤ 10):4				
Product Inventory				
ID	Title	Category	Price (\$)	Stock
9	Clavier RGB	Computer Components and Accessories	29.00	16
10	Mouse Rizer	Computer Components and Accessories	43.00	24
3	Macbook air m2	Computers and Laptops	1,560.00	6
4	Hp maven 5	Computers and Laptops	1,099.00	15
7	Ps5 controller	Electronics	59.00	35
8	Beat Casque	Electronics	199.00	7
5	PC Gamer MSI	Gaming and Entertainment	1,399.00	8
6	Razer Gamer	Gaming and Entertainment	3,099.00	12
2	iphone 15 pro	Mobile Devices	756.00	8
1	iphone 16 pro max	Mobile Devices	879.00	21

Category Summary		
Category	Number of Products	Total Stock
Mobile Devices	2	29
Mobile Devices	2	29
Computers and Laptops	2	21
Gaming and Entertainment	2	20
Electronics	2	42
Computer Components and Accessories	2	40

FIGURE 4.14 – Exemple de rapport détaillé

## 4.5 Export Excel

À partir de l'AdminPanel, l'export Excel ([GET /api/export/{type}/excel](#)) est possible pour différents types de données (Produits, Commandes, Utilisateurs).

#### 4.5.1 Bouton d'export Excel

Users List					
<b>Hamid Elawarchani</b> (Hamid_EL) Hamid@gmail.com 0654987321 Tanger_Dradeb					
<b>Saad Kharoub</b> (Saad_02) Saad@gmail.com 0632659874 Marrakech_daoudiat					
<b>Halima Aboudi</b> (Halima_Ab) Halima@gmail.com 0612356479					

FIGURE 4.15 – Bouton “Export Excel”

#### 4.5.2 Résultat de l'export Excel

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	<b>Id</b>	<b>Address</b>	<b>Phone Number</b>	<b>Username</b>	<b>First Name</b>	<b>Last Name</b>	<b>Email</b>												
2	1	Tanger_Dradeb	0654987321	Hamid_EL	Hamid	Elawarchani	Hamid@gmail.com												
3	2	Marrakech_daoudiat	0632659874	Saad_02	Saad	Kharoub	Saad@gmail.com												
4	3	Casablanca_Sidi-Maroud	0612356479	Halima_Ab	Halima	Aboudi	Halima@gmail.com												
5	4	Rabat_Riyad	0612356498	Houssam_M	Houssam	Marouani	Houssam@gmail.com												

FIGURE 4.16 – Exemple de fichier Excel exporté

# Chapitre 5

## Interaction avec le Backend et Méthodes HTTP

Le frontend communique avec le backend via des endpoints REST. Parmi les plus utilisés :

- **GET** /api/products/all pour charger les produits.
- **POST** /api/commands/create pour créer une commande.
- **GET** /api/export/{type}/excel pour exporter en Excel.
- **POST** /api/products, **PUT** /api/products, **DELETE** /api/products/{id} pour le CRUD des produits.

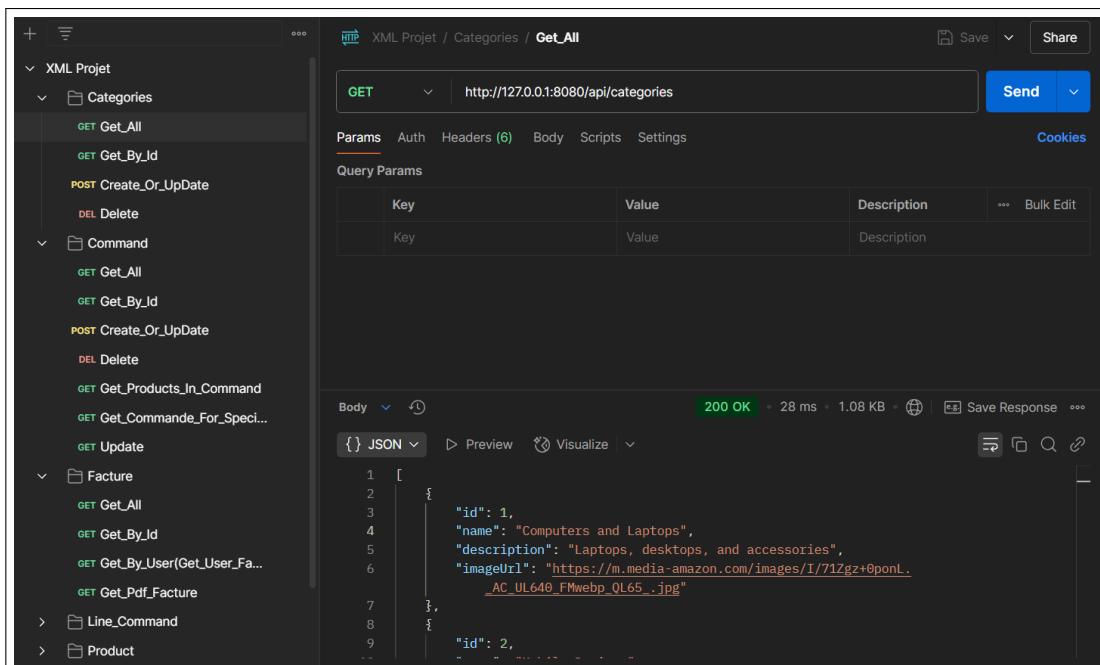


FIGURE 5.1 – Flux de requêtes/réponses entre frontend et backend

# Chapitre 6

## Résumé du Fonctionnement Global (Frontend)

### 6.1 Cycle de Vie Simplifié

1. **Chargement initial** : L'application (`App.jsx`) lance un `GET /api/products/all` et `GET /api/categories` pour récupérer le catalogue et les catégories.
2. **Navigation** : L'utilisateur explore Home, Cart, Checkout, AdminPanel, etc.
3. **Panier** : Chaque ajout au panier est géré par le `CartContext`.
4. **Commande** : Sur `Checkout`, `POST /api/commands/create` envoie la commande ; le serveur génère un PDF (facture) en réponse.
5. **Administration** : L'administrateur se connecte (`AdminLogin`), puis gère Produits, Utilisateurs, Commandes, Rapports, Exports Excel.

# Chapitre 7

## Architecture du Projet (Backend)

### 7.1 UML (Diagramme de Classes)

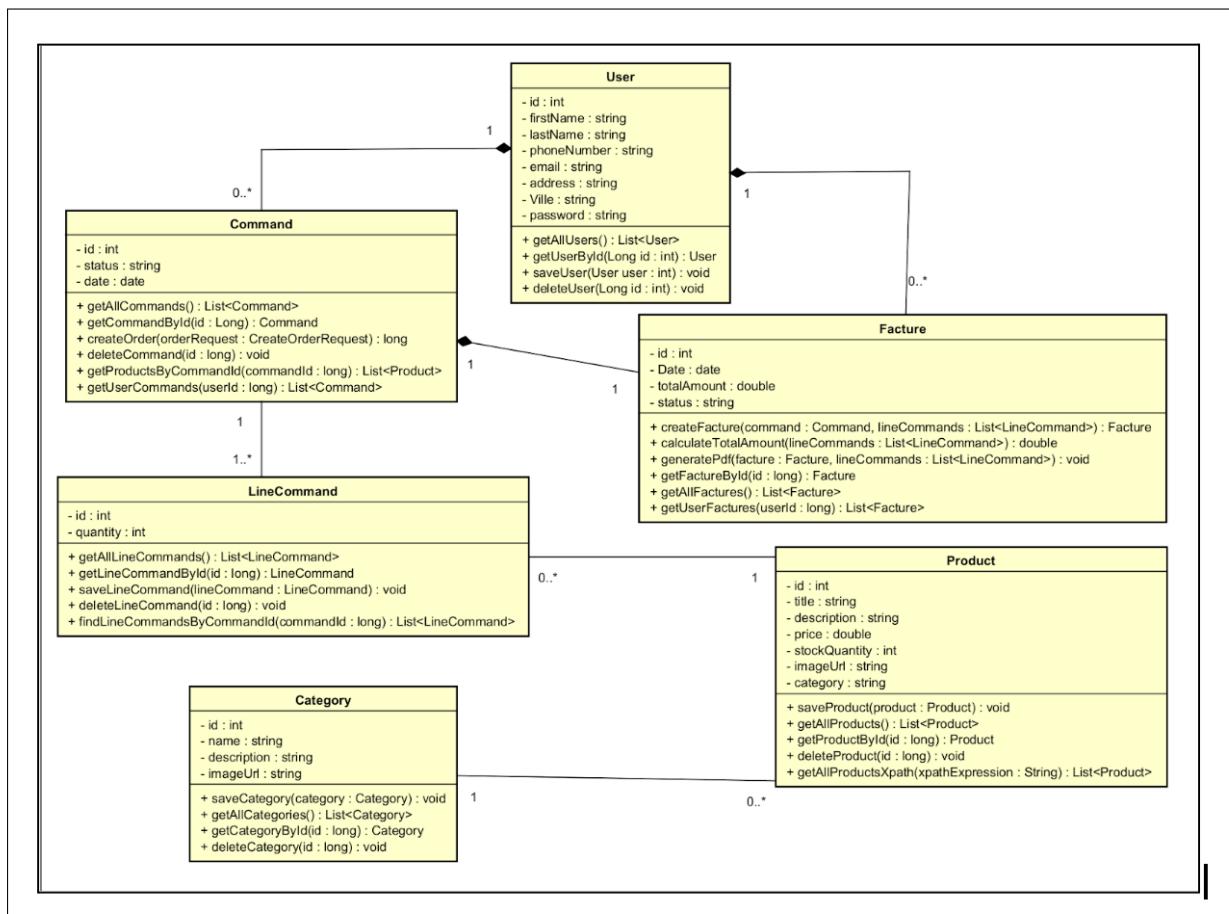
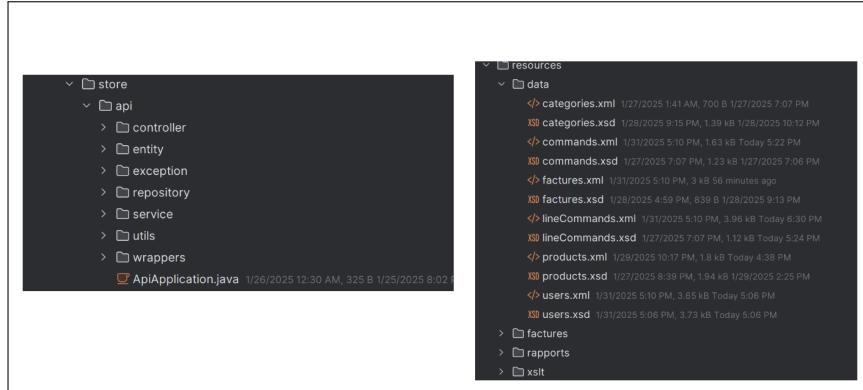


FIGURE 7.1 – Exemple de diagramme de classes UML (placeholder)

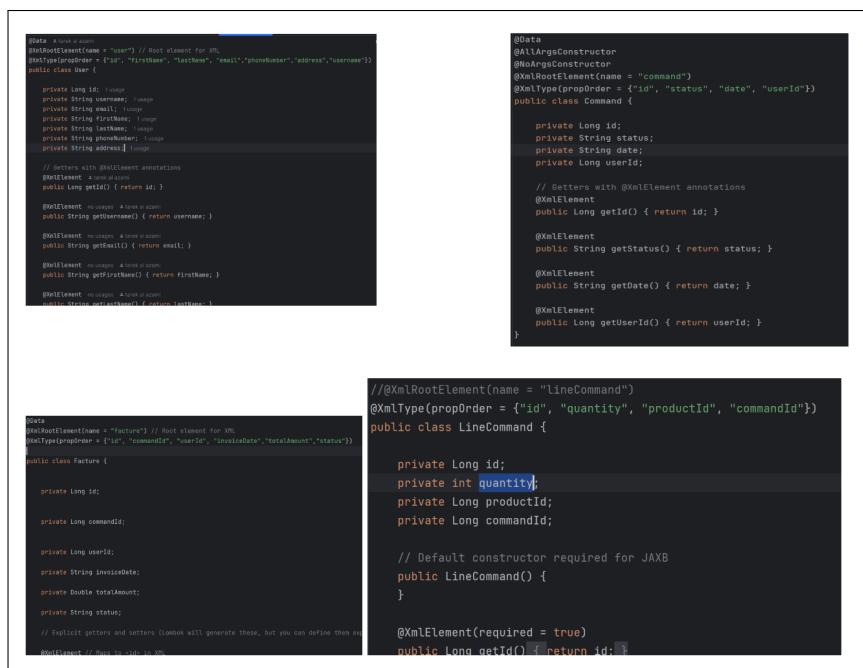
## 7.2 Architecture en Couches

Le projet suit une approche en couches, où chaque couche a un rôle bien défini pour assurer une séparation claire des responsabilités :



### 7.2.1 Couche Entity (Entités)

Rôle : Modéliser les objets métiers et les données utilisées dans l'application. Exemple :



```
@XmlRootElement(name = "product")
@XmlType(propOrder = {"id", "title", "description", "price", "stockQuantity", "imageUrl", "category"})

public class Product {

    private Long id;
    private String title;
    private String description;
    private Double price;
    private Integer stockQuantity;
    private String imageUrl;
    private String category; // Reference to the category by ID

    // Getters with @XmlElement annotations
    @XmlElement
    public Long getId() { return id; }

    @XmlElement
    public String getTitle() { return title; }

    ...
}
```

FIGURE 7.2 – Classe User : entité XML

La classe **User** est une entité représentant un utilisateur dans le système. Elle est conçue pour être sérialisée/désérialisée en XML via JAXB (`@XmlRootElement(name = "user")`).

### 7.2.2 Couche Util (Utilitaires)

**Rôle** : Fournir des outils transversaux comme la validation et la gestion des fichiers XML. Exemple :

```
public class XmlUtil {

    // Read XML with validation
    public static <T> T readXml(String xmlPath, Class<T> clazz, String xsdPath)
        throws JAXBException, SAXException, FileNotFoundException {

        File file = new File(xmlPath);
        if (!file.exists()) {
            throw new FileNotFoundException("XML file not found: " + xmlPath);
        }
        if (file.length() == 0) {
            throw new SAXParseException("XML file is empty: " + xmlPath, null);
        }

        JAXBContext context = JAXBContext.newInstance(clazz);
        Unmarshaller unmarshaller = context.createUnmarshaller();

        // Load XSD and enforce validation
        Schema schema = loadSchema(xsdPath);
        unmarshaller.setSchema(schema);

        // Add validation event handler
        unmarshaller.setEventHandler(handleEvent(event) -> {
            // Log validation errors
            System.err.println("Validation error: " + event.getMessage());
            return false; // Continue after validation errors
        });

        return clazz.cast(unmarshaller.unmarshal(file));
    }
}
```

FIGURE 7.3 – XmlUtil : gestion et validation XML

```
// Write XML with validation
public static <T> void writeXml(String xmlPath, T object, String xsdPath)
    throws JAXBException, SAXException {

    JAXBContext context = JAXBContext.newInstance(object.getClass());
    Marshaller marshaller = context.createMarshaller();

    // Load XSD and enforce validation
    Schema schema = loadSchema(xsdPath);
    marshaller.setSchema(schema);

    // Add validation event handler
    marshaller.setEventHandler(handleEvent(event) -> {
        // Log validation errors
        System.err.println("Validation error: " + event.getMessage());
        return false; // Continue after validation errors
    });

    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

    // First validate by marshalling to a StringWriter
    StringWriter sw = new StringWriter();
    marshaller.marshal(object, sw);

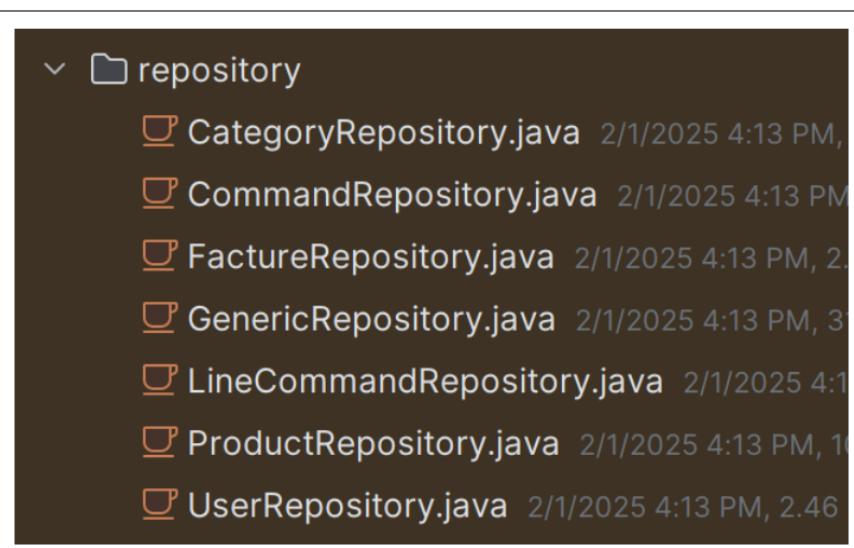
    // If validation passes, write to file
    marshaller.marshal(object, new File(xmlPath));
}
```

FIGURE 7.4 – Lecture et écriture de fichiers XML

`XmlUtil`, une classe dédiée à la lecture et l'écriture de fichiers XML avec validation par un schéma XSD.

### 7.2.3 Couche Repository (Persistance)

Rôle : Gérer les opérations de persistance (CRUD) sur les données XML. Exemple :



```
@Repository
public class UserRepository implements GenericRepository<User, Long>{

    private static final String USERS_FILE = "src/main/resources/data/users.xml";
    private static final String USERS_XSD = "src/main/resources/data/users.xsd"; // Path to XSD

    // Read all users from the XML file
    public List<User> findAll() throws Exception {...}

    // Find a user by ID
    public Optional<User> findById(Long id) throws Exception {...}

    public Optional<User> findByPhoneNumber(String phoneNumber) throws Exception {...}

    // Find a check user by phone number
    public Boolean checkUserExistByPhoneNumber(String phoneNumber) throws Exception {...}

    // Save a user (add or update)
    public void save(User user) throws Exception {...}

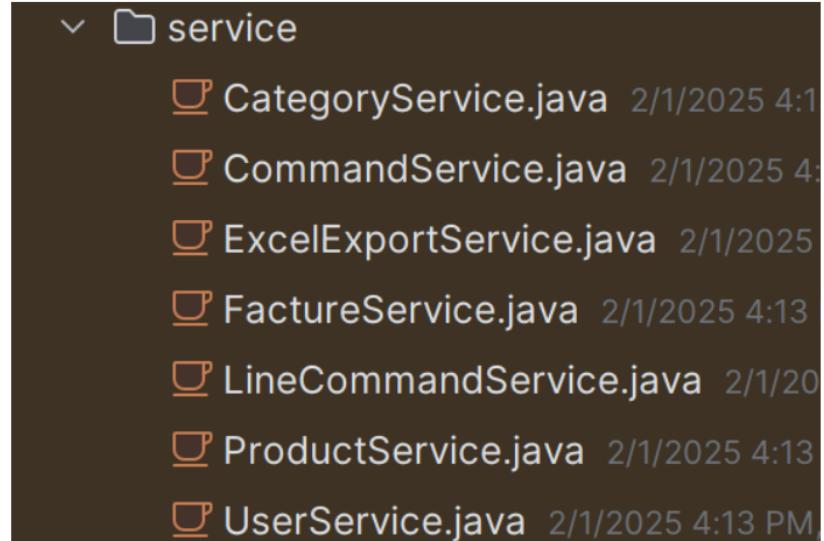
    // Delete a user by ID
    public void deleteById(Long id) throws Exception {...}

    //
}
```

UserRepository gère la persistance des utilisateurs en XML, assurant les opérations CRUD tout en validant les données via `users.xsd`.

#### 7.2.4 Couche Service (Logique Métier)

**Rôle :** Implémenter la logique métier et orchestrer les différentes opérations de l'application. Exemple :



```

public class CommandService {

    @Autowired
    private CommandRepository commandRepository;

    @Autowired
    private LineCommandRepository lineCommandRepository;

    @Autowired
    private ProductRepository productRepository;

    @Autowired
    private FactureService factureService;

    @Autowired
    private UserRepository userRepository;

    // Create new Order
    @PostMapping("/orders")
    public ResponseEntity<Order> createOrder(@RequestBody CreateOrderRequest orderRequest) {
        try {
            Long orderId = commandService.createOrder(orderRequest);
            return ResponseEntity.ok(orderId);
        } catch (Exception e) {
            return ResponseEntity.status(INTERNAL_SERVER_ERROR).build();
        }
    }

    // Delete a command by ID
    @DeleteMapping("/{id}")
    public void deleteCommand(@PathVariable Long id) throws Exception {
        commandService.deleteCommand(id);
    }

    // Get products for a specific command
    @GetMapping("/{commandId}/products")
    public List<Product> getProductsByCommandId(@PathVariable Long commandId) throws Exception {
        return commandService.getProductsByCommandId(commandId);
    }

    // Get all commands for a specific user
    @GetMapping("/userCommands/{userId}")
    public List<Command> getUserCommands(@PathVariable long userId) throws Exception {
        return commandService.getUserCommands(userId);
    }

    // Get all commands
    @GetMapping("/allCommands")
    public List<Command> getAllCommands() throws Exception {
        return commandRepository.findAll();
    }

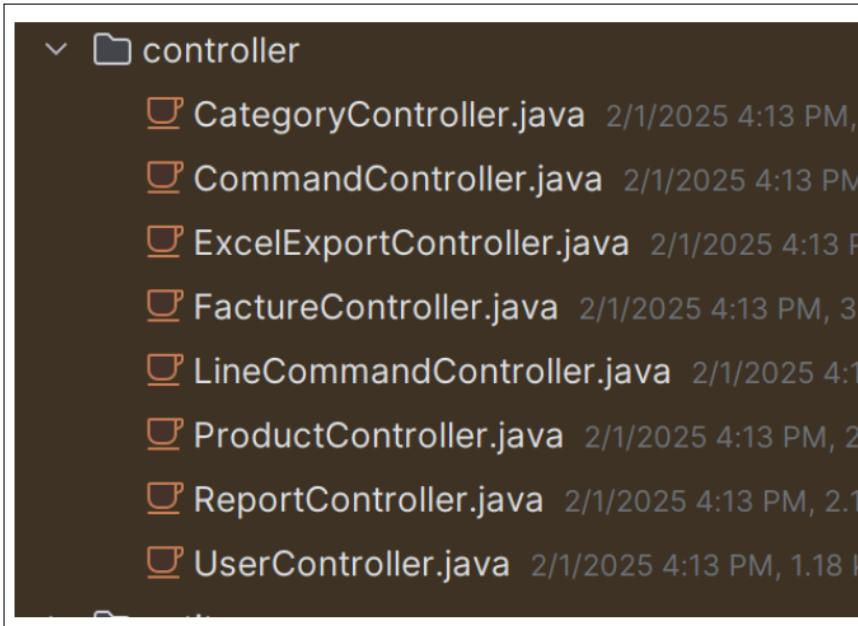
    // Get a command by ID
    @GetMapping("/{id}")
    public Optional<Command> getCommandById(@PathVariable Long id) throws Exception {
        return commandRepository.findById(id);
    }
}

```

**CommandService**, qui gère la création, récupération et suppression des commandes, en intégrant produits, utilisateurs, lignes de commande et factures.

### 7.2.5 Couche Controller (Gestion des Requêtes)

**Rôle** : Gérer les requêtes HTTP et les réponses de l'application. Exemple :



```

public class CommandController {

    @Autowired
    private CommandService commandService;

    // Get all commands
    @GetMapping
    public List<Command> getAllCommands() throws Exception {
        return commandService.getAllCommands();
    }

    // Get a command by ID
    @GetMapping("/{id}")
    public Optional<Command> getCommandById(@PathVariable Long id) throws Exception {
        return commandService.getCommandById(id);
    }

    // Create or update a command
    @PostMapping
    public void saveCommand(@RequestBody Command command) throws Exception {
        commandService.saveCommand(command);
    }

    // Create new Order
    @PostMapping("orders")
    public ResponseEntity<Order> createOrder(@RequestBody CreateOrderRequest orderRequest) {
        try {
            Long orderId = commandService.createOrder(orderRequest);
            return ResponseEntity.ok(orderId);
        } catch (Exception e) {
            return ResponseEntity.status(INTERNAL_SERVER_ERROR).build();
        }
    }

    // Delete a command by ID
    @DeleteMapping("{id}")
    public void deleteCommand(@PathVariable Long id) throws Exception {
        commandService.deleteCommand(id);
    }

    // Get products for a specific command
    @GetMapping("{commandId}/products")
    public List<Product> getProductsByCommandId(@PathVariable Long commandId) throws Exception {
        return commandService.getProductsByCommandId(commandId);
    }

    // Get all commands for a specific user
    @GetMapping("userCommands/{userId}")
    public List<Command> getUserCommands(@PathVariable long userId) throws Exception {
        return commandService.getUserCommands(userId);
    }

    // Get all commands
    @GetMapping("allCommands")
    public List<Command> getAllCommands() throws Exception {
        return commandRepository.findAll();
    }

    // Get a command by ID
    @GetMapping("{id}")
    public Optional<Command> getCommandById(@PathVariable Long id) throws Exception {
        return commandRepository.findById(id);
    }
}

```

**CommandController**, définissant les endpoints API (**GET** , **POST** , **DELETE** ) pour la gestion des commandes.

# Chapitre 8

## Manipulation des Données XML

### 8.1 Stockage des Produits en XML

Les produits sont représentés sous forme de fichiers XML (ex. `products.xml`). Chaque produit possède un `id`, un `title`, une `description`, un `price`, etc. Une validation XSD garantit l'intégrité des informations (types, contraintes, unicité d'`id`, etc.).

```
<product>
  <id>1</id>
  <title>Smartphone X</title>
  <description>A high-end smartphone with advanced features.</description>
  <price>799.99</price>
  <stockQuantity>50</stockQuantity>
  <imageUrl>http://example.com/smartphone-x.jpg</imageUrl>
  <category>Phone</category>
</product>
```

### 8.2 Validation des Fichiers XML avec XSD

Pour assurer la conformité des fichiers XML, un XSD (XML Schema Definition) est défini. Il permet de :

- Définir la structure (ex. `<products>` contenant des `<product>`).
- Vérifier les types (ex. `price` doit être un `double`).
- Imposer des contraintes (unicité de l'`id`, etc.).

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <!-- Define the root element 'products' -->
    <xss:element name="products">
        <xss:complexType>
            <xss:sequence>
                <!-- Define the 'product' element as a sequence of child elements -->
                <xss:element name="product" maxOccurs="unbounded" minOccurs="0">
                    <xss:complexType>
                        <xss:sequence>
                            <!-- Define the child elements in the specified order -->
                            <xss:element name="id">
                                <xss:simpleType>
                                    <xss:restriction base="xs:long">
                                        <xss:minInclusive value="1"/>
                                    </xss:restriction>
                                </xss:simpleType>
                            </xss:element>
                            <xss:element name="title" type="xs:string"/>
                            <xss:element name="description" type="xs:string"/>
                            <xss:element name="price" type="xs:double"/>
                            <xss:element name="stockQuantity" type="xs:int"/>
                            <xss:element name="imageUrl" type="xs:string"/>
                            <xss:element name="category" type="xs:string"/>
                        </xss:sequence>
                    </xss:complexType>
                </xss:element>
            </xss:sequence>
        </xss:complexType>
    </xss:element>
</xss:schema>
```

## 8.3 Stockage et Validation des Users en XML / XSD

De manière similaire, les utilisateurs (`User`) sont stockés dans `users.xml`, validés par `users.xsd`. On y retrouve `id`, `username`, `firstName`, `lastName`, `phoneNumber`, etc., avec des règles (ex. email unique, format du `phoneNumber`, etc.).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<users>
    <user>
        <id>1</id>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <email>john.doe@example.com</email>
        <phoneNumber>0638557931</phoneNumber>
        <address>123 Main St, Springfield</address>
        <username>john_doe</username>
    </user>

```

```

<xsd:element name="user">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="id" maxOccurs="unbounded">
        <xsd:simpleType>
          <xsd:restriction>
            <!-- id: Must be a positive long integer -->
            <xsd:element name="id">
              <xsd:simpleType>
                <xsd:restriction base="xs:long">
                  <xsd:minInclusive value="1"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:element>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="firstName" type="xs:string"/>
      <xsd:element name="lastName" type="xs:string"/>
      <xsd:element name="email" type="xs:string">
        <xsd:restriction>
          <!-- email: Required string with email pattern -->
          <xsd:pattern value="([a-zA-Z0-9_.+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,})"/>
        </xsd:restriction>
      </xsd:element>
      <xsd:element name="username" type="xs:string">
        <xsd:restriction>
          <!-- username: Required string with exactly 10 digits -->
          <xsd:pattern value="([0-9]{10})"/>
        </xsd:restriction>
      </xsd:element>
      <xsd:element name="phoneNumber" type="xs:string">
        <xsd:restriction>
          <!-- Pattern to accept exactly 10 digits -->
          <xsd:pattern value="([0-9]{10})"/>
        </xsd:restriction>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
  
```

The left pane shows the XML instance of a user object:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<commands>
  <command>
    <id>1</id>
    <status>Pending</status>
    <date>2023-10-01</date>
    <userId>1</userId>
  </command>
</commands>
  
```

The right pane shows the corresponding XSD schema definition:

```

<xsd:unique name="uniqueUserID">
  <xsd:selector xpath="user"/>
  <xsd:field xpath="id"/>
</xsd:unique>
<!-- Unique constraint on email -->
<xsd:unique name="uniqueEmail">
  <xsd:selector xpath="user"/>
  <xsd:field xpath="email"/>
</xsd:unique>
<!-- Unique constraint on phone-->
<xsd:unique name="uniquePhoneNumber">
  <xsd:selector xpath="user"/>
  <xsd:field xpath="phoneNumber"/>
</xsd:unique>
<!-- Unique constraint on username -->
<xsd:unique name="uniqueUsername">
  <xsd:selector xpath="user"/>
  <xsd:field xpath="username"/>
</xsd:unique>
</xsd:element>
</xsd:schema>
  
```

## 8.4 Stockage et Validation des Commandes, Factures, Lignes de Commande, Catégories

Le principe reste le même : chaque entité possède son fichier XML et un XSD correspondant. Les **contraintes** (unicité d'`id`, intégrité référentielle, types de champs) sont définies dans le schéma XSD.

The left pane shows the XML instance of a command object:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<commands>
  <command>
    <id>1</id>
    <status>Pending</status>
    <date>2023-10-01</date>
    <userId>1</userId>
  </command>
</commands>
  
```

The right pane shows the corresponding XSD schema definition:

```

<xsd:element name="commands" elementFormDefault="qualified">
  <!-- Define the root element 'commands' -->
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="command" maxOccurs="unbounded" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <!-- Define the child elements in the specified order -->
            <xsd:element name="id" type="xs:long"/>
            <xsd:element name="status" type="xs:string"/>
            <xsd:element name="date" type="xs:string"/>
            <xsd:element name="userId" type="xs:long"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
  
```

The left pane shows the XML instance of a facture object:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<factures>
  <facture>
    <id>1738081215668</id>
    <commandId>10</commandId>
    <userId>1</userId>
    <invoiceDate>2025-01-28</invoiceDate>
    <totalAmount>6099.95</totalAmount>
    <status>CREATED</status>
  </facture>
</factures>
  
```

The right pane shows the corresponding XSD schema definition:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="factures">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="facture" type="factureType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  
```

The `factureType` complex type definition is shown below:

```

<xsd:complexType name="factureType">
  <xsd:sequence>
    <xsd:element name="id" type="xs:long"/>
    <xsd:element name="commandId" type="xs:long"/>
    <xsd:element name="userId" type="xs:long"/>
    <xsd:element name="invoiceDate" type="xs:string"/>
    <xsd:element name="totalAmount" type="xs:double"/>
    <xsd:element name="status" type="xs:string"/>
  </xsd:sequence>
</xsd:complexType>
  
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lineCommands>
  <lineCommand>
    <id>3</id>
    <quantity>3</quantity>
    <productId>2</productId>
    <commandId>1</commandId>
  </lineCommand>
</lineCommands>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <!-- Root element: LineCommands -->
  <xsd:element name="LineCommands">
    <xsd:complexType>
      <xsd:sequence>
        <!-- Child element: lineCommand (multiple) -->
        <xsd:element name="LineCommand" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="id" type="xs:long"/>
              <xsd:element name="quantity" type="xs:int"/>
              <xsd:element name="productId" type="xs:long"/>
              <xsd:element name="commandId" type="xs:long"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:element>
  </xsd:complexType>
<xsd:unique name="uniqueLineCommandID">
  <xsd:selector xpath="LineCommand"/>
  <xsd:field xpath="id"/>
</xsd:unique>
</xsd:element>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<categories>
  <category>
    <id>101</id>
    <name>Electronics</name>
    <description>Electronic devices</description>
    <imageUrl>http://example.com/electronics.jpg</imageUrl>
  </category>
</categories>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <!-- Define the root element 'categories' -->
  <xsd:element name="categories">
    <xsd:complexType>
      <xsd:sequence>
        <!-- Define the 'category' element as a sequence of child elements -->
        <xsd:element name="category" maxOccurs="unbounded" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <!-- Define the child elements in the specified order -->
              <xsd:element name="id" type="xs:long"/>
              <xsd:element name="name" type="xs:string"/>
              <xsd:element name="description" type="xs:string"/>
              <xsd:element name="imageUrl" type="xs:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:element>
  </xsd:complexType>
<xsd:unique name="uniqueCategoryName">
  <xsd:selector xpath="category"/>
  <xsd:field xpath="name"/>
</xsd:unique>
<xsd:unique name="uniqueCategory">
  <xsd:selector xpath="category"/>
  <xsd:field xpath="id"/>
</xsd:unique>
</xsd:element>
</xsd:schema>
```

# Chapitre 9

## Génération de Fichiers depuis XML (HTML, Excel, PDF)

### 9.1 Rapports HTML via XSLT

Les données XML peuvent être converties en **rapports HTML** à l'aide de feuilles XSLT. Par exemple, `products.xml` peut être transformé en un rapport de *stock* ou de *ventes* avec mise en forme HTML.

Summary				
Total Products:	10			
Total Items in Stock:	15			
Low Stock Items (Quantity ≤ 10):	4			

```
public class ReportController {  
    private static final Logger logger = LoggerFactory.getLogger(ReportController.class);  
    private static final String PRODUCTS_FILE = "src/main/resources/data/products.xml";  
  
    @Autowired no usages  
    private ReportGeneratorUtil reportGeneratorUtil;  
  
    @GetMapping("/monthly/products") no usages  
    public ResponseEntity<?> generateMonthlyProductStockSummary() {...}  
}
```

```
@Component  
public class ReportGeneratorUtil {  
    private static final Logger logger = LoggerFactory.getLogger(ReportGeneratorUtil.class);  
    private static final String REPORTS_DIR = "src/main/resources/reports/";  
    private static final String XSLT_DIR = "src/main/resources/xslt/";  
  
    public String generateMonthlyReport(String xmlFilePath, String reportType) throws ReportGenerationException {  
        try {  
            // Validate input files exist  
            validateInputFiles(xmlFilePath, reportType);  
  
            // Create directories if they don't exist  
            createDirectories();  
  
            // Generate unique filename for the report  
            String timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMdd_HHmss"));  
            String outputFileName = String.format("%s_%s_report.html", reportType, timestamp);  
            String outputPath = REPORTS_DIR + outputFileName;  
  
            // Create transformer  
            TransformerFactory transformerFactory = TransformerFactory.newInstance();  
            String xsltPath = XSLT_DIR + reportType + ".xslt";  
  
            logger.info("Loading XSLT from: {}", xsltPath);  
            Source xslt = new StreamSource(new File(xsltPath));  
            Transformer transformer = transformerFactory.newTransformer(xslt);  
  
            // Set up input and output  
            logger.info("Loading XML from: {}", xmlFilePath);  
            Source xml = new StreamSource(new File(xmlFilePath));  
            Result output = new StreamResult(new File(outputPath));
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>

    <!-- Variable to store total value -->
    <xsl:variable name="total-value">
        <xsl:value-of select="sum(//product/price)"/>
    </xsl:variable>

    <xsl:template match="/">
        <html>
            <head>...</head>
            <body>
                <h1>■ Monthly Store Report</h1>
                <div class="summary">
                    <h2>Summary</h2>
                    <p><strong>Total Products:</strong> <xsl:value-of select="count(//product)"/></p>
                    <p><strong>Total Items in Stock:</strong> <xsl:value-of select="sum(//product/stockQuantity)"/></p>
                    <p><strong>Low Stock Items (Quantity ≤ 10):</strong> <xsl:value-of select="count(//product[stockQuantity &lt;= 10])"/></p>
                </div>

                <h2>■ Product Inventory</h2>
                <table>
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>Title</th>
                            <th>Category</th>
                            <th>Price ($)</th>
                            <th>Stock</th>
                        </tr>
                    </thead>
                    <xsl:apply-templates select="//product">
```

```

                    <th>Stock</th>
                </tr>
                <xsl:apply-templates select="//product">
                    <xsl:sort select="category"/>
                </xsl:apply-templates>
            </table>

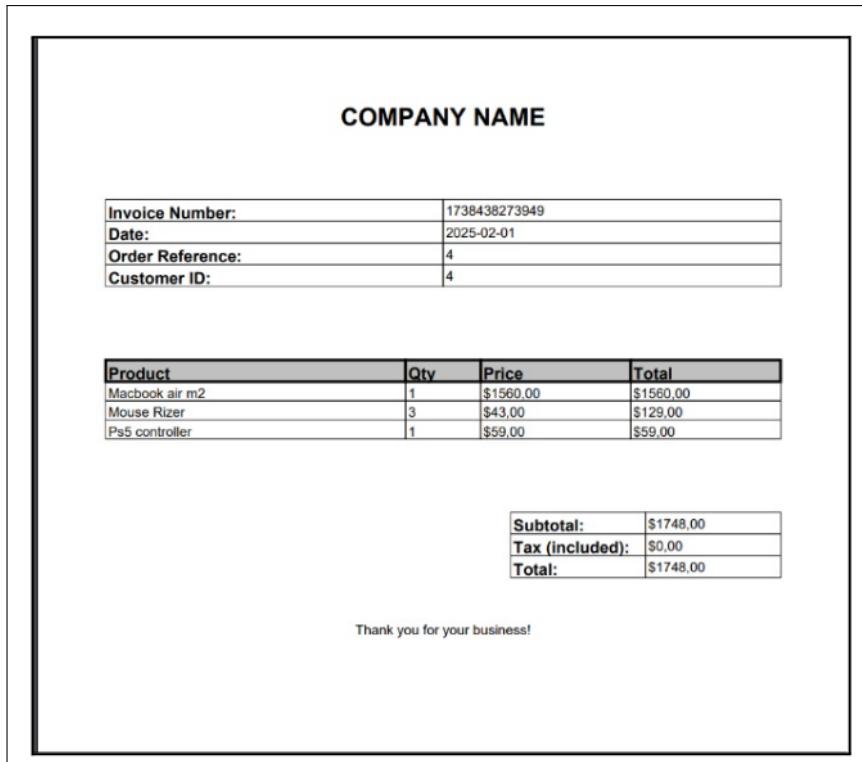
            <h2>■ Category Summary</h2>
            <table>
                <thead>
                    <tr>
                        <th>Category</th>
                        <th>Number of Products</th>
                        <th>Total Stock</th>
                    </tr>
                </thead>
                <xsl:for-each select="//product[not(category=preceding::product/category)]/category">
                    <xsl:variable name="current-category" select=".//category"/>
                    <tr>
                        <td><xsl:value-of select="$current-category"/></td>
                        <td><xsl:value-of select="count(//product[category=$current-category])"/></td>
                        <td><xsl:value-of select="sum(//product[category=$current-category]/stockQuantity)"/></td>
                    </tr>
                </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
```

```
<xsl:template match="product">
    <tr>
        <xsl:if test="stockQuantity &lt;= 10">
            <xsl:attribute name="class">low-stock</xsl:attribute>
        </xsl:if>
        <td><xsl:value-of select="id"/></td>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="category"/></td>
        <td><xsl:value-of select="format-number(price, '#,##0.00')"/></td>
        <td><xsl:value-of select="stockQuantity"/></td>
    </tr>
</xsl:template>
</xsl:stylesheet>
```

FIGURE 9.1 – Feuilles XSLT pour transformer products.xml

## 9.2 Factures PDF

La classe `FactureService` (ou équivalent) permet de générer un PDF à partir d'informations de commande : ligne(s) de commande, total, taxes, etc. Ces informations sont également stockées en XML.



```
// Add footer
document.add(Chunk.NEWLINE);
Paragraph footer = new Paragraph("Thank you for shopping in our store!", NORMAL_FONT);
footer.setAlignment(Element.ALIGN_CENTER);
document.add(footer);
```

```

public void generatePdf(Facture facture, List<LineCommand> lineCommands) {
    try {
        Document document = new Document(PageSize.A4, 50, 50, 50, 50);
        String pdfPath = PDF_DIRECTORY + "facture_" + facture.getId() + ".pdf";
        PdfWriter.getInstance(document, new FileOutputStream(pdfPath));

        document.open();

        // Add company logo/header
        Paragraph header = new Paragraph("COMPANY NAME", TITLE_FONT);
        header.setAlignment(Element.ALIGN_CENTER);
        document.add(header);
        document.add(Chunk.NEWLINE);

        // Add invoice details
        PdfPTable infoTable = new PdfPTable(2);
        infoTable.setWidthPercentage(100);

        addInfoCell(infoTable, "Invoice Number:", facture.getId().toString());
        addInfoCell(infoTable, "Date:", facture.getInvoiceDate());
        addInfoCell(infoTable, "Order Reference:", facture.getCommandId().toString());
        addInfoCell(infoTable, "Customer ID:", facture.getUserId().toString());
        document.add(infoTable);
        document.add(Chunk.NEWLINE);

        // Add items table
        PdfPTable itemsTable = new PdfPTable(new float[]{4, 1, 2, 2});
        itemsTable.setWidthPercentage(100);

        // Add table headers
        addTableHeader(itemsTable);

        // Add line items
        double subtotal = 0;
        for (LineCommand line : lineCommands) {...}

        document.add(itemsTable);
        document.add(Chunk.NEWLINE);

        // Add totals
        PdfPTable totalsTable = new PdfPTable(2);
        totalsTable.setWidthPercentage(40);
        totalsTable.setHorizontalAlignment(Element.ALIGN_RIGHT);

        addTotalRow(totalsTable, "Subtotal:", String.format("%.2f", subtotal));
        addTotalRow(totalsTable, "Tax (included):", String.format("%.2f", facture.getTotalAmount() - subtotal));
        addTotalRow(totalsTable, "Total:", String.format("%.2f", facture.getTotalAmount()));

        document.add(totalsTable);
    }
}

```

## 9.3 Génération des Fichiers Excel

L'exportation des données en **Excel** permet aux utilisateurs d'extraire, d'analyser et de partager des informations de manière structurée. Dans ce projet, un système d'exportation *dynamique* a été implémenté, permettant de générer des fichiers Excel pour différentes entités (Utilisateurs, Produits, Commandes, Catégories).

### Exemples d'export possible :

- **Users** : Liste des utilisateurs (identifiants, emails, noms, adresses, etc.).

A	B	C	D	E	F	G
ID	Username	Email	First Name	Last Name	Phone Number	Address
1	john_doe	john.doe@example.com	John	Doe	0638557931	123 Main St, Springfield
2	jane_smith	jane.smith@example.com	Jane	Smith	9876543210	456 Elm St, Springfield
3	arbi_allam	arbi.allam@example.com	Arbi	Allam	5551234567	789 Oak St, Springfield
4	tarek_alazami	tarek.alazami@example.com	Tarek	Al Azami	0638557531	321 Pine St, Springfield

- **Products** : Liste des produits (titre, description, catégorie, prix, stock).

A	B	C	D	E	F	G
Id	Description	Title	Price	Image Url	Stock Quantity	Category
1	1 A high-end smartphone with advanced features.	Smartphone X	799.99	http://example.com/smartphone-x.jpg	50	Phone
2	2 A powerful laptop designed for gaming and productivity.	Gaming Laptop	1499.99	http://example.com/gaming-laptop.jpg	10	Phone
3	3 Noise-cancelling wireless earbuds with long battery life.	Wireless Earbuds	199.99	http://example.com/wireless-earbuds.jpg	100	Laptop
4	4 A 55-inch 4K Ultra HD Smart TV with HDR support.	4K Smart TV	899.99	http://example.com/4k-smart-tv.jpg	15	Electronics
5	5 A high-performance laptop (Updated)	tarek arbi	1099.99	http://example.com/laptop-updated.jpg	5	Phone
6	6 A high-performance laptop (Updated)	tarek arbi	1099.99	http://example.com/laptop-updated.jpg	5	Phone
7	7 A high-performance laptop (Updated)	tarek arbi	1099.99	http://example.com/laptop-updated.jpg	5	Phone

- **Commandes** : Historique des commandes avec date, statut, total, etc.

1	Id	Date	Status	User Id
2	2	2023-10-02	Shipped	102
3	10	2023-10-02	Pending	1
4	11	2023-10-02	Pending	1
5	12	2023-10-02	arbi	1

Le contrôleur `ExcelExportController` est responsable de :

- Gérer les requêtes d'exportation via l'API (par exemple, GET `/api/export/{type}/excel`).
- Identifier l'entité demandée (`users`, `products`, `commands`, `categories`).
- Appeler le service `ExcelExportService` pour générer le fichier Excel correspondant.
- Retourner le fichier en réponse pour le téléchargement.

## 9.4 Recherche et Filtrage de Produits via XPath

Ce module ajoute une fonctionnalité avancée de recherche et de filtrage des produits à partir des fichiers XML en utilisant **XPath**. Il permet d'effectuer des requêtes flexibles sur les produits en fonction de plusieurs critères (titre, catégorie, description, prix, quantité en stock, etc.).

```

@GetMapping("/{entityType}/excel")
public ResponseEntity<?> exportToExcel(@PathVariable String entityType) {
    try {
        String filePath;
        switch (entityType.toLowerCase()) {
            case "users":
                filePath = excelExportService.exportToExcel(
                    userService.getAllUsers(), "Users");
                break;
            case "products":
                filePath = excelExportService.exportToExcel(
                    productService.getAllProducts(), "Products");
                break;
            case "commands":
                filePath = excelExportService.exportToExcel(
                    commandService.getAllCommands(), "Commands");
                break;
            case "categories":
                filePath = excelExportService.exportToExcel(
                    categoryService.getAllCategories(), "Categories");
                break;
            // Add other entity types here
            default:
                return ResponseEntity.badRequest()
                    .body("Unsupported entity type: " + entityType);
        }
    }
}

```

```

// Find products by XPath expression
// Enhanced findByXPath method supporting complex queries
public List<Product> findByXPath(String xpathExpression) throws Exception {...}

private String processXPathExpression(String rawExpression) {...}

// Helper method to convert XML nodes to Product objects
private List<Product> convertNodesToProducts(NodeList nodes) {...}

// Helper method to set product fields
private void setProductField(Product product, String fieldName, String value) {...}

}

xpathBuilder.append(" and ");
xpathBuilder.append(
    "(contains(translate(category, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz'), '" + word + "') or " +
    "contains(translate(title, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz'), '" + word + "') or " +
    "contains(translate(description, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz'), '" + word + "'))"
);
}

```

#### 9.4.1 Recherche de produits avec XPath

- Analyse et exécute des expressions XPath pour extraire les produits correspondants.

- 
- Transforme les noeuds XML en objets Product.

```
findByXPath(String xpathExpression)
```

- Charge et parse le fichier XML (`products.xml`).
- Compile et exécute l'expression XPath sur le document.
- Convertit les noeuds correspondants en objets Product.

```
processXPathExpression(String rawExpression)
```

- Convertit l'entrée de l'utilisateur en une expression XPath valide.
- Ajoute dynamiquement des conditions pour la recherche multi-critères.
- Gère les filtres numériques (prix, stock) et les expressions contenant plusieurs mots.

```
convertNodesToProducts(NodeList nodes)
```

- Parcourt les résultats XPath et convertit chaque noeud XML en objet Product.
- Associe chaque balise XML à son champ correspondant dans la classe Product.

```
setProductField(Product product, String fieldName, String value)
```

- Assigne les valeurs des noeuds XML aux attributs de l'objet Product.
- Gère les conversions de types (de `String` vers `Long`, `Double`, `Integer`) et évite les erreurs.

## 9.5 Conclusion

Ce projet fournit un système robuste et modulaire permettant de :

- Stocker et gérer les données en XML avec validation XSD.
- Exporter les données en **Excel**, **PDF** et **HTML** pour une meilleure analyse.
- Automatiser le traitement des commandes et factures pour optimiser la gestion des ventes.
- Offrir une API REST intuitive pour accéder et télécharger les fichiers générés.

En définitive, cette solution e-commerce repose sur un *écosystème XML* pour la persistance et sur un **frontend React** moderne, permettant une expérience complète pour les utilisateurs et les administrateurs.