RAPPORT DE PROJET

ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

GESTION DE MAGASIN TECHSTORE

SUPERVISION RÉSEAU

Réalisé par :BOUGHABA Nada
Filière : 3ème année IIR
Encadré par : TLEMÇANI Khadija

École Marocaine des Sciences de l'Ingénieur
Année Universitaire 2025/2026

Remerciement

Avant toute chose, je tiens à exprimer ma profonde gratitude à Dieu Tout- Puissant pour m'avoir accordé la force, la patience et la persévérance nécessaires pour mener à bien ce stage et réaliser ce rapport.

Nous adressons nos sinc`eres remerciements `a Madame TLEMÇANI Khadija, notre encadrante, pour son accompagnement, ses conseils pr ´ecieux et sa disponibilit´e tout au long de ce projet. Ses orientations nous ont permis de mener `a bien ce travail dans les meilleures conditions.

Nous remercions également l'ensemble du corps professoral de l'EMSI - Ecole Marocaine des Sciences de l'Ingénieur, pour la qualité de l'enseignement dispensé et les compétences techniques acquises durant notre formation en Ingénierie Informatique et Réseaux

Enfin, je dedie une pensee toute particuliere a ma famille et a mes proches pour leur soutien moral, leurs encouragements constants, et leur confiance, sans lesquels ce parcours n'aurait pas ete possible.

A tous, merci du fond du coeur.

Sommaire

Résumé:

Ce rapport présente le développement d'une application web Single Page Application (SPA) de gestion de magasin de produits technologiques, réalisée dans le cadre de notre formation en 3ème année Ingénierie Informatique et Réseaux à l'EMSI Casablanca.

L'application a été développée en HTML5, CSS3 et JavaScript Vanilla, fonctionnant entièrement côté client avec persistance via LocalStorage. Elle offre une solution complète de gestion de magasin comprenant trois modules distincts : gestion CRUD des produits, gestion des catégories, et un dashboard avec intégration API externe.

Les fonctionnalités principales incluent un système de gestion complète des produits (ajout, modification, suppression, recherche, tri), un module de gestion des catégories avec synchronisation automatique, un tableau de bord avec indicateurs de performance (KPIs) et visualisation graphique via Chart.js, et une intégration de l'API FakeStore pour l'import de données externes.

L'interface utilisateur moderne, développée avec Bootstrap 5 et un design glassmorphism, garantit une expérience optimale et responsive sur tous les appareils. La persistance des données est assurée par LocalStorage, permettant un fonctionnement complet hors ligne.

Ce projet nous a permis de mettre en pratique nos connaissances en développement web frontend, architecture SPA, manipulation du DOM, programmation orientée objet en JavaScript, et intégration d'APIs externes.

Introduction générale:

Dans un contexte de digitalisation des commerces, la gestion informatisée des stocks et produits est devenue essentielle pour optimiser les opérations commerciales. Les applications web modernes offrent aux gestionnaires la possibilité de suivre leurs inventaires en temps réel, d'analyser leurs performances et de prendre des décisions éclairées.

Ce projet s'inscrit dans cette dynamique de transformation numérique et vise à développer une application web complète de gestion de magasin de produits technologiques. L'objectif principal est de concevoir une plateforme moderne, intuitive et performante permettant aux gestionnaires de gérer leurs produits, catégories et d'analyser leurs statistiques via un tableau de bord interactif.

Le choix d'une architecture Single Page Application (SPA) en JavaScript Vanilla s'explique par la volonté de maîtriser les fondamentaux du développement web moderne sans dépendre de frameworks lourds. L'utilisation de LocalStorage permet un fonctionnement autonome sans nécessiter de serveur backend, idéal pour un prototype fonctionnel.

L'architecture modulaire adoptée garantit une séparation claire des responsabilités, facilitant ainsi la maintenance et l'évolutivité de l'application. Le design glassmorphism avec animations procure une expérience utilisateur moderne et engageante.

Ce rapport détaille l'ensemble du processus de développement, depuis l'analyse des besoins jusqu'à la mise en œuvre des fonctionnalités, en passant par les choix technologiques, l'architecture de l'application, et les solutions apportées aux différentes problématiques rencontrées.

1. Introduction générale:

Dans un contexte de digitalisation des commerces, la gestion informatisée des stocks et produits est devenue essentielle pour optimiser les opérations commerciales. Les applications web modernes offrent aux gestionnaires la possibilité de suivre leurs inventaires en temps réel, d'analyser leurs performances et de prendre des décisions éclairées.

Ce projet s'inscrit dans cette dynamique de transformation numérique et vise à développer une application web complète de gestion de magasin de produits technologiques. L'objectif principal est de concevoir une plateforme moderne, intuitive et performante permettant aux gestionnaires de gérer leurs produits, catégories et d'analyser leurs statistiques via un tableau de bord interactif.

Le choix d'une architecture Single Page Application (SPA) en JavaScript Vanilla s'explique par la volonté de maîtriser les fondamentaux du développement web moderne sans dépendre de frameworks lourds. L'utilisation de LocalStorage permet un fonctionnement autonome sans nécessiter de serveur backend, idéal pour un prototype fonctionnel.

L'architecture modulaire adoptée garantit une séparation claire des responsabilités, facilitant ainsi la maintenance et l'évolutivité de l'application. Le design glassmorphism avec animations procure une expérience utilisateur moderne et engageante.

Ce rapport détaille l'ensemble du processus de développement, depuis l'analyse des besoins jusqu'à la mise en œuvre des fonctionnalités, en passant par les choix technologiques, l'architecture de l'application, et les solutions apportées aux différentes problématiques rencontrées.

Structure du rapport :

Le présent document est organisé comme suit : la section 2 présente les technologies et outils utilisés, la section 3 détaille le développement des différents modules de l'application, la section 4 décrit le système de dashboard et de visualisation des données, les sections 5 et 6 abordent respectivement les tests effectués et les difficultés rencontrées, la section 7 présente les résultats finaux avec des captures d'écran, et enfin la section 8 conclut ce travail en ouvrant sur des perspectives d'amélioration.

2. Technologies utilisées:

2.1 Langages utilisés

- HTML5 : Structure sémantique de l'application avec balises modernes
- CSS3 : Mise en forme avancée avec glassmorphism, animations et transitions
- JavaScript : Logique applicative avec classes, modules, async/await

2.2 Bibliothèques et frameworks

- Bootstrap 5.3.0 : Framework CSS pour les composants UI et le design responsive
- Chart.js : Bibliothèque de visualisation de données pour les graphiques
- Font Awesome 6.4.0 : Bibliothèque d'icônes vectorielles
- Google Fonts (Inter) : Typographie moderne et lisible

2.3 Outils de développement

- VS Code : Éditeur de code principal avec extensions JavaScript
- Chrome DevTools : Débogage et inspection du code
- Git : Contrôle de version du code source
- GitHub : Hébergement du dépôt de code
- Live Server : Serveur de développement local avec rechargement automatique

2.4 APIs externes

- FakeStore API : API REST publique pour l'import de produits de démonstration
- Endpoint : https://fakestoreapi.com/products
- Format : JSON
- Méthode : GET
-

3. Développement de l'application

3.1 Structure générale de l'application

3.1.1 Organisation des fichiers

L'application suit une structure modulaire claire :

```
projetjs/
├── index.html              # Page principale SPA
├── css/
│   └── styles.css          # Styles personnalisés (966 lignes)
├── js/
│   ├── app.js              # Application principale (113 lignes)
│   ├── modules/
│   │   ├── produits.js     # Gestion produits (373 lignes)
│   │   ├── categories.js   # Gestion catégories (181 lignes)
│   │   └── dashboard.js    # Dashboard & API (252 lignes)
│   └── utils/
│       └── storage.js      # Gestion LocalStorage (118 lignes)
└── README.md               # Documentation
```

Total : ~2 500 lignes de code

3.1.2 Architecture SPA

L'application implémente une architecture Single Page Application avec navigation dynamique :

Navigation (app.js)

- Classe App comme contrôleur principal
- Gestion des sections sans rechargement de page
- Synchronisation sidebar et navbar
- Mise à jour dynamique du contenu
- État actif des liens de navigation

Flux de navigation

1. Utilisateur clique sur un lien de navigation
2. App.showSection() masque toutes les sections
3. Affiche la section demandée
4. Rafraîchit les données du module concerné
5. Met à jour l'état actif dans la navigation

3.1.2 Architecture SPA

Stockage des données (storage.js)

- Classe StorageManager comme singleton
- Gestion centralisée de LocalStorage
- Méthodes CRUD pour produits et catégories
- Calculs de statistiques agrégées
- Initialisation des données par défaut

3.2 Module 1 : Gestion des Produits

3.2.1 Architecture du module

```
// js/modules/produits.js
class ProduitsManager {
    constructor() {
        this.storage = window.storageManager;
        this.currentEditId = null;
        this.initEventListeners();
    }

    initEventListeners() {
        // Formulaire d'ajout
        document.getElementById('productForm')
            .addEventListener('submit', (e) => this.handleSubmit(e));

        // Recherche en temps réel
        document.getElementById('searchProduct')
            .addEventListener('input', (e) => this.handleSearch(e));

        // Tri dynamique
        document.getElementById('sortProducts')
            .addEventListener('change', (e) => this.handleSort(e));
    }
}
```

3.2.2 Ajout et modification d'un produit

Formulaire de produit :

```html
<form id="productForm">
    <input type="text" id="productName" placeholder="Nom du produit" required>
    <select id="productCategory" required>
        <!-- Options dynamiques depuis catégories -->
    </select>
    <input type="number" id="productPrice" placeholder="Prix"
           min="0" step="0.01" required>
    <input type="number" id="productStock" placeholder="Stock"
           min="0" required>
    <input type="url" id="productImage" placeholder="URL de l'image">
    <textarea id="productDescription" placeholder="Description"></textarea>
    <button type="submit">Ajouter le produit</button>
</form>
```

Logique d'ajout/modification :

```avascript
handleSubmit(e) {
    e.preventDefault();

    const product = {
        nom: document.getElementById('productName').value.trim(),
        categorie: document.getElementById('productCategory').value,
        prix: parseFloat(document.getElementById('productPrice').value),
        stock: parseInt(document.getElementById('productStock').value),
        image: document.getElementById('productImage').value.trim(),
        description: document.getElementById('productDescription').value.trim()
    };

    // Validation
    if (!this.validateProduct(product)) {
        this.showAlert('Veuillez remplir tous les champs correctement', 'danger');
        return;
    }

    if (this.currentEditId) {
        // Modification
        this.storage.updateProduct(this.currentEditId, product);
        this.showAlert('Produit modifié avec succès!', 'success');
        this.currentEditId = null;
    } else {
        // Ajout
        this.storage.addProduct(product);
        this.showAlert('Produit ajouté avec succès!', 'success');
    }

    this.resetForm();
    this.displayProducts();
}

validateProduct(product) {
    if (!product.nom || product.nom.length < 2) return false;
    if (!product.categorie) return false;
    if (product.prix <= 0) return false;
    if (product.stock < 0) return false;
    return true;
}
```

3.2.3 Affichage et recherche

Affichage des produits en cartes :

```javascript
displayProducts(productsToDisplay = null) {
    const products = productsToDisplay || this.storage.getProducts();
    const container = document.getElementById('productsList');

    if (products.length === 0) {
        container.innerHTML = `
            <div class="col-12 text-center py-5">
                <i class="fas fa-box-open fa-3x mb-3 text-muted"></i>
                <p class="text-muted">Aucun produit disponible</p>
            </div>
        `;
        return;
    }

    container.innerHTML = products.map(product => `
        <div class="col-md-6 col-lg-4 mb-4">
            <div class="card h-100 product-card">
                <img src="${product.image || 'https://via.placeholder.com/300'}"
                     class="card-img-top" alt="${product.nom}"
                     onerror="this.src='https://via.placeholder.com/300'">
                <div class="card-body">
                    <h5 class="card-title">${product.nom}</h5>
                    <p class="card-text text-muted">${product.description || 'Pas de description'}</p>
                    <div class="d-flex justify-content-between align-items-center mb-3">
                        <span class="badge bg-primary">${product.categorie}</span>
                        <span class="badge ${this.getStockBadgeClass(product.stock)}">
                            Stock: ${product.stock}
                        </span>
                    </div>
                    <div class="d-flex justify-content-between align-items-center">
                        <h4 class="text-primary mb-0">${product.prix.toFixed(2)} DH</h4>
                        <div class="btn-group">
                            <button onclick="produitsManager.showDetails(${product.id})"
                                    class="btn btn-sm btn-info">
                                <i class="fas fa-eye"></i>
                            </button>
                            <button onclick="produitsManager.editProduct(${product.id})"
                                    class="btn btn-sm btn-warning">
                                <i class="fas fa-edit"></i>
                            </button>
                            <button onclick="produitsManager.deleteProduct(${product.id})"
                                    class="btn btn-sm btn-danger">
                                <i class="fas fa-trash"></i>
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    `).join('');
}
getStockBadgeClass(stock) {
    if (stock === 0) return 'bg-danger';
    if (stock < 10) return 'bg-warning';
    return 'bg-success';
}
```

Recherche en temps réel :

```javascript
handleSearch(e) {
    const searchTerm = e.target.value.toLowerCase().trim();

    if (!searchTerm) {
        this.displayProducts();
        return;
    }

    const products = this.storage.getProducts();
    const filtered = products.filter(product =>
        product.nom.toLowerCase().includes(searchTerm) ||
        product.description.toLowerCase().includes(searchTerm)
    );

    this.displayProducts(filtered);
}
```

3.2.3 Affichage et recherche

```javascript
handleSort(e) {
    const sortBy = e.target.value;
    let products = [...this.storage.getProducts()];

    switch(sortBy) {
        case 'name-asc':
            products.sort((a, b) => a.nom.localeCompare(b.nom));
            break;
        case 'name-desc':
            products.sort((a, b) => b.nom.localeCompare(a.nom));
            break;
        case 'price-asc':
            products.sort((a, b) => a.prix - b.prix);
            break;
        case 'price-desc':
            products.sort((a, b) => b.prix - a.prix);
            break;
        case 'stock-asc':
            products.sort((a, b) => a.stock - b.stock);
            break;
        case 'stock-desc':
            products.sort((a, b) => b.stock - a.stock);
            break;
    }
    this.displayProducts(products);
}
```

3.2.5 Modal de détails

```javascript
showDetails(productId) {
    const product = this.storage.getProducts().find(p => p.id === productId);
    if (!product) return;

    const modalContent = `
        <div class="modal fade" id="productModal" tabindex="-1">
            <div class="modal-dialog modal-lg">
                <div class="modal-content">
                    <div class="modal-header">
                        <h5 class="modal-title">${product.nom}</h5>
                        <button type="button" class="btn-close"
                                data-bs-dismiss="modal"></button>
                    </div>
                    <div class="modal-body">
                        <div class="row">
                            <div class="col-md-6">
                                <img src="${product.image}" class="img-fluid rounded">
                            </div>
                            <div class="col-md-6">
                                <p><strong>Catégorie:</strong> ${product.categorie}</p>
                                <p><strong>Prix:</strong> ${product.prix.toFixed(2)} DH</p>
                                <p><strong>Stock:</strong> ${product.stock} unités</p>
                                <p><strong>Description:</strong> ${product.description}</p>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    `;

    document.body.insertAdjacentHTML('beforeend', modalContent);
    const modal = new bootstrap.Modal(document.getElementById('productModal'));
    modal.show();
}
```

3.2.6 Suppression avec confirmation

```javascript
deleteProduct(productId) {
    if (!confirm('Êtes-vous sûr de vouloir supprimer ce produit?')) {
        return;
    }

    this.storage.deleteProduct(productId);
    this.showAlert('Produit supprimé avec succès!', 'success');
    this.displayProducts();
}
```

3.3 Module 2 : Gestion des Catégories

3.3.1 Architecture du module

```javascript
// js/modules/categories.js
class CategoriesManager {
    constructor() {
        this.storage = window.storageManager;
        this.initEventListeners();
    }

    init() {
        this.displayCategories();
        this.updateCategorySelect();
    }
}
```

3.1.2 Architecture SPA

```javascript
handleSubmit(e) {
    e.preventDefault();

    const category = {
        nom: document.getElementById('categoryName').value.trim(),
        description: document.getElementById('categoryDescription').value.trim()
    };

    if (!category.nom || category.nom.length < 2) {
        this.showAlert('Le nom de la catégorie doit contenir au moins 2 caractères', 'danger');
        return;
    }

    // Vérifier si la catégorie existe déjà
    const existingCategories = this.storage.getCategories();
    if (existingCategories.find(c => c.nom.toLowerCase() === category.nom.toLowerCase())) {
        this.showAlert('Cette catégorie existe déjà!', 'warning');
        return;
    }

    this.storage.addCategory(category);
    this.showAlert('Catégorie ajoutée avec succès!', 'success');
    this.resetForm();
    this.displayCategories();
    this.updateCategorySelect();
}
```

3.3.3 Affichage avec compteur de produits

```javascript
displayCategories() {
    const categories = this.storage.getCategories();
    const container = document.getElementById('categoriesList');

    if (categories.length === 0) {
        container.innerHTML = `
            <div class="col-12 text-center py-5">
                <i class="fas fa-tags fa-3x mb-3 text-muted"></i>
                <p class="text-muted">Aucune catégorie disponible</p>
            </div>
        `;
        return;
    }

    container.innerHTML = categories.map(category => {
        const productCount = this.storage.getProductsByCategory(category.nom).length;
        const iconClass = this.getCategoryIcon(category.nom);
        const colorClass = this.getCategoryColor(category.nom);

        return `
            <div class="col-md-6 col-lg-4 mb-4">
                <div class="card h-100 category-card ${colorClass}">
                    <div class="card-body">
                        <div class="d-flex justify-content-between align-items-start mb-3">
                            <div>
                                <i class="${iconClass} fa-2x mb-2"></i>
                                <h5 class="card-title">${category.nom}</h5>
                            </div>
                            <span class="badge bg-light text-dark">
                                ${productCount} produits
                            </span>
                        </div>
                        <p class="card-text text-muted">
                            ${category.description || 'Pas de description'}
                        </p>
                        <button onclick="categoriesManager.deleteCategory(${category.id})"
                                class="btn btn-sm btn-danger"
                                ${productCount > 0 ? 'disabled' : ''}>
                            <i class="fas fa-trash"></i> Supprimer
                        </button>
                    </div>
                </div>
            </div>
        `;
    }).join('');
}
```

```
getCategoryIcon(categoryName) {
    const icons = {
        'Ordinateurs': 'fas fa-laptop',
        'Smartphones': 'fas fa-mobile-alt',
        'Tablettes': 'fas fa-tablet-alt',
        'Accessoires': 'fas fa-headphones'
    };
    return icons[categoryName] || 'fas fa-box';
}

getCategoryColor(categoryName) {
    const colors = {
        'Ordinateurs': 'border-primary',
        'Smartphones': 'border-success',
        'Tablettes': 'border-info',
        'Accessoires': 'border-warning'
    };
    return colors[categoryName] || 'border-secondary';
}
```

3.4 Module 3 : Dashboard et API

3.4.1 Calcul et affichage des KPIs

```javascript
// js/modules/dashboard.js
class Dashboard {
    constructor() {
        this.storage = window.storageManager;
        this.chart = null;
    }

    init() {
        this.updateKPIs();
        this.updateChart();
    }

    updateKPIs() {
        const stats = this.storage.getStatistics();

        // Total Produits
        document.getElementById('totalProducts').textContent = stats.totalProducts;

        // Stock Total
        document.getElementById('totalStock').textContent = stats.totalStock;

        // Valeur du Stock
        document.getElementById('stockValue').textContent =
            `${stats.stockValue.toFixed(2)} DH`;

        // Total Catégories
        document.getElementById('totalCategories').textContent = stats.totalCategories;
    }
}
```

3.4.2 Visualisation avec Chart.js

```javascript
javascript
updateChart() {
    const categories = this.storage.getCategories();
    const chartData = categories.map(category => ({
        label: category.nom,
        count: this.storage.getProductsByCategory(category.nom).length
    }));

    const ctx = document.getElementById('categoryChart').getContext('2d');

    // Détruire l'ancien graphique s'il existe
    if (this.chart) {
        this.chart.destroy();
    }

    this.chart = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: chartData.map(d => d.label),
            datasets: [{
                label: 'Nombre de produits par catégorie',
                data: chartData.map(d => d.count),
                backgroundColor: [
                    'rgba(99, 102, 241, 0.8)',   // Indigo
                    'rgba(16, 185, 129, 0.8)',   // Vert
                    'rgba(6, 182, 212, 0.8)',    // Cyan
                    'rgba(245, 158, 11, 0.8)'    // Orange
                ],
                borderColor: [
                    'rgba(99, 102, 241, 1)',
                    'rgba(16, 185, 129, 1)',
                    'rgba(6, 182, 212, 1)',
                    'rgba(245, 158, 11, 1)'
                ],
                borderWidth: 2
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            plugins: {
                legend: {
                    display: false
                },
                title: {
                    display: true,
                    text: 'Répartition des produits par catégorie',
                    font: {
                        size: 16
                    }
                }
            },
            scales: {
                y: {
                    beginAtZero: true,
                    ticks: {
                        stepSize: 1
                    }
                }
            }
        }
    });
}
```

3.4.3 Intégration API FakeStore

```javascript
async loadFromAPI() {
    const button = document.getElementById('loadAPIBtn');
    button.disabled = true;
    button.innerHTML = '<span class="spinner-border spinner-border-sm"></span> Chargement...';

    try {
        const response = await fetch('https://fakestoreapi.com/products');

        if (!response.ok) {
            throw new Error(`Erreur HTTP: ${response.status}`);
        }

        const products = await response.json();
        let imported = 0;

        products.forEach(apiProduct => {
            // Créer la catégorie si elle n'existe pas
            const categoryName = this.normalizeCategoryName(apiProduct.category);
            this.ensureCategoryExists(categoryName);

            // Ajouter le produit
            const product = {
                nom: apiProduct.title,
                categorie: categoryName,
                prix: apiProduct.price * 10, // Conversion USD -> DH approximative
                stock: Math.floor(Math.random() * 50) + 10,
                image: apiProduct.image,
                description: apiProduct.description
            };

            this.storage.addProduct(product);
            imported++;
        });

        this.showAlert(`${imported} produits importés avec succès!`, 'success');
        this.updateKPIs();
        this.updateChart();

    } catch (error) {
        console.error('Erreur lors du chargement:', error);
        this.showAlert('Erreur lors du chargement des données depuis l\'API', 'danger');
    } finally {
        button.disabled = false;
        button.innerHTML = '<i class="fas fa-download"></i> Charger depuis l\'API';
    }
}

normalizeCategoryName(apiCategory) {
    const mapping = {
        'electronics': 'Ordinateurs',
        'jewelery': 'Accessoires',
        "men's clothing": 'Accessoires',
        "women's clothing": 'Accessoires'
    };
    return mapping[apiCategory] || 'Accessoires';
}

ensureCategoryExists(categoryName) {
    const categories = this.storage.getCategories();
    if (!categories.find(c => c.nom === categoryName)) {
        this.storage.addCategory({
            nom: categoryName,
            description: `Catégorie ${categoryName}`
        });
    }
}
```

4. Dashboard et visualisation des données

4.1 Indicateurs de performance (KPIs)

Le dashboard affiche quatre KPIs principaux sous forme de cartes colorées :

```
<div class="row g-4 mb-4">
    <!-- Total Produits -->
    <div class="col-md-6 col-lg-3">
        <div class="card kpi-card border-primary">
            <div class="card-body">
                <div class="d-flex justify-content-between align-items-center">
                    <div>
                        <h6 class="text-muted mb-2">Total Produits</h6>
                        <h3 id="totalProducts" class="mb-0">0</h3>
                    </div>
                    <div class="icon-box bg-primary">
                        <i class="fas fa-box fa-2x text-white"></i>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <!-- Stock Total -->
    <div class="col-md-6 col-lg-3">
        <div class="card kpi-card border-success">
            <div class="card-body">
                <div class="d-flex justify-content-between align-items-center">
                    <div>
                        <h6 class="text-muted mb-2">Stock Total</h6>
                        <h3 id="totalStock" class="mb-0">0</h3>
                    </div>
```

4.2 Graphique de répartition

```
html
<div class="card">
    <div class="card-header">
        <h5 class="mb-0">
            <i class="fas fa-chart-bar me-2"></i>
            Répartition des produits
        </h5>
    </div>
    <div class="card-body">
        <div style="height: 400px;">
            <canvas id="categoryChart"></canvas>
        </div>
    </div>
</div>
```

4.3 Import API FakeStore

Interface d'import :

```html
<div class="card mb-4">
    <div class="card-header">
        <h5 class="mb-0">
            <i class="fas fa-cloud-download-alt me-2"></i>
            Importer des données
        </h5>
    </div>
    <div class="card-body">
        <p class="text-muted">
            Importez des produits de démonstration depuis l'API FakeStore
        </p>
        <button id="loadAPIBtn" class="btn btn-primary">
            <i class="fas fa-download"></i> Charger depuis l'API
        </button>
    </div>
</div>
```

5. Tests et validation

5.1 Tests fonctionnels

Tests du module Produits :

- Ajout de produit avec tous les champs
- Modification d'un produit existant
- Suppression avec confirmation
- Recherche en temps réel
- Tri par nom, prix, stock (croissant/décroissant)
- Affichage des détails en modal
- Gestion des images manquantes (placeholder)

Tests du module Catégories :

- Ajout de catégorie
- Affichage avec compteur de produits
- Suppression bloquée si produits liés
- Synchronisation avec le formulaire produits

Tests du Dashboard :

- Calcul correct des KPIs
- Mise à jour en temps réel
- Affichage du graphique Chart.js
- Import API FakeStore
- Gestion des erreurs réseau

5.2 Tests de validation

Validation des formulaires :

- Champs requis vérifiés
- Validation des types (nombre, texte, URL)
- Validation des valeurs minimales (prix > 0, stock ≥ 0)
- Messages d'erreur clairs
- Feedback visuel (classes Bootstrap)

Validation des données :

- Noms de produits/catégories non vides
- Prix et stock numériques
- URLs d'images valides (format)
- Unicité des catégories

5.3 Tests de compatibilité

Navigateurs testés :

- Chrome 120+ (principal)
- Firefox 121+
- Safari 17+
- Edge 120+

Appareils testés :

- Desktop (1920x1080, 1366x768)
- Tablette (iPad, Android)
- Mobile (iPhone, Android)

Responsive design :

- Sidebar rétractable sur mobile
- Cartes adaptatives (grid Bootstrap)
- Formulaires optimisés mobile
- Navigation touch-friendly

5.4 Tests de performance

LocalStorage :

- Stockage de 100+ produits sans ralentissement
- Recherche instantanée
- Tri rapide (< 100ms)
- Mise à jour KPIs (< 50ms)

Chargement :

- Temps de chargement initial < 1s
- Navigation entre sections instantanée
- Animations fluides (60fps)

6. Difficultés rencontrées et solutions apportées

6.1 Gestion de l'état de l'application

Problème : Synchronisation des données entre modules (produits, catégories, dashboard)

Solution :

- Création d'un StorageManager singleton comme source unique de vérité
- Méthodes de notification entre modules lors des modifications
- Rafraîchissement automatique lors du changement de section

6.2 Persistance des données avec LocalStorage

Problème : Gestion des IDs auto-incrémentés et éviter les doublons

Solution :

- Stockage d'un compteur séparé pour chaque entité
- Génération d'IDs uniques lors de l'ajout
- Vérification d'unicité pour les catégories

6.3 Gestion des images manquantes

Problème : URLs d'images invalides ou images supprimées

Solution :

- Attribut onerror sur les balises <img>
- Placeholder par défaut via placeholder.com
- Validation optionnelle des URLs

6.4 Appels API asynchrones

Problème : Gestion des erreurs réseau et timeout API

Solution :

- Utilisation de async/await avec try/catch
- Feedback visuel pendant le chargement
- Messages d'erreur explicites
- Bouton désactivé pendant le chargement

6.5 Suppression de catégories avec produits liés

Problème : Éviter la suppression de catégories utilisées par des produits

Solution :

- Vérification des dépendances avant suppression
- Compteur de produits affiché sur les cartes
- Bouton de suppression désactivé si produits liés
- Message explicatif à l'utilisateur

6.6 Responsive design de la sidebar

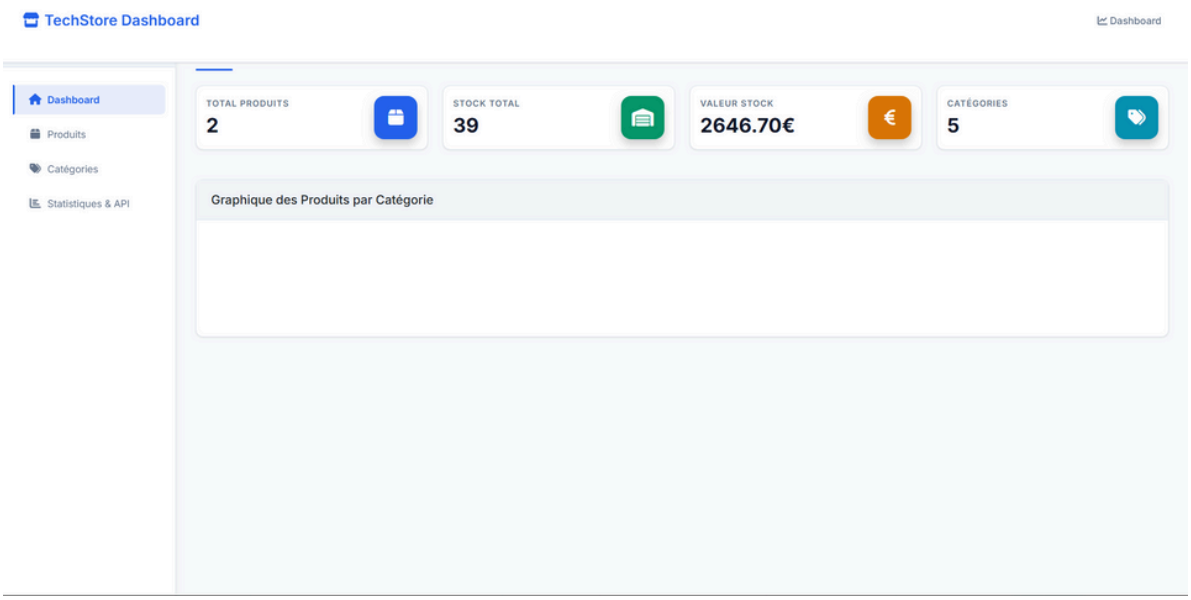
Problème : Sidebar fixe qui empiète sur le contenu mobile

Solution :

- Sidebar cachée par défaut sur mobile
- Bouton toggle visible uniquement sur petit écran
- Overlay sombre pour fermer la sidebar
- Transitions CSS fluide

7. Résultats et captures d'écran

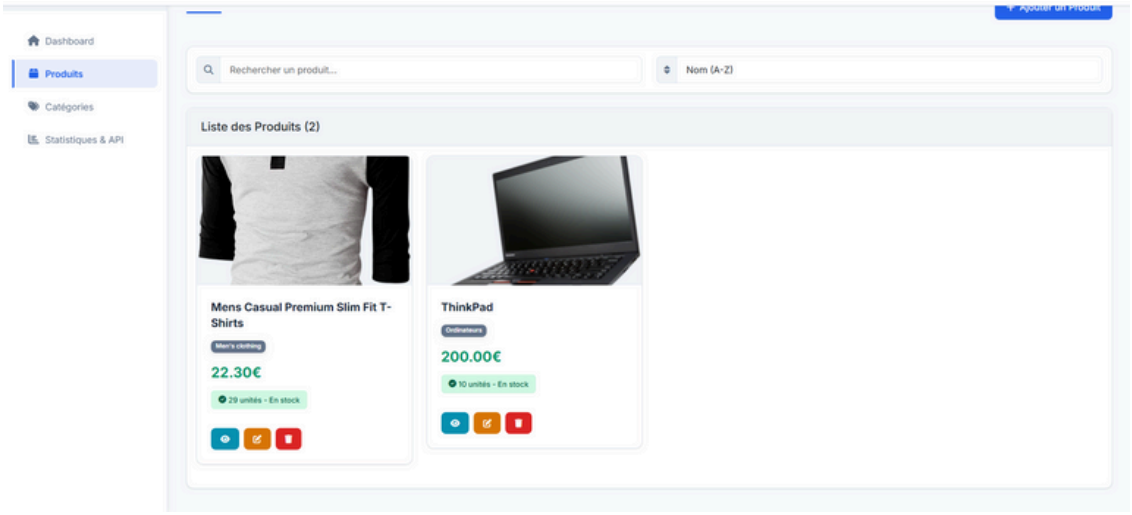
7.1 Interface Dashboard



Le dashboard présente une interface moderne avec :

- 4 KPIs colorés en haut (Produits, Stock, Valeur, Catégories)
- Graphique en barres Chart.js pour la répartition
- Section d'import API avec bouton de chargement
- Design glassmorphism avec effets de transparence

7.2 Gestion des Produits



Liste des produits en grille

- Cartes produits avec images
- Badges de stock colorés (vert > 10, orange < 10, rouge = 0)
- Prix en grand format
- Boutons d'action (Voir, Modifier, Supprimer)
- Barre de recherche en temps réel
- Menu déroulant de tri

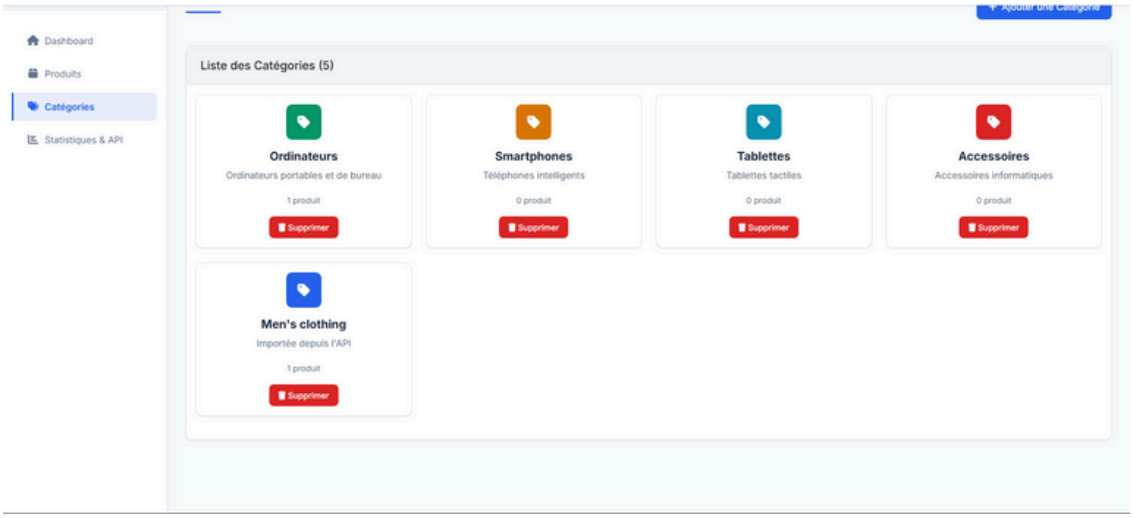
Formulaire d'ajout/modification

- Champs avec validation visuelle
- Sélection de catégorie dynamique
- Upload d'image par URL
- Zone de texte pour description
- Bouton d'action adaptatif (Ajouter/Modifier)

Modal de détails

- Affichage grand format de l'image
- Toutes les informations du produit
- Design responsive avec Bootstrap modal

7.3 Gestion des Catégories



Liste des catégories

- Cartes colorées par catégorie
- Icônes Font Awesome spécifiques
- Compteur de produits liés
- Bouton de suppression avec état conditionnel
- Grid responsive 3 colonnes

8. Conclusion

8.1 Objectifs atteints

Développement d'une SPA fonctionnelle

- Navigation fluide sans rechargement
- Architecture modulaire claire
- Code organisé et maintenable

Module CRUD Produits complet

- Création, lecture, modification, suppression
- Recherche et tri multi-critères
- Validation des données
- Gestion des images

Module Catégories avec synchronisation

- Gestion simplifiée des catégories
- Protection contre suppression inappropriée
- Mise à jour automatique des sélecteurs

Dashboard interactif

- KPIs en temps réel
- Visualisation avec Chart.js
- Intégration API FakeStore
- Statistiques automatiques

8.2 Compétences développées

JavaScript avancé :

- Programmation orientée objet (classes ES6)
- Modules ES6 et architecture modulaire
- Programmation asynchrone (async/await)
- Manipulation du DOM
- Gestion d'événements
- LocalStorage API

Intégration d'APIs :

- Fetch API pour requêtes HTTP
- Gestion d'erreurs réseau
- Parsing JSON
- Transformation de données

Frontend moderne :

- HTML5 sémantique
- CSS3 avancé (animations, gradients, glassmorphism)
- Bootstrap 5 pour composants UI
- Responsive design mobile-first
- Chart.js pour visualisation

Bonnes pratiques :

- Code propre et commenté
- Séparation des responsabilités
- Gestion d'erreurs robuste
- Validation des données
- Expérience utilisateur (UX)

8.3 Perspectives d'amélioration

Fonctionnalités futures :

- Système d'authentification : Login/logout avec gestion des sessions
- Gestion des ventes : Module de facturation et historique des ventes
- Statistiques avancées : Graphiques de tendances, analyses prédictives
- Notifications : Alertes de stock faible, notifications push
- PWA : Application web progressive installable
- Multilingue : Support de plusieurs langues
- Thèmes : Mode sombre/clair personnalisable
- Export PDF : Génération de rapports PDF

Améliorations techniques :

- **Backend :** API REST avec Node.js/Express pour persistance serveur
- **Sécurité :** JWT pour authentification, validation côté serveur
- **Performance :** Pagination pour grandes listes, lazy loading des images
- **Tests :** Tests unitaires avec Jest, tests E2E avec Cypress
- **Build :** Webpack/Vite pour bundling et optimisation
- **Déploiement :** CI/CD avec GitHub Actions, hébergement Vercel/Netlify

UX/UI :

- **Filtres avancés** : Filtrage multi-critères combiné
- **Recherche intelligente** : Suggestions, recherche floue
- **Actions groupées** : Suppression/modification en masse
- **Import/Export** : CSV, Excel pour données
- **Galerie d'images** : Upload multiple, gestion d'albums
- **Tableaux interactifs** : Tri, pagination, colonnes personnalisables

9. Annexes

A. Structure complète du code

```
projetjs/
├─── index.html          # Page principale SPA (394 lignes)
│   └─── Sections: Dashboard, Produits, Catégories
│
├─── css/
│   └─── styles.css       # Styles personnalisés (966 lignes)
│     ├─── Variables CSS
│     ├─── Styles généraux
│     ├─── Sidebar & Navigation
│     ├─── Cartes & Composants
│     ├─── Animations
│     └─── Media queries responsive
│
├─── js/
│   ├─── app.js          # Application principale (113 lignes)
│   │   └─── Classe App : Navigation SPA
│   │
│   ├─── modules/
│   │   ├─── produits.js     # Gestion produits (373 lignes)
│   │   │   ├─── Classe ProduitsManager
│   │   │   ├─── CRUD complet
│   │   │   ├─── Recherche & Tri
│   │   │   └─── Validation
│   │   │
│   │   ├─── categories.js    # Gestion catégories (181 lignes)
│   │   │   ├─── Classe CategoriesManager
│   │   │   ├─── CRUD simplifié
│   │   │   └─── Synchronisation
│   │   │
│   │   └─── dashboard.js     # Dashboard & API (252 lignes)
│   │     ├─── Classe Dashboard
│   │     ├─── KPIs
│   │     ├─── Chart.js
│   │     └─── API FakeStore
│   │
│   └─── utils/
│     └─── storage.js     # Gestion LocalStorage (118 lignes)
│       ├─── Classe StorageManager
│       ├─── CRUD Produits & Catégories
│       └─── Statistiques
│
└─── README.md          # Documentation (107 lignes)
```

B. Guide d'installation

Prérequis :

- Navigateur web moderne (Chrome, Firefox, Safari, Edge)
- Éditeur de code (VS Code recommandé)
- Live Server ou serveur web local

Installation :

1. Cloner le projet

bash
git clone [URL_DU_DEPOT]
cd projetjs

1. Ouvrir avec Live Server

bash
VS Code : clic droit sur index.html > "Open with Live Server"
ou utiliser un serveur local
python -m http.server 8000

1. Accéder à l'application

http://localhost:8000
Aucune dépendance npm à installer - Toutes les bibliothèques sont chargées via CDN.

C. Configuration des bibliothèques CDN

Bootstrap 5.3.0 -->
https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css
https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js

Font Awesome 6.4.0 -->
https://cdnjs.cloudflare.com/ajax/libs/fontawesome/6.4.0/css/all.min.css

Chart.js -->
https://cdn.jsdelivr.net/npm/chart.js

Google Fonts -->
https://fonts.googleapis.com/css2family=Inter:wght@300;400;500;600;700&display=swap

D. API FakeStore - Documentation

Endpoint utilisé :
GET https://fakestoreapi.com/products
Réponse (exemple) :

```json
[
  {
    "id": 1,
    "title": "Fjallraven - Foldsack No. 1 Backpack",
    "price": 109.95,
    "description": "Your perfect pack for everyday use...",
    "category": "men's clothing",
    "image": "https://fakestoreapi.com/img/81fPKd-2AYL._AC_SL1500_.jpg"
  }
]
```

Traitement dans l'application :

- Conversion USD ↠ DH (x10 approximatif)
- Mapping des catégories API vers catégories locales
- Génération de stock aléatoire (10-60 unités)
- Création automatique des catégories manquantes

E. Structure LocalStorage

Clés utilisées :

```javascript
// Données
localStorage.getItem('products')          // Array<Product>
localStorage.getItem('categories')        // Array<Category>

// Compteurs d'IDs
localStorage.getItem('nextProductId')     // Number
localStorage.getItem('nextCategoryId')    // Number

```

Format des données :

```javascript
// Product
{
  id: 1,
  nom: "MacBook Pro",
  categorie: "Ordinateurs",
  prix: 25000,
  stock: 15,
  image: "https://...",
  description: "...",
  createdAt: "2026-01-15T10:30:00.000Z"
}

// Category
{
  id: 1,
  nom: "Ordinateurs",
  description: "Ordinateurs portables et de bureau"
}
```

Références et ressources

Documentation officielle :

- MDN Web Docs - JavaScript
- Bootstrap 5 Documentation
- Chart.js Documentation
- Font Awesome Icons

APIs utilisées :

- FakeStore API
- LocalStorage API

Concepts appliqués :

- Single Page Application (SPA)
- ES6 Modules
- Async/Await
- Glassmorphism Design