|  |  |  |
| --- | --- | --- |
| C:\wamp\www\PROJET\fsts.png |  |  |

**UNIVERSITE HASSAN 1ER**

**FACULTE DES SCIENCES ET TECHNIQUES**

**-SETTAT-**

**Rapport de Projet de fin d’études**

Pour l’Obtention du Diplôme

**LICENCE SCIENCES ET TECHNIQUES**

**Spécialité : Génie Informatique**

**Ecomerce Dashboard administrative**

Réalisé par :t

**Youssef Taoussi**

Sous la direction de :

**M. Said EL KAFHALI,** professeur à laFST de Settat

**M. Noureddine DARKAOUI,** ACOS Academy

*Soutenu le :***12 juin 2025**

**JURY**

**Haitam ETTAZI** professeur à la FST de Settat

**Pr. Fatima Ezzahraa BENBOUAZZA** professeur à la FST de Settat

**Pr. Doha MALKI**, professeur à la FST de Settat

**Année Universitaire : 2024-2025**

Table of Contents

[Dédicace 4](#__RefHeading___Toc2940_950592567)

[**CHAPITRE 1** : Contexte générale du projet 5](#__RefHeading___Toc2794_749061618)

[Presentation et context de project  5](#__RefHeading___Toc2796_749061618)

[Context  6](#__RefHeading___Toc2798_749061618)

[Organigramme  7](#__RefHeading___Toc2800_749061618)

[Critique de l’existant 7](#__RefHeading___Toc2802_749061618)

[Problématique 7](#__RefHeading___Toc2804_749061618)

[Objectifs du projet 8](#__RefHeading___Toc2806_749061618)

[Solution proposée  9](#__RefHeading___Toc2808_749061618)

[Conduite du projet / Méthodologie de travail 10](#__RefHeading___Toc2818_749061618)

[Planification du projet / Planning prévisionnel 11](#__RefHeading___Toc2834_749061618)

[Conclusion 12](#__RefHeading___Toc2838_749061618)

[**Chapitre II :** Analyse et conception 13](#__RefHeading___Toc2840_749061618)

[Besoins fonctionnels : 13](#__RefHeading___Toc2842_749061618)

[Identification et description des acteurs : 14](#__RefHeading___Toc2844_749061618)

[Diagramme de Cas d’Utilisation 14](#__RefHeading___Toc2846_749061618)

[Diagrammes de Séquences 15](#__RefHeading___Toc2848_749061618_Copy_1)

[Diagramme d’activités – « Accès et utilisation du Dashboard Admin » 16](#__RefHeading___Toc2850_749061618)

[Diagrame des class 16](#__RefHeading___Toc2852_749061618)

[Besoins non fonctionnels / techniques 17](#__RefHeading___Toc2854_749061618)

[Performance 17](#__RefHeading___Toc2856_749061618)

[Sécurité 17](#__RefHeading___Toc2858_749061618)

[La sécurité est essentielle pour un système manipulant des données sensibles : 17](#__RefHeading___Toc2942_950592567)

[Maintenance 18](#__RefHeading___Toc2944_950592567)

[Portabilité 18](#__RefHeading___Toc2862_749061618)

[**Chapitre III :** Etude technique 20](#__RefHeading___Toc2864_749061618)

[Présentation des technologies utilisées : 21](#__RefHeading___Toc2866_749061618)

[Choix du SGBDR : MySQL 22](#__RefHeading___Toc2888_749061618)

[Les outils de développement utilisés 23](#__RefHeading___Toc2890_749061618)

[**Chapitre IV :** Réalisation/Mise en œuvre de la solution 25](#__RefHeading___Toc2892_749061618)

[Structure du projet 26](#__RefHeading___Toc2894_749061618)

[Schéma de la base de données 29](#__RefHeading___Toc2924_749061618)

[Présentation des interfaces de l’application /Captures écrans  30](#__RefHeading___Toc2926_749061618)

[Page de connexion (Login) : 30](#__RefHeading___Toc2134_3647396143)

[Présentation de l’interface : Page Dashboard 31](#__RefHeading___Toc2136_3647396143)

[Présentation de l’interface : Page Tous les Produits (All Products) 32](#__RefHeading___Toc2138_3647396143)

[Présentation de l'interface : Page de Création de Produit 33](#__RefHeading___Toc2140_3647396143)

[Présentation de l'interface : Page de Création de Catégorie 35](#__RefHeading___Toc2142_3647396143)

[Présentation de l'interface : Page de Détail du Produit 37](#__RefHeading___Toc2144_3647396143)

[Présentation de l'interface : Page de Détail de Catégorie 39](#__RefHeading___Toc2146_3647396143)

[Présentation de l'interface : Page des commandes (orders) 41](#__RefHeading___Toc2148_3647396143)

[Présentation de l'interface : Page de Détail de détail du commande 43](#__RefHeading___Toc2150_3647396143)

[Présentation de l'interface : Page des clients (customers) 45](#__RefHeading___Toc2152_3647396143)

[Présentation de l'interface : Page de détail de client (customer detail) 46](#__RefHeading___Toc2152_3647396143_Copy_1)

[Tests Réalisés 48](#__RefHeading___Toc2946_950592567)

[La qualité du code 52](#__RefHeading___Toc2948_950592567)

[Évaluation de la Qualité du Code Front-end 52](#__RefHeading___Toc2950_950592567)

[Évaluation de la Qualité du Code Back-end 53](#__RefHeading___Toc2952_950592567)

## ****Dédicace****

Je dédie ce projet à ma famille et à mes amis qui m'ont soutenu tout au long de cette aventure. Leur présence, leurs encouragements, leur patience et leur confiance en moi ont été une source précieuse de motivation. À ma famille, pour leur amour inconditionnel et leur soutien moral dans les moments les plus difficiles. À mes amis, pour leur aide, leurs conseils et leur bonne humeur, qui ont rendu ce parcours plus léger et agréable. Ce travail est aussi le fruit de votre présence à mes côtés. Merci du fond du cœur.

# **CHAPITRE 1 :** **Contexte générale du projet**

## Presentation et context de project

Avec l'essor du commerce en ligne au cours de la dernière décennie, les plateformes e-commerce se sont multipliées, devenant des outils essentiels pour les entreprises souhaitant élargir leur portée et mieux répondre aux attentes de leurs clients. Toutefois, cette croissance rapide s’accompagne d’un besoin croissant de **maîtrise, d’analyse et de gestion des données** générées quotidiennement par les interactions des utilisateurs, les ventes, les livraisons, les produits et les retours. Afin de répondre à ces enjeux, les **dashboards (tableaux de bord)** sont devenus des éléments incontournables dans les systèmes de gestion modernes.

Ce projet s’inscrit dans cette logique et consiste à concevoir et développer un **dashboard administratif pour une application e-commerce**. L’objectif principal est de fournir aux administrateurs une interface centralisée, intuitive et performante permettant d’accéder rapidement à des indicateurs clés de performance (KPI), de visualiser l’état global de l’activité commerciale, et de faciliter la prise de décisions stratégiques. Le dashboard intègre des modules essentiels comme la gestion des produits, des utilisateurs, des commandes, des revenus, et des statistiques avancées de ventes. Il permet également de suivre l'évolution de l'activité grâce à des graphiques dynamiques, des filtres de recherche, et des outils d’analyse en temps réel.

La réalisation de ce projet repose sur l’utilisation de technologies modernes adaptées aux besoins du développement web. Le backend a été développé en utilisant le framework **Laravel**, connu pour sa robustesse, sa sécurité, et sa facilité de maintenance. Pour le frontend, des technologies comme **React** (ou Blade selon le cas), associées à des bibliothèques de visualisation telles que **Chart.js** ou **Recharts**, ont été utilisées afin d’offrir une expérience utilisateur fluide et agréable. Le projet intègre également une base de données relationnelle (comme MySQL), permettant de stocker et organiser efficacement l’ensemble des données nécessaires au bon fonctionnement du tableau de bord.

Ce rapport retrace toutes les étapes de la réalisation du projet, en commençant par l’analyse des besoins fonctionnels, la modélisation de la base de données, la conception de l’interface utilisateur, jusqu’à l’implémentation technique et les tests. Il vise à présenter les choix technologiques effectués, les difficultés rencontrées ainsi que les solutions apportées, dans le but de fournir une plateforme fiable, évolutive et adaptée aux exigences réelles du commerce électronique.

## Context

Avec la croissance continue du commerce en ligne, les entreprises ont besoin d’interfaces de gestion efficaces, ergonomiques et sécurisées. Les dashboards e-commerce permettent aux administrateurs de superviser les commandes, gérer les stocks, suivre les ventes et avoir une vue d’ensemble sur l’activité. Cependant, les solutions existantes sont souvent :

* + complexes à utiliser pour des petites structures,
  + peu personnalisables,
  + ou payantes.

Ainsi, j’ai souhaité développer une solution **moderne, intuitive et personnalisable**, intégrant :

* une architecture **full stack**,
* un design responsive basé sur **React.js** et **Tailwind CSS** (avec **ShadCN UI….**),
* un backend avec **PHP, Larvel**,
* une base de données relationnelle **MySQL** .
* L’objectif de ce projet est double :

**Mettre en pratique** les compétences acquises durant la formation (conception, développement, architecture logicielle, UX/UI…

**Proposer une base solide** pour une future application e-commerce réelle ou comme produit open-source.

## ****Organigramme****

Le projet ayant été réalisé dans le cadre d’un travail personnel validé par un encadrant académique, l’organigramme du projet est simplifié comme suit :

* **Encadrant universitaire** : Responsable du suivi pédagogique du projet, émet des recommandations, valide les étapes du projet.
* **Étudiant développeur (moi-même)** : Responsable de l’ensemble des tâches liées au projet, à savoir :
  + L’analyse des besoins fonctionnels et techniques,
  + La conception de l’architecture logicielle et de la base de données,
  + Le développement du frontend et du backend,
  + La mise en œuvre des interfaces utilisateur,
  + La rédaction de la documentation technique et du rapport final

## ****Critique de l’existant****

Il existe aujourd’hui plusieurs plateformes e-commerce comme **Shopify**, **WooCommerce** ou **Prestashop** qui proposent des interfaces d’administration. Cependant, ces solutions ne sont pas toujours adaptées pour :

* Les petits projets ou les développeurs débutants,
* Les besoins simples et personnalisés,
* L’apprentissage et la mise en pratique des compétences techniques.

De plus, ces outils sont souvent :

* Complexes à utiliser,
* Difficiles à personnaliser,
* Payants ou avec des fonctionnalités limitées en version gratuite.

Dans le cadre d’un projet de fin d’études, utiliser ces plateformes ne permet pas de montrer ses compétences en développement. C’est pourquoi j’ai choisi de créer **mon propre dashboard e-commerce** pour gérer les produits, les clients et les commandes.

## ****Problématique****

Dans un contexte où le commerce en ligne occupe une place de plus en plus importante, les entreprises ont besoin d’**outils de gestion efficaces** pour piloter leurs activités. Les tableaux de bord proposés par les solutions e-commerce existantes sont souvent **complexes**, **coûteux**, ou **difficiles à personnaliser** pour des besoins spécifiques, notamment dans les petites structures ou les projets pédagogiques.

Face à cette situation, se pose la question suivante :

**Comment développer une solution web personnalisée et accessible, permettant à un administrateur de gérer de manière centralisée les produits, les clients et les commandes d’une boutique en ligne, tout en assurant une bonne ergonomie, une visualisation claire des données, et une architecture logicielle robuste ?**

Cette problématique implique des défis à la fois techniques (architecture full-stack, sécurité, performance) et fonctionnels (interface intuitive, gestion fluide des entités, visualisation de statistiques).

## Objectifs du projet

L’objectif principal de ce projet est de concevoir et développer un **dashboard e-commerce d’administration** complet et performant, permettant à un administrateur de gérer efficacement les produits, clients et commandes d’une boutique en ligne.

Les objectifs spécifiques sont :

* **Concevoir une interface utilisateur ergonomique et robuste** avec **React** en **TypeScript**, pour bénéficier d’un typage statique, d’une meilleure maintenabilité et d’une expérience développeur améliorée. Le design est moderne et responsive, avec l’utilisation de **Tailwind CSS** et **Shadcn UI**
* **Développer un backend sécurisé et performant** avec **Laravel**, qui expose une **API RESTful** pour gérer la logique métier, l’authentification, la validation des données, et la persistance dans une base de données relationnelle **MySQL**.
* **Assurer une séparation claire entre frontend et backend** via des API REST, facilitant la maintenance et la scalabilité.
* **Implémenter les fonctionnalités essentielles** : gestion complète des produits (création, modification, suppression, inventaire), gestion des commandes (suivi, statuts, filtres) et administration des clients (profil, historique d’achats).
* **Afficher des indicateurs clés de performance (KPI)** dans un tableau de bord interactif pour une visualisation synthétique des ventes, commandes, et autres données importantes.
* **Appliquer les bonnes pratiques** de développement web full stack, notamment en termes de sécurité (authentification JWT ou Sanctum), validation, tests unitaires et fonctionnels, ainsi que documentation technique.
* Ce projet permet ainsi de mettre en œuvre des compétences avancées en développement full stack moderne, avec des technologies largement utilisées en industrie.
* **Respecter les bonnes pratiques de développement**, notamment en matière de sécurité (authentification, gestion des sessions, validation des données), de tests et de documentation.

Ce projet vise à la fois un usage pédagogique pour valider les compétences acquises et un prototype fonctionnel pouvant servir de base à un futur produit ou projet personnel.

## Solution proposée

Pour répondre à la problématique posée, j’ai développé une solution complète basée sur une architecture **full stack moderne** qui sépare clairement la partie frontend et backend, afin d’assurer modularité, maintenance et évolutivité.

**Frontend**

* **Technologie** : React avec **TypeScript**  
  L’utilisation de TypeScript permet d’améliorer la robustesse du code grâce à un typage statique, facilitant la détection d’erreurs dès la phase de développement.
* **Design** : Interface utilisateur moderne, responsive et intuitive, réalisée avec Tailwind CSS, offrant une expérience utilisateur fluide sur desktop et mobile.
* **Fonctionnalités** :
  + Gestion des produits, clients et commandes via des formulaires dynamiques.
  + Tableau de bord affichant les indicateurs clés de performance (ventes, commandes en cours, statistiques clients).
  + Recherche, filtres et pagination pour faciliter la navigation dans les listes.
  + Authentification sécurisée avec gestion des sessions utilisateur.

**Backend**

* **Technologie** : Laravel (PHP)  
  **Laravel** fournit un cadre structuré et sécurisé pour créer une **API RESTful**, responsable de la logique métier, la gestion des utilisateurs, la validation des données et la communication avec la base de données.
* **API RESTful** :  
  Le backend expose des endpoints pour le frontend, permettant les opérations **CRUD** (Create, Read, Update, Delete) sur les entités principales (produits, clients, commandes).
* **Sécurité** :  
  Authentification sécurisée via **JWT**, contrôle d’accès basé sur les rôles, protection contre les attaques courantes (injection SQL, etc.).
* **Base de données** :  
  Utilisation d’une base de données relationnelle **MySQL** pour stocker les données de façon structurée et fiable.

**Architecture**

* **Séparation Frontend / Backend** :  
  Le frontend et le backend communiquent via des appels API, permettant une indépendance technologique et facilitant la maintenance et les évolutions futures.
* **Modularité et évolutivité** :  
  Le code est organisé en modules clairs, facilitant l’ajout de nouvelles fonctionnalités (ex : gestion des promotions, notifications, etc.).

**Avantages de la solution**

* Adaptée aux besoins réels d’un administrateur e-commerce.
* Solution personnalisée, légère et évolutive.
* Utilisation de technologies modernes et reconnues, permettant un apprentissage approfondi et une application professionnelle.
* Interface utilisateur claire et agréable, simplifiant la gestion quotidienne.

## Conduite du projet / Méthodologie de travail

Pour mener à bien ce projet, j’ai adopté une méthodologie agile et itérative permettant de livrer un produit fonctionnel rapidement tout en restant flexible face aux évolutions et aux retours.

#### Analyse des besoins

* + - Identification des fonctionnalités principales (gestion produits, commandes, clients).
    - Rédaction d’un cahier des charges simplifié pour cadrer les objectifs et le périmètre.
    - Priorisation des fonctionnalités selon leur valeur métier.

#### Conception

* + - Modélisation de la base de données schéma relationnel MySQL.
    - Conception des maquettes UI / UX sous forme de wireframes pour valipder l’ergonomie.
    - Définition de l’architecture technique (séparation frontend/backend, choix des technologies).

#### Développement itératif

* + - Découpage du projet en sprints de 1 à 2 semaines.
    - Développement du backend avec Laravel pour exposer les APIs REST nécessaires.
    - Développement parallèle du frontend avec React + TypeScript, consommation des APIs.
    - Intégration progressive des fonctionnalités, tests unitaires et fonctionnels.

#### Tests et validation

* + - Tests manuels pour vérifier la conformité aux besoins.
    - Correction des bugs et ajustements UI/UX.
    - Tests de sécurité et validation des accès (authentification, rôles).

#### Documentation

* + - Rédaction d’une documentation technique pour faciliter la maintenance.
    - Guide utilisateur simple pour la prise en main de l’application.

#### Déploiement

* + - Mise en production sur un serveur adapté (hébergement Laravel + frontend build).
    - Configuration des environnements (production, développement).
    - Surveillance post-déploiement pour détecter d’éventuels problèmes.

#### Suivi et amélioration continue

* + - Recueil des retours utilisateurs pour identifier des pistes d’amélioration.
    - Planification d’évolutions futures selon les besoins métiers.

## Planification du projet / Planning prévisionnel

Le projet a été découpé en 5 grandes phases, chacune avec des tâches précises et une durée estimée. Ce découpage m’a permis de suivre une progression logique et de garder une bonne visibilité sur l’avancement.

|  |  |  |
| --- | --- | --- |
| ****Phase**** | **Tâches principales** | **Tâches principales** |
| **1. Préparation** | - Analyse des besoins  - Cahier des charges  - Choix des technologies | 1 semain |
| **2 . Conception** | - Maquettes UI (figma ou autre, Dribble)  - Modélisation de la base de données  - Architecture de l’application (front + back) | 1 semaine |
| **3**. **Développement Backend** | - Création du projet Laravel  - Mise en place de l’authentification  - Création des routes et contrôleurs API  - Gestion base de données (migrations, seeders) | 2 semaines |
| **4. Développement Frontend** | - Initialisation du projet React avec TypeScript  - Connexion aux API Laravel  - Création des composants (produits, clients, commandes, tableau de bord, etc.)  - Navigation, validations, messages d’erreur | 2 à 3 semaines |
| **5. Tests & Déploiement** | - Tests manuels et débogage  - Améliorations UI/UX  - Documentation utilisateur + technique  - Déploiement final | 1 semaine |

### Outils de suivi

* **Git** : suivi des versions du code, branches pour chaque fonctionnalité.

## ****Conclusion****

Ce premier chapitre a permis de situer le projet dans son environnement global. Nous avons présenté l’organisme d’accueil ainsi que son organisation, ses activités principales et son secteur d’intervention. Ensuite, nous avons détaillé le contexte dans lequel s’inscrit le projet, en mettant en évidence les limites de l’existant, les besoins identifiés, ainsi que la problématique à résoudre.

À partir de cette analyse, nous avons défini les objectifs à atteindre et proposé une solution adaptée, fondée sur une architecture moderne combinant Laravel pour le backend et React avec TypeScript pour le frontend. La conduite du projet s’est appuyée sur une méthodologie agile et un découpage en phases bien structurées, accompagnées d’un planning prévisionnel clair et réaliste.

Ce cadre général nous permettra d’aborder dans le chapitre suivant les aspects techniques liés à la conception et au développement de l’application, en assurant une continuité logique dans la présentation du travail effectué.

# **Chapitre II : Analyse et conception**

## Besoins fonctionnels :

Les besoins fonctionnels décrivent les services que le système doit offrir à ses utilisateurs. Dans le cadre de notre projet, les besoins fonctionnels identifiés sont les suivants :

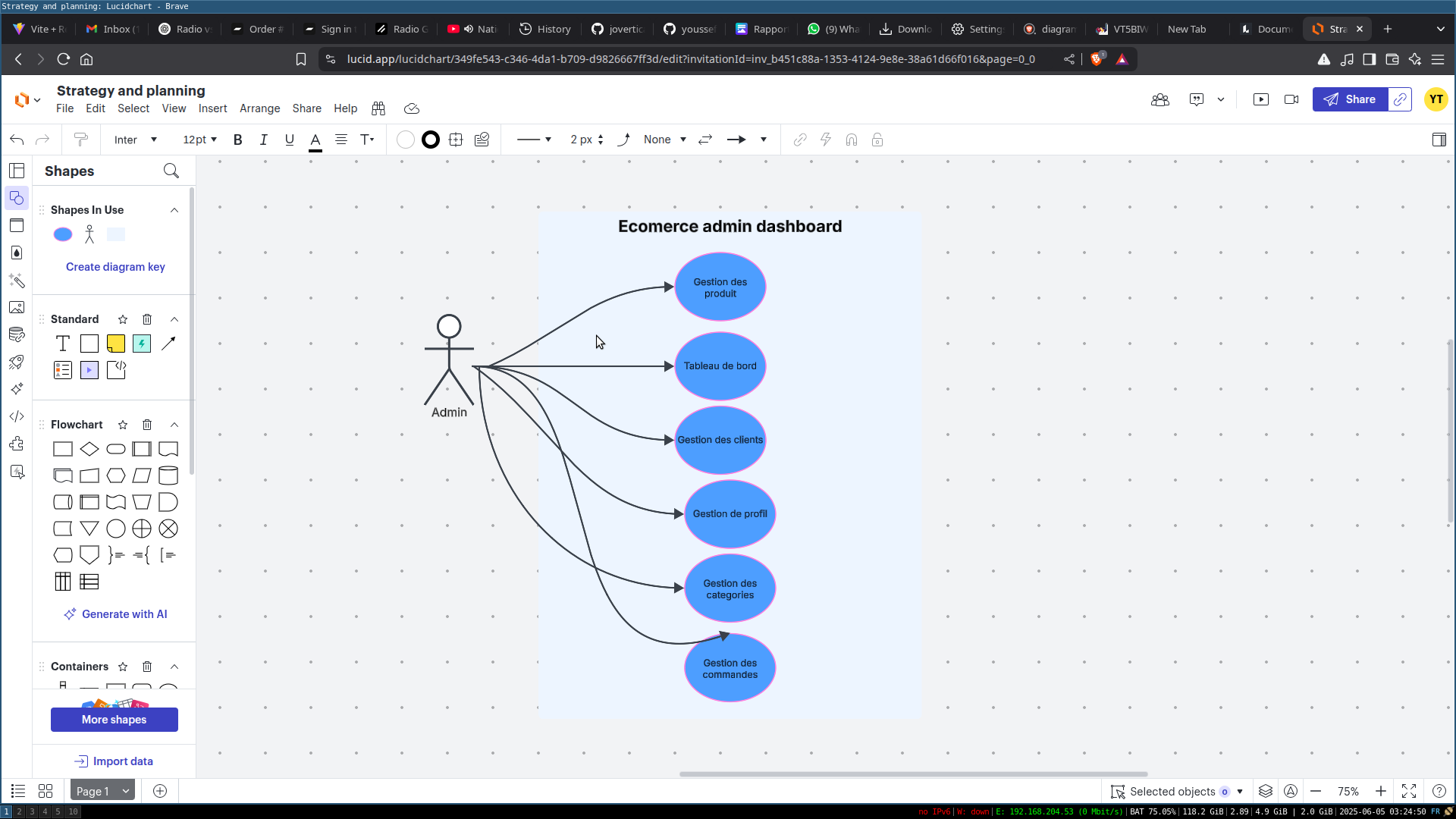
* Authentification
* Gestion des produits : ajout, modification, suppression, visualisation.
* Gestion des commandes : suivi des commandes, changement de statut.
* Gestion des clients : informations personnelles, historique d’achat.
* Tableau de bord pour l’administration avec des statistiques (ventes, utilisateurs actifs…).
* Interface utilisateur fluide et responsive.

#### Identification et description des acteurs :

|  |  |
| --- | --- |
| **Acteur** | **Description** |
| **Administrateur** | Gère les utilisateurs, produits, commandes, statistiques… |

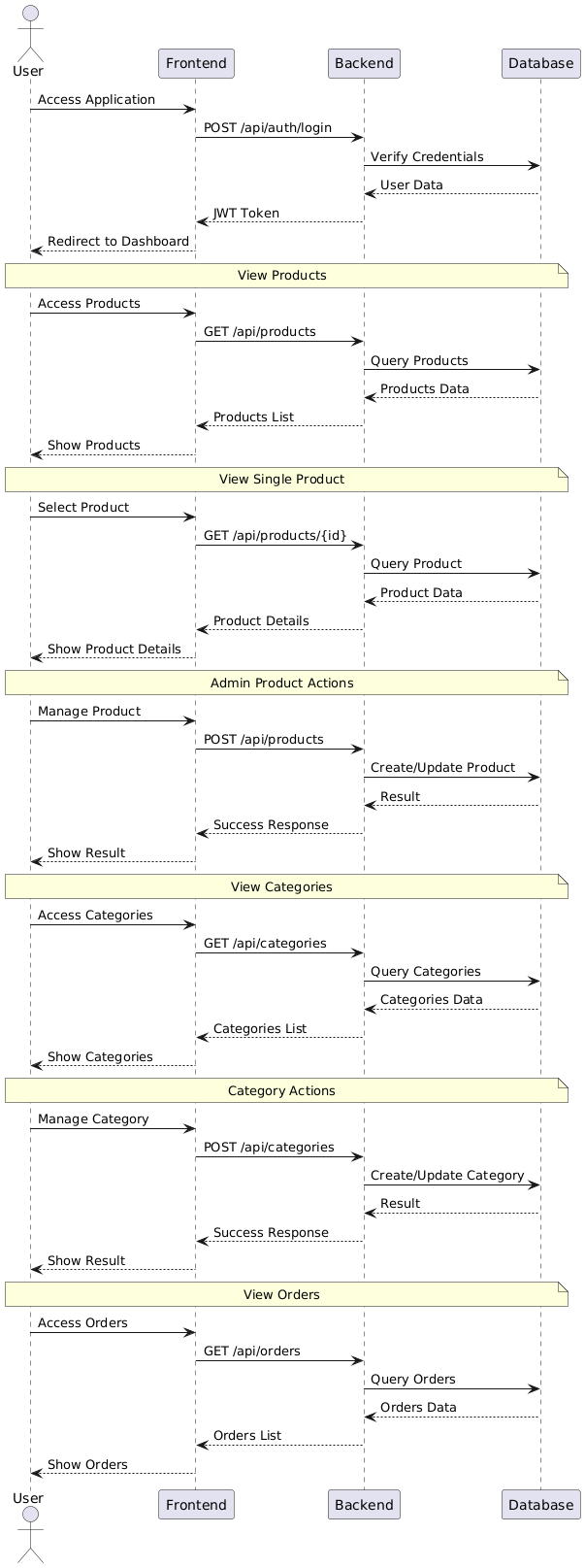
#### ****Diagramme de Cas d’Utilisation****

Le **diagramme de cas d'utilisation** (Use Case Diagram) est un outil de modélisation UML (Unified Modeling Language) permettant de représenter les interactions entre les **acteurs** dusystème et les **fonctionnalités (cas d'utilisation)** qu’ils peuvent exécuter. Il permet de visualiser **qui fait quoi** dans le système de manière simple et intuitive.



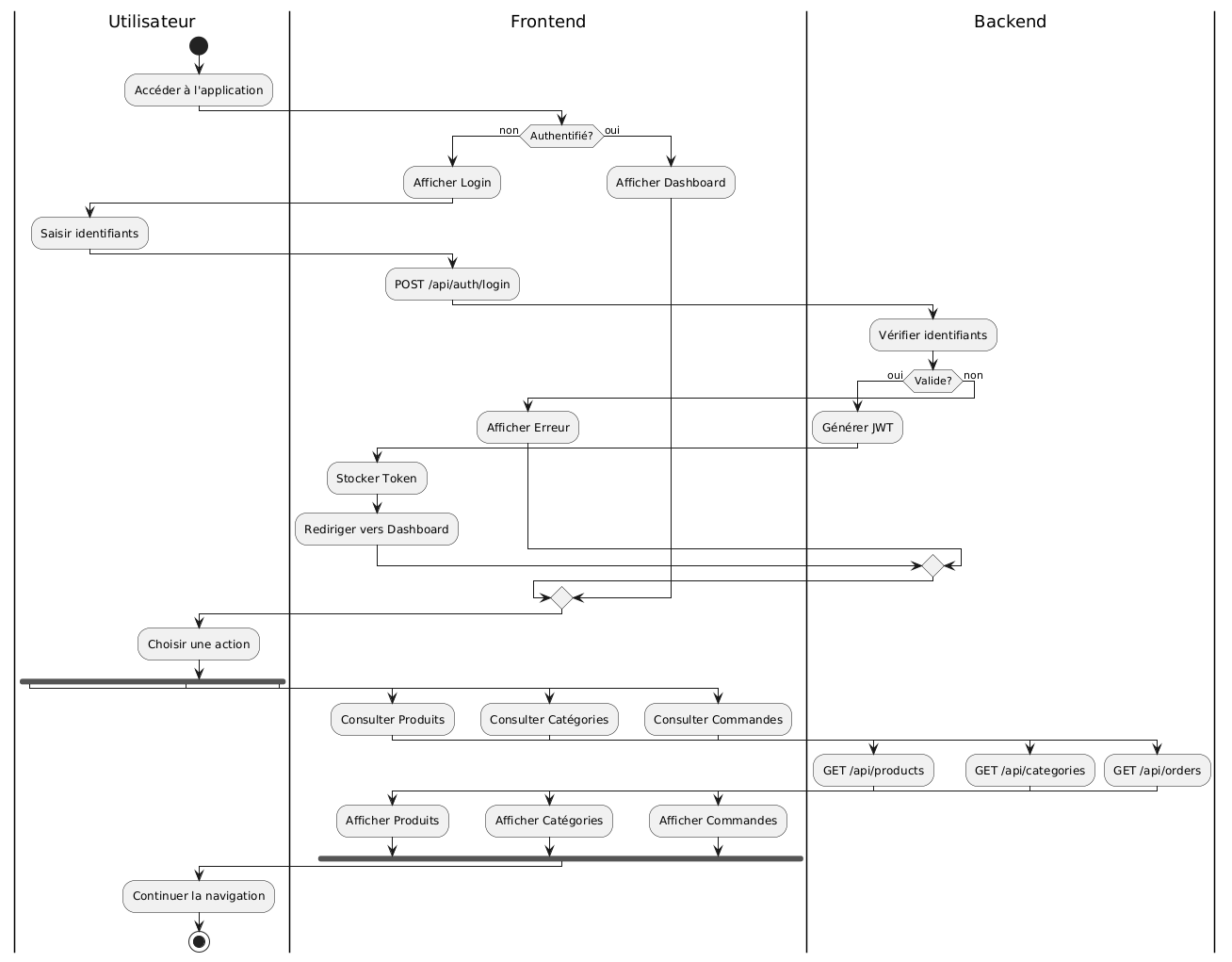
#### ****Diagrammes de Séquences****

**diagramme de séquence est un diagramme UML qui décrit l’enchaînement des messages échangés entre les objets ou composants d’un système au cours d’unscénario précis. Il illustre l’ordre chronologique des interactions et permet de comprendre la logique du traitement d’un cas d’utilisation.**



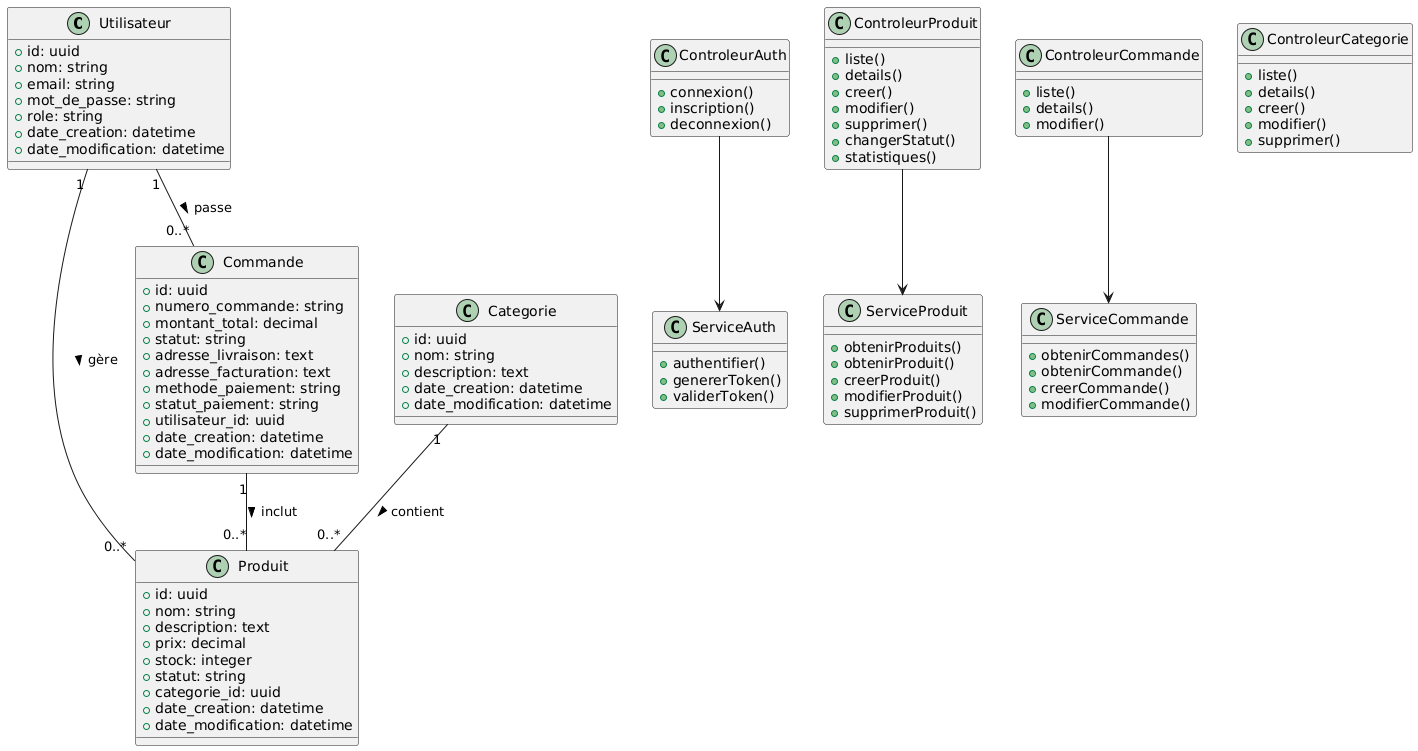
#### Diagramme d’activités – « Accès et utilisation du Dashboard Admin »

Ce diagramme montre les différentes étapes et décisions liées à l'accès et à l'utilisation du dashboard par l'administrateur.



#### Diagrame des class

Le **diagramme de classes** est un diagramme structurel de l'UML (Unified Modeling Language) qui permet de représenter **les différentes classes du système**, ainsi que **leurs relations**, **attributs** et **méthodes**.  
Il décrit **la structure statique du système** et sert de base pour la conception orientée objet.



## Besoins non fonctionnels / techniques

#### ****Performance****

* Le système doit être rapide et réactif :
* Le tableau de bord admin doit afficher les données (utilisateurs, commandes, produits, etc.) en **moins de 2 secondes**.
* L’utilisation de React avec TypeScript garantit une **interface dynamique**, réactive et fluide grâce au DOM virtuel.
* L’API Laravel est optimisée avec un **chargement paresseux** (lazy loading) et la **pagination** pour réduire les requêtes massives en base de données.

#### ****Sécurité****

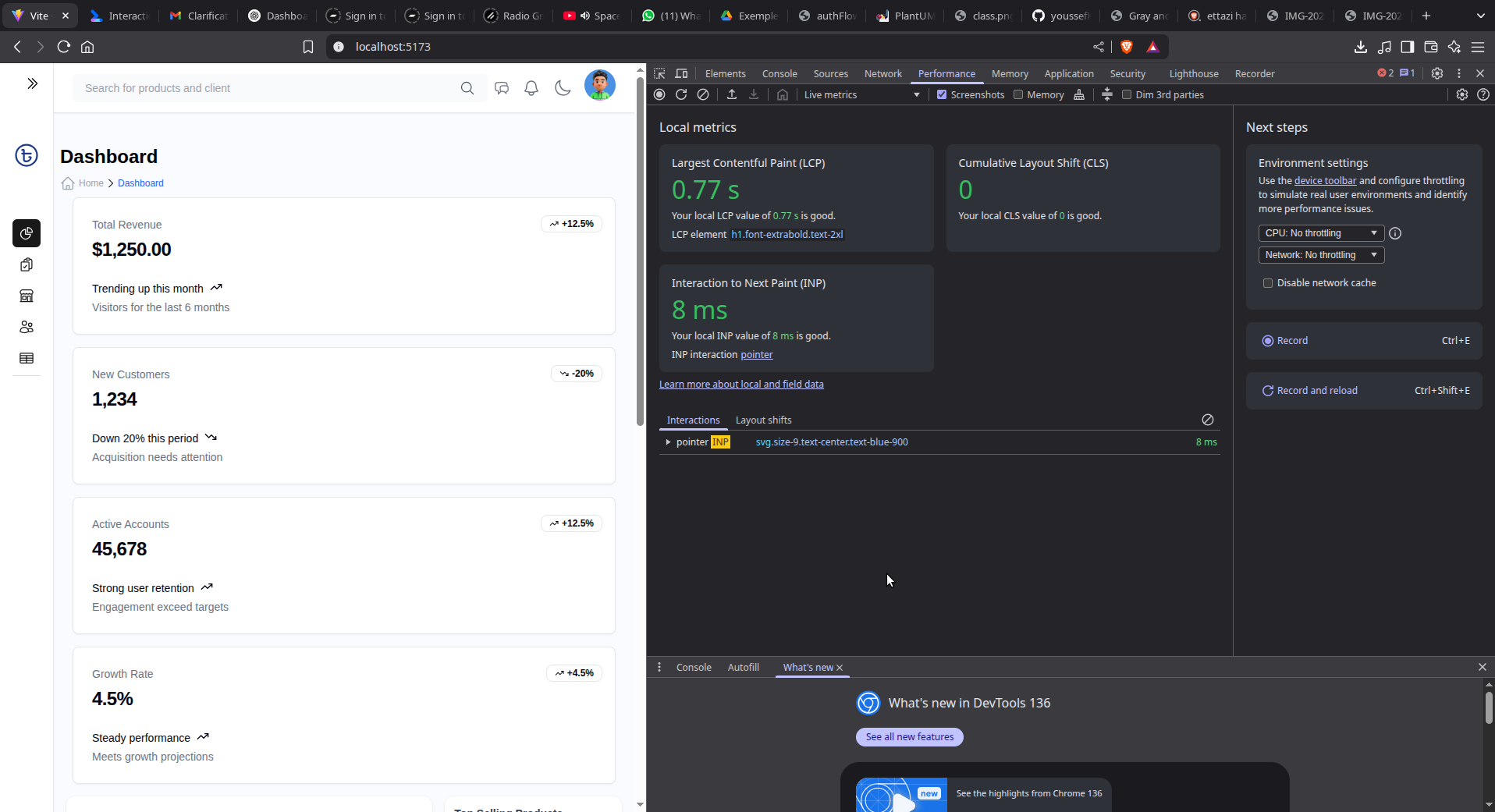
#### La sécurité est essentielle pour un système manipulant des données sensibles :

* Authentification sécurisée via **tokens JWT** .
* Gestion des rôles et permissions : **seuls les administrateurs** peuvent accéder au dashboard.
* Données sensibles chiffrées (ex. : mots de passe avec bcrypt).
* Protection contre les attaques injection SQL assurée par les mécanismes de Laravel et les bonnes pratiques front-end.

#### ****Maintenance****

Pour faciliter la maintenance et les évolutions futures :

* Architecture en **composants réutilisables** côté React.
* Utilisation de **principes SOLID** et d’un **framework MVC** (Laravel) pour un code clair et organisé.
* Documentation du code et des API.
* Facilité de mise à jour des dépendances grâce à **npm** et **composer**



#### ****Portabilité****

**Architecture et Dépendances**

* + Utilisation de Laravel (back-end) et React (front-end), deux frameworks largement supportés
  + Séparation claire front-end/back-end
  + Utilisation de Docker pour la conteneurisation
  + Structure modulaire avec services et interfaces

**Configuration et Environnement**

* + Utilisation des fichiers .env pour la configuration
  + Variables d'environnement pour les clés API
  + Configuration Docker pour l'environnement de développement

**Base de Données**

* + Utilisation de migrations Laravel
  + Structure de base de données standard
  + Support de Mysql

**Services Externes**

* + - Abstraction des services via interfaces
    - Service de stockage modulaire

**Front-end**

* + Composants React réutilisables
  + TypeScript pour la sécurité du typage
  + Structure modulaire

**API et Communication**

* + API RESTful standard
  + Documentation des endpoints
  + Gestion des erreurs standardisée

# **Chapitre III : Etude technique**

## ****Présentation des technologies utilisées :****

Dans le cadre du développement de cette application web, plusieurs technologies modernes ont été adoptées pour assurer la performance, la sécurité, la maintenabilité et l’ergonomie du projet. Ces technologies se divisent en deux grandes catégories : **le front-end** (interface utilisateur) et **le back-end** (logique métier, base de données et API).

#### ****Front-end :****

**React avec TypeScript**

* **React** est une bibliothèque JavaScript développée par Facebook permettant de créer des interfaces utilisateur dynamiques et réactives.
* **TypeScript** est un sur-ensemble de JavaScript qui ajoute le typage statique, ce qui permet d’éviter de nombreuses erreurs lors du développement.
* Ensemble, ils facilitent la création de composants réutilisables, typés, maintenables et faciles à tester

****Tailwind CSS****

* Un framework CSS utilitaire permettant de concevoir des interfaces rapidement sans écrire de styles CSS classiques.
* Il permet un **design responsive** et **modulaire**, tout en maintenant une base de code propre et organisée.

**shadcn/ui**

* Une bibliothèque de composants UI accessible, construite sur Tailwind CSS et Radix UI.
* Elle fournit des composants modernes, stylés et facilement personnalisables comme des **modals**, **toasts**, **formulaires**, etc.

****Yup****

* Une bibliothèque JavaScript utilisée pour la **validation des schémas de données**, notamment pour les formulaires.
* Elle est intégrée avec des outils comme React Hook Form pour valider facilement les champs côté client.

**Autres bibliothèques**

* **React Hook Form** : pour la gestion des formulaires.
* **Axios** : pour effectuer des appels HTTP vers l’API Laravel

#### ****Back-end****

****Laravel (PHP)****

* Un framework web robuste basé sur PHP, orienté objet et structuré en MVC (Modèle-Vue-Contrôleur).
* Laravel propose :
* Un système d’authentification,
* Une gestion simplifiée des bases de données avec Eloquent ORM,
* Des routes API RESTful simples à mettre en œuvre,
* Un environnement sécurisé avec middleware, validation, etc.

**MySQL**

* Un système de gestion de base de données relationnelle (SGBDR).
* Utilisé pour stocker toutes les données liées aux utilisateurs, étudiants, résultats, etc.

****Cloudinary****

Une plateforme de **stockage et gestion d’images** dans le cloud.

Elle permet :

* L’upload sécurisé des images via API,
* Le redimensionnement et l’optimisation des images,
* La récupération rapide via une URL unique.

## **Choix du SGBDR : **MySQL****

**Le **Système de Gestion de Base de Données Relationnelle (SGBDR)**** choisi pour ce projet est **MySQL**. Ce choix a été motivé par plusieurs raisons techniques et pratiques :

* **Simplicité d’utilisation**

MySQL est simple à installer, configurer et utiliser. Il est parfaitement adapté aux projets web de taille petite à moyenne, comme celui développé dans ce projet.

* **Compatibilité avec Laravel**

MySQL est **nativement supporté** par le framework Laravel. L’intégration avec **Eloquent ORM** est fluide, ce qui facilite la gestion des modèles, migrations et requêtes.

* **Performance et fiabilité**

**MySQL** est reconnu pour sa **rapidité d'exécution** des requêtes et sa **fiabilité** dans les environnements de production. Il convient bien aux applications nécessitant une bonne performance en lecture et en écriture.

* **Documentation abondante**

Il existe une **grande communauté** et une **documentation riche**, ce qui facilite la résolution des problèmes éventuels et accélère le développement.

* **Portabilité**

**MySQL** est **open source**, multiplateforme, et peut être utilisé aussi bien en local que sur des serveurs cloud. Cela permet une migration facile entre les environnements de développement et de production.

## **Les outils de développement utilisés**

Le développement de l’application s’est appuyé sur une combinaison d’outils modernes côté front-end et back-end pour optimiser la productivité, la qualité du code et la maintenance.

****Éditeurs et Environnement de développement****

* **Neovim** : éditeur de code léger, hautement personnalisable avec support LSP, idéal pour coder rapidement en React, TypeScript et PHP.
* **LAMP** (Linux, Apache, MySQL, PHP) : environnement local utilisé pour faire tourner Laravel.

****Outils de développement back-end****

* **Laravel** : framework PHP utilisé pour créer l’API RESTful, la logique métier et l’authentification.
* **Artisan** : outil en ligne de commande fourni par Laravel pour générer des fichiers, exécuter des migrations, lancer le serveur local, etc.
* **Composer** : gestionnaire de dépendances PHP, utilisé pour installer Laravel et les packages tiers.

**Outils de développement front-end**

* + **React (avec TypeScript)** : bibliothèque JavaScript utilisée pour construire une interface utilisateur réactive et typée.
  + **Tailwind CSS** : framework CSS utilitaire permettant un développement rapide et propre.
  + **shadcn/ui** : bibliothèque de composants UI construite sur Tailwind CSS.
  + **Yup** : bibliothèque de validation de formulaires côté client.
  + **Vite** : outil de bundling utilisé pour le front-end React, avec rechargement à chaud rapide.
  + **npm** : gestionnaire de packages Node.js utilisé pour installer et gérer les dépendances front-end.

**Outils d’API et de test**

* + **Insomnia** : outil de test d’API pour simuler les requêtes HTTP vers l’API Laravel.
  + **Cloudinary** : plateforme de gestion d’images en ligne utilisée pour stocker et optimiser les images envoyées par les utilisateurs.

****Outils de versionnage****

* + **Git** : pour la gestion du code source, le suivi des modifications et la création de branches.
  + **GitHub** : pour héberger le projet, suivre les issues, collaborer et documenter.

# Chapitre IV : Réalisation/Mise en œuvre de la solution

## Structure du projet

### Structure du Projet Front-end (Reactjs, Typescript) :

Le projet est structuré de manière modulaire pour faciliter la lisibilité, la maintenabilité et l’évolutivité de l'application. Voici l’arborescence principale du projet **React** + **TypeScript** :

src/

├── api/ # Configuration d'Axios et fichiers d’appel aux APIs

├── assets/ # Images, icônes, fichiers statiques

├── components/ # Composants UI réutilisables

│ ├── common/ # Boutons, inputs, modals, etc.

│ ├── containers/ # Composants spécifiques à certaines vues

│ └── ui/ # Design system (badges, cartes, tableaux, etc.)

├── config/ # Fichiers de configuration globaux

├── context/ # Contextes React (authentification, notifications, etc.)

├── hooks/ # Hooks personnalisés (ex: useProduct, useCategory…)

├── interfaces/ # Interfaces TypeScript (produits, utilisateurs, commandes…)

├── layouts/ # Layouts pour les pages (dashboard, invité, etc.)

├── lib/ # Fonctions utilitaires globales

├── pages/ # Pages principales de l’application (dashboard, produits…)

│ ├── dashboard/

│ ├── products/

│ ├── orders/

│ └── login/

├── routes/ # Fichiers de définition des routes et protections

├── services/ # Services pour stockage local, cookies, etc.

├── types/ # Types TypeScript partagés

└── utils/ # Fonctions utilitaires

#### api/

Contient la configuration Axios pour interagir avec l’API backend.

* axios.config.ts : définit les en-têtes, l’URL de base, etc.
* index.ts : point d’entrée pour les appels API.

#### components/

Répertoire des composants réutilisables dans toute l'application :

* **common/ :** composants génériques (boutons, modales, tableaux, etc.)
* **containers/ :** composants spécifiques aux vues ou fonctionnalités.
* **Profil/ :** composants liés à la gestion de profil.
* **ui/ :** éléments UI stylisés (bouton, carte, checkbox, input...).

#### context/

Contient les contextes React (équivalent du state global partagé) :

* authContext, NotificationContext, etc. : pour gérer l’état de l’utilisateur, les notifications, etc.

#### HOC/

(Higher Order Components) pour encapsuler la logique commune à plusieurs pages, comme un dashboardWrapper.

#### hooks/

Hooks personnalisés pour réutiliser la logique côté client : gestion de produits, catégories, wishlist, etc.

#### interfaces/ et types/

Définissent les types TypeScript pour garantir la cohérence des données manipulées (produits, utilisateurs, commandes…).

#### layouts/

Composants qui définissent l’apparence globale des pages : sidebar, navbar, layout dashboard.

#### lib/ et utils/

Fonctions utilitaires ou bibliothèques internes utilisées un peu partout dans l'app.

#### pages/

Chaque sous-dossier représente une page ou vue majeure de l’application : dashboard, produits, clients, etc.

* Organisation hiérarchique : par domaine fonctionnel (produits, clients, commandes...).
* Les sous-dossiers components et hooks dans chaque page renforcent la séparation des responsabilités.

#### routes/

Définit les routes protégées ou accessibles aux invités selon l’authentification.

#### services/

Services comme **localStorageService** pour stocker/retrouver des données dans le navigateur.

#### Fichiers principaux

* **App.tsx :** point d’entrée principal de l’app.
* **main.tsx** : point de montage React.
* **index.css** : styles globaux.
* **vite-env.d.ts :** types pour Vite.

### Structure du Projet Backend (Laravel) :

Le backend est structuré de manière **modulaire** et **organisée par responsabilités**, afin d’assurer une **maintenabilité**, une **extensibilité** et une **lisibilité** optimales du code.

app/

├── Http/

│ ├── Controllers/

│ │ └── Api/ # Contrôleurs RESTful pour les ressources principales

│ ├── Kernel.php # Enregistrement des middlewares

│ ├── Middleware/ # Middleware globaux

│ ├── Middlewares/Api/ # Middleware spécifiques à l’API

│ ├── Requests/ # Form Requests pour la validation des données entrantes

│ ├── Resources/ # Formatage des données JSON retournées (API Resources)

│ ├── routes/ # Définition des routes de l’API

│ └── Traits/Api/ # Trait commun pour les réponses JSON

├── Interfaces/ # Interfaces pour services et dépendances

├── Models/ # Modèles Eloquent (entités métier)

├── Providers/ # Fournisseurs de services Laravel

└── Services/ # Services métiers (ex: Cloudinary)

**Http/Controllers/Api/**

Contient les contrôleurs de l'API REST :

* + - AuthController.php : Authentification (login, logout, etc.)
    - CategoryController.php, ProductController.php : gestion des produits et catégories
    - CustomerController.php, ClientController.php : gestion des clients
    - OrderController.php : gestion des commandes
    - PermissionController.php, RoleController.php : gestion des permissions et rôles (RBAC)
    - Controller.php : contrôleur parent (hérité par tous les autres)

**Http/Middleware/ & Http/Middlewares/Api/**

Contient les middlewares :

* ApiTokenAuth.php : Middleware de base pour authentification par token
* isAdmin.php : Vérifie les privilèges admin
* ValidateToken.php : Middleware de validation de jeton

**Http/Requests/**

Form Requests pour la validation des données côté serveur :

* LoginRequest.php, ProfileUpdateRequest.php : règles de validation

**Http/Resources/**

Définit la structure de sortie des réponses JSON :

* Exemple : ProductResource.php, OrderResource.php
* Collections et ressources simples pour une API propre et cohérente

**Http/routes/**

Organisation des routes par domaine fonctionnel :

* auth.php, category.php, product.php, order.php, etc.

**Http/Traits/Api/**

* ApiResponse.php : Trait pour normaliser les réponses API (succès/erreur)

**Interfaces/**

* StorageServiceI.php : Interface pour abstraction des services de stockage (ex: Cloudinary)

**Models/**

Modèles Eloquent représentant les entités de la base de données :

* Product.php, Category.php, Order.php, User.php, etc.

**Providers**/

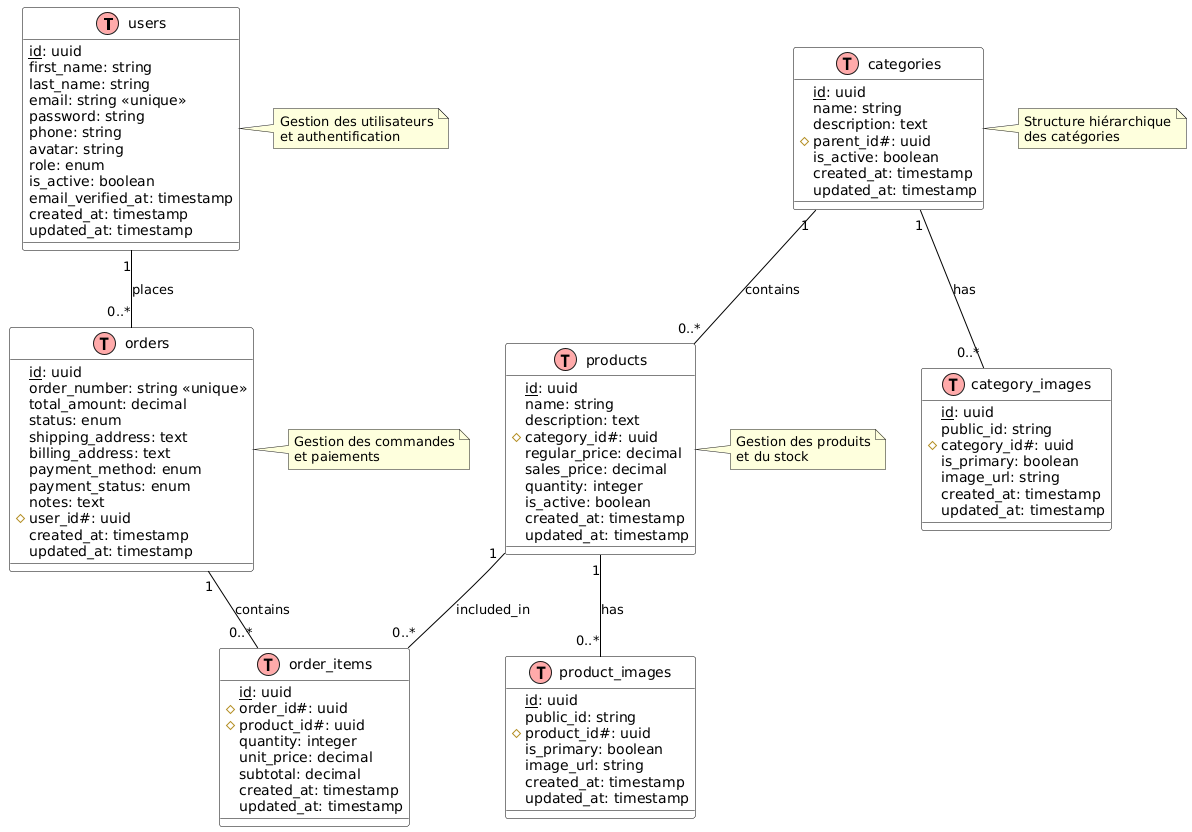
* AppServiceProvider.php : enregistrement de services personnalisés, binding d'interfaces, etc.

**Services/**

Services métiers et d’intégration externe :

* CloudinaryService.php : gestion des uploads d’images vers Cloudinary

## Schéma de la base de données

Dans le cadre de notre projet, la conception de la base de données a été pensée pour répondre aux besoins fonctionnels de l’application backend, assurant une gestion cohérente des utilisateurs, produits, commandes et autres entités clés.

## Présentation des interfaces de l’application /Captures écrans

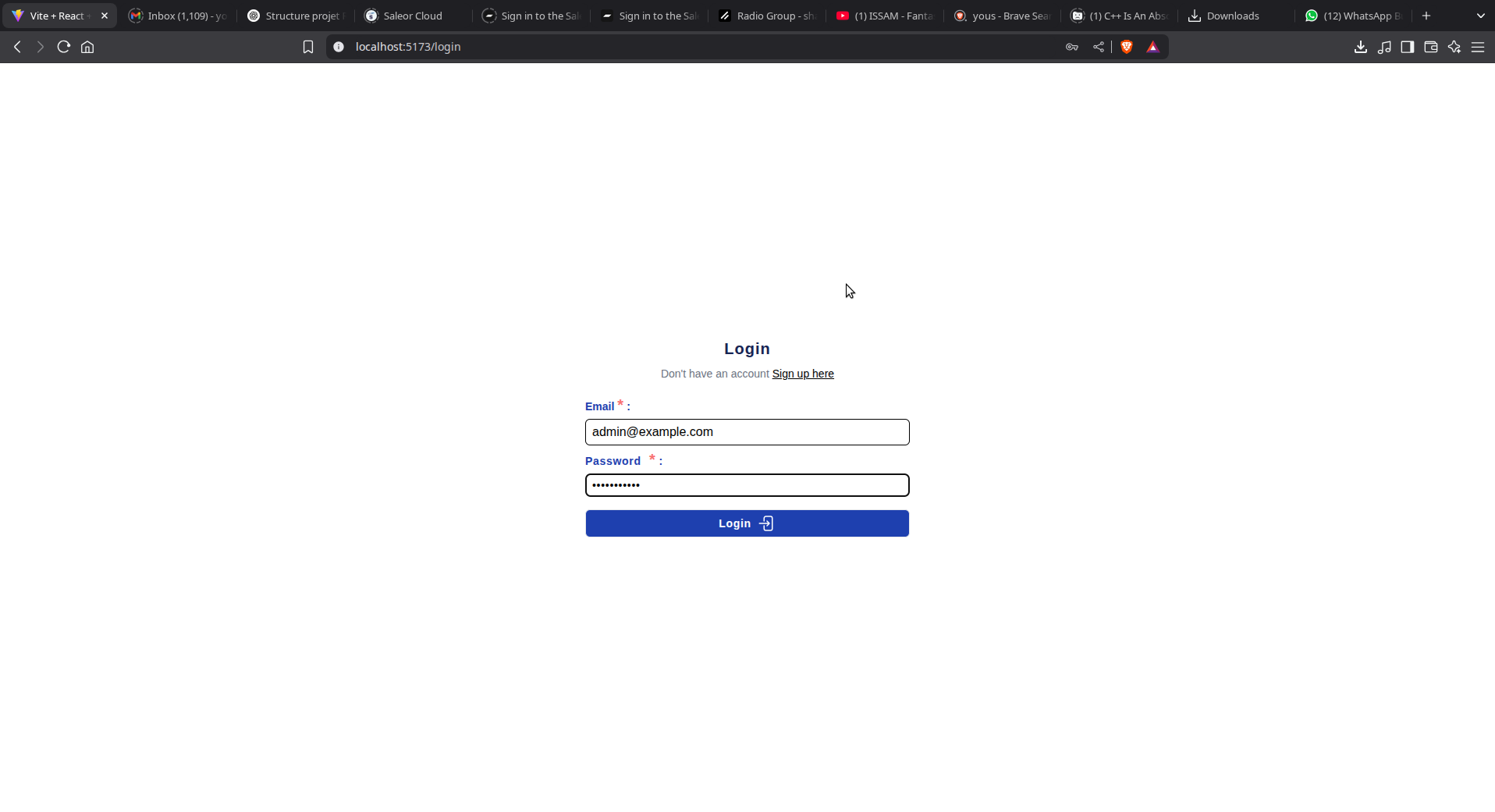
### Page de connexion (Login) :

**Description générale**

La page de connexion est la porte d’entrée de l’application, permettant aux utilisateurs de s’authentifier pour accéder aux fonctionnalités réservées. Cette page est conçue pour être simple, intuitive et sécurisée.

* Le formulaire de login utilise **Formik**, une bibliothèque React qui facilite la gestion des formulaires, notamment la gestion des états, des validations et des erreurs.
* La validation des champs (email et mot de passe) est effectuée en temps réel pour améliorer l’expérience utilisateur.
* En cas d’erreur de validation (champs vides, format incorrect, mot de passe trop court), un message d’erreur clair est affiché à côté du champ concerné.
* Les erreurs d’authentification (ex : mauvais identifiants) sont affichées sous forme de notifications visibles par l’utilisateur.

**Capture d’écran de la page Login**



### Présentation de l’interface : Page Dashboard

**Description générale**

La page Dashboard constitue le tableau de bord principal de l’application, offrant une vue d’ensemble synthétique des indicateurs clés de performance (KPI) et des données essentielles pour le suivi des activités commerciales. Elle est conçue pour fournir rapidement des informations pertinentes aux utilisateurs, notamment aux administrateurs et aux responsables.

**Composants principaux du Dashboard**

* **Cards des KPI (Indicateurs Clés)**

Ces cartes présentent en un coup d’œil les données essentielles suivantes :

* **Total Revenu** : le chiffre d’affaires généré sur la période sélectionnée.
* **Total Customers** : nombre total de clients actifs.
* **Total Orders** : nombre total de commandes effectuées.
* **Total Growth Rate** : taux de croissance comparé à la période précédente.

Chaque carte est stylisée pour attirer l’attention avec des icônes et des couleurs adaptées selon le contexte (croissance positive, alertes, etc.).

* **Graphique de performance des ventes (Sales Performance)**

Un graphique multi-lignes présente l’évolution mensuelle des commandes selon leur statut :

* **Pending** (En attente)
* **Completed** (Terminées)
* **Cancelled** (Annulées)
* **Processing** (En cours de traitement)

Le graphique permet d’observer les tendances, pics et baisses des ventes et d’analyser la santé commerciale sur l’année.

tableau ou liste affiche les produits qui génèrent le plus de ventes. Pour chaque produit, on affiche :

* Nom du produit
* Quantité vendue
* Chiffre d’affaires généré

Cela permet d’identifier rapidement les articles phares et d’orienter les décisions commerciales.

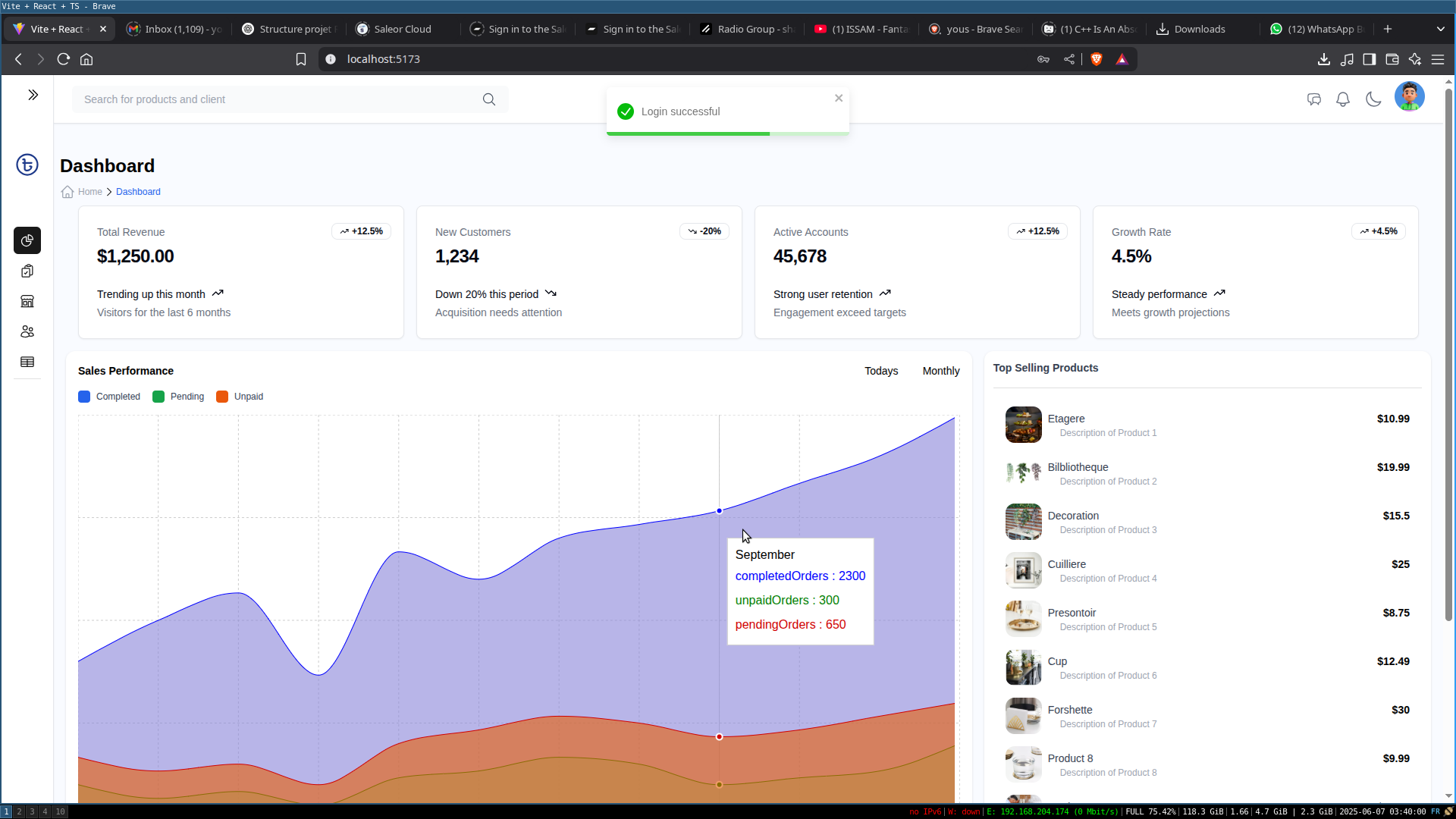
* **Tableau des commandes récentes (Recent Orders)**

Ce tableau liste les dernières commandes effectuées avec les colonnes suivantes :

* Numéro de commande
* Nom du client
* Date de la commande
* Statut (Pending, Completed, etc.)
* Montant total

Il offre une vue rapide sur l’activité récente et facilite la gestion des commandes.

**Capture d’écran type de la page Dashboard**



### Présentation de l’interface : Page Tous les Produits (All Products)

**Description générale**

La page **All Products** permet aux utilisateurs de consulter l’ensemble des produits disponibles sous forme de cartes, avec la possibilité de filtrer les résultats via une recherche par nom ou par catégorie. Cette interface facilite la navigation et la gestion des produits dans l’application. Un bouton visible permet également de naviguer vers une page dédiée à la création d’un nouveau produit.

**Fonctionnalités**

* **Barre de recherche**
  + Recherche dynamique par **nom du produit**.
  + Filtre par **catégorie** via un menu déroulant ou un champ de sélection.
  + Mise à jour instantanée de la liste des produits affichés selon les critères saisis.
* **Affichage des produits en cartes**

Chaque produit est représenté par une carte comprenant :

* Image du produit
* Nom du produit
* Catégorie
* Prix
* Un bouton popup pour voir plus de détails ou modifier le produit (
* Les cartes sont disposées en grille responsive.
* Bouton de navigation vers la création de produit

Un bouton clairement visible, généralement en haut ou en bas de la page, permet à l’utilisateur de naviguer vers la page dédiée à la **création d’un nouveau produit**.

**Capture d’écran de la page de tous les produits**

### Présentation de l'interface : Page de Création de Produit

**Description générale**

La page de création de produit est une interface dédiée à l'ajout de nouveaux produits dans le catalogue de l'application e-commerce. Elle offre une expérience utilisateur intuitive et structurée pour la saisie des informations essentielles des produits.

**Composants principaux de la page**

**Formulaire de creation de produit**

**Nom du produit** (obligatoire)

* + - * + Champ de saisie textuel
        + Validation en temps réel

**Description du produit** (obligatoire)

* + - * + Zone de texte pour une description détaillée
        + Validation de la longueur minimale

**Quantité** (obligatoire)

* + - * + Champ numérique
        + Validation de la valeur minimale

**Prix**

* + - * + Prix régulier (obligatoire)
        + Prix de vente (obligatoire)
        + Validation des montants

**Catégorie** (obligatoire)

* + - * + Sélecteur de catégorie
        + Liste déroulante des catégories existantes

**Gestion des médias**

* + - * + Upload d'images
        + Zone de dépôt de fichiers
        + Support multi-images
        + Prévisualisation des images
        + Option de suppression d'image
        + Validation des formats acceptés

**Barre d'actions**

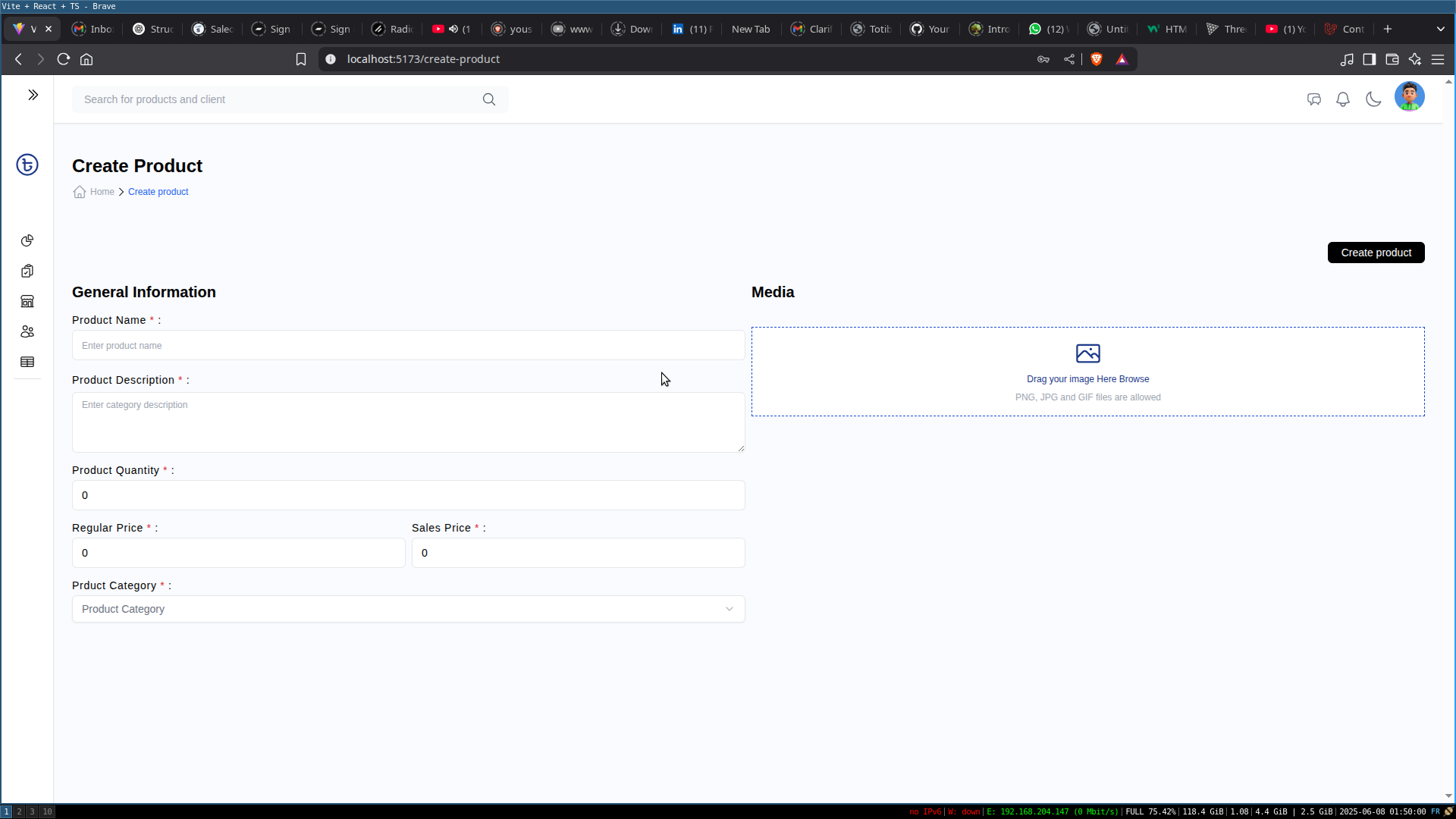
* + - * + Bouton "Create product" en haut à droite
        + Indicateur de chargement pendant la soumission
        + Messages de notification pour le succès/échec

**Fonctionnalités principales**

* Validation en temps réel des champs
* Gestion des erreurs avec messages explicites
* Upload multiple d'images
* Prévisualisation des images avant envoi
* Sélection de catégorie avec interface dédiée
* Sauvegarde des données en format multipart/form-data
* Gestion des erreurs
* Le système affiche des messages d'erreur pour :
* Champs obligatoires non remplis
* Formats d'images invalides
* Valeurs numériques incorrectes
* Échec de création du produit

Cette interface est conçue pour offrir une expérience utilisateur fluide et efficace dans la création de nouveaux produits, avec une attention particulière portée à la validation des données et à la gestion des médias.

**Capture d’écran de la page Creation de Product**

****

### Présentation de l'interface : Page de Création de Catégorie

**Description générale**

La page de création de catégorie est une interface dédiée à l'ajout de nouvelles catégories dans le système de classification des produits. Elle offre une expérience utilisateur simple et efficace pour la gestion des catégories de l'application e-commerce.

**Composants principaux de la page :**

**Formulaire de création de catégorie**

* + - **Nom de la catégorie** (obligatoire)
    - Champ de saisie textuel
    - Validation en temps réel
    - Message d'erreur si invalide
    - **Description de la catégorie** (obligatoire)
    - Zone de texte pour une description détaillée
    - Validation de la longueur minimale
    - Message d'erreur si invalide

**Gestion de l'image**

* + - **Image de la catégorie** (obligatoire)
    - Upload de fichier unique
    - Prévisualisation de l'image
    - Validation du format
    - Message d'erreur si invalide

**Barre d'actions**

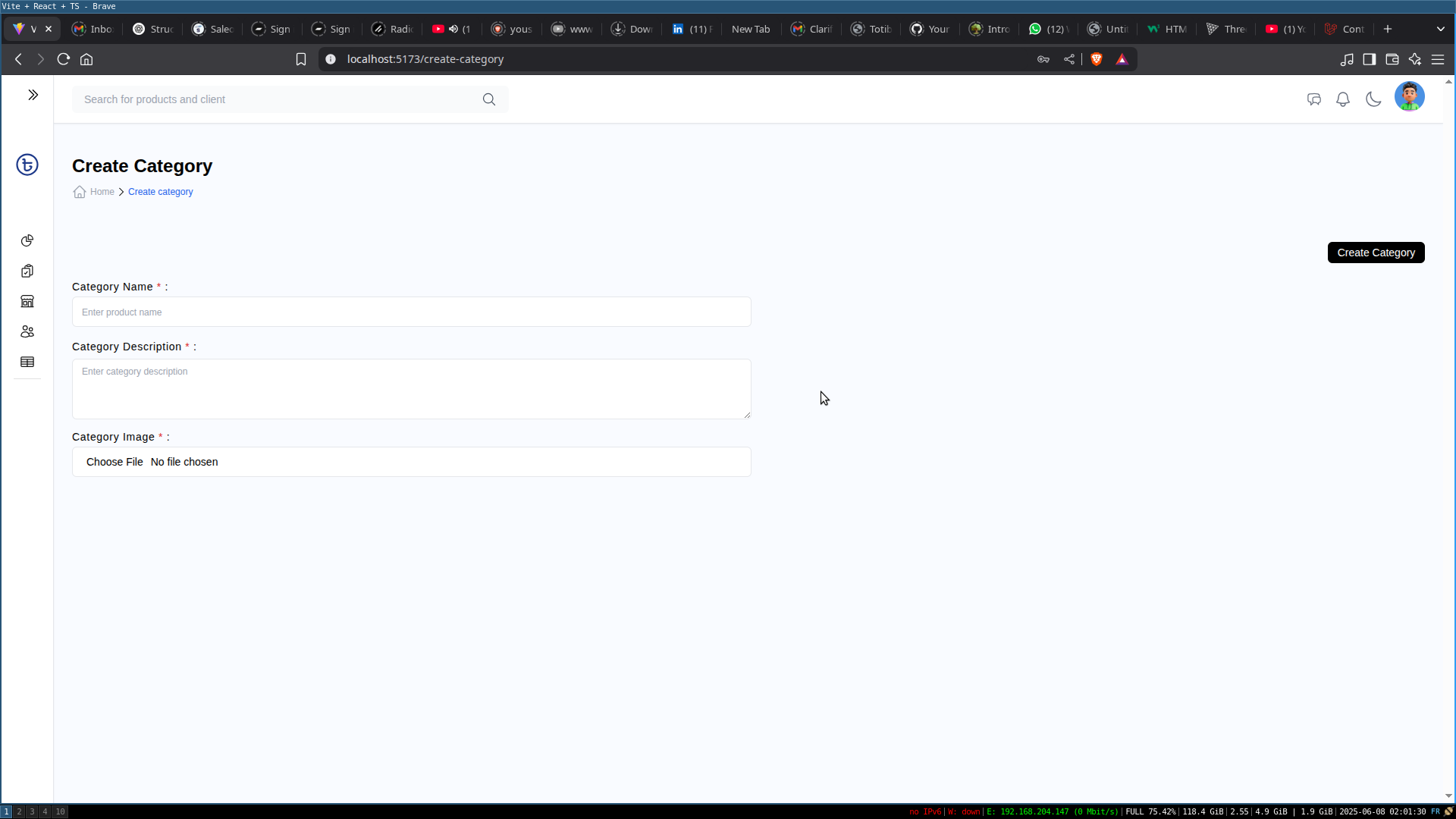
* + - * Bouton "Create Category" en haut à droite
      * Indicateur de chargement pendant la soumission
      * Messages de notification pour le succès/échec

**Fonctionnalités principales**

* Validation en temps réel des champs
* Gestion des erreurs avec messages explicites
* Upload d'image avec prévisualisation
* Sauvegarde des données en format multipart/form-data
* Navigation intuitive avec fil d'Ariane (breadcrumb)
* Gestion des erreurs
* Le système affiche des messages d'erreur pour :
* Champs obligatoires non remplis
* Format d'image invalide
* Échec de création de la catégorie

Cette interface est conçue pour offrir une expérience utilisateur claire et efficace dans la création de nouvelles catégories, avec une attention particulière portée à la validation des données et à la gestion des médias.

**Capture d’écran de la page Create Product**



### Présentation de l'interface : Page de Détail du Produit

**Description générale**

La page de détail du produit est une interface complète permettant de visualiser et gérer les informations détaillées d'un produit spécifique. Elle offre une vue détaillée des caractéristiques du produit et permet sa modification.

**Composants principaux de la page**

* **Informations du produit**
  + - * **Nom du produit**

Affichage en mode lecture

Champ de saisie en mode édition

* + - * **Description du produit**

Affichage en mode lecture

Zone de texte en mode édition

* + - * **Catégorie**

Affichage avec image et nom

Sélecteur en mode édition

* + - * **Prix**

Prix régulier

Prix de vente

* + - * **Quantité en stock**
      * **Gestion des images**

Galerie d'images

Affichage des images existantes

Option de suppression d'image

Upload de nouvelles images

Prévisualisation des nouvelles images

* **Barre d'actions**
  + - * **Boutons de contrôle**

Édition

Annulation

Sauvegarde

Suppression

* + - * Indicateurs de chargement pour les actions

**Fonctionnalités principales**

* Mode lecture/édition
* Gestion des images (ajout/suppression)
* Modification des informations du produit
* Sauvegarde des modifications
* Suppression du produit
* Affichage des statistiques
* Gestion des avis

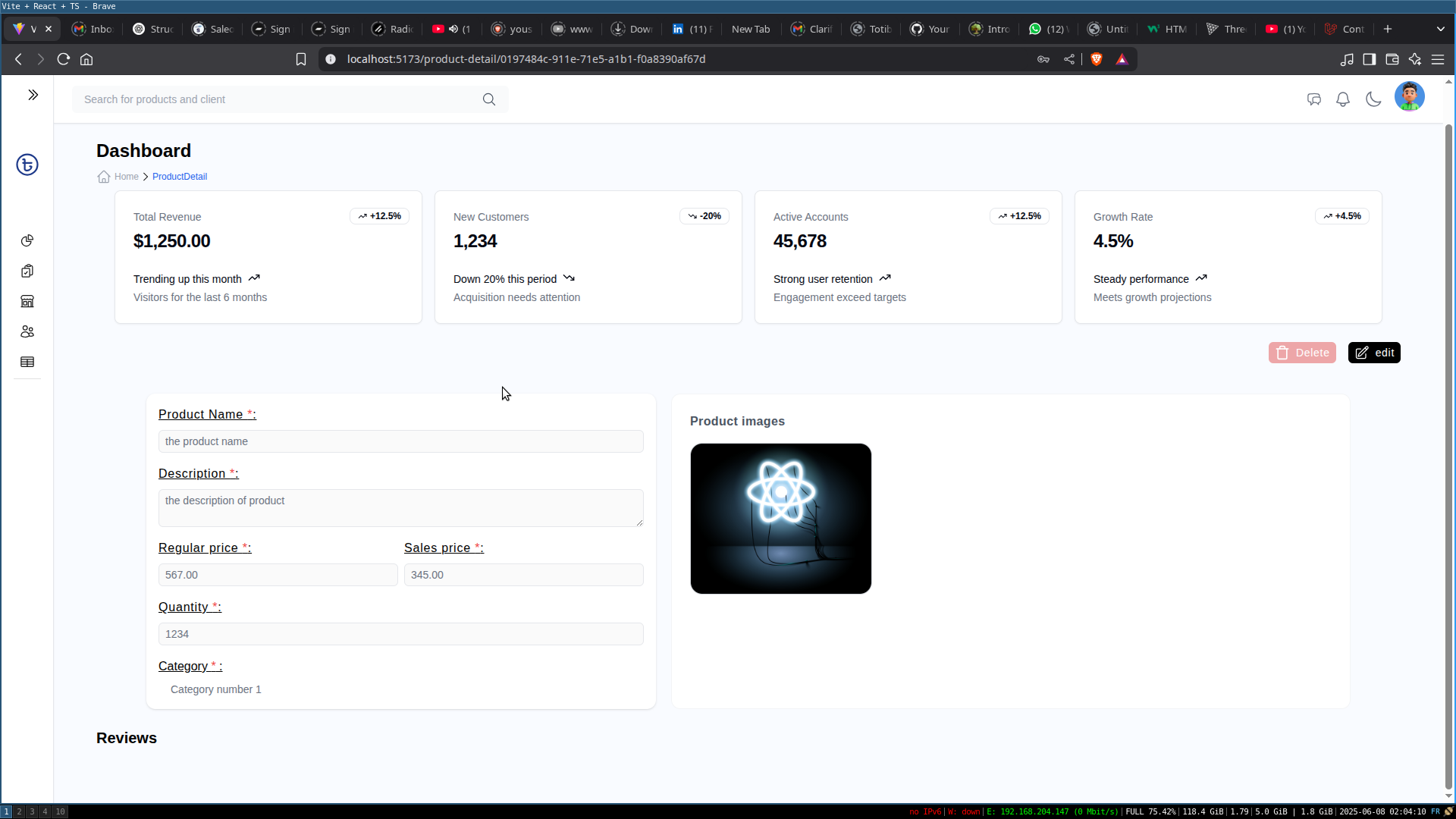
**Gestion des erreurs**

Le système affiche des messages de notification pour :

* + Succès de la mise à jour
  + Échec de la mise à jour
  + Erreurs de chargement
  + Problèmes de suppression

Cette interface est conçue pour offrir une expérience utilisateur complète dans la gestion des détails d'un produit, avec une attention particulière portée à la facilité d'édition et à la gestion des médias.

**Capture d’écran de la page de detail de produit**



### Présentation de l'interface : Page de Détail de Catégorie

**Description générale**

La page de détail de catégorie est une interface complète permettant de visualiser et gérer les informations détaillées d'une catégorie spécifique. Elle offre une vue détaillée des caractéristiques de la catégorie et permet sa modification.

**Composants principaux de la page**

* **Section d'édition de la catégorie**
  + - * + **Informations de la catégorie**
      * **Nom de la catégorie**
      * Affichage en mode lecture
      * Champ de saisie en mode édition
      * **Description de la catégorie**
      * Affichage en mode lecture
      * Zone de texte en mode édition
    - **Gestion de l'image**
    - Affichage de l'image principale
    - Upload de nouvelle image en mode édition
    - Prévisualisation de la nouvelle image
* **Barre d'actions**
  + - **Boutons de contrôle**
    - Édition
    - Annulation
    - Sauvegarde
    - Suppression
    - **Indicateurs de chargement** pour les actions
* **Section des produits associés**
  + - **Liste des produits** de la catégorie
    - **Statistiques** des produits
    - **Recherche** de produits
    - **Tableau** des produits avec :
    - Image du produit
    - Nom du produit
    - Navigation vers le détail du produit

**Fonctionnalités principales**

* + Mode lecture/édition
  + Gestion de l'image (modification)
  + Modification des informations de la catégorie
  + Sauvegarde des modifications
  + Suppression de la catégorie
  + Affichage des statistiques
  + Gestion des produits associés

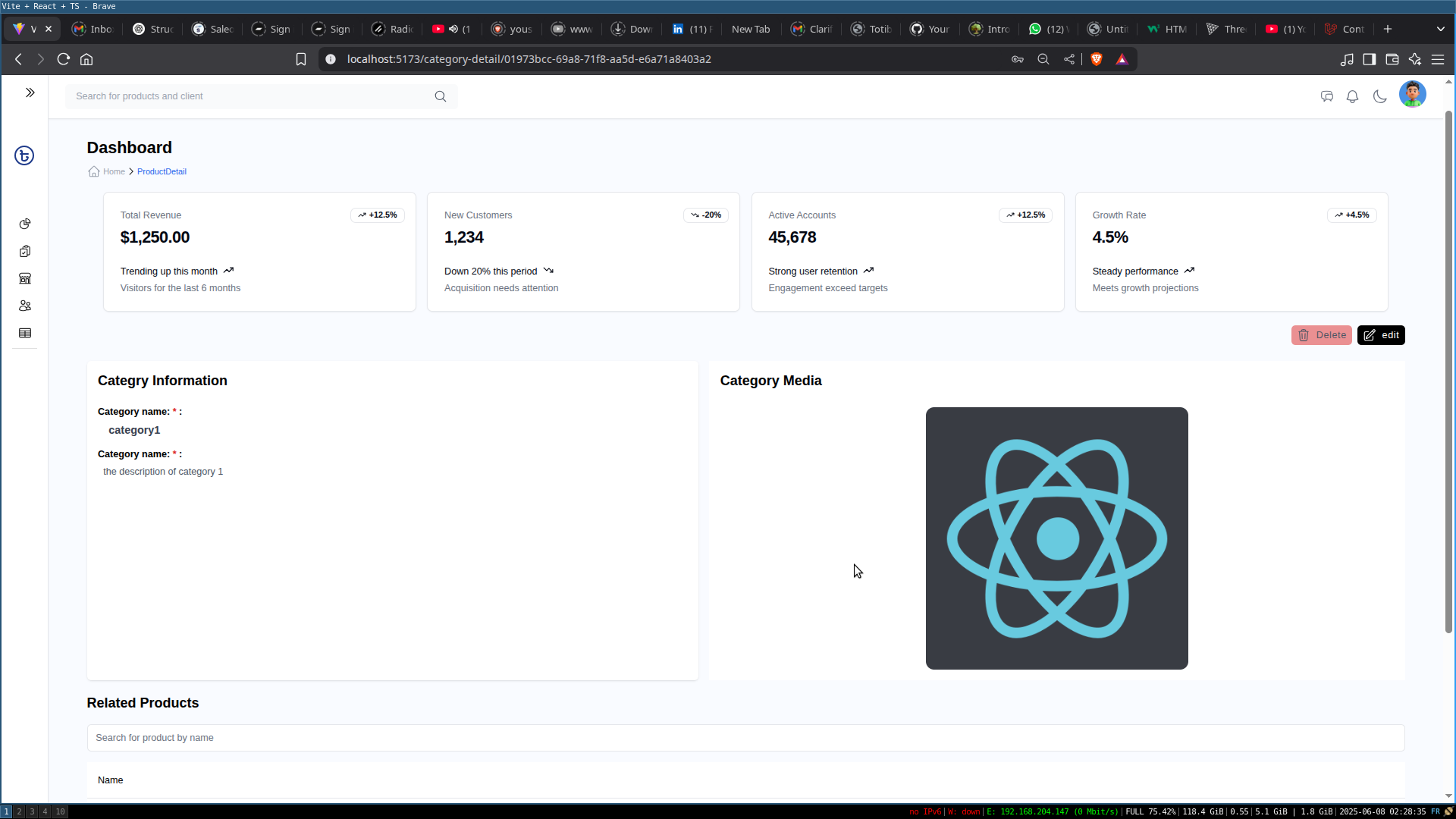
**Gestion des erreurs**

Le système affiche des messages de notification pour :

* + Succès de la mise à jour
  + Échec de la mise à jour
  + Erreurs de chargement
  + Problèmes de suppression

Cette interface est conçue pour offrir une expérience utilisateur complète dans la gestion des détails d'une catégorie, avec une attention particulière portée à la facilité d'édition et à la gestion des produits associés.

**Capture d’écran de la page Category Detail**



### **Présentation de l'interface : Page des commandes (orders)**

**Description générale**

La page des commandes est une interface complète permettant de visualiser et gérer toutes les commandes du système. Elle offre une vue d'ensemble des commandes avec des statistiques détaillées et des fonctionnalités de filtrage.

**Composants principaux de la page**

* **Section des statistiques (OrdersStats)**

**Cartes de statistiques** affichant :

* + - Revenu total
    - Revenu du mois
    - Commandes du mois
    - Commandes complétées
    - Commandes en attente
    - Total des commandes
* **Filtres (FilterOrders)**
  + - **Barre de recherche**
    - Recherche par nom ou email du client
    - **Filtre par statut**
    - Pending
    - Cancelled
    - Completed
    - Processing
* **Tableau des commandes (OrdersTable)**
  + - **Colonnes** :
      * Nom du client (avec avatar)
      * Email du client
      * Statut de la commande
      * Nombre d'articles
      * Date
      * Montant total
    - **Fonctionnalités** :
      * Navigation vers le détail de la commande
      * Affichage du statut avec code couleur
      * Formatage des dates

**Fonctionnalités principales**

* Filtrage des commandes
* Recherche en temps réel
* Affichage des statistiques
* Navigation vers les détails
* Gestion des statuts
* Pagination des résultats

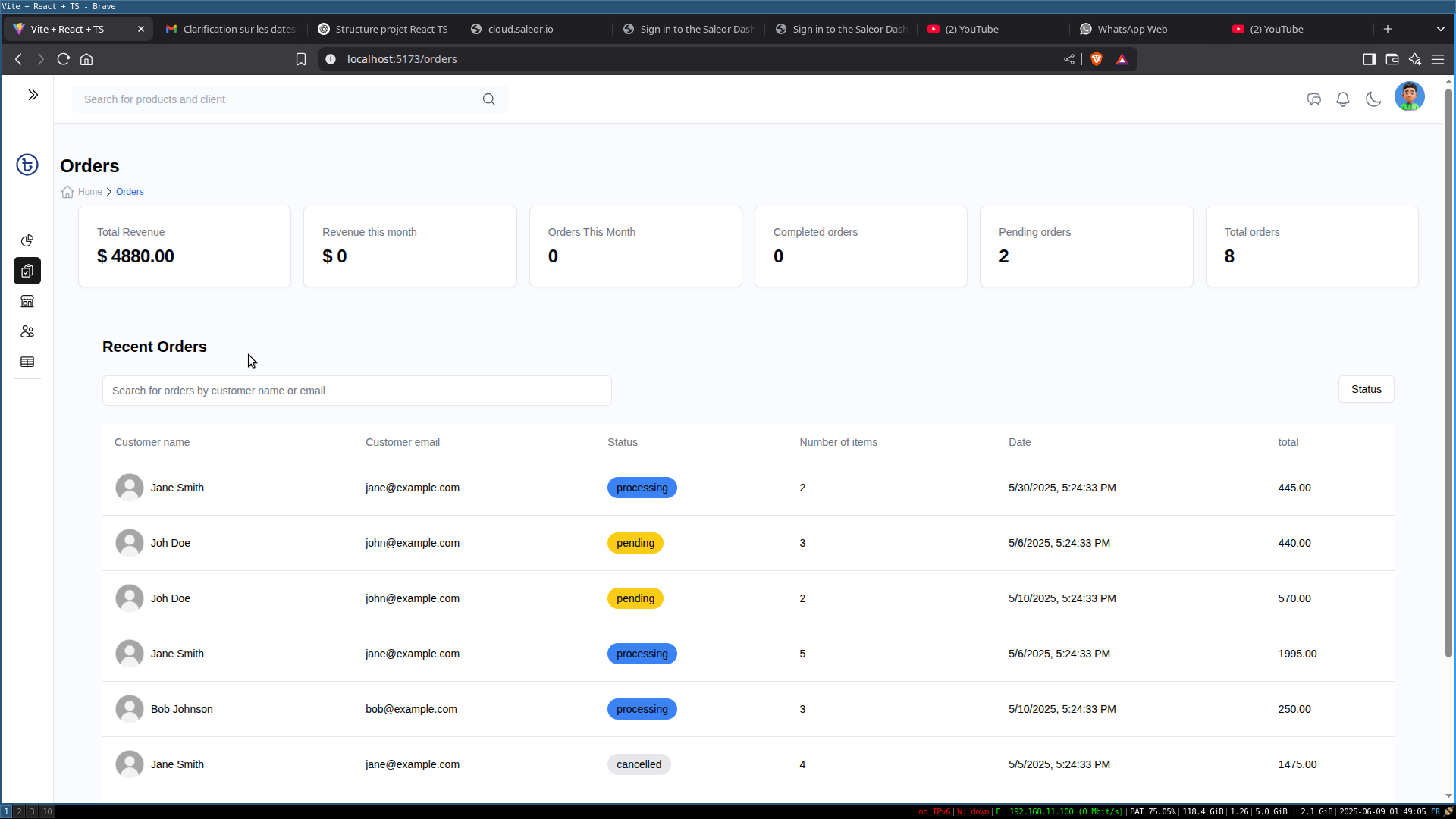
**Gestion des erreurs**

Le système affiche des messages de notification pour :

* Erreurs de chargement
* Problèmes de filtrage
* Échecs de mise à jour

Cette interface est conçue pour offrir une expérience utilisateur complète dans la gestion des commandes, avec une attention particulière portée à la visualisation des données et à la facilité de filtrage.

**Capture d’écran de la page des commandes**



### **Présentation de l'interface : Page de Détail de détail du commande**

**Description générale**

La page de détail des commandes est une interface détaillée permettant de visualiser et gérer une commande spécifique. Elle offre une vue complète des informations de la commande, des produits commandés et des options de gestion.

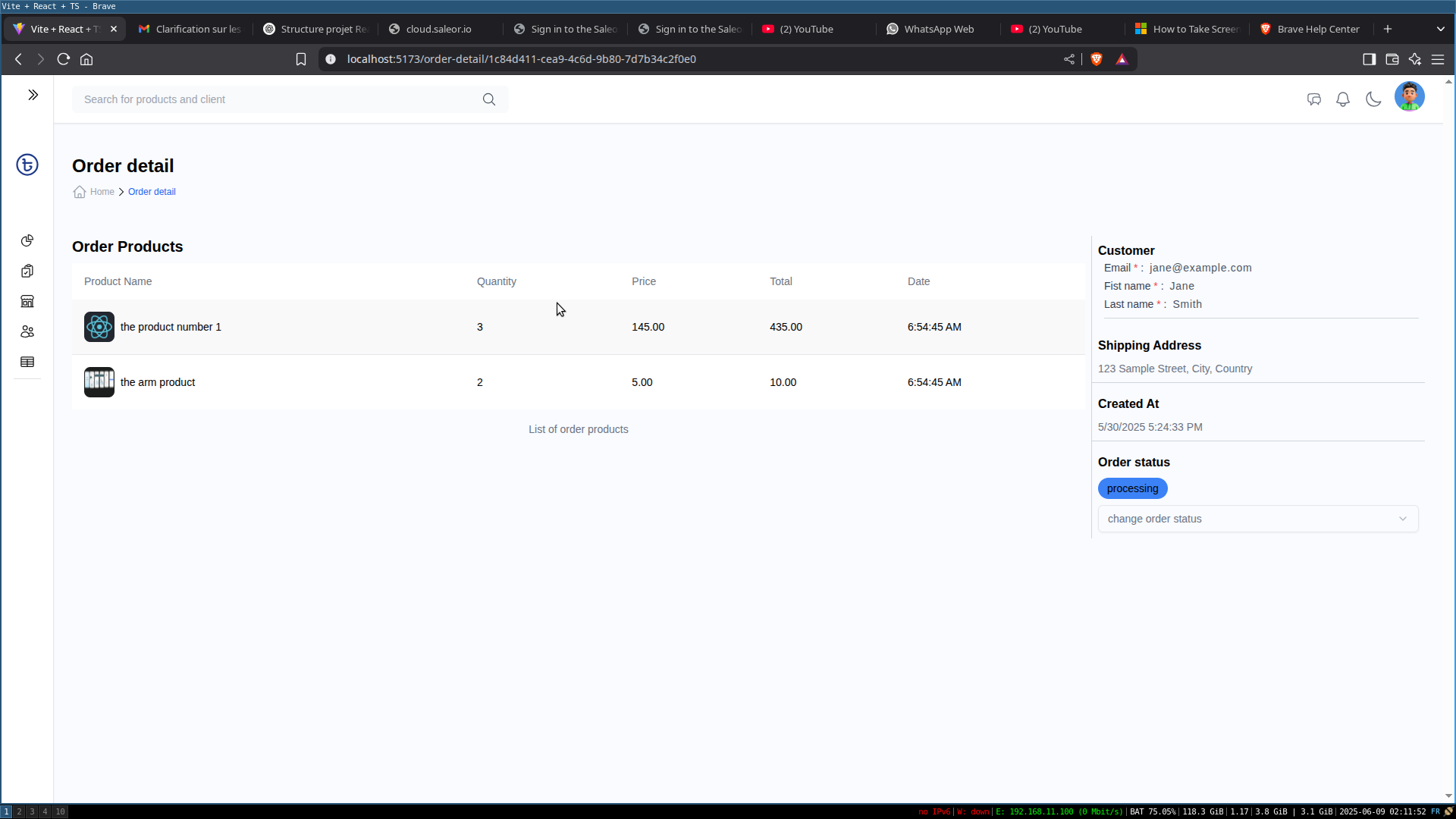
**Composants principaux de la page**

* **Section des produits de la commande (OrderProducts)**
  + **Tableau des produits** avec les colonnes :
  + Nom du produit (avec image)
  + Quantité
  + Prix unitaire
  + Total
  + Date
  + **Fonctionnalités** :
    - Navigation vers le détail du produit
    - Affichage des images des produits
    - Calcul des sous-totaux
* **Section des informations de la commande (OrderInformation)**
  + **Informations générales** :
    - Numéro de commande
    - Date de création
    - Statut de la commande
  + **Gestion du statut** :
    - Affichage du statut actuel
    - Menu déroulant pour modifier le statut
    - Options disponibles :
    - Pending
    - Processing
    - Completed
    - Cancelled
  + **Informations client** :
    - Nom du client
    - Email
    - Adresse de livraison

**Fonctionnalités principales**

* Visualisation détaillée des produits
* Modification du statut de la commande
* Navigation vers les détails des produits
* Affichage des informations client
* Suivi du paiement

Cette interface est conçue pour offrir une expérience utilisateur complète dans la gestion des détails d'une commande, avec une attention particulière portée à la visualisation des informations et à la facilité de modification du statut.

**Capture d’écran de la page de détail du commande**

### **Présentation de l'interface : Page des clients (customers)**

**Description générale**

La page des clients est une interface complète permettant de visualiser et gérer tous les clients du système. Elle offre une vue d'ensemble des clients avec des fonctionnalités de recherche et de filtrage.

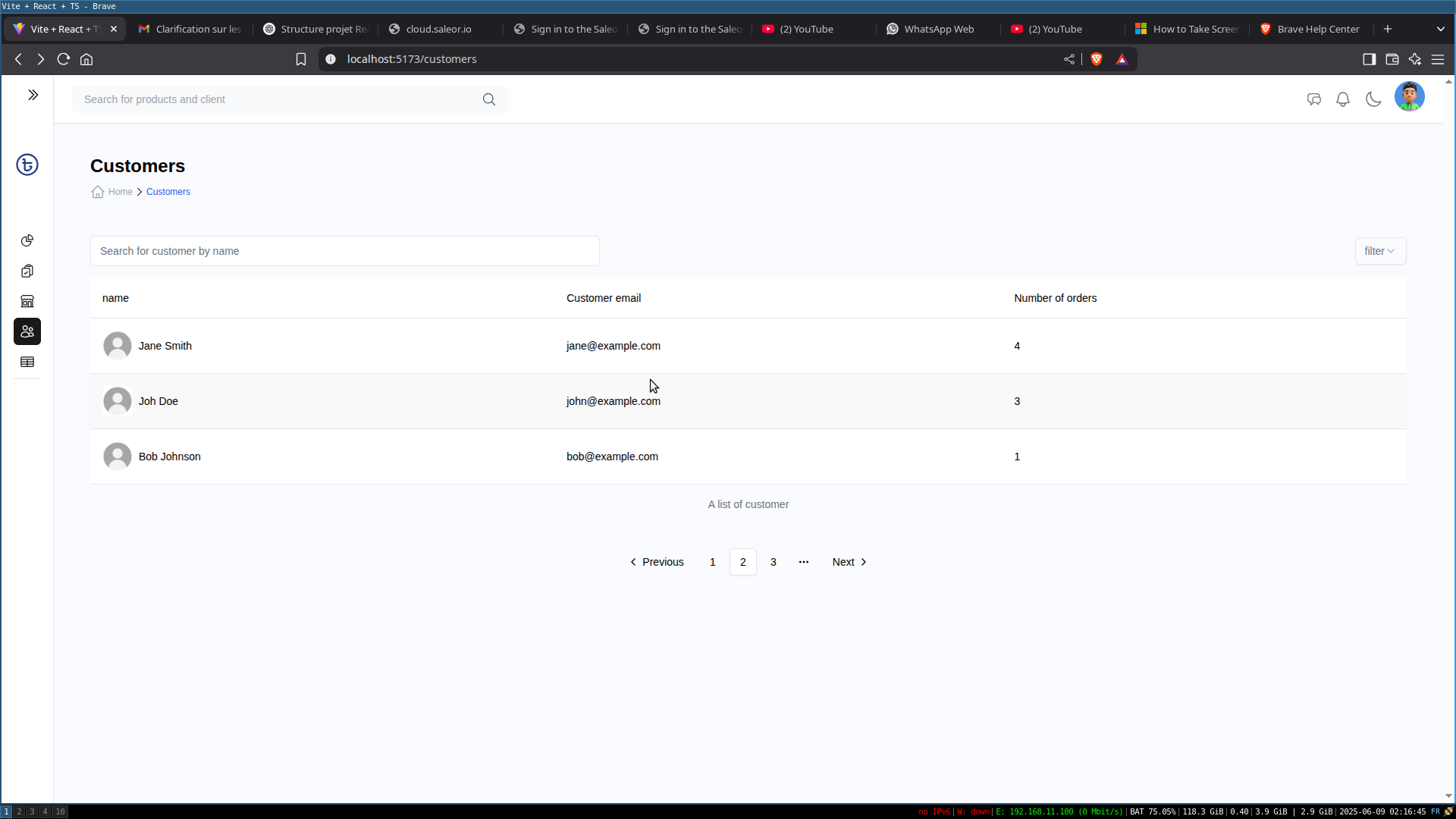
**Composants principaux de la page**

* **Section de filtrage (CustomersFilter)**
  + **Barre de recherche**
    - Recherche par nom de client
* **Tableau des clients (CustomersTable)**
  + **Colonnes** :
    - Nom du client (avec avatar)
    - Email du client
    - Nombre de commandes
  + **Fonctionnalités** :
    - Navigation vers le détail du client
    - Affichage des avatars
    - Indicateur de chargement

**Fonctionnalités principales**

* Visualisation détaillée des produits
* Modification du statut de la commande
* Navigation vers les détails des produits
* Affichage des informations client
* Suivi du paiement

Cette interface est conçue pour offrir une expérience utilisateur complète dans la gestion des clients, avec une attention particulière portée à la recherche et au filtrage des données.

**Capture d’écran de la page des clients (customers)**

### **Présentation de l'interface : Page de détail de client (customer detail)**

**Description générale**

La page de détail des clients est une interface complète permettant de visualiser et gérer les informations détaillées d'un client spécifique. Elle offre une vue détaillée des informations personnelles du client et de son historique de commandes.

**Composants principaux de la page**

* **Barre d'actions (EditCustomerButtons)**
  + - **Mode édition** :
      * Bouton "Edit" pour modifier les informations
      * Bouton "Delete" pour supprimer le client
    - **Mode modification** :
      * Bouton "Save Changes" pour enregistrer les modifications
      * Bouton "Cancel" pour annuler les modifications
* **Informations générales (CustomerGeneralInformation)**
  + - **Informations personnelles** :
    - Prénom (champ obligatoire)
    - Nom (champ obligatoire)
    - Email (non modifiable)
    - Numéro de téléphone (champ obligatoire)
    - **Fonctionnalités** :
      * Champs éditables en mode modification
      * Validation des champs obligatoires
      * Affichage des indicateurs de champs requis
* **Historique des commandes (CustomerOrders)**
  + - **Tableau des commandes** avec les colonnes :
      * ID de la commande
      * Statut de la commande
      * Date
      * Montant total
    - **Fonctionnalités** :
      * Navigation vers le détail de la commande
      * Affichage du statut avec code couleur
      * Indicateur de chargement

**Fonctionnalités principales**

* Fonctionnalités principales
* Modification des informations client
* Suppression du client
* Visualisation de l'historique des commandes
* Navigation vers les détails des commandes
* Gestion des états de chargement

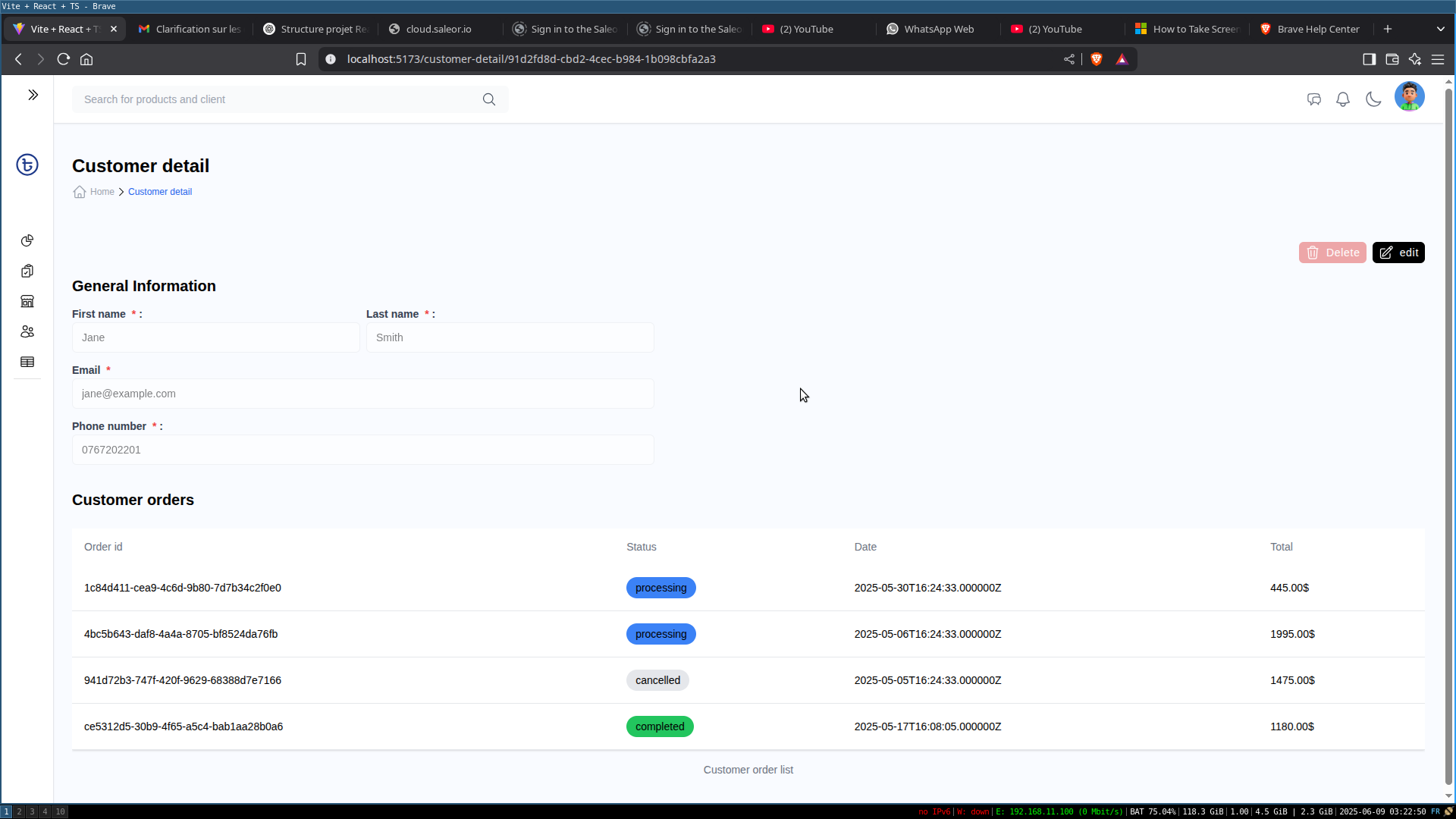
**Gestion des erreurs**

Le système affiche des messages de notification pour :

* + Erreurs de chargement des données
  + Échecs de sauvegarde des modifications
  + Problèmes de suppression
  + Erreurs de validation des champs

Cette interface est conçue pour offrir une expérience utilisateur complète dans la gestion des clients, avec une attention particulière portée à la recherche et au filtrage des données.

**Capture d’écran de la page de détail des clients (customers detail )**



## Tests Réalisés

**Tests d'Authentification (auth.php)**

* **Tests de Connexion**
  + **POST /auth/login**
    - Test avec identifiants valides
    - Test avec identifiants invalides
    - Vérification du token JWT retourné
* **Tests de Déconnexion**
  + **GET /auth/logout**
    - Test de déconnexion avec token valide
    - Vérification de l'invalidation du token

**Tests de Gestion des Produits (product.php)**

* **Tests CRUD Produits**
  + **GET /products**
    - Test de récupération de la liste des produits
    - Test avec filtres et pagination
  + **GET /products/{id}**
    - Test de récupération d'un produit spécifique
    - Test avec ID invalide
  + **POST /products**
    - Test de création d'un nouveau produit
    - Vérification des données requises
  + **POST /products/update/{id}**
    - Test de mise à jour d'un produit
    - Vérification des modifications
  + **DELETE /products/{id}**
    - Test de suppression d'un produit
    - Vérification de la suppression
* **Tests de Fonctionnalités Spécifiques**
  + **GET /products/{id}/toggle-status**
    - Test de changement de statut
    - Vérification du nouveau statut
  + **GET /products/statistics**
    - Test de récupération des statistiques
    - Vérification des données statistiques

**Tests de Gestion des Catégories (category.php)**

* **Tests CRUD Catégories**
  + **GET /categories**
    - Test de récupération de la liste des catégories
    - Test avec filtres
  + **GET /categories/{id}**
    - Test de récupération d'une catégorie spécifique
    - Test avec ID invalide
  + **POST /categories**
    - Test de création d'une nouvelle catégorie
    - Vérification des données requises
  + **POST /categories/update/{id}**
    - Test de mise à jour d'une catégorie
    - Vérification des modifications

**Tests de Fonctionnalités Spécifiques**

* + **GET /categories/{id}/toggle-status**
    - Test de changement de statut
    - Vérification du nouveau statut
  + **GET /categories/statistics**
    - Test de récupération des statistiques
    - Vérification des données statistiques

**Tests de Gestion des Commandes (order.php)**

* + - **Tests de Consultation**
    - **GET /orders**
      * Test de récupération de la liste des commandes
      * Test avec filtres
    - **GET /orders/{id}**
      * Test de récupération d'une commande spécifique
      * Test avec ID invalide
    - **GET /orders/customer/{id}**
      * Test de récupération des commandes d'un client
      * Vérification des données retournées
  + **Tests de Statistiques**
    - **GET /orders/statistics**
      * Test de récupération des statistiques
      * Vérification des données statistiques

**Tests de Gestion des Clients (customers.php)**

* **Tests CRUD Clients**
  + **GET /customers**
    - Test de récupération de la liste des clients
    - Test avec filtres
* **GET /customers/{id}**
  + - Test de récupération d'un client spécifique
    - Test avec ID invalide
* **PUT /customers/{id}**
  + - Test de mise à jour d'un client
    - Vérification des modifications
* **DELETE /customers/{id}**
  + - Test de suppression d'un client
    - Vérification de la suppression

**Tests de Fonctionnalités Spécifiques**

* **POST /users/{id}/toggle-status**
  + Test de changement de statut
  + Vérification du nouveau statut
* **GET /users/statistics**
  + Test de récupération des statistiques
  + Vérification des données statistiques

**Intégration Front-end**

* Tests d'intégration de toutes les API avec le front-end
* Vérification du bon fonctionnement des composants
* Tests des formulaires et des interactions utilisateur
* Validation des réponses et des erreurs

**Résultats des Tests**

* Toutes les routes API sont fonctionnelles
* Les réponses sont conformes aux attentes
* L'intégration front-end est réussie
* Les fonctionnalités de base sont opérationnelles

Cette suite de tests couvre l'ensemble des fonctionnalités implémentées dans votre application, avec une attention particulière aux routes API et à leur intégration avec le front-ent

## La qualité du code

### Évaluation de la Qualité du Code Front-end

**Architecture et Structure**

* Organisation claire des composants avec une structure de dossiers logique
* Séparation des responsabilités entre composants, hooks et types
* Utilisation de composants réutilisables (ex: Breadcrumb, Button, Table)
* Implémentation du pattern de composition avec des composants UI de base

**Sécurité**

* Validation des formulaires avec Formik et Yup
* Gestion sécurisée des tokens d'authentification
* Protection des routes sensibles

**Performance**

* Utilisation de useDebounce pour optimiser les recherches
* Gestion efficace des états avec useState et useEffect
* Composants optimisés avec des re-rendus contrôlés

**Maintenabilité**

* Code TypeScript bien typé
* Composants modulaires et réutilisables
* Structure de projet claire

**Documentation (6/10)**

* Types TypeScript bien définis
* Commentaires sur les fonctions principales

**Gestion des Erreurs**

* Utilisation de try/catch pour la gestion des erreurs
* Notifications utilisateur pour les erreurs
* Gestion des états de chargement

**Standards de Code**

* Style de code cohérent
* Utilisation de conventions de nommage
* Code bien formaté
* Utilisation de composants UI modernes

**Conclusion**

Le code front-end présente une bonne qualité globale avec une architecture solide et des bonnes pratiques de développement React/TypeScript.

### Évaluation de la Qualité du Code Back-end

**Architecture et Structure**

* Organisation claire suivant le pattern MVC
* Séparation des responsabilités entre Controllers, Models et Services
* Utilisation des middlewares pour la gestion des requêtes
* Structure de dossiers logique et cohérente

**Sécurité**

* Validation des données avec des règles strictes
* Protection des routes avec middleware d'authentification
* Gestion sécurisée des tokens JWT
* Validation des entrées utilisateur

**Performance**

* Utilisation efficace des requêtes Eloquent
* Gestion des relations entre modèles

**Maintenabilité**

* Code bien structuré et lisible
* Utilisation de services pour la logique métier
* Gestion claire des dépendances

**Gestion des Erreurs**

* Utilisation de try/catch pour la gestion des erreurs
* Réponses d'erreur standardisées
* Logging des erreurs

**Standards de Code**

* Style de code cohérent
* Utilisation de conventions de nommage
* Code bien formaté
* Respect des standards PSR

**Conclusion**

Le code back-end présente une bonne qualité globale avec une architecture solide et des bonnes pratiques de développement Laravel.