ROYAUME DU MAROC

ECOLE NATIONALE SUPERIEURE DES MINES DE RABAT

Rapport du **P**rojet **F**édérateur

Département : Informatique

Filière : Génie Informatique

Sous le thème :

Développement d'une application de tableau de bord financier

Réalisé par :

BITAR Achraf

Encadré par :

M. TOULNI Hamza

Année universitaire : 2024 / 2025

Dédicace

Je dédie ce travail à mes parents pour leur amour et leur soutien inconditionnel, à mes amis pour leur présence et leurs encouragements, ainsi qu'à mes professeurs dont les enseignements et les conseils m'ont guidé tout au long de ce parcours.

Remerciements

Je tiens à exprimer ma profonde gratitude envers mes parents, qui incarnent pour moi le symbole de la bienveillance. Leur amour inconditionnel, leur soutien sans faille, et leur dévouement constant ont été une source inépuisable de motivation tout au long de mes études.

Je remercie également mes amis pour leur soutien indéfectible et leur précieuse assistance durant cette période de stage et tout au long de l'année.

Je souhaite exprimer ma reconnaissance à mon encadrant, Monsieur TOULNI Hamza, pour ses conseils avisés et son appui constant tout au long de ce projet.

Enfin, je remercie chaleureusement mes enseignants du département d'Informatique de l'École Nationale Supérieure des Mines de Rabat, pour la qualité exceptionnelle de leurs enseignements qui ont grandement contribué à ma formation.

Résumé

Ce projet consiste à développer une application de tableau de bord financier interactive et moderne en utilisant la stack **MERN** (MongoDB, Express.js, React et Node.js) avec l'intégration de modèles d'apprentissage automatique pour prédire les tendances financières.

L'objectif est d'aider les décideurs financiers à visualiser clairement leurs données (revenus, profits, dépenses) grâce à des graphiques interactifs et ergonomiques tout en anticipant les évolutions futures grâce aux prédictions générées par des algorithmes de Machine Learning.

Grâce à React.js pour l'interface utilisateur, Redux Toolkit pour la gestion d'état, et Recharts pour les visualisations, l'application offre une expérience fluide et intuitive, même pour les utilisateurs non experts.

Le backend développé en Node.js et Express.js assure la communication avec une base de données MongoDB pour centraliser les données et une API Python pour les prédictions. Ce projet répond au besoin croissant des entreprises d'avoir des outils simples, performants et intelligents pour faciliter la prise de décision financière.

Abstract

This project involves developing an interactive and modern financial dashboard application using the MERN stack (MongoDB, Express.js, React and Node.js) with the integration of machine learning models to predict financial trends.

The objective is to help financial decision-makers clearly visualize their data (income, profits, expenses) thanks to interactive and ergonomic graphs while anticipating future developments thanks to predictions generated by Machine Learning algorithms.

Thanks to React.js for the user interface, Redux Toolkit for state management, and Recharts for visualizations, the application provides a smooth and intuitive experience, even for non-expert users.

The backend developed in Node.js and Express.js ensures communication with a MongoDB database to centralize data and a Python API for predictions. This project responds to the growing need for businesses to have simple, efficient and intelligent tools to facilitate financial decision-making.

Table of Contents

Liste des figures

Introduction générale

Dans un monde où les décisions financières jouent un rôle crucial dans le succès des entreprises, il est essentiel de disposer d'outils modernes et intuitifs pour analyser les données et anticiper les tendances. Les solutions traditionnelles, souvent complexes et rigides, ne répondent plus aux besoins actuels de clarté et de rapidité d'analyse.

Ce projet vise à développer une application de tableau de bord financier interactive qui combine visualisation des données et prédictions financières grâce à l'intelligence artificielle. Conçue avec des technologies modernes comme React, Node.js, et MongoDB, l'application permettra de centraliser les données financières, de les présenter de manière claire et compréhensible via des graphiques interactifs, et d'anticiper les évolutions futures à l'aide de modèles de Machine Learning**.**

L'objectif est de proposer un outil simple, fluide et accessible, qui aide les décideurs, même non-experts, à mieux comprendre leurs finances et à prendre des décisions éclairées pour l'avenir.

Chapitre 1 : Introduction et contexte du projet

1. Contexte et motivation

Dans le monde des affaires d'aujourd'hui, les données financières jouent un rôle clé dans la prise de décision. Cependant, ces données sont souvent dispersées et difficiles à interpréter, ce qui complique le travail des décideurs. Les entreprises recherchent donc des outils modernes qui leur permettent de visualiser leurs indicateurs financiers de manière claire et interactive.

De plus, il ne suffit plus d'analyser les performances passées. Les entreprises ont besoin d'anticiper les tendances futures pour prendre des décisions éclairées et rester compétitives. Grâce à l'apprentissage automatique, il est désormais possible de prédire ces tendances et d'identifier les opportunités ou les risques à venir. Ce projet répond à ce double besoin de visualisation intuitive et de prédictions fiables, tout en offrant une expérience utilisateur adaptée aux décideurs, même non experts en technologie.

2. Problématique :

Le projet vise à résoudre plusieurs défis majeurs liés à l'analyse et à la gestion des données financières :

1. **Comment centraliser et visualiser les données financières de manière claire et compréhensible ?**
    - Les entreprises utilisent souvent plusieurs outils pour gérer leurs finances, rendant difficile l'accès à une vision globale et synthétique.
    - Une interface interactive doit permettre de représenter visuellement les indicateurs clés comme les revenus, les dépenses et les profits.
2. **Comment anticiper les tendances financières grâce à des modèles de machine learning ?**
    - L'intégration de modèles prédictifs permet d'identifier les tendances à venir et d'adapter les stratégies en conséquence.
3. **Comment offrir une expérience utilisateur fluide et interactive, adaptée aux décideurs financiers ?**
    - L'application doit proposer une navigation intuitive et une accessibilité optimisée pour une prise en main rapide.

3.Objectives

Ce projet a pour objectif de concevoir une application moderne et efficace permettant aux entreprises de mieux exploiter leurs données financières.

1. **Développer une application interactive pour visualiser les indicateurs financiers**
    - Permettre aux utilisateurs de consulter en temps réel les informations clés via des graphiques et des tableaux dynamiques.
    - Faciliter l'interprétation des données grâce à des représentations visuelles claires et intuitives.
2. **Intégrer des algorithmes de machine learning pour réaliser des prédictions financières**
    - Mettre en place des modèles capables de prévoir l'évolution des revenus, des dépenses et autres indicateurs stratégiques.
    - Aider les entreprises à anticiper les risques et opportunités grâce à des analyses basées sur l'intelligence artificielle.

Chapitre 2 : Analyse des Besoins

1.Introduction

L'analyse des besoins est une étape essentielle pour définir les attentes des utilisateurs et les fonctionnalités clés de l'application. Elle permet d'identifier les exigences fonctionnelles et techniques du projet en se basant sur les besoins des entreprises et des décideurs financiers.

2. Expression des Besoins

L'application s'adresse principalement aux acteurs suivants :

- **Les dirigeants et décideurs financiers :**
  - Besoin d'une vision globale et synthétique de la situation financière.
  - Capacité à prendre des décisions stratégiques sur la base de données précises.
- **Les analystes financiers et comptables :**
  - Analyser les tendances et détecter des anomalies financières.
  - Générer des rapports et proposer des recommandations adaptées.
- **Les gestionnaires et responsables de budget :**
  - Suivre les dépenses et revenus pour une gestion budgétaire optimisée.
  - Ajuster les stratégies en fonction des prévisions financières.

2.2 Besoins Fonctionnels

L'application doit proposer plusieurs fonctionnalités clés :

- **Centralisation et gestion des données financières :**
  - Stockage sécurisé dans une base de données MongoDB.
  - Importation et exportation de fichiers (CSV, Excel, API).
- **Visualisation interactive des indicateurs clés** :
  - Tableaux de bord dynamiques avec graphiques interactifs (Recharts).
  - Filtres et options de personnalisation des données.
- **Prédictions financières avec Machine Learning** :
  - Analyse des tendances grâce à des algorithmes d'apprentissage automatique.
  - Affichage des prévisions sous forme de graphiques évolutifs.

2.3 Besoins Non Fonctionnels

- **Performance et rapidité** : Optimisation des requêtes API pour un chargement rapide.
- **Scalabilité** : Architecture évolutive facilitant l'ajout de nouvelles fonctionnalités.
- **Sécurité** : Protection contre les attaques XSS, CSRF, injections SQL.

- **Interopérabilité** : Compatibilité avec d'autres systèmes via API REST.
- **Expérience utilisateur fluide** : Interface intuitive adaptée aux utilisateurs non experts.

3. Cahier des Charges

Fonctionnalité	Description
Tableaux de bord financiers	Affichage des indicateurs clés sous forme de graphiques interactifs.
Prédictions financières	Intégration de modèles de Machine Learning pour prévoir les tendances.
Gestion des données	Importation et exportation des données financières (CSV, Excel).

3.2 Architecture Technique

L'application repose sur une architecture MERN (MongoDB, Express.js, React, Node.js) avec une intégration de Machine Learning via Python.

- **Frontend** : React.js + Redux Toolkit + Material UI + Recharts.
- **Backend** : Node.js + Express.js.
- **Base de données** : MongoDB hébergé sur MongoDB Atlas.
- **Machine Learning** : API en Python (Flask/FastAPI) intégrant les modèles prédictifs.

3.3 Contraintes et Exigences

1. **Performance** :
    - Chargement optimisé des données avec Redux Toolkit.
    - Réduction du temps de réponse grâce à l'optimisation des requêtes API.
2. **Interopérabilité** :
    - Intégration avec d'autres outils financiers ou ERP via API.
3. **Expérience Utilisateur** :
    - Interface moderne et ergonomique avec Material UI.
    - Application responsive adaptée aux différents supports (PC, tablette, mobile).

Chapitre 3 : Architecture et Conception de l’application

1. Vue d'Ensemble de l'Architecture

L'application de tableau de bord financier repose sur une architecture **MERN stack** (MongoDB, Express.js, React, Node.js), complétée par une intégration de modèles de **Machine Learning** via une API Python. Cette architecture a été choisie pour sa flexibilité, sa scalabilité et sa capacité à gérer des applications modernes et interactives.

L'architecture est divisée en trois couches principales :

- **Frontend** : Développé avec **React.js**, il est responsable de l'interface utilisateur et de l'interaction avec l'utilisateur.
- **Backend** : Développé avec **Node.js** et **Express.js**, il gère la logique métier, la communication avec la base de données et l'intégration des prédictions financières.
- **Base de Données** : **MongoDB** est utilisé pour stocker les données financières de manière structurée et flexible.
- **Machine Learning** : Une API Python (avec **Flask** ou **FastAPI**) est utilisée pour exécuter les modèles de prédiction et renvoyer les résultats au backend.

2. Frontend

2.1 Technologies Utilisées

- **React.js** : Bibliothèque JavaScript pour la création d'interfaces utilisateur interactives.
- **Redux Toolkit** : Gestion d'état centralisée pour une meilleure gestion des données et des interactions.
- **Material UI** : Bibliothèque de composants UI pour une interface moderne et responsive.
- **Recharts** : Bibliothèque de visualisation de données pour la création de graphiques interactifs.

2.2 Structure des Composants

Le frontend est structuré en plusieurs composants React modulaires, chacun responsable d'une fonctionnalité spécifique :

- **Dashboard** : Affiche les graphiques et les indicateurs financiers en temps réel.
- **Prediction** : Affiche les prédictions financières générées par les modèles de Machine Learning.

- **Navigation** : Barre de navigation pour une expérience utilisateur fluide.

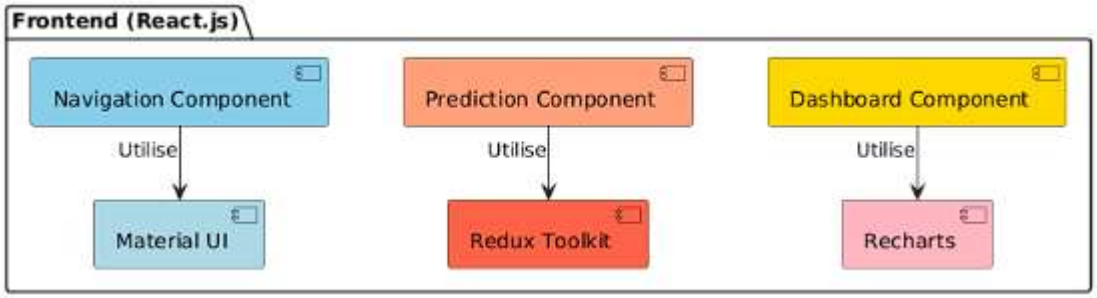

Figure 1: Composants Frontend

2.3 Interaction avec le Backend

Le frontend communique avec le backend via des **requêtes HTTP** (GET, POST) pour récupérer les données financières et les prédictions. Les données sont ensuite affichées sous forme de graphiques interactifs grâce à **Recharts**.

3. Backend

3.1 Technologies Utilisées

- **Node.js** : Environnement d'exécution JavaScript pour le développement côté serveur.
- **Express.js** : Framework pour la création d'API RESTful.
- **MongoDB** : Base de données NoSQL pour le stockage des données financières.
- **Mongoose** : Bibliothèque pour la modélisation des données MongoDB.

3.2 Structure de l'API

Le backend expose plusieurs routes API pour gérer les différentes fonctionnalités de l'application :

- **/api/financial-data** : Récupère les données financières stockées dans MongoDB.
- **/api/predictions** : Envoie les données financières à l'API Python pour obtenir des prédictions.

3.3 Gestion des Données

Les données financières sont stockées dans une base de données **MongoDB** hébergée sur **MongoDB Atlas**. Les collections MongoDB sont structurées pour stocker les informations suivantes :

- **Revenus** : Montants des revenus par période.
- **Dépenses** : Montants des dépenses par catégorie.
- **Profits** : Calcul des profits basés sur les revenus et les dépenses.

3.4 Sécurité

Le backend intègre des mesures de sécurité pour protéger les données et les API :

- **Authentification** : Utilisation de JSON Web Tokens (JWT) pour sécuriser les routes API.
- **Validation des Données** : Validation des entrées utilisateur pour prévenir les injections SQL et les attaques XSS

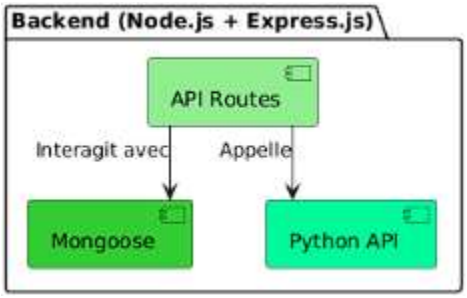

Figure 2: Composant Backend

4. Base de Données

4.1 MongoDB

MongoDB a été choisi pour sa flexibilité et sa capacité à gérer des données non structurées. Les données financières sont stockées dans des collections MongoDB, avec des schémas définis pour chaque type de données (revenus, dépenses, profits).

4.2 Schéma des Données

Exemple de données :


```
{
  "date": "2023-10-01",
  "prix": 50000,
  "categorie": "Ventes",
  "description": "Ventes du mois d'octobre"
}
```


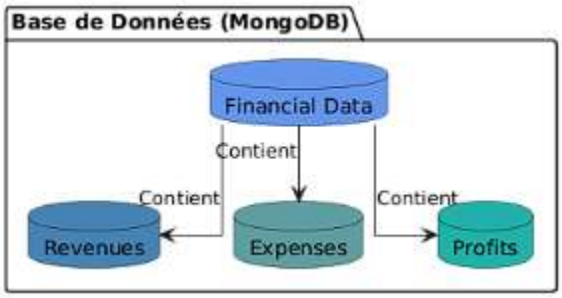

Figure 3: Composants Base de Données

5. Machine Learning

5.1 Technologies Utilisées

- **Python** : Langage de programmation pour le développement des modèles de Machine Learning.
- **Flask/FastAPI** : Framework pour la création de l'API de prédiction.
- **Scikit-learn** : Bibliothèque pour l'entraînement des modèles de Machine Learning.
- **Pandas** : Bibliothèque pour la manipulation des données financières.

5.2 Modèles de Prédiction

Les modèles de Machine Learning sont entraînés sur des données historiques pour prédire les tendances financières. Les algorithmes utilisés incluent :

- **Régression Linéaire** : Pour prédire les revenus futurs.
- **ARIMA** : Pour l'analyse des séries temporelles et la prédiction des dépenses.
- **LSTM** : Pour des prédictions plus complexes basées sur des réseaux de neurones.

5.3 Intégration avec le Backend

L'API Python est appelée par le backend via des requêtes HTTP. Les données financières sont envoyées à l'API, qui renvoie les prédictions sous forme de JSON. Ces prédictions sont ensuite affichées dans le frontend.

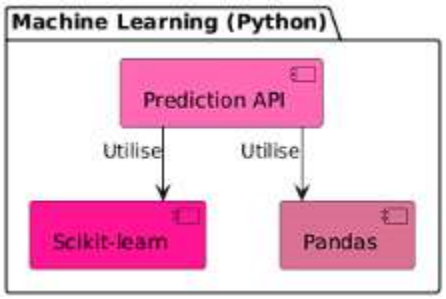

Figure 4: Composants de l'API de prediction

6. Conception de l'application

La conception de l'application est une étape cruciale qui permet de définir la structure et les interactions entre les différents composants du système. Cette section détaille les diagrammes UML utilisés pour modéliser l'application, ainsi que les choix de conception qui ont été faits pour répondre aux besoins fonctionnels et non fonctionnels.

6.1 Diagramme de classe

Le diagramme de classe représente la structure statique de l'application en définissant les entités principales, leurs attributs et les relations entre elles.

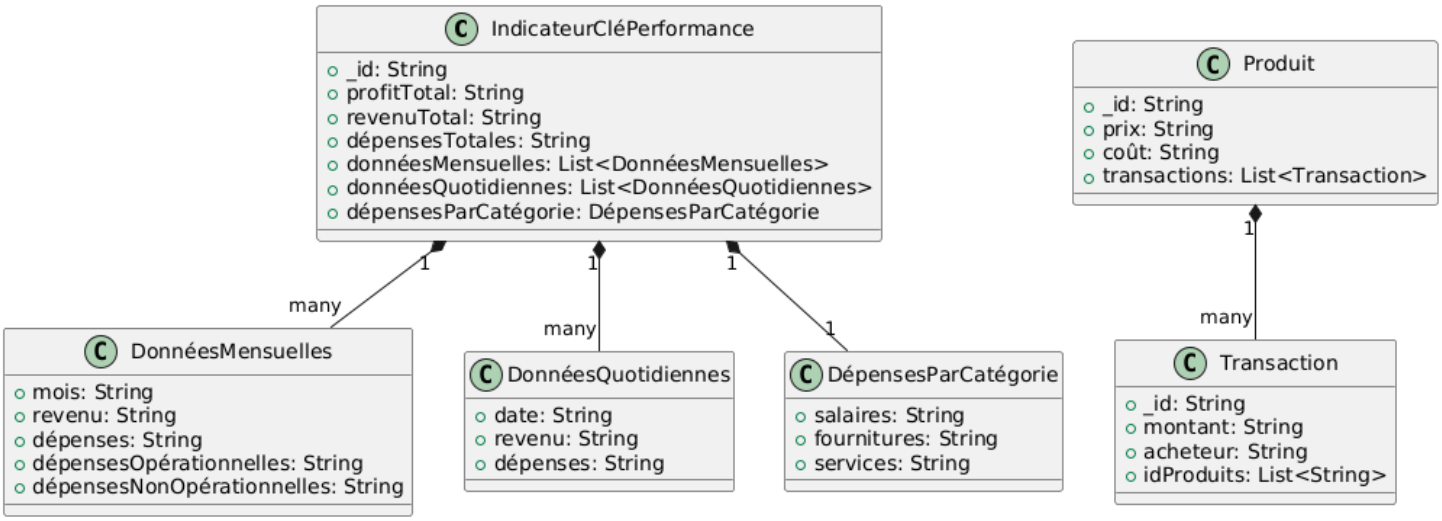

Figure 5*: Diagramme de classe*

Classes Principales :

1. **IndicateurCléPerformance** :
    - **Attributs** :
        - id: String
        - profitTotal: String
        - revenuTotal: String
        - dépensesTotales: String
        - donnéesMensuelles: List<DonnéesMensuelles> (relation "many")
        - donnéesQuotidiennes: List<DonnéesQuotidiennes> (relation "many")
        - dépensesParCatégorie: DépensesParCatégorie (relation "one")
    - **Méthodes** :
        - calculerProfitTotal()
        - calculerRevenuTotal()
        - calculerDépensesTotales()
2. **DonnéesMensuelles** :
    - **Attributs** :

        - mois: String
        - revenu: String
        - dépenses: String
        - dépensesOpérationnelles: String
        - dépensesNonOpérationnelles: String
    - **Méthodes** :
        - calculerDépensesOpérationnelles()
        - calculerDépensesNonOpérationnelles()
3. **DonnéesQuotidiennes** :
    - **Attributs** :
        - date: String
        - revenu: String
        - dépenses: String
    - **Méthodes** :
        - calculerRevenuQuotidien()
        - calculerDépensesQuotidiennes()
4. **DépensesParCatégorie** :
    - **Attributs** :
        - salaires: String
        - fournitures: String
        - services: String
    - **Méthodes** :
        - calculerTotalDépensesParCatégorie()
5. **Produit** :
    - **Attributs** :
        - id: String
        - prix: String

        - coût: String
        - transactions: List<Transaction> (relation "many")
    - **Méthodes** :
        - calculerProfit()
        - ajouterTransaction(transaction: Transaction)

6. **Transaction** :
    - **Attributs** :
        - id: String
        - montant: String
        - acheteur: String
        - idProduits: List<String> (relation "many")
    - **Méthodes** :
        - calculerMontantTotal()

6.2 Diagramme de Séquence

Le diagramme de séquence permet de visualiser le déroulement des échanges entre le frontend, le backend et la base de données.

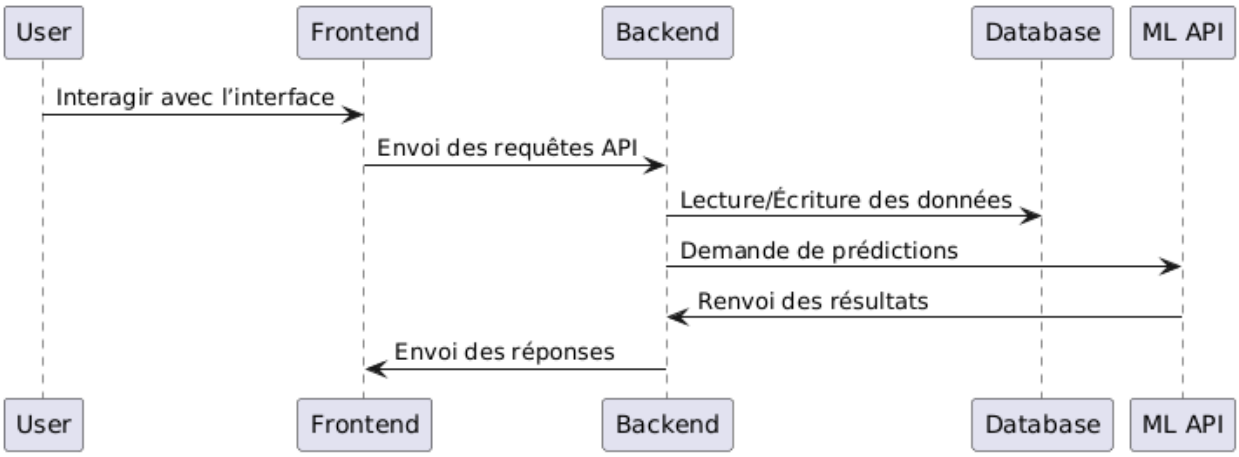

Figure 6: Diagramme de séquencee

6.3 Diagramme de déploiement

Le diagramme de déploiement montre comment les différents composants de l'application sont déployés sur les serveurs et les services cloud :

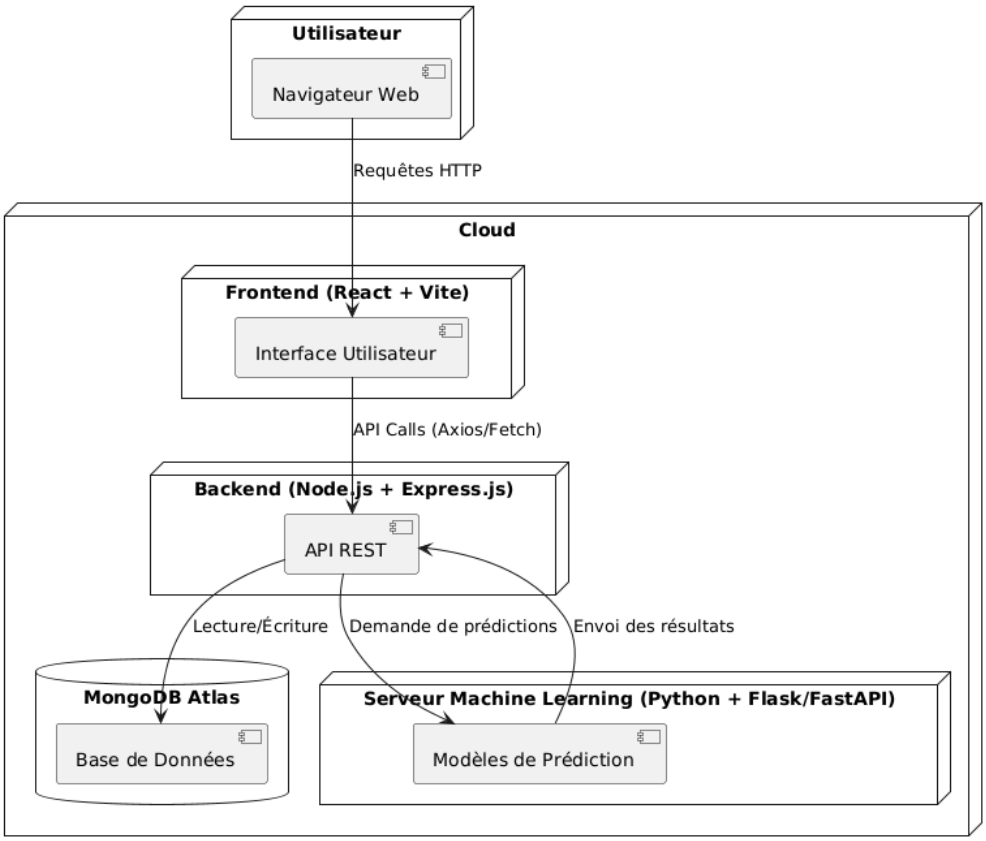

Figure 7: Diagramme de déploiement

6.5 Choix de conception

- **Modularité**

L'application a été conçue avec une architecture modulaire pour faciliter la maintenance et l'ajout de nouvelles fonctionnalités. Chaque composant (frontend, backend, base de données, API Python) est indépendant et communique via des API REST.

- **Scalabilité**

L'utilisation de services cloud comme MongoDB Atlas, Heroku, et AWS permet de garantir la scalabilité de l'application. Les services cloud peuvent être facilement mis à l'échelle pour gérer un nombre croissant d'utilisateurs et de données.

Chapitre 4 : Mise en œuvre et développement

1. Introduction

Ce chapitre décrit les étapes de développement de l'application, en mettant l'accent sur les choix techniques, les défis rencontrés et les solutions apportées.

2. Développement du Frontend

2.1 Configuration de l'environnement de développement

- **Initialisation du projet** : Utilisation de « create-react-app » pour configurer rapidement l'environnement de développement.
- **Gestion des dépendances** : Installation des bibliothèques nécessaires (React, Redux Toolkit, Material UI, Recharts).

2.2 Création des composants principaux

- **Dashboard** : Développement des composants pour afficher les graphiques interactifs (revenus, dépenses, profits).

  Exemple de composant :

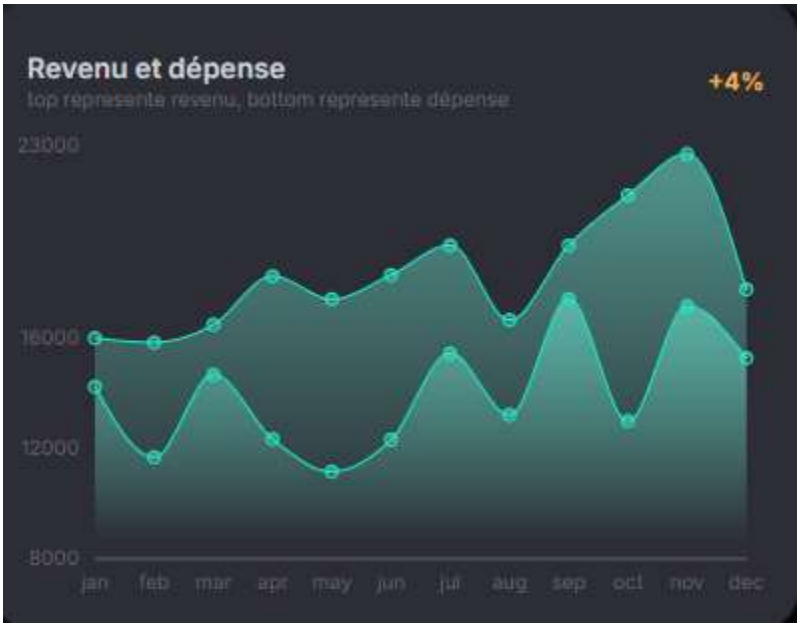

Figure 8: Composant « Revenu et dépense »

- **Prédictions** : Intégration des résultats de prédiction dans des graphiques évolutifs.

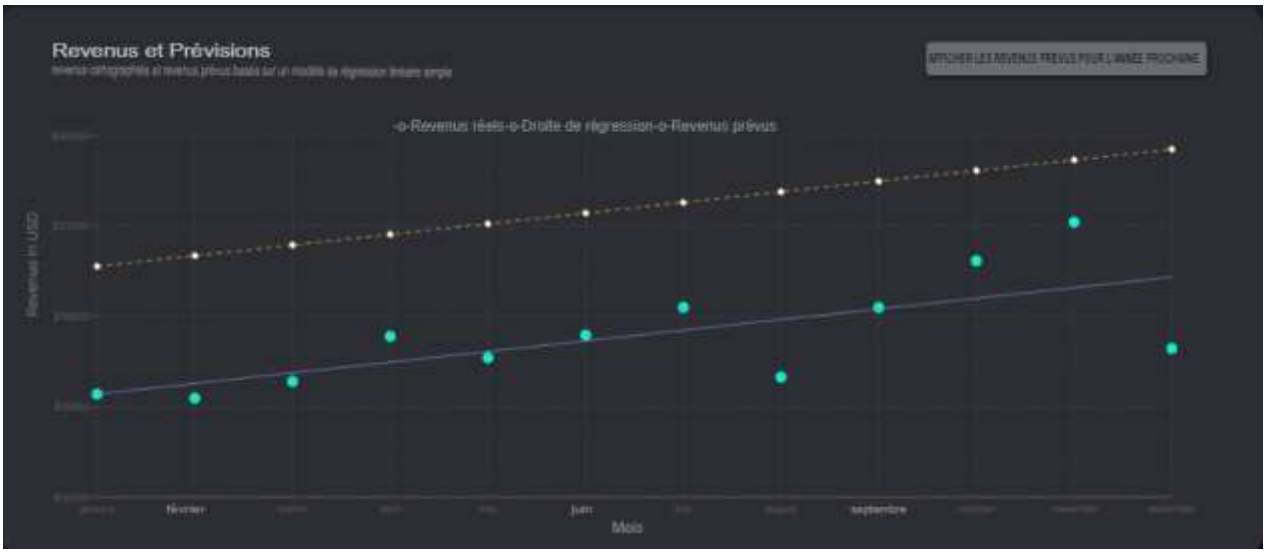

Figure 9: Page de prédiction

- **Navigation** : Création d'une barre de navigation responsive avec Material UI.

Figure 10: Material UI barre de navigation

2.3 Intégration des graphiques avec Recharts

- **Personnalisation des graphiques** : Ajout de filtres et d'options de personnalisation pour les utilisateurs.
- **Interactivité** : Mise en place de tooltips et de zoom pour une meilleure expérience utilisateur.

Figure 11: Dashboard interactive

3. Développement du Backend

3.1 Configuration de l'environnement backend

- **Initialisation du projet** : Utilisation de Express.js pour créer le serveur backend.
- **Gestion des dépendances** : Installation des bibliothèques nécessaires (Mongoose, JWT, etc.).

3.2 Création des routes API

- **Récupération des données financières** : Développement de la route /api/financial-data pour récupérer les données depuis MongoDB.
- **Prédictions financières** : Création de la route /api/predictions pour interagir avec l'API Python.

3.3 Gestion des données avec MongoDB

- **Connexion à MongoDB Atlas** : Configuration de la connexion à la base de données cloud.

- **Modélisation des données** : Utilisation de Mongoose pour définir les schémas des collections :

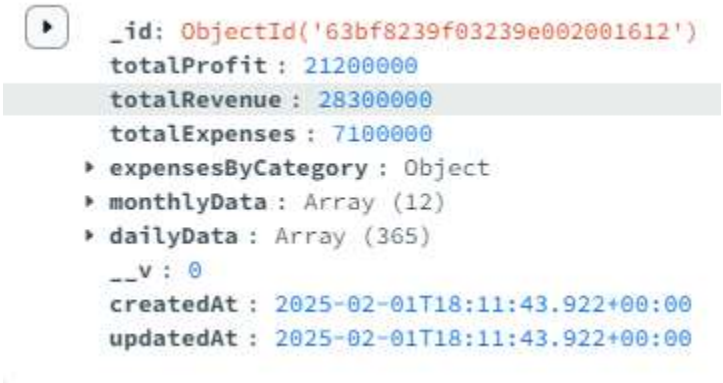

Figure 12: Instance de kpi

Figure 13: Instance de produit

Figure 14: Instance de transaction

3.4 Sécurité et authentification

- **JWT (JSON Web Tokens)** : Mise en place d'un système d'authentification sécurisé pour protéger les routes API.

4. Intégration de l'API Machine Learning

4.1 Développement de l'API Python

- **Choix du framework** : Utilisation de Flask ou FastAPI pour créer l'API de prédiction.
- **Entraînement des modèles** : Utilisation de Scikit-learn et Pandas pour entraîner les modèles de Machine Learning.

4.2 Intégration avec le backend

- **Communication entre backend et API Python** : Envoi des données financières à l'API Python via des requêtes HTTP.
- **Récupération des prédictions** : Affichage des résultats de prédiction dans le frontend.

5. Test ET validation

5.1 Tests du frontend

- **Tests des composants React** : Utilisation de Jest et React Testing Library pour tester les composants individuels.
- **Tests des actions Redux** : Vérification du bon fonctionnement des actions et des reducers.

5.2 Tests du backend

- **Tests des routes API** : Utilisation de Mocha et Chai pour tester les routes Express.js.
- **Tests de la base de données** : Vérification des requêtes MongoDB avec Mongoose.

5.3 Tests d'intégration

- **Interaction frontend-backend** : Vérification de la communication entre le frontend et le backend.
- **Intégration avec l'API Python** : Tests des prédictions financières pour s'assurer de leur exactitude.

Chapitre 5 : Conclusion et perspectives

1. Conclusion

Ce chapitre résume les réalisations du projet, les défis surmontés et les résultats obtenus. Il met en avant les points forts de l'application et son impact potentiel sur la prise de décision financière.

2. Perspectives

2.1 Améliorations possibles

- **Intégration de nouveaux modèles de Machine Learning** : Exploration de modèles plus avancés pour améliorer la précision des prédictions.
- **Personnalisation des tableaux de bord** : Ajout de fonctionnalités pour permettre aux utilisateurs de personnaliser davantage leurs tableaux de bord.

2.2 Élargissement du public cible

- **Adaptation à d'autres secteurs** : Extension de l'application pour répondre aux besoins d'autres secteurs comme la santé ou l'éducation.
- **Internationalisation** : Ajout de la prise en charge de plusieurs langues pour toucher un public international.

2.3 Recherche et développement

- **Collaboration avec des experts en finance** : Travail en collaboration avec des experts pour affiner les modèles de prédiction.
- **Exploration de nouvelles technologies** : Intégration de technologies émergentes comme l'IA générative pour enrichir les fonctionnalités de l'application.

Annexes

1. Glossaire

- **MERN Stack** : Acronyme pour MongoDB, Express.js, React, et Node.js.
- **Machine Learning** : Domaine de l'intelligence artificielle qui permet aux systèmes d'apprendre à partir de données.
- **API REST** : Interface de programmation d'application qui utilise des requêtes HTTP pour communiquer entre le client et le serveur.

2. Bibliographie

- **Documentation officielle de React** : https://reactjs.org/docs/getting-started.html
- **Documentation officielle de Node.js** : https://nodejs.org/en/docs/
- **Documentation officielle de MongoDB** : https://docs.mongodb.com/

3. Code source

- **Lien vers le dépôt GitHub** : https://github.com/AchrafBitar/ProjetFederateurS5