Cahier des Charges – Application de gestion de budget (PocketTrack-app)

# 1. Introduction

## 1.1. Contexte et objectif

Dans un monde où les technologies web évoluent rapidement, il est essentiel pour les développeurs d’acquérir des compétences solides en React.js et Node.js, deux des technologies les plus utilisées pour le développement d’applications web modernes.  
  
Cette application vise à aider les utilisateurs à mieux gérer leurs finances personnelles en leur offrant une interface simple pour visualiser leurs revenus et dépenses.  
  
Objectif : Offrir une application web avec authentification sécurisée, tableau de bord et visualisation des revenus/dépenses à l’aide de graphiques.  
  
Ce projet a pour but d’initier les apprenants au développement full-stack en leur proposant de créer une application de gestion de budget (Pocket-Tracker). À travers ce projet, ils apprendront :  
• Les bases de React.js : composants, état, hooks, événements et communication avec une API.  
• Le développement backend avec Node.js et Express : création d’une API REST pour gérer le budget.  
• La connexion avec une base de données SQL Server pour stocker et manipuler les données.

## 1.2. Description Fonctionnelle

Application web développée dans ce projet comprendra les fonctionnalités suivantes :  
  
Pages principales :  
• Page d’authentification :  
 - Inscription (nom, email, mot de passe)  
 - Connexion sécurisée

- Modification des informations   
 - Déconnexion (bouton Logout)  
 **- Gestion des sessions via JWT**  
  
• Dashboard (Tableau de bord) :  
 - Affichage d’un graphique (statique) montrant l’évolution globale du budget  
 - Vue d’ensemble : solde actuel, résumé revenus/dépenses  
  
• Page Revenus :

- Formulaire pour ajouter un revenu (source, montant, date)  
 - Liste des revenus  
 - Graphique en courbe statique pour visualiser les revenus

- Supprime des revenus

• Page Dépenses :

- Formulaire pour ajouter une dépense (catégorie, montant, date)  
 - Liste des dépenses  
 - Graphique en courbe statique pour visualiser les dépenses

- Supprime des dépenses  
  
Fonctionnalités principales :

• **Authentification utilisateur (JWT, mot de passe hashé avec Bcrypt)**  
• Ajouter ou supprimer un revenu ou une dépense  
• Afficher un graphique statique pour les revenus et dépenses  
• Navigation fluide entre les pages  
• Communication frontend-backend : Utilisation d’Axios pour interagir avec l’API  
• Stockage des données : Implémentation d’une base de données Postgres.

# 2. Architecture du système

## 2.1. Description globale

L'application de gestion de budget repose sur une architecture full-stack utilisant React.js pour le frontend et Node.js avec Express pour le backend. Elle permet aux utilisateurs d'ajouter, de supprimer et de consulter leurs budgets via une interface web interactive.  
  
L’architecture repose sur un modèle client-serveur :  
• Le client (frontend) est développé avec React.js et communique avec le serveur via des requêtes HTTP.  
• Le serveur (backend) est développé avec Node.js et Express. Il gère les requêtes des utilisateurs et interagit avec la base de données.  
• La base de données est gérée avec Postgres, permettant de stocker les informations des budgets et des utilisateurs.  
  
Le système suit une approche API RESTful, où chaque ressource (budgets, utilisateurs) est accessible via des endpoints spécifiques.

## 2.2. Architecture logicielle

L’architecture logicielle est organisée en trois couches principales :  
  
1. Frontend (React.js)  
• Interface utilisateur interactive.  
• Gestion des événements et état des composants avec React Hooks.  
• Communication avec le backend via Axios (requêtes HTTP).  
• Routing avec React Router pour la navigation entre les pages.  
• Bibliothèques : React Router, Axios, Chart.js ou Recharts pour les graphiques.  
  
2. Backend (Node.js + Express.js)  
• Serveur RESTful qui gère les requêtes HTTP.  
• Routes pour gérer les revenus (/api/income/), les dépenses (/api/expense/) et les utilisateurs (/api/auth/).  
• Middleware pour la gestion des erreurs et de la sécurité (CORS, validation des entrées).  
• Interaction avec la base de données Postgres via le package "pg".  
  
3. Base de données (PostgreSQL)  
• Stockage des revenus (source, amount, date, icone).  
• Stockage des dépenses (category, amount, date, icone).  
• Stockage des utilisateurs (email, mot de passe haché, fullname, profile\_pic).

## 2.3. Schéma d’architecture

Un schéma d’architecture illustrant l’interaction entre le frontend, le backend et la base de données peut être ajouté ici.



# 3. Base de données

Conception : diagramme de classe

## 3.1. Modèle logique (MLD)

Voici la structure logique des tables utilisées dans l’application :

* **Modèle : User**

|  |  |  |
| --- | --- | --- |
| Champ | Type | Description |
| id | UUID / SERIAL | Identifiant unique (clé primaire) |
| fullname | VARCHAR(100) | Nom complet de l'utilisateur |
| email | VARCHAR(255) | Email unique |
| password | VARCHAR(255) | Mot de passe (haché) |
| profile\_pic | TEXT | URL de la photo de profil |
| created\_at | TIMESTAMP | Date d’inscription |

* **Modèle : Income**

|  |  |  |
| --- | --- | --- |
| Champ | Type | Description |
| id | UUID / SERIAL | Identifiant unique (clé primaire) |
| user\_id | UUID / INT | Clé étrangère vers User |
| source | VARCHAR(255) | Source du revenu (ex: salaire, bonus…) |
| amount | DECIMAL(10, 2) | Montant du revenu |
| icon | VARCHAR(255) | Icon |
| date | DATE | Date du revenu |
| created\_at | TIMESTAMP | Date d’ajout de l’enregistrement |

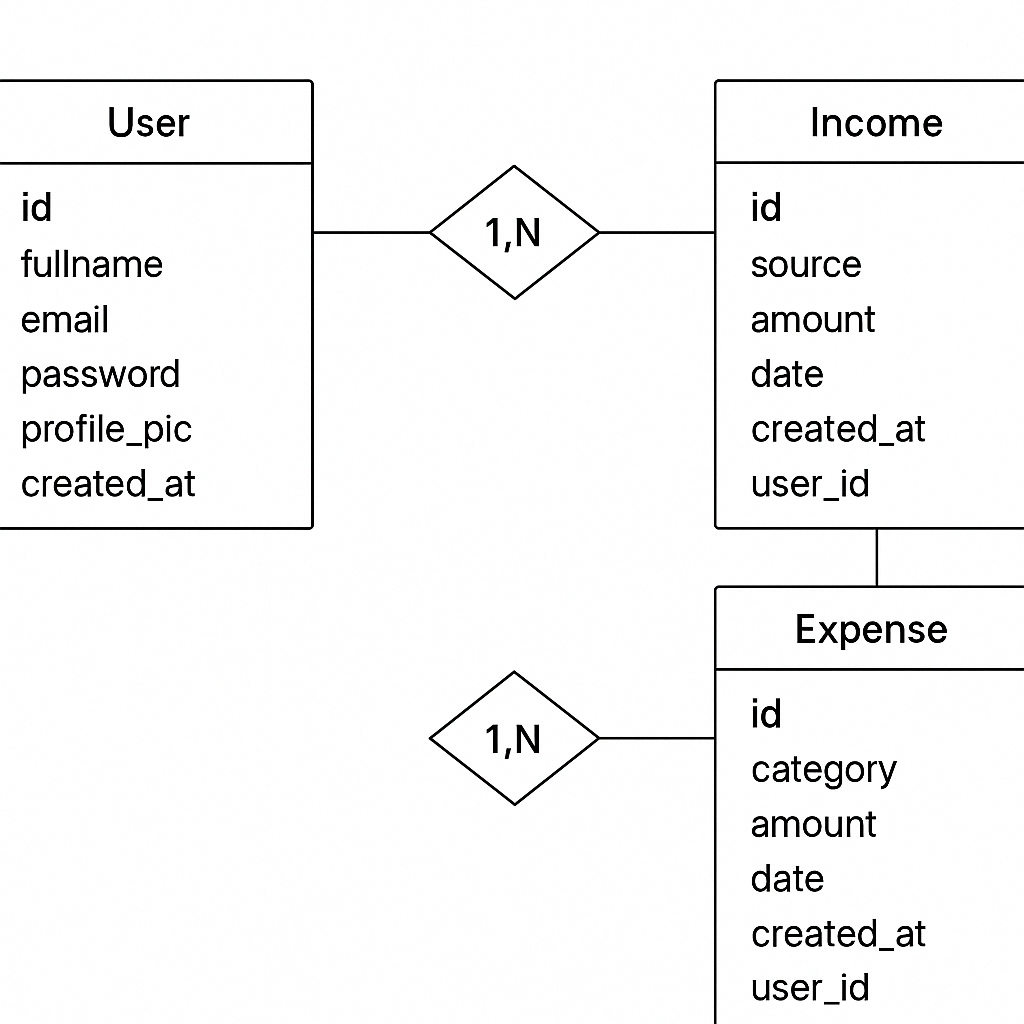
* **Modèle : Expense**

|  |  |  |
| --- | --- | --- |
| Champ | Type | Description |
| id | UUID / SERIAL | Identifiant unique (clé primaire) |
| user\_id | UUID / INT | Clé étrangère vers User |
| category | VARCHAR(255) | Catégorie de dépense (ex: nourriture…) |
| amount | DECIMAL(10, 2) | Montant de la dépense |
| icon | VARCHAR(255) | Icon |
| date | DATE | Date de la dépense |
| created\_at | TIMESTAMP | Date d’ajout l’enregistrement |

🔐 Relations :  
• Un utilisateur peut avoir plusieurs revenus et dépenses  
• user\_id est une clé étrangère dans les tables Income et Expense

## 3.2. Modèle conceptuel (MCD)

Voir le schéma MCD ci-dessous :



# 4. Technologies et outils

## 4.1. Technologies utilisées

🔹 Back-end : Node.js avec Express.js

• Express.js est un framework minimaliste et flexible pour Node.js, bien adapté pour construire une API RESTful.

• Il est largement adopté, dispose d’une grande communauté, et permet une mise en place rapide.

🔹 Front-end : React.js

• Utilisation de composants réutilisables pour une interface dynamique et interactive.

• Intégration de bibliothèques telles que React Router pour la navigation, Axios pour les appels API, et Chart.js ou Recharts pour l'affichage des graphiques.

🔹 Base de données : PostgreSQL

• Utilisation du système de gestion de base de données relationnelle PostgreSQL pour le stockage des utilisateurs, revenus, et dépenses.

• Connexion avec le backend via le module 'pg' de Node.js.

🔹 Interface utilisateur (UI/UX)

• Tailwind CSS : pour un design léger, moderne et facile à personnaliser.

• Material-UI (MUI) : pour bénéficier de composants visuellement attractifs et prêts à l’emploi.

## 4.2. Outils de développement

Voici les outils principaux utilisés pour le développement de l’application :

* Visual Studio Code : éditeur de code principal
* Postman : test des requêtes API
* Pg Admin : gestion de la base de données PostgreSQL
* Git et GitHub : gestion du versionnement du code
* Figma (ou autre outil de maquette) : conception UI/UX
* Nodemon : redémarrage automatique du serveur en développement
* ESLint & Prettier : pour le formatage et la qualité du code