📘 Rapport Technique

Application Budget Manager – Stack MERN

---

1. Choix et justification du projet

1.1 Contexte et problématique

La gestion financière personnelle représente un défi quotidien pour de nombreuses personnes. Beaucoup éprouvent des difficultés à :

- Suivre précisément leurs dépenses
- Respecter des budgets mensuels
- Comprendre la répartition réelle de leur argent

Les solutions existantes sont souvent **complexes**, **payantes**, ou peu adaptées aux besoins réels des utilisateurs débutants.

1.2 Solution proposée

Le projet **Budget Manager** propose une solution :

- **Simple et intuitive**
- **Gratuite**
- Accessible via une application web moderne

Elle permet à l'utilisateur de :

- Visualiser sa situation financière en temps réel
- Définir des budgets avec des périodes précises
- Catégoriser ses dépenses
- Suivre l'évolution de ses finances via des statistiques dynamiques

---

1.3 Pertinence pédagogique

Ce projet a une forte valeur pédagogique car il permet de démontrer la maîtrise de la **stack MERN complète**, notamment :

- Gestion de relations complexes en base de données (1–1, 1–N, N–N)
- Authentification sécurisée avec JWT

- Architecture MVC côté backend
- Conception d'une API RESTful sécurisée
- Développement d'une interface moderne et responsive avec React

---

2. Architecture générale de l'application

L'application adopte une architecture **client–serveur** avec séparation claire des responsabilités :

- **Frontend** : React (SPA)
- **Backend** : Node.js + Express (API REST)
- **Base de données** : MongoDB via Mongoose

---

3. Architecture Backend (Pattern MVC)

3.1 Modèles (Models) – Couche données

L'application repose sur cinq modèles Mongoose principaux :

1. User (Utilisateur)

- Champs : `name`, `email`, `password`
- Relation : **1-to-1** avec `Profile`
- Contient uniquement les données sensibles liées à l'authentification

2. Profile (Profil utilisateur)

- Relation **1-to-1** avec `User`
- Garantit qu'un profil appartient à un seul utilisateur (`unique: true`)
- Contient les informations non sensibles (bio, téléphone, etc.)

3. Budget

- Relation **1-to-Many** avec `User`
- Un utilisateur peut créer plusieurs budgets
- Champs principaux : `name`, `amount`, `startDate`, `endDate`

4. Expense (Dépense)

- Relations **1-to-Many multiples**
- Chaque dépense est liée à :
    - un utilisateur
    - un budget

    - une catégorie
- Permet une analyse financière détaillée

5. Category (Catégorie)

- Relation **Many-to-Many** avec `User`
- Une catégorie peut être partagée entre plusieurs utilisateurs
- Supporte les catégories personnelles et communes

3.2 Résumé des relations

Type de relation	Entités concernées	Description
1-to-1	User ↔ Profile	Un utilisateur a un seul profil
1-to-Many	User → Budgets	Un utilisateur possède plusieurs budgets
1-to-Many	User → Expenses	Un utilisateur a plusieurs dépenses
1-to-Many	Budget → Expenses	Un budget contient plusieurs dépenses
Many-to-Many	Users ↔ Categories	Catégories partagées ou privées

---

3.3 Contrôleurs (Controllers) – Logique métier

Chaque contrôleur gère les opérations CRUD d'une entité spécifique :

- `authController` : inscription, connexion, récupération du profil
- `budgetController` : gestion des budgets
- `expenseController` : gestion des dépenses
- `categoryController` : gestion des catégories
- `profileController` : gestion du profil utilisateur

Principe clé :
Toutes les requêtes sont filtrées par utilisateur afin de garantir que chaque utilisateur accède uniquement à ses propres données.

3.4 Routes et flux d'une requête

Chaque endpoint suit le flux suivant :

Client → Route → Validator → Middleware JWT → Controller → Model → MongoDB

Ce flux garantit :

- Validation des données
- Authentification sécurisée
- Séparation claire des responsabilités

3.5 Validation des données

La bibliothèque **express-validator** est utilisée pour :

- Vérifier le type des données
- Imposer des contraintes (montant > 0, dates valides, etc.)
- Empêcher les données malveillantes ou incorrectes

3.6 Middleware d'authentification

Le middleware JWT :

- Extrait le token depuis le header `Authorization`
- Vérifie sa validité
- Injecte l'utilisateur dans `req.user`
- Bloque l'accès si le token est invalide ou expiré

---

4. Architecture Frontend (React)

4.1 Organisation en couches

Pages (Vues)
↓
Services API (Axios)
↓
Context (État global)
↓
Composants réutilisables

4.2 Gestion de l'état global

Le **AuthContext** :

- Stocke l'utilisateur connecté

- Gère le token JWT
- Persiste la session via `localStorage`
- Redirige automatiquement en cas de déconnexion

4.3 Services API

Les appels HTTP sont centralisés dans des services dédiés :

- `budgetService`
- `expenseService`
- `categoryService`
- `profileService`

Avantages :

- Réutilisabilité
- Testabilité
- Maintenance facilitée

4.4 Routing

Le routage est géré avec **React Router v6** :

- Routes protégées
- Redirection automatique vers `/login` si non authentifié
- Navigation fluide sans rechargement de page

---

5. Fonctionnalités clés

5.1 Authentification sécurisée

- Hashage des mots de passe avec bcrypt
- JWT avec expiration (30 jours)
- Protection complète des routes sensibles

5.2 Gestion des budgets

- Création, modification, suppression
- Validation stricte des montants et dates
- Isolation par utilisateur

5.3 Suivi des dépenses

- Association obligatoire à un budget et une catégorie
- Tri par date
- Vérification d'appartenance du budget

5.4 Catégories personnalisées

- Catégories privées ou partagées
- Relation Many-to-Many flexible
- Réutilisation entre utilisateurs

5.5 Dashboard

- Statistiques en temps réel
- Calcul automatique :
    - Total budgets
    - Total dépenses
    - Montant restant

5.6 Profil utilisateur

- Relation 1-to-1 sécurisée
- Création automatique si inexistant
- Modification des informations personnelles

---

6. Sécurité de l'application

Mesures implémentées

- Hashage des mots de passe
- JWT dans les headers (pas de cookies)
- Validation backend + frontend
- CORS configuré
- Variables sensibles dans `.env`

Améliorations possibles

- Rate limiting
- Captcha après tentatives échouées
- HTTPS en production

---

7. Choix technologiques

Stack MERN

- **MongoDB** : flexibilité et performance
- **Express.js** : middleware et API REST
- **React** : composants réutilisables, SPA
- **Node.js** : asynchrone et performant

Librairies complémentaires

- bcrypt
- jsonwebtoken
- axios
- react-router-dom
- react-hot-toast
- lucide-react

---

8. Conclusion

Le projet **Budget Manager** démontre :

- Une maîtrise complète de la stack MERN
- Une architecture claire et scalable
- Une sécurité robuste
- Une interface moderne et responsive
- Un code structuré et maintenable

L'application est fonctionnelle et prête pour une mise en production avec des améliorations futures possibles telles que :

- Tests unitaires
- Graphiques avancés
- Export PDF
- Notifications intelligentes

Auteur : Sarah Ghabri
Date : Janvier 2025
Technologies : MongoDB, Express.js, React, Node.js