Rapport de Stage

Développement d'un logiciel du chiffrage

eco-pilot

Stagiaire : Mayar Briki
Encadrant académique : Nom Prénom
Encadrant professionnel : Nom Prénom
Date : October 22, 2025

Chapter 1

Introduction

Durant mon stage de deux mois, j'ai eu l'opportunité de découvrir et de participer à des projets liés à la gestion de projets et à la gestion de budgets à travers un logiciel de chiffrage. Ce stage m'a permis de comprendre l'importance de l'organisation, du suivi des coûts et de l'optimisation des ressources pour assurer le succès des projets, tout en développant mes compétences pratiques dans un environnement professionnel. **eco-pilote**.

Chapter 2

Présentation générale du projet

2.1 Contexte et objectifs

Ce projet a été réalisé dans le cadre d'un stage de deux mois.

2.2 Utilisateurs cibles

L'application s'adresse principalement aux professionnels impliqués dans la gestion de projets et le chiffrage budgétaire. Les utilisateurs cibles incluent :

- **Chefs de projet :** responsables de la planification, du suivi et de la gestion des ressources et des coûts des projets.
- **Responsables financiers :** chargés de la validation des budgets, du contrôle des dépenses et de l'optimisation des coûts.
- **Membres d'équipes techniques :** intervenant dans la saisie des besoins, la consultation des articles et la remontée d'informations sur l'avancement des tâches.
- **Administrateurs :** assurant la gestion des utilisateurs, des droits d'accès et la configuration générale de l'application.
- **Clients ou partenaires externes :** pouvant accéder à certaines informations ou suivre l'avancement des projets selon les droits qui leur sont attribués.

Cette diversité d'utilisateurs implique une gestion fine des rôles et des permissions afin de garantir la sécurité et la pertinence des informations accessibles à chacun.

2.3 Fonctionnalités principales

Les principales fonctionnalités du projet incluent :

- **Gestion des utilisateurs :** Création, modification, suppression et gestion des rôles des utilisateurs. Chaque utilisateur dispose d'un profil personnalisé avec des droits d'accès adaptés à sa fonction (administrateur, chef de projet, membre technique, etc.). Un système d'authentification sécurisé permet de garantir la confidentialité des données et la traçabilité des actions.

- **Gestion des articles :** Ajout, mise à jour, suppression et consultation des articles. Les articles représentent les ressources, matériaux ou prestations nécessaires à la réalisation des projets. Un catalogue centralisé facilite la recherche, la comparaison et la sélection des articles, tout en permettant la gestion des stocks et des coûts associés.

- **Gestion des projets :** Suivi, planification et gestion des différents projets. Cette fonctionnalité permet de créer de nouveaux projets, d'y associer des utilisateurs, des articles et des budgets, et de suivre l'avancement des tâches. Des outils de reporting et de visualisation (tableaux de bord, graphiques) aident à piloter efficacement les projets et à anticiper les dérives budgétaires ou organisationnelles.

Chapter 3

Cahier des charges

3.1 Besoins fonctionnels

Les besoins fonctionnels de l'application **eco-pilot** sont les suivants :

- Authentification et gestion sécurisée des utilisateurs (inscription, connexion, gestion des rôles).
- Gestion des projets : création, modification, suppression, consultation.
- Gestion des articles : ajout, modification, suppression, recherche et consultation.
- Gestion des budgets associés aux projets, suivi des coûts prévus et réels.
- Attribution des utilisateurs aux projets et gestion des droits d'accès.
- Visualisation de l'avancement des projets (tableaux de bord, graphiques).
- Historique des actions et traçabilité.
- Exportation de rapports (PDF, Excel).

3.2 Besoins techniques

Pour garantir le bon fonctionnement et la maintenabilité de l'application **eco-pilot**, plusieurs besoins techniques ont été identifiés et structurés autour des axes suivants :

- **Environnement de développement :**

- Ordinateur avec système d'exploitation moderne (Windows, macOS ou Linux) et ressources suffisantes (RAM, CPU) pour supporter les outils de développement.
- Installation de Node.js (version 16 ou supérieure) et npm pour la gestion des dépendances et l'exécution du backend.
- Installation de PostgreSQL (version 13 ou supérieure) pour la gestion locale de la base de données relationnelle.
- Navigateur web moderne (Chrome, Firefox, Edge) pour le développement et les tests du frontend.
- Outils de gestion de version (Git) pour le suivi collaboratif du code source.
- Éditeur de code performant (Visual Studio Code recommandé) avec extensions pour JavaScript/TypeScript, linting et gestion des environnements.

- **Architecture logicielle :**
  - Séparation stricte entre le frontend (React) et le backend (Node.js/Express) pour une meilleure maintenabilité et évolutivité.
  - Utilisation d'une API RESTful pour la communication entre les deux couches.
  - Base de données relationnelle PostgreSQL pour la gestion structurée et fiable des données.
  - Gestion des variables d'environnement via fichiers `.env` pour séparer les configurations locales, de test et de production.
  - Respect des bonnes pratiques de structuration des dossiers et du code (architecture modulaire, séparation des responsabilités).

- **Sécurité :**
  - Authentification sécurisée basée sur JWT (JSON Web Token) pour la gestion des sessions utilisateurs.
  - Hashage des mots de passe avec bcrypt ou équivalent pour garantir la confidentialité des données sensibles.
  - Mise en place de middlewares de sécurité (helmet, cors) pour protéger l'API contre les attaques courantes (XSS, CSRF, injections SQL).
  - Utilisation du protocole HTTPS en production pour chiffrer les échanges de données.

  - Gestion des droits d'accès et des rôles utilisateurs pour limiter l'accès aux ressources sensibles.

- **Déploiement et hébergement :**
  - Hébergement du frontend sur Vercel, permettant le déploiement continu, la gestion des builds et la distribution via CDN.
  - Hébergement du backend sur Render, avec gestion automatique du scaling, de la surveillance et des redémarrages.
  - Configuration des variables d'environnement et des secrets sur les plateformes cloud pour garantir la sécurité des accès.
  - Accès à une base de données PostgreSQL hébergée (Render ou équivalent) avec sauvegardes automatiques.
  - Possibilité de rollback rapide en cas de problème lors d'un déploiement.

- **Tests et qualité logicielle :**
  - Mise en place de tests unitaires et d'intégration (Jest, Supertest) pour garantir la fiabilité du code.
  - Utilisation d'outils d'analyse statique et de linters (ESLint, Prettier) pour assurer la qualité et la cohérence du code.
  - Documentation technique et API (Swagger ou équivalent) pour faciliter la maintenance et l'intégration de nouveaux développeurs.
  - Intégration continue (CI) pour automatiser les tests et les déploiements.

- **Scalabilité, performance et maintenance :**
  - Architecture modulaire et découplée pour faciliter l'ajout de nouvelles fonctionnalités ou la modification de modules existants.
  - Possibilité de scaling horizontal/vertical sur Render et Vercel pour s'adapter à la montée en charge.
  - Mise en place de logs et de monitoring pour détecter rapidement les anomalies et optimiser les performances.
  - Sauvegarde régulière de la base de données et plan de reprise en cas d'incident.

Ces besoins techniques assurent la robustesse, la sécurité, la performance et l'évolutivité de la solution développée, tout en facilitant la collaboration et la maintenance sur le long terme.

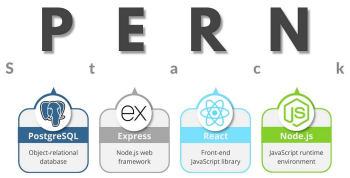
3.3 Contraintes

- Respect des délais de livraison du projet (2 mois).
- Respect des normes de sécurité et de confidentialité des données.
- Compatibilité multiplateforme (navigateurs récents, systèmes d'exploitation courants).
- Utilisation exclusive de technologies open-source.
- Facilité de prise en main et d'utilisation pour les utilisateurs finaux.
- Documentation technique et utilisateur à fournir.

Chapter 4

Étude et choix technologiques

Présentation du choix de la stack **PERN (PostgreSQL, Express, React, Node.js)** et justification.



Présentation détaillée de la stack PERN

La stack **PERN** est composée de quatre technologies principales, chacune jouant un rôle spécifique dans l'architecture de l'application :

- **PostgreSQL :** Système de gestion de base de données relationnelle open-source reconnu pour sa robustesse, sa conformité aux standards SQL et ses fonctionnalités avancées (transactions, gestion des relations complexes, extensibilité). Il permet de garantir l'intégrité et la sécurité des données, tout en offrant de bonnes performances pour des applications à grande échelle.

- **Express.js :** Framework minimaliste pour Node.js, Express simplifie la création d'API RESTful et la gestion des routes, des middlewares et des requêtes HTTP. Il favorise une architecture modulaire et maintenable, tout en restant léger et performant.

- **React :** Bibliothèque JavaScript développée par Facebook pour la création d'interfaces utilisateur dynamiques et réactives. React permet

de construire des composants réutilisables, d'optimiser le rendu grâce au Virtual DOM et de gérer efficacement l'état de l'application côté client.

- **Node.js :** Environnement d'exécution JavaScript côté serveur, Node.js se distingue par son modèle événementiel non bloquant, idéal pour les applications nécessitant une forte scalabilité et un traitement asynchrone des requêtes.

Justification du choix de la stack PERN

La stack **PERN** (PostgreSQL, Express, React, Node.js) a été choisie pour plusieurs raisons :

- **JavaScript full-stack :** Avec Node.js et React, le même langage est utilisé côté serveur et côté client, ce qui simplifie le développement et la maintenance.
- **Base de données relationnelle robuste :** PostgreSQL offre une gestion avancée des données, des relations complexes et une grande fiabilité pour les applications critiques.
- **Performance et scalabilité :** Node.js, grâce à son modèle non-bloquant et orienté événement, permet de gérer efficacement un grand nombre de requêtes simultanées.
- **Flexibilité et modularité :** Express.js facilite la création d'API RESTful flexibles, tandis que React permet de construire une interface utilisateur dynamique et réactive.
- **Écosystème riche :** Chaque composant de la stack dispose d'une communauté active et de nombreux packages, accélérant le développement et l'intégration de fonctionnalités avancées.

Comparaison avec d'autres stacks populaires

Il existe d'autres stacks populaires telles que **MERN** (MongoDB, Express, React, Node.js) et **LAMP** (Linux, Apache, MySQL, PHP). Le choix de PERN s'explique par les avantages suivants :

- **Par rapport à MERN :** PostgreSQL offre des fonctionnalités relationnelles avancées et une meilleure gestion des transactions que MongoDB, ce qui est crucial pour des applications nécessitant une forte cohérence des données.
- **Par rapport à LAMP :** La stack PERN permet un développement full JavaScript, réduisant la courbe d'apprentissage et facilitant la communication entre les différentes couches de l'application. De plus, Node.js offre de meilleures performances pour les applications temps réel.
- **Évolutivité et modernité :** La stack PERN est particulièrement adaptée aux applications web modernes nécessitant une interface utilisateur réactive et une API performante.

En résumé, la stack PERN combine la puissance d'une base de données relationnelle, la flexibilité du JavaScript full-stack et la modernité des outils de développement web actuels, ce qui en fait un choix pertinent pour le projet **eco-pilot**.

Chapter 5

Conception du projet

5.1 Architecture de l'application

Inclure un schéma de l'architecture PERN.

5.2 Diagrammes UML

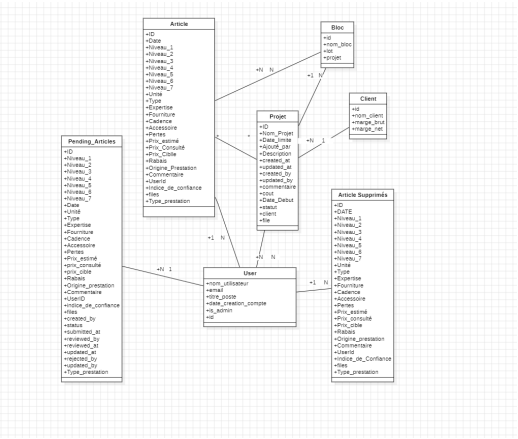


Figure 5.1: Diagramme UML principal de l'application

Description du diagramme :

Le diagramme UML ci-dessus illustre l'architecture principale de l'application **eco-pilot**. On y distingue les entités essentielles et leurs relations :

- **Utilisateur (User) :** Représente les différents profils accédant à l'application (administrateur, chef de projet, membre technique, etc.). Chaque util-

isateur possède des attributs comme l'identifiant, le nom, l'email, le mot de passe (hashé) et un rôle.

- **Projet (Project) :** Entité centrale du système, chaque projet est associé à un ou plusieurs utilisateurs (gestion de l'équipe projet), à un budget, à des articles et à un état d'avancement.
- **Article :** Représente les ressources, matériaux ou prestations nécessaires à la réalisation des projets. Chaque article possède des informations telles que le nom, la description, le coût unitaire et le stock disponible.
- **Budget :** Associé à chaque projet, il permet de suivre les coûts prévus et réels, ainsi que les écarts éventuels.
- **Relations :**
  - Un utilisateur peut participer à plusieurs projets, et un projet peut avoir plusieurs utilisateurs (relation plusieurs-à-plusieurs).
  - Un projet peut contenir plusieurs articles, chaque article pouvant être utilisé dans plusieurs projets (relation plusieurs-à-plusieurs).
  - Chaque projet dispose d'un budget unique, mais le modèle peut être étendu pour gérer des budgets intermédiaires ou par lot.

Ce diagramme met en évidence la structure relationnelle de l'application et sert de base à la conception de la base de données et des API.

5.3 Modèle de la base de données

Le modèle de la base de données repose sur une structure relationnelle adaptée à la gestion de projets, d'utilisateurs, d'articles et de budgets. Voici un aperçu des principales tables et de leurs relations :

- **users** (`id`, `name`, `email`, `password_hash`, `role`)
- **projects** (`id`, `name`, `description`, `status`, `budget_id`)
- **articles** (`id`, `name`, `description`, `unit_cost`, `stock`)
- **budgets** (`id`, `project_id`, `planned_cost`, `actual_cost`)
- **project_users** (`project_id`, `user_id`)
- **project_articles** (`project_id`, `article_id`, `quantity`)

Les relations principales sont :

- Un utilisateur peut participer à plusieurs projets (relation plusieurs-à-plusieurs via `project_users`).
- Un projet peut contenir plusieurs articles (relation plusieurs-à-plusieurs via `project_articles`).
- Chaque projet dispose d'un budget unique.

Chapter 6

Implémentation

6.1 Backend (Node.js + Express)

Le backend est développé avec Node.js et Express. Il expose une API RESTful permettant la gestion des utilisateurs, des projets, des articles et des budgets. Les principales fonctionnalités implémentées sont :

- Authentification JWT et gestion des rôles.
- Endpoints CRUD pour les entités principales (utilisateurs, projets, articles, budgets).
- Middleware de sécurité (helmet, cors).
- Validation des données avec Joi.
- Gestion des erreurs centralisée.
- Documentation de l'API avec Swagger.

6.2 Frontend (React)

Le frontend est réalisé avec React. Il propose une interface utilisateur moderne et réactive :

- Authentification et gestion de session.
- Tableaux de bord pour la visualisation des projets, budgets et articles.
- Formulaires dynamiques pour la création et la modification des entités.
- Visualisation graphique de l'avancement des projets (charts).

- Gestion des notifications et des erreurs.
- Responsive design pour une utilisation sur desktop et mobile.

6.3 Base de données (PostgreSQL)

La base de données PostgreSQL stocke toutes les informations relatives aux utilisateurs, projets, articles et budgets. Les scripts de migration permettent de créer et de mettre à jour le schéma de la base. Des index sont ajoutés pour optimiser les requêtes fréquentes.

6.4 Sécurité et authentification (JWT, hashage)

- Authentification basée sur JWT pour sécuriser les endpoints.
- Hashage des mots de passe avec bcrypt.
- Vérification des rôles et des permissions pour chaque action sensible.
- Protection contre les attaques courantes (XSS, CSRF, injections SQL).
- Utilisation de HTTPS en production.

6.5 Tests et validation

- Tests unitaires sur les fonctions critiques du backend (Jest).
- Tests d'intégration des endpoints API (Supertest).
- Validation manuelle de l'interface utilisateur.
- Vérification de la conformité aux besoins fonctionnels.

Chapter 7

Déploiement

7.1 Environnement de développement

Le développement s'est effectué localement sur des machines équipées de Node.js, npm et PostgreSQL. Le code source est versionné avec Git et organisé en deux répertoires principaux : `client` (React) et `server` (Node.js/Express). Les variables d'environnement sont gérées via des fichiers `.env` pour séparer les configurations locales et de production.

7.2 Environnement de production

Pour le déploiement de l'application, deux plateformes cloud distinctes ont été utilisées afin de séparer le frontend et le backend :

- **Frontend (React) :** Déployé sur **Vercel**, une plateforme spécialisée dans l'hébergement d'applications front-end modernes. Vercel offre un déploiement continu à partir du dépôt GitHub, une gestion automatique des builds, un CDN performant et des URLs de prévisualisation pour chaque branche ou pull request. Cela permet de garantir une mise en ligne rapide et fiable de l'interface utilisateur.
- **Backend (Node.js/Express) :** Hébergé sur **Render**, un service cloud qui facilite le déploiement d'API Node.js. Render gère automatiquement le scaling, la surveillance, les redémarrages en cas de panne et l'intégration continue. La connexion à la base de données PostgreSQL est également configurée sur Render, assurant la sécurité et la disponibilité des données.

La communication entre le frontend (Vercel) et le backend (Render) se fait via des appels API sécurisés (HTTPS). Les variables d'environnement de production sont configurées directement sur les tableaux de bord Vercel et Render pour garantir la confidentialité des clés et des accès.

7.3 Gestion de versions (Git/GitHub)

Le projet est versionné avec Git et hébergé sur GitHub. Les branches principales sont :

- `main` : version stable et déployée.
- `dev` : développement des nouvelles fonctionnalités.
- branches de fonctionnalités : pour chaque nouvelle fonctionnalité ou correction de bug.

Des pull requests sont utilisées pour la revue de code et l'intégration continue.

Chapter 8

Résultats obtenus

L'application **eco-pilot** a permis de :

- Centraliser la gestion des projets, des articles et des budgets.
- Améliorer la traçabilité et la transparence des coûts.
- Offrir une interface intuitive pour les différents profils d'utilisateurs.
- Réduire les erreurs de saisie et accélérer la prise de décision grâce aux tableaux de bord.

Chapter 9

Difficultés rencontrées et solutions

- **Problème de synchronisation entre frontend et backend :** Résolu par la mise en place d'une documentation Swagger et de tests d'intégration.
- **Gestion des rôles complexe :** Utilisation d'un middleware de vérification des permissions pour chaque endpoint.
- **Déploiement sur Render et Vercel :** Adaptation des variables d'environnement et gestion des CORS.
- **Optimisation des requêtes SQL :** Ajout d'index et refactoring des requêtes lentes.
- **Formation des utilisateurs :** Rédaction d'un guide utilisateur et organisation de sessions de démonstration.

Chapter 10

Perspectives d'amélioration

- Ajout d'un module de notifications en temps réel (WebSocket).
- Intégration d'un système de gestion documentaire (upload de fichiers).
- Extension du modèle de budget pour gérer des sous-budgets par lot ou phase.
- Mise en place d'un système de commentaires et de suivi des tâches.
- Internationalisation de l'interface (multilingue).
- Automatisation des tests end-to-end (Cypress).

Chapter 11

Conclusion

Ce stage m'a permis de mettre en pratique mes compétences en développement web full-stack, de découvrir les enjeux de la gestion de projets et d'approfondir mes connaissances sur la stack PERN. J'ai pu contribuer à la conception, au développement et au déploiement d'une application complète, tout en apprenant à travailler en équipe et à respecter des contraintes professionnelles. Cette expérience a renforcé mon autonomie, ma rigueur et ma capacité à résoudre des problèmes techniques complexes.

Appendix A

Annexes

A.1 Extraits de code

```
// Exemple d'endpoint Express pour la création d'un utilisateur
app.post('/api/users', async (req, res) => {
    // Validation des données
    // Hashage du mot de passe
    // Insertion dans la base de données
});
```

A.2 Documentation API

La documentation complète de l'API est disponible via Swagger à l'adresse : `https://eco-pilot-api-docs.example.com`

A.3 Diagrammes complets

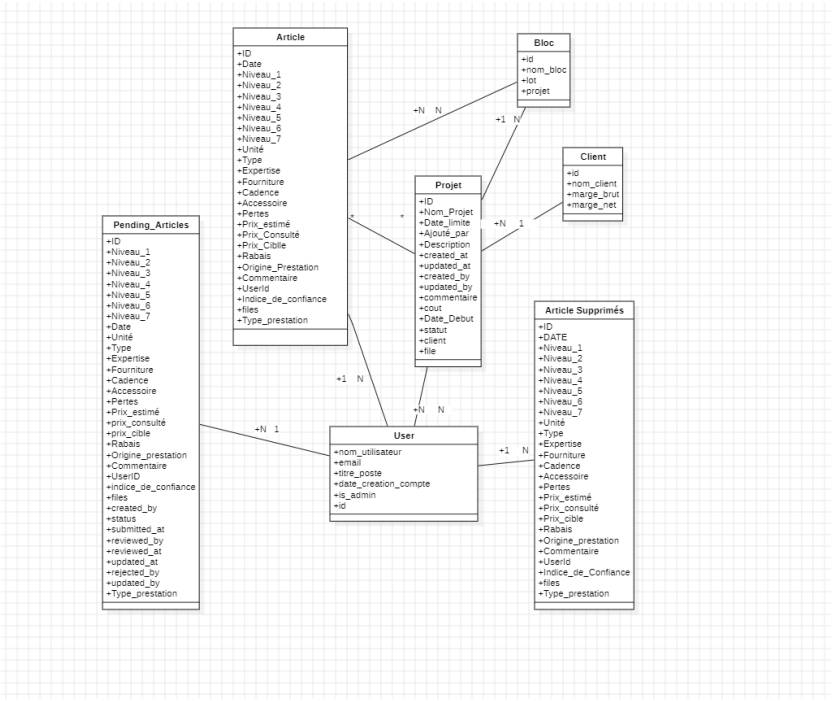


Figure A.1: Diagramme UML complet de l'application

Bibliography

[1] Documentation officielle MERN : https://www.mongodb.com/mern-stack

[2] Documentation React : https://react.dev

[3] Documentation Node.js : https://nodejs.org

[4] Documentation Express : https://expressjs.com

[5] Documentation PostgreSQL : https://www.postgresql.org/docs/

[6] JWT Introduction : https://jwt.io/introduction