Passion Lecture

Sottile, Teixeira, Bamert – GRP2D
Lausanne - Vennes
24 périodes
Grégory Charmier

Table des matières

1 SPÉCIFICATIONS

1.1 Titre

Passion Lecture P_web295

1.2 Description

Le projet Passion Lecture est un projet qui se réalise par groupe de 3 dans le cadre du module 295. Il vise à concevoir et développer le backend d'une application web permettant aux utilisateurs de partager leur passion pour la lecture. L'application offrira une API REST complète qui servira de base pour le développement d'un frontend futur. GitHub et Github projects doivent être utilisés pour assurer la mise à jour continue du projet et optimiser la collaboration entre les membres de l'équipe.

2 ANALYSE

2.1 GitHub Project

Vous trouverez ici notre GitHub Project

2.2 Tableau des routes

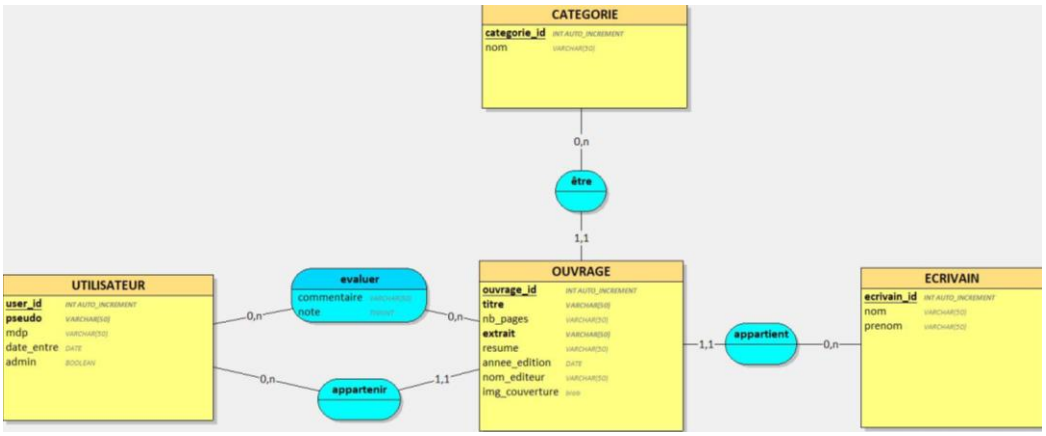
Verbe HTTP	Route	JSON envoyé (Body)	Description
GET	/api/books	Aucun	Récupère la liste de tous les livres selon des paramètres (titre ou/et nom de l'éditeur).
GET	/api/books/:id	Aucun	Récupère les détails d'un livre spécifique.
POST	/api/books	`{ "titre": "titre", "nb_pages": 420, "extrait": "extrait.pdf", "categorie_fk": 1, "resume":"resume", "annee_edition": "2020-07-15", "nom_editeur":"Bragelonne", "user_fk": 10, "ecrivain_fk": 17 }`	Ajoute un nouveau livre à la collection (requiert une authentification).
PUT	/api/books/:id	`{ "titre": "Nouveau titre", "auteur": "Nouvel auteur" }`	Modifie un livre existant.
DELETE	/api/books/:id	Aucun	Supprime un livre.
GET	/api/categories	Aucun	Récupère la liste des catégories de livres.
GET	/api/categories/ :id	Aucun	Récupère une catégorie selon l'id
GET	/api/categories/ :id/books	Aucun	Récupère tous les livres d'une catégorie
GET	/api/books/:id/notes	Aucun	Récupère la note du livre
POST	/api/login	`{ "pseudo": "pseudo", "password": "motdepasse" }`	Connecte un utilisateur et génère un jeton JWT.
GET	/api/authors	Aucun	Récupère la liste des auteurs de livres.
GET	/api/authors/:id	Aucun	Récupère un auteur selon l'id
GET	/api/authors/:id/books	Aucun	Récupère la liste des livres d'un auteur spécifique.
POST	/api/books/:id/evaluations	`{ "commentaire": "Super livre!", "note": 5 }`	Ajoute un commentaire et une note à un livre (utilisateur connecté requis).
GET	/api/books/:id/evaluations	Aucun	Récupère toutes les évalutations d'un livre.

DELETE	/api/comments/:id	Aucun	Supprime un commentaire (uniquement si l'utilisateur en est l'auteur ou admin).

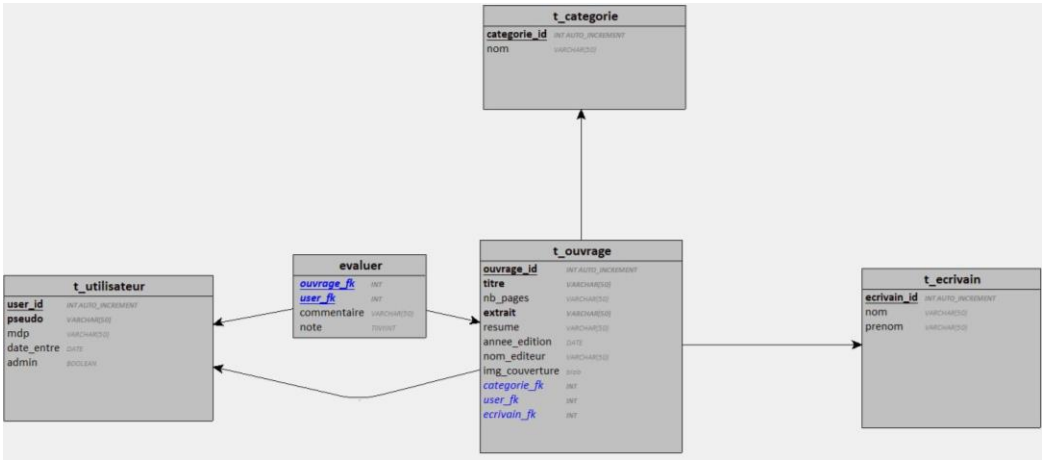
Les catégories et les auteurs n'ont pas besoin de routes pour la création, la modification ou la suppression, car ces actions sont réservées aux administrateurs, qui les effectueront directement dans la base de données si nécessaire.

2.3 Base de données

2.3.1 MCD



2.3.2 MLD

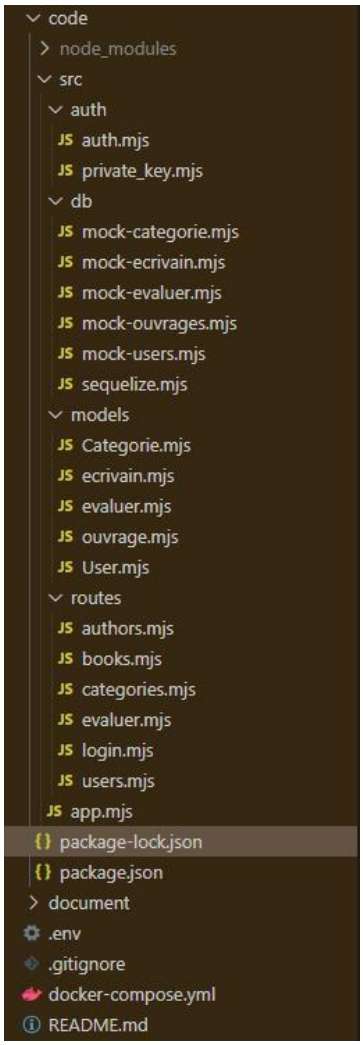


Nous avons choisi de créer une table « **Évaluer »**, regroupant à la fois le commentaire et la note, plutôt que de les séparer en deux tables distinctes. Cette approche est plus logique, car un utilisateur soumet simultanément une note et un commentaire, ce qui simplifie la gestion des données et optimise la structure de la base.

La table « **t_categorie** » ne contient qu'un seul champ, ce qui pourrait suggérer qu'il serait plus simple d'ajouter directement un champ « **categorie** » dans la table « **t_ouvrage** ». Cependant, afin de faciliter la gestion des catégories, il est plus pertinent de les stocker

dans une table distincte. Cette approche permet d'éviter les redondances, d'assurer une meilleure organisation des données et de simplifier d'éventuelles modifications ou ajouts de nouvelles catégories.

2.4 Structure du code



auth/ : Contient les fichiers liés à l'authentification, notamment :
auth.mjs : Gère les fonctionnalités d'authentification des utilisateurs avec les Jeton Web Token.
private_key.mjs : Stocke une clé privée pour des opérations sécurisées.

db/ : Regroupe les fichiers liés à la base de données.
Chaque mock contient des valeurs fictives pour les insérer dans la base de données.

models/ : Définit les tables de la base de données avec les spécifications des champs dans chaque table.

Routes/ : Contient les routes de l'API
Chacun des fichiers de routes/ contienne la gestion des routes spécifiques à une ressource.

Fichier principal *app.mjs* est le point d'entrée de l'application, qui initialise les routes et la connexion à la base de données.

package.json : Liste les dépendances et scripts du projet.

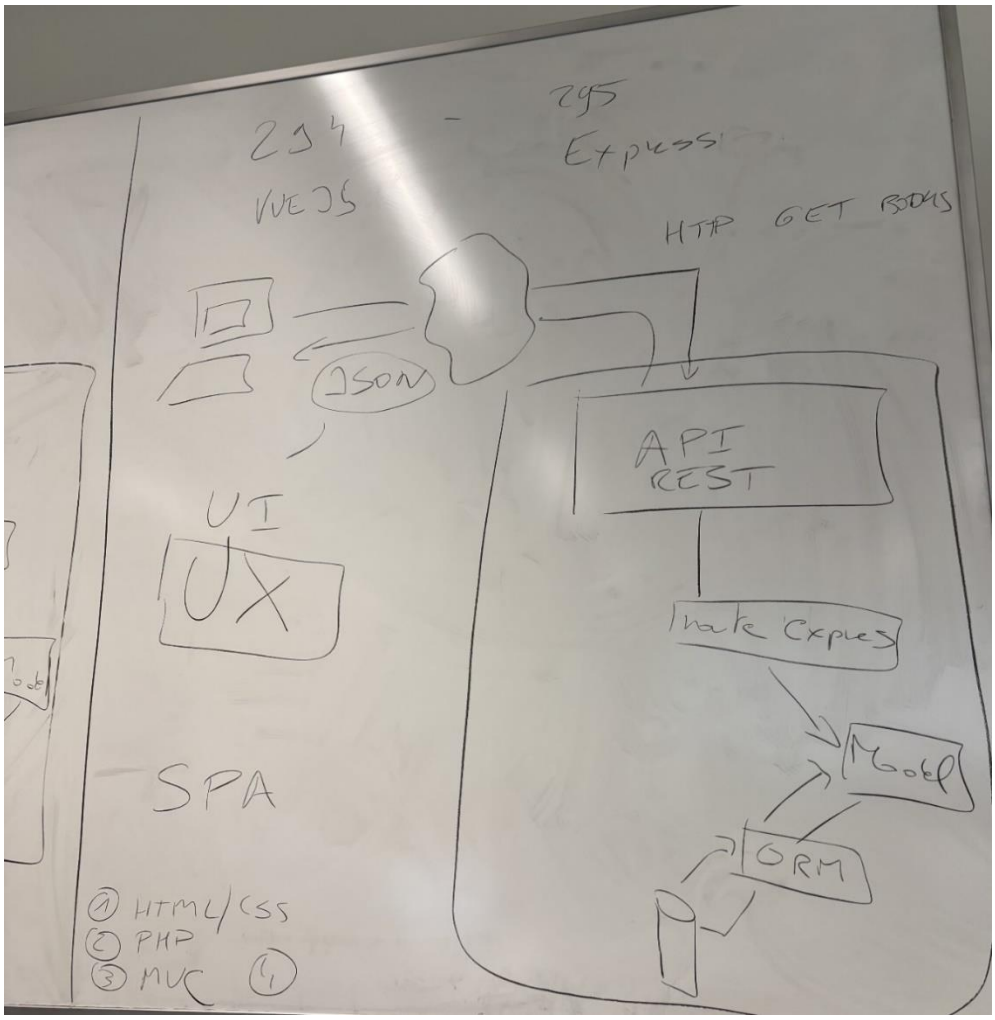
.env : Contient le nom de notre container docker

.gitignore : Indique les fichiers et dossiers à exclure du suivi Git.

docker-compose.yml : Indique la présence d'une configuration Docker pour le déploiement.

README.md : Contient le nom de notre projet.

2.5 Schéma entre Backend et Frontend



3 RÉALISATION

3.1 Figma

Vous trouverez ici notre Figma

3.2 Authentification

Login.mjs :

```
import express from "express";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import { User } from "../db/sequelize.mjs";
import { privateKey } from "../auth/private_key.mjs";

const loginRouter = express();

loginRouter.post("/", (req, res) => {
  User.findOne({ where: { pseudo: req.body.pseudo } })
    .then((user) => {
      if (!user) {
        const message = `L'utilisateur demandé n'existe pas`;
        return res.status(404).json({ message });
      }

      bcrypt.compare(req.body.mdp, user.mdp).then((isPasswordValid) => {
        console.log(req.body.mdp);
        console.log(user.mdp);
        if (!isPasswordValid) {
          const message = `Le mot de passe est incorrecte.`;
          return res.status(401).json({ message });
        } else {
          // JWT
          const token = jwt.sign(
            { userId: user.id, admin: user.admin },
            privateKey,
            {
              expiresIn: "1y",
            }
          );
          const message = `L'utilisateur a été connecté avec succès`;
          return res.json({ message, data: user, token });
        }
      });
    })
    .catch((error) => {
      const message = `L'utilisateur n'a pas pu être connecté. Réessayez dans quelques instants`;
      return res.json({ message, data: error });
    });
});

export { loginRouter };
```

L'algorithme mis en place pour gérer l'authentification repose sur **Express.js**, **bcrypt** pour le hachage des mots de passe et **jsonwebtoken** (JWT) pour la gestion des sessions. Voici son fonctionnement détaillé :

Recherche de l'utilisateur dans la base de données :

- Lorsqu'un utilisateur tente de se connecter, son **pseudo** est recherché dans la table des utilisateurs grâce à `User.findOne({ where: { pseudo: req.body.pseudo } })`.
- Si aucun utilisateur correspondant n'est trouvé, une réponse 404 (Not Found) est renvoyée.

Vérification du mot de passe :

- Si l'utilisateur **existe**, son mot de passe haché est comparé avec celui fourni dans la requête via `bcrypt.compare(req.body.mdp, user.mdp)`.
- Si les mots de passe ne correspondent pas, une réponse 401 (Unauthorized) est envoyée avec un message d'erreur.

Génération du jeton JWT :

- Si l'authentification réussit, un jeton JWT (JSON Web Token) est généré à l'aide de `jwt.sign()`.
- Ce jeton inclut les informations essentielles de l'utilisateur (ID et rôle admin) et est signé avec une clé privée (`privateKey`).
- Le jeton est configuré pour expirer après 1 an (`expiresIn: "1y"`), garantissant une session longue durée.

Réponse au client :

- Une réponse JSON est renvoyée contenant :
- Un message de succès
- Les données de l'utilisateur
- Le token JWT qui pourra être utilisé pour authentifier les futures requêtes

Gestion des erreurs :

- Si une erreur survient durant le processus, une réponse appropriée est envoyée au client avec un message d'erreur générique.

3.3 Aspect sécurité

Hachage des mots de passe :

Pour protéger les données sensibles des utilisateurs, les mots de passe ne sont jamais stockés en clair dans la base de données. À la place, ils sont transformés en une version hachée grâce à un algorithme de cryptographie (bcrypt). Ainsi, même en cas de fuite de données, les mots de passe restent illisibles et donc inutilisables. Lorsqu'un utilisateur tente de se connecter, le mot de passe qu'il saisit est comparé à celui stocké dans la base de données en utilisant la méthode **bcrypt.compare**.

Utilisation des JSON Web Tokens (JWT) :

Une fois la connexion où l'enregistrement validée, un jeton JWT est généré pour l'utilisateur. Ce jeton contient des informations essentielles, comme

son identifiant unique et son rôle (utilisateur ou administrateur). Il est signé avec une clé privée, ce qui empêche toute modification ou falsification. Le jeton a une durée de validité d'un an (expiresIn: '1y'), ce qui permet de maintenir une session sécurisée sans nécessiter de connexion fréquente. Contrairement aux sessions traditionnelles, où les informations de connexion sont stockées côté serveur, le JWT est envoyé à chaque requête et permet d'authentifier l'utilisateur sans avoir à conserver de données sensibles sur le serveur.

Gestion des erreurs et prévention des attaques :
Pour éviter de donner trop d'informations aux éventuels attaquants, le système d'authentification a été conçu pour renvoyer des messages d'erreur génériques. Par exemple, si un utilisateur entre un pseudo inexistant ou un mot de passe erroné, le message d'erreur ne précise pas lequel des deux est incorrect. Ce qui empêche les attaques par énumération de comptes, où un attaquant pourrait tester différents pseudos pour voir lesquels existent réellement.

3.4 Fonctionnalité

Routes :
L'API REST comprend un ensemble de routes pour gérer différentes entités comme les ouvrages, les utilisateurs, les catégories, les auteurs et les évaluations des livres.

```
import { loginRouter } from "./routes/login.mjs";
app.use("/api/login", loginRouter);

import { booksRouter } from "./routes/books.mjs";
app.use("/api/books", auth, booksRouter);

import { userRouter } from "./routes/users.mjs";
app.use("/api/users", auth, userRouter);

import { categoriesRouter } from "./routes/categories.mjs";
app.use("/api/categories", auth, categoriesRouter);

import { authorsRouter } from "./routes/authors.mjs";
app.use("/api/authors", auth, authorsRouter);

import { evaluerRouter } from "./routes/evaluer.mjs";
app.use("/api/evaluations", auth, evaluerRouter);

import { sequelize, initDb } from "./db/sequelize.mjs";
sequelize
  .authenticate()
  .then((_) =>
    console.log("La connexion à la base de données a bien été établie")
  )
  .catch((error) => console.error("Impossible de se connecter à la DB"));
initDb();

app.use(({ res }) => {
  const message =
    "Impossible de trouver la ressource demandée ! Vous pouvez essayer une autre URL.";
  res.status(404).json(message);
});
```

Validation des données :

Toutes les données fournies par le consommateur de l'API sont validées. Par exemple, dans le modèle de produit défini dans models/products.mjs, les validations sont mises en place pour les champs name et price :

```
const UserModel = (sequelize, DataTypes) => {
  return sequelize.define("t_utilisateur", {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
    },
    pseudo: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: { msg: "Ce username est déjà pris." },
      validate: {
        notEmpty: {
          msg: "Le pseudo ne peut pas être vide.",
        },
        notNull: {
          msg: "Le pseudo est une propriété obligatoire.",
        },
      },
    },
    mdp: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        notEmpty: {
          msg: "Le mot de passe ne peut pas être vide.",
        },
        notNull: {
          msg: "Le mot de passe est une propriété obligatoire.",
        },
      },
    },
    date_entre: {
      type: DataTypes.DATEONLY,
      allowNull: true,
      validate: {
        isDate: {
          msg: "La date d'entrée doit être une date valide.",
        },
      },
    },
    admin: {
      type: DataTypes.BOOLEAN,
      allowNull: false,
      defaultValue: false,
    },
  });
};

export { UserModel };
```

Gestion des statuts HTTP et des erreurs :
L'API gère les statuts HTTP appropriés (200, 3xx, 4xx, 5xx) et les erreurs.

```
.catch((error) =>
  res.status(404).json({
    message: `L'ouvrage dont l'id vaut ${ouvragesId} n'existe pas`,
    error: error,
  })
);
```

Recherche :
L'API permet la recherche sur les entités comme les livres.

```
booksRouter.get("/", (req, res) => {
  let whereClause = {};

  if (req.query.titre) {
    if (req.query.titre.length > 2) {
      whereClause.titre = { [Op.like]: `%${req.query.titre}%` };
    } else {
      return res
        .status(400)
        .json(
          "Vous devez entrer au moins deux caractères pour filtrer sur le nom"
        );
    }
  }
  if (req.query.categorie) {
    whereClause.categorie_fk = { [Op.like]: `%${req.query.categorie}%` };
  }
  if (req.query.nom_editeur) {
    whereClause.nom_editeur = { [Op.like]: `%${req.query.nom_editeur}%` };
  }
  Ouvrage.findAll({ where: whereClause }).then((ouvrages) => {
    if (ouvrages.length === 0) {
      res.status(404).json({ error: "Aucun ouvrages n'a été trouvé" });
    } else {
      const books = Promise.all(
        ouvrages.map(async (ouvrage) => {
          const cat = await Categorie.findByPk(ouvrage.categorie_fk);
          const ecri = await Ecrivain.findByPk(ouvrage.ecrivain_fk);
          return {
            ouvrage: ouvrage,
            categorie: cat.nom,
            écrivain: ecri.nom + " " + ecri.prenom,
          };
        })
      );
      books
        .then((result) => {
          res.status(200).json({
            message: "La liste des ouvrages a été trouvée",
            data: result,
          });
        })
        .catch((error) => {
          res.status(500).json({
            message: "Erreur lors de la récupération des ouvrages",
            error: error.message,
          });
        });
    }
  });
});
```

Authentification JWT :
L'API utilise un système d'authentification basé sur les jetons JWT.

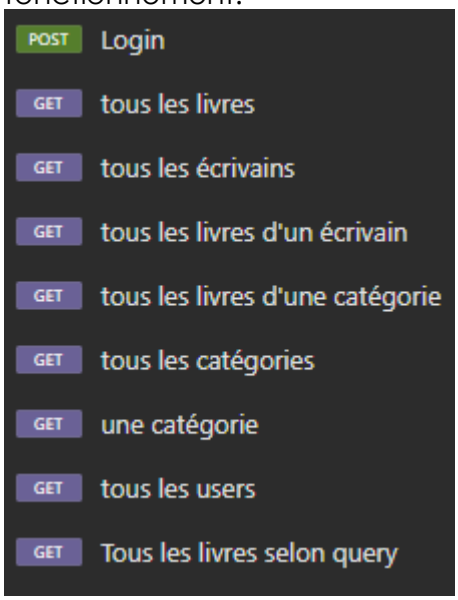
```
import jwt from "jsonwebtoken";
import { privateKey } from "./private_key.mjs";
const auth = (req, res, next) => {
  const authorizationHeader = req.headers.authorization;
  if (!authorizationHeader) {
    const message = `Vous n'avez pas fourni de jeton d'authentification. Ajoutez-en un dans l'en-tête de la requête.`;
    return res.status(401).json({ message });
  } else {
    const token = authorizationHeader.split(" ")[1];
    const decodedToken = jwt.verify(
      token,
      privateKey,
      (error, decodedToken) => {
        if (error) {
          const message = `L'utilisateur n'est pas autorisé à accéder à cette ressource.`;
          return res.status(401).json({ message, data: error });
        }
        const userId = decodedToken.userId;
        if (req.body.userId && req.body.userId !== userId) {
          const message = `L'identifiant de l'utisateur est invalide`;
          return res.status(401).json({ message });
        } else {
          next();
        }
      }
    );
  }
};

export { auth };
```

Documentation Swagger :

L'API est documentée avec Swagger. La documentation Swagger est définie dans swagger.mjs et les routes des ouvrages sont annotées avec des commentaires Swagger :

```
import swaggerJSDoc from "swagger-jsdoc";
const options = {
  definition: {
    openapi: "3.0.0",
    info: {
      title: "API self-service-machine",
      version: "1.0.0",
      description:
        "API REST permettant de gérer l'application self-service-machine",
    },
    servers: [
      {
        url: "http://localhost:3000",
      },
    ],
    components: {
      securitySchemes: {
        bearerAuth: {
          type: "http",
          scheme: "bearer",
          bearerFormat: "JWT",
        },
      },
      schemas: {
        Teacher: {
          type: "object",
          required: ["name", "price", "created"],
          properties: {
            id: {
              type: "integer",
              description: "L'identifiant unique du produit.",
            },
            name: {
              type: "string",
              description: "Le nom du produit.",
            },
            price: {
              type: "float",
              description: "Le prix du produit.",
            },
            created: {
              type: "string",
              format: "date-time",
              description: "La date et l'heure de l'ajout d'un produit.",
            },
          },
        },
        // Ajoutez d'autres schémas ici si nécessaire
      },
    },
    security: [
      {
        bearerAuth: [],
      },
    ],
  },
  apis: ["./src/routes/*.mjs"], // Chemins vers vos fichiers de route
};
const swaggerSpec = swaggerJSDoc(options);
```

Tests avec Insomnia ou Postman :
L'API a été testée manuellement avec Insomnia pour vérifier son bon fonctionnement.



Tests automatisés avec Vitest, intégration continue avec GitHub Actions et dockerisation du backend :
Étant donné que ces fonctionnalités étaient optionnelles et que nous manquions de temps, nous ne les avons pas implémentées.

4 CONCLUSION

4.1 Explication organisation du groupe

Pour le développement de ce projet, nous avons adopté une approche agile en intégrant des principes de la méthode Scrum. Nous avons privilégié la communication aux processus stricts, en favorisant des interactions régulières tout au long du travail.

Chaque session de travail débute par un daily scrum de 5 à 10 minutes, durant lequel nous faisons le point sur les avancées précédentes et répartissons les tâches à venir. Nous n'hésitions pas à nous regrouper lors des séances pour dissiper des doutes ou clarifier certains points.

Pour la gestion des versions de l'application, nous avons utilisé GitHub, tandis que GitHub Projects nous a permis d'organiser et d'assigner efficacement les différentes tâches du projet.

4.2 Eco-conception Web

On a fait en sorte de n'ajouter que des fonctionnalités vraiment utiles, sans rien de superflu. Chaque élément de notre produit répond à un besoin concret et est réellement utilisé par les utilisateurs. Cette approche d'éco-conception nous

permet d'optimiser les ressources, de limiter l'empreinte écologique de notre solution et de garantir une efficacité maximale sur tout le cycle de vie du produit.

4.3 Personnel

4.3.1 Diego Teixeira Nunes

Ce projet m'a vraiment permis de mieux comprendre les défis techniques du développement backend, notamment la sécurisation des données et la mise en place d'une API REST complète. Travailler en équipe m'a aussi aidé à améliorer ma collaboration, à mieux répartir les tâches et à m'adapter aux exigences d'un projet concret.

4.3.2 Mathieu Joshua Bamert

Ce projet m'a permis de mieux comprendre le fonctionnement d'une application web et d'acquérir des bases solides en développement. Travailler en équipe m'a appris à organiser un projet de manière efficace, ce qui reflète bien les conditions du monde professionnel. J'ai également pu renforcer mes compétences en JavaScript en développant une API REST.

4.3.3 Evan Sottile

J'ai apprécié le projet, car il m'a permis d'appliquer concrètement les connaissances acquises en cours, tout en évitant le simple copier-coller redondant. J'ai aussi aimé travailler en équipe, car cela reflète les conditions réelles du monde professionnel et m'a préparé à la collaboration en entreprise.

4.4 Général

Ce projet a été une expérience enrichissante à plusieurs niveaux, tant sur le plan technique que collaboratif. Il nous a permis de renforcer nos compétences en développement backend, notamment à travers la mise en place d'une API REST sécurisée et fonctionnelle. L'apprentissage et l'application de bonnes pratiques, comme l'utilisation de GitHub et des méthodologies agiles, nous ont préparés aux exigences du monde professionnel.

En travaillant en équipe, nous avons amélioré notre capacité à organiser un projet, à répartir les tâches efficacement et à nous adapter aux défis rencontrés. Cette collaboration nous a également permis de mieux comprendre l'importance de la communication et du travail collectif dans un environnement professionnel.

Enfin, ce projet nous a offert une véritable opportunité d'apprentissage pratique, en nous permettant de mettre en œuvre nos connaissances théoriques de manière concrète et de gagner en autonomie. C'est une expérience précieuse qui nous servira pour la suite de notre parcours.

4.5 Critique planification

La planification initiale du projet présente des déséquilibres dans la granularité des tâches, avec certaines trop larges et d'autres trop petites. Une approche plus structurée et équilibrée permettra une meilleure gestion du projet, facilitant son exécution et son suivi. En appliquant des méthodes adaptées et en ajustant la taille des tâches de manière optimale, le projet pourra gagner en efficacité et en clarté.

5 WEBOGRAPHIE

bcrypt - npm
express - npm
Sequelize
Methods Sequelize
express.json() Function - GeeksforGeeks
jsonwebtoken - npm

Collectif Conception responsable de service numérique