Développer la partie back-end d’une application web ou web mobile sécurisée

(CCP2)

Portfolio

Sommaire

Introduction

Objectif du projet :

Développer une application MERN (MongoDB, Express, React, Node.js) permettant de gérer et d'afficher des compétences via un portfolio dynamique. L'application doit être sécurisée, responsive (mobile-first) et conforme aux exigences RGPD, incluant un système de gestion des cookies (Tarteaucitron.js) et un Captcha (Google reCAPTCHA ou Tarteaucitron).

Compétences évaluées et technologies utilisées :

1. **Maquetter des interfaces utilisateur :**
    - **Outils :** Figma, Adobe XD
    - **Bonnes pratiques UX/UI :**
        - Design mobile-first
        - Navigation intuitive entre le portfolio et le Dashboard
        - Affichage des compétences sous forme de cartes interactives
    - **Astuce :** Utilisation de Flexbox ou Grid pour un design fluide.

2. **Base de données relationnelle (MongoDB + Mongoose) :**
    - **Collections MongoDB :**
        - `users`: nom, email, mot de passe (haché avec bcrypt), rôle (admin/user)
        - `skills`: titre, catégorie, niveau (débutant, intermédiaire, expert), image (Cloudinary)
        - `settings`: préférences utilisateur, choix des cookies
    - **Astuce :** Indexation des champs fréquents pour optimiser les requêtes.

3. **Composants d'accès aux données (NoSQL) :**
    - **Backend :** Express + MongoDB
    - **API REST (CRUD) :** `/api/skills`
    - **Upload d'images :** Cloudinary via Multer
    - **Authentification sécurisée :** JWT & bcrypt
    - **Astuce :** Utilisation de httpOnly cookies pour protéger le JWT des attaques XSS.

4. **Composants métier côté serveur :**
    - **Sécurité :**
        - `helmet` pour protéger les en-têtes HTTP
        - `cors` pour gérer les accès cross-origin
        - `Tarteaucitron.js` pour la gestion des cookies (RGPD)
        - `reCAPTCHA` ou `Tarteaucitron CAPTCHA` pour bloquer les bots
        - Middleware de rôles (admin, utilisateur) pour sécuriser le Dashboard
    - **Astuce :** Utilisation de Winston ou Morgan pour le logging des actions utilisateurs.

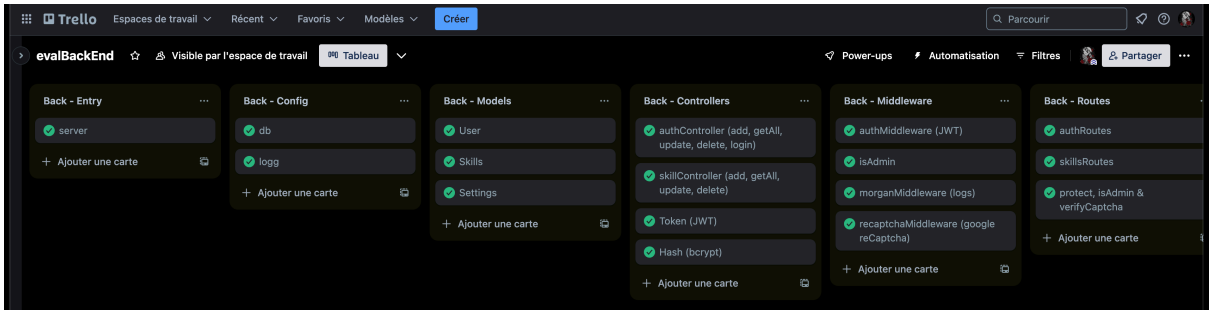
5. **Documentation du déploiement :**
    - **README.md** détaillant :
        - Installation et exécution du projet
        - Déploiement backend (Render, Railway, Heroku)
        - Déploiement frontend (Vercel, Netlify)
        - Collection Postman pour tester l'API
    - **Astuce :** Création d'un schéma d'architecture pour clarifier l'organisation du projet.
6. **Interfaces utilisateur statiques et dynamiques (React) :**
    - **Frontend :** React avec approche Mobile-First & Responsive
    - **Fonctionnalités :**
        - Portfolio dynamique pour afficher les compétences
        - Dashboard sécurisé pour gérer les compétences (ajout, modification, suppression)
        - Formulaire de connexion avec reCAPTCHA
        - Gestion des cookies avec Tarteaucitron.js
        - Utilisation de React Router et Axios pour la navigation et les requêtes
        - Mise en page responsive avec Tailwind CSS ou Bootstrap
    - **Astuce :** Utilisation de React Query ou Redux Toolkit pour optimiser les requêtes.

Livrables attendus :

- Code source hébergé sur GitHub avec un **README.md** clair
- **Rapport de projet** expliquant les choix techniques
- **Diaporama de présentation** pour la soutenance
- **Collection Postman** pour les tests API

Oganisation et suivi

Pour l'organisation des tâches je me suis aidé de l'outil en ligne Trello :



Extrait du trello

J'ai hébergé mon projet sur github et je l'ai envoyé via commande terminal git

github & git logos

Github est un service d'hebergement cloud pour le code. Il sert à gérer le code et à faire du versionning

Git est un système de contrôle de version, il permet une collaboration fluide et un contrôle des versions et des contributeurs.

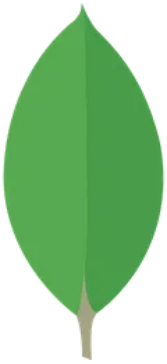
Développement

Pour réaliser ce projet j'ai eu besoin de plusieurs outils :

Pour développer j'ai utilisé Visual Studio Code car je l'utilise depuis plusieurs années et je me sens à l'aise avec.

Il y a un grand choix d'extensions qui m'aident beaucoup dans mes projets.

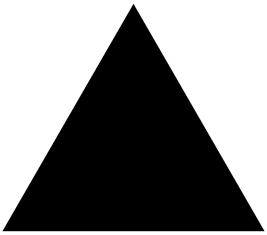
De plus VSCode simplifie pas mal le lien avec github pour ceux qui ont du mal avec les commandes git sur le terminal.



La base de donnée utilisée est MongoDB.

Nous l'avions déjà utilisé de nombreuses fois en cours et je me sentais assez à l'aise pour la réutiliser aisément.

Pour le déploiement du backend j'ai utilisé Render, hébergeur que nous avions déjà utilisé lors de la formation et qui nous a été recommendé par notre formateur.



Quant au frontend il a été déployé sur Vercel. Idem que Render, nous l'avions déjà utilisé au cours de la formation.

Tout d'abord il faut initier le projet node pour le backend avec la commande `npm init`

```
MacBook-Air-de-alpha:back alpha$ npm init
```

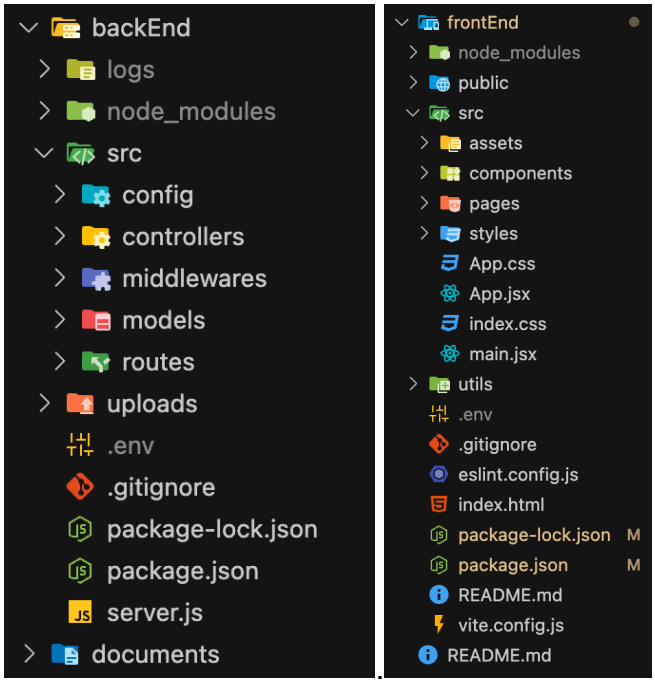
Ensuite il faut initier le projet vite react pour le frontend à la racine du projet avec la commande ` npm create vite@latest`

```
MacBook-Air-de-alpha:back alpha$ npm create vite@latest
```

Après ceci il faut installer les dépendances néxessaires dans chaque parties du projet (front et back) avec la commande `npm i <le nom des dependances>`

```
MacBook-Air-de-alpha:back alpha$ npm i express jsonwebton bcrypt mongoose etc...
```

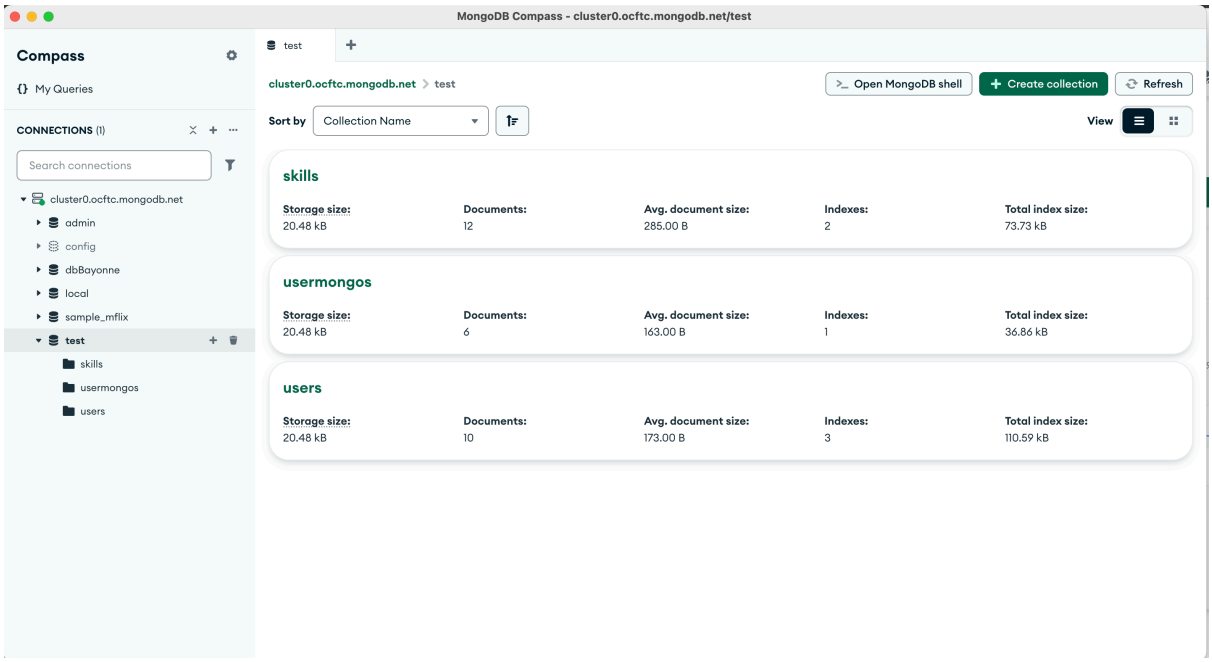
Une fois les dépendances installées il faut créer l'arborescence du projet



Base de données

Pour la base de donnée j'ai crée un compte sur Mongo Atlas puis j'ai récupéré le MONGO_URI que j'ai ajouté dans le fichier .env qui sert à avoir des variables d'environnement qui peuvent être des données sensibles qui ne doivent pas êtres publiques.

```
MONGO_URI=mongodb+srv://ryan
```



Ensuite j'ai configuré la connexion à base de données avec mongoose dans un fichier db.js

```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log(`Base de donnée connectée`);
  } catch (error) {
    console.error(`Erreur lors de la connexion à la base de donnée`, error);
  }
}

module.exports = connectDB
```

Pour finir j'ai lancé la connexion de la base de donnée

```
// Express
const express = require('express');
const app = express();
```

```
// DB
const connectDB = require('./src/config/db');
```

```
// Server
app.listen(PORT, () => {
  connectDB();
  console.log(`Le server tourne sur : http://localhost:${PORT}`);
});
```

CRUD (create, read, update, delete)

Dans cette application il y a deux CRUD différents, un pour les utilisateurs et l'autre pour les compétences.

Dans un premier temps j'ai crée la gestion d'authentification avec :

- Création d'utilisateur avec hachage de mot de passe pour la sécurité (bcrypt)

```
exports.registerUser = async (req, res) => {
  const { name, email, password, role } = req.body;
  if (!name || !email || !password) {
    return res.status(400).json({ message: `Tous les champs doivent être remplis` });
  }
  try {
    const hashedPassword = await bcrypt.hash(password, saltRounds);      You, 2 days ago
    const user = await User.create({ name, email, password: hashedPassword, role });
    res.status(201).json({ message: `L'utilisateur a bien été crée: `, user });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: `Erreur lors de la création de l'utilisateur` });
  }
};
```

```
_id: ObjectId('67b73e534aea368517ce62b6')
name : "Ryan"
email : "ryanadmin@mail.com"
password : "$2b$10$2JItu1EO8PZKPkzMIf6lauK3Fn0aoyuAgwM4aTaeFyqxD0xvfhBpa"
role : "admin"
createdAt : 2025-02-20T14:38:11.776+00:00
updatedAt : 2025-02-20T14:38:11.776+00:00
__v : 0
```

- Récupération de tous les utilisateurs pour les afficher

```
exports.getAllUsers = async (req, res) => {
  try {
    const users = await User.find().select("-password");
    res.status(200).json({ users });
  } catch (error) {
    res.status(500).json({ message: `Erreur lors de la récupération des utilisateurs` })
  }
}
```

- Modification d'un utilisateur via son id

```
exports.updateUser = async (req, res) => {
  try {
    const { id } = req.params;
    const { name, password } = req.body
    const updated = { name };
    if (password) {
      updated.password = await bcrypt.hash(password, saltRounds);
    }
    const updatedUser = await User.findByIdAndUpdate(id, updatedFields, { new: true }).select("-password");
    res.status(200).json({ message: `L'utilisateur a bien été mis à jour`, user: updatedUser });
  } catch (error) {
    res.status(500).json({ message: `Erreur lors de la modification de l'utilisateur`, error });
  }
}
```

- Suppression d’un utlisateur via son id

```
exports.deleteUser = async (req, res, next) => {
  try {
    const { id } = req.params
    const user = await User.findByIdAndDelete(id)
    res.status(200).json({ message: `Utilisateur supprimé avec succès`, user });
  } catch (error) {
    next(error);      You, 2 days ago • commit
  }
};
```

- Connexion d’utilisateur et génération d’un token pour vérifier son identité

```
exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const userLogin = await User.findOne({ email });

    if (!userLogin) {
      return res.status(404).json({ message: `Cet utilisateur n'existe pas` })
    }

    const isMatch = await bcrypt.compare(password, userLogin.password);
    if (!isMatch) {
      return res.status(401).json({ message: `Email ou mot de passe incorrect` })
    }

    const token = await generateToken(userLogin._id);
    res.cookie('jwt', token, {
      httpOnly: true,
      secure: true,
    });

    res.status(201).json({ userLogin, token })
  } catch (error) {
    res.status(500).json({ message: `Erreur lors de la connexion`, error: error.message })
  }
}
```

Documentation du projet

Pour la documentation j'ai utilisé un Readme que j'ai placé à la racine du projet. Dans celui-ci j'y ai indiqué plusieurs informations qui sont les suivantes :

- Le sujet
- Les fonctionnalités
- Les technologies utilisées
- Les prérequis
- Comment l'installer, le configurer et le lancer
- Les dépendances installées
- Des identifiants d'un adminisrateur pour effectuer les tests
- L'arborescence du projet
- La collection postman pour les tests des routes
- Et les déploiements front-end et back-end

Conclusion

En résumé, ce projet incorpore des fonctionnalités robustes et des technologies modernes pour offrir une solution efficace et performante. Grâce à des prérequis bien définis et une documentation détaillée sur l'installation, la configuration et le lancement, les utilisateurs et administrateurs peuvent naviguer et tester aisément le système. L'intégration des dépendances essentielles et la fourniture d'identifiants administrateurs garantissent un testing complet et efficace. Avec une arborescence de projet bien structurée et une collection Postman dédiée, chaque route est testée avec précision. Enfin, les déploiements front-end et back-end assurent une flexibilité et une adaptabilité maximales. Ce projet promet donc une implémentation fluide et un fonctionnement optimal.