Projet REMINDR

Documentation éditée le : 20 décembre 2023

Dernière mise à jour le : 22 décembre 2023

Auteurs :

- Maxence SABATIER
- Célian JUBLOT
- Corentin GUILBERT

Table des matières

Livrables

1. **Code source :** Le code source du projet est disponible sur Github : https://github.com/Maxence-S/remindr
   → **La branche à cloner est la branche “main”.**

2. **Documentation technique :** Ce présent document a pour objectif d’expliquer nos choix technologiques, la structure de notre projet ainsi que les détails.

Choix technologiques

Afin de mener à bien ce projet, plusieurs technologies ont été utilisées. On retrouve HTML, CSS pour la réalisation de l’interface graphique de l’application. Pour la partie back-end, Node.js et SQLite ont été utilisés pour la gestion de l’application et de la base de données.

Principaux modules Node.js

Express.js

Express.js est un framework web pour Node.js, conçu pour simplifier la création d'applications web en fournissant des fonctionnalités middleware et des outils pour la gestion des routes. Des modules complémentaires tels que express-session sont également utilisés pour faciliter la gestion de l’application.

Prisma.js

Prisma.js est une bibliothèque d'accès aux données (ORM) qui facilite l'interaction avec la base de données dans les applications Node.js et TypeScript. Il offre une abstraction de la base de données et facilite la gestion des opérations CRUD (Create, Read, Update, Delete).

Handlebars

Handlebars est un moteur de modèle pour la création de vues dans les applications web. Il permet d'intégrer du code JavaScript dans les modèles HTML, facilitant ainsi la création de pages dynamiques.

Pourquoi ces modules ?

Combinés ensemble, ces trois modules pour Node.js permettent de gérer efficacement une application web, ce qui convient parfaitement dans le cadre de notre application de gestion de rappels.

SQLite

SQLite est un système de gestion de base de données relationnelle léger, autonome et sans serveur. Conçu pour être incorporé directement dans des applications, il offre une solution efficace pour le stockage et la gestion de données locales.

Nous avons décidé de choisir ce SGBDR car il offre un système léger, autonome et facile à prendre en main. De plus, il est compatible avec l'utilisation de Prisma.js.

Structure du projet

Sommaire

Le projet est architecturé en plusieurs répertoires et organisé autour d'un fichier **index.js** qui est la base de l'application. Celui-ci initialise les différents modules, la session utilisateur ainsi que les routeurs.

Répertoires :

- Routers
- Controllers
- Middleware
- Template
- Services
- public

De plus, on retrouve plusieurs fichiers annexes à la racine du projet, tels que :

- database.db
- schema.prisma et .env
- package.json et package-lock.json

Routers

Ce répertoire a pour objectif de gérer les différentes routes de l'application qui permettent à l'utilisateur de se déplacer au sein de celle-ci.

- **appRouter.js** : Routeur principal de l'application. Il est responsable de la grande majorité des routes de l'application.
- **groupRouter.js** : Routeur secondaire qui contient les routes concernant les pages de groupes.

Controllers

Ce répertoire contient toutes les fonctions principales exécutées par les routes.

- **appController.js** : Contrôleur principal de l'application. Il contient les fonctions permettant l'authentification et la création de compte, les fonctions permettant d'afficher les pages principales, d'ajouter un rappel ou encore de se déconnecter de l'application.

- **groupsController.js** : Contrôleur qui va de pair avec groupRouter.js. Il gère les fonctions d'affichage d'un groupe ou de tous les groupes d'un utilisateur, ainsi que la création d'un groupe ou l'ajout d'un utilisateur à un groupe spécifique.

Middlewares

Ce répertoire contient toutes les fonctions annexes exécutées avant les fonctions des controllers.

- **bodyparser.js** : Middleware permettant de lire les données d'une requête POST et permet d'éviter l'affichage des données d'un formulaire dans l'URL

- **groups_middlewares.js** : Middlewares permettant d'ajouter un groupe à la base de donnée, de rajouter un utilisateur dans un groupe (dans la base de donnée) ou encore de vérifier l'accès d'un utilisateur à un groupe en particulier.

- **isConnected.js** : Middleware permettant de s'assurer que la session est active et que l'utilisateur est connecté. Il s'agit d'une fonction généralement exécutée avant l'accès à une page d'un utilisateur de l'application.

- **login_register.js** : Middlewares permettant de créer un nouvel utilisateur dans la base de donnée ou de vérifier si les identifiants entrés dans le formulaire de connexion sont corrects.

- **reminder_middleware.js** : Middleware permettant d'ajouter un nouveau rappel dans la base de donnée pour le groupe correspondant.

Template

Ce répertoire contient les modèles de pages web utilisés par l'application. Un sous-répertoire **connexion** contient les pages permettant de se connecter ou de s'inscrire.

- **connexion/connexion_page.html** : Page de connexion. Il s'agit de la page par défaut sur lequel on arrive au lancement de l'application.
- **connexion/register.html** : Page permettant de s'inscrire sur l'application.
- **dashboard.hbs** : Modèle Handlebars pour la page d'accueil d'un utilisateur. Elle affichera les principales informations sur celui-ci et lui permettra de naviguer dans l'application.
- **file_rappel.html** : Page censée afficher tous les rappels de l'utilisateur peu importe le groupe. Cependant celle-ci n'a pas pu être terminée.
- **groupes.hbs** : Modèle Handlebars affichant tous les groupes d'un utilisateur.
- **one_group.hbs** : Modèle Handlebars qui sert à afficher un groupe en particulier.

Services

Répertoire contenant des scripts secondaires nécessaires à certaines fonctions de l'application.

- **utils.js** : Contient une classe d'erreurs personnalisée et une fonction utile à certaines fonctions des contrôleurs.

public

Répertoire contenant le style (fichiers CSS) et images de l'application, ainsi que la barre de navigation.

Autres fichiers

Comme dit précédemment, on retrouve plusieurs fichiers supplémentaires à la racine du projet, hormis **index.js**.

- **database.db** : Base de données SQLite utilisée par l'application.
- **schema.prisma, .env** : Fichiers permettant de générer la base de données grâce à prisma.js, et d'établir la connexion avec celle-ci.
- **package.json, package-lock.json** : Fichiers permettant le bon lancement de l'application avec Node.js

Détails de mise en oeuvre

Utilisation de Github

Durant ce projet l'outil nous permettant de centraliser le code et de garder en vue l'avancement du projet était GitHub. Nous avons travaillé via différentes branches qui était ensuite merge sur le main dès lors que la fonctionnalité était testé et fonctionnel.

Utilisation de Discord

Discord à également été un de nos moyens de communication afin de transférer de courte partie de code. Mais également pour pouvoir se répartir les tâches correctement et en garder une trace écrite et claire.

Utilisation de NodeJS

Non seulement Node.js a été la plateforme centrale qui nous a permis de créer l'application, mais cet outil nous a également permis d'héberger localement l'application, en plus de nous permettre d'installer et de mettre en place les modules JS utilisés.

Sécurisation de l'application

Nous avons mis en place plusieurs mesures de sécurité en ce qui concerne la base de données. Avant tout via l'usage de Prisma nous permettant d'éviter toute injection de code SQL pouvant endommager la base de données. Nous avons également usé de Bcrypt pour garantir la sécurité des mot de passe dans la base de données, ces derniers se retrouvant hachés et donc illisibles sans le code originel.

De plus, les middlewares contenus dans **isConnected.js** et **groups_middlewares.js** nous ont permis d'assurer la sécurité de l'accès aux pages. En effet, un utilisateur non connecté ne pourra pas accéder à l'application, tandis qu'un utilisateur connecté le pourra. De plus, un utilisateur sera limité aux groupes dont il est membre.