**Application de gestion de rappels – RemindR**

Modèle de Base de Données :

La base de données de l'application est conçue pour stocker des informations liées aux utilisateurs, aux groupes et aux rappels. Le modèle de données est représenté à l'aide du langage Prisma permettant de faciliter l'interaction avec la base de données.  
  
**Modèle de Donnée**

**1. User**

Le modèle **User** représente les utilisateurs de l'application. Chaque utilisateur est identifié par un identifiant unique (**userId**) et possède un nom d'utilisateur unique (**username**), une adresse e-mail (**useremail**), et un mot de passe (**password**).

Relations:

* Un utilisateur peut appartenir à plusieurs groupes (**groupe**).
* Un utilisateur peut créer plusieurs rappels (**rappelle**).

**2. Group**

Le modèle **Group** représente les groupes auxquels les utilisateurs peuvent appartenir. Chaque groupe est identifié par un nom unique (**groupename**).

Relations:

* Un groupe peut avoir plusieurs utilisateurs (**users**).
* Un groupe peut avoir plusieurs rappels (**rapelles**).

**3. Rappelle**

Le modèle **Rappelle** représente les rappels créés par les utilisateurs. Chaque rappel est identifié par un identifiant unique (**rappelleId**) et possède un nom de rappel (**rappellename**), une description (**description**), une couleur (**couleur**), une date d'échéance (**dateEcheance**), un créateur unique (**creator**), un groupe parent (**groupeParent**), et le nom du groupe parent (**groupeParentName**).

Relations :

* Un rappel est créé par un seul utilisateur (**creator**).
* Un rappel appartient à un seul groupe (**groupeParent**).

**4. UserGroup**

Le modèle **UserGroup** est une table intermédiaire qui représente la relation many-to-many entre les utilisateurs et les groupes.

Relations :

* Un utilisateur peut appartenir à plusieurs groupes (**groupe**).
* Un groupe peut avoir plusieurs utilisateurs (**user**).

Choix technique :

**1. Langage de Programmation : JavaScript**

JavaScript a été choisi comme langage de programmation pour le développement du site web. Cette décision nous a été imposé mais offre une cohérence dans la technologie utilisée sur l'ensemble de l'application.

**2. Système de Gestion de Base de Données : MySQL**

MySQL a été sélectionné comme système de gestion de base de données relationnelle. Cette décision a été prise car nous ne connaissons pas d’autre système. Son intégration avec l'application permet de stocker et de récupérer efficacement les données nécessaires au bon fonctionnement du site.

**3. Frameworks**

**a. Express.js**

Express.js a été choisi comme framework web pour le côté serveur de l'application. Il offre une approche minimaliste du développement web tout en fournissant des fonctionnalités essentielles telles que le routage, la gestion des requêtes et des réponses, ainsi que la facilité d'intégration avec d'autres modules.

**b. Prisma**

Prisma est utilisé comme ORM (Object-Relational Mapping) pour simplifier les opérations de base de données MySQL. Il offre une syntaxe de requête type-safe, facilitant ainsi les interactions avec la base de données tout en assurant la sécurité et la fiabilité des opérations.

**c. Handlebars**

Handlebars est le moteur de modèle choisi pour la création de vues HTML. Il offre une syntaxe simple et expressive pour générer des pages web dynamiques en utilisant des modèles.

Structure du Projet :

**1. node\_modules/**Ce répertoire est généré automatiquement lors de l'installation des dépendances Node.js à l'aide de npm. Il contient les modules externes nécessaires à l'application.

**2. views/**Ce répertoire contient les fichiers de modèles Handlebars utilisés pour générer les vues HTML.

**3. Prisma/**   
Répertoire contenant les fichiers de configuration Prisma, y compris le schéma de la base de données.

**4. Controller/**

Le répertoire controllers contient des fichiers définissant les contrôleurs, qui sont responsables de la gestion de la logique métier et de la manipulation des données. Chaque contrôleur est associé à des routes spécifiques et interagit avec les modèles Prisma pour accéder à la base de données.

**5. Routes/**

Le répertoire routes regroupe les fichiers définissant les routes Express pour l'application. Chaque fichier de route spécifie les points d'entrée de l'API et utilise des contrôleurs du répertoire controllers pour gérer la logique métier associée.

**6. Middleware/**

Le répertoire **middleware** contient des fichiers définissant des middleware Express personnalisés.

**7.public/style**

Le répertoire contient un main.css, permettant de gérer l’aspect visuel du site

**8. package.json** et **package-lock.json**  
Fichiers de configuration de Node.js contenant la liste des dépendances et leurs versions.

**9.gitignore**   
Fichier spécifiant les fichiers et répertoires à ignorer lors de l'ajout au système de contrôle de version Git.

**10. index.js**  
Point d'entrée de l'application, où Express est configuré et les routes sont définies.

**11.env**  
Fichier de configuration pour les variables d'environnement. Il peut contenir des informations sensibles comme les clés d'API, mais ne doit pas être versionné.

Pour ce qui est de la partie technique, nous avons d’abord les routes principal du projet :

Pour l'authentification :

/register : Affiche le formulaire d'inscription.

/register (POST) : Traite le formulaire d'inscription.

/login : Affiche le formulaire de connexion.

/login (POST) : Traite le formulaire de connexion.

/logout : Déconnecte l'utilisateur.

Pour les groupes :

/mesgroupes : Affiche la liste des groupes de l'utilisateur.

/mesgroupes/create-group (POST) : Crée un nouveau groupe.

/mesgroupes/:groupename/members : Affiche les membres d'un groupe.

/mesgroupes/:groupename/add-member (POST) : Ajoute un membre à un groupe.

Pour les rappels :

/mesgroupes/:groupename/edit-reminder/:rappelleId : Affiche le formulaire de modification d'un rappel.

/mesgroupes/:groupename/edit-reminder\_process/:rappelleId (POST) : Traite la modification d'un rappel.

/mesgroupes/:groupename/createreminder (POST) : Crée un nouveau rappel.

/mesgroupes/:groupename/delete-reminder/:rappelleId (POST) : Supprime un rappel.

Pour le tableau de bord (dashbord)

/dashboard : Affiche le tableau de bord.

/dashboard/remindeGrise (POST) : Affiche les rappels expirés en option.

/dashboard (POST) : Affiche tous les rappels.

Concernant les sessions :

Les sessions sont gérées avec le module express-session.

Pour ce qui est des midlewares :

checkIfLoggedIn : Vérifie si l'utilisateur est déjà connecté.

requireLogin : Force la connexion pour accéder à certaines pages.

Pour l’affichage avec handlebars :

**layouts/main.handlebars**

Fonction : Il s'agit du modèle principal qui englobe toutes les autres vues. Il définit la structure HTML de base, inclut des liens vers des feuilles de style et d'autres dépendances.

**partials/footer.handlebars**

Fonction : Définit le pied de page de l'application. Il affiche des informations sur l'application, et s'il y a un utilisateur connecté, il affiche également le nom d'utilisateur du tableau de bord.

**partials/header.handlebars**

Fonction : Définit l'en-tête de l'application. Il inclut le titre de l'application et un menu de navigation. Le contenu du menu change en fonction de la connexion de l'utilisateur.

**views/dashboard.handlebars**

Fonction : Affiche le tableau de bord de l'utilisateur. Il comprend des boutons pour afficher différents rappels, une table pour afficher les rappels à venir, et un formulaire pour ajouter un rappel.

**views/edit-reminder.handlebars**

Fonction : Affiche le formulaire pour modifier un rappel spécifique appartenant à un groupe. Il permet à l'utilisateur de modifier le nom du rappel, la description, la date d'échéance et la couleur.

**views/group-members.handlebars**

Fonction : Affiche la liste des membres d'un groupe particulier. Il permet également à l'utilisateur d'ajouter un nouveau membre au groupe.

**views/create-reminder.handlebars**

Fonction : Affiche le formulaire pour créer un nouveau rappel dans le contexte d'un groupe. Il permet à l'utilisateur de spécifier le nom du rappel, la date d'échéance, la couleur et la description.

**views/group-reminders.handlebars**

Fonction : Affiche la liste des rappels d'un groupe spécifique. Il permet à l'utilisateur de supprimer un rappel du groupe ou de le modifier.

**views/login.handlebars**

Fonction : Affiche le formulaire de connexion. Les utilisateurs peuvent saisir leur nom d'utilisateur et leur mot de passe pour se connecter.

**views/register.handlebars**

Fonction : Affiche le formulaire d'inscription. Les utilisateurs peuvent saisir un nom d'utilisateur, une adresse e-mail et un mot de passe pour créer un nouveau compte.

**views/mesgroupes.handlebars**

Fonction : Affiche la liste des groupes de l'utilisateur, permet à l'utilisateur de voir les membres du groupe et de créer un nouveau groupe.

Ensuite, pour rendre le tout plus joli, j’utilise du code css que j’ai mi dans un fichier accessible n'importe où : **app.use(express.static('public'));**

C’et le fichier **public/styles/main.css**

Pour certains fonctionnalités à vérifier avant la route comme la vérification qu’un utilisateur est connecté, j’ai utilisé des Middleware comme :

requireLogin : Pour ne pas avoir accès à une page si on est pas connecté

checkIfLoggedIn : Pour ne pas avoir accès à la page de login si on est déjà connecté

Gestion des erreurs :

Quand l’utilisateur rentre une information incorrecte, je régénère la page avec un message d’erreur.

* s’il entre le nom d’un utilisateur qui n’existe pas
* s’il veut créer un group avec un nom déjà attribué

Aussi les formulaires ne peuvent pas être validés si un champ est vide.

Si un utilisateur reçoit une adresse mail sans ‘@’ son inscription n’est pas acceptée.

Après, au niveau de la sécurité pour l’identification, nous avons décidé de crypté le mot-de-passe avec la fonction : **bcrypt**

Pour qu’un utilisateur se connecte, le mot de passe entré doit être identique au mot de passe entré au moment de l’inscription et l’identifiant doit exister dans la base et correspondre.

Pour les sessions, la clef de sécurité est très longue et fait de caractère aléatoire. Cela assure une sécurité en plus dans certaines mesures.

Au niveau des controllers il y a 5 fichiers :

**AuthController.js**

**DashboardController.js**

**GetData.js**

**GroupeController.js**

**RemindsController.js**

C’est dans ces fichiers que toute la logique de vérification et d'interaction est faite. Les fichiers routes sont courts et appellent simplement la bonne fonction qui vient de ces fichiers Controller.js.