Fauchart
Lucas

Campus
Carlos Acutis

RAPPORT DE STAGE

BTS SIO option SLAM 2024-2025

Stage du 6 janvier 2025 au 14 février 2025

REMERCIEMENTS

Je tiens à exprimer ma profonde gratitude à Monsieur Gaulier pour son aide précieuse tout au long de mon stage. Son expertise technique et ses conseils m'ont permis de surmonter plusieurs défis et d'améliorer mes compétences en développement. Je remercie également le GIRV et Monsieur Gambier de nous avoir offert cette opportunité de stage, nous permettant de travailler sur un projet concret et enrichissant. Enfin, un grand merci à mon établissement, Carlos Acutis, pour nous avoir accueillis durant cette période et pour nous avoir fourni les ressources nécessaires, notamment l'accès à la connexion internet, afin de mener à bien notre mission.

SOMMAIRE

PRESENTATION DE L'ENTREPRISE

GIRV

Le Groupement Interprofessionnel de la Région de Vernon (GIRV) est une association créée en 1944 par un groupe d'industriels de la région, dans le but de relancer l'activité économique après la guerre. À cette époque, les entreprises locales devaient faire face à de nombreux défis, et l'objectif du GIRV était d'unir leurs forces pour reconstruire et dynamiser le tissu économique. Depuis sa création, le GIRV n'a cessé d'évoluer pour répondre aux besoins des entreprises et accompagner leur développement.

Aujourd'hui, il regroupe plus de 130 entreprises de différents secteurs, représentant plus de 8 200 emplois. Son siège est situé au Campus de l'Espace, un lieu stratégique à Vernon qui favorise l'innovation et la collaboration entre entreprises. L'association fonctionne sous le statut de loi 1901, ce qui signifie qu'elle est à but non lucratif et agit avant tout pour le bien des entreprises locales et de l'économie régionale.

Le rôle principal du GIRV est de créer un réseau solide entre les entreprises de la région afin de favoriser la coopération et les échanges. Il organise régulièrement des

événements professionnels, tels que des conférences, des forums et des rencontres, qui permettent aux dirigeants d'entreprises de partager leurs expériences, d'échanger des idées et de trouver de nouvelles opportunités de développement. Grâce à ces rencontres, les entreprises peuvent se soutenir mutuellement, identifier des partenaires potentiels et bénéficier de conseils précieux pour améliorer leur activité.

Actions Transition
Les entreprises s'engagent pour la résilience

Le GIRV s'investit aussi dans la transition écologique en sensibilisant et en accompagnant les entreprises vers des pratiques plus respectueuses de l'environnement. Il encourage la réduction de la consommation d'énergie, la mise en place du recyclage et la préservation de la biodiversité. De plus en plus d'entreprises prennent conscience de l'importance du développement durable, et le GIRV les aide à intégrer ces nouvelles pratiques dans leur fonctionnement. Cet engagement pour l'environnement est essentiel pour assurer un avenir plus responsable et durable aux entreprises de la région.

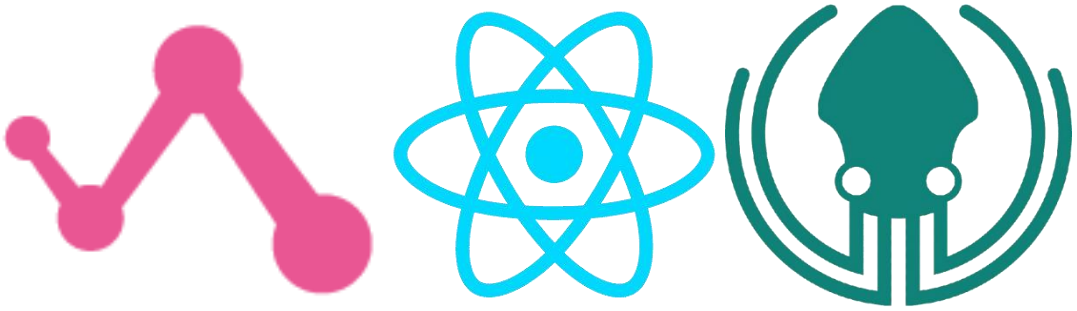
PRESENTATION DU SERVICE D'ACCUEIL

Le GIRV est l'entreprise qui m'accueille pour mon stage. Comme ils ne possèdent pas de locaux, ils ont demandé à mon établissement Carlos Acutis de nous accueillir pour que nous puissions travailler dans de bonnes conditions. Notre mission principale durant ce stage est de développer une application dédiée à l'inter professionnalisation, un outil qui aidera les professionnels à mieux collaborer. Pour nous accompagner dans ce projet, nous sommes encadrés par Monsieur Gaullier, chef de projet, qui nous guide dans la création de l'application, et Monsieur Gambier, représentant du GIRV et superviseur du stage, qui veille à son bon déroulement. Ce stage est une opportunité pour nous de mettre en pratique nos connaissances et de découvrir le travail en équipe sur un projet concret.

MOYEN INFORMATIQUE, HUMAIN ET TECHNIQUE

Moyens informatiques

Pour développer l'application, nous avons utilisé React Native et Expo pour la partie mobile et Node.js avec Express pour le serveur. Le langage principal était JavaScript, avec CSS pour la mise en forme. La base de données était gérée avec MySQL et phpMyAdmin, et hébergée sur Always Data. Nous avons utilisé Visual Studio Code comme éditeur de code et GitKraken et GitHub pour le suivi des versions et le travail collaboratif. Pour tester l'application sur un environnement local, nous avons utilisé WampServer.

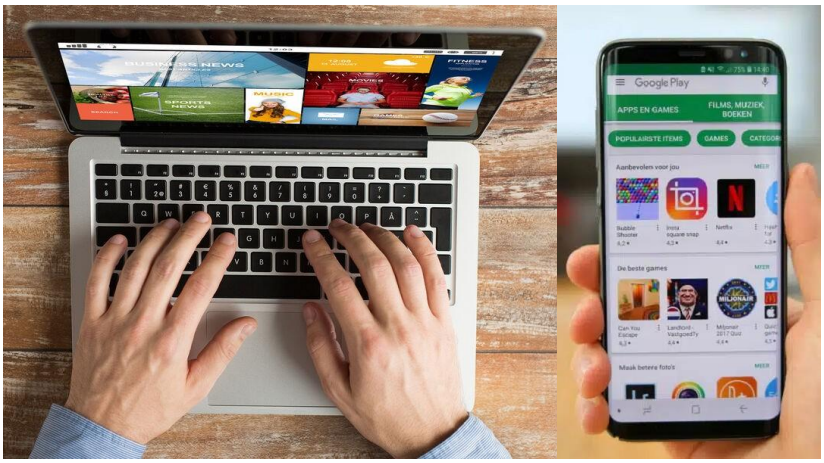


Moyens humains

Nous étions une équipe de trois développeurs, accompagnés par un professeur qui nous aidait lorsque nous rencontrions des difficultés. De plus, un représentant du GIVR nous donnait des retours réguliers pour s'assurer que l'application correspondait aux attentes.

Moyens techniques

Nous avons travaillé sur nos PC personnels pour développer l'application. Pour certaines parties graphiques, nous avons utilisé Canvas. L'organisation du projet s'est faite avec Trello pour la gestion des tâches et Teams pour la communication entre nous. Pour tester l'application, nous avons utilisé Android Studio afin de simuler des appareils virtuels, ainsi que des téléphones physiques sous Android et iOS. Enfin, la connexion internet de l'établissement nous a permis d'accéder aux outils en ligne et de tester notre application en réseau.



PRESENTATION DU SUJET

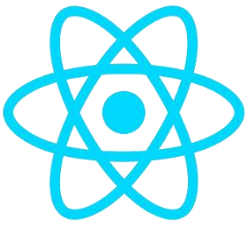
Le sujet de stage consistait à développer une application mobile en React Native pour le GIRV. Le GIRV est une association qui regroupe des chefs d'entreprise, des indépendants et des professionnels de Vernon, avec pour objectif de favoriser les échanges et la collaboration entre ses membres afin de dynamiser l'économie locale.

L'application à concevoir devait s'inspirer de Ki&Ki, une application déjà existante, tout en étant adaptée aux besoins spécifiques du GIRV. Son but principal était de mettre en relation les chefs d'entreprise de Vernon en leur offrant une plateforme où ils pourraient créer un profil, échanger via une messagerie intégrée, et accéder aux événements organisés par le GIRV. L'application devait aussi proposer un espace de discussions en groupe ou en privé afin de faciliter les interactions professionnelles et encourager le networking.

Le projet devait être réalisé par une équipe de trois développeurs, en mettant l'accent sur une interface fluide et intuitive, une navigation simplifiée et un système de notifications permettant de tenir les utilisateurs informés des nouvelles interactions et des événements à venir.

DEVELOPPEMENT DE NOTRE PROJET

Présentation des outils utilisés



React Native est un outil qui nous a permis de créer une seule application compatible avec Android et iOS, en utilisant le même code. Il nous a fait gagner du temps et simplifié le développement.

Expo est une plateforme qui facilite le développement avec React Native. Grâce à Expo, nous avons pu tester rapidement notre application sur nos téléphones sans avoir à tout reconfigurer à chaque modification.

Node.js permet d'exécuter du JavaScript sur un serveur, et Express aide à gérer les requêtes. Ces deux outils nous ont permis de connecter notre application à la base de données et d'envoyer des informations en toute sécurité.

JavaScript nous a permis de créer la logique de l'application, tandis que CSS nous a servi à organiser et styliser l'interface pour qu'elle soit agréable et facile à utiliser.

MySQL est un système de base de données que nous avons utilisé pour stocker et organiser nos informations. Avec phpMyAdmin, nous avons pu gérer cette base de données de façon plus simple grâce à une interface en ligne.

Always Data est un service d'hébergement que nous avons utilisé pour stocker notre base de données en ligne. Cela nous a permis d'y accéder depuis n'importe où sans avoir besoin d'un serveur physique.

Nous avons utilisé un serveur VPS (Virtual Private Server) chez Hostinger pour héberger notre backend. Cela nous a permis d'avoir un serveur performant et sécurisé, accessible en permanence.

FileZilla est un logiciel qui permet de transférer des fichiers entre notre ordinateur et notre serveur en ligne.

Nous avons utilisé nos propres ordinateurs pour coder, tester et améliorer l'application. Chaque développeur avait les outils nécessaires installés sur son PC.

Nous avons testé l'application sur nos téléphones pour vérifier qu'elle fonctionnait bien sur des appareils réels, en plus des tests sur simulateur.

Canvas nous a permis de créer des maquettes pour planifier l'apparence et les fonctionnalités de l'application avant de commencer à coder.

Trello nous a aidés à organiser nos tâches et à suivre l'avancement du projet en attribuant des missions à chaque membre de l'équipe.

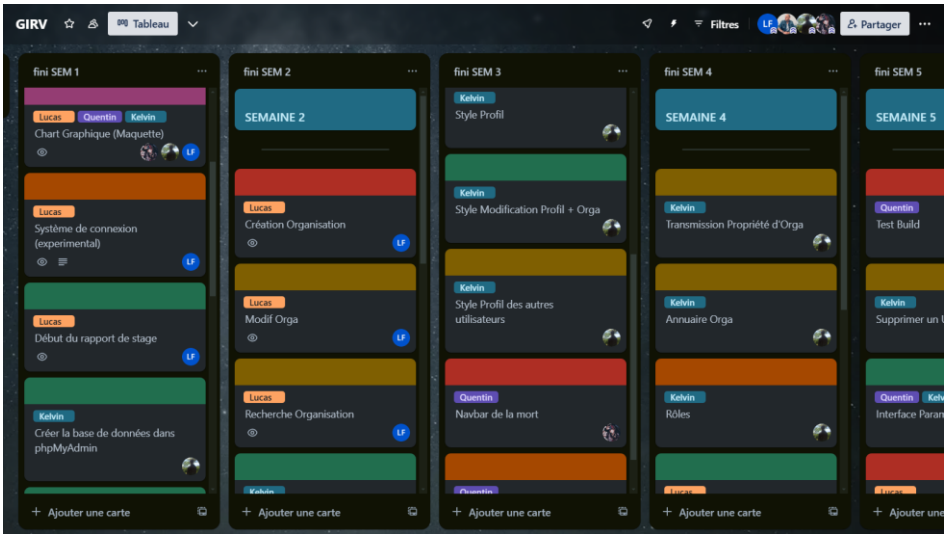
Nous avons utilisé Teams pour communiquer facilement, partager des fichiers et organiser des réunions en ligne. Cela nous a permis de bien nous coordonner, même à distance.

Pour tester l'application sans forcément utiliser un téléphone, nous avons utilisé le simulateur d'Android Studio. Cela nous a permis de voir le rendu de l'application sur différents types d'appareils et de détecter des erreurs plus facilement.

GitKraken est un outil graphique pour gérer les versions de notre code avec Git. Il nous a permis de suivre l'évolution de notre projet, de gérer les différentes branches de notre code, et de fusionner facilement les modifications. Avec son interface claire, GitKraken a simplifié la gestion des conflits de code et nous a permis de collaborer plus efficacement en équipe. Chaque développeur pouvait ainsi travailler sur des parties différentes du projet sans risquer de perdre des modifications importantes.

Présentation des méthodes utilisées

Pendant notre stage, nous avons adopté une approche où chaque membre de l'équipe choisissait ses tâches et avançait à son rythme. Pour organiser et suivre l'avancement du projet, nous avons utilisé Trello, un outil de gestion de projet qui nous a permis de diviser les tâches en différentes catégories, d'assigner des responsabilités et de suivre l'état de chaque tâche en temps réel. Nous avons ainsi pu mieux coordonner nos efforts et respecter les délais tout au long du projet.

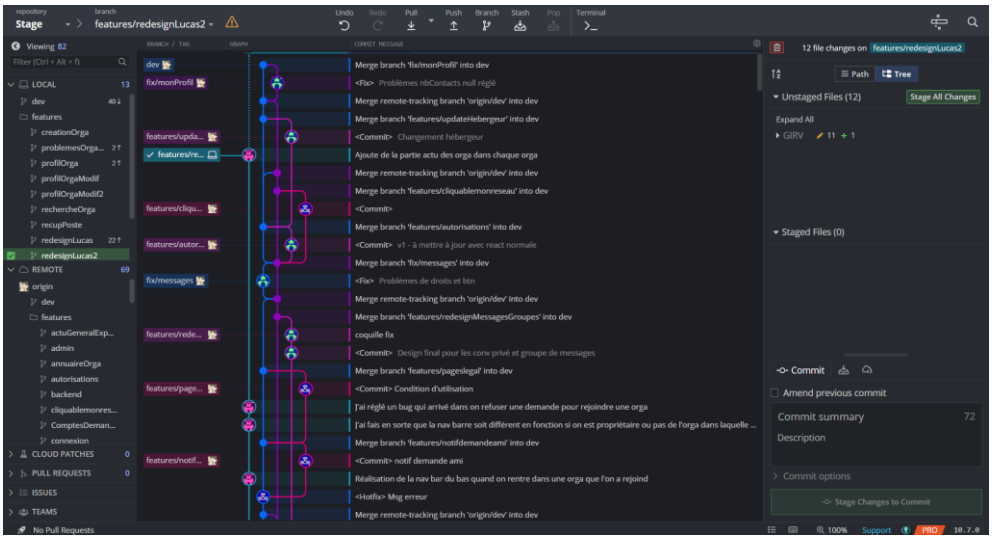


Le développement de l’application s’est fait avec React Native et Expo, ce qui nous a permis de créer une application multiplateforme et de tester rapidement l’application sur iOS et Android, tant sur des appareils physiques que via le simulateur Android Studio. Nous avons conçu les maquettes de l'application avec Canvas pour valider l’interface utilisateur avant de commencer le développement.

Pour la gestion du backend, nous avons d'abord utilisé WAMP Server en local pour tester notre API et la base de données MySQL. Ensuite, nous avons migré l’API vers AlwaysData pour la rendre accessible en ligne, puis, pour améliorer les performances et la stabilité, nous avons décidé de la déployer sur un VPS Hostinger.

Côté technique, nous avons utilisé des bibliothèques comme React Native Paper pour créer des composants réutilisables et améliorer l'interface utilisateur. Pour gérer les requêtes API, nous avons choisi Axios, et pour la navigation entre les écrans, nous avons intégré React Navigation. La sécurité des utilisateurs a été assurée grâce à JWT et bcrypt pour le chiffrement des mots de passe. Nous avons également utilisé Socket.io pour gérer les échanges en temps réel, notamment pour les notifications et la messagerie.

En termes d’organisation du code, nous avons utilisé GitKraken comme outil de gestion de version. GitKraken nous a facilité la gestion des branches, la réconciliation des modifications et le suivi des différentes versions du projet. Nous avons également utilisé GitHub pour stocker le code source et collaborer à distance, permettant à chaque membre de l’équipe de travailler sur différentes fonctionnalités sans risque de conflits.



Les tests ont été réalisés sur des smartphones physiques Android et iOS grâce à Expo Go, ainsi que sur le simulateur Android via Android Studio. Après plusieurs phases d’amélioration et de correction, l’application a été déployée sur le VPS Hostinger, assurant ainsi une performance optimale pour la mise en production.

Planification des tâches confiées

Création de maquettes pour l'interface utilisateur (UI)

Conception du Modèle Conceptuel de Données (MCD) :

- Création d'un modèle pour structurer la base de données et de la base de données (avec les autres développeurs)
- Ajout de nouvelles tables pour gérer les likes et les postes.

Installation des dépendances pour l'API et structure de l'API

- Organisation du code dans différents fichiers pour gérer la base de données et les routes.
- Développement de fichiers comme bd.js, middleware.js, index.js et différents fichiers pour gérer les routes comme connexion.js, inscription.js, organisation.js, etc.

Mise en place des routes API.

- Routes de connexion et d'inscription : gestion des utilisateurs, vérification des emails, et cryptage des mots de passe.
- Routes d'organisation : création d'organisations, affichage d'informations détaillées, modification du profil d'organisation.
- Routes de gestion des postes : création, modification et suppression des postes, ainsi que la gestion des likes.
- Routes de gestion des demandes d'adhésion et de publication de postes : gestion des demandes de rejoindre une organisation, ainsi que l'acceptation ou le refus de ces demandes.

Développement des interfaces utilisateur (UI).

- Conception des pages de connexion, d'inscription, de création d'organisation, de modification de profil, de demande d'adhésion, etc.
- Mise en place de design épuré et moderne avec des éléments visuels comme des bulles animées et des cartes stylisées.

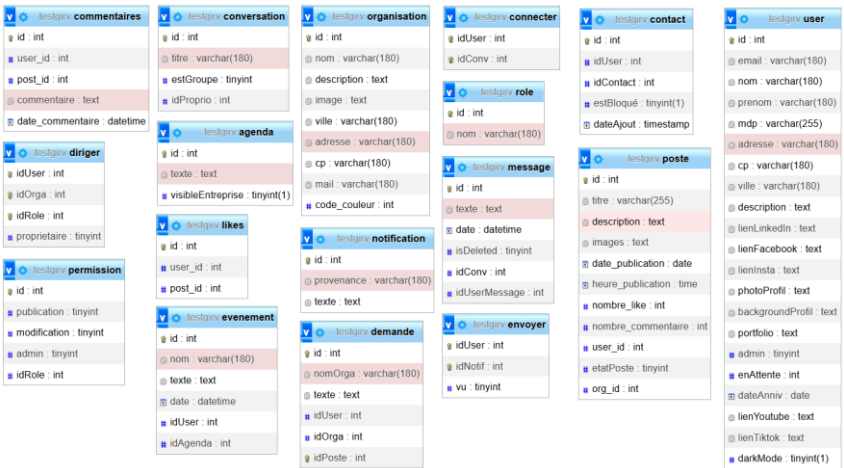
Gestion de la navigation pour la partie organisation dans l'application.

Explication technique des tâches confiées

La réalisation de plusieurs maquettes a été essentielle pour la conception et le développement du projet, notamment dans le cadre de mon stage. Elles ont permis de visualiser l'interface utilisateur (UI) avant de commencer le développement réel de l'application. Ces maquettes ont servi à définir l'ergonomie, la navigation et

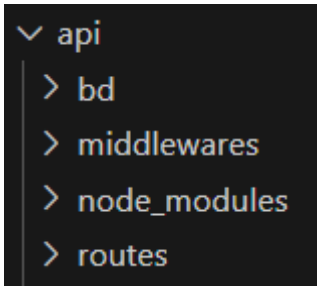
l'apparence générale de l’application, et ont facilité la validation des choix visuels et fonctionnels.

En parallèle, avec les deux autres développeurs de mon stage, nous avons créé un Modèle Conceptuel de Données (MCD) pour structurer la base de données du projet. Ce MCD a permis de définir les entités et leurs relations. Cependant, au fur et à mesure du projet, nous avons remarqué qu'il manquait certaines tables importantes dans la base de données, comme celles pour les likes et les postes. Ces tables étaient essentielles pour permettre aux utilisateurs d'interagir avec les publications. J’ai donc dû ajouter de nouvelles tables pour gérer les likes et les postes. Ces ajustements ont permis d'enrichir la base de données et de rendre l'application plus fonctionnelle. Cela a aussi mis en lumière l'importance d'une réflexion approfondie sur les besoins dès le début du projet pour éviter de tels oublis.



Dans le cadre du projet, plusieurs dépendances ont été installées pour gérer les différentes fonctionnalités de l'API. Express est utilisé pour gérer les requêtes HTTP, les routes et la logique côté serveur. JWT permet la gestion des jetons d'authentification, notamment avec la fonction verify pour vérifier un jeton et generateToken pour en créer un à partir d'une clé secrète et de l'ID utilisateur. MySQL est utilisé pour établir la connexion avec la base de données via phpMyAdmin. Body-Parser permet de traiter les requêtes en JSON pour les envoyer au front-end. Cors facilite l'accès aux routes API depuis le front-end. Bcrypt est utilisé pour sécuriser les mots de passe en les cryptant grâce à la méthode hash. Enfin, Nodemon assure une mise à jour automatique du serveur lorsqu'une modification est effectuée dans le code.

Ajout de plusieurs dépendances aussi pour la partie visuelle de l'application @react-native-async-storage/async-storage est utilisé pour stocker localement des données de manière asynchrone. @react-navigation/native, @react-navigation/native-stack, et @react-navigation/stack sont des bibliothèques permettant de gérer la navigation entre les écrans de l'application, en offrant différentes structures comme la navigation en pile (stack). Axios est une bibliothèque permettant de faire des requêtes HTTP pour communiquer avec l'API. Moment est utilisé pour manipuler et formater les dates de manière simple et efficace. React et React Native sont les bibliothèques de base permettant de construire l'interface et la logique de l'application mobile. react-native-flash-message sert à afficher des messages de notification temporaires aux utilisateurs. react-native-image-picker permet de sélectionner et gérer des images depuis la galerie ou l'appareil photo. react-native-safe-area-context et react-native-screens assurent une meilleure gestion de l'affichage en fonction des zones sûres des appareils modernes. Enfin, react-native-vector-icons fournit une collection d'icônes personnalisables pour améliorer l'interface utilisateur.



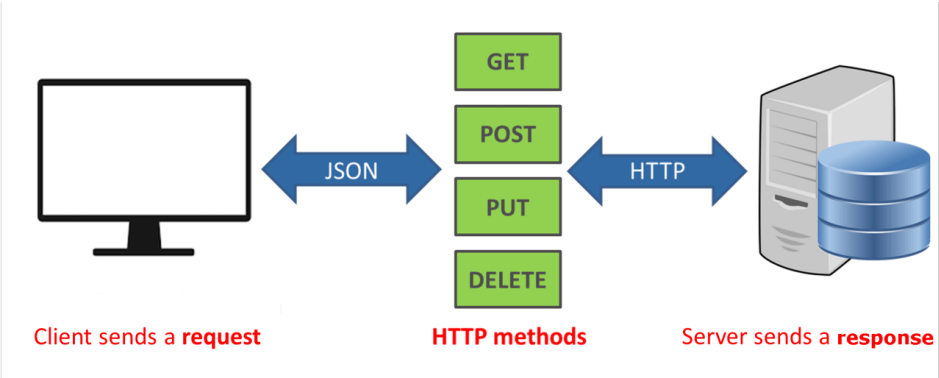
L'API est structurée dans un dossier spécifique contenant plusieurs fichiers. Le fichier bd.js est chargé de gérer la connexion à la base de données hébergée sur phpMyAdmin, AlwaysData ou Hostinger. Le fichier middleware.js permet de vérifier si l'utilisateur possède un jeton d'authentification avec JWT et d'en extraire ses informations pour les utiliser dans les différentes routes sécurisées.

Le fichier index.js est le point d'entrée principal de l'API. Il importe Express, qui est stocké dans une variable app, ainsi que Cors et Body-Parser pour gérer les requêtes. Il importe également tous les fichiers contenant les routes de l'API pour les rendre accessibles au front-end. Ce fichier définit également le port sur lequel l'API sera hébergée : 3000 en local et 990 sur AlwaysData.

D'autres fichiers sont dédiés aux différentes routes de l'API. Par exemple, connexion.js gère les routes liées à l'authentification, tandis que inscription.js, organisation.js et poste.js contiennent les routes correspondantes. Tous ces fichiers importent bd.js pour

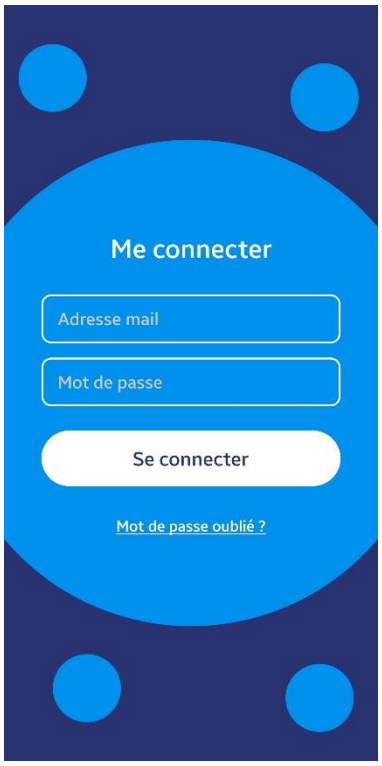
interagir avec la base de données et middleware.js pour sécuriser l'accès à certaines routes. Ils incluent également des messages d'erreur pour gérer les éventuels problèmes lors des requêtes.

Chaque route utilise une méthode HTTP spécifique : GET pour récupérer des données, POST pour en envoyer, PUT pour les modifier et DELETE pour les supprimer.



Dans connexionRoute.js, une seule route /connexion est mise en place. Elle récupère les informations saisies par l'utilisateur dans le formulaire de connexion (email et mot de passe), puis vérifie si l'email existe en base de données. Si l'email est trouvé, le mot de passe est comparé avec celui stocké grâce à bcrypt. Si tout est valide, un jeton d'authentification est généré en utilisant l'ID utilisateur et une clé secrète avec JWT, permettant ainsi une connexion sécurisée.

```
app.post("/connexion", (req, res) => {
  const { email, password } = req.body;

  // Requête SQL pour trouver l'utilisateur avec son email
  const query = "SELECT * FROM user WHERE email = ?";
  appConnection.query(query, [email], async (err, results) => {
    if (err) {
      console.error("Erreur de la base de données :", err);
      return res.status(500).json({ error: "Erreur de la base de données" });
    }

    // Vérifie si un utilisateur existe avec cet email
    if (results.length === 0) {
      console.log("Aucun utilisateur trouvé avec cet email :", email);
      return res.status(401).json({ error: "Utilisateur non trouvé avec cette adresse mail" });
    }

    const user = results[0];

    // Compare le mot de passe crypté
    const passwordMatch = await bcrypt.compare(password, user.mdp);
    if (!passwordMatch) return res.status(401).json({ error: "Mot de passe incorrect" });

    // Crée le token
    const secretKey = process.env.JWT_SECRET || "MaCléSecrèteTrèsSécurisée";
    const payload = {id : user.id}
    const token = generateToken(payload, secretKey);
    console.log("Token : ", token)

    res.json({ token });
  });
});
```



L'interface présente un design épuré et moderne. En haut de l'écran, on retrouve un fond de couleur sombre, avec plusieurs éléments circulaires de couleur bleue qui flottent en arrière-plan, créant un effet dynamique. Le titre "Me connecter" est centré et en gros caractères blancs, offrant une bonne visibilité. En dessous, l'utilisateur voit deux champs de saisie : un pour l'adresse e-mail et un autre pour le mot de passe. Ces champs sont stylisés avec des bordures blanches et un fond transparent, ce qui les rend visibles mais subtils. Le champ du mot de passe est sécurisé, ce qui signifie que le texte saisi est masqué. Un bouton "Se connecter" apparaît en dessous des champs de saisie. Ce bouton est large, de couleur blanche, avec un texte en bleu, incitant l'utilisateur à cliquer pour se connecter. Un lien "Mot de passe oublié ?" est également disponible sous le bouton de connexion, permettant à l'utilisateur de récupérer son mot de passe si nécessaire. L'interface utilise des vibrations légères lors de certaines actions, comme la tentative de connexion ou la récupération du mot de passe.

Dans inscriptionRoute.js, la route /inscription récupère les informations saisies (email, mot de passe, prénom et nom). Elle vérifie si l'email est déjà utilisé en base de données, puis crypte le mot de passe avec bcrypt avant d'insérer les informations dans la table users.

```
app.post("/inscription", (req, res) => {
    const { email, password, nom, prenom } = req.body;

    if (!email || !prenom || !nom || !password) {
        return res.status(400).json({ error: "Tous les champs sont obligatoires." });
    }

    try {
        // Vérifier si l'utilisateur existe déjà
        const query = "SELECT * FROM user WHERE email = ?";
        appConnection.query(query, [email], async (err, results) => {
          if (err) {
            console.error("Erreur de la base de données :", err);
            return res.status(500).json({ error: "Erreur serveur." });
          }

          if (results.length > 0) {
            return res.status(400).json({ error: "Un utilisateur avec cet email existe déjà." });
          }

          // Chiffrer le mot de passe
          const hashedPassword = await bcrypt.hash(password, 10);

          // Insérer l'utilisateur dans la base de données
          const insertQuery = "INSERT INTO user (email, prenom, nom, mdp) VALUES (?, ?, ?, ?)";
          appConnection.query(insertQuery, [email, nom, prenom, hashedPassword], (err) => {
            if (err) {
              console.error("Erreur lors de l'insertion de l'utilisateur :", err);
              return res.status(500).json({ error: "Erreur serveur." });
            }

            res.status(201).json({ message: "Utilisateur créé avec succès !" });
          });
        });
    } catch (error) {
        console.error("Erreur lors de la création de l'utilisateur :", error);
        res.status(500).json({ error: "Erreur serveur." });
    }
});
```

En haut de l'écran, on retrouve des éléments visuels, tels que des bulles animées de couleur bleue qui ajoutent une touche esthétique. Le titre de la page "Inscription" est affiché en grand, au centre. Ensuite, un formulaire est présenté avec plusieurs champs de saisie : l'adresse e-mail, le mot de passe, le nom et le prénom de l'utilisateur. Chaque champ est accompagné d'un texte de placeholder en gris pour guider l'utilisateur sur ce qu'il doit entrer. Tous les champs sont stylisés avec des bordures blanches et un fond transparent pour garantir une bonne lisibilité. Sous ces champs, un bouton "Inscription" permet de soumettre le formulaire. Ce bouton est large, avec un fond blanc et un texte en bleu foncé. L'interface est simple et épurée, avec un fond bleu foncé qui contraste bien avec les éléments blancs et bleus. Ce design favorise une expérience utilisateur claire et agréable.

Dans organisationRoute.js, plusieurs routes sont créées pour gérer les organisations. La route /creationOrganisation permet d'ajouter une organisation en base de données en récupérant les informations saisies par l'utilisateur (nom, description, image, adresse, etc.) ainsi que son ID grâce au middleware. Une vérification est effectuée pour s'assurer que l'organisation n'existe pas déjà. Une fois créée, l'utilisateur est ajouté à la table diriger avec un rôle de propriétaire.

```
router.post("/creationOrga", verifyToken, (req, res) => {
  const { nom, description, image, ville, adresse, codePostal, email } = req.body;
  const idUser = req.user.id; // Récupérer l'ID de l'utilisateur connecté depuis le middleware

  // Vérification des champs requis
  if (!nom || !description || !image || !ville || !adresse || !codePostal || !email) {
    return res.status(400).json({ error: "Tous les champs sont obligatoires." });
  }
  try {
    // Vérifier si l'organisation existe déjà
    const query = "SELECT * FROM organisation WHERE nom = ?";
    dbConnection.query(query, [nom], (err, results) => {
      if (err) {
        console.error("Erreur de la base de données :", err);
        return res.status(500).json({ error: "Erreur serveur." });
      }

      if (results.length > 0) {
        return res.status(400).json({ error: "Une organisation existe déjà avec ce nom." });
      }

      // Insérer l'organisation dans la base de données
      const insertQuery = "INSERT INTO organisation (nom, description, image, ville, adresse, cp, mail) VALUES (?, ?, ?, ?, ?, ?, ?)";
      dbConnection.query(insertQuery, [nom, description, image, ville, adresse, codePostal, email], (err, orgResult) => {
        if (err) {
          console.error("Erreur lors de l'insertion de l'organisation :", err);
          return res.status(500).json({ error: "Erreur serveur." });
        }

        const orgId = orgResult.insertId; // Récupérer l'ID de l'organisation insérée

        // Insérer dans la table diriger
        const insertDirigerQuery = `
          INSERT INTO diriger (idUser, idOrga, idRole, proprietaire)
          VALUES (?, ?, ?, ?)
        `;
        dbConnection.query(insertDirigerQuery, [idUser, orgId, 0, 1], (err) => {
          if (err) {
            console.error("Erreur lors de l'insertion dans diriger :", err);
            return res.status(500).json({ error: "Erreur serveur." });
          }

          // Réponse après succès
          res.status(201).json({ message: "Organisation créée avec succès et utilisateur associé comme propriétaire !" });
        });
      });
    });
  } catch (error) {
    console.error("Erreur lors de la création de l'organisation:", error);
    res.status(500).json({ error: "Erreur serveur." });
  }
});
```

L'interface permet à l'utilisateur de créer une nouvelle organisation en remplissant un formulaire. Le formulaire est organisé dans une carte stylisée avec des coins arrondis et une ombre pour donner un effet de profondeur. L'utilisateur doit renseigner plusieurs champs : le nom de l'organisation, une description, une image (sous forme d'URL), la ville, l'adresse, le code postal, et l'adresse mail. Chaque champ est clairement étiqueté et dispose d'un espace pour saisir les informations. Les champs de saisie sont conçus pour accepter un texte limité, avec des champs comme la description étant multiligne pour permettre des entrées plus longues. Un bouton "Créer mon organisation" permet de soumettre les informations saisies. En bas de l'écran, un menu de navigation est présent. L'interface est épurée et bien structurée pour offrir une expérience utilisateur fluide.

La route /infoOrganisation/:idOrganisation permet d'afficher les informations d'une organisation créée par l'utilisateur. Elle récupère l'ID utilisateur et l'ID de l'organisation pour afficher les données depuis la base de données.

```
router.get("/infoOrga/:id", verifyToken, (req, res) => {
  const userId = req.user?.id; // Récupération de l'ID de l'utilisateur via le token
  const idOrga = req.params.id; // Récupération de l'ID de l'organisation depuis l'URL

  if (!userId) {
    console.error("User ID not found in req.user");
    return res.status(400).json({ error: "Utilisateur non authentifié" });
  }

  if (!idOrga) {
    return res.status(400).json({ error: "ID de l'organisation requis" });
  }

  // Requête SQL pour récupérer l'organisation spécifique dont l'utilisateur est dirigeant
  const query = `
    SELECT o.id AS orga_id, o.nom, o.description, o.image, o.ville, o.adresse, o.cp, o.mail
    FROM diriger d
    JOIN organisation o ON d.idOrga = o.id
    WHERE d.idUser = ? AND o.id = ?
  `;

  dbConnection.query(query, [userId, idOrga], (err, results) => {
    if (err) {
      console.error("Erreur de la base de données :", err);
      return res.status(500).json({ error: "Erreur lors de la récupération des données de l'organisation" });
    }

    if (results.length === 0) {
      return res.status(404).json({ error: "Aucune organisation trouvée pour cet utilisateur avec cet ID" });
    }

    // Retourner les informations de l'organisation spécifique
    res.json(results[0]);
  });
});
```

L'interface présente les informations détaillées d'une organisation de manière claire et structurée. En haut de l'écran, un en-tête avec des boutons de navigation permet à l'utilisateur d'accéder à d'autres parties de l'application. L'écran affiche d'abord une grande image en arrière-plan, qui représente visuellement l'organisation. Cette image peut être remplacée par une image par défaut si celle de l'organisation n'est pas disponible. En dessous de l'image, le nom de l'organisation est mis en avant dans une carte avec un style distinctif. Ensuite, plusieurs cartes affichent des informations essentielles telles que la description de l'organisation, son adresse, et son adresse e-mail. Ces informations sont organisées de manière lisible et esthétique. Un bouton d'édition permet à l'utilisateur de modifier les informations de l'organisation, avec une icône en forme de crayon. Tout au long de l'écran, des éléments sont disposés de manière équilibrée pour assurer une expérience utilisateur fluide. Enfin, un menu de navigation situé en bas de l'écran permet d'accéder à d'autres options liées à l'organisation.

La route /modificationProfilOrganisation permet à un utilisateur de modifier les informations de son organisation. L'ID utilisateur est récupéré via le middleware et utilisé pour identifier l'organisation qu'il possède avant d'effectuer la mise à jour en base de données.

```
router.put("/modificationProfilOrga", verifyToken, (req, res) => {
  const userId = req.user.id; //ID de l'utilisateur connecté depuis le middleware
  console.log(userId)
  const { nom, description, image, ville, adresse, cp, mail } = req.body;

  if (!nom || !description || !image || !ville || !adresse || !cp || !mail) {
    return res.status(400).json({ error: "Tous les champs sont obligatoires." });
  }

  // Requête pour récupérer l'orgId lié à l'utilisateur
  const query = "SELECT idOrga FROM diriger WHERE idUser = ?";
  dbConnection.query(query, [userId], (err, results) => {
    if (err) {
      console.error("Erreur SQL :", err);
      return res.status(500).json({ error: "Erreur lors de la récupération de l'organisation." });
    }

    if (results.length === 0) {
      return res.status(404).json({ error: "Aucune organisation trouvée pour cet utilisateur." });
    }

    const orgId = results[0].idOrga;

    console.log(orgId);
    console.log("Mise à jour de l'organisation avec les données suivantes :", { nom, description, image, ville, adresse, cp, mail, orgId });

    // Mettre à jour l'organisation
    const updateQuery = `
      UPDATE organisation
      SET nom = ?, description = ?, image = ?, ville = ?, adresse = ?, cp = ?, mail = ?
      WHERE id = ?
    `;

    dbConnection.query(updateQuery, [nom, description, image, ville, adresse, cp, mail, orgId], (err, updateResults) => {
      if (err) {
        console.error("Erreur lors de la mise à jour :", err);
        return res.status(500).json({ error: "Erreur lors de la mise à jour de l'organisation." });
      }

      if (updateResults.affectedRows === 0) {
        return res.status(404).json({ error: "Organisation non trouvée." });
      }

      res.json({ message: "Organisation mise à jour avec succès." });
    });
  });
});
```

L'interface de l'écran "Modifier le Profil de l'Organisation" permet à l'utilisateur de mettre à jour les informations de son organisation. En haut de l'écran, un en-tête avec des boutons de navigation permet de revenir à l'écran précédent ou de recharger la page. Le formulaire est organisé en une série de champs de saisie, chacun permettant de modifier différentes informations de l'organisation, telles que le nom, la description, l'image, la ville, l'adresse, le code postal et l'adresse email. Chaque champ est clairement étiqueté, et les utilisateurs peuvent y entrer des informations texte. Les champs sont présentés dans un cadre avec un fond clair, des bords arrondis et une ombre légère pour un effet de profondeur. Sous les champs de saisie, un bouton qui permet à l'utilisateur de : mettre à jour les informations de l'organisation. Ce bouton est visuellement bien distinct grâce à sa taille et sa couleur. L'ensemble de l'interface est conçu de manière propre et fonctionnelle, avec des éléments espacés pour une meilleure lisibilité et une expérience utilisateur fluide.

De la même manière, la route /detailsOrganisation/:idOrganisation affiche les informations d'une organisation à laquelle l'utilisateur n'est pas propriétaire mais souhaite consulter les détails.

```
router.get("/detailsOrga/:id", verifyToken, (req, res) => {
  const idOrga = req.params.id;
  console.log("ID Orga reçu par l'API :", idOrga);
  const userId = req.user?.id;

  console.log("ID Organisation:", idOrga);
  console.log("ID Utilisateur:", userId);

  if (!userId) {
    return res.status(400).json({ error: "Utilisateur non authentifié" });
  }

  // Requête principale pour récupérer les infos de l'organisation
  const queryOrga = `
    SELECT id AS orga_id, nom, description, image, ville, adresse, cp, mail, code_couleur
    FROM organisation
    WHERE id = ?
  `;

  // Requête pour vérifier si l'utilisateur est dans la table "diriger"
  const queryMembership = `
    SELECT proprietaire
    FROM diriger
    WHERE idUser = ? AND idOrga = ?
  `;

  dbConnection.query(queryOrga, [idOrga], (err, orgaResults) => {
    if (err) {
      console.error("Erreur SQL :", err);
      return res.status(500).json({ error: "Erreur lors de la récupération des données." });
    }

    if (orgaResults.length === 0) {
      return res.status(404).json({ error: "Organisation non trouvée." });
    }

    const organisation = orgaResults[0];

    dbConnection.query(queryMembership, [userId, idOrga], (err, memberResults) => {
      if (err) {
        console.error("Erreur SQL :", err);
        return res.status(500).json({ error: "Erreur lors de la vérification de l'adhésion." });
      }

      const estMembre = memberResults.length > 0; // Vrai si l'utilisateur est dans "diriger"
      const estProprietaire = estMembre && memberResults[0].proprietaire === 1;

      // Ajouter une clé "peutRejoindre" si l'utilisateur N'EST PAS membre
      organisation.peutRejoindre = !estMembre;

      res.json(organisation);
    });
  });
});
```

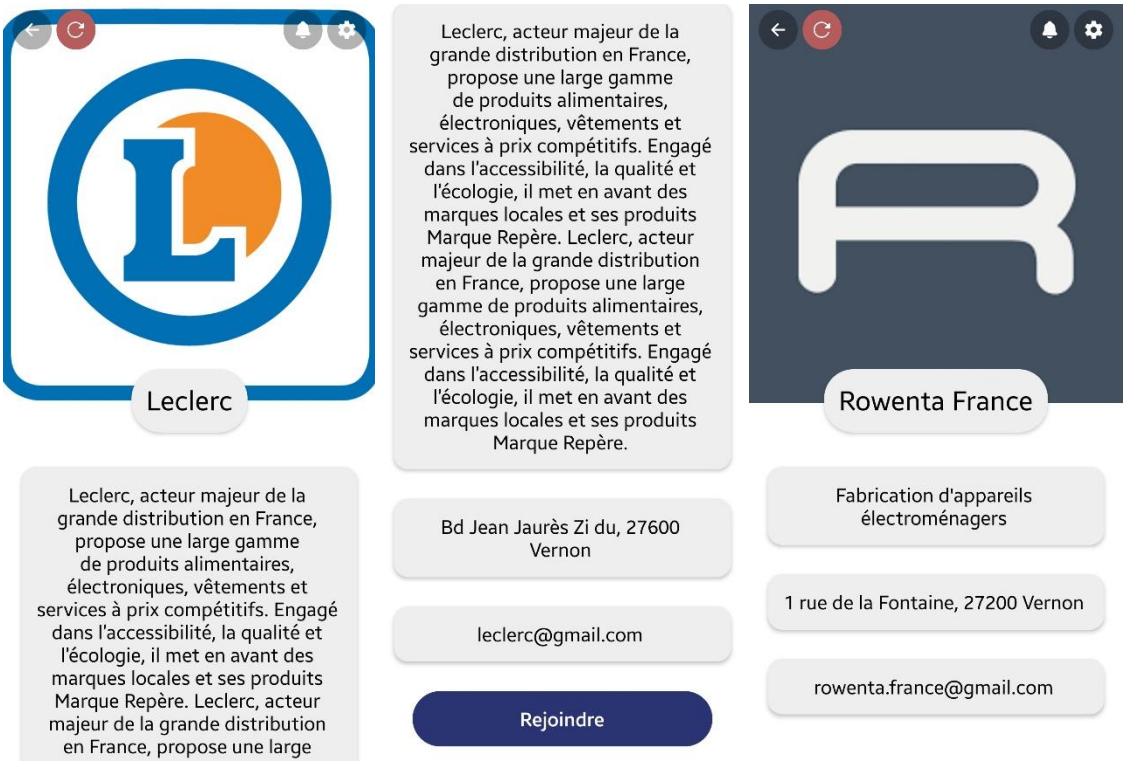
La route /demandeRejoindreOrganisation permet à un utilisateur d'envoyer une demande pour rejoindre une organisation. L'ID de l'utilisateur et de l'organisation sont récupérés pour vérifier si une demande existe déjà. Si ce n'est pas le cas, elle est enregistrée en base de données.

```
router.post('/demandeRejoindreOrga', verifyToken, (req, res) => {
  const { idOrga } = req.body;
  const userId = req.user?.id;

  if (!userId) {
      return res.status(401).json({ error: "Utilisateur non authentifié" });
  }

  // Vérifier si une demande existe déjà
  const checkQuery = `
      SELECT id FROM demande
      WHERE idUser = ? AND idOrga = ?
  `;

  dbConnection.query(checkQuery, [userId, idOrga], (err, result) => {
      if (err) {
          console.error("Erreur SQL :", err);
          return res.status(500).json({ error: "Erreur lors de la vérification de la demande." });
      }

      if (result.length > 0) {
          return res.status(400).json({ error: "Une demande est déjà en attente." });
      }

      // Si aucune demande en attente, on l'insère
      const insertQuery = `
          INSERT INTO demande (idUser, idOrga)
          VALUES (?, ?)
      `;

      dbConnection.query(insertQuery, [userId, idOrga], (err) => {
          if (err) {
              console.error("Erreur SQL :", err);
              return res.status(500).json({ error: "Erreur lors de l'envoi de la demande." });
          }

          res.status(201).json({ message: "Demande envoyée avec succès." });
      });
  });
});
```

L'interface affiche de manière claire et structurée les informations détaillées d'une organisation. En haut de l'écran, une image de couverture représentant l'organisation occupe une grande partie de l'espace. Cette image est dynamique et peut être changée en fonction de l'URL fournie. En dessous de l'image, un bloc contient le nom de l'organisation, affiché en grand et centré. Ensuite, un ensemble de cartes présente la description de l'organisation, son adresse complète (y compris le code postal et la ville), et son adresse e-mail. Ces informations sont présentées de manière claire avec un fond léger pour une meilleure lisibilité. Le design est épuré et bien organisé, avec une navigation facilitée grâce à un menu en bas de l'écran, permettant d'accéder rapidement à d'autres fonctionnalités de l'application. L'ensemble de l'interface est conçu pour offrir une expérience utilisateur fluide et agréable.

L'interface présente les informations détaillées d'une organisation spécifique. En haut de l'écran, on trouve une barre de navigation avec des boutons permettant à l'utilisateur de revenir à l'écran précédent ou de recharger les informations de l'organisation. L'image de couverture de l'organisation occupe une grande partie de l'écran, offrant une vue d'ensemble esthétique de l'entité. Sous cette image, le nom de l'organisation est affiché dans une carte avec un fond léger, suivi d'une autre carte contenant une description détaillée de l'organisation. Plus bas, on retrouve l'adresse complète de l'organisation ainsi que son adresse email, également affichés dans des

cartes individuelles. Si l'utilisateur n'est pas déjà membre de l'organisation et que l'option est disponible, un bouton "Rejoindre" s'affiche, permettant de faire une demande pour rejoindre l'organisation. L'interface est conçue de manière claire et structurée, avec des éléments visuels soignés tels que des cartes, des boutons stylisés et un arrière-plan harmonieux pour rendre l'expérience utilisateur fluide et agréable. Lorsque les données sont en cours de chargement, un indicateur de chargement est visible pour informer l'utilisateur.



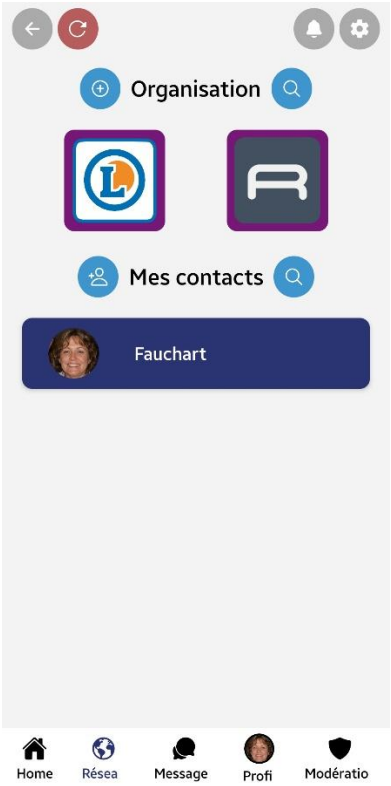
La route /recuperationMesOrganisation récupère les organisations dont l'utilisateur est propriétaire. Grâce au middleware, l'ID utilisateur est extrait et une requête SQL filtre les organisations où la valeur propriétaire est 1.

```
router.get('/recupMesOrga', verifyToken, (req, res) => {
  const idUser = req.user.id; // L'ID de l'utilisateur extrait du token

  // Requête SQL pour récupérer les organisations dont l'utilisateur est propriétaire
  const query = `
    SELECT o.*
    FROM organisation o
    JOIN diriger d ON o.id = d.idOrga
    WHERE d.idUser = ? AND d.proprietaire = 1;
  `;

  // Exécution de la requête
  dbConnection.query(query, [idUser], (err, results) => {
    if (err) {
      console.error('Erreur lors de la récupération des organisations :', err);
      return res.status(500).json({ message: 'Erreur serveur', error: err.message });
    }

    // Si l'utilisateur est propriétaire de certaines organisations
    if (results.length > 0) {
      return res.json(results); // Retourner les organisations
    } else {
      return res.status(404).json({ message: 'Aucune organisation trouvée pour cet utilisateur.' });
    }
  });
});
```

De même, la route /recuperationAutreOrganisation permet à un utilisateur de récupérer les organisations dont il est membre et non propriétaire, en sélectionnant celles où propriétaire est égale à 0.

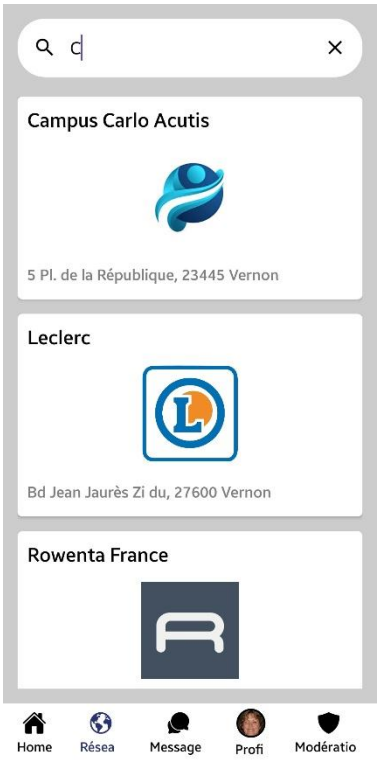
```
router.get('/recupAutreOrga', verifyToken, (req, res) => {
  const idUser = req.user.id; // L'ID de l'utilisateur extrait du token

  // Requête SQL pour récupérer les organisations dont l'utilisateur est propriétaire
  const query = `
    SELECT o.*
    FROM organisation o
    JOIN diriger d ON o.id = d.idOrga
    WHERE d.idUser = ? AND d.proprietaire = 0;
  `;

  // Exécution de la requête
  dbConnection.query(query, [idUser], (err, results) => {
    if (err) {
      console.error('Erreur lors de la récupération des organisations :', err);
      return res.status(500).json({ message: 'Erreur serveur', error: err.message });
    }

    // Si l'utilisateur est propriétaire de certaines organisations
    if (results.length > 0) {
      return res.json(results); // Retourner les organisations
    } else {
      return res.status(404).json({ message: 'Aucune organisation trouvée pour cet utilisateur.' });
    }
  });
});
```

Les deux routes précédentes permettent d'afficher sur l'interface du réseau les organisations créées par l'utilisateur connecté ainsi que celles qu'il a rejointes.



La route /rechercheOrganisation permet de rechercher une organisation en fonction d'un mot-clé saisi par l'utilisateur. Une requête SQL avec LIKE permet de récupérer toutes les organisations dont le nom correspond à la recherche.

```
router.get("/rechercheOrga", verifyToken, (req, res) => {
  const keyword = req.query.keyword; // Mot-clé envoyé depuis le frontend

  if (!keyword) {
    return res.status(400).json({ error: "Un mot-clé est requis pour effectuer la recherche." });
  }

  // Construire la requête pour rechercher les entreprises correspondant au mot-clé
  const query = `
    SELECT id, nom, description, image, ville, adresse, cp, mail
    FROM organisation
    WHERE nom LIKE ?
  `;

  const searchTerm = `%${keyword}%`; // Ajouter les jokers pour la recherche partielle

  dbConnection.query(query, [searchTerm, searchTerm, searchTerm], (err, results) => {
    if (err) {
      console.error("Erreur SQL :", err);
      return res.status(500).json({ error: "Erreur lors de la récupération des données." });
    }

    if (results.length === 0) {
      return res.status(404).json({ error: "Aucune organisation trouvée." });
    }

    res.json(results); // Retourner toutes les organisations trouvées
  });
});
```



L'interface de l'écran "Rechercher une organisation" permet à l'utilisateur de rechercher facilement des organisations en utilisant une barre de recherche située en haut de l'écran. Cette barre de recherche est équipée d'un champ où l'utilisateur peut entrer un mot-clé pour trouver une organisation spécifique. Dès que l'utilisateur tape dans le champ, les résultats s'affichent sous forme de liste dynamique. Si des organisations sont trouvées, chaque élément de la liste montre le nom de l'organisation, son adresse, et éventuellement une image de celle-ci. Lorsqu'un élément de la liste est sélectionné, l'utilisateur est redirigé vers une page de détails de l'organisation. Si aucune organisation n'est trouvée ou en cas d'erreur de connexion, un message d'erreur est affiché sous la barre de recherche. La liste des résultats est présentée de manière claire, chaque élément étant un bloc cliquable avec une légère ombre pour améliorer la lisibilité et l'interaction. En bas de l'écran, un menu de navigation est également présent, offrant des options supplémentaires pour l'utilisateur.

La route /demandes permet de récupérer toutes les demandes d'adhésion reçues par un utilisateur propriétaire d'une organisation, ainsi que les demandes de publication de poste soumises par les membres.

```
router.get("/demandes", verifyToken, (req, res) => {
  const userId = req.user?.id; // Récupère l'ID de l'utilisateur depuis le token

  if (!userId) {
      return res.status(400).json({ error: "Utilisateur non authentifié" });
  }

  // Vérifier si l'utilisateur est propriétaire d'une organisation
  const query = `
      SELECT d.idOrga
      FROM diriger d
      WHERE d.idUser = ? AND d.proprietaire = ?
  `;

  dbConnection.query(query, [userId, 1], (err, orgaResults) => {
      if (err) {
          console.error("Erreur SQL :", err);
          return res.status(500).json({ error: "Erreur lors de la récupération des organisations." });
      }

      if (orgaResults.length === 0) {
          return res.status(403).json({ error: "Vous n'êtes propriétaire d'aucune organisation." });
      }

      // Extraire les idOrga sous forme de tableau
      const idOrgas = orgaResults.map(orga => orga.idOrga);

      // Récupérer les demandes pour ces organisations ET les demandes de poste
      const demandeQuery = `
          SELECT d.id, d.idUser, d.idOrga, d.idPoste, u.nom, u.prenom, u.email, p.titre AS posteTitre, d.texte AS demandeTexte
          FROM demande d
          JOIN user u ON d.idUser = u.id
          LEFT JOIN poste p ON d.idPoste = p.id
          WHERE d.idOrga IN (${idOrgas.map(() => "?").join(",")})
      `;

      dbConnection.query(demandeQuery, idOrgas, (err, demandesResults) => {
          if (err) {
              console.error("Erreur SQL :", err);
              return res.status(500).json({ error: "Erreur lors de la récupération des demandes." });
          }

          res.json(demandesResults); // Retourner les demandes (d'adhésion ou de poste)
      });
  });
});
```

La route /demandeAjouterPoste permet à un utilisateur de faire une demande pour rejoindre un poste dans une organisation. L'utilisateur envoie une requête avec l'ID de l'organisation, et si aucune demande en attente n'existe déjà, elle est insérée dans la base de données. Un message de succès est ensuite renvoyé pour confirmer que la demande a bien été envoyée.

```
router.post('/demandeAjoutPoste', verifyToken, (req, res) => {
    const { idOrga } = req.body;
    const userId = req.user?.id;

    if (!userId) {
        return res.status(401).json({ error: "Utilisateur non authentifié" });
    }

    // Si aucune demande en attente, on l'insère
    const insertQuery = `
        INSERT INTO demande (idUser, idOrga, idPoste)
        VALUES (?, ?, ?)
    `;

    dbConnection.query(insertQuery, [userId, idOrga], (err) => {
        if (err) {
            console.error("Erreur SQL :", err);
            return res.status(500).json({ error: "Erreur lors de l'envoi de la demande." });
        }

        res.status(201).json({ message: "Demande envoyée avec succès." });
    });
});
```

Enfin, la route /accepteDemandePoste permet d'accepter une demande d'ajout à un poste au sein d'une organisation. Elle prend l'ID de la demande, met à jour l'état du poste à actif (etatPoste = 1), puis supprime la demande de la base de données. En cas de succès, un message de confirmation est renvoyé. Si une erreur survient lors du processus, un message d'erreur est renvoyé à l'utilisateur.

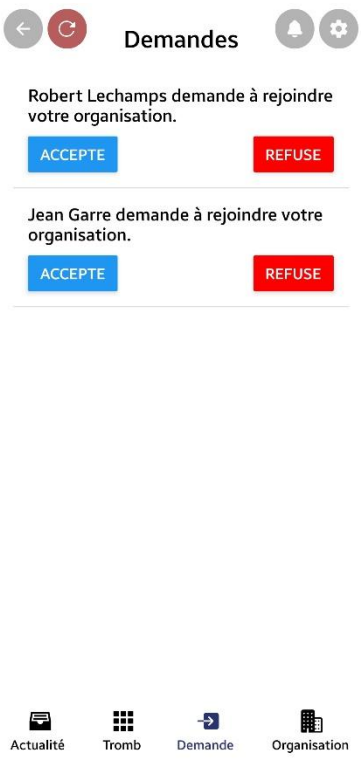
```
router.post("/accepteDemandePoste", verifyToken, async (req, res) => {
    try {
        const { idDemande } = req.body;

        if (!idDemande) {
            return res.status(400).json({ error: "L'ID de la demande est requis." });
        }

        // Mettre à jour etatPoste à 1 dans la table `poste`
        const updateQuery = `
            UPDATE poste
            SET etatPoste = 1
            WHERE id IN (SELECT idPoste FROM demande WHERE id = ?)
        `;

        dbConnection.query(updateQuery, [idDemande], (err, result) => {
            if (err) {
                console.error("Erreur SQL :", err);
                return res.status(500).json({ error: "Erreur lors de la mise à jour du poste." });
            }

            if (result.affectedRows === 0) {
                return res.status(404).json({ error: "Demande introuvable ou déjà traitée." });
            }

            // Suppression de la demande après l'acceptation
            const deleteQuery = `DELETE FROM demande WHERE id = ?`;
            dbConnection.query(deleteQuery, [idDemande], (err) => {
                if (err) {
                    console.error("Erreur lors de la suppression de la demande :", err);
                    return res.status(500).json({ error: "Erreur lors de la suppression de la demande." });
                }

                res.status(200).json({ success: true, message: "Demande acceptée et poste activé." });
            });
        });

    } catch (error) {
        console.error("Erreur serveur :", error);
        res.status(500).json({ error: "Erreur interne du serveur." });
    }
});
```

La route /accepteDemandeRejoindreOrganisation permet d'accepter une demande d'adhésion. L'idDemande est vérifié en base de données et, si elle existe, l'utilisateur est ajouté à l'organisation avec un rôle par défaut. Une fois l'ajout réussi, la demande est supprimée de la base de données. En cas d'échec, un message d'erreur est retourné.

```
router.post('/accepteDemandeRejoindreOrga', (req, res) => {
  const { idDemande } = req.body;
  console.log("ID Demande reçu :", idDemande);

  // Récupération de la demande
  const fetchQuery = 'SELECT * FROM demande WHERE id = ?';

  dbConnection.query(fetchQuery, [idDemande], (fetchErr, fetchResult) => {
      if (fetchErr || fetchResult.length === 0) {
          res.status(404).send('Demande non trouvée');
      } else {
          const { idUser, idOrga } = fetchResult[0];

          // Ajouter l'utilisateur à l'organisation
          const insertQuery = 'INSERT INTO diriger (idUser, idOrga, idRole, proprietaire) VALUES (?, ?, ?, ?)';
          dbConnection.query(insertQuery, [idUser, idOrga, 0, 0], (insertErr) => {
              if (insertErr) {
                  res.status(500).send('Erreur lors de l\'ajout de l\'utilisateur à l\'organisation');
              } else {
                  // Supprimer la demande après acceptation
                  const deleteQuery = 'DELETE FROM demande WHERE id = ?';
                  dbConnection.query(deleteQuery, [idDemande], (deleteErr) => {
                      if (deleteErr) {
                          res.status(500).send('Erreur lors de la suppression de la demande');
                      } else {
                          res.status(200).send('Demande acceptée et utilisateur ajouté à l\'organisation');
                      }
                  });
              }
          });
      }
  });
});
```

Enfin, la route /refuserDemandeRejoindreOrganisation permet de refuser une demande d'adhésion. Elle commence par vérifier que l'idDemande est bien fourni, puis exécute une requête SQL pour la supprimer de la base de données. En cas de succès, un message de confirmation est envoyé à l'utilisateur, sinon un message d'erreur est retourné.

```
router.post("/refuserDemandeRejoindreOrga", verifyToken, (req, res) => {
  try {
      const { idDemande } = req.body;

      if (!idDemande) {
          console.log("Requête invalide : ID de la demande manquant");
          return res.status(400).json({ error: "L'ID de la demande est requis." });
      }

      console.log("Tentative de suppression de la demande avec ID :", idDemande);
      const deleteQuery = "DELETE FROM demande WHERE id = ?";

      dbConnection.query(deleteQuery, [idDemande], (err, result) => {
          if (err) {
              console.error("Erreur SQL lors de la suppression de la demande :", err);
              return res.status(500).json({ error: "Erreur lors du refus de la demande." });
          }
          console.log("Demande supprimée avec succès, ID:", idDemande);
          res.status(200).json({ message: "Demande refusée avec succès." });
      });
  } catch (error) {
      console.error("Erreur inattendue lors du traitement de la requête :", error);
      res.status(500).json({ error: "Erreur serveur." });
  }
});
```



L'interface affiche une liste de demandes en attente d'approbation, divisées en demandes pour rejoindre une organisation ou ajouter un poste. En haut de l'écran, un en-tête permet à l'utilisateur de revenir à l'écran précédent ou de recharger la liste des demandes. Si aucune demande n'est présente, un message indique que l'utilisateur n'a aucune demande à traiter. Si des demandes existent, elles sont listées sous forme d'éléments avec des informations sur la personne qui a fait la demande et le type de demande (organisation ou poste). Pour chaque demande, deux boutons sont proposés : "Accepter" et "Refuser". L'utilisateur peut ainsi prendre des décisions pour chaque demande. Si une erreur se produit lors du chargement des données ou lors de l'interaction avec une demande, un message d'erreur est affiché. L'interface est également équipée d'une barre de navigation pour faciliter la navigation entre les écrans.

Dans le fichier posteRoute.js, plusieurs routes sont définies pour gérer les postes au sein de l'application. La route /recuperationPoste permet de récupérer les postes, qu'ils soient généraux ou spécifiques à une organisation, selon les paramètres fournis. Si un identifiant d'organisation est passé en paramètre, seuls les postes associés à cette organisation sont retournés. Si aucun identifiant d'organisation n'est spécifié, tous les postes généraux sont récupérés. En plus des informations sur chaque post, la requête SQL compte également le nombre de "likes" pour chaque post et récupère des informations utilisateur, comme l'avatar. Le résultat inclut également une vérification pour savoir si l'utilisateur connecté peut supprimer chaque post.

```
router.get('/recupPoste', verifyToken, (req, res) => {
    try {
        const user_id = req.user.id;
        const orgId = req.query.orgId; // Paramètre optionnel pour filtrer les posts par organisation
        let sql;

        if (orgId) {
            // Récupérer les posts d'une organisation spécifique
            sql = `
                SELECT p.id, p.titre, p.description, p.images, p.date_publication,
                       (SELECT COUNT(*) FROM likes WHERE post_id = p.id) AS nombre_like,
                       u.id as user_id, u.nom as user_nom, u.photoProfil as user_avatar
                FROM poste p
                JOIN user u ON p.user_id = u.id
                WHERE p.org_id = ?
                AND etatPoste = 1
                ORDER BY p.date_publication DESC
            `;
        } else {
            // Récupérer tous les posts généraux
            sql = `
                SELECT p.id, p.titre, p.description, p.images, p.date_publication,
                       (SELECT COUNT(*) FROM likes WHERE post_id = p.id) AS nombre_like,
                       u.id as user_id, u.nom as user_nom, u.photoProfil as user_avatar
                FROM poste p
                JOIN user u ON p.user_id = u.id
                WHERE p.org_id IS NULL
                AND etatPoste = 1
                ORDER BY p.date_publication DESC
            `;
        }

        dbConnection.query(sql, [orgId], (err, result) => {
            if (err) {
                return res.status(500).json({ error: "Erreur lors de la récupération des posts." });
            }

            const posts = result.map(post => ({
                ...post,
                canDelete: post.user_id === user_id
            }));

            res.status(200).json(posts);
        });
    } catch (error) {
        res.status(500).json({ error: "Erreur interne du serveur." });
    }
});
```

La route /like permet à un utilisateur de liker ou de retirer son like d'un post. La fonction commence par vérifier si l'utilisateur a déjà liké ce post. Si c'est le cas, le like est supprimé et le nombre de likes est mis à jour. Si ce n'est pas le cas, un like est ajouté et le nombre de likes est mis à jour en conséquence. Après chaque opération, un message est renvoyé pour indiquer si l'action a été réussie, avec le nombre de likes actuels.

```
router.post("/like", verifyToken, (req, res) => {
    const user_id = req.user.id;
    const { post_id } = req.body;

    if (!post_id) {
        return res.status(400).json({ error: "L'ID du post est requis." });
    }

    const checkLikeQuery = `SELECT * FROM likes WHERE user_id = ? AND post_id = ?`;
    dbConnection.query(checkLikeQuery, [user_id, post_id], (err, results) => {
        if (err) return res.status(500).json({ error: "Erreur SQL lors de la vérification du like." });

        if (results.length > 0) {
            // Supprimer le like
            const deleteLikeQuery = `DELETE FROM likes WHERE user_id = ? AND post_id = ?`;
            dbConnection.query(deleteLikeQuery, [user_id, post_id], (err) => {
                if (err) return res.status(500).json({ error: "Erreur lors de la suppression du like." });

                // Récupérer le nombre total de likes après suppression
                dbConnection.query(`SELECT COUNT(*) AS nombre_like FROM likes WHERE post_id = ?`, [post_id], (err, result) => {
                    if (err) return res.status(500).json({ error: "Erreur mise à jour des likes après suppression." });

                    return res.json({ success: true, message: "Like supprimé !", nombre_like: result[0].nombre_like });
                });
            });
        } else {
            // Ajouter un like
            const addLikeQuery = `INSERT INTO likes (user_id, post_id) VALUES (?, ?)`;
            dbConnection.query(addLikeQuery, [user_id, post_id], (err) => {
                if (err) return res.status(500).json({ error: "Erreur lors de l'ajout du like." });

                // Récupérer le nombre total de likes après ajout
                dbConnection.query(`SELECT COUNT(*) AS nombre_like FROM likes WHERE post_id = ?`, [post_id], (err, result) => {
                    if (err) return res.status(500).json({ error: "Erreur mise à jour des likes après ajout." });

                    return res.json({ success: true, message: "Like ajouté !", nombre_like: result[0].nombre_like });
                });
            });
        }
    });
});
```

La route /likes permet de récupérer le nombre de likes pour chaque post. Elle effectue une requête SQL pour compter les likes de chaque post et retourne les résultats sous forme d'un objet où chaque post_id est associé à son nombre de likes.

```
router.get("/likes", verifyToken, (req, res) => {
    const getLikesQuery = `SELECT post_id, COUNT(*) AS nombre_like FROM likes GROUP BY post_id`;

    dbConnection.query(getLikesQuery, (err, results) => {
        if (err) {
            return res.status(500).json({ error: "Erreur SQL lors de la récupération des likes." });
        }

        // Transformer les résultats en objet clé-valeur { post_id: nombre_like }
        const likesMap = {};
        results.forEach(row => {
            likesMap[row.post_id] = row.nombre_like;
        });

        res.json(likesMap);
    });
});
```

La route /supprimer/:id permet à un utilisateur de supprimer l'un de ses propres posts. Elle commence par vérifier si l'utilisateur est bien le propriétaire du post qu'il souhaite supprimer. Si ce n'est pas le cas, un message d'erreur est renvoyé. Si l'utilisateur est bien propriétaire, le post est supprimé et un message de succès est envoyé.

```
router.delete('/supprimerPoste/:id', verifyToken, async (req, res) => {
    try {
        const postId = req.params.id;
        const userId = req.user.id; // On récupère l'ID utilisateur depuis le token

        // Vérifier si le post appartient à l'utilisateur
        const sqlCheckPost = "SELECT * FROM poste WHERE id = ? AND user_id = ?";
        dbConnection.query(sqlCheckPost, [postId, userId], (err, result) => {
            if (err) {
                console.error("Erreur SQL :", err);
                return res.status(500).json({ error: "Erreur lors de la vérification du post." });
            }

            if (result.length === 0) {
                return res.status(403).json({ error: "Vous ne pouvez pas supprimer ce post." });
            }

            // Si le post appartient à l'utilisateur, on peut le supprimer
            const sqlDeletePost = "DELETE FROM poste WHERE id = ?";
            dbConnection.query(sqlDeletePost, [postId], (err, result) => {
                if (err) {
                    console.error("Erreur SQL :", err);
                    return res.status(500).json({ error: "Erreur lors de la suppression du post." });
                }

                return res.status(200).json({ success: true, message: "Post supprimé avec succès." });
            });
        });
    } catch (error) {
        console.error("Erreur serveur :", error);
        res.status(500).json({ error: "Erreur interne du serveur." });
    }
});
```

La route /organisationPoste/:idOrganisation permet de vérifier si un utilisateur est propriétaire d'une organisation en fonction de son ID. Une requête est exécutée pour vérifier l'association entre l'utilisateur et l'organisation avec un rôle de propriétaire. Si l'utilisateur est effectivement propriétaire, une réponse positive est renvoyée, sinon un message d'erreur est renvoyé.

```
router.get('/orgaPoste/:idOrga', verifyToken, async (req, res) => {
    const orgId = req.params.idOrga;
    const userId = req.user.id;

    try {
      // Vérification si l'utilisateur est propriétaire de l'organisation
      const query = `
        SELECT o.id AS orgId
        FROM organisation o
        JOIN diriger d ON d.idOrga = o.id
        WHERE d.idOrga = ? AND d.idUser = ? AND d.proprietaire = 1
      `;

      // Exécution de la requête SQL
      const results = dbConnection.query(query, [orgId, userId]);

      // Si la requête retourne un résultat (au moins une ligne)
      if (results.length === 0) {
        return res.status(404).json({ error: 'Organisation non trouvée ou vous n\'êtes pas le propriétaire.' });
      }

      const org = results[0];  // Prenez le premier objet de la réponse

      // Si l'utilisateur est le propriétaire, renvoyer les informations de l'organisation
      res.json({
        ownerId: userId,  // ID de l'utilisateur qui est le propriétaire
        isOwner: true,  // Signaler que l'utilisateur est le propriétaire
      });

    } catch (error) {
      console.error('Erreur serveur:', error);
      res.status(500).json({ error: 'Erreur serveur' });
    }
});
```

L'interface de l'écran "Actualités Générales" est conçue pour afficher une liste de postes dans un format fluide et interactif. En haut de l'écran, on trouve une barre de navigation avec des boutons permettant d'accéder à différentes sections. Chaque post est présenté dans un bloc bien délimité avec un fond clair, contenant des informations importantes comme l'avatar de l'utilisateur qui a créé le post, son nom, ainsi que la date de publication formatée. En dessous du titre du post, une image peut être affichée si elle est présente, suivie d'une description du contenu. Les utilisateurs peuvent interagir avec chaque post en aimant like ou en supprimant le post s'ils en ont l'autorisation. Le bouton like est représenté par une icône de pouce en haut ou bas, changeant de couleur en fonction de l'état du like. De plus, un bouton de suppression, accessible via un appui long, permet de retirer un post si l'utilisateur en est autorisé. Les postes sont affichés un à un dans une liste déroulante, avec une vue détaillée à chaque sélection. Un indicateur de chargement est visible pendant que les postes sont récupérés du serveur. Si une erreur se produit lors de la récupération des données, un message d'erreur s'affiche à l'utilisateur. Il y a aussi une page d'actualité pour les organisations créer et rejointe qui est la même que celle générale.

La route /creationPoste permet de créer un post général. L'utilisateur doit fournir un titre, une description, et éventuellement une image. La route vérifie que toutes les informations nécessaires sont présentes, puis insère-le post dans la base de données avec l'utilisateur comme auteur. Une fois le post ajouté, un message de succès est renvoyé, accompagné de l'ID généré du post.

```
router.post('/creationPoste', verifyToken, async (req, res) => {
    try {
        console.log("Requête reçue pour créer un post :", req.body);
        console.log("je suis serveur")

        const { titre, description, image } = req.body;
        const user_id = req.user.id; // On récupère l'ID utilisateur depuis le token

        if (!user_id || !titre || !description) {
            console.warn("Champs manquants :", { user_id, titre, description });
            return res.status(400).json({ error: "Tous les champs sont requis." });
        }

        console.log("Image reçue :", image || "Aucune image fournie");

        const sql = "INSERT INTO poste (user_id, titre, description, images, etatPoste) VALUES (?, ?, ?, ?, ?)";

        dbConnection.query(sql, [user_id, titre, description, image, 0], (err, result) => {
            if (err) {
                console.error("Erreur SQL :", err);
                return res.status(500).json({ error: "Erreur lors de l'ajout du post." });
            }

            console.log("Post ajouté avec succès - ID :", result.insertId);
            res.status(201).json({ success: true, message: "Post ajouté avec succès.", postId: result.insertId });
        });
    } catch (error) {
        console.error("Erreur serveur :", error);
        res.status(500).json({ error: "Erreur interne du serveur." });
    }
});
```

La route /creationPosteOrganisation/:idOrganisation permet à un utilisateur de créer un post spécifique à une organisation. L'utilisateur doit être authentifié et fournir un titre, une description, ainsi que l'ID de l'organisation. Si toutes les informations sont valides, le post est créé pour cette organisation avec l'ID de l'utilisateur, et un message de succès est renvoyé.

```
router.post("/creationPosteOrga/:idOrga", verifyToken, async (req, res) => {
    try {
        console.log("Requête reçue pour créer un post d'organisation :", req.body);

        const { titre, description, image } = req.body;
        const { idOrga } = req.params; // Récupération de l'id de l'organisation via l'URL
        const user_id = req.user.id; // Récupération de l'utilisateur depuis le token

        if (!user_id || !titre || !description || !idOrga) {
            console.warn("Champs manquants :", { user_id, titre, description, idOrga });
            return res.status(400).json({ error: "Tous les champs sont requis, y compris idOrga." });
        }

        console.log(`Création d'un post pour l'organisation ID: ${idOrga} par l'utilisateur ID: ${user_id}`);

        const sql = "INSERT INTO poste (user_id, titre, description, images, etatPoste, org_id) VALUES (?, ?, ?, ?, ?, ?)";

        dbConnection.query(sql, [user_id, titre, description, image, 0, idOrga], (err, result) => {
            if (err) {
                console.error("Erreur SQL :", err);
                return res.status(500).json({ error: "Erreur lors de l'ajout du post." });
            }

            console.log("Post ajouté avec succès - ID :", result.insertId);
            res.status(201).json({ success: true, message: "Post ajouté avec succès.", postId: result.insertId });
        });
    } catch (error) {
        console.error("Erreur serveur :", error);
        res.status(500).json({ error: "Erreur interne du serveur." });
    }
});
```

L'interface de l'écran "Ajouter un post" présente un formulaire simple et intuitif. En haut de l'écran, on trouve un en-tête avec des boutons de navigation permettant à l'utilisateur de revenir en arrière ou de recharger la page. Le formulaire lui-même est contenu dans un bloc avec un fond clair et des bords arrondis, accompagné d'une ombre pour ajouter un effet de profondeur. Ce bloc contient trois champs principaux : un pour le titre du post, un autre pour la description, et enfin un champ où l'utilisateur peut entrer l'URL d'une image. Chaque champ est étiqueté avec des textes clairs comme "Titre", "Description" et "Image (URL)", permettant à l'utilisateur de comprendre facilement ce qu'il doit remplir. La description peut être saisie sur plusieurs lignes grâce à un champ de texte multi-lignes. Sous ces champs, un bouton "Publier" permet à l'utilisateur de soumettre les informations. Le design est soigné, avec des bordures colorées et des espaces bien définis entre les éléments pour rendre l'expérience utilisateur agréable et fluide. Il y a aussi une page de création de poste pour les organisations créer qui ressemble à celle-ci

Cette barre de navigation située en bas de l'écran. Cette barre contient quatre boutons de navigation, chacun accompagné d'une icône et d'un texte. Les boutons permettent de naviguer vers différentes sections de l'application : "Actualités", "Trombi" (l'annuaire), "Demandes" et "Organisation". Lorsque l'utilisateur sélectionne un bouton, celui-ci est mis en évidence par un changement de couleur, signalant ainsi l'onglet actif. La barre de navigation est simple et intuitive, avec des icônes de type "Ionicons" représentant chaque section (comme un document pour les actualités, un trombone pour l'annuaire, etc.). En plus de cela, la barre est responsive et s'adapte à différentes plateformes (iOS et Android). En cas de clavier affiché à l'écran, la barre de navigation disparaît pour ne pas interférer avec l'interface utilisateur. En haut de l'écran, des informations sur l'utilisateur sont récupérées et stockées, bien que la section de profil ne soit pas visible directement dans cette interface. Quand nous l'allons dans une organisation qui nous appartient pas nous ne voyions pas l'icône demandes dans la barre de navigation

BILAN DE MON SUJET

Mon stage a consisté à développer une application mobile pour le GIRV de Vernon en utilisant React Native, avec une base de données gérée sous PHPMyAdmin et une communication en temps réel via Socket.io. L'application a pour objectif de mettre en relation les chefs d'entreprise en leur permettant de créer un profil, d'échanger via une messagerie intégrée et d'accéder aux événements organisés par le GIRV. Tout au long du projet, j'ai suivi plusieurs étapes : la mise en place de l'environnement de développement, la création de l'API, l'implémentation des principales fonctionnalités et l'optimisation de l'application. À la fin de mon stage, l'application était fonctionnelle dans sa grande majorité : les utilisateurs peuvent s'inscrire, gérer leur profil, échanger des messages en temps réel et consulter les événements. Seule la gestion de l'importation d'images reste à finaliser. Ce projet m'a permis d'acquérir des compétences solides en développement mobile, gestion de bases de données et communication en temps réel, tout en développant mon autonomie et ma capacité à gérer un projet de A à Z.

BILAN GENERAL DE MON STAGE

Ce stage m'a permis de travailler sur un projet concret en développant une véritable application répondant aux besoins du GIRV de Vernon. Grâce à cette expérience, j'ai renforcé mes compétences en React Native, en gestion de bases de données avec PHPMyAdmin et en communication entre le front-end et le back-end via une API. J'ai également appris à gérer l'hébergement d'une application, aussi bien en local qu'en ligne. Ce projet m'a confronté à différents défis techniques, ce qui m'a poussé à chercher des solutions par moi-même, à améliorer ma capacité d'adaptation et à affiner ma méthodologie de développement. Sur le plan personnel, ce stage m'a permis de gagner en autonomie, en rigueur et en confiance en moi, notamment pour aborder de futurs projets avec plus de sérénité. Cette expérience a été formatrice et enrichissante, tant sur le plan technique que professionnel, et elle me motive à approfondir encore mes connaissances dans le développement d'applications.