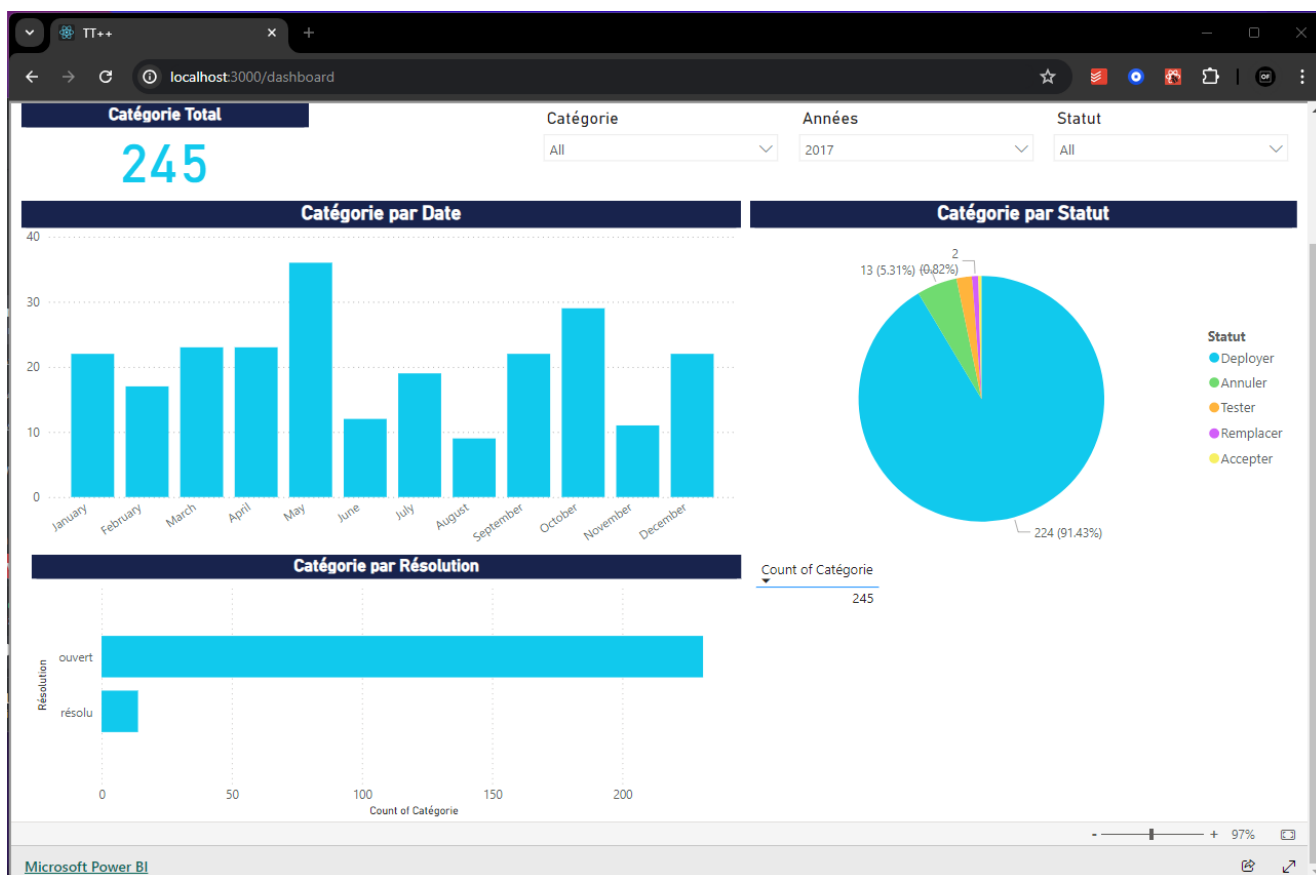


## Déclaration du Composant Dashboard

```
const Dashboard = () => {
  return (
    <div className="dashboard">
      <Navbar />
      <h1 className="dashboard-header">Dashboard</h1>
      <PowerBIEmbed
```



Fonctionnalité : Définir et retourner la structure du composant Dashboard avec une barre de navigation et un titre.

## Configuration de l'Intégration Power BI

```
embedConfig={{
  type: "report",
  id: "1bcc9648-be7f-408b-801a-7d3d1c08389b",
  embedUrl: "https://app.powerbi.com/view?r=eyJrIjoiaZjM5YmZmMWUtMjdjNy00Y2Q2LWI4ODctM2M0MzExOWI4ZjhhIiwidCI6ImRiZDY2NjRkLTRlYjktNDZlYi05OWQ4LTVjNDNiYTE1M2M2MSIsImMiOjI9",
  accessToken: token,
  tokenType: models.TokenType.Embed,
  settings: {
    panes: {
      filters: {
```

```

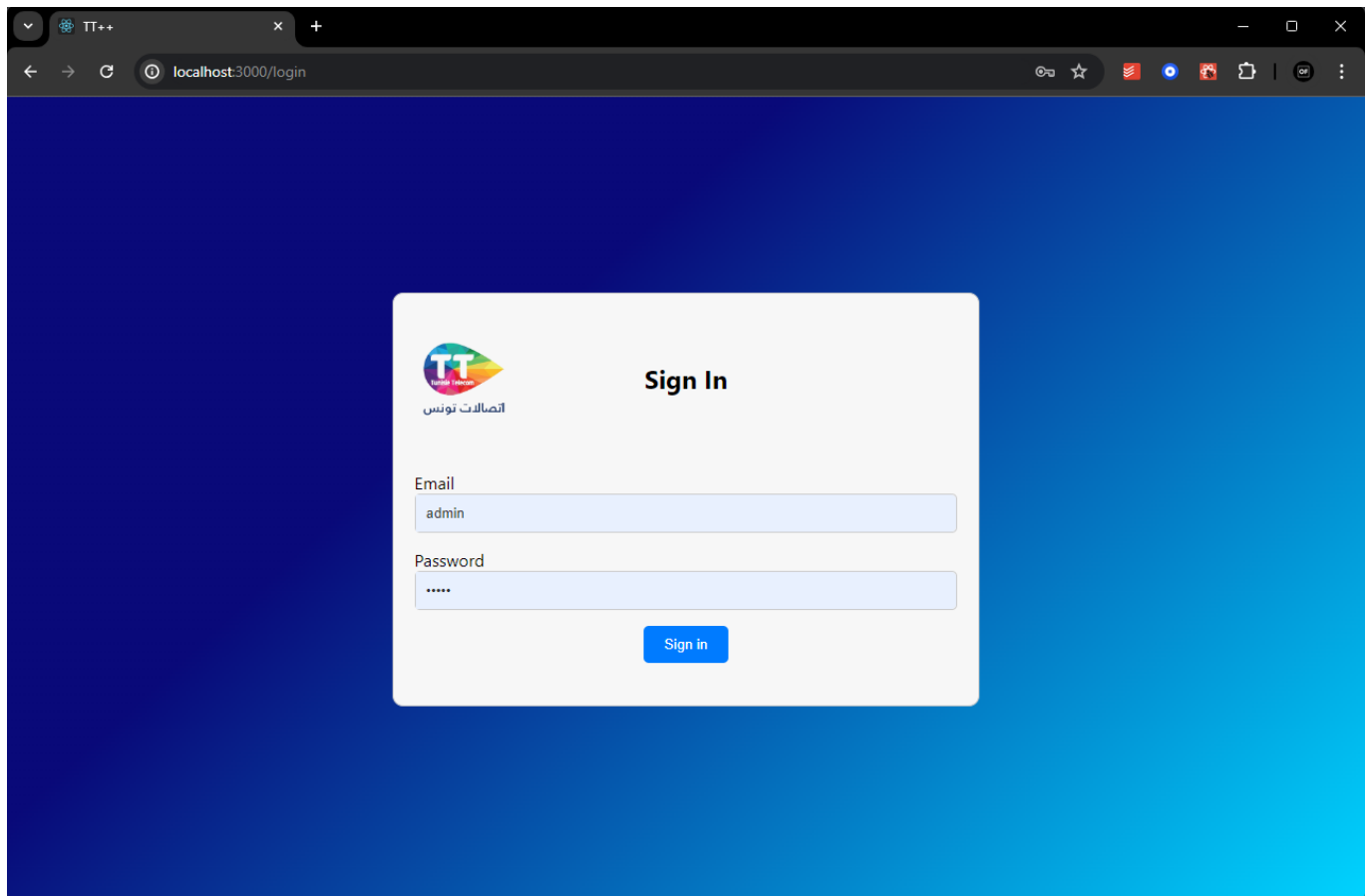
        expanded: false,
        visible: false,
      },
    },
    background: models.BackgroundColor.Transparent,
  },
}

```

Fonctionnalité : Configurer l'intégration du rapport Power BI, y compris l'ID du rapport, l'URL d'intégration, le jeton d'accès, le type de jeton et les paramètres d'affichage.

## Explication du Composant Login

Authentifier un utilisateur avec Firebase.



## Composants

```

const Login = () => {
  const [mail, setMail] = useState("");
  const [Password, setPassword] = useState("");
  const navigation = useNavigate();

```

Fonctionnalité: Déclarer les états mail et Password pour stocker les valeurs saisies par l'utilisateur et initialiser la fonction de navigation.

## Fonction de Connexion

```
try {
  await signInWithEmailAndPassword(auth, mail, Password).then(
    (userCredential) => {
      // Signed in
      const user = userCredential.user;
      navigation("/dashboard");
      console.log(user);
    }
  );
} catch (error) {
  alert("Bad Credential");
}
};
```

Fonctionnalité: Définir la fonction OnLogin pour authentifier l'utilisateur avec Firebase et rediriger vers le tableau de bord en cas de succès ou afficher une alerte en cas d'erreur.

## Initialisation de **Firebase**

Cette partie explique comment nous avons configuré et initialisé **Firebase** dans notre application JavaScript.

### Configuration Firebase

```
const firebaseConfig = {
  apiKey: "AIzaSyA40pH0xbYm2SgrzbKn1XxFghruj3A0YWw",
  authDomain: "telecom-5e921.firebaseio.com",
  projectId: "telecom-5e921",
  storageBucket: "telecom-5e921.appspot.com",
  messagingSenderId: "107097147337",
  appId: "1:107097147337:web:6294dd4a303c60ea5c8a18",
  measurementId: "G-605WVM26JW"
};
```

Fonctionnalité Nous avons défini l'objet firebaseConfig qui contient les informations de configuration nécessaires pour connecter notre application à Firebase. Ces informations incluent :

- apiKey: La clé API de notre projet Firebase.
- authDomain: Le domaine d'authentification de notre projet Firebase.
- projectId: L'ID de notre projet Firebase.
- storageBucket: Le bucket de stockage de notre projet Firebase.
- messagingSenderId: L'ID de l'expéditeur de messages de notre projet Firebase.

- appId: L'ID de l'application de notre projet Firebase.
- measurementId: L'ID de mesure de notre projet Firebase.

## Initialisation de Firebase

```
// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
const auth = getAuth(app);
```

Fonctionnalité: Nous avons initialisé Firebase avec la configuration définie précédemment.

- initializeApp(firebaseConfig): Initialise notre application Firebase avec la configuration fournie.
- getAnalytics(app): Initialise et obtient l'instance d'Analytics pour notre application.
- getAuth(app): Initialise et obtient l'instance d'authentification pour notre application.

## Parties Principales du Composant Gestion

### Gestion de l'État

Le composant utilise plusieurs variables d'état pour suivre les informations des employés, gérer les formulaires d'ajout et de modification, et contrôler les interactions utilisateur.

### Récupération des Employés

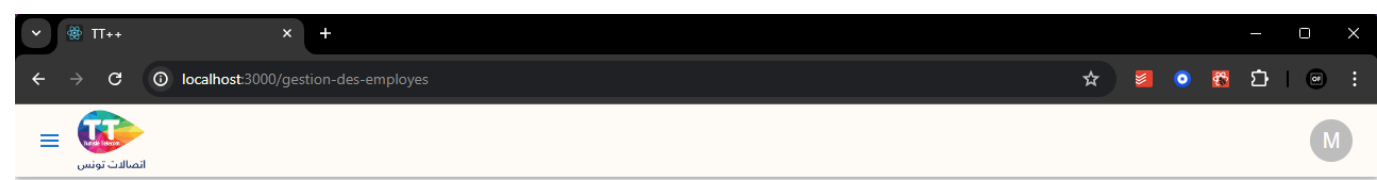
Lorsque le composant se charge, il récupère les données des employés à partir de la base de données Firestore et les stocke pour les afficher.

```
useEffect(() => {
  const getEmployees = async () => {
    const data = await getDocs(employeeCollectionRef);
    setEmployees(data.docs.map((doc) => ({ ...doc.data(), id: doc.id })));
  };

  getEmployees();
}, []);
```

### Onglets pour Voir et Ajouter des Employés

L'interface utilisateur utilise des onglets pour permettre aux utilisateurs de basculer entre la vue de la liste des employés et un formulaire pour ajouter un nouvel employé.



## Gestion des Employés

VOIR LES EMPLOYÉS    AJOUTER UN EMPLOYÉ

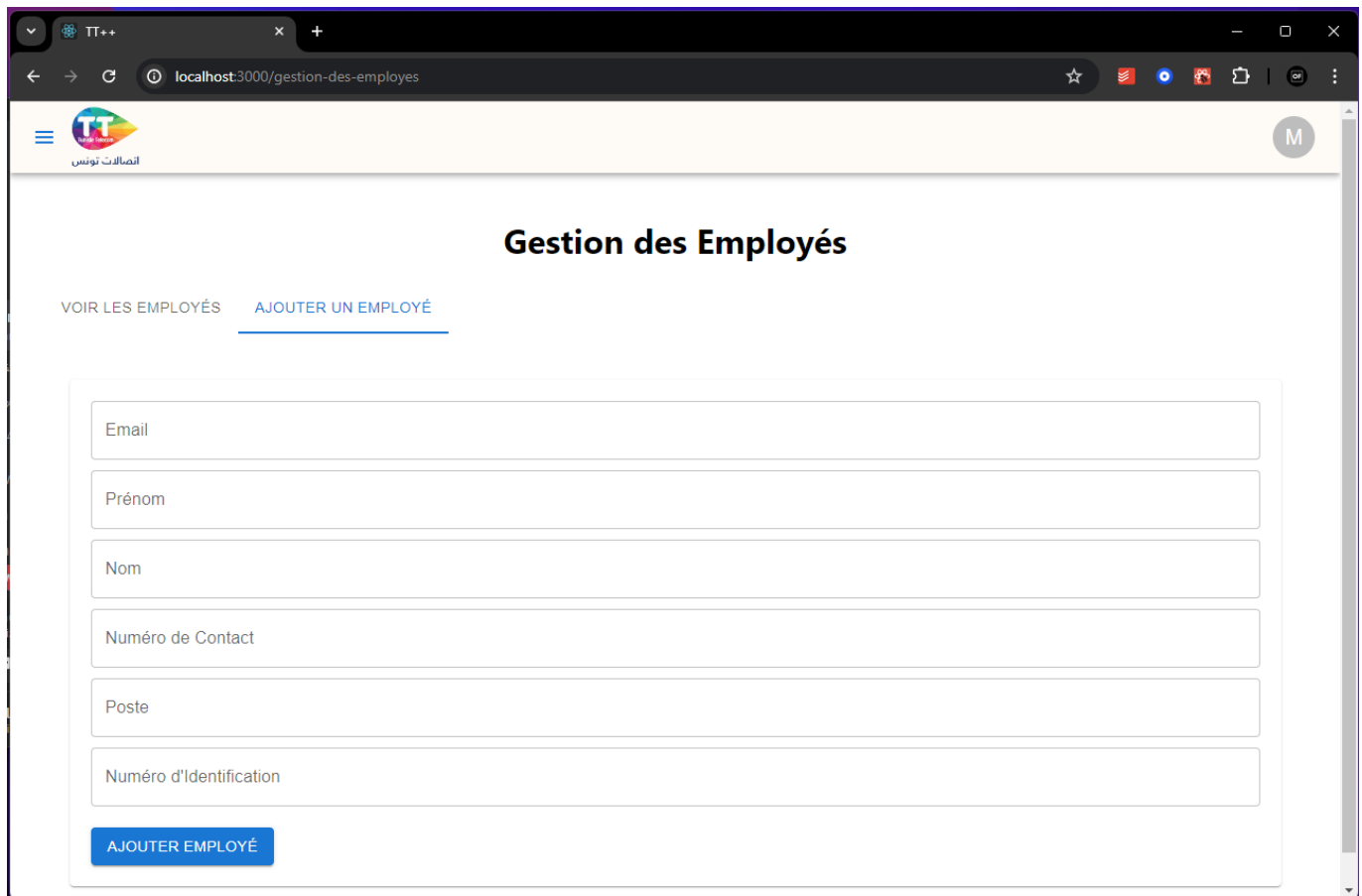
Email	Prénom	Nom	Numéro de Contact	Poste	Numéro d'identification	Actions	ENVOYER EMAIL
overflowtn@gmail.com	test	test	123	1234	123434	<div>MODIFIER</div> <div>SUPPRIMER</div>	<input type="checkbox"/>

```
<Tabs value={value} onChange={handleChange} aria-label="onglets de gestion des employés">
  <Tab label="Voir les Employés" />
  <Tab label="Ajouter un Employé" />
</Tabs>
```

## Actions sur les Employés

### - Création d'Employés

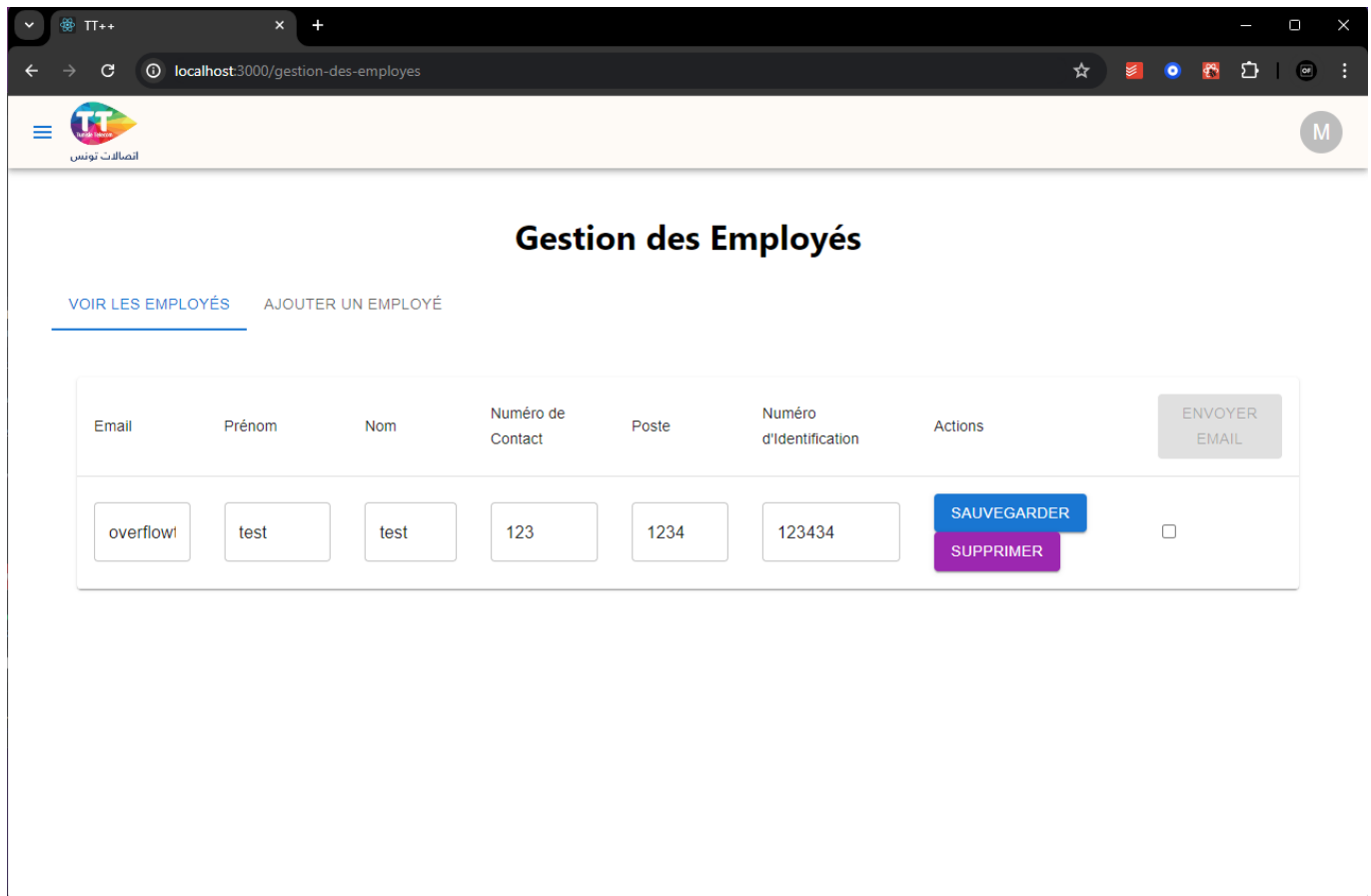
Les utilisateurs peuvent ajouter un nouvel employé en remplissant un formulaire. Lorsqu'ils soumettent le formulaire, les données sont envoyées à la base de données et la liste des employés est mise à jour.



```
const createEmployee = async () => {
  if (newEmployee.email && newEmployee.firstName && newEmployee.lastName &&
    newEmployee.contactNumber && newEmployee.post && newEmployee.socialId) {
    await addDoc(employeeCollectionRef, newEmployee);
    setNewEmployee({
      email: '',
      firstName: '',
      lastName: '',
      contactNumber: '',
      post: '',
      socialId: '',
    });
    const data = await getDocs(employeeCollectionRef);
    setEmployees(data.docs.map((doc) => ({ ...doc.data(), id: doc.id })));
  }
};
```

### - Modification d'Employés

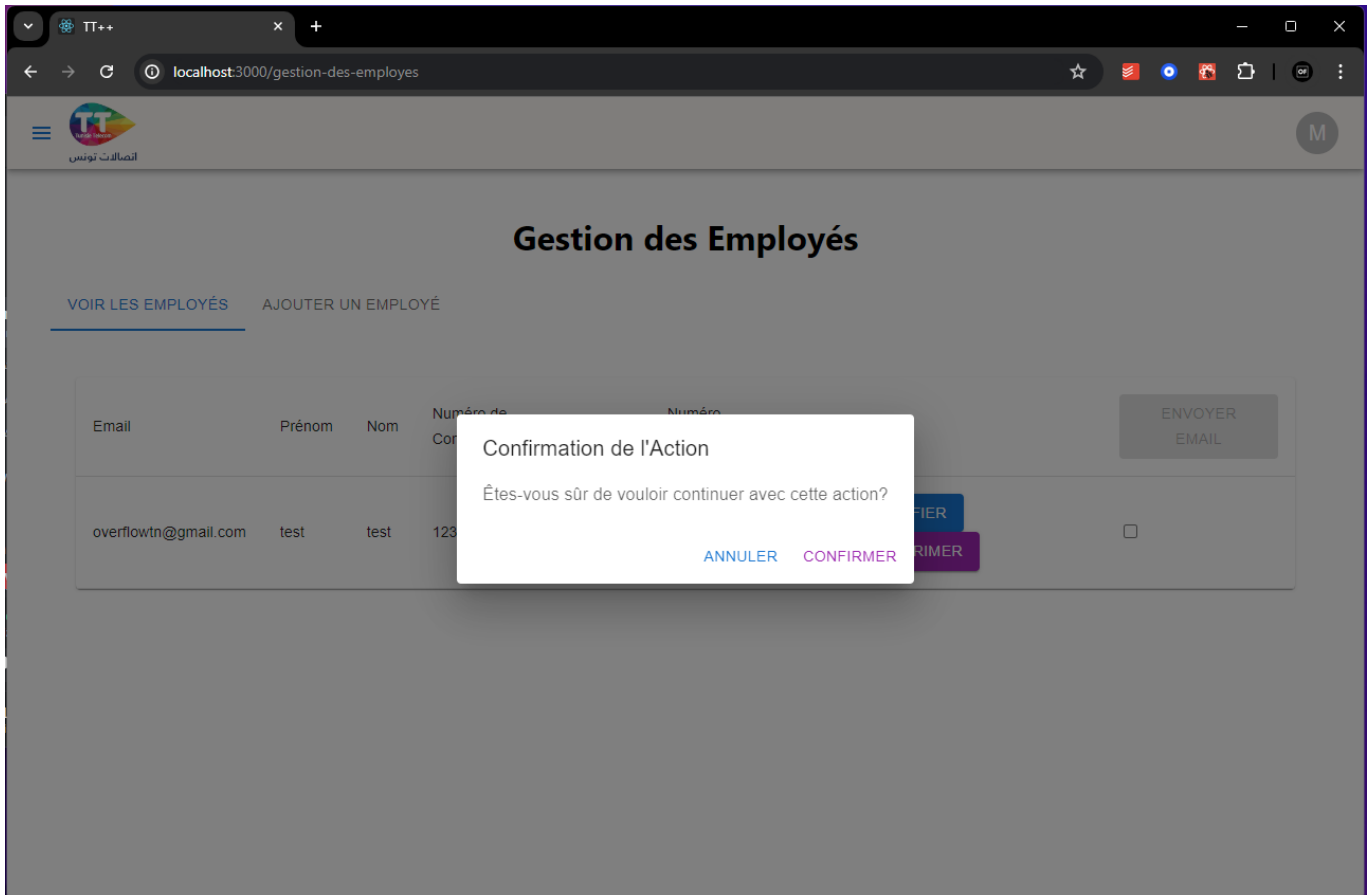
Les utilisateurs peuvent modifier les informations des employés directement dans la liste. Les modifications sont enregistrées dans la base de données et la liste est mise à jour.



```
const updateEmployee = async (id, updatedEmployee) => {
  const employeeDoc = doc(db, 'employees', id);
  await updateDoc(employeeDoc, updatedEmployee);
  setEditEmployeeId(null);
  const data = await getDocs(employeeCollectionRef);
  setEmployees(data.docs.map((doc) => ({ ...doc.data(), id: doc.id })));
};
```

### - Suppression d'Employés

Les utilisateurs peuvent supprimer un employé de la liste. La suppression est effectuée dans la base de données et la liste est mise à jour.



```
const deleteEmployee = async (id) => {
  const employeeDoc = doc(db, 'employees', id);
  await deleteDoc(employeeDoc);
  const data = await getDocs(employeeCollectionRef);
  setEmployees(data.docs.map((doc) => ({ ...doc.data(), id: doc.id })));
};
```

## - Envoi d'Emails

Les utilisateurs peuvent sélectionner plusieurs employés et envoyer des emails. L'adresse email des employés sélectionnés est utilisée pour envoyer le message.

```
const handleSendEmail = async () => {
  const selectedEmails = selectedEmployees.map((emp) => emp.email).join(', ');

  try {
    const response = await fetch('http://localhost:3001/sendEmail', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        to_email: selectedEmails,
        subject: 'test',
        text: 'test',
      })
    });
  }
};
```



```
    }},  
  });  
  
  if (response.ok) {  
    console.log('Email envoyé avec succès');  
  } else {  
    console.error('Échec de l\'envoi de l\'email');  
  }  
} catch (error) {  
  console.error('Erreur lors de l\'envoi de l\'email :', error);  
}  
};
```

## Backend pour l'Envoi d'Emails

---

### Configuration du Serveur Express

Ce code configure notre serveur Express pour envoyer des emails en utilisant Nodemailer :

```
const express = require('express');  
const nodemailer = require('nodemailer');  
const cors = require('cors');  
  
const app = express();  
app.use(express.json());  
app.use(cors());  
  
const smtpTransport = nodemailer.createTransport({  
  service: 'gmail',  
  auth: {  
    user: 'yassineahmedali02@gmail.com',  
    pass: 'cqdvokidxpgvnlhj',  
  },  
});
```

### Point d'Accès POST /sendEmail

Ce point d'accès écoute les requêtes POST sur /sendEmail. Il extrait les informations nécessaires de la requête JSON, configure les options de l'email, envoie l'email via SMTP, et gère les erreurs.

```
app.post('/sendEmail', async (req, res) => {  
  const { to_email, subject, text } = req.body;  
  
  try {  
    const mailOptions = {  
      from: 'yassineahmedali02@gmail.com',  
      to: to_email,  
      subject: subject,  
      text: text,  
    };  
    await smtpTransport.sendMail(mailOptions);  
    res.status(200).json({ message: 'Email envoyé avec succès' });  
  } catch (error) {  
    console.error('Erreur lors de l\'envoi de l\'email :', error);  
    res.status(500).json({ message: 'Erreur lors de l\'envoi de l\'email' });  
  }  
});
```

```
    text: text
  };

  await smtpTransport.sendMail(mailOptions);
  console.log('Email envoyé avec succès');
  res.status(200).send('Email envoyé');
} catch (error) {
  console.error('Erreur lors de l\'envoi de l\'email :', error);
  res.status(500).send('Erreur lors de l\'envoi de l\'email');
}
});
```

## Gestion des Erreurs

Nous gérons les erreurs lors de l'envoi d'email de manière appropriée. Si quelque chose ne fonctionne pas, nous répondons avec un statut HTTP 500 et un message d'erreur détaillé.

## Démarrage du Serveur

Le serveur démarre sur le port spécifié (process.env.PORT ou par défaut 3001) et affiche un message pour indiquer qu'il écoute les requêtes entrantes.

```
const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Le serveur écoute sur le port ${PORT}`);
});
```