

Rapport de Stage

2025



WHERE DREAMERS CONNECT WITH DOERS

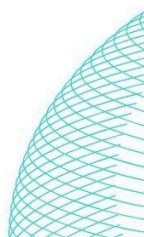
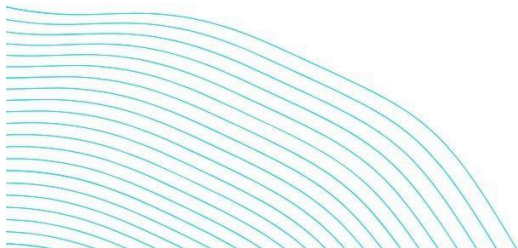
Tutrice : Asma ELKABIR
6 janvier - 20 février

YOHANN GONTHIER



Sommaire

01	Introduction Introduction.....Page 3
02	Présentation Présentation.....Page 4
03	Localisation Localisation.....Page 5
04	Missions Missions.....Page 6-24
05	Remerciement Remerciement.....Page 25



Introduction

Durant ma formation en BTS Services Informatiques aux Organisations (SIO), spécialité Solutions Logicielles et Applications Métiers (SLAM), j'ai effectué un stage du 6 janvier au 20 février au sein de l'entreprise Xpilab, située à Brunoy, sous la tutelle de Madame Asma EL KABIR.

Après plusieurs semaines de recherche, j'ai trouvé Xpilab, qui a accepté de me recevoir en tant que stagiaire. J'ai envoyé ma candidature par e-mail et, à la suite de cela, l'entreprise m'a proposé un entretien en présentiel afin d'évaluer mes motivations, les langages informatiques maîtrisés ainsi que mes formations personnelles. Une semaine plus tard, j'ai reçu une réponse positive m'informant que mon stage de six semaines et quatre jours était accepté.

L'objectif principal de ce stage était d'acquérir de l'expérience professionnelle et d'approfondir mes connaissances en informatique, tout en mettant en pratique les compétences acquises lors de ma deuxième année de BTS SIO ainsi que durant mes jours de formation personnelle.

Présentation de l'entreprise

1. Introduction

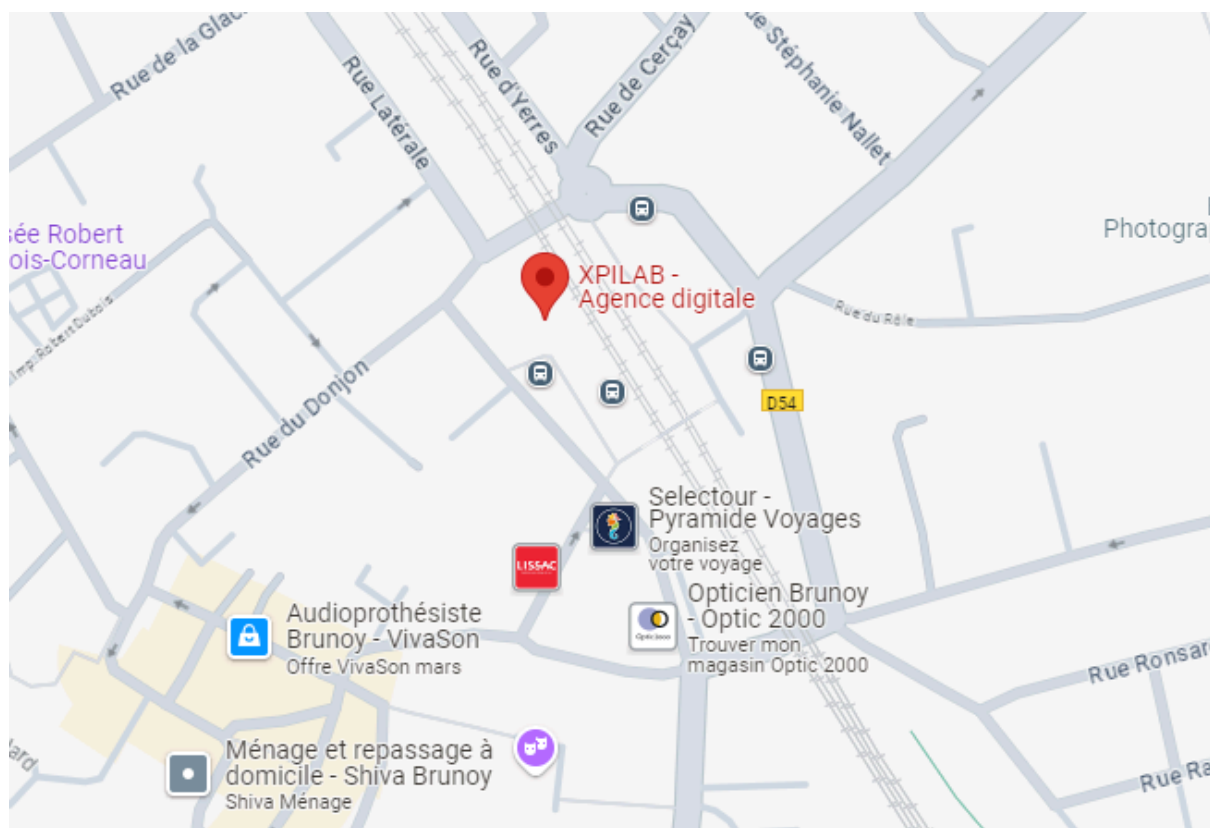
Xpilab est une agence digitale située à Brunoy, en région Île-de-France, au 22, place de la Gare, 91800 Brunoy. L'entreprise est spécialisée dans l'accompagnement des entreprises françaises dans leur transformation numérique.

Xpilab propose une gamme de services diversifiés pour répondre aux besoins numériques des entreprises et aider ses clients à améliorer leur présence en ligne et leur efficacité opérationnelle à travers des solutions adaptées à leurs besoin, comme :

- **Création de sites web : sites vitrines, sites e-commerce, et plateformes personnalisées.**
- **Conception d'identités visuelles : Réalisation de logos, chartes graphiques et autres éléments graphiques pour renforcer l'image de marque.**
- **Automatisation des services : Aide à l'optimisation des processus métiers.**
- **Formations au numérique : Accompagnement des entreprises et particuliers pour mieux utiliser les outils numériques modernes.**
- **Conseils en cybersécurité : Sensibilisation et recommandations pour protéger les données et les systèmes d'information.**

Xpilab accorde une grande importance à l'innovation, ce qui lui permet de proposer des solutions modernes et créatives. L'entreprise met également un point d'honneur à la satisfaction de ses clients et à l'accessibilité, rendant ainsi les outils numériques compréhensibles et utilisables par toutes les entreprises.

Localisation



Missions

Missions 1:

Lors de cette mission, ma tâche principale était de mettre en place un système d'inscription et de connexion pour les utilisateurs. J'ai commencé par créer un formulaire d'inscription avec validation des données. Pour mener à bien cette missions j'ai utilisé:

- Backend : Node.js avec Express
- Front-end : React, CSS3
- Base de données : MongoDB
- Authentification : JWT (JSON Web Token)
- Gestion des emails : Nodemailer
- Librairie: SweetAlert, Boxicons

1.1. Inscription de l'utilisateur

L'inscription d'un utilisateur commence par la collecte des informations de base nécessaires, telles que le nom, l'adresse e-mail et le mot de passe. J'ai mis en place un formulaire de saisie côté frontend, où l'utilisateur peut renseigner ces informations. Une fois le formulaire soumis, les données sont envoyées au backend via une requête HTTP POST. (Doc 1)

J'ai créé une route appelée /register. À l'intérieur de cette route, j'ai créé une variable existingUser afin de vérifier dans la base de données si l'utilisateur existe déjà. Si l'utilisateur existe, alors j'utilise la méthode de salage avec le hachage du mot de passe grâce à la librairie bcrypt. Ensuite, je crée l'utilisateur avec le nom, l'email, le mot de passe haché (password: hash), le rôle par défaut 'membre', et isMailVerified à false pour indiquer que l'email n'est pas encore vérifié après l'inscription. (Doc 2)

Côté front-end, j'ai créé une variable formData et setFormData, ce qui permet de récupérer les données transmises par l'utilisateur. Ensuite, on envoie une requête à la route /register du back-end en utilisant la méthode fetch(). On la définit en méthode POST, puis on convertit la variable formData en format JSON avec JSON.stringify() avant de l'envoyer au back-end, contenant ainsi toutes les informations de l'utilisateur. (Doc 3)

Doc 1:

The image shows a registration form titled "Inscription" centered on a blue gradient background. The form is enclosed in a dashed white border and contains the following elements:

- Nom :** A text input field.
- Email :** A text input field.
- Mot de passe :** A text input field.
- Confirmer votre mot de passe :** A text input field.
- S'inscrire**: A blue button.
- Connexion**: A green button.

Doc 2:

```
router.post("/register", async (req, res) => {
  //Vérifie si l'utilisateur existe déjà dans la base de données
  const existingUser = await User.findOne({ email });
  if (existingUser) {
    return res.status(400).json({ message: "Cet email est déjà utilisé." });
  }

  const salt = await bcrypt.genSalt(10);
  const hash = await bcrypt.hash(password, salt);

  const user = new User({
    name: name,
    email,
    password: hash,
    role: role || "member",
    projectId: projectId,
    isEmailVerified: false,
  });

  await user.save();
  console.log(`Utilisateur enregistré : ${user._id}`);
});
```

Doc 3:

```
const [formData, setFormData] = useState({
  name: '',
  email: '',
  password: '',
  confirmPassword: '',
});
```

```
const response = await fetch('http://localhost:3001/api/register', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(formData),
});
```

1.3. Vérification de l'email de l'utilisateur

J'ai mis en place un formulaire de connexion côté frontend, où l'utilisateur entre ses informations. Une fois le formulaire soumis et l'adresse email saisie, les données sont envoyées au backend via une requête HTTP POST. Le backend vérifie alors si l'adresse e-mail existe dans la base de données et si le compte utilisateur est vérifié. Si l'utilisateur est trouvé, son mot de passe est comparé avec celui stocké dans la base de données (après hachage). Si les identifiants sont valides, une session est créée pour l'utilisateur et un jeton d'authentification (token(JWT)) est généré. Ce jeton est ensuite renvoyé au frontend, où il est stocké dans le localStorage pour être utilisé lors des futures requêtes.

Pour commencer, à partir de la route /register, j'ai factorisé mon code (Don't Repeat Yourself) afin de le rendre plus propre et plus lisible. Dans le fichier middleware, j'ai créé une fonction generateToken qui regroupe toutes les informations de l'utilisateur, comme l'ID, l'email, le rôle et le nom. Une fois cette fonction appelée, je peux choisir quelles informations exporter. Dans mon cas, je récupère uniquement l'email et l'ID unique de l'utilisateur. (Doc 1)

Ensuite, j'ai créé une route /confirm-email en appliquant une méthode de factorisation. Dans le fichier sendMailer(), j'ai créé une fonction transporterMailer() qui utilise la bibliothèque Nodemailer. Cette fonction configure le transporteur avec la clé de l'application et l'adresse email de l'expéditeur, que j'ai stockées dans un fichier .env. Ensuite, j'ai défini une variable MailOptions contenant l'objet et le texte du message que l'expéditeur enverra à l'utilisateur. Enfin, j'envoie l'email en appelant sendMail().(Doc 2)

Du côté front-end, j'ai créé une fonction fléchée confirmEmail qui prend cinq paramètres :

- tokenURL → L'URL contenant le token de confirmation.
- navigate → Une fonction de navigation.
- setIsVerifEmail → Une fonction setState pour mettre à jour un état.
- setLoading → Une autre fonction setState pour gérer le chargement.
- setConfirmationMessage → Pour afficher un message à l'utilisateur.

Ensuite, j'ai utilisé fetch() pour envoyer une requête vers la route /confirm-email avec la méthode GET, car nous devons simplement récupérer le lien de confirmation. Si l'email est bien confirmé, alors user.isEmailVerified = true. (Doc 2)

Doc 1 :


```

router.post("/register", async (req, res) => {
  // ...
  console.log(`Utilisateur enregistré : ${user._id}`);

  // Créer et envoier le token
  const token = await middleware.generateToken(user._id, user.email);

  await sendMailer.sendConfirmationEmail(user.email, user._id);

  res
    .status(200)
    .json({ message: "Un email de confirmation a été envoyé.", token });
} catch (err) {
  console.error(err);
  res
    .status(500)
    .json({
      error: "Erreur lors de l'enregistrement.",
      details: err.message,
    });
}
});

```

```

async function generateToken(user) {
  // Génère un token avec toutes les informations de l'utilisateur
  return jwt.sign(
    { id: user._id, email: user.email, role: user.role, name: user.name },
    process.env.JWT_SECRET,
    { expiresIn: '30d' }
  );
}

module.exports = { generateToken }

```

Doc 2:

```

Const token=req.query.token; // Prendre le token depuis la query string
Const decoded = jwt.verify(token, process.env.JWT_SECRET);
const{ userId, email}= decoded;
Const user = awaitUser.findById(userId);

```

```

router.get("/confirm-email", async (req, res) => {
  user.isEmailVerified = true;
  await user.save();

  const transporter = await sendMailer.transporterMailer();

  const mailOptions = {
    from: process.env.EMAIL_USER,
    to: user.email,
    subject: "Confirmation de votre adresse email",
    text: "Votre adresse email a été confirmée avec succès.",
  };

  transporter.sendMail(mailOptions, async (error, info) => {
    if (error) {
      console.error("Erreur d'envoi d'email:", error);
      return res
        .status(500)
        .json({ success: false, message: "Erreur d'envoi de l'email." });
    }

    await middleware.generateToken(user._id, user.email);

    console.log("Email confirmé !");
    return res.status(200).json({ message: "Email confirmé avec succès." });
  });
} catch (error) {
  console.error(error);
  return res.status(400).json({ error: "Token invalide ou expiré." });
}
});

```

```

async function transporterMailer() {
  return nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: process.env.EMAIL_CREATOR,
      pass: process.env.EMAIL_PASS,
    },
  });
}

```

Dans le fichier VerifEmail.jsx:

```

const confirmEmail = (tokenURL, navigate, setIsVerifEmail, setLoading, setConfirmationMessage) => {
  // Appel à l'API pour confirmer l'email avec le token
  fetch(`http://localhost:3001/api/confirm-email?token=${tokenURL}`, {
    method: 'GET',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
      Origin: 'http://localhost:3000',
    },
  })
    .then((response) => response.json())
    .then((data) => {
      console.log('Réponse du backend:', data);
      if (data.message === 'Email confirmé.') {
        setConfirmationMessage('Email confirmé !');
        setIsVerifEmail(true); // L'email a été vérifié donc true
        setLoading(false); // Pas de chargement
      }
    });
}

```

Doc 3:

```
{
  "_id": ObjectId('67ab2ff04a0077b3a9ad2eba'),
  "name": "Yann",
  "email": "yohann.gonthier.jr@gmail.com",
  "password": "$2a$10$iw8s5Q2dyNxC9ae8ej6sh.6kVLI9bahsAPN.H9ZSWq9FZH3Rm1/ES",
  "role": "admin",
  "projectId": ObjectId('67ab2fe54a0077b3a9ad2eb8'),
  "isEmailVerified": true,
  "__v": 0
}
```



xpilabconnect@gmail.com
to me ▾

Tue, Feb 11, 11:50 AM (7 days ago) ☆ 😊 ↩ ⋮

Cliquez sur ce lien pour confirmer votre email: http://localhost:3000/confirm-email?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQlOiNzMDI2NDgwIiwiaWwiOiJ5b2hhbm4uZ29udGhpZXlueHBpbGFIQGRtYWIsImNvbSIsImhhdCI6MTczOTI3MTAyNiwiZXhwIjoxNzQxODYzMDI2fQ.E8-GA3FKNdsyB_82ioJWTDtSzpCyoPaJl1hnx9VsJk

1.4. Connexion de l'utilisateur

L'utilisateur saisit ses identifiants dans le formulaire de connexion. Le backend vérifie l'email, le mot de passe et l'état de vérification du compte. Si tout est valide, un jeton JWT est généré et renvoyé au frontend, où il est stocké pour les prochaines requêtes.

J'ai créé une nouvelle route `/login`, contenant une variable `passwordValid` avec la librairie `bcrypt` afin de comparer le mot de passe saisi par l'utilisateur et celui enregistré dans la base de données. Ensuite, je vérifie si le mot de passe est conforme.

J'ai créé une variable `authHeader = [authorization]`. Mais je pouvais aussi effectuer la requête `authHeader` en faisant `authHeader = req.body`. Ensuite, on vérifie le token utilisateur avec son jeton JWT. S'il est valide, on retourne un message JSON indiquant que la connexion est réussie. (Doc 1)

Si la connexion réussit, alors la fonction `generateToken` que j'ai défini dans le fichier `middleware` permettra de générer un nouveau token.

Côté utilisateur, j'ai cherché la route `/login` avec un `fetch()` prenant en compte l'email et le mot de passe. Si tout est bon, on stocke le token dans le `localStorage`, on envoie un message indiquant que la connexion est réussie, puis on redirige automatiquement l'utilisateur vers la page protégée. (Doc 2)

Doc 1:

```
const { email, password } = req.body; //Récupère les données saisi par l'utilisateur
const existing User = await User.findOne({ email });
```

```

router.post("/login", async (req, res) => {
  // Vérification si l'utilisateur existe déjà dans la base de données
  const existingUser = await User.findOne({ email: req.body.email });
  if (!existingUser) {
    return res.status(400).json({ message: "Cet email n'existe pas." });
  }

  const passwordValid = await bcrypt.compare(password, existingUser.password);
  if (!passwordValid) {
    return res.status(400).json({ message: "Mot de passe incorrect." });
  }

  // Vérification si un token valide existe déjà dans les headers
  const authHeader = req.headers["authorization"];
  if (authHeader) {
    const existingToken = authHeader.split(" ")[1];
    try {
      const verifiedUser = jwt.verify(existingToken, process.env.JWT_SECRET);
      console.log("Token existant vérifié :", verifiedUser);
      return res.status(200).json({
        message: "Connexion réussie avec un token existant.",
        user: { id: verifiedUser.id, email: verifiedUser.email },
        token: existingToken,
      });
    } catch (error) {
      // Token invalide
    }
  }
});

```

```

// Génération d'un nouveau token
const token = await middleware.generateToken(existingUser); //Méthode de factorisation

console.log("Nouveau token généré :", token);

// Réponse avec le nouveau token
res.status(200).json({
  message: "Connexion réussie.",
  user: {
    id: existingUser._id,
    name: existingUser.name,
    email: existingUser.email,
  },
  token,
});

```

Doc 2

```

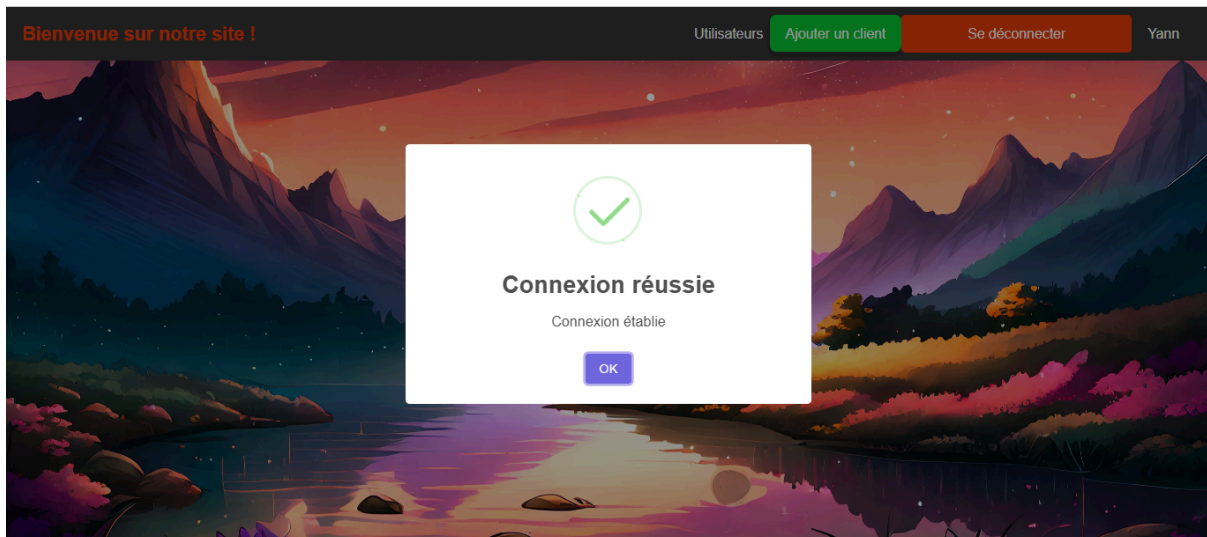
const Login = () => {
  const handleSubmit = async (e) => {
    const response = await fetch('http://localhost:3001/api/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        Accept: 'application/json',
        Origin: 'http://localhost:3000',
      },
      body: JSON.stringify({ email: email, password: password }),
    });

    const data = await response.json();
    console.log('Réponse du serveur :', data);

    if (response.ok) {
      localStorage.setItem('token', data.token);
      console.log('Connexion réussie, token enregistré :', data.token);
      alert(data.message);
      navigate('/protected');
    }
  };
};

```

Si mot de passe correct:



Si mot de passe incorrect:

A screenshot of a login form titled "Connexion" enclosed in a dashed blue border. Below the title, a red error message "Mot de passe incorrect." is displayed. The form contains two input fields: "Email" with the value "yohann.gonthier.jr@gmail.c" and "Mot de passe" with masked characters ".....". Below the password field is a link "Mot de passe oublié". At the bottom are two buttons: a blue "Se connecter" button and a green "Inscription" button.

1.5. Déconnexion de l'utilisateur

Lors de la déconnexion, le jeton d'authentification (JWT) stocké côté frontend est supprimé du localStorage. Cela empêche l'utilisateur d'accéder aux routes protégées sans se reconnecter.

J'ai créé une nouvelle route appelée /logout, qui renvoie simplement des messages JSON (Doc 1).

Côté front-end, j'ai appelé cette route avec fetch, puis j'ai créé une variable `setIsLoggedIn(false)` qui permet de déconnecter l'utilisateur. Enfin, j'ai utilisé `localStorage.removeItem('token')` pour supprimer le token utilisateur puis redirige automatiquement l'utilisateur vers la page /login. (Doc 2)

Doc 1 : Déconnexion

```
// Route pour se déconnecter
router.post("/logout", (req, res) => {
  try {
    res.status(200).send("Déconnecté avec succès.");
  } catch (err) {
    console.error("Erreur lors de la déconnexion :", err);
    res.status(500).send("Erreur interne du serveur lors de la déconnexion.");
  }
});
```

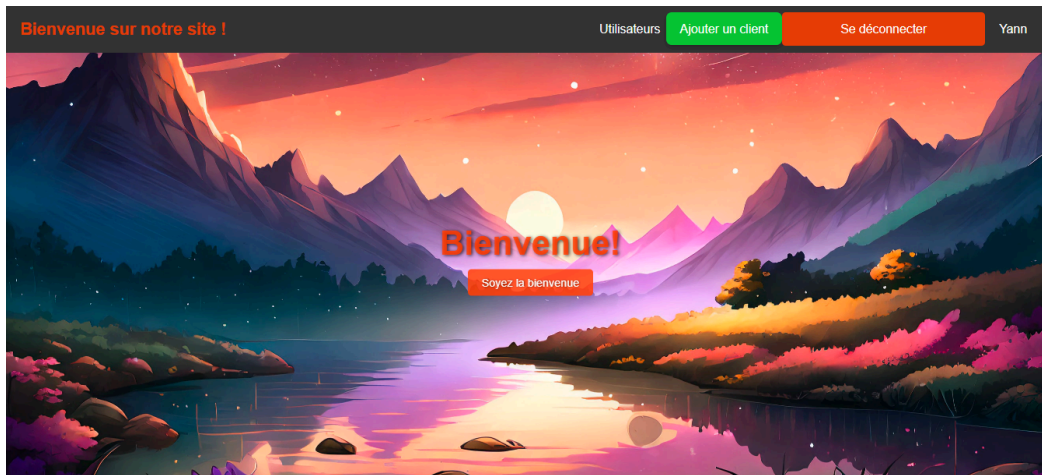
Doc 2:

```
const Navbar = () => {
  const handleLogout = async () => {
    try {
      const response = await fetch('http://localhost:3001/api/logout', {
        method: 'POST',
        credentials: 'include',
      });

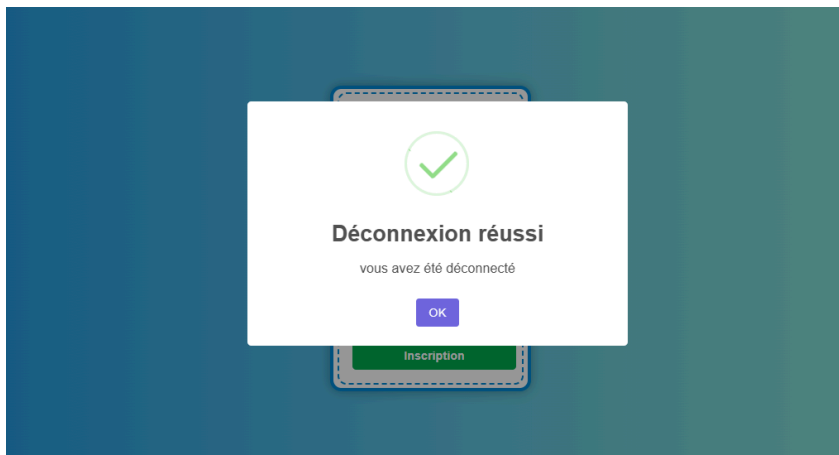
      if (response.ok) {
        setIsLoggedIn(false);
        localStorage.removeItem('token');
        navigate('/login'); // Redirection après déconnexion
      } else {
        alert('Erreur lors de la déconnexion.');
      }
    } catch (err) {
      console.error('Erreur lors de la déconnexion :', err);
    }
  };
};
```

Doc 3 :

Avant:



Après:



Missions 2: Création d'une page utilisateur unique

Ma deuxième mission consistait à développer une page unique pour chaque utilisateur, affichant ses informations personnelles.

Après l'authentification, l'utilisateur est redirigé vers sa page personnelle. Le backend fournit une route sécurisée qui récupère ses données à partir de son ID contenu dans le token JWT. Le frontend affiche ensuite dynamiquement ces informations, garantissant que chaque utilisateur ne voit que ses propres données. (rôle, email, nom)

J'ai créé une route `/authenticated` qui récupère le token utilisateur via `req.query.token`. Cela permet de transmettre le token JWT de l'utilisateur dans l'URL afin de le vérifier.

Ensuite, j'ai décodé le token JWT pour identifier l'utilisateur, puisque chaque utilisateur possède son propre jeton.

J'ai ensuite créé une variable `user` qui permet de rechercher l'utilisateur dans la base de données en fonction de son ID, grâce à la méthode `User.findById(decoded.id)`. Le paramètre `decoded` contient les données décodées du JWT. (Voir Doc 1)

Côté front-end. J'ai récupéré la route `/authenticated` et vérifié les données renvoyées par le backend. Si `data` possède une propriété `authenticated` et qu'elle n'est pas `null`, alors `setUserInfo` stocke toutes les informations de l'utilisateur tant que `data` est valide. (Voir Doc2)

Doc 1:

```
// Route pour vérifier si l'utilisateur est authentifié
router.get("/authenticated", async (req, res) => {
  const token = req.query.token; // Récupère le token de l'URL
  if (!token) {
    return res
      .status(401)
      .json({ authenticated: false, message: "Token manquant." });
  }

  // Vérifie le token de manière asynchrone
  jwt.verify(token, process.env.JWT_SECRET, async function (err, decoded) {
    if (err) {
      console.error("Erreur lors de la vérification du token:", err);
      return res
        .status(403)
        .json({ authenticated: false, message: "Token invalide ou expiré." });
    }

    console.log(`Décodé : ${decoded}`);

    // Recherche l'utilisateur dans la base de données
    const user = await User.findById(decoded.id);

    if (!user) {
      console.log("Utilisateur non trouvé.");
      return res
        .status(403)
        .json({ authenticated: false, message: "Utilisateur non trouvé." });
    }
  })
})
```



```

// Vérifie si l'email est vérifié
if (!user.isEmailVerified) {
  return res
    .status(403)
    .json({ authenticated: false, message: "Email non vérifié." });
}

// Si tout est validé, renvoie les informations de l'utilisateur
res.status(200).json({
  authenticated: true,
  message: "Utilisateur authentifié avec succès.",
  user: {
    userId: user._id,
    name: user.name,
    email: user.email,
    role: user.role,
  },
});
});
});

```

Doc 2:

Const token = localStorage.getItem('token');

```

}

fetch(`http://localhost:3001/api/authenticated?token=${token}`, {
  method: 'GET',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
    Origin: 'http://localhost:3000',
  },
})
.then(response => response.json())
.then(data => {
  if (data && data.authenticated && data.user) {
    setUserInfo({ email: data.user.email, role: data.user.role, name: data.user.name });
  } else {
    setError('Accès refusé ou session expirée.');
```

```

return (
  <div className="user-page">
    <div className="user-header">
      <h1>Bonjour, {userInfo.name || "Non disponible"} 🙌</h1>
    </div>

    <div className="user-content">
      <p className="welcome-text">Bienvenue sur ta page personnelle</p>

      <div className="user-info-card">
        <h2>Informations personnelles</h2>
        <ul>
          <li><strong>Email :</strong> {userInfo.email || "Non disponible"}</li>
          <li><strong>Rôle :</strong> {userInfo.role || "Non disponible"}</li>
          <li><strong>Nom :</strong> {userInfo.name || "Non disponible"}</li>
        </ul>
      </div>

      <div className="profile-container">
        
      </div>
    </div>
  </div>
)

```



Missions 3: Réinitialisation du mot de passe

J'ai implémenté un système permettant aux utilisateurs de réinitialiser leur mot de passe. Après avoir saisi leur email, ils reçoivent un code de vérification temporaire. Si le code est valide, ils accèdent à une page pour définir un nouveau mot de passe, qui est ensuite haché et mis à jour dans la base de données.

J'ai créé la route `/reset-password` avec la méthode **POST** afin d'envoyer des données depuis le client. Je récupère le token passé dans l'URL en utilisant la méthode **HTTP GET**. J'ai également utilisé une expression régulière (regex) pour valider un ensemble de chaînes de caractères, afin d'améliorer la sécurité (Doc 1)

Ensuite, j'ai recherché l'ID de l'utilisateur dans la base de données, puis j'ai haché son mot de passe avec la bibliothèque bcrypt, en utilisant un facteur de salage répété 10 fois. Enfin, j'ai enregistré ses informations mises à jour dans la base de données. (Doc 2)

Ensuite, le token de l'utilisateur est stocké dans le localStorage, puis une requête POST à la route /reset-password est émise, le corps de la requête avec `JSON.stringify({password})`. Si l'action est réussie, alors `setSuccess(true)` met à jour l'état de success à true afin d'afficher un message de succès, avec la librairie SweetAlert2. De plus, `setPassword("")` et `setConfirmPassword("")` réinitialisent les champs de mot de passe et de confirmation. (Doc 3).

Doc 1:

```
router.post("/reset-password", async (req, res) => {
  console.log("Entre dans le reset....");
  const token = req.query.token;
  const { password } = req.body;
  console.log("Token reçu :", token);
  console.log("Mot de passe reçu :", password);

  const regexPassword =
    /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*(),.?":{}|<>]).{12,}$/;

  if (!regexPassword.test(password)) {
    return res
      .status(400)
      .json({
        message:
          "Le mot de passe est invalide. Il doit contenir au moins une majuscule, u
```

Doc 2:

```
try {
  const userId = req.body.userId;

  const user = await User.findOne({ userId });

  if (!user) {
    return res.status(400).send("Token invalide ou expiré.");
  }

  const hashedPassword = await bcrypt.hash(password, 10);
  console.log("Mot de passe hashé :", hashedPassword);

  user.password = hashedPassword;
  user.resetToken = null;

  await user.save();
```

Doc 3:

```
// Appel à l'API pour réinitialiser le mot de passe
const token = localStorage.getItem('token');
fetch(`http://localhost:3001/api/reset-password?token=${token}`, {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
    Origin: 'http://localhost:3000',
  },
  body: JSON.stringify({
    password: password,
  }),
})
```

```
.then(response => response.json())
.then(data => {
  if (data.success) {
    setSuccess(true);
    setPassword("");
    setConfirmPassword("");
    Swal.fire({ icon: "success",
      text: 'Mot de passe modifié !'
    })
  }
})
```

Missions 3.1 : Code de vérification

J'ai créé la route `/code-mail-verify`, qui permet d'envoyer un code de vérification à l'utilisateur une fois inscrit avec son adresse e-mail. Cela servira à réinitialiser son mot de passe tout en s'assurant qu'il s'agit bien de lui, afin d'éviter qu'une personne malveillante ne puisse modifier son mot de passe en connaissant simplement son adresse e-mail.

J'ai récupéré l'e-mail de l'utilisateur avec `req.body`, ce qui permet de traiter les données envoyées par l'utilisateur.

Pour la vérification du code, j'ai utilisé `Math.random()` afin de générer un chiffre aléatoire compris entre 1000 et 9000, avec une expiration de 5 minutes grâce à la librairie `moment`.

J'ai également créé une variable `mailOptions` pour définir le message envoyé à l'utilisateur. Ensuite, avec la variable `transporter`, j'ai factorisé l'envoi des e-mails via `sendMailer`, qui contient toutes les informations liées à ma clé APP (le compte utilisé pour l'envoi des e-mails). (Doc 1)

Enfin, j'ai utilisé `.sendMailer(mailOptions)` pour envoyer le code de vérification à l'utilisateur avec le message défini. (Doc 2)

Ensuite, dans le frontend, j'ai appelé la route avec `fetch`. Si `data.success` est `true`, alors l'utilisateur sera redirigé en quelques secondes vers la page `/codeVerif` avec le code de vérification aléatoire qui lui a été envoyé. (Doc 3)

Si le code est valide, l'utilisateur sera redirigé vers la page `/reset-password`. J'ai utilisé une méthode Regex pour valider le mot de passe, ainsi que le hash et le salage pour sécuriser le stockage du mot de passe. Enfin, j'ai utilisé `await user.save()` pour enregistrer les nouvelles informations de l'utilisateur dans la base de données. (Doc 5)

Côté front-end, j'ai appelé la route `/reset-password` et vérifié si `data.success` est `true`. Si c'est le cas, les champs du mot de passe sont réinitialisés, et un message de confirmation est affiché à l'utilisateur pour l'informer que la réinitialisation a été effectuée avec succès. (Doc 6)

Doc 1:

```
const { email } = req.body;
const user = await User.findOne({ email });
if (!user) {
  return res
    .status(400)
    .json({ success: false, message: "Email non trouvé." });
}
console.log("Utilisateur trouvé au verify", user);
// Génération du code de vérification
const verificationCode = Math.floor(1000 + Math.random() * 9000); // Code à 4 chiffres

const expirationTime = moment().add(5, "minutes"); // Code expire dans 5 minutes
console.log(expirationTime);

user.resetCode = verificationCode;
user.resetCodeExpiration = expirationTime;
await user.save();

const mailOptions = {
  from: process.env.EMAIL_USER,
  to: email,
  subject: "Code de vérification pour réinitialisation du mot de passe",
  text: `Votre code de vérification est : ${verificationCode}.`,
};

const transporter = await sendMailer.transporterMailer();
```

Doc 2:

```
transporter.sendMail(mailOptions, (error, info) => {
  if (error) {
    console.error("Erreur d'envoi d'email:", error);
    return res
      .status(500)
      .json({ success: false, message: "Erreur d'envoi de l'email." });
  }
  res.json({
    success: true,
    message: "Un code de vérification a été envoyé.",
  });
});
} catch (error) {
  console.error("Erreur serveur:", error);
  res.status(500).json({ success: false, message: "Erreur serveur" });
}
```

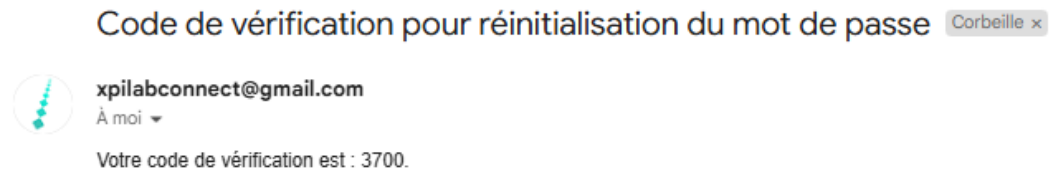
Doc 3:

```
try {
  console.log('Envoi de l\'email:', email);

  const response = await fetch('http://localhost:3001/api/code-mail-verify', {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
      Origin: 'http://localhost:3000',
    },
    body: JSON.stringify({ email }),
  });
  console.log(email)

  const data = await response.json();
  if (data.success) {
    Swal.fire('Succès', 'Le code de vérification a été envoyé.', 'success');
    setTimeout(() => {
      navigate('/codeVerif')
    }, 3000); //3 secondes
  } else {
    console.error('Erreur serveur:', data); // Logge l'erreur serveur
    Swal.fire('Erreur', data.message, 'error');
    setError(data.message);
  }
} catch (error) {
  Swal.fire('Erreur', 'Une erreur est survenue', 'error');
  setError('Une erreur est survenue');
} finally {
  setLoading(false);
}
};
```

Doc 4:



Doc 5:

```
const regexPassword =
  /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*(),.?":{}|<>]).{12,}$/;

if (!regexPassword.test(password)) {
  return res
    .status(400)
    .json({
      message:
        "Le mot de passe est invalide. Il doit contenir au moins une majuscule,
    });
}

try {
  const userId = req.body.userId;

  const user = await User.findOne({ userId });

  if (!user) {
    return res.status(400).send("Token invalide ou expiré.");
  }

  const hashedPassword = await bcrypt.hash(password, 10);
  console.log("Mot de passe hashé :", hashedPassword);

  user.password = hashedPassword;
  user.resetToken = null;

  await user.save();
}
```

Doc 6 :

```
if (password !== confirmPassword) {  
  Swal.fire({icon: "error",  
    text: "Les mots de passe ne correspondent pas."  
  });  
  setSuccess(false);  
} else {  
  setError(null);  
}
```

```
const token = localStorage.getItem('token');  
fetch(`http://localhost:3001/api/reset-password?token=${token}`, {  
  method: 'POST',  
  headers: {  
    Accept: 'application/json',  
    'Content-Type': 'application/json',  
    Origin: 'http://localhost:3000',  
  },  
  body: JSON.stringify({  
    password: password,  
  }),  
})  
  .then(response => response.json())  
  .then(data => {  
    if (data.success) {  
      setSuccess(true);  
      setPassword("");  
      setConfirmPassword("");  
      Swal.fire({ icon: "success",  
        text: 'Mot de passe modifié !'  
      })  
    } else {  
      window.alert(data.message || 'Erreur lors de la réinitialisation du mot de passe.');    }  
  })  
})
```


Remerciement

Je tiens à exprimer ma sincère gratitude à Madame Asma EL KABIR, responsable du système d'information de l'entreprise Xpilab, pour m'avoir offert l'opportunité d'effectuer mes six semaines de stage au sein de son équipe.

Asma Elkabir a été une tutrice exemplaire, m'accompagnant tout au long de mon stage. Elle a su répondre avec patience à toutes mes questions et à mes besoins, ce qui m'a permis de m'intégrer rapidement et de progresser efficacement.

Je la remercie chaleureusement pour les précieuses connaissances qu'elle m'a transmises, ainsi que pour sa gentillesse et sa capacité à expliquer clairement les concepts. Son soutien a été inestimable et a grandement enrichi mon expérience professionnelle.