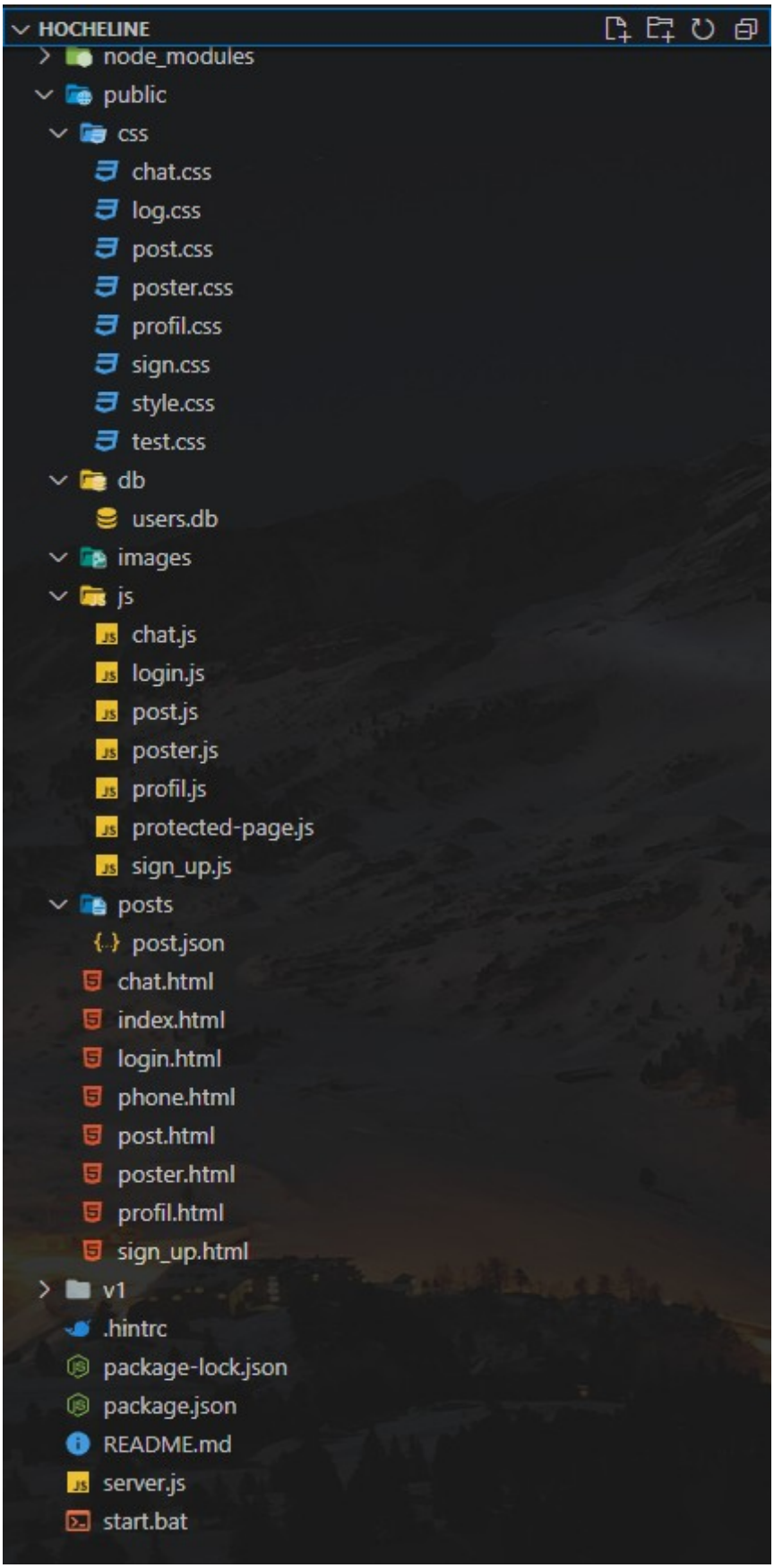
HOCHELINE DOCUMENTATION

Sommaire

HOCHELINE DOCUMENTATION

I - Arborescence



HOCHELINE DOCUMENTATION

II – Server.js

1) Code

```
const express = require('express');
const fs = require('fs');
const app = express();
const http = require('http').createServer(app);
const io = require('socket.io')(http);
const multer = require('multer');
const path = require('path');
const port = 3000;
const sqlite3 = require('sqlite3').verbose();

const filePath = 'public/posts/post.json';

app.use(express.static('public'));

let onlineCount = 0;
let sql;

var postVar = "";

const loginDB = new sqlite3.Database('public/db/users.db', sqlite3.OPEN_READWRITE,
(err) => {
  if(err) return console.error(err.message);
});

sql = 'CREATE TABLE IF NOT EXISTS users(id INTEGER PRIMARY
KEY,username,password,email,admin)';
loginDB.run(sql);

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/public/index.html');
});

io.on('connection', (socket) => {

  const clientIp = socket.handshake.address;

  onlineCount++;

  io.emit('display post', postVar);

  socket.on('disconnect', () => {
    onlineCount--;
```

```
  });

  socket.on('getIP', () => {
    return clientIp;
  });

  socket.on('chat message', (msgNom) => {
    io.emit("display message", msgNom);
  });

  socket.on('register post', (innerHTML) => {

    fs.readFile(filePath, 'utf8', (err, data) => {
      let jsonData = {};

      if (err) {
          if (err.code !== 'ENOENT') {
            console.error('Erreur lors de la lecture du fichier JSON :', err);
            return;
          }
      } else {
          if (data.trim() !== '') {
              try {
                jsonData = JSON.parse(data);
              } catch (jsonErr) {
                console.error('Erreur lors de l\'analyse du JSON existant :',
jsonErr);
                return;
              }
          }
      }
      if (!jsonData.elements) {
          jsonData.elements = [];
      }

      if(innerHTML){
        jsonData.elements.push({
          innerHTML: innerHTML
        });
      }

      const updatedJsonContent = JSON.stringify(jsonData, null, 2);

      fs.writeFile(filePath, updatedJsonContent, 'utf8', (writeErr) => {
          if (writeErr) {
            console.error('Erreur lors de l\'écriture du fichier JSON :',
writeErr);
            return;
          }
```

```
    });
    postVar = creerListeDepuisObjet(jsonData.elements);
    io.emit("display post", postVar);
  });
});

// DATABASE HANDLER
socket.on("login", (username, password) => {
  sql = "SELECT * FROM users WHERE username = ? AND password = ?";
  loginDB.get(sql, [username, password], (err, row) => {
    if (err) {
      console.error(err.message);
      return;
    }
    if (row) {
      const data = {
        username: username,
        password: password,
        url: "http://82.121.132.29:3000/"
      };
      socket.emit('redirect', data);
    }
  });
});
socket.on("sign up", (userData) =>{
  const username = userData.username;
  const password = userData.password;
  const email = userData.email;
  sql = 'INSERT INTO users (username, password, email) VALUES (?, ?, ?)';
  loginDB.run(sql, [username, password, email], (err) => {
    console.log(username, password, email);
    const data = {
      username: username,
      password: password,
      url: "http://82.121.132.29:3000/"
    };
    socket.emit('redirect', data);
    if (err) {
      console.error(err.message);
      return;
    }
  });
});
socket.on("check admin", (messageData) => {
  sql = "SELECT * FROM users WHERE username = ? AND admin = 'true'";
  loginDB.get(sql, [messageData.username], (err, row) => {
    if (err) {
      console.error(err.message);
      return;
```

```
      }
      if (row) {
        messageData.check = "admin";
        messageData.message = messageData.username + " >>> " +
messageData.message;
        socket.emit("display message checked", messageData);
      } else {
        messageData.check = "user";
        messageData.message = messageData.username + " >>> " +
messageData.message;
        socket.emit("display message checked", messageData);
      }
    });
  });
  socket.on("getUserData", (username) => {
    sql = "SELECT email, password, admin FROM users WHERE username = ?";
    loginDB.get(sql, [username], (err, row) => {
      if (err) {
        console.error(err.message);
        return;
      }
      if (row) {
        socket.emit("fill edit form", row);
      }
    });
  });
  socket.on("getUserEmail", (data1) => {
    sql = "SELECT email, admin, password FROM users WHERE username = ?";
    loginDB.get(sql, [data1.username], (err, row) => {
      if (err) {
        console.error(err.message);
        return;
      }
      if (row) {
        const data = {
          email: row.email,
          event: data1.event,
          admin: row.admin,
          username: data1.username,
          password: row.password
        };
        console.log(data);
        socket.emit("popup info", data);
      }
    });
  });
  socket.on("edit profil", (data) => {
    sql = "UPDATE users SET username = ?, email = ?, password = ? WHERE username =
?";
```

```
    loginDB.run(sql, [data.username, data.email, data.password,
data.old_username], (err, row) => {
      if (err) {
        console.error(err.message);
        return;
      }
      socket.emit("profil edited", data);
    });
  });
  socket.on("delete profil", (username) => {
    sql = "DELETE FROM users WHERE username = ?";
    loginDB.run(sql, [username], (err, row) => {
       if (err) {
         console.error(err.message);
         return;
       }
       socket.emit("profil deleted", username);
    });
   });
});

function creerListeDepuisObjet(obj) {
  try {
    if (Array.isArray(obj)) {
      const liste = [];

      obj.forEach((element) => {
        if (element.innerHTML) {
          liste.push(element.innerHTML);
        } else {
          liste.push('[Object sans propriété nom]');
        }
      });

      return liste;
    } else {
      console.error('L\'objet ne représente pas un tableau.');
      return null;
    }
  } catch (error) {
    console.error('Erreur lors de la création de la liste :', error);
    return null;
  }
}

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, path.join(__dirname, 'public', 'images'));
  },
```

HOCHELINE DOCUMENTATION

```
  filename: function (req, file, cb) {
    cb(null, file.originalname);
  },
});

const upload = multer({ storage: storage });

app.post('/upload', upload.single('image'), (req, res) => {
  res.send('Fichier téléchargé avec succès !');
});

app.use(express.static(path.join(__dirname, 'public')));

app.listen(port, () => {
  console.log(`
`);
});

http.listen(port, '0.0.0.0', () => {
  console.log(`Serveur bien démarré sur le port ${port}`);
});
```

2) Documentation

Ce code représente un serveur pour un réseau social basé sur le framework Node.js avec Express et Socket.io, et une base de données SQLite pour gérer les utilisateurs. La fonctionnalité principale du réseau social est de permettre aux utilisateurs de s'inscrire, de se connecter, de publier des messages, et de modifier leurs profils.

Fonctionnalités Principales :

Express Setup :

HOCHELINE DOCUMENTATION

Le serveur utilise Express pour créer une application web.

Le dossier public est configuré comme répertoire statique pour servir des fichiers statiques tels que les fichiers HTML, CSS, et images.

javascript

```
const express = require('express');

const fs = require('fs');

const app = express();

const http = require('http').createServer(app);

const io = require('socket.io')(http);

const multer = require('multer');

const path = require('path');

const port = 3000;

const sqlite3 = require('sqlite3').verbose();
```

Socket.io Setup :

Socket.io est utilisé pour permettre la communication en temps réel entre le serveur et le client.

javascript

```
const io = require('socket.io')(http);
```

Base de Données SQLite Setup :

Une base de données SQLite est utilisée pour stocker les informations des utilisateurs.

Javascript

```
const loginDB = new sqlite3.Database('public/db/users.db', sqlite3.OPEN_READWRITE, (err) => {
  if(err) return console.error(err.message);
});
// Création de la table 'users' si elle n'existe pas encore
sql = 'CREATE TABLE IF NOT EXISTS users(id INTEGER PRIMARY KEY,username,password,email,admin)';
loginDB.run(sql);
```

Endpoints HTTP :

Configuration des points de terminaison pour le serveur web.

HOCHELINE DOCUMENTATION

Javascript

```
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/public/index.html');
});
app.post('/upload', upload.single('image'), (req, res) => {
  res.send('Fichier téléchargé avec succès !');
});
```

Gestion des Connexions Socket.io :

Les connexions socket.io sont gérées pour compter le nombre d'utilisateurs en ligne, afficher les messages du chat en temps réel, et gérer la publication de messages.

javascript

```
io.on('connection', (socket) => {
  // ...
  socket.on('disconnect', () => {
    onlineCount--;
  });
  // ...
});
```

Gestion des Utilisateurs :

La base de données SQLite est utilisée pour gérer les opérations liées aux utilisateurs telles que l'inscription, la connexion, la vérification des administrateurs, l'obtention de données d'utilisateur, la modification de profil, et la suppression de profil.

javascript

```
socket.on("login", (username, password) => {
  // ...
});

socket.on("sign up", (userData) => {
  // ...
});
```

```
socket.on("check admin", (messageData) => {
  // ...
});

socket.on("getUserData", (username) => {
  // ...
});

socket.on("edit profil", (data) => {
  // ...
});

socket.on("delete profil", (username) => {
  // ...
});
```

Fonctions Utilitaires :

Une fonction utilitaire est définie pour créer une liste à partir d'un objet JSON.

javascript

```
function creerListeDepuisObjet(obj) {
  // ...
}
```

Multer Configuration :

Multer est configuré pour gérer le téléchargement d'images sur le serveur.

javascript

```
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, path.join(__dirname, 'public', 'images'));
  },
  filename: function (req, file, cb) {
```

```
  cb(null, file.originalname);
 },
});
const upload = multer({ storage: storage });
```

Démarrage du Serveur :

Le serveur est démarré sur le port 3000.

javascript

```
http.listen(port, '0.0.0.0', () => {
 console.log(`Serveur bien démarré sur le port ${port}`);
});
```

Remarques :

- Les messages du chat et les publications sont partagés en temps réel avec tous les utilisateurs connectés.
- L'interface d'administration est gérée en vérifiant si un utilisateur est un administrateur dans la base de données.
- Les informations des utilisateurs sont stockées de manière sécurisée dans une base de données SQLite.
- Note :
- Il est important de prendre des mesures de sécurité supplémentaires lors du déploiement d'une application en production, telles que la validation des entrées utilisateur, la gestion des sessions, le chiffrement des mots de passe, etc.

II – Pages HTML

1) index.html

a) Code

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/test.css">

    <title>Hocheline</title>
</head>
<body>
    <header>
        <div class="header">
            <h1 id="title">Hocheline - Username</h1>
```

```
            <nav>
                <a href="index.html">accueil</a>
                <a href="chat.html">chat</a>
                <a href="post.html">post</a>
                <a href="poster.html">poster</a>
                <a href="profil.html">compte</a>
            </nav>
        </div>
    </header>
    <div class="container">
        <div class="top-left">
            <h2>Hocheline ?</h2>
            <p>
                Bienvenue sur Hocheline, le réseau social conçu sur mesure pour
les élèves de la cité scolaire Hoche ! Hocheline offre une plateforme interactive
qui combine un chat en direct et un système de publication, créant ainsi un espace
virtuel vibrant et engageant où les membres peuvent interagir, partager et se
connecter. C'est un espace conçu pour renforcer les liens au sein de la
communauté, faciliter la collaboration éducative, et célébrer les moments uniques
qui font du lycée une expérience inoubliable.
            </p>
        </div>
        <div class="bottom-left">
            <h2>Conditions d'utilisation</h2>
            <p>
                <p>En utilisant Hocheline, le réseau social dédié à la cité
scolaire Hoche, vous acceptez les conditions suivantes, élaborées dans le but de
créer une communauté en ligne positive, sécurisée et enrichissante pour tous les
membres :</p>

                <h3>1. Eligibilité :</h3>
                <p> Hocheline est exclusivement destiné aux élèves, enseignants et
membres du personnel de la cité scolaire Hoche. L'accès est limité aux personnes
ayant une affiliation active avec l'établissement.</p>


                <h3>2. Respect et Civilité :</h3>
                <p>Les utilisateurs sont tenus de respecter les opinions,
croyances et points de vue des autres membres. Les commentaires ou publications
offensants, diffamatoires, discriminatoires ou irrespectueux ne seront pas
tolérés.</p>

                <h3>3. Confidentialité et Sécurité :</h3>
                <p>La confidentialité des membres est une priorité. Évitez de
partager des informations personnelles sensibles, telles que les adresses, numéros
de téléphone ou informations bancaires.
                Signalez immédiatement tout comportement suspect ou toute
violation de la sécurité.</p>
```

```
                <h5>/!\ Ce site web n'a pas un sécurité infaible, veuillez à
utiliser un mot de passe unique à ce site !</h5>


                <h3>4. Contenu Approprié :</h3>
                <p>Les publications doivent respecter les normes de décence,
éviter le contenu explicite, obscène ou illégal. Tout contenu susceptible de
perturber le bon fonctionnement de la communauté sera supprimé.</p>

                <h3>5. Utilisation Responsable :</h3>
                <p>Les membres s'engagent à utiliser Hocheline de manière
responsable, en évitant toute activité pouvant nuire à la réputation du Lycée
Hoche ou de ses membres.</p>

                <h3>6. Propriété Intellectuelle :</h3>
                <p>Le contenu publié sur Hocheline doit respecter les droits
d'auteur et autres droits de propriété intellectuelle. Les utilisateurs sont
responsables du contenu qu'ils partagent.</p>

                <h3>7. Modération et Sanctions :</h3>
                <p></p>L'équipe de modération veillera au respect des conditions
d'utilisation. Les violations peuvent entraîner des avertissements, des
suspensions temporaires ou la désactivation définitive du compte, selon la gravité
de l'infraction.

                <h3>8. Modifications des Conditions :</h3>
                <p>Hocheline se réserve le droit de mettre à jour ou de modifier
les conditions d'utilisation. Les utilisateurs seront informés des changements
significatifs. En utilisant Hocheline, vous vous engagez à respecter ces
conditions d'utilisation. Tout manquement peut entraîner des mesures appropriées,
y compris la résiliation du compte. Nous vous encourageons à contribuer
positivement à la création d'une communauté virtuelle exemplaire au sein du Lycée
Hoche de Versailles.</p>
            </p>
        </div>
        <div class="right">
            <h1>Fonctionnalités et caractéristiques principales :</h1>
            <h3>Chat en Direct :</h3>
            <p>Hocheline favorise la communication instantanée au sein de la
communauté. Que ce soit pour discuter de devoirs, partager des idées
passionnantes, ou simplement échanger des nouvelles, le chat en direct permet aux
utilisateurs de rester connectés en temps réel.</p>
            <h3>Système de Publication :</h3>
            <p>Exprimez-vous et partagez vos pensées, photos, et expériences à
travers le système de publication intuitif. Que ce soit pour publier des annonces
importantes, des projets créatifs, ou des moments mémorables, Hocheline offre une
plateforme pour mettre en avant la diversité et la richesse de la vie
lycéenne.</p>
            <h3>Communauté Scolaire :</h3>
```

```
            <p>Hocheline est exclusivement dédié à la communauté du Lycée Hoche,
créant ainsi un espace sûr et familial où les élèves, les enseignants, et le
personnel peuvent interagir de manière transparente et enrichissante.</p>
        </div>
    </div>
    <div class="footer">
        <p>© 2023-2024 Hocheline - crée par des <a href="#" id="h2__popup"
class="h2__popup">élèves</a> du Lycée Hoche</p>
    </div>
    <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
</body>
</html>
```

b) Documentation

Le code HTML représente la structure de la page d'accueil du réseau social "Hocheline". Voici une documentation expliquant chaque partie du code :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
    <link rel="stylesheet" href="css/test.css">
    <title>Hocheline</title>
</head>
<body>
```

- **<!DOCTYPE html>** : Déclaration du type de document HTML.
- **<html lang="en">** : Balise HTML avec l'attribut lang spécifiant la langue (anglais).
- **<head>** : Section contenant les métadonnées de la page.
    - **<meta charset="UTF-8">** : Définit l'encodage des caractères comme UTF-8.
    - **<meta name="viewport" content="width=device-width, initial-scale=1.0">** : Configuration de la vue pour les appareils mobiles.
    - **<link rel="stylesheet" href="css/test.css">** : Liaison vers une feuille de style externe ("test.css").
    - **<title>Hocheline</title>** : Titre de la page affiché dans l'onglet du navigateur.
- **</head>** : Fin de la section contenant les métadonnées de la page.

HOCHELINE DOCUMENTATION

```
<header>
    <div class="header">
        <h1 id="title">Hocheline - Username</h1>
        <nav>
            <a href="index.html">accueil</a>
            <a href="chat.html">chat</a>
            <a href="post.html">post</a>
            <a href="poster.html">poster</a>
            <a href="profil.html">compte</a>
        </nav>
    </div>
</header>
```

<header> : Section de l'en-tête de la page.

- ** ** : Conteneur pour l'en-tête.
    - **<h1 id="title">Hocheline - Username</h1>** : Titre principal avec un éventuel nom d'utilisateur.
    - **<nav>** : Navigation contenant des liens vers différentes sections du site.
- ** ** : Fin du conteneur pour l'en-tête.

</header> : Fin de la section de l'en-tête de la page.

```
<div class="container">
    <div class="top-left">
        <!-- Contenu de la partie supérieure gauche -->
    </div>
    <div class="bottom-left">
        <!-- Contenu de la partie inférieure gauche -->
    </div>
    <div class="right">
        <!-- Contenu de la partie droite -->
    </div>
</div>
```

** ** : Conteneur principal de la page, divisé en trois parties.

- **<top-left>** : Section supérieure gauche.
- **<bottom-left>** : Section inférieure gauche.
- **<right>** : Section droite.

** ** : Conteneur principal de la page, divisé en trois parties.

```
 <div class="footer">
        <p>© 2023-2024 Hocheline - crée par des <a href="#" id="h2__popup"
class="h2__popup">élèves</a> du Lycée Hoche</p>
```

```
    </div>
```

** ** : Pied de page**.**

- **<p>** : Paragraphe contenant l'année de création et la mention "créé par des élèves du Lycée Hoche".
- **élèves** : Lien avec une classe pour éventuellement afficher un popup.

** ** : Fin du pied de page**.**

```
<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
```

<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script> : Inclusion du script client de Socket.io pour la communication en temps réel.

Résumé :

- La page HTML est structurée avec des sections distinctes pour l'en-tête, le contenu principal, et le pied de page.
- Les liens de navigation permettent d'accéder aux différentes sections du site.
- Des divs avec des classes spécifiques sont utilisées pour diviser la page en plusieurs parties.
- Un script client Socket.io est inclus pour la communication en temps réel.

Note :

- Pour une meilleure compréhension, les parties spécifiques du contenu (top-left, bottom-left, right) n'étaient pas fournies dans le code HTML fourni.

2) profil.html

a) Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/profil.css">
    <title>Document</title>
</head>
<body>
```

```
    <header>
        <div class="header">
            <h1 id="title">Hocheline - Username</h1>
            <nav>
                <a href="index.html">accueil</a>
                <a href="chat.html">chat</a>
                <a href="post.html">post</a>
                <a href="poster.html">poster</a>
                <a href="profil.html">compte</a>
            </nav>
        </div>
    </header>
    <div class="container"></div>
    <div class="container">
        <div class="box">
            <h2></h2>
            <label for="title">Username:</label>
            <input type="text" id="username-input" name="username" required>

            <label for="description">Email:</label>
            <input type="text" id="email-input" name="email" required>

            <label for="image">Password</label>
            <input type="text" id="password-input" name="password" required>

            <button class="submit btn" id="submit">Valider les
changements</button>
        </div>
        <div class="box">
            <div class="checkbox-container">
                <input type="checkbox" id="receiveEmails"
name="receiveEmails">
                <label for="receiveEmails">Recevoir des emails</label>
            </div>
            <button id="deco" class="btn">Se déconnecter</button>
            <button id="delete" class="delete">Supprimer votre
compte</button>
            <h4>Attention ! Toute suppression de compte est définitive !
</h4>
        </div>
    </div>

    <div class="footer">
        <p>© 2023-2024 Hocheline - crée par des <a href="#" id="h2__popup"
class="h2__popup">élèves</a> du Lycée Hoche</p>
    </div>
    <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
    <script src="js/profil.js"></script>
    <script src="js/protected-page.js"></script>
```

```
</body>
</html>
```

b) Documentation

Le code HTML ci-dessous représente la structure de la page de profil du réseau social "Hocheline". Voici une documentation détaillée expliquant chaque partie du code :

html

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <link rel="stylesheet" href="css/profil.css"> <title>Document</title> </head> <body>

- **<!DOCTYPE html>** : Déclaration du type de document HTML.
- **<html lang="en">** : Balise HTML avec l'attribut lang spécifiant la langue (anglais).
- **<head>** : Section contenant les métadonnées de la page.
    - **<meta charset="UTF-8">** : Définit l'encodage des caractères comme UTF-8.
    - **<meta name="viewport" content="width=device-width, initial-scale=1.0">** : Configuration de la vue pour les appareils mobiles.
    - **<link rel="stylesheet" href="css/profil.css">** : Liaison vers une feuille de style externe ("profil.css").
    - **<title>Document</title>** : Titre de la page.

html

<header> <h1 id="title">Hocheline - Username</h1> <nav> accueil chat post poster compte </nav> </header>

- **<header>** : Section de l'en-tête de la page.
    - ** ** : Conteneur pour l'en-tête.
        - **<h1 id="title">Hocheline - Username</h1>** : Titre principal avec un éventuel nom d'utilisateur.
        - **<nav>** : Navigation contenant des liens vers différentes sections du site.

html

   <!-- Contenu de la première boîte -->   <!-- Contenu de la deuxième boîte -->

- ** ** : Conteneur vide.
- ** ** : Conteneur principal de la page, divisé en deux boîtes.
    - **.box** : Boîte contenant des informations de profil.
        - **<h2></h2>** : Titre de la boîte (vide dans l'exemple).
        - **<label for="title">Username:</label>** : Libellé pour le champ de saisie du nom d'utilisateur.

- **<input type="text" id="username-input" name="username" required>** : Champ de saisie pour le nom d'utilisateur.
- Autres champs similaires pour l'email et le mot de passe.
- **<button class="submit btn" id="submit">Valider les changements</button>** : Bouton de validation des changements.

- Deuxième **.box** : Boîte contenant des options de profil.
  - ** ** : Conteneur pour une case à cocher.
    - **<input type="checkbox" id="receiveEmails" name="receiveEmails">** : Case à cocher pour recevoir des emails.
    - **<label for="receiveEmails">Recevoir des emails</label>** : Libellé associé à la case à cocher.
  - **<button id="deco" class="btn">Se déconnecter</button>** : Bouton pour se déconnecter.
  - **<button id="delete" class="delete">Supprimer votre compte</button>** : Bouton pour supprimer le compte.
  - **<h4>Attention ! Toute suppression de compte est définitive !</h4>** : Avertissement sur la suppression de compte.

html

 © 2023-2024 Hocheline - crée par des élèves du Lycée Hoche <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script> <script src="js/profil.js"></script> <script src="js/protected-page.js"></script> </body> </html>

- ** ** : Pied de page.
  - ** ** : Paragraphe contenant l'année de création et la mention "créé par des élèves du Lycée Hoche".
  - **élèves** : Lien avec une classe pour éventuellement afficher un popup.
- **<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>** : Inclusion du script client de Socket.io pour la communication en temps réel.
- **<script src="js/profil.js"></script>** : Inclusion d'un script JavaScript spécifique pour la page de profil.
- **<script src="js/protected-page.js"></script>** : Inclusion d'un autre script JavaScript pour les pages protégées.

Résumé :

- La page HTML est structurée avec des sections distinctes pour l'en-tête, le contenu principal, et le pied de page.
- Elle contient deux boîtes dans la section principale, chacune avec des informations et des options de profil.
- Des champs de saisie et des boutons sont utilisés pour interagir avec le profil de l'utilisateur.
- Des scripts JavaScript spécifiques sont inclus pour gérer les fonctionnalités de la page de profil.
- Socket.io est inclus pour la communication en temps réel.

HOCHELINE DOCUMENTATION

Note :

- Certains détails spécifiques tels que les titres des boîtes, les noms des champs, et les fonctionnalités des scripts ne sont pas fournis dans le code HTML. Ces détails peuvent être implémentés dans les fichiers CSS et JavaScript associés.

3) chat.html

a) Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/chat.css">
    <title>Document</title>
</head>
<body>
    <header>
        <div class="header">
            <h1 id="title">Hocheline - Username</h1>
            <nav>
                <a href="index.html">accueil</a>
                <a href="chat.html">chat</a>
                <a href="post.html">post</a>
                <a href="poster.html">poster</a>
                <a href="profil.html">compte</a>
            </nav>
        </div>
    </header>
    <div class="container" id="message-container"></div>
    <div class="send-message">
        <input type="text" id="user-message" placeholder="envoyer un
message">
        <button id="send-btn">→</button>
    </div>

    <div class="footer">
        <p>© 2023-2024 Hocheline - crée par des <a href="#" id="h2__popup"
class="h2__popup">élèves</a> du Lycée Hoche</p>
    </div>
    <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
    <script src="js/chat.js"></script>
    <script src="js/protected-page.js"></script>
</body>
</html>
```

b) Documentation

HOCHELINE DOCUMENTATION

Le code HTML ci-dessous représente la structure de la page de chat du réseau social "Hocheline". Voici une documentation détaillée expliquant chaque partie du code :

html

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <link rel="stylesheet" href="css/chat.css"> <title>Document</title> </head> <body>

- **<!DOCTYPE html>** : Déclaration du type de document HTML.
- **<html lang="en">** : Balise HTML avec l'attribut lang spécifiant la langue (anglais).
- **<head>** : Section contenant les métadonnées de la page.
    - **<meta charset="UTF-8">** : Définit l'encodage des caractères comme UTF-8.
    - **<meta name="viewport" content="width=device-width, initial-scale=1.0">** : Configuration de la vue pour les appareils mobiles.
    - **<link rel="stylesheet" href="css/chat.css">** : Liaison vers une feuille de style externe ("chat.css").
    - **<title>Document</title>** : Titre de la page.

html

<header> <h1 id="title">Hocheline - Username</h1> <nav> accueil chat post poster compte </nav> </header>

- **<header>** : Section de l'en-tête de la page.
    - ** ** : Conteneur pour l'en-tête.
        - **<h1 id="title">Hocheline - Username</h1>** : Titre principal avec un éventuel nom d'utilisateur.
        - **<nav>** : Navigation contenant des liens vers différentes sections du site.

html

- ** ** : Conteneur principal de la page pour afficher les messages du chat. L'ID "message-container" sera probablement utilisé pour manipuler cet élément dans le script JavaScript associé.

html

 <input type="text" id="user-message" placeholder="envoyer un message"> <button id="send-btn">→</button>

- ** ** : Section pour envoyer des messages.
    - **<input type="text" id="user-message" placeholder="envoyer un message">** : Champ de saisie pour l'utilisateur entrer un message.
    - **<button id="send-btn">→</button>** : Bouton pour envoyer le message.

HOCHELINE DOCUMENTATION

html

 © 2023-2024 Hocheline - crée par des élèves du Lycée Hoche

- ** ** : Pied de page.
    - ** ** : Paragraphe contenant l'année de création et la mention "créé par des élèves du Lycée Hoche".
    - **élèves** : Lien avec une classe pour éventuellement afficher un popup.

html

<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script> <script src="js/chat.js"></script> <script src="js/protected-page.js"></script> </body> </html>

- **<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>** : Inclusion du script client de Socket.io pour la communication en temps réel.
- **<script src="js/chat.js"></script>** : Inclusion du script JavaScript spécifique pour la gestion du chat.
- **<script src="js/protected-page.js"></script>** : Inclusion d'un autre script JavaScript pour les pages protégées.

Résumé :

- La page HTML est structurée avec des sections distinctes pour l'en-tête, le contenu principal (chat), et le pied de page.
- Elle contient un conteneur pour afficher les messages du chat.
- Un champ de saisie et un bouton sont fournis pour que l'utilisateur puisse envoyer des messages.
- Socket.io est inclus pour la communication en temps réel.
- Des scripts JavaScript spécifiques ("chat.js" et "protected-page.js") sont inclus pour gérer les fonctionnalités de la page de chat et des pages protégées, respectivement.

4) post.html

a) Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/post.css">
    <title>Document</title>
</head>
<body>
    <header>
        <div class="header">
            <h1 id="title">Hocheline - Username</h1>
            <nav>
                <a href="index.html">accueil</a>
```

```
                <a href="chat.html">chat</a>
                <a href="post.html">post</a>
                <a href="poster.html">poster</a>
                <a href="profil.html">compte</a>
            </nav>
        </div>
    </header>
    <div class="container" id="post-container">
    </div>

    <div class="footer">
        <p>© 2023-2024 Hocheline - crée par des <a href="#" id="h2__popup"
class="h2__popup">élèves</a> du Lycée Hoche</p>
    </div>
    <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
    <script src="js/post.js"></script>
    <script src="js/protected-page.js"></script>
</body>
</html>
```

b) Documentation

Le code HTML ci-dessous représente la structure de la page des publications du réseau social "Hocheline". Voici une documentation détaillée expliquant chaque partie du code :

htmlCopy code

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <link rel="stylesheet" href="css/post.css"> <title>Document</title> </head> <body>

- **<!DOCTYPE html>** : Déclaration du type de document HTML.
- **<html lang="en">** : Balise HTML avec l'attribut lang spécifiant la langue (anglais).
- **<head>** : Section contenant les métadonnées de la page.
    - **<meta charset="UTF-8">** : Définit l'encodage des caractères comme UTF-8.
    - **<meta name="viewport" content="width=device-width, initial-scale=1.0">** : Configuration de la vue pour les appareils mobiles.
    - **<link rel="stylesheet" href="css/post.css">** : Liaison vers une feuille de style externe ("post.css").
    - **<title>Document</title>** : Titre de la page.

htmlCopy code

<header> <h1 id="title">Hocheline - Username</h1> <nav> accueil chat post poster compte </nav> </header>

- **<header>** : Section de l'en-tête de la page.

- ** ** : Conteneur pour l'en-tête.
    - **<h1 id="title">Hocheline - Username</h1>** : Titre principal avec un éventuel nom d'utilisateur.
    - **<nav>** : Navigation contenant des liens vers différentes sections du site.

htmlCopy code

- ** ** : Conteneur principal de la page pour afficher les publications. L'ID "post-container" sera probablement utilisé pour manipuler cet élément dans le script JavaScript associé.

htmlCopy code

 © 2023-2024 Hocheline - crée par des élèves du Lycée Hoche

- ** ** : Pied de page.
    - ** ** : Paragraphe contenant l'année de création et la mention "créé par des élèves du Lycée Hoche".
    - **élèves** : Lien avec une classe pour éventuellement afficher un popup.

htmlCopy code

<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script> <script src="js/post.js"></script> <script src="js/protected-page.js"></script> </body> </html>

- **<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>** : Inclusion du script client de Socket.io pour la communication en temps réel.
- **<script src="js/post.js"></script>** : Inclusion du script JavaScript spécifique pour la gestion des publications.
- **<script src="js/protected-page.js"></script>** : Inclusion d'un autre script JavaScript pour les pages protégées.

Résumé :

- La page HTML est structurée avec des sections distinctes pour l'en-tête, le contenu principal (publications), et le pied de page.
- Elle contient un conteneur pour afficher les publications.
- Socket.io est inclus pour la communication en temps réel.
- Des scripts JavaScript spécifiques ("post.js" et "protected-page.js") sont inclus pour gérer les fonctionnalités de la page des publications et des pages protégées, respectivement.

5) poster.html

a) Code

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/poster.css">
    <title>Document</title>
</head>
<body>
    <header>
        <div class="header">
            <h1 id="title">Hocheline - Username</h1>
            <nav>
                <a href="index.html">accueil</a>
                <a href="chat.html">chat</a>
                <a href="post.html">post</a>
                <a href="poster.html">poster</a>
                <a href="profil.html">compte</a>
            </nav>
        </div>
    </header>
    <div class="container">
        <h1>Poster une publication</h1>
        <form action="">
            <label for="title">Titre:</label>
            <input type="text" id="title-input" name="title" required>

            <label for="description">Description:</label>
            <textarea id="description" name="description"
required></textarea>

            <label for="image">Image:</label>
            <input type="file" id="image" name="image" accept="image/*"
required>

            <button class="submit">Poster</button>
        </form>
    </div>

    <div class="footer">
        <p>© 2023-2024 Hocheline - crée par des <a href="#" id="h2__popup"
class="h2__popup">élèves</a> du Lycée Hoche</p>
    </div>
    <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
    <script src="js/poster.js"></script>
    <script src="js/protected-page.js"></script>
</body>
</html>
```

b) Documentation

HOCHELINE DOCUMENTATION

Le code HTML ci-dessous représente la structure de la page permettant aux utilisateurs de poster une publication sur le réseau social "Hocheline". Voici une documentation expliquant chaque partie du code :

htmlCopy code

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <link rel="stylesheet" href="css/poster.css"> <title>Document</title> </head> <body>

- **<!DOCTYPE html>** : Déclaration du type de document HTML.
- **<html lang="en">** : Balise HTML avec l'attribut lang spécifiant la langue (anglais).
- **<head>** : Section contenant les métadonnées de la page.
    - **<meta charset="UTF-8">** : Définit l'encodage des caractères comme UTF-8.
    - **<meta name="viewport" content="width=device-width, initial-scale=1.0">** : Configuration de la vue pour les appareils mobiles.
    - **<link rel="stylesheet" href="css/poster.css">** : Liaison vers une feuille de style externe ("poster.css").
    - **<title>Document</title>** : Titre de la page.

htmlCopy code

<header> <h1 id="title">Hocheline - Username</h1> <nav> accueil chat post poster compte </nav> </header>

- **<header>** : Section de l'en-tête de la page.
    - ** ** : Conteneur pour l'en-tête.
        - **<h1 id="title">Hocheline - Username</h1>** : Titre principal avec un éventuel nom d'utilisateur.
        - **<nav>** : Navigation contenant des liens vers différentes sections du site.

htmlCopy code

 <h1>Poster une publication</h1> <form action=""> <label for="title">Titre:</label> <input type="text" id="title-input" name="title" required> <label for="description">Description:</label> <textarea id="description" name="description" required></textarea> <label for="image">Image:</label> <input type="file" id="image" name="image" accept="image/*" required> <button class="submit">Poster</button> </form>

- ** ** : Conteneur principal de la page.
    - **<h1>Poster une publication</h1>** : Titre de la section pour poster une publication.
    - **<form action="">** : Formulaire pour soumettre une publication.
        - **<label for="title">Titre:</label>** : Étiquette pour le champ du titre.
        - **<input type="text" id="title-input" name="title" required>** : Champ de saisie du titre.
        - **<label for="description">Description:</label>** : Étiquette pour le champ de description.

- **<textarea id="description" name="description" required></textarea>** : Champ de saisie pour la description.
- **<label for="image">Image:</label>** : Étiquette pour le champ de téléchargement d'image.
- **<input type="file" id="image" name="image" accept="image/*" required>** : Champ pour télécharger une image.
- **<button class="submit">Poster</button>** : Bouton de soumission du formulaire.

htmlCopy code

  © 2023-2024 Hocheline - crée par des élèves du Lycée Hoche

- ** ** : Pied de page.
    - ** ** : Paragraphe contenant l'année de création et la mention "créé par des élèves du Lycée Hoche".
    - **élèves** : Lien avec une classe pour éventuellement afficher un popup.

htmlCopy code

<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script> <script src="js/poster.js"></script> <script src="js/protected-page.js"></script> </body> </html>

- **<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>** : Inclusion du script client de Socket.io pour la communication en temps réel.
- **<script src="js/poster.js"></script>** : Inclusion du script JavaScript spécifique pour la gestion du formulaire de publication.
- **<script src="js/protected-page.js"></script>** : Inclusion d'un autre script JavaScript pour les pages protégées.

Résumé :

- La page HTML est structurée avec des sections distinctes pour l'en-tête, le contenu principal (formulaire de publication), et le pied de page.
- Elle contient un formulaire pour permettre aux utilisateurs de soumettre une nouvelle publication.
- Socket.io est inclus pour la communication en temps réel.
- Des scripts JavaScript spécifiques ("poster.js" et "protected-page.js") sont inclus pour gérer les fonctionnalités de la page de publication et des pages protégées, respectivement.

III – CSS

Est ici illustré le code css de la page d'accueille (index.html). Toutes les autres pages possèdent un style similaire.

1) Style.css

a) code

```
body{
    background-color: #00212b;
    font-family: Arial, sans-serif;
}
#title{
    margin-left: 1rem;
}
div.header {
    border-color: #046b90;
    text-decoration: none;
    color: #046b90;
    border: solid 5px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}
a {
    color: inherit;
    text-decoration: none;
    margin-left: 2rem;
    margin-right: 2rem;
    font-size: 1.5rem;
}
div.footer{
    padding: 1rem;
    margin-top: 3rem;
    border: solid 5px;
    border-color: #046b90;
    color: #046b90;
    display: flex;
    justify-content: center;
}
div.container{
    margin-top: 2rem;
    height: 45rem;
    max-height: 100%;
    border: solid 5px;
```

```
    border-color: #046b90;
    overflow-y: scroll;
}
div.container::-webkit-scrollbar{
    width: 0px;
}
div.send-message{
    border: solid 5px;
    border-color: #046b90;
    padding: 1rem;
    margin-top: 0.7rem;
    display: flex;
    justify-content: space-between;
}
input{
    background-color: #00212b;
    color: #046b90;
    border: none !important;
    width: 100%;
    height: 2rem;
}
input:focus{
    outline: none;
}
button{
    width: 4rem;
    border: solid 2px #00212b !important;
    background-color: #046b90;
}
button:hover{
    cursor: pointer;
    color: #046b90;
    background-color: #00213b;

}
/* MESSAGE */
.message {
    margin: 0.2rem;
    height: fit-content;
    border: solid 2px #046b90 !important;
    background-color: #00213b;
    border-radius: 10px;
    padding: 10px;
}

.admin2-message {
  color: black;
  font-size: larger;
  background-color: #046b90;
```

```
}
a.h2__popup{
  font-size: medium;
  margin: 0;
}
/* Styles for the popup */
.popup {
    position: fixed;
    color: #046b90;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    padding: 20px;
    background-color: #00212b;
    border: 1px solid #046b90;
    box-shadow: 0 2px 10px #046b90;
    z-index: 1000;
    border: solid 2px;
    border-color: #046b90;
  }

  /* Styles for popup's close button */
  .close {
    position: absolute;
    size: 20px;
    top: 10px;
    right: 10px;
    cursor: pointer;
  }
  .popup__div{
    cursor: move;
  }
```

b) Documentation

- Le fond de la page (**body**) a une couleur **#00212b**.
- La police par défaut est définie comme Arial ou toute police sans empattement (**sans-serif**).
- La section d'en-tête (**div.header**) a une bordure, une couleur de texte, et est stylisée pour afficher une barre de navigation.
- Les liens (**a**) ont une couleur et une mise en page spécifiques.
- La section de pied de page (**div.footer**) a une bordure, une couleur de texte, et est centrée.
- La section principale de la page (**div.container**) a une hauteur fixe avec une barre de défilement invisible.
- La section pour envoyer un message (**div.send-message**) a une bordure, une couleur de fond, et est stylisée pour contenir un champ d'entrée et un bouton.

- Les champs de saisie (**input**) ont une couleur de fond spécifique et aucun contour. Lorsqu'ils sont en surbrillance, ils n'ont pas de contour.
- Les boutons (**button**) ont une largeur fixe, une bordure, et une couleur de fond qui change au survol.
- Les messages (**message**) sont stylisés avec une bordure, une couleur de fond, une bordure arrondie, et un rembourrage.
- Les messages de l'administrateur (**admin2-message**) ont une couleur de texte différente et un fond différent.
- Les liens dans les messages popup (**a.h2__popup**) ont une taille de police spécifique.
- Le popup (**popup**) est stylisé avec une position fixe, une couleur de fond, une bordure, et une ombre.
- Le bouton de fermeture du popup (**close**) a une position absolue et un curseur de type pointeur.
- La classe pour déplacer le popup (**popup__div**) a un curseur de type déplacer.

IV – JavaScript

1) ProtectedPages.js

a) Code

```
const greetHeader = document.getElementById('title');
var popup = document.getElementById("h2__popup");

if (localStorage.getItem('isLoggedIn') == "false") {

    window.location.href = 'login.html';
}else{
    if (window.innerWidth <= 768 && window.innerHeight <= 1024) {
        window.location = 'http://82.121.132.29:3000/phone.html';
    } else {
        greetHeader.innerHTML = "Hocheline - " +
localStorage.getItem('username');
    }
}

popup.addEventListener('click', function(event) {
    createPopup(event);
});


function createPopup(event) {
    const popupContainer = document.createElement('div');
    popupContainer.classList.add('popup');
```

```
    const closeButton = document.createElement('span');
    closeButton.classList.add('close');
    closeButton.innerHTML = '&times;';
    closeButton.addEventListener('click', function () {
      document.body.removeChild(popupContainer);
    });

    const content = document.createElement('div');
    content.classList.add('popup__div');
    content.innerHTML = `
      <p>Ce site a été créé par</p>
      <h3>Charles Dubesset en 2nd7</h3>
    `;

    popupContainer.style.left = `${event.pageX}px`;
    popupContainer.style.top = `${event.pageY}px`;

    let isDragging = false;
    let offsetX, offsetY;

    popupContainer.addEventListener('mousedown', function (mousedownEvent) {
      isDragging = true;

      offsetX = mousedownEvent.clientX -
popupContainer.getBoundingClientRect().left - 121;
      offsetY = mousedownEvent.clientY -
popupContainer.getBoundingClientRect().top - 65;

      document.body.style.userSelect = 'none';
    });

    document.addEventListener('mouseup', function () {
      isDragging = false;

      document.body.style.userSelect = '';
    });

    document.addEventListener('mousemove', function (mousemoveEvent) {
      if (isDragging) {
        popupContainer.style.left = `${mousemoveEvent.clientX - offsetX}px`;
        popupContainer.style.top = `${mousemoveEvent.clientY - offsetY}px`;
      }
    });

    popupContainer.appendChild(closeButton);
    popupContainer.appendChild(content);

    document.body.appendChild(popupContainer);
  }
```

b) Documentation

- La variable greetHeader est utilisée pour sélectionner l'élément avec l'id 'title', généralement le titre de la page.
- La variable popup est utilisée pour sélectionner l'élément avec l'id 'h2__popup'.
- Le code vérifie si l'utilisateur est connecté en vérifiant la valeur dans le stockage local (localStorage), redirigeant vers la page de connexion si nécessaire.
- Si la taille de la fenêtre est inférieure à certaines valeurs, l'utilisateur est redirigé vers une version mobile.
- Le nom d'utilisateur est affiché dans l'en-tête.
- Un écouteur d'événements est ajouté au clic sur l'élément avec l'id 'h2__popup', déclenchant la fonction createPopup.
- La fonction createPopup crée un popup avec un contenu spécifique et la possibilité de le déplacer sur la page.

2) Profil.js

a) Code

```
const disconect = document.getElementById('deco');
const username = document.getElementById('username-input');
const email = document.getElementById('email-input');
const password = document.getElementById('password-input');
const submit = document.getElementById('submit');
const delete_profil = document.getElementById('delete');
const socket = io();

socket.emit("getUserData", localStorage.getItem("username"));

socket.on("fill edit form", (row) => {
    username.value = localStorage.getItem('username');
    email.value = row.email;
    password.value = row.password;
    localStorage.setItem('password', row.password);
    localStorage.setItem('email', row.email);
});

socket.on("profil edited", (data) => {
    localStorage.setItem('username', data.username);
    localStorage.setItem('password', data.password);
    localStorage.setItem('email', data.email);
    username.value = localStorage.getItem('username');
    email.value = localStorage.getItem('email');
    password.value = localStorage.getItem('password');
    window.location.reload();
});
```

```
socket.on("profil deleted", (username) => {
    localStorage.setItem('username', "");
    localStorage.setItem('password', "");
    localStorage.setItem('email', "");
    localStorage.setItem('isLoggedIn', "false");
    window.location.reload();
});

submit.addEventListener('click', function(event){
    if(localStorage.getItem("username") == username.value ||
localStorage.getItem("password") == password.value ||
localStorage.getItem("email") == email.value){
        const data = {
            old_username: localStorage.getItem("username"),
            username: username.value,
            password: password.value,
            email: email.value,
          };
        socket.emit("edit profil", data);
    }
});

delete_profil.addEventListener('click', function(event){
    if (window.confirm("Are you sure you want to delete your account ?")){
        socket.emit("delete profil", localStorage.getItem('username'));
    }
});

disconect.addEventListener('click', function(event){
    localStorage.setItem('isLoggedIn', "false");
    localStorage.setItem('username', "Username");
    window.location.href = 'login.html';
});
```

b) Documentation

- Le script utilise Socket.io pour établir une connexion en temps réel avec le serveur.
- L'événement "getUserData" est émis pour obtenir les données de l'utilisateur actuel.
- Les événements "fill edit form", "profil edited", et "profil deleted" gèrent respectivement le remplissage du formulaire d'édition, la mise à jour du profil, et la suppression du profil.
- Les boutons "Valider les changements" et "Supprimer votre compte" émettent des événements "edit profil" et "delete profil" respectivement.
- Le bouton de déconnexion émet un événement "disconnect" et redirige vers la page de connexion.

3) chat.js

a) Code

```
const socket = io();

const messageContainer = document.getElementById('message-container');
const messageInput = document.getElementById('user-message');
const sendButton = document.getElementById('send-btn');

socket.on('display message', (msgNom) => {
  socket.emit("check admin", msgNom);
});

socket.on("display message checked", (messageData) => {
  const messageElement = document.createElement('div');
  messageElement.classList.add('message');
  messageElement.classList.add('h2__popup');
  messageElement.classList.add(messageData.check === "admin" ? "admin2-
message" : "server-message");
  messageElement.textContent = messageData.message;
  messageContainer.appendChild(messageElement);
  messageContainer.scrollTop = messageContainer.scrollHeight;
});

function sendMessage(){
  const message = messageInput.value;
  if (message.trim() !== '') {
    const msgNom = {
      username: localStorage.getItem("username"),
      message: message
    };
    socket.emit('chat message', msgNom);
    messageInput.value = '';
  }
}

sendButton.addEventListener('click', () => {
  socket.emit('send message');
  sendMessage();
});

messageInput.addEventListener('keydown', (event) => {
  if (event.key === 'Enter') {
    sendMessage();
  }
});
```

```
messageContainer.addEventListener('click', function (event) {
  const clickedElement = event.target.closest('.h2__popup');
  console.log("clicked");

  if (clickedElement && clickedElement.innerText !== "élèves") {
    console.log(clickedElement.innerText.split(" ")[0]);
    const data1 = {
      username: clickedElement.innerText.split(" ")[0],
      event: event
    };
    socket.emit("getUserEmail", (data1));
  }
});

socket.on("popup info", (data) => {
  console.log(data.username, data.event, data.email, data.admin);
  createPopupPost(data.username, data.event, data.email, data.admin);
});

function createPopupPost(username, event, email, admin) {
  const popupContainer = document.createElement('div');
  popupContainer.classList.add('popup');

  const closeButton = document.createElement('span');
  closeButton.classList.add('close');
  closeButton.innerHTML = '&times;'; // '×' symbol for close
  closeButton.addEventListener('click', function () {
    document.body.removeChild(popupContainer);
  });

  const content = document.createElement('div');
  content.classList.add('popup__div');
  content.innerHTML = `
    <p>Username: ${username}</p>
    <p>Email: ${email}</p>
  `;
  if(admin === "true"){
    content.innerHTML = `
    <p>Username: ${username}</p>
    <p>Email: ${email}</p>
    <p>Infos supplémentaires: Cet utilisateur est administrateur</p>
  `;
  }

  popupContainer.style.left = `${event.pageX}px`;
  popupContainer.style.top = `${event.pageY}px`;

  let isDragging = false;
  let offsetX, offsetY;
```

```
    popupContainer.addEventListener('mousedown', function (mousedownEvent) {
      isDragging = true;

      offsetX = mousedownEvent.clientX -
  popupContainer.getBoundingClientRect().left - 200;
      offsetY = mousedownEvent.clientY -
  popupContainer.getBoundingClientRect().top - 80;

      document.body.style.userSelect = 'none';
    });

    document.addEventListener('mouseup', function () {
      isDragging = false;

      document.body.style.userSelect = '';
    });

    document.addEventListener('mousemove', function (mousemoveEvent) {
      if (isDragging) {
        popupContainer.style.left = `${mousemoveEvent.clientX - offsetX}px`;
        popupContainer.style.top = `${mousemoveEvent.clientY - offsetY}px`;
      }
    });

    popupContainer.appendChild(closeButton);
    popupContainer.appendChild(content);

    document.body.appendChild(popupContainer);
  }
```

b) Documentation

- Le script utilise Socket.io pour établir une connexion en temps réel avec le serveur.
- Les événements "display message" et "display message checked" gèrent respectivement l'affichage des messages et la vérification des messages pour les administrateurs.
- La fonction sendMessage est appelée pour envoyer un message lorsque le bouton d'envoi est cliqué ou lorsque la touche "Enter" est enfoncée.
- Un écouteur d'événement sur la zone de message permet également d'envoyer un message lorsque la touche "Enter" est enfoncée.
- L'événement "click" sur le conteneur de messages est utilisé pour afficher les informations de l'utilisateur en popup.
- La fonction createPopupPost crée la popup avec les informations de l'utilisateur lorsqu'elle est appelée.

4) post.js

a) Code

```
const socks = io();
const container = document.getElementById("post-container");
const popups = document.getElementsByClassName("h2__popup");

socks.emit('register post', "")

socks.on("display post", (postList) => {
  var rumor = document.getElementById("post-container");
  rumor.innerHTML = "";
  for (var element of postList) {
    const article = document.createElement('div');
    article.classList.add("cta");
    article.innerHTML = element;
    rumor.appendChild(article);
  }
});

container.addEventListener('click', function (event) {
  const clickedElement = event.target.closest('.h2__popup');

  if (clickedElement && clickedElement.innerText !== "élèves") {
    console.log(clickedElement.innerText.split(" ")[0]);
    const data1 = {
      username: clickedElement.innerText.split(" ")[0],
      event: event
    };
    socks.emit("getUserEmail", (data1));
  }
});

socks.on("popup info", (data) => {
  console.log(data.username, data.event, data.email, data.admin);
  createPopupPost(data.username, data.event, data.email, data.admin);
});

function createPopupPost(username, event, email, admin) {
  const popupContainer = document.createElement('div');
  popupContainer.classList.add('popup');

  const closeButton = document.createElement('span');
  closeButton.classList.add('close');
  closeButton.innerHTML = '&times;'; // '×' symbol for close
  closeButton.addEventListener('click', function () {
```

```
      document.body.removeChild(popupContainer);
    });

    const content = document.createElement('div');
    content.classList.add('popup__div');
    content.innerHTML = `
      <p>Username: ${username}</p>
      <p>Email: ${email}</p>
    `;
    if(admin === "true"){
      content.innerHTML = `
      <p>Username: ${username}</p>
      <p>Email: ${email}</p>
      <p>Infos supplémentaires: Cet utilisateur est administrateur</p>
    `;
    }

    popupContainer.style.left = `${event.pageX}px`;
    popupContainer.style.top = `${event.pageY}px`;

    let isDragging = false;
    let offsetX, offsetY;

    popupContainer.addEventListener('mousedown', function (mousedownEvent) {
      isDragging = true;

      offsetX = mousedownEvent.clientX -
  popupContainer.getBoundingClientRect().left - 200;
      offsetY = mousedownEvent.clientY -
  popupContainer.getBoundingClientRect().top - 80;

      document.body.style.userSelect = 'none';
    });

    document.addEventListener('mouseup', function () {
      isDragging = false;

      document.body.style.userSelect = '';
    });

    document.addEventListener('mousemove', function (mousemoveEvent) {
      if (isDragging) {
        popupContainer.style.left = `${mousemoveEvent.clientX - offsetX}px`;
        popupContainer.style.top = `${mousemoveEvent.clientY - offsetY}px`;
      }
    });

    popupContainer.appendChild(closeButton);
    popupContainer.appendChild(content);
```

```
    document.body.appendChild(popupContainer);
  }
```

b) Documentation

- Le script utilise Socket.io pour établir une connexion en temps réel avec le serveur.
- L'événement "register post" est émis pour sauvegarder les posts.
- L'événement "display post" est écouté pour afficher les publications lorsqu'elles sont mises à jour.
- L'événement "click" sur le conteneur de publications est utilisé pour afficher les informations de l'utilisateur en popup.
- La fonction createPopupPost crée la popup avec les informations de l'utilisateur lorsqu'elle est appelée.

5) poster.js

a) Code

```
const socks = io();

const submit = document.querySelector('.submit');

submit.addEventListener('click', () => {

  var titleInput = document.getElementById('title-input');
  var descriptionInput = document.getElementById('description');
  var imageInput = document.getElementById('image');

  var titleValue = titleInput.value;
  var descriptionValue = descriptionInput.value;
  var selectedFile = imageInput.files[0];

   var formData = new FormData();
   formData.append('title', titleValue);
   formData.append('description', descriptionValue);
   formData.append('image', selectedFile);

   fetch('/upload', {
     method: 'POST',
     body: formData,
   })
     .then((response) => response.text())
     .then((data) => {
       console.log(data);
     })
     .catch((error) => {
       console.error('Erreur:', error);
     });

  var article = document.createElement('div');
  article.classList.add('cta');
```

```
    var textColumn = document.createElement('div');
    textColumn.classList.add('cta__text-column');
    var h2 = document.createElement('h2');
    h2.classList.add("h2__popup")
    if (titleValue) {
      h2.textContent = localStorage.getItem("username") + " >>> " +
titleValue;
    }
    var p = document.createElement('p');
    if (descriptionValue) {
      p.textContent = descriptionValue;
    }
    var img = document.createElement('img');
    if (selectedFile) {
      var imageUrl = URL.createObjectURL(selectedFile);
      img.src = "../images/" + selectedFile.name;
    }
    textColumn.appendChild(h2);
    textColumn.appendChild(p);
    article.appendChild(img);
    article.appendChild(textColumn);

    socks.emit("register post", article.innerHTML);

    titleInput.value = "";
    descriptionInput.value = "";
    imageInput.value = "";

    window.location.reload();
    window.location.href = "post.html";
});
```

b) Documentation

- Le script utilise Socket.io pour établir une connexion en temps réel avec le serveur.
- L'événement "click" sur le bouton de soumission déclenche l'envoi des données du formulaire au serveur via l'API Fetch.
- Une fois les données envoyées avec succès, un nouvel élément d'article est créé pour représenter la publication localement.
- Cet élément est ajouté à la page pour afficher la publication sans avoir à recharger la page.
- Enfin, un événement Socket.io est émis pour enregistrer la nouvelle publication sur le serveur, permettant aux autres utilisateurs de voir la publication en temps réel.

6) login.js

a) Code

```
const submit = document.getElementById("submit");
const socket = io();

submit.addEventListener('click', () => {
    var username = document.getElementById("username").value;
    var password = document.getElementById("password").value;
    socket.emit('login', username, password);
    document.getElementById("username").value = "";
    document.getElementById("password").value = "";
});

socket.on('redirect', (data) => {
    localStorage.setItem('isLoggedIn', 'true');
    localStorage.setItem('username', data.username);
    window.location.href = data.url;
});
```

b) Documentation

- Le script utilise Socket.io pour établir une connexion en temps réel avec le serveur.
- L'événement "click" sur le bouton de soumission déclenche l'envoi des données du formulaire au serveur via l'API Fetch.
- Une fois les données envoyées avec succès, un nouvel élément d'article est créé pour représenter la publication localement.
- Cet élément est ajouté à la page pour afficher la publication sans avoir à recharger la page.
- Enfin, un événement Socket.io est émis pour enregistrer la nouvelle publication sur le serveur, permettant aux autres utilisateurs de voir la publication en temps réel.

7) sign_up.js

a) Code

```
const submit = document.getElementById("submit");
const socket = io();

submit.addEventListener('click', () => {
    const username = document.getElementById("username").value;
    const password = document.getElementById("password").value;
```

```
    const email = document.getElementById("email").value;

    const userData = {
        username: username,
        password: password,
        email: email
    };

    socket.emit("sign up", userData);
    socket.emit("login", username, password);
    localStorage.setItem("isLoggedIn", "true");
    localStorage.setItem("username", username);
    localStorage.setItem("password", password);
    localStorage.setItem("email", email);
    window.location.href = 'index.html';

    document.getElementById("username").value = "";
    document.getElementById("password").value = "";
    document.getElementById("email").value = "";
});
```

b) Documentation

- Le script utilise Socket.io pour établir une connexion en temps réel avec le serveur.
- L'écouteur d'événement est attaché au bouton de soumission du formulaire de connexion.
- Lorsque l'utilisateur clique sur le bouton, le script récupère les valeurs des champs du formulaire (nom d'utilisateur et mot de passe).
- Ensuite, il émet un événement 'login' avec ces informations d'identification vers le serveur via Socket.io.
- Le serveur traite les informations d'identification et, en cas de succès, émet un événement 'redirect' avec les données nécessaires pour rediriger l'utilisateur.
- L'écouteur d'événement 'redirect' sauve les informations de connexion dans le stockage local et redirige l'utilisateur vers l'URL spécifiée par le serveur.

V – Conclusion

L'application Hocheline offre une plateforme interactive et dynamique pour la communication sociale. Son architecture repose sur plusieurs technologies clés, combinant l'utilisation de HTML, CSS, JavaScript côté client, avec une communication en temps réel via Socket.io côté serveur. Cette combinaison crée une expérience utilisateur fluide et interactive. Examinons de plus près les aspects clés de son fonctionnement.

1. Structuration HTML et CSS :

- L'application utilise une structuration HTML claire et sémantique, définissant des sections distinctes pour l'en-tête, le contenu principal et le pied de page.
- Les classes CSS spécifiques sont employées pour styliser chaque section, assurant une présentation visuelle cohérente.

2. Communication en Temps Réel avec Socket.io :

- Socket.io est intégré pour permettre une communication bidirectionnelle en temps réel entre le client et le serveur.
- Cela se manifeste dans diverses fonctionnalités, notamment le chat en direct, les mises à jour en temps réel des publications, et la gestion des utilisateurs connectés.

3. Fonctionnalités Clés :

- **Inscription et Connexion :** Les utilisateurs peuvent s'inscrire et se connecter de manière transparente grâce à des formulaires interactifs.

- **Chat en Direct :** La fonction de chat en direct permet aux utilisateurs de communiquer instantanément avec d'autres utilisateurs connectés.
- **Publication de Contenu :** Les utilisateurs peuvent publier des messages et des images, créant un flux de contenu dynamique.
- **Édition de Profil :** Les utilisateurs ont la possibilité de personnaliser leur profil, y compris la modification des informations d'identification.

4. Interface Utilisateur Intuitive :

- L'interface utilisateur est conçue de manière intuitive avec une navigation fluide entre les différentes sections de l'application.
- Des éléments interactifs, tels que des liens de navigation, des boutons et des formulaires, facilitent la navigation et l'interaction.

5. Gestion d'Utilisateurs :

- Le serveur gère efficacement les données utilisateur, y compris l'enregistrement des utilisateurs, la gestion des connexions et la récupération d'informations de profil.

6. Sécurité et Contrôle d'Accès :

- Des mécanismes de sécurité tels que la gestion des sessions et les vérifications côté serveur sont mis en place pour assurer un accès sécurisé aux fonctionnalités sensibles.