Documentation Deuscord

Pap Alexandre, Sailly Kevin, Fermon Romain

BUT3 TD1

Table des matières

Modules

authentification.js

Gérer l'enregistrement et la connexion des utilisateurs.

Dépendances utilisées
- bcrypt : Module pour le hachage sécurisé des mots de passe.
- mysql : Module pour interagir avec la base de données MySQL.

Structure de la base de données utilisée :
Une table nommée `user` avec les colonnes suivantes :
- `id` (identifiant unique de l'utilisateur)
- `pseudo` (pseudo de l'utilisateur)
- `email` (adresse e-mail de l'utilisateur)
- `password` (mot de passe haché de l'utilisateur)
- `creation_date` (date de création du compte utilisateur)

Fichiers requis :
- mysql_connection.js : Fichier contenant les informations de connexion à la base de données MySQL.

Fonction exportée :
register(req, res, connection) :
Fonction exportée qui est appelée lorsqu'un utilisateur essaie de s'inscrire.
Paramètres :
req : objet de requête HTTP contenant les données envoyées par l'utilisateur.
res : objet de réponse HTTP pour envoyer une réponse à l'utilisateur.
connection : objet de connexion à la base de données SQL.
Fonctionnement :
La fonction register(req, res, connection) vérifie :
- la validité du pseudo et de l'adresse e-mail de l'utilisateur.
- si le pseudo ou l'adresse e-mail existe déjà dans la base de données.
- si non trouvé, elle enregistre l'utilisateur dans la base de données après avoir haché son mot de passe.
La fonction stocke le pseudo et l'ID de l'utilisateur dans la session et redirige l'utilisateur vers `/app` après l'inscription réussie.

Fonction exportée :
login(req, res, connection)
Fonction exportée qui est appelée lorsqu'un utilisateur essaie de se connecter.
Paramètres :
req : objet de requête HTTP contenant les données envoyées par l'utilisateur.
res : objet de réponse HTTP pour envoyer une réponse à l'utilisateur.
connection : objet de connexion à la base de données SQL.
Fonctionnement :
La fonction login(req, res, connection) vérifie :
- validité du login (pseudo ou e-mail) de l'utilisateur.
- vérifie si l'utilisateur existe dans la base de données.
- compare le mot de passe fourni avec celui stocké dans la base de données après le hachage.

- si les informations de connexion sont correctes, stocke le pseudo et l'ID de l'utilisateur dans la session et redirige vers `/app`.
- sinon, elle redirige vers `/register` avec un message d'erreur.

images.js

Gère la récupération et l'affichage des images.

Dépendances utilisées :
- fs : Module FileSystem pour interagir avec les fichiers sur le serveur.
- path`: Module pour la gestion des chemins d'accès aux fichiers.

Type de contenu supproté :
- HTML: 'text/html'
- Texte: 'text/plain'
- CSS: 'text/css'
- GIF: 'image/gif'
- JPEG: 'image/jpeg'
- PNG: 'image/png'
- SVG: 'image/svg+xml'
- JavaScript: 'application/javascript'

Fonction exportée :
getImage(req, res) :
Cette fonction est appelée lorsqu'un utilisateur demande une image.
Paramètres :
- req: objet de requête HTTP contenant les données envoyées par l'utilisateur, notamment le nom de l'image demandée.
- res: objet de réponse HTTP pour envoyer une réponse à l'utilisateur.
Fonctionnement :
- Vérifie la validité du nom de l'image demandée pour éviter les failles.
- Détermine le type de contenu de l'image en fonction de son extension.
- Crée un flux de lecture (`ReadStream`) pour envoyer le fichier image au client.
- En cas de succès, configure l'en-tête de réponse avec le type de contenu et envoie l'image au client.
- En cas d'erreur de lecture ou de fichier introuvable, renvoie une réponse 404 avec le message "Not found".

Sécurité :
La fonction getImage(req, res) effectue une vérification sur le nom de l'image demandée pour éviter les failles de sécurité potentielles.

Gestion des Erreurs :
Si l'image demandée n'est pas trouvée ou s'il y a une erreur lors de la lecture du fichier, la fonction getImage(req, res) renvoie une réponse 404 avec le message "Not found".

mysql_connection.js

Fonction exportée :
getConnexion() :
Cette fonction crée et retourne une connexion à la base de données MySQL.

Dépendances Utilisées :
- `mysql`: Module pour interagir avec la base de données MySQL.

Configuration de la Base de Données :
- Hôte (`host`): 127.0.0.1 (localhost).
- Utilisateur (`user`): root (utilisateur par défaut de MySQL).
- Mot de passe (`password`): Vide (pas de mot de passe défini).
- Base de données (`database`): deuscord.

Sécurité :
- Aucune gestion de mot de passe n'est implémentée dans cette configuration, ce qui peut être un problème de sécurité.

Paramètres :
Aucun
Retour :
Objet de connexion à la base de données SQL.
Fonctionnement :
- Utilise `mysql.createConnection()` pour créer une nouvelle connexion à la base de données.
- Paramètres de connexion :
- `host`: L'adresse IP du serveur MySQL (127.0.0.1 dans ce cas, ce qui correspond à la machine locale).
- `user`: Le nom d'utilisateur pour se connecter à la base de données (root dans ce cas).
- `password`: Le mot de passe pour l'utilisateur (vide dans ce cas).
- database : Le nom de la base de données à laquelle se connecter (deuscord dans ce cas).
- Retourne l'objet de connexion créé.

socket_channels_functions.js

Gère la création, la récupération, la mise à jour et la suppression de canaux de discussion.

Fonctions Exportées :

1. create_chanel(name, description, position, connection, socket, callback) :
   - Cette fonction est appelée pour créer un nouveau canal de discussion.
   - **Paramètres :**
     - `name`: Nom du canal.
     - `description`: Description du canal.
     - `position`: Position du canal.
     - `connection`: Objet de connexion à la base de données MySQL.
     - `socket`: Objet socket.io.
     - `callback`: Fonction de retour appelée après l'opération.
   - **Fonctionnement :**

- Vérifie la validité du nom du canal.
- Insère les données du canal dans la base de données.
- Appelle le callback avec un statut OK ou une erreur.

2. get_chanel_list(connection, socket, callback) :
- Cette fonction est appelée pour récupérer la liste des canaux de discussion.
- **Paramètres :**
  - `connection`: Objet de connexion à la base de données MySQL.
  - `socket`: Objet socket.io.
  - `callback`: Fonction de retour appelée après l'opération.
- **Fonctionnement :**
  - Récupère la liste des canaux de discussion depuis la base de données.
  - Appelle le callback avec la liste des canaux ou une erreur le cas échéant.

3. delete_chanel(chanel_id, connection, socket, callback) :
- Cette fonction est appelée pour supprimer un canal de discussion.
- **Paramètres :**
  - `chanel_id`: Identifiant du canal à supprimer.
  - `connection`: Objet de connexion à la base de données MySQL.
  - `socket`: Objet socket.io.
  - `callback`: Fonction de retour appelée après l'opération.
- **Fonctionnement :**
  - Supprime le canal de discussion de la base de données, ainsi que les messages associés.
  - Appelle le callback avec un statut OK ou une erreur.

4. update_chanel(name, description, position, chanel_id, connection, socket, callback) :
- Cette fonction est appelée pour mettre à jour les informations d'un canal de discussion.
- **Paramètres :**
  - `name`: Nouveau nom du canal.
  - `description`: Nouvelle description du canal.
  - `position`: Nouvelle position du canal.
  - `chanel_id`: Identifiant du canal à mettre à jour.
  - `connection`: Objet de connexion à la base de données MySQL.
  - `socket`: Objet socket.io.
  - `callback`: Fonction de retour appelée après l'opération.
- **Fonctionnement :**
  - Vérifie la validité des nouvelles informations du canal.
  - Met à jour les données du canal dans la base de données.
  - Appelle le callback avec un statut OK ou une erreur.

Dépendances Utilisées :
- `mysql`: Module pour interagir avec la base de données MySQL.

Sécurité :
- Les fonctions sont en partie sécurisées en vérifiant la validité des entrées utilisateur (nom du canal).

Gestion des Erreurs :
- Les erreurs liées à la base de données sont gérées et loguées.

socket_function.js

Gère la création, la récupération, la mise à jour et la suppression de canaux de discussion.

Fonctions Exportées :

1. create_chanel(name, description, position, connection, socket, callback) :
   - Cette fonction est appelée pour créer un nouveau canal de discussion.
   - **Paramètres :**
     - `name`: Nom du canal.
     - `description`: Description du canal.
     - `position`: Position du canal.
     - `connection`: Objet de connexion à la base de données MySQL.
     - `socket`: Objet socket.io.
     - `callback`: Fonction de retour appelée après l'opération.
   - **Fonctionnement :**
     - Vérifie la validité du nom du canal.
     - Insère les données du canal dans la base de données.
     - Appelle le callback avec un statut OK ou une erreur.

2. get_chanel_list(connection, socket, callback) :
   - Cette fonction est appelée pour récupérer la liste des canaux de discussion.
   - **Paramètres :**
     - `connection`: Objet de connexion à la base de données MySQL.
     - `socket`: Objet socket.io.
     - `callback`: Fonction de retour appelée après l'opération.
   - **Fonctionnement :**
     - Récupère la liste des canaux de discussion depuis la base de données avec les informations suivantes : ID du canal, nom, sujet, propriétaire, position et pseudo du propriétaire.
     - Appelle le callback avec la liste des canaux ou une erreur le cas échéant.

3. delete_chanel(chanel_id, connection, socket, callback) :
   - Cette fonction est appelée pour supprimer un canal de discussion.
   - **Paramètres :**
     - `chanel_id`: Identifiant du canal à supprimer.
     - `connection`: Objet de connexion à la base de données MySQL.
     - `socket`: Objet socket.io.
     - `callback`: Fonction de retour appelée après l'opération.
   - **Fonctionnement :**
     - Supprime le canal de discussion de la base de données.
     - Appelle le callback avec un statut OK ou une erreur.

4. update_chanel(name, description, position, chanel_id, connection, socket, callback) :
   - Cette fonction est appelée pour mettre à jour les informations d'un canal de discussion.
   - **Paramètres :**
     - `name`: Nouveau nom du canal.
     - `description`: Nouvelle description du canal.
     - `position`: Nouvelle position du canal.
     - `chanel_id`: Identifiant du canal à mettre à jour.
     - `connection`: Objet de connexion à la base de données MySQL.

- `socket`: Objet socket.io.
- `callback`: Fonction de retour appelée après l'opération.

- Fonctionnement :
- Vérifie la validité des nouvelles informations du canal.
- Met à jour les données du canal dans la base de données.
- Appelle le callback avec un statut OK ou une erreur.

Dépendances Utilisées :
- `mysql`: Module pour interagir avec la base de données MySQL.

Sécurité :
- Les fonctions sont en partie sécurisées en vérifiant la validité des entrées utilisateur (nom du canal, position).
- Des vérifications de permissions sont mentionnées mais pas encore implémentées (marquées comme `TODO`).

Gestion des Erreurs :
- Les erreurs liées à la base de données sont gérées et loguées.

socket_messages_functions.js

Gère les actions liées aux canaux de discussion, telles que le changement de canal et l'envoi de messages.

Fonctions Exportées :
1. change_chanel(chanel_id, connection, socket, io, callback) :
   - Cette fonction est appelée lorsque le client souhaite changer de canal.
   - Paramètres :
   - `chanel_id`: Identifiant du canal auquel le client souhaite se connecter.
   - `connection`: Objet de connexion à la base de données MySQL.
   - `socket`: Objet socket.io représentant la connexion du client.
   - `io`: Objet socket.io utilisé pour communiquer avec tous les clients.
   - `callback`: Fonction de retour appelée après l'opération.
   - Fonctionnement :
   - Vérifie la validité de l'identifiant du canal.
   - Quitte tous les canaux actuels du client.
   - Rejoint le nouveau canal spécifié par `chanel_id`.
   - Récupère les 50 derniers messages du canal depuis la base de données.
   - Envoie les messages récupérés au client via le callback.

2. send_message(message, connection, socket, io, callback) :
   - Cette fonction est appelée lorsque le client envoie un message dans un canal.
   - Paramètres :
   - `message`: Contenu du message envoyé par le client.
   - `connection`: Objet de connexion à la base de données MySQL.
   - `socket`: Objet socket.io représentant la connexion du client.
   - `io`: Objet socket.io utilisé pour communiquer avec tous les clients.
   - `callback`: Fonction de retour appelée après l'opération.
   - Fonctionnement :
   - Vérifie si la longueur du message est inférieure ou égale à 2000 caractères.
   - Insère le message dans la base de données avec l'identifiant du canal actuel du client.

- Envoie le message à tous les clients dans le canal actuel via `io.emit`.

Fonctions Non Exportées :

1. clear_client_rooms(socket) :
   - Cette fonction est utilisée pour quitter tous les canaux actuels du client, sauf celui par défaut.
   - **Paramètres :**
     - `socket`: Objet socket.io représentant la connexion du client.

2. get_client_chanel(socket) :
   - Cette fonction est utilisée pour obtenir le nom du canal actuel du client.
   - **Paramètres :**
     - `socket`: Objet socket.io représentant la connexion du client.
   - **Retour :** Nom du canal actuel du client.

3. get_client_name(socket) :
   - Cette fonction est utilisée pour obtenir le pseudo du client connecté.
   - **Paramètres :**
     - `socket`: Objet socket.io représentant la connexion du client.
   - **Retour :** Pseudo du client.

Dépendances Utilisées :
- `mysql`: Module pour interagir avec la base de données MySQL.

Sécurité :
- Les messages sont limités à 2000 caractères pour éviter les abus.

Gestion des Erreurs :
- Les erreurs liées à la base de données sont gérées et loguées.

Vues

chanels_list.ejs

Gère la création et la mise à jour de la liste des canaux de discussion.

Fonctions :

1. add_chanel_div(nom, id, description, owner, position, pseudo) :
   - Cette fonction ajoute un nouveau canal à la liste des canaux.
   - **Paramètres :**
     - `nom` : Nom du canal.
     - `id` : Identifiant unique du canal.
     - `description` : Description du canal.
     - `owner` : ID du propriétaire du canal.
     - `position` : Position du canal.
     - `pseudo` : Pseudo du propriétaire du canal.
   - **Fonctionnement :**
     - Clone le modèle de div de canal.

- Affiche la nouvelle div de canal.
- Affecte les valeurs aux champs cachés correspondants.
- Ajoute la nouvelle div de canal à la liste des canaux.

2. clear_chanel_list() :
- Cette fonction efface tous les canaux de la liste des canaux.
- **Fonctionnement :**
  - Supprime tous les éléments ayant la classe "chanel" qui sont enfants de l'élément ayant l'ID "chanels_list".

3. create_chanel_list() :
- Cette fonction utilise Socket.io pour demander la liste des canaux au serveur et les afficher.
- **Fonctionnement :**
  - Émet un événement "get_chanel_list" au serveur via Socket.io.
  - Pour chaque canal dans la réponse du serveur, appelle `add_chanel_div()` pour l'ajouter à la liste des canaux.

4. regen_chanel_list() :
- Cette fonction efface la liste actuelle des canaux et en recrée une nouvelle.
- **Fonctionnement :**
  - Appelle d'abord `clear_chanel_list()` pour effacer la liste des canaux.
  - Ensuite, appelle `create_chanel_list()` pour recréer la liste des canaux.

Dépendances Utilisées :
- JQuery est nécessaire pour l'utilisation des fonctions `$()`.

Socket.io :
- Utilisé pour interagir avec le serveur pour obtenir la liste des canaux de discussion.

Fonctionnement :
- Lorsque la page est chargée ou lorsqu'une mise à jour est nécessaire, la fonction `regen_chanel_list()` est appelée pour rafraîchir la liste des canaux de discussion en demandant au serveur la liste mise à jour.

cookies.ejs

Manipule les cookies dans le navigateur.

Fonctions :

1. createCookie(name, value, days) :
- Cette fonction crée un nouveau cookie avec le nom spécifié, la valeur donnée et une durée de vie optionnelle.
- **Paramètres :**
  - `name` : Nom du cookie.
  - `value` : Valeur du cookie.
  - `days` : Durée de vie du cookie en jours (optionnel).
- **Fonctionnement :**
  - Si `days` est spécifié, calcule la date d'expiration en ajoutant le nombre de jours à la date actuelle.

- Crée une chaîne de caractères pour l'expiration du cookie.
- Définit le cookie avec le nom, la valeur et l'expiration spécifiés, ainsi que le chemin "/" pour le rendre accessible à tout le site.

2. readCookie(name) :
- Cette fonction lit la valeur d'un cookie spécifié par son nom.
- **Paramètres :**
  - `name` : Nom du cookie à lire.
- **Retour :** La valeur du cookie s'il existe, sinon `null`.
- **Fonctionnement :**
  - Divise la chaîne des cookies actuels en un tableau.
  - Parcourt le tableau pour trouver le cookie avec le nom spécifié.
  - Retourne la valeur du cookie si trouvé, sinon `null`.

3. eraseCookie(name) :
- Cette fonction supprime un cookie en définissant sa durée de vie à une date passée.
- **Paramètres :**
  - `name` : Nom du cookie à supprimer.
- **Fonctionnement :**
  - Appelle `createCookie()` avec une durée de vie négative, ce qui entraîne l'expiration immédiate du cookie.

Utilisation :
- `createCookie()` est utilisée pour créer de nouveaux cookies.
- `readCookie()` est utilisée pour lire la valeur d'un cookie existant.
- `eraseCookie()` est utilisée pour supprimer un cookie en le rendant expiré.

messages.ejs

Gére l'affichage des messages dans une interface utilisateur et pour communiquer avec le serveur via Socket.io pour le changement de canal et l'envoi de messages.

Fonctions :

1. add_message(pseudo, text, id) :
- Cette fonction ajoute un nouveau message à l'interface utilisateur.
- **Paramètres :**
  - `pseudo` : Pseudo de l'utilisateur ayant envoyé le message.
  - `text` : Contenu du message.
  - `id` : Identifiant unique du message.
- **Fonctionnement :**
  - Clone le modèle de div de message.
  - Affiche la nouvelle div de message.
  - Affecte les valeurs aux champs de pseudo et de texte correspondants.
  - Ajoute la nouvelle div de message à l'interface utilisateur.

2. clear_messages() :
- Cette fonction efface tous les messages de l'interface utilisateur.
- **Fonctionnement :**
  - Supprime tous les éléments ayant la classe "message" qui sont enfants de l'élément ayant l'ID

"central_div".

3. changer_de_chanel(chanel_id) :
  - Cette fonction utilise Socket.io pour demander au serveur de changer de canal.
  - Paramètres :
    - `chanel_id` : Identifiant du canal vers lequel changer.
  - Fonctionnement :
    - Émet un événement "change_chanel" au serveur via Socket.io avec l'identifiant du canal.
    - Une fois la réponse reçue, si le statut est "OK", efface les messages actuels et affiche les nouveaux messages.
    - Si le statut est "Erreur", affiche une alerte avec l'information d'erreur.

4. send_message(message) :
  - Cette fonction utilise Socket.io pour envoyer un message au serveur.
  - Paramètres :
    - `message` : Contenu du message à envoyer.
  - Fonctionnement :
    - Émet un événement "send_message" au serveur via Socket.io avec le contenu du message.
    - Une fois la réponse reçue, si le statut est "OK", le message est envoyé avec succès.
    - Si le statut est "Erreur", affiche une alerte avec l'information d'erreur.

Utilisation :
- Ces fonctions sont appelées en réponse à des événements utilisateur, tels que le changement de canal ou l'envoi d'un message.

Dépendances Utilisées :
- JQuery est utilisé pour manipuler le DOM et pour émettre des événements via Socket.io.

app.ejs

Interface utilisateur pour une application de messagerie basée sur des canaux.

Structure HTML :

1. Divisions Principales :
  - `server_div` : Contient un lien pour les paramètres du serveur.
  - `chanel_list` : Contient la liste des canaux.
  - `chanel_infos` : Affiche les informations sur le canal sélectionné.
  - `central_div` : Contient les messages échangés dans le canal sélectionné.
  - `input_message` : Permet à l'utilisateur d'entrer un nouveau message.
  - `online_list` : Liste des utilisateurs en ligne.
  - `user_area` : Affiche le nom de l'utilisateur connecté.

2. Modèles :
  - `chanel` : Modèle de canal. Cloné pour chaque canal affiché.
  - `message` : Modèle de message. Cloné pour chaque message affiché.

Fonctionnalités JavaScript :

1. Gestion des Clics sur les Panels :

- Les clics sur les flèches des panels (`div_chanels_panel` et `div_online_panel`) entraînent l'ouverture/la fermeture des panels correspondants.
- Les panels peuvent être ouverts ou fermés en fonction de leur état actuel.

2. Affichage des Canaux et des Messages :
- Lorsqu'un canal est sélectionné dans la liste des canaux, les informations du canal sont affichées dans `chanel_infos` et les messages associés sont affichés dans `central_div`.
- L'élément `message_text_input` est affiché lorsque le canal est sélectionné, permettant à l'utilisateur d'entrer un nouveau message.

3. Gestion des Événements Utilisateur :
- Lorsque l'utilisateur clique sur un canal dans la liste des canaux, la fonction `changer_de_chanel()` est appelée pour changer de canal.
- Lorsque l'utilisateur appuie sur la touche "Entrée" dans `message_text_input`, la fonction `send_message()` est appelée pour envoyer le message.

4. Communication avec le Serveur via Socket.io :
- Les événements "recive_message" sont écoutés pour afficher les nouveaux messages reçus en temps réel.

Dépendances Utilisées :
- jQuery est utilisé pour manipuler le DOM et pour gérer les événements utilisateur.
- Socket.io est utilisé pour la communication en temps réel avec le serveur.

chanels_admin.ejs

Interface utilisateur pour la gestion des canaux.

Structure HTML :

1. Divisions Principales :
- `central_div` : Contient les éléments de l'interface pour la gestion des canaux.

2. Sous-Divisions :
- `div_left_pane` : Contient des liens et des éléments pour ajouter et fermer des canaux.
- `chanels_div` : Contient la liste des canaux.

3. Modèles :
- `chanel` : Modèle de canal. Cloné pour chaque canal affiché.

4. Formulaire pour la Modification des Canaux :
- `chanel_form` : Permet de modifier ou de créer un nouveau canal.
- `chanel_modify_submit_button` : Bouton pour soumettre les modifications ou la création du canal.

Fonctionnalités JavaScript :

1. Gestion des Événements Utilisateur :
- Lorsque l'utilisateur clique sur un canal, ses informations sont chargées dans le formulaire de modification.

- Lorsque l'utilisateur clique sur le bouton "Ajouter Canal", un formulaire vide est affiché pour créer un nouveau canal.

2. Communication avec le Serveur via Socket.io :
- Lorsque l'utilisateur supprime un canal, une requête est envoyée au serveur pour effectuer la suppression.
- Lorsque l'utilisateur soumet le formulaire de modification/création de canal, une requête est envoyée au serveur pour effectuer la modification/création.

Dépendances Utilisées :
- jQuery est utilisé pour manipuler le DOM et pour gérer les événements utilisateur.
- Socket.io est utilisé pour la communication en temps réel avec le serveur.

login.ejs

Page de connexion.

Structure HTML :

1. Divisions Principales :
- `central_div` : Contient les éléments de la page de connexion.

2. Formulaire de Connexion :
- `form` : Permet à l'utilisateur de saisir son login et son mot de passe.
- `input` : Champs pour le login et le mot de passe.
- `button` : Bouton de soumission du formulaire.

3. Messages d'Erreur :
- `error_div` : Div pour afficher les messages d'erreur.
- `error_p` : Paragraphe pour afficher le message d'erreur.

4. Liens de Navigation :
- `a` : Liens pour récupérer un mot de passe oublié et pour accéder à la page d'inscription.

Fonctionnalités JavaScript :
- Le script récupère les éventuels messages d'erreur dans l'URL (paramètre `error`) et affiche le message d'erreur correspondant à l'utilisateur.
- Les messages d'erreur sont affichés dans la `error_div` en fonction du type d'erreur.

register.ejs

Page d'inscription.

Structure HTML :

1. Divisions Principales :
- `central_div` : Contient les éléments de la page d'inscription.

2. Formulaire d'Inscription :
- `form` : Permet à l'utilisateur de saisir son pseudo, son adresse e-mail et son mot de passe.
- `input` : Champs pour le pseudo, l'adresse e-mail et les mots de passe (initial et de confirmation).
- `button` : Bouton de soumission du formulaire.

3. Messages d'Erreur :
- `error_div` : Div pour afficher les messages d'erreur.
- `error_p` : Paragraphe pour afficher le message d'erreur.

Fonctionnalités JavaScript :
- Le script récupère les éventuels messages d'erreur dans l'URL (paramètre `error`) et affiche le message d'erreur correspondant à l'utilisateur.
- Les messages d'erreur sont affichés dans la `error_div` en fonction du type d'erreur.

Serveur Web

server.js

Crée un serveur web utilisant Express pour gérer les routes et les requêtes HTTP.

Dépendances NPM utilisées :

- express : Framework Node.js pour créer des serveurs web.
- morgan : Middleware pour logger les requêtes HTTP.
- ejs : Moteur de template pour générer des vues HTML dynamiques.
- body-parser : Middleware pour récupérer les données des formulaires HTML.
- mysql : Client MySQL pour interagir avec une base de données MySQL.
- bcrypt : Bibliothèque de chiffrement pour sécuriser les mots de passe.
- redis : Système de gestion de base de données en mémoire.
- express-session : Middleware pour gérer les sessions utilisateurs.
- connect-redis : Stockage des sessions dans Redis.

Fonctionnalités :

1. **Connexion à la base de données MySQL :** Utilise le module `mysql_connection` pour établir une connexion avec la base de données.

2. **Gestion des sessions avec Redis :** Utilise `express-session` pour stocker les sessions dans Redis, un système de gestion de base de données en mémoire.

3. **Routes pour les pages de login et d'inscription :** Définit des routes GET et POST pour les pages de login et d'inscription.

4. **Authentification et enregistrement des utilisateurs :** Les données sont vérifiées et enregistrées dans la base de données MySQL en utilisant le module `authentification`.

5. **Gestion des vues avec EJS :** Utilise EJS comme moteur de template pour rendre les pages HTML dynamiques.

6. **Log des requêtes HTTP avec Morgan :** Les requêtes HTTP sont loggées pour faciliter le débogage et la surveillance du serveur.

7. **Redirections et rechargements des pages :** Après les opérations de login, logout et d'inscription, les utilisateurs sont redirigés vers les pages appropriées.