

Documentation technique concernant l'application

Serveur :

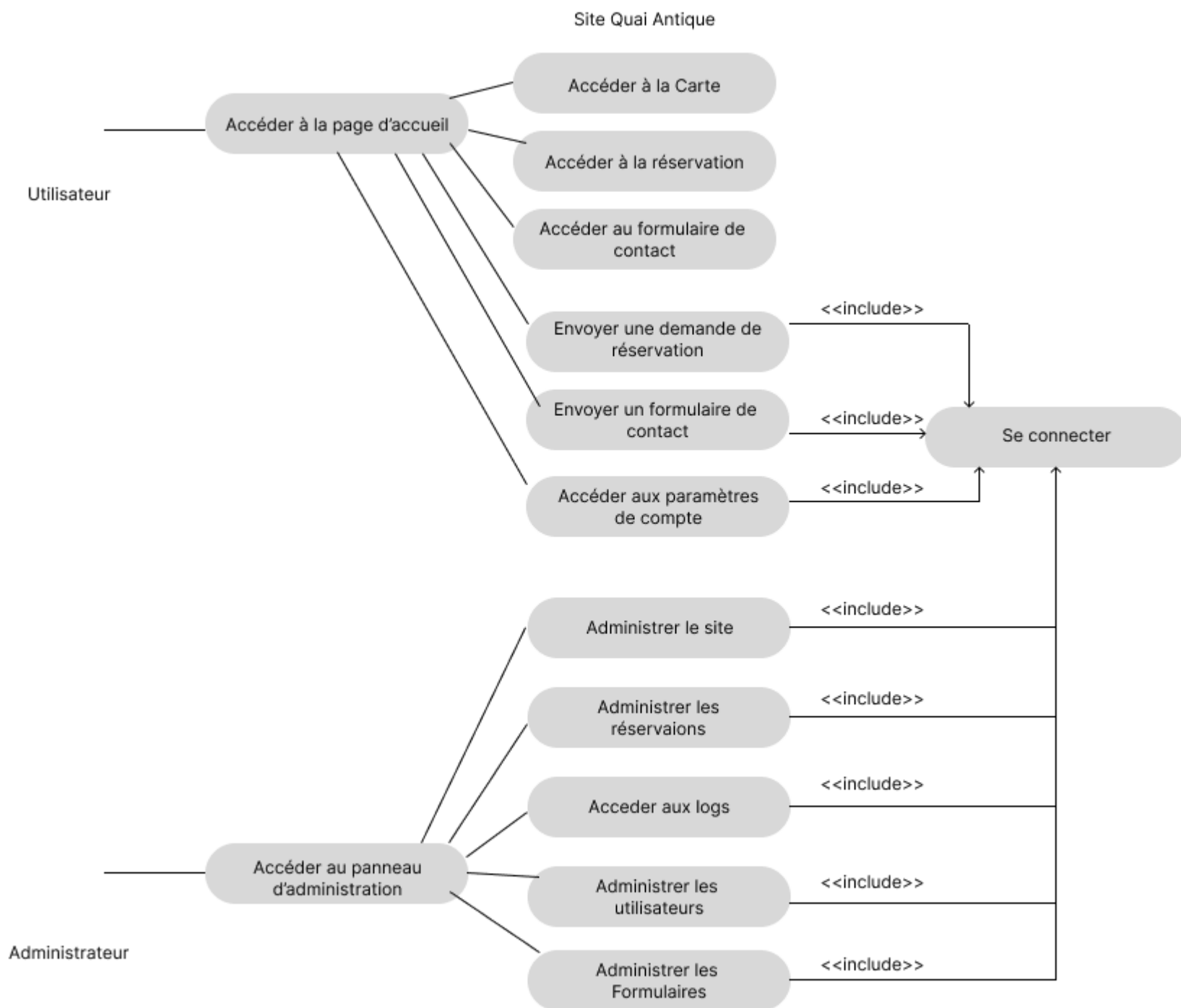
- Fly.io

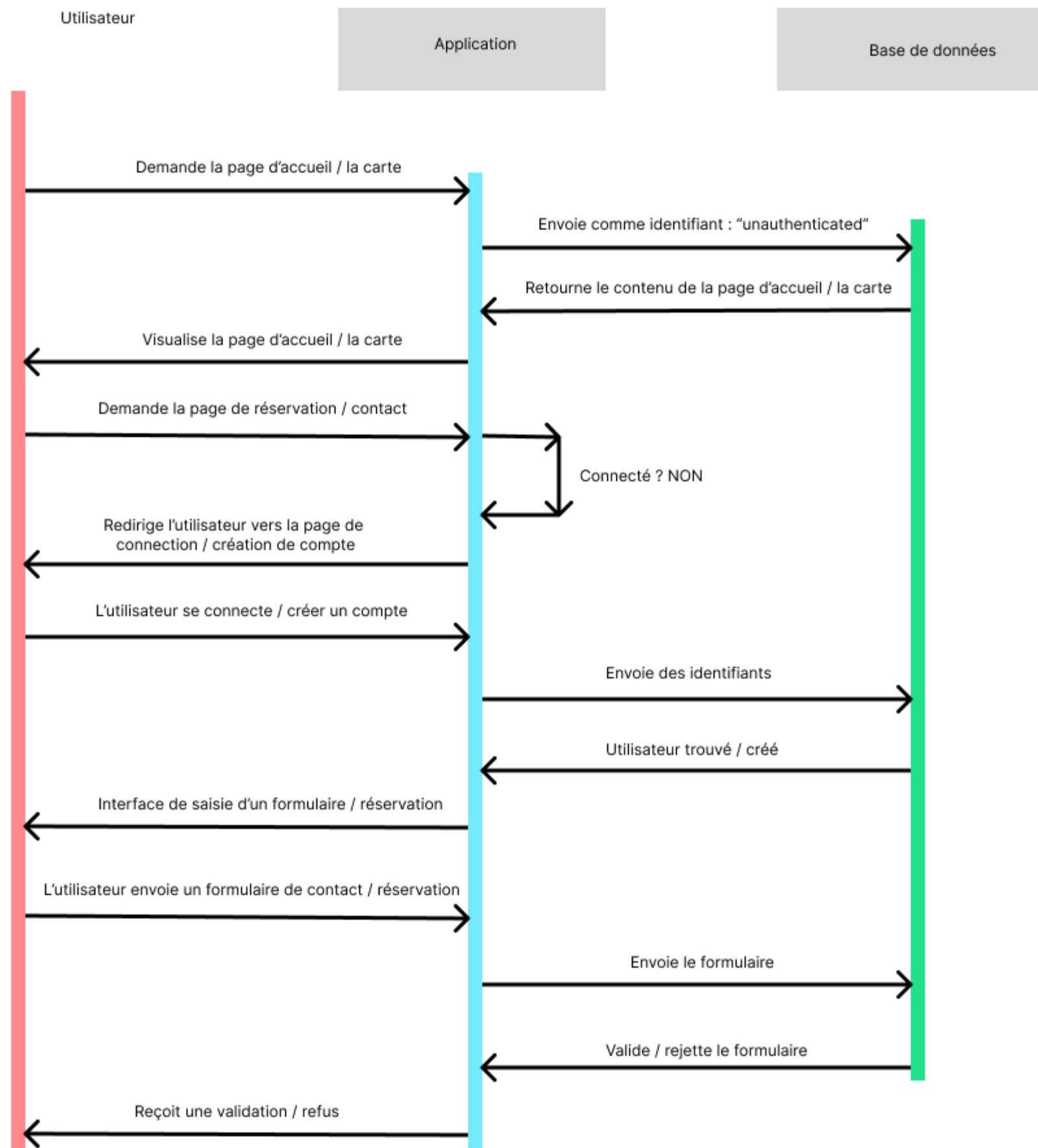
Front :

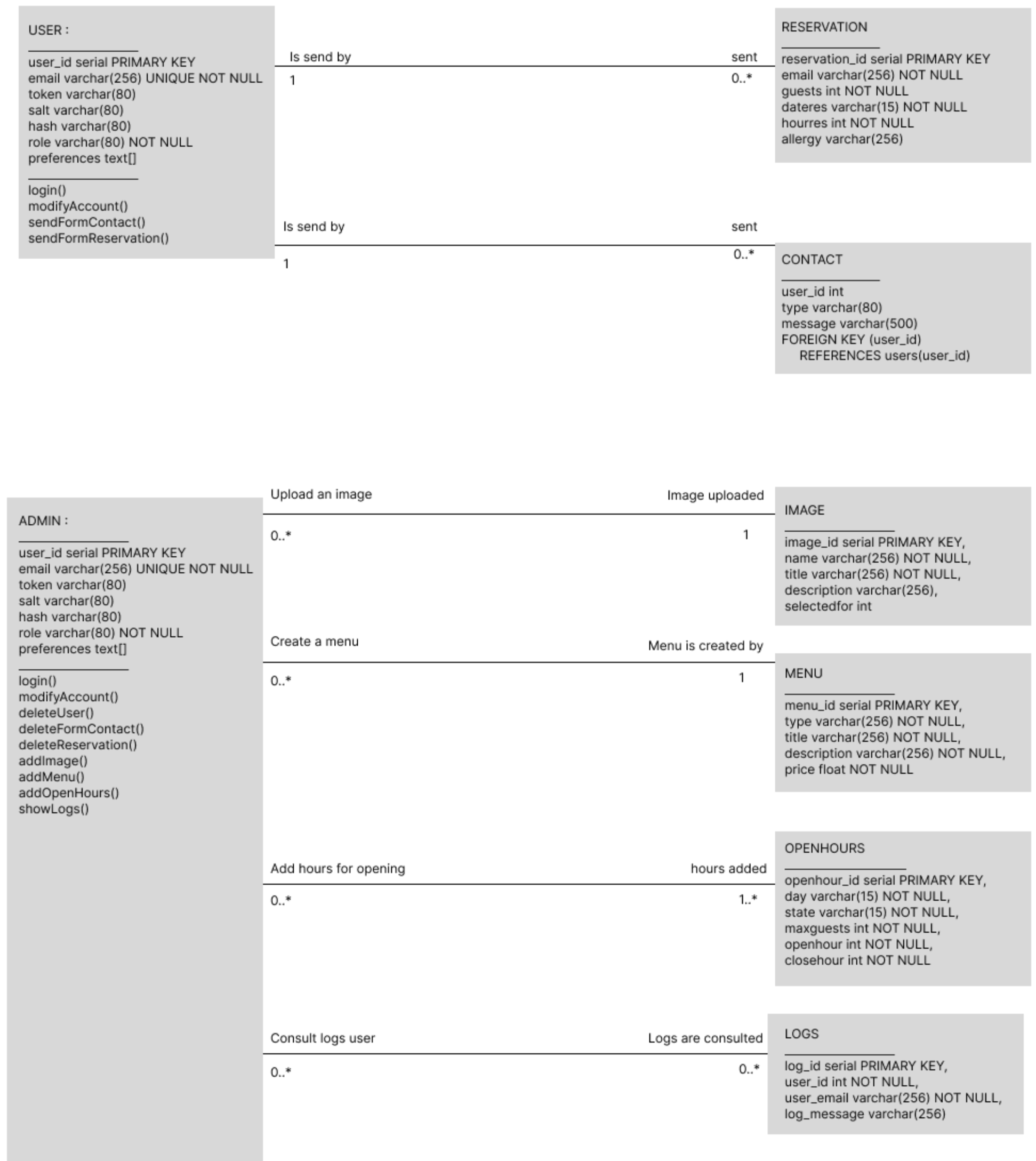
- HTML 5
- CSS 3
- Bootstrap
- JavaScript
- Twig

Back :

- Express.js
- Node
- npm
- PostgreSQL







BackOffice

Paquets installés

- **nodemon**

nodemon permet de redémarrer le serveur à chaque modification du code de manière automatique.

- **express**

Express.js est le framework standard pour le développement de serveur basés sur Node.js

- **cookieparser**

CookieParser me permet de gérer les cookies comme le token d'itentification

- **pg (postgresql)**

Depuis un shell LINUX pour se connecter :

```
sudo -i -u postgres
```

puis :

```
psql
```

pg est le paquet permettant de faire communiquer avec plus de simplicité la BDD (postgresql) avec le code du framework Express.js

- **fs**

fs est le paquet me permettant d'executer des script d'extensions .sql, de ce fait je peux créer des scripts plus complexes que de simple requête et les injecter dans mon code JavaScript pour y gagner en lisibilité.

- **uid2 / crypto-js**

uid2 et crypto-js sont les paquets me permettant d'encrypter les informations sensibles utilisateurs comme le mot de passe.

- **env-cmd**

env-cmd me permet d'intégrer et lire des fichier d'environnement

- **express-fileupload**

express-fileupload comme sont nom l'indique permet de gérer le téléchargement de fichier depuis un formulaire.

Construction du BackOffice

index.js est le fichier qui execute le serveur en mode developpement.

secureSSL.js est le fichier qui execute le serveur en mode production. Sous contrainte d'avoir des certificats SSL valident.

Le BackOffice est constitué d'un dossier *./Controllers*, d'un dossier *./Routes*, d'un dossier *./Models*, d'un dossier *./Utils* et d'un dossier *./Fixtures*.

J'ai décidé volontairement de ne pas utiliser de librairie me permettant de générer des "models" ou des "entités" (EX : sequelize). Dans la consigne il est stipulé que vous souhaitez vérifier certains script SQL, mon application étant simple j'ai choisis de coder mes script SQL qui représentent mes modèles. N'ayant pas de librairies pour générer les modèles, j'ai donc des requêtes SQL dans mes controllers.

- **Controllers**

Dans le dossier *./Controllers*, se trouve un fichier *user.js* qui définit les fonctions liées au modèle *USER*.

Vous y trouverez 8 fonctions asynchrones qui composent le CRUD user et la gestion de l'entité :

- `usersGet()`
- `userCreate()`
- `resetPassword()`
- `userUpdate()`
- `userDelete()`
- `settingsPreferences()`
- `userLogin()`
- `userLogout()`

Elles représentent donc le CRUD du modèles *USER* ainsi que la connection et la déconnection du client au BackOffice (l'authentification sera vu un peu plus loin).

Dans le dossier *./Controllers* se trouve aussi un fichier *app.js* qui définit les fonctions liées à l'utilisation de l'application.

Vous y trouverez 12 fonctions asynchrones qui composent le système de redirection de l'application :

- ***redirectHomepage()***
- ***redirectContact()***
- ***redirectSuscribe()***
- ***redirectLogin()***
- ***redirectCreateAccount()***
- ***redirectInformation()***
- ***redirectSettings()***
- ***redirectMenu()***

- ***redirectReservation()***
- ***selectDayReservation()***
- ***selectHourReservation***
- ***valideReservation***

Dans le dossier */Controllers* se trouve pour finir un fichier */dashboard.js* qui définit les fonctions liées à l'utilisation de l'interface backoffice pour les adminisitrateurs. Comme son nom l'indique il s'agit d'un tableau de bord pour visualiser et gérer les utilisateurs.

vous y trouverez 15 fonctions asynchrones qui composent le système de gestion et de redirection du dashboard admin :

- ***redirectDashboard()***
 - ***redirectShowUser()***
 - ***redirectLogs()***
 - ***redirectFormcontact()***
 - ***redirectManageSite()***
 - ***uploadImage()***
 - ***deleteImage()***
 - ***selectImage()***
 - ***addMenu()***
 - ***deleteMenu()***
 - ***addFormule()***
 - ***deleteFormule()***
 - ***addOpenHours()***
 - ***deleteOpenHours()***
 - ***redirectShowReservations()***
-
- ***deleteReservation()***

• Routes

Dans le dossier */Routes*, se trouve un fichier ***user.js*** qui définit les relations entre les appels API et les fonctions qui y sont liées.

Dans le dossier */Routes*, se trouve aussi un fichier ***app.js*** qui définit les relations entre les appels de redirections et les fonctions qui y sont liées.

Dans le dossier */Routes*, se trouve pour finir un fichier ***dashboard.js*** qui définit les relations entre les appels tu tableau de bord administrateur et les fonctions qui y sont liées.

• Models

Dans le dossier */Models* se trouve les fichiers ***user.sql***, ***logs.sql***, ***contacts.sql***, ***images.sql***, ***menus.sql***, ***openhours.sql***, ***reservations.sql*** et ***formules.sql*** qui permettent de construire les tables SQL si celles ci ne le sont pas déjà au sein de la BDD. On y trouve aussi le fichier ***init_models.js*** qui est appelé dans le script shell d'initialisation de la BDD pour créer les tables nécessaires.

• Fixtures

Dans le dossier */Fixtures* se trouve un fichier ***load.js*** qui est appelé dans ***/bin/init_models.js*** lors de l'initialisation de la BDD.

• Utils

Les deux fichiers ***encryptPassword.js*** et ***decryptPassword.js***, permettent d'encrypter le mot de passe utilisateur et de faire correspondre un mot de passe avec une combinaison de ***SALT*** et de ***HASH***.

encryptPassword.js contient une fonction `encryptPassword()` qui depuis un mot de passe génère le jeu de données ***SALT***, ***HASH*** et ***TOKEN***. A la création d'un nouvelle utilisateur, le jeu de données est sauvegardé en BDD.

decryptPassword.js contient une fonction `decryptPassword()` qui prend pour paramètre un jeu de données ***SALT*** et ***HASH*** ainsi qu'un mot de passe, si le mot de passe est le bon, la fonction retourne le ***TOKEN*** lié à l'utilisateur. Lorsqu'un utilisateur parvient à se connecter le ***TOKEN*** en question est conservé côté client pour une durée de 25 minutes, ou de 1 an selon le choix de l'utilisateur, permettant de continuer à authentifier ce dernier.

getRolesMiddleware.js contient une fonction `getRolesMiddleware()` qui lors d'un appel à l'API vérifie si l'utilisateur est déjà connecté par le biais du ***TOKEN***, si c'est le cas, elle renvoie un role qui correspond à l'utilisateur sauvegardé en BDD. Ce rôle servira dans les controllers à confirmer ou non l'accès à certains appels API.

logger.js contient une fonction `newLog()` qui permet d'ajouter un log en BDD pour historiser les actions des utilisateurs.

generatePassword.js comme son nom l'indique permet de générer aléatoirement un mot de passe, utile si un utilisateur souhaite réinitialiser son mot de passe.

db.js contient une constante qui est décrit comme ceci :

```
const pool = new Pool({
```

```
user: process.env.POSTGRES_USER,
host: process.env.HOST,
database: process.env.DATABASE,
password: process.env.PASSWORD,
})
```

Cette constante permet de connecter les requêtes SQL à la BDD en se basant sur les informations du fichier *.env.NODE_ENV*.

FrontOffice

Paquets installés

- **Twig**

Twig en moteur de template, ayant déjà travaillé avec Symfony auparavant, twig me semblait une solution simple de mise en oeuvre. Il me permet de transmettre des données au front et de le conditionner selon certains paramètres. Typiquement, si l'utilisateur est connecté ou non.

- **body-parser**

body-parser me permet de récupérer de manière simple et efficace le contenu du body de la requête transmise par les formulaires.

- **bootstrap**

Bootstrap pour la mise en page, aussi pour l'adaptation mobile.

- **jQuery**

jQuery est nécessaire pour faire fonctionner Bootstrap avec Express.

Construction du FrontOffice

Le FrontOffice est constitué de deux dossiers, ***/Views*** qui contient les pages html.twig et d'un dossier ***/Public*** qui est déclaré static dans mon index.js, qui lui contient les fichiers nécessaire au rendu coté client.

- **Views**

Le dossier ***/Views*** contient un fichier ***base.html.twig*** qui sert de parent aux autres templates. Les appels de script JS ainsi que les appels de style CSS se font dans ce fichier pour être répercutés sur toute l'application, me permettant de mettre en place un système de thème par exemple. Le dossier ***/Views/Templates*** contient les templates de l'application, les pages qui seront affichées à l'utilisateur final. Il s'y trouve aussi un dossier ***/Views/Templates/AdminDashboard*** qui contient les fichiers nécessaire au rendu du tableau de bord administrateur.

- **Public**

Le dossier ***/Public*** me permet de transporter les modules utiles au bon fonctionnement du front, comme la bibliothèque Bootstrap, jQuery ou encore les fichiers personnalisés CSS et les images.

Utilisation du FrontOffice

Côté front si vous êtes ***administrateur*** vous aurez accès à un dashboard, permettant de gérer les utilisateurs, de visualiser les logs, d'accéder aux formulaires de contacts envoyer par les utilisateurs, visualiser les réservations clients et modifier certains aspects du site.

Il est aussi possible de lister les utilisateurs en entrant un patern dans la barre de recherche.

Vous pourrez accéder aux logs d'une manière global mais aussi en consultant une fiche utilisateur, uniquement les logs concernant cet utilisateur seront listés.

Si vous êtes ***utilisateur*** vous pourrez visualiser la page d'accueil, et le menu. Si vous avez créé un compte, vous pourrez contacter l'équipe via un formulaire et faire une réservation.

Amélioration possible du code

- **Gestion de la validation des formulaires**

Les formulaires fonctionnent mais la gestion de la données n'est pas entièrement implémentée. Cela viendra. Typiquement, il est possible de rentrer une chaîne de caractère dans le champs d'un numéro de téléphone, le serveur va renvoyer une erreur. Pour le formulaire de réservation, beaucoup est à faire même s'il fonctionne pour l'instant.

- **Redirection après validation d'un formulaire**

Les redirections ne sont pas toujours logique, il faudrait que j'ajoute des messages de confirmation lorsqu'on effectue une action comme la création d'un compte et la connexion. Actuellement un retour JSON est envoyé lorsqu'une erreur est relevée.

- **Ajout de popup**

Il faut que j'ajoute une couche de confirmation sous forme de popup côté utilisateur, quand on veut supprimer son compte, quand on veut confirmer une réservation..

- **Gérer la cohérence du code**

En 72h de code je me permet de faire une précision, oui il y a clairement des améliorations dans la cohérence du code ! les conditions, et la manière dont j'appelle les variables. La redirection des routes qui ne suit pas toujours la même logique. Mais surtout la partie template du front et les styles CSS.

Outils de développement

- **VScod**

Une capture d'écran de mon espace de travail :

J'ai deux écrans, l'un me sert à afficher Postman et la documentation, le second me sert à afficher VScode.

J'ai deux fenêtrages de code, souvent celui de droite est le code sur lequel je me base, typiquement mes modèles et celui de gauche le code que je travaille.

J'ai deux invites de commandes, l'un me sert à afficher le prompt et le retour d'erreur du serveur Express et le second à installer mes paquets et naviguer sur ma machine.

Les extensions installées sur VScode sont nombreuses, je ne vais citer que les plus pertinentes pour ce projet.

- **Github** Pour pouvoir effectuer mes commits, changement de branche et fusion de manière assistée et éviter les erreurs
- **PowerShell** et **SSH FS** qui me permettent d'obtenir un prompt connecté en SSH à une machine distante
- **Microsoft Edge Tool** pour obtenir une fenêtre de navigateur dans VScode, peut être très utile lorsque je souhaite de la documentation et que mon second écran est utilisé.
- **ChatGPT**, qui me sert EXCLUSIVEMENT et j'insiste, à vérifier des connaissances ou me donner des noms de librairies lorsque je ne connais pas bien le framework.
- **HTMLHint**, pour relever les erreurs HTML.
- **Live Sass Compiler**, pour compiler mes fichiers Scss.
- **Twig Syntax highlighting**, pour relever les erreurs Twig.
- **Tabnine**, pour me faciliter la tâche dans mon code en général en appliquant une couche d'autocomplétion intelligente.

user.js - studiECF2023_Fredj_Corentin

File Edit Selection View Go Run Terminal Help

EXPLORER

- STUDIECF2023_FREDJ_CORENTIN
 - Controllers
 - JS user.js
 - ECFimages
 - jiraFeuilleDeRoute.png
 - Models
 - user.sql
 - node_modules
 - Routes
 - JS user.js
 - Utils
 - JS decryptPassword.js
 - JS encryptPassword.js
 - JS getRolesMiddleware.js
 - lockCopie à rendre ECF Graduate Developp... U
 - Copie à rendre ECF Graduate Developpeur.doc
 - JS index.js
 - package-lock.json
 - package.json
 - README.md

Controllers > JS user.js > [?] <unknown> > 🔗 userLogout

```
79         res.send(row+' Succefully updated !')
80     });
81 }
82 }
83 }
84 }
85
86 // Delete a user by pseudo selection
87 async function userDelete(req, res) {
88     if (req.role != "ROLE_ADMIN") {
89         return res.json("Only ADMIN can do that !")
90     }
91     else {
92         const target_pseudo = req.body.target_pseudo
93         if (!req.body.target_pseudo) {
94             return res.send("Missing pseudo to confirm wich one you want to delete")
95         }
96         else {
97             await pool.query(`SELECT user_id FROM users WHERE pseudo = '${target_pseudo}'`)
98             if (error) {
99                 throw error
100             }
101             const selectedUser = results.rows[0]
102             if (!selectedUser) {
103                 return res.json('No user found with this pseudo !')
104             }
105             else {
106                 pool.query(`DELETE FROM users WHERE pseudo = '${target_pseudo}'`)
107                 if (error) {
108                     throw error
109                 }
110             }
111         }
112     }
113 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS


```
syscall: 'open',
code: 'ENOENT',
path: './Utils/initDB.sql'
}

Node.js v18.13.0
[nodemon] app crashed - waiting for file changes before starting...
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Server app listening on port 3000
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Server app listening on port 3000
```


> OUTLINE


> TIMELINE


man*

 **studiECF2023_Fred...**
Projet logiciel


PLANIFICATION


 Feuille de route


 Backlog


 **Tableau**

D  VELOPPEMENT

 Code

 Pages de projet

 Ajouter un raccourci

 Param  tres du pro...

Vous faites partie d'un projet g  r   par l'  quipe

En savoir plus

D  bloquez davantage de fonctionnalit  s gr  ce    un [essai gratuit de Jira Software](#)

Projets / studiECF2023_Fredj_Corentin

Gestion Utilisateur FRONT



















Compl  ter la partie Utilisateur en FrontOffice







Epic 

A FAIRE 12 TICKETS	EN COURS 2 TICKETS	FINI 3 TICKETS 
<div>L'utilisateur doit choisir son pseudo</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-6 </div>	<div>L'utilisateur peut demander �� rester connect��</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-5 </div>	<div>L'utilisateur doit entrer u pour valider la connectio</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-12</div>
<div>L'utilisateur doit choisir un MDP</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-7 </div>	<div>L'utilisateur peut demander un nouveau MDP si oubli��</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-4 </div>	<div>L'utilisateur doit entrer u Pseudo pour se connecte</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-10</div>
<div>L'utilisateur doit choisir un @</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-9 </div>		<div>L'utilisateur peut se d��connecter</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-13</div>
<div>L'utilisateur peut supprimer son compte</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-14 </div>		
<div>L'utilisateur peut modifier son MDP</div> <div>L'UTILISATEUR FRONT</div> <div> ECF-16 </div>		

	fév.	MARS
Sprints		<div>Ges...</div>
<div><div>▼</div><div><div><div>✚</div><div>ECF-1</div><div>L'utilisateur se connecte</div></div></div></div>		<div></div>
<div><div><div>📌</div><div>ECF-4</div><div>L'utilisateur peut demander un nouveau MDP si oublié</div></div><div>À FAIRE</div><div>CORENTIN</div></div>		<div></div>
<div><div><div>📌</div><div>ECF-5</div><div>L'utilisateur peut demander à rester connecté</div></div><div>À FAIRE</div><div>CORENTIN</div></div>		<div></div>
<div><div><div>📌</div><div>ECF-12</div><div>L'utilisateur doit entrer un MDP pour valider la conne...</div></div><div>À FAIRE</div><div>CORENTIN</div></div>		<div></div>
<div><div><div>📌</div><div>ECF-10</div><div>L'utilisateur doit entrer un @ ou Pseudo pour se con...</div></div><div>À FAIRE</div><div>CORENTIN</div></div>		<div></div>
<div><div>▶</div><div><div><div>✚</div><div>ECF-2</div><div>L'utilisateur créé un compte</div></div></div></div>		<div></div>
<div><div>▶</div><div><div><div>✚</div><div>ECF-3</div><div>L'utilisateur se déconnecte</div></div></div></div>		<div></div>
<div><div>▶</div><div><div><div>✚</div><div>ECF-15</div><div>L'utilisateur peut mettre à jour son profil</div></div></div></div>		<div></div>
<div><div><div>✚</div><div>ECF-19</div><div>L'admin se connecte</div></div></div>		
<div><div><div>✚</div><div>ECF-21</div><div>L'admin se déconnecte</div></div></div>		
<div><div>▶</div><div><div><div>✚</div><div>ECF-22</div><div>L'admin peut gérer les utilisateurs</div></div></div></div>		<div></div>

Liens utiles

- [Github](#)
- [Jira](#)