

# ***SAE 302 : Application communicante***

## ***Rapport***

**ÉLÈVES:**  
**MONTEGU Jeremie**  
**LEBON Johan**  
**LEPERLIER Aymeric**

<b>Notre projet :</b>	<b>3</b>
<b>Notre équipe :</b>	<b>4</b>
<b>Mise en garde pour l'utilisation du projet.</b>	<b>4</b>
<b>Support pris en charge :</b>	<b>5</b>
<b>Prérequis.</b>	<b>5</b>
<b>Développement du Back-End de HiddenKey.</b>	<b>6</b>
1. Base de données.	6
2. Inscription.	7
3. Connexion.	8
4. Ajout de mots de passe.	9
5. Liste de mots de passe.	9
6. Supprimer un mots de passe.	10
7. Modifier un mots de passe.	11
<b>Développement du Front-End de HiddenKey.</b>	<b>12</b>
1. Accueil.	12
2. Inscription.	13
3. Connexion.	14
4. Ajout de mots de passe.	15
5. Liste de mots de passe.	16
5.1 Visualisation du mots de passe.	17
5.2 Modification du mots de passe.	17
5.3 Suppression du mots de passe.	18
<b>Déploiement sur le serveur VPS.</b>	<b>19</b>
1. Préparation du Serveur VPS.	19
2. Transfert des Fichiers avec WinSCP.	20
3. Installation des Dépendances du Projet.	20
4. Configuration du Serveur Web avec Apache.	21
4.1. Configuration de base d'Apache.	21
4.2. Détails de la Configuration Apache.	21
5. Gestion du Backend avec PM2.	22
6. Problèmes Rencontrés.	22
6.1. Problème de Ports Occupés.	22
6.2. Erreur 404 pour les Routes API.	22
6.3. Fonctionnement en Local vs Réseau Universitaire.	22
7. Commandes Principales.	23
8. Résultat Final.	24
9. Création de trois utilisateurs pour test.	24
10. Instructions pour tester l'application.	25

## **Notre projet :**

Une web app de gestionnaire de mots de passes cryptés, fonctionnant sur tout appareil disposant d'un navigateur internet

Pour réaliser ce projet, nous avons utilisé un serveur web React, JavaScript, HTML, CSS et Bootstrap.

Nous nous sommes inspirés de l'application Keepass et d'autres modèles d'applications de gestionnaire de mots de passes pour faire notre propre application qu'on a nommé "[HiddenKey](#)"

Les problèmes divers que nous avons rencontrés sont le développement et la compréhension de JS due à notre connaissance basique de ce langage et également le déploiement de l'application sur le serveur VPS.

## **Notre équipe :**

Nom & Prénom	Rôle
MONTEGU Jeremie	Responsable Client/Serveur
LEPERLIER Aymeric	Responsable Hébergement/Serveur
LEBON Johan	Responsable Rapport/HTML/CSS

## **Mise en garde pour l'utilisation du projet**

Nous souhaitons attirer votre attention sur un point important concernant l'accès au projet.

Le projet que nous vous transmettons est **une version adaptée pour un usage local** (localhost). En effet, la version complète utilisée pour le déploiement sur le serveur VPS est trop volumineuse pour être téléchargée directement depuis le serveur, en raison de la taille des fichiers et des configurations spécifiques nécessaires à l'hébergement.

Cependant, nous avons veillé à ce que **toutes les étapes de déploiement, de configuration et de paramétrage** soient correctement expliquées dans ce compte rendu.

Pour les tests, nous vous recommandons de **lire attentivement tout le compte rendu** avant de procéder. Une fois le compte rendu parcouru, nous vous invitons à tester directement les fonctionnalités en ligne, à l'adresse suivante : <http://88.222.215.101>.

Le projet fourni est principalement destiné à vous permettre de consulter les codes sources et de comprendre la logique et la structure du projet. Les tests fonctionnels et pratiques devraient se faire directement sur le site déployé, car il est configuré pour refléter toutes les parties techniques nécessaires au bon fonctionnement en environnement en ligne.

---

## **Support pris en charge :**

Notre application est hébergée et donc elle est accessible via un navigateur internet à l'adresse suivante : <http://88.222.215.101> donc accessible depuis : Téléphone, Ordinateur, Smart TV, etc...

## **Prérequis**

Pour mener à bien notre projet et qu'on puisse développer l'application correctement on a dû installer quelques pré requis tel que

- Express.js pour le développement côté serveur de l'application
- Axios pour les requêtes HTTP
- React pour le développement côté client de l'application
- Bootstrap pour le style
- Intro.js pour le côté Tutoriel de l'application
- Crypto.js et bcrypt.js pour le côté cryptage des mots de passe
- jsonwebtoken pour pouvoir différencier les utilisateurs avec des token

## **Développement du Back-End de HiddenKey**

Pour le développement du backend nous avons utilisé des requêtes HTTP avec les différents routes de notre applications

### **1.Base de données**

Pour notre base de données nous avons décidé d'utiliser un fichier JSON qui est stocker en local sur le serveur

Donc pour pouvoir lire et écrire dans ce fichier JSON nous avons dû créer un les fonctions suivantes

```
// Fonction pour lire les données du fichier JSON
function readData() {
  if (!fs.existsSync(DATA_FILE)) {
    fs.writeFileSync(DATA_FILE, JSON.stringify({ users: [],
passwords: [] }, null, 2));
  }
  const rawData = fs.readFileSync(DATA_FILE);
  return JSON.parse(rawData);
}

// Fonction pour écrire les données dans le fichier JSON
function writeData(data) {
  fs.writeFileSync(DATA_FILE, JSON.stringify(data, null, 2));
}
```

Notre basé de donnés est organisée de cette façon :  
Tout les utilisateur sont rangés dans la partie “**users**”

```
{
  "users": [
    {
      "username": "jo",
      "password": "$2a$10$kF4FE3pHvkm6hI8Syfe2Xu4c8YvxAw.wCTDXv3heWJFTv2zzaGaam"
    },
  ],
}
```

Et tous les mots de passe sont rangés dans la partie “**passwords**” en fonction de leur ID et de leur user

```
"passwords": [  
  {  
    "id": "8ca8dab2-786d-4892-a7a8-3198358894a9",  
    "service": "go",  
    "password": "U2FsdGVkX1++7nYxAXG00IAh35twv7ck8nvTnl7viG0=",  
    "username": "lo",  
    "user": "jo"  
  },  
]
```

Tous les mots de passe de la base de données sont cryptées

## 2. Inscription

Ensuite pour l'inscription d'un utilisateur nous avons utiliser un requêtes POST avec le chemin de la page register

```
// Inscription d'un nouvel utilisateur
app.post('/api/register', async (req, res) => {
  const { username, password } = req.body;
  if (!username || !password) {
    return res.status(400).send('Nom d'utilisateur et mot de passe requis');
  }

  const data = readData();
  // Vérifie si l'utilisateur existe déjà
  const existingUser = data.users.find(user => user.username === username);
  if (existingUser) {
    return res.status(409).send('Nom d'utilisateur déjà pris');
  }

  // Hache le mot de passe et l'ajoute à la liste des utilisateurs
  const hashedPassword = await bcrypt.hash(password, 10);
  data.users.push({ username, password: hashedPassword });
  writeData(data);
  res.send('Inscription réussie !');
  console.log(username + " a été inscrit");
});
```



### 3. Connexion

Ensuite pour la connexion de l'utilisateur nous avons utiliser un requêtes POST avec le chemin de la page login

```
// Connexion d'un utilisateur
app.post('/api/login', async (req, res) => {
  const { username, password } = req.body;
  const data = readData();
  const user = data.users.find(user => user.username === username);

  if (!user || !(await bcrypt.compare(password, user.password))) {
    return res.status(401).send('Nom d'utilisateur ou mot de passe incorrect');
  }

  // Génère un token JWT
  const token = jwt.sign({ username: user.username }, JWT_SECRET);
  res.json({ token });
  console.log(username + " s'est connectée");
  console.log('Utilisateur authentifié :', username);
});
```

## 4. Ajout de mots de passe

Ensuite pour l'ajout de mots de passe dans notre password manager nous avons utilisé un requête post avec le chemin de la page add-password

```
// Route pour ajouter un mot de passe
app.post('/api/add-password', authenticateToken, (req, res) => {
  const { service, username, password } = req.body;
  const data = readData();
  const { v4: uuidv4 } = require('uuid');
  const id = uuidv4(); // Génère un ID unique

  // Ajoute le mot de passe avec un ID unique (index-based)
  //const id = data.passwords.length; // index-based id
  const newPassword = { id, service, password, username: username,
user: req.user.username };
  data.passwords.push(newPassword);
  writeData(data);
  res.json(newPassword); // Répond avec l'objet mot de passe créé
});
```

## 5. Liste de mots de passe

Ensuite pour voir notre liste de mots de passe dans notre password manager nous avons utiliser un requêtes get avec le chemin de la page passwords

```
// Route pour voir tous les mots de passe
app.get('/api/passwords', authenticateToken, (req, res) => {
  const data = readData();
  // Filtre les mots de passe pour ne retourner que ceux de
  l'utilisateur connecté
  console.log(req.user.username)
  const userPasswords = data.passwords.filter(entry => entry.user ===
req.user.username);
  res.json(userPasswords); // Retourne les mots de passe de
l'utilisateur
});
```

## 6. Supprimer un mots de passe

Ensuite pour supprimer un mots de passe de liste de mots de passe nous avons utiliser un requêtes delete avec le chemin de la page passwords/:id

```
// Route DELETE pour supprimer un mot de passe par ID
app.delete('/api/passwords/:id', (req, res) => {
  const id = req.params.id; // ID reçu dans l'URL
  const data = readData();

  // Trouve le mot de passe à supprimer
  const passwordEntry = data.passwords.find((password) => password.id
=== id);

  // Vérifie si le mot de passe existe et appartient à l'utilisateur
  if (!passwordEntry) {
    return res.status(404).json({ message: 'Mot de passe non trouvé'
});
  }

  // Filtre les mots de passe pour exclure celui à supprimer
  data.passwords = data.passwords.filter((password) => password.id !==
id);
  writeData(data);

  res.status(200).json({ message: 'Mot de passe supprimé avec succès'
});
});
```

## 7. Modifier un mots de passe

Ensuite pour modifier un mots de passe de notre liste de mots de passe nous avons utiliser un requêtes put avec le chemin de la page passwords/:id

```
// Route PUT pour mettre à jour un mot de passe
app.put('/api/passwords/:id', (req, res) => {
  const { id } = req.params;
  const updatedData = req.body;

  fs.readFile(DATA_FILE, 'utf8', (err, data) => {
    if (err) {
      return res.status(500).json({ message: 'Erreur lors de la
lecture des données.' });
    }

    // Parse du fichier JSON
    const fileContent = JSON.parse(data);
    const passwords = fileContent.passwords; // Accéder au tableau
`passwords`

    // Vérifier que `passwords` est un tableau
    if (!Array.isArray(passwords)) {
      return res.status(500).json({ message: 'Les données ne sont
pas dans un format valide.' });
    }

    // Trouver l'index de l'élément à modifier
    const index = passwords.findIndex((p) => p.id === id);
    if (index === -1) {
      return res.status(404).json({ message: 'Mot de passe non
trouvé.' });
    }

    // Mettre à jour l'élément
    passwords[index] = { ...passwords[index], ...updatedData };

    // Sauvegarder dans le fichier JSON
    fileContent.passwords = passwords;
    fs.writeFile(DATA_FILE, JSON.stringify(fileContent, null, 2),
(err) => {
```

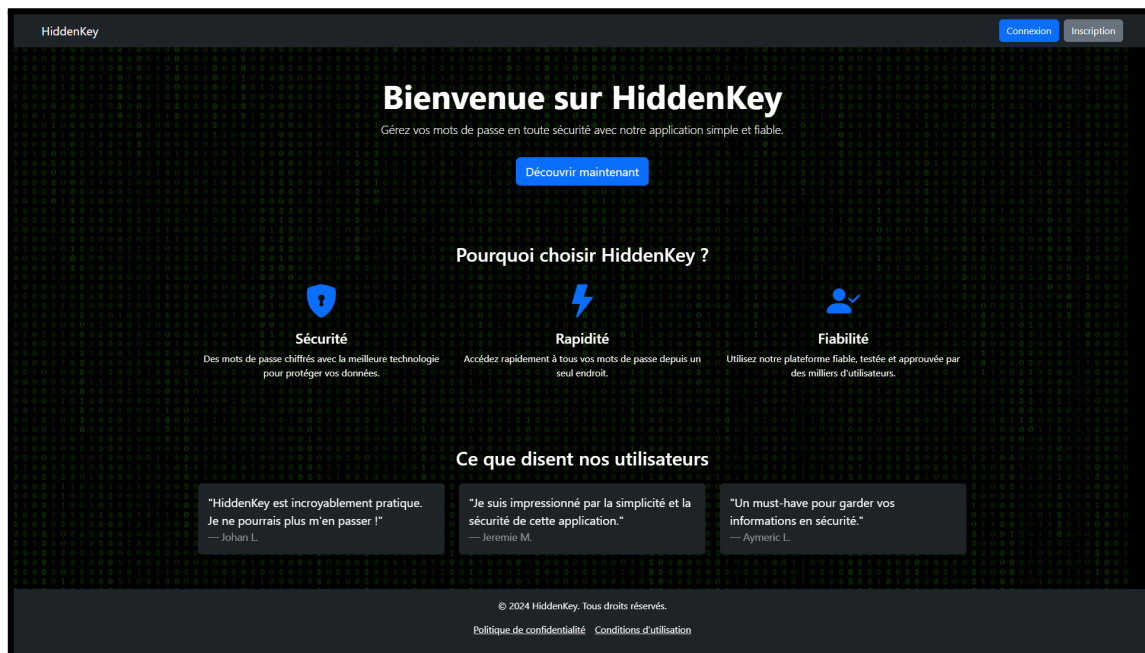
```
        if (err) {  
            return res.status(500).json({ message: 'Erreur lors de la  
sauvegarde des données.' });  
        }  
  
        res.json({ message: 'Mot de passe mis à jour avec succès.',  
updatedPassword: passwords[index] });  
    });  
});
```

## **Développement du Front-End de HiddenKey**

Pour le développement du frontend nous avons utilisé React pour nous permettre de créer l'interface utilisateur

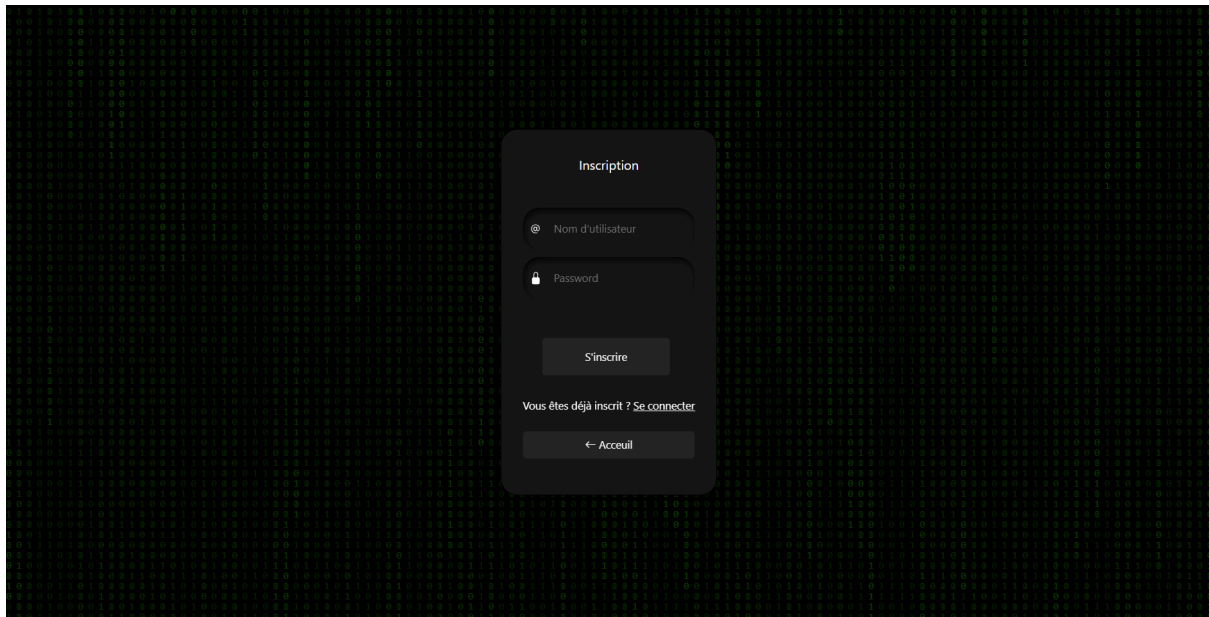
### **1. Accueil**

Pour la page d'accueil nous avons décidé de faire une petite présentation de HiddenKey et des exemples que peut donner des utilisateurs pour rendre l'application la plus réaliste possible



## 2. Inscription

Lorsque nous cliquons sur le bouton “Découvrir maintenant” ça nous renvoie vers notre page d’inscription qui permet à l'utilisateur de s’inscrire dans notre base de donnée



The screenshot shows a registration form titled "Inscription" centered on a dark background with a green binary code (Matrix-style) pattern. The form contains the following elements:

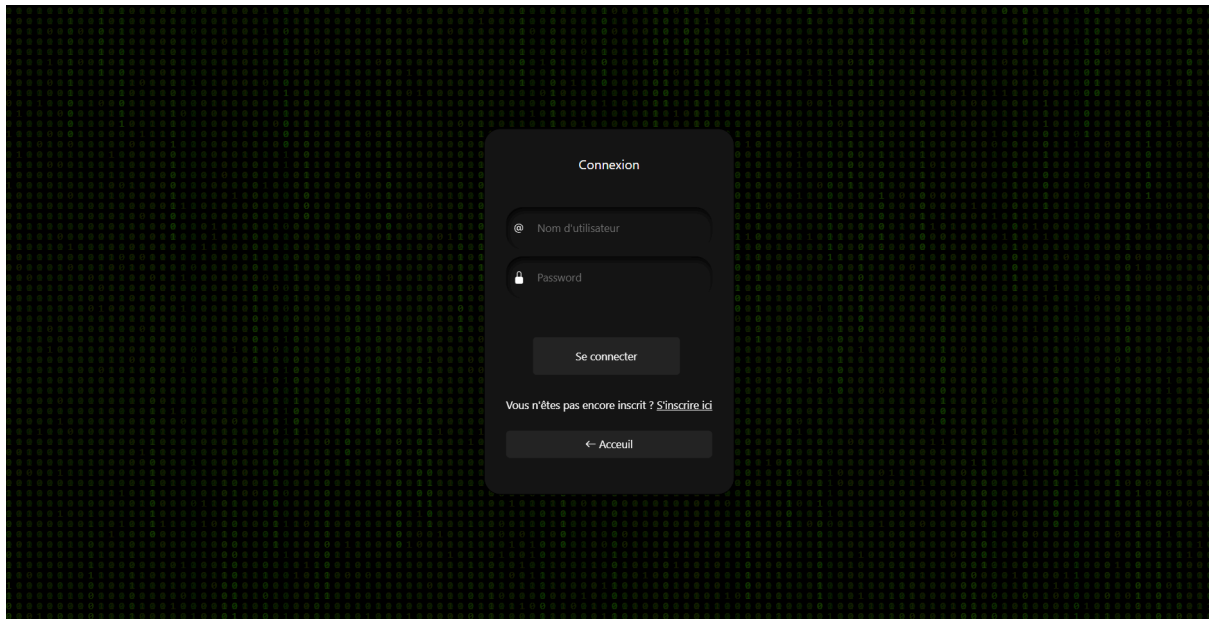
- A text input field with an email icon and the placeholder text "Nom d'utilisateur".
- A password input field with a lock icon and the placeholder text "Password".
- A button labeled "S'inscrire".
- A link below the button that reads "Vous êtes déjà inscrit ? [Se connecter](#)".
- A button at the bottom labeled "← Accueil".

Nous avons rajouté un option en dessous du bouton “S’inscrire” qui permet à l'utilisateur déjà inscrit que de connecter



### 3. Connexion

Lorsque l'utilisateur s'est inscrit nous le renvoyons vers la page de connexion pour qu'il puisse se connecter a son gestionnaire de mots de passe



Nous avons rajouté un option en dessous du bouton “Se connecter” qui permet à l'utilisateur pas encore inscrit de s'inscrire

## 4. Ajout de mots de passe

Lorsque l'utilisateur est connecté il est renvoyé sur son gestionnaire de mots de passe

**Ajouter un Mot de Passe** ⓘ

Service

Nom d'utilisateur

Mot de Passe

+ Générer un mot de passe

+ Ajouter

Afficher la Liste des Mots de Passe

Sur celui-ci il a la possibilité d'ajouter un mots de passe en fonction du "Service", son "Nom d'utilisateur", et son "Mot de passe"

**Ajouter un Mot de Passe** ⓘ

Service

Nom d'utilisateur

Mot de Passe

L'utilisateur à même la possibilité de générer un mots de passe si il le souhaite

Mot de Passe

+ Générer un mot de passe

Veuillez renseigner ce champ.

Nous avons également une barre qui permet d'évaluer la complexité du mots de passe

The image shows three sequential screenshots of a password strength evaluator interface. Each screenshot features a dark background with a grid of small, colorful dots. At the top of each panel is a label 'Mot de Passe' and a button 'Générer un mot de passe'. Below the input field is a horizontal bar representing the password strength, with the text 'Force du mots de passe' above it.

- First screenshot:** The input field contains the number '1'. The strength bar is red and labeled 'Facilement trouvable' (Easily found).
- Second screenshot:** The input field contains the alphanumeric string 'ufthiueiuh56564464'. The strength bar is yellow and labeled 'Moyennement trouvable' (Moderately found).
- Third screenshot:** The input field contains the complex string 'wH:8\$(NL&-1;lqW'. The strength bar is green and labeled 'Introuvable' (Unfindable).

Tous les mots de passe qui sont ajouter son crypter dans la base de données

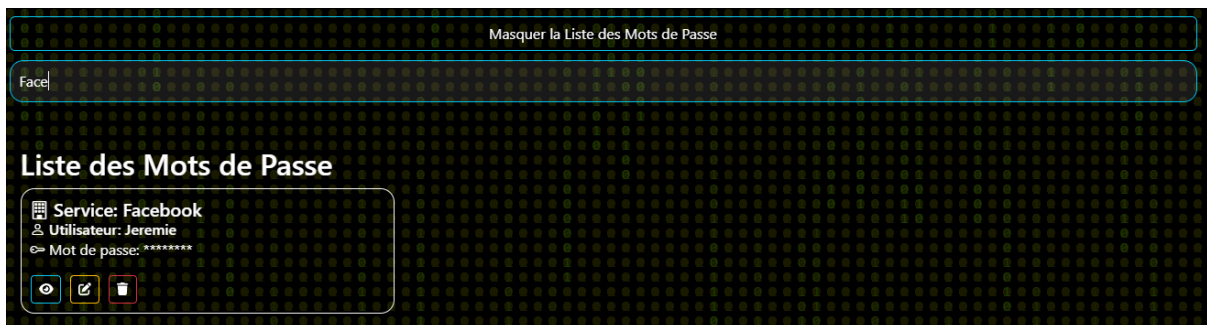
```
const encryptedPassword = CryptoJS.AES.encrypt(password,  
secretKey).toString();
```

## 5. Liste de mots de passe

Lorsque nous cliquons sur le bouton “Afficher la liste des mots de passe” nous pouvons voir tous les mots de passe que nous avons ajouté à notre application



Nous avons également ajouté une barre de recherche qui permet de retrouver les mots de passe que nous voulons sans devoir les rechercher dans toute la liste

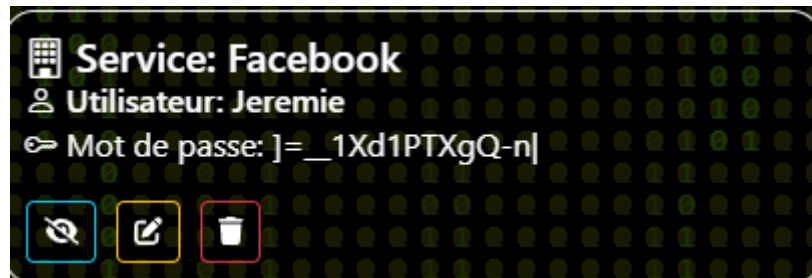


Nous avons également ajouté 3 boutons pour un pour visualiser le mot de passe, un autre qui permet de modifier le mot de passe et un dernier qui permet de le supprimer

### 5.1 Visualisation du mot de passe

Lorsque nous cliquons sur le bouton 

L'utilisateur peut voir son mot de passe décrypter




## 5.2 Modification du mots de passe

Lorsque nous cliquons sur le bouton 

Nous avons un modal bootstrap qui s'ouvre qui permet à l'utilisateur de modifier ses informations

## 5.3 Suppression du mots de passe

Lorsque nous cliquons sur le bouton 

Nous avons un modal bootstrap qui s'ouvre qui permet à l'utilisateur de confirmer la suppression du mots de passe

### Ajouter un Mot de Passe ⓘ

Service

e

Nom d'utilisateur

Nom d'utilisateur

Mot de Passe

Générer un mot de passe

Intéressant

Face

Confirmation de suppression

tes-vous sûr de vouloir supprimer ce mot de passe ?

+ Ajouter

Annuler Supprimer

### Liste des Mots de Passe

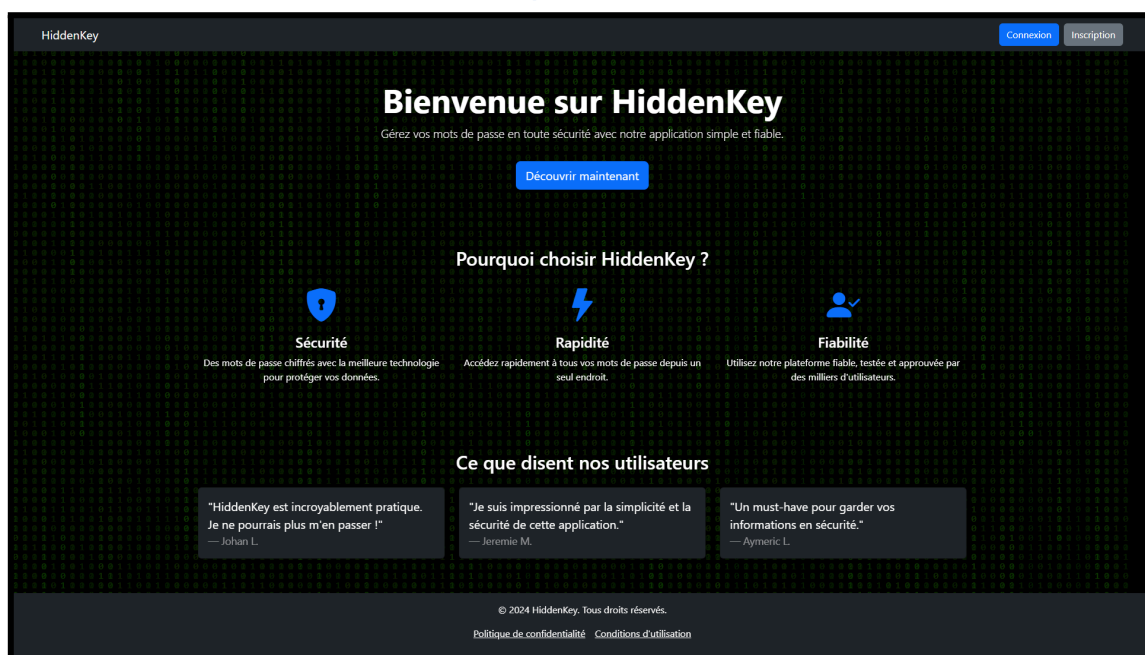
Service: Facebook

Utilisateur: Jeremie

Mot de passe: ]=\_1Xd1PTXgQ-n]

## Déploiement sur le serveur VPS

Pour rendre notre application accessible depuis n'importe où, nous avons opté pour un serveur VPS. Ce choix nous permet de déployer notre application de manière stable, tout en ayant un contrôle total sur l'environnement d'hébergement. L'application est accessible publiquement à l'adresse suivante: <http://88.222.215.101>



## 1. Préparation du Serveur VPS

Dès l'achat du VPS, nous avons préparé le serveur pour accueillir notre projet. Cela a impliqué :

- Connexion SSH : Nous nous sommes connectés en SSH au serveur pour effectuer toutes les manipulations nécessaires.

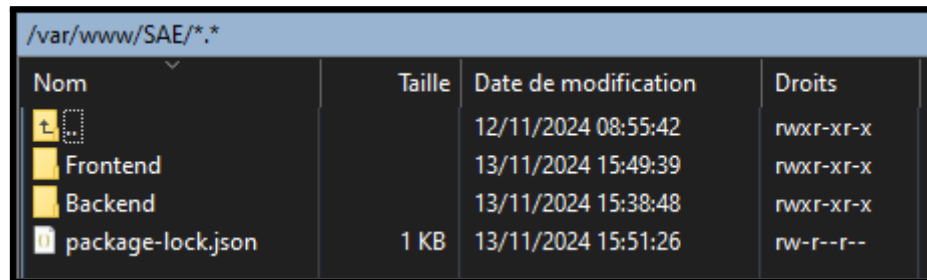
```
login as: root
root@88.222.215.101's password:
Linux srv622013 6.1.0-26-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.112-1 (2024-09-30) x86_64
```





- Installation des outils requis : Nous avons installé Node.js pour exécuter le backend ainsi qu'Apache pour gérer la redirection et la mise en ligne du frontend.



## 2. Transfert des Fichiers avec WinSCP

Pour transférer les fichiers de notre application vers le serveur, nous avons utilisé WinSCP, un outil qui facilite la copie de fichiers via le protocole SCP :



Nom	Taille	Date de modification	Droits
		12/11/2024 08:55:42	rwXr-Xr-X
 Frontend		13/11/2024 15:49:39	rwXr-Xr-X
 Backend		13/11/2024 15:38:48	rwXr-Xr-X
 package-lock.json	1 KB	13/11/2024 15:51:26	rw-r--r--

- Les dossiers backend et frontend ont été transférés depuis notre machine locale vers le répertoire du VPS.
- Cette étape a permis de disposer des fichiers directement sur le serveur, prêts à être configurés et exécutés.

### 3. Installation des Dépendances du Projet

Après le transfert des fichiers, nous avons installé les dépendances nécessaires à l'aide de la commande "npm install" dans les répertoires backend et frontend du serveur VPS. Cette étape garantit que toutes les bibliothèques et modules utilisés dans notre application sont bien présents et opérationnels.

## 4. Configuration du Serveur Web avec Apache

### 4.1. Configuration de base d'Apache

a. Nous avons configuré Apache pour qu'il :

- Serve le frontend (l'application React construite) directement.
- Redirige les requêtes de l'API vers le backend via un proxy.

b. Extrait de la configuration d'Apache :

- ProxyPass et ProxyPassReverse : Configurés pour rediriger toutes les requêtes commençant par /api vers le backend sur le port 3001.

### 4.2. Détails de la Configuration Apache

Voici le contenu du fichier de configuration Apache (/etc/apache2/sites-available/000-default.conf) utilisé pour rediriger les requêtes API vers le backend :

```
VirtualHost *:80>
    ServerAdmin webmaster@localhost

    # Configuration pour le Frontend
    DocumentRoot /var/www/SAE/Frontend/build

    <Directory "/var/www/SAE/Frontend/build">
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    # Proxy inverse pour rediriger les requêtes vers le backend
    ProxyPass /api http://localhost:3001/api
    ProxyPassReverse /api http://localhost:3001/api

    # Logs pour le port 80
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Cette configuration permet de rediriger toutes les requêtes /api vers le backend.

## 5. Gestion du Backend avec PM2

Pour garantir la stabilité du backend, nous utilisons PM2, un gestionnaire de processus qui permet de :

- Garder le backend en fonctionnement permanent.
- Redémarrer automatiquement le backend en cas de crash.

La commande pour démarrer le backend avec PM2 est la suivante :

```
pm2 start /var/www/SAE/Backend/server.js --name backend
```

---

## **6. Problèmes Rencontrés**

### **6.1. Problème de Ports Occupés**

Initialement, des erreurs indiquaient que le port 3000 était déjà utilisé par Apache. Pour résoudre ce conflit, nous avons configuré le backend pour écouter sur le port 3001.

### **6.2. Erreur 404 pour les Routes API**

Malgré la redirection configurée, les routes API renvoyaient une erreur 404. Ce problème a été résolu en s'assurant que les appels API incluaient bien /api dans leurs chemins.

### **6.3. Fonctionnement en Local vs Réseau Universitaire**

Le projet fonctionnait correctement en local, ou encore dans un réseau domestique, mais rencontrait des problèmes sur le réseau de l'université. Cela était dû aux restrictions du réseau qui bloquaient l'accès aux ports non standards (autres que 80 et 443).

## 7. Commandes Principales

- Démarrer le Backend :

Cette commande a permis de lancer le backend en arrière-plan.

```
pm2 start /var/www/SAE/Backend/server.js --name backend
```

```
root@srv622013:~# pm2 status backend
```

id	name	mode	⌘	status	cpu	memory
0	backend	fork	363	online	0%	68.0mb

- Redémarrer Apache :

Pour lancer Apache, nous avons exécuté :

```
sudo systemctl restart apache2
```

```
root@srv622013:~# sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; preset: enab
   Active: active (running) since Thu 2024-11-14 06:23:36 UTC; 7h ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 47434 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/S
   Main PID: 47439 (apache2)
      Tasks: 55 (limit: 4686)
     Memory: 12.9M
        CPU: 2.132s
    CGroup: /system.slice/apache2.service
            └─47439 /usr/sbin/apache2 -k start
              └─47440 /usr/sbin/apache2 -k start
                └─47441 /usr/sbin/apache2 -k start

Nov 14 06:23:36 srv622013 systemd[1]: Starting apache2.service - The Apache HTTP
Nov 14 06:23:36 srv622013 systemd[1]: Started apache2.service - The Apache HTTP
```

- Construire le Frontend :

Pour générer la version de production du frontend, nous avons utilisé :

```
npm run build
```

```
The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:

  https://cra.link/deployment

root@srv622013:/var/www/SAE/Frontend#
```

## 8. Résultat Final

Après plusieurs ajustements et tests, le projet est maintenant pleinement opérationnel, que ce soit en local ou sur le réseau de l'université. Le frontend est servi par Apache sur le port 80, tandis que les requêtes API sont redirigées de manière transparente vers le backend Node.js sur le port 3001. Cette configuration garantit une accessibilité stable et un bon fonctionnement de l'application sur n'importe quel réseau.

## 9. Création de trois utilisateurs pour test

Dans le cadre de tests pour le gestionnaire de mots de passe, nous avons créé trois utilisateurs fictifs avec leurs identifiants et une liste de services associés. Ce compte rendu présente les détails de ces utilisateurs et explique comment les utiliser pour effectuer les tests nécessaires.

### **1. Utilisateur : David**

- **Nom d'utilisateur** : David
- **Mot de passe principal** : cl8Jv6ny@QBv

### **2. Utilisateur : Bob**

- **Nom d'utilisateur** : Bob
- **Mot de passe principal** : ajgbo@)MgJwU

### **3. Utilisateur : Alice**

- **Nom d'utilisateur** : Alice
- **Mot de passe principal** : eBfaov3XOKoP

Ces trois utilisateurs et leurs services associés permettront de tester les fonctionnalités suivantes du gestionnaire de mots de passe :

- Création et gestion d'utilisateurs.
- Ajout, modification, et suppression de services associés.
- Vérification des mots de passe.



---

## 10. Instructions pour tester l'application

Pour vérifier le bon fonctionnement de l'application, nous recommandons de suivre ces étapes simples :

1. **Créer un compte :**
  - Inscrivez-vous avec un identifiant et un mot de passe de votre choix.
2. **Connexion :**
  - Connectez-vous avec les identifiants que vous avez créés.
3. **Tester les fonctionnalités :**
  - Une fois connecté, un petit "i" d'information situé sur l'interface vous guidera pour tester les fonctionnalités principales : ajout, modification, et suppression de mots de passe.
  - Ces tests permettront de valider que tout fonctionne correctement.