

Projet de Chiffrement et Déchiffrement des Données avec Flask et React

1. Titre

Présentation du projet de sécurité des données avec stockage en base de données SQLite

2. Introduction

Contenu :

- ***Objectif du Projet :***

- Créer un système de chiffrement et de déchiffrement des données à l'aide de Flask (Backend) et React (Frontend).
- Les données chiffrées seront stockées dans une base de données SQLite.
- L'application devra permettre à l'utilisateur de chiffrer une donnée, la stocker, et la récupérer pour déchiffrement.

❖ 3. Téléchargement de GLPI et configuration de l'App Token pour la sécurité

Explication :

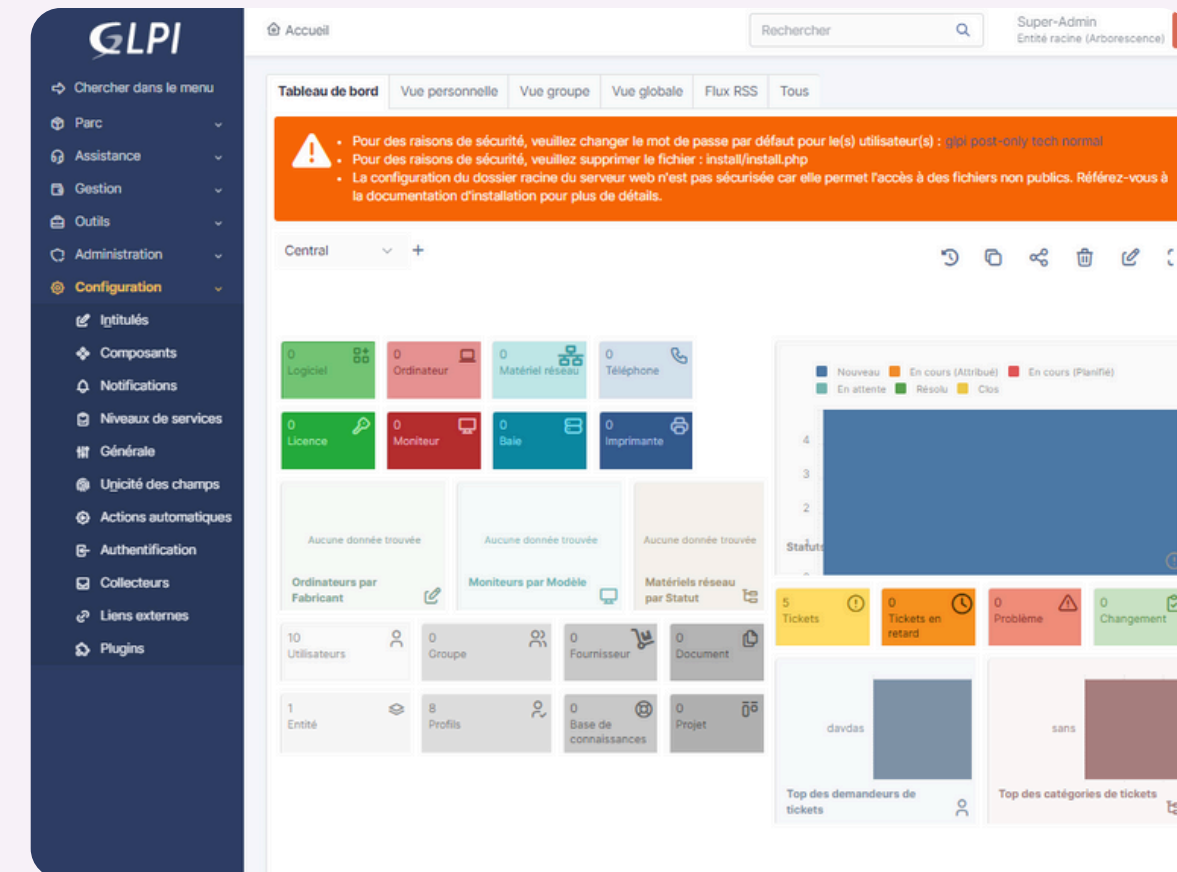
- Téléchargez Glpi sur <https://glpi-project.org/fr/telecharger-glpi/>
- Ajoutez-le sur votre serveur local, par exemple dans le dossier htdocs de XAMPP.
- Décompressez l'archive GLPI et lancez-le sur votre serveur local. Vérifiez que toutes les dépendances sont à jour.
- Connectez-vous avec les identifiants par défaut :
- Identifiant : glpi
- Mot de passe : glpi
- Dans le menu de gauche, accédez à Configuration → Générale → API.



Presentation 1



Presentation 2

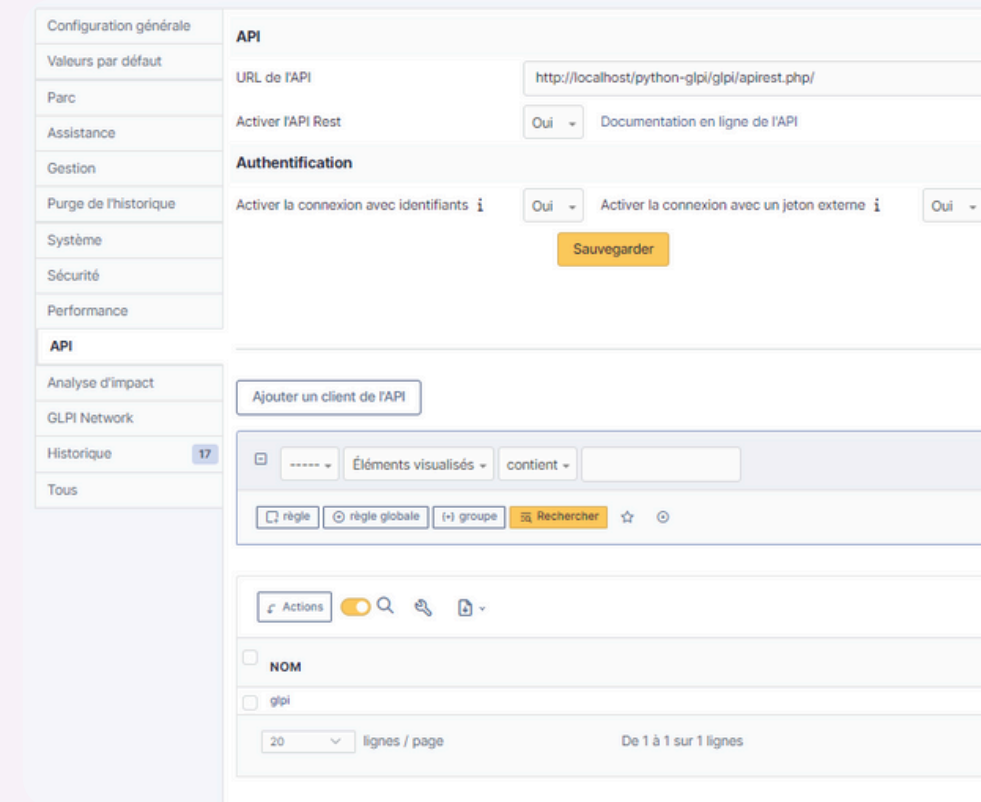


Presentation 3

3. Telechargement de Glpi et App_token pour l'aspect sécurité

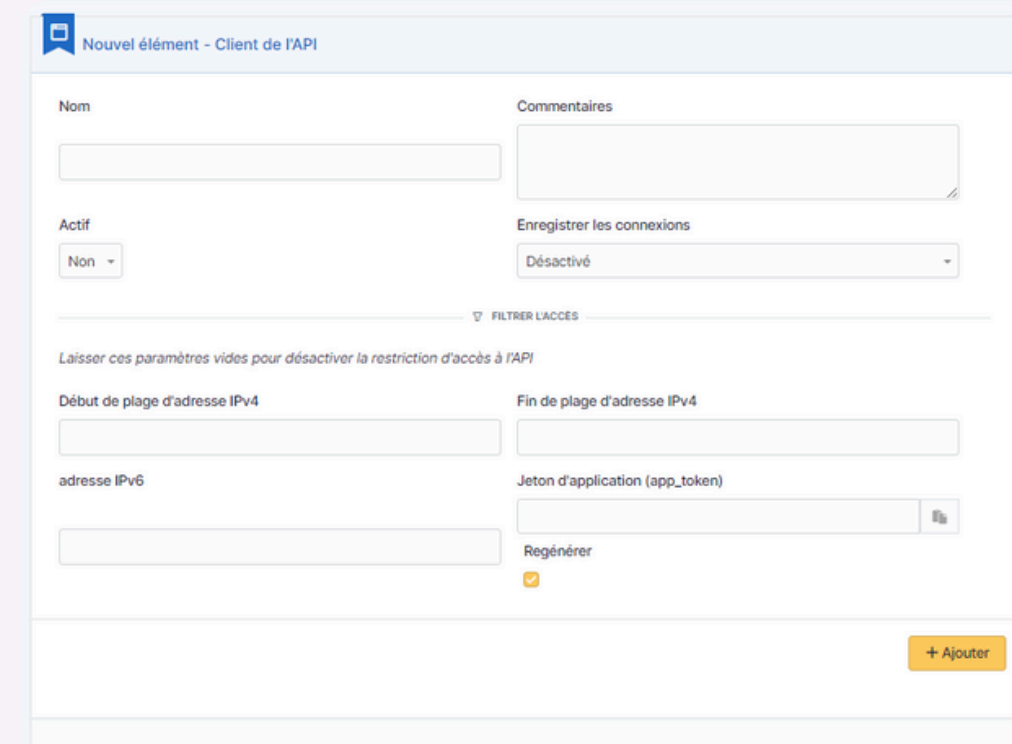
Explication :

- Vérifiez que l'URL de l'API est correcte, puis activez les autres options.
- Cliquez sur "Ajouter un client de l'API", puis remplissez le formulaire selon vos préférences.
- Votre nom s'affichera en bas de la section "Ajouter un client de l'API".
- En enregistrant, vous obtiendrez votre App Token, qui vous permettra d'effectuer des manipulations via l'API.



The screenshot shows the 'API' configuration page in GLPI. On the left is a sidebar menu with options like 'Configuration générale', 'Valeurs par défaut', 'Parc', 'Assistance', 'Gestion', 'Purge de l'historique', 'Système', 'Sécurité', 'Performance', 'API', 'Analyse d'impact', 'GLPI Network', 'Historique', and 'Tous'. The main content area is titled 'API' and includes sections for 'API' (with a text input for 'URL de l'API' set to 'http://localhost/python-glpi/glpi/apirest.php/'), 'Authentification' (with checkboxes for 'Activer l'API Rest' and 'Activer la connexion avec un jeton externe'), and a list of API clients. A 'Sauvegarder' button is visible. Below the configuration, there's a section to 'Ajouter un client de l'API' with a search bar and a table showing one client named 'glpi'.

Presentation 4



The screenshot shows the 'Nouvel élément - Client de l'API' form. It has fields for 'Nom', 'Commentaires', 'Actif' (a dropdown menu), and 'Enregistrer les connexions' (a dropdown menu). Below these are fields for 'Début de plage d'adresse IPv4', 'Fin de plage d'adresse IPv4', 'adresse IPv6', and 'Jeton d'application (app_token)'. There is a 'Regénérer' checkbox and a '+ Ajouter' button at the bottom right.

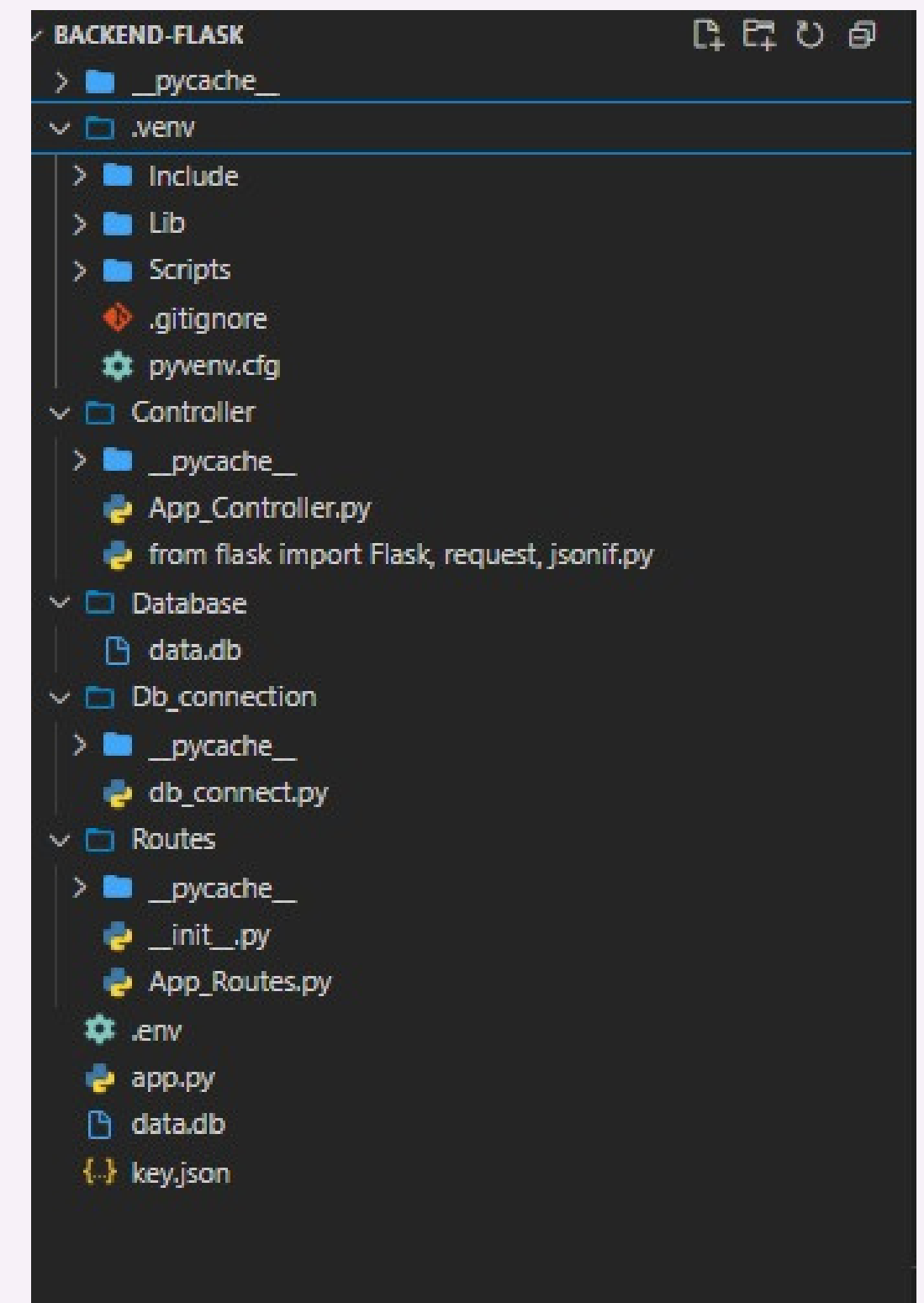
Presentation 5

4. Structure du Projet

Explication :

Description : La structure du projet est organisée de manière claire pour séparer les différentes parties du projet (Frontend, Backend, Base de données). Cela permet de maintenir une bonne organisation et facilite la gestion des différents fichiers et modules.

```
/python-test
├── /backend-flask
│   ├── /Db_connection
│   │   └── db_connect.py    # Connexion à la base de données SQLite
│   ├── /Controller
│   │   └── App_Controller.py # Gestion du chiffrement et déchiffrement
│   ├── /Routes
│   │   └── App_Routes.py    # Routes Flask pour les requêtes
│   ├── /database
│   │   └── data.db          # Base de données SQLite
│   ├── app.py              # Point d'entrée de l'application Flask
│   ├── .env                # Fichier des variables d'environnement
│   ├── requirements.txt     # Dépendances du backend
│   └── key.json             # Clé de chiffrement générée
├── /frontend-input
│   ├── /src
│   │   ├── App.js          # Composant principal React
│   │   └── EncryptForm.js  # Formulaire d'encryptage et de décryptage
│   ├── package.json        # Dépendances du frontend React
│   ├── .env                # Variables d'environnement du frontend
│   ├── README.md           # Documentation générale du projet
│   └── .gitignore           # Fichiers à ignorer par Git
```



5. Fonctionnement du Système

Le code est organisé en plusieurs fonctions pour le rendre modulaire et réutilisable.

Explication :

- Chiffrement (Backend Flask)
- L'utilisateur envoie des données via le frontend. Flask génère une clé de chiffrement. Les données sont chiffrées avec cette clé et stockées dans une base de données SQLite.
- La clé de chiffrement est ensuite sauvegardée dans un fichier key.json.
- Déchiffrement (Backend Flask)
- L'utilisateur envoie un identifiant de donnée à déchiffrer.
- Flask récupère la donnée chiffrée depuis la base de données.
- La donnée est déchiffrée avec la clé correspondante et renvoyée au frontend.

6. Le Choix de la Méthode de Chiffrement

Le code est organisé en plusieurs fonctions pour le rendre modulaire et réutilisable.

Explication :

- Méthode choisie : Fernet (Cryptography)
- Avantages :
- Sécurité élevée : Utilisation de clés symétriques et sécurisées.
- Facilité d'utilisation : La bibliothèque cryptography offre une API simple.
- Performances : Rapide pour chiffrer et déchiffrer des données en temps réel.
- Inconvénients :
- Limité à des données de taille modérée.

8. Code Backend (Flask)

Explication :

- Description : Le backend est développé avec Flask. Il gère les routes pour le chiffrement et le déchiffrement, ainsi que la gestion de la base de données.

App.py:

```
from flask import Flask
from flask_cors import CORS
from Routes.App_Routes import routes
from Db_connection.db_connect import init_db

app = Flask(__name__)
CORS(app)

app.register_blueprint(routes)

if __name__ == "__main__":
    print("\n Initialisation de la base de données...")
    init_db()
    print("\n Serveur Flask démarré sur http://localhost:5000")
    app.run(debug=True)
```

Routes.py :

```
@routes.route("/encrypt", methods=["POST"])
def encrypt_data_route():
    data = request.json.get("data")
    if not data:
        return jsonify({"error": "Aucune donnée fournie"}), 400

    data_id, error = encrypt_data(data)
    if error:
        return jsonify({"error": error}), 500
    return jsonify({"message": f"Donnée chiffrée avec succès, ID: {data_id}"}), 201
```

9. Code Backend (Flask)

Explication :

- Chiffrement : Le Controller prend les données, génère une clé de chiffrement (si elle n'existe pas), chiffre les données et les insère dans la base de données.
- Déchiffrement : Le Controller prend un identifiant de données, récupère les données chiffrées depuis la base de données, les déchiffre à l'aide de la clé et renvoie la donnée déchiffrée.

1. Code du Controller – Fonction get_key()

Objectif : Gérer la clé de chiffrement. Si la clé existe déjà, elle est lue depuis un fichier JSON. Sinon, une nouvelle clé est générée et stockée.

```
def get_key():
    try:
        if os.path.exists("key.json"):
            with open("key.json", "r") as file:
                return json.load(file)["key"].encode()
        else:
            key = Fernet.generate_key()
            with open("key.json", "w") as file:
                json.dump({"key": key.decode()}, file)
            return key
    except Exception as e:
        print(f"Erreur lors de la gestion de la clé: {e}")
        return None
```

2. Code du Controller – Fonction encrypt_data()

Objectif : Chiffrer les données et les insérer dans la base de données.

```
def encrypt_data(data):
    key = get_key()
    if key is None:
        return None, "Erreur de clé"

    cipher = Fernet(key)
    encrypted_data = cipher.encrypt(data.encode())

    try:
        conn = get_db_connection()
        if conn:
            cursor = conn.cursor()
            cursor.execute("INSERT INTO encrypted_data (data) VALUES (?)", (encrypted_data.decode(),))
            conn.commit()
            data_id = cursor.lastrowid
            conn.close()
            return data_id, None # Return ID and no error
        else:
            return None, "Erreur de connexion à la base de données"
    except Exception as e:
        return None, f"Erreur de base de données: {e}"
```

3. Code du Controller – Fonction decrypt_data()

- Objectif : Déchiffrer les données à partir de leur identifiant et renvoyer la donnée déchiffrée.

```
def decrypt_data(data_id):
    key = get_key()
    if key is None:
        return None, "Erreur de clé"

    cipher = Fernet(key)

    try:
        conn = get_db_connection()
        if conn:
            cursor = conn.cursor()
            cursor.execute("SELECT data FROM encrypted_data WHERE id = ?", (data_id,))
            row = cursor.fetchone()
            conn.close()

            if row:
                decrypted_data = cipher.decrypt(row["data"].encode()).decode()
                return decrypted_data, None # Return decrypted data and no error
            else:
                return None, "Donnée non trouvée"
        else:
            return None, "Erreur de connexion à la base de données"
    except Exception as e:
        return None, f"Erreur de base de données: {e}"
```

10. Code Frontend – React

Le frontend de l'application est développé avec React, un framework populaire de JavaScript qui permet de construire des interfaces utilisateurs dynamiques et interactives. Dans cette application, React est utilisé pour créer un formulaire permettant à l'utilisateur de chiffrer et de déchiffrer des données. L'application frontend interagit avec le backend Flask via des requêtes HTTP, en utilisant fetch API pour envoyer les données vers le serveur pour traitement (chiffrement/déchiffrement). Les utilisateurs peuvent soumettre des données texte à chiffrer (via la route /encrypt) ou un ID de donnée pour déchiffrer (via la route /decrypt/<data_id>). Le frontend reçoit ensuite les résultats et les affiche dans l'interface utilisateur.

Objectifs du Code Frontend :

1. Permettre à l'utilisateur de saisir des données à chiffrer ou à déchiffrer.
2. Envoyer les données au serveur Flask via une requête POST (pour le chiffrement) ou GET (pour le déchiffrement).
3. Afficher les réponses du serveur (chiffrement/déchiffrement ou messages d'erreur).

```
function EncryptForm() {
  const [data, setData] = useState('');
  const [responseMessage, setResponseMessage] = useState('');
  const [dataId, setDataId] = useState('');
  const [decryptedData, setDecryptedData] = useState('');

  const handleInputChange = (event) => {
    setData(event.target.value);
  };

  const handleEncryptSubmit = async (event) => {
    event.preventDefault();
    const response = await fetch('http://localhost:5000/encrypt', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ data })
    });

    const result = await response.json();
    setResponseMessage(result.message || result.error);
  };

  const handleDecryptSubmit = async (event) => {
    event.preventDefault();
    const response = await fetch(`http://localhost:5000/decrypt/${dataId}`);
    const result = await response.json();
    setDecryptedData(result.decrypted_data || result.error);
  };

  return (
    <div>
      {/* Formulaires de chiffrement et déchiffrement */}
    </div>
  );
}
```


11. Requêtes SQL – Affichage des Données

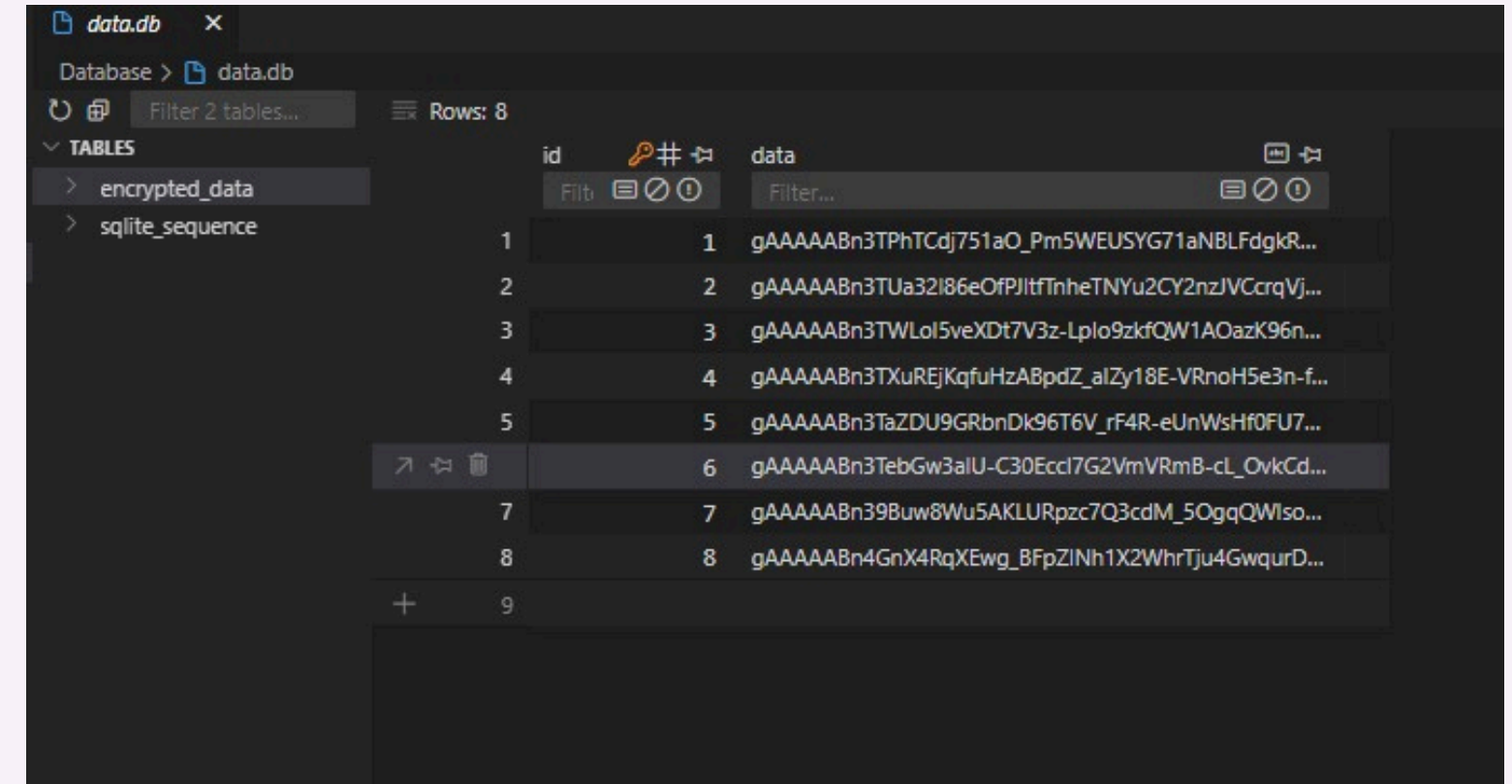
Dans l'application, les données chiffrées sont stockées dans une base de données SQLite. La table `encrypted_data` contient deux colonnes principales :

- ***id*** : Un identifiant unique généré automatiquement pour chaque entrée.
- ***data*** : La donnée elle-même, qui est stockée sous forme chiffrée.

Les données sont insérées dans cette table après avoir été chiffrées et peuvent être récupérées par leur identifiant unique pour être déchiffrées par la suite.

Visuel :

Vous pouvez ajouter une image illustrant la table `encrypted_data` de la base de données, montrant une ligne avec l'id et la donnée chiffrée.

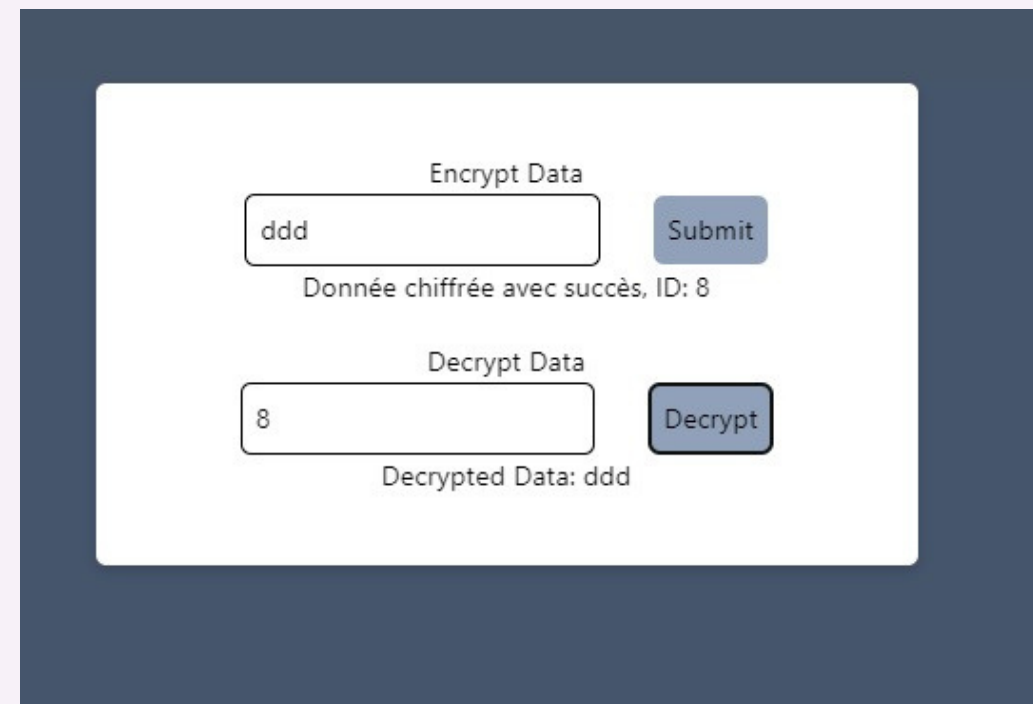


The screenshot shows a SQLite database viewer interface. The database is named 'data.db'. The table 'encrypted_data' is selected, showing 8 rows. The columns are 'id' and 'data'. The data is encrypted and appears as long strings of characters.

id	data
1	gAAAAABn3TPhTCdj751aO_Pm5WEUSYG71aNBkFdgkR...
2	gAAAAABn3TUa32l86eOfPjItTnheTNYu2CY2nzJVCcrqVj...
3	gAAAAABn3TWLol5veXDt7V3z-Lplo9zkfQW1AOazK96n...
4	gAAAAABn3TXuREjKqfuHzABpdZ_alZy18E-VRnoH5e3n-f...
5	gAAAAABn3TaZDU9GRbnDk96T6V_rF4R-eUnWsHf0FU7...
6	gAAAAABn3TebGw3aIU-C30EccI7G2VmVRmB-cl_OvkCd...
7	gAAAAABn39Buw8Wu5AKLURpzc7Q3cdM_5OgqQWiso...
8	gAAAAABn4GnX4RqXEwg_BFpZINh1X2WhrTju4GwqurD...

10. Présentation des résultats

L'interface React permet de chiffrer et déchiffrer des données facilement. Après le chiffrement, un message de succès avec un ID est affiché. Lors du déchiffrement, l'utilisateur obtient la donnée déchiffrée via l'ID.



The screenshot shows a web interface with a dark blue background. It contains two main sections: 'Encrypt Data' and 'Decrypt Data'. In the 'Encrypt Data' section, there is a text input field containing 'ddd' and a blue 'Submit' button. Below the input field, the text 'Donnée chiffrée avec succès, ID: 8' is displayed. In the 'Decrypt Data' section, there is a text input field containing '8' and a blue 'Decrypt' button. Below the input field, the text 'Decrypted Data: ddd' is displayed.

Ce projet démontre comment sécuriser le chiffrement et le déchiffrement des données en utilisant Flask pour le backend et React pour l'interface frontend. L'intégration de bibliothèques comme cryptography et SQLite a permis de garantir la sécurité des données tout en offrant une expérience utilisateur fluide et intuitive.

N'hésitez pas si vous avez des questions.
Merci pour votre attention !