

Promotion 2023
Lucie Bottin & Céline KHAUV

SÉCURITÉ INFORMATIQUE

Projet | SQL Injection



SOMMAIRE

INTRODUCTION

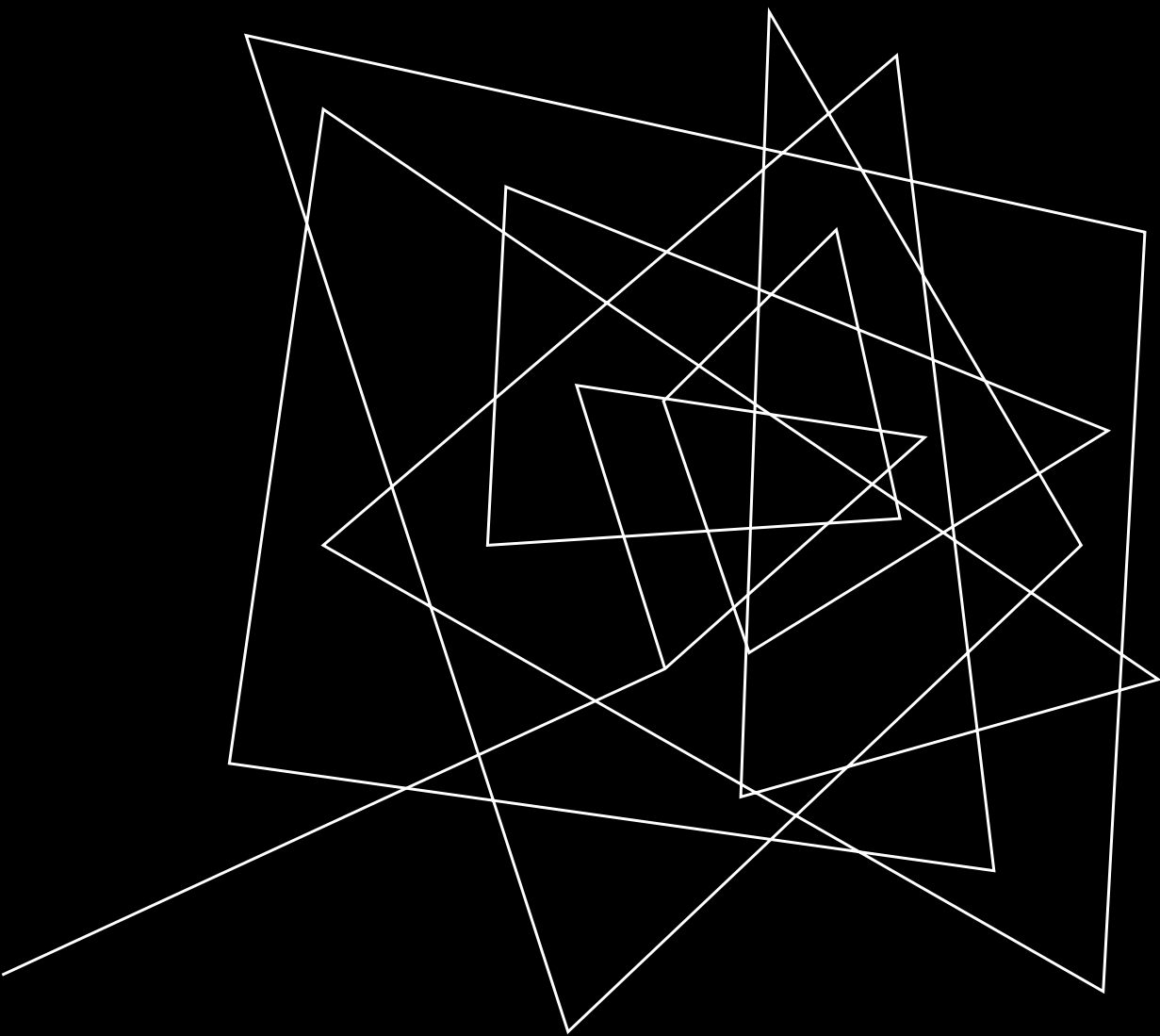
Présentation du Javascript et de PostgreSQL

LA BASE DES INJECTIONS SQL

Explications et exemples

NOTRE PROJET

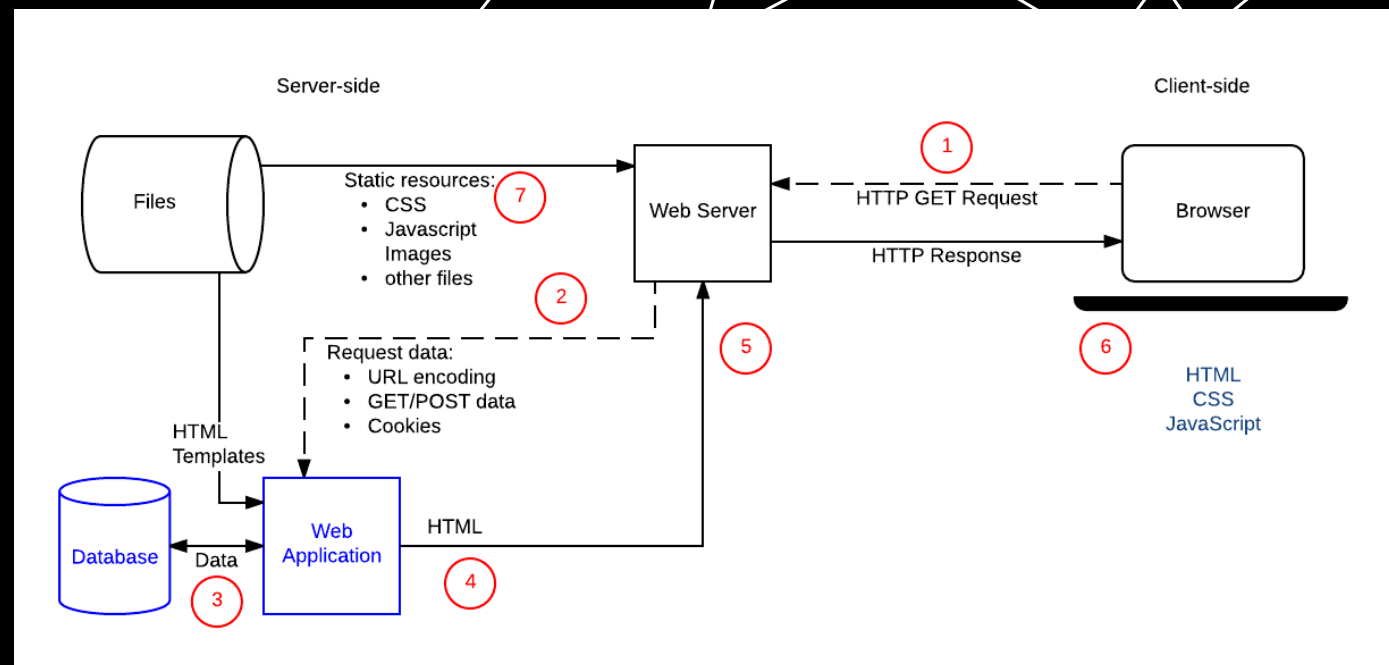
Application des injections SQL dans notre projet et solutions



INTRODUCTION

QU'EST CE QUE LE JAVASCRIPT ?

JavaScript est un langage de script populaire permettant d'ajouter des fonctionnalités interactives et d'autres contenus dynamiques aux pages web. Les exemples les plus connus de contenu JavaScript sont les formulaires à remplir, les diaporamas de galeries de photos et les graphiques animés.



Source : https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview

EXEMPLES DE CODE

1

```
function myJsFunction(){  
  var x = document.getElementById('input1').value;  
  document.write(x);  
}
```

2

```
// gets data from API and sets the content of #result div  
const getData = function () {  
  result.innerText = "Loading...";  
  fetch("https://api.github.com/users/gulshansainis")  
    .then((res) => res.json())  
    .then((data) => {  
      result.innerText = JSON.stringify(data, null, 2);  
    })  
    .catch((error) => console.log(error));  
};
```



QU'EST CE QUE POSTGRESQL ?

PostgreSQL est un système de gestion de base de données relationnelle orienté objet puissant et open source qui est capable de prendre en charge en toute sécurité les charges de travail de données les plus complexes.

NODE.JS

Définition

NodeJS est un environnement d'exécution permettant d'utiliser le JavaScript côté serveur. Grâce à son fonctionnement non bloquant, il permet de concevoir des applications en réseau performantes, telles qu'un serveur web, une API ou un job CRON.

node-postgres

node-postgres est une bibliothèque JavaScript pure qui permet d'interagir avec une base de données PostgreSQL.

node-postgres peut être facilement installé dans un projet en installant le paquet pg : *npm install pg*

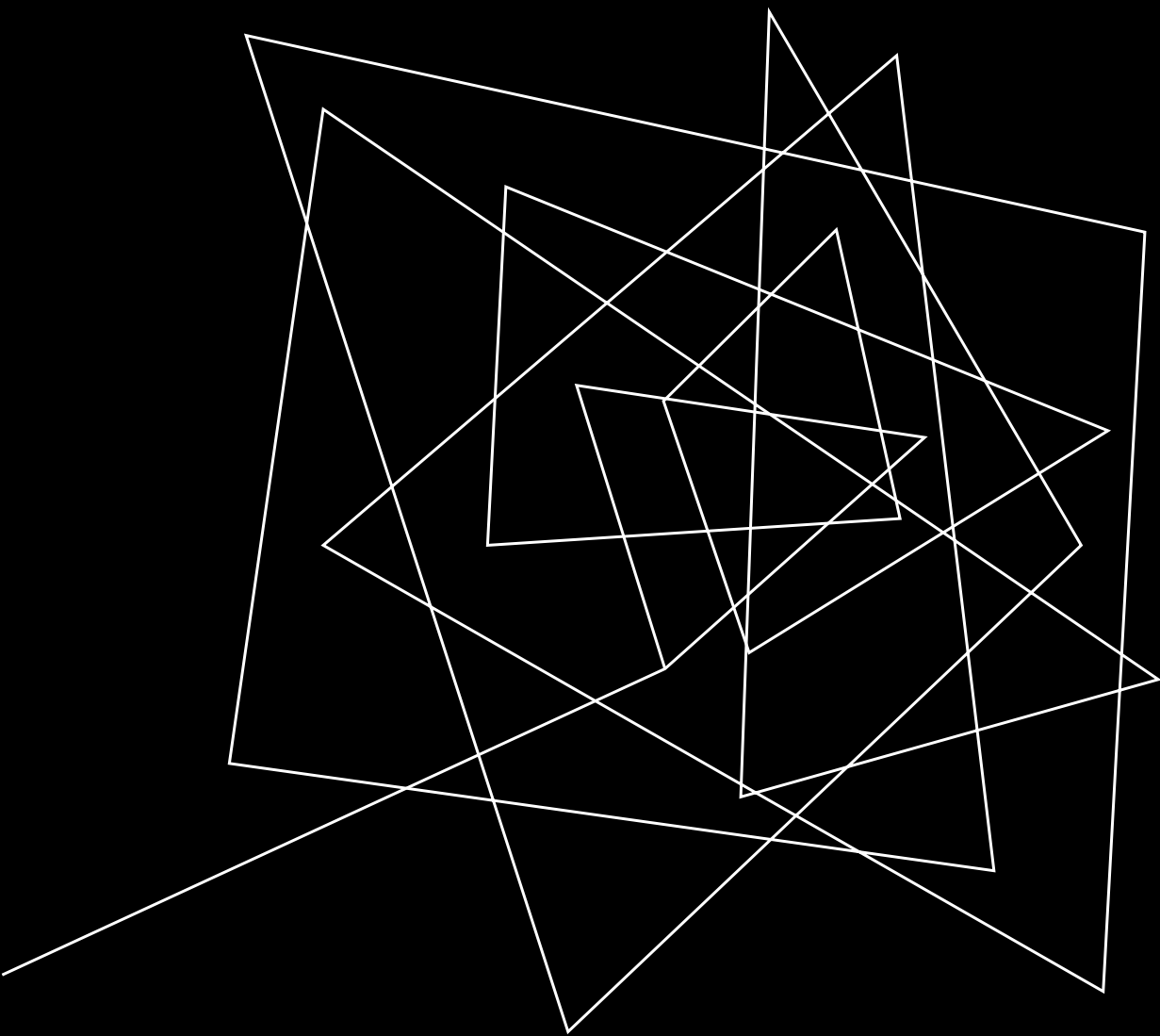
```
const { Pool } = require("pg");

const credentials = {
  user: "postgres",
  host: "localhost",
  database: "efrei",
  password: "yourpassword",
  port: 5432,
};

async function poolQuery() {
  const pool = new Pool(credentials);
  const now = await pool.query("SELECT * FROM users", (err, response) => {
    if (err) {
      throw err;
    }
    if (response.rows.length > 0) {
      console.log(response.rows);
    }
  });
  await pool.end();
  return now;
}
```

SE CONNECTER À POSTGRES DEPUIS NODE

Il existe plusieurs façons de se connecter à une base de données. Par exemple en utilisant un pool de connexion ou simplement en instanciant un client.

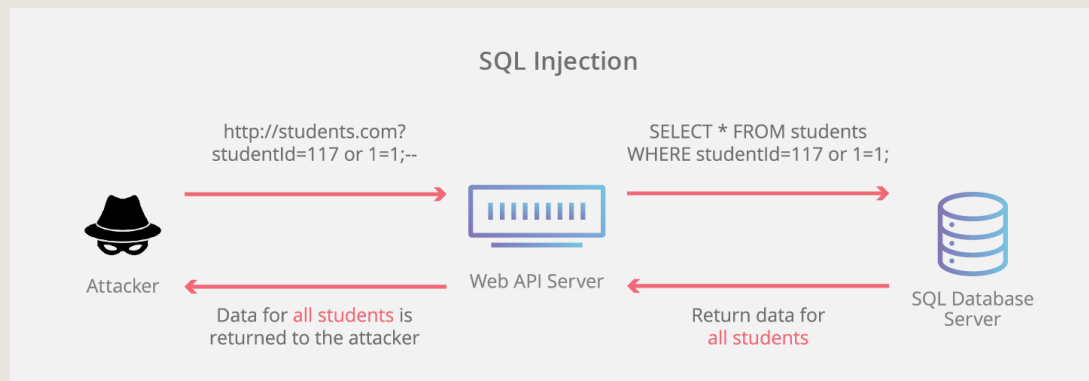


LA BASE DES INJECTIONS SQL

INTRODUCTION

SQL (Structured Query Language, en français, langage de requête structurée) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. La partie langage de manipulation des données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles

L'injection SQL est une technique où un pirate modifie une requête SQL existante pour afficher des données cachées, pour écraser des valeurs importantes ou encore exécuter des commandes dangereuses pour la base. Cela se fait lorsque l'application prend les données envoyées par l'internaute et l'utilise directement pour construire une requête SQL.



EXPLOITATION

Prenons l'exemple d'un forme où l'on doit rentrer un username et un mot de passe. La requête suivante sera exécutée pour vérifier l'existence de l'utilisateur ainsi que son mot de passe associé :

```
SELECT * FROM users where name = 'admin' and password = 'password'
```

Il suffit que l'attaquant rentre username'# dans le premier champs pour avoir :

```
"SELECT * FROM users where name = 'admin'# and password = 'password'
```

Le reste de la requête est passé en commentaire et sera ignoré. L'attaquant à maintenant accès à toutes les données de l'utilisateur 'admin'.

ATTAQUE BOOLÉENNE

Lors d'une injection SQL booléenne, l'attaquant cherche les entrées d'un utilisateur qui peuvent être vulnérables aux injections SQL en essayant ces requêtes :

- "... and 1=1"
- "... and 1=2"

Il peut également modifier la une requête. Reprenons l'exemple du formulaire, on obtient :

```
SELECT * FROM users where name = 'admin' and password = 'password' or 1= 1
```

La condition sera toujours vraie donc le mot de passe ne sera jamais vérifié.

ATTAQUE BASÉE SUR L'UNION

Grâce à l'union, l'attaquant peut exécuter une autre requête et afficher son résultat.
Par exemple :

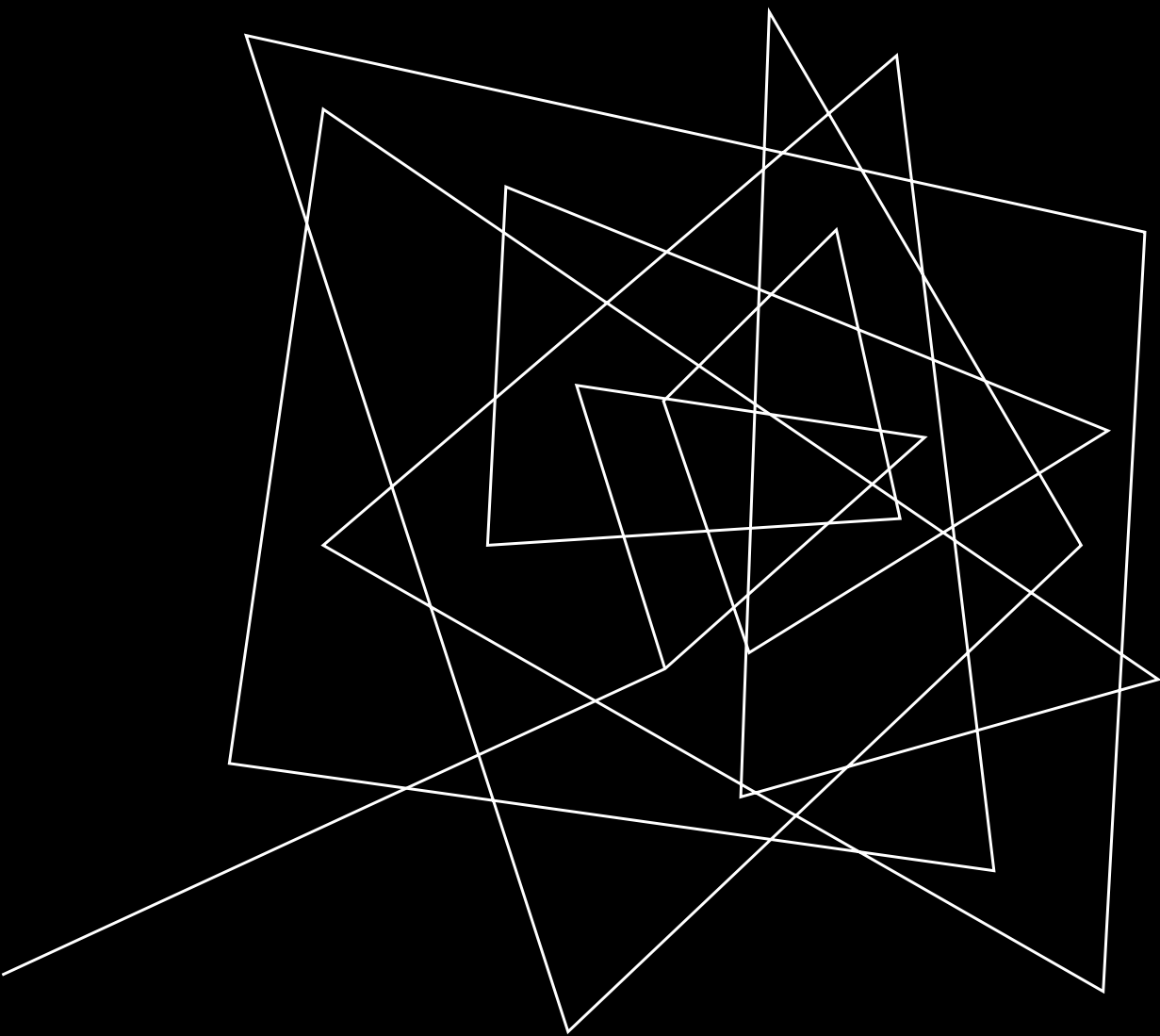
```
SELECT title FROM post WHERE id = 10 UNION  
SELECT * FROM users WHERE name = 'admin' AND password = 'password'
```

ATTAQUE BASÉE SUR LES ERREURS

Le but est d'obtenir plus d'informations sur la structure de la base de données et les noms des tables que l'application web suit. Par exemple, un message d'erreur peut contenir le nom de la table inclus dans la requête et les noms des colonnes de la table. Ces données peuvent ensuite être utilisées pour créer de nouvelles attaques.

COMMENT FAIRE UNE ATTAQUE ?

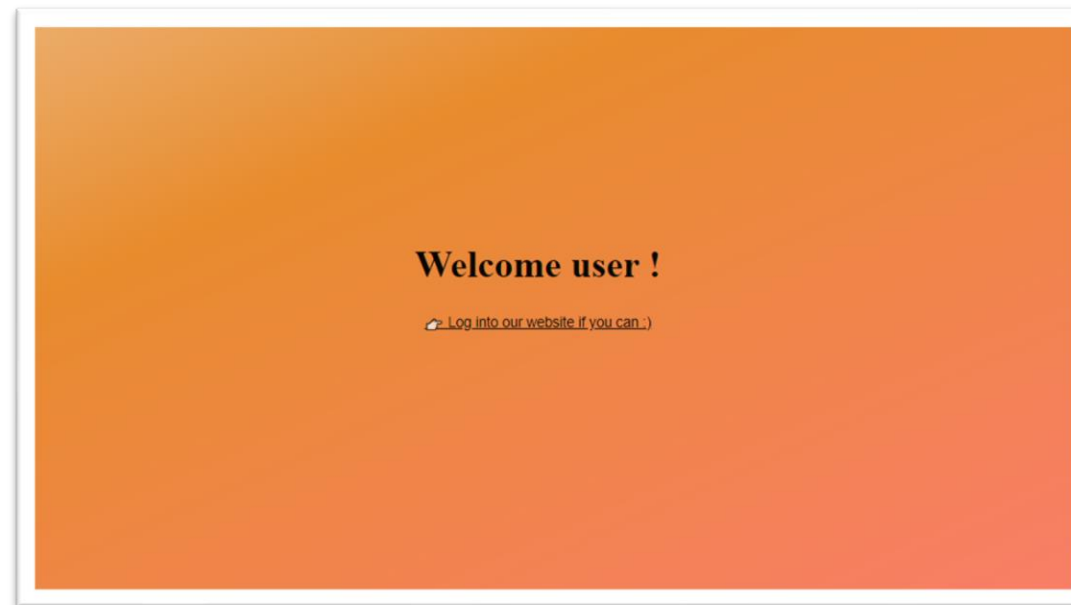
Une attaque SQL peut être fait de plusieurs manières : dans les entrées de l'utilisateurs, sur l'url de site web, via les en-têtes HTTP, etc...



NOTRE PROJET

PRÉSENTATION DU SITE

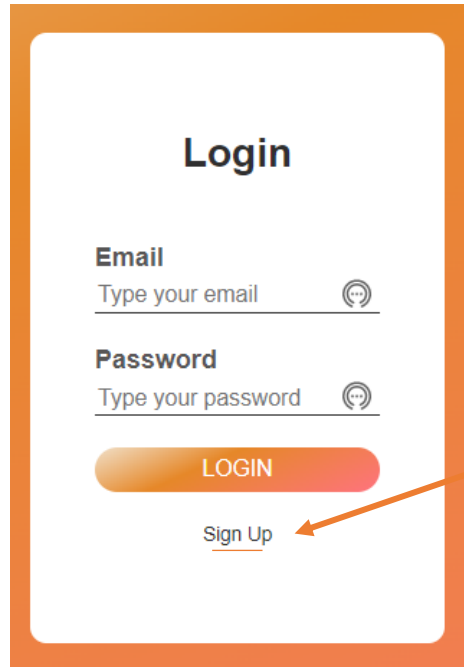
Lorsque l'utilisateur arrive sur le site voici la première vue qui lui est présentée



PRÉSENTATION DU SITE

<http://localhost:4000/users/login>

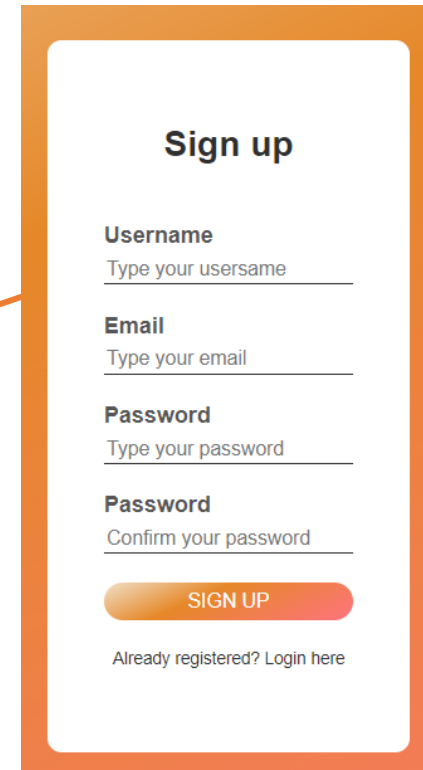
Sur la page de *login*,
l'utilisateur pourra
entrer ses identifiants
si il possède déjà un
compte



The login form is titled "Login" and is enclosed in an orange border. It contains two input fields: "Email" with the placeholder "Type your email" and a clear button, and "Password" with the placeholder "Type your password" and a clear button. Below the password field is a red "LOGIN" button. At the bottom, there is a link "Sign Up" with an arrow pointing to the register form.

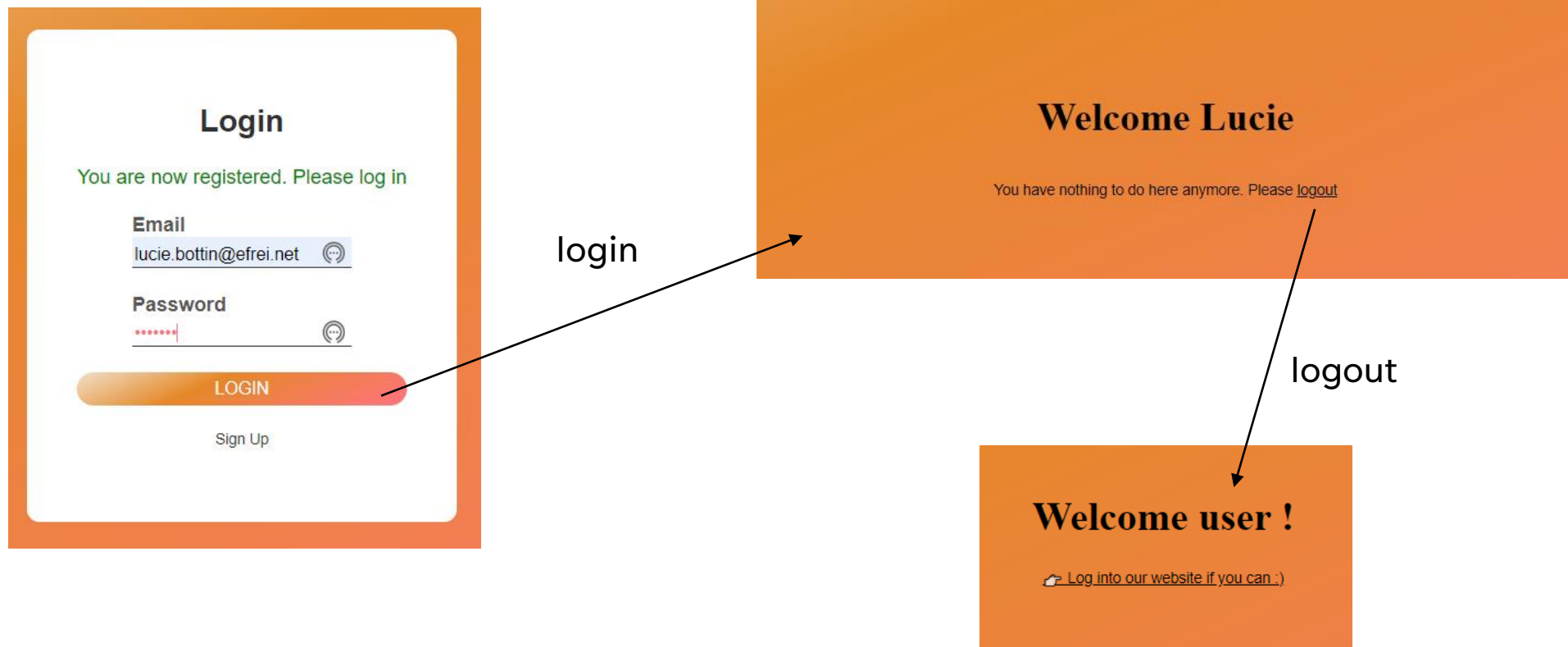
<http://localhost:4000/users/register>

Si il n'a pas encore de
compte, il pourra
s'enregistrer sur la
page *register*



The sign up form is titled "Sign up" and is enclosed in an orange border. It contains three input fields: "Username" with the placeholder "Type your username", "Email" with the placeholder "Type your email", and "Password" with the placeholder "Type your password". Below the password field is a second "Password" field with the placeholder "Confirm your password". At the bottom is a red "SIGN UP" button. Below the button is a link "Already registered? Login here".

LOGIN DE L'UTILISATEUR



BASE DE DONNÉES POSTGRESQL

Nous enregistrons les données des utilisateurs dans une base de données PostgreSQL. Lorsque ce dernier s'enregistre, ses identifiants sont gardés ici automatiquement.

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane shows the database structure: Databases (2) > SecuInfo > Schemas (1) > public > Tables (1) > users. The 'Query Editor' pane shows the following SQL query:

```
1 SELECT * FROM public.users
2 ORDER BY id ASC
```

The 'Data Output' pane displays the results of the query in a table format:

	name character varying (200)	email character varying (200)	password character varying (200)	id [PK] integer
1	Lucie	lucie.bottin@efrei.net	passwr	2

Voici les données d'un utilisateur :

- Name
- Email
- Password

SÉCURITÉ DES URLS

```
✓ app.get("/users/register", checkAuthenticated, (req, res) => {  
  res.render("register");  
});  
  
✓ app.get("/users/login", checkAuthenticated, (req, res) => {  
  res.render("login");  
});  
  
✓ app.get("/users/dashboard", checkNotAuthenticated, (req, res) => {  
  res.render("dashboard", { user: req.user.name });  
});
```

checkAuthenticated

Permet de s'assurer qu'un utilisateur est authentifié pour accéder aux pages qu'il entre dans l'url

```
function checkAuthenticated(req, res, next) {  
  if (req.isAuthenticated()) {  
    return res.redirect("/users/dashboard");  
  }  
  next();  
}  
  
function checkNotAuthenticated(req, res, next) {  
  if (req.isAuthenticated()) {  
    return next();  
  }  
  res.redirect("/users/login");  
}
```

checkNotAuthenticated

Permet de s'assurer que si l'utilisateur n'est pas authentifié il doit être redirigé vers la page de login

INJECTIONS SQL BOOLÉENNES

Avec les identifiants suivants :

- **Email** : lucie.bottin@efrei.net
- **Password** : ' or '1'='1

```
email: lucie.bottin@efrei.net  
password: ' or '1'='1  
query: SELECT name FROM users where email = 'lucie.bottin@efrei.net' and password = '' or '1'='1'
```

La query se retrouve face à une condition qui sera toujours vraie : 1=1.

Donc le « or » permet de se connecter à un compte utilisateur sans aucune difficulté.

Afin de faciliter les injections SQL, nous avons décidé de ne pas hash les mots de passe des utilisateurs.

SOLUTION POUR EMPÊCHER LES INJECTIONS SQL

```
function initialize(passport) {
  const authenticateUser = (email, password, done) => {
    console.log(email, password);
    pool.query(
      `SELECT * FROM users WHERE email = ${1}`,
      [email],
      (err, results) => {
        if (err) {
          throw err;
        }
        console.log(results.rows);

        if (results.rows.length > 0) {
          const user = results.rows[0];

          if (password == user.password) {
            return done(null, user);
          } else {
            return done(null, false, { message: "Password is not correct" });
          }
        } else {
          // No user
          return done(null, false, {
            message: "User doesn't exist",
          });
        }
      }
    );
  };
}
```

Cette fonction va permettre de vérifier que le password entré par l'utilisateur est le même que celui qui correspond à cet utilisateur dans la base de données.

Ainsi, nous avons supprimé toute faille dans l'authentification de l'utilisateur

SÉCURITÉ SUPPLÉMENTAIRE

```
<div class="input-wrap">
  <label for="email">Email</label>
  <input type="email" placeholder="Type your email" name="email" />
</div>
```

L'input de l'email est de type « email »
ce qui signifie que si un hacker entre
une injection sql booléenne à la place,
elle ne sera pas prise en compte car se
n'est pas au format email

```
if (!username || !email || !password || !password2) {
  errors.push({ message: "Please enter all fields" });
}
```

Tous les champs doivent être remplis
avant de pouvoir s'inscrire ou
s'identifier. On ne peut donc pas
contourner cela en tentant de se login
sans password par exemple

```
if (password != password2) {
  errors.push({ message: "Passwords do not match" });
}
```

Lors de l'inscription nous avons aussi
ajouté une sécurité sur la
confirmation du mot de passe



SYNTHÈSE

Ce projet nous a permis de découvrir des notions que nous ne connaissions pas et qui sont très intéressantes à aborder. Nous pouvons maintenant nous rendre compte de l'importance de la sécurité dans l'informatique et toujours y faire attention lors de nos développements de sites, applications...