

SECURITY BY DESIGN



l'école d'ingénierie
informatique



Intervenant:

M^{me} Marie Amina



Réalisé par :

- AMROUN Ikram

Contents

1	Mesures mises en place	3
2	Risques couverts	5
3	Tests réalisés et résultats obtenus	6
3.1	1. Test de l'injection SQL	6
3.2	2. Test des attaques XSS	6
3.3	3. Analyse des cookies	7
4	Script de démarrage	7
5	Conclusion	7

Introduction

Le but de ce projet est de créer un **formulaire de contact sécurisé**, avec les bonnes pratiques de sécurité pour protéger les données des utilisateurs. Ce projet couvre :

- La mise en place de l'authentification utilisateur.
- La sécurisation du formulaire via HTTPS.
- L'intégration du reCAPTCHA pour prévenir les attaques automatisées.
- Le stockage sécurisé des données sensibles.

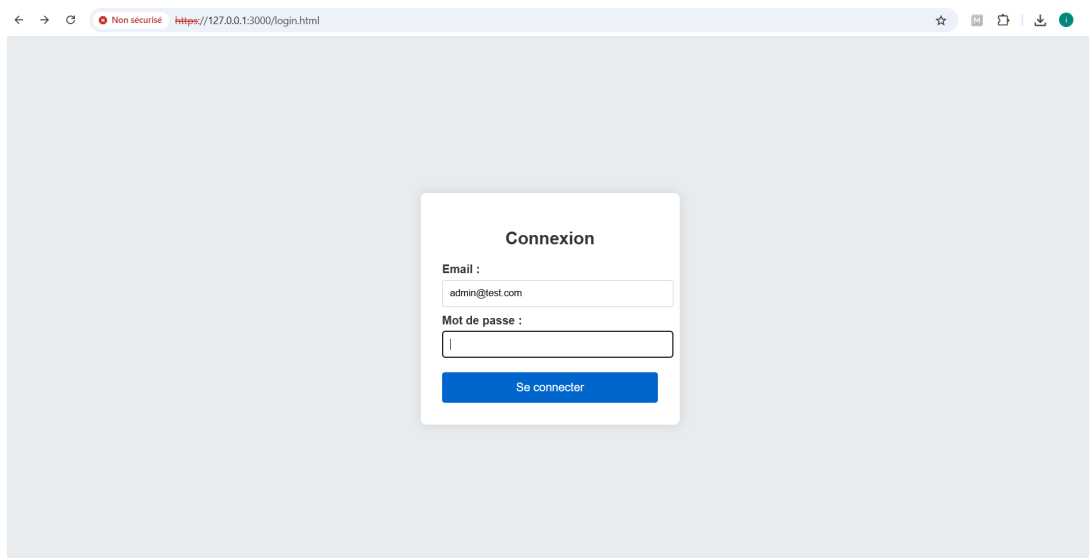


Figure 1: Interface de connexion

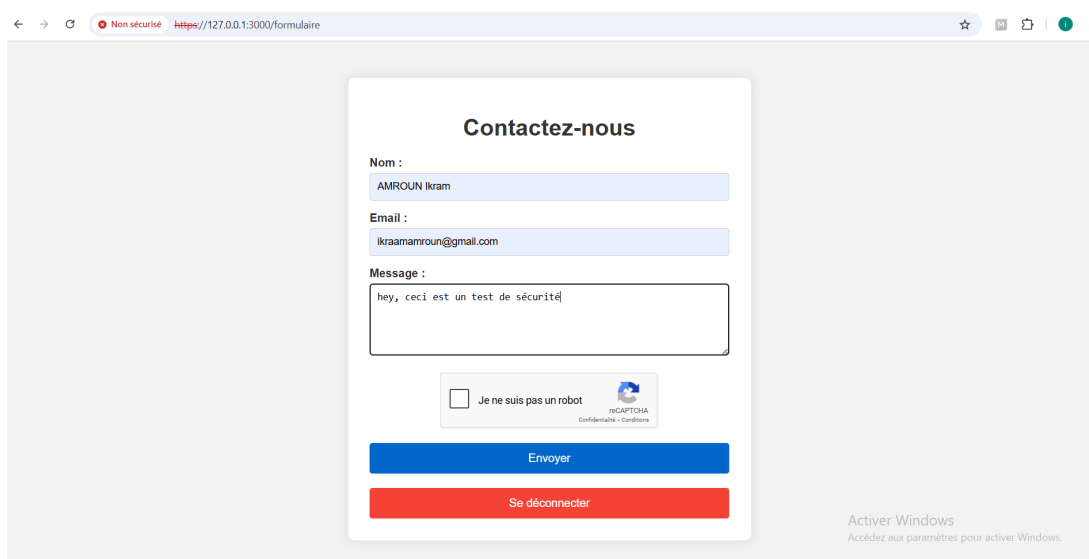


Figure 2: Interface de contact avec CAPTCHA

1 Mesures mises en place

- **Validation CAPTCHA** : Utilisation de Google reCAPTCHA pour vérifier que l'utilisateur est bien humain.

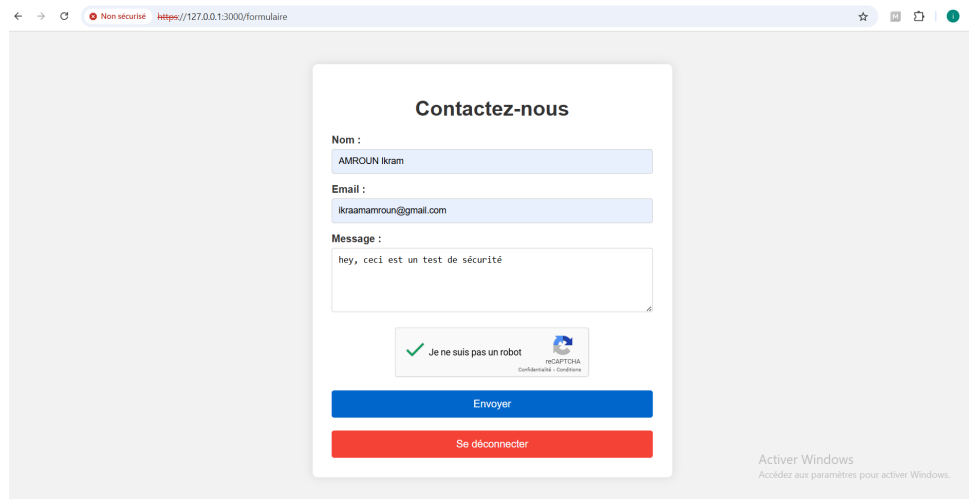


Figure 3: Captcha

- **Authentification sécurisée** : Système d'authentification avec un mot de passe haché et des sessions sécurisées.

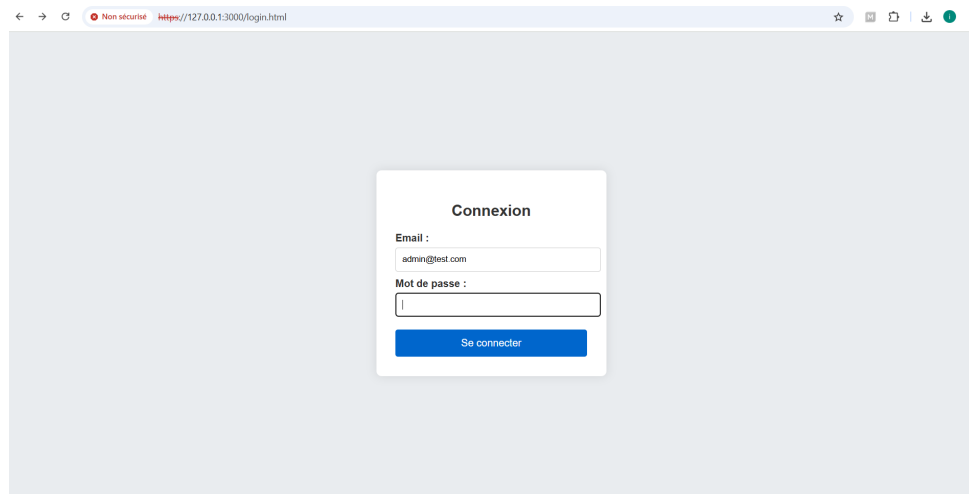


Figure 4: Interface d'authentification

```

try {
  const response = await axios.post(verifyUrl);
  if (!response.data.success) {
    return res.status(403).send('Captcha invalide');
  }

  const hashedEmail = await bcrypt.hash(email, 10);
  const encryptedMsg = Buffer.from(message).toString('base64');

  fs.appendFileSync('./logs/access.log', `Nom: ${name} - Email: ${hashedEmail}\n`);
  res.send('Message reçu et sécurisé.');
```

```

} catch (err) {
  console.error("Erreur lors de la vérification du captcha :", err);
  res.status(500).send('Erreur serveur');
}
});

```

Figure 5: Hashage des identifiants

- **HTTPS avec SSL/TLS** : Le serveur utilise HTTPS avec un certificat SSL auto-signé pour sécuriser la communication.

```

C:\Program Files>mkcert localhost 127.0.0.1 :1
mkcert n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

C:\Program Files>cd C:\Program Files
C:\Program Files>mkcert -install
mkcert n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

C:\Program Files>mkcert -install
The local CA is now installed in the system trust store!
The local CA is now installed in Java's trust store!

C:\Program Files>mkcert localhost 127.0.0.1 :1
Created a new certificate valid for the following names
- "localhost"
- "127.0.0.1"
- ":1"

The certificate is at ".\localhost+2.pem" and the key at ".\localhost+2-key.pem"
It will expire on 12 August 2027

C:\Program Files>

```

Figure 6: Capture d'écran de la configuration des SSL

- **Logs d'accès** : Tous les accès et tentatives de connexion sont enregistrés dans un fichier log sécurisé.

Figure 7: Les accès enregistrés dans un fichier

- **Sécurisation des cookies** : Les cookies sont configurés avec les flags `HttpOnly` et `Secure` pour empêcher l'accès côté client et s'assurer que les cookies sont envoyés uniquement via HTTPS.

```

24
25 // Sessions sécurisées
26 app.use(session({
27     secret: 'monsecret',
28     resave: false,
29     saveUninitialized: false,
30     cookie: {
31         httpOnly: true,
32         secure: true
33     }
34 }));
35

```

Figure 8: Les cookies de sécurisation

2 Risques couverts

Le système couvre plusieurs risques de sécurité, notamment :

- **Injection SQL** : Le formulaire et les entrées de données sont protégés contre les injections SQL, bien qu'aucune base de données ne soit utilisée actuellement.
- **Attaques XSS** : Les entrées de l'utilisateur sont nettoyées pour éviter l'injection de scripts malveillants dans le formulaire.

- **Vol de session** : L'utilisation des cookies sécurisés et la gestion des sessions empêchent le vol de sessions par des attaquants.
- **Interception de données** : Le serveur utilise HTTPS pour garantir la confidentialité et l'intégrité des données échangées, empêchant ainsi les attaques de type Man-in-the-Middle.

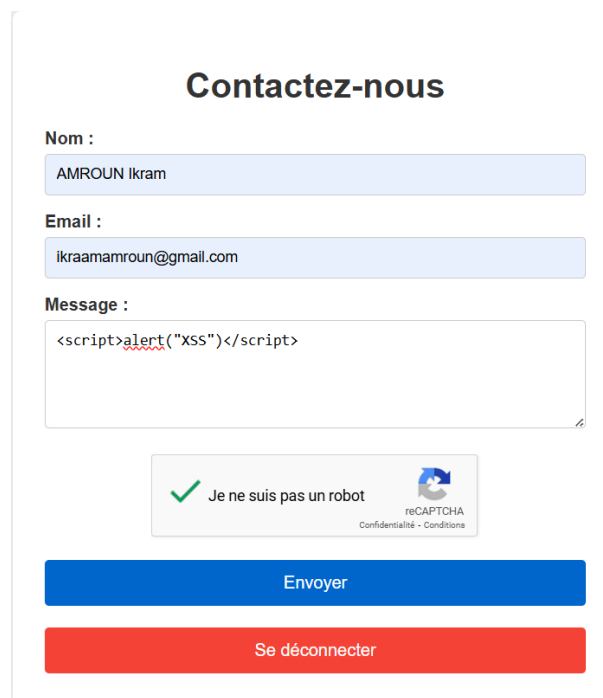
3 Tests réalisés et résultats obtenus

3.1 1. Test de l'injection SQL

Lors des tests d'injection SQL, aucune vulnérabilité n'a été trouvée. En effet, les entrées ne sont pas directement utilisées dans des requêtes SQL, ce qui élimine les risques d'injection SQL.

3.2 2. Test des attaques XSS

Le test avec un script malveillant injecté dans le champ "Message" a échoué. Le script n'a pas été exécuté, garantissant ainsi que le formulaire est protégé contre les attaques XSS.



Contactez-nous

Nom :
AMROUN Ikram

Email :
ikraamamroun@gmail.com

Message :
<script>alert("XSS")</script>

Je ne suis pas un robot

Envoyer

Se déconnecter

Figure 9: Les cookies de sécurisation

3.3 3. Analyse des cookies

Les cookies ont été vérifiés dans les DevTools de Chrome. Ils sont bien configurés avec les flags **Secure** et **HttpOnly**, ce qui empêche leur accès côté client et garantit leur envoi uniquement sur HTTPS.

4 Script de démarrage

Voici le script de démarrage pour exécuter le serveur localement avec HTTPS :

```
# Assurez-vous que Node.js est installé
# Installez les dépendances du projet
npm install express helmet express-session bcrypt axios morgan body-parser xss

# Démarrez le serveur HTTPS
node src/server.js
```

Une fois le serveur démarré, vous pouvez accéder à l'application via `https://127.0.0.1:3000`.

5 Conclusion

Ce projet met en place un formulaire de contact sécurisé avec une protection contre les attaques courantes telles que les injections SQL et les attaques XSS. Le système d'authentification sécurisé avec sessions et HTTPS garantit également la protection des données des utilisateurs.