esaip
École d'Ingénieurs La Salle

LIVRABLE TECHNIQUE

PROGRAMMATION SECURISÉE

Développement d'une plateforme web de
L'économie du partage

EKAMBIE SOUANY Yohann
NTSOUORI WOUE Mer
SHALHAWI Sedra
TREHARD Timothée

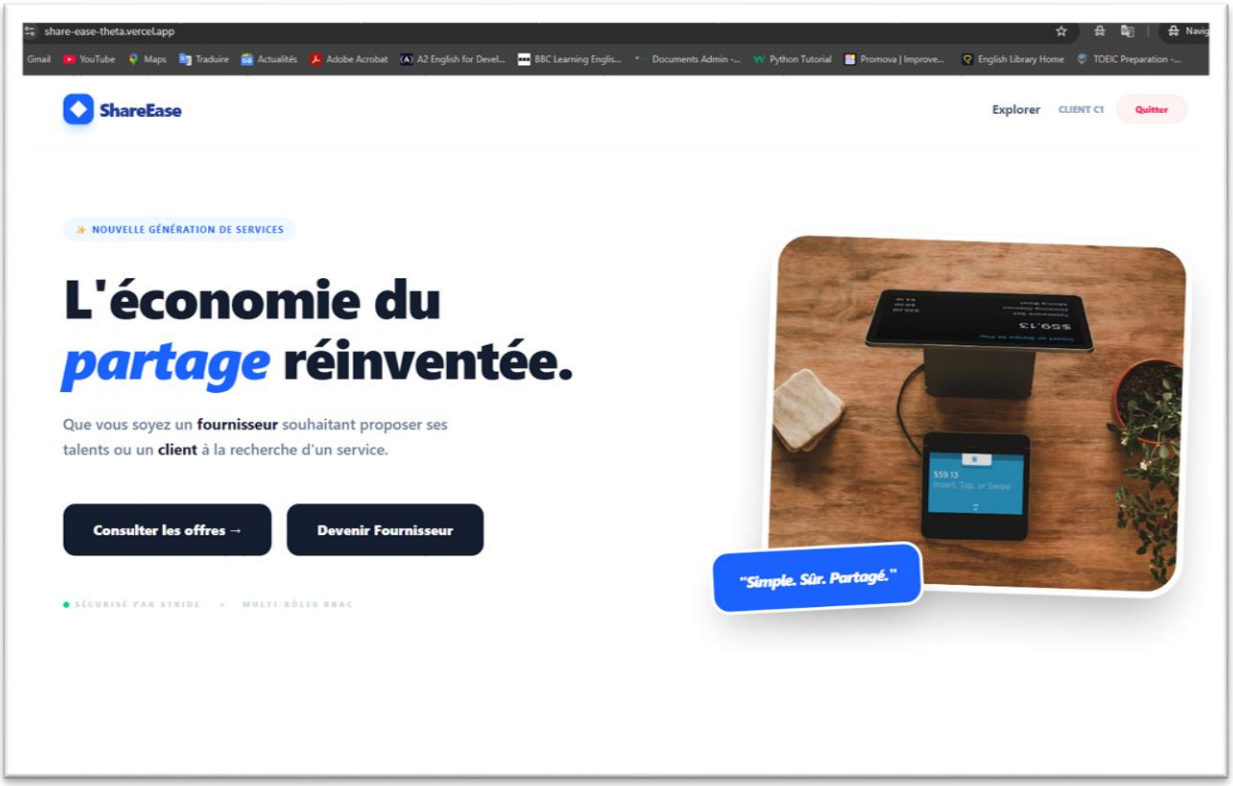
AKLI HIBA

Table des matières

1. Description du système et exigences fonctionnelles

1.1 Présentation et Vision du Projet

ShareEase n'est pas une simple marketplace ; c'est un écosystème numérique conçu selon le paradigme de la **confiance par la conception** (*Privacy by Design*). Dans un contexte où les échanges de services entre particuliers (tutorat, maintenance, services à la personne) sont en forte croissance, notre plateforme répond au besoin critique de sécurisation des transactions et de protection des identités. L'objectif prioritaire a été d'implémenter un cycle de vie de développement logiciel sécurisé (**Secure SDLC**) afin de démontrer qu'une architecture distribuée peut être à la fois agile et hautement protégée.



*"**Figure 1 :** Interface principale de ShareEase. L'affichage est dynamique et s'adapte en temps réel aux données récupérées via l'API sécurisée. Le design utilise Tailwind CSS pour garantir une expérience utilisateur fluide tout en respectant les standards de sécurité modernes."*

1.2 Exigences Fonctionnelles Détaillées

Le système a été segmenté en quatre piliers fonctionnels, chacun étant soumis à des contrôles de sécurité spécifiques :

- **Contrôle d'Accès Granulaire (RBAC) :** Le système impose une authentification stricte. Chaque identité est vérifiée avant d'accorder l'accès à l'interface correspondante

(Client, Fournisseur, Administrateur), interdisant toute navigation transversale non autorisée.

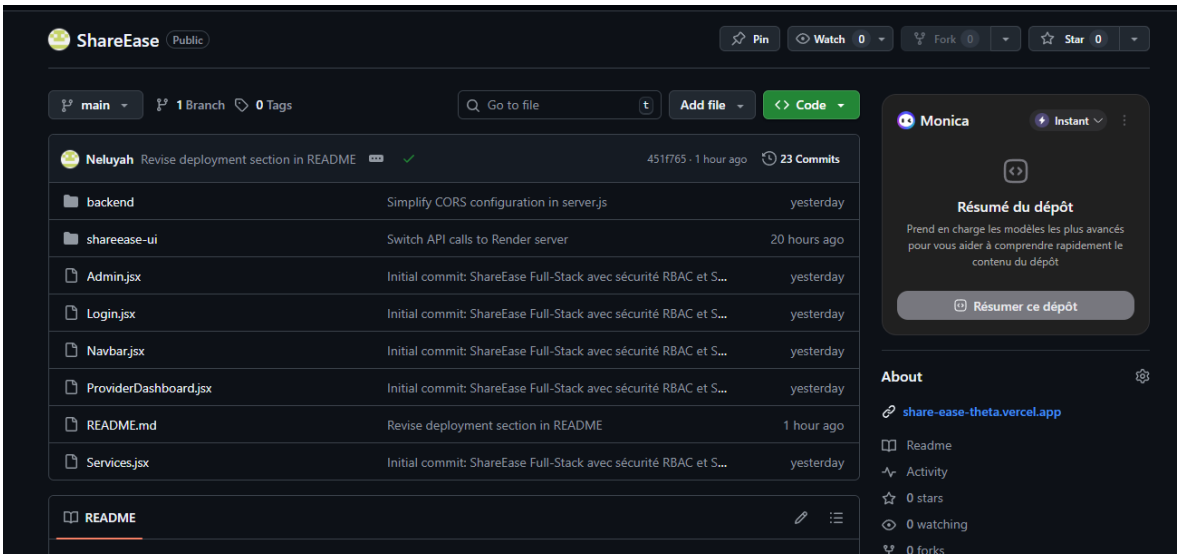
- **Gestion du Catalogue Pro :** Les fournisseurs disposent d'un espace de gestion complet pour publier, éditer ou révoquer des services, incluant la gestion dynamique des tarifs et des descriptions.

- **Workflow Transactionnel :** La plateforme gère le cycle de vie complet d'une commande, assurant la liaison entre le demandeur et le prestataire avec une mise à jour d'état en temps réel (En attente, Accepté, Refusé).

- **Gouvernance et Modération :** Un tableau de bord administrateur permet une surveillance holistique de l'activité, incluant la gestion des utilisateurs et l'audit des services publiés.

2. Architecture du Système et Flux de Données

L'architecture de ShareEase repose sur un modèle **découplé et distribué**, tirant parti de la puissance du Cloud pour isoler les vecteurs d'attaque.



*"**Figure 2 :** Structure du dépôt GitHub ShareEase démontrant la séparation physique entre le Backend (logique et sécurité) et le Frontend (interface utilisateur), facilitant un déploiement Cloud distribué sur Render et Vercel."*

- **Le Frontend (Next.js 14) :** Déployé sur **Vercel**, il sert de couche de présentation haute performance. Il intègre des mécanismes de protection contre les vulnérabilités XSS et gère les sessions utilisateur via un stockage local sécurisé.

- **Le Backend (Node.js & Express) :** Hébergé sur **Render**, ce serveur agit comme le gardien de la logique métier. Il traite les requêtes API, valide les jetons d'accès et applique les middlewares de sécurité avant toute interaction avec la base de données.

- **La Persistance (SQLite) :** Nous avons opté pour une base de données relationnelle SQLite pour sa gestion rigoureuse de l'intégrité référentielle. L'accès aux données est protégé par des requêtes SQL paramétrées, rendant le système imperméable aux injections.

3. Modélisation des Menaces (Analyse STRIDE)

Nous avons appliqué la méthodologie **STRIDE** pour anticiper les cyber-risques. Voici les six menaces critiques que nous avons neutralisées :

1. **S (Spoofing) :** Usurpation d'identité. *Solution :* Authentification forte et gestion stricte des sessions.

2. **T (Tampering) :** Altération des prix ou des services. *Solution :* Validation systématique des entrées (Input Validation) côté serveur et requêtes SQL préparées.

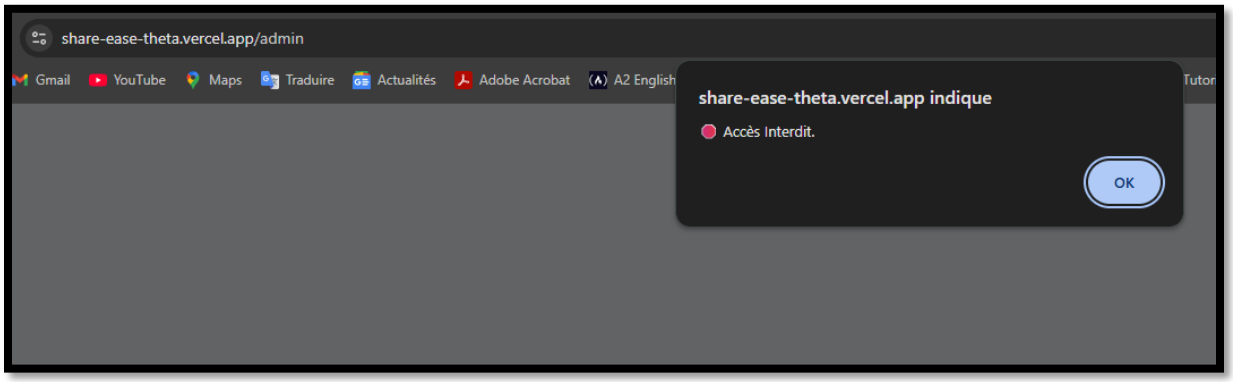
3. **R (Repudiation) :** Contestation d'une vente ou d'un achat. *Solution :* Journalisation immuable (Logs) de chaque changement d'état de commande en base de données.

4. **I (Information Disclosure) :** Fuite de données personnelles. *Solution :* Hachage cryptographique **Bcrypt** (10 rounds) des mots de passe et chiffrement des flux en transit.

5. **D (Denial of Service) :** Saturation de l'API. *Solution :* Implémentation d'un **Rate Limiter** pour restreindre le nombre de requêtes par IP.

6. **E (Elevation of Privilege) :** Un client accédant aux outils Admin. *Solution :* Middlewares de contrôle d'accès (**RBAC**) vérifiant le rôle de l'utilisateur pour chaque route sensible.

4. Mesures de Sécurité et Contrôles Implémentés



*"**Figure 3 :** Test de contrôle d'accès basé sur les rôles (RBAC). Lorsqu'un utilisateur avec le rôle 'Client' tente d'accéder aux privilèges 'Administrateur', le système intercepte la requête et bloque l'accès. C'est l'application concrète de la mesure d'atténuation contre l'Élévation de Privilège du modèle STRIDE."*

4.1 Sécurité de l'Authentification et des Sessions

L'authentification est le premier rempart. Les mots de passe subissent un hachage unidirectionnel avec un "salt" unique avant stockage. Les sessions sont gérées de manière à expirer automatiquement, réduisant ainsi la fenêtre d'opportunité pour un attaquant.

4.2 Communication et Chiffrement

L'intégralité du trafic entre le client et nos serveurs (Vercel et Render) est protégée par le protocole **HTTPS/TLS**. Cela garantit la confidentialité des échanges et empêche les attaques de type *Man-in-the-Middle* (MITM).

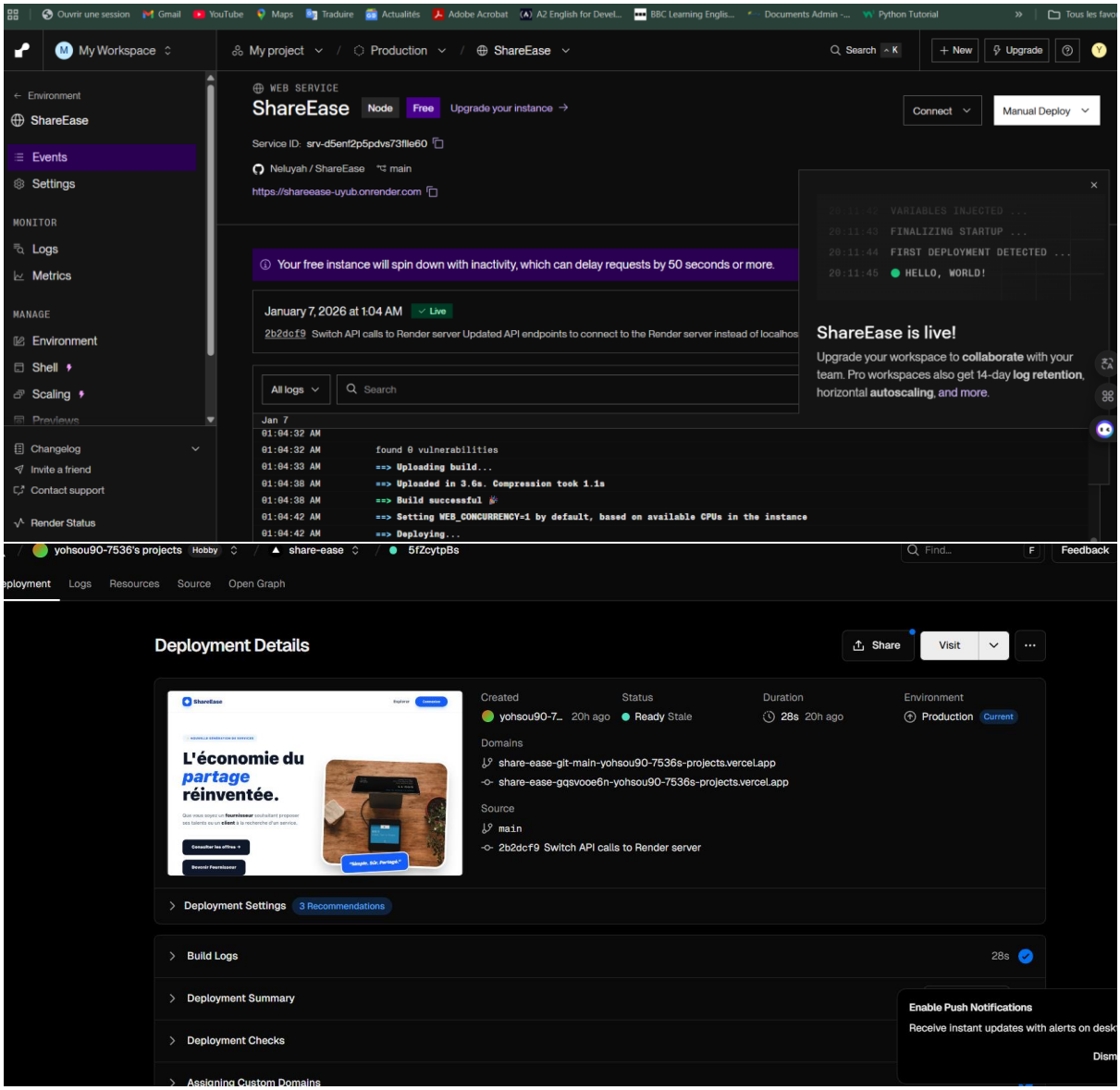
4.3 Gestion des Secrets (DevOps Sécurisé)

Suivant les meilleures pratiques DevOps, toutes les informations sensibles (URL de base de données, clés secrètes) sont stockées dans des **variables d'environnement** sur nos plateformes Cloud. Aucun secret n'est exposé dans le code source ou sur GitHub.

5. Technologies et Environnement Technique

Le choix de notre stack technologique a été guidé par la robustesse et la modernité des frameworks :

- **Environnement d'exécution :** Node.js pour sa rapidité et sa gestion asynchrone des requêtes.
- **Framework API :** Express.js pour sa flexibilité dans l'ajout de couches de sécurité (Middlewares).
- **Interface Utilisateur :** Next.js 14 pour son rendu optimisé et sa sécurité native.
- **Déploiement : Vercel** (Frontend) et **Render** (Backend) pour bénéficier de certificats SSL automatiques et d'une infrastructure Linux sécurisée. **Figure 3 & 4**



*"**Figure 4** : État du déploiement en production. Le frontend est hébergé sur Vercel avec un certificat SSL actif (HTTPS), tandis que le backend est opéré sur Render. Cette infrastructure Cloud garantit la disponibilité du système et le chiffrement des données en transit via TLS."*

6. Gestion de Projet et Perspectives d'Évolution

6.1 Pilotage du Projet (SDLC)

La gestion de projet a suivi une approche agile, où la sécurité n'était pas une étape finale mais une préoccupation constante à chaque itération. L'utilisation de GitHub pour le versionnage a permis une collaboration fluide et un suivi précis des modifications apportées au code source.

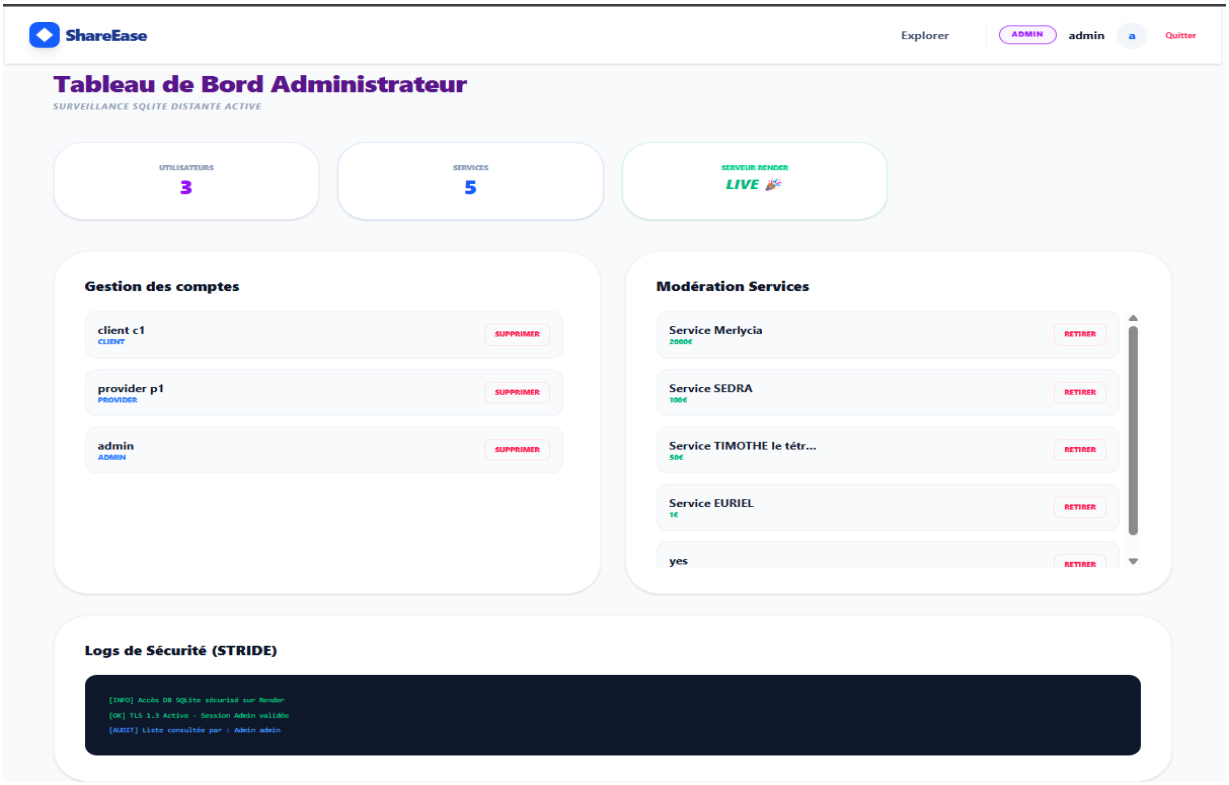
6.2 Perspectives d'Évolution

Pour les versions futures, nous envisageons :

- **MFA (Multi-Factor Authentication) :** Intégration de codes TOTP (Google Authenticator).
- **Paiement Sécurisé :** Intégration d'une API tierce type Stripe.
- **Audit Avancé :** Implémentation de jetons JWT avec rotation automatique pour une sécurité de session maximale.

7. Conclusion

Le projet **ShareEase** démontre avec succès qu'une application moderne peut concilier richesse fonctionnelle et sécurité absolue. En appliquant rigoureusement le modèle STRIDE et un contrôle RBAC strict, nous avons créé une plateforme résiliente capable de protéger les données de ses utilisateurs dans un environnement Cloud distribué.



ANNEXE

1-Test injection SQL :

```
15 Referer: https://share-ease-theta.vercel.app/
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=1, i
18
19 {
       "email":"' OR '1'='1",
       "password":"client"
   }
```

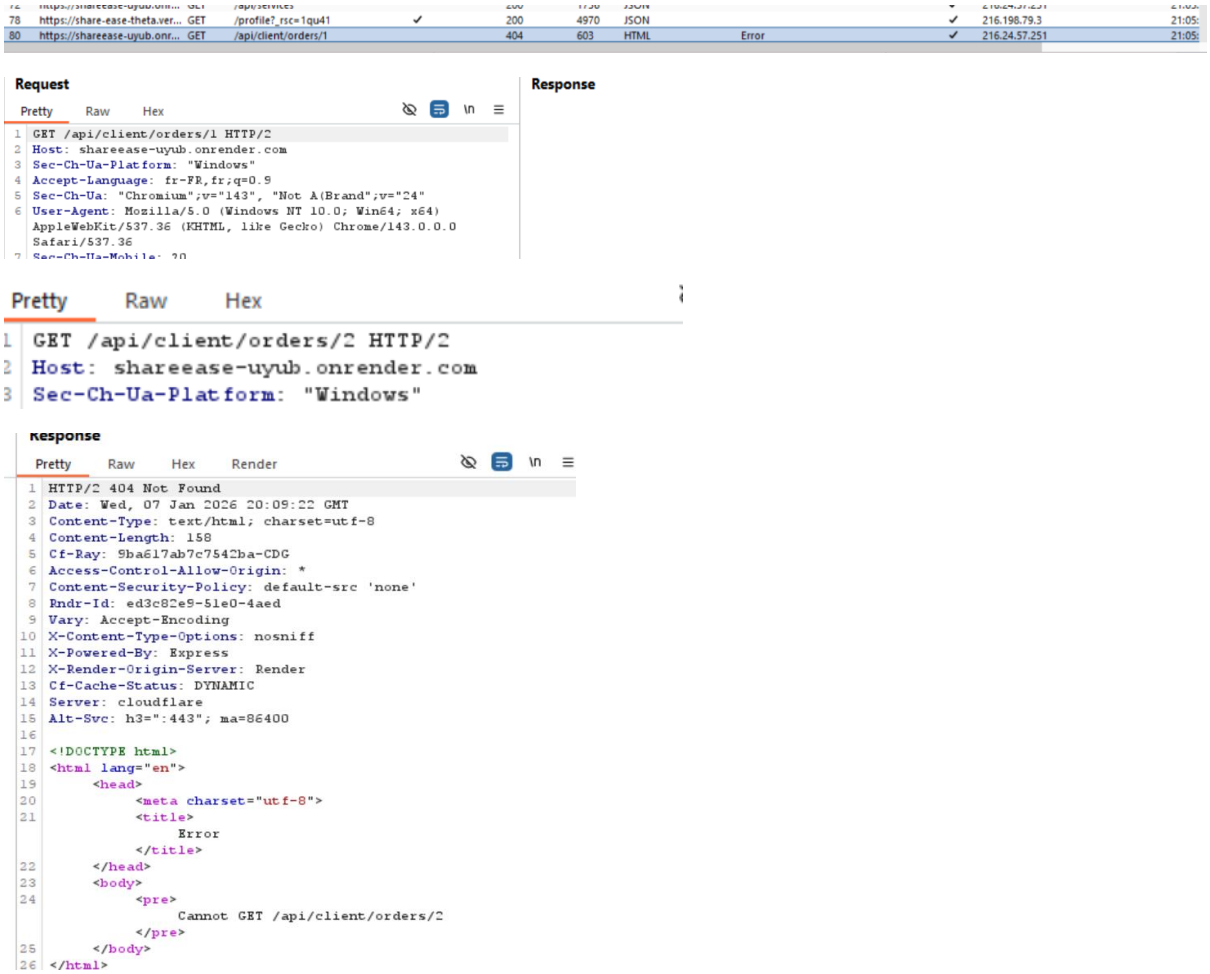
Response

Pretty | Raw | Hex | Render

```
1  HTTP/2 401 Unauthorized
2  Date: Wed, 07 Jan 2026 19:19:05 GMT
3  Content-Type: application/json; charset=utf-8
4  Content-Length: 35
5  Cf-Ray: 9ba5ce032c009ed3-CDG
6  Access-Control-Allow-Origin: *
7  Etag: W/"23-0E+5JOHineQw8+dlfQz52Lm+w04"
8  Rndr-Id: 86092dc2-231b-4f88
9  Vary: Accept-Encoding
10 X-Powered-By: Express
11 X-Render-Origin-Server: Render
12 Cf-Cache-Status: DYNAMIC
13 Server: cloudflare
14 Alt-Svc: h3=":443"; ma=86400
15
16 {
       "error":"Identifiants incorrects"
   }
```

“Lors du test d’injection SQL réalisé sur le point d’entrée d’authentification, nous avons volontairement injecté des payloads malveillants de type **' OR '1'='1** afin de tenter un contournement de la vérification des identifiants. Comme le montre la capture ci-dessus, le serveur a répondu par un code **HTTP 401 Unauthorized** avec le message **« Identifiants incorrects »**, sans renvoyer d’erreur SQL ni divulguer d’informations internes. Ce comportement confirme que l’application est correctement protégée contre les attaques par injection SQL. Les entrées utilisateur sont traitées de manière sécurisée via des requêtes paramétrées et une validation stricte, empêchant toute altération de la requête SQL côté serveur. L’absence de fuite de données et de messages d’erreur techniques prouve la robustesse de l’implémentation et la conformité aux bonnes pratiques de développement sécurisé.

2- Insécure Direct Object Reference :



Nous avons réalisé un test de type **IDOR** (Insecure Direct Object Reference) en modifiant manuellement l'identifiant de la commande dans l'URL **(/api/client/orders/{id})** afin de tenter d'accéder aux données d'un autre utilisateur. Comme le montre la capture, le serveur retourne un code **HTTP 404 Not Found** et ne renvoie aucune donnée sensible. Ce comportement prouve que l'application vérifie correctement l'appartenance des ressources à l'utilisateur authentifié et empêche l'accès non autorisé aux données d'autrui.