



Secure Notes

**Analyse des vulnérabilités OWASP
Top 10 - PoC & Corrections**

Présentation du projet

Le projet Secure Notes est une application web composée de :

- un backend (Node.js, Express, MariaDB),
- un frontend simple (HTML / JavaScript),
- dont l'objectif pédagogique est de mettre en évidence les principales failles de sécurité OWASP Top 10, puis de les corriger.

L'approche adoptée est volontairement la suivante :

1. Implémentation intentionnelle de failles de sécurité,
2. Réalisation de preuves de concept (PoC),
3. Application de correctifs,
4. Validation du comportement sécurisé après correctio

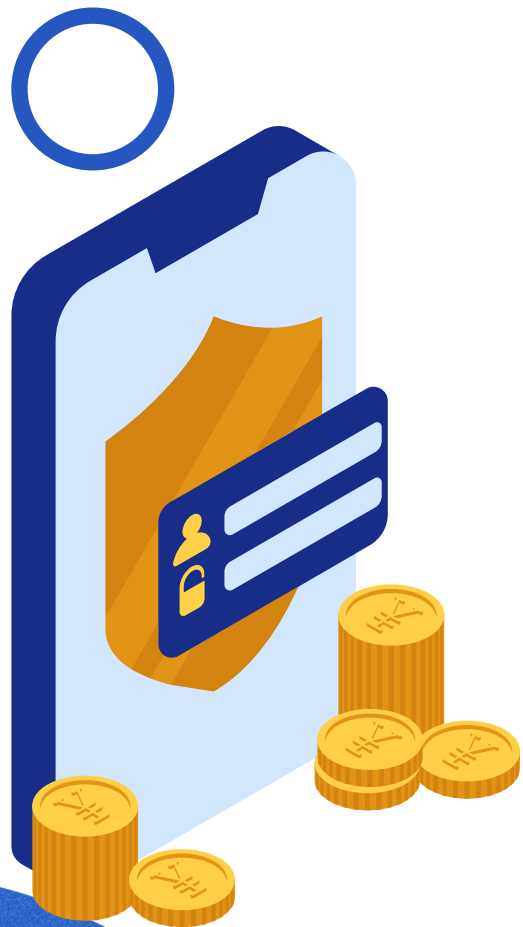


Le projet Secure Notes est une application web composée de :

- un backend (Node.js, Express, MariaDB),
- un frontend simple (HTML / JavaScript),
- dont l'objectif pédagogique est de mettre en évidence les principales failles de sécurité OWASP Top 10, puis de les corriger.

L'approche adoptée est volontairement la suivante :

1. Implémentation intentionnelle de failles de sécurité,
 2. Réalisation de preuves de concept (PoC),
 3. Application de correctifs,
 4. Validation du comportement sécurisé après correction.
-



Objectif fonctionnel de l'application

Secure Notes est une application web permettant aux utilisateurs de :

- Créer un compte
- S'authentifier
- Créer et consulter des notes personnelles

Acteurs du système

L'application distingue deux types d'acteurs :

Utilisateur (User)

- S'inscrire (register)
- Se connecter (login)
- Créer des notes
- Consulter ses propres notes
- (Phase vulnérable : accès à des ressources non autorisées)

Administrateur (Admin)

- Accéder aux fonctionnalités d'administration
- Consulter la liste des utilisateurs
- Accéder aux statistiques globales

Dans la première version, cette distinction est volontairement mal implémentée pour illustrer :
A01 – Broken Access Control et A04 – Insecure Design

Fonctionnalités principales

Authentification :

- Inscription avec email / mot de passe
- Hashage des mots de passe avec bcrypt
- Authentification par JWT

Gestion des notes

- Création de notes
- Consultation des notes
- Association des notes à un propriétaire (owner_id)

Administration

- Consultation de données sensibles
- Accès réservé à l'administrateur (en théorie)

Règles de gestion de l'application

Règle :	Description :
RG1	Authentification obligatoire pour accéder aux ressources
RG2	Un utilisateur ne peut consulter que ses propres notes
RG3	Les fonctionnalités admin sont réservées aux admins
RG4	Les mots de passe ne sont jamais stockés en clair
RG5	Les communications doivent être sécurisées via HTTPS

Certaines règles sont volontairement violées dans la première version pour démontrer les vulnérabilités OWASP.

Architecture technique retenue

Secure Notes repose sur une architecture 3-tiers :

```
[ Frontend ] ⇄ [ API Backend ] ⇄ [ Base de données ]
```

Ce choix permet :

- Une séparation claire des responsabilités
- Une meilleure maintenabilité

Une analyse précise des failles à chaque niveau

Couche Frontend (client Web)

Technologies :

- HTML
- JavaScript (vanilla)

Rôle :

- Interface utilisateur
- Envoi des requêtes HTTP vers l'API
- Affichage des notes

Utilisé volontairement pour démontrer une faille XSS (innerHTML)

Couche Backend (API REST)

Technologies :

- Node.js
- Express.js

Rôle :

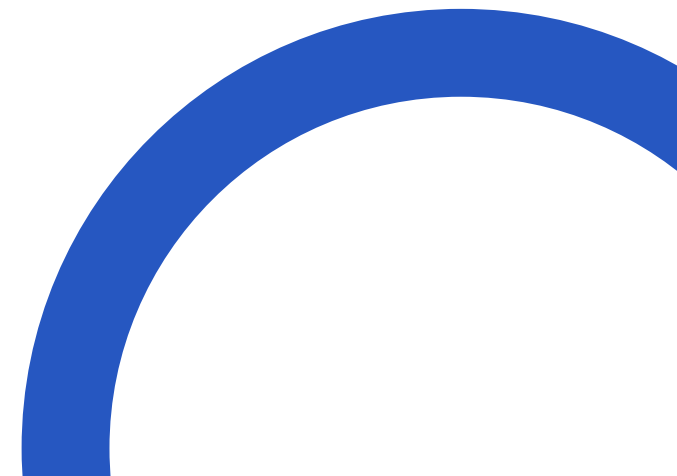
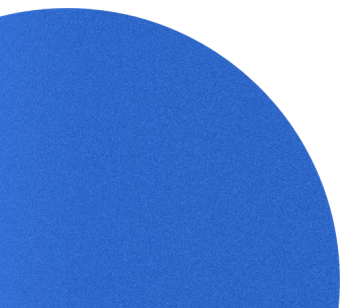
- Logique métier
- Authentification
- Contrôle d'accès
- Communication avec la base de données

Rôle de HTTPS dans l'architecture

L'utilisation de HTTPS garantit :

- La confidentialité des identifiants
- La protection du JWT (Authorization: Bearer)
- L'intégrité des données échangées

Sans HTTPS, l'authentification serait facilement compromise.



Sécurité intégrée transversalement

Backend

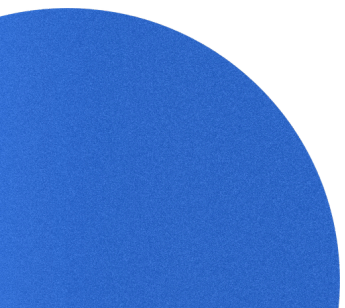
- Hashage bcrypt
- JWT
- Middlewares de contrôle d'accès
- Gestion centralisée des erreurs

Frontend

- Tests XSS
- Affichage contrôlé après correction

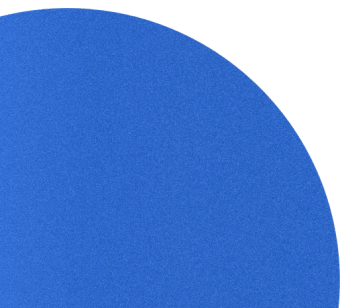
Base de données

- Requêtes préparées
- Aucune donnée sensible en clair



Justification des choix techniques

Choix :	Justification :
Node.js / Express	Léger, idéal pour API REST
MariaDB	Relationnel, adapté à la démonstration SQLi
JWT	Authentification stateless
Architecture 3-tiers	Séparation claire, exigence du module
HTTPS	Sécurisation des échanges
Frontend simple	Focus sur la sécurité, pas le design



Limites volontaires (phase vulnérable)

Dans la première version :

- Pas de rate limiting
- Pas de logs avancés
- Contrôles d'accès incomplets
- Secrets codés en dur

Ces limites sont intentionnelles et servent de support pédagogique pour l'analyse OWASP.

A01 – Broken Access Control (IDOR)

Une vulnérabilité IDOR (Insecure Direct Object Reference) permet à un utilisateur d'accéder à des ressources qui ne lui appartiennent pas, simplement en modifiant un identifiant dans la requête.

📍 Localisation

backend/src/routes/notes.js

```
router.get("/", async (req, res) => {
  const { owner_id } = req.query;

  const [rows] = await db.query(
    "SELECT id, title, content, owner_id FROM notes WHERE owner_id = ?",
    [owner_id]
  );

  res.json(rows);
});
```


Preuve de concept (PoC)

The screenshot displays the Thunder Client interface with a new request configured for a POST to `http://localhost:4000/auth/register`. The request body is a JSON object: `{ "email": "user1@test.com", "password": "Test1234!" }`. The response is a 201 Created status with a JSON body: `{ "message": "registered" }`. The bottom terminal shows the server running with Nodemon, indicating a restart due to changes.

Thunder Client Interface:

- Activity Panel:** Lists recent requests, including GET requests to `localhost:4000/health`, `localhost:4000/boom`, `localhost:4000/debug...`, and `localhost:4000/notes`, as well as the current POST request to `localhost:4000/auth...`.
- Request Editor:** Shows the POST request to `http://localhost:4000/auth/register` with a JSON body.
- Response Panel:** Displays the response status (201 Created), size (24 Bytes), and time (98 ms). The response body is `{ "message": "registered" }`.
- Terminal:** Shows the command `npm run dev` being executed, with output indicating the server is running on `http://localhost:4000` and restarting due to changes.

New Request

ActivityCollectionsEnv

filter activity

GETlocalhost:4000/health8 mins ago

GETlocalhost:4000/boom14 mins ago

GETlocalhost:4000/debug...17 mins ago

GETlocalhost:4000/notes10 hours ago

POSTlocalhost:4000/auth...just now

POSTlocalhost:4000/auth...10 hours ago

POSThttp://localhost:4000/auth/registerSend

QueryHeaders2AuthBody1TestsPre Run

JSONXMLTextFormForm-encodeGraphQLBinary

JSON ContentFormat

1 {
2 "email": "user2@test.com",
3 "password": "Test1234!"
4 }

Status: 201 CreatedSize: 24 BytesTime: 59 ms

ResponseHeaders8CookiesResultsDocs

1 {
2 "message": "registered"
3 }

PROBLEMSOUTPUTDEBUG CONSOLETERMINALPORTS

PS C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend> npx nodemon server.js

API: http://localhost:4000

Error: Boom! Debug stack exposed
at C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\server.js:40:9
at Layer.handleRequest (C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\lib\layer.js:152:17)
at next (C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\lib\route.js:157:13)
at Route.dispatch (C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\lib\route.js:117:3)
at handle (C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\index.js:435:11)
at Layer.handleRequest (C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\lib\layer.js:152:17)
at C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\index.js:295:15
at processParams (C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\index.js:582:12)
at next (C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\index.js:291:5)
at Layer.handleRequest (C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\backend\node_modules\router\lib\layer.js:147:12)

[nodemon] restarting due to changes...
[nodemon] starting `node server.js`

API: http://localhost:4000

[nodemon] restarting due to changes...
[nodemon] starting `node server.js`

A01 – Broken Access Control (IDOR)

Une vulnérabilité IDOR (Insecure Direct Object Reference) permet à un utilisateur d'accéder à des ressources qui ne lui appartiennent pas, simplement en modifiant un identifiant dans la requête.

Code corrigé

```
router.get("/", requireAuth, async (req, res) => {
  const userId = req.user.id;

  const [rows] = await db.query(
    "SELECT id, title, content FROM notes WHERE owner_id = ?",
    [userId]
  );

  res.json(rows);
});
```

Résultat

- Un utilisateur ne peut accéder qu'à ses propres notes
- IDOR supprimé

OWASP A02 – Security Misconfiguration

Description de la faille

Une mauvaise configuration du serveur peut exposer :

- des informations sensibles,
- des stack traces,
- des chemins internes.

📍 Localisation

✗ Code vulnérable

- backend/server.js

```
app.use(cors({ origin: "*", credentials: true }));

app.get("/boom", (req, res) => {
  throw new Error("Boom! Debug stack exposed");
});

app.use((err, req, res, next) => {
  res.status(500).json({
    message: err.message,
    stack: err.stack
  });
});
```

Code corrigé

```
app.use(cors({
  origin: "http://localhost:5173",
  credentials: true
}));

app.use((err, req, res, next) => {
  res.status(500).json({ message: "Internal server error" });
});
```

Résultat

- CORS restrictif
- Suppression de l'endpoint /boom
- Handler d'erreur générique

```
95 | <div class="row">
96 |   <h2>Admin</h2>
97 |   <button id="btnLoadUsers" class="btn btn--ghost">Load users</button>
98 | </div>
99 | <div id="adminUsers" class="list"></div>
100 | </section>
101 |
102 | <footer class="footer">
103 |   <small class="muted">Projet Sécurité Web – Secure Notes</small>
104 | </footer>
105 | </main>
106 |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\micro\OneDrive\Desktop\Bureau\secure-notes\frontend> npx serve .
Need to install the following packages:
serve@14.2.5
Ok to proceed? (y) y

Serving!

- Local: http://localhost:3000
- Network: http://192.168.1.20:3000

Copied local address to clipboard!

OWASP A03 – Injection (SQL Injection)

Description de la faille

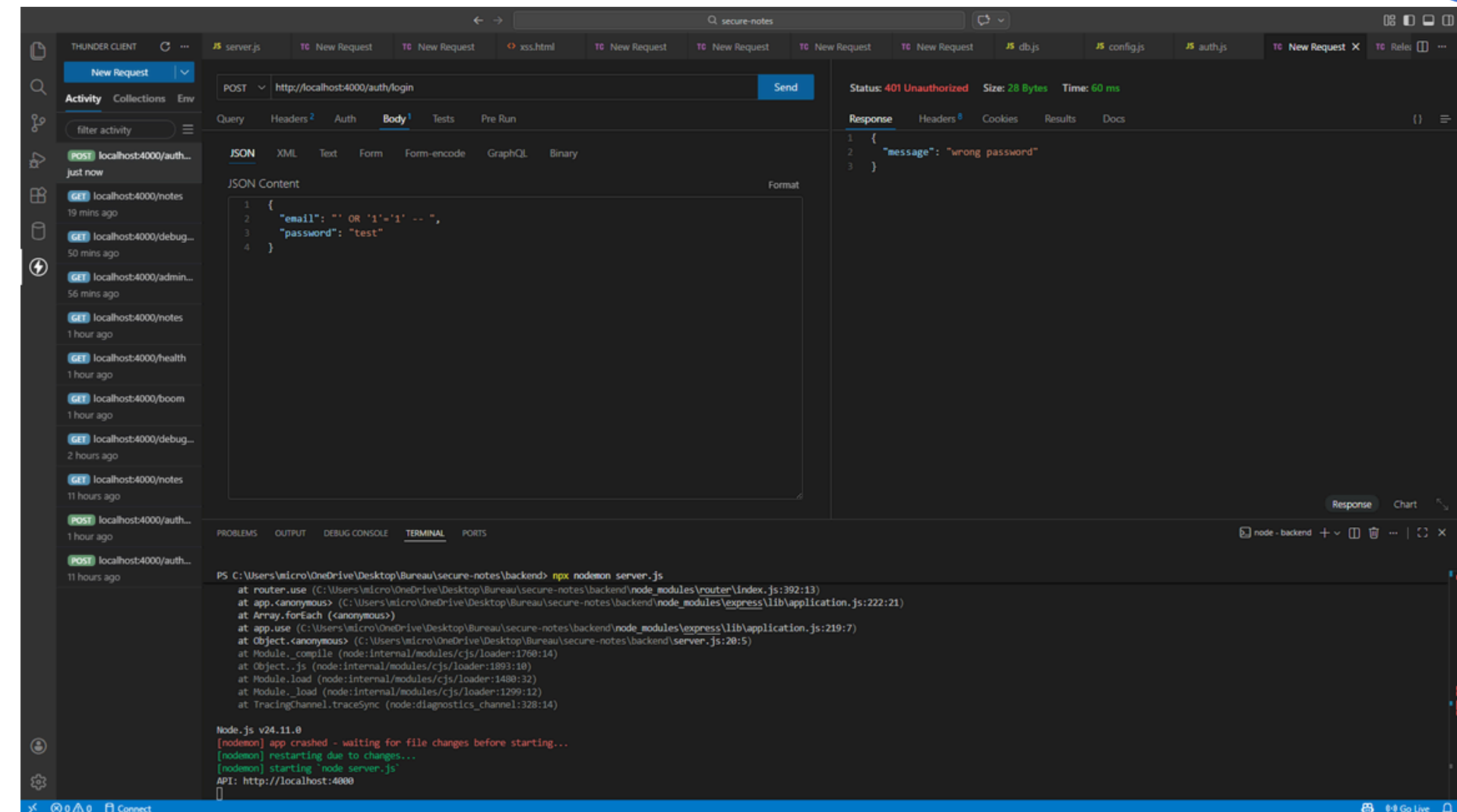
- Une injection SQL permet de modifier une requête SQL via une entrée utilisateur non sécurisée.

Localisation

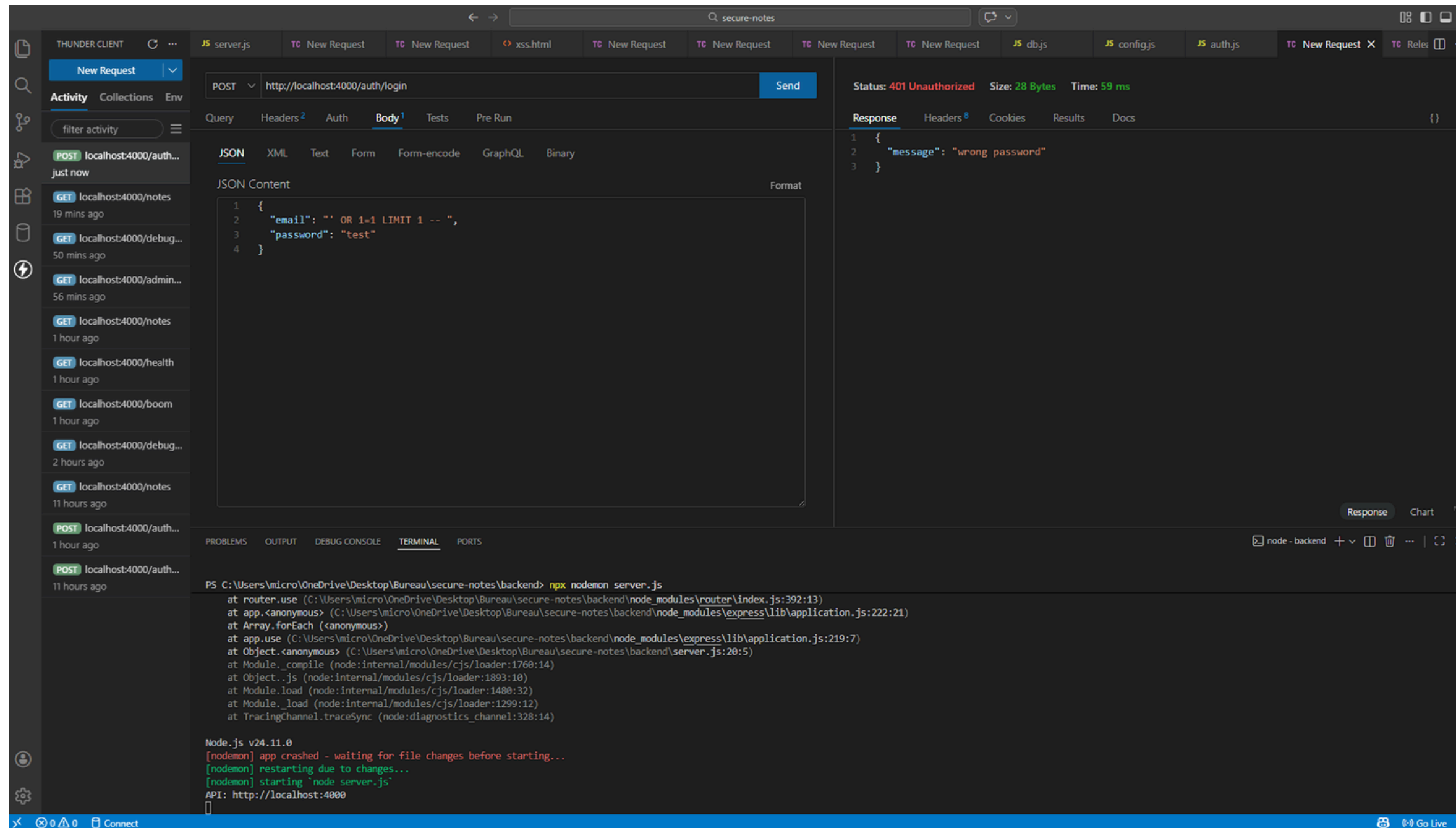
backend/src/routes/auth.js

✗ Code vulnérable

```
const sql =  
  "SELECT id, email, password_hash FROM users WHERE email = '" + email + "'";  
  
const [rows] = await db.query(sql);
```



Preuve de concept (PoC)



Résultat :

- Erreur SQL
- Stack trace exposée
- Requête manipulée

✓ Correction appliquée

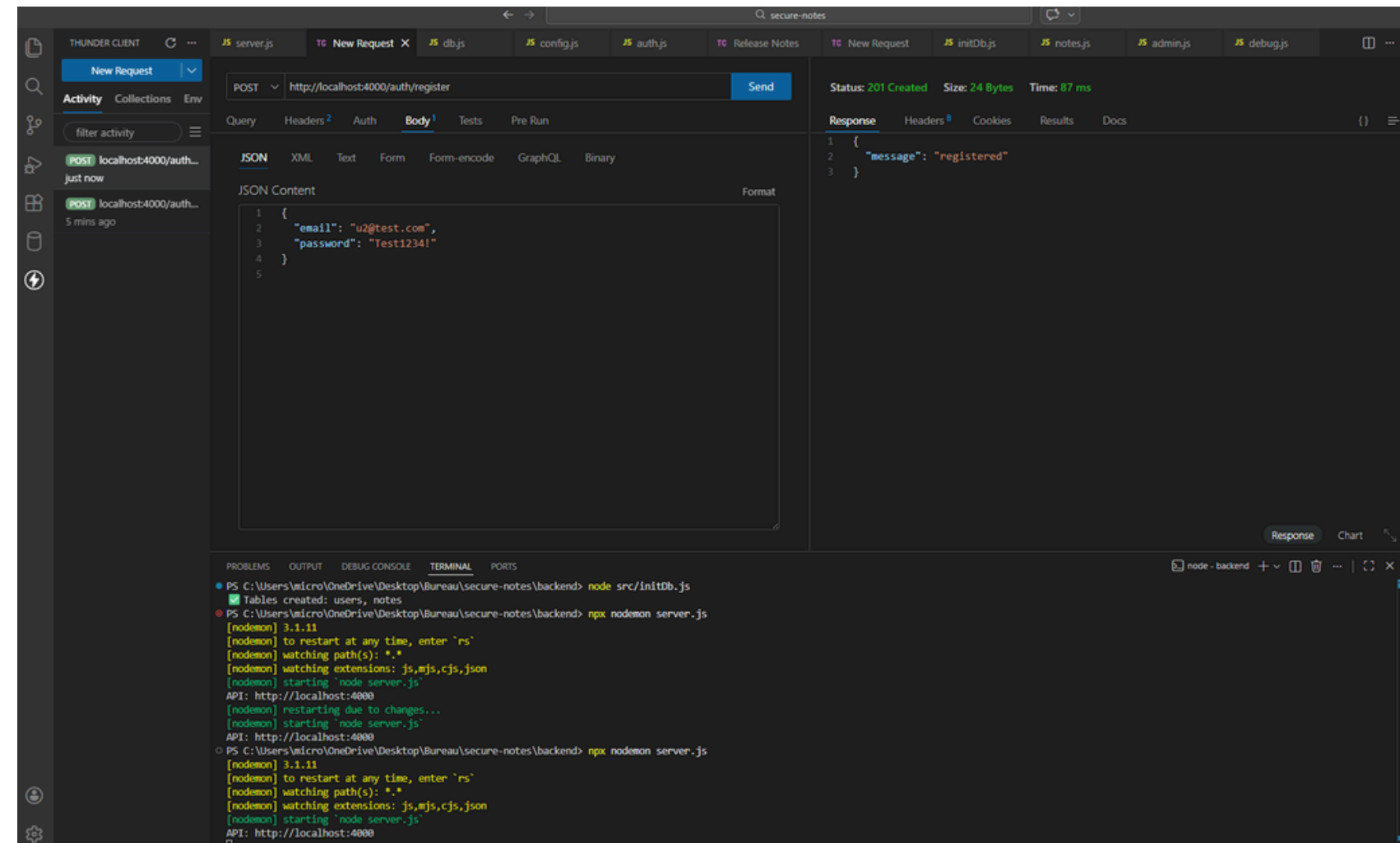
Utilisation de requêtes préparées

✓ Code corrigé

```
const [rows] = await db.query(  
  "SELECT id, email, password_hash FROM users WHERE email = ?",  
  [email]  
);
```

● Résultat

- Injection impossible
- Requête sécurisée



✓ Corrections appliquées

- Message unique
- JWT durée réduite
- Rate limiting (optionnel)

```
return res.status(401).json({ message: "Invalid credentials" });
```

OWASP A04 - Insecure Design

● Description

Absence ou contournement de règles de contrôle d'accès.

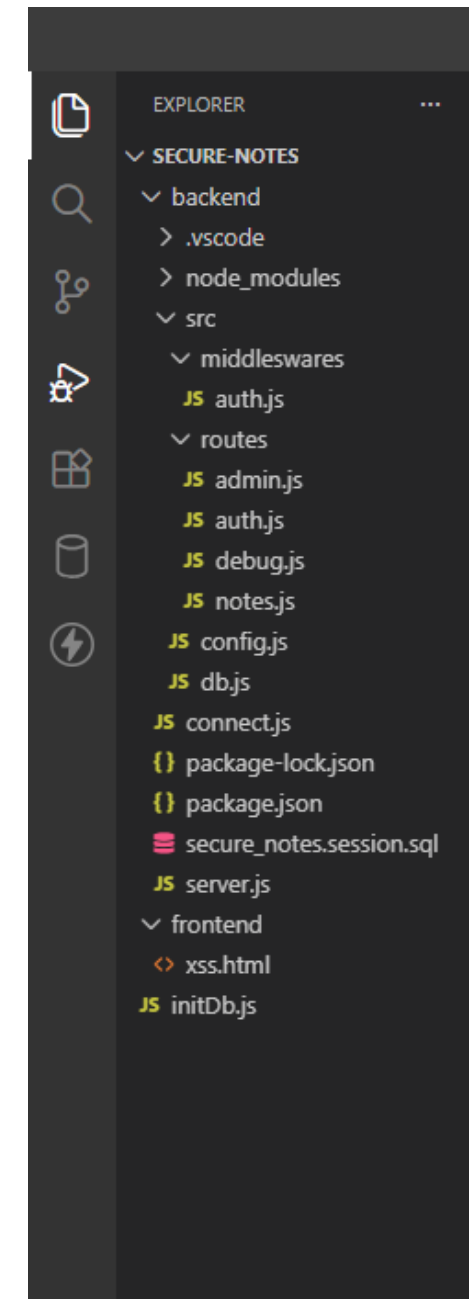
✗ Code vulnérable

```
if (OWASP.A04_INSECURE_DESIGN) return next();
```

➡ Accès admin possible sans rôle admin.

```
if (req.user.role !== "admin") {  
  return res.status(403).json({ message: "Admin only" });  
}
```

✗ structure propre



OWASP A05 - Cryptographic Failures

● Problème

Secret JWT codé en dur.

```
const JWT_SECRET = "DEV_SECRET_IN_CODE";
```

✓ Correction

Utilisation de variables d'environnement.

```
const JWT_SECRET = process.env.JWT_SECRET;
```

OWASP A06 - Outdated Components

Risques

- Des dépendances obsolètes peuvent contenir des vulnérabilités connues (RCE, XSS indirect, etc.)
- Impact : compromission du serveur / fuite de données

