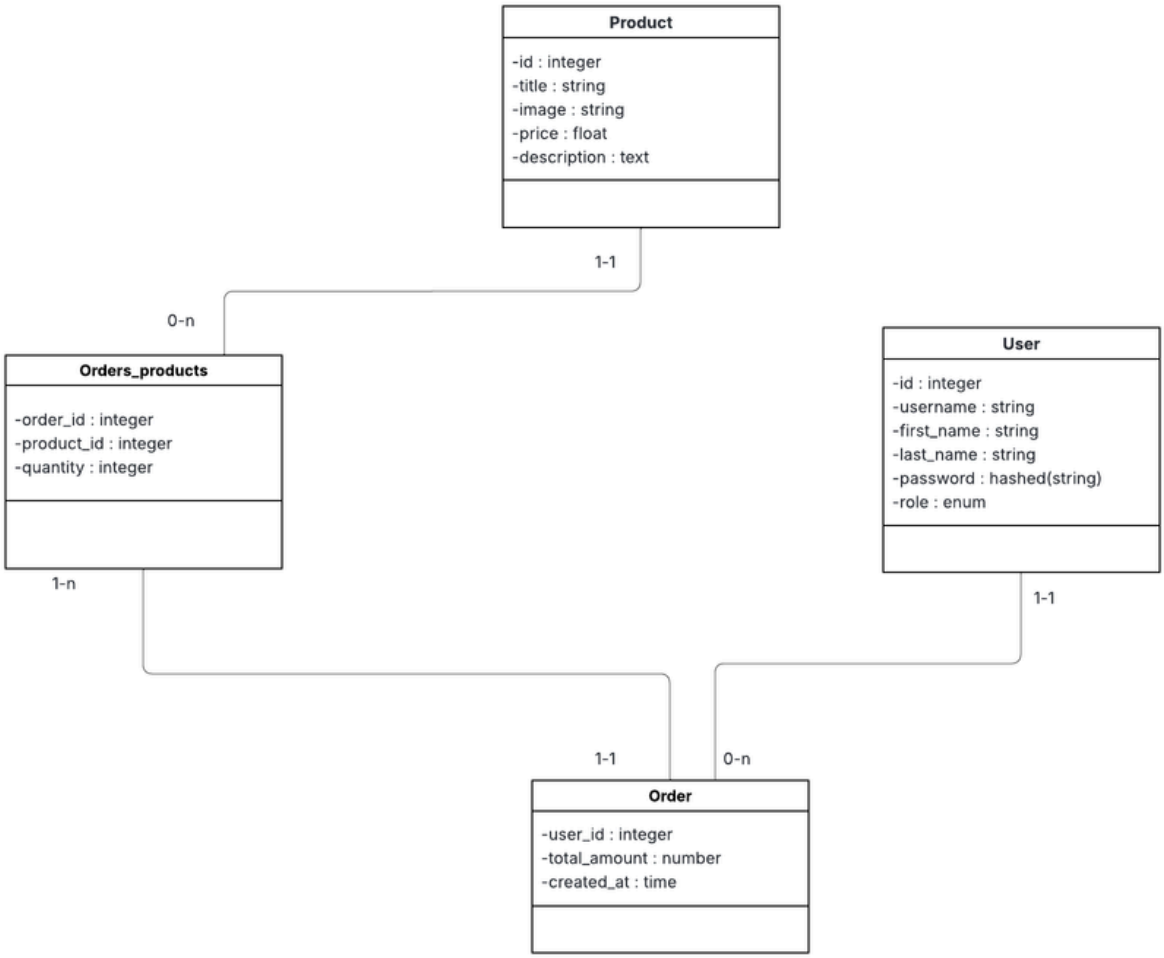
Rapport : Sécurité Web

1- Le diagramme de la base de données :



2- Les contre-mesures aux vulnérabilités OWASP :

1- Broken Access Control : utiliser des middlewares pour les routes qui nécessitent une authentification (un middleware pour authentification normale et un pour admin).

```
function authMiddleware(req, res, next) {
  const token = req.cookies.token;

  if (!token) return res.status(401).json({ message: 'Token manquant' });


  try {
    const decoded = verifyToken(token);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(403).json({ message: 'Token invalide' });
  }
}




function isAdmin(req, res, next) {
  if (req.user && req.user.role === 'ADMIN') {
    console.log(req.user.role)
    next();
  } else {
    res.status(403).json({ message: 'Accès refusé : rôle administrateur requis.' });
  }
}
```

2- Cryptographic Failures : utiliser la bibliothèque bcrypt pour hasher le mot de passe avant de le stocker !

```
  const hashed = await bcrypt.hash(password, 10);

  const user = await prisma.user.create({
    data: { username, firstname, lastname, password: hashed },
  });
```

3- Injection : valider les données utilisateurs pour l'authentification avec la bibliothèque express-validator,
Utiliser l'ORM Prisma pour les requêtes paramétrées afin d'éviter les attaques d'injection.

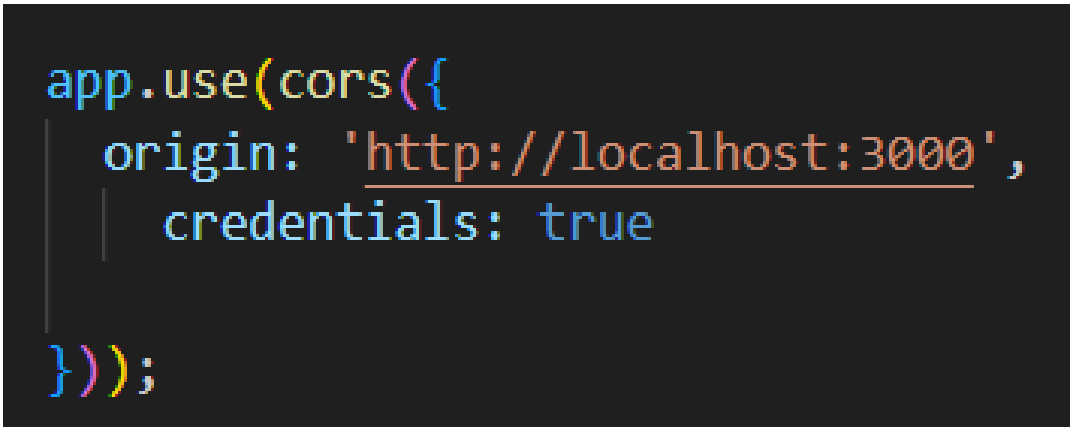
```
  body('lastname').notEmpty().withMessage('Nom requis'),
  body('password').isLength({ min: 8 }).withMessage('Mot de passe ≥ 8 caractères'),
```

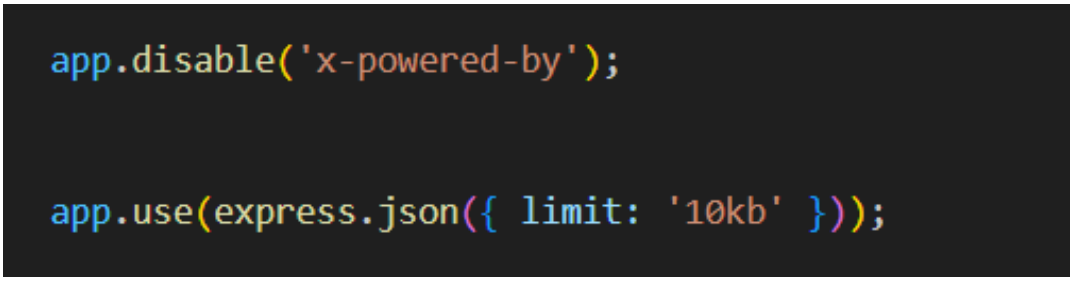
```
  const user = await prisma.user.create({
    data: { username, firstname, lastname, password: hashed },
  });
```

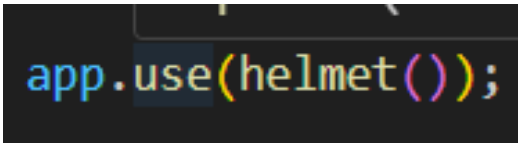
4- Insecure Design : faire une conception de base de données sécurisée, utiliser des user stories, utiliser le modèle Architecture Zero Trust (ne jamais faire confiance au client, tout valider côté serveur).

5- Security Misconfiguration :

- supprimer l'en-tête HTTP X-Powered-By avec app.disable('x-powered-by');
- ajouter des headers de sécurité automatiquement avec app.use(helmet());
- limiter la taille des données reçues avec app.use(express.json({ limit: '10kb' }));
- limiter le CORS (qui peut accéder à mon API)
- faire des messages d'erreur généraux sans donner de détails techniques à l'utilisateur !







6- Vulnerable and Outdated Components : détecter les vulnérabilités avec npm audit, corriger celles-ci avec npm audit fix, et mettre à jour automatiquement les packages avec npm update.

7- Identification and Authentication Failures :
Envoyer le token dans un cookie sécurisé avec l'attribut HttpOnly pour protéger le token d'authentification en le rendant inaccessible via JavaScript, ce qui renforce la sécurité contre les attaques XSS et CSRF.
limiter le nombre de requêtes des clients avec rateLimit.

```
const jwt = require('jsonwebtoken');

const JWT_SECRET = process.env.JWT_SECRET

function generateToken(user) {
  return jwt.sign(
    { userId: user.id, role: user.role },
    JWT_SECRET,
    { expiresIn: '1d' }
  );
}

function verifyToken(token) {
  return jwt.verify(token, JWT_SECRET);
}

module.exports = { generateToken, verifyToken};
```

```
res.cookie('token', token, {
  httpOnly: true,
  secure: process.env.NODE_ENV === 'production',
  sameSite: 'Strict',
  maxAge: 24 * 60 * 60 * 1000,
});
```

```
app.use(rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 50
}));
```

8- Software and Data Integrity Failures : Mettre en place un pipeline CI/CD avec GitHub Actions pour tester le backend et le déployer sur Render, en gérant les secrets et les variables d'environnement afin d'assurer la sécurité.

```yaml
name: CI + Deploy

on:
  push:

    branches: [ main ]

jobs:
  test-and-deploy:
    runs-on: ubuntu-latest

    env:
      JWT_SECRET: ${{ vars.JWT_SECRET }}
      DATABASE_URL: ${{ vars.DATABASE_URL }}

    steps:
      - name: Checkout le code
        uses: actions/checkout@v3

      - name: Installer les dépendances
        run: npm install

      - name: Lancer les tests
        run: npm test

      - name: Déploiement sur Render (si les tests passent)
        run: curl "${{ secrets.RENDER_DEPLOY_HOOK }}"
```

```
describe("Route protégée avec token dans cookie", () => {
  let token;

  beforeAll(() => {
    token = jwt.sign({ userId: 123 }, process.env.JWT_SECRET, { expiresIn: "1h" });
  });

  test("sans cookie token => 401", async () => {
    const res = await request(app).get("/products");
    expect(res.status).toBe(401);
    expect(res.body.message).toBe("Token manquant");
  });

  test("avec cookie token invalide => 401", async () => {
    const res = await request(app)
      .get("/products")
      .set("Cookie", ["token=fauxToken"]);
    expect(res.status).toBe(403);
    expect(res.body.message).toBe("Token invalide.");
  });

  test("avec cookie token valide => 200", async () => {
    const res = await request(app)
      .get("/products")
      .set("Cookie", [`token=${token}`]);
    expect(res.status).toBe(200);
  });
```

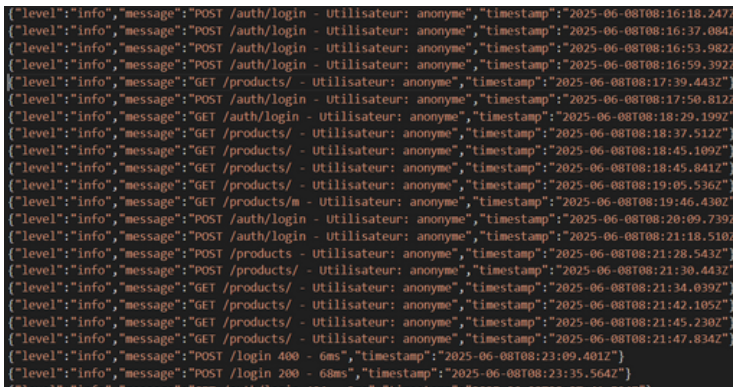
9- Security Logging and Monitoring Failures : Logger les requêtes HTTPS entrantes avec Winstonlogger.

```
const winston = require('winston');

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    new winston.transports.File({ filename: 'logs/combined.log' }),
  ],
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.simple(),
  }));
}

module.exports = logger;
```



```
{"level":"info","message":"POST /auth/login - Utilisateur: anonyme","timestamp":"2025-06-08T08:16:18.247
{"level":"info","message":"POST /auth/login - Utilisateur: anonyme","timestamp":"2025-06-08T08:16:37.084
{"level":"info","message":"POST /auth/login - Utilisateur: anonyme","timestamp":"2025-06-08T08:16:53.982
{"level":"info","message":"POST /auth/login - Utilisateur: anonyme","timestamp":"2025-06-08T08:16:59.392
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:17:39.443Z"
{"level":"info","message":"POST /auth/login - Utilisateur: anonyme","timestamp":"2025-06-08T08:17:50.812
{"level":"info","message":"GET /auth/login - Utilisateur: anonyme","timestamp":"2025-06-08T08:18:29.199Z
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:18:37.512Z"
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:18:45.109Z"
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:18:45.841Z"
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:19:05.536Z"
{"level":"info","message":"GET /products/m - Utilisateur: anonyme","timestamp":"2025-06-08T08:19:46.430Z"
{"level":"info","message":"POST /auth/login - Utilisateur: anonyme","timestamp":"2025-06-08T08:20:09.739
{"level":"info","message":"POST /auth/login - Utilisateur: anonyme","timestamp":"2025-06-08T08:21:18.510
{"level":"info","message":"POST /products - Utilisateur: anonyme","timestamp":"2025-06-08T08:21:28.543Z"
{"level":"info","message":"POST /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:21:30.443Z"
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:21:34.039Z"
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:21:42.105Z"
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:21:45.230Z"
{"level":"info","message":"GET /products/ - Utilisateur: anonyme","timestamp":"2025-06-08T08:21:47.834Z"
{"level":"info","message":"POST /login 400 - 6ms","timestamp":"2025-06-08T08:23:09.401Z"}
{"level":"info","message":"POST /login 200 - 68ms","timestamp":"2025-06-08T08:23:35.564Z"}
```

10- Server-Side Request Forgery (SSRF) : Aucune route ne doit effectuer de requête HTTP vers une URL contrôlée par l'utilisateur, afin de prévenir les attaques SSRF.