Déploiement de l’application Vite et Gourmand

1. Mise en place de l’environnement serveur

Pour le déploiement, j’ai utilisé une instance AWS EC2 sous Ubuntu 24.04 LTS.

Après connexion via SSH, la première étape a été la mise à jour du système :

```
sudo apt update
sudo apt upgrade -y
```

Cette étape est importante pour garantir que toutes les dépendances système soient à jour avant d’installer Docker.

Ensuite, j’ai installé Docker ainsi que Docker Compose via les dépôts officiels :

- sudo install -m 0755 -d /etc/apt/keyrings
- curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
- sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Après installation, le système indiquait qu’un redémarrage était recommandé suite à une mise à jour du kernel.
J’ai redémarré l’instance afin d’assurer la cohérence entre le kernel installé et celui en cours d’exécution.

Versions finale :

Ubuntu 24.04.4 LTS

Kernel 6.17.0-1007-aws

Cela garantit un environnement propre et stable.

2. Architecture du déploiement

L'application est déployée sous forme de services conteneurisés :

- Frontend : React + Vite + Nginx
- Backend : Node.js + Express
- PostgreSQL
- MongoDB

Tous les services sont orchestrés via Docker Compose.

Le frontend est servi par Nginx, qui joue également le rôle de reverse proxy vers le backend pour les requêtes API.

Ce choix permet :

- séparation claire des responsabilités
- meilleure maintenabilité
- environnement reproductible

3. Gestion des variables d'environnement

J'ai créé un fichier .env.develop sur le serveur contenant :

- paramètres de connexion PostgreSQL
- URI MongoDB
- JWT_SECRET
- PORT

Les variables d'environnement sont essentielles, notamment pour :

- sécuriser les secrets

éviter de hardcoder les credentials

adapter la configuration selon l'environnement

Un point important :
Vite injecte les variables d'environnement au moment du build.
Donc toute modification de VITE_API_URL nécessite un rebuild du frontend.

4. Build et démarrage des conteneurs

Une fois le projet cloné sur l'instance :

```
docker compose up --build -d
```

L'option (flague) --build était indispensable pour forcer la reconstruction des images.

Vérification :

```
docker compose up --build -d
```

Le backend était marqué healthy, ce qui signifie que le healthcheck configuré fonctionnait correctement.

5. Initialisation de la base de données (Seed)

Lors du premier lancement, une erreur est apparue :

```
relation "menus" does not exist
```

Cela signifiait que le schéma PostgreSQL n'avait pas encore été appliqué, il etait vide.

J'ai donc exécuté le seed depuis le conteneur backend :

```
docker compose run backend npm run seed-menu-dev
```

Après exécution réussie, les tables ont été créées et remplies.

L'endpoint /api/menus a ensuite commencé à retourner des données correctement.

6. Problèmes rencontrés et résolution

Erreur 405 sur /auth/register

Cause : mauvaise configuration de la base URL côté frontend et proxy Nginx.

Correction :

- Vérification du VITE_API_URL
- Rebuild complet du frontend
- Vérification du proxy_pass dans Nginx

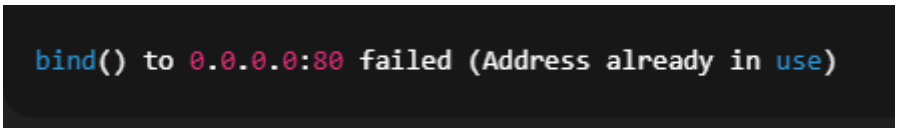
Erreur JavaScript : Cannot read properties of undefined (filter)

Cette erreur était liée à une réponse 500 backend qui envoyait des données vides.
Le frontend essayait d'appliquer .filter() sur un objet undefined.

Après correction du seed et du schéma, le problème a disparu.

Problème Certbot (port 80 occupé)

Lors de la tentative d'installation d'un certificat SSL :



Le port 80 était déjà utilisé par le conteneur de Nginx.

C` etait un defi, la configuration a faire :

Quand on utilise Docker pour le reverse proxy, on ne peut pas utiliser en parallèle un Nginx système sur le même port.

La solution correcte serait :

intégrer Certbot dans l'architecture Docker ou

utiliser un reverse proxy dédié (Traefik, Nginx externe, etc.)

7. Analyse personnelle

Ce déploiement m'a permis de comprendre concrètement :

- la différence entre environnement local et environnement cloud
- l'importance des variables d'environnement
- le rôle critique du seed et du schéma SQL
- la logique du reverse proxy
- l'importance du rebuild dans Vite

J'ai également l`imperssion que la majorité des erreurs en production ne viennent peut-etre pas du code, mais :

de la configuration

des ports

des variables d'environnement

des bases non initialisées

8. Conclusion

Le déploiement de Vite et Gourmand sur AWS m’a permis de mettre en pratique :

- conteneurisation avec Docker

- l’orchestration avec Docker Compose

- configuration serveur Linux

- gestion de base de données en environement similaire de la production

- résolution de bugs liés à l’infrastructure

Même si le processus peut sembler simple en théorie, il nécessite une compréhension globale de l’architecture applicative.

Ce déploiement constitue une des mes rares occasions de s`occuper de la mise en production fonctionnelle de l’application, et une étape importante vers une architecture plus robuste et scalable je l`espere sincerment.