Documentation technique

1. Réflexions initiales technologiques

Contexte du projet

Le projet **Vite & Gourmand** a été réalisé dans le cadre de l'ECF du titre professionnel **Développeur Web et Web Mobile** (Studi).
Il s'agit d'une application web permettant à un service traiteur de présenter ses menus, de gérer les commandes clients et de suivre l'activité de l'entreprise via des statistiques.

L'application distingue plusieurs rôles (visiteur, utilisateur, employé, administrateur) et couvre l'ensemble du cycle de commande, de la consultation des menus jusqu'à la confirmation et au suivi.

Choix technologiques et justifications

Figma

L'outil **Figma** a été utilisé lors de la phase de conception afin de réaliser les maquettes des interfaces utilisateur.
Il a permis de définir la structure des pages, l'organisation des contenus et la cohérence graphique avant le développement, facilitant ainsi la mise en œuvre du front-end et la conformité avec la charte graphique du projet.

Draw.io

L'outil **draw.io** a été utilisé pour la modélisation des diagrammes UML, notamment le diagramme de classes, le diagramme de cas d'utilisation et le diagramme de séquence.
Son utilisation permet de formaliser la conception technique de l'application, de clarifier les relations entre les composants et d'illustrer le fonctionnement global du système.

XAMPP

L'environnement **XAMPP** a été utilisé pour le développement local de l'application.
Il permet de disposer rapidement d'un serveur **Apache**, de **PHP** et de **MySQL** dans un environnement cohérent et stable, sans configuration complexe.
Ce choix facilite la mise en place du projet, les tests locaux et le débogage.

Visual Studio Code

L'éditeur **Visual Studio Code** a été retenu comme environnement de développement (IDE) pour ce projet.
Il offre un support efficace pour le développement PHP, JavaScript, HTML et CSS.

Son utilisation permet d'améliorer la productivité, la lisibilité du code et le respect des bonnes pratiques de développement.

PHP sans framework

Le choix de **PHP sans framework** a été volontaire afin de :

- Maîtriser les **fondamentaux du langage** (sessions, sécurité, PDO, architecture),
- Éviter l'abstraction excessive induite par un framework.

Ce choix permet de démontrer la capacité à structurer une application complète sans dépendance lourde.

Architecture MVC

Une architecture **MVC personnalisée** a été mise en place afin de :

- Séparer clairement les responsabilités :
    - **Model** : logique métier et accès aux données,
    - **View** : affichage et interfaces utilisateur,
    - **Controller** : gestion des requêtes et orchestration,
- Améliorer la **lisibilité**, la **maintenabilité** et l'évolutivité du projet,
- Rester cohérent avec les bonnes pratiques enseignées dans la formation.

MySQL (base relationnelle)

MySQL a été retenu pour stocker les données structurées :

- Utilisateurs,
- Menus,
- Plats,
- Commandes,
- Avis.

Ce choix est justifié par :

- La **nature relationnelle** des données,
- La nécessité de garantir l'intégrité référentielle,
- La compatibilité avec PDO et les environnements d'hébergement classiques.

MongoDB (base NoSQL)

MongoDB est utilisé exclusivement pour la gestion des **statistiques** (chiffre d'affaires, commandes par période, menus les plus commandés).

Ce choix permet :

- De dissocier les données analytiques des données transactionnelles,
- D'exploiter un stockage **NoSQL orienté agrégation**,
- De répondre à un cas d'usage concret et pertinent du NoSQL.

En local, MongoDB est lancé **via Docker**.
En production, MongoDB est hébergé sur **MongoDB Atlas**.

JavaScript et AJAX

Le projet utilise **JavaScript vanilla**, sans framework front-end, afin de :

- Maîtriser les mécanismes fondamentaux du DOM et des événements,
- Utiliser **AJAX** pour dynamiser certaines interactions (filtres, formulaires, confirmations) sans rechargement de page.

Chart.js

La librairie **Chart.js** est utilisée pour :

- Représenter graphiquement les statistiques,
- Améliorer la lisibilité des données pour l'administrateur,
- Produire des graphiques simples, clairs et interactifs.

PHPMailer

PHPMailer est utilisé pour :

- L'envoi d'emails de confirmation de commande,

- Garantir une gestion fiable et standardisée des emails,
- Éviter l'utilisation directe de la fonction mail() de PHP.

Docker

Docker est utilisé uniquement pour MongoDB en environnement local, afin de :

- Simplifier l'installation,
- Garantir un environnement cohérent,
- Éviter l'installation directe de MongoDB sur la machine hôte.
-

Hébergement

L'application est hébergée sur **Heroku**, conformément aux recommandations de la formation Studi.

2. Configuration de l’environnement de travail

Prérequis techniques

Avant toute installation, les éléments suivants doivent être disponibles :

- **Système d’exploitation** : Windows
- **Serveur local** : XAMPP
- **PHP** : version 8.2
- **MySQL** : version 8.x
- **Docker** : installé et fonctionnel (MongoDB)
- **Navigateur web** : Chrome, Firefox ou équivalent

Installation du projet en local

Clonage du dépôt

git clone https://github.com/Gatien37/Vite-Gourmand.git

Placer ensuite le dossier du projet dans le répertoire htdocs de XAMPP.

Démarrage des services

Dans XAMPP, démarrer :

- Apache
- MySQL

Vérifier que Docker est actif afin de permettre le lancement de MongoDB.

Configuration de la base MySQL

Création de la base

Créer une base de données nommée :

vite_gourmand

Encodage recommandé : utf8mb4.

Import des scripts SQL

Importer les fichiers suivants dans l'ordre :

1. database/schema.sql
2. database/seed.sql

Configuration de la connexion MySQL

Le fichier suivant n'est **pas versionné pour des raisons de sécurité** :

config/database.php

Il doit être créé à partir de l'exemple fourni dans le dépôt :

config/database.example.php

MongoDB (statistiques)

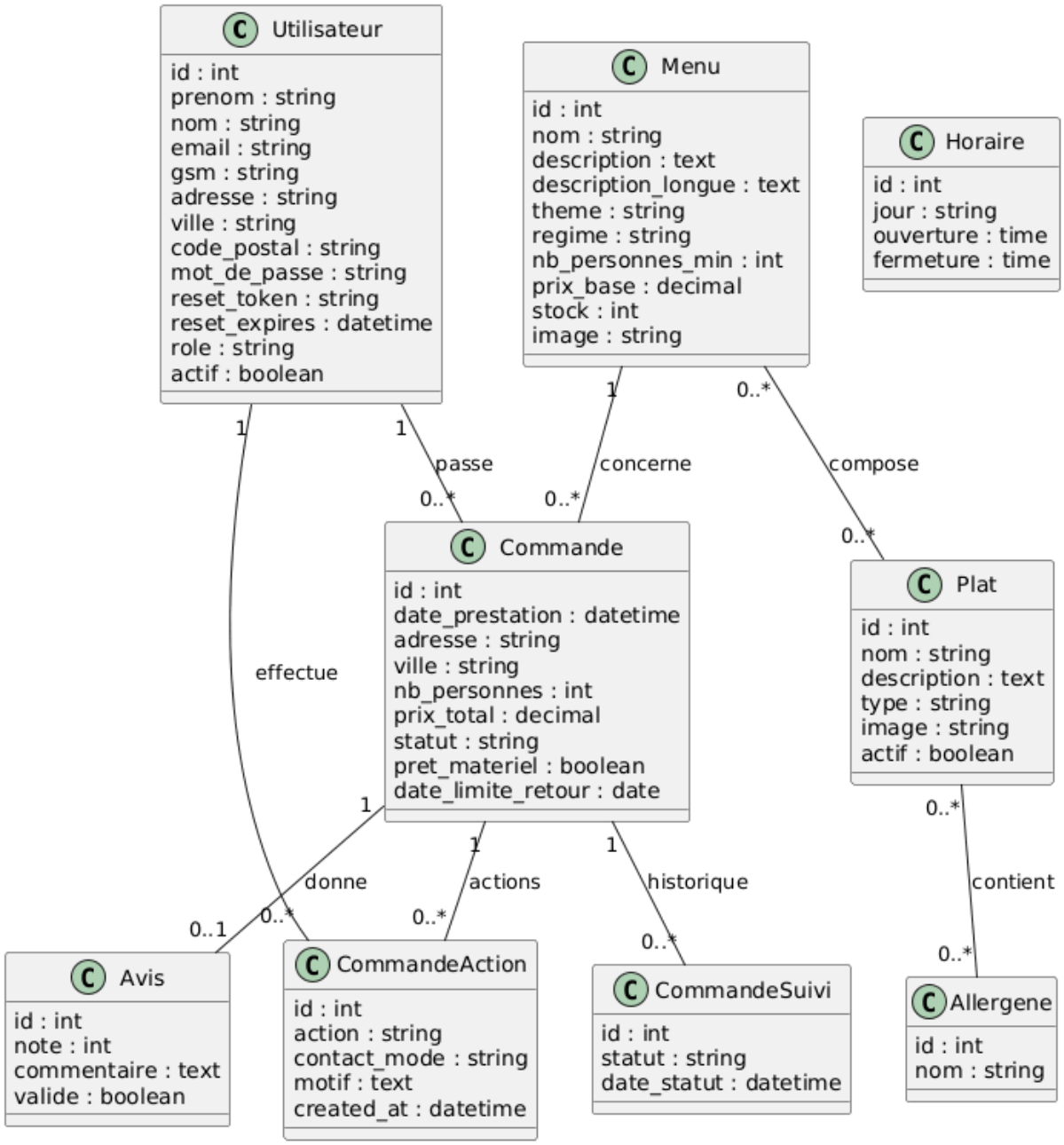
- MongoDB est utilisé pour le stockage des statistiques
- Il est lancé via Docker
- La base et les collections sont créées automatiquement
- Aucune configuration manuelle n'est requise

Lancement de l'application

Une fois les services actifs, l'application est accessible à l'adresse :

http://localhost/vite-gourmand/public/index.php

3. Modèle conceptuel de données



Le modèle de données repose sur une structure relationnelle claire, permettant de gérer les utilisateurs, les menus, les commandes et les avis.

Entités principales

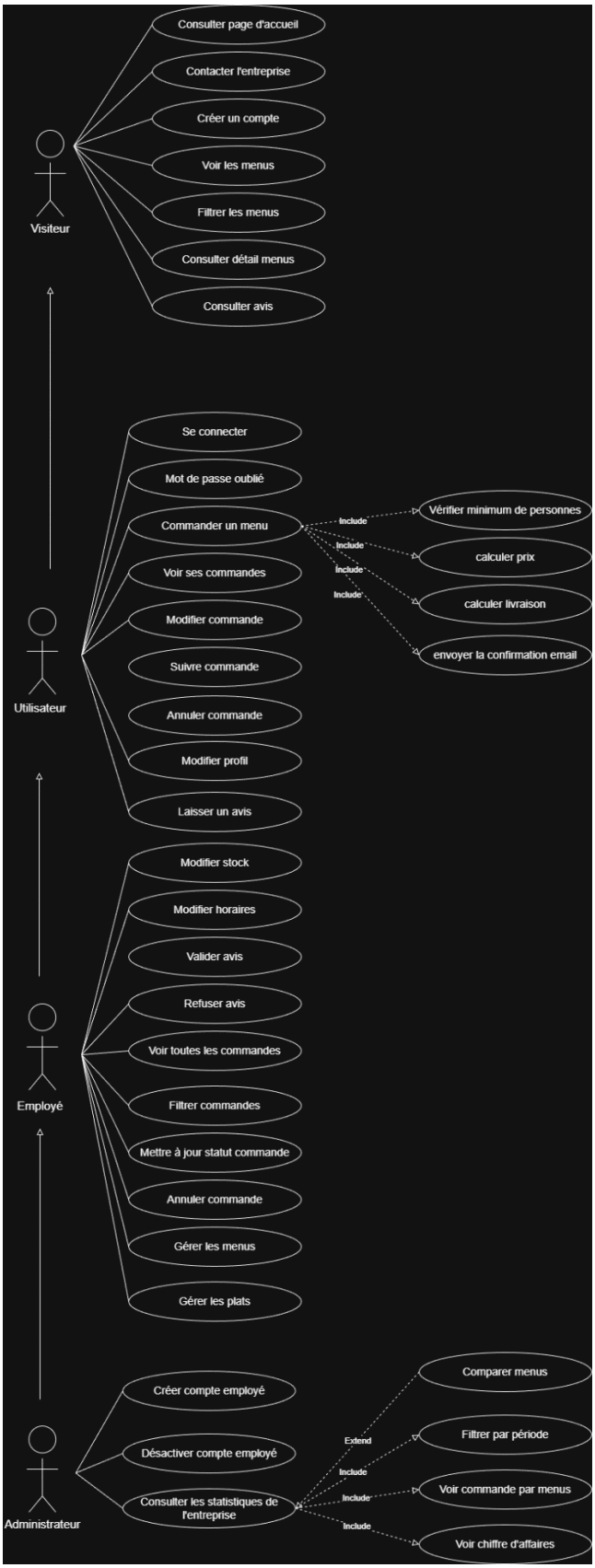
- **Utilisateur** : clients, employés et administrateurs (gestion par rôle)
- **Menu** : offres proposées par le traiteur

- **Plat** : éléments composant les menus
- **Commande** : prestations commandées par les utilisateurs
- **Avis** : retours clients après une commande
- **CommandeSuivi** : historique des statuts d’une commande
- **CommandeAction** : actions internes liées à une commande
- **Horaire** : plages d’ouverture
- **Allergène** : informations associées aux plats

Les relations entre entités respectent les cardinalités fonctionnelles (1-N, N-N) et assurent la cohérence des données.

4. Diagrammes UML

Diagramme de cas d’utilisation

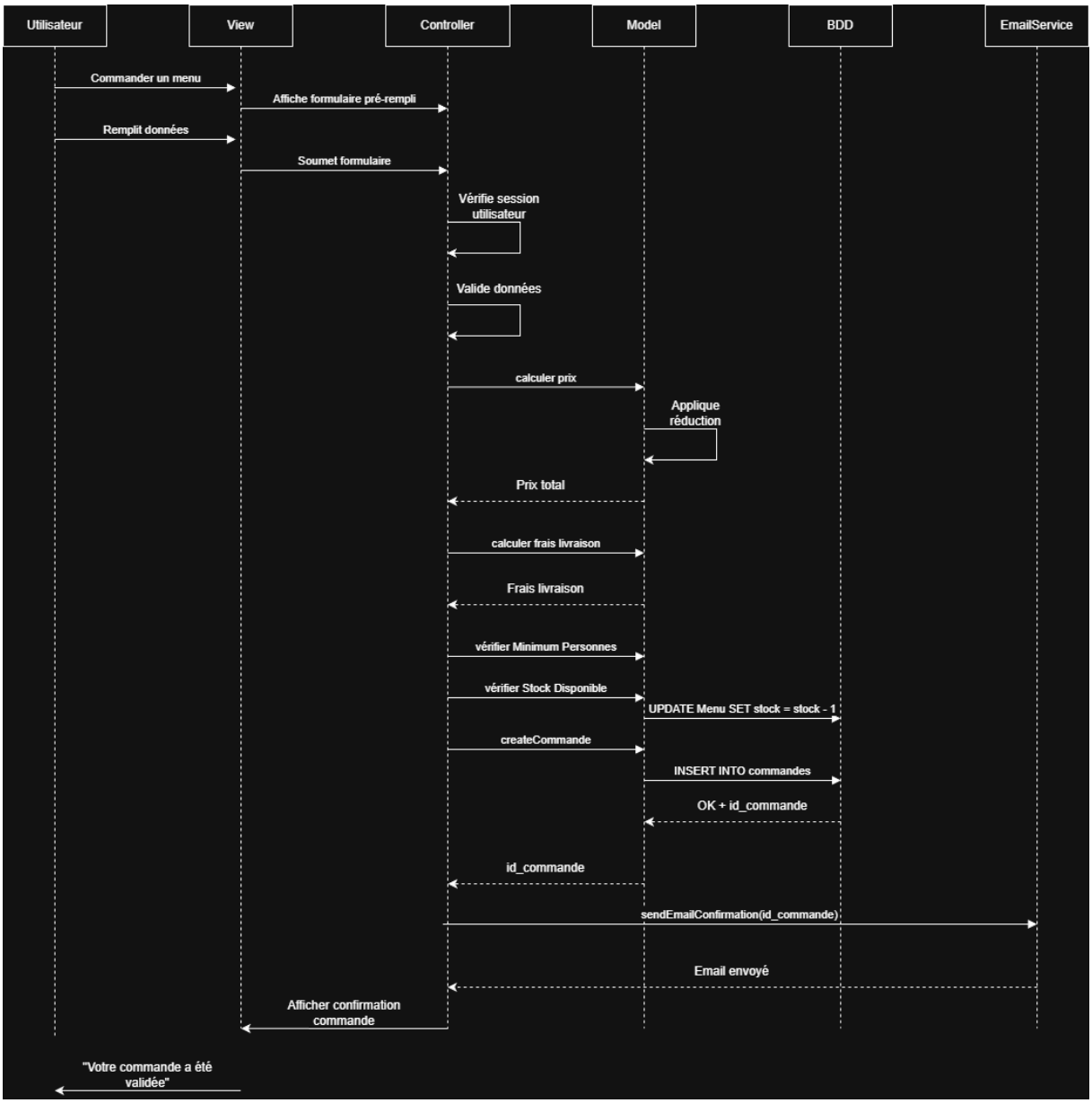


Le diagramme de cas d’utilisation identifie clairement les fonctionnalités accessibles selon le rôle :

- Visiteur : consultation, création de compte
- Utilisateur : commande, suivi, avis
- Employé : gestion des commandes, menus et avis
- Administrateur : gestion des employés et statistiques

Les relations <<include>> permettent de factoriser les règles métier (calcul du prix, vérification du minimum, livraison, email).

Diagramme de séquence



Le diagramme de séquence représente le scénario **« Commander un menu »**.
Il illustre :

- L'interaction utilisateur / interface,
- La validation côté contrôleur,
- L'application des règles métier,
- L'enregistrement en base,
- L'envoi de l'email de confirmation.

Ce diagramme met en évidence le fonctionnement MVC et la circulation des données.

5. Sécurité de l’application

Authentification et gestion des mots de passe

L’application met en œuvre une authentification basée sur les sessions PHP. Les mots de passe utilisateurs ne sont jamais stockés en clair : ils sont **hachés** lors de l’inscription et lors des changements de mot de passe en utilisant password_hash().

Une politique de complexité de mot de passe est appliquée (minimum 10 caractères, présence de majuscule, minuscule, chiffre et caractère spécial) afin de limiter les risques liés aux mots de passe faibles.

Gestion des rôles et contrôle d’accès

L’application implémente un contrôle d’accès basé sur les rôles stockés dans la table utilisateur (champ rôle).
Les zones sensibles (espace utilisateur, employé, administrateur) sont protégées par des middlewares dédiés (requireUtilisateur.php, requireEmploye.php, requireAdmin.php). Ces mécanismes vérifient systématiquement les droits avant l’exécution des pages concernées, ce qui empêche l’accès par simple saisie d’URL lorsque le rôle n’est pas autorisé.

Sessions et sécurité des cookies

La gestion des sessions repose sur les sessions PHP natives ($_SESSION) et un fichier d’initialisation (initSession.php).
Les cookies de session sont configurés avec des attributs de sécurité renforcés :

- HttpOnly pour limiter l’accès au cookie depuis JavaScript,
- Secure activé automatiquement lorsque l’application est utilisée en HTTPS,
- SameSite=Strict afin de réduire les risques de CSRF via des requêtes cross-site.

Protection CSRF

Les formulaires et actions nécessitant une authentification sont protégés par un **token CSRF**. Ce jeton est généré côté serveur, stocké en session et vérifié à la soumission des formulaires afin de s’assurer que la requête provient bien de l’application et de l’utilisateur authentifié. Cette protection couvre l’ensemble des espaces nécessitant une connexion (utilisateur, employé, administrateur).

Sécurisation des accès à la base de données (SQL)

L'accès à la base relationnelle MySQL est effectué via **PDO**. Les requêtes manipulant des entrées utilisateur utilisent des **requêtes préparées** (prepare() / execute()), ce qui permet de prévenir les injections SQL en séparant strictement la structure de la requête et les données.

Validation des données et prévention des failles XSS

Les données reçues via formulaires (inscription, connexion, commande, etc.) font l'objet d'une **validation côté serveur** afin d'éviter qu'un contournement du JavaScript ne permette d'injecter des valeurs invalides ou dangereuses.
À l'affichage, les sorties sont sécurisées via htmlspecialchars() pour limiter les risques de **Cross-Site Scripting (XSS)** lors de l'affichage de données potentiellement contrôlées par l'utilisateur.

Emails sensibles et réinitialisation de mot de passe

La fonctionnalité "mot de passe oublié" repose sur l'envoi d'un email contenant un **token unique** de réinitialisation, associé à une **durée de validité** (1 heure).
Aucun mot de passe n'est transmis par email : le lien permet uniquement de définir un nouveau mot de passe via un processus contrôlé côté serveur.

Protection des configurations et secrets

Les fichiers de configuration contenant des informations sensibles, notamment la connexion à la base de données (database.php), **ne sont pas versionnés** dans le dépôt GitHub pour des raisons de sécurité.
Un fichier d'exemple (database.example.php) est fourni afin de permettre l'installation en local sans exposer de données confidentielles.

En environnement de production, les accès aux bases de données (MySQL via JAWSDB_URL, MongoDB via MONGO_URI) sont gérés exclusivement via des **variables d'environnement**.
La gestion des erreurs de connexion est volontairement générique afin d'éviter toute divulgation d'informations techniques sensibles (hôte, schéma, identifiants).

Sécurité MongoDB (statistiques)

MongoDB est utilisé uniquement pour les statistiques et est hébergé sur **MongoDB Atlas**, accessible via une URI sécurisée. L'accès est réalisé côté serveur ; les données analytiques sont dissociées des données transactionnelles, ce qui limite l'exposition et le périmètre fonctionnel de la partie NoSQL.

6. Déploiement

L'application est déployée sur Heroku.
La base relationnelle est fournie via un addon MySQL Heroku, configuré à l'aide de variables d'environnement.
La base NoSQL MongoDB est hébergée sur MongoDB Atlas et intégrée à l'application via une URI sécurisée.

Architecture retenue

- **Heroku**
  - Hébergement PHP
  - Déploiement via Git et buildpack PHP
- **MySQL**
  - Base relationnelle via addon Heroku (JawsDB)
- **MongoDB Atlas**
  - Base NoSQL managée externe
  - Connexion sécurisée via URI

Étapes de déploiement

1. Création de l'application Heroku

heroku login

heroku create vite-gourmand

Heroku détecte automatiquement PHP et installe l'environnement nécessaire.

Si le point d'entrée est public/, un Procfile est ajouté :

web: vendor/bin/heroku-php-apache2 public/

2. Ajout de MySQL (addon)

heroku addons:create jawsdb:kitefin

La variable d'environnement JAWSDB_URL est automatiquement fournie.

3. Connexion MySQL via variables d’environnement

La connexion est configurée dynamiquement :

```
$url = parse_url(getenv('JAWSDB_URL'));


$pdo = new PDO(
  "mysql:host={$url['host']};dbname=" . ltrim($url['path'], '/'),
  $url['user'],
  $url['pass'],
  [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
);
```

4. MongoDB Atlas

- Création d’un cluster gratuit (M0)
- Création d’un utilisateur
- Autorisation IP
- Récupération de l’URI MongoDB

Configuration sur Heroku :

```
heroku config:set MONGO_URI="mongodb+srv://..."
```

5. Déploiement final

```
git push heroku main
heroku open
```