Vite! Gourmand

Documentation Technique

Architecture, Base de Données & Sécurité

Date: 19/02/2026

Table des Matières

1. Présentation du Projet

Vite! Gourmand est une application de commande en ligne (Click & Collect) pour un service traiteur. Le projet vise à fournir une expérience utilisateur fluide pour la commande de repas, tout en offrant une interface d'administration robuste pour la gestion des produits et des commandes.

2. Stack Technique

- **Langage** : PHP 8.2
- **Framework Backend** : Symfony 7.0
- **Base de Données** : SQLite (Environnement de développement/POC)
- **ORM** : Doctrine ORM
- **Frontend** : Twig, CSS (Vanilla), JavaScript (Stimulus)
- **Gestionnaire de paquets** : Composer (PHP), NPM (JS/CSS)
- **Serveur Web** : Apache / PHP Built-in Server

3. Architecture Logicielle

L'application suit le pattern **MVC (Modèle-Vue-Contrôleur)** imposé par Symfony.

Structure des Dossiers

```
/src
  /Controller       # Logique métier (Admin, User, Security, etc.)
  /Entity           # Classes mapping base de données (Model)
  /Form             # Gestion des formulaires (Symfony Form)
  /Repository       # Requêtes SQL (Doctrine)
  /Security         # Gestion des accès (Voters, Authenticators)
/templates          # Vues Twig (Frontend)
/public             # Point d'entrée, images, assets compilés
/assets             # Sources CSS/JS (AssetMapper)
```

Contrôleurs Principaux

- `HomeController` : Page d'accueil et présentation.
- `MenuController` : Affichage public des menus et produits.
- `OrderController` : Gestion du panier et validation de commande.
- `Admin/*Controller` : Back-office protégé (CRUD Produits/Menus).
- `Registration/SecurityController` : Authentification.

4. Modèle de Données (MCD/MLD)

La base de données relationnelle est gérée via Doctrine ORM.

Entités Principales

User (Utilisateur)

- `id`, `email`, `password` (haché), `roles` (JSON), `firstName`, `lastName`.
- Relation *OneToMany* avec **Order**.

Product (Produit)

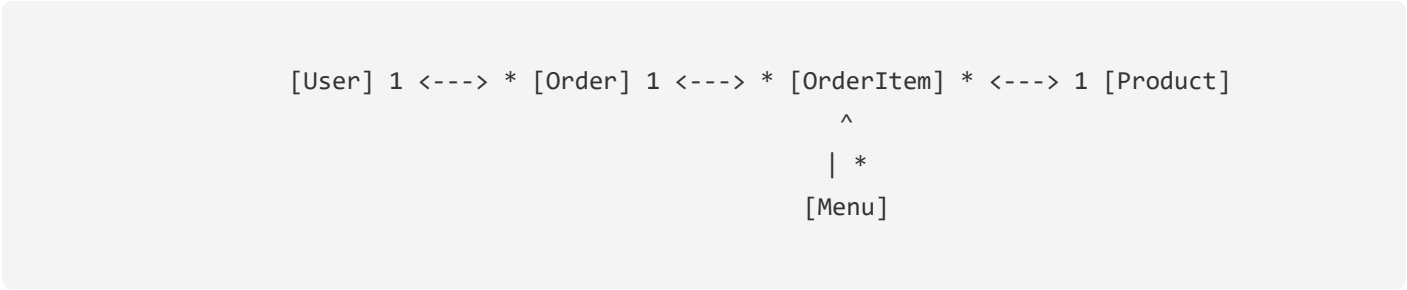
- `id`, `name`, `price`, `category` (starter/main/dessert/drink), `imageName`.
- Relation *ManyToMany* avec **Allergen, Diet, Theme**.
- Relation *ManyToMany* avec **Menu** (Inverse side).

Menu

- `id`, `name`, `price`, `stock`, `minPeople`.
- Relation *ManyToMany* avec **Product** (Un menu contient plusieurs produits).

Order (Commande)

- `id`, `status` (cart, paid, delivered...), `totalPrice`, `deliveryTime`.
- Relation *ManyToOne* avec **User**.
- Relation *OneToMany* avec **OrderItem**.



5. Sécurité

Authentification

Utilisation du composant `SecurityBundle` de Symfony.

- **Provider** : EntityUserProvider (basé sur l'entité User).
- **Hasher** : `auto` (Argon2id ou Bcrypt selon disponibilité).
- **Protection CSRF** : Activée sur tous les formulaires.

Contrôle d'Accès (Roles)

Rôle	Accès
`ROLE_USER`	Accès au profil, historique de commandes, laisser un avis.
`ROLE_EMPLOYEE`	Accès au back-office (lecture seule ou modification limitée).
`ROLE_ADMIN`	Accès complet (Gestion utilisateurs, configuration).

6. Frontend & Assets

Moteur de Template

Utilisation de **Twig** pour le rendu HTML sécurisé (auto-escaping XSS).

Asset Mapper

Le nouveau composant **AssetMapper** de Symfony 6.4+ est utilisé pour gérer les fichiers CSS et JS sans étape de build complexe (pas de Webpack Encore/Node.js obligatoire en prod).

Design System

- **CSS** : Vanilla CSS avec variables CSS (`--primary-color` , etc.).
- **Style** : Approche "Glassmorphism" (transparence, flou) pour une esthétique moderne.
- **Icônes** : FontAwesome 6 (CDN).

- **Responsive** : Design adaptatif (Mobile First) via Flexbox et CSS Grid.