

# Documentation Technique

## Méthodologie et gestion de projet

Le développement de Vite & Gourmand a été conduit selon une démarche agile itérative, inspirée des principes Scrum.

Le projet a été structuré par phases successives :

modélisation de la base de données, mise en place de l'architecture sécurisée, implémentation du frontend, gestion d'état globale, puis optimisation progressive.

L'organisation des tâches a été réalisée via Trello, permettant :

- la priorisation des fonctionnalités
- le suivi de l'avancement
- l'adaptation continue des objectifs
- une vision claire de la progression du projet

- 🔗 Lien Trello : <https://trello.com/b/G02hHQge/vite-gourmand>
- 🔗 Dépôt GitHub <https://github.com/Mayk-ITdS/vite-et-gourmand>
- 🔗 Application déployée : <http://51.20.182.243>

Cette approche a permis de privilégier la solidité architecturale et la sécurité dès les premières phases du développement, avant l'extension fonctionnelle des espaces avancés (Employé / Administrateur).

## Architecture, choix technologiques et intégration

### Architecture générale du système

Le projet *Vite & Gourmand* repose sur une architecture client–serveur structurée, conforme aux standards actuels du développement full-stack. L'application est construite autour d'une séparation claire entre la couche de présentation (frontend), la couche de logique métier et d'API (backend), ainsi que la couche de persistance des données (base relationnelle).

Cette organisation répond à un objectif stratégique : garantir évolutivité, maintenabilité et sécurité. Le système a été pensé dès sa conception comme une plateforme capable d'évoluer vers une solution professionnelle complète, intégrant gestion opérationnelle, supervision administrative et optimisation logistique.

L'ensemble de l'application est conteneurisé via Docker et orchestré à l'aide de Docker Compose. Cette approche assure la reproductibilité de l'environnement, facilite le déploiement et garantit une cohérence entre les environnements de développement, de test et de production.

## **Frontend - Expérience utilisateur et performance**

La couche frontend est développée en React avec TypeScript et Vite, et stylisée via TailwindCSS, en s'appuyant sur des primitives accessibles et des composants prédéfinis issus de bibliothèques spécialisées. Le choix de React s'inscrit dans une volonté de construire une interface dynamique, modulaire et maintenable. L'architecture basée sur des composants réutilisables favorise la scalabilité et la reproductibilité, notamment dans la perspective d'implémentation complète des espaces employé et administrateur.

TypeScript a été intégré dès le départ afin d'introduire un typage statique strict. Cette décision améliore la robustesse du code, réduit les erreurs à l'exécution et facilite la refactorisation à long terme. Dans un projet de cette envergure, ce choix démontre une approche professionnelle et structurée.

Vite a été retenu comme bundler pour sa rapidité exceptionnelle au démarrage et lors du hot module replacement. Cela permet un cycle de développement fluide et productif, particulièrement pertinent dans un projet itératif nécessitant de nombreuses évolutions visuelles.

TailwindCSS adopte une approche utility-first. Ce choix favorise la cohérence visuelle et permet une création rapide, directement dans les composants, tout en conservant une configuration centralisée des styles. L'ajustement visuel peut ainsi être réalisé rapidement et de manière testable.

L'état global de l'application est assuré par Redux Toolkit. Cette solution centralise les données critiques telles que l'authentification, les menus et les réservations. Elle permet également d'éviter des appels réseau redondants grâce à l'exploitation du filtrage local et à la mémoïsation des sélecteurs, améliorant ainsi la performance perçue.

### **Bibliothèques de composants : MUI, shadcn/ui et Radix UI**

L'interface utilisateur de Vite & Gourmand repose sur une approche hybride combinant TailwindCSS, shadcn/ui, Radix UI et ponctuellement MUI.

Cette combinaison n'est pas le fruit d'un empilement arbitraire, mais d'un choix structuré visant à concilier flexibilité stylistique, accessibilité et robustesse fonctionnelle.

### **Radix UI - Accessibilité et primitives solides**

Radix UI fournit des primitives accessibles et non stylisées.

Ces composants (Dialog, Dropdown, Tabs, etc.) sont conçus avec une attention particulière à l'accessibilité (ARIA, gestion du focus, navigation clavier).

L'intégration de Radix permet :

- une base technique fiable pour les interactions complexes
- une conformité aux standards d'accessibilité
- une séparation entre logique comportementale et style visuel

Radix agit ainsi comme fondation technique des composants interactifs.

### **shadcn/ui - Design system composable**

shadcn/ui s'appuie sur Radix et Tailwind pour proposer des composants stylisés mais entièrement personnalisables.

Ce choix permet :

- un design cohérent et moderne
- une personnalisation complète sans dépendance lourde
- une intégration fluide dans une architecture Tailwind

shadcn/ui offre un compromis intéressant entre rapidité de développement et maîtrise totale du design system.

### **MUI – Composants structurés et formulaires complexes**

MUI (Material UI), basé sur les principes du Material Design, est utilisé pour certains composants fonctionnels nécessitant une gestion avancée des états, notamment :

- formulaires
- cartes structurées
- composants nécessitant validation et gestion d'erreurs

MUI apporte :

- robustesse des composants
- gestion fine des états (hover, focus, error, disabled)
- conformité aux standards d'accessibilité
- documentation mature et éprouvée

L'objectif n'est pas d'uniformiser tout le design autour de Material Design, mais d'exploiter la maturité technique de la bibliothèque pour des éléments critiques.

### **Cohérence de l'approche UI**

La combinaison Tailwind + Radix + shadcn + MUI repose sur une logique claire :

- Tailwind pour la direction artistique et la flexibilité visuelle
- Radix pour la fiabilité comportementale
- shadcn pour la cohérence design composable
- MUI pour les composants structurés nécessitant une robustesse renforcée

Cette architecture permet de dissocier :

- la logique d'interaction
- le design system
- les composants fonctionnels avancés

Le résultat est une interface premium, modulaire et évolutive, adaptée à un projet en croissance.

### **Backend - Logique métier et sécurité**

Le backend est développé en Node.js avec Express.js et TypeScript.

Node.js a été choisi pour sa capacité à gérer efficacement un grand nombre de connexions simultanées grâce à son modèle asynchrone basé sur l'event loop. Dans le contexte d'une

plateforme événementielle susceptible de recevoir de nombreuses requêtes, ce choix assure scalabilité et performance.

Express.js offre une structure légère pour la construction d'une API REST. Sa simplicité permet une organisation claire des routes, middlewares, contrôleurs, services et repositories.

Le backend adopte une architecture en couches distinctes : contrôleurs, services métier, repositories d'accès aux données et DTOs. Cette séparation des responsabilités garantit une meilleure testabilité, une maintenance facilitée et une conformité aux bonnes pratiques de développement logiciel.

La sécurité repose sur une authentification par JWT. Les middlewares backend assurent la vérification du token et la gestion des rôles. La séparation stricte entre utilisateur, employé et administrateur est assurée via un système relationnel robuste (roles et user\_roles).

## **Architecture orientée sécurité et approche “Database First”**

L'architecture de Vite & Gourmand repose sur un principe fondamental : la base de données constitue la source centrale de vérité du système.

Le choix a été fait d'adopter une approche dite “Database First”. Cela signifie que la structure des données, les contraintes métier et une partie des mécanismes de validation sont définis directement au niveau de PostgreSQL avant même l'implémentation complète de la logique applicative.

Cette stratégie repose sur plusieurs constats :

- la base de données est l'élément le plus stable d'un système,
- elle doit garantir l'intégrité indépendamment de la couche applicative,
- elle constitue la dernière ligne de défense en matière de sécurité et de cohérence.

## **Sécurité par conception (Security by Design)**

La sécurité n'est pas uniquement assurée au niveau du backend via les middlewares JWT. Elle est intégrée à plusieurs niveaux complémentaires.

Sécurité structurelle au niveau de la base

PostgreSQL est utilisé de manière avancée afin d'imposer des règles strictes :

- Types ENUM (diet\_enum, reservation\_status\_enum, product\_type\_enum) empêchant l'insertion de valeurs incohérentes
- Contraintes CHECK garantissant la validité métier
- Clés étrangères assurant la cohérence relationnelle
- Index ciblés optimisant les performances
- Triggers permettant la normalisation automatique des données
- Fonctions stockées sécurisées (SECURITY DEFINER) encapsulant la logique sensible

L'écriture de fonctions telles que ingest\_stock ou insert\_new\_user illustre cette approche : la logique critique est exécutée directement dans la base, réduisant le risque d'incohérence côté application

## Typage comme mécanisme de sécurité

Le typage statique via TypeScript ne constitue pas uniquement un confort de développement ; il représente une couche de sécurité supplémentaire.

Grâce aux interfaces, DTOs et types stricts :

- les structures de données sont validées dès la compilation,
- les accès incorrects aux propriétés sont bloqués,
- les erreurs de transformation sont détectées précocement,
- les appels API respectent un contrat formel.

La combinaison PostgreSQL + TypeScript crée ainsi une double validation :

- validation à la compilation (TypeScript),
- validation à l'exécution et à la persistance (PostgreSQL).

Cette redondance volontaire renforce la robustesse globale du système.

## Gestion d'état avec Redux

Le choix d'implémenter Redux Toolkit répond à une volonté de centraliser et sécuriser la gestion de l'état applicatif.

Bien que les hooks React puissent suffire pour des applications simples, la complexité croissante du projet - gestion des rôles, authentification, menus dynamiques et réservations — nécessitait une solution plus structurée.

Redux offre :

- une gestion prévisible et fiable de l'état global,
- une séparation claire entre logique métier et composants visuels,
- une traçabilité explicite des mutations,
- une meilleure maîtrise des flux de données.

Les thunks typés permettent de définir précisément les entrées, les sorties et les erreurs possibles des appels API. Cette capacité à typer simultanément les payloads d'entrée et les réponses renforce le contrat entre frontend et backend.

L'intégration de redux-persist permet une persistance contrôlée de l'état, notamment du token d'authentification. Celui-ci est encapsulé dans le store Redux plutôt que manipulé directement via localStorage, améliorant la cohérence et la sécurité.

## Cohérence globale

L'ensemble des choix technologiques n'est pas arbitraire. Chaque technologie répond à un besoin précis :

React pour l'expérience utilisateur dynamique, TypeScript pour la robustesse et le contrôle contractuel.

**PostgreSQL** pour l'intégrité métier, **Docker** pour la portabilité et la reproductibilité.

**JWT** et middleware pour la sécurité, **Nginx** pour une mise en production professionnelle.