# 📘 Portfolio BTS SIO SLAM

# Nom : Marie Marc-Antoine Session : 2025

# 📑 Sommaire

# [Tableau de Synthèse](#1-tableau-de-synth%C3%A8se)

# [Attestation de Travail](#2-attestation-de-travail)

# [Descriptions Succinctes des Projets](#3-descriptions-succinctes-des-projets)

# [Projet 1 : 1ClickAllEat](#projet-1-1clickalleat)

# Projet 2 : LDAP - Telora

# [Projet 3 : Portfolio Web](#projet-2-portfolio-web)

# [Veille Technologique](#4-veille-technologique)

# [Annexes](#5-annexes)

# [Annexes Projets](#annexes-projets)

# [Annexes Veille](#annexes-veille)

# Tableau de Synthèse (tableau au complet annexe 1)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Gérer le patrimoine informatique** | **Répondre aux incidents et aux demandes d’assistance et d’évolution** | **Développer la présence en ligne de l’organisation** | **Travailler en mode projet** | **Mettre à disposition des utilisateurs un service informatique** | **Organiser son développement professionnel** |
| **Portfolio** |  |  | X | X |  | X |
| **1ClickAllEat** |  |  | X | X | X | X |
| **Logiciel LDAP** | X | X |  | X | X | X |

# 2. Attestation de Travail

# *À insérer : attestation fournie par l’entreprise ou le tuteur de stage/alternance si applicable.*

# 3. Descriptions Succinctes des Projets

# Projet 1 : 1ClickAllEat

# Contexte : Projet réalisé en autonomie dans le cadre du BTS SIO SLAM, pour répondre à un besoin concret : fluidifier la gestion des réservations et des commandes dans la restauration, tout en offrant une expérience utilisateur moderne, rapide et sécurisée.

# Partenaires : Aucun (projet individuel)

# Environnement technique :

# *Front-end :* Blade (Laravel), Bootstrap 5, HTML5, CSS3, responsive design

# *Back-end :* Laravel 10 (PHP 8), architecture MVC

# *Base de données :* SQLite (développement), MySQL/PhpMyAdmin (production)

# *Outils :* Breeze (authentification), Faker (données de test), GitHub, VS Code, hébergement HTTPS

# Liste des tâches :

# Analyse du besoin, rédaction du cahier des charges et du diagramme de Gantt

# Modélisation de la base de données et création des migrations Laravel

# Développement des fonctionnalités principales : réservation, commande, gestion des utilisateurs, gestion des restaurants/tables/menus

# Implémentation de l’authentification sécurisée multi-profils (clients, restaurateurs)

# Développement d’un tableau de bord pour restaurateur (statistiques, gestion des réservations et commandes)

# Tests unitaires, validation des formulaires et gestion des erreurs

# Déploiement sur serveur distant, configuration du HTTPS et optimisation de la sécurité

# Rédaction de la documentation utilisateur et technique

# Difficultés rencontrées / solutions :

# Découverte et prise en main de Laravel (documentation officielle, tutoriels, forums)

# Gestion des droits et de la sécurité (middlewares, tests, validation)

# Déploiement et configuration serveur (documentation OVH, Let’s Encrypt, adaptation .env)

# Pas de maquette initiale, adaptation de templates Bootstrap (recherche d’exemples, tests responsive)

# Bilan personnel :

# Fier d’avoir mené ce projet seul, en développant des fonctionnalités avancées et en progressant énormément sur Laravel et l’écosystème PHP.

# Gain d’autonomie, de rigueur, et de capacité à résoudre des problèmes complexes.

# Pour la suite : intégrer un paiement en ligne, notifications avancées, API mobile.

# Projet 2 : Portfolio Web

# Date, lieu : Septembre 2024 à Mars 2025, personnel

# Contexte : Création de mon site portfolio pour présenter mes réalisations et compétences

# Partenaires : Aucun (projet individuel)

# Environnement technique : PHP, HTML, CSS, Bootstrap, XAMPP

# Bilan personnel :

# Amélioration de la rigueur en front-end

# Découverte du responsive design et de l’accessibilité

# 4. Veille Technologique

# Outils de veille utilisés :

# Feedly (agrégateur de flux RSS)

# LinkedIn, Twitter (veille réseaux sociaux professionnels)

# Newsletters spécialisées (AFUP, PHP Weekly, JavaScript Weekly)

# Google Alerts

# Forums : StackOverflow, OpenClassrooms

# Sujets de veille suivis :

# PHP : nouveautés (PHP 8.3), sécurité (CVE-2024-4577), bonnes pratiques, frameworks (Laravel)

# JavaScript : nouveautés ES2024, pipeline operator, Temporal API, sécurité XSS

# Sécurité web : failles récentes, bonnes pratiques, outils d’audit

# Tendances développement web : frameworks, accessibilité, performance

# 5. Annexes

# Annexes Projets

# Cahier des charges 1ClickAllEat (p.\*\* à \*\*)

# Diagrammes UML

# Captures d’écran de l’application

# Extraits de code commentés

# Diagramme de Gantt

# Annexes Veille

# Copies d’articles lus (PDF, captures)

# Tableaux comparatifs de frameworks

# Notes personnelles de synthèse

**Cahier des Charges - Projet 1ClickAllEat**

**1. Contexte**

La startup 1ClickAllEat nous mandate pour développer une application de réservation de restaurant au préalable. Cette solution vise à simplifier et rendre plus accessibles les restaurants traditionnels qui sont concurrencés par les fast-foods.

**2. Objectif Principal**

Proposer une solution optimisée pour plusieurs restaurants qui proposent des menus différents. L'application permettra d'optimiser les échanges entre les différents acteurs du restaurant (administration, service, cuisine, clients).

**3. Public Cible**

Restaurateurs et leur personnel

**4. Expression des Besoins**

L'application 1ClickAllEat doit offrir une gestion complète et optimisée des restaurants, permettant de suivre en temps réel les menus, les commandes et les réservations. L'ensemble des opérations sera parfaitement synchronisé avec les horaires d'ouverture, assurant une expérience fluide et sur-mesure pour chaque client.

La plateforme fournira aux restaurateurs les outils nécessaires à l'ajustement des offres, à la gestion des réservations et au suivi des commandes, accessibles en ligne pour un pilotage efficace des établissements à tout moment.

**5. Fonctionnalités**

**5.1. Fonctionnalités MVP (Minimum Viable Product)**

* **Gestion des restaurants**
  + Création et paramétrage des profils de restaurants
  + Configuration des horaires d'ouverture
  + Gestion des tables et capacités
* **Authentification des utilisateurs**
  + Système d'inscription/connexion sécurisé
  + Récupération de mot de passe
  + Différents niveaux d'accès selon les rôles
* **Accès spécifique restaurant**
  + Interface dédiée aux restaurateurs
  + Tableau de bord de gestion
  + Statistiques de base sur l'activité
* **Accès spécifique client**
  + Interface utilisateur intuitive
  + Recherche et filtrage des restaurants
  + Historique des commandes/réservations
* **Gestion de la carte/des menus**
  + Classification des articles au menu par catégorie
  + Ajout/modification/suppression d'articles
  + Gestion des prix et disponibilités
* **Gestion des commandes/réservations**
  + Création et suivi des réservations
  + Gestion des statuts des commandes
  + Notifications aux clients et restaurateurs

**5.2. Fonctionnalités MLP (Minimum Lovable Product)**

* **Génération de QR code**
  + Lien d'accès à la commande pour chaque carte de restaurant
  + QR code dynamique et personnalisable
* **Responsive**
  + Adaptation à tous les formats d'écran
  + Expérience utilisateur optimisée sur mobile, tablette et desktop
* **Paramétrage de la charte graphique**
  + Personnalisation des couleurs et logo du restaurant
  + Adaptation visuelle de la carte selon l'identité du restaurant
* **Paiement en ligne via API Stripe**
  + Intégration complète avec le package Laravel Cashier
  + Gestion sécurisée des transactions
  + Suivi des paiements et remboursements

**6. Contraintes Techniques**

* **Framework de développement**
  + Laravel (Back-end)
  + Interface utilisateur moderne et intuitive
* **Déploiement**
  + Sur un nom de domaine dédié
  + Hébergement sécurisé et performant
* **Intégration continue**
  + Automatisation des migrations de base de données
  + Gestion du vidage des caches de Laravel
  + Optimisation des fichiers pour l'environnement de production
* **Template**
  + Design responsive et moderne
  + Adaptable à l'identité visuelle des différents restaurants
* **Sécurité**
  + Certificat SSL (HTTPS) pour sécuriser les communications
  + Protection des données personnelles
* **Tests unitaires**
  + Mise en place de tests automatisés
  + Code qui teste le code

**Document préparé pour :** *1ClickAllEat*

Diagramme UML :

Diagramme de package 🡪

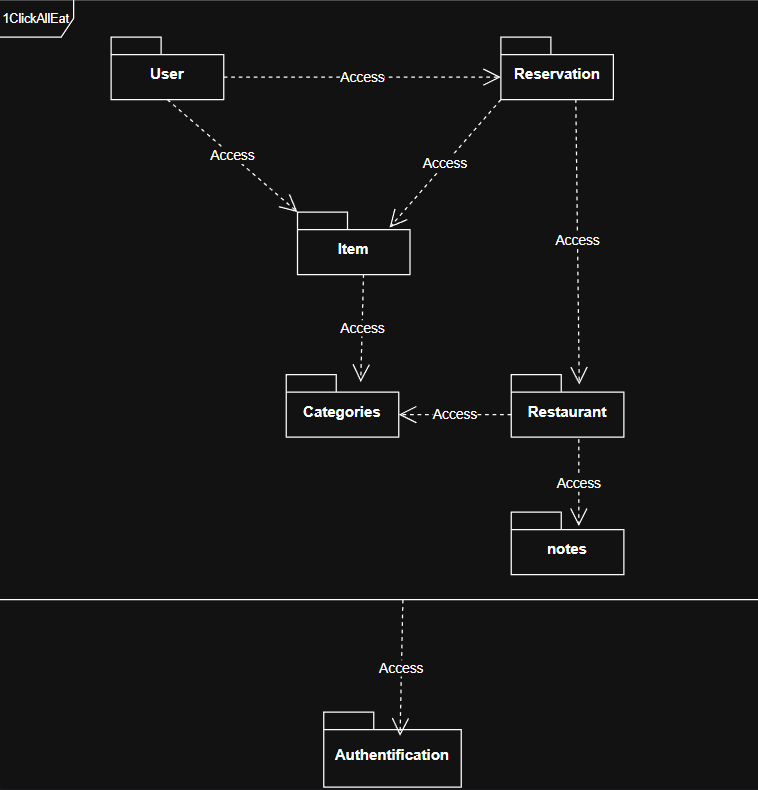
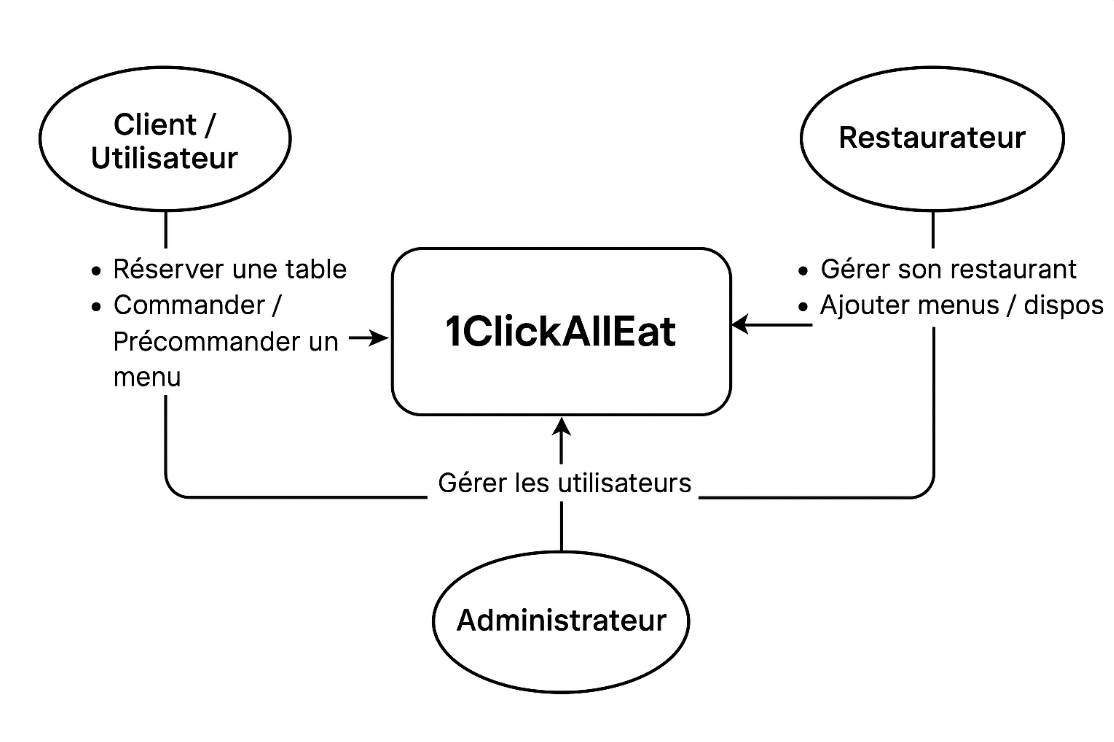


Diagramme de context 🡪



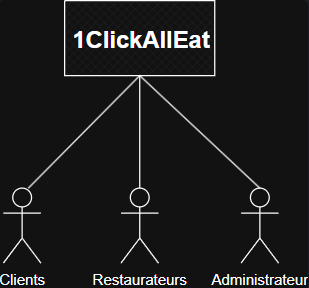
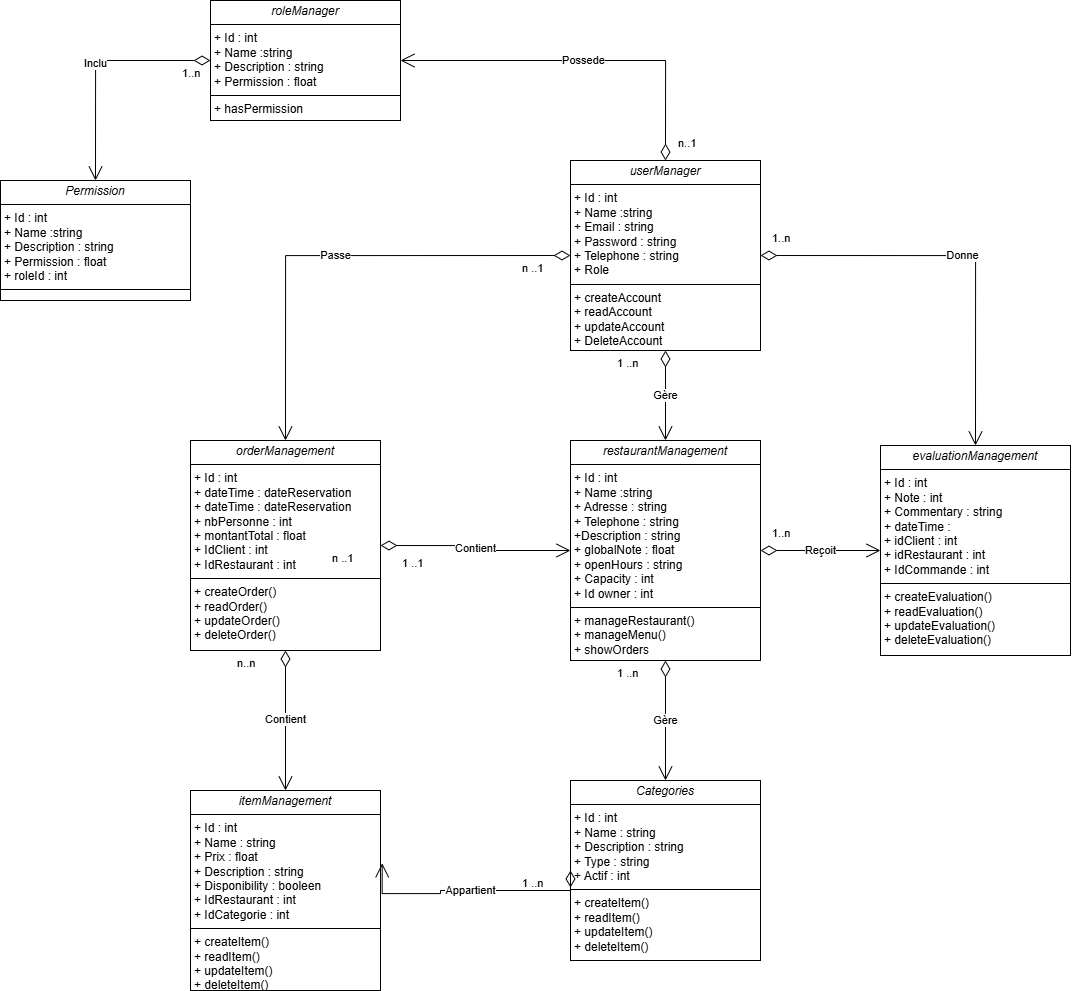
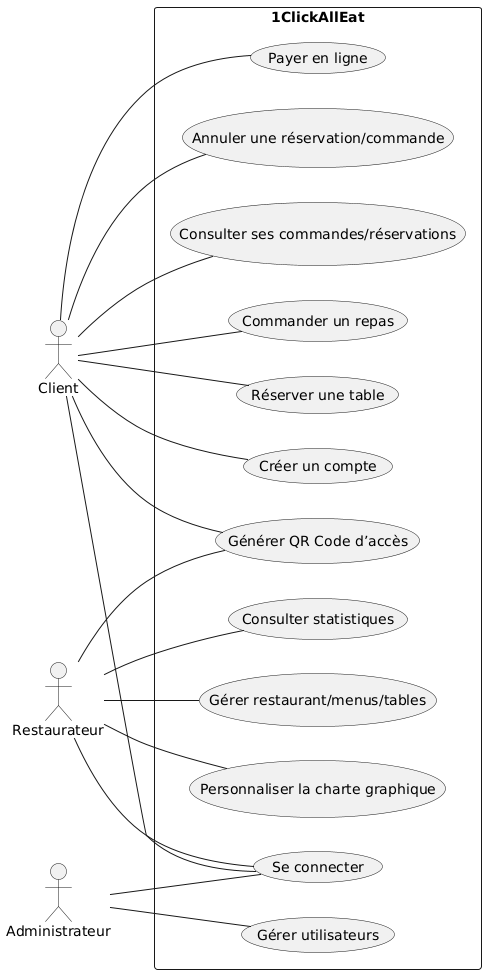


Diagramme de classe 🡪



Use\_case 🡪



Plan de test qualité 🡪

**Plan de Test Qualité – 1ClickAllEat**

**Objectif**

Fournir un cadre structuré pour valider que l’application 1ClickAllEat répond à toutes les exigences fonctionnelles (MVP, MLP), techniques et de qualité : gestion des restaurants, authentification, accès différencié, gestion des menus/commandes, QR Code, responsive, personnalisation graphique, paiement en ligne, sécurité, CI/CD, etc.

**Périmètre du test**

**Fonctionnalités MVP**

* Gestion des restaurants (création, modification, suppression)
* Authentification et gestion des utilisateurs (clients, restaurateurs)
* Accès différencié (dashboard restaurateur, espace client)
* Gestion des menus/cartes (catégorisation, affichage)
* Gestion des commandes (création, suivi, annulation)

**Fonctionnalités MLP**

* Génération et affichage du QR Code pour accès rapide
* Design responsive (mobile, tablette, desktop)
* Personnalisation graphique (charte, couleurs, logos)
* Paiement en ligne (API Stripe)

**Contraintes techniques**

* Laravel (PHP 8+)
* Déploiement HTTPS (certificat SSL)
* Intégration continue (migrations, cache, optimisation)
* Tests unitaires/fonctionnels automatisés

**Méthodologie**

* **Tests manuels et automatisés** : PHPUnit, Laravel Dusk, BrowserStack
* **Tests fonctionnels** : Vérification de chaque exigence du cahier des charges
* **Tests d’intégration** : Vérification de la communication entre modules
* **Tests de performance** : Simulation de charge, temps de réponse
* **Tests de sécurité** : SSL, authentification, API Stripe, gestion des rôles
* **Tests de compatibilité/responsive** : Desktop, tablette, mobile, principaux navigateurs
* **Environnement de préproduction** : Identique à la production

**Tableau du plan de test**

| **Périmètre testé** | **Date** | **Description du test** | **Résultat attendu** | **Résultat obtenu** | **Réussite/Échec** | **Testé par** |
| --- | --- | --- | --- | --- | --- | --- |
| Authentification utilisateur | 2025-04-10 | Connexion/déconnexion (client, restaurateur, admin) | Accès sécurisé, redirection correcte, session isolée | À renseigner | À renseigner | QA |
| Gestion des restaurants | 2025-04-11 | CRUD restaurant via dashboard restaurateur | Opérations sans erreur, affichage immédiat des changements | À renseigner | À renseigner | QA |
| Accès restaurateur | 2025-04-12 | Dashboard, gestion menus, commandes, réservations | Accès réservé, outils visibles uniquement pour restaurateur | À renseigner | À renseigner | QA |
| Accès client | 2025-04-12 | Consultation menus, réservation, commande | Accès restreint, affichage personnalisé selon le restaurant | À renseigner | À renseigner | QA |
| Gestion menus/cartes | 2025-04-13 | CRUD catégories/articles, affichage dynamique | Modifications visibles, catégorisation correcte | À renseigner | À renseigner | QA |
| Gestion des commandes | 2025-04-14 | Passer/annuler commande, suivi en temps réel | Commande enregistrée, notification cuisine, historique mis à jour | À renseigner | À renseigner | QA |
| Génération QR Code | 2025-04-15 | Génération, scan, redirection vers menu/commande | QR code lisible, redirection correcte | À renseigner | À renseigner | QA |
| Responsive design | 2025-04-16 | Tests sur desktop, tablette, mobile, Chrome, Firefox, Safari | Interface cohérente, aucune perte de fonctionnalité | À renseigner | À renseigner | QA |
| Personnalisation graphique | 2025-04-17 | Modification couleurs, logos, polices via interface | Rendu conforme à la charte définie | À renseigner | À renseigner | QA |
| Paiement en ligne (Stripe) | 2025-04-18 | Transaction test, validation, génération reçu | Paiement validé, reçu généré, sécurité respectée | À renseigner | À renseigner | QA |
| CI/CD et déploiement | 2025-04-19 | Migrations, cache, optimisation lors d’un déploiement | Scripts CI déclenchés, déploiement sans interruption, rollback possible | À renseigner | À renseigner | DevOps |
| Sécurité SSL/API | 2025-04-20 | Vérification HTTPS, sécurité Stripe, rôles | Aucune faille détectée, accès sécurisé, logs d’accès complets | À renseigner | À renseigner | Sécurité |

**Exécution, reporting et validation**

* Chaque test est réalisé par l’équipe QA selon le planning.
* Les anomalies sont tracées dans l’outil de suivi (Jira, GitHub Issues…).
* Un rapport de test est généré à chaque fin de cycle.
* La mise en production n’est validée que si tous les tests critiques sont passés et les anomalies bloquantes corrigées.

**Conclusion**

Ce plan de test garantit la conformité de l’application 1ClickAllEat avec les attentes fonctionnelles et techniques du projet. Il sera mis à jour à chaque évolution majeure du produit ou retour utilisateur.

Extrait de code 🡪

**1. Réservation d’une table**

php

CopyInsert

// Extrait de ReservationController.php

public function store(Request $request, Restaurant $restaurant)

{

$validated = $request->validate([

'table\_id' => ['required', 'exists:restaurant\_tables,id'],

'date\_reservation' => ['required', 'date', 'after:now'],

'heure\_reservation' => ['required', 'date\_format:H:i'],

]);

$table = RestaurantTable::findOrFail($validated['table\_id']);

if (!Auth::check()) {

return redirect()->back()->with('error', 'Vous devez être connecté pour réserver une table.');

}

$dateTime = $validated['date\_reservation'] . ' ' . $validated['heure\_reservation'];

$exists = $table->reservations()

->where('date\_reservation', $dateTime)

->whereIn('status', ['pending', 'confirmed'])

->exists();

if ($exists) {

return redirect()->back()->with('error', 'Cette table est déjà réservée pour ce créneau.');

}

$reservation = Reservation::create([

'user\_id' => Auth::id(),

'restaurant\_id' => $restaurant->id,

'table\_id' => $table->id,

'date\_reservation' => $dateTime,

'status' => 'pending',

]);

// Création automatique d'une commande liée à la réservation

$order = new \App\Models\FoodOrder([

'client\_id' => Auth::id(),

'restaurant\_id' => $restaurant->id,

'status' => 'pending',

'total\_price' => 0

]);

$order->save();

}

**Explication détaillée**  
Cette méthode permet à un utilisateur connecté de réserver une table :

* Elle valide que la table existe et que la date/heure sont correctes.
* Elle vérifie que l’utilisateur est bien connecté.
* Elle contrôle que la table n’est pas déjà réservée à ce créneau.
* Si tout est OK, elle crée la réservation et génère automatiquement une commande vide associée, facilitant le workflow côté client et restaurateur.

**2. Affichage différencié des commandes et réservations (client vs restaurateur)**

php

CopyInsert

// Extrait de FoodOrderController.php

public function index()

{

if (Auth::user()->role->name === 'client') {

$orders = Auth::user()->clientFoodOrders()->with(['restaurant', 'items'])->latest()->get();

$reservations = \App\Models\Reservation::with(['restaurant', 'table'])

->where('user\_id', Auth::id())

->orderBy('date\_reservation', 'asc')

->get();

return view('orders.index', compact('orders', 'reservations'));

} else {

$orders = Auth::user()->restaurantFoodOrders()->with(['client', 'items'])->latest()->get();

return view('orders.index', compact('orders'));

}

}

**Explication détaillée**  
Cette méthode affiche les commandes et réservations dans l’espace utilisateur :

* Si l’utilisateur est un **client** : il voit ses propres commandes et réservations.
* Si c’est un **restaurateur** : il voit toutes les commandes passées dans son restaurant.
* Cela permet une expérience personnalisée et sécurisée selon le rôle, en limitant l’accès aux bonnes informations.

**3. Passage d’une commande (création d’une commande à partir du panier)**

php

CopyInsert

// Extrait de FoodOrderController.php

public function store(Request $request)

{

$validated = $request->validate([

'restaurant\_id' => ['required', 'exists:restaurants,id'],

'items' => ['required', 'array'],

]);

$itemIds = array\_keys($validated['items']);

$items = Item::findMany($itemIds);

$selectedItems = [];

foreach ($items as $item) {

$qty = intval($validated['items'][$item->id]['quantity'] ?? 0);

if ($qty > 0) {

$selectedItems[] = [

'id' => $item->id,

'quantity' => $qty,

'price' => $item->effective\_price,

];

}

}

if (count($selectedItems) === 0) {

return back()->with('error', 'Veuillez sélectionner au moins un plat.');

}

$restaurant = Restaurant::findOrFail($validated['restaurant\_id']);

// Création de la commande

$order = FoodOrder::create([

'client\_id' => Auth::id(),

'restaurant\_id' => $restaurant->id,

'status' => 'pending',

'total\_price' => array\_sum(array\_map(fn($i) => $i['quantity'] \* $i['price'], $selectedItems)),

]);

// Ajout des plats à la commande (OrderItem)

foreach ($selectedItems as $item) {

$order->items()->create([

'item\_id' => $item['id'],

'quantity' => $item['quantity'],

'price' => $item['price'],

]);

}

return redirect()->route('orders.index')->with('success', 'Commande enregistrée !');

}

**Explication détaillée**

* Cette méthode gère la validation du panier et la création d’une commande.
* Elle vérifie que le panier contient bien des plats, puis crée la commande avec le total calculé.
* Chaque plat sélectionné est ajouté à la commande via la relation

items()

.

* Cela garantit la cohérence des données et permet un suivi précis des commandes côté cuisine.

**4. Annulation d’une réservation**

php

CopyInsert

// Extrait de ReservationController.php

public function cancel(Reservation $reservation)

{

// Vérifie que seul le propriétaire de la réservation peut l’annuler

if ($reservation->user\_id !== Auth::id()) {

abort(403);

}

$reservation->status = 'cancelled';

$reservation->save();

return redirect()->back()->with('success', 'Réservation annulée.');

}

**Explication détaillée**

* Cette méthode permet à un utilisateur d’annuler sa propre réservation.
* Elle vérifie l’autorisation (propriétaire uniquement).
* Elle change le statut en “cancelled” et sauvegarde la modification.
* Cela protège la logique métier et évite les suppressions accidentelles ou malveillantes.

**5. Ajout d’un plat au panier (extrait simplifié)**

php

CopyInsert

// Extrait de CartController.php

public function add(Request $request, $itemId)

{

$item = Item::findOrFail($itemId);

$cart = session()->get('cart', []);

$cart[$itemId] = [

'name' => $item->name,

'quantity' => ($cart[$itemId]['quantity'] ?? 0) + 1,

'price' => $item->effective\_price,

];

session(['cart' => $cart]);

return back()->with('success', 'Plat ajouté au panier.');

}

**Explication détaillée**

* Cette méthode ajoute un plat au panier stocké en session.
* Si le plat existe déjà, on incrémente la quantité.
* Le panier est persistant tant que la session de l’utilisateur reste active.
* Cela permet une expérience utilisateur fluide et rapide, sans base de données pour le panier temporaire.

Diagramme de gantt 🡪

