Page de garde

Table des matières

[Annexe 9 – 1 B 4](#_Toc196751979)

[Annexe 10 – B 8](#_Toc196751980)

[1ClickAllEat 10](#_Toc196751981)

[Cahier des charges 10](#_Toc196751982)

[Diagramme de Gantt : 14](#_Toc196751983)

[1ClickAllEat 15](#_Toc196751984)

[UML 15](#_Toc196751985)

[1. Diagramme de contexte 15](#_Toc196751986)

[2. Diagramme de packages 15](#_Toc196751987)

[15](#_Toc196751988)

[3. Diagramme Use\_Case 16](#_Toc196751989)

[4. Cartographie technique 17](#_Toc196751990)

[5. Diagramme de classe 17](#_Toc196751991)

[1ClickAllEat 18](#_Toc196751992)

[Merise – Modélisation BDD 18](#_Toc196751993)

[1. Dictionnaire de données 18](#_Toc196751994)

[2. MCD (Modelé Conceptuel de données) 20](#_Toc196751995)

[20](#_Toc196751996)

[3. MLD (Modèle Logique de données) 20](#_Toc196751997)

[4. MPD (Modèle Physique de données) 21](#_Toc196751998)

[21](#_Toc196751999)

[1ClickAllEat 22](#_Toc196752000)

[Extraits de code commentés 22](#_Toc196752001)

[1. Réservation d’une table 22](#_Toc196752002)

[2. Affichage différencié des commandes et réservations (client vs restaurateur) 23](#_Toc196752003)

[3. Passage d’une commande (création d’une commande à partir du panier) 24](#_Toc196752004)

[4. Annulation d’une réservation 26](#_Toc196752005)

[5. Ajout d’un plat au panier (extrait simplifié) 27](#_Toc196752006)

[1ClickAllEat 29](#_Toc196752007)

[Plan de test qualité 29](#_Toc196752008)

[Objectif 29](#_Toc196752009)

[Fonctionnalités MVP 29](#_Toc196752010)

[Fonctionnalités MLP 29](#_Toc196752011)

[Contraintes techniques 29](#_Toc196752012)

[Méthodologie 30](#_Toc196752013)

[Tableau du plan de test 30](#_Toc196752014)

[Exécution, reporting et validation 31](#_Toc196752015)

[Conclusion 31](#_Toc196752016)

[1ClickAllEat 32](#_Toc196752017)

[Guide utilisateur 32](#_Toc196752018)

[Section 1: Interface Restaurateur 32](#_Toc196752019)

[Section 2: Interface Client 33](#_Toc196752020)

[Section 1: Interface Restaurateur 36](#_Toc196752021)

[LDAP – Telora 42](#_Toc196752022)

[Cahier des charges détaillé 42](#_Toc196752023)

[Diagramme de gantt 47](#_Toc196752024)

[LDAP - Telora 48](#_Toc196752025)

[UML 48](#_Toc196752026)

[1. Diagramme de contexte 48](#_Toc196752027)

[2. Diagramme de package 48](#_Toc196752028)

[Diagramme use\_case 49](#_Toc196752029)

[Cartographie technique 50](#_Toc196752030)

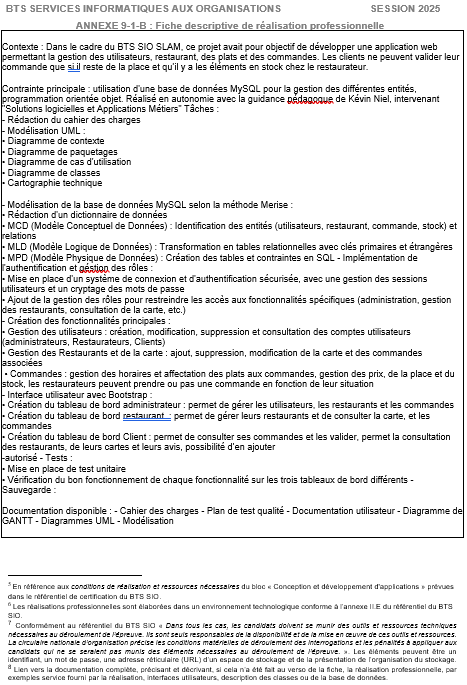
[Diagramme de classe 50](#_Toc196752031)

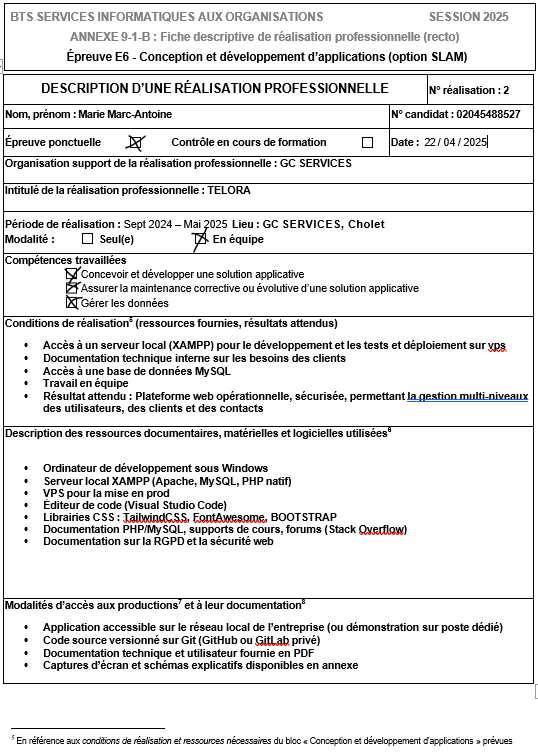
[LDAP – Telora 51](#_Toc196752032)

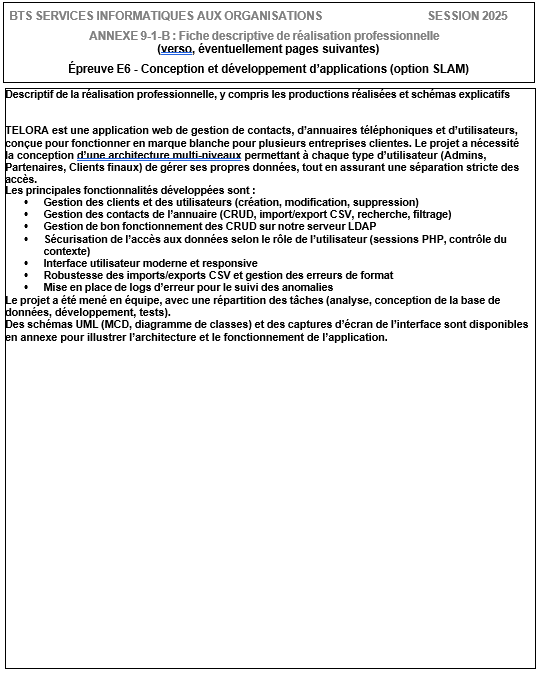
[Merise – Modélisation BDD 51](#_Toc196752033)

[1. Dictionnaire de données 51](#_Toc196752034)

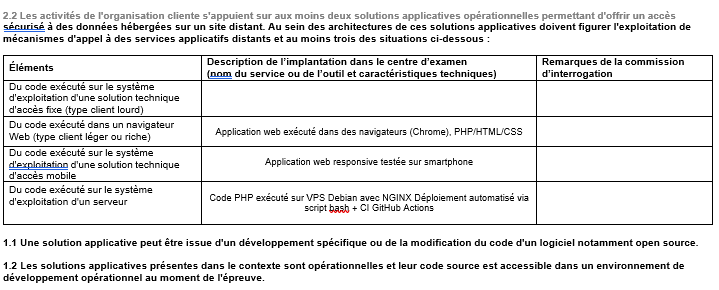
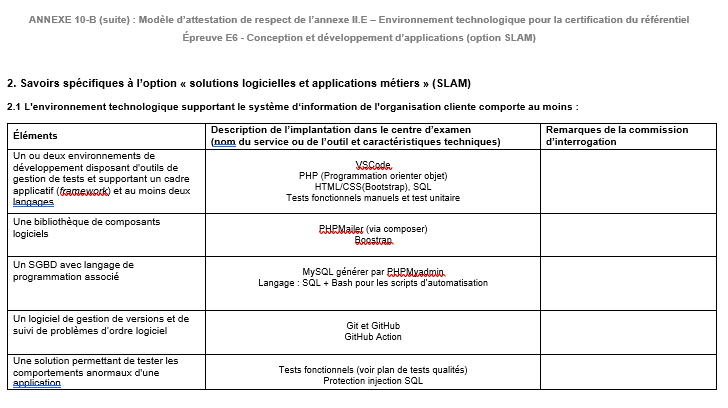
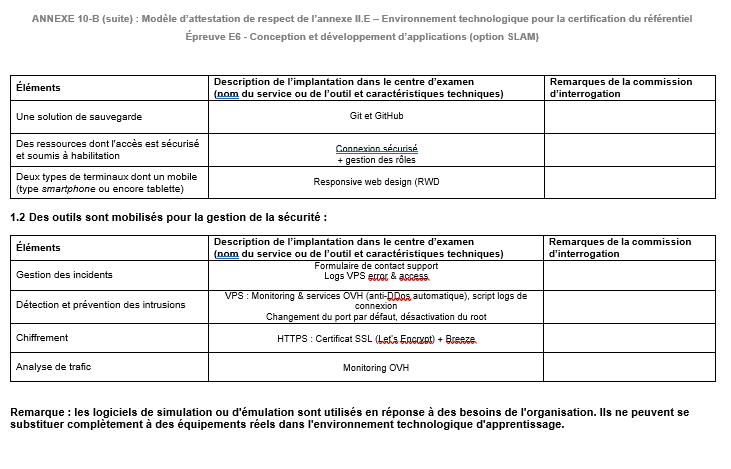
# Annexe 9 – 1 B







# Annexe 10 – B



# 1ClickAllEat

## Cahier des charges

**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**

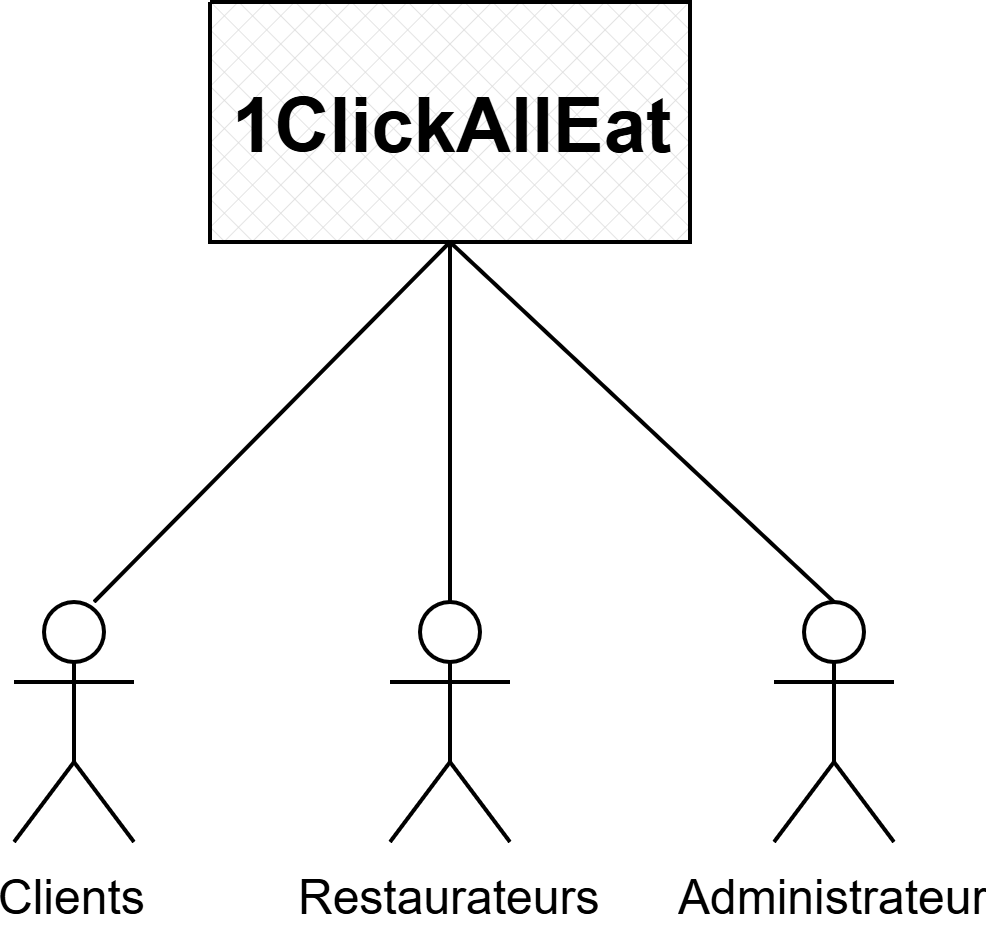
* **Framework de développement**
  + Laravel (Back-end)
  + Interface utilisateur moderne et intuitive
* **Déploiement**
  + Sur un nom de domaine dédié
  + Hébergement sécurisé et performant
* **Intégration continue**
  + Automatisation des migrations de base de données
  + Gestion du vidage des caches 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

# Diagramme de Gantt :

# 1ClickAllEat

## UML

### 1. Diagramme de contexte



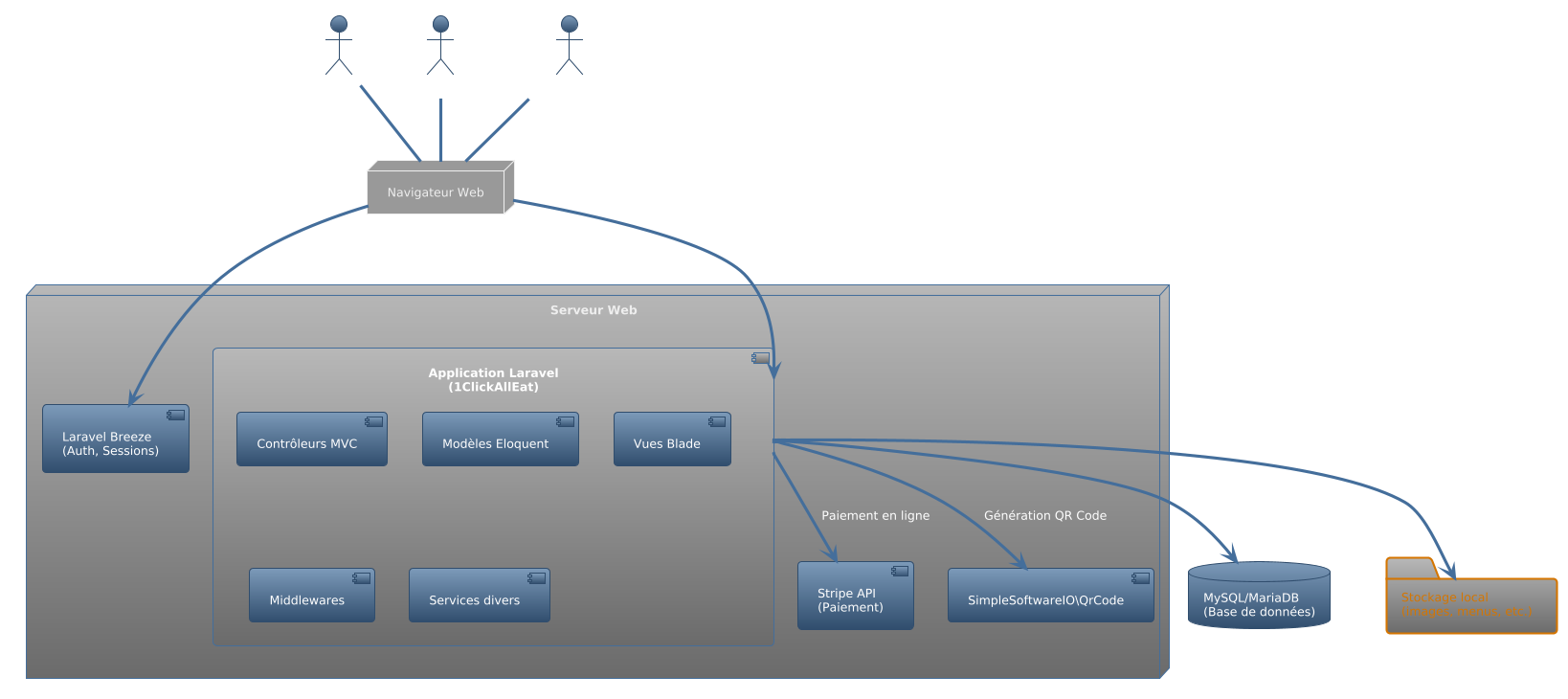
### 2. Diagramme de packages

## 

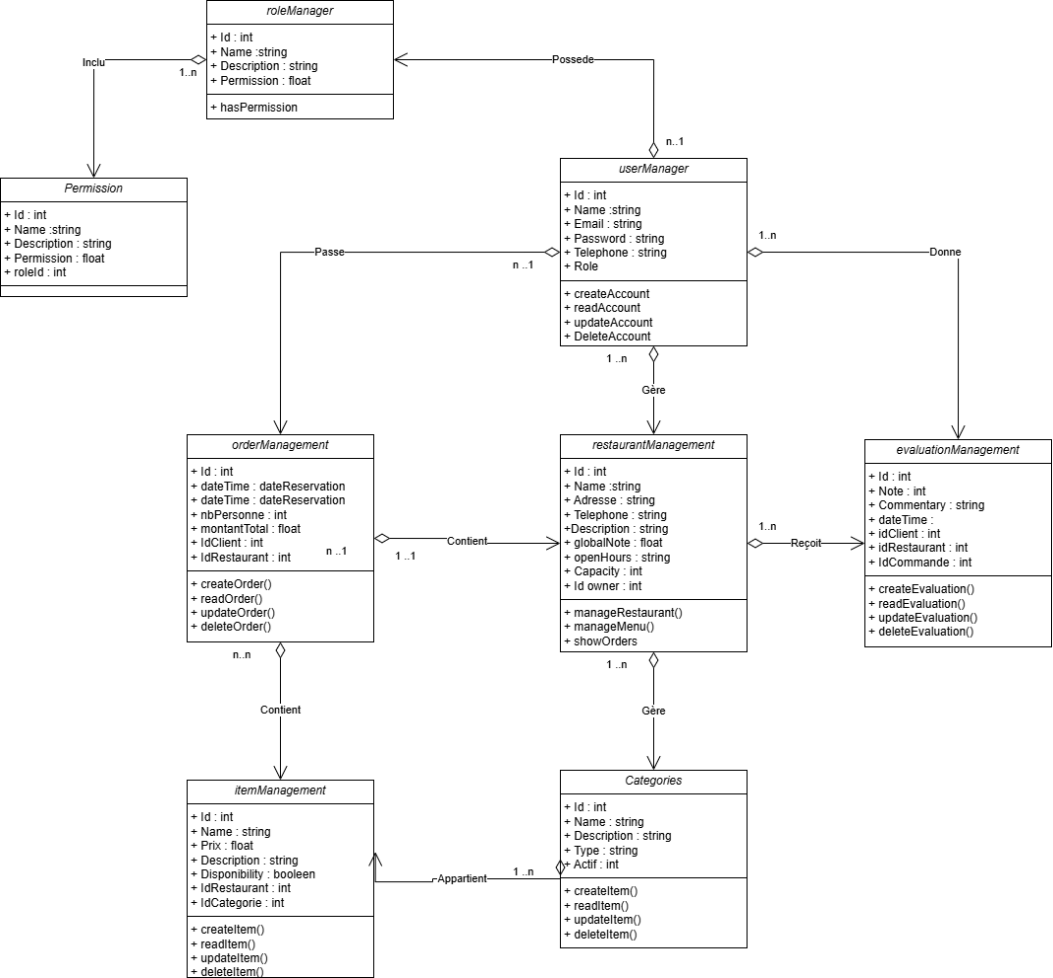
## 

### 3. Diagramme Use\_Case

### 4. Cartographie technique



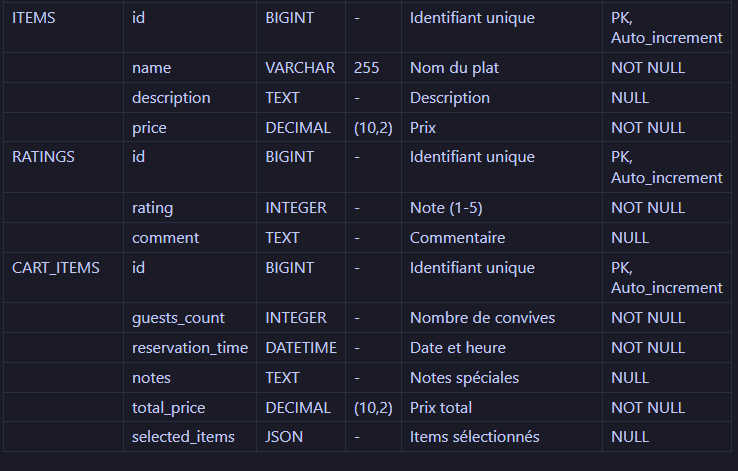
### 5. Diagramme de classe



# 1ClickAllEat

## Merise – Modélisation BDD

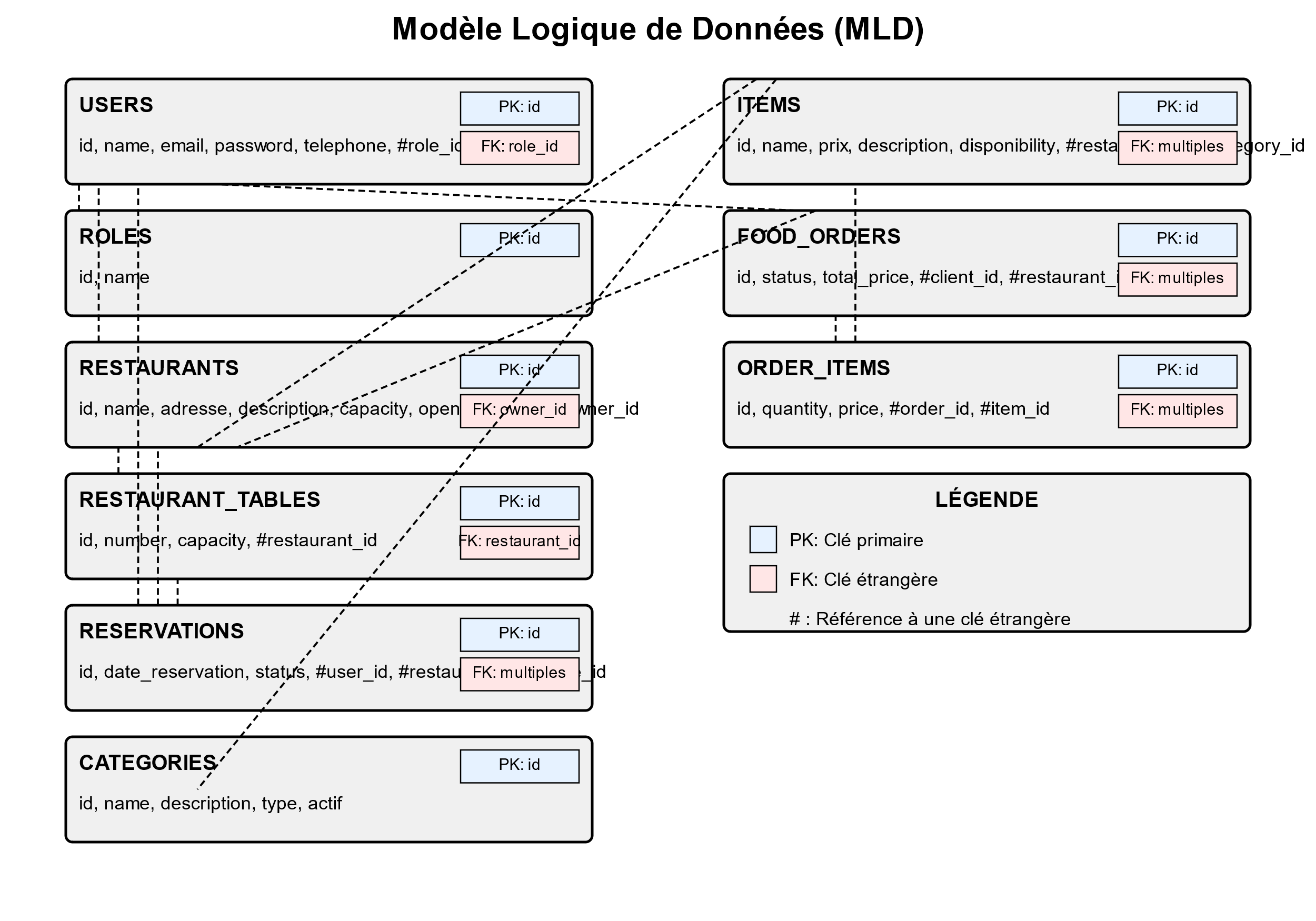
### 1. Dictionnaire de données



### 2. MCD (Modelé Conceptuel de données)

### 

### 3. MLD (Modèle Logique de données)



### 4. MPD (Modèle Physique de données)

### 

# 1ClickAllEat

## Extraits de code commentés

### 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.

# 1ClickAllEat

## Plan de test qualité

### 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, responsive, personnalisation graphique, 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,
* 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, 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

* Les anomalies sont tracées dans l’outil de suivi (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

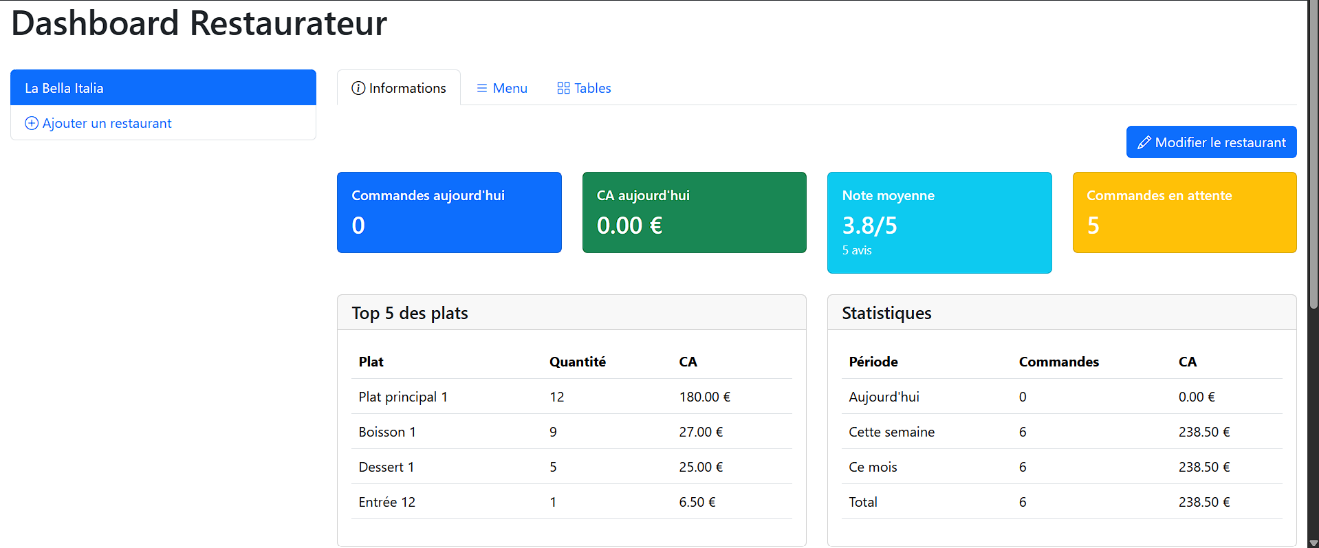
Ce plan de test garantit la conformité de l’application 1ClickAllEat avec les attentes fonctionnelles et techniques du projet. Il sera mis à jour à chaque évolution majeure du produit ou retour utilisateur.

# 1ClickAllEat

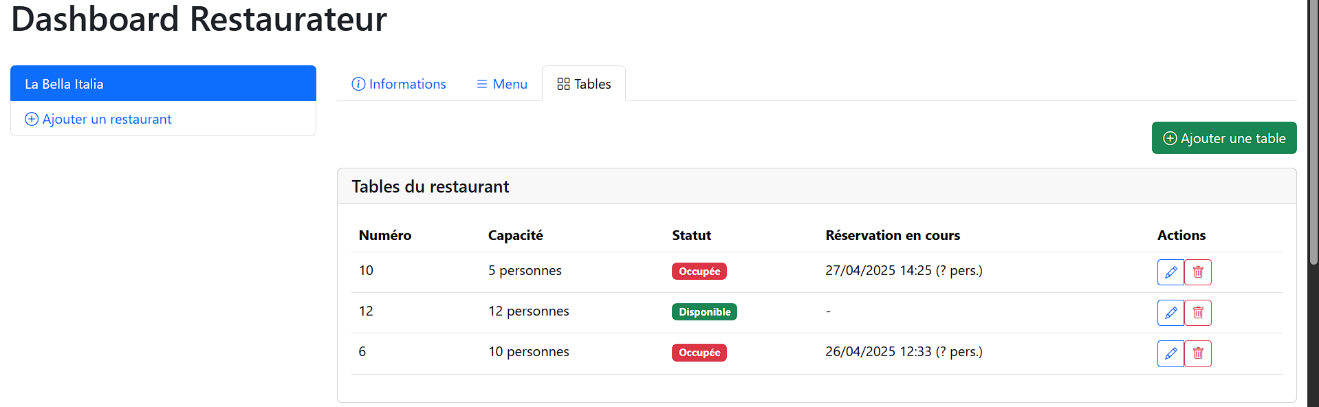
## Guide utilisateur

### Section 1: Interface Restaurateur

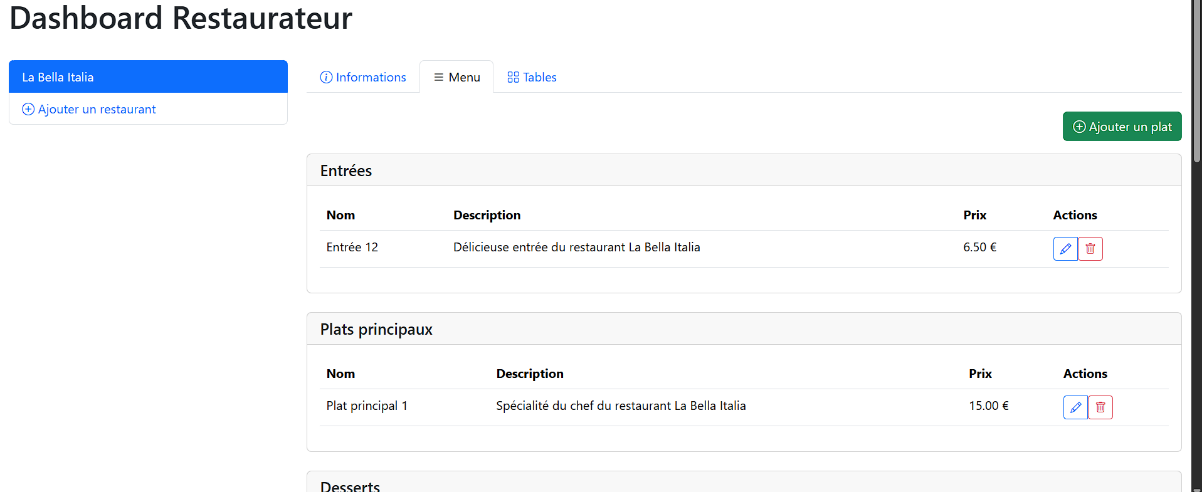
1. Dashboard principal



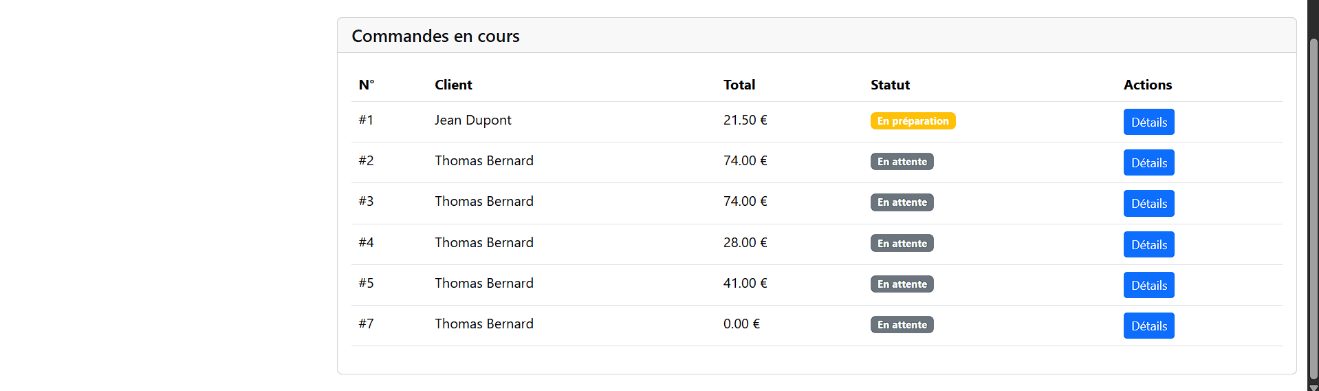
1. Gestion des tables



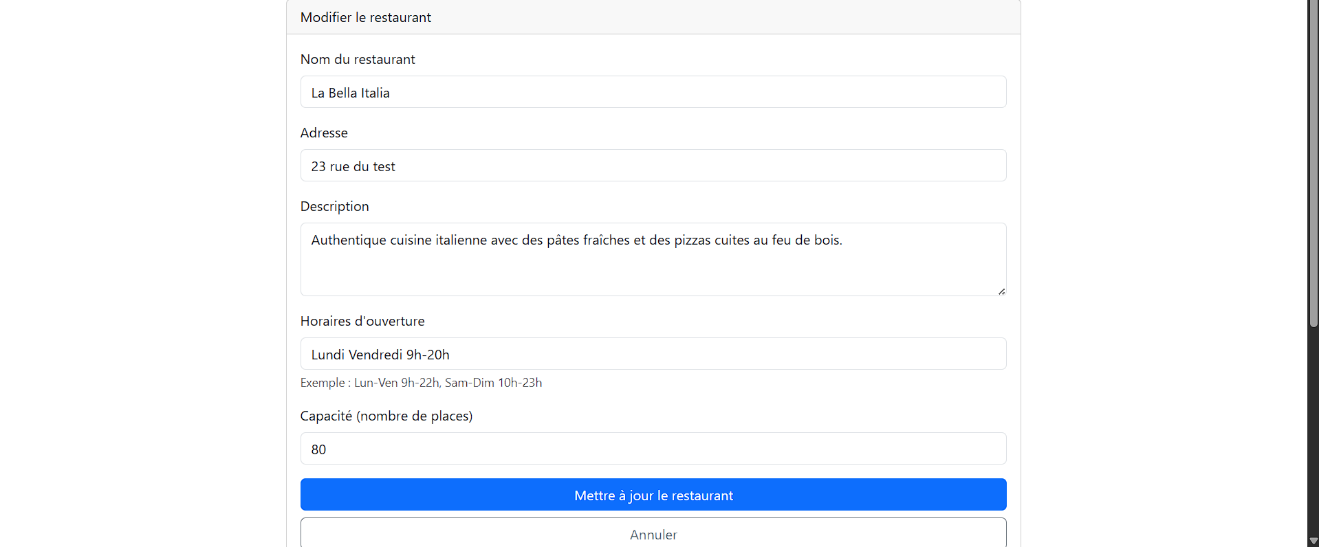
1. Gestion du menu



1. Gestion des commandes

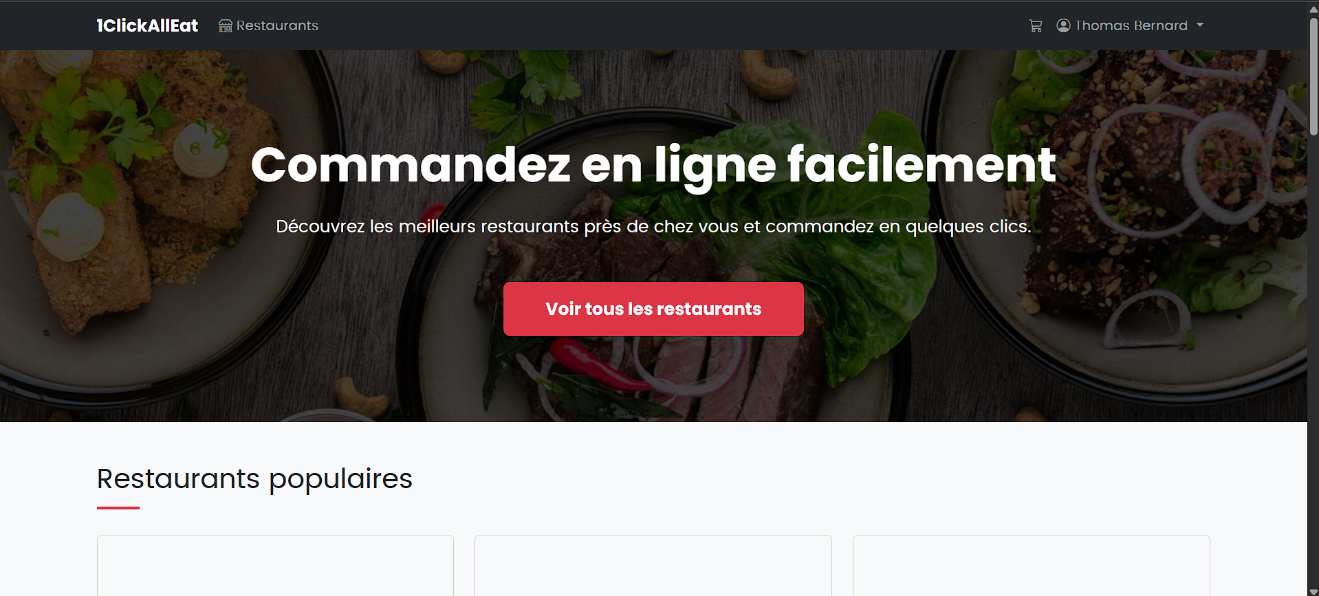


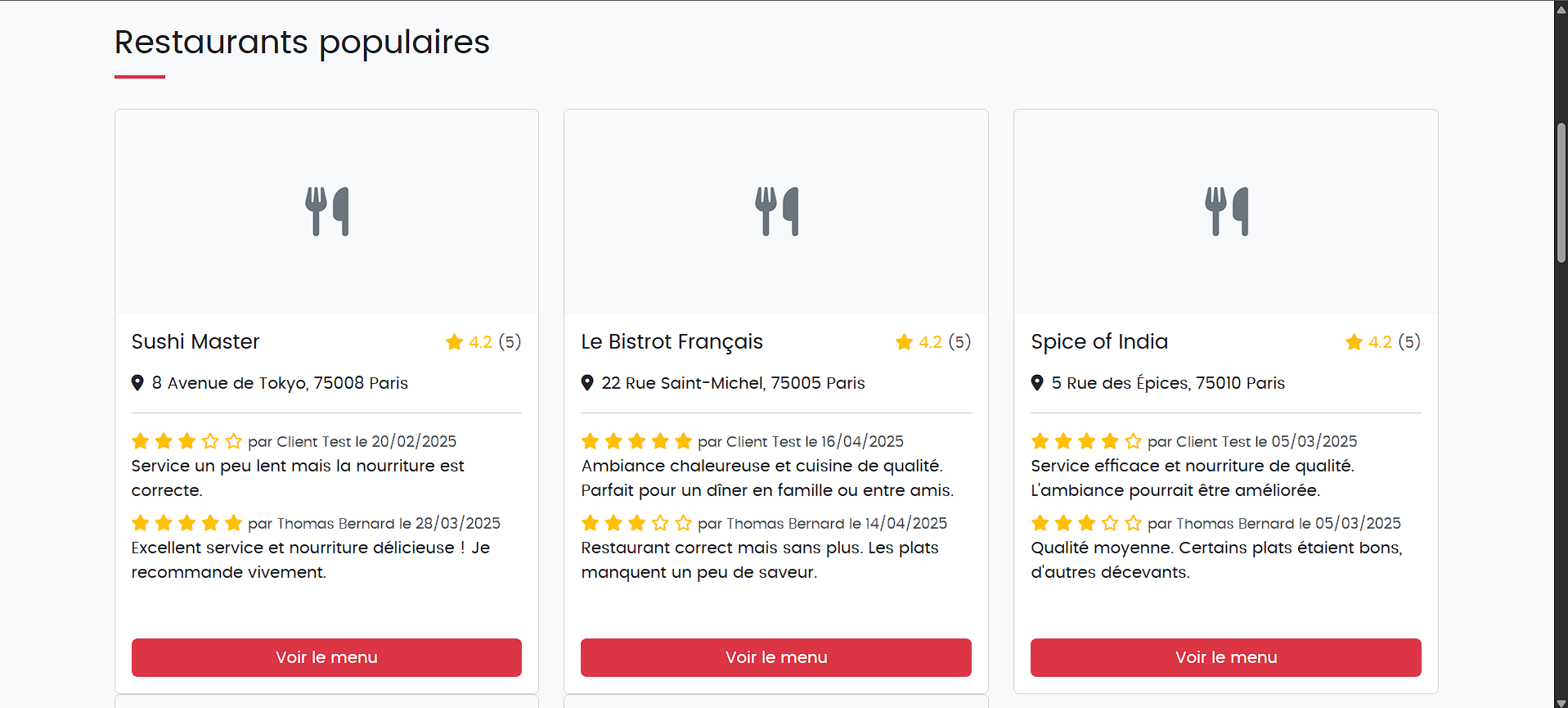
1. Paramètres du restaurant

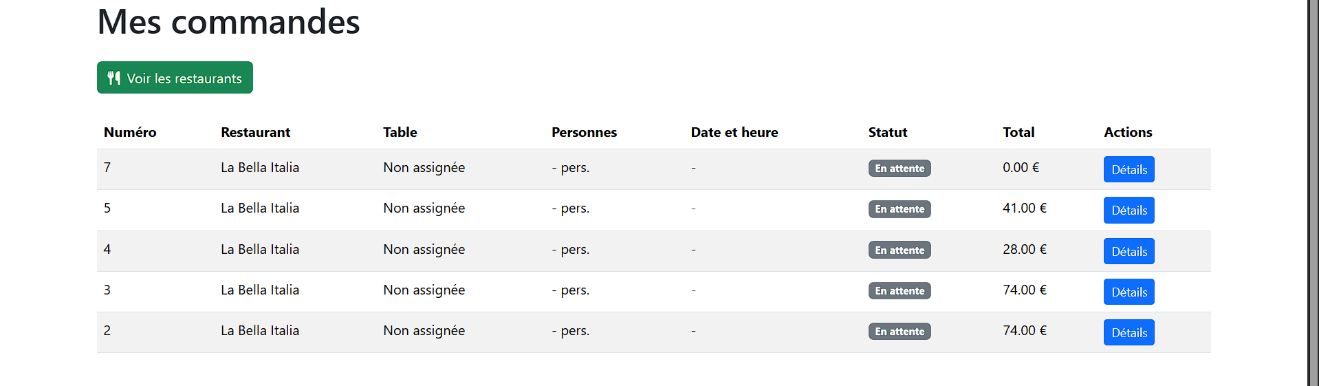
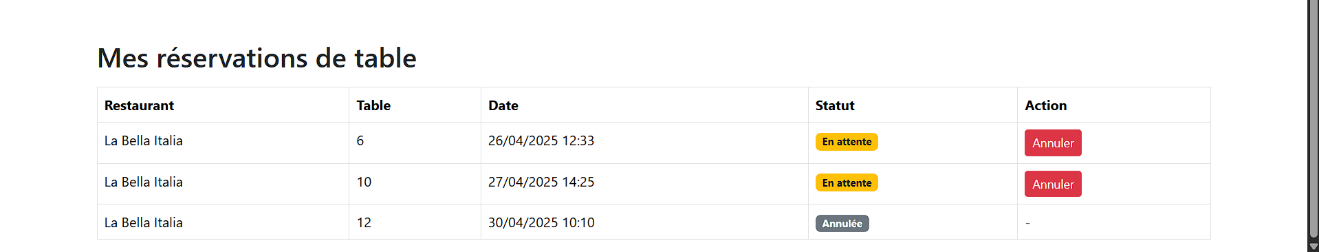


### Section 2: Interface Client

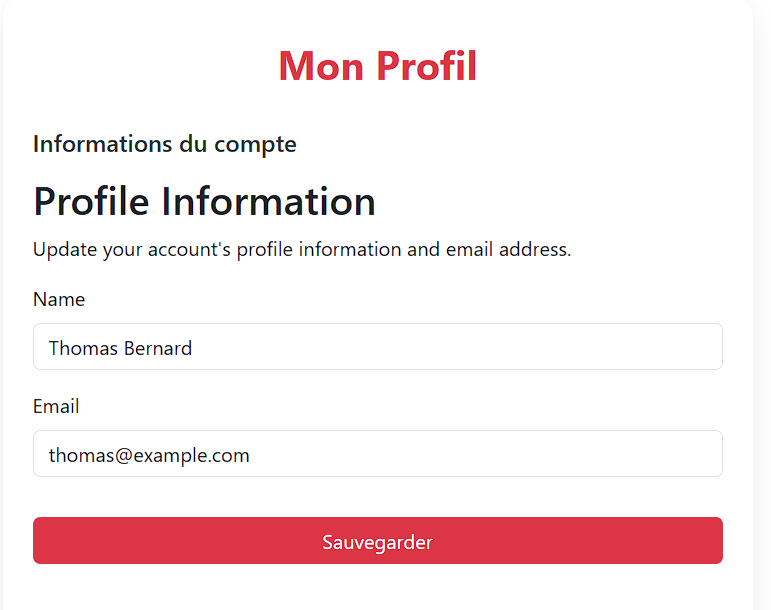
1. Parcourir les restaurants

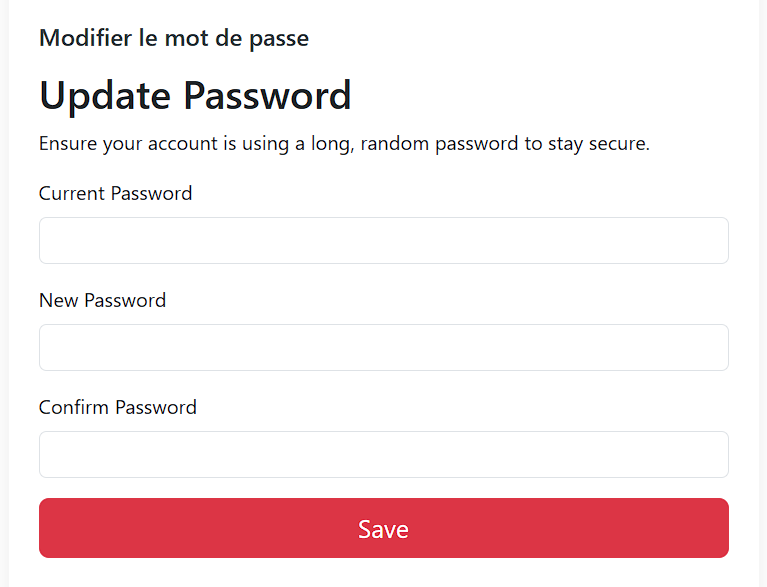


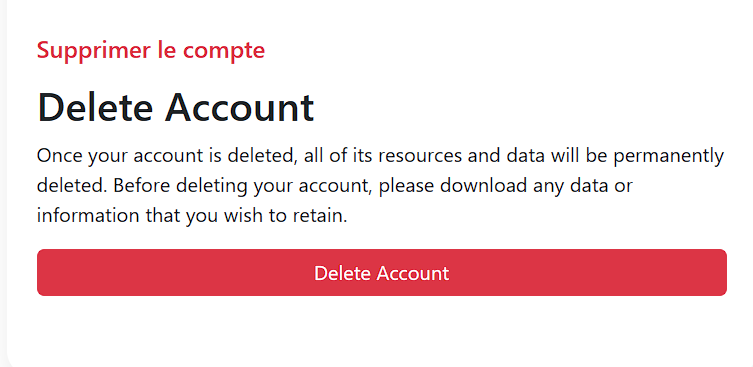
1. Commander en ligne
2. Suivi des commandes et réservations



1. Profil utilisateur







### Section 1: Interface Restaurateur

**1. Dashboard principal**

Le Dashboard Restaurateur est votre centre de contrôle principal pour gérer votre établissement sur la plateforme 1ClickAllEat. Il affiche:

* Le nom de votre restaurant (ex: La Bella Italia)
* Navigation principale: Informations, Menu, Tables
* Statistiques essentielles:
  + Commandes aujourd'hui (nombre)
  + Chiffre d'affaires (CA) du jour
  + Note moyenne et nombre d'avis (ex: 3.8/5 basé sur 5 avis)
  + Commandes en attente (nombre)
* Top 5 des plats les plus vendus avec quantités et CA générés
* Statistiques de performance par période (aujourd'hui, semaine, mois, total)

Le bouton "Modifier le restaurant" en haut à droite vous permet d'accéder rapidement aux paramètres de votre établissement.

**2. Gestion des tables**

L'onglet "Tables" vous permet de visualiser et gérer l'ensemble des tables de votre restaurant:

* Liste complète avec: Numéro, Capacité, Statut (Disponible/Occupée)
* Affichage des réservations en cours pour les tables occupées
* Pour chaque table, vous pouvez:
  + Modifier les informations (icône crayon)
  + Supprimer la table (icône poubelle)
* Bouton "Ajouter une table" pour créer une nouvelle table avec:
  + Numéro de table
  + Capacité (nombre de personnes)

Cette vue vous aide à optimiser l'occupation de votre restaurant et à suivre les réservations en temps réel.

**3. Gestion du menu**

L'onglet "Menu" vous permet de créer et gérer l'ensemble de votre carte:

* Organisation par catégories: Entrées, Plats principaux, Desserts, etc.
* Pour chaque plat, vous visualisez:
  + Nom du plat
  + Description
  + Prix
  + Actions (modifier/supprimer)
* Bouton "Ajouter un plat" pour enrichir votre menu avec:
  + Nom du plat
  + Description détaillée
  + Prix en euros
  + Catégorie (à sélectionner dans le menu déroulant)

Gardez votre menu à jour pour offrir la meilleure expérience à vos clients.

**4. Gestion des commandes**

La section "Commandes en cours" vous permet de suivre toutes les commandes reçues:

* Numéro de commande
* Nom du client
* Montant total
* Statut (En préparation, En attente)
* Action "Détails" pour consulter le détail complet

Cette vue vous aide à traiter efficacement les commandes et à suivre leur progression.

**5. Statistiques et performances**

Le dashboard principal affiche des indicateurs clés de performance:

* Top 5 des plats les plus populaires avec:
  + Nom du plat
  + Quantité vendue
  + Chiffre d'affaires généré
* Récapitulatif financier par période:
  + Aujourd'hui (0 commandes, 0.00 €)
  + Cette semaine (6 commandes, 238.50 €)
  + Ce mois (6 commandes, 238.50 €)
  + Total (6 commandes, 238.50 €)

Ces données vous permettent d'analyser les tendances et d'optimiser votre offre.

**6. Paramètres du restaurant**

Le formulaire "Modifier le restaurant" vous permet de mettre à jour les informations essentielles:

* Nom du restaurant
* Adresse complète
* Description détaillée
* Horaires d'ouverture (format suggéré: Lun-Ven 9h-22h, Sam-Dim 10h-23h)
* Capacité totale (nombre de places)

Gardez ces informations à jour pour que les clients puissent trouver facilement votre établissement.

**Section 2: Interface Client**

**1. Parcourir les restaurants**

La page d'accueil client présente:

* Bannière principale "Commandez en ligne facilement"
* Courte description: "Découvrez les meilleurs restaurants près de chez vous et commandez en quelques clics"
* Bouton "Voir tous les restaurants" pour accéder au catalogue complet
* Section "Restaurants populaires" présentant les établissements les plus appréciés
* Filtres de recherche avec:
  + Nom du restaurant
  + Note minimale
  + Adresse/Ville (Ex: Paris)
  + Tri (A-Z)

Cette interface permet aux clients de découvrir rapidement des restaurants qui correspondent à leurs préférences.

**2. Réserver une table**

La section "Mes réservations de table" permet aux clients de:

* Visualiser leurs réservations existantes avec:
  + Nom du restaurant
  + Numéro de table
  + Date et heure
  + Statut (En attente, Annulée)
  + Action possible (Annuler)
* Réserver de nouvelles tables dans les restaurants participants

Les réservations sont synchronisées avec le dashboard restaurateur, permettant une gestion efficace des places.

**3. Commander en ligne**

Les clients peuvent passer commande directement depuis la plateforme:

* Parcourir le menu du restaurant organisé par catégories
* Ajouter des plats à leur panier
* ²Définir la quantité pour chaque article
* Voir le total de leur commande en temps réel
* Valider et payer en ligne

**4. Suivi des commandes et réservations**

La section "Mes commandes" permet aux clients de:

* Voir l'historique complet de leurs commandes
* Consulter le statut de chaque commande (En attente, En préparation)
* Accéder aux détails complets (articles, prix, restaurant)
* Voir le montant total de chaque commande
* Les commandes sont numérotées pour faciliter le suivi

Cette transparence améliore l'expérience client et réduit les demandes d'information.

**5. Profil utilisateur**

Le menu utilisateur (accessible depuis l'avatar en haut à droite) permet d'accéder à:

* "Mon profil": informations personnelles
* "Mes commandes": historique des commandes
* "Déconnexion": quitter la session en cours

Les clients peuvent ainsi gérer facilement leur compte et accéder à leur historique.

# LDAP – Telora

## Cahier des charges détaillé

**1. Contexte et Présentation générale**

Telora est une application web de gestion de contacts, d’annuaires téléphoniques et d’utilisateurs, pensée pour un fonctionnement en marque blanche. Elle vise à permettre à différents types d’utilisateurs (Admins, Partenaires, Clients finaux) de gérer leurs propres données, via une interface moderne, responsive et sécurisée.

Le projet est développé en PHP/MySQL côté serveur, avec une interface HTML/CSS/JS enrichie par TailwindCSS et FontAwesome.

**2. Objectifs principaux**

- Centraliser la gestion des contacts et des utilisateurs pour des clients variés (partenaires, entreprises, utilisateurs finaux).

- Offrir une séparation stricte des accès selon le rôle.

- Proposer une expérience utilisateur fluide, moderne et sécurisée.

- Permettre l’import/export de contacts au format CSV, avec robustesse face aux erreurs de format.

**3. Public cible**

- Administrateurs système (super-admins)

- Partenaires (entreprises clientes de Telora)

- Clients finaux (utilisateurs des partenaires)

**4. Expression des besoins et fonctionnalités**

**4.1. Gestion multi-niveaux et rôles**

- Admins : gestion globale des partenaires, clients, utilisateurs, et contacts.

- Partenaires : gestion de leurs propres clients finaux et de leurs contacts.

- Clients finaux : gestion de leurs propres contacts et informations.

**4.2. Gestion des entités**

- CRUD complet (création, lecture, modification, suppression) pour :

- Clients

- Utilisateurs

- Contacts de l’annuaire

- Import/export CSV des contacts (encodage UTF-8, format compatible)

- Recherche, filtrage, pagination des contacts

**4.3. Sécurité et navigation**

- Séparation stricte des accès selon le rôle (aucune fuite de données entre clients)

- Gestion du contexte de navigation via la session PHP

- Redirection automatique vers la page de connexion en cas d’accès non autorisé

- Logs d’erreur activés pour le suivi des bugs (sans fuite d’informations sensibles)

- Jamais de modification des variables de session à partir de l’URL côté client

- Vérification stricte des droits d’accès à chaque page sensible

**4.4. Interface utilisateur**

- Interface responsive, moderne, centrée, avec header harmonisé sur toutes les pages

- Utilisation de containers pour centrer et limiter la largeur du contenu (max 1200px)

- Design cohérent entre tous les formulaires et tableaux

- Utilisation de boutons modernes, feedback utilisateur (succès/erreur), modals pour les actions sensibles

- Intégration de TailwindCSS et FontAwesome

**4.5. Points d’attention**

- Maintien du contexte utilisateur lors de la navigation (pas de fuite de session)

- Cohérence visuelle et UX sur toutes les pages (header, boutons, tableaux, formulaires)

- Robustesse de la gestion des imports/exports CSV (gestion des erreurs de format)

- Tests sur différents rôles et scénarios d’accès

**5. Contraintes techniques**

- Technologies : PHP (backend), MySQL (base de données), HTML/CSS/JS (frontend), TailwindCSS, FontAwesome

- Organisation : séparation claire entre la logique métier (PHP, dossier classes/api), les vues (HTML, dossiers admin, annuaire, clientlist, utilisateurdetail…), et le style (CSS, partials)

- Sécurité : sessions PHP, contrôle d’accès, logs d’erreur, protection contre la modification des sessions côté client

- Ergonomie : centrage du contenu, largeur limitée, responsive design

- Tests : activation des logs, rédaction d’un cahier de tests fonctionnels, tests manuels/automatisés selon disponibilité

**6. Architecture technique (modules/dossiers)**

- admin/ : gestion des administrateurs et des partenaires

- annuaire/ : gestion des contacts et de l’annuaire téléphonique

- clientlist/ et clientdetail/ : gestion des clients

- utilisateurdetail/ : gestion des utilisateurs

- api/ : endpoints pour l’interface ou l’intégration externe

- classes/ : logique métier et objets PHP

- config/, database/ : configuration et scripts SQL

- login/ : authentification et gestion des sessions

- partials/ : fragments d’interface (header, footer…)

- utils/ : utilitaires divers

- docs/, Documentation\_LDAP/ : documentation technique et d’intégration

**7. Livrables attendus**

- Code source complet (PHP, CSS, JS, SQL)

- Documentation technique (structure, installation, sécurité)

- Cahier de tests fonctionnels (scénarios d’accès, sécurité, import/export)

- Fichiers d’exemple pour l’import/export CSV

**8. Déploiement et maintenance**

- Déploiement sur un serveur compatible PHP/MySQL, avec configuration sécurisée (HTTPS recommandé)

- Scripts d’installation et de migration de base de données fournis

- Procédures de sauvegarde et de restauration des données à documenter

**9. Sécurité et RGPD**

- Protection des données personnelles (sessions, accès, logs limités)

- Conformité RGPD à assurer (mentions légales, droits des utilisateurs)

- Certificat SSL recommandé pour toutes les communications

**10. Tests et validation**

- Rédaction d’un cahier de tests couvrant tous les rôles et scénarios critiques

- Tests d’import/export CSV avec gestion des erreurs

- Validation finale par démonstration auprès du tuteur

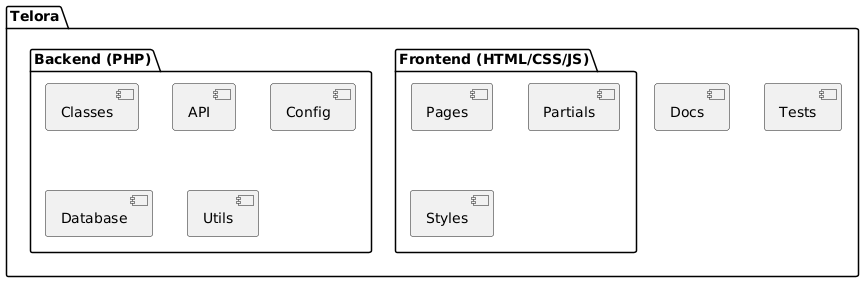
# Diagramme de gantt

# LDAP - Telora

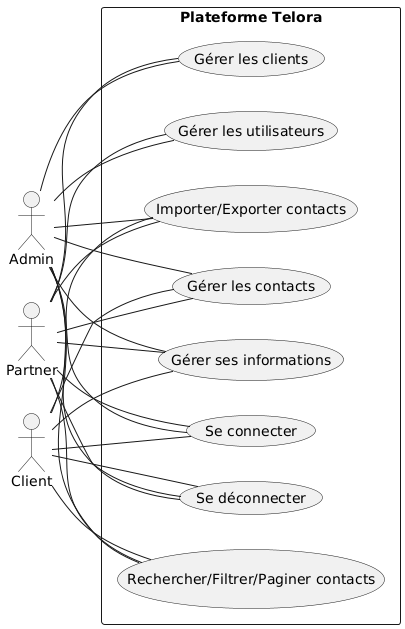
## UML

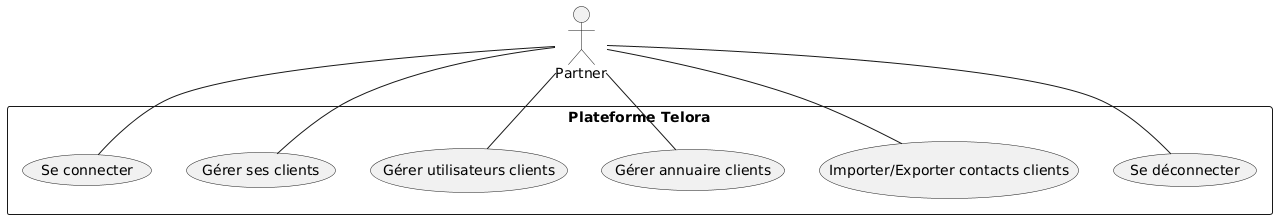
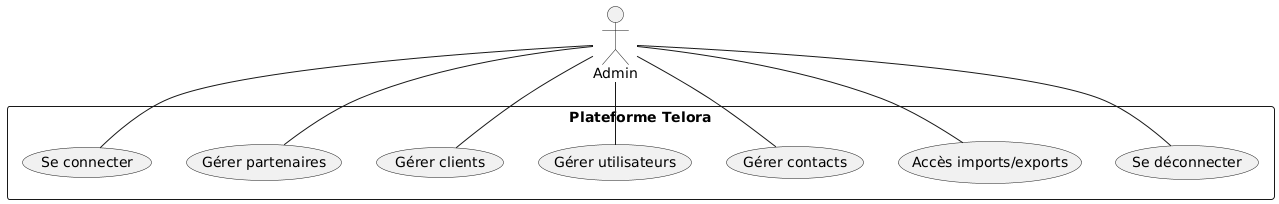
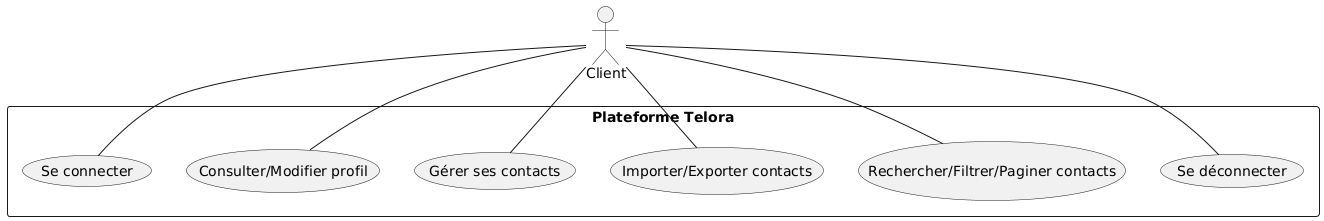
### 1. Diagramme de contexte

### 2. Diagramme de package

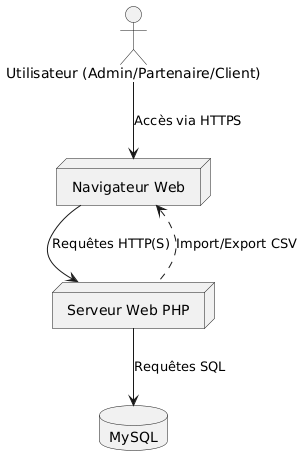


### 3. Diagramme use\_case

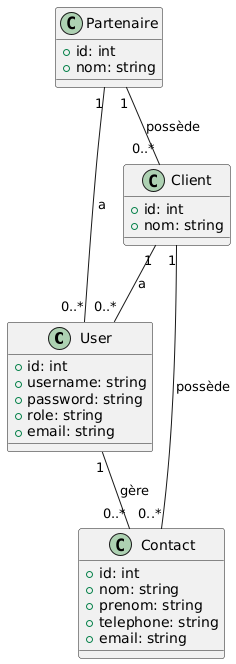




### 4. Cartographie technique



### 5. Diagramme de classe



# LDAP – Telora

## Merise – Modélisation BDD

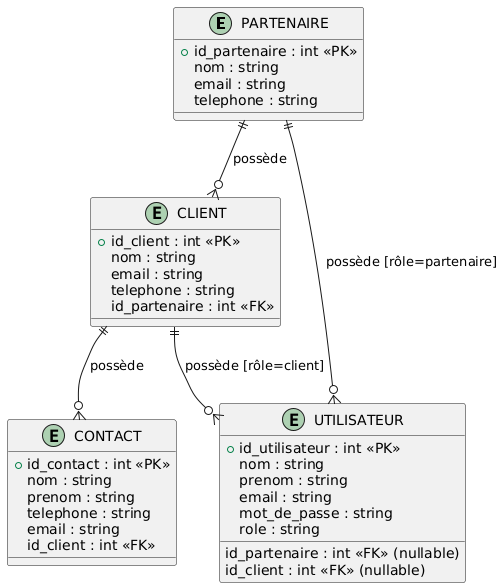
### 1. Dictionnaire de données

### 2. MCD (Modelé conceptuel de données)

### 

### 

### 3. MLD (Modèle logique de données)



### 4. MPD (Modèle physique de données)

# 1ClickAllEat

## Extraits de code commentés

### 1. Authentification et gestion des rôles

# php

# CopyInsert

# // Extrait de annuaire/editcontact\_form.php

# session\_start();

# if (!isset($\_SESSION['role'])) {

# header('Location: ' . $rootPath . '/login/login.php');

# exit();

# }

# // Récupération du rôle et des IDs en session

# $role = $\_SESSION['role'];

# $partnerId = $\_SESSION['partner\_id'] ?? null;

# Explication détaillée :

# session\_start()

# démarre la session PHP, permettant d’accéder aux variables de session pour suivre l’état de connexion de l’utilisateur.

# La condition

# if (!isset($\_SESSION['role']))

# vérifie si l’utilisateur est authentifié (un rôle doit être défini en session après connexion).

# Si l’utilisateur n’est pas connecté, il est redirigé vers la page de login, ce qui protège toutes les pages sensibles contre l’accès non autorisé.

# Les variables de session comme

# role

# et

# partner\_id

# sont utilisées pour adapter dynamiquement l’interface et les droits d’accès. Par exemple, un partenaire ne pourra voir que ses propres clients, un client que ses propres contacts, etc.

# Ce mécanisme centralise la gestion des droits et garantit la séparation stricte des accès selon le rôle, comme exigé dans le cahier des charges.

# **2. Récupération et affichage paginé des contacts**

# php

# CopyInsert

# // Extrait de annuaire/annuaire.php

# $page = isset($\_GET['page']) ? max(1, (int)$\_GET['page']) : 1;

# $perPage = isset($\_GET['perPage']) ? (int)$\_GET['perPage'] : 10;

# $totalContacts = $annuaireManager->countAnnuaireByClient($clientsId);

# $offset = ($page - 1) \* $perPage;

# $totalPages = max(1, ceil($totalContacts / $perPage));

# $contacts = $annuaireManager->getAnnuaireByClientPaginated($clientsId, $offset, $perPage);

# **Explication détaillée :**

# Cette séquence gère la pagination pour l’annuaire des contacts d’un client.

# Elle récupère la page courante et le nombre d’éléments à afficher par page depuis l’URL (GET), avec des valeurs par défaut sécurisées.

# **countAnnuaireByClient($clientsId)** compte le nombre total de contacts pour le client, ce qui permet de calculer le nombre total de pages.

# L’offset (

# $offset

# ) détermine à partir de quel contact commencer l’affichage pour la page courante.

# **getAnnuaireByClientPaginated($clientsId, $offset, $perPage)** récupère uniquement les contacts à afficher pour cette page, optimisant les performances même pour de gros volumes de données.

# Ce mécanisme améliore l’ergonomie et la rapidité de navigation dans l’annuaire.

# **3. Modification d’un contact**

# php

# CopyInsert

# // Extrait de database/Annuaire\_request.php

# function updateContact($contactId, $prenom, $nom, $email, $societe, $adresse, $ville, $telephone, $commentaire) {

# try {

# $sql = "UPDATE User\_annuaire

# SET Prenom = :prenom, Nom = :nom, Email = :email, Societe = :societe,

# Adresse = :adresse, Ville = :ville, Telephone = :telephone, Commentaire = :commentaire

# WHERE idUserAnnuaire = :contactId";

# $stmt = $this->pdo->prepare($sql);

# $stmt->bindParam(":contactId", $contactId, PDO::PARAM\_INT);

# $stmt->bindParam(":prenom", $prenom, PDO::PARAM\_STR);

# $stmt->bindParam(":nom", $nom, PDO::PARAM\_STR);

# $stmt->bindParam(":email", $email, PDO::PARAM\_STR);

# $stmt->bindParam(":societe", $societe, PDO::PARAM\_STR);

# $stmt->bindParam(":adresse", $adresse, PDO::PARAM\_STR);

# $stmt->bindParam(":ville", $ville, PDO::PARAM\_STR);

# $stmt->bindParam(":telephone", $telephone, PDO::PARAM\_STR);

# $stmt->bindParam(":commentaire", $commentaire, PDO::PARAM\_STR);

# return $stmt->execute();

# } catch (PDOException $e) {

# throw new Exception("Erreur lors de la modification du contact : " . $e->getMessage());

# }

# }

# **Explication détaillée :**

# Cette méthode met à jour toutes les informations d’un contact dans la base de données.

# Elle utilise une requête SQL préparée, ce qui protège contre les injections SQL et garantit la sécurité des données.

# Chaque champ du contact (prénom, nom, email, etc.) est lié à un paramètre de la requête pour éviter toute altération malveillante.

# En cas d’erreur lors de la mise à jour (par exemple, si la base est inaccessible), une exception claire est levée, facilitant le debug et la robustesse de l’application.

# Cette méthode est appelée lors de la modification d’un contact depuis l’interface utilisateur.

# **4. Import de contacts depuis un fichier CSV**

# php

# CopyInsert

# // Extrait de database/Annuaire\_request.php

# function importAnnuaireFromCSV($clientId, $filePath) {

# // Ouvre le fichier, lit chaque ligne, nettoie et insère chaque contact pour le client donné

# // Gestion des erreurs de format et des doublons

# }

# **Explication détaillée :**

# Cette fonction permet d’importer en masse des contacts pour un client à partir d’un fichier CSV.

# Elle lit chaque ligne du fichier, nettoie les données (suppression des caractères spéciaux, normalisation des champs) et insère chaque contact dans la base.

# Elle gère les erreurs de format (colonnes manquantes, emails invalides, etc.) et évite les doublons, assurant la qualité des données importées.

# Ce mécanisme facilite la migration de données ou l’intégration de nouveaux clients disposant déjà d’un annuaire.

# **5. Suppression d’un client**

# php

# CopyInsert

# // Extrait de database/clients\_request.php

# public function deleteClient($clientId) {

# $stmt = $this->pdo->prepare("DELETE FROM clients WHERE idclient = ?");

# return $stmt->execute([$clientId]);

# }

# **Explication détaillée :**

# Cette méthode supprime un client de la base de données à partir de son identifiant.

# L’utilisation d’une requête préparée garantit la sécurité de l’opération.

# Elle permet de gérer la suppression d’un client depuis l’interface d’administration ou de gestion partenaire, en respectant les droits d’accès.

# **6. Connexion utilisateur**

# php

# CopyInsert

# // Extrait de database/login\_request.php

# public function login($login, $password) {

# $stmt = $this->pdo->prepare("SELECT \* FROM utilisateurs WHERE login = ?");

# $stmt->execute([$login]);

# $user = $stmt->fetch();

# if ($user && password\_verify($password, $user['motdepasse'])) {

# // Authentification réussie, création de la session

# $\_SESSION['user\_id'] = $user['id'];

# $\_SESSION['role'] = $user['role'];

# // Redirection selon le rôle

# $this->redirectBasedOnRole($user['role'], $user);

# } else {

# // Erreur d’authentification

# return false;

# }

# }

# **Explication détaillée :**

# Cette méthode gère la connexion d’un utilisateur à l’application.

# Elle recherche l’utilisateur dans la base à partir du login, puis vérifie le mot de passe avec

# password\_verify

# , qui prend en charge le hachage sécurisé.

# Si la connexion réussit, elle initialise la session et stocke l’identifiant et le rôle de l’utilisateur, permettant de personnaliser l’accès à l’application.

# La fonction

# redirectBasedOnRole

# permet de rediriger l’utilisateur vers l’espace qui lui correspond (admin, partenaire, client).

# En cas d’échec, la méthode retourne

# false

# , ce qui permet d’afficher un message d’erreur à l’utilisateur.