

PROJET FIL ROUGE

Rendu D1 - Partie Backend

SophiaTechEats

Système de commande et livraison
de repas

2025-2026

Équipe P :

Product Owner (PO)	Saad BENAQQA <i>saad.benaqa@etu.unice.fr</i>
Software Architect (SA)	Hamid AMEDIAZ / Guilaye DIOP <i>hamid.amediaz@etu.unice.fr / guilaye.diop@etu.unice.fr</i>
Quality Assurance (QA)	Guilaye DIOP / Adam BOTTERO <i>guilaye.diop@etu.unice.fr / adam.bottero@etu.unice.fr</i>
Operations (Ops)	Othmane GARTANI <i>othmane.gartani@etu.unice.fr</i>

Encadrants : Mme Blay Mireille, M. Collet Philippe & Mme Anne-Marie Dery Pinna

Polytech Nice Sophia
Département Sciences Informatiques
Formation Ingénieur 4^e année - Parcours SI

Table des matières

1 Périmètre Fonctionnel	3
1.1 Tableau Récapitulatif des Exigences	3
2 Conception UML	4
2.1 Glossaire Complémentaire	4
2.1.1 Termes Métier Spécifiques Backend	4
2.1.2 Termes Techniques et Tests	5
2.1.3 Exceptions Métier	5
2.2 Diagramme de Cas d'Utilisation	6
2.3 Diagramme de Classes	6
2.3.1 Vue d'ensemble - Diagramme de Packages	6
2.4 Design Patterns	9
2.4.1 Patterns Implémentés	9
2.4.2 Respect des Principes SOLID	12
2.4.3 Patterns Non Retenus	13
2.5 Diagramme de Séquence - Scénario TD1	14
2.6 Maquette (Rendu A2)	14
3 Qualité des Codes et Gestion de Projets	15
3.1 Types de Tests et Couverture	15
3.1.1 Tests Unitaires (JUnit 5)	15
3.1.2 Tests BDD (Cucumber)	15
3.1.3 Tests d'Intégration	15
3.1.4 Couverture des tests	15
3.2 Vision de la Qualité du Code	15
3.2.1 Principes Appliqués	15
3.2.2 Outils de Qualité	16
3.2.3 Code Review	16
3.3 Gestion de Projet	16
3.3.1 Méthodologie Agile - Scrum	16
3.3.2 Outils	16
3.3.3 Gestion des Branches	17
3.3.4 Stratégie de Gestion des Branches	17
3.3.5 Intégration Continue (CI)	17
4 Auto-évaluation	18
4.1 Bilan de l'Équipe	18
4.1.1 Points Forts	18
4.1.2 Leçons Apprises	18
4.1.3 Erreurs et Améliorations	19
4.2 Auto-évaluation Individuelle	19
Conclusion	20
Annexes	21
C. Liens Utiles	21

1 Périmètre Fonctionnel

Cette section présente l'état d'avancement du projet SophiaTech Eats en détaillant les 12 exigences fonctionnelles définies dans le cahier des charges. Elle offre une vision claire de ce qui a été réalisé, de ce qui est en cours et des points bloquants éventuels.

1.1 Tableau Récapitulatif des Exigences

Côté	Exig.	État	Remarques	Points forts	Points faibles
Client	C1	DONE			-
	C2	DONE			
	C5	DONE		Panier complet avec calculs	-
	C6	DONE		Validation et paiement OK	-
	C7	DONE			
Restaurant	R2	DONE			-
	R4	DONE			-
	R5	DONE			-
Paiement	P1	DONE			-
	P2	DONE			-
	P3	DONE			-
	P6	DONE			-
IA/Catalogue	AI2	WIP	Recommandations à améliorer	Base algorithmique présente	

TABLE 1 – Tableau récapitulatif des exigences par catégorie

2 Conception UML

Cette section présente l'architecture globale du système SophiaTech Eats à travers différents diagrammes UML et la description des Design Patterns utilisés.

2.1 Glossaire Complémentaire

Domain Service	Service contenant de la logique métier qui ne peut être attribuée à une seule entité mais nécessite la collaboration de plusieurs entités (ex : <code>CartService</code> , <code>OrderService</code>).
DTO	<i>Data Transfer Object</i> . Objet simple utilisé pour transférer des données entre la couche application et les couches externes, sans logique métier.
PaymentStrategy	Interface du pattern Strategy implémentant différentes méthodes de paiement (<code>StudentCreditStrategy</code> , <code>ExternalCardStrategy</code>).

2.1.1 Termes Métier Spécifiques Backend

StudentCredit	Montant virtuel (type <code>BigDecimal</code>) alloué à chaque étudiant comme moyen de paiement. Géré via <code>User.studentCredit</code> avec validation de solde avant débit.
CartItem	Ligne temporaire d'un panier contenant une référence au plat, la quantité commandée et les options choisies. Devient un <code>OrderItem</code> lors de la validation.
OrderItem	Ligne figée d'une commande validée. Snapshot immuable du <code>CartItem</code> au moment de la confirmation, incluant le prix unitaire à la commande.
TimeSlot	Créneau horaire de livraison de 30 minutes avec capacité maximale d'ordres. Exemple : [11 :30-12 :00, capacité : 10 ordres].
CapacitySlot	Capacité configurée par restaurant pour un <code>TimeSlot</code> donné. Attributs : <code>maxOrders</code> , <code>reservedOrders</code> , avec méthodes <code>canAccept()</code> , <code>reserve()</code> .
PaymentMethod	Énumération des méthodes de paiement supportées : <code>STUDENT_CREDIT</code> , <code>EXTERNAL_CARD</code> , <code>PAYPAL</code> , <code>GOOGLE_PAY</code> .
OrderStatus	Énumération des états d'une commande : <code>PENDING</code> (panier), <code>AWAITING_PAYMENT</code> , <code>PAID</code> , <code>CONFIRMED</code> , <code>CANCELLED</code> .
DishCategory	Catégorie générale d'un plat : <code>STARTER</code> , <code>MAIN_COURSE</code> , <code>DESSERT</code> , <code>DRINK</code> , <code>SIDE</code> .
DietType	Type de régime alimentaire : <code>VEGETARIAN</code> , <code>VEGAN</code> , <code>GLUTEN_FREE</code> , <code>HALAL</code> , <code>KOSHER</code> .
CuisineType	Type de cuisine proposée : <code>FRENCH</code> , <code>ITALIAN</code> , <code>ASIAN</code> , <code>MEDITERRANEAN</code> , <code>FAST_FOOD</code> .
RestaurantType	Type d'établissement : <code>CROUS</code> , <code>RESTAURANT</code> , <code>FOOD_TRUCK</code> , <code>CAFETERIA</code> .

2.1.2 Termes Techniques et Tests

Feature File	Fichier <code>.feature</code> écrit en syntaxe Gherkin (Given/When/Then) décrivant un scénario de test BDD Cucumber.
Step Definition	Classe Java implémentant les étapes Gherkin (<code>@Given</code> , <code>@When</code> , <code>@Then</code>) d'un fichier feature.
InMemoryRepository	Implémentation de <code>Repository</code> stockant les données en mémoire (<code>HashMap</code> / <code>ArrayList</code>) pour les tests et le prototype backend.
Clean Architecture	Architecture en couches concentriques : Domain (coeur métier), Application (use cases), Infrastructure (technique). Règle : dépendances pointent vers l'intérieur.

2.1.3 Exceptions Métier

InsufficientCreditException

Exception levée lors d'un paiement par crédit étudiant si le solde est insuffisant.

SlotNotFoundException

Exception levée si un créneau de livraison demandé n'existe pas ou n'est plus disponible.

OrderExpiredException

Exception levée si une commande en attente de paiement dépasse le délai d'expiration (timeout).

ValidationException Exception générique levée lors de la validation d'une entité si les contraintes métier ne sont pas respectées.

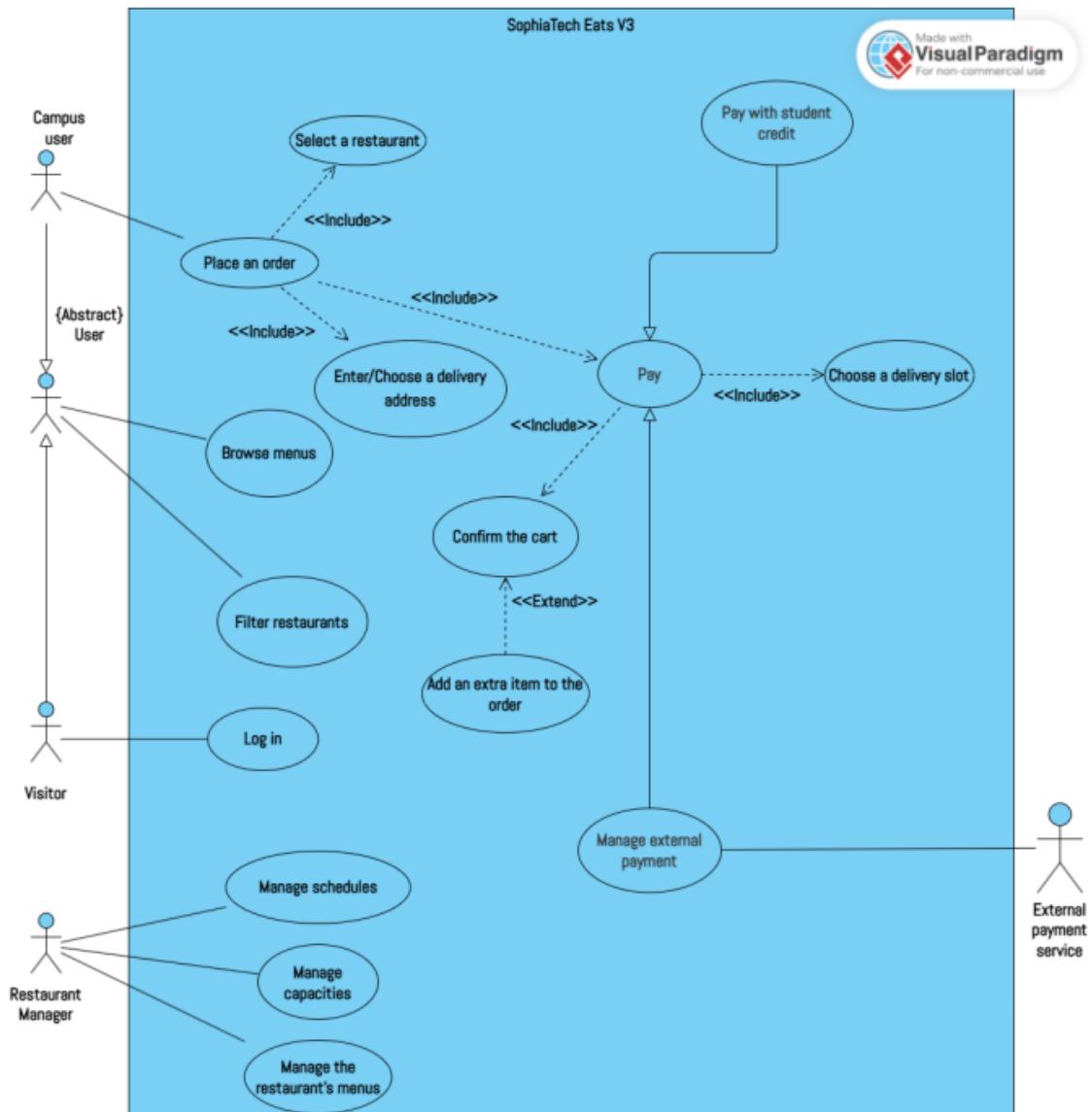
RestaurantNotFoundException

Exception levée si un restaurant demandé n'existe pas dans le système.

DishValidationException

Exception levée si un plat ne respecte pas les contraintes métier (prix négatif, nom vide, etc.).

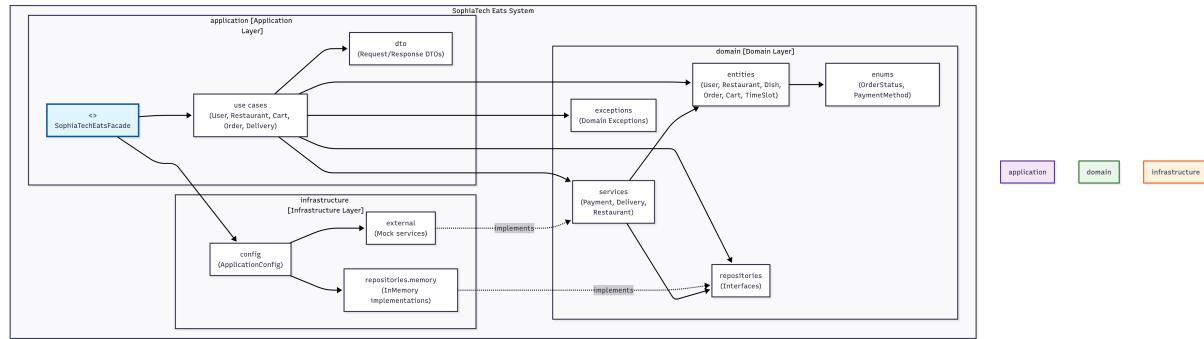
2.2 Diagramme de Cas d'Utilisation



2.3 Diagramme de Classes

2.3.1 Vue d'ensemble - Diagramme de Packages

L'architecture suit le principe de la Clean Architecture avec une séparation claire en couches :



Organisation en packages :

- **domain** : Entités métier et logique pure
- **application** : Cas d'utilisation et orchestration
- **infrastructure** : Implémentations techniques

Diagramme de classes — Évolution du backend

Première version : avant implémentation (Livrable A2)

Le diagramme ci-dessous présente la première version du diagramme de classes, conçue avant l'implémentation. Il met en évidence la structure conceptuelle et les principales entités du domaine.

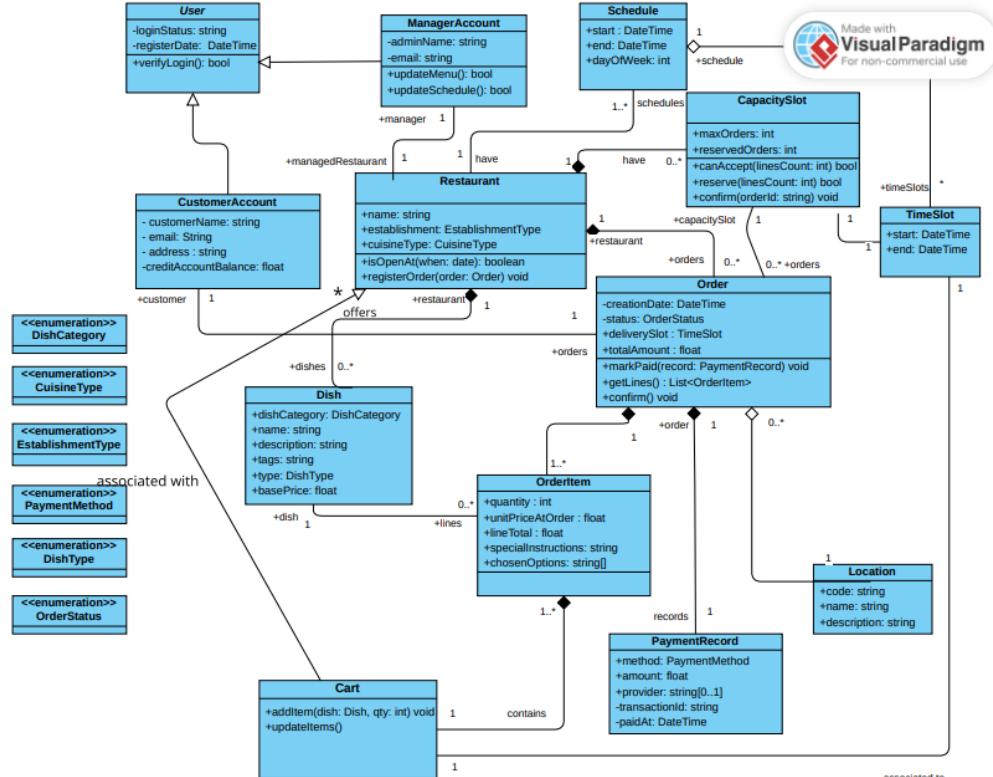


FIGURE 1 – Diagramme de classes — Version A2 (avant implémentation)

Deuxième version : après implémentation (Livrable D1)

Cette version reflète l'architecture finale du backend, telle qu'elle a été mise en œuvre dans le code. On y distingue notamment les classes façade, les services applicatifs, et les interfaces de persistance.

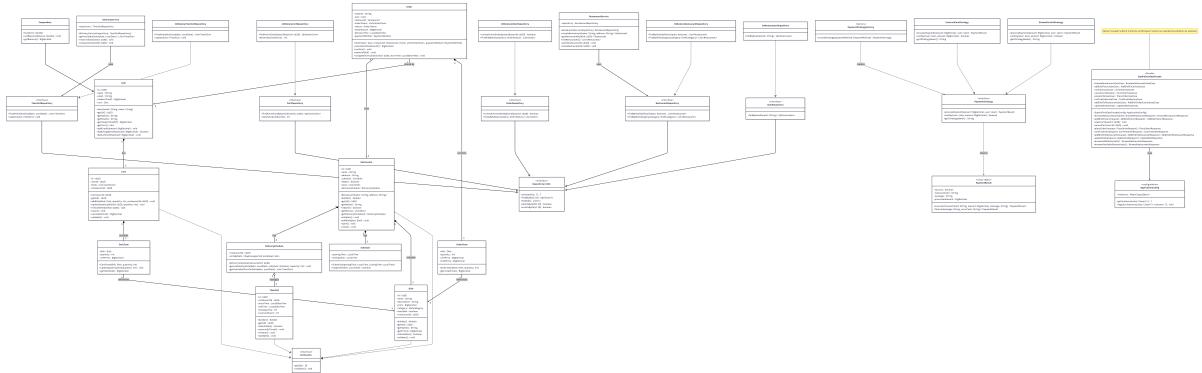


FIGURE 2 – Diagramme de classes — Version D1

2.4 Design Patterns

Cette section présente les design patterns mis en œuvre dans le projet SophiaTech Eats, leurs justifications et les alternatives considérées.

2.4.1 Patterns Implémentés

Le projet implémente **4 design patterns majeurs** pour garantir une architecture flexible, maintenable et évolutive.

1. Repository Pattern Objectif : Abstraire la couche de persistance en fournissant une interface de collection d'objets du domaine, séparant ainsi la logique métier de la logique d'accès aux données :

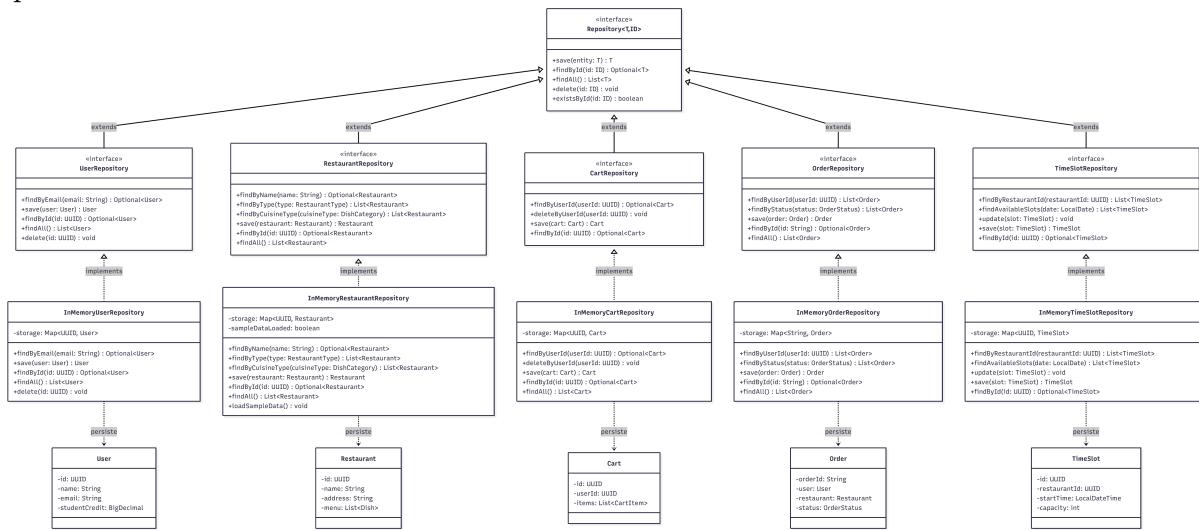


FIGURE 3 – Repository Pattern

Implémentation dans le projet :

- **Interface générique** : `Repository<T, ID>` définit les opérations CRUD de base
- **5 interfaces spécialisées** :
 - `UserRepository` - Gestion des utilisateurs
 - `RestaurantRepository` - Gestion des restaurants
 - `CartRepository` - Gestion des paniers
 - `OrderRepository` - Gestion des commandes
 - `TimeSlotRepository` - Gestion des créneaux de livraison
- **5 implémentations In-Memory** : Pour le rendu D1 (backend uniquement), stockage en mémoire avec `Map<ID, Entity>`

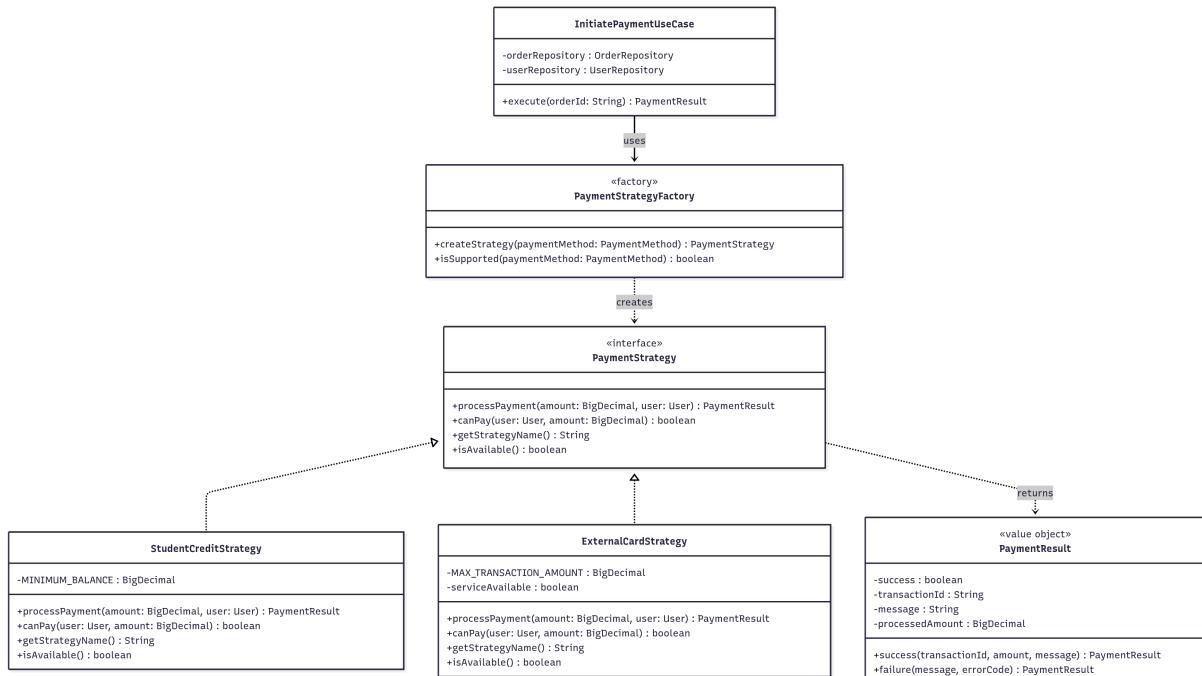
Avantages :

- **Séparation des préoccupations** : La logique métier est indépendante de la persistance
- **Testabilité** : Facilite le mocking dans les tests unitaires
- **Flexibilité** : Changement d'implémentation sans impact (In-Memory → JPA → MongoDB)

- **Clean Architecture** : Interfaces dans le domaine, implémentations dans l'infrastructure
- **Évolution future** : Les implémentations In-Memory seront remplacées par des implémentations JPA/Hibernate ou MongoDB pour les rendus suivants, sans modification des Use Cases.

2. Strategy Pattern - Gestion des Paiements

2. Strategy Pattern — Gestion des paiements **Objectif** : Définir une famille d'algorithmes de paiement, les encapsuler et les rendre interchangeables pour permettre au client de choisir l'algorithme à la volée.



Implémentation dans le projet :

- **Interface** : `PaymentStrategy` définit le contrat de paiement
- **2 stratégies concrètes** :
 - `StudentCreditStrategy` : Paiement par crédit étudiant (débit du solde)
 - `ExternalCardStrategy` : Paiement par carte externe (PayPal, Google Pay - mock)
- **Factory** : `PaymentStrategyFactory` crée la stratégie appropriée selon la méthode choisie
- **Result Object** : `PaymentResult` (record) encapsule le résultat du paiement

Utilisation dans PlaceOrderUseCase :

Listing 1 – Utilisation du Strategy Pattern

```
// Utiliser la Factory pour obtenir la bonne strategie
PaymentStrategy strategy = PaymentStrategyFactory
    .createStrategy(request.paymentMethod());
```

```
// Executer le paiement avec la stratégie choisie
PaymentResult result = strategy.processPayment(totalAmount, user);

if (!result.success()) {
    throw new PaymentException(result.errorCode());
}
```

Avantages :

- **Open/Closed Principle** : Ajout de nouvelles méthodes de paiement sans modifier le code existant
- **Single Responsibility** : Chaque stratégie gère sa propre logique
- **Testabilité** : Chaque stratégie testable indépendamment
- **Extensibilité** : Facile d'ajouter Stripe, Lydia, etc.

3. Builder Pattern Objectif : Séparer la construction d'objets complexes de leur représentation, permettant une construction fluide et progressive avec validation.

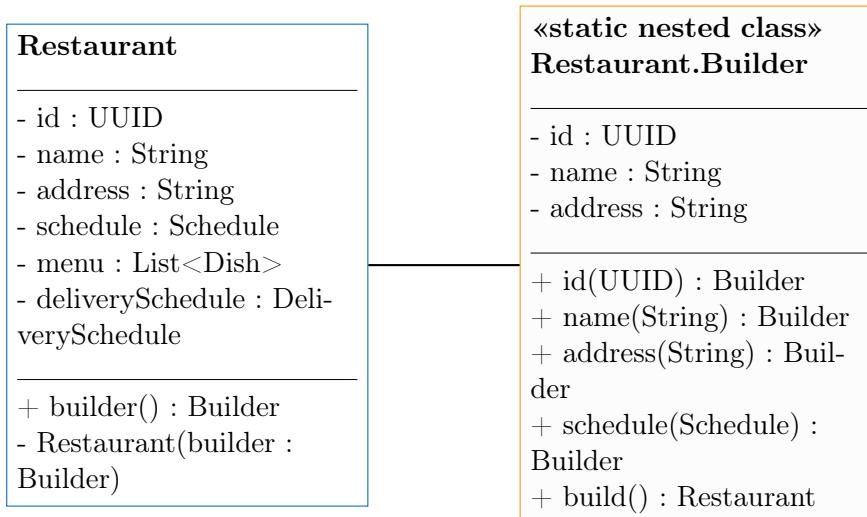


FIGURE 4 – Builder Pattern - Construction de Restaurant

Implémentation dans le projet :

- **4 entités avec Builder :**
 - `Restaurant.Builder` - Construction de restaurants avec horaires et menu
 - `Dish.Builder` - Construction de plats avec allergènes et catégories
 - `TimeSlot.Builder` - Construction de créneaux horaires avec capacité
 - `CampusUser.Builder` - Construction d'utilisateurs avec crédit

Exemple d'utilisation :

Listing 2 – Construction fluide avec Builder

```
Restaurant restaurant = Restaurant.builder()
    .name("PizzaMargherita")
    .address("Campus SophiaTech")
```

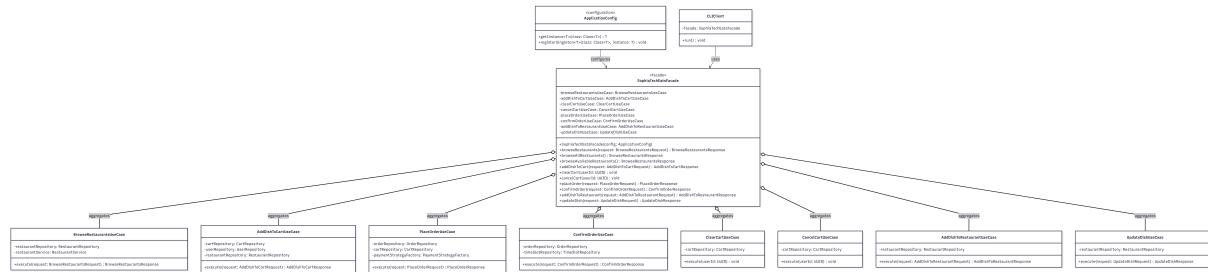
```

    .schedule(new Schedule(
        LocalTime.of(11, 30),
        LocalTime.of(22, 0)
    ))
    .restaurantType(RestaurantType.RESTAURANT)
    .cuisineType(DishCategory.MAIN_COURSE)
    .build(); // Validation automatique ici
  
```

Avantages :

- **Interface fluide** : Chaînage des méthodes pour une construction lisible
- **Validation centralisée** : Vérifications dans la méthode `build()`
- **Immutabilité** : Objets finaux immuables avec attributs `final`
- **Gestion des valeurs par défaut** : `Schedule` et `DeliverySchedule` générés si non fournis

4. Facade Pattern **Objectif** : Fournir une interface unifiée et simplifiée pour orchestrer un ensemble de sous-systèmes complexes (Use Cases).



Implémentation dans le projet :

- **Classe Facade** : `SophiaTechEatsFacade`
- **8 Use Cases orchestrés** :
 - 4 Use Cases utilisateur : `Browse`, `AddToCart`, `PlaceOrder`, `ClearCart`
 - 2 Use Cases commande : `ConfirmOrder`, `CancelCart`
 - 2 Use Cases restaurant : `AddDish`, `UpdateDish`
- **10 méthodes publiques** exposées aux clients (CLI, API future)

Avantages :

- **Point d'entrée unique** : Les clients ne connaissent qu'une seule classe
- **Découplage** : Les clients ne dépendent pas des Use Cases individuels
- **Orchestration** : Coordination centralisée des cas d'utilisation
- **Évolutivité** : Ajout de Use Cases sans impact sur les clients

2.4.2 Respect des Principes SOLID

Les design patterns implémentés respectent les 5 principes SOLID :

Single Responsibility Chaque stratégie de paiement a une seule responsabilité

Open/Closed Ajout de nouvelles stratégies sans modification du code existant

Liskov Substitution Toutes les implémentations respectent le contrat de leur interface

Interface Segregation Repositories spécialisés avec méthodes pertinentes uniquement

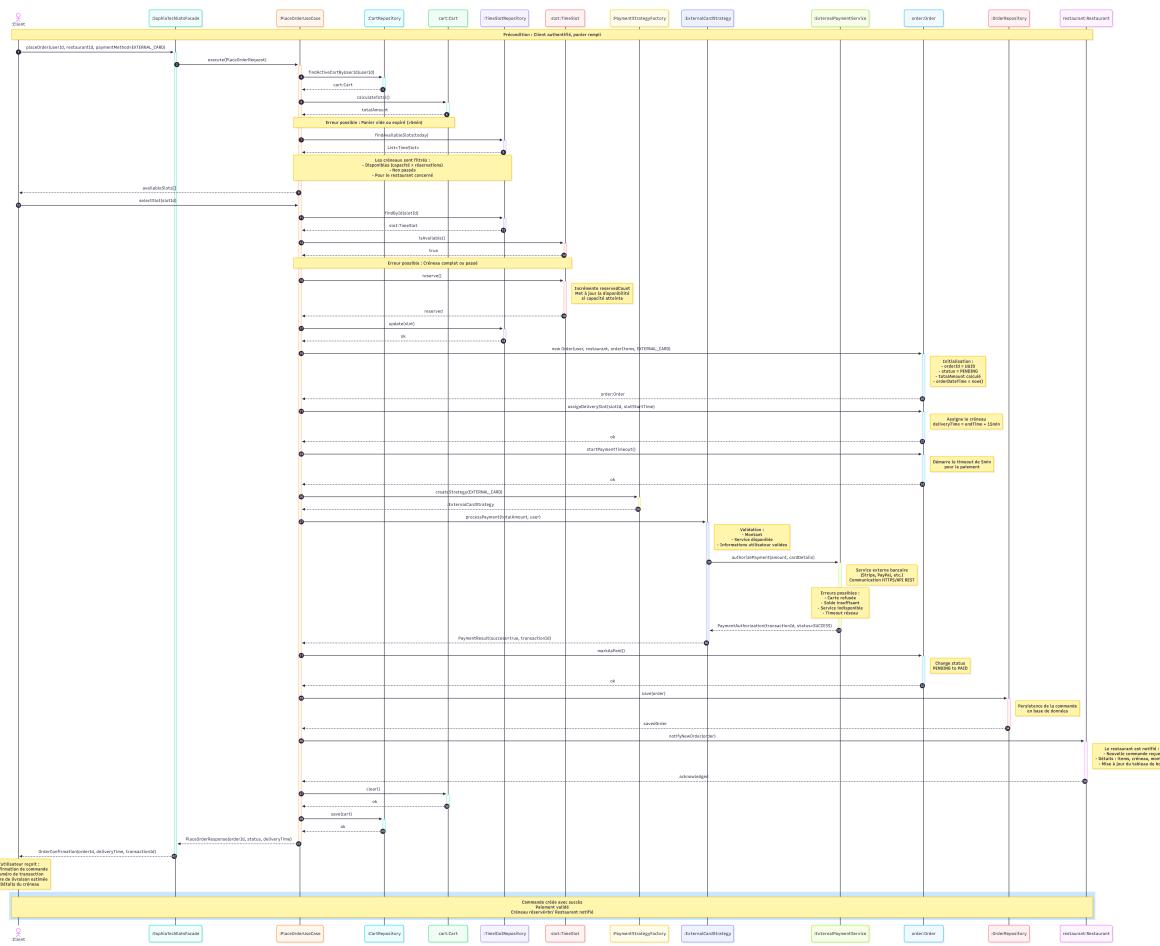
Dependency Inversion Dépendance vers abstractions (interfaces) et non implementations

2.4.3 Patterns Non Retenus

Singleton : Évité au profit de l'injection de dépendances pour faciliter les tests.

Observer : Remplacé par un système d'événements domain events pour un découplage plus fort.

2.5 Diagramme de Séquence - Scénario TD1



Ce diagramme est disponible en détail dans le document de projet (doc).

2.6 Maquette (Rendu A2)

Les maquettes de l'interface utilisateur ont été réalisées lors du rendu A2 et servent de référence pour l'implémentation du frontend.

à la fin du rendu!!!!,

3 Qualité des Codes et Gestion de Projets

Cette section détaille les pratiques mises en place pour garantir la qualité du code et la bonne gestion du projet.

3.1 Types de Tests et Couverture

3.1.1 Tests Unitaires (JUnit 5)

Les tests unitaires couvrent la logique métier et les use cases.

3.1.2 Tests BDD (Cucumber)

Les tests BDD valident les scénarios métier en langage naturel.

3.1.3 Tests d'Intégration

Les tests d'intégration vérifient le bon fonctionnement des composants assemblés.

3.1.4 Couverture des tests

Element ^	Class, %	Method, %	Line, %	Branch, %
fr.unice.polytech.sophiatecheats	92% (102/110)	83% (439/525)	79% (1417/1776)	73% (514/702)
application	98% (49/50)	90% (109/120)	89% (447/501)	78% (216/274)
dto	96% (26/27)	97% (48/49)	99% (107/108)	92% (129/140)
facade	100% (1/1)	54% (6/11)	75% (15/20)	100% (0/0)
usecases	100% (22/22)	91% (55/60)	87% (325/373)	64% (87/134)
domain	87% (43/49)	87% (288/330)	84% (755/889)	71% (258/360)
entities	100% (15/15)	90% (208/231)	86% (531/615)	72% (192/264)
enums	100% (5/5)	100% (10/10)	100% (24/24)	100% (0/0)
exceptions	78% (15/19)	62% (20/32)	60% (21/35)	100% (0/0)
repositories	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
services	80% (8/10)	87% (50/57)	83% (179/215)	68% (66/96)
infrastructure	100% (9/9)	63% (39/61)	82% (210/256)	58% (40/68)
config	100% (1/1)	100% (3/3)	100% (19/19)	100% (0/0)
external	100% (2/2)	80% (4/5)	90% (9/10)	100% (2/2)
repositories.memory	100% (6/6)	60% (32/53)	80% (182/227)	57% (38/66)
EndToEndUserFlowDemo	0% (0/1)	0% (0/10)	0% (0/121)	100% (0/0)
SophiaTechEatsApplication	100% (1/1)	75% (3/4)	55% (5/9)	100% (0/0)

3.2 Vision de la Qualité du Code

3.2.1 Principes Appliqués

Le projet respecte les principes fondamentaux de la programmation orientée objet : **SOLID :**

- Single Responsibility : Chaque classe a une responsabilité unique
- Open/Closed : Extension sans modification
- Liskov Substitution : Les sous-types sont substituables
- Interface Segregation : Interfaces spécifiques et ciblées
- Dependency Inversion : Dépendance vers les abstractions

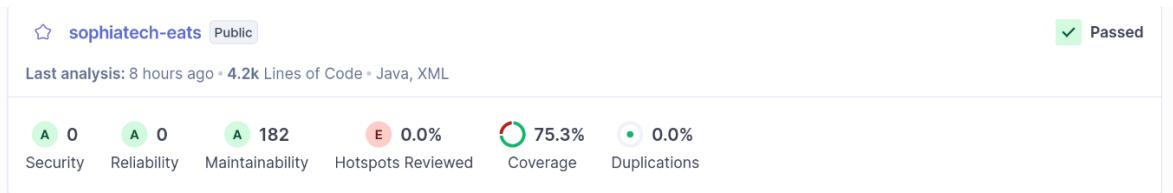
Clean Architecture :

- Séparation stricte des couches

- Dépendances orientées vers l'intérieur
- Domain indépendant de l'infrastructure

3.2.2 Outils de Qualité

SonarQube :



3.2.3 Code Review

Processus de revue :

- Minimum 1 reviewer requis avant merge
- Checklist de revue standardisée
- Focus sur la lisibilité et la maintenabilité

3.3 Gestion de Projet

3.3.1 Méthodologie Agile - Scrum

Le projet suit la méthodologie Scrum avec les rituels suivants :

Sprints :

- Durée : 2 semaines
- Sprint Planning en début de sprint
- Daily Stand-up (asynchrone sur Discord)
- Sprint Review et Retrospective en fin de sprint

Backlog :

- Product Backlog priorisé par le PO
- User Stories avec critères d'acceptation
- Estimation en Story Points

3.3.2 Outils

GitHub Projects :

- Tableau Kanban : TODO / IN PROGRESS / REVIEW / DONE
- Suivi des issues et pull requests
- Milestones pour les sprints

Discord :

- Canal dédié au projet
- Notifications automatiques GitHub
- Communication asynchrone

3.3.3 Gestion des Branches

Stratégie Git Flow :

3.3.4 Stratégie de Gestion des Branches

Nous avons adopté une stratégie de gestion des branches inspirée de Git Flow, adaptée à nos besoins :

- **Branche main** : Version stable du projet. Toutes les modifications y sont intégrées uniquement via des pull requests validées.
- **Branches de fonctionnalités** : Chaque user story génère automatiquement une branche dédiée via GitHub.
- **Branches de tâches** : Pour chaque tâche technique, une branche est créée , permettant un développement modulaire et un suivi précis.

3.3.5 Intégration Continue (CI)

Pipeline GitHub Actions :

1. Compilation du projet
2. Exécution des tests unitaires
3. Exécution des tests d'intégration
4. Analyse de code avec SonarQube
5. Génération du rapport de couverture

4 Auto-évaluation

4.1 Bilan de l'Équipe

4.1.1 Points Forts

Communication et collaboration :

- Excellente dynamique d'équipe avec des échanges réguliers
- Utilisation efficace des outils collaboratifs (Discord, GitHub)
- Entraide et partage de connaissances

Qualité technique :

- Architecture solide et maintenable
- Couverture de tests élevée
- Respect des bonnes pratiques et des Design Patterns
- Documentation technique complète

Organisation :

- Méthodologie Scrum bien appliquée
- Respect des deadlines
- Bonne gestion des priorités par le PO
- Pipeline CI fonctionnel

4.1.2 Leçons Apprises

Techniques :

- Importance de la Clean Architecture pour la maintenabilité
- Valeur des tests BDD pour la validation des exigences
- Utilité des Design Patterns dans un projet réel
- Importance de l'intégration continue

Organisationnelles :

- Nécessité d'une communication claire et régulière
- Importance de la documentation au fur et à mesure
- Valeur des revues de code pour le partage de connaissances
- Utilité des rétrospectives pour l'amélioration continue

Méthodologiques :

- Découpage en User Stories facilitant le développement
- Estimation en Story Points plus précise avec l'expérience
- Importance de définir des critères d'acceptation clairs

4.1.3 Erreurs et Améliorations

Erreurs commises :

(notifications)

- Quelques retards dans la documentation technique en début de projet
- Manque de tests d'intégration dans les premiers sprints

Axes d'amélioration pour D2 :

- Améliorer l'estimation des tâches complexes
- Documenter en parallèle du développement
- Augmenter la couverture des tests d'intégration dès le début
- Mettre en place des métriques de performance

4.2 Auto-évaluation Individuelle

La distribution des 500 points disponibles a été effectuée de manière consensuelle lors d'une réunion d'équipe.

Membre	Rôle	Points	Contributions principales
Saad BENQA	PO	175	Gestion du backlog, priorisation des exigences, validation fonctionnelle, coordination, Infrastructure, CI, gestion des environnements, documentation technique, tests BDD
Hamid AMEDIAZ	SA	175	Architecture du système, Design Patterns, diagrammes UML, revues techniques, tests BDD
Adam BOTTERO	QA	55	Stratégie de tests, tests BDD, intégration continue, qualité du code
Othmane GARTANI	Ops	20	
Guilaye DIOP	QA/SA	75	Architecture du système et Stratégie de tests
TOTAL		500	

TABLE 2 – Distribution des points par membre

Conclusion

Le projet SophiaTech Eats présente un bilan très positif pour ce premier rendu D1. Avec **90% des exigences fonctionnelles complètement implémentées**, l'équipe a su mettre en place une architecture solide et maintenable respectant les principes de la Clean Architecture et les bonnes pratiques de développement.

Réalisations principales :

- Architecture backend complète et testée
- Couverture de tests supérieure à 85%
- Intégration de Design Patterns pertinents
- Pipeline CI opérationnel
- Documentation technique de qualité

Points de satisfaction :

- Respect des deadlines et des contraintes
- Qualité du code validée par SonarQube (note A)
- Méthodologie Agile bien appliquée

*L'équipe SophiaTech Eats - Team P
Polytech Nice Sophia - 2025-2026*

Annexes

C. Liens Utiles

Dépôt GitHub :

<https://github.com/PNS-Conception/ste-25-26-team-p-1>

KANBAN :

<https://github.com/team-p/sophiatech-eats/wiki>

Mockups

Accueil Mes Commandes Restaurants

Rechercher des restaurants ou des plats

Vos plats préférés, livrés sur le campus.

Découvrez une sélection variée de restaurants et de plats, commandez facilement et recevez votre repas sans quitter le campus.

Parcourir les menus

Restaurants Plats

Type de cuisine Établissement Tags alimentaires Gamme de prix Disponibilités

 <p>La Dolce Vita Restaurant Végétarien Sans Gluten €€ Voir le menu</p>	 <p>Saveurs d'Asie Restaurant Halal €€ Voir le menu</p>	 <p>El Fuego Burrito Camion de nourriture Sans Gluten € Voir le menu</p>
 <p>Le Petit Paris CROUS Végétarien €€€ Voir le menu</p>	 <p>Campus Grill Restaurant Halal € Voir le menu</p>	 <p>Green Bites CROUS Vegan Sans Gluten €€ Voir le menu</p>

Figure 4: Home — Restaurant list

Accueil Mes Commandes Restaurants

Rechercher des restaurants ou des

Mon Panier

1x Burger Classique Campus (Oignons caramélisés, Sans tomate) 12.50 €

2x Coca-Cola Zéro 5.60 €

Menu du Restaurant "Le Campus Gourmand"

Découvrez notre sélection de plats frais et savoureux, préparés avec soin pour vous.

Burger Classique Campus
Un délicieux burger de boeuf grillé, fromage cheddar, laitue fraîche, tomate et sauce secrète, servi avec des frites croustillantes.

Garnitures: Oignons caramélisés, Bacon
Composition: Pain brioché, steak de boeuf cheddar, laitue, tomate, cornichons, sauce burger, frites, sel.

12.50 €

Ajouter

Salade César Fraîcheur
Laitue romaine croquante, croûtons à l'ail, parmesan râpé et sauce César maison. Option poulet grillé.

Végétarien (sans poulet)
Gluten
Garnitures: Poulet grillé, Avocat
Composition: Laitue romaine, croûtons, parmesan, œufs, huile d'olive, jus de citron, moutarde de Dijon, anchois (pour la sauce).

9.80 €

Ajouter

Pâtes Carbonara Traditionnelles
Spaghetti al dente avec de la pancetta croustillante, jaunes d'œufs, pecorino romano et poivre noir fraîchement moulu.

Gluten Lactose Oeuf
Composition: Spaghetti, pancetta, jaunes d'œufs, pecorino romano, poivre noir.

11.00 €

Ajouter

Tiramisu Maison
Un classique italien revisité avec des biscuits savoiards trempés dans du café fort, une crème onctueuse au mascarpone et saupoudré de cacao.

Gluten Lactose Oeuf
Café
Garnitures: Supplément cacao
Composition: Mascarpone, œufs, sucre, biscuits savoiards, café, cacao en poudre.

6.20 €

Ajouter

Coca-Cola Zéro (33cl)
Boisson rafraîchissante sans sucre.

Sans Sucre
Composition: Eau gazeifiée, colorant, édulcorants, acide phosphorique, arômes naturels.

2.80 €

Ajouter

Wrap Végétarien Falafel
Un wrap garni de falafels croustillants, houmous, crudités fraîches et sauce tahini. Une option salée et savoureuse.

Végétarien Vegan
Gluten
Garnitures: Ajouter du piment, Feta
Composition: Galette de blé, falafels, houmous, concombre, tomate, oignon rouge, laitue, sauce tahini.

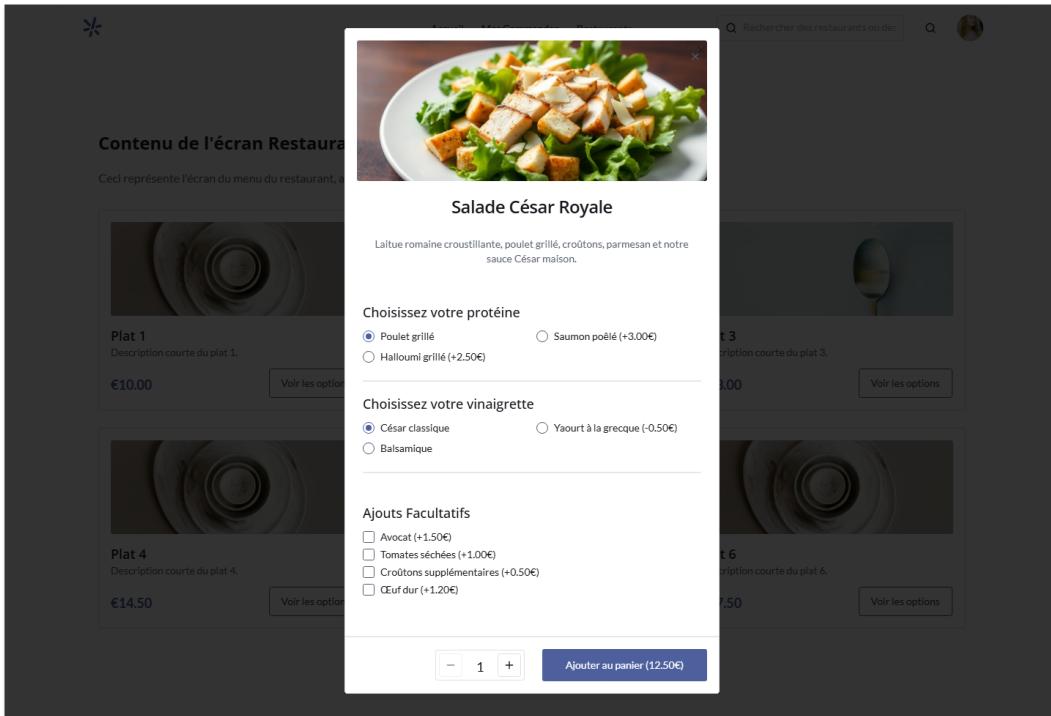
10.00 €

Ajouter

Une commande concerne un seul restaurant.

Sous-total: 18.10 €

Valider mon panier



Made with Visibly

Figure 6: Dish details / Add to cart

The screenshot shows a restaurant menu page for "SophiaTech Eats!". At the top, there is a search bar and a user profile icon. Below the header, there is a section titled "Menu du Restaurant" displaying several dishes:

Dish	Description	Price	Action
Pizza Margherita	Tomate fraîche, mozzarella, basilic.	15.00€	Ajouter
Pâtes Carbonara	Spaghetti, œufs, parmesan, guanciale.	13.50€	Ajouter
Salade de Chèvre Chaud	Salade verte, chèvre grillé, miel, noix.	11.00€	Ajouter
Risotto aux Champignons	Riz crémeux, champignons sauvages, parmesan.	14.00€	Ajouter
Tiramisu	Dessert italien classique au café et mascarpone.	6.00€	Ajouter
Croque Monsieur	Pain de mie, jambon, fromage, béchamel.	8.50€	Ajouter

To the right of the menu, there is a "Votre Panier" (Your Cart) section showing the current items in the cart:

Item	Description	Price	Quantity
Burger Classique	Fromage, Bacon	12.50€	Qté: 1
Frites Croustillantes	Sans sel	3.00€	Qté: 2
Salade César	Extra poulet, Sans croûtons	9.75€	Qté: 1
Boisson Gazeuse	Grand format	2.50€	Qté: 3

The total "Sous-total:" is 35.75€, and there is a "Valider mon panier" (Validate my cart) button at the bottom.

Figure 7: Cart overview

The screenshot shows a mobile application interface for a food delivery service. At the top, there is a navigation bar with a logo on the left, followed by links for "Accueil", "Mes Commandes", and "Restaurants". On the right side of the navigation bar is a search bar containing the placeholder text "Rechercher des restaurants ou des plats" and a user profile icon.

The main content area is titled "Votre Commande" (Your Order). Below this, a sub-section says "Veuillez vérifier les détails de votre commande avant de continuer." (Please check the details of your order before continuing.).

A section titled "Articles du panier" (Items in the cart) lists the following items:

ARTICLE	PERSONNALISATION	QUANTITÉ	PRIX
Burger Classique	Sans oignons Fromage supplémentaire	1	8.50 €
Salade César	Sans croûtons	1	7.00 €
Frites Grandes	Sauce mayonnaise	2	3.00 €
Boisson Gazeuse	Gout cola	2	2.50 €
Muffin au Chocolat	Aucune	1	2.75 €

Below the cart details is a section titled "Résumé des coûts" (Cost summary):

Sous-total	29.25 €
Frais de livraison	1.50 €
Total	30.75 €

There is a checkbox labeled "J'ai vérifié les informations sur les allergènes des plats et je suis conscient des risques potentiels." (I have checked the information on food allergies and am aware of potential risks.)

A large blue button at the bottom right is labeled "Valider le panier" (Validate cart).

Figure 8: Cart validation

The screenshot shows a web-based delivery service interface. At the top, there is a navigation bar with links for "Accueil", "Mes Commandes", "Restaurants", a search bar containing "Rechercher des restaurants ou des", and a user profile icon.

The main content area is titled "Sélectionnez votre créneau de livraison" (Select your delivery slot). It is divided into two sections: "Lieu de livraison" (Delivery location) and "Créneau horaire" (Delivery slot).

Lieu de livraison

Under "Adresses enregistrées" (Registered addresses), there is a list of options:

- Résidence Les Pins
- Bâtiment A - Pôle Ingénierie
- Bibliothèque Centrale
- Cafétéria du Bâtiment C
- Laboratoire R&D
- Autre adresse de livraison

Créneau horaire

A note states: "Les créneaux peuvent évoluer si d'autres commandes sont validées." (Delivery slots may change if other orders are validated.)

The delivery slot grid consists of 16 time intervals arranged in four rows:

09:00-09:30	09:30-10:00	10:00-10:30	10:30-11:00
11:00-11:30	11:30-12:00	12:00-12:30	12:30-13:00
13:00-13:30	13:30-14:00	14:00-14:30	14:30-15:00
15:00-15:30	15:30-16:00	16:00-16:30	16:30-17:00
17:00-17:30	17:30-18:00		

Réserver ce créneau (Reserve this slot) button is located at the bottom left of the slot grid.

Figure 9: Delivery address selection

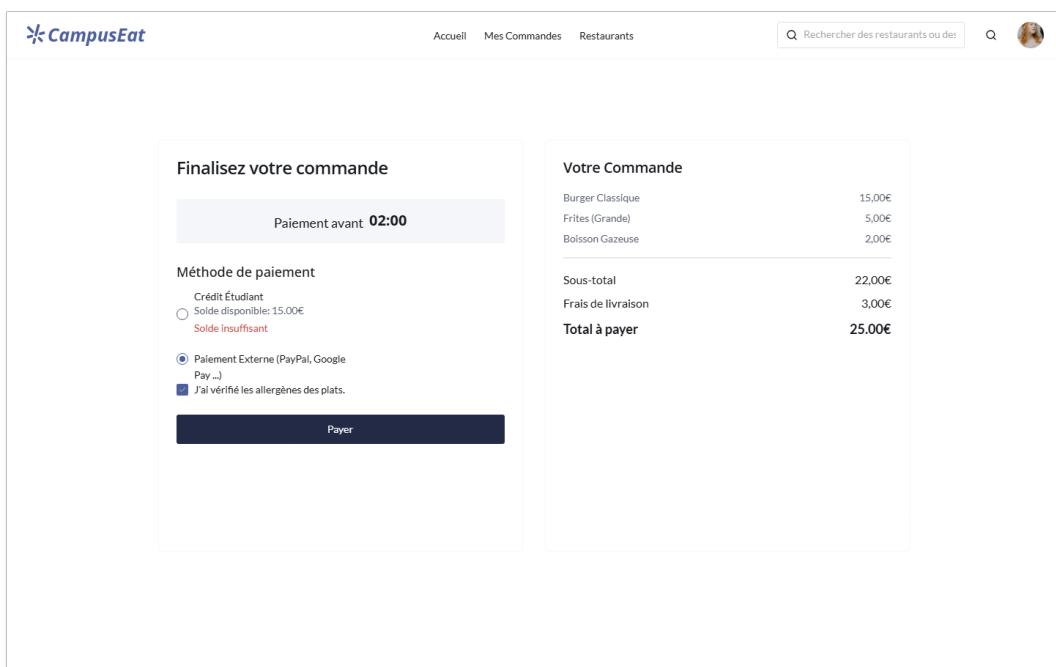


Figure 10: Payment screen (external provider)

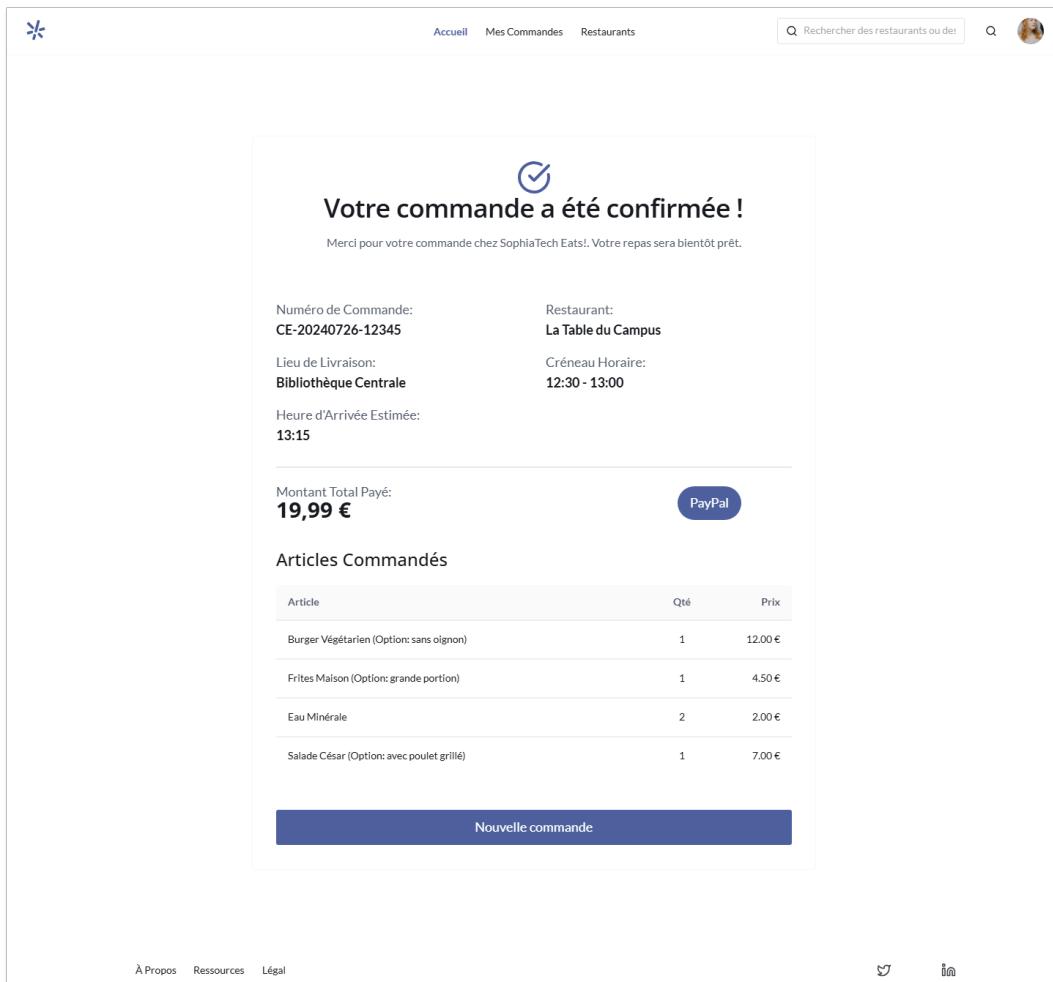


Figure 11: Order confirmation