Cahier des charges techniques

Projet : Refonte SI Gestion de colis Digicheese

(Projet TP7 - Bloc 3 & 4)

Travail réalisé par :

- Clementine Ducournau
- Florian Jordany
- Axel Bernat

Sommaire

1.a Contexte du projet

La fromagerie Digicheese a besoin de moderniser son SI qui est obsolète par rapport aux technologies actuelles.

L'ancienne application est basée sur des fichiers statiques (Access pour le front et le back).

Les problèmes rencontrés étaient :

- Forte instabilité (bugs réguliers)
- Problèmes de maintenance
- Faible possibilité d'évolution de développement
- Manque de fluidité, d'accessibilité et de visibilité pour les utilisateurs
- Besoin d'avoir les fichiers pour lancer l'application

1.b Objectifs du projet

- L'objectif est de moderniser l'ancien SI afin de répondre aux besoins du client qui consistent à avoir une application simple, performante et sécurisée, tout en respectant ses contraintes et ses exigences.

- Les données doivent être stockées dans une base de données.

- L’application doit être facilement maintenable et évolutive.

- L'application doit pouvoir effectuer les mêmes actions que l'ancien système, sans régression dans le nouveau SI.

- Les bugs de l'ancienne application doivent être résolus dans la nouvelle version.

Nous avons réalisé une analyse et une proposition pour le nouveau SI qui a été validée par le P.O. (Lien vers le Powerpoint)

1.c Parties prenantes

Le nouveau SI sera livré à l'entreprise Digicheese, les End Users sont les utilisateurs finaux de l'applications et les Key Users sont là pour suivre l'état d'avancement et pour valider le projet pendant son développement.

Cotés client :

Effectif (Métier)	Key user (nombre)	End user
Gestionnaire de colis	1	Oui
Client	1	Oui
Product Owner	Non	Non
DSI	Non	Non

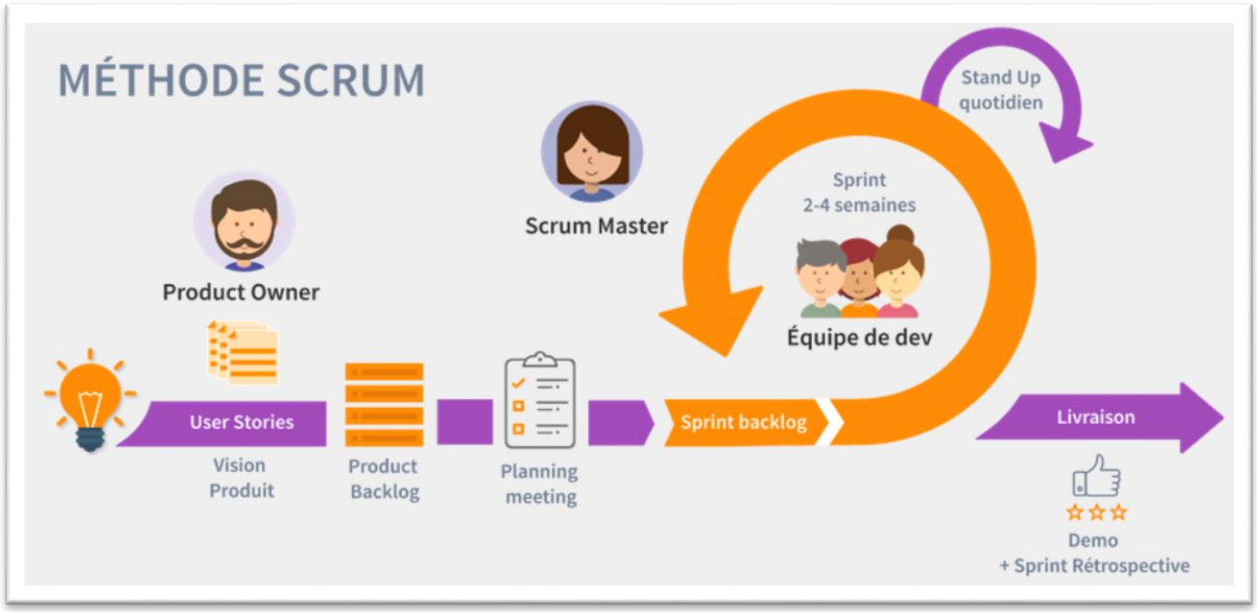
Cotés Prestataires :

2 Développeurs (Florian, Axel)

1 Scrum master (Chef de projet / Clémentine)

1.d Méthodologie de développement

Nous avons choisi d'adopter la méthodologie agile (SCRUM) pour le développement de notre application. La méthodologie agile permet une approche itérative et collaborative, favorisant la flexibilité, la réactivité et l'adaptation aux changements. Elle nous permettra de livrer des fonctionnalités de manière itérative, d'obtenir des retours fréquents des parties prenantes et d'ajuster notre approche en conséquence.



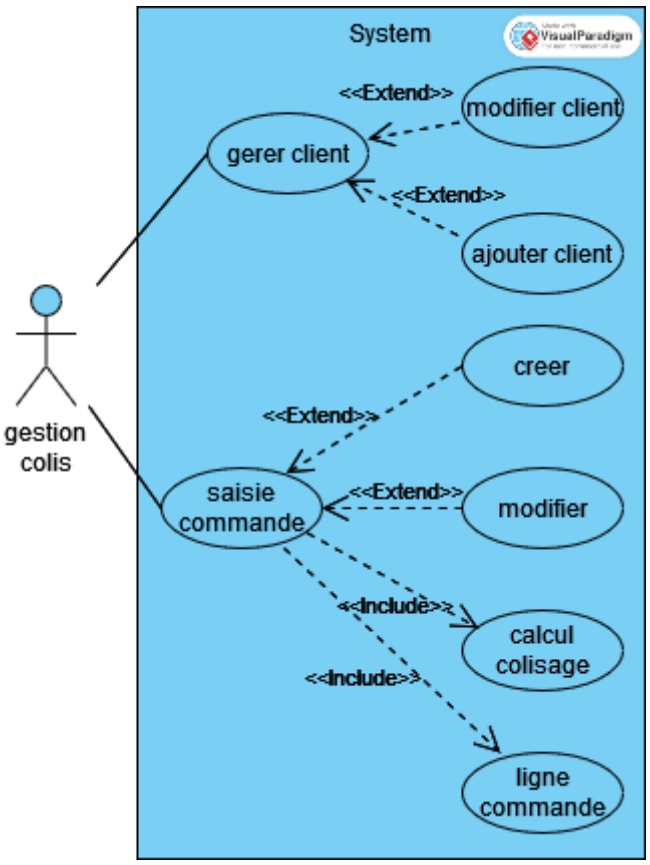
2.a Besoins clients et fonctionnalités principales

Le client a besoin d'une application simple, performante et sécurisé.

Les données doivent être stocker en ligne, l'application doit être facilement maintenable et évolutive.

L'utilisateur doit pouvoir faire les mêmes actions qu'avec l'ancien système, sans régression dans le nouveau SI.

2.b Cas d'utilisation et objectifs fonctionnels



Pour la gestion de colis, il faut pouvoir gérer les clients et saisir les commandes, le gestionnaire de colis est responsable de ces tâches.

La gestion de client comprend le fait d'ajouter un client et de modifier ses informations.

La gestion de commande comprend le fait de créer une commande et de modifier les informations d'une commande existante.

La gestion du stock permet de réaliser un inventaire et de l'imprimer

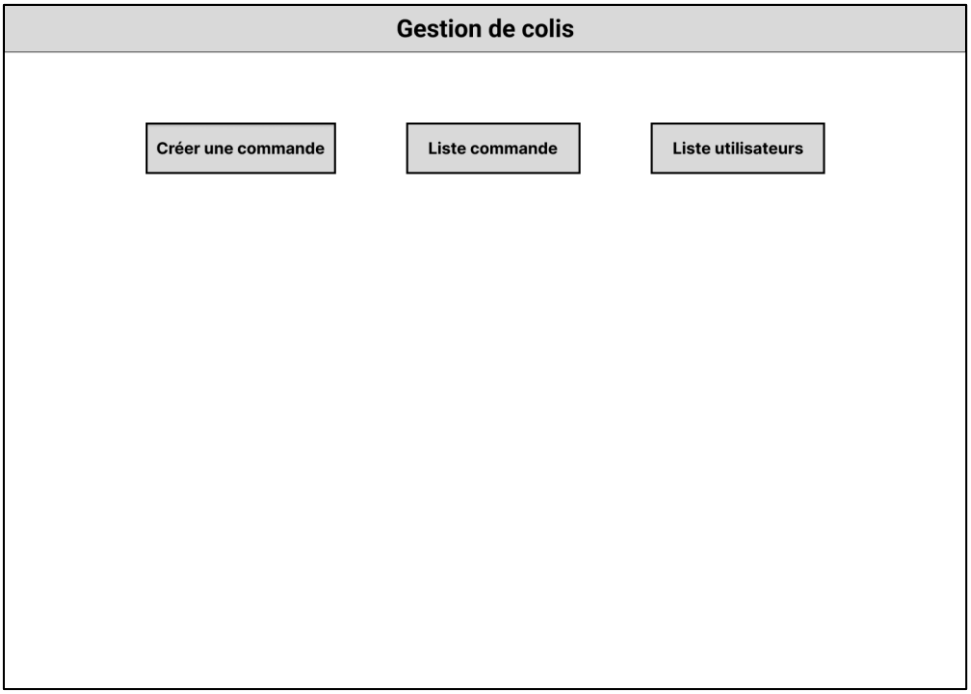
L'administration permet de gérer les rôles des différents utilisateurs ainsi que de gérer les

2.c Exigences spécifiques

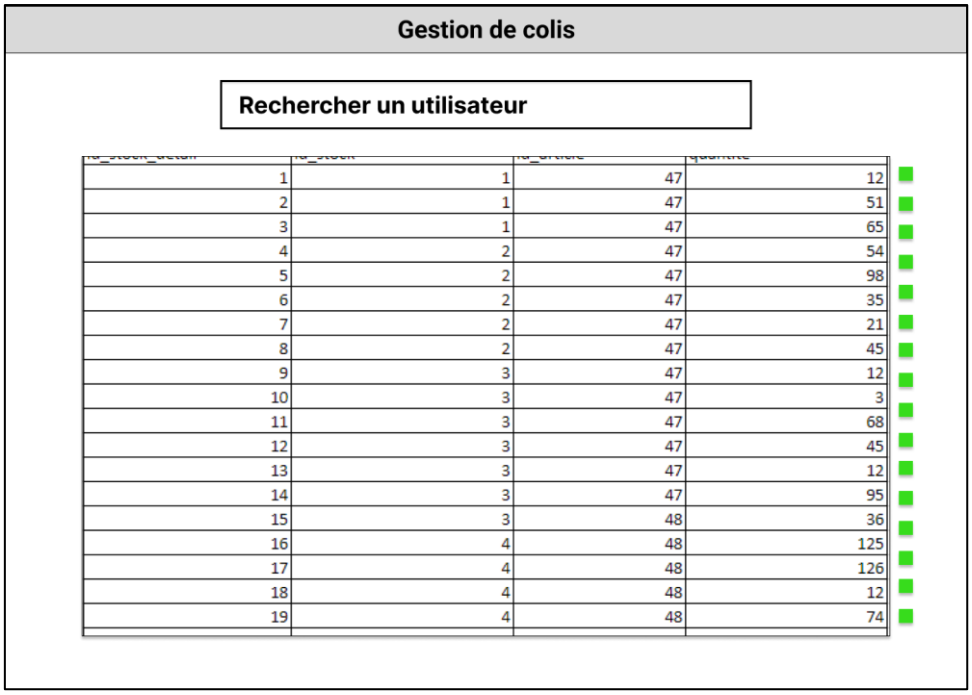
- Pas de perte de données
- Centralisation des données en ligne
- Pas de régression par rapport à l'ancienne application
- Une interface simple et agréable

3.a Maquettes

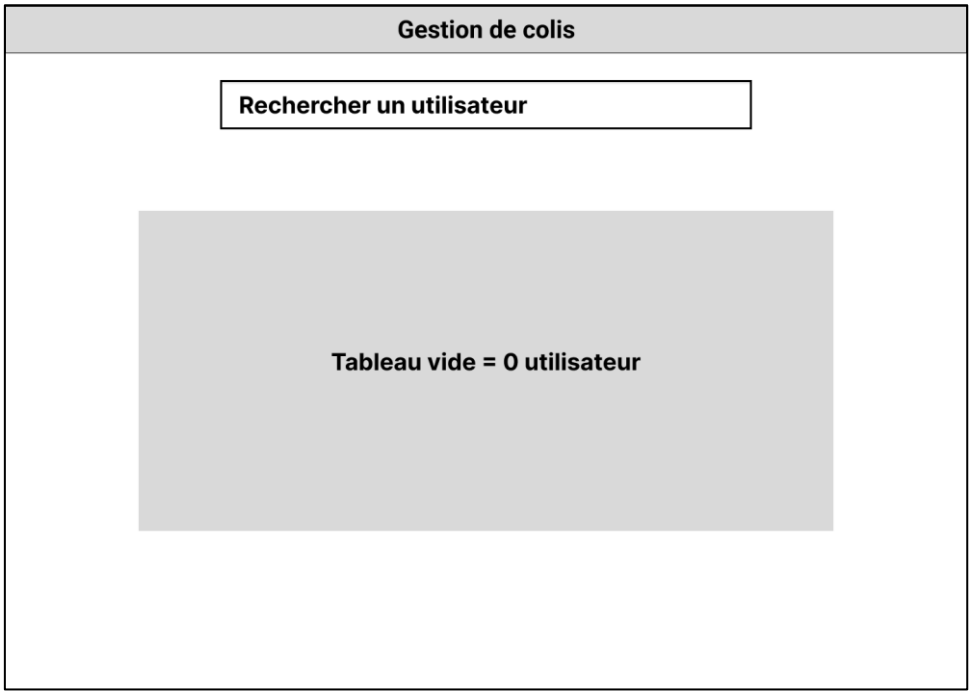
Les maquettes, qui décrivent les interfaces utilisateurs, sont présentées juste après pour illustrer de manière visuelle l'apparence et la disposition des éléments de l'application.



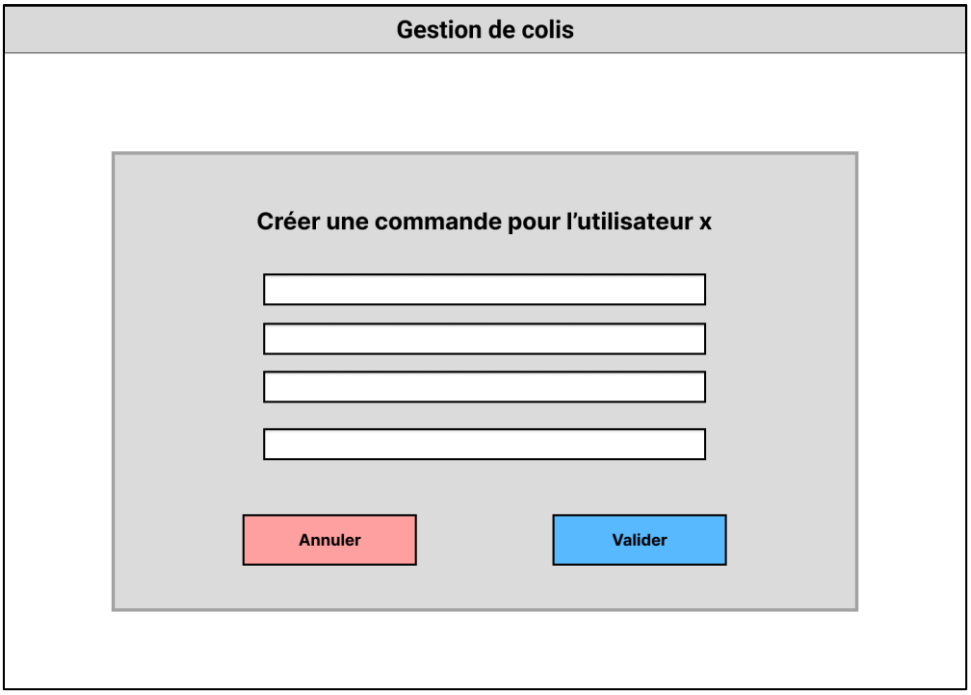
Accueil



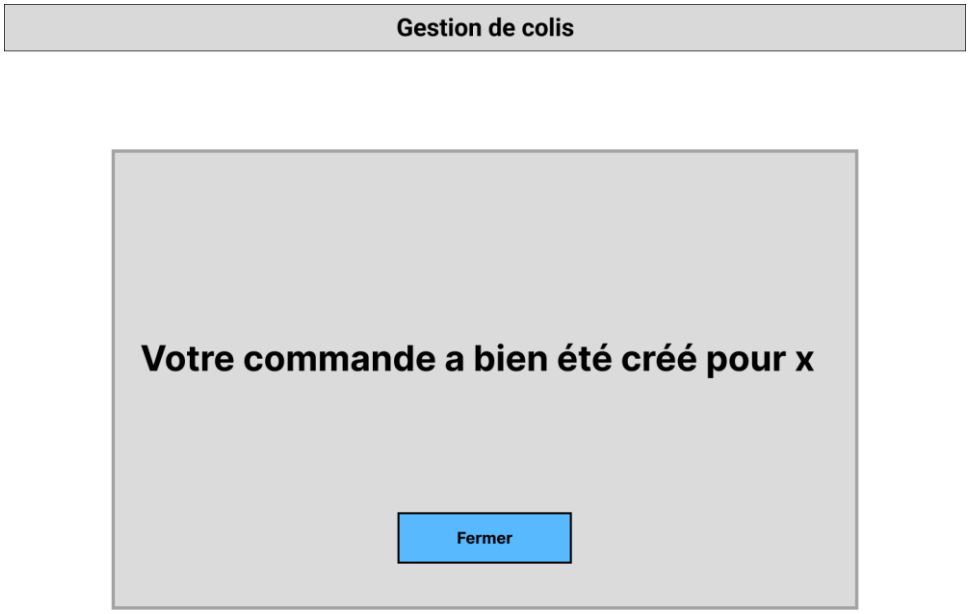
Recherche utilisateur



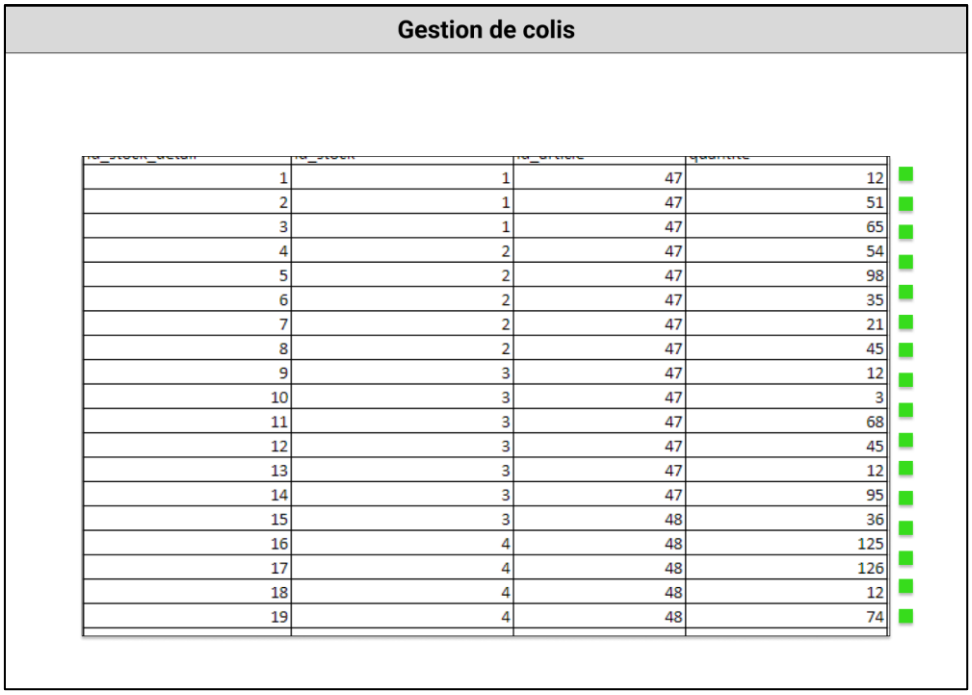
Pas d'utilisateur



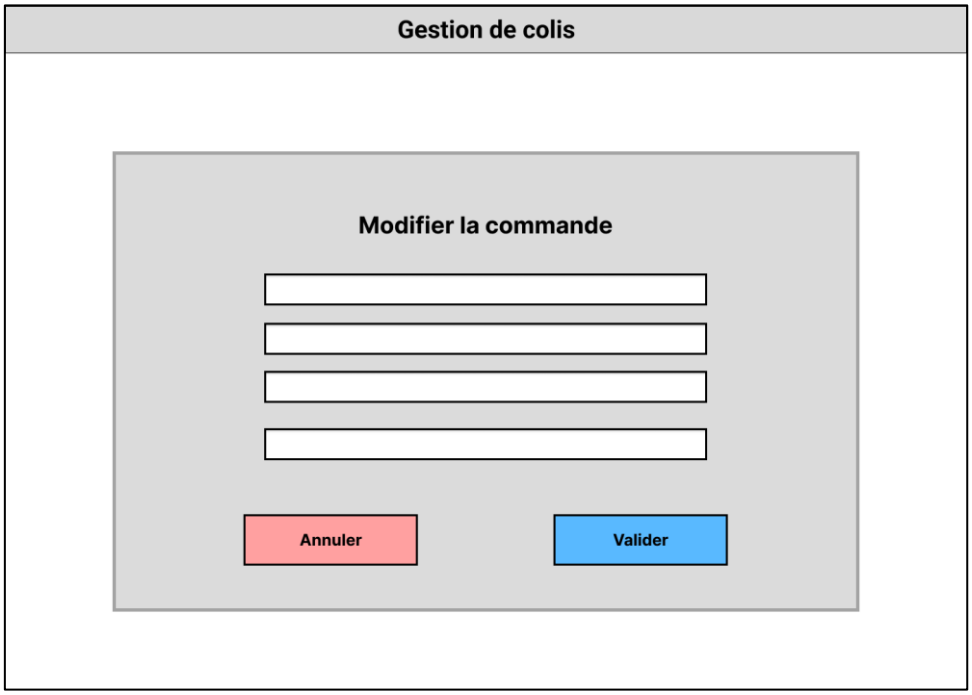
Création de commande pour un utilisateur



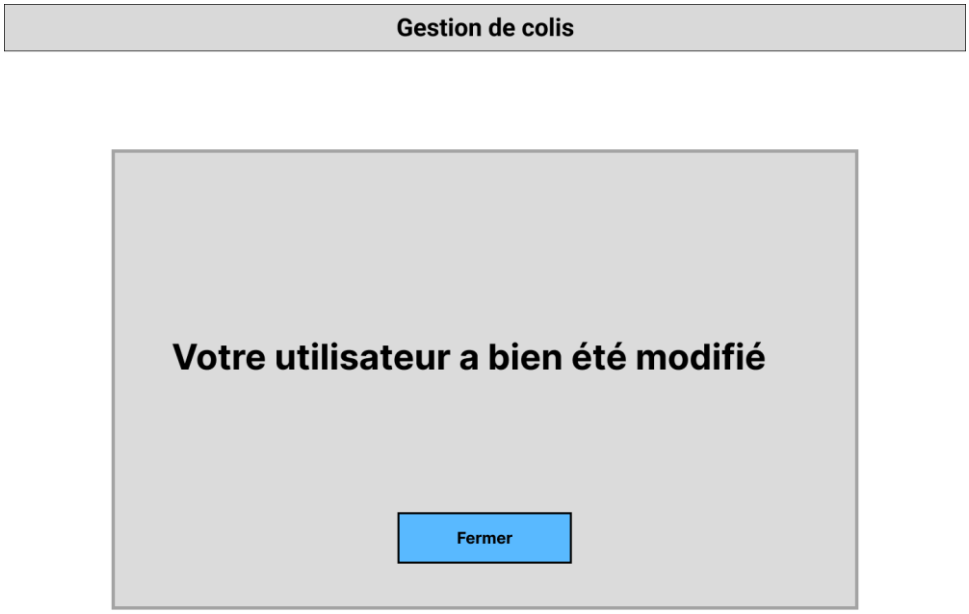
Confirmation création commande



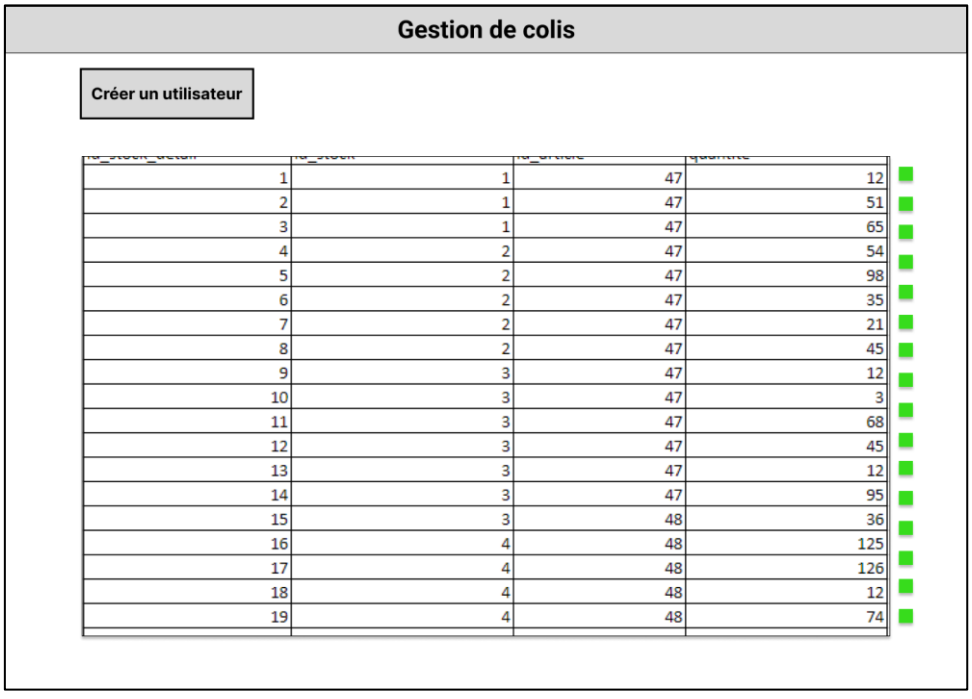
Liste des commandes



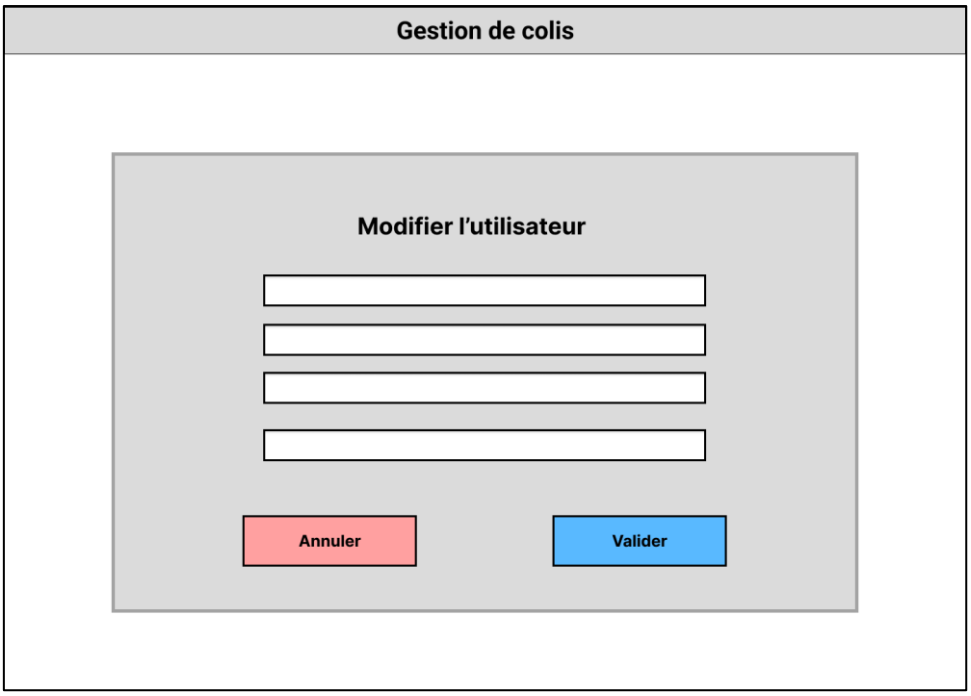
Modification commande



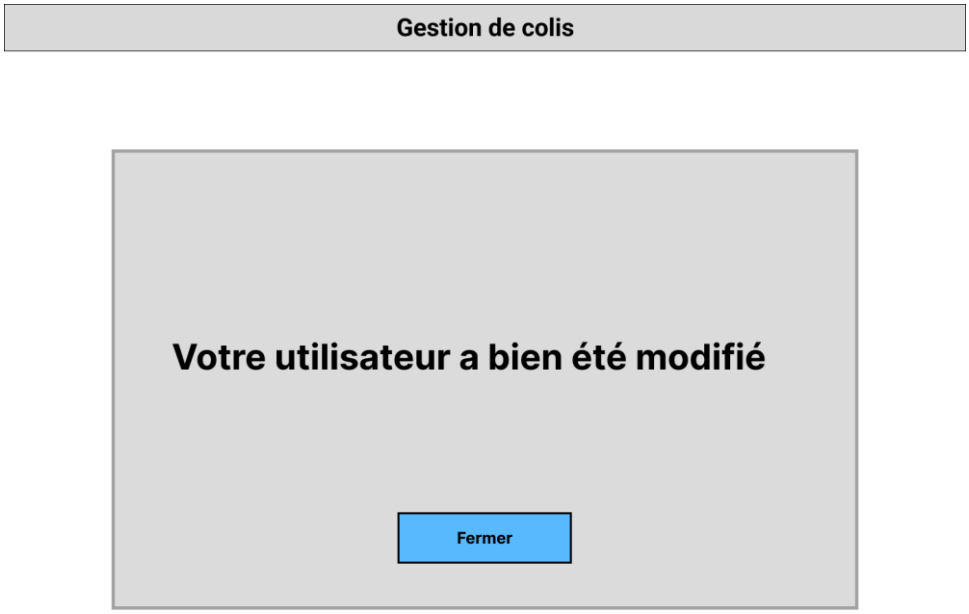
Confirmation modification commande



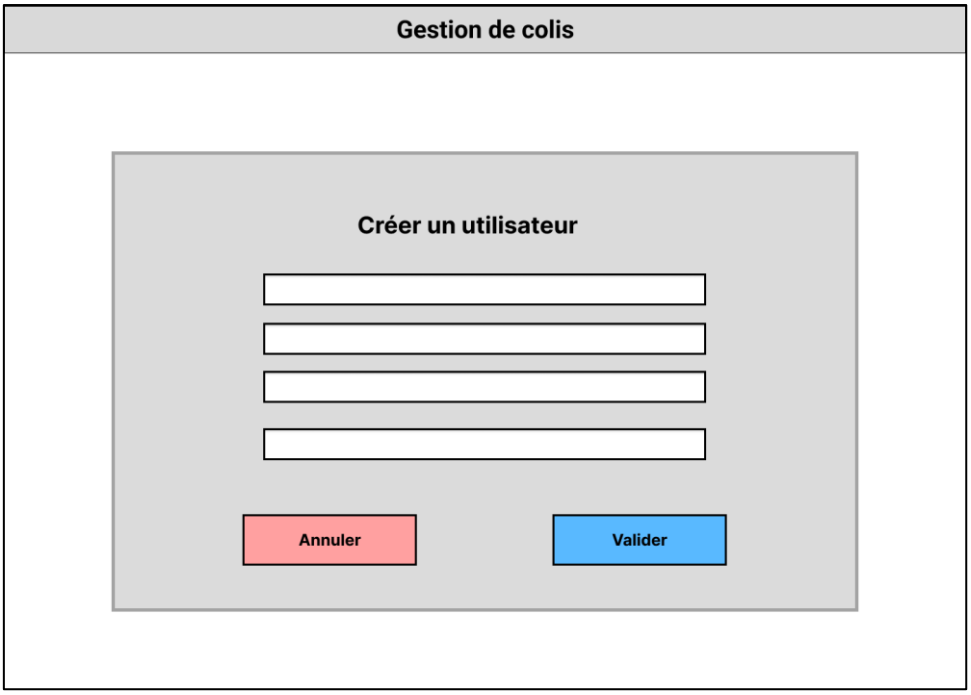
Liste utilisateurs



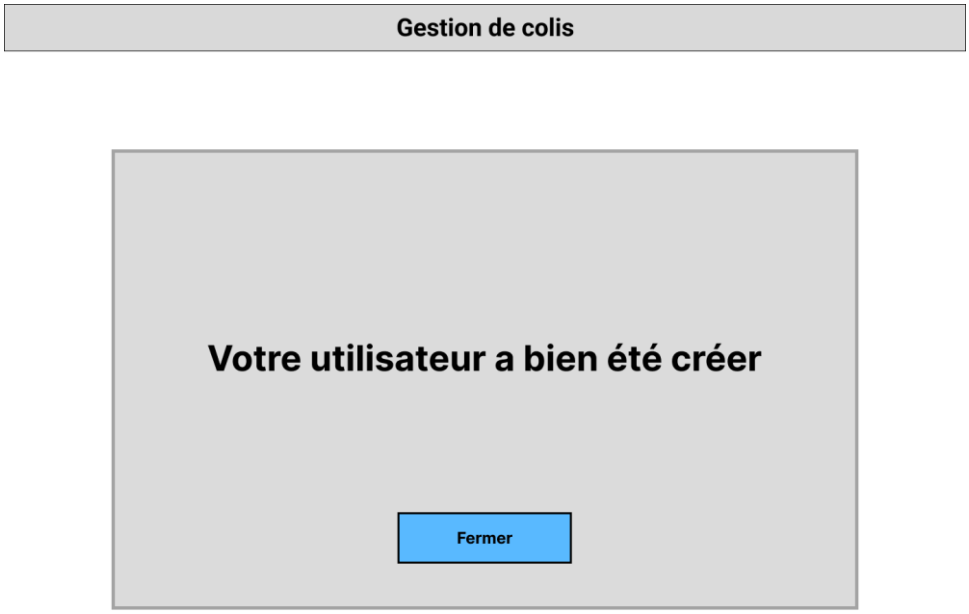
Modification utilisateur



Confirmation modification utilisateur



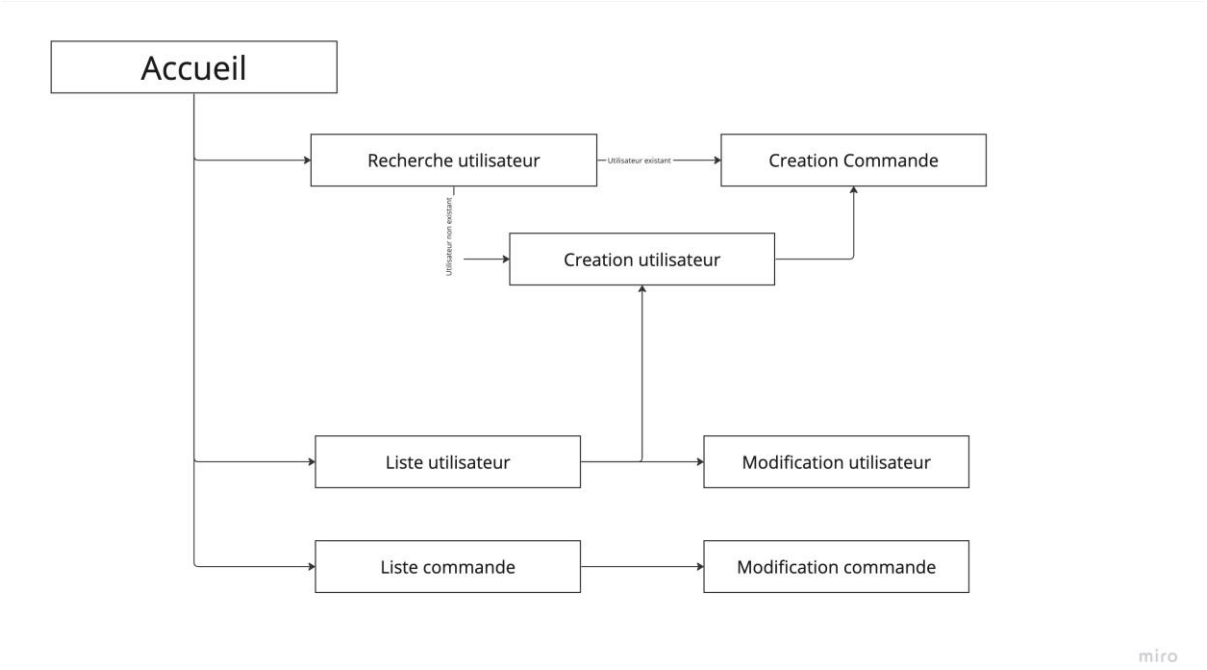
Création utilisateur



Confirmation création utilisateur

3.b Flux de navigation

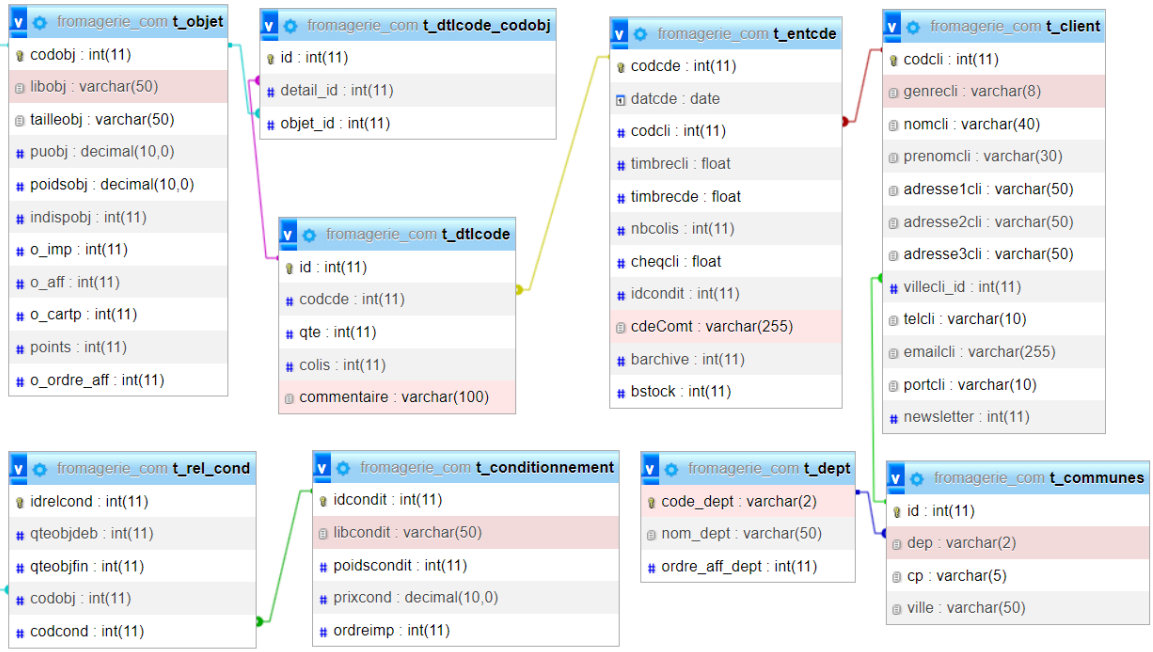
L'image ci-dessous présente les différentes pages de l'application ainsi que la navigation entre elles. Elle offre une vue d'ensemble de l'interface utilisateur et de la façon dont les utilisateurs peuvent interagir avec les différentes fonctionnalités.

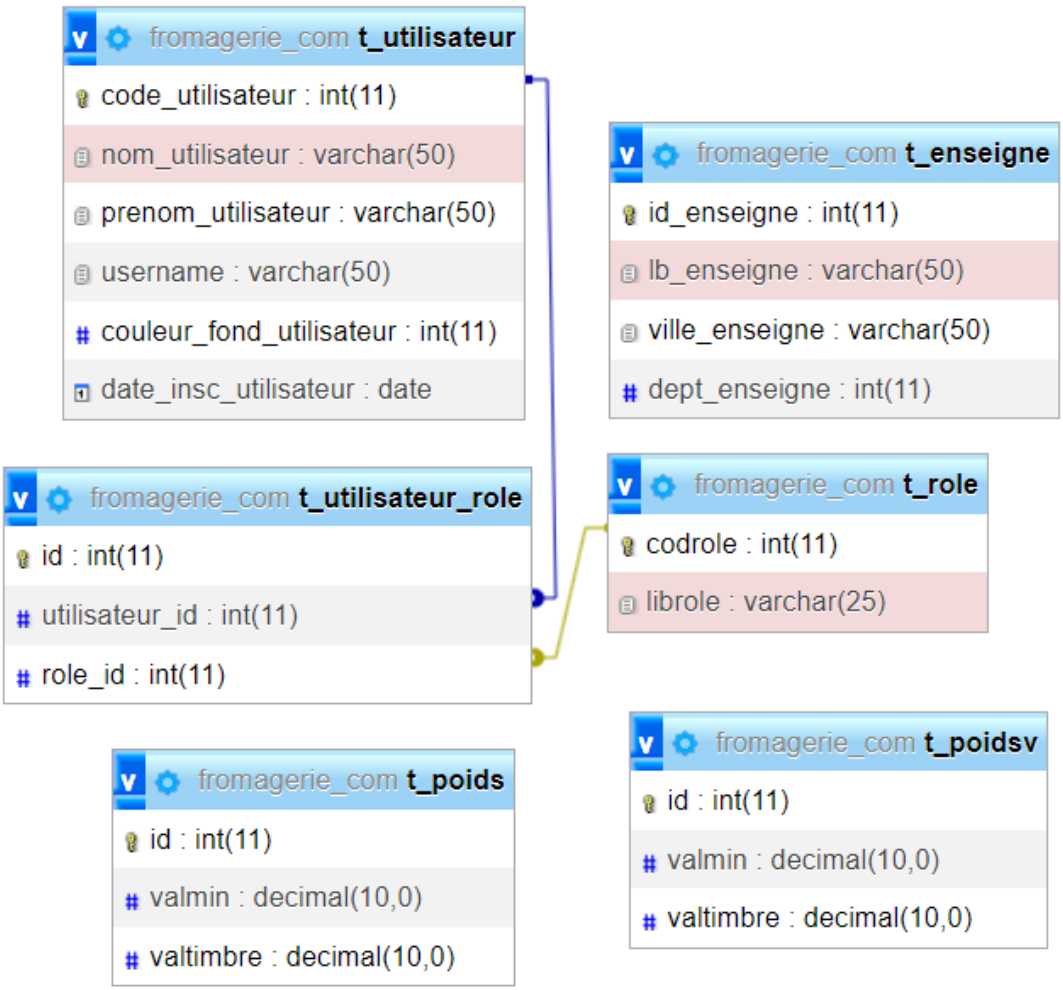


4.a Structure de la base de données

La Base de données (mysql) a été conçu à partir des models fournits par le P.O.

Pour la concevoir nous avons utilisé un ORM pour matérialiser les classes en entités dans la BDD, une api a été créée pour communiquer avec elle.




fromagerie_com t_utilisateur
code_utilisateur : int(11)
nom_utilisateur : varchar(50)
prenom_utilisateur : varchar(50)
username : varchar(50)
couleur_fond_utilisateur : int(11)
date_insc_utilisateur : date
fromagerie_com t_enseigne
id_enseigne : int(11)
lb_enseigne : varchar(50)
ville_enseigne : varchar(50)
dept_enseigne : int(11)
fromagerie_com t_utilisateur_role
id : int(11)
utilisateur_id : int(11)
role_id : int(11)
fromagerie_com t_role
codrole : int(11)
librole : varchar(25)
fromagerie_com t_poids
id : int(11)
valmin : decimal(10,0)
valtimbre : decimal(10,0)
fromagerie_com t_poidsv
id : int(11)
valmin : decimal(10,0)
valtimbre : decimal(10,0)

5.a Architecture matérielle

Dans le cadre de la nouvelle application, nous vous proposons cette infrastructure matérielle :

Back end :

- Petit serveur physique ou virtuel
- Processeur de milieu de gamme (4 cœurs)
- 8 Go de RAM
- Espace de stockage de 100 Go
- Système d'exploitation : Linux (Debian)

Front end :

- Serveur physique ou virtuel de petite taille
- Processeur de milieu de gamme (2 à 4 cœurs)
- 4 à 8 Go de RAM
- Système d'exploitation : Linux (Debian)

Base de données :

- Serveur de base de données de petite taille (MySQL)
- Installation sur le même serveur que le back end
- 10 à 50 Go d'espace de stockage

Réseau :

- Routeur et commutateur réseau
- Configuration des pare-feu et des règles de sécurité réseau appropriées

5.b Environnement technique

Back-End :

Ces packages et dépendances sont utilisés pour mettre en place différentes fonctionnalités et assurer le bon fonctionnement de l'application basée sur Python, FastAPI et SQLAlchemy.

Voici la liste des packages utilisés dans l'application :

alembic==1.11.1

blinker==1.6.2

click==8.1.3

itsdangerous==2.1.2

Jinja2==3.1.2

Mako==1.2.4

MarkupSafe==2.1.2

PyMySQL==1.0.3

SQLAlchemy==2.0.15

typing_extensions==4.6.2

Werkzeug==2.3.4

En outre, les dépendances suivantes sont également utilisées pour le développement de l'application :

fastapi~=0.95.2

pydantic~=1.10.8

setuptools~=65.5.1

python-dotenv~=1.0.0

Front-End (Angular):

Dans l'architecture actuelle, notre application se concentre principalement sur la couche backend, où nous utilisons FastAPI et SQLAlchemy pour gérer les fonctionnalités, les données et les opérations côté serveur.

En utilisant Angular, nous pouvons développer une interface utilisateur dynamique et réactive, offrant une expérience utilisateur optimale. Ces frameworks front-end nous permettent de communiquer avec notre API backend en effectuant des requêtes HTTP vers les endpoints exposés par l'API.

En utilisant les requêtes HTTP, nous pouvons échanger des données entre le front-end et le backend de manière asynchrone, en récupérant des données à afficher, en soumettant des formulaires, en effectuant des opérations CRUD (Create, Read, Update, Delete), et bien plus encore. Cela permet de séparer clairement les responsabilités entre la couche front-end et la couche backend, favorisant une conception modulaire et une évolutivité de l'application.

Le choix d'Angular, il offre une large gamme de fonctionnalités, une grande communauté de développeurs et une documentation abondante pour faciliter le développement de l'interface utilisateur et la communication avec l'API backend.

Hébergement et déploiement :

Les fichiers de l'application seront hébergés sur GitHub, ce qui permettra une gestion centralisée du code source et des versions. De plus, le développement pourra se faire en

local sur les machines des développeurs, offrant ainsi un environnement de développement flexible et personnalisé. Des environnements distincts de test et de production seront également mis en place pour valider les fonctionnalités de l'application, effectuer des tests et permettre un déploiement stable.

Pour assurer une cohérence dans le déploiement de l'application, il est envisagé d'utiliser Docker. Cette technologie de conteneurisation permettra de créer des conteneurs légers et portables qui encapsuleront l'application ainsi que toutes ses dépendances. Ainsi, quelle que soit l'environnement de déploiement, que ce soit en local, en test ou en production, il sera possible d'utiliser les mêmes conteneurs Docker pour garantir une version identique de l'application. Cela facilitera la gestion des dépendances et assurera une exécution cohérente de l'application sur différents systèmes.

5.c Sécurité

Cette documentation présente les mesures de sécurité mises en place dans notre application, en utilisant les technologies suivantes :

Framework de développement sécurisé

L'application est développée en utilisant les bonnes pratiques de sécurité recommandées par le framework FastAPI.

Les fonctionnalités de sécurité intégrées, telles que la gestion des sessions, la protection contre les attaques CSRF (Cross-Site Request Forgery) et la validation des données entrantes, sont utilisées pour renforcer la sécurité de l'application.

Les dépendances sont vérifiées et validées avant leur intégration dans le code de l'application, en s'assurant qu'elles proviennent de sources fiables et sécurisées.

Authentification et autorisation

L'application utilise des mécanismes d'authentification et d'autorisation basés sur les standards de sécurité établis.

Les utilisateurs doivent s'authentifier à l'aide d'identifiants sécurisés tels que des noms d'utilisateur et des mots de passe cryptés.

Les autorisations d'accès aux fonctionnalités sensibles sont gérées de manière granulaire, en s'assurant que seuls les utilisateurs autorisés peuvent accéder à ces ressources.

Chiffrement des données

Les données sensibles stockées dans la base de données sont chiffrées à l'aide d'algorithmes de chiffrement robustes.

Les communications entre le client et le serveur sont sécurisées à l'aide du protocole HTTPS, garantissant le chiffrement des données en transit.

L'application est protégée contre les attaques courantes telles que l'injection SQL, les attaques par force brute et les attaques de contournement d'authentification.

Les entrées utilisateur sont systématiquement validées et filtrées pour prévenir les attaques par injection de code malveillant.

5.d Interactions entre les composants

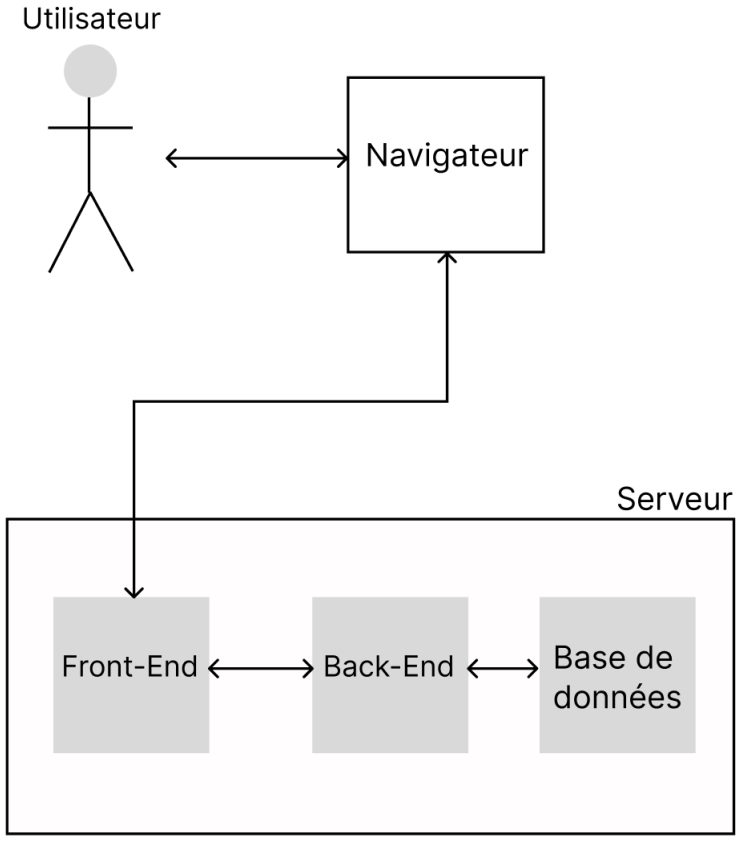


Diagramme de composants

Cette documentation présente les principales interactions entre les composants de notre application, basée sur les spécificités de notre architecture technique.

Backend - Frontend

Le backend de l'application, développé avec FastAPI, expose une API RESTful qui permet au frontend d'interagir avec les données et les fonctionnalités.

Le frontend, développé avec Angular, communique avec l'API backend en effectuant des requêtes HTTP pour récupérer des données, soumettre des formulaires et effectuer des opérations CRUD.

Les données échangées entre le backend et le frontend sont généralement au format JSON.

Backend - Base de données

Le backend de l'application utilise SQLAlchemy, un ORM (Object-Relational Mapping), pour interagir avec la base de données.

Les modèles de données de l'application sont définis en utilisant SQLAlchemy, ce qui permet de créer, lire, mettre à jour et supprimer des enregistrements dans la base de données de manière efficace.

Les requêtes SQL nécessaires sont générées automatiquement par SQLAlchemy à partir des opérations effectuées sur les modèles.

Backend - Services externes (Nouvelles fonctionnalités)

L'application peut intégrer des interactions avec des services externes tels que des API tierces, des services de paiement, des services de messagerie, etc.

A l’avenir ces interactions pourraient être réalisées en utilisant des bibliothèques spécifiques à chaque service ou en effectuant des requêtes HTTP vers les endpoints appropriés.

Authentification et autorisation

Le processus d'authentification et d'autorisation est géré par le backend de l'application.

Lorsqu'un utilisateur tente d'accéder à des ressources protégées, il doit fournir des informations d'identification valides (nom d'utilisateur, mot de passe, jeton d'accès, etc.).

Le backend vérifie ces informations, authentifie l'utilisateur et génère un jeton d'accès sécurisé qui sera utilisé pour les futures requêtes.

Sécurité des données

Les données sensibles, telles que les mots de passe des utilisateurs, sont stockées de manière sécurisée en utilisant des algorithmes de hachage forts.

Les communications entre le frontend, le backend et les services externes sont sécurisées à l'aide du protocole HTTPS, garantissant le chiffrement des données en transit.

Gestion des erreurs et des exceptions

Le backend de l'application met en place un système de gestion des erreurs et des exceptions pour garantir un comportement fiable et prévisible.

Les erreurs sont gérées de manière appropriée, en renvoyant des codes d'état HTTP appropriés, des messages d'erreur clairs et des informations supplémentaires si nécessaire.

En comprenant ces interactions clés entre les différents composants de notre application, nous pouvons assurer un fonctionnement harmonieux et sécurisé de l'ensemble du système.

Test de l'API avec Swagger

Nous utilisons Swagger pour tester et documenter notre API backend. Swagger fournit une interface conviviale pour explorer les différentes routes et méthodes disponibles, ainsi que pour envoyer des requêtes et afficher les réponses.

Lancez l'application backend sur votre environnement local ou distant.

Ouvrez votre navigateur web et accédez à l'URL suivante : http://localhost:8000/docs (remplacez localhost:8000 par l'URL appropriée si vous utilisez un environnement distant).

En utilisant l'interface Swagger, vous pouvez tester toutes les fonctionnalités de notre API backend, vérifier les réponses, les codes de statut et les données renvoyées. Cela facilite également la compréhension de la structure des requêtes et des réponses attendues pour chaque route.

6.b Étapes de développement

1 - Analyse des besoins : Comprendre les exigences du client, les spécifications fonctionnelles et non fonctionnelles de l'application.

2 - Conception : Concevoir l'architecture logicielle, les modèles de données, les interfaces utilisateur et les interactions entre les composants.

3 - Développement backend : Implémenter la logique métier, les API, les services et les interactions avec la base de données en utilisant FastAPI, SQLAlchemy et les autres technologies backend.

4 - Développement frontend : Concevoir et développer l'interface utilisateur en utilisant Angular, en se basant sur les maquettes et les spécifications fonctionnelles.

5 - Tests unitaires : Effectuer des tests unitaires pour vérifier le bon fonctionnement des différentes parties de l'application.

6 - Intégration et tests système : Intégrer les différentes composantes de l'application et effectuer des tests de bout en bout pour valider le bon fonctionnement global de l'application.

7 - Déploiement Test et validation métier : Déployer l'application dans un environnement de test pour une validation finale et effectuer les ajustements nécessaires.

8 - Déploiement Production : Déployer l'application dans un environnement de production.

9 - Documentation : Rédiger la documentation technique, les guides d'utilisation et les instructions d'installation pour faciliter la maintenance et l'utilisation de l'application.

10 - Livraison et support : Livrer l'application au client, assurer un support continu, résoudre les problèmes éventuels et effectuer des mises à jour et des améliorations en fonction des besoins.

6.c Ressources nécessaires

Les ressources nécessaires pour mener à bien le développement de l'application comprennent :

Une équipe de développement compétente comprenant des développeurs backend et frontend, des testeurs, un chef de projet (Scrum master).

Un environnement de développement comprenant des ordinateurs avec les spécifications requises, une connexion Internet stable et les outils de développement appropriés (IDE : PyCharm, contrôle de version, etc.).

Un accès à un serveur de base de données MySQL ou un environnement de base de données en cloud.

Une infrastructure de test pour effectuer des tests unitaires, des tests d'intégration et des tests système.

6.d Échéancier du projet (Planning)

L'échéancier du projet est établi en tenant compte des différentes étapes de développement, des dépendances entre les tâches et des contraintes de temps.

Itération 1 : Gestion de colis

Étape de développement	Durée estimée	Dates prévues
Analyse des besoins	2 jours	12/06/23 - 13/06/23
Conception	2 jours	14/06/23 - 15/06/23
Développement	5 jours	16/06/23 - 22/06/23
Test	1 jour	23/06/23
Mise en production	1 jour	26/06/23

Itération 2 : Gestion de stock

Étape de développement	Durée estimée	Dates prévues
Gestion de stock	10 jours	27/06/23 - 10/07/23

Itération 3 : Administration

Étape de développement	Durée estimée	Dates prévues
Administration	10 jours	11/07/23 - 25/07/23

7.a Contraintes techniques

Les contraintes techniques sont les limitations ou exigences spécifiques liées à notre projet. Elles peuvent inclure des contraintes matérielles, logicielles, de performance, de compatibilité, etc. Les principales contraintes techniques identifiées pour notre projet sont les suivantes :

Environnement de développement : Nous devons assurer que notre application est compatible avec les environnements de développement spécifiés, tels que Python 3.11 et les versions spécifiques des bibliothèques et des frameworks utilisés.

Performance : Nous devons garantir que notre application est performante et réactive, en minimisant les temps de réponse et en optimisant les requêtes à la base de données.

Sécurité : Nous devons mettre en place des mesures de sécurité appropriées pour protéger les données sensibles, prévenir les attaques potentielles et garantir la confidentialité et l'intégrité des informations échangées.

Scalabilité : Nous devons concevoir notre application de manière à pouvoir la faire évoluer facilement en cas d'augmentation de la charge utilisateur ou d'ajout de nouvelles fonctionnalités.

7.b Contraintes de temps et de budget estimé

Nous avons établi un calendrier pour le développement de notre application, en tenant compte des différentes étapes et des délais impartis pour chaque tâche.

Il est essentiel de respecter ces contraintes de temps afin de livrer le projet dans les délais convenus.

De plus, nous avons défini un budget estimé pour le développement de l'application, en tenant compte des ressources nécessaires, des coûts associés et des éventuelles dépenses supplémentaires.

Nous constatons que vos PC sont obsolètes et fonctionnent avec un système d'exploitation dépassé. Nous recommandons de remplacer les PC des utilisateurs. Il est estimé qu'ils nécessitent une configuration similaire à celle-ci :

- Processeur: I7-13700
- RAM: 16Go
- Disque SSD: 512Go

Un exemple d'ordinateur correspondant à cette configuration est le Dell Inspiron 3020 Small Desktop. Il est suffisamment puissant pour répondre à vos besoins et assurer une durabilité à long terme. Son coût unitaire est de 723 € HT. Le prix de notre estimation comprend leur installation.

Taches	Budget Estimé
Equipe Technique	43000 €
Infrastructure SI	5000 €
Frais Annexes	2000 €
Formation (Optionnel)	>= 5000 €
Plan de maintenance Annuel (Optionnel)	20000 €
5 Ordinateurs (Optionnel)	4000 €
Total	50000 - 79000 €

Voici une justification du prix estimé pour chaque tâche :

1. Equipe Technique - 42 000 € :

L'équipe technique est composée de plusieurs membres qui seront impliqués dans le développement du projet. Cela comprend des développeurs et un chef de projet. Le budget estimé de 42 000 € couvre les coûts de salaires et de charges sociales de l'équipe technique pendant la durée du projet. Les coûts peuvent varier en fonction du nombre de membres de l'équipe, de leur expertise, de leur expérience et des normes du marché. Il est important d'avoir une équipe technique compétente et bien organisée pour assurer la réussite du projet.

2. Infrastructure SI - 5 000 € :

La mise en place de l'infrastructure informatique requiert l'achat de serveurs, de matériel réseau, de systèmes de stockage, ainsi que l'installation et la configuration des logiciels nécessaires. Les coûts incluent également les licences logicielles et les éventuels frais de maintenance. Le budget estimé dépend de la complexité de l'infrastructure requise pour le projet.

3. Frais Annexes - 2 000 € :

Les frais annexes englobent les coûts indirects associés au projet, tels que les frais administratifs, les frais de communication, les fournitures de bureau, les frais de déplacement, etc. Ces frais peuvent varier en fonction de la taille du projet et des besoins spécifiques de l'équipe.

4. Formation (Optionnel) - >= 5 000 € :

Si une formation supplémentaire est nécessaire pour l'équipe technique afin de maîtriser de nouvelles technologies ou de se familiariser avec des outils spécifiques utilisés dans le projet, des frais de formation peuvent être inclus. Le budget estimé dépend du nombre de membres de l'équipe à former, de la durée de la formation et des coûts associés à celle-ci.

5. Plan de maintenance Annuel (Optionnel) - 20 000 € :

Si un plan de maintenance annuel est prévu pour assurer le bon fonctionnement continu de l'application après sa livraison, des frais supplémentaires peuvent être nécessaires. Ces frais comprennent la surveillance, la maintenance corrective et évolutive, les mises à jour, les correctifs de sécurité, etc. Le budget estimé dépend de l'étendue de la maintenance prévue et des besoins spécifiques du projet.

Le total estimé du budget pour toutes les tâches s'élève à 74 000 €. Cependant, il convient de noter que ces chiffres sont basés sur une estimation et peuvent varier en fonction des spécificités du projet et des facteurs externes.

7.c Risques identifiés

Lors de l'analyse du projet, nous avons identifié certains risques potentiels qui pourraient avoir un impact sur le développement et la livraison de l'application.

Ces risques comprennent :

1 - Retards dans la fourniture des ressources : Si certaines ressources, telles que les serveurs de base de données ou les outils de développement, ne sont pas disponibles en temps voulu, cela pourrait entraîner des retards dans le développement.

2 - Problèmes de compatibilité : Des problèmes de compatibilité entre les différentes technologies utilisées, par exemple des incompatibilités entre les versions de bibliothèques ou des problèmes de configuration, pourraient entraîner des retards et nécessiter des efforts supplémentaires pour résoudre ces problèmes.

3 - Vulnérabilités de sécurité : Si des vulnérabilités de sécurité sont découvertes dans les technologies utilisées ou si des erreurs de configuration sont commises, cela pourrait compromettre la sécurité de l'application et nécessiter des correctifs urgents.

4 – Non-disponibilité des équipes métiers : pas de Key Users disponibles pour les tests qui entraînerait des retards sur la date de mise en production

7.d Solutions de contournement

Pour atténuer les risques identifiés, nous avons prévu les solutions de contournement suivantes :

Planification et suivi rigoureux : Nous maintiendrons un suivi régulier du projet et nous assurerons que les ressources nécessaires sont disponibles en temps voulu. En cas de retard prévu, nous mettrons en place des mesures correctives pour minimiser l'impact sur le développement.

Tests et vérifications : Nous effectuerons des tests approfondis pour identifier les problèmes de compatibilité dès le début du développement. Nous mettrons également en place des protocoles de sécurité pour détecter et résoudre les vulnérabilités potentielles.

Communication et collaboration : Nous maintiendrons une communication ouverte avec toutes les parties prenantes du projet, en partageant régulièrement les mises à jour sur l'avancement, les problèmes rencontrés et les mesures prises pour y remédier.

En mettant en œuvre ces solutions de contournement et en restant attentifs aux risques potentiels, nous minimiserons les impacts négatifs sur le développement de l'application et garantirons la livraison réussie du projet.

8.a Plan de test

Le plan de test vise à assurer la qualité et la fiabilité de notre application en identifiant et en vérifiant les fonctionnalités clés. Voici les principales étapes de notre plan de test :

Test unitaire : Nous réaliserons des tests unitaires pour chaque composant individuel de notre application, en nous assurant que chaque fonctionnalité est correctement implémentée et fonctionne comme prévu.

Test d'intégration : Nous effectuerons des tests d'intégration pour vérifier le bon fonctionnement des interactions entre les différents composants de notre application, en nous assurant qu'ils communiquent correctement et que les données sont échangées de manière appropriée.

Test de système : Nous réaliserons des tests de système pour évaluer le fonctionnement global de notre application, en vérifiant son comportement dans des situations réelles et en nous assurant que toutes les fonctionnalités principales répondent aux attentes.

Test de performance : Nous effectuerons des tests de performance pour évaluer les performances de notre application, en vérifiant les temps de réponse, la capacité de charge et l'évolutivité dans différentes conditions.

Test de sécurité : Nous réaliserons des tests de sécurité pour identifier les éventuelles vulnérabilités et garantir que notre application est sécurisée contre les attaques potentielles.

Test d'interface utilisateur : Nous effectuerons des tests d'interface utilisateur pour vérifier que toutes les interactions avec l'utilisateur sont intuitives, conviviales et répondent aux besoins du client.

Test de compatibilité : Nous testerons notre application sur différents navigateurs, appareils et systèmes d'exploitation pour nous assurer qu'elle est compatible et fonctionne correctement dans tous les environnements cibles.

8.b Critères de validation

Les critères de validation sont les indicateurs qui nous permettent de déterminer si notre application répond aux exigences et aux attentes du client. Voici les principaux critères de validation pour notre application :

Fonctionnalités complètes : Toutes les fonctionnalités définies dans le cahier des charges doivent être correctement implémentées et fonctionner comme prévu.

Précision des résultats : Les résultats produits par notre application doivent être précis, cohérents et conformes aux attentes du client.

Réactivité et performances : Notre application doit être réactive, offrir des temps de réponse rapides et être capable de gérer une charge utilisateur élevée sans compromettre ses performances.

Sécurité : Toutes les mesures de sécurité définies dans la documentation doivent être implémentées et fonctionner efficacement pour protéger les données et les utilisateurs de notre application.

Interface utilisateur : L'interface utilisateur de notre application doit être intuitive et offrir une expérience utilisateur agréable.

Compatibilité : Notre application doit être compatible avec les navigateurs, les appareils et les systèmes d'exploitation spécifiés dans les exigences techniques.

En réalisant des tests rigoureux et en vérifiant que notre application satisfait ces critères de validation, nous nous assurerons de la qualité et de la conformité de notre solution par rapport aux attentes du client.

9.a Plan de maintenance

Le plan de maintenance est optionnel et en supplément (devis) pour le client, il vise à assurer le bon fonctionnement continu de notre application, à résoudre les problèmes éventuels et à maintenir sa performance et sa sécurité. Voici les principales activités incluses dans notre plan de maintenance :

Surveillance proactive : Nous mettrons en place des outils de surveillance pour suivre les performances, la disponibilité et la sécurité de notre application en temps réel. Cela nous permettra de détecter rapidement les problèmes potentiels et de prendre des mesures préventives pour les résoudre.

Maintenance corrective : En cas de dysfonctionnement ou de bug identifié, nous interviendrons rapidement pour résoudre le problème. Nous effectuerons des analyses approfondies pour identifier la cause et apporter les correctifs nécessaires.

Mises à jour de sécurité : Nous resterons vigilants quant aux nouvelles vulnérabilités et aux correctifs de sécurité. Nous mettrons à jour régulièrement notre application pour garantir que toutes les mesures de sécurité sont à jour et protéger les données et les utilisateurs contre les menaces potentielles.

Maintenance préventive : Nous effectuerons des tâches de maintenance régulières, telles que l'optimisation des performances, la gestion des bases de données, la gestion des sauvegardes et la suppression des fichiers obsolètes. Cela garantira que notre application reste efficace et évitera l'accumulation de problèmes techniques.

PS : Nous offrons une période de maintenance gratuite de 14 jours à partir de la livraison de l'application. Pendant cette période, nous nous engageons à résoudre gratuitement tout dysfonctionnement ou problème identifié.

9.b Évolutivité et futures fonctionnalités

Nous avons conçu notre application avec une architecture extensible et modulaire, ce qui permet d'ajouter facilement de nouvelles fonctionnalités et de faire évoluer l'application pour répondre aux besoins changeants du client. Voici notre approche pour l'évolutivité et les futures fonctionnalités :

- Automatisation de la prise de commande par les clients
- Automatiser le statut de la commande après l’envoi (statut : expédié)

10.a Diagrammes supplémentaires

Diagramme d’activité

Il modélise le flux d'activités et de décisions dans un système, mettant l'accent sur les actions, les transitions et les conditions qui régissent le déroulement des processus. Il permet de représenter visuellement les différentes étapes d'un processus et les décisions prises tout au long de son exécution, facilitant ainsi la compréhension et l'analyse du comportement d'un système.

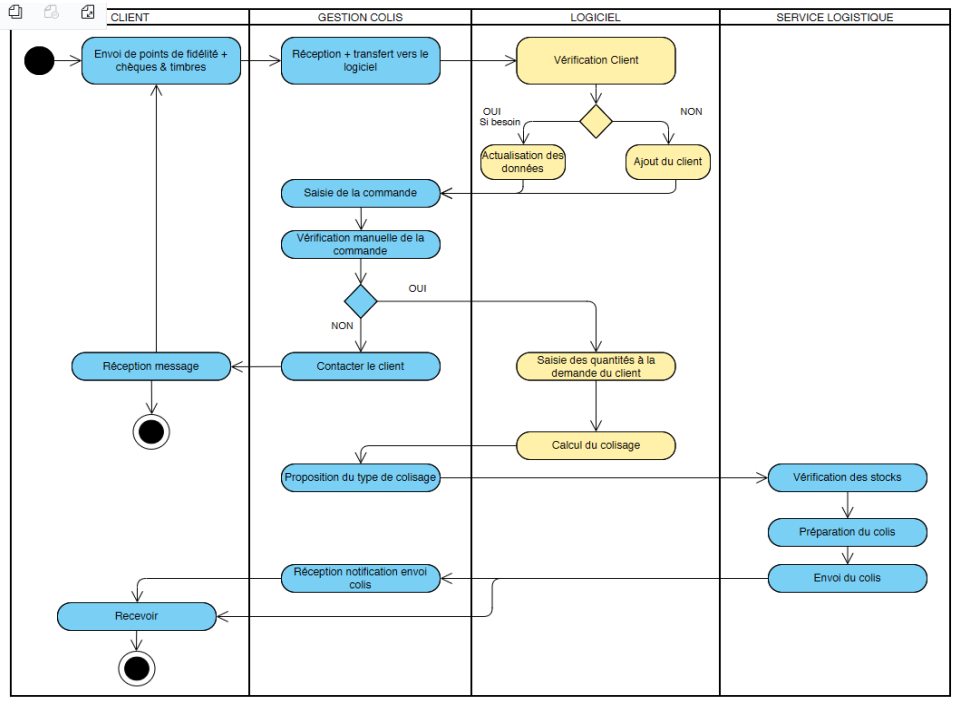
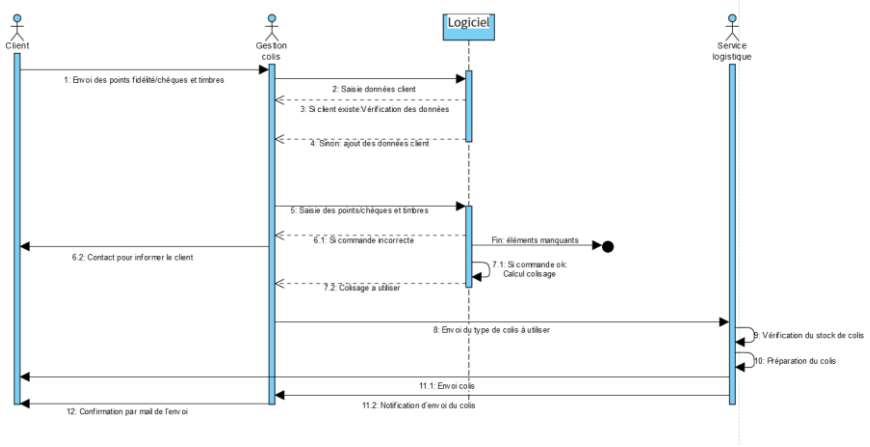


Diagramme de séquence

Il représente la séquence temporelle des interactions entre objets ou acteurs d'un système, permettant ainsi de comprendre et de modéliser le comportement dynamique d'un système logiciel. Il aide à visualiser les messages échangés et à analyser les collaborations entre les différentes entités du système.



10.c Glossaire

Enfin, nous incluons un glossaire des termes techniques et spécifiques utilisés dans ce cahier des charges. Ce glossaire vise à clarifier le sens des termes et à fournir des définitions précises pour une meilleure compréhension globale du document.

1. API : L'acronyme API (Application Programming Interface) désigne un ensemble de règles et de protocoles qui permettent à des logiciels de communiquer entre eux.

2. ORM : L'acronyme ORM (Object-Relational Mapping) fait référence à une technique qui permet de faire le lien entre des objets d'un langage de programmation et une base de données relationnelle, facilitant ainsi les opérations de manipulation des données.

3. JSON : JSON (JavaScript Object Notation) est un format de données léger et facile à lire et à écrire. Il est largement utilisé pour l'échange de données entre un serveur et un client, notamment dans les API RESTful.

4. HTTPS : L'acronyme HTTPS (Hypertext Transfer Protocol Secure) indique que les communications entre un client et un serveur sont sécurisées par chiffrement. Cela garantit la confidentialité et l'intégrité des données échangées.

5. CRUD : L'acronyme CRUD (Create, Read, Update, Delete) désigne les opérations de base effectuées sur des données persistantes. Ces opérations incluent la création, la lecture, la mise à jour et la suppression d'enregistrements dans une base de données.

6. Jeton d'accès (Token) : Un jeton d'accès est une chaîne de caractères utilisée pour authentifier un utilisateur ou une application lorsqu'il effectue des requêtes vers une API. Il permet de vérifier les autorisations et les droits d'accès associés à cet utilisateur ou cette application.

7. Hachage : Le hachage est une fonction mathématique qui transforme une donnée en une empreinte numérique unique. Cela permet de stocker de manière sécurisée des informations sensibles, comme les mots de passe des utilisateurs, sans les exposer directement.

8. Code d'état HTTP : Les codes d'état HTTP sont des chiffres à trois chiffres envoyés par un serveur en réponse à une requête effectuée par un client. Ils indiquent le statut de la requête et fournissent des informations sur le résultat de l'opération.

9. Docker : Docker est une plate-forme open-source qui permet de créer, déployer et exécuter des applications dans des conteneurs légers et isolés. Cela facilite le déploiement et la gestion d'applications dans des environnements variés.

10. Backend : Le backend désigne la partie de l'application qui est responsable du traitement des requêtes et de la gestion des données. Il est généralement situé du côté du serveur et communique avec le frontend.

11. Frontend : Le frontend désigne la partie de l'application qui est visible et accessible par les utilisateurs. Il est responsable de l'interface utilisateur, de l'affichage des données et de l'interaction avec le backend.

12. Architecture logicielle : L'architecture logicielle fait référence à la conception et à l'organisation générale d'un système logiciel, en définissant la structure, les composants, les interactions et les principes qui régissent l'application.

13. Base de données : Une base de données est un système de stockage organisé et structuré qui permet de stocker, gérer et manipuler des données de manière efficace.

14. ORM : Acronyme de "Object-Relational Mapping". C'est une technique de programmation qui permet de représenter les objets d'une application dans une base de données relationnelle et de gérer leur persistance.

Réalisé par :

Clémentine Ducournau - Florian Jordany - Axel Bernat

Nous attestons avoir réalisé ce cahier des charges conformément aux exigences et spécifications définies par le client, Digicheese. Nous confirmons également que toutes les informations fournies dans ce document sont complètes, précises et reflètent les besoins du projet tels que discutés avec le client.

Date :

Clementine Ducournau Signature :

Date :

Client Signature :

Date :

Florian Jordany Signature :

Date :

Axel Bernat Signature :