Elective développement web avancé

Membres du groupe :

- GUELLATI Mohamed
- Zamouche Nadir
- HAMHAMI El Rayan
- MESSAOUI Afnane

RAPPORT

Table des matières :

I. Présentation de l'équipe :

- **GUELLATI Mohamed :** "Je suis développeur web spécialisé en Node.js et React.js. J'ai une solide expérience dans le développement de microservices et la création d'interfaces utilisateur interactives et réactives. Mon expertise inclut également la gestion des bases de données MongoDB et l'implémentation de pipelines CI/CD avec GitHub. "
- **Zamouche Nadir :** " Je suis développeur d'applications mobiles spécialisé en React Native. J'ai travaillé sur plusieurs projets où j'ai utilisé des composants UI avancés et géré des interactions de données complexes avec Axios. Mon expérience comprend également l'optimisation des performances d'applications et l'intégration de services tiers. "
- **HAMHAMI El Rayan :** "Je suis développeur back-end spécialisé en Node.js et Express.js. Je suis également compétent en gestion de bases de données relationnelles comme SQL Server et MySQL. Mon expertise comprend la conception et le développement d'API RESTful, ainsi que la mise en place de conteneurisation avec Docker et l'orchestration avec Kubernetes. "
- **MESSAOUI Afnane :** "Je suis développeuse full-stack avec une expertise en TypeScript et dans l'optimisation des routes de livraison en utilisant des algorithmes comme l'Ant Colony Optimization (ACO). J'ai également une expérience approfondie dans la gestion des services cloud et l'intégration de solutions de microservices avec une attention particulière aux contraintes de fenêtres temporelles."

II. Reformulation du besoin :

Notre projet vise à transformer la gestion des offres commerciales dans le domaine de la restauration grâce à une plateforme logicielle distribuée innovante. En tenant compte des besoins diversifiés de multiples utilisateurs, cette plateforme est conçue pour offrir une gamme de services variés adaptés à chaque profil utilisateur.

Parmi les utilisateurs, nous trouvons l'utilisateur final, le restaurateur, le livreur, le développeur tiers, ainsi que les services commerciaux et techniques. Chaque groupe bénéficie d'outils et de fonctionnalités spécifiques pour optimiser leurs interactions et leurs processus opérationnels.

Le but de ce projet est de concevoir, réaliser, déployer, tester et utiliser cette plateforme de manière efficace et intuitive. Notre ambition est de créer un écosystème intégré qui non seulement améliore l'efficacité opérationnelle, mais aussi accroît la satisfaction des utilisateurs. Grâce à l'utilisation des technologies

modernes et aux meilleures pratiques de développement logiciel, nous sommes déterminés à fournir une solution évolutive et performante, répondant aux exigences du marché actuel de la restauration.

III. Identification des points critiques et analyse des risques :

- Développer une plateforme logicielle distribuée en utilisant une architecture basée sur les microservices.
- Créer cinq applications distinctes pour différents utilisateurs : livreur, restaurateur, client final, commercial/technique, développeur tiers.
- Structurer chaque application autour de quatre couches principales : application, services, composants et données.
- Intégrer deux bases de données distinctes : SQL (utilisant SQL Server) pour la gestion des utilisateurs et NoSQL (utilisant MongoDB) pour d'autres données.

IV. Analyse du cahier des charges :

L'objectif de notre projet est de développer une plateforme logicielle innovante pour la gestion des commandes et des livraisons de repas. Cette plateforme doit répondre aux besoins variés de plusieurs types d'utilisateurs : les clients finaux, les restaurateurs, les livreurs, les développeurs tiers, ainsi que les services commercial et technique.

Pour garantir que chaque groupe d'utilisateurs bénéficie d'une expérience optimale, nous avons identifié et analysé les besoins spécifiques et les fonctionnalités clés nécessaires pour chacun. Cette analyse nous permettra de concevoir une solution qui est non seulement fonctionnelle et efficace, mais aussi intuitive et adaptée à ses différents utilisateurs.

- **Utilisateur Final :**
- Besoins principaux : gestion de compte, commandes, historique, suivi de livraison, parrainage, notifications.
- Fonctionnalités clés : gestion de compte, commandes, suivi de livraison, notifications.
- **Restaurateur :**
- Besoins principaux : gestion de compte, menu, commandes, statistiques, parrainage, notifications.
- Fonctionnalités clés : gestion de compte, menu, commandes, statistiques, notifications.

- **Livreur :**
- Besoins principaux : gestion de compte, gestion des livraisons, QR code, parrainage, notifications.
- Fonctionnalités clés : gestion de compte, gestion des livraisons, QR code, notifications.
- **Développeur Tiers :**
- Besoins principaux : gestion de compte, intégration des composants, utilisation de l'API, téléchargement de composants.
- Fonctionnalités clés : intégration des composants, utilisation de l'API, téléchargement de composants.
- **Service Commercial :**
- Besoins principaux : gestion des comptes clients, suivi des processus de commande, notifications.
- Fonctionnalités clés : gestion des comptes clients, suivi des processus de commande, notifications.
- **Service Technique :**
- Besoins principaux : gestion des composants, surveillance des performances, déploiement de services.
- Fonctionnalités clés : gestion des composants, surveillance des performances, déploiement de services.

V. Architecture :

Une architecture logicielle est une représentation des éléments et des interactions d'un ou plusieurs services informatiques. Quatre types d'architectures logicielles sont couramment utilisés : le monolithique, l'orientée services (SOA), les microservices et le serverless.

- Monolithique : Cette architecture regroupe tous les composants d'une application en un seul programme sur une plate-forme unique.

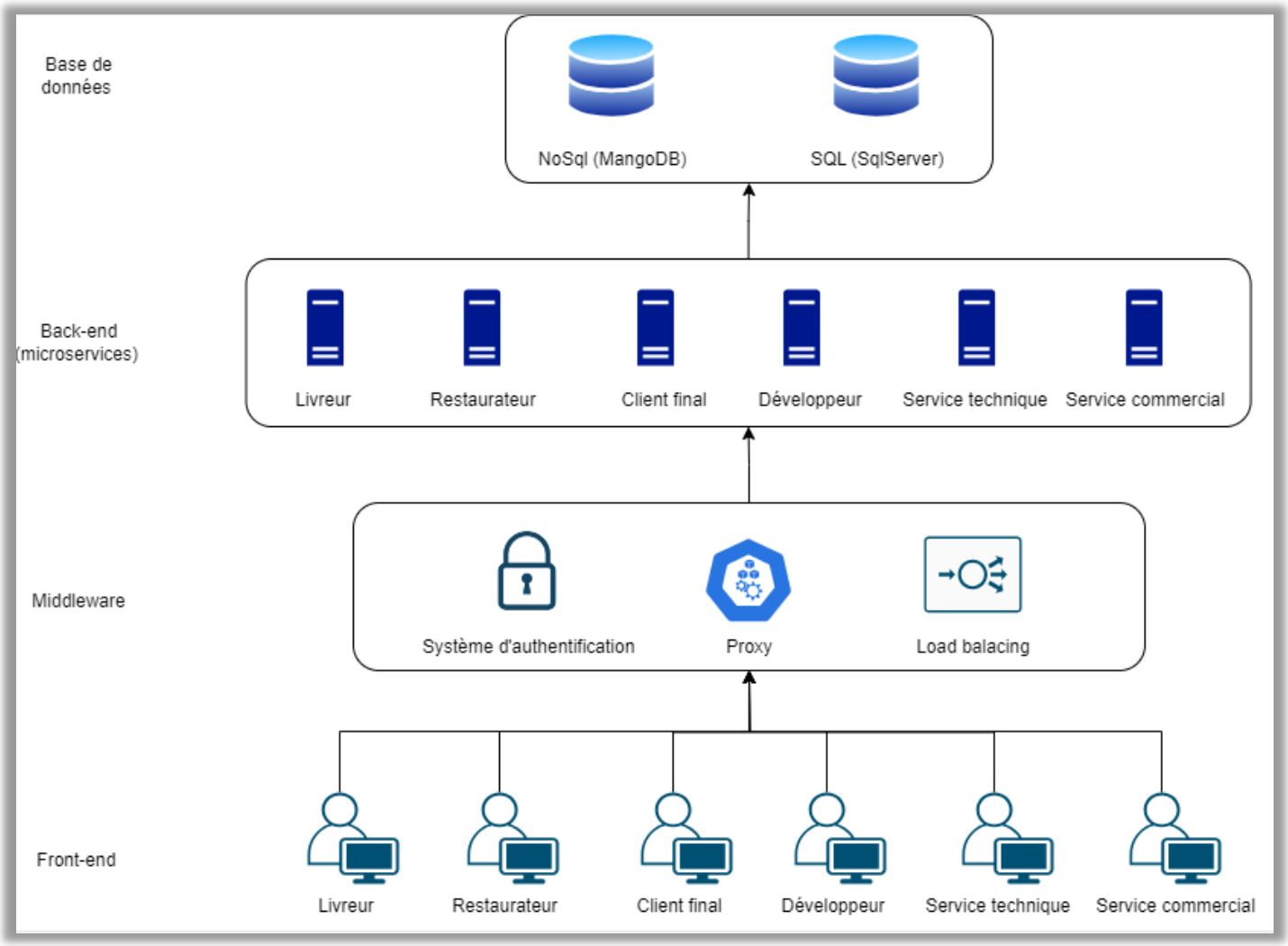
- SOA : L'architecture orientée services se compose d'agents logiciels indépendants et faiblement couplés, offrant des services distincts à d'autres parties de l'application.

- Microservices : Les microservices sont des composants autonomes qui travaillent ensemble pour constituer une application. Contrairement aux monolithes, ils sont conçus de manière indépendante et communiquent via des API.

- Serverless : Cette approche de cloud computing permet de développer et d'exécuter des applications sans gérer l'infrastructure sous-jacente. Le code est déployé sur des serveurs gérés par le fournisseur de cloud, libérant ainsi les développeurs de la gestion des serveurs.

Dans notre cas, nous avons opté pour l'architecture des microservices. Cette décision repose sur plusieurs avantages, notamment la scalabilité et l'évolutivité. En effet, avec les microservices, chaque composant peut

être développé, déployé et mis à jour indépendamment, ce qui assure la continuité du service même en cas de maintenance ou de panne d'un service spécifique.



VI. Maquette de l'application finale :

Nous avons créé une maquette pour chaque type d'utilisateur en fonction du cahier des charges. Chaque maquette offre une interface adaptée aux besoins spécifiques de chaque utilisateur, allant de la gestion des comptes à la consultation des commandes et des livraisons.

VII. Présentation d'un plan d'intégration et de déploiement continu :

- Chaque module de l'architecture, du front-end aux microservices virtualisés, nécessite un pipeline CI dédié.
- Pour le front-end, le pipeline CI comprendra des étapes telles que la compilation des assets, l'exécution des tests unitaires JavaScript, et le déploiement sur un environnement de test.

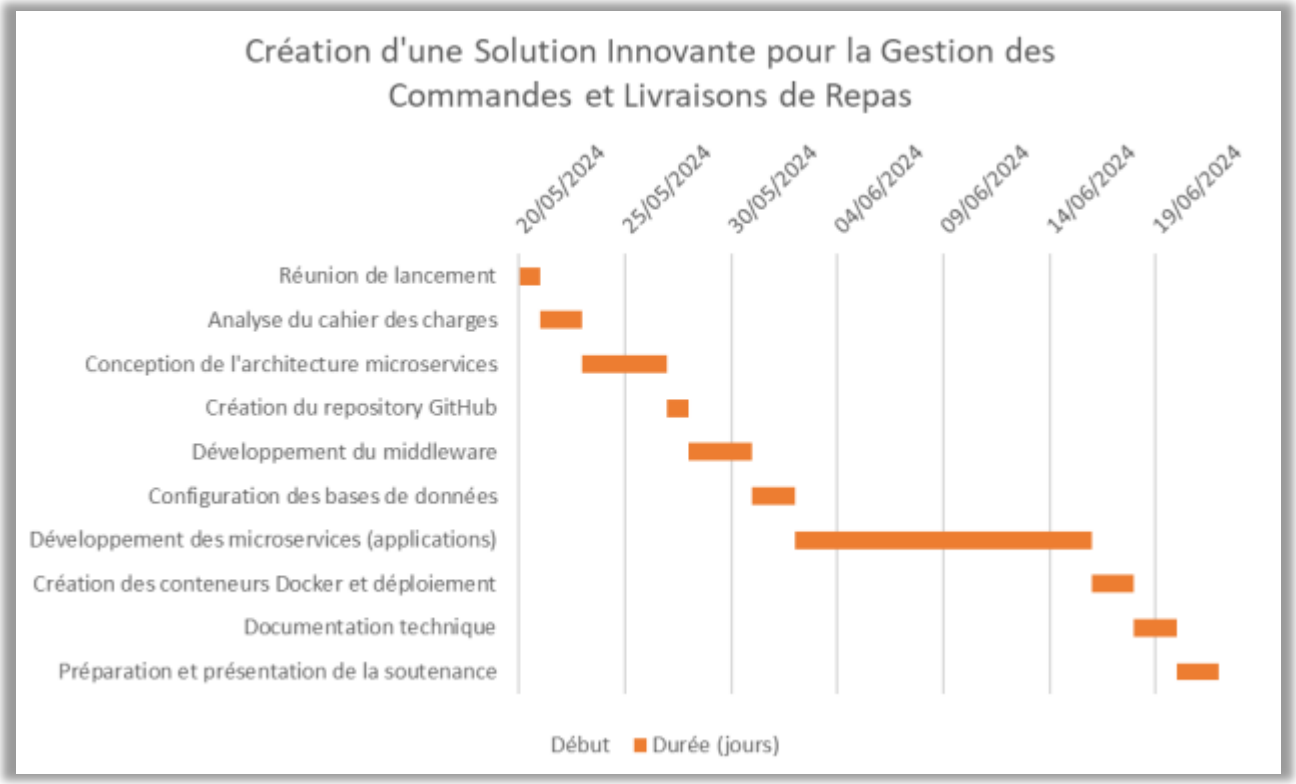
- Pour les microservices virtualisés, le pipeline CI inclura des étapes telles que la construction de l'image Docker, l'exécution des tests d'intégration avec les dépendances simulées, et la publication de l'image sur un registre Docker.

- Chaque couche de l'architecture nécessite un pipeline CD pour garantir un déploiement rapide et fiable des services et des composants.
- Le pipeline CD pour la couche services comprendra des étapes telles que la construction de l'application, l'exécution des tests automatisés, et le déploiement sur un environnement de préproduction pour une validation finale.

- Pour les composants virtualisés, le pipeline CD inclura des étapes similaires, mais en se concentrant sur la construction de l'image Docker et le déploiement sur des clusters Kubernetes.

VIII. Planning prévisionnel :

Nous avons conçu ce diagramme de Gantt pour planifier et suivre le développement de notre solution pour la gestion des commandes et livraisons de repas. Nous avons commencé par une réunion de lancement pour nous aligner sur les objectifs. Ensuite, nous avons analysé le cahier des charges, élaboré l'architecture microservices et mis en place notre repository GitHub. Le développement s'est ensuite concentré sur le middleware, la configuration des bases de données et les microservices. Nous avons terminé par la création des conteneurs Docker, la documentation technique et la préparation de notre soutenance. Grâce à cette planification rigoureuse, nous pensons pouvoir progresser de manière organisée et efficace vers notre date limite du 22 juin.



Création d'une Solution Innovante pour la Gestion des Commandes et Livraisons de Repas

Tâches	Début	Durée (jours)	Fin
Réunion de lancement	20/05/2024	1	21/05/2024
Analyse du cahier des charges	21/05/2024	1	22/05/2024
Maquettes des applications	22/05/2024	2	24/05/2024
Conception de l'architecture microservices	24/05/2024	4	28/05/2024
Création du repository GitHub	28/05/2024	1	29/05/2024
Développement du middleware	29/05/2024	3	01/06/2024
Configuration des bases de données	01/06/2024	2	03/06/2024
Développement des microservices (applications)	03/06/2024	14	17/06/2024
Création des conteneurs Docker et déploiement	17/06/2024	2	19/06/2024
Documentation technique	19/06/2024	2	21/06/2024
Préparation et présentation de la soutenance	21/06/2024	1	22/06/2024

IX. Questions à débattre :

1. Choix d'architecture : Devons-nous privilégier une architecture orientée service ou une architecture orientée microservices pour notre solution informatique ?

2. Technologies à utiliser : Quels langages de programmation, frameworks et outils devrions-nous utiliser pour développer chaque composant de l'application ?

3. Gestion de la sécurité : Comment devons-nous gérer la sécurité des données et des communications entre les différents modules de l'application ?

4. Interchangeabilité des composants : Comment garantirons-nous que chaque composant de l'application soit interchangeable sur toutes les plateformes développées, comme spécifié dans les spécifications techniques ?

5. Normalisation des messages : Comment devons-nous normaliser les messages de retour et les éléments UI pour assurer une expérience utilisateur cohérente et intuitive ?

6. Stockage des données : Quelle est la meilleure approche pour le stockage des données applicatives, en particulier concernant l'utilisation d'une base de données NoSQL comme MongoDB ?

7. Documentation des APIs : Comment devons-nous organiser et rédiger la documentation des APIs pour permettre à tout développeur de les consommer efficacement ?

8. Déploiement et supervision : Quels outils et processus devons-nous mettre en place pour déployer et superviser efficacement notre plateforme logicielle distribuée ?