Titre du Projet			Réf. Du projet
GitMeMoney			02_GMM_01
Historique	Guide de recommandations		
Version	Rédacteur	Modification	Date
1.0	Jules MESSOLO		08/08/2021

Table des matières

1. Description générale de l'architecture....3
2. Technologies préconisés....4
3. Méthode de développement : l'agilité avec Scrum....4
4. Environnement de travail....5
5. Mise en place un pipeline CI/CD....5
6. Convention de Codage....6
7. Amélioration continue des pratiques de code....7
8. Les Tests, privilégier une approche TDD....7
9. Suivez le story-board de l'application «GitMeMoney»....8
10. Implémentation des microservices....8
11. Implémentation de la Gateway....8
12. Implémentation de la couche front-end....9
13. Suivi à l'aide d'outils de monitoring....9

1. Description générale de l'architecture

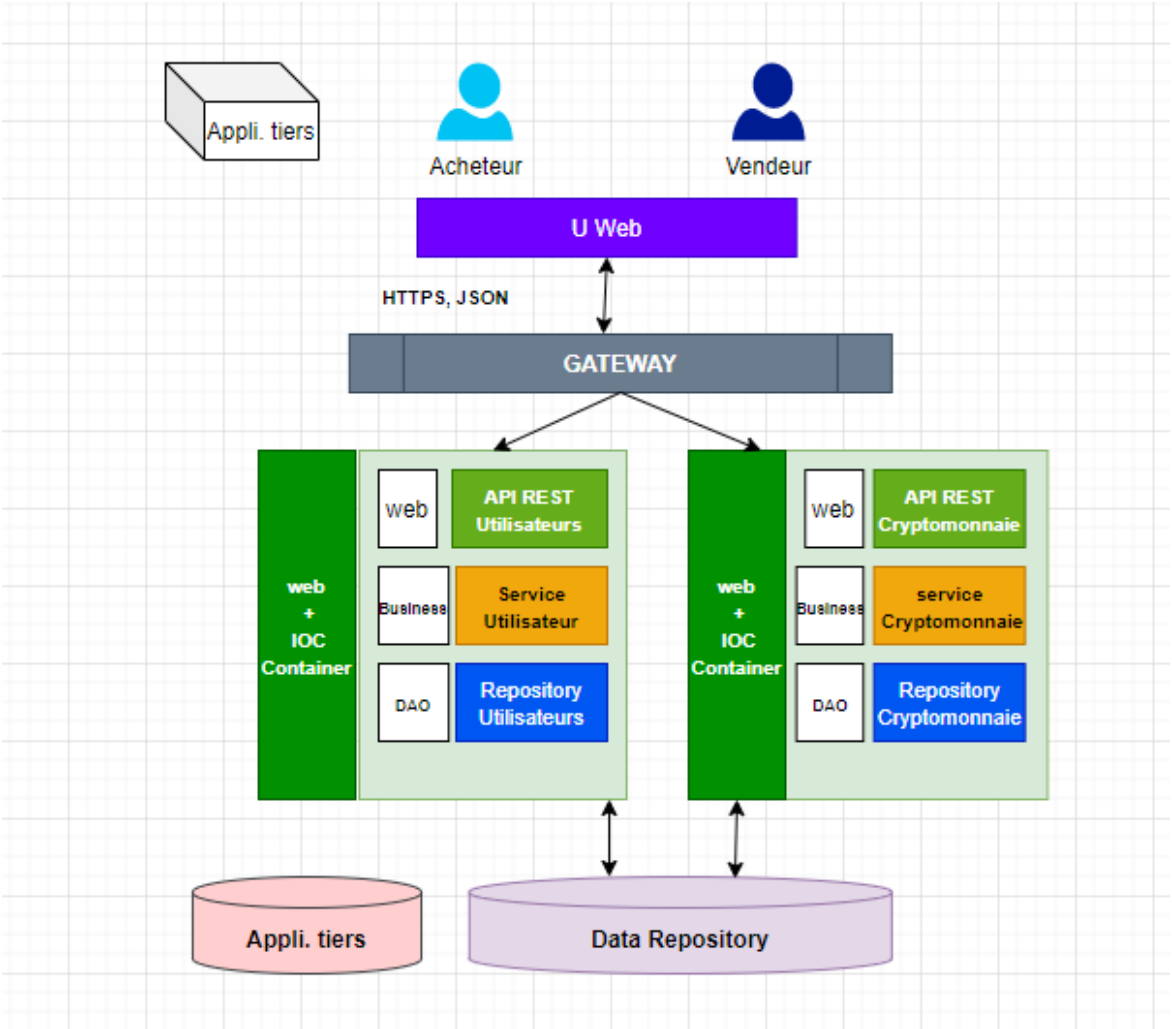
Architecture Microservice qui se décompose en trois parties :

• **La partie Front-end** ou interface utilisateur basé sur le framework Angular qui permet aux utilisateurs de se connecter à l'application et accessible depuis un navigateur web.

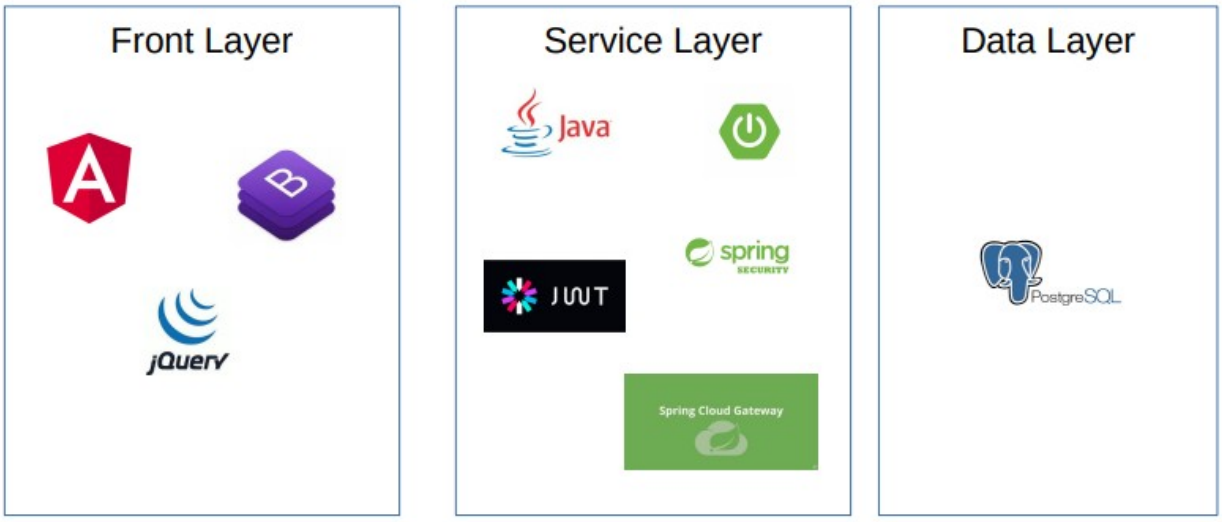
• **La partie Back-end** ou la couche services basée sur Spring, Spring Data JPA et Hibernate. Elle est découpée en microservices exposant des API Restful.

Chaque Microservice représente un groupe de fonctionnalités indépendants de l'application accessibles via des requêtes HTTPS.

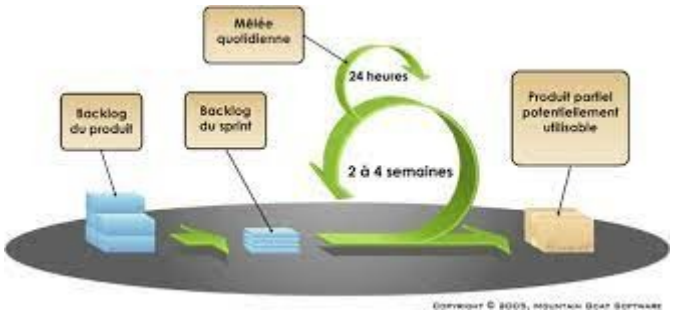
• **La partie Data** avec deux bases de données relationnelles



2. Technologies préconisés



3. Méthode de développement : l'agilité avec Scrum.



L'implémentation du projet est découpé en sprint (2 à 3 semaine)pour permettre au équipe de développer les fonctionnalités présentent dans le Tableau kanban.

Chaque sprint débute par un ***sprint planning***, afin de parcourir le contenu du sprint et les objectifs à atteindre. Occasion de rappeler des règles de fonctionnement au sein de l'équipe.

Durant le sprint des courtes réunions d'échange seront effectués chaque jours, d'où la nécessité d'être ponctuel au rendez-vous du ***Daily.*** C'est la réunion du matin que permettra à l'équipe de faire un point sur les taches de la journées et les difficultés rencontrés ou avenir.

Chaque fin de sprint sera suivi d'une «***Séquence retro***» d'un jour permettant à l'équipe de traiter les éventuelles retours du client qui ont été faite lors des démonstration de fin de

sprint en présence de toute l'équipe. Période durant laquelle l'équipe pourra à travers les réunions «***sprint retro***» faire un point sur les méthodes de fonctionnement de l'équipe et le déroulé du sprint passé.

4. Environnement de travail

Pour éviter l'interruption des services et assurer ***implémentation***, le ***déploiement*** et la ***maintenance*** de la solution, il est recommander de mettre en place différents environnements distincts et bien cloisonnés. Avec des accès limités aux personnes concernés A savoir :

- DEV :un environnement de développement dédié aux développeurs,
- TEST-SIT: un environnement de test, dédié aux tests et démonstration de fin de sprints
- UAT: Un environnement de pré-production (User Acceptance Test) permettant d'avoir un environnement qui se rapproche le plus de la production.Il permettra d'effectuer les test de charge, des test de performance, test utilisateurs.
- PROD: Un environnement de production

5. Mise en place un pipeline CI/CD

Outil de gestion de version : **GitLab**, assure la livraison continue du code.
Outils de gestion de projet (suivi des bugs, Gestionnaire des taches et documentation): **Jira**

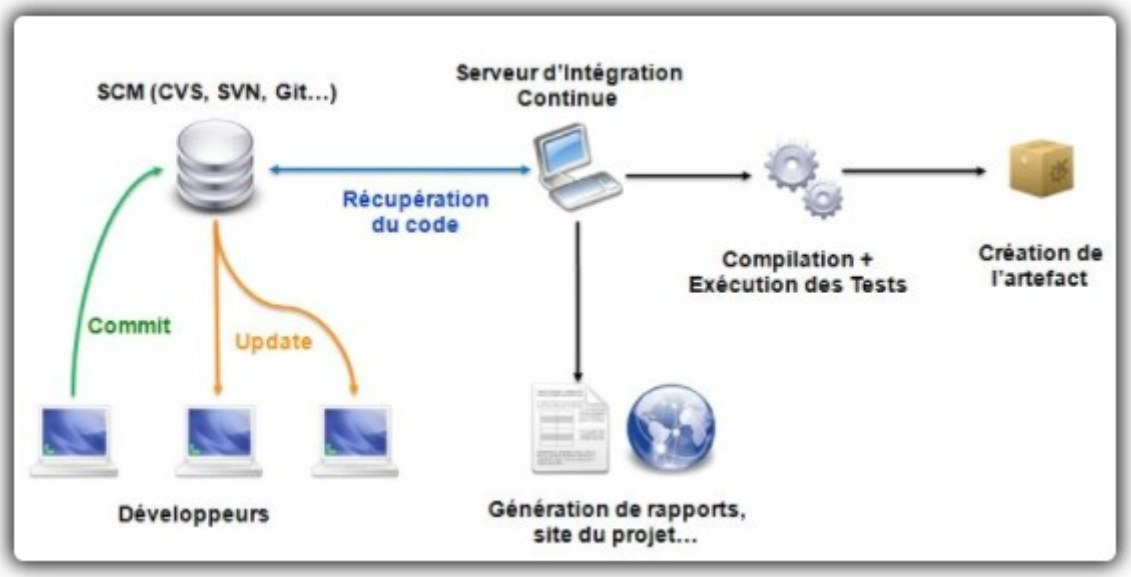
Intégration continue

L'intégration continue est un ensemble de pratiques qui consiste à ce que les développeurs intègrent régulièrement leurs modifications de code à un référentiel centralisé afin de s'assurer que ces modifications ne produisent pas de régression dans l'application développée.
Des outils permettent de réunir les étapes nécessaire à la mise en place de l'intégration continue.

Livraison continue

La livraison continue est une méthode de développement de logiciels dans le cadre de laquelle les modifications de code sont automatiquement préparées en vue de leur publication dans un environnement de production à n'importe quel moment.

Elle permet aux développeurs d'automatiser les tests au-delà des simples tests d'unité, afin de vérifier différents aspects d'une mise à jour d'application. Il peut s'agir de tests d'interface, de charge, d'intégration, de fiabilité de l'API, etc. De cette manière, les développeurs peuvent vérifier de façon plus complète les mises à jour et détecter les éventuels problèmes à corriger avant le déploiement.



6. Convention de Codage

Pendant la phase de codage, pour une meilleure qualité et compréhension du code, il est préconisé d'utiliser des convention de codage pour le nommage des variables manipulés.

L'écriture *CamlCase* ou *la notation hongroise* (chaque nom de variable est précédé par la lettre représentant son type.
Pour la base de donnée, on peut figer les règles suivantes afin de toujours mieux comprendre le code. Toutes les tables sont écrites au pluriel en lettre minuscule, et les champs au singulier. Enfin la nomination de l'id se résume à une concaténation entre le nom de la table et le mot «id».

7. Amélioration continue des pratiques de code

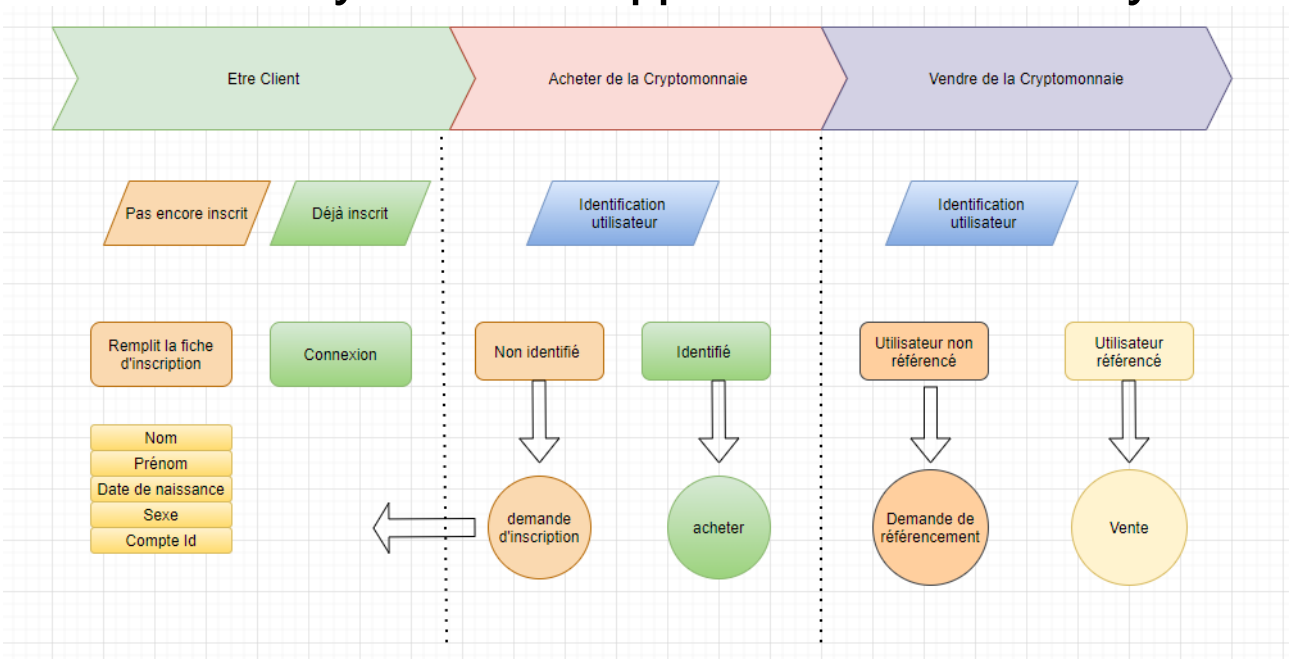
Des outils intégrés à votre IDE Eclipse peuvent vous permettre d'améliorer la qualité et le format de votre code.
SonarCloud est un outillage permettant d'effectuer l'audit de code et de l'afficher sous forme de rapport détaillés et interactifs. Il est entièrement gratuit pour les projets open source.
Des outils de build comme Maven et Gradle permettent d'automatiser le lancement des tests mais aussi de la vérification de qualité de code.

8. Les Tests, privilégier une approche TDD

Test-Driven Development ou Développements Pilotés par les Tests, est une méthode de développement de logiciel, qui consiste à concevoir un logiciel par petits pas, de façon itérative et incrémentale, en écrivant chaque test avant d'écrire le code source en remaniant le code continuellement.

Outil préconisé : Cucumber, qui est un framework BDD aide pour la traduction de Gherkins en langage Java.
JUnit pour la rédaction des tests unitaires.

9. Suivez le story-board de l'application «GitMeMoney»



10. Implémentation des microservices

Chaque microservice expose une API REST accessible via des requêtes HTTPS et uniquement par des clients authentifié. L'échange d'information se fait au format JSON. Format plus léger et performant que le XML.
Tout comme le pattern ***MVC*** utilisé tout au long du développement de l'application, l'utilisation de pattern tel que l'***inversion des contrôles*** est fortement préconisé lors du développement du microservices. Chaque microservice sera accessible au travers de l'API Gateway et pourra être versionné.
L'application requière le développement de deux microservices:
Gestion des utilisateurs et Gestion de la Cryptomonnaie.

11. Implémentation de la Gateway

Point d'entrée vers le Back-end et les différents services de l'application. Celle ci doit être un point d'attention pour éviter de devenir un point de défaillance centralisé de l'application.
Elle est implémentée en aval des microservices, et permet de rediriger la requête du client vers l' API qui convient.

12. Implémentation de la couche front-end

Il est important de penser enchaînement des écrans avant de commencer l'implémentation de la solution front-end.

S'assurer que l'interface utilisateur couvre toute les User stories présentent dans le backlog. Simuler les données afin de permettre aux développeurs front-end d'avancer en toute autonomie (JSON Server pour l'application Angular).

Il existe des frameworks de test pour les applications front, Jasmine et Karma vous permettront de tester à minima votre développement.

13. Suivi à l'aide d'outils de monitoring

La maintenance de l'application passe par la détection en amont de possible défaillance. Écoute continue des services permet de dénicher de potentielle régression ou anomalie. Pour cela la mise en place d'outils de surveillance dit de «Monitoring» de l'application est fortement préconisé. Le suivi log permet aussi de comprendre les défaillances ou anomalies.