M1AL

**Compte-rendu**

**CleanCode - CycleConnect**

**2023 - 2024**

Flacelière Matthieu

Sauvage Timothée

Couderc Quentin

**Plan :**

1. Analyse du besoin
2. Workflow
3. Diagramme de classe
4. Choix des capteur
5. Spécifications technique
6. Réalisation du projet et principes de programmation :
   1. TDD
   2. SOLID
   3. YAGNI, KISS et DRY

I - Analyse du besoin :

Le projet consiste à transformer un vélo d'appartement non connecté en un vélo d'appartement connecté.

Les étapes incluent le choix des équipements, la conception d'une architecture logicielle, le développement d'une application POC en utilisant TDD et en respectant les principes de conception logicielle SOLID, DRY, KISS et YAGNI.

Les besoins fonctionnels de l'application incluent la connexion, le démarrage et la fin d'un parcours avec lecture vidéo en fonction de la vitesse de pédalage, la visualisation de l'historique des parcours, et la déconnexion.

Des fonctionnalités complémentaires peuvent être intégrées parmi les options suivantes : connexion à une Apple Watch, suivi par un coach, ou participation à une course multi-participants.

II - Workflow de notre application :

Pour commencer, l'utilisateur crée un compte sur la plateforme, lui permettant d'accéder à toutes les fonctionnalités.

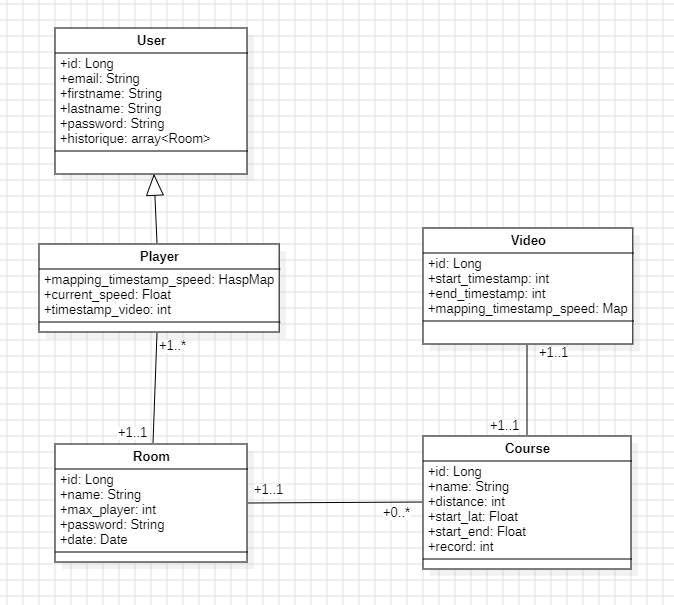
Ensuite, il connecte son PC au capteur via Bluetooth pour capturer en temps réel les données de sa performance. Une fois connecté, il a la possibilité de rejoindre un salon multijoueur où il pourra jouer avec d'autres utilisateurs.

Après avoir choisi une course et paramétrer ses objectifs, il reçoit et charge la course depuis le serveur. Le lancement de la course se fait alors, synchronisé avec le ratio entre la vitesse captée par le capteur et la vitesse de la vidéo originale.

À la fin de la course, il peut visualiser son historique de courses pour suivre ses progrès. De plus, il a la possibilité d'accéder à des rapports de performance exploitant l'intelligence artificielle pour comparer ses différentes courses et obtenir des conseils personnalisés pour améliorer ses performances.

Ce workflow offre une expérience interactive et enrichissante pour les amateurs de fitness et apporte une portabilité à moindre coût si celui-ci dispose d’un ordinateur et d’un écran.

III - Diagramme de classe :



IV - Choix du capteur :

Pour le choix du capteur, nous avons fait le choix d’utiliser un capteur bluetooth afin d’éviter d’ajouter un câble qui pourrait être encombrant et nous avons privilégié l’utilisation du protocole ANT+ qui est un protocole généralisé dans les objets connectés sportif et compatible avec de nombreux autres objets connectés.   
De plus ce capteur ANT+ utilise une technologie à faible consommation d’énergie ce qui peut être intéressant pour éviter de le recharger fréquemment.

Dans notre cas nous avons repéré un modèle de capteur qualitatif et pas trop cher que nous recommandons à nos futur client qui est le modèle Magene S3+

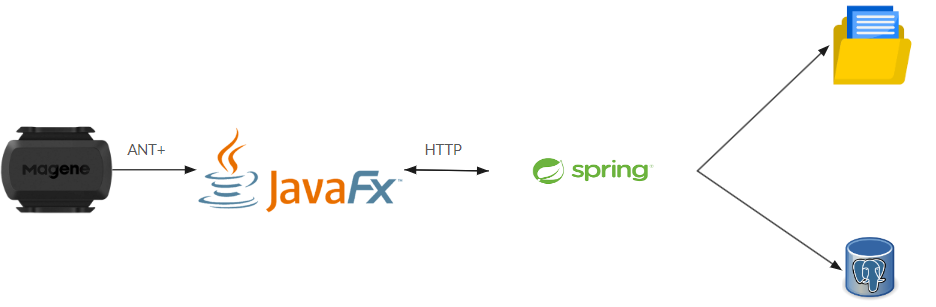


V - Spécifications technique :

Comme présenté ci-dessous, dans un premier temps le capteur va collecter les informations de vitesse et de cadence du vélo d’appartement.

Celui-ci va envoyer ces données avec le protocole ANT+ vers un client JavaFx installé au préalable par le client sur son ordinateur. Par la suite, le client enverra des requêtes vers le serveur Spring et recevra des réponses avec le protocole http.

Enfin, le serveur utilisera une base de données Postgres et un serveur de fichier afin de stocker les vidéos et différentes données de l’application.



VI - Réalisation du projet et principes de programmation:

A) TDD (Test-Driven Development) :

Afin de réaliser le projet, nous avons utiliserons la méthode TDD pour Test-Driven Development qui consiste dans un premier temps à développer le test unitaire d’une fonctionnalité, puis de réaliser le code nécessaire pour résoudre ce test pour enfin s’en servir de base de code pour réaliser notre fonction en ajoutant les implémentations nécessaire à cette fonction.  
Cette méthode de développement permet une bonne maintenabilité du projet dans le temps ainsi qu’une amélioration de la fiabilité du service qui sera couvert par des tests unitaires constamment. Elle permet aussi une réduction du nombre des tests pour une couverture similaire.

B) SOLID :

Pour mener à bien ce projet et assurer sa maintenabilité dans le temps, nous utiliserons plusieurs autres principes de programmation.

Nous utiliserons le principe SOLID :

SOLID : SOLID est un acronyme qui représente cinq principes de conception de classes et d'objets en programmation orientée objet. Chaque lettre représente un principe différent :   
 S : Single Responsibility Principle (SRP) - Principe de responsabilité unique : Consiste à ce qu’une classe n’ait qu’une seule responsabilité ou tâche à accomplir dans le système.

O : Open/Closed Principle (OCP) - Principe d'ouverture/fermeture : Consiste à ce que les entités logicielles (classes, modules, fonctions, etc.) soient ouvertes aux extensions mais fermées aux modifications.

L : Liskov Substitution Principle (LSP) - Principe de substitution de Liskov : Consiste à ce que les objets d'un type de base doivent pouvoir être remplacés par des objets de sous-types sans altérer la cohérence du programme.

I : Interface Segregation Principle (ISP) - Principe de ségrégation des interfaces : Consiste à ce que les clients ne soient pas impactés par les interfaces qu'ils n'utilisent pas.

D : Dependency Inversion Principle (DIP) - Principe d'inversion de dépendance : Consiste à éviter que les modules de haut niveau ne dépendent de modules de bas niveau. Les deux doivent dépendre d'abstractions.

C) YANGI, KISS et DRY :

DRY (Don't Repeat Yourself) : Ne vous répétez pas. Ce principe recommande d'éviter la duplication de code en factorisant le code répété dans des fonctions, des classes ou des modules réutilisables. Cela favorise la maintenabilité du code en réduisant la propagation des changements et en améliorant la cohérence.

KISS (Keep It Simple, Stupid) : Gardez-le simple, idiot. Ce principe suggère de préférer les solutions simples et directes plutôt que des solutions complexes et sophistiquées. Les solutions simples sont généralement plus faciles à comprendre, à maintenir et à déboguer.

YAGNI (You Aren't Gonna Need It) : Vous n'en aurez pas besoin. Ce principe conseille de ne pas ajouter de fonctionnalités ou de complexité au code tant qu'elles ne sont pas nécessaires. Éviter d'anticiper des fonctionnalités ou des besoins futurs qui pourraient ne jamais se matérialiser, car cela peut conduire à une sur-ingénierie et à un code inutilement complexe.

Ce qu’il faut garder à l’esprit c’est que ces différents principes de programmation permettent d'épurer le code et d’assurer sa perduration dans le temps sans difficulté.