Clean-Bike

Robin - Yannick - Yohann

1. Contexte

Le but du projet est de réaliser une solution qui permet à un home trainer, vélo d'appartement, de devenir un vélo connecté.

2. Les Choix Techniques

a. Le capteur

Dans un premier temps nous avons dû choisir le matériel nécessaire à cette transformation. Le premier élément essentiel de ce projet est d'être capable de récupérer les informations de notre utilisateur. C'est pour cela que nous avons besoin d'installer un capteur. Nous avons choisi d'utiliser un capteur de cadence. Grâce à ce type de capteur nous pouvons avoir les informations sur le nombre de tour de pédale par minutes de l'utilisateur.

Dans notre cas 2 capteurs était en concurrence, le capteur de cadence Sensor 2 de la marque Garmin et le capteur Fitness RPM de la marque Wahoo.

Caractéristique	Garmin Cadence 2	Wahoo Fitness RPM
Marque	Garmin	Wahoo Fitness
Prix	39.99	40.00
Connectivité	ANT+ , Bluetooth	Bluetooth, ANT+
Compatibilité	iOS, Android	iOS, Android
Poids	6 grammes	7 grammes
Étanchéité	IPX7	IPX7
Type de fixation	Aimant	Sans aimant
Applications compatibles	Garmin Connect, Zwift, strava	Wahoo Fitness, Strava

Suivant le tableau de comparaison ci-dessous, et aussi la notoriété de la marque Garmin nous avons choisi le Capteur cadence Sensor 2.

b. L’écran d’affichage

Le deuxième élément nécessaire à notre solution est un tableau ou un système muni d’Android, smartphone, écran connecté… Pour notre cas d’utilisation nous sommes partie sur une tablette Samsung galaxy tab pour le plus grand confort de notre utilisateur. On à choisi le support tablette car c’est le compromis parfait pour avoir le système android et avoir une grande surface d’affichage pour une meilleure immersion.

c. La Connectivité

Nous avions le choix entre deux protocoles de connexion, le Bluetooth et l ' ANT+. On à choisi de relier notre capteur et notre support d’affichage via **Bluetooth**. Les principaux critères qui nous ont poussés à prendre le Bluetooth plutôt que ANT+ sont la sécurité et la facilité d’utilisation. Effectivement le Bluetooth offre une connexion unique du capteur à notre appareil d’affichage et évite que d'autres solutions puissent récupérer les informations de notre utilisateur.

d. Le type de solution

Plusieurs types d’application était possible de réaliser. Celles que nous avons retenue et mis en concurrences étaient les suivantes :

- Les applications Mobile
- Les WebApp

Nos critères étaient simples, pouvoir utiliser notre solution sur le plus d’appareil possible, qu’elle soit le plus accessible possible et ne nécessite pas d’acheter beaucoup de matériel et pouvoir être utilisé hors ligne.
La seule application qui regroupe tous ces critères est **l’application mobile.** Grâce à ce choix, toutes les personnes ayant un téléphone pourrons accéder à notre application, une fois l’application téléchargée ils pourront accéder à celle-ci sans connexion.

Nous avons choisi de développer notre application en JAVA car c’est un langage répandu et que nous maîtrisons tous les trois.

3. Développement

a. Recherche de solutions

Afin de réaliser le développement de notre application POC (Proof Of Concept), nous avons commencé par chercher des solutions techniques permettant de répondre à nos besoins fonctionnels.
Premièrement, nous avons cherché une méthode pour récupérer les données envoyées par notre capteur. Les applications développées pour Android sont conçues pour récupérer des données de différents capteurs intégrés dans les téléphones et tablettes.
Nous avons donc trouvé dans la documentation Android une interface ***SensorEventListener*** permettant de récupérer les données d'un capteur et de déclencher les actions voulues lors de la réception de nouvelles données du capteur.
Dans notre cas, dès que le capteur nous envoie une information, nous souhaitons adapter la vitesse de la vidéo affichée sur l'écran de l'utilisateur.
Pour cela, il fallait d'abord afficher une vidéo. Une fois de plus, le développement d'applications Android prévoit une solution pour cela, il s'agit de la classe ***MediaPlayer***.
En plus de nous permettre de lire une vidéo, cette classe nous permet de gérer différents paramètres, et notamment sa vitesse de lecture.
Nous avons donc les éléments nécessaires pour répondre à nos besoins fonctionnels.

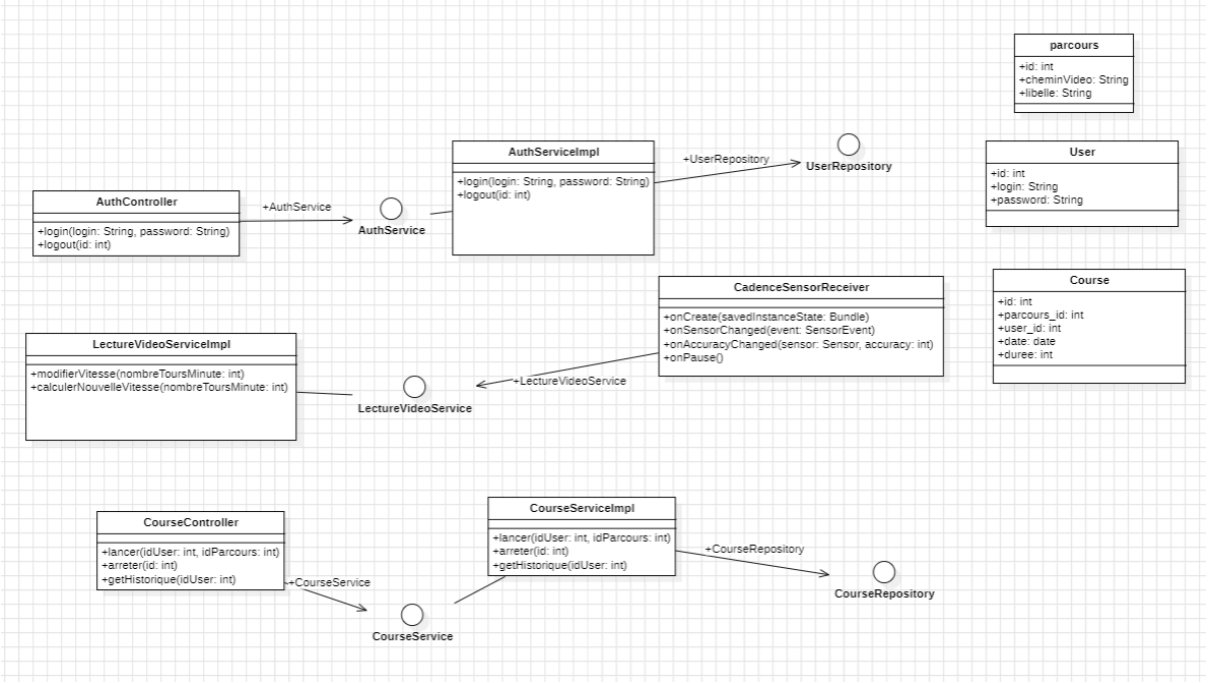
b. Conception du POC

L'objectif du Proof of Concept (POC) était d'évaluer la pertinence de notre architecture logicielle et de montrer les différents choix que nous avons effectués.
Pour cela, nous avons opté pour une approche où le développement est séparé en deux projets distincts : un projet Android pour le frontend et un projet Java Spring pour le backend. Cette décision de conception a été prise après une analyse des besoins de l'application, ainsi que des avantages et inconvénients associés à chacune des architectures.

Pour réaliser une version concrète de cette application, les 2 projets devraient être réunis afin de rendre les solutions trouvées précédemment utilisables et fonctionnelles.

c. Diagramme de classes et code

Sur notre projet POC java cleanbike nous avons mis en place les éléments présents dans le diagramme de classe :



Ces composants forment la partie backend de notre architecture et respectent des normes de développement établies.
Notre application utilise une architecture par couche (controller, service, repository) semblable au modèle utilisé par le framework Java Spring.
La composition du projet est divisée en plusieurs grands modules pour limiter la responsabilité de chaque classe :

- Auth : authentification à l'application, connexion déconnexion.
- Course : système de gestion des courses, lancement d'une course, historique par utilisateur…
- LectureVideo et CadenceSensorReceiver : gestion du capteur de cadence et de la vitesse de lecture de vidéo.

Ainsi, chaque classe de chaque module possède un rôle spécifique par rapport à l'ensemble du projet.

Pour les dépendances injectées dans nos classes, nous passerons par des interfaces pour garantir d'utiliser les bonne méthodes et pour gagner en flexibilité et en maintenabilité.

Pour ces interfaces, nous veillerons à ne pas leur déclarer trop de méthode, de manière à ce que les classes qui les implémentent ne soient pas forcées à implémenter des méthodes qu'elles n'utiliseront pas.

De cette manière, on garde de la cohérence dans notre manière de développer.

Par exemple, dans la classe CadenceSensorReceiver qui a pour responsabilité de récupérer les information du capteur, on implémente l'interface LectureVideoService qui possède une seule méthode : modifierVitesse().

Cette méthode est donc appelée uniquement dans la classe CadenceSensorReceiver et est override dans LectureVideoServiceImpl qui implémente l'interface.

d. Fonctionnement et Tests

Pour répondre à l'objectif général du projet de pouvoir accélérer la vitesse de lecture d'une vidéo en fonction de la fréquence de tours d'un vélo, nous avons mis en œuvre plusieurs classes dans notre POC qui utiliseront les solutions techniques Android choisies.

Premièrement, pour gérer notre capteur de cadence, nous avons la classe CadenceSensorReceiver évoquée plus tôt.
Cette classe implémente l'interface SensorEventListener ainsi que quelques méthodes servant à la configuration et à la gestion des événements du capteur.
Parmi ces méthodes, on retrouve la fonction onSensorChanged, qui est appelée lors d'un changement de valeur du capteur.

Cette fonction récupère donc la valeur en tours/minutes du capteur et appellera donc la méthode modifierVitesse(), de l'interface LectureVideoService en lui passant la nouvelle valeur.
Sur l'implémentation de cette interface (LectureVideoServiceImpl) nous calculerons la nouvelle vitesse de lecture de la vidéo.

Pour passer cette nouvelle vitesse de lecture à l'activity Android correspondante (coté front) tout en respectant les bonnes pratiques de développement, on utilise la librairie JmsTemplate du framework Spring, permettant d'envoyer des messages.
On injecte donc une propriété de type JmsTemplate dans notre service LectureVideoServiceImpl et nous poturrons ainsi envoyer la nouvelle vitesse sur l'activity d'affichage de la vidéo.

Enfin, pour mettre à jour cette vitesse de lecture, nous utiliserons les classes MediaPlayer et PlaybackParams d'Android permettant de gérer plus globalement la lecture de vidéo.

Test de création d'un parcours :

```java
@SpringBootTest
class ParcoursRepositoryTest {
    3 usages
    @Autowired
    private ParcoursRepository parcoursRepository;

    unknown
    @BeforeEach
    void deleteAllParcours() { parcoursRepository.deleteAll(); }

    unknown
    @Test
    @Order(1)
    void shouldAjouterParcours(){
        parcoursRepository.save(new Parcours( id: 1L,  cheminVideo: "/test",  libelle: "Course 1"));
        assertEquals( expected: 1L, parcoursRepository.count());
    }
}
```

Détails :

- On utilise le parcoursRepository (sans le mocker pour réellement appeler le repository).
- Avant chaque test, on supprime l'ensemble des données de parcours pour partir d'une base propre.
- On exécute le test qui ajoute un parcours avec des données arbitraire, puis on vérifie que ce parcours a correctement été ajouté.

Test de récupération de l'ensemble des courses d'un utilisateur (notamment utilisé pour l'historisation de course)

```java
@SpringBootTest
class CourseRepositoryTest {
    5 usages
    @Autowired
    private CourseRepository courseRepository;

    unknown
    @BeforeEach
    void deleteAllCourse() { courseRepository.deleteAll(); }

    unknown
    @Test
    @Order(1)
    void shouldReturnCoursesByUserId(){
        Long userId = 123L;
        courseRepository.save(new Course( id: 1L,  parcoursId: 1L, userId, new Date()));
        courseRepository.save(new Course( id: 2L,  parcoursId: 3L, userId, new Date()));
        courseRepository.save(new Course( id: 3L,  parcoursId: 2L, userId, new Date()));
        List<Course> courses = courseRepository.findAllByUserId(userId);
        assertEquals( expected: 3, courses.size());
    }
}
```

Détails :

- On utilise le courseRepository (aussi sans le mocker pour réellement appeler le service et ses fonctions).
- Avant chaque test, on supprime l'ensemble des données de course pour avoir une base propre.
- Le test en lui-même ajoute quelques courses avec des parcours différents mais à un seul utilisateur (ici userId).
- On récupère ensuite la liste des courses par utilisateur avec la fonction findAllByUserId du courseRepository (gérée automatiquement par le framework)
- Enfin, on vérifie que le résultat obtenu correspond bien aux courses de l'utilisateur, ajoutées précédemment.