Département : MMI

Filière : M1 Intelligence Artificielle

Projet IFO0704

sous le thème

Course Vélo

Réalisé par

- **Ahmed Lafdhal**
- **El Mehdi Outbarkate**
- **Sarah Moali**
- **M'hand Sahari**

Encadré par

- **Mr Marwane Ayaida**

Table des matières

Travail demandé

Il est demandé de crée une application mobile pour la gestion des courses de vélo d'une manière distribuée qui permette chaque capteur envoie sa position au serveur et calculer la distance parcouru et l'accélération moyenne ce serveur doit récupérer les données envoyées par l'application, les stocker dans une base de données puis les afficher pour l'utilisateur.

I. Description technique et justification des choix

Le projet se base sur une structure client/serveur avec un client android. Le serveur est développé en python, et nous avons utilisé android studio pour le client. La communication entre les 2 est assurée à l'aide de websockets.

1. Notre serveur

Le serveur AEMM est développé en python. Nous avons opté pour cette option car nous avions tous un peu d'expérience avec ce langage. Pour notre utilisation, il présente beaucoup d'avantages : c'est un langage très populaire, d'où découle une abondance de documentation accessible facilement sur internet. Il est aussi réputé pour ses nombreuses bibliothèques facilitant le développement déjà relativement rapide en comparaison avec d'autres langages dû à son approche haut niveau limitant les sources de bugs et offrant de nombreuses fonctionnalités très pratiques dans ses fonctions natives.

2. Communications client-serveur

Nous avons utilisé la bibliothèque socketio pour faire communiquer le client et le serveur. Cette bibliothèque introduit le paradigme de programmation événementielle en offrant la possibilité de facilement gérer les évènements en créant des "handlers" ainsi que d'en émettre. Les événements peuvent avoir des données, Ces événements sont envoyés par le serveur vers les clients pour les informer de la participation d'un joueur à la course ou de sa déconnexion et des clients vers le serveur pour envoyer les données tels que les noms, leur courses ou il participera et à la fin de la course la distance qu'il a parcourue .

Lorsqu'un joueur s'authentifie sur l'application, il est envoyé dans la room principale, depuis laquelle il a accès à toutes les courses qu'il peut rejoindre ou de créer sa propre course ensuite

il est renvoyé dans une room spéciale pour la course qu'il a rejointe ou crée. Nous nous servons des room pour différencier les joueurs d'une course à l'autre par exemple les joueurs de la course 1 font leur propre élection sans que les joueurs d'une course2 participent à cette élection vue que ce n'est pas la même course.

3. Généralités et fonctionnement :

Le système de parties permet de créer et de gérer plusieurs "rooms" qui s'exécutent indépendamment les unes des autres. Dans le menu principal, un joueur connecté peut choisir de créer une nouvelle partie ou d'en rejoindre une qui n'a pas son nombre de participants au complet. Pour rejoindre une course, il peut en choisir une sur la liste. Le système de parties est représenté sous la forme d'un dictionnaire associant le nom
d'une partie avec une liste des joueurs . Dans le code il s'agit de la variable globale **"room_users".**
La structure est de la forme : nom_course : [joueur1,joueur2..etc]

4. Contexte et messages échangés entre back-end et front-end

Dans le code, l'évènement join permet de rejoindre une partie mais aussi d'en créer une si elle n'existe pas et l'événement "creer_course" permet d'ajouter au dictionnaire un nouveau item
Syntaxe du message pour créer/rejoindre une partie
join_room : {room=nom_room,users=listedesjoueurs}

5. Réception de la liste des courses disponibles

A chaque fois qu'une course est créée on envoie avec l'émit "update_table" au serveur la liste des courses disponibles

6. Page d'attente et commencement de la course

Quand un joueur cree ou rejoint une partie il sera redirigé vers une salle d'attente l'algorithme d'élection se fera à nouveau.
Donc normalement chaque utilisateur aura le même leader vu que l'élection se fait à chaque arrivée d'un joueur.
Le leader est le seul qui aura le droit de commencer la course , s'il la commence, le serveur reçoit par l'événement "commencer" avec nom_course comme donnée et le serveur de son tour

envoie à tous les joueurs de cette course un emit “commencer” pour commencer la course ensuite tous les joueurs sont dirigés vers la page parties qui contient les détailles de chaque joueurs.
Si un joueur se déconnecte, le serveur le détecte et envoie aux joueurs de la course dont le joueur qui est déconnecté appartenait un emit “join_room” qui sert à faire la mise à jour de la liste des joueurs de la course et l'élection se fait aussi à nouveau .

7. Fin de la course

Seul le leader a le droit de finir la course , en cliquant sur le bouton finir, il envoie l’événement “finir” au serveur, ce dernier renvoie l'événement “finir” aux joueurs, qui à leurs tours, ils renvoient les "données" avec comme paramètres “room” pour nom de la course, “nom” pour le nom du joueur et “distance” pour la distance qu’il a parcourue.

II. Base de données:

L’utilisation de SQL brut pour effectuer des opérations CRUD sur la base de données peut être fastidieuse.
Au lieu de cela, SQLAlchemy, le Python Toolkit est un puissant OR Mapper, qui fournit aux développeurs d’applications toutes les fonctionnalités et la flexibilité de SQL.
Flask-SQLAlchemy est une extension Flask qui ajoute la prise en charge de SQLAlchemy à l’application Flask.Toutes les communications avec la BDD se font via un ORM qui abstrait la couche SQL en code python.

1. Qu’est-ce que l’ORM ?

ORM est l’abréviation de Object Relation Mapping (parfois Object Relationship Mapping). La plupart des plates-formes de langage de programmation sont orientées objet.
Les données du serveur SGBDR sont stockées dans des tables. Le mappage objet-relationnel est une technique qui mappe les paramètres d’objet à la structure d’une table SGBDR de couche. L’API ORM permet d’effectuer des opérations CRUD sans écrire d’instructions SQL brutes.

2. Tables :

Nous avons créé trois tables :

table_**bikers** : Le biker est enregistré dans la base à chaque login.

table_**courses** : Une course est ajoutée à la table après chaque nouvelle création

table_**distance** : A la fin de chaque course, une distance parcourue par un biker est enregistrée et affiché sur le serveur.

III. Algorithme d'élection de Leader :

L'élection de Leader se fera d'une manière distribuée, chaque Biker reçoit une table des participants pour faire sa propre élection par la comparaison des Login de chaque biker pour voir le poids du plus grand.

Par exemple si on a quatre Bikers :

Login 1 = 'Ahmed', Login 2 = 'El Mehdi' Login 3 = 'Mhand' Login 4 ='Mohammad' Notre algorithme va choisir ***Mohammad***.

```
public  void election() {
    List<String> joueurs = new ArrayList<>();
    joueurs = MainActivity.joueurs;
    String max;
    max= Collections.max(joueurs, comp: null);
    TextView myAwesomeTextView = (TextView)findViewById(R.id.leadertext);
    myAwesomeTextView.setText(max);
    leader=max;

    if(leader.equals(MainActivity.nom)){
        btnCommencer.setVisibility(View.VISIBLE);
    }
    else {
        btnCommencer.setVisibility(View.GONE);
    }
}
```

IV. Conception

1. Acteur et taches :

Un acteur : C’est l'ensemble des rôles joué par un utilisateur, nous présentons dans ce qui se suit les acteurs principaux intervenants dans notre application :

- **Un biker :** Désigne tout utilisateur qui se connecte à l’application.

Une tâche : Est l’ensemble des différentes fonctions affectées à un acteur auxquelles il peut accéder

Chaque cas d’utilisation représente une tâche qu’on associe à un acteur du système, le déroulement de ces tâches est représenté sous forme de scénario que nous présentons dans le tableau suivant :

Acteurs	**Taches**	**Scénario**
Biker	**T1:** Login	**S1:** Saisir un nom de login S2: Cliquer sur le bouton <Login>
	T2: Créer une course	**S1:** Cliquer sur le bouton <Créer une nouvelle course>
	T3: Rejoindre une course	**S1:** Choisir le nom de la course à rejoindre dans la liste. **S2:** Cliquer sur le <bouton rejoindre>
Leader	**T1:** Commencer une course	**S1:** Cliquer sur le bouton <Commencer la course>
	T2: quitter la partie	**S1:** Cliquer sur le bouton <quitter le jeu>

Table 1-1 - Acteurs et leur tâches et leur taches

2. Diagramme de contexte

Le diagramme de contexte est utile en début de projet pour clarifier le domaine d'étude car il permet de :

- Le situer dans son environnement.
- Identifier les acteurs avec lesquels il communique.
- Délimiter ce qu'il y à faire et ne pas faire.

La figure ci-dessous représente le diagramme de contexte de notre future application.

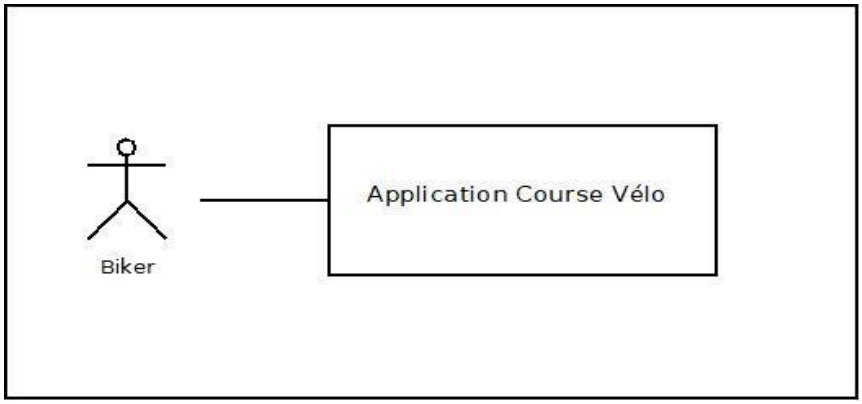


Figure 1-1 Diagramme de contexte

3. Diagramme de cas d’utilisation:

Les diagrammes des cas d’utilisations donnent une vision globale du comportement fonctionnel de l’application en mettant en évidence les besoins exprimés par l’utilisateur ainsi que son interaction avec l’application.

Pour une meilleure lisibilité et compréhension de tous les cas d'utilisation recensés, nous élaborons un diagramme de cas d'utilisation global pour mettre en évidence les cas généraux.

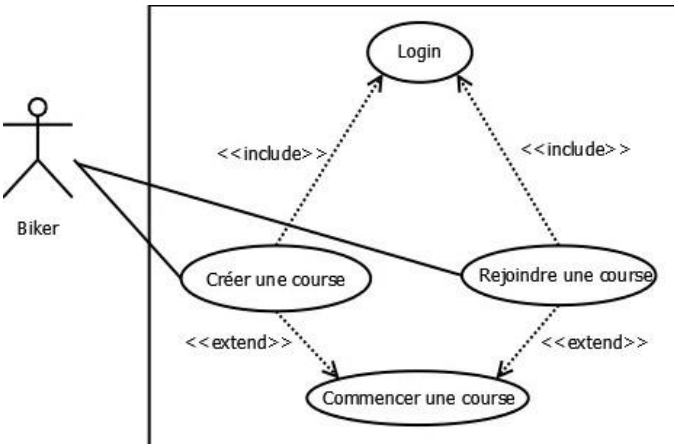


Figure 1-1 Diagramme de cas d'utilisation global

V. Travail réalisé:

1. Interfaces de l'application

1.1 Page d'acceuil:

Au lancement de l'application, on obtient la page d'acceuil suivante où un biker peut se connecter.

1.2 Interface principale:

Le biker peut soit créer une nouvelle course ou rejoindre une course déja crée

1.3 Interface salle d'attente:

Figure cas leader

Figure cas biker simple

Ici les bikers sont entrain d'attendre le leader a ce quil commence la course. Comme on a dit auparavant, seul le leader aura cette posibilité de commencer uen course.

1.4 Interface de partie:

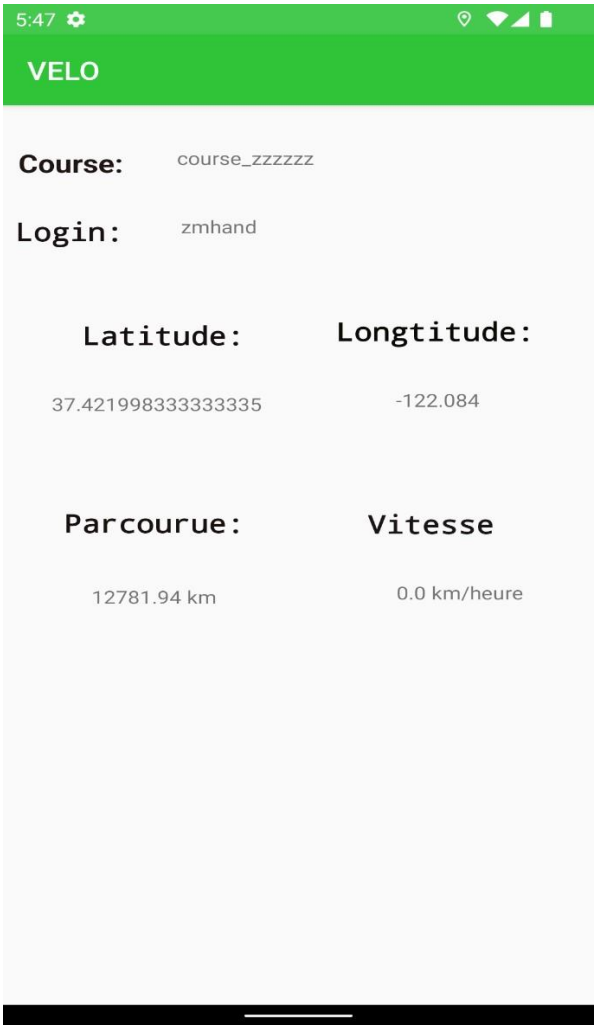


Figure cas leader

Figure cas biker simple

Ici on affiche les détails d'une course du biker, comme déjà cité, seul le leader peut arrêter la partie.

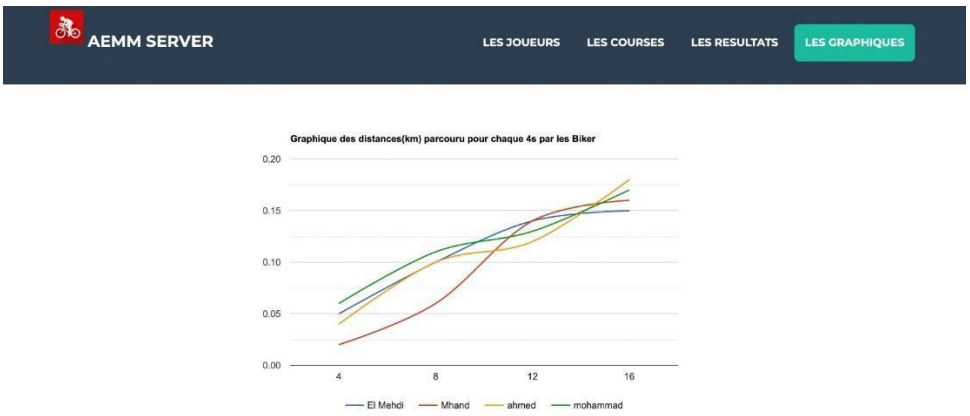
2. Server AEMM

2.1 Bikers

2.2 Courses

2.3 Resultats

2.4 Graphiques



VI. Difficultés rencontrées

- Dans le calcul des distances utilisant latitude et longitude qu'on se retrouve dans un espace fermé
- La distance ça change quelquefois avant de bouger puisque le GPS n'est exact.
- On n'avait pas assez de temps pour implémenter les graphiques dans le site web serveur.