Université de Sherbrooke Québec Canada

**********

Faculté des Sciences

**********

Département Informatique

**********

# IFT 604 - Applications Internets et Mobilité

## TP2 : HockeyNite Live, le Serveur et le Client Web

Rédigé par

Moustapha M. P. F. SOUMANOU Matricule : *18 176 714*

Boubacar. DEMBELE Matricule : *19 155 432*

Enseignant : M. Sylvain Giroux

***Automne 2019***

# Introduction

Afin d'approfondir les connaissances enseignées pour le cours applications Internet et mobilité, nous avons pour notre tp1 (pour rappel), réaliser un système hockeyNiteLive, qui est un système permettant de suivre "en direct", sur un téléphone cellulaire Android, le déroulement d'un match de hockey et de parier en-ligne. Toujours dans cet objectif et pour approfondir les enseignements, les connaissances apprises au cours et améliorer ce qui à été fait, nous présentons dans ce rapport le client web et le serveur mise à jour qui prend en compte la communication REST.

Pour la réalisation de ce tp nous avons utiliser un serveur kotlin (framework KTOR) et un client web (Framework DJANGO du langage Python). Dans les lignes à suivre nous expliquerons le fonctionnement de ses parties et les différentes communications entre elles.

# I. Outils, Installation, Configuration et lancement

Nous avons pour notre tp, un serveur développé en Ktor (Kotlin) avec l'aide intellij. Notre serveur pour la sauvegarde des données et pour répondre aux clients, utilise une base de donnée mysql. Nous avons aussi le client web développé avec le framework Django de python qui communique directement avec notre serveur.

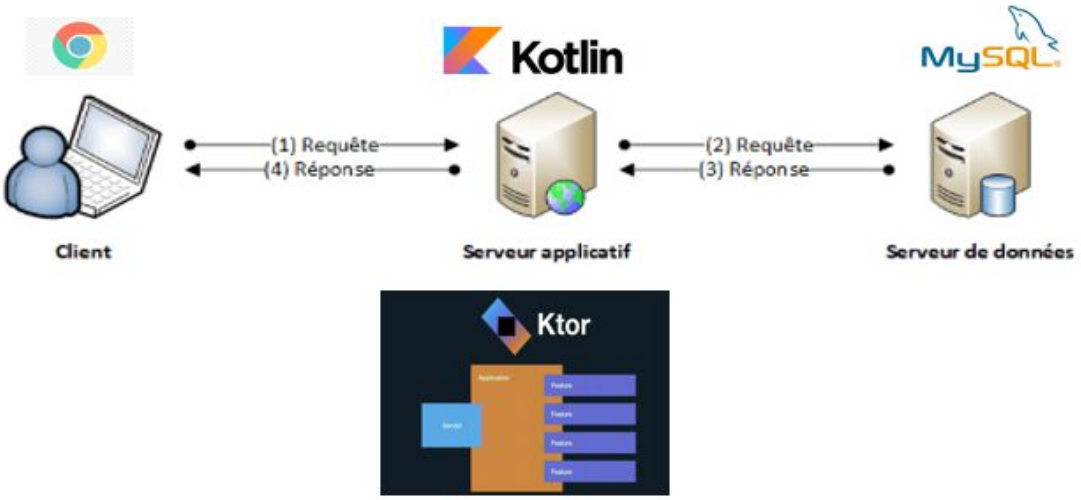
Pour lancer l'application, on démarre la base de donnée, on lance le serveur sur intellij et il se met à l'écoute sur le port désigné et ensuite, on lancer le client par son lien sur une page web. On pourra alors utiliser l'application qui sera alors lancé sur la page web et communiquer avec le serveur.

NB : le fichier README.TXT contient toutes les étapes d'installation et d'utilisation.

# II. Description et justification des choix

## A. Architecture Client Serveur

Le schéma ci-dessus décrit l'architecture de client serveur utilisé :



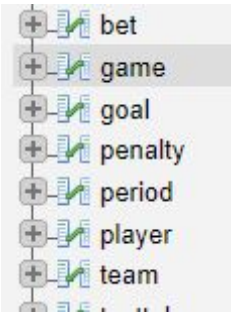
## B. Base de donnée

Toutes les informations sont enregistrées dans une base de donnée Mysql, les références et le relations sont gérées côté kotlin avec Exposed.

Ex Table de paris :

```
object Bet : Table("bet") {

  val id = integer("id").primaryKey().autoIncrement()

  val idGame = (integer("idGame") references Game.id).nullable()

  val choice = integer("choice")

  val bet = float("bet")

}
```

Coté table sur phpMyAdmin, on a juste déclarer les tables comme suivant.




### 1. Sérialisation

Pour l'envoie des données on les a sérialisées en JSON, dans la classe serialize.

Elle dispose de deux méthodes Serialize et unSerialize qui permettent d'adapter les données transférer.

On a fait le choix d'utiliser JSON, par ce que ça permet d'encapsuler l'objet et le désencapsuler rapidement et on dispose déjà de fonctions pour faire ça en Kotlin `GsonBuilder()`

Ex de JSON envoyé:

```
{
 "destination": "127.0.0.1",
 "destinationPort": 6779,
 "value": [
  {
   "id": 1,
   "team1Id": 1,
   "team2Id": 3,
```

```
    "date": "2019-10-14T13:23:42.000-04:00",
    "ended": 1
  },
  ]
}
```

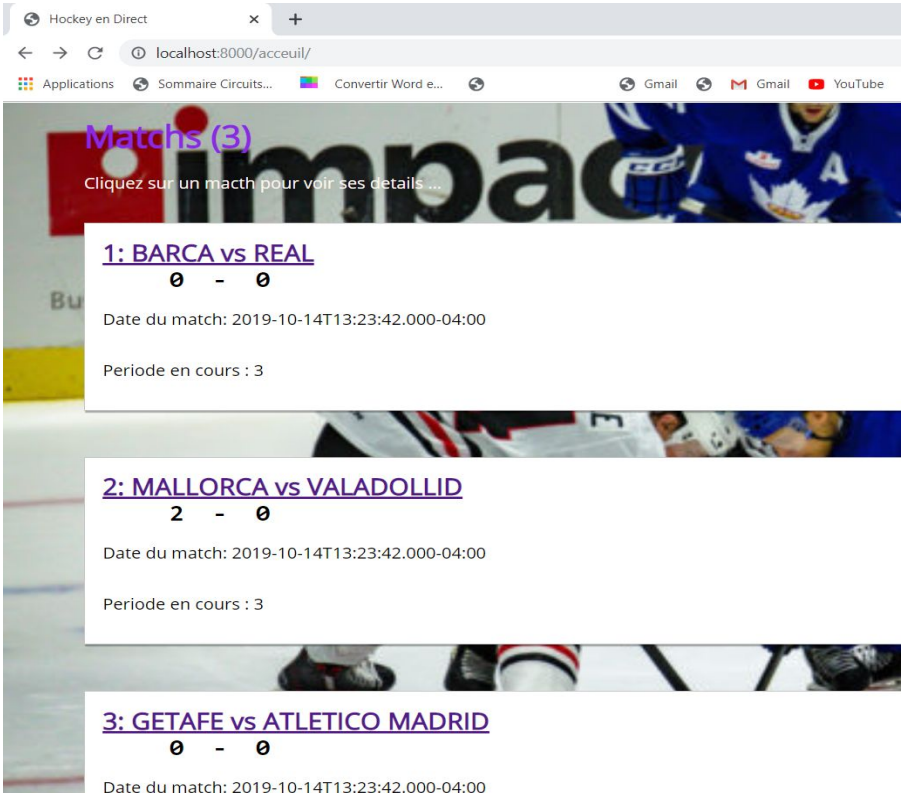

2. Concurrence

   La gestion de concurrence dans le serveur se fait pour tous les accès en R/W à la base de données. Pour gérer ça on a utilisé les attributs @synchronized devant les courtines qui accèdent à la BD.

## C. Côté Client Web

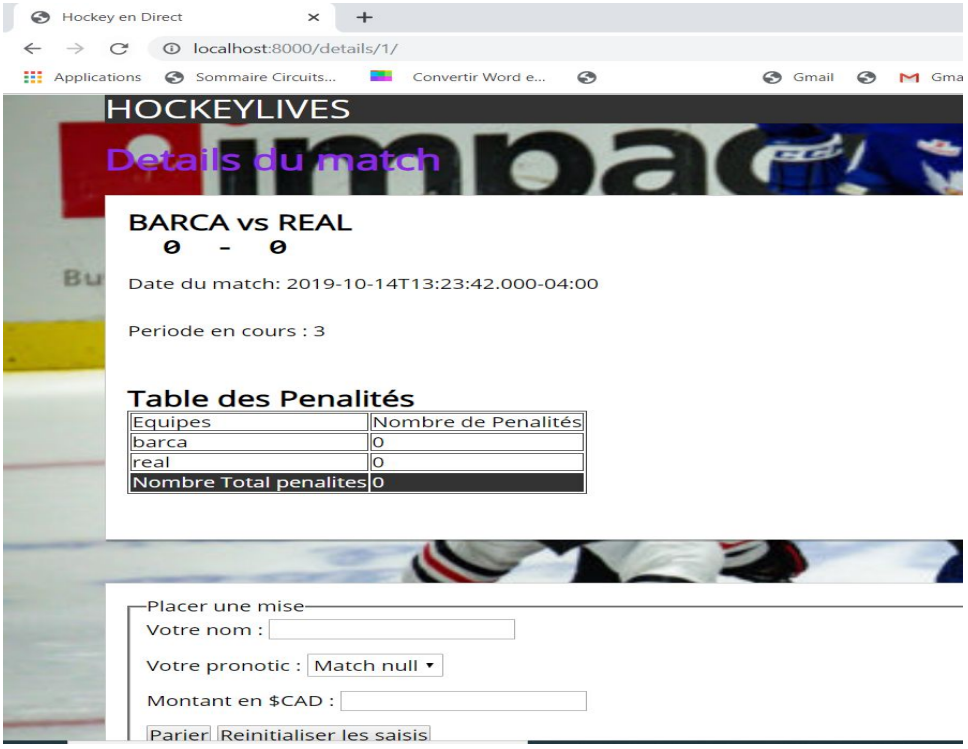
- Game_List:

  Sur la page d'accueil, elle permet d'afficher la liste des matchs.



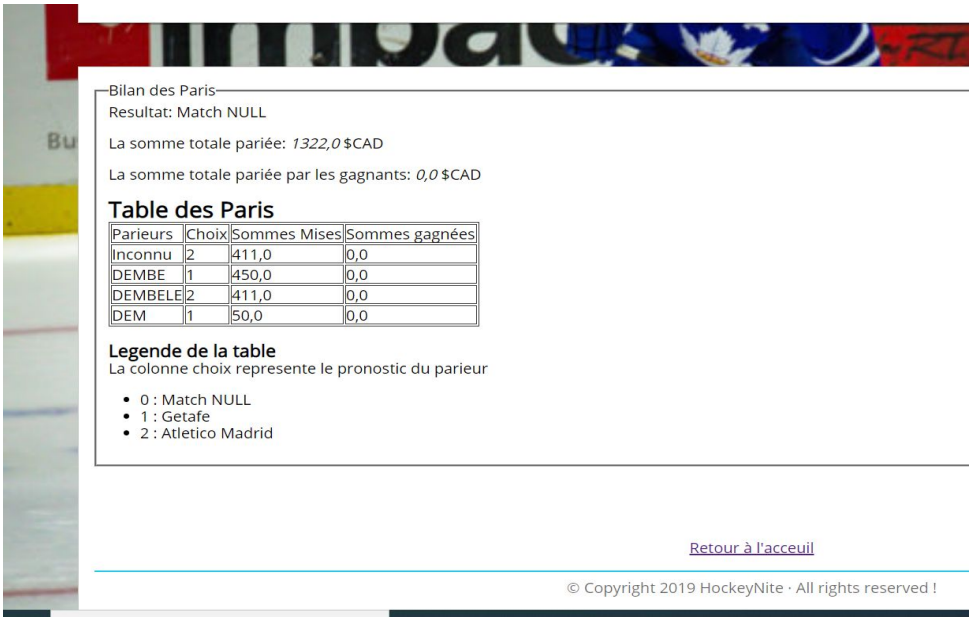

- Game_Detail :

Il nous affiche tous les détails sur un match donné sélectionné par l'utilisateur avec les pénalités




- Game_Pari :

Cette page présente le pari, le montant total parié, la liste des paris, et le gain par parieur.



## **D.** Services

- Communication (Service non connecté) : le serveur ktor présente une api rest en localhost sur le port 8080 permettant d'échanger avec le client web qui récupère les données nécessaire à son usage suivant le code ci-dessous.

```
routing { this: Routing
    //Test fonctionnement server
    get( path: "/") { this: PipelineContext<Unit, ApplicationCall>
        call.respond(Response(status = "OK"))
    }

    post( path: "/") { this: PipelineContext<Unit, ApplicationCall>
        val request = call.receive<Request>()
        call.respond(request)
    }

    //Envoie de la liste des matchs
    get( path: "/ListMatchs") { this: PipelineContext<Unit, ApplicationCall>
        call.respond(ListGames.getAllGames())
    }

    //Envoie des details d'un match
    post( path: "/Match") { this: PipelineContext<Unit, ApplicationCall>
        val gameID = call.receive<GameID>()
        println("Id:"+gameID.id)
        call.respond(Games.getGame(gameID.id.toInt()) as DetailGame)
    }

    //Reception de données sur les paris
    put( path: "/Pari"){ this: PipelineContext<Unit, ApplicationCall>
        val dataPari= call.receive<DataPari>()
        val res = Bets.placeBet(dataPari.id, dataPari.choice, dataPari.amount, dataPari.user)
        if (res){
            call.respond(Response(status = "OK"))
        }
```

```
    //Reception de données sur les paris
    put( path: "/Pari"){ this: PipelineContext<Unit, ApplicationCall>
        val dataPari= call.receive<DataPari>()
        val res = Bets.placeBet(dataPari.id, dataPari.choice, dataPari.amount, dataPari.user)
        if (res){
            call.respond(Response(status = "OK"))
        }
        else{
            call.respond(Response(status = "Imposible"))
        }
    }

    //Envoie du bilan d'un jeu terminé
    post( path: "/Bilan") { this: PipelineContext<Unit, ApplicationCall>
        val gameID = call.receive<GameID>()
        println("Id:"+gameID.id)
        call.respond(Games.getBilanGame(gameID.id.toInt()))
    }

}
}.start(wait = true)
```

## Conclusion

L'implémentation de ce TP nous a permis de mieux voir et appréhender et d'approfondir les notions appris au cours lors de l'apprentissage de ce IFT604.

De façon pratique nous avons pu créer un serveur avec kotlin et des clients léger, Android et Web qui communique avec ce serveur pour consulter et parié directement en ligne entre le score des équipes de hockey. Ce qui nous à été très bénéfique pour l'assimilation de ce cours.