POLYTECH GRENOBLE

RICM5

# Dossier de Conception Système

## MesCourses

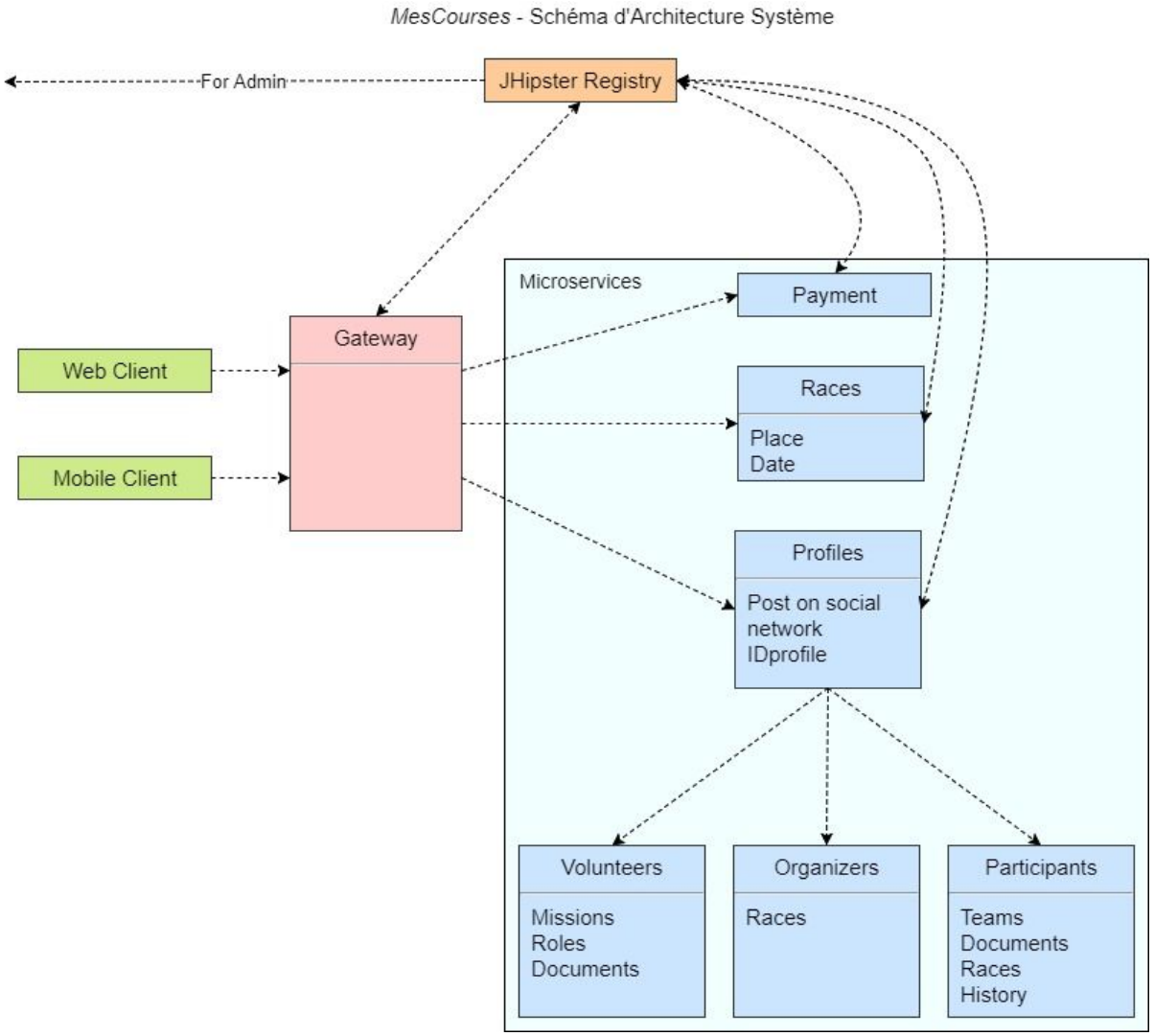
AUBERTIN Alicia - BROCHIER Aymeric - NASSIK Ahmed
ODIEVRE Boris - TURRIN Vincent

# Table des matières

# Introduction

Ce document présente la conception du système MesCourses qui est un projet généré à l'aide de l'outil JHipster et qui a donc une architecture qui en découle. Ce projet est décomposé initialement en deux front end (web et mobile), un gateway et un microservice. Ce microservice sera éventuellement divisé en plusieurs microservices au cours du développement si l'équipe el juge nécessaire. Sa composition est détaillée dans le schéma ci-dessous et dans le reste de ce document. Il sera basé sur le micro-framework SpringBoot.

# Schéma d'architecture système



# Front-end

## Web

Cette partie du front end sera générée avec JHipster et permettra l'inscription et la gestion de tous les membres inscrits ainsi que la gestion de toutes les courses.

## Mobile

Le front-end mobile sera développé en utilisant Cordova. Il permettra aux amis et aux bénévoles de consulter en temps réel les positions des coureurs et/ou de scanner les dossard de ces derniers. Il présentera donc des fonctionnalités plus limitées que la partie Web.

# Gateway

L'entrée de tous nos microservices : le gateway, sera générée à l'aide du générateur JHipster prévu à cet effet. Ce portail d'entrée gérera la redirection des requêtes HTTP vers les microservices correspondants, ainsi que le load-balancing, pour assurer une équité dans l'utilisation des serveurs.

# Registry

Le registre utilisé est le registre de base de JHipster. Celui-ci est relié à tous les microservices et permet un monitoring assez précis de ceux-ci. Celui-ci fournit des informations importantes au gateway afin que celui-ci puisse assurer le load balancing.

# Microservice

Le microservice de notre application est organisé selon la structure décrite plus haut. Il utilisera une base de donnée en MySQL. Plus tard ce microservice sera séparé en plusieurs sous microservices.

# Races

## Data :

Liste de courses (List<Race>)

## Attributs d'une course

**organisateur** (int) : La personne ayant créé la course
**admins** (List<int>) : Les personnes assignées en tant qu'administrateurs par le gérant de la course, ceux-ci possèdent les mêmes droits que l'organisateur
**city** (String) : La ville dans laquelle se déroule la course
**date** (date) : Date à laquelle se déroule l'évènement
**route** (List<Coordinate>) : Liste de coordonnées. Utilisation d'une API par l'organisateur pour tracer la course
**documents** (List<int>) : Les documents (par identifiant) nécessaires à l'inscription
**price** (int) : Prix d'inscription à la course
**volunteers** (List<int(id), Disponibilite, int(role), position>) : Ceux-ci seront associé à un point de bénévolat (position) automatiquement ou bien par l'organisateur manuellement.
**competitors** (List<int, role>) : Coureurs de la course, avec leur rôle associé
**pendingVolunteer**(List<int>) : Bénévoles en cours d'approbation

## API

### Gestion courses

**@who** : Utilisable pour tous les profils
**@brief** : Permet la création d'une course et son ajout dans la base de données
Le créateur possède alors le statut d'organisateur de la course
**@returns** : Renvoie true si la course à été correctement créée, false sinon
boolean createRace(String city, int day, int month, int year, int entryPrice)

**@who** : L'organisateur de la course
**@parameters** : idRace : l'identifiant de la course à modifier
**@returns** : Renvoie true si la course à été correctement modifiée, false sinon
**@comments** : Le prix ne peut pas être modifié
boolean modifyRace(int idRace, String city, int day, int month, int year)

**@Who** : Utilisable par tous les profils
**@brief** : Vérifie l'existence d'une certaine course
**@returns** : Renvoie true si la course existe, false sinon
**@comments** : Les paramètres sont omis pour la recherche s'il ne sont pas renseignés
boolean existingRace(String city, int day, int month, int year, int lowBoundPrice, highBoundPrice)

## Gestion utilisateurs

**@Who** : L'organisateur de la course
**@brief** : Permet d'octroyer le grade d'administrateur à un utilisateur donné
**@returns** : Renvoie true si l'utilisateur a été correctement ajouté, false sinon
**@comments** : L'idUser est récupéré grâce à une recherche dans "profils"
boolean addAdmin(int idRace, int idUser)

**@Who** : L'organisateur et les administrateurs de la course
**@brief** : Permet d'accepter un bénévole pour une course donnée
**@returns** : Renvoie true si l'utilisateur a été correctement ajouté, false si la personne n'est pas ajoutée refus ou erreur
**@comments** : L'organisateur peut accepter un bénévole qui se serait inscrit
boolean acceptVolunteer(int idRace, int idVolunteer)

## Fonctions pour utilisateurs

**@Who** : Toute personne n'étant pas bénévole/organisateur de la course peut concourir à celle-ci
**@brief** : Permet à un utilisateur de s'inscrire à la course, pour le rôle **role**
**@returns** : Renvoie true si l'utilisateur a été correctement inscrit, false sinon
**@comments** : Lien avec le microservice paiement pour le paiement de l'inscription
boolean joinRaceRunner(int idRace, int idUser, List documents, int role)

**@Who** : Tous profils
**@brief** : Permet à l'utilisateur d'afficher l'itinéraire de la route via une API
**@comments** : Cela enclenche un popup d'une carte avec l'itinéraire de la course
boolean raceRoute(int idRace)

**@Who** : Sera utilisée par l'organisateur et les administrateurs pour accepter un bénévole à la course
**@brief** : Permet à un administrateur d'accepter un bénévole à la course
**@returns** : Renvoie true si l'utilisateur a été correctement inscrit, false sinon
**@comments** : Lien avec le microservice paiement pour le paiement de l'inscription
boolean addBenevole(int idRace, int idUser)

**@Who** : Toute personne n'étant pas bénévole
**@brief** : Permet à un utilisateur de s'inscrire à la course
**@returns** : Renvoie true si l'utilisateur a été correctement inscrit, false sinon
**@comments** : Lien avec le microservice paiement pour le paiement de l'inscription
boolean joinRaceBenevole(int idRace, int idUser, List Documents)

# Profiles

Ce composant sert à faire le lien entre le gateway et les différents utilisateurs. Il gère les fonctions de base liées à la gestion d'un utilisateur et de sa connexion. Il permet de regrouper les fonctions communes aux participants, organisateurs et bénévoles qui hériteront de ce composant.

## Data

Liste d'utilisateurs (List<User>)

## Attributs

- **idUser** (int) : identifiant de l'utilisateur
- **recapitulatif** (list <Course>) : historique des courses réalisées
- **documents** (list <file, int>) : liste de documents (associé à leur identifiant de document)

## API

**@Who** : Tout le monde
**@brief** : Permet à cette personne d'upload un document sur son compte
**@returns** : Renvoie true si le document a bien été ajouté dans la base de données
boolean addDocument(File document)

**@Who** : Organisateur
**@brief** : Permet de trouver les bénévoles pour une course
**@returns** : Renvoie la liste des bénévoles d'une course
**@comments** : critère de spécialité optionnel ne prend pas en compte les bénévole déjà affecté à une autre course sur la même période (prendre en compte les refus ne pas redemander si un bénévole a déjà refusé)
list<int> searchBenevole(int idRace)
list<Benevole> searchBenevole(int idRace, list<String> specialties)

**@Who** : Tous
**@brief** : Permet de voir les courses auquel l'utilisateur a participé
**@returns** : Renvoie la liste des course passées avec le rôle associé.
**@comments** : peut renvoyer une liste vide
list<course, role> viewRecap(int idUser)

Inscription

**@Who** : Utilisateurs non inscrits
**@brief** : Permet de se créer un compte
**@returns** : true si le compte est créé false sinon (si par exemple le nom est pris)

boolean createAccount(String username, String email, Date birthdate)

Désinscription

**@Who** : Propriétaire du compte
**@brief** : Permet de supprimer son compte
**@returns** : true si le compte est supprimé false sinon
**@comments** :
boolean removeAccount()

Modification données du compte

**@Who** : Propriétaire du compte
**@brief** : Permet de modifier son compte
**@returns** : true si la modification est effective false sinon
**@comments** :
boolean moveMail(String newMail)
boolean moveBirthDate(Date realBirthday)

# Volunteers

Ce composant sert à gérer les bénévoles, et les informations qui leur sont liées.

- list <Disponibilite> Disponibilites

Définition du type Disponibilite :
Date start
Date end
list <String> cites

**@who** : Tout utilisateur non inscrit en tant que coureur à la course
**@brief** : Inscription d'un bénévole à la course
**@returns** : true si la demande d'inscription a bien été prise en compte, false sinon
**@comments** : L'organisateur peut s'inscrire en tant que bénévole s'il le désire
boolean applyRace(int idRace)

## API

Spécialité des bénévoles

**@Who** : Bénévole

**@brief** : Permet à un Bénévole de définir une spécialité
**@returns** : Renvoie true si la spe est correctement ajoute, false sinon
**@comments** : Peut il avoir plusieurs spécialités ? contrainte de choisir une spe dans une liste ou il ajoute ce qu'il veut ?
addSpecialty(String specialty)

**@Who** : Bénévole
**@brief** : Permet à un Bénévole de modifier une spécialité
**@returns** : Renvoie true si la spe est correctement modifié, false sinon
**@comments** : On suppose le nombre de specialité limite à définir
moveSpecialty(String oldSpecialty,String newSpecialty)

**@Who** : Bénévole
**@brief** : Permet à un Bénévole de supprimer une spécialité
**@returns** : Renvoie true si la spe est correctement supprimé, false sinon
**@comments** :
removeSpecialty(String Specialty)

**@Who** : Bénévole
**@brief** : Permet à un Bénévole de supprimer toute ses spécialité
**@returns** : Renvoie true si la/les spe sont correctement supprimés, false sinon
**@comments** :
removeAllSpecialty()

Disponibilité des bénévole

**@Who** : Bénévole
**@brief** : Permet à un Bénévole de définir une date ou période de disponibilité
**@returns** : Renvoie true si la disponibilité est correctement ajoute, false sinon
**@comments** : Possibilité de contraindre cette disponibilité par un lieu ou de choisir une période si pas de contrainte de lieu on suppose qu'il est dispo partout : any ?

addDisponibility(Date date)
addDisponibility(Date start, Date end)
addDisponibility(Date date,list<String> cities)
addDisponibility(Date start, Date end,list<String> cities)

**@Who** : Bénévole
**@brief** : Permet à un Bénévole de modifier les date ou les lieu d'une de ses disponibilités
**@returns** : Renvoie true si la/les spe sont correctement supprimés, false sinon

**@comments** : contrainte start <= end
moveDisponibility(int idDisponibility,Date start)
moveDisponibility(int idDisponibility,Date end)
moveDisponibility(int idDisponibility,Date start,Date end)
moveDisponibility(int idDisponibility,list<String> cities)

**@Who** : Bénévole
**@brief** : Permet à un Bénévole de supprimer une disponibilité
**@returns** : Renvoie true si la disponibilité est correctement supprimé, false sinon
**@comments** :
removeSpecialite(int idDisponibility)

Fiche de missions
**@Who** : Bénévole
**@brief** : Permet à un Bénévole de confirmer sa présence sur une course
**@returns** : Renvoie true si le bénévole est bien ajouté à la course (bénévole appro -> bénévole voir attribut course), false sinon
**@comments** :
acceptMission(int idRace)

**@Who** : Bénévole
**@brief** : Permet à un Bénévole de se désister sur une course
**@returns** : Renvoie true si le bénévole est bien supprimé de la liste des bénévole en attente d'approbation de la course (benevole appro -> X voir attribut course), false sinon
**@comments** :
refuseMission(int idRace)

App mobile bénévole

**@Who** : Bénévole
**@brief** : Permet de charger l'application pour les bénévoles
**@returns** : Renvoie true si appli telecharger false sinon
**@comments** : contrainte seulement avant le jour de la course ?
boolean loadApp()

# Organizers

Peuvent utiliser les fonctions de profils

## API

**@Who** : Organisateur d'une course donnée

**@brief** : Permet d'accepter un bénévole sur une mission
**@returns** : Renvoie true si le bénévole est accepté
**@comments** : Contrainte : seulement avant le jour de la course
acceptVolunteer(int idUser)

# Participants

Peuvent utiliser les fonctions de profils

**@comments** : encapsule la fonction joinRaceRunner

**@Who** : Participants
**@brief** : Permet à un participant d'enregistrer ses justificatifs
**@returns** : Renvoie true siles documents sont correctement ajoute, false sinon
**@comments** : Nombre de document obligatoire à définir
boolean joinRace(int idRace, List Documents)

**@Who** : Personne en train de s'inscrire à la course
**@brief** : Permet à cette personne d'aller chercher un document qu'il aurait enregistré sur son compte
**@returns** : Renvoie le document que la personne est allée sélectionner
File addDocumentRace(int idRace, File document)

# Payment

Ce composant sert à faire le lien entre l'API de la banque et notre application. Il utilise les coordonnées bancaires transmises par le coureur durant l'inscription. L'historique des paiement est sauvegardé par ce microservice.

## API

**@Who** : Coureurs
**@brief** : Paiement de l'inscription à la course
**@returns** : retourne true pour un paiement valide
**@comments** : Permet de réaliser le paiement (interaction avec l'api bancaire)
boolean racePayment(int idRace, float price, coordonnéesBancaire )

**@Who** : Microservice course
**@brief** : vérification du paiement d'inscription à la course
**@returns** : retourne 0 pour un paiement valide, un code d'erreur dans le cas contraire
**@comments** : Permet de vérifier le paiement.
int racePayment(int idRace, idCoureur )