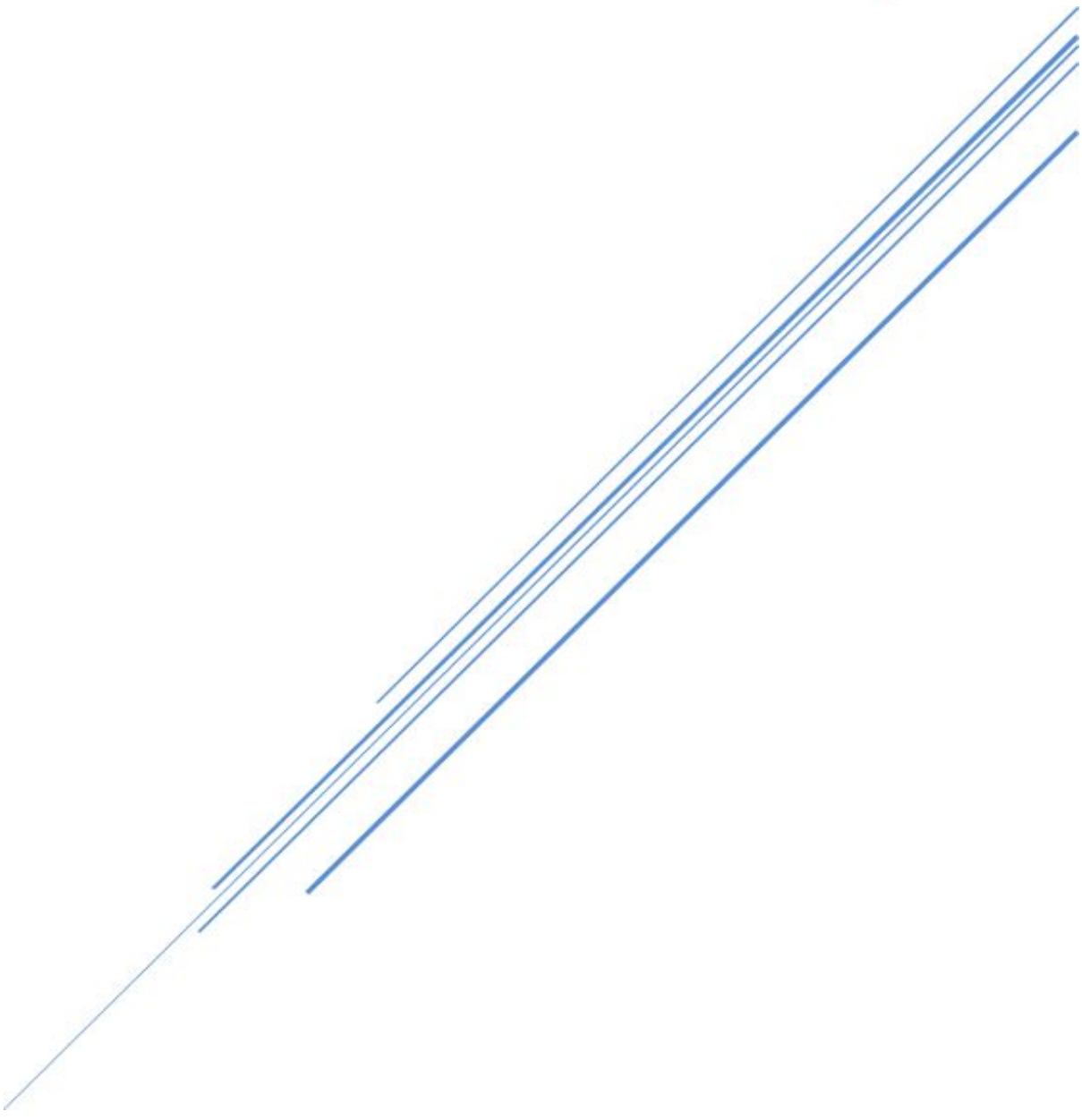
Sport Matching

Projet Web
IG3 - 2017



Kévin Hassan

Polytech Montpellier, France

Introduction

Etudiant en IG3 à Polytech Montpellier et redoublant, j'ai été pour la deuxième fois amené à réaliser un projet Web. Fort de mon expérience dans les projets piscine ainsi que le projet web de l'an dernier, j'ai voulu mettre en application ce que j'avais appris.

Ainsi, l'idée de mon projet a germé d'un constat que j'ai pu faire lorsque j'essayais avec mes camarades d'organiser une simple partie de basket. En effet, j'ai alors remarqué à quel point il pouvait être difficile de trouver un terrain de sport à libre accès même à Montpellier. Et pour peu que nous en trouvions un, il était déjà occupé.

C'est alors que m'est venu l'idée de mettre en place une plateforme web collaborative de cartographie des terrains de sport et d'organisation de match.

I. Cadre fonctionnel

J'ai alors commencé par chercher si l'idée que j'avais eu existait et comment j'aurais pu l'améliorer. Par la suite, j'ai pu me rendre compte que la mairie de Montpellier par exemple, listait les terrains en fonction du sport choisi mais impossible de connaître l'occupation.

C'est alors qu'en ajoutant cet aspect là ainsi qu'une dimension communautaire mon idée devenait inédite.

1) Analyse

Par la suite j'ai réfléchis aux différents cas d'utilisation afin de m'inspirer pour le développement de la future plateforme. J'en suis alors arrivé à ces points :

- il peut ajouter un terrain quelque soit sa ville
- il peut voir tous les terrains de sa ville ainsi que les parties organisées
- il peut se mettre sur liste d'attente s'il considère que lui et ses amis ne sont pas suffisamment nombreux pour jouer
- il peut organiser une partie quelque soit le nombre de participant et tous les utilisateurs en attente et à proximité recevront une notification
- il peut accepter ou non de rejoindre une partie

De cette analyse, j'en ai déduis la réel nécessité pour la plateforme d'ajouter un ou plusieurs administrateurs pour maintenir la plateforme et lutter contre les abus.
Par conséquent un administrateur peut :

- gérer les utilisateurs (Suppression, Ajouter des droits)
- gérer les villes et les terrains cartographiés

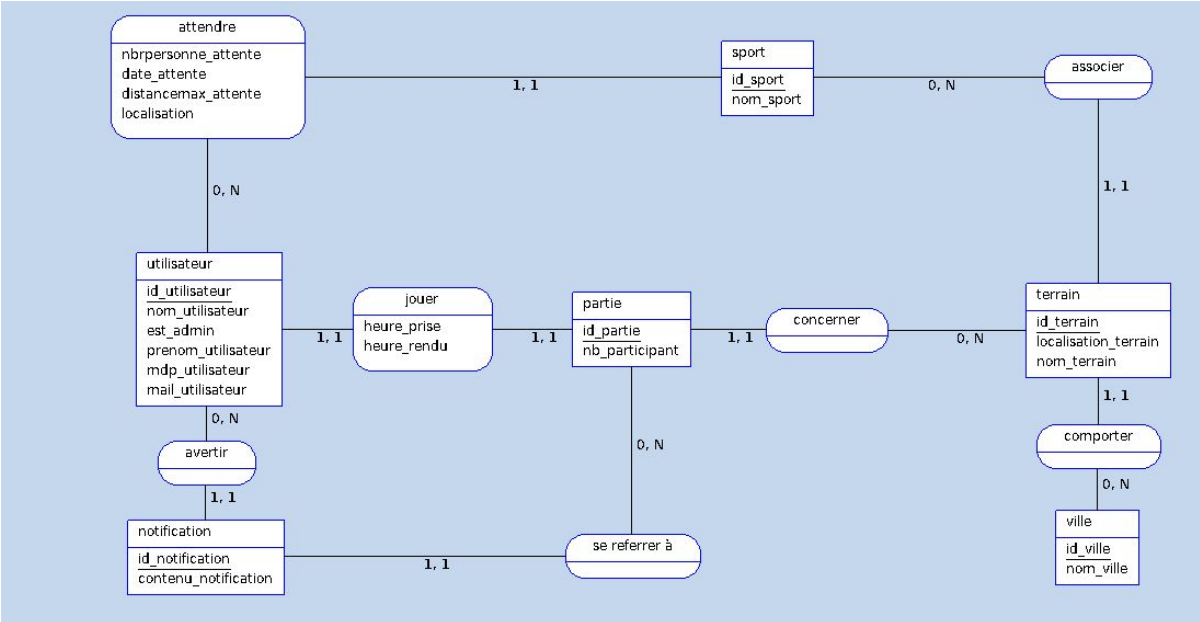
2) Modélisation

Dans un soucis de modélisation, je n'ai pas considéré les terrains multisports qui complexifient le système pour l'organisation de partie. J'ai alors volontairement imposé qu'un terrain sportif ne pouvait accueillir qu'un seul sport.

J'ai aussi considéré que les parties organisées par les membres ne pouvaient pas accueillir plus que le nombre maximum de joueur du sport (exemple : 10 joueurs maximum au basket).

De cette analyse, j'ai pu modéliser mon système sous la forme d'un MCD (voir ci-dessous).

1) Modèle conceptuel de données



2) Schéma relationnel

Après dérivation du diagramme ci-dessus, j'obtiens le schéma relationnel ci-dessous:

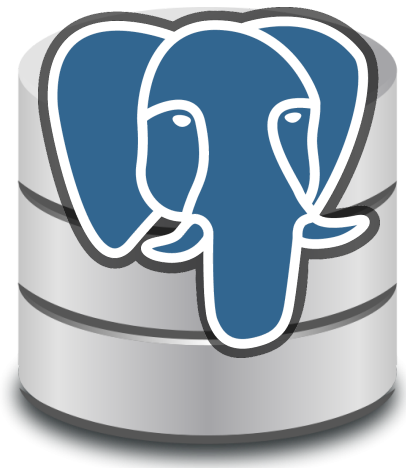
- utilisateur (id_utilisateur, nom_utilisateur, est_admin, prenom_utilisateur, mdp_utilisateur, mail_utilisateur)
- partie (id_partie, nb_participant, heure_prise, heure_rendu, #id_terrain)
- notification (id_notification, contenu_notification, #id_utilisateur, #id_partie)
- sport (id_sport, nom_sport)
- terrain (id_terrain, localisation_terrain, nom_terrain, #id_sport, #id_ville)
- ville (id_ville, nom_ville)
- attendre (#id_utilisateur, #id_sport, nbrpersonne_attente, date_attente, distancemax_attente,localisation)

II. Architecture technique

1) Base de données

Pour ma base de données j'ai fais le choix d'utiliser l'outil libre PostgreSQL, un SGBD qui nous a souvent été conseillé par nos enseignants, il m'a alors paru légitime car celui-ci s'adapte parfaitement à mon système qui aura pour vocation de stocker de très gros volume de données (coordonnées GPS de terrain dans toute la France).

De plus, j'ai dû rajouter le système d'information géographique PostGIS afin de stocker et de calculer des distances entre des points géographiques.



a) Tables et contraintes

```
create table attendre
(
  nbr_personne integer default 1 not null,
  date timestamp with time zone not null,
  distance_max integer not null,
  localisation geometry(Point,4326) not null,
  sport_id integer not null
    constraint attendre_sport_id_fkey
      references sport
        on update cascade on delete cascade,
  utilisateur_id integer not null
    constraint attendre_utilisateur_id_fkey
      references utilisateur
        on update cascade on delete cascade,
  constraint attendre_pkey
    primary key (sport_id, utilisateur_id)
);
create table notification
(
  id serial not null
    constraint notification_pkey
      primary key,
  contenu varchar(255) not null,
  utilisateur_id integer
    constraint notification_utilisateur_id_fkey
      references utilisateur
        on update cascade on delete cascade,
  partie_id integer
    constraint notification_partie_id_fkey
      references partie
        on update cascade on delete cascade
);
create table partie
(
  id serial not null
    constraint partie_pkey
      primary key,
  nbr_participant integer not null,
  heure_prise timestamp with time zone not null,
  heure_rendu timestamp with time zone not null,
  terrain_id integer
    constraint partie_terrain_id_fkey
      references terrain
        on update cascade on delete cascade,
  utilisateur_id integer
    constraint partie_utilisateur_id_fkey
      references utilisateur
        on update cascade on delete cascade
);
create table sport
(
  id serial not null
    constraint sport_pkey
      primary key,
  nom varchar(255) not null,
  nbr_max_participant integer not null
);
create table terrain
(
  id serial not null
    constraint terrain_pkey
      primary key,
  localisation geometry(Point,4326) not null
    constraint terrain_localisation_key
      unique,
  nom varchar(255),
  sport_id integer
    constraint terrain_sport_id_fkey
      references sport
        on update cascade on delete cascade,
  ville_id integer
    constraint terrain_ville_id_fkey
      references ville
        on update cascade on delete cascade
);
create table utilisateur
(
  id serial not null
    constraint utilisateur_pkey
      primary key,
  nom varchar(255) not null,
  prenom varchar(255) not null,
  mail varchar(255) not null
    constraint utilisateur_mail_key
      unique,
  mdp varchar(255) not null,
  est_admin boolean default false not null
);
create table ville
(
  id serial not null
    constraint ville_pkey
      primary key,
  nom varchar(255) not null
);
```

J’ai fait le choix d’imposer aux contraintes de clé étrangère un “ON DELETE CASCADE” afin de maintenir la cohérence dans la base de données pour ne pas avoir de trace inutile en cas de suppression d’une entité impliqué dans une relation avec une autre.

De plus, pour le stockage des coordonnées j’ai choisi de le faire sur le SRID 4326 car c’est la référence utilisée par de nombreuses API et librairie (Google Maps et Leaflet)

b) Trigger

Le trigger est déclenché dès qu’une partie est ajoutée. Il consiste à prendre le sport concerné par cette partie à récupérer le nombre maximum de participant pour ce sport et à comparer avec les utilisateurs en attente de partie afin que la somme :

Nombre de participant + Nombre de personne en attente < Nombre de personne maximum

De plus, grâce à PostGIS je calcule les distances entre ces utilisateurs et le terrain, et j’exclus ceux dont la distance maximale est franchie. Ainsi un utilisateur répondant à tous ces critères recevra une notification.

```
CREATE OR REPLACE FUNCTION check_partie_nbr() RETURNS TRIGGER AS
$BODY$
  DECLARE
    nb_max INTEGER;
    idSport INTEGER;
    personne INTEGER;
  BEGIN
    --récupérer le nombre maximum de joueur pour le sport concerné
    nb_max =(SELECT nbr_max_participant from terrain INNER JOIN sport ON
terrain.sport_id = sport.id WHERE
      new.terrain_id = terrain.id);
    idSport=(SELECT sport.id from sport,terrain WHERE terrain.id=NEW.terrain_id
 and terrain.sport_id=sport.id);

    IF (new.nbr_participant<nb_max) THEN--Cas ou il reste de la place pour d'
autre joueur
      --Regarder tous les joueurs en attente à proximité du terrain
      --Récupérer le nombre de joueur en attente (seulement s'ils sont
inférieurs au nombre maximum du sport)

      FOR personne IN SELECT utilisateur_id
                      FROM attendre,terrain
                      WHERE attendre.sport_id=idSport AND new.nbr_participant+
attendre.nbr_personne <= nb_max
      AND NEW.terrain_id = terrain.id AND (SELECT ST_Distance(
            ST_Transform(attendre.localisation,26986),
            ST_Transform(terrain.localisation,26986)
          ))/1000<=attendre.distance_max
      LOOP
        --Ajouter une notification à l'utilisateur concerné
        INSERT INTO notification(contenu,utilisateur_id,partie_id)
        VALUES('Partie disponible à proximité',personne,NEW.id);
      END LOOP;
    END IF;
    RETURN NEW;
  END;
$BODY$
LANGUAGE plpgsql;
```

2) API Rest NodeJS

a) Choix techniques

Pour réaliser ce projet j'ai fait le choix d'utiliser des technologies assez récentes comme NodeJS pour le backend. En effet, c'est une technologie de plus en plus souvent utilisée sur des projets. De plus, le framework Express permet d'abstraire le bas niveau de NodeJS.

En effet, la réalisation d'une application RESTful qui utilise le protocole HTTP avec des URI simples tout en respectant ses standards s'y prête bien.

Afin de distribuer convenablement les routes de mon API j'ai utilisé le module "Router" d'Express tout en veillant à respecter les verbes HTTP (GET, POST, DELETE, PUT).

De plus, j'ai ajouté à NodeJS l'ORM Sequelize basé sur les promises de Bluebird qui me permet de définir mes models et les requêtes tout en s'abstrayant du dialecte utilisé (Mysql, Postgres,...).

Et enfin, j'utilise l'API de Google Maps afin récupérer le nom de la ville correspondant aux coordonnées géographique grâce à sa fonction de "georeverse".

b) Authentification et sécurité

D'autre part, mon application requiert d'être authentifié sur presque toutes les routes (sauf **/login** et **/register**). Par conséquent, je devais mettre en place une authentification sans mettre en péril la scalabilité de mon application. Pour cela, 2 choix se portaient à moi : **Cookie** ou **Token**.

J'ai fait le choix d'utiliser des tokens car ils permettent d'échanger entre un serveur et différentes applications clientes des informations d'utilisateur et de rôles de manière stateless. Ils sont signés et cryptés pour éviter d'être modifiés côté client et sauvegardé dans le WebStorage. La génération du token se fait à chaque nouvelle connexion et à chaque requête le token est passé dans le Header.

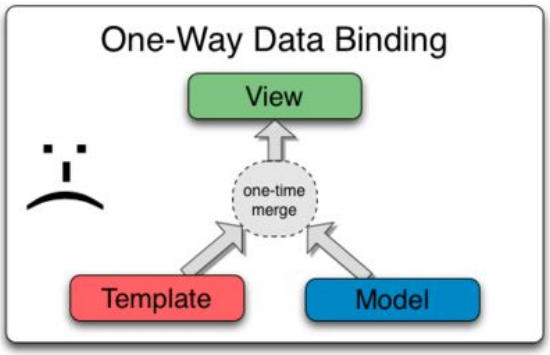
J'ai alors décidé de stocker dans le payload l'id de l'utilisateur, son rôle ainsi qu'une date d'expiration (1j) crypté avec l'algorithme "HS512".
De plus, les mots de passe des utilisateurs sont cryptés avec le module "crypto" en AES-256 et un salt y est ajouté.

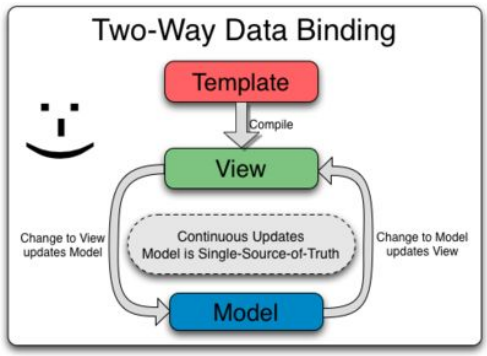
3) Client AngularJS

a) Choix techniques

J'ai fait le choix d'implémenter mon client en AngularJS car le cours d'introduction sur cette technologie m'a redonné envie de l'utiliser malgré un précédent échec.

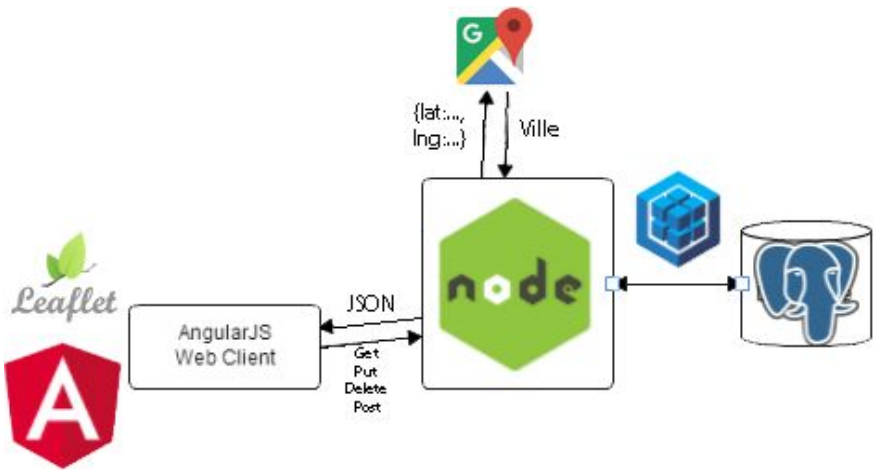
Car malgré tout ce framework by Google est très bien documenté et le Two-Way-Data Binding apporte un dynamisme aux pages non négligeable.





De plus, j'ai pris exemple sur la manière de structurer un code client grâce à "angular-seed" disponible sur Github. Dès lors, l'implémentation de mon client avec les controllers, les factories et les services m'a paru beaucoup plus clair.
En effet, j'utilise les factories qui consomment les services de mon API via le service $http.
En outre, j'ai dû utiliser la librairie open-source "LeafletJS" pour la cartographie intéractive, celle-ci étant basé sur le layer d'OpenStreetMap.

Et enfin j'utilise le navigator geolocation en html5 pour récupérer les coordonnées géographique de la personne afin de les traiter.



b) Design

Le design du site repose presque intégralement sur l'utilisation du framework CSS Semantic-UI que j'aime beaucoup et dont la documentation est très riche.

III. Architecture de déploiement

J'ai choisi Heroku comme solution de déploiement car d'une part il est gratuit et d'autre part il propose une base de donnée en Postgres.
De plus, il s'avère simple à utiliser ce qui facilite le déploiement.
Néanmoins, il y avait quelques restrictions sur l'offre gratuite :

- 10000 entrées (dont ~500 réservées par PostGIS)
- 20 connexions simultanées
- L'application reste éveillée 18/24h

Malgré cela, cette solution gratuite est suffisante pour répondre aux besoins de l'application

Le serveur d'API :
https://hassan-webproject.herokuapp.com/api/v1
Le client AngularJS :
https://sportmatching.herokuapp.com/

IV. Conclusion

Ce projet m'avait déjà apporté beaucoup l'année dernière même si je considère toujours que c'était un échec. J'ai appris de mes erreurs et j'ai essayé de ne plus les reproduire et il me semble pas que ça soit le cas.

De plus, j'ai pris plaisir à travailler avec certains de mes camarades. Certains rencontrés des problèmes et j'ai essayé de les aider le plus possible même si nous n'utilisions pas les même technologies. Et j'ai aussi reçu de l'aide quand j'en avais besoin. En effet, au-delà de savoir ce qu'on peut sortir comme application web en un temps restreint tout en étant soumis à des contraintes, c'est avant tout l'aventure humaine qui me marque le plus cette année.

Pour conclure, je considère avoir parfois perdu beaucoup de temps à lire la documentation des modules et des librairies que j'utilisais pour bien comprendre comment les utiliser plutôt que de "coder". Néanmoins, je suis content du travail que j'ai apporté et des connaissances que j'ai pu accumuler aux cours de ces nombreuses recherches.
J'espère que vous en serez satisfait.