RENDU PROJET WEB

MAMUSCU

Hugo Maitre IG3 –2017

Sommaire

Introduction

Le projet

MaMuscu est un site qui permet à un utilisateur d'organiser ses différents entraînements de musculation et lui donnant la possibilité d'avoir un suivi de ses différentes performances.

En effet l'utilisateur à la possibilité de gérer ses données qui lui sont propres. Aucune donnée n'est partagée entre les utilisateurs.

Arrivé sur le site l'utilisateur rajoutera ses différentes machines de musculation en sa possession, les différents exercices qui lui plaisent puis les différents entraînements qui seront composés de ses exercices.

Ce site est très simple utilisation et permet de ne pas perdre de temps lors de la composition de différents entraînements.

De plus, l'utilisateur peut rajouter des performances sur ses différents entraînements ce qui lui permet d'avoir un suivi de sa progression au fil du temps.

MaMuscu est donc une application incontournable pour tous les sportifs qui veulent aller au plus vite pour la création et la consultation de ses différents entraînements.

Ce projet à un but d'utilisation purement personnel ou à un groupe réduit de personne.

Pourquoi ce projet ?

L'été arrive à grands pas et beaucoup de gens veulent être en forme et bien sculpté pour aller sur la plage. Car être en bonne santé et en forme change tout sur le bien-être d'une personne.

C'est mon cas en tout cas. Et ce qui me dérange c'est de ne pas pouvoir être efficace lors de la composition de mes entraînements et de ne pas avoir d'interfaces simples permettant de consulter les différents exercices que je dois faire lors de mes séances…

C'est pour cela que j'ai décidé de réaliser ce projet.

Projet Web Polytech 2017 – Hugo Maitre

Spécification fonctionnelle

Analyse

Tout d'abord, il m'a fallu voir si l'application existait déjà.
Sans grande surprise j'ai remarqué que quelques applications de ce type étaient déjà sur le marché. Mais je les trouvais toutes trop compliquées pour l'utilisation que je voulais en faire. Je me suis donc dit que mon projet était assez différent et donc avait une place à prendre dans le monde des applications.

Ensuite, il m'a fallu trouver les différentes cibles de mon projet. Et je me suis rendu compte que mon application devait être accessible par tout le monde, que ce soit pour des enfants que pour des vieilles personnes. Il fallait donc que je crée une « friendly interface » dans le but de rendre l'utilisation de mon application la plus simple possible.

Opérations

Les différentes opérations possibles sur l'application :

- Pas de gestion admin, utilisation personnelle ou à comité restreint, donc accès direct à la base de données.

- Un utilisateur à ses propres données et ne sont pas partagés.

- Gérer ses données, pouvoir les supprimer.

- Modification non possible : c'est un choix qui a été dur à prendre mais lorsqu'un utilisateur cré une quelconque donnée il ne peut que la supprimer. C'est pour éviter de fausser les résultats sur les différentes performances. (Explication techniques plus bas).

- Création de différents entraînements grâce aux données créées en amont par l'utilisateur. En donnant un temps approximatif de l'entraînement complet.

- Ajouter des performances liées aux entraînements créés.

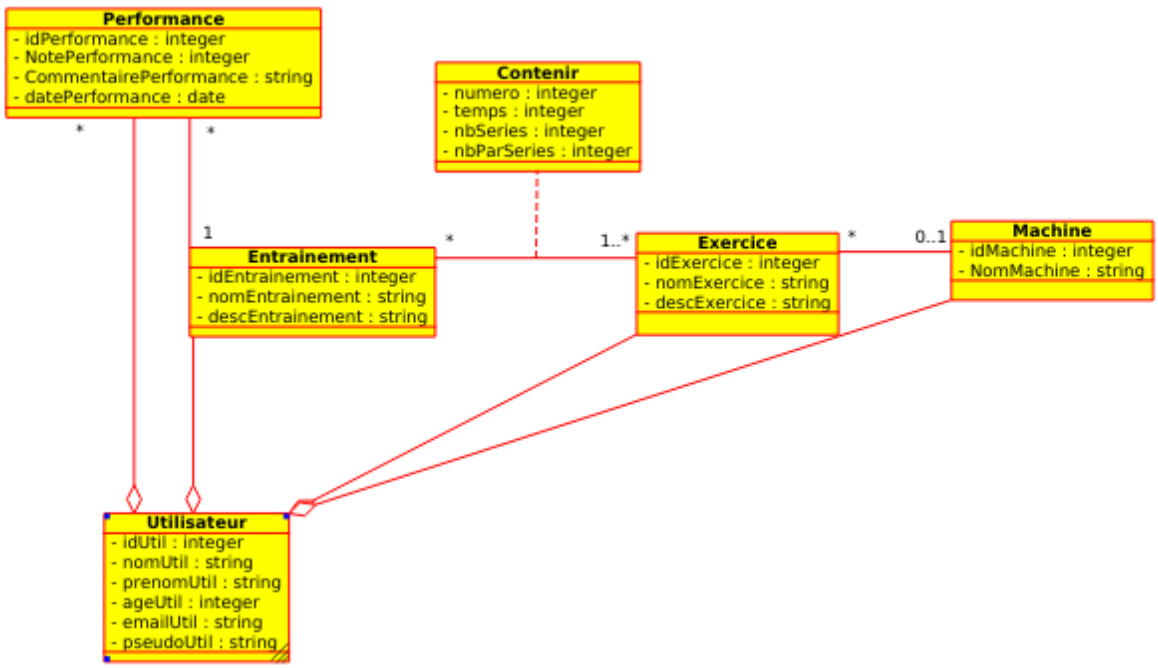
- Pouvoir consulter ses différents entraînements grâce à une interface simple lors d'une séance.

- Pouvoir consulter ses performances et quelques statistiques liés à l'utilisateur.

Modélisation

UML

Grâce à ses différentes directives j'ai pu réaliser en UML les différentes interactions entre les différents objets de mon application :



Mon uml se compose donc de 6 classes.

- Un utilisateur est composé de ses données (toutes les tables reliées à lui).

- Un exercice a au maximum une machine, une machine peut avoir de multiples exercices

- Un entraînement est composé de multiples exercices et un exercice peut être utilisé dans de multiples entraînements (même plusieurs fois dans le même ! Important).

- Une performance est lié à un entraînement et un entraînement peut avoir plusieurs performances.

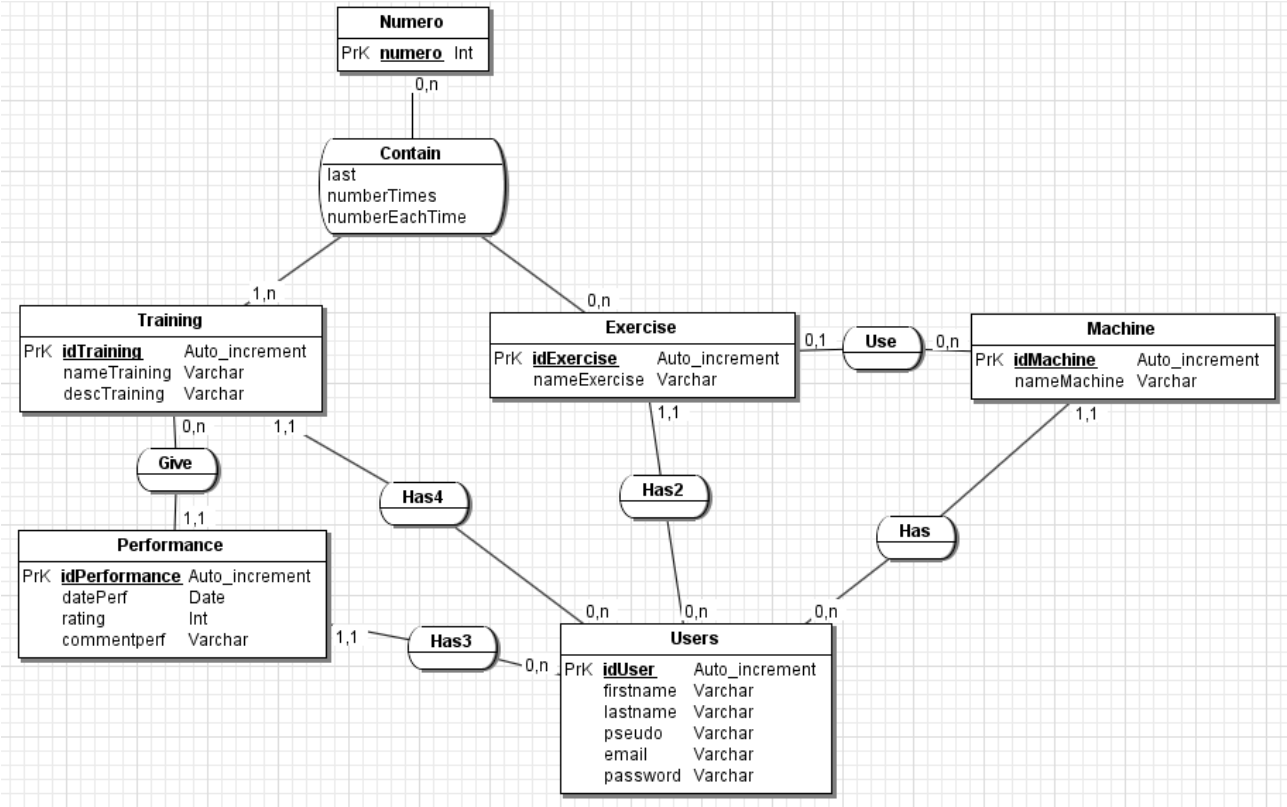
- Un exercice est soit du type Cardio il aura donc ses données nbSeries et nbParSerie non existantes ou du type musculation et nbSeries et nbParSerie sera donc existantes. Les exercices sont classés par numéro dans un entraînement ce qui permet d'avoir une chronologie dans l'entraînement. Et chaque exercice dure n minutes dans l'entraînement.

On peut bien voir ici que lorsque l'on modifie une donnée cela impacte le reste des données. Et du coup les performances sont faussées. Je vois le site comme quelque chose qui dure dans le temps. Chaque donnée créée doit perdurer jusqu'à la fermeture du compte. Bien sûre on peut laisser l'utilisateur supprimer une donnée s'il s'est tromper lors de l'ajout. Mais pour ce qui est de la

modification je ne veux pas qu'il puisse modifier car les performances ne feront plus référence à l'entraînement effectué à la base.

Dérivation

MCD



Par dérivation on arrive donc au MLD suivant :

Users (iduser, firstname, lastname, pseudo, email, password)

Machine (idmachine, namemachine, iduser)

Exercise (idexercise, nameexercise, descexercise, iduser, idmachine)

Training (idtraining, nametraining, desctraining, iduser)

Contain (idtraining, idexercise, numero, last, numbertimes, numbereachtime)

! Ici idtraining, idexercise et numero sont tous les trois des clés primaires car un exercice peut se trouver plusieurs fois dans un entraînement mais à une position différente dans le temps. Le numéro permet de créer cette chronologie.

Performance (idperformance, dateperf, rating, comment, iduser, idtraining)

SGBD

Pour ma base de données j'ai choisi d'utiliser postgresql.

Ce choix a été fait grâce aux différents conseils et cours que j'ai eu au cours de l'année. Je me sentais déjà plus en sécurité car je connaissais cet SGBD. De plus il s'est avéré que la documentation de postgresql est très bien faite et a une communauté assez présente malgré que la communauté de MySQL soit beaucoup plus grosse. Mais après quelques recherches j'ai remarqué que postgresql offre beaucoup plus d'outils que MySQL et que peut-être ces outils m'auraient été utile. Il s'est avéré que non mais on ne sait jamais.
De plus, en regardant quelques benchmarks de postgresql comparé à ceux de MySQL, on peut voir clairement que postgresql est beaucoup plus performant lors de l'écriture que de la lecture même si postgresql est un petit peu en dessous au niveau de la lecture. Ces différents critères ont fait que j'ai penché pour l'utilisation de cet SGBD.

Mes tables SQL :

```sql
CREATE TABLE Users (
  idUser serial,
  firstname character varying(32) NOT NULL,
  lastname character varying(32) NOT NULL,
  pseudo character varying(32) NOT NULL,
  email character varying(64) NOT NULL,
  password character varying(64) NOT NULL,
  CONSTRAINT pk_users PRIMARY KEY (iduser),
  CONSTRAINT un_users UNIQUE (pseudo, email)
);


CREATE TABLE Machine (
  idMachine serial,
  nameMachine varchar(50) NOT NULL,
  iduser integer NOT NULL,
  CONSTRAINT pk_machine PRIMARY KEY (idMachine),
  CONSTRAINT fk_machine_user FOREIGN KEY (iduser) REFERENCES users (iduser) ON DELETE CASCADE
);


CREATE TABLE Exercise (
  idExercise serial,
  nameExercise varchar(32) NOT NULL,
  descExercise varchar(128) NOT NULL,
  idMachine integer,
  iduser integer NOT NULL,
  CONSTRAINT pk_exercise PRIMARY KEY (idExercise),
  CONSTRAINT fk_exercise_user FOREIGN KEY (iduser) REFERENCES users (iduser) ON DELETE CASCADE,
  CONSTRAINT fk_exercise_machine FOREIGN KEY (idmachine) REFERENCES machine (idmachine) ON DELETE SET NULL
);
```

```sql
CREATE TABLE Training (
  idTraining serial,
  nameTraining varchar(32) NOT NULL,
  descTraining varchar(128) NOT NULL,
  idUser integer NOT NULL,
  CONSTRAINT pk_training PRIMARY KEY (idTraining),
  CONSTRAINT fk_training_user FOREIGN KEY (iduser) REFERENCES users (iduser) ON DELETE CASCADE
);

CREATE TABLE Performance (
  idPerformance serial,
  dateperf date NOT NULL,
  rating integer NOT NULL,
  comment varchar(256) NOT NULL,
  iduser integer NOT NULL,
  idtraining integer NOT NULL,
  CONSTRAINT pk_performance PRIMARY KEY (idPerformance),
  CONSTRAINT fk_performance_user FOREIGN KEY (iduser) REFERENCES users (iduser) ON DELETE CASCADE,
  CONSTRAINT fk_performance_training FOREIGN KEY (idTraining) REFERENCES Training (idTraining) ON DELETE CASCADE
);

CREATE TABLE Contain (
  idExercise integer NOT NULL,
  idTraining integer NOT NULL,
  numero integer NOT NULL,
  last integer NOT NULL,
  numberTimes integer,
  numberEachTime integer,
  CONSTRAINT pk_contain PRIMARY KEY (idExercise, idTraining, numero),
  CONSTRAINT fk_contain_training FOREIGN KEY (idTraining) REFERENCES Training (idTraining) ON DELETE CASCADE,
  CONSTRAINT fk_contain_exercice FOREIGN KEY (idExercise) REFERENCES Exercise (idExercise) ON DELETE CASCADE
);
```

Les principales contraintes utilisées sont des clés primaires et clés étrangères. Presque toutes sont caractérisés par ON DELETE CASCADE qui permet de supprimer un tuple d'une autre table si cette table contenait la clé étrangère d'un tuple que l'on vient de supprimer. Un seul utilise la contrainte ON DELETE SET NULL car lorsque je supprime une machine je ne veux pas forcément supprimer les exercices qui sont liés à cette machine mais je veux quand même changer la valeur de la machine dans cet exercice par NULL.

Trigger

Pour ce projet j'ai utilisé un trigger.

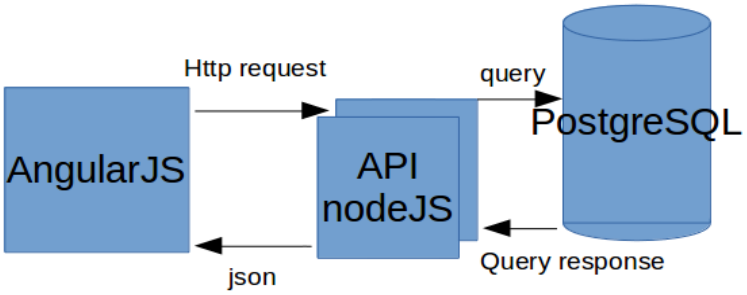
```sql
CREATE OR REPLACE FUNCTION supp_empty_trainings() RETURNS trigger AS
$BODY$
DECLARE id integer;
BEGIN
  FOR id IN ((SELECT idTraining FROM training WHERE iduser=OLD.iduser)
                EXCEPT
             (SELECT idTraining FROM training NATURAL JOIN contain WHERE iduser= OLD.iduser))
  LOOP
    DELETE FROM training WHERE idtraining = id;
  END LOOP;
  RETURN NEW;
END
$BODY$ LANGUAGE plpgsql VOLATILE COST 100;

CREATE TRIGGER trigg_supp_empty_trainings
AFTER DELETE ON exercise
    FOR EACH ROW EXECUTE PROCEDURE supp_empty_trainings();
```

Ce trigger a pour but de supprimer les entraînements qui ne contiennent plus aucun exercice existant. A chaque suppression d'exercice on va vérifier si tous les entraînements ont bien au moins un exercice qui lui sont liés. Généralement un utilisateur ne supprimera pas un exercice mais lorsqu'il le fera il faut penser à vérifier les entraînements qui le contient.

Les différentes technologies utilisées

Il y a plusieurs semaines je me suis beaucoup intéressés sur le langage javascript. Je me suis rendu compte que ce langage était tout à fait exceptionnel. Il a quelques lacunes mais comparés à d'autres langages il met à disposition des outils assez abstraits comme les événements asynchrones qui permettent au code de s'exécuter avec une fluidité majestueuse.

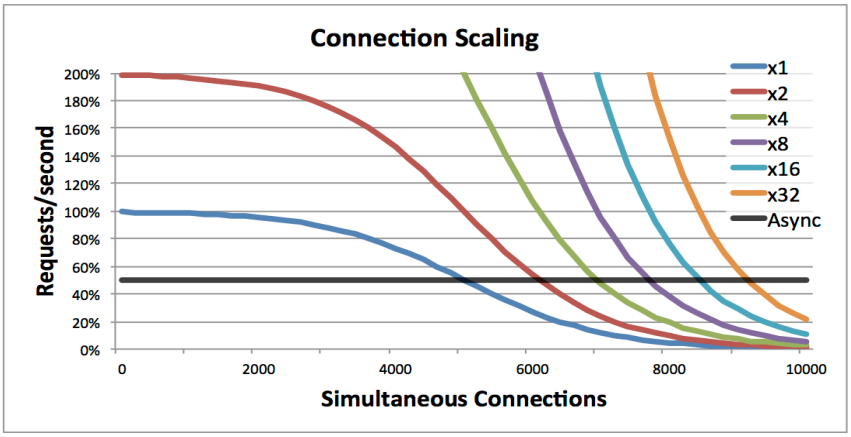
C'est pour cela que j'ai choisi pour mon projet d'utiliser une implémentation sofea (Service-Oriented Front-Architecture) qui utilise NodeJS pour l'API et AngularJS pour l'application cliente.



End

NodeJS

Pour le projet j'ai décidé d'utiliser NodeJS couplé au framework express. Express est juste une couche se trouvant au-dessus du code source de NodeJS très avancé. Ce qui a permis de simplifier le code permettant d'implémenter le parcours des différentes requêtes lorsqu'elles arrivent au serveur. J'ai utilisé NodeJS car comme je l'ai dit précédemment, c'est du javascript et je voulais vraiment apprendre ce langage. Il permet d'avoir un serveur très performant et rapide grâce à son côté asynchrone. Tous les modules sont téléchargés grâce à npm qui est un gestionnaire de librairies public.



Comment le serveur fonctionne ?

Chaque requête passe par des fonctions ordonnées une par une validant au fur et à mesure la requête ou pour récupérer les différentes informations la concernant. Ces fonctions s'appellent « middlewares » et appartiennent au Framework express.

Les derniers « middlewares » sont celles qui permettent de renvoyer une réponse à l'utilisateur (sans erreur). Ces différents « middlewares » sont appelés des routes car ce sont elles qui sont choisies en fonction de l'URI passé en paramètre.

Les différentes routes du serveur :

Authentification	
POST	/authenticate
POST	/register

Machines	
GET	/api/machines
GET	/api/machines/:id
POST	/api/machines
DELETE	/api/machines/:id

Exercises	
GET	/api/exercises
POST	/api/exercises
DELETE	/api/exercises/:id

Trainings	
GET	/api/trainings
GET	/api/trainings/:id
POST	/api/trainings
POST	/api/trainings/:idtraining/exercises/:idexercise
DELETE	/api/trainings/:id

Performances	
GET	/api/performances
GET	/api/performances/:id
GET	/api/performances/statistiques
POST	/api/performances
DELETE	/api/performances/:id

Users	
GET	/api/user
DELETE	/api/user

Je ne mets en avant ici seulement les routes utilisées.
Il y a principalement des GET qui permettent de récupérer des informations liées à toutes les tables. Les POST qui créent de l'informations et les DELETE qui suppriment de l'information. Dans le code j'ai écrit quelques fonctions de modifications liées au verbe PUT mais c'était au cas où. Je les laisse pour ne pas avoir à les réécrire si un jour je trouve une utilité à leur existence.
On peut voir qu'il y a deux types de routes distinctes, celles ayant /api dedans et celles qui ne l'ont pas. Cela permet tout simplement de filtrer les requêtes ou l'utilisateur doit être identifié et celle où l'on n'est pas obligé de l'être. Pour les requêtes /api l'utilisateur doit être authentifié, pour les autres non.

Grâce à un middleware, chaque requête contenant /api au début passe par une fonction de vérification, grâce à un token on vérifie si l'utilisateur à le droit de passer ou non.

Authentification

En parlant d'utilisateur authentifié, j'utilise pour gérer les connexions des utilisateur jsonwebtoken. Grâce à un module de npm, il est très simple de l'implémenter. Et en ayant trouvé un tutoriel simple sur internet, j'ai pu très facilement implémenter mon authentification et la création de comptes (ref : « http://thejackalofjavascript.com/architecting-a-restful-node-js-app/ ») que ce soit au niveau du serveur que du client.

Ce tutoriel ne m'a pas simplement aidé dans la gestion des connexions mais aussi pour structurer mon API.

Et pour ce qui est de la sécurité, j'utilise bcrypt qui est aussi un module npm utilisable qui permet de crypter très facilement un mot de passe et de comparer. Le grain de sel est compris dans le mot de passe crypté et est basé sur un entier entré en paramètre lors du cryptage.

AngularJS

Pour ce qui est de l'application client, j'ai utilisé AngularJS.

Le fonctionnement d'AngularJS

AngularJS créée par des équipes de Google n'est autre qu'une couche supérieure au javascript en particulier une couche supérieure à jquery qui permet de gérer tous les événements de manière locale. En effet grâce à différents modules tels que « routeProvider » et à sa directive nommé « ng-view » on peut créer des routes au niveau du client. On n'a plus besoin de demander au serveur pour aller sur telle ou telle page. Tout est géré dans le navigateur internet de l'utilisateur. Les seules interactions avec le serveur sont celles qui permettent d'échanger de la donnée entre le navigateur et le serveur. Ce qui permet de rendre totalement indépendant le serveur du client. De plus comme je l'ai dit précédemment, le javascript permet de gérer des événements de manière totalement asynchrone. Ce mélange parfait ne peut rendre l'interaction entre le navigateur et l'utilisateur que très fluide et rapide.

Modules utilisés

Pour mon projet j'ai donc utilisé le module $http permettant de faire des requêtes ajax à mon API, $routeProvider pour la gestion des différentes pages localement, $scope pour avoir accès aux différents éléments angular présent dans la page. Ajouté à des « filters » qui permettent de faire de la recherche basée sur des noms d'objets et donc de simplifier la recherche pour un entrainement ou un exercice.

Pour l'affichage d'une échelle de 1 à 10 avec des étoiles j'ai utilisé un module nommé jkratingstar qui propose une directive affichant les étoiles et récupère les valeurs.

Angular propose de créer des services, c'est à dire des objets utilisables dans les controllers grâce à des injections. J'ai principalement créés es services appelés factory qui permettent de factoriser des fonctions. Et ces factory sont principalement utilisés dans mon projet pour faire les requêtes ajax au serveur.

De plus grâce à un système de stockage local présent dans les navigateurs web, j'ai pu implémenter, avec l'aide du tutoriel au-dessus, un système de session local. En effet les informations de l'utilisateur et le token sont stockés localement et même si l'utilisateur recharge sa page il restera connecté. Tout cela grâce à des événement asynchrones utilisé par la méthode run de $rootScope.

Les différentes pages

Mon site se compose de 7 pages :

- La page de connexion
- La page d'inscription
- La page d'accueil : montre les dernières performances et des statistiques
- La page des entrainements : gestion des entrainements de l'utilisateur
- La page des exercices : gestion des exercices et machines de l'utilisateur
- La page des données de l'utilisateur : montre les données de l'utilisateur et possibilité de supprimer son compte
- La page montrant un entrainement en particulier : accessible en cliquant sur un entrainement dans la page des entrainements

Le CSS

Pour le style de mon application j'ai utilisé le framework css semantic ui. Mais n'ayant pas du tout une âme d'artiste j'ai eu du mal à trouver la combinaison parfaite. Que ce soit au niveau des couleurs que la disposition des différents éléments du DOM. De plus avec angular, semantic n'est fonctionnel qu'à moitié car semantic a besoin d'activer ses différents modules manuellement au chargement de chaque page. Ce qui a pour conséquence de ne pas actualiser les éléments qui changent continuellement. J'ai donc juste gardé le côté esthétique de semantic mais laissé de côté les éléments dynamiques de ce framework css. Ensuite j'ai créé quelques éléments css à la main.

De plus j'ai utilisé quelques icônes sur l'application. Ces icônes viennent de l'API google et de font awesome.
Mon site est, pour la plupart des pages responsive.
Je me considère vraiment comme un débutant en css et j'ai eu du mal à tout bien placer comme il faut. Le css est sûrement mon plus gros point faible.

Le déploiement

Pour déployer mon application j'ai utilisé heroku et mon raspberry.

Sur heroku j'ai créé deux applications, une nommé api-mamuscu.herokuapp.com pour l'API et un nommé mamuscu.herokuapp.com pour l'application cliente.
Sur mon raspberry mon application cliente est disponible à l'adresse mamuscu.sytes.net:8080 et mon api est disponible à l'adresse mamuscu.sytes.net:3000.

Peu importe l'adresse, les deux API discutent sur la même base de données liée à l'application heroku mamuscu.herokuapp.com mais se trouvant sur les serveurs d'amazon.

Adresses web:

https://api-mamuscu.herokuapp.com (API)

https://mamuscu.herokuapp.com (CLIENT)

http://mamuscu.sytes.net:3000 (API)

http://mamuscu.sytes.net:8080 (CLIENT)

Github

Pour avoir accès à mon code de partout j'ai utilisé Github. Il m'a permis de pouvoir coder sur différents périphériques sans problèmes et vu que heroku utilise git pour déployer des applications, j'ai vite fait mon choix.

Adresse Github :

https://github.com/HmFlashy/WebProject

Conclusion

Pour conclure, ce projet a été très éprouvant. J'ai beaucoup avancé la première semaine. Et peaufiner le code la deuxième semaine. Et ajouté aux différents partiels ça a été une période assez difficile et je me dis que les efforts donnés lors de la première semaine n'ont pas été en vain.

Mais ce projet m'a permis de démultiplier mes connaissances en javascript même s'il me reste le plus gros je pense à découvrir.

Au tout début je n'avais pas du tout ce projet en tête, mais c'est en écrivant le cahier des charges de l'autre application que je me suis rendu compte que mon projet de départ aurait été impossible à terminer dans les temps car il mettait en avant une base de données beaucoup trop importante.

Ce projet était parfait car il a une taille parfaite pour deux semaines de travails, il mettait en œuvre toutes les compétences requises et plus. De plus je vais l'utiliser personnellement pour mes entraînements.

Je pense qu'il y a quelques problèmes que je n'ai pas réussi à identifier mais dans le futur j'essaierai de l'améliorer et qui sais peut-être ramener une communauté sur mon application. Créer une sorte de réseau social, avec un système d'amis etc. Mais pour le moment, l'application reste à utiliser de manière personnelle car il faut avoir accès à la base de données pour administrer.

Je n'ai pas réellement eu de soucis à réaliser ce projet mis à part le début du projet où je me suis mis une grosse pression qui m'a permis d'avancer vite mais qui m'a un petit peu déstabilisé.

Si je devais recommencer je pense que je changerai de framework css, semantic ui n'est pas le meilleur couplé à angular malgré qu'il donne un beau style au site et surtout aux formulaires.
Je pense aussi que je réfléchirai plus sur comment j'aurais pu garder des informations même en modifiant des données. Ici quand je supprime une quelconque donnée, cela à une influence sur le reste des données dont des données qui devraient rester constante, ce qui est un petit peu problématique.

Je n'avais jamais encore codé en pure javascript, mis à part quelques bouts de code en jquery dans le passé. Cela faisait juste plusieurs semaines que je m'étais renseigné partout sur internet pour comprendre les coulisses de ce langage. Grâce à ce projet, j'ai pu vraiment approfondir mes connaissances dans le développement web et les différentes architectures existantes. Ce projet a été très bénéfique pour moi. Et il va m'être très utile pour le stage que je vais faire début juin.