
WIMF
What's In My Fridge
Projet Web IG3
AFONSO Benjamin

Table des matières

Introduction

Étudiant en IG3 à Polytech Montpellier et issu de Peip Montpellier, j'ai été assigné à la conception d'une application web. N'ayant quasiment jamais fait de web auparavant, j'ai pu apprendre de nombreux concepts et notions.

Tout projet démarre d'une idée, c'est ce qu'il m'a fallu afin de pouvoir démarrer ce périple. J'ai tout d'abord pensé à réaliser une application de covoiturage entre la France et le Luxembourg (pour les frontaliers) mais j'ai vite abandonné l'idée faute d'originalité. Je me suis donc lancé dans un projet plus original et dans lequel j'ai pu m'épanouir beaucoup plus : la gestion du contenu de son frigo mais aussi de ses placards !

Je vais dérouler ce rapport comme une histoire. J'expliquerai mes choix et mes problèmes au fur et à mesure afin de leur donner une véritable dimension.

I. Analyse de l'existant

Dans un premier temps, j'ai tenté de confirmer que l'idée que je venais d'avoir était vraiment novatrice. Après quelques recherches, il s'est avéré que l'idée avait déjà été pensée. Mais heureusement pour moi, ces applications étaient vieilles, et peu jolies (d'après moi). Ces applications étaient basées sur mobile seulement et n'étaient probablement plus à jour (car basées sur d'anciennes versions iOS).

II. Architecture technique

1. Analyse

C'est donc très peu de temps après que j'entamai l'analyse. Ma première étape a été d'établir des cas d'utilisations ainsi que des frontières à mon application.

Pour qui ?

Un petit peu tout le monde, étudiants, familles, …

Pourquoi ?

J'ai découvert un besoin lorsque j'ai voulu faire des crêpes peu de temps avant. Alors que j'étais pourtant certain d'avoir de la farine chez moi, j'ai acheté tout le nécessaire hormis la farine. Et quand je suis rentré, pas de farine et donc pas de crêpes. J'en ai également parlé à des proches (amis, familles) qui m'ont confirmé que l'idée leur plaisait et que le besoin était réel.

Les différents cas d'utilisations auxquels j'ai pensé :

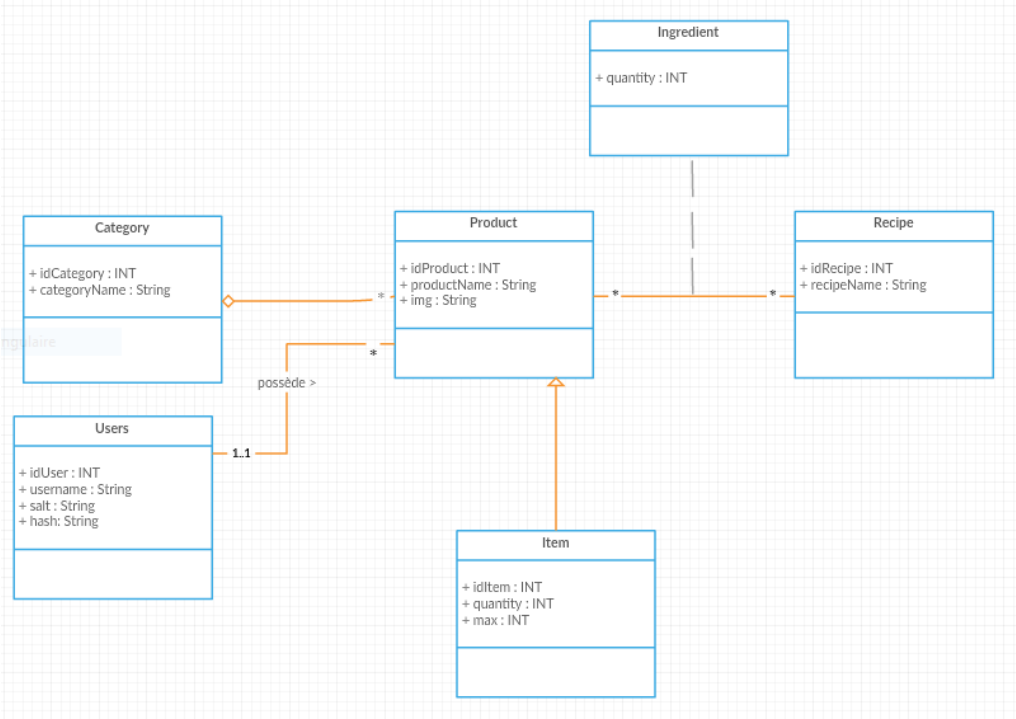
- Chaque utilisateur disposerait de son propre espace de gestion
- Pouvoir ajouter / retirer / modifier des produits
- Pouvoir associer des quantités (lots) à ces produits
- Configurer des unités à chaque lot de produit
- Rechercher des produits
- Avoir un aperçu des produits à faible stocks

- Disposer de différentes catégories (produits frais, viande, …)
- Pouvoir ajouter, supprimer et modifier les catégories à sa guise
- Pouvoir ajouter des produits à certaines catégories
- Pouvoir marquer la consommation d'un certain lot (réduire la quantité restante)
- Ajouter / Supprimer / Modifier des recettes (listes prédéfinies de certaines quantités de produits)
- Ajouter / Supprimer / Modifier des ingrédients d'une recette

c. Modélisation

Après avoir établi les différents cas d'utilisations, je voyais déjà apparaître mon diagramme UML et mes débuts de schémas de bases de données. C'est donc très tard dans la nuit que j'ai tenté de modéliser mon application web. Ce qui s'est avéré être une mauvaise idée car la fatigue a eu raison de moi et m'a poussé à faire des erreurs qui m'ont coûté cher pas la suite.

Je suis donc finalement arrivé sur ce diagramme UML :



Une catégorie est composée de produits.

Un produit est une généralisation d'item.

Un utilisateur possède des produits.

Pour un produit et une recette, on obtient un ingrédient.

Après la dérivation de ce diagramme en gardant une forte dépendance avec l'utilisateur.

Users (idUser, username, salt, hash)

Products (idProduit, #idUser, #idCategory, productName, img)

Items (idItem, #idProduct, quantity)

Recipes (idRecipe, recipeName)

Ingredient (#idRecipe, #idProduct, quantity)

Categories (idCategory, #idUser, categoryName)

J'ai par la suite voulu rajouter une fonctionnalité de journal de consommation pour chaque produit afin de permettre à l'utilisateur de suivre sa consommation d'un produit donné. Cependant je n'ai pas implémenté cette fonctionnalité à mon diagramme UML.

Une entrée journal est faite pour chaque consommation / ajout d'une certaine quantité d'un lot (Item). Elle est composée de l'identifiant du produit associé à l'item consommé, d'une date de consommation ainsi que de la quantité consommée / ajoutée.

2. Base de données

J'étais fin prêt à taper frénétiquement des lignes de codes sur mon clavier. Il ne me manquait qu'une seule chose : le choix du SGBD.

a. Choix technique

Conseillé par Mme Laurent quant à l'utilisation de *PostgreSQL*, j'ai commencé à effectuer des recherches. Documentation, différences avec les autres SGBD, inconvénients, avantages.

Et il s'est avéré que ce dernier était très bien documenté. Bien qu'un peu moins connu et utilisé qu'*Oracle* ou encore *MySQL*, j'ai remarqué que *PostgreSQL* disposait d'une communauté très active ; et donc que je serais capable de trouver solution à mes problèmes avec ce SGBD. De plus, la démarche d'installation de *postgre* sur ma *raspberry* était très simple. Et les utilitaires d'accès à la base de données étaient intéressants (psql que j'ai trouvé plus pratique que sqltool, pgAdminIII même si je ne m'en suis que très peu servi).

b. Tables et contraintes

Je posais enfin les premières briques de mon application en tapant mon premier script de création de tables.

```
CREATE TABLE Users (
  idUser SERIAL PRIMARY KEY,
  username VARCHAR(30) NOT NULL CONSTRAINT unique_username UNIQUE,
  password TEXT NOT NULL,
  salt TEXT NOT NULL,
  role role_domain NOT NULL DEFAULT 'user'
);
```

```
CREATE TABLE Products (
  idProduct SERIAL PRIMARY KEY,
  idCategory INT references Categories(idCategory) ON DELETE SET NULL,
  productName VARCHAR(50) NOT NULL,
  img TEXT,
  idUser INT references Users(idUser) ON DELETE CASCADE
);
```

```
CREATE TABLE Items (
  idItem SERIAL PRIMARY KEY,
  idProduct INT references Products(idProduct) ON DELETE CASCADE,
  idUser INT references Users(idUser) ON DELETE CASCADE,
  quantity NUMERIC NOT NULL DEFAULT 0,
  unit VARCHAR(15) NOT NULL,
  max NUMERIC,
  created_at DATE DEFAULT CURRENT_DATE
);
```

```
CREATE TABLE Recipes (
  idRecipe SERIAL PRIMARY KEY,
  recipeName VARCHAR(50),
  idUser INT references Users(idUser) ON DELETE CASCADE
);
```

```
CREATE TABLE Categories (
  idCategory SERIAL PRIMARY KEY,
  categoryName VARCHAR(50),
  idUser INT references Users(idUser) ON DELETE CASCADE
);
```

```
CREATE TABLE Ingredients (
  idRecipe INT references Recipes(idRecipe) ON DELETE CASCADE,
  idProduct INT references Products(idProduct) ON DELETE CASCADE,
  quantity NUMERIC,
  PRIMARY KEY (idRecipe, idProduct)
);
```

```
CREATE TABLE Logs (
  idProduct INT references Products(idProduct) ON DELETE CASCADE,
  quantity NUMERIC NOT NULL,
  recorded_on DATE DEFAULT CURRENT_DATE
);
```

J'ai choisi d'utiliser toutes ces contraintes '*ON DELETE CASCADE*' afin d'assurer la cohérence de la base données et ne pas laisser de traces inutiles lors de la suppression de certaines relations ; par exemple, si l'on supprime une recette, on supprime les ingrédients associés. J'utilise également une contrainte de domaine lors de la création d'un utilisateur qui n'autorise seulement les valeurs '*admin*' ou '*user'* ; la valeur par défaut étant '*user*'. Le reste des tables étant assez simple, je ne développerai pas ce point.

c. Déclencheurs (triggers)

Afin d'assurer et de maintenir la cohérence des données dans le SGBD (et aussi car c'était imposé), j'ai du ajouter des Triggers à ma base de données.

Vérification AVANT insertion d'un ingrédient

```
-- Verifier si le produit et la recette concernent le même utilisateur
-- lors d'une insertion dans Ingredients
CREATE OR REPLACE FUNCTION proc_check_user_on_insert() RETURNS TRIGGER AS $check_user_on_insert$
  DECLARE
    recipe_user_id INTEGER;
    product_user_id INTEGER;
  BEGIN
    recipe_user_id = (SELECT DISTINCT idUser FROM Recipes WHERE idRecipe = NEW.idrecipe);
    product_user_id = (SELECT DISTINCT idUser FROM Products WHERE idProduct = NEW.idProduct);
    IF recipe_user_id != product_user_id THEN
      RAISE EXCEPTION 'the user and the recipe dont have the same owner';
      RETURN NULL;
    END IF;
    RETURN NEW;
  END;
$check_user_on_insert$ LANGUAGE plpgsql;

CREATE TRIGGER check_user_on_insert BEFORE INSERT
ON Ingredients
FOR EACH ROW
EXECUTE PROCEDURE proc_check_user_on_insert();
```

Ce trigger vérifie avant l'insertion d'un ingrédient (afin de maintenir la cohérence de la BD) si le produit et la recette de l'ingrédient concernent le même utilisateur. Si c'est le cas, la requête est libérée. Sinon, elle est annulée et une erreur est renvoyée.

Insertion de la consommation d'un produit dans un journal

```
-- Enregistrer dans un journal chaque consommation d'un produit
-- afin de permettre par la suite des statistiques et un suivi
-- de la consommation pour chaque produit
CREATE OR REPLACE FUNCTION proc_write_consumption_logs() RETURNS TRIGGER AS $write_consumption_logs$
  DECLARE
    delta_stock NUMERIC;
  BEGIN
    --
    -- Write each product consumption on the Log table
    --
    IF (TG_OP = 'DELETE') THEN
      INSERT INTO Logs (idProduct, quantity) VALUES (OLD.idProduct, -OLD.quantity);
    ELSIF (TG_OP = 'UPDATE') THEN
      delta_stock = (NEW.quantity - OLD.quantity);
      -- If the stock is the same, do nothing
      IF (delta_stock = 0) THEN RETURN NULL; END IF;
      INSERT INTO Logs (idProduct, quantity) VALUES (NEW.idProduct, delta_stock);
    ELSIF (TG_OP = 'INSERT') THEN
      INSERT INTO Logs (idProduct, quantity) VALUES (NEW.idProduct, NEW.quantity);
    END IF;
    RETURN NULL;
  END;
$write_consumption_logs$ LANGUAGE plpgsql;
```

Celui-ci insert de façon automatique la quantité consommée ou ajoutée d'un produit après chaque Insertion, Suppression ou Modification d'un item. La date est la date du jour par défaut (Contraintes).

Suppression des entrées journal d'un produit ayant été supprimé

```
-- Supprime les entrées journal d'un produit lors de la suppression de ce dernier
CREATE OR REPLACE FUNCTION proc_remove_log_entries() RETURNS TRIGGER AS $remove_log_entries$
  BEGIN
    DELETE FROM Logs WHERE idProduct = OLD.idProduct;
    RETURN NULL;
  END;
$remove_log_entries$ LANGUAGE plpgsql;

CREATE TRIGGER remove_log_entries AFTER DELETE
ON Products
FOR EACH ROW
EXECUTE PROCEDURE proc_remove_log_entries();
```

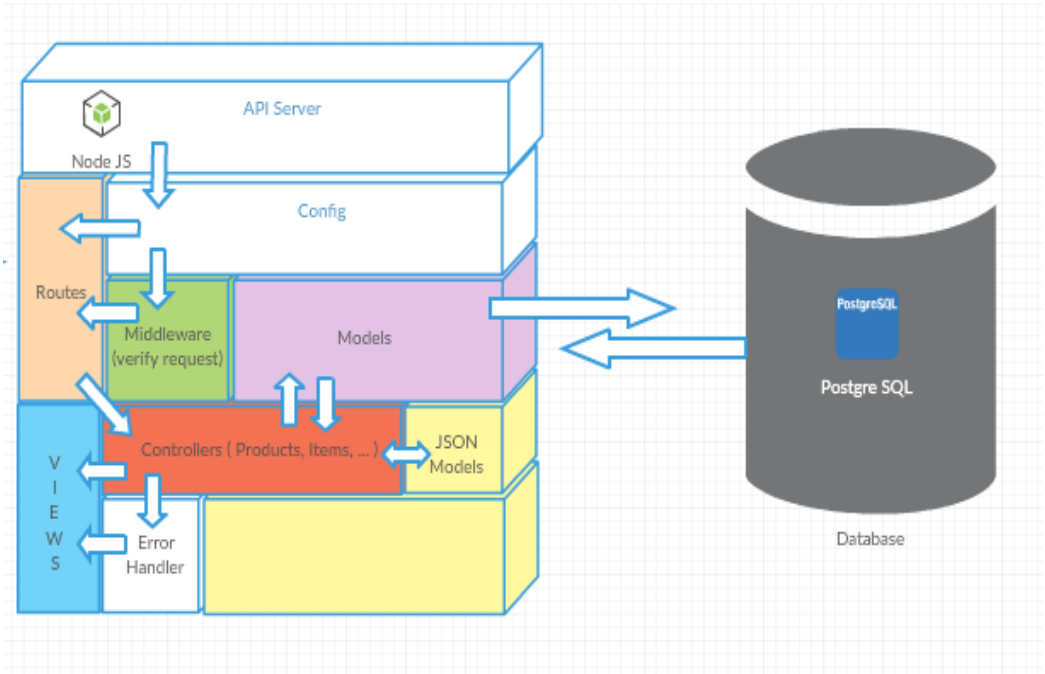
Ce trigger a pour but de supprimer les entrées journal de consommation d'un produit ayant été supprimé. Il m'aurait été également possible d'obtenir le même résultat avec une contrainte *ON DELETE CASCADE*.

3. REST API (Back-end)

a. Choix techniques

J'ai choisi de faire une application RESTful car après de nombreuses recherches, j'ai été séduit par la façon de distribuer les ressources. J'aimais vraiment l'idée d'avoir un contrôle de plus à la sortie du SGBD, et c'était un excellent moyen d'utiliser convenablement le protocole HTTP.

J'avais également beaucoup entendu parler de NodeJS, de ce qu'il était possible de réaliser avec, ainsi que l'aspect 'bas niveau' qui permettait de gérer les choses au plus bas. Après de longues recherches je me suis rendu compte que ce côté n'était pas seulement bénéfique et j'ai donc utilisé le framework 'Express' afin d'abstraire légèrement les concepts de NodeJS.



Architecture technique serveur

b. Gestion des ressources

J'ai organisé l'accès aux ressources de la façon que j'ai jugée la plus 'intuitive', 'naturelle' tout en respectant les verbes HTTP et en proposant des URIs simples.

Méthode	URI	Description
POST	/login	Connexion d'un utilisateur
POST	/register	Création d'un utilisateur
GET	/api/v2/products	Récupérer tous les produits
POST	/api/v2/products	Créer un produit
GET	/api/v2/products/:id	Récupérer un produit
PUT	/api/v2/products/:id	Modifier un produit
DELETE	/api/v2/products/:id	Supprimer un produit
GET	/api/v2/categories	Récupérer toutes les catégories
POST	/api/v2/categories	Ajouter une catégorie
PUT	/api/v2/categories/:id	Modifier une catégories
DELETE	/api/v2/categories/:id	Supprimer une catégorie
GET	/api/v2/products/:product_id/items	Récupérer les items d'un produit
POST	/api/v2/products/:product_id/items	Ajouter un item à un produit
GET	/api/v2/products/:product_id/items/:id	Récupérer un item
PUT	/api/v2/products/:product_id/items/:id	Modifier un item
DELETE	/api/v2/products/:product_id/items/:id	Supprimer un item
GET	/api/v2/recipes	Récupérer toutes les recettes
POST	/api/v2/recipes	Ajouter une recette
GET	/api/v2/recipes/:id	Récupérer une recette
PUT	/api/v2/recipes/:id	Modifier une recette
DELETE	/api/v2/recipes/:id	Supprimer une recette
GET	/api/v2/recipes/:recipe_id/ingredients	Récupérer les ingrédients
POST	/api/v2/recipes/:recipe_id/ingredients	Ajouter un ingrédient à une recette
PUT	/api/v2/recipes/:recipe_id/ingredients/:product_id	Modifier un ingrédient
DELETE	/api/v2/recipes/:recipe_id/ingredients/:product_id	Supprimer un ingrédient
GET	/api/v2/users/me	Récupérer ses informations (profil)
GET	/api/v2/admin/users	Récupérer les liste des utilisateurs
PUT	/api/v2/admin/users/:id	Modifier un utilisateur

Routes accessibles par n'importe qui
Routes accessible seulement aux utilisateurs connectés
Routes uniquement accessibles par les administrateurs

Afin de parvenir à distribuer les ressources, j'ai utilisé le module 'router' du framework Express. Ce module m'a permis de définir très simplement les règles d'accès (verbes, etc..) pour l'accès de chaque ressource.

Il m'a fallu ensuite traiter la façon de gérer les images. Je me suis tout d'abord renseigné sur les services de stockage dans le cloud (*AmazonWebServices*, …), mais ces derniers étaient, pour la plupart, payants. J'ai donc décidé de laisser l'utilisateur chercher lui même l'URL des images qu'il désirait stocker. Ainsi la base de données ne stockait que les URLs. Cependant j'ai trouvé que l'utilisateur devenait en quelque sorte dépendant de trop de sites, de par le nombre de sources d'images qu'il utiliserait. J'ai alors utilisé une API (*reSmushIt*) afin de compresser les images et les stocker sur leur serveur. Le service étant gratuit, j'ai mis en place ce système un peu plus tard.

Cependant, j'ai par la suite découvert que la durée de stockage des images sur ce serveur était instable dans le temps (temps de stockage très limité ~40 minutes). J'ai alors abandonné cette idée.

c. Authentification et sécurité

En effet la plupart des ressources requièrent une autorisation. Je vais expliquer ici comment j'ai procédé afin de sécuriser ces dernières.

L'authentification par token : j'ai mis en place ce type d'authentification car, n'ayant jamais réalisé de système d'authentification, j'avais un point de vue vraiment objectif et j'ai donc pu peser le pour et le contre de chacun (tokens et cookies).

Les tokens sont Cross-domain tandis que les cookies le sont moins. En utilisant les tokens, on est capable d'effectuer des requêtes AJAX à n'importe quel serveur car on utilise le HTTP Header pour transmettre l'information de l'utilisateur. Le token est *stateless* et on est moins dépendant du stockage client et donc plus adapté à un support mobile.

Finalement, j'ai choisi le token car c'est une technologie plus adaptée à la consommation d'une API et de plus en plus utilisée. C'est pourquoi j'ai jugé plus intéressant d'apprendre cette méthode plutôt que les cookies.

Comment sont générés les tokens ?

Les tokens sont générés à chaque connexion d'un utilisateur. J'ai utilisé le module proposé par JsonWebTokens afin de générer ces derniers. Le fonctionnement de JsonWebToken est assez simple ; en fonction d'une phrase secrète, un objet JSON est encrypté, constitué d'un *header* (type d'encryption, …), d'un *payload* (informations d'un utilisateur sous forme d'objet JSON) et d'une signature.


```
eyJhbGciOiJIUzI1NiIsInR5cC
I6IkpXVCJ9.eyJ1c2VyX2lkIjo
iMTIiLCJ1c2VybmFtZSI6IkJlb
mphbWluIEFmb25zbyIsImFkbWl
uIjp0cnVlfQ.f1y2Ui0NTt7tE_
Z76pauyFqzZKQBJwnKGt6yiQeb
BK8
```


Illustration 1: Token encodé

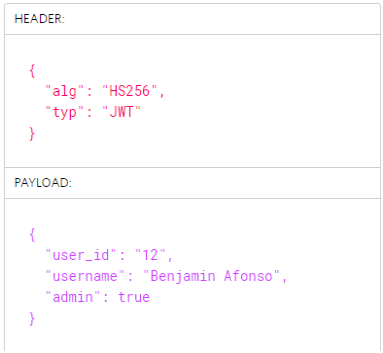


Illustration 2: Token décodé

Comment sont gérées les données sécurisées ?

J'ai utilisé le module 'crypto' afin de gérer l'encryptage des mots de passe. Il était très complet et permettait un nombre conséquent de possibilités d'encryptage.

J'ai choisi un encryptage asymétrique. Cependant je me suis rendu compte que le cryptage n'est pas totalement asymétrique car le message est reçu non crypté par le serveur tandis que pour un cryptage de ce type, le message devrait être envoyé chiffré puis déchiffré à l'aide d'une clé unique, le '*salt*', de l'utilisateur. Si c'était à refaire je procéderais un peu différemment.

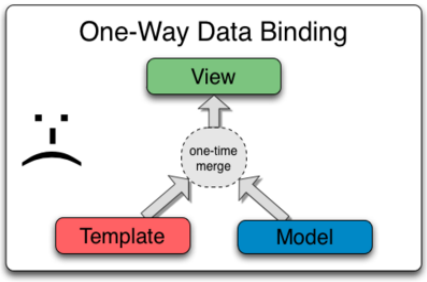
Actuellement lors de chaque création d'utilisateur, un '*salt*' constitué de 128bits est généré aléatoirement. Le mot de passe est donc encrypté à partir du '*salt*' précédemment généré (12000 itérations ~300ms) afin de retourner un '*hash*'. Le couple '*hash*' / '*salt*' est stocké dans la base de données. A chaque connexion d'un utilisateur, on va donc chercher le '*salt*' associé à cet utilisateur, on procède au cryptage du mot de passe fourni et on compare le '*hash*' obtenu au '*hash*' stocké en base de données.

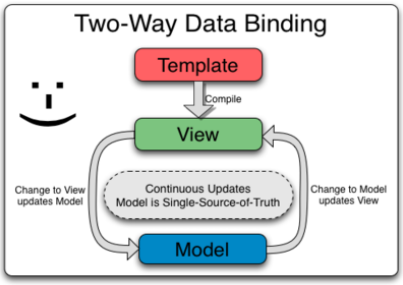
J'ai rencontré un autre problème qui est tout simplement que les informations transitent en clair entre le client et le serveur. L'utilisation du protocole https permettrait de palier à ce problème.

4. Angular Client (Front-end)

a. Choix technique

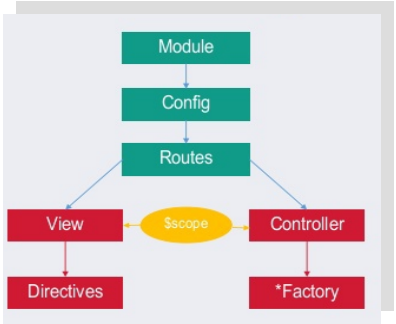
J'ai choisi Angular car c'est une technologie que je voulais vraiment apprendre une fois de plus. Ce framework étant basé sur du Javascript dont j'ai pu comprendre les subtilités avec NodeJS, je n'ai pas eu énormément de mal à le prendre en main. J'ai également été séduit par le Two-way-data-binding introduit par AngularJS qui permet de rendre l'affichage des ressources plus asynchrone et dynamique au sein de l'application.



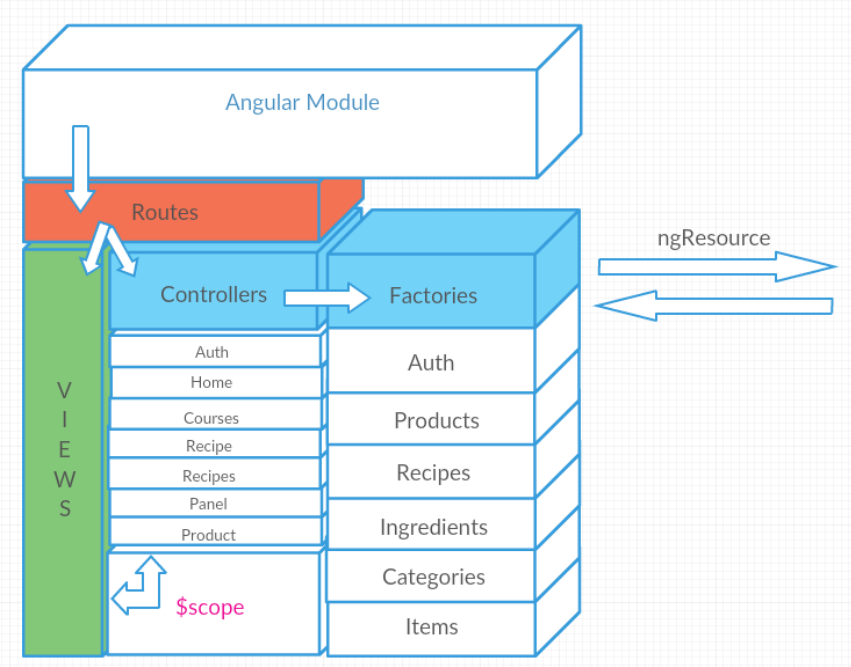


Un autre point assez important dans ce choix était la documentation disponible qui été très conséquente. En effet, Google a réussi à mettre Angular sur le devant de la scène ces dernières années, notamment en promouvant un tutoriel CodeSchool qui m'a été d'une grande aide (même si je l'avais réalisé il y longtemps par pur plaisir).

Et enfin j'ai trouvé qu'Angular permettait d'apprendre très simplement les bonnes pratiques, ainsi que la façon pour bien structurer son application. J'ai trouvé le modèle proposé par Angular très intuitif et bien pensé. Ce dernier est appelé MV* ou MVW (Model View Whatever) dont la structure est la suivante :



b. Structure client



Architecture technique client

c. Accès aux ressources

J'avais tout d'abord utilisé le module $http d'Angular qui me permettait d'effectuer mes requêtes HTTP au sein de mes 'Factories'. Cependant, le code était lourd et redondent, je devais définir chaque fonction entièrement et ce pour chaque type de ressource. Ce changement de choix m'a coûté une quantité non négligeable de temps de modification. J'ai finalement opté pour le module ngResource qui permet de disposer des verbes de bases sur une URI de ressources. Ce qui m'a permis d'alléger très largement mon code.

Mais ngResource m'a également causé un gros problème que je vais expliciter ici :

```
{
  "status": 200,
  "message": "Retrieved products successfully",
  "products": [
    {
```

API Version 1

```
[
  {
    "idProduct": 40,
```

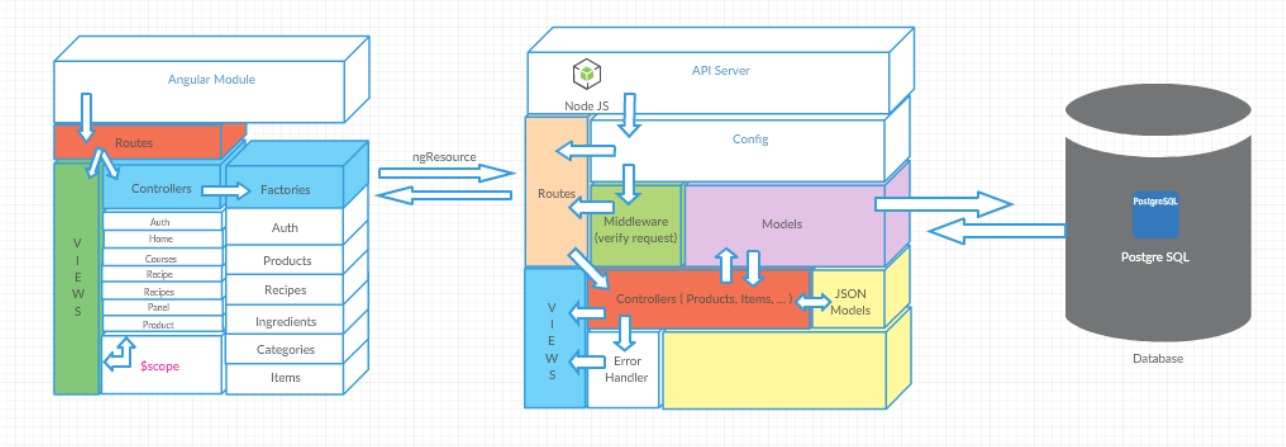
API Version 2

En effet, j'ai du modifier mon API car ngResource gérait très mal le fait que je ne renvoie pas uniquement les ressources demandées mais également quelques informations complémentaires. C'est pourquoi j'ai eu l'impression de perdre le contrôle quant à l'utilisation de ngResource. J'ai donc privilégié le fait que l'ensemble soit géré plus proprement et opté pour la modification de mon API. Choix qui m'a été confirmé suite à une question que j'ai posé sur StackOverflow.

Cette décision ne m'a pas été handicapante du tout car les codes de retour sont de toute façon présents dans les requêtes HTTP. Et les messages sont toujours envoyés en cas d'erreurs.

Donc j'ai jugé bon dans l'ensemble d'utiliser ngResource.

Architecture générale de l'application



d. Style

Étant déjà quelque peu familier avec les frameworks CSS, j'ai décidé de faire du CSS *vanilla*, car je connaissais seulement les notions de bases et je voulais en apprendre plus. Je trouve également que l'utilisation d'un framework CSS nous limite à ce qui nous est proposé et rend encore plus difficile les modifications du code existant.

J'ai donc pu apprendre et ressentir le calvaire que cela représentait pour un novice. J'ai, je pense, réussi à y faire face, mais j'ai du imposer quelques limites à ma créativité, et ce à cause de mes compétences restreintes.

J'ai utilisé Sass, et je suis bel et bien conscient que je ne me suis pas servi de cette 'extension' de langage à son plein potentiel mais plus pour le découvrir pas à pas. J'ai pu notamment me familiariser avec les variables qui permettent un grand gain de temps pour modifier rapidement le code couleur de l'application. Mais également avec les héritages de classes css et les *mixins* dont je me suis servis lors de mon utilisation de Susy Grid.

Susy est une base de grilles créée sur la base de Sass permettant de concevoir ses propres grilles et de les personnaliser. Les grilles sont élaborées sur demande et calculées directement dans le css. Cela permet d'alléger le code html et de rendre au CSS son véritable intérêt contrairement à Bootstrap qui, selon moi, complique et alourdit le code html.

J'ai également utilisé 'Breakpoints' (lui aussi construit avec Sass) qui m'a permis d'effectuer des *media-queries* de manière plus intuitive et de rendre mon site plus *responsive*.

Et pour finir, j'ai eu recours à 'Material Design Lite' de Google afin de donner une belle allure à mes composants. Cependant ces derniers n'étaient pas simples à modifier car ils fixaient pour un même composant, plusieurs tailles, en pixel par exemple. Mais Material Design Lite m'en a fait voir de toutes les couleurs. En effet, les composants étant liés au DOM, lors de l'utilisation des routes Angular, ils ne se rechargeaient pas avec les *templates*. Mes composants MDL n'étaient donc visibles que sur la première page visitée ou lors d'un rafraîchissement complet de la page. C'est pourquoi j'ai du trouver un script pour remédier à ce problème.

Lors de la réalisation du design de mon application, j'ai rencontré de nombreux problèmes, c'est pour cela que je qualifie cette partie de « périple » et que je la considère comme la plus fastidieuse du projet. Redimensionner, adapter, structurer mes conteneurs. J'ai causé ces problèmes en concevant mal mes conteneurs, décidant d'ajouter tel ou tel contenu, … Bien qu'ayant une idée globale de ce à quoi mon application devait ressembler, j'ai eu du mal à transposer mes idées et donc à les retranscrire.

Si c'était à refaire, je passerais beaucoup plus de temps à dessiner mes pages et à identifier les différents conteneurs dont j'aurais besoin. Et je n'utiliserais plus Material Design Lite mais plutôt Ionic qui semble, selon moi, être bien plus adapté à une utilisation mobile et qui a été créée dans le but d'être utilisé avec AngularJS.

III. Déploiement

J'ai opté pour deux environnements de déploiement : la liberté et la sécurité.

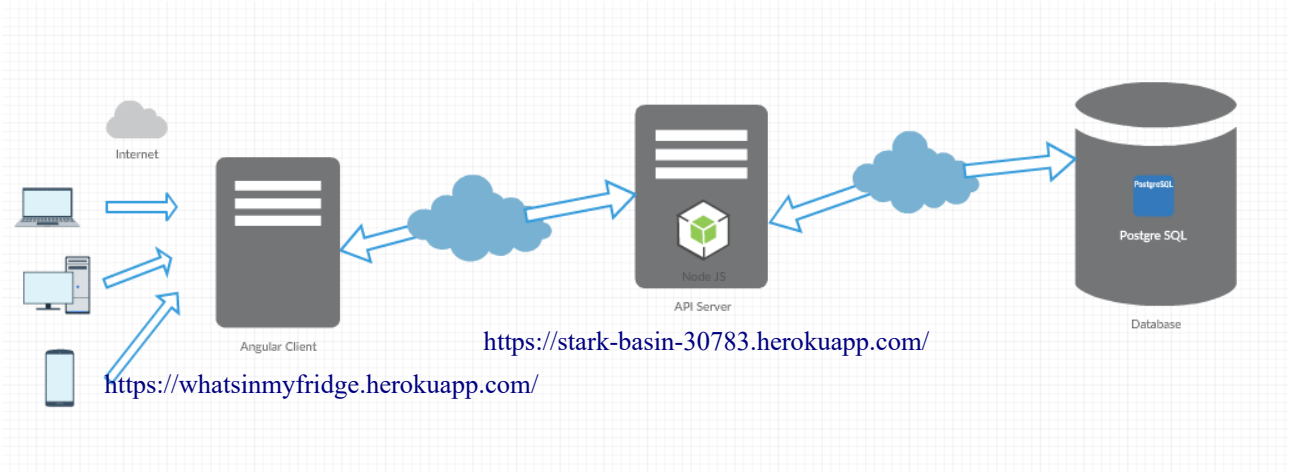
Le premier, la liberté, est situé chez moi et connecté à ma box internet :une Raspberry Pi 2B, assez performante pour faire tourner le tout et assumer un faible trafic.

Le second, la sécurité, est situé à trois endroits différents en Europe, trois serveurs d'applications fournis par Heroku.

J'ai choisi le second seulement afin d'assurer la disponibilité de mon application à l'heure fatidique car ce dernier ne permet la disponibilité de l'application seulement 18 heures par 24 heures. Le serveur ne se mettant en route seulement lors de trafic, cette offre est déjà plus qu'acceptable pour une offre gratuite.

A ma grande satisfaction, Heroku ne fournit seulement des bases de données PostgreSQL et l'offre autorisait 20 connexions simultanées ainsi que 10,000 lignes dans le SGBD. Cette offre étant une fois de plus très avantageuse, j'ai également choisi Heroku pour le déploiement de la base de données.

Afin d'optimiser les temps d'accès aux ressources, j'ai automatisé certaines tâches lors du développement qui m'ont ensuite facilité la mise en production. Notamment la minification et concaténation de tous mes fichiers javascript et css. De plus, j'ai aussi utilisé bower afin de ne prendre seulement les morceaux que j'ai utilisés dans les frameworks utilisés (Angular, Material Design, …) et ainsi d'éviter à l'utilisateur de charger des frameworks entiers bien que minifiés.



Ci-dessous un schéma de mon architecture de déploiement :

Environnement de déploiement n°1 :

Serveur API	https://stark-basin-30783.herokuapp.com
Serveur client	https://whatsinmyfridge.herokuapp.com
Base de données	ec2-54-228-219-2.eu-west-1.compute.amazonaws.com:5432/d7esbe6ba62akt

Environnement de déploiement n°2 :

Serveur API	http://benjaminafonso.ml:3000
Serveur client	http://benjaminafonso.ml:8080
Base de données	31.32.127.70:5432/whatsinmyfridge

Conclusion

Durant ce projet, j'ai eu l'occasion d'apprendre de très nombreuses notions et ce même avant. Car j'ai commencé à m'y intéresser dès que j'ai su qu'on aurait un projet à effectuer, par plaisir. J'ai pu apprendre des technologies qui m'intéressaient et m'exercer sur une application originale que j'ai montée de toutes pièces. J'ai tout d'abord appris AngularJS et ses concepts, puis NodeJS et son côté asynchrone, Sass et sa beauté d'écriture et enfin Grunt pour l'automatisation. J'ai aussi appris comment lier le tout et transformer mes connaissances en ressources. Ce projet m'a été très bénéfique et j'ai eu la chance de ne pas m'imposer une pression trop importante, car je m'y suis préparé à l'avance. Et donc cela m'a permis d'entamer le projet avec plus de sérénité.

Si c'était à refaire, je le referais sans hésiter. Mais mieux, en apprenant de mes erreurs. J'essaierais de reprendre toutes les choses négatives que j'ai rencontrées et aussi de faire en sortes qu'elles le soient moins. J'essaierais d'optimiser les choses qui ne m'ont pas posé de problèmes.

Durant ce projet, j'ai également pu aider d'autres étudiants qui avaient choisis les mêmes technologies que moi, ce qui m'a permis de valider des concepts que j'avais appris en les explicitant à d'autres. Mon acuité à détecter les erreurs et les corriger est également sortie grandie de cette expérience.

Pour conclure, ce projet m'a permis d'accomplir un véritable challenge. Il m'a également permis d'assimiler de précieuses connaissances dans des domaines qui m'intéressaient déjà auparavant.