

Rapport

PRO3600 – Développement informatique

Encadrant : Daniel RANC

Groupe-projet : PRO3600-20-RAN-37
Emilien VIMONT Armand ROBERGE
Théophane DENEUVE et Nordine MARIE



Projet :



DietApp

Une application multiplateforme (iOS, android et autres) conçue en HTML, CSS, JavaScript associée au framework AngularJS. DietApp a pour vocation de remplacer les conseillers diététiques grâce à des algorithmes de comparaison et d'optimisation de produits, l'accent est mis sur une interface soignée accessible à tous.



Table des matières :

I. Pré-rapport	3
1. Cahier des charges	3
2. Spécification fonctionnelle	6
3. Description du flux des données inter-module	10
II. Conception détaillé	11
4. Dépendances	11
5. Les principaux objets manipulés	12
6. Fonctionnalités détaillées	15
7. Cordova – Application builder	31
III. Problèmes rencontrées et Anecdotes	32
IV. Tests	36
V. Déploiement	36
VI. Bilan	37
VII. Manuel utilisateur	39
VIII. Bibliographie	46

I. Pré-rapport

1. Cahier des charges

a. Présentation du projet

Au début de notre projet nous avons fait le constat de la diversité des produits alimentaires présents sur le marché. Pour un même type de produit, les caractéristiques : le poids, les ingrédients, les apports nutritionnelles et même le nutri-score, varient fortement. Cela oblige d'une part les supermarchés à garder à jour ces données mais cela oblige surtout les clients à redoubler d'attention quant à la qualité des produits.

Dans ce sens le but du projet « DietApp » est de développer une application multiplateforme pouvant aider les consommateurs à gérer leurs achats et leur liste de course, les conseiller dans le choix des produits selon leur profil. Ce profil contiendra des informations sur l'utilisateur comme son âge, son genre, ses goûts ainsi qu'un champ nous renseignant sur ses « ambitions diététiques » (Sportif / Régime minceur / Prise de masse). Ces informations stockées uniquement localement seront utiles à l'application pour calculer les besoins journaliers, les potentiels conseils et l'équilibrage des repas de chaque utilisateur.

L'affichage de l'application sera dirigé vers une utilisation des clients. Cette application devra contenir plusieurs fonctionnalités définies rapidement ici et approfondies plus tard : la gestion d'une liste de courses, la gestion d'un profil utilisateur, la suggestion de produits selon les intérêts et le profil de l'utilisateur, la suggestion de produits alternatifs ou un service de renseignement sur les caractéristiques des articles pouvant être obtenus par scan de code-barres.

b. Objectifs de l'application

La vocation de « DietApp » est de fournir une application multiplateforme s'adaptant à l'utilisateur qui serait en mesure de remplacer un conseiller diététique notamment grâce à :

- un algorithme de comparaison proposant une sélection de produits alternatifs, triés du meilleur au pire pour la santé, pour chaque produit sélectionné par l'utilisateur.
- un algorithme de conseil de produits qui a pour but d'optimiser le panier courant vers le profil recommandé pour l'utilisateur, tout

en se basant sur les produits déjà achetés, regardés ou mis en favori.

L'objectif de « DietApp » est alors clair : aider ses utilisateurs à atteindre leurs ambitions en matière de diététique.

c. Cibles de l'application

L'application cible principalement les particuliers d'âge supérieur à 16 ans qui souhaite au moins une des assertions suivantes :

- Maîtriser leur consommation
- Associer leur entraînement sportif à un régime prévu à cet effet
- Mincir/grossir

d. Langages de développement

- HTML pour la représentation de l'UI (User Interface)
- CSS pour la présentation de l'UI
- JavaScript et le framework AngularJS pour la logique de l'application

Nous avons choisi ces langages à défaut du Java pour deux raisons principales :

- Langages multiplateformes : à défaut de Java et d'Android Studio ,qui, comme son nom l'indique, permet uniquement de coder des applications android. Les langages choisis permettent d'être ensuite compiler en .apk pour Android et .ipa pour iOS grâce au script Cordova, qui injecte un navigateur basé respectivement sur Chrome et Safari.
- Pérennité des langages : En effet, il paraît invraisemblable que ces technologies web s'effondrent du jour au lendemain. A contrario, de Android Studio, qui est maintenu tant que android le sera. Cela est encore plus vrai lorsque l'on sait que Google travaille actuellement sur un nouvel OS (non-android) : « Fuchsia ».

e. Charte graphique

Logo, palette de couleurs et police :

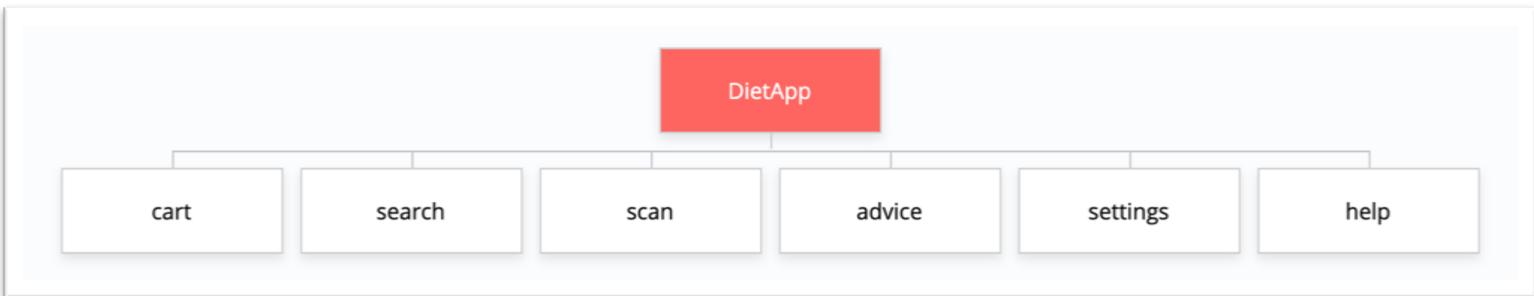


Frameworks CSS :

- **W3.css** : pour les éléments HTML
 - **fontawesome** : pour les icônes
 - **loading.css** : pour les barres de chargement
-

2. Spécification fonctionnelle

Remarque importante : Tous les termes techniques en italiques sont définis plus en profondeur dans la partie « Conception détaillée ».



a. Description générale des rubriques

- **cart :**

Contenu du panier utilisateur sous forme d'une liste d'objet *product* qui contient son nom *product_name* mais aussi d'autres caractéristiques.

- **search :**

Recherche de produits par mots-clés dans la base de données d'OpenFoodFacts par saisie.

- **scan :**

Recherche de produits par code-barres dans la base de données d'OpenFoodFacts par saisie ou par scan grâce à QuaggaJS.

- **advice :**

- Affichage des statistiques nutritifs en accord avec le profil
- Comparaison des proportions en macronutriments du panier courant avec celles du profil choisi
- Affichage de la progression calorique journalière
- Conseil de produit afin de faire tendre les statistiques du panier courant vers celles du profil recommandé.

- **settings :**
 - Configuration du profil utilisateur
 - Gestion des cartes de fidélité
 - Gestion des favoris

- **help :**

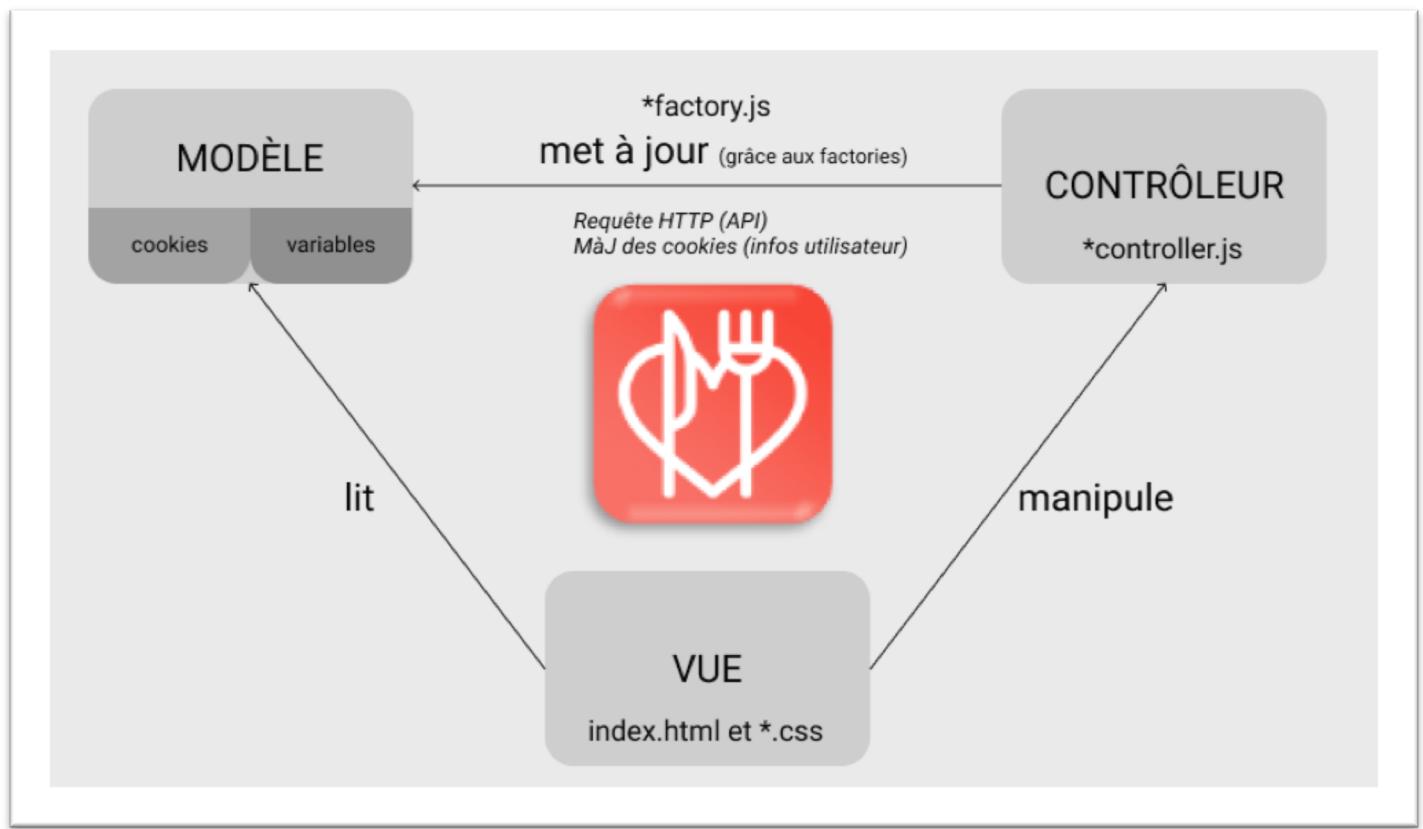
Accès au support et au contact

b. Description détaillée des fonctionnalités

Rubrique	Fonctionnalité	Description
cart	Affichage du panier	Affichage du panier sous forme d'une liste ordonnée par ordre d'ajout
	In.Dé.crémentation	Modification de l'attribut <i>quantity</i> d'un <i>product</i> du panier
	Suppression	Supprime un <i>product</i> du panier
	Bilan carbone	Affichage du bilan carbone total du panier
search	Recherche par mots-clés	Recherche par mots-clés de produit dans la base de données d'OpenFoodFacts
	Carte détaillée d'un produit	Par clique sur un produit dans les résultats de recherche, apparaît une carte détaillée du produit cliqué
	Ajout au panier	Bouton qui permet d'ajouter le <i>product</i> sélectionné au panier
	Ajout au favori	Bouton qui permet d'ajouter le <i>product</i> sélectionné aux favoris
	Produits alternatifs	Dans la carte détaillée d'un <i>product</i> , un carrousel met en avant des produits alternatifs semblables mais de meilleure composition
scan	Recherche par code-barres	Recherche par code-barres de produit dans la base de données d'OpenFoodFacts par saisie clavier ou par scan du code-barre avec la caméra de l'appareil

advice	Configuration du profil utilisateur (1 ^{ère} utilisation)	Fenêtre de configuration du profil : Prénom, genre, âge, ambition diététique
	Comparaison des statistiques	Affichage comparatif des proportions en lipides, protéines et glucides du panier courant et du profil recommandé
	Progression calorique journalière	Affichage de la progression calorique journalière en % en fonction de la configuration du profil utilisateur
	Conseil d'articles	Conseil d'articles supplémentaires visant à faire tendre les statistiques du panier courant vers celles du profil recommandé ; conseils basés sur les produits fréquemment achetés/vus ou mis en favoris
settings	Modification du profil utilisateur	Modification du profil : Prénom, genre, âge, ambition diététique
	Gestion cartes de fidélités	Ajout/suppression de carte de fidélité par saisie du code-barres et d'un nom la désignant
help	Signaler un bug	Formulaire de signalement de bug : adresse-mail, description du bug et moyen de le reproduire
	Contact	Paragraphe de contact avec les adresses-mail et lien vers page LinkedIn des créateurs.

3. Description du flux des données inter-module



II. Conception détaillée

1. Dépendances :

Les diverses dépendances (JS, AngularJS) sont stockées dans le dossier jslibs. En cliquant sur leurs noms ci-dessous vous serez redirigé vers leurs documentations respectives.

a. AngularJS :

- **angular-cookies** : gestionnaire de cookies navigateur
- **angular-chart** : affichage de graphique
- **angular-ui-router** : gestionnaires de routes

b. Plain Javascript :

- **quagga** : lecteur de codes-barres
-

2. Les principaux objets manipulés

Des exemples de structure des objets principalement manipulés vont être donnés, les attributs explicites (e.g `product_name`) seront laissés au lecteur.

b. objet : *product/offproduct*

Tout d'abord nous allons lever une ambiguïté, dans notre application en réalité on représente un produit sous 2 types d'objets : *product* et *offproduct*. Ce dernier correspond à l'objet produit tel que renvoyé par l'API OpenFoodFacts. Vous vous posez certainement la question suivante : pourquoi redéfinir un autre objet produit que celui renvoyé par l'API ? En réalité, il y a deux réponses à cela :

Premièrement, un *offproduct* est beaucoup trop lourd en mémoire et ne peut donc pas être stocké dans un cookie, qui rappelons-le a une taille limitée à 4 ko, il fallait définir un nouvel objet *product* qui serait une version « light » de *offproduct* comportant uniquement les attributs d'un produit qui nous sont utiles. De plus, *product* comporte l'attribut `code` permettant si on le souhaite d'obtenir son équivalent *offproduct* par requête HTTP.

Deuxièmement, OpenFoodFact étant ouvert et libre il n'est pas rare de trouver des produits avec des attributs manquants ou vides. Ainsi, définir un nouvel objet produit nous permettait d'avoir un objet produit avec les attributs principaux jamais manquants et quasiment jamais vides.

Exemple de structure de l'objet *product* :

```
{
  product_name: "Nutella"
  brands: ["ferrero", "nutella"]
  categories: "Petit-déjeuners,Produits à tartiner,Produits à tartiner sucrés,Pâtes à tartiner,Pâtes à tartiner aux noisettes,Pâtes à tartiner au chocolat,Pâtes à tartiner aux noisettes et au cacao"
  co2: "33.8"
  code: "3017620422003"
  energy: 2255
  generic_name: "Pâte à tartiner aux noisettes et au cacao"
  image_front_thumb_url:
  "https://static.openfoodfacts.org/images/products/301/762/042/2003/front_fr.168.100.jpg"
  nutriscore_score: 26
  nutriscore_tag: "e"
  origins: []
  plg_list: [6.3, 61.8, 113.8]
  quantity: 1
}
```

co2 (string) : empreinte carbonique en gCO₂/100g**code (string)** : code barre**energy (int)** : énergie en kCal**image_front_thumb_url (string)** : URL de la miniature du produit**plg_list (Array[float])** : proportion en Protides/Lipides/Glucides**quantity (int)** : attribut utile seulement au panierExemple de structure de l'objet *offproduct* :<https://wiki.openfoodfacts.org/API/Read/Product>c. objet : *user*Exemple de structure de l'objet *user* :

```
{
  profile: "healthy",
  gender: 0,
  age: 22,
  surname: "Nordine"
}
```

gender (int) : 0 pour homme, 1 pour femme. Pourquoi pas l'inverse ? C'est simple : ayant peur des mouvements ultra-féministes je me suis assuré que le test 0 (*homme*) < 1 (*femme*) soit *true*.

profile (string) : définit le profil/ambition diététique (cf. prochain objet)

d. objet : *profiles*

Exemple de structure de l'objet *profiles* :

```
{
  loseweight : {proportion : [0.15,0.4,0.45],
    // entre 16 et 40 ans
    adult : {energyHF: [2350,1900]},
    // plus de 40 ans
    old : {energyHF: [2200,1750]}
  },
  healthy : {proportion : [0.15,0.375,0.475],
    // entre 16 et 40 ans
    adult : {energyHF: [2650,2150]},
    // plus de 40 ans
    old : {energyHF: [2450,2000]}
  },
  gainweight : {proportion : [0.2,0.3,0.5],
    // entre 16 et 40 ans
    adult : {energyHF: [3250,2500]},
    // plus de 40 ans
    old : {energyHF: [3050,2350]}
  }
}
```

3 profil diététiques différents :

loseweight = Minceur

healthy = Santé

gainweight = Prise de poids

avec leur *proportion* en Protides/Lipides/Glucides correspondante et leur besoin calorique en fonction de la tranche d'âge (*adult* ou *old*) et du genre (*energyHF[0]* correspondant à celui de l'homme).

3. Fonctionnalités détaillées

a. Message de bienvenue :

welcomer est la fonction qui permet d'afficher un message de bienvenue :

```
welcomer = function() {
    var cookie = $cookies.getObject("welcomeCookie") ;
    var text = document.getElementById('welcome_div')
    var now = new Date() ;
    if (typeof cookie == "undefined") { // New user
        var welcookie = {lastconnexion: now.getDay()} ;
        $cookies.putObject("welcomeCookie", welcookie) ;
        console.log("Welcome new user") ;
        text.innerHTML = "<h4 style='text-align: center;'>Bienvenue sur
<strong>DietApp</strong>,<br> ton nouveau diététicien personnel !</h4>" ;
        displayModal('welcome') ; // 'welcome' est l'id du modal à afficher
        setTimeout(function() {displayModal('welcome')}) //callback
            ,2500) ;// temps d'attente en ms
    }
    else if (cookie.lastconnexion != now.getDay()) { // 1st daily connection
        var welcookie = {lastconnexion: now.getDay()} ;
        $cookies.putObject("welcomeCookie", welcookie) ;
        var user = $cookies.getObject("userCookie") ;
        console.log("Welcome " + $scope.surname + " !") ;
        text.innerHTML = "<h4 style='text-align: center;'>Bienvenue sur
DietApp <strong>" + user.surname + "</strong>,<br> nous sommes ravis de vous
retrouver parmi nous !</p>" ;
        displayModal('welcome') ;
        setTimeout(function() {displayModal('welcome')}) //callback
            ,2500) ;// temps d'attente en ms
    }
}
```

À la première utilisation, la fonction initialise le cookie *welcomeCookie* contenant la dernière connexion et affiche un message de bienvenue.

Lors d'une connexion ultérieure, la fonction vérifie si l'utilisateur ne s'est pas encore déjà connecté aujourd'hui et dans ce cas elle affiche un message de bienvenue personnalisée grâce au cookie *userCookie* contenant les informations de l'utilisateur

Dans tous les cas le message apparaît puis disparaît automatiquement après 2.5s grâce aux fonctions *displayModal* et *setTimeout*.

b. Recherche de produits par mots-clés

Les recherches se font par requête à l'API d'OpenFoodFacts :

```
"https://fr.openfoodfacts.org/cgi/search.pl?search_terms=[query]&search_simple=1&action=process&json=1"
```

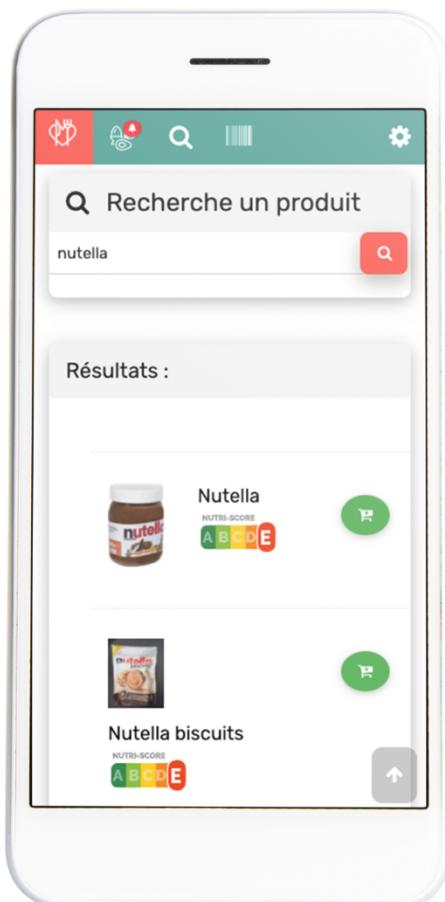
search_terms = [query] : correspond aux termes de la recherche

search_simple = 1 : recherche de base

json = 1 : Résultat de la requête en JSON

On extrait alors la liste de *product* qu'on affecte à une variable du *\$scope* du *searchcontroller.js* de la page *search.html*. On affiche alors cette liste de *product* grâce à la directive *ng-repeat*.

- Affichage



c. Recherche d'un produit par code-barre

- Obtention du code-barres

Le code-barre d'un produit est obtenu par saisie ou par scan nous ne détaillerons pas la saisie qui est simplement l'association d'un *input* grâce à la directive *ng-model* à une variable du *\$scope*.

Pour le scan du code on utilise une dépendance : QuaggaJS on configure son initialisation de la façon suivante :

```
Quagga.init({
    inputStream: {
        frequency: 0.5,
        name: "Live",
        type: "LiveStream",
        target: document.querySelector('#camera')
    },
    decoder: {
        readers: ["ean_reader"]
    }
}, function(err) {
    if (err) {
        console.log(err);
        var errordisplay = document.getElementById('camera') ;
        errordisplay.innerHTML = "<div style='width: 200%;text-align: center;'><span style='font-size:40px'>Oups !</span><br>DietApp ne peut accéder à la camera<br><strong>Donnez lui l'autorisation</strong> dans les paramètres système</div>";
        return
    }
    console.log("Quagga initialization finished. Ready to start");
    Quagga.start();
})
```

On détaillera les éléments importants, pour plus de détails se référer à la documentation de QuaggaJS

Permet d'utiliser la camera comme source (à défaut d'une image ou d'une vidéo pré-enregistrée)

Sous forme de sélecteur CSS on donne le conteneur où l'on voudra afficher la camera, dans notre cas : une balise `div` avec pour id "camera".

Liste des types de code-barres que QuaggaJS cherchera à lire, pour éviter les faux positifs mieux vaut en mettre le moins possible. Dans notre cas on lit seulement les EAN, qui correspondent aux codes-barres commerciaux européens.

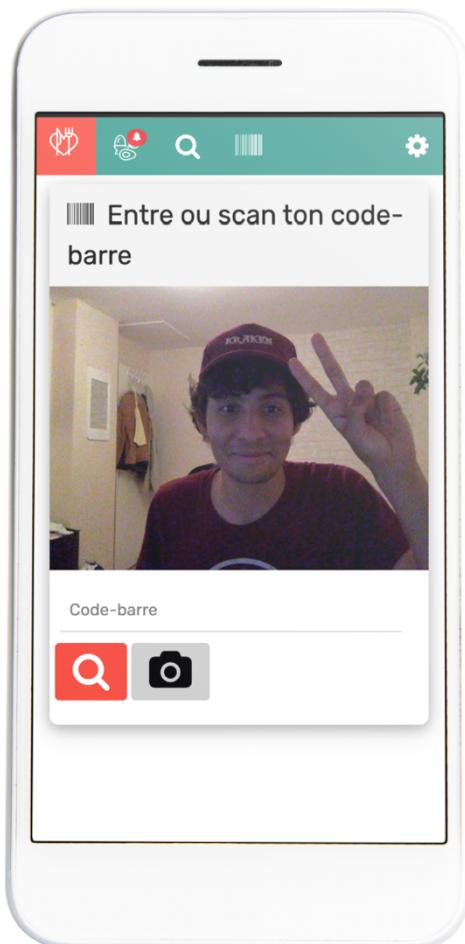
Fonction d'erreur d'initialisation : dans notre cas l'erreur d'initialisation correspond généralement à l'absence de permission sur la caméra, c'est donc le message que l'on affiche.

- Recherche du produit grâce au code

Une fois le code obtenu on obtient le produit par requête HTTP à l'API OpenFoodFacts :

"[https://world.openfoodfacts.org/api/v0/product/\[code\].json](https://world.openfoodfacts.org/api/v0/product/[code].json)

- Affichage



d. Gestion du panier

cart = {products = Array(product)} est contenu dans le cookie **cartCookie**

- Primitives (js/cartfactory.js) :

Des primitives du panier contenus dans js/cartfactory.js permettant de gérer le panier sont explicites, elles ne seront donc pas détaillées :

init() -> null : si 1^{ère} utilisation, la fonction initialise le *cart* à `{products: []}`

get() -> cart : renvoie le *cart*

addProduct(product) -> null : ajoute *product* à *cart.product*

delProduct(code) -> null : supprime s'il existe le *product* ayant pour code barre *code*

clear() -> null : efface le contenu du panier

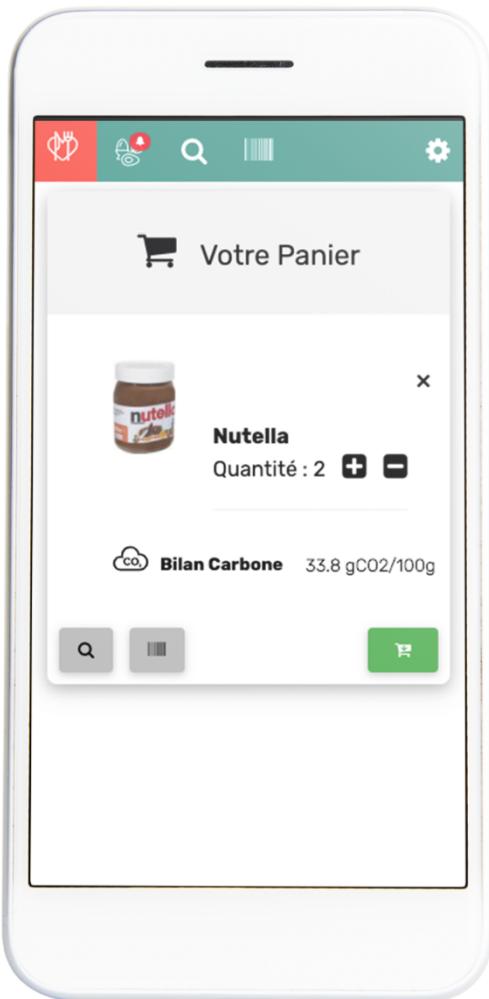
incr(code) -> null : incrémente l'attribut *quantity* du *product* ayant pour code barre *code*, s'il existe

decr(code) -> null : décrémente l'attribut *quantity* du *product* ayant pour code barre *code*, s'il existe

carbonFootprint() -> int : calcule la somme des empreintes carboniques des *product*

Les favoris fonctionnent sur le même principe à la différence qu'elles utilisent uniquement les primitives : *init*, *get*, *addProduct*, *delProduct*, *clear*, pour des raisons évidentes.

- Affichage :



e. Proposition d'alternative

Cette fonctionnalité est très semblable à la fonctionnalité de recherche, en effet lorsque l'utilisateur sélectionne un produit, l'application effectue la requête à l'API d'OpenFoodFacts suivante :

```
$http.get("https://fr.openfoodfacts.org/cgi/search.pl?action=process&tagtype_0=categories&tag_contains_0=contains&tag_0="+ acategorie +"&json=true")
```

Détaillons cette requête :

Si on veut chercher des produits semblables à un certain *aproduct* on prend le premier élément de sa liste de *categories*, qui est censé être le plus discriminant selon la documentation d'OpenFoodFacts, on a ainsi *acategorie = aproduct.categories[0]*. Ensuite on effectue une recherche sur les produits se trouvant dans la même catégorie grâce aux attributs :

tagtype_0 = categories Définit le type du tag n°0 (dans notre cas une catégorie)

tag_contains_0 = contains : Permet de renseigner que les tags des produits recherchés devront contenir le tag n°0

tag_0 = "Biscuit chocolatée" : Permet de renseigner le tag n°0

Un fois la liste de produits alternatifs obtenus on les ordonne du nutriscore le moins élevé au plus élevé (ie. du meilleur au pire nutritivement) grâce à la relation d'ordre définie par la méthode suivante :

```
alterproducts.sort(
    function(product_a,product_b) {
        return (product_a.nutriscore_score - product_b.nutriscore_score)
    }
);
```

- Affichage :



f. Proposition de conseils

Voici la fonctionnalité phare de l'application. En effet, contrairement à la proposition d'alternative qui est la même pour tous car se basant seulement sur le nutri-score, la proposition de conseils, elle, est totalement personnalisée en fonction du profil utilisateur vu précédemment. Cela consiste en la discrimination de différents produits pour trouver le produit optimal à conseiller.

Toutefois cette fonctionnalité a **deux problématiques** :

1. Sur quelle liste de produit discriminer ?

En effet, bien que la complexité temporelle d'une telle fonction soit linéaire, on ne peut se permettre d'utiliser l'ensemble de la base de données pour des raisons évidentes (à l'heure où ce rapport est écrit il y'a 1 390 189 produits dans la base de données OpenFoodFact mondiale)

2. Comment discriminer des produits, pour déterminer le produit optimal à conseiller ?

Nous allons donc étudier ces deux problématiques plus en profondeur

1. Sur quelle liste de produit discriminer :

En réalité, DietApp utilise deux listes de produits pour établir des conseils

- 1.1 Les produits favoris

On utilise simplement la méthode `get` de `js/favoritefactory.js` pour obtenir cette liste de produits mise en favoris, sur laquelle on cherchera à trouver le produit optimal à conseiller.

- 1.2 Les produits fréquents

D'un point de vue mathématiques, la liste des produits fréquents est simple à définir et à maintenir : En effet, il suffirait à première vue de maintenir une liste de tuple (`product, nb`) où `nb` est le nombre de fois où on a ajouté `product` au panier et alors :

$$f_{aproduct} = \frac{nb_{aproduct}}{\sum_{product} nb_{product}}$$

Toutefois cette implémentation a des inconvénients techniques mais aussi pratiques :

En effet, pour calculer la fréquence on a une complexité linéaire due au calcul de $\sum_{product} nb_{product}$. Ainsi on change la structure en un objet qui permet d'obtenir ce dernier en $O(1)$ grâce à un attribut *total* mis à jour à chaque achat de produit.

Par ailleurs, il est inutile de garder les produits achetés uniquement une fois, qu'on ne rachètera plus, car en plus d'être inutile, à long terme cela poserait des problèmes de complexité spatiale.

Enfin, cette implémentation ne prend pas en compte les changements d'habitudes liés à la saison/période ou à la sortie de nouveaux produits.

Par exemple : Imaginons que j'adore le nutella, ça fait 2 ans que j'en achète fréquemment, $f_{nutella}$ est donc très élevé, mais aujourd'hui ça m'écoeure (j'ai dit imaginons...) et j'arrête totalement d'en acheter. Toutefois, l'implémentation qu'il restera considéré comme un objet fréquent pendant un bon bout de temps encore...

Ainsi, il fallait permettre l'actualisation de ces fréquences sans pour autant perdre toute les données car sinon les conseils en seraient affectés. On ajoute donc un attribut *month* et on initialise l'objet produit fréquents de cette manière :

```
var frequently = {
  month: now.getMonth(),
  total: 0,
  products: [] // un produit sera de la forme {product_name, code, nb, plg_list}
};
```

Intéressons-nous à l'actualisation (mensuelle) de cette liste de produits fréquents :

Grâce à l'attribut *month* et à la méthode *Date().getMonth*, on détermine si il y'a une actualisation à effectuer. Dans ce cas on effectue les opérations suivantes :

On calcule les fréquences des produits enregistrés de la manière suivante :

```
var nbArr = [] ;
for (var product of frequently.products) {
    nbArr.push(product.nb) ;
}
var freqArr = nbArr.map(x => x/ frequently.total) ;
```

Ensute, on définit une constante float *seuil* qui est la fréquence seuil d'actualisation. Dès lors, on parcourt la liste *frequently.products* et on supprime les *product* qui vérifient $f_{product} < f_{seuil}$, cela permet d'éviter de stocker une multitude de *product* avec des fréquences quasiment nulles.

Enfin, afin de s'adapter au mieux aux changements de comportements/habitudes les produits gardés vont voir leur compteur remis à 0 (ainsi que l'attribut *total* de *frequently*)

C'est ainsi qu'on a défini une implémentation pérenne d'une liste de produits fréquents à la complexité correcte.

2. Comment discriminer des produits pour déterminer le produit optimal à conseiller ?

Rappelons-le : nous voulons discriminer par rapport au profil diététique de l'utilisateur, ainsi, le produit optimal à conseiller est le produit qui, s'il est ajouté, nous rapprocherait le plus possible de notre profil choisi (i.e. : Minceur ou Santé ou Sportif). Toutefois il est nécessaire de quantifier ce rapprochement, c'est ce qui est décrit ci-dessous :

Il faut définir une fonction de distance entre 2 listes Protides/Lipides/Glucides. Dans notre cas on utilise tout simplement la distance associée à la norme euclidienne en dimension 3.

$$d(A, B) = \sqrt{(B.\text{protides} - A.\text{protides})^2 + (B.\text{lipides} - A.\text{lipides})^2 + (B.\text{glucides} - A.\text{glucides})^2}$$

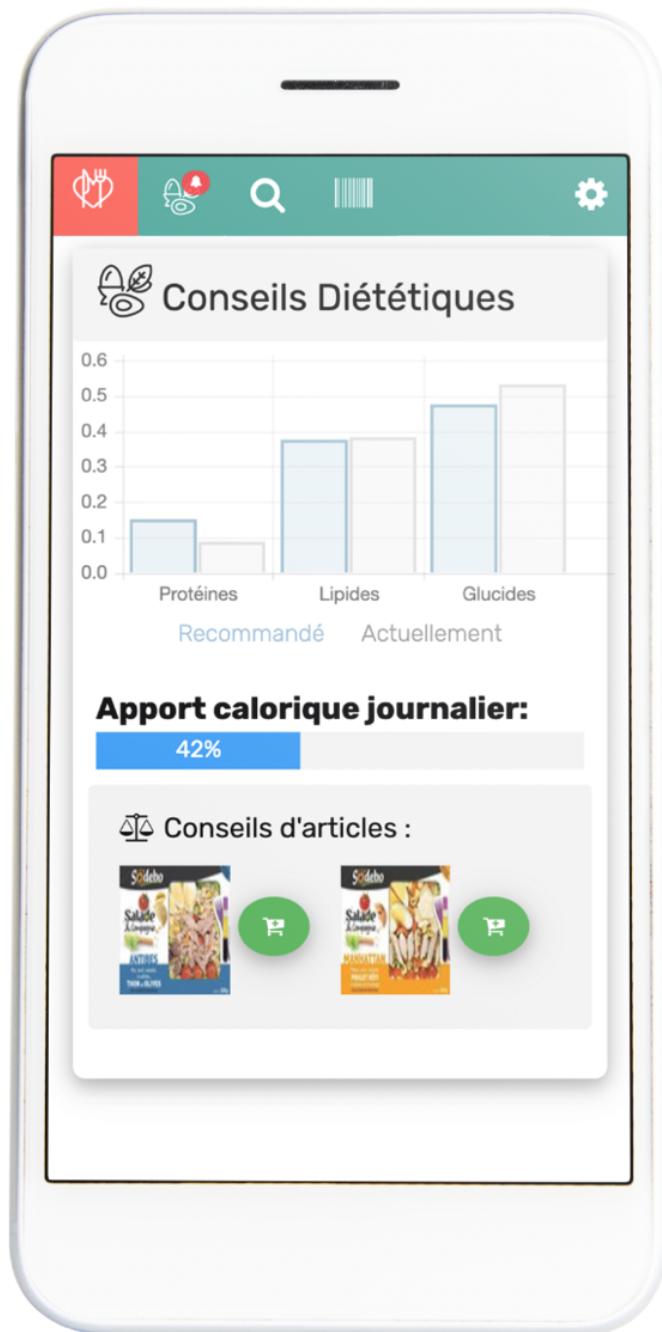
Ensuite, par parcours de la liste de produits (fréquents ou favoris) on détermine le *aproduct* qui minimise la distance entre le tuple (Protides,Lipides,Glucides) du panier utilisateur réel auquel on a ajouté *aproduct* et le tuple (Protides,Lipides,Glucides) correspondant à ses ambitions diététiques :

```
for (var product of products) {
    var sum =
plg[0]+product.plg_list[0]+plg[1]+product.plg_list[1]+plg[2]+product.plg
_list[2] ;
    var newplg_ratio = [(plg[0]+product.plg_list[0])/sum,
(plg[1]+product.plg_list[1])/sum, (plg[2]+product.plg_list[2])/sum];
    var newdiscr = discr(newplg_ratio,plg_ratioR) ;
    if (newdiscr <= mindiscr) { // lorsqu'on trouve un meilleur
produit à conseiller
        mindiscr = newdiscr ;
        best = product
    }
}
if(best == null) { return best;} // Permet de rattraper l'erreur
au niveau de l'affichage HTML
else { return best.code;} // le code suffit à identifier un
produit
```

où **discr** correspond à notre fonction de distance $d(A,B)$

C'est ainsi qu'on détermine (ou pas) les produits à conseiller parmi les produits favoris et/ou fréquents. On les intègre ensuite dans le fichier *advice/advice.html* grâce au contrôleur *advice/advicecontroller.js*

- Affichage



g. Diagramme double des macronutriments



en JavaScript dans advice/advicecontroller.js :

```
$scope.labels = ['Protéines', 'Lipides', 'Glucides'];
$scope.series = ['Recommandé', 'Actuellement'];
$scope.data = [
  adviceFactory.getProportion(adviceFactory.getUserProfile()), // Recommandé
  adviceFactory.evalCartProportion() // Actuellement
```

En HTML dans advice/advice.html

```
<canvas width="" height="" style="margin-left: 5px;" id="line" class="chart
chart-bar" chart-data="data" chart-labels="labels"></canvas>
```

La partie HTML est simplement obtenue par la documentation de Chart.js nous nous intéresserons plus particulièrement à la partie JavaScript :

`adviceFactory.getUserProfile()` renvoie le profil diététique de l'utilisateur ie. 'loseweight' ou 'healthy' ou 'gainweight'.

`adviceFactory.getProportion(aprofile)` renvoie les proportions
Protides/Lipides/Glucides du profil *aprofile*
Ainsi `adviceFactory.getProportion(adviceFactory.getUserProfile())`
renvoie la proportion en macronutriments du profil d'ambition diététique
de l'utilisateur et donc ce qui lui est recommandée.

`adviceFactory.evalCartProportion()` renvoie tout simplement la
proportion en macronutritif du panier courant.

On compare ces deux dernières listes de proportion de macronutriments au
moyen de diagramme en les nommant respectivement : « Recommandée »
et « Actuellement »

h. Cartes de fidélité

On définit l'objet *card* ayant la structure suivante :

```
card = {
    name: name, // nom de la carte e.g Carrefour, G20
    code: code, // code de la carte
    type: "EAN8", // EAN8 ou EAN13
    imgsrc : URL // url de l'image du code-barres
}
```

où **URL** = "https://barcode.tec-it.com/barcode.ashx?data=" + **code** +
"&code=EAN8&multiplebarcodes=false&translate-
esc=false&unit=Fit&dpi=96&imagetype=Gif&rotation=0&color=%23000000&bgc
olor=%23ffffff&codepage=&qunit=Mm&quiet=0"

On utilise ainsi l'API de barcode.tec-t.com de manière un peu cachée, puisqu'il n'y a pas de données à recevoir au travers du module AngularJS \$http, simplement une image du code-barre à afficher grâce à une balise *img* avec pour attribut *src=URL*.

- Affichage :



4. Cordova – (Application Builder)

Cordova est l'outil qui permet de compiler son code web (HTML, CSS, JS) en application multiplateforme. Toutefois, avant d'arriver à cette prouesse il est nécessaire de télécharger bon nombre de librairies, je laisse donc au lecteur curieux le choix de se renseigner davantage sur [la documentation de Cordova](#) s'il veut en faire de même. De plus vous trouverez à la rubrique *Problèmes rencontrés et Anecdotes* de ce rapport des informations très intéressantes mais absentes de la documentation.

III. Problèmes rencontrés et Anecdotes :

Les différents problèmes rencontrés et anecdotes de programmation vous sont détaillés ci-dessous dans l'ordre chronologique de rencontre lors de ce projet de développement informatique. À chaque problème est associé notre solution (qui n'est pas forcément la meilleure) ainsi que des liens/commandes qui nous ont été utiles.

a. Apache

Contrairement aux autres sites-web que j'avais codé en pure HTML et CSS. Un site-web utilisant AngularJS nécessite d'être héberger sur un serveur Apache.

Solution :

Installer un serveur Apache sur son ordinateur

```
$ sudo apt-get update
$ sudo apt-get install apache2
```

<https://httpd.apache.org/>

b. Gestion de la taille des cookies

- Les objets produits tels que renvoyés par l'API d'OpenFoodFacts sont beaucoup trop lourd en mémoire,
- Le cookie a un taille maximale de 4ko

Or, le panier de l'utilisateur est stocké dans un cookie...

Résultat : le panier de l'utilisateur ne peut contenir qu'un seul produit avant que le cookie stockant le panier ne soit plein en mémoire, pas très utile le panier...

Solution :

Définir 2 objets produits : une version complète « offproduct » tel que renvoyé par OpenFoodFact et une version light « product » contenant uniquement les attributs utiles à DietApp. De cette manière le panier peut contenir plus de 1000 produits

c. Faux positifs de QuaggaJS

Lors du scan de nos premiers code-barres nous remarquons que souvent le code renvoyé n'est pas celui qui a été scanné, parfois il s'agit d'une simple erreur d'un chiffre et parfois même du code entier.

Solution :

Si on n'indique pas à QuaggaJS quel type de code-barre il doit lire, il va être amené à vouloir scanner tous les types de code-barre qu'il a à sa connaissance et cela engendre de nombreux faux-positifs

Il suffit donc de correctement initialiser l'objet *decoder* avec :

```
readers: ['ean_reader'],
```

d. Mise en cache du site-web

Pour éviter de recharger constamment les mêmes pages certains navigateurs tel que Chrome ou Safari ont eu la bonne idée de mettre pendant une certaine période certaines données en cache. Toutefois, lorsque que l'on code activement notre site-web et que même après rafraîchissement de la page le site-web n'est pas mis à jour, cela pose un vrai problème pour coder tout en vérifiant nos ajouts...

Solution :

Désactiver la mise en cache :

Sous Chrome, il suffit de faire un clic-droit sur la page, cliquer sur « Inspecter l'élément », puis dans l'onglet « Network » de l'inspecteur cocher la case « Disable cache ».

e. Permission de la camera sur Android

C'est certainement le problème qui m'a pris le plus de temps à résoudre et à comprendre.

Lors de la compilation vers l'APK (application android) grâce à Cordova je pensais que lorsque QuaggaJS utilisait la caméra il y aurait une demande de permission d'accès à la caméra comme sur la version Chrome Desktop, car, rappelons-le Cordova compile notre site web en application android grâce à une version mobile de Chromium. Toutefois, rien ne se passe la caméra n'est pas démarrée, je comprends tout de suite qu'il s'agit d'un problème de permission.

Je parcours alors la documentation de Cordova à la recherche de mot-clés tel que « Camera permission » ou « Camera access » et je tombe sur des plugins Cordova d'acquisition vidéo. Toutefois ces plugins intègrent leur propre lecteur vidéo et accorde la permission seulement à leur lecteur, tandis que mon lecteur QuaggaJS lui n'a toujours pas la permission...

Ensuite, je me penche du côté de la documentation d'Android Studio, car je le rappelle Cordova utilise les librairies d'Android Studio pour créer l'APK. Je lis dans leur documentation que les permissions sont gérées dans le fichier AndroidManifest.xml, miracle il est aussi présent dans chaque dossier d'application (avant compilation) je suis la documentation et j'ajoute les lignes suivantes :

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Alors je pense que c'est gagné, mais en fait non : lors de la compilation Cordova renvoie justement une erreur de compilation.

Je cherche sur le net et je me rends compte que je ne suis pas le seul dans ce cas et bien sûr et je tombe alors sur un repo* d'un plugin non-officiel de Cordova qui permet d'ajouter n'importe quelle permission à l'ensemble de l'application compilée. J'utilise ce dernier et l'installe grâce à la commande :

```
$ cordova plugin add https://github.com/dpa99c/cordova-custom-config.git
```

Et en ajoutant dans le fichier de configuration de Cordovva config.xml les lignes suivantes :

```
<custom-config-file parent="/*" target="AndroidManifest.xml">
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
</custom-config-file>
```

De cette manière la permission est demandée à l'utilisateur au moment de l'installation de l'application android.

* repo de cordova-custom-config : <https://github.com/dpa99c/cordova-custom-config>

IV. Tests

Nous avons respecté les conditions d'utilisation des API qui étaient :

- **OpenFoodFacts :**
Ne pas scraper la base de données, c'est-à-dire de faire une/des requête(s) dans le but d'obtenir la totalité de la base de données en effet des liens de téléchargement sont prévus à cet effet.
- **Barcode generator :** Mettre un lien de retour vers leur site-web.

Nous avons veillé à la terminaison des fonctions codées en rattrapant les erreurs, et plus particulièrement celles dues aux types Javascript : undefined, null et NaN.

V. Déploiement

Tout d'abord, commencez par cloner le repo de DietApp :

```
git clone http://gitlabens.imtbs-tsp.eu/nordine.marie/pro3600-20-ran-37.git
```

Si vous voulez déployer l'application deux méthodes s'offre à vous :

- Compilez vous-même le code sur votre plateforme désirée au moyen de Cordova en copiant le dossier *dev/dietapp* dans votre projet Cordova cf la documentation de Cordova pour plus de détails
- Installez directement les versions distribuables (.apk et .ipa) de DietApp présentes dans le dossier *dist*

VI. Bilan

L'état de l'application au moment du rendu final respecte le cahier des charges. Tout d'abord l'application livrée est bien compatible avec Android comme IOS. Pour rappel, les fonctions de DietApp prévues dans le cahier des charges étaient d'aider les consommateurs à gérer leurs achats et liste de course et d'apporter des conseils **personnalisés** sur le choix des produits. En effet le but de créer un traitement unique pour chacun des utilisateurs a été le centre de notre réflexion et le suivi personnel promis dans le cahier des charges a été respecté.

Le profil d'une personne est composé des champs prévus, à savoir son âge, son genre, ses ambitions diététiques et nous pouvons même grâce à un ajout bonus prendre en compte ses goûts via la possibilité pour l'utilisateur de gérer ses articles Favoris, et la mise en mémoire de produits fréquemment consommé. Chaque utilisateur reçoit alors des suggestions différentes.

De plus l'utilisateur est aisément capable de gérer son expérience sur l'application. Il a accès aux informations nutritionnelles de tous les produits simplement en scannant le code-barres. Il pourra ainsi l'ajouter au panier, aux produits favoris en connaissance de cause. Nous lui fournissons aussi un outil de visualisation de l'équilibre de ses repas. En effet un graphique représentant ses **besoins** nutritionnels en lipides / glucides / protéines et les **apports** nutritionnels en lipides / glucides / protéines du repas qu'il compose est présent.

Dans le but d'amener un critère de sélection supplémentaire intéressant en lien avec l'environnement, il a aussi été ajouté une option de filtre en fonction de l'empreinte carbone de chaque produit. Pour faciliter l'expérience utilisateur lorsque celui-ci utilise notre application alors qu'il fait ses courses en magasin, un module d'enregistrement de carte de fidélité a été implémenté.

Les interfaces sont prévues pour être accessible facilement pour une utilisation par le consommateur. L'intégralité du cahier des charges a été respecté, il y a eu même quelques ajouts supplémentaires qui nous ont semblé utiles le moment venu.

Finalement, malgré la fin de ce projet, nous pensons que cette application a encore du potentiel et qu'il est possible de la rendre encore plus performante, pratique, utile au quotidien. Nous avions pensé par exemple à

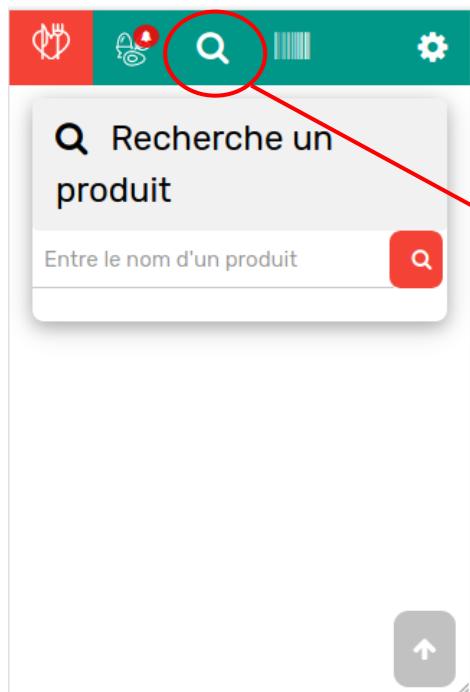
un champ allergie dans le profil, de sorte que les suggestions évitent les produits contenant ces allergènes. Ensuite il est évidemment possible d'affiner le graphique : séparer glucides dont sucres ou non, pouvoir sélectionner les doses sur les produits (par exemple il n'est pour le moment pas possible d'avoir l'apport d'une tartine à la confiture, le graphique nous montre les caractéristiques du paquet de tartines et du pot de confiture en entier). Ensuite comme la finalité peut être l'utilisation dans les centres commerciaux, la possibilité d'avoir accès au plan du centre que trace le parcours à prendre selon la liste, ou une fois dans le magasin nous suggère les articles en promotion. De plus, il semble réalisable d'intégrer aussi un module qui vous propose des recettes selon vos besoins en nutriments, et ajoute directement les ingrédients au panier. Enfin, si nous devions rendre l'application commerciale nous nous tourneront certainement vers des bases de données de supermarchés, car si OpenFoodFacts présentait l'avantage d'être gratuit et ouvert, il a de nombreux inconvénients : produits doublons, information manquante, erronée et j'en passe.

VII. Manuel utilisateur

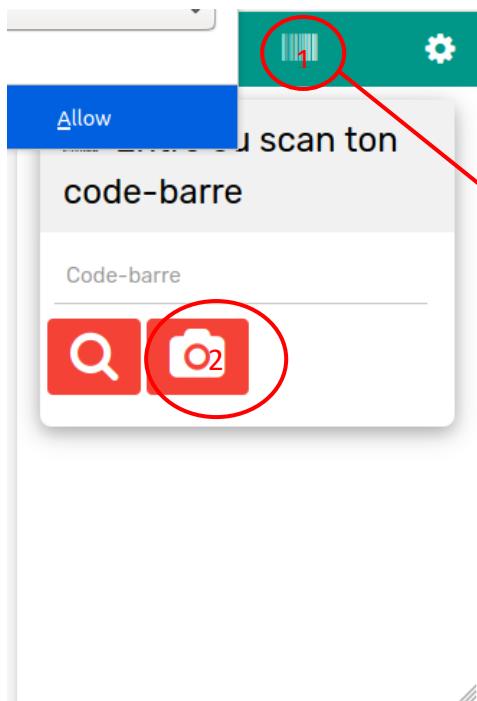
L'utilisation de l'application est ici détaillée. L'application étant rendu disponible sur android via un APK ou sur iOS via un IPA, il suffit d'abord à l'utilisateur d'ouvrir l'application sur son téléphone, celui-ci arrivera alors sur l'écran d'accueil de DietApp :



Lors de sa première connexion, l'utilisateur doit d'abord renseigner son profil, il fournit alors son nom, son âge, son sexe ainsi que le régime qu'il souhaite suivre afin que l'application lui fournisse des conseils en correspondance avec ses besoins.

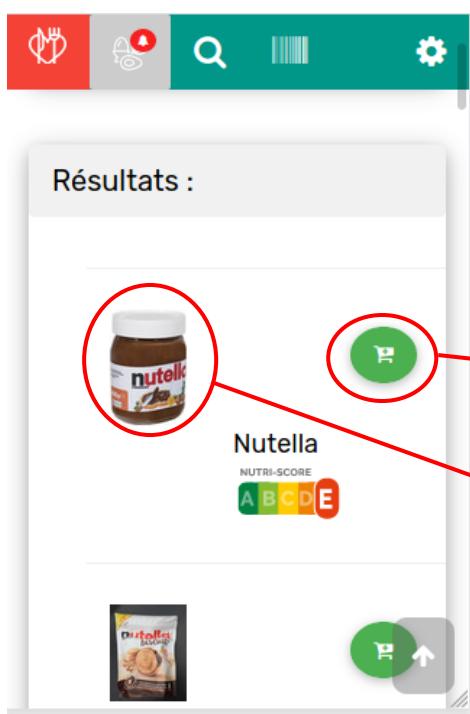


L'utilisateur peut ensuite accéder à la première fonctionnalité de l'application : la recherche de produit. Pour cela, l'utilisateur clique sur le troisième onglet du menu.



Une autre manière de rechercher le produit souhaité pour l'utilisateur serait de scanner le produit qu'il a en main : pour disposer de cette fonctionnalité il suffit alors de cliquer sur la dernière option du menu (1) puis de cliquer sur l'appareil photo (2). Il est toujours possible d'effectuer une recherche par mot-clé en cliquant sur la loupe.

N.B. : le test ayant été réalisé sur ordinateur dans le cadre de la rédaction de ce guide, on remarquera que l'ordinateur demande l'autorisation pour utiliser la webcam, le code barre peut alors être scanné et le produit ajouté au panier.



Notre utilisateur choisit finalement de cliquer sur la loupe et utilise la fonction recherche par mot clé. En recherchant une célèbre marque de pâte à tartiner celui-ci accède aux résultats de sa recherche. L'application affiche alors le nutri-score et permet à l'utilisateur d'ajouter le produit au panier. Cependant, si l'utilisateur souhaite plus d'information sur le résultat de sa recherche celui-ci peut cliquer sur l'image associée au produit afin d'obtenir davantage d'informations.

Nutella

Pâte à tartiner aux noisettes et au cacao

NUTRI-SCORE

A B C D E

Marque ferrero nutella

Origine

Empreinte Carbone 33.8 (gCO₂/100g)

Ajouter au panier

Nous vous conseillons aussi :

Une fenêtre d'informations s'affiche alors. Il est alors possible de connaître l'empreinte carbone du produit, sa marque et son origine lorsque ces données sont renseignées. Il est également possible d'ajouter le produit à ses favoris en cliquant sur le cœur à côté du nutri-score et du descriptif du produit. La possibilité d'ajouter le produit au panier est toujours disponible.

aux noisettes et au cacao

Marque ferrero nutella

Origine

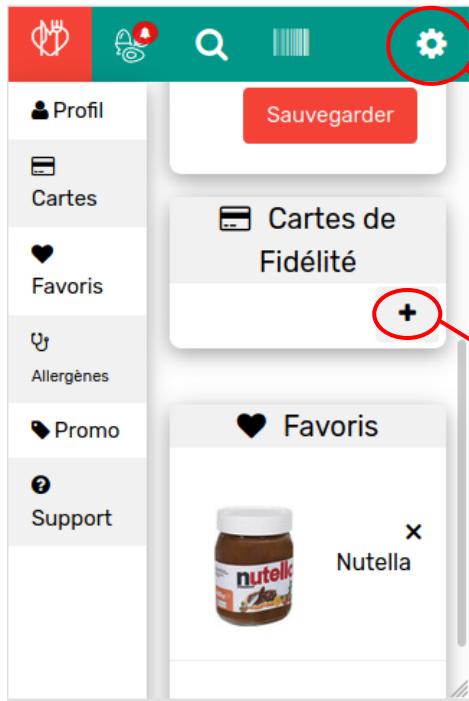
Empreinte Carbone 33.8 (gCO₂/100g)

Ajouter au panier

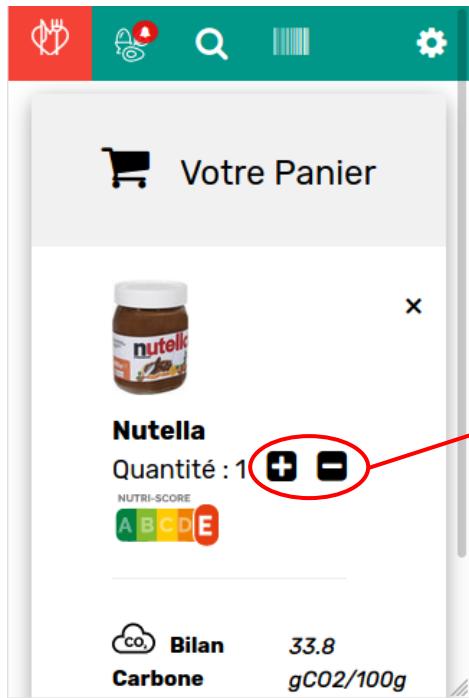
Nous vous conseillons aussi :

Jocciolata

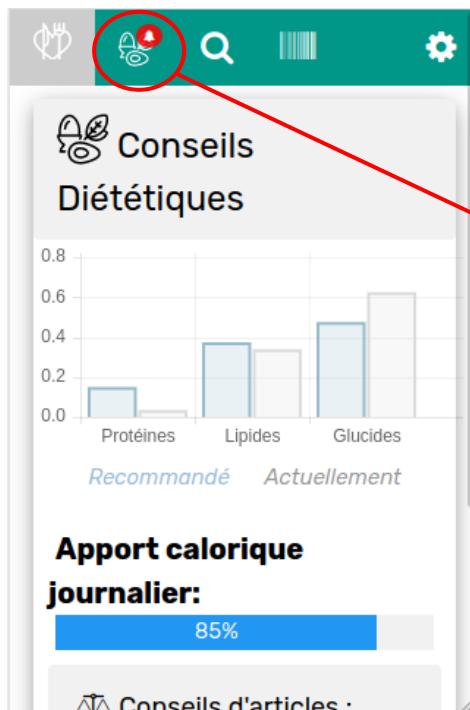
Enfin, en bas de cette fenêtre se trouve également un onglet de produits plus sains conseillées à l'utilisateur sur la base du nutri-score.



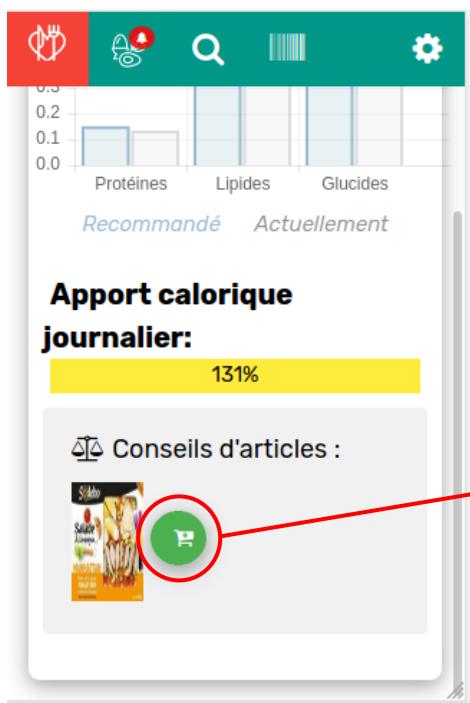
L'utilisateur ajoute finalement le produit à ses favoris. Pour consulter l'ensemble de ses produits favoris, l'utilisateur clique sur l'onglet le plus à droite du menu afin d'accéder à ses paramètres, en bas de cet onglet se situe alors la liste de ses produits favoris.
L'utilisateur peut également entrer ses cartes de fidélité dans l'application.



L'utilisateur peut également consulter son panier en retournant sur la page d'accueil, cette interface lui permet de consulter le nutri-score ainsi que d'incrémenter ou décrémenter la quantité du produit choisi. L'application contient également une fonctionnalité permettant d'afficher le bilan carbone du panier.



L'utilisateur peut ainsi passer à la phase d'équilibre de son panier. Celui-ci se rend alors dans l'onglet équilibre (deuxième onglet du menu). L'utilisateur peut alors comparer la teneur en macronutriments ainsi que l'apport calorique de son panier avec les apports recommandés pour une journée pour une personne correspondant à son profil

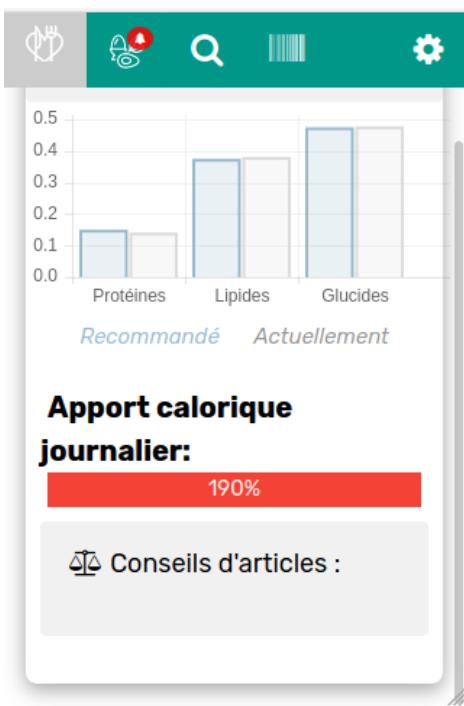


Enfin, si l'utilisateur descend il peut se voir conseiller des articles pour atteindre les apports journaliers et ainsi équilibrer son panier. Il peut alors ajouter les produits à son panier.

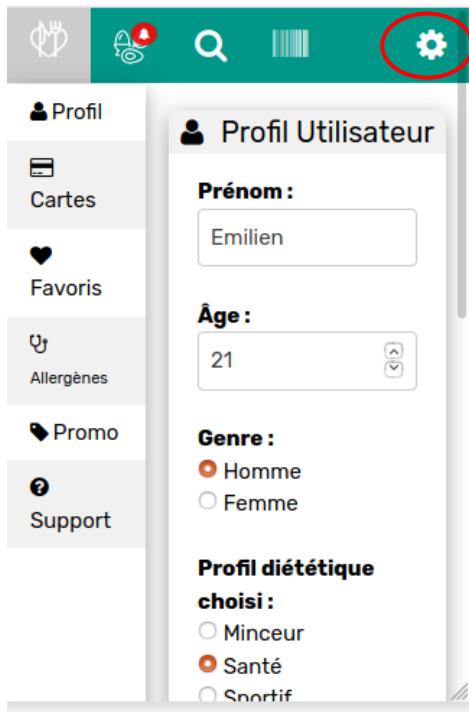
N.B : La salade a déjà été ajouté d'où la différence de calories entre les deux captures d'écrans.



L'utilisateur constate alors en se renseignant sur le produit conseillé par son descriptif que celui-ci semble compenser ses écarts, il peut alors l'ajouter au panier.



L'utilisateur peut alors continuer de suivre les conseils prodigués par l'application, celle-ci s'arrêtera de fournir des conseils lorsque l'apport caloriques sera jugé trop important. On remarque de plus que l'équilibre des macronutriments est optimal une fois que les conseils ont été donné.



Enfin, si l'utilisateur a fêté son anniversaire récemment ou s'il souhaite changer son type de régime, celui-ci peut accéder aux paramètres rentrés lors de la première utilisation à l'aide de l'onglet paramètre (onglet le plus à droite). L'utilisateur peut alors par exemple suivre un profil sportif.



L'utilisateur remarque alors que ces changements modifient ses besoins journaliers.

VIII. Bibliographie

CSS Frameworks

W3CSS - Modern & Responsive CSS

<https://www.w3schools.com/w3css/>

fontawesome - Icônes

<https://fontawesome.com/>

Loading.io – Animations de chargement

<https://loading.io/>

Modules JavaScript

ChartJS - Visualisation de données

<https://www.chartjs.org/docs/latest/>

QuaggaJS - Lecteur de code-barres

<https://serratus.github.io/quaggaJS/>

Modules AngularJS

ui-router - Gestion des routes

<https://ui-router.github.io/ng1/>

ng-cookies- Gestion des cookies

[https://docs.angularjs.org/api/ngCookies/service/\\$cookies](https://docs.angularjs.org/api/ngCookies/service/$cookies)

APIs

OpenFoodFacts - Base de données et API de produits alimentaires

<https://documenter.getpostman.com/>

Barcode generator - API génératrice de code-barres

<https://barcode.tec-it.com/>

Divers

Apache – HTTP server

<https://httpd.apache.org/>

Cordova – Multiple platforms compiler

<https://cordova.apache.org/>

Custom config – Cordova custom configuration

<https://github.com/dpa99c/cordova-custom-config>

Atom - a hackable text editor

<https://atom.io/>