IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

Luc Fabresse - *luc.fabresse@imt-nord-europe.fr*
Rémy Pinot

Projet IDAW : $i\mathcal{M}$anger$\mathcal{M}$ieux ($i\mathcal{MM}$)

16 mars 2023

1 Sujet

Vous devez réaliser une application Web permettant de maintenir un journal de tous les aliments que vous consommez. L'idée d'un "tracker" de nourriture permet de me mieux contrôler ses différents apports énergétiques.

Un utilisateur de l'application est caractérisé par son login. Par exemple, en fonction de sa tranche d'age (<40, <60, 60+), de son sexe (homme, femme) et de son niveau de pratique sportive (bas, moyen, élevé), il est possible de calculer ses besoins énergétiques journaliers.

Après avoir renseigné son profil, l'utilisateur doit pouvoir entrer les aliments qu'il consomme et en quelle quantité à une date donnée. L'historique des aliments consommés pourront être visualisés sous la forme d'un tableau. Il doit être possible de filtrer ce tableau :

— sur une période donnée (jour, semaine, mois, tout)
— par type d'aliment
— ...

Notez qu'un aliment peut être composé d'autres aliments. Par exemple : une ratatouille est généralement composée de tomates, courgettes, aubergines, ... avec un % de composition qui permettra de déduire la quantité par sous aliments. Cette notion doit être prise en compte dans la base de données. Par contre, elle pourra être ignorée lors de la création des formulaires de gestion des aliments (cf. Section 4).

A vous de déterminer les caractéristiques des aliments que vous souhaitez intégrer dans votre application ainsi que leur granularité. A titre d'exemple :

— ratios glucide, lipide, protéine, sel, sucre, ...
— type d'aliment : légume, fruit, produit transformé, viande, poisson, ...
— ...

A terme, l'application doit calculer et afficher des indicateurs intéressants pour une période donnée (jour, semaine, mois, ...). Par exemple : quantité de calories moyenne consommée, quantité de sel ingérée, quantité de sucre, type d'aliments consommés, évolution de la charge glycémique, ... Ces indicateurs sont à confronter avec les recommandations nutritionnelles officielles. Par exemple, l'OMS [1] recommande d'ingérer une quantité maximum de sel par jour. L'idée est de proposer un tableau de bord synthétique à l'utilisateur afin de l'alerter sur les recommandations qu'il ne suivrait pas.

A minima, les fonctionnalités suivantes sont attendues :

— affichage d'un tableau CRUD affichant une liste d'aliments
— aliments stockés en base dans une table aliment
— un formulaire permettant d'ajouter et éditer les aliments

L'architecture de votre application devra être conforme l'architecture REST (cf. Figure 1) et donc découpée en 2 parties : le backend et le frontend.

1. https://www.who.int/fr/news-room/fact-sheets/detail/salt-reduction

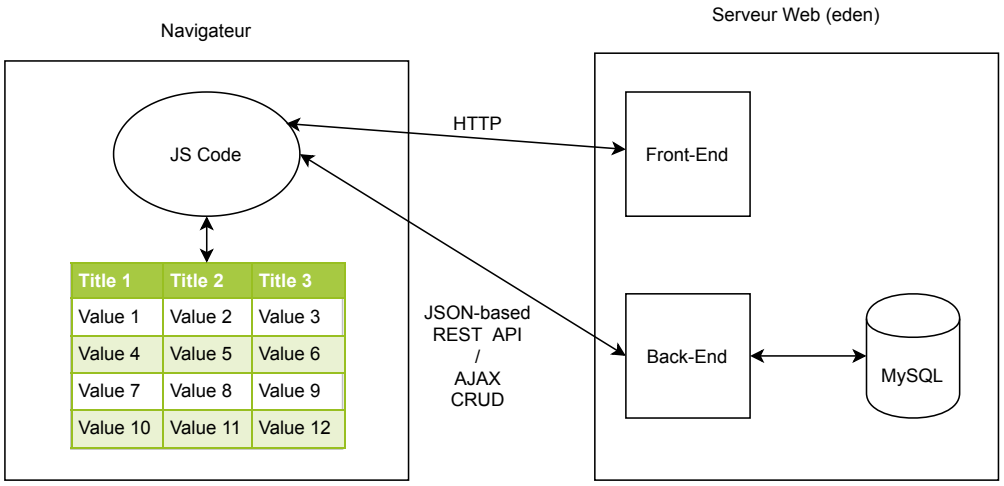


FIGURE 1 – Architecture de votre application Web

2 Jalon 1 : Le backend

Il s'agit de code PHP (pas de HTML, pas de CSS, ni de JS) qui reçoit des requêtes HTTP et retourne des réponses HTTP contenant du JSON conformément à une API que vous devez construire.

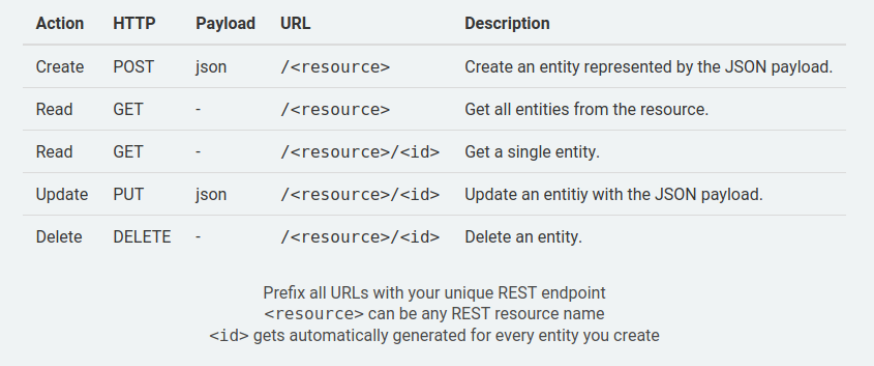
Organisation attendue des fichiers du backend de votre projet dans votre git :

```
Projet
   |- backend
         |- sql/
             |- database.sql
         |- config.php
         |- aliments.php
         |- ...
         |- tests/
         |- README.md
```

Travaux à réaliser :

— `sql/database.sql` Script SQL de création des tables (`create table ...`) avec insertion des données (`insert into ...`). On doit pouvoir exécuter ce script dans PhpMyAdmin pour construire toutes les tables et insérer les tuples. Votre base devra stocker les aliments, les entrées du journal, ... Il est impératif que votre site contienne des données de tests. Pour les aliments, vous pouvez par exemple facilement extraire une cinquantaine d'aliments d'OpenFoodFacts, les télécharger au format csv et les importer dans votre base. Vous ajouterez également un ou plusieurs utilisateurs de test, des entrées dans leur journal afin que les indicateurs que vous aurez implémentez soient bien affichés sur leur page d'accueil.

— `README.md` description de votre API REST c'est-à-dire les endpoints, les méthodes HTTP attendues (GET, POST, PUT, DELETE, ...), les paramètres et les résultats attendus JSON ou erreurs de manière similaire à Punk API (https://punkapi.com/documentation/v2). Votre API REST doit permettre de gérer toutes les actions CRUD (Create, Read, Update, Delete) des entités de votre application (e.g. aliment, entrée journal, ...) dont les valeurs seront stockées dans votre base de données. Votre API REST utilisera le format JSON pour retourner les résultats (cf. Figure 2).



Action	HTTP	Payload	URL	Description
Create	POST	json	/<resource>	Create an entity represented by the JSON payload.
Read	GET	-	/<resource>	Get all entities from the resource.
Read	GET	-	/<resource>/<id>	Get a single entity.
Update	PUT	json	/<resource>/<id>	Update an entitiy with the JSON payload.
Delete	DELETE	-	/<resource>/<id>	Delete an entity.

Prefix all URLs with your unique REST endpoint
<resource> can be any REST resource name
<id> gets automatically generated for every entity you create

FIGURE 2 – Exemple d'API REST. Figure extraite du site https://crudcrud.com/

— `config.php` contient des variables globales d'initialiation de votre backend. Par exemple, l'accès à la BD doit être configurable dans ce fichier et uniquement ce fichier. De même, le préfixe de tous vos endpoints doit être configurable dans ce fichier et uniquement ce fichier.
— autres fichiers PHP qui implémentent votre API
— (bonus) tests unitaires de vos endpoints

Pour le backend, vous n'utiliserez que du PHP standard i.e. aucun framework ou bibliothèque externe.

3 Jalon 2 : Le frontend

Le frontend regourpe le code HTML, CSS, JS et PHP qui permet d'envoyer la partie cliente de l'application au navigateur. Il s'agit d'un site Web classique (cf. TP 1 et 2). Toutefois, tout accès aux données devra **obligatoirement** passer par des requêtes AJAX en JS et l'API REST fournie par le backend. Dans votre projet vous aurez au minimum (ajoutez-en d'autres si nécessaire) les pages suivantes :

index qui affiche un dashboard i.e. présente les indicateurs que vous aurez choisis : consommation de sel journalière sur les 7 derniers jours, de calories, ... combien de fruits et/ou légumes par jours sur les 7 jours, ... possibilité de changer la période ... vous pouvez afficher des graphiques à l'aide bibliothèques JS (cf. TP4 exo 6)

profil page permettant de renseigner vos informations

aliments page affichant les aliments de la base avec la possibilité d'en ajouter, d'en supprimer, de les modifier, ... Attention, comme il est possible d'avoir un grand nombre d'aliments en base, il est important de prévoir une pagination.

journal page affichant son journal avec possibilité d'ajouter une entrée

Organisation des fichiers de votre projet dans votre git :

```
Projet
   |- frontend
         |- js/
         |- css/
         |- imgs/
         |- config.php
         |- index.php
         |- profil.php
```

```
|- aliments.php
|- journal.php
|- README.md
```

Travaux à réaliser :

- `README.md` description de votre frontend. Différents login/pass pour tester.
- `config.php` contient des variables gloables d'initialiation de votre frontend. Par exemple, le préfixe de tous vos endpoints doit être configurable dans ce fichier et uniquement ce fichier.
- **IMPORTANT :** l'architecture REST impose une séparation stricte entre le frontend et le backend. Vous ne devez donc **JAMAIS** inclure le code PHP du backend dans le frontend.

Pour le frontend, vous pouvez utiliser : Bootstrap CSS (https://getbootstrap.com), JQuery (https://jquery.com), Datatables (https://datatables.net). Il est par contre interdit d'utiliser un framework PHP comme Symphony ou Laravel car ils reposent sur des notions non abordées en IDAW comme MVC (cf. UV CDAW1). Il est également interdit d'utiliser un framework JS comme Vue.js ou react.

4 Améliorations Possibles

Gestion des aliments composites. Modifiez votre code pour gérer la décomposition en sous-aliments dans le formulaire de gestion de la base de données des aliments.

Proposition d'aliments pré-remplis. Lorsque l'utilisateur souhaite ajouter un aliment dans son journal, il commence par chercher un aliment parmi ceux présents dans sa base. Si l'aliment n'est pas présent, l'utilisateur se verra proposé des aliments issus de la base d'OpenFoodFacts (https://fr.openfoodfacts.org/data) via son API REST. Si l'utilisateur sélectionne un tel aliment ainsi proposé, il sera ajouté dans la base et se vera ainsi directement proposé la prochaine fois.

Multi-utilisateurs. Ajoutez la possibilité de se créer un compte sur le site, de se connecter et se déconnecter via un mécanisme de session. Bien évidemment, chaque utilisateur aura son propre journal. Par contre, la liste des aliments sera partagée.

5 Rendu

Le rendu de votre site est constitué par : le code sous git, un email, une vidéo de 3min max de démonstation de votre site.

Organisation attendu des fichiers de votre projet dans votre dépôt git :

```
IDAW
  |- Projet/
    |- backend/
        ...
    |- frontend/
        ...
```

Template du mail de rendu à m'envoyer :

```
Sujet : [IDAW Projet] xxx
Contenu :
   Membres du groupe : XXXXXX.XXXXXX et YYYYYYY.YYYYYYY
   URL du dépôt git : http://github.com/ZZZZZZ/IDAW.git

   <décrire les points notables de votre projet si nécessaire>

   Vidéo : URL / lien de partage
```

6 Procédure de test de votre projet

```
git clone  http://github.com/XXXXXX/IDAW.git NOM_Prenom
cd NOM_Prenom

Configuration :
    1. import de votre base et de vos données via le fichier backend/sql/database.sql
        dans PHPMyAdmin local
    2. édition des identifiants de connexion à la BD dans backend/config.php
    3. édition du prefixe de l'URL de l'API REST http://localhost/NOM_Prenom
        dans backend/config.php et frontend/config.php

Test de votre site via l'URL http://localhost/NOM_Prenom/Projet/frontend/
```