Rapport Projet Web

Lundi 12 Novembre 2018

Yoan Rock - Axel Morvan - Olivier Marie-Louise
Adèle Wagner - Jérôme Hennin - Mehdi Khelifa

Sommaire

Présentation de Phat'Advisor

Contexte

Dans le cadre d'un projet web, il nous a été demandé de réaliser une application web. Aucun sujet n'était imposé afin de favoriser les sujets libres. Cependant, plusieurs exigences ont été exprimées (cf. Présentation du projet).

Parmi les divers sujets proposés lors de la première séance, il y en a un qui a retenu notre attention, Phat'Advisor était né.

Objectifs

L'objectif principal de Phat'Advisor est de répondre aux questions que les étudiants de l'ENSIIE se posent tous les jours : "Qu'est-ce qu'on mange ? Où est-ce qu'on mange ?".

Pour répondre à ce besoin nous avons choisis de fournir aux étudiants de l'ENSIIE une application centralisant l'ensemble des endroits où manger, qui sont proche de l'école, et indiquant les offres étudiantes et les partenariats du BDE.

L'intérêt pour l'équipe du projet, est de pouvoir livrer cette application au BDE afin qu'elle soit utilisable par les étudiants mais également par le personnel travaillant à l'ENSIIE.

Fonctionnalités

Phat'Advisor, en plus de centraliser et de recenser les restaurants et leurs offres étudiantes, dispose de plusieurs fonctionnalités afin de parfaire l'expérience de l'utilisateur.

Une fois connecté, l'utilisateur, a la possibilité de filtrer la liste des restaurants selon plusieurs critères (badges ou catégories), ou bien sélectionner un restaurant.

Les badges et catégories représentent un moyen ludique d'effectuer une recherche depuis l'application. Les catégories servent à différencier les différents types de restaurants, selon ce qui est servi (type de nourriture) ou encore selon le type d'endroit (restaurant, stand). Les badges ajoutent un moyen de "récompenser" un restaurant, ce sont des facteurs de confiance. Ils peuvent indiquer l'origine de la nourriture (Halal, Végétarien, Bio), le service assuré (livraison, à emporter, commande par téléphone) ou encore les offres étudiantes et les partenariats du BDE.

Pour chaque restaurant, il est indiqué :

- Une note globale
- Les commentaires des utilisateurs et leur note
- L’adresse du restaurant,
- Une description
- Les badges et les catégories

Présentation de l'équipe

Les membres

Le projet a été développé par une équipe de six personnes :

- Yoan ROCK
- Axel MORVAN
- Olivier MARIE-LOUISE
- Adèle WAGNER
- Jérôme HENNIN
- Mehdi KHELIFA

Répartition du travail

Modèle MVC

Lors de la phase conception du projet, nous avons ensemble défini les différentes user stories afin de faire apparaître les tâches à effectuer. Pour chaque tâche, nous avons réfléchi aux besoins nécessaires pour sa réalisation en terme de Modèle, Vue ou Contrôleur. Selon les affinités techniques de chacun, nous nous sommes divisés en trois groupes de deux personnes. Ces trois groupes correspondent au Back, au Front et à la base de données.

Back

Yoan et Olivier ont travaillé sur le Back, la mise en place des sessions, et les redirections en cas d'erreur sur le site. Ils se sont réparti le travail de façon égale dès le début et, si une tâche était trop complexe et avait une probabilité élevée d'engendrer des conflits, ils l'ont l'accomplis en pair programming à l'école.

Front

Axel et Adèle se sont occupés du Front.

Adèle a également réalisé les différents visuels et les icônes de l'application (cf. exemples), du carrousel et ainsi de la plupart des formulaires du site.

Axel s'est occupé de certaines vues du site et des scripts js pour rendre le site "user-friendly"

BD et Configuration (certificats, yaml)

Après un recensement des besoins de l'équipe, Jérôme assisté de Mehdi, a réalisé la base de données.

Mehdi a également pris en charge la partie configuration du projet sur les points suivants:

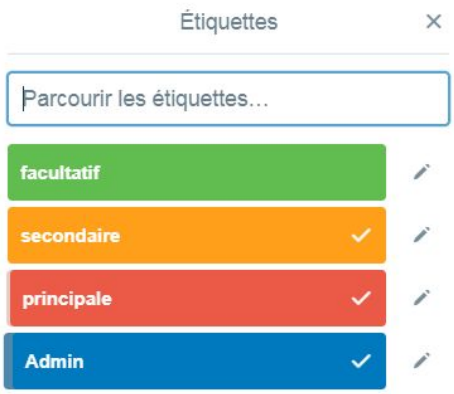
- Modifications du Makefile

- Génération de certificats auto-signés pour le https et modification du Dockerfile nginx (nécessaire pour avoir accès aux maps de Google) et du docker-compose
- Création de l'API

Déroulement du projet

★ Nous avons trié l'ensemble des key-users sous forme de tâches dans le backlog (colonne "Todo") du trello.

★ Ensuite, nous avons assigné des étiquettes aux différentes tâches (celle-ci pouvait avoir plusieurs étiquettes)
Les étiquettes étaient les suivantes :



à implémenter après le rendu du projet

à implémenter pour le rendu du projet

à implémenter pour la présentation

fonctionnalité lié aux administrateurs du site

★ En parallèle, nous avons mis au point un diagramme de cas d'utilisation et un diagramme de classes représentant la base de données le plus fidèlement.

★ Ensuite, les personnes se sont regroupées par "sous-services MVC" pour choisir et donc s'affecter aux différentes tâches du trello. La période de développement débuta à ce moment.

★ Au début de la période de développement, il n'y avait pas encore d'html, de css ni de js disponible et utilisable. "l'équipe back" a donc d'abord implémenté des scripts php simple retournant des tableaux de données. Ces scripts étaient couplés avec des pages fronts basiques sans aucun css ni js. Cela a permis à Yoan et Olivier de tester directement leurs scripts php sans dépendre des autres équipes.

★ L'équipe Modèle a dès le début implémenté presque en totalité la base de données, cependant si l'équipe back ou l'équipe front avaient remarqué des points d'améliorations ou manquants, on envoyait un message sur le groupe discord (exemple : oublie d'un

NOT NULL dans le champ d'une table ou taille maximum de l'URL du site web d'un restaurant trop faible).

Choix effectués

Nous avons décidé de rendre notre application web robuste. C'est-à-dire que les vérifications des champs et des accès aux pages (droits utilisateurs/ utilisateur non connecté) sont vérifiés et implémentés côté client et serveur pour éviter les problèmes.

Pour permettre un code plus propre et un site fonctionnel, nous avons pris la décision (la dernière semaine) d'implémenter moins de fonctionnalités mais d'améliorer la qualité de celles déjà existantes. Nous avons donc mis de côté les fonctionnalités facultatives et les secondaires les moins importantes

Dès que l'équipe Front a terminé les pages les plus importantes du projet, nous avons fait une réunion sur discord pour changer notre façon de s'échanger les informations entre les vue et les contrôleurs, désormais il a fallu changer et réécrire les scripts pour qu'ils renvoient du JSON que la vue pourrait interpréter directement dans du JS.

Pour améliorer l'expérience utilisateur et éviter les rafraîchissements de page, on a décidé de se rapprocher du principe "Single Page application" nous avons utilisé beaucoup d'ajax.

Architecture de l'application

Base de données

Afin de créer un modèle de données cohérent, nous avons effectué un brainstorming autour des fonctionnalités principales du projet. Il en est ressorti différents points que l'on a pu regrouper afin de former des tables et des associations qui permettent d'être utilisés facilement par le groupe Back-Office. Nous nous sommes alors attelés à créer un Modèle conceptuel des données avec le logiciel Jmerise (cf. Annexe).

Notre modèle de données se constitue de quatre tables principales (Restos, Favoris, Persons et Comments). Les tables Catégories et Badge ont été rajoutées dans un second temps afin de permettre un tri plus simple des restaurants sur la page d'accueil du site. Les tables « Cat-Resto » et « Badge_Resto » sont quant à elle des associations qui permettent le lien entre la table Restos et les Tables Catégories et Badge.

L'ensemble de cette base de données est codée dans le fichier phat-advisor.sql qui se situe dans le dossier data du projet. Ce dossier contient également des données qui permettent au projet de tourner dès la fin de l'installation du conteneur du projet.

API

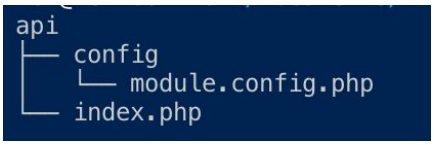
Objectifs

Dans un souci de modularité et de séparation des responsabilités techniques, nous avons opté pour la mise en place d'une API REST. La philosophie générale visée était la suivante:

- Le backend en PHP fait les calculs, fournit la liaison avec la base de données et renvoie les résultats en JSON
- Le frontend va lui présenter ces résultats à l'utilisateur
- l'API doit pouvoir être développée en parallèle de l'existant et sans impact.

Fonctionnement

Une route a été ajoutée côté NGINX afin de ne pas modifier le router de base
L'API est accessible via https://localhost:8000/api/.
Les accès sont gérés par le fichier index.php



Les requêtes peuvent être de ce format:

- GET Récuperation un Element /api/{entity}/{id}
- GET Récupération Collection /api/{entity}
- POST Creation d'Elements /api/{entity}
- DELETE Effacer Element /api/{entity}/{id}
- PUT Modifier un Element /api/{entity}/{id}

Au départ, le index.php se voulait générique mais intimement lié à l'arborescence du projet. L'inconvénient majeur est le manque de flexibilité. Si par exemple, il était devenu nécessaire de revoir l'arborescence du projet, l'API aurait subi de lourdes modifications.
Il a donc été décidé d'apporter une couche d'abstraction via un fichier de configuration (/api/config/module.config.php). Et ce fonctionnement cumule des avantages non négligeables:

- plus de clarté dans le fonctionnement de l'API.
- Si un autre membre du projet a besoin d'ajouter des éléments à l'API, il peut désormais le faire via ce fichier.
- Facilement sécuriser l'API en limitant les méthodes HTTP ou en filtrant certaines données, et ce, par entité

Le fichier de configuration (extrait):

```
"users" =>[
    "entity"           => Entity\User::class,
    "hydrator"         => Hydrator\User::class,
    "repository"       => Repository\User::class,
    "service"          => Service\User::class,
    "api-methods"      => ['GET','POST','PUT','DELETE'],
    "POST-action"      => "../public/user-register.php",
    "PUT-action"       => "../public/user-update.php",
    "GET-hidden-fields" => ["secret_user"]
],
```

On a ici la route /api/users qui donne accès à l'entité **User** (et les éléments associés hydrator, repository).
Lors de la récupération de données (**GET**), le mot de passe (*secret_user*) ne sera pas renvoyé dans le JSON.
Les responsabilités de création (**POST**) et de mise à jour (**PUT**) ont été laissées aux fichiers php initiaux créés par l'équipe BACKEND.
On a donc réutilisé le travail qui a été fait en attendant que l'API soit pleinement opérationnelle.

Mise en oeuvre

Une fois l'API réalisée, les modifications ont majoritairement été apportés sur les fichiers JS qui gèrent les pages de formulaire. Les submit sont gérés par des call AJAX, il a suffi de changer l'URL du call AJAX.
L'équipe BACKEND a pu être autonome sur le reste du projet quant à l'enrichissement de l'API.

Outils utilisés

PostMan: Tests de l'API

Problèmes rencontrés

Par manque de temps, nous n'avons pas eu le temps d'implémenter certaines de nos fonctionnalités que nous jugions comme secondaires mais nécessaires, par exemple :

- Connexion via Arise
- Système de suggestions
- Envoi de mails (pour les suggestions, la validation des commentaires, etc ..)
- Visualisation d'un menu d'un restaurant

De plus, nous avons réalisé la majorité des tests de notre site sur un seul navigateur : Google Chrome.

Et nous nous sommes rendus compte que notre site présentait quelques problèmes lorsqu'on passait sur d'autres navigateurs. Nous n'avons pas encore géré cette compatibilité au niveau du CSS et de certaines fonctions JavaScript.
Pour l'instant, nous recommandons d'utiliser Google Chrome ou la version libre Chromium pour utiliser notre site, mais ce défaut d'interopérabilité entre navigateurs est un des problèmes majeurs à résoudre pour une future version de notre site web.

Solutions apportées

Afin de palier au problème des notes des restaurants, nous avons décidé de créer un trigger qui a pour but de mettre à jour la note d'un restaurant grâce aux notes des utilisateurs de l'application.

Ce trigger s'exécute comme suit après l'ajout d'un commentaire par un utilisateur :

- Récupération de la note actuellel du restaurant
- Recherche du nombre de notes sur le restaurant
- Calcul de la nouvelle note
- Modification de la note du restaurant
- Ajout du commentaire dans la base

Conclusion

Dès l'expression des users stories nous savions que certaines fonctionnalités ne pourraient pas être implémentées pour le rendu. Puisqu'à terme nous souhaitons pouvoir fournir l'application, nous avons réfléchi à des fonctionnalités permettant aux utilisateurs de se contacter via l'application et de communiquer par chat. L'objectif, est que les utilisateurs puissent réaliser des commandes et des départs groupés. Par ailleurs, pour aller beaucoup plus loin, nous avons également évoqué l'idée d'analyser les choix des utilisateurs afin de mettre en avant les restaurants les plus sollicités.

Enfin, nous prévoyons de faire une refonte graphique afin de donner à notre application le pep's dont elle a besoin.

Annexes

Exemples de Badges et Catégories

Figure : Badges "Partenariat BDE" et "Halal"

Figure : Catégories "Burgers" et "Pizza"

Base de données

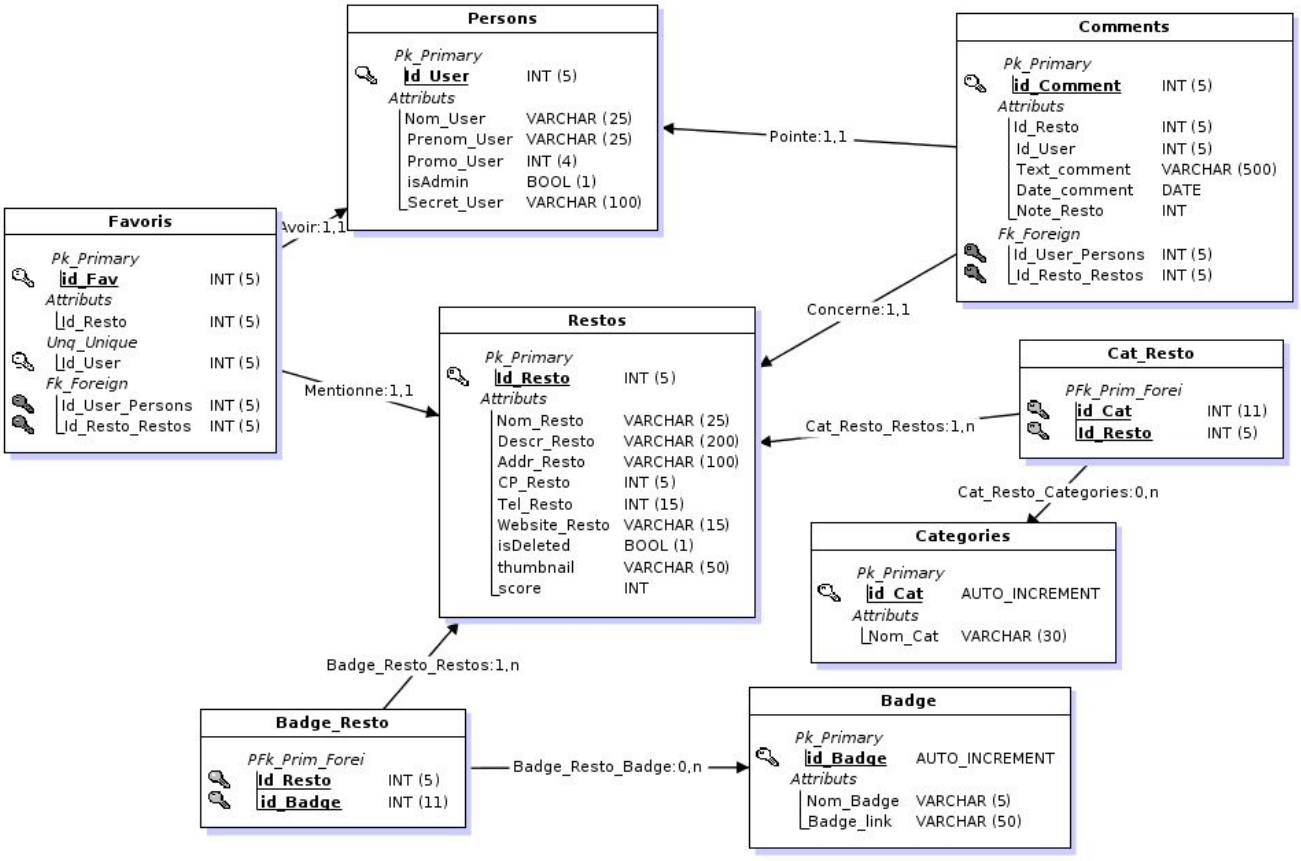


Figure : Modèle de la base de données