EPICEVRY

Mathieu TRUC, Loic MARY, Manal KOURI, Bryan ROY

12 mai 2020

Table des matières

1 Introduction

Motivés par l'ouverture de l'épicerie solidaire étudiante de l'université d'Evry, nous souhaitons développer le site web "EpicEvry", permettant aux étudiants de commander leurs aliments en ligne et de les récupérer à un drive de l'épicerie.

Nous aimerions permettre aux utilisateurs d'échanger aussi sur le thème de l'alimentation via notre plateforme. En effet, les différents utilisateurs pourront publier des recettes.

2 Installation du projet

2.1 Makefile

Le Makefile dispose de 4 commandes :

- make start qui permet de lancer le serveur sur *localhost:8080*
- make db.init qui initialise la db a partir du fichier *./data/init.sql*
- make db.drop qui détruit la db
- make db.reset qui détruit puis initialise la db

2.2 Accès à la base de données

- Pour que le code applicatif accède à la db il faut changer les paramètres dans le fichier *./src/config/config.php*.
- Pour que le makefile se connecte a la db il faut changer les paramètres directement dans le fichier *Makefile*.

3 Approches importantes

3.1 Structure du site

La page d'accueil est le fichier *index.php*. En haut figure un menu avec les onglets suivants:

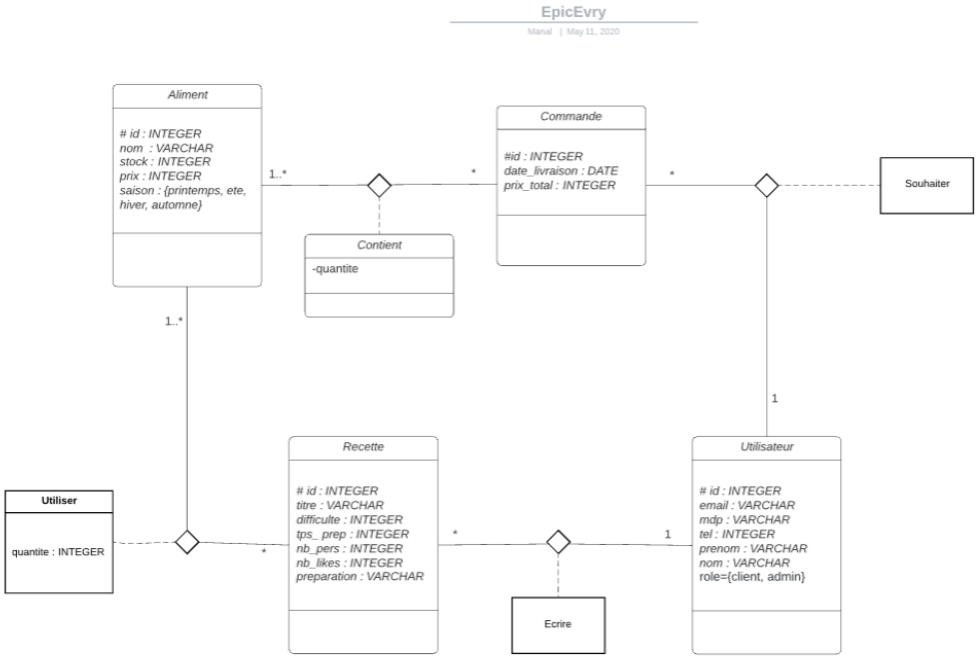
- Accueil : pour revenir à la page d'accueil
- A propos (*page_ presentation.php*) : pour avoir des informations sur l'épicerie
- Aliment (*catalogue_ accueil.php*) : c'est le catalogue des différents produits proposés par EpicEvry avec leurs prix, saisons et types. Il est important de noter qu'on ne peut pas ajouter un article au panier si on n'est pas connecté. C'est la raison pour laquelle il n'a pas l'option "ajouter au panier" dans la page Aliment de l'accueil
- Connexion (*page_ connexion.php*)
- Recettes (*page_ recette.php*) : accessible à tout le monde, cet onglet permet de voir les différentes recettes publiées par les utilisateurs
- Contact (*contact.php*) : pour poser des questions ou donner des conseils afin d'améliorer le mode de fonctionnement de l'épicerie.

Il y a 2 types d'utilisateurs : clients et administrateur .
Le client peut commander et choisir la date de livraison. De plus , il peut écrire une recette et la partager sur le site.
L'administrateur quant à lui a les mêmes droits que le client avec en plus la possibilité de suivre toutes les commandes de ces derniers.

3.2 Structure de la base de données

Nous avons choisi d'utiliser une base de données répartie sur les tables suivantes:

- Aliment(#id, nom, stock, prix :, saison : Ete, Printemps, Hiver, Automne) : contient la liste des produits et leurs caractéristiques
- Commande(#id , date_livraison, prix_total, id_client ⇒ utilisateur.id): contient les informations sur les commandes
- Recette(#id, titre, difficulte, tps_prep, nb_pers, nb_like, id_auteur ⇒ utilisateur.id): contient l'ensemble des informations sur les recettes publiées par les utilisateurs
- Contient(#id_commande ⇒ commande.id, #id_aliment ⇒ aliment.id, quantite : INTEGER)
- Utilise(#id_recette ⇒> recette.id, #id_aliment ⇒ aliment.id, quantite)
- Utilisateur(#id, email, mdp, role=client, admin, tel, prenom, nom) : contient tous les utilisateurs et leurs informations personnelles



3.3 Structure des objets php

Tous les clients enregistrés ont accès après connexion à 3 nouveaux onglets “Panier” “Mon Profil” , “Déconnexion”. De plus, il est maintenant possible d’ajouter un article du catalogue dans le panier et de consulter ce dernier. Il est également possible de choisir la date de livraison. De plus, grâce à l’onglet “Profil”, l’utilisateur peut modifier les informations relatives à son profil (nom,prénom,mot de passe, etc.) et il peut surtout consulter la commande en cours et la supprimer si elle ne lui convient pas. Enfin, un client peut publier une recette qui sera accessible par n’importe quel utilisateur connecté ou pas.

3.4 Fonctionnalités pour l’administrateur

L’administrateur a accès aux mêmes fonctionnalités qu’un client. Mais il a en plus, la possibilité de voir et de manipuler l’ensemble des commandes effectuées par tous les clients, grâce à l’onglet “gestion des commandes”.

4 Organisation interne du groupe

Nous avons mis notre projet sur GitLab et chacun a créé une branche git pour effectuer des modifications de notre côté. Régulièrement nous faisions des fu-

sions de nos branches pour rassembler notre travail.

Nous nous sommes aussi répartis les tâches, Loïc et Mathieu ont plus travaillé sur le Backend tandis que Manal et Bryan sur le Frontend.

Voici donc ce que chacun a fait :

- Bryan s'est tout d'abord occupé du profil utilisateur et de son édition en produisant le code des pages *userprofile.php* et *userprofileedition.php*. Il a ensuite configuré le css du corps de certaines. Il a veillé à la cohérence de la mise en page du site (harmonisation du fond, des tableaux, des boutons et des différentes parties de texte) pour produire un visuel esthétique optimal. Il est également à l'origine du logo d'EpicEvry. Sur le plan SQL, Bryan a initié la conception de la base de données en implémentant les tables relatives aux utilisateurs et aux commandes.

- Mathieu s'est occupé de créer les objet php UserRepository, User, Commande Repository, Commande et de l'une partie de Aliment et AlimentRepository. Il a établit le système de session et le corps des formulaires d'inscription et de connexion. Il a aussi participé au profil utilisateur et de son édition, notamment pour permettre une bonne interaction avec la base de données ainsi que l'affichage du corps recettes et commande de l'utilisateur sur cette page. Il s'est ensuite chargé du corps des pages de gestion de commandes administrateurs, ainsi que de la page de détail des commandes. Il a aussi fait les corps des pages de publication de recette, ainsi que la partie de concrète du passage de commande après avoir remplis un panier utilisateur.

- Manal, de son côté, s'est occupée de la création des bases de données relatives aux aliments et aux recettes dans le *init.sql* . Ceci avant de contribuer à la partie visuelle du site en produisant, entre autres, le header et le footer utilisés sur toutes les pages,la page d'accueil *index.php*, la page de présentation *page_ presentation.php* et la page de contact *contact.php*. Elle a aussi veillé, avec Bryan, à assurer un aspect unifié pour le site, et ceci en fournissant des modèles facilement adaptables pour les pages de styles et les structures utilisées (tableaux, par exemple) et corrigeant les éventuelles incohérences.

- Quant à Loïc, il s'est chargé de mettre en place le catalogue de l'épicerie et le panier. En effet, il s'est occupé de l'affichage dynamique de l'ensemble des articles et de l'ajout des articles au panier, de la suppression de produits si l'utilisateur le souhaite. Il a notamment implémenté des fonctions dans *AlimentRepository.php* (add , fetchAll_panier et delete) et il a créé la classe panier dans le fichier *panier_ class.php* De plus, il a légèrement modifié les données que l'on entre dans la table Aliment pour que les photos des articles collent parfaitement avec le nom de l'article. En plus, il a modifié le visuel du panier.

5 Aspects techniques

5.1 Système de session

La table des utilisateurs représente les comptes auxquels nous pouvons nous connecter sur le site. Le système de connexion au compte se base sur les sessions php. Le fichier *connect.php* vérifie que le nom d'utilisateur entré dans le formulaire de connexion correspond à celui d'un utilisateur dans la base de données, puis vérifie que le mot de passe entré correspond bien à celui dans la base de donnée. Si les deux conditions sont vérifiées alors le fichier initialise une session php en initialisant l'id de l'utilisateur, son pseudonyme ainsi que son rôle (1 pour administrateur et 0 pour utilisateur). Les autres pages peuvent alors détecter si un utilisateur est connecté en utilisant session_start() et en vérifiant si $_SESSION['id'] est initialisé. La page de connexion *page_ connexion.php* est une page de formulaire classique qui récupère les éventuelles erreurs de *connect.php* pour afficher le bon message d'erreur à l'utilisateur. On utilise la validation de formulaire d'HTML5 pour vérifier que les champs sont non vides. La page n'est accessible que si l'utilisateur n'est pas déjà connecté.

5.2 Système d'inscription

La page d'inscription *page_ inscription.php* est très similaire à *page_ connexion.php*. *signup.php* est le script php qui ajoute l'utilisateur dans la base de données. La classe UserRepository est la classe gérant l'interaction entre les contrôleurs et la base de données des utilisateurs. Grâce à la classe UserRepository on peut facilement interagir avec la base de données depuis le script. La fonction addUser de UserRepository permet d'ajouter un utilisateur à la base de donnée grâce à un INSERT en s'assurant que le nom d'utilisateur n'est pas déjà pris et retourne une exception dans le cas contraire.

5.3 Profil utilisateur

Nous avons d'abord décidé que les profils utilisateurs pouvaient être accessible à tous les utilisateurs du site. C'est pourquoi dans la page *userprofil.php* qui affiche les profils des utilisateurs, c'est la variable $_GET['id'] qui permet de choisir le profil de l'utilisateur dont correspond l'id dans la base de données. Cependant les commandes de l'utilisateur ne sont visible que si c'est l'utilisateur qui est connecté consulte son profil sur la page *userprofile.php*. On peut aussi modifier son profil et publier une recette de cette page. Cette page récupère les recettes et commandes associée à l'utilisateur, en fonction de quel utilisateur consulte le profil, via les objets AlimentRepository et RecetteRepository et les affiche. Pour consulter ses commandes un utilisateur peut cliquer sur détails qui l'amène sur la page *detailscommande.php*. Cette page récupère et affiche les informations générales de la commande, ainsi que les aliments commandés via les objets AlimentRepository et CommandeRepository. UserRepository permet de savoir qui visite la page.

5.4 Modification du profil

La modification de profil se fait via *userprofileedition.php*, cette page fait à la fois les modifications dans la base de données via l'objet UserRepository et sert de formulaire pour entrer les modification. Le formulaire fait appel à la même page qui s'occupe d'effectuer les modifications si besoin. Cette page n'est accessible que si l'utilisateur est connecté et la modification se fait sur l'utilisateur connecté.

5.5 Gestion des conmmandes

La gestion des commandes est possibles de deux manières différentes. Soit l'utilisateur est un administrateur, dans ce cas il a accès à la page *page_ admin_ commande.php*. Soit l'utilisateur n'est pas administrateur et il n'a donc accès qu'à la gestion de ses commandes via son profil utilisateur. La partie sur le profil utilisateur est détaillé en 3.3. Lorsque qu'un administrateur est connecté il à accès à la page *page_ admin_ commande.php*. Cette page récupère toute les commandes en cours sur le site via les objet CommandeRepository et UserRepository pour afficher les clients. L'administrateur peut afficher les détails de la commande de la même manière que décrit en 3.3 via detailscommande.php. Et il peut également supprimer les commandes.

5.6 Catalogue

Le catalogue utilise la table "Aliment". Il faut distinguer deux catalogues qui sont implémentés par deux fichier différents le premier figure sur la page d'accueil et il ne possède pas l'option "ajout au panier" (*catalogue_ accueil.php*) le deuxième présente la dite option et est accessible lorsque l'on est connecté (*catalogue.php*) Les deux catalogues présentent l'ensemble des produits que notre épicerie propose. Il s'agit donc de récupérer le nom des produits, une image (qui ne figure pas dans la base de données) , le prix , la saison, le type et le stock. Le fichier *panier_ class.php* permet d'initialiser la session et d'entrer les différents id des produits dans le tableau $_SESSION['panier']. Puis grâce à une fonction fetchAll() implémentée dans *AlimentRepository.php* un parcours du tableau et des fonction Get, l'ensemble des informations sur les produits s'affiche sur la page. Concernant l'affichage des images pour les aliments, nous avons nommé le fichier comme suit *Id_ de_ l'_ aliment.png*, ainsi, en récupérant l'id du produit, la photo correspond bien au produit. Puis, pour l'option "Ajout au panier", lorsque l'on clique sur l'image une fonction Javascript (dans app.js) se réfère à la fonction *addPanier.php* qui ajoute dans un tableau l'id et et le nom du produit sélectionnée. De plus, la bibliothèque JQuery présente la fonction Javascript permet d'afficher une fenêtre pour demander à l'utilisateur s'il souhaite consulter son panier.

5.7 Panier

Le fichier *panier.php* permet de visualiser les produits présents dans le panier, de voir le total à payer , de connaître la quantité de chaque article et de supprimer un article si l'envie nous prend. La démarche est assez similaire que celle expliquée dans le catalogue. On récupère l'id des articles présents dans le tableau renvoyé par la fonction add et qui figure dans $_SESSION. Puis on récupère les différentes fonctions de ces produits avec les Getters. La fonction total() de *panier_ class.php* permet d'obtenir le prix total du panier. Ensuite concernant la suppression d'article, le fait de cliquer sur l'image de corbeille enclenche la fonction del () de *panier_ class.php* qui décrémente de un la quantité de l'article. De plus, en revenant sur la page catalogue de l'utilisateur, le stock des aliments s'actualise selon les quantités prises dans le panier.

5.8 Commandes

Une fois le panier remplis, un champs de date est remplis par l'utilisateur dans la page panier.php. Le bouton ajouter commande fait appel à *addCommand.php* qui est le fichier qui récupère via la méthode POST et le $_SESSION['panier'] les informations de la nouvelle commande. Via l'objet CommandeRepository le script ajoute la commande dans un premier temps dans la base de données, puis récupère tous les aliments du panier pour les insérer dans la relation Contient. On vide ensuite le panier en effaçant $_SESSION['panier'].

5.9 Visualisation des recettes

Pour assurer que l'utilisateur ait accès, de manière claire, aux données relatives à chaque recette, on a opté pour un affichage simple et épure. En effet, pour chaque recette, on récupère le titre dans un premier temps, ensuite on regroupe dans un tableau certaines informations sur la recette: le niveau de difficulté, le temps de préparation et le nombre de portions. Un second tableau juste après sert à présenter les ingrédients qu'on peut trouver à EpicEvry et qui figurent dans la recette ainsi que leurs quantités, et à la fin on trouve les étapes de présentation de chaque recette. Ces données proviennent de la base de données et sont récupérées au besoin à l'aide de requêtes php. On remarque aussi que les recettes ajoutées par les utilisateurs eux-même figurent instantanément sur la page de recette juste après leur ajout, étant donné qu'elles sont aussitôt stockées dans la base de données.

5.10 Publication d'une recette

Un utilisateur connecté peut décider de publier une recette via un lien sur son profil. L'édition de la recette se fait via la page *recetteedition.php*. Cette page contient un formulaire permettant de renseigner les informations relatives à la recette. Ensuite via un POST on transmet ces information à la page *addrecette.php* qui va ajouter la recette dans la base de donnée via l'objet RecetteRepository

puis transmettre les informations de la recette nouvellement crée via un POST qui s'envoie automatiquement via un formulaire javascript. La page *legumeedition.php* permet d'ajouter la liste des légumes de la recette présents dans le catalogue du site. Un formulaire demandant le nom de l'aliment à ajouter à la recette envoie l'information à la page *addlegume.php* via un POST. Cette page s'occupe d'ajouter l'ingrédient à la base de donnée via l'utilisation des objets RecetteRepository et AlimentRepository. Elle renvoie ensuite les informations sur la recette via un POST automatiquement à la page *addlegume.php* pour que l'utilisateur puisse ajouter d'autres légumes.

6 Difficultés rencontrées et solutions

- Beaucoup de nos fonctionnalités requièrent des interactions avec la base de données. Il fallait donc trouver un moyen efficace pour gérer ces interactions. Nous avons donc décidé de faire les différentes classes php et Repository pour pouvoir gérer ces interactions dans des fichiers séparés.

- Une autre difficulté était de pouvoir correctement supprimer les éléments de notre base de données, par exemple la table commande est référencée comme clé étrangère dans la table contient donc pour pouvoir supprimer une commande nous avons décidé de supprimer la relation où apparaissait la commande avant de supprimer la commande de la table "Commande". Cela était fait automatiquement dans le CommandeRepository.

- La publication de recette et les nouvelles commandes étaient compliquées car il fallait d'abord créer la bonne recette ou commande, puis ajouter dans les relations Utilise les ingrédients de la recette, et dans Contient les ingrédients de la commande. Pour les commandes nous avons pu les faires rajoutée automatiquement dans *addCommande.php* grâce à la structure du panier. En revanche pour les recettes nous avons du faire une page séparée qui ajoute au fur et à mesure les aliments que les utilisateurs veulent dans leurs recettes.

- Ainsi, le fait de récupérer les données de recettes par requêtes php était un peu contraignant au niveau de la mise en page. Ceci nous a en partie empêché d'utiliser des méthodes d'affichage plus épurés. Par exemple, Pour la préparation de chaque recette figurant dans la base de données, nous n'avions pas pu le faire sous forme de liste personnalisable, étant donné que toutes les parties étaient récupérées en une fois de la base de données.

- Une autre difficulté que nous avons rencontré fut au niveau du header. Quand un utilisateur est connecté, on remarque l'apparition de nouveaux onglets dans la barre du header, ce qui dérègle les paramètres d'affichage des onglets, et fait en disparaître certaines. Ainsi pour pallier à ce problème nous avons utilisé un système de header conditionnel pour gérer différemment les caractéristiques. Quand un utilisateur est connecté, on fait

appel à une page dont le style est géré par la page *header_ connected.css* et qui, donc gère les icônes de manière à ce qu'on puisse toutes les voir. Et quand le visiteur du site est déconnecté, il apperçoit un header dont la page de style est *header_ disconnected.css* dont les attributs des sections diffèrent à certains endroits de ceux de l'autre fichier. Ceci étant fait, le grand nombre d'icônes en cas de connexion nous a empêché de les afficher bien alignées avec le logo comme en cas de déconnexion, et selon les navigateurs utilisés, une petite différence peut être visible au niveau de la hauteur de la ligne des icônes. En particulier pour les navigateurs Firefox et Edge , nous n'avons pas noté de problèmes. Par contre, ce n'est pas le cas pour Chrome et Safari où de petits dysfonctionnements sont visibles.

- Aussi, nous avons essayé au maximum d'avoir un site responsive qui s'adapte à la taille d'écran utilisé lors de l'affichage. Nous avons réussi à le faire sur une grande partie des pages ("Contact" ou "A propos", entre autres) et aussi sur le header et footer de toutes les pages.