École Nationale Supérieure d'Informatique
pour l'Industrie et l'Entreprise

ENSIIE-Évry

Projet Web 2019-2020

Nicolas MIR, Terence NGO, Aymeric KEYEWA

12 mai 2020

Rapport du projet de Web

Nicolas MIR, Terence NGO, Aymeric KEYEWA

ENSIIE, Évry

Table des matières

1 Introduction

Un fan de films, séries ou livres peut facilement être dérouté par la diversité des choix qui lui sont proposés : d'un côté les méthodes de diffusion des oeuvres se développent et se multiplient (cf streaming, youtube) et de l'autre la production d'oeuvres est toujours plus importante.

Il est donc nécessaire de remettre de l'ordre dans ces prises de décisions, mais de quelle manière ? Comment le site Medialiste peut-il répondre à ce défi ?

2 Fonctionnalités du site web

2.1 2.1 Fonctionnalités générales

- **Inscription**
  Un utilisateur peut s'inscrire en saisissant quelques informations le concernant.

- **Authentification**
  Afin d'utiliser toutes les fonctionnalités disponibles du site un utilisateur doit se connecter.

- **Compte utilisateur**
  Tout utilisateur quel qu'il soit peut modifier certaines informations qu'il a entré lors de son inscription.

- **Recherche d'une oeuvre**
  Tout utilisateur peut entrer le nom d'une oeuvre présente sur le site dans la barre de recherche puis accéder à une page contenant quelques informations essentielles sur l'oeuvre dont la note moyenne donnée par les utilisateurs sur le site.

  En bas de page se trouve un espace commentaire où les utilisateurs peuvent réagir sur l'oeuvre présentée.

- **Notation d'une oeuvre**
  Tout utilisateur peut donner une note à une oeuvre allant de 1 à 5.

- **Ajout de commentaire**
  Tout utilisateur peut ajouter un commentaire à l'espace commentaire pour donner un avis sur une oeuvre.

- **Suppression de commentaire**
  Tout utilisateur peut supprimer un commentaire qu'il a créé dans l'espace commentaire.

- **Accès aux listes d'un utilisateur**
  Tout utilisateur peut accéder aux listes d'oeuvres d'un utilisateur qui a posté un commentaire dans l'espace commentaire.

- **Signaler un utilisateur**
  Tout utilisateur peut signaler un autre utilisateur en choisissant la raison du signalement la plus adaptée parmi le spam, la vulgarité ou encore la publicité.

- **Suivre un utilisateur**
  Un utilisateur peut décider de suivre un autre utilisateur. Dans ce cas, l'utilisateur pourra avoir accès aux listes des utilisateurs qu'il suit dans son profil. A tout moment un utilisateur peut décider de ne plus suivre un autre utilisateur.

- **Gestion de listes d'oeuvres**
  Tout utilisateur peut créer des listes d'oeuvres et ajouter les oeuvres qu'ils souhaitent à ces dernières. Il peut aussi décider de supprimer des oeuvres aux listes ou bien encore de supprimer des listes d'oeuvres.

- **Ajout de like et annulation de like sur la liste d'un utilisateur**
  Tout utilisateur peut aimer la liste d'un autre utilisateur ou arrêter de l'aimer à tout moment.

- **Visualisation des listes les plus likées dans le home et l'onglet Oeuvres**
  Tout utilisateur peut accéder à un onglet présentant les listes d'oeuvres ayant le plus de likes sur le site.

- **Visualisation des oeuvres du site via l'onglet Oeuvres**
  Tout utilisateur peut accéder à l'ensemble des oeuvres présentes sur le site.

- **Visualisation des oeuvres qui vont bientôt sortir via l'onglet Agenda**
  Tout utilisateur peut accéder à l'ensemble des oeuvres sur le point de sortir d'être publié pour un livre, de sortir au cinéma pour un film...

2.2 Fonctionnalités supplémentaires pour l'administrateur

- **Visualisation des signalements**
  Tout administrateur peut voir l'ensemble des signalements.

- **Suppression d'un commentaire**
  Tout administrateur peut supprimer un commentaire qu'il juge non conforme aux règles du site.

- **Suppression du droit de parole d'un utilisateur**
  Tout administrateur peut muter un utilisateur pour le sanctionner de son comportement. Ainsi ce dernier ne pourra plus poster de commentaire ni signaler d'autres utilisateurs.

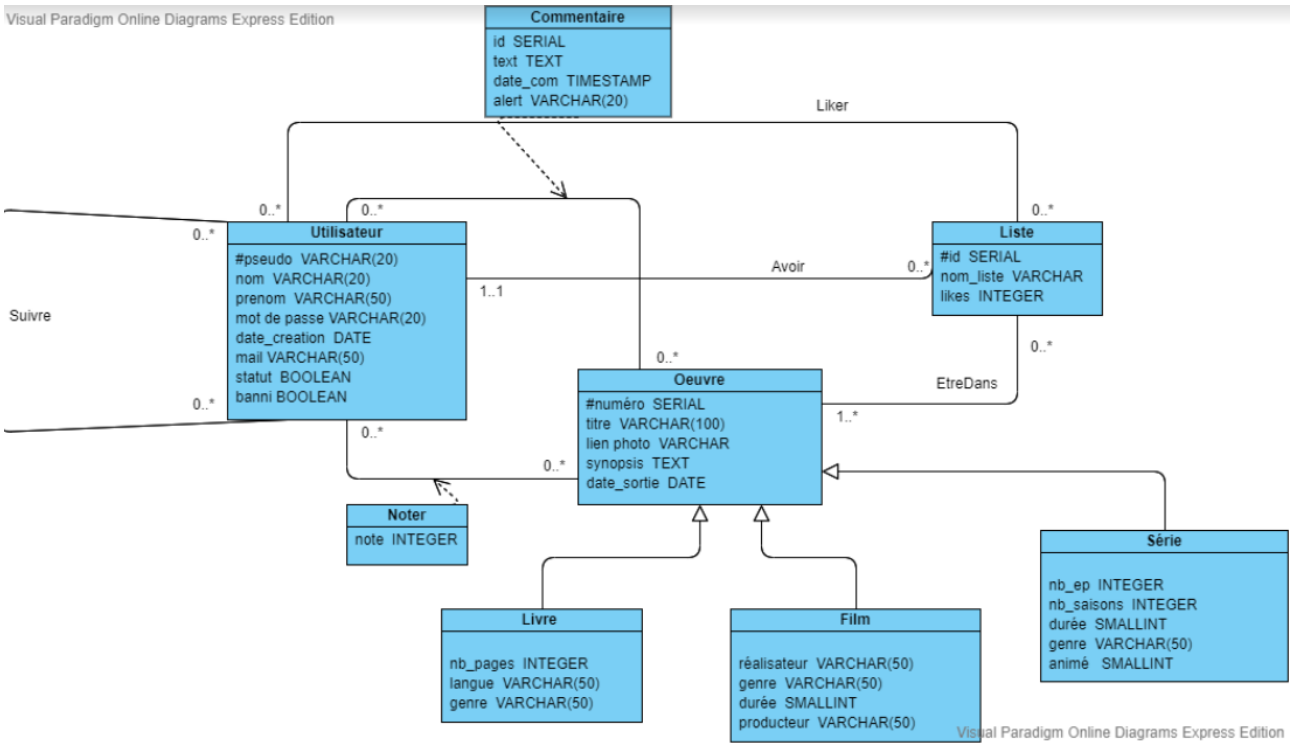
- **Redonner le droit de parole à un utilisateur**
  Tout administrateur peut redonner le droit de parole à un utilisateur.

3 Analyse et conception

Cette phase comprend les décisions de créations des classes et leurs mises en relation au vu des fonctionnalités du site.

3.1 Schéma conceptuel

Nous avons d'abord établi un diagramme UML dont voici la présentation:



- **Cardinalités**
  Les cardinalités ont été choisies de manière à restreindre le moins possible les fonctionnalités du site: 2 oeuvres peuvent par exemple porter le même nom et former 2 oeuvres distinctes, ou bien un utilisateur peut avoir 2 listes ayant le même nom.

- **Administrateurs**
  Les administrateurs sont caractérisés par l'attribut "statut" mis à true. Ils régulent l'activité du site et ont le droit de bannir des utilisateurs du forum, chez qui l'attribut "banni" est à true.

3.2 Schéma relationnel

- **Vues**
  Le schéma relationnel contient les vues de l'héritage de Oeuvre. Chacune des classes ayant un nombre non négligeable d'attributs, l'héritage se fait par ces classes filles.

- textbfRelations
  Les relations entre les tables "Avoir" et "Etre dans" du diagramme UML sont modélisés par 2 tables Ravoir et EtreDans.

- **Choix des types**
  Le type SERIAL de l'attribut id facilite les insertions d'oeuvres et de listes. Notons que les Mi_clé des classe filles sont cependant des INTEGER puisqu'ils doivent référencer les id correspondant dans Oeuvre ou Liste.

3.3 Implémentation

Le fichier init.sql du dossier data contient toutes les créations de tables et les insertions d'instances en PostgreSQL.

4 Réalisation technique

4.1 Organisation de travail

Les tâches suivantes n'ont pas été aussi délimitées, mais voici globalement ce qui ressort:

- Terence a défini et maintenu la BDD à jour, codé les fichiers de type affichage, add et delete, et développé la fonctionnalité "Suivre la liste d'un utilisateur".
- Nicolas a développé plusieurs fonctionnalités essentielles pour faire fonctionner le site: l'inscription, la connexion, la page de recherche d'une oeuvre (espace commentaire),les listes...
- Aymeric a créé la page index et travaillé la mise en forme du site via l'intégration de Font Awesome et l'implémentation de fonctions Javascript graphiques(Carousel, Menu déroulant, etc)

4.2 Chronologie

Après avoir trouvé notre BDD et implémenté l'index ainsi que divers fichiers d'affichages, nous avons mis en place les commentaires avant d'élaborer l'agenda du site. La fonctionnalité Mes-Listes s'est développée en parallèle de l'implémentation des fichiers correspondants.

Le suivi des listes s'est fait à posteriori puisque les listes devaient déjà fonctionner.

Enfin, nous avons mis en forme le site et corrigé les quelques problèmes restants.

4.3 Technologies

Nous avons utilisé PostgreSQL pour les interactions avec la base de données, HTML pour structurer les pages, PHP et JavaScript pour les rendre dynamiques et interactives, CSS pour leur mise en forme.

Pour cette dernière tâche, nous avons eu recours au CSS de Bootstrap. Nous avons également utilisé le CSS de Font Awesome pour l'intégration d'icônes.

4.4 Détails du code

Les fichiers peuvent être séparés en différents groupes types: il y a les Repository.php, les nom_classe.php, affichage_classe.php, file.css, add_instance.php ou encore les signX.php entre

autres.

Certains fichiers ne peuvent pas appartenir à une catégorie, c'est le cas par exemple de index.php ou top.php; nous les détaillerons donc à part.

- **Fichiers types de Public**
  - **index.php**
    Il s'agit du menu d'accueil su site. Il est composé d'un carousel qui affiche les n (La variable est paramétrable dans le fichier) oeuvres les mieux notées par les utilisateurs et de deux parties :
    Une partie montrant les dernières oeuvres ajoutées par l'utilisateur à sa liste Favoris (Elle est cachée s'il n'y a pas d'utilisateur connecté)
    Une partie affichant les listes crées par les utilisateurs de la plus likée à la moins likée.
  - **affichage_classe.php**
    Ces fichiers sont souvent appelés par d'autres pages pour afficher un certain contenu.

    Ce contenu est alors récupéré à partir d'une variable globale qui contient la valeur fournie par la méthode POST (depuis la page appelante).

    Les includes des Repository.php en tête de fichier servent à accéder à leurs fonctions, notamment à fetchAll, qui permet de récupérer les données à afficher.

    Un foreach est enfin utilisé pour parcourir toutes les instances récupérées, que l'on présente sous la forme d'un tableau avec ¡table¿.

    Remarque:
    Pour certains fichiers, la variable globale n'est pas donnée par POST, typiquement lorsque la page appelante réfère affichage_classe.php par un "onclick".
  - **add_instance.php**
    Ces fichiers ajoutent des instances de classe dans la BDD à partir de données entrées par l'utilisateur et récupérées par méthode POST.

    L'insertion ne se fait que si les paramètres respectent les conditions établies par la BDD (NOT NULL, taille limite), ce qu'on vérifie avec if.

    Il en est de même pour les fichiers delete.

  - signX.php
    Ces fichiers sont composés de formulaires qui demandent des informations sur l'utilisateur et dont les actions sont réalisées avec la méthode POST.

    Pour chaque formulaire on teste que les données entrées par l'utilisateur conviennent, par exemple on vérifie qu'un utilisateur entre bien un mail avec le bon format. Ses vérifications sont réalisées à l'aide de PHP, Javascript est utilisé pour mettre en valeur les champs qui ont mal été entré par l'utilisateur.

- **top.php**
  Ce fichier permet d'afficher le top 10 commu. On récupère dans $listes les listes rangés dans l'ordre décroissant (ORDER BY) du nombre de likes (DESC) en limitant leur nombre à 10 (LIMIT).

  Un foreach permet de les traiter cas par cas.

  La variable i et le test de divisibilité sur celui-ci permettent d'obtenir un affichage sur 3 colonnes: on change de ligne (class="row") toutes les 3 listes.

  En incrémentant i à chaque itération du foreach, sa valeur correspond au rang de la liste actuelle dans le top commu, ce qu'on affiche avec Rang : ¤¡= $i?¿.

  Pour chaque liste, on récupère le propriétaire $owner via une requête sur la jointure entre Liste et Ravoir au niveau de l'attribut id.

  De même, on récupère les oeuvres qui s'y trouvent en effectuant une jointure entre EtreDans et Oeuvre (sur l'attribut id), puis en sélectionnant les lignes dont l'identifiant id correspond à celle de la liste en question.

  On obtient le nombre d'oeuvres $n se trouvant dans la liste à l'aide de COUNT (sur les titres des oeuvres) pour préciser à l'utilisateur cette information. Enfin, pour

  chaque oeuvre obtenue, un fetch permet d'accéder à son attribut "lien_photo", et donc d'afficher l'image la décrivant.

- **fichiers réalisant une action**
  Ces fichiers sont appelés par des méthodes POST dans des formulaires et effectuent des requêtes sur la base de données. Par exemple, ajout-commentaire.php permet d'ajouter un commentaire dans l'espace commentaire d'une oeuvre, liker.php permet de liker une liste...

- **file.css**
  Le squelette de la plupart des fichiers CSS viennent de Boostrap. Nous avons ajouté leurs classes automatiquement créées dans nos fichiers .php.

- **Fichiers types de Source**

  - **nom_classe.php**
    On y définit des getters et setters pour les classes, car les attributs sont privés pour plus de sécurité. On implémente donc des fonctions getAttribut() et setAttribut(valeur) pour chaque attribut.

    Il faut aussi préciser un namespace en début de fichier. L'héritage de Oeuvre a été gérée avec un "extends" placé dans les fichiers des classes filles.

  - **Repository.php**
    A chaque fichier nom_classe.php est associé un classeRepository.php qui reprend le namespace du premier et définit une classe "classeRepository" dans laquelle sont implémentées des fonctions nécessaires, notamment les fetchAll, add et delete.

Ces fichiers ont en commun de posséder l'attribut dbAdapter et une fonction "construct" pour mettre en place le PDO.

Le dbAdapter est ensuite utilisé dès lors qu'une requête est nécessaire, souvent à travers "prepare" et "execute".

4.5 Difficultés rencontrées et solutions

- **PDO**
  Problèmes d'accès à la BDD dû au nombre conséquent de relations entre les classes: pour un seul test de code, il fallait penser à insérer beaucoup d'instances dépendants les unes des autres à la fois. Par exemple, l'insertion d'une nouvelle instance de EtreDans exige les existences de la liste et de l'oeuvre ciblées, ainsi qu'une instance dans Ravoir précisant que utilisateur en est le propriétaire.

- **Barres de recherche**
  Sur la page mylists.php, lorsque l'on cherchait à ajouter ou supprimer un oeuvre dans une liste, l'auto-suggestion était affichée au niveau de la barre de recherche car toutes les balises spans avaient le même id.

  Pour que cet id soit différent d'une liste à une autre et aussi du bouton ajouter au bouton supprimer il a fallu créer une deuxième fonction permettant l'auto-suggestion qui prenait une chaîne de caractère et affichait l'auto-suggestion au niveau de l'id de la chaîne de caractère correspondante pour chaque liste selon que le bouton était ajouter ou supprimer.

- **Création de colonnes**
  Faire en sorte qu'un nombre donné d'éléments s'affiche sur une ligne avant de remplir la suivante n'a pas été simple.

  Nous avons pour cela introduit des variables servant de compteur (comme présenté dans top.php) en testant leurs divisibilités par le nombre voulu de colonnes avant chaque affichage d'éléments.

- **Gestion de l'héritage**
  L'héritage de la classe Oeuvre n'a pas posé de véritables difficultés, mais nous a contraint à complexifier les requêtes par des jointures et multiplier les includes de Repository dans les fichiers.

- **Dimensions des images**
  Dans notre site chaque oeuvre est associée à une image. Or les images ont très rarement les mêmes dimensions ce qui pose problème à l'affichage. Il a fallu choisir des images de dimensions similaires.

- **Insertion d'instances**
  Entrer les insertions à la main est une tâche fastidieuse; de plus, cette méthode n'est pas optimale pour le maintien de la cohérence des informations dès lors que la taille de la BDD augmente.

  Pour certaines classes telle que "Utilisateur", nous avons donc généré les instances automatiquement via le site Mockaroo.

4.6 Limites du site et du code

- **Réutilisation du code**
  Les codes de certaines parties fichiers sont très similaires à certaines parties et il aurait été plus judicieux de créer un fichier template qu'on appellerait au besoin. Le problème c'est que les variables n'avaient pas toujours le même nom d'un fichier à l'autre et il était donc plus difficile de le généraliser.

5 Conclusion

La création de listes permet de structurer facilement les oeuvres préférées des utilisateurs. De plus, la possibilité de suivre les listes des autres membres de Medialiste donne immédiatement accès à du contenu organisé. Le développement de ce site pourrait par ailleurs déboucher sur d'autres fonctionnalités intéressantes, comme le fait de pouvoir catégoriser les listes elle-mêmes.