Projet Web - Création d'un site pour l'association d'ŒnologIIE

Sophie GERMAIN, Tony RANINI,
Baptiste SIGNOLLE, Jérémie SPIESSER

12 mai 2020

Table des matières

Introduction

Nous avons choisi un thème libre pour ce projet : la confection d'un site web pour ŒnologIIE, l'association des amateurs de vin de l'ENSIIE.

Après de grandes réflexions sur les besoins des futurs utilisateurs de la plateforme que nous nous apprêtions à développer, nous avons convenu de l'implémentation des fonctionnalités générales suivantes :

- la consultation d'une base de données de vins, que les membres pourraient consulter et compléter
- pour chaque vin, un système de commentaires et de favoris
- la gestion personnelle du profil utilisateur pour chaque membre
- un rôle différent pour les membres du bureau, afin qu'ils soient en mesure de modérer le contenu du site.

Vous trouverez dans ce rapport la gestion de projet mise en place, puis l'éventail technique mis en oeuvre pour assurer le meilleur résultat possible.

1 Gestion de projet

1.1 Rédaction d'un cahier des charges

La viabilité d'un projet et son acceptation par les futurs utilisateurs dépendent entièrement de son adaptation à leurs véritables besoins. Dans ce but, nous avons tout d'abord entrepris une démarche de collecte des besoins exprimés par les membres de l'association ŒnologIIE, pour déterminer leurs attentes et discuter avec eux de leurs priorités. A l'issue de cette démarche, nous avions en notre possession un semblant de **cahier des charges**, qui nous aura guidé durant tout le développement de la solution.

Par ailleurs, le choix de ce sujet était en partie motivé par la présence dans notre équipe du président d'ŒnologIIE lui-même. Ses choix, ainsi que l'appel ponctuel à certains autres membres de l'association pour obtenir leurs avis tout au long du projet, nous ont permis d'**éviter l'effet "tunnel"**, dans une démarche que l'on pourrait qualifier de partiellement Agile.

1.2 Architecture du site

Après avoir collecté les différentes fonctionnalités que nous aurions à implémenter, nous nous sommes mis d'accord sur l'architecture du site. Quelles pages, contenant quelles données ? Comment accéder à chacune d'elles et comment organiser l'information ? Telles sont les questions que nous nous sommes posées pour arriver à une architecture logique et pratique :

- une **page d'accueil**, sur laquelle n'importe qui pourra trouver les informations sur la dernière réunion de l'association
- une **page pour chercher** les vins, idem pour les années, les domaines et les types de vins, mais aussi les membres de l'association. Ces pages sont consultable également par tous
- une **page pour chaque vin**, domaine, année, ..., où y consulter les informations mais aussi voir les commentaires laissés par les membres à son propos
- une **page d'ajout** de vins, accessible seulement par les utilisateur connectés
- une **page de profil** éditable par chacun des membres.

Quelques croquis ont été réalisés pour l'association, afin que nous soyons sûrs de ne pas partir sur une **architecture** trop différente de celle attendue. Après qu'ils ont eu fait leur choix, nous nous sommes penchés sur l'aspect qu'aurait le site. Il a ainsi été décidé que la **charte graphique** serait centrée sur les teintes bordeaux, en hommage au thème de l'association.

1.3 Modèle de base de données

Pour une appréciation visuelle des relations et champs de chaque table, un **diagramme de modélisation** de la base de données est disponible à la racine du projet.

Globalement, il existe les tables suivantes : **Wines, Years, Types, Domains, Users, Comments et Roles**.

Les multiplicités sont telles que :

— Users 1 ——[propose]—— * Wines
— Users * ———[a]——— 1 Roles
— Users 1 ——[poste]—— * Comments
— Comments * —[commente]— 1 Wines
— Comments * —[commente]— 1 Years (idem envers Types et Domains)
— Domains 1 –[caractériser]– * Wines (idem pour Years et Types envers Wines)
— Comments * —[répond à]— 1 Comments
— Users * ——[aime]—— * Comments
— Users * ——[favoris]—— * Wines

Nous avons donc dû implémenter 2 relations de multiplicité **Many-to-Many**.

1.4 Répartition des rôles et organisation

Au début du projet, nous avions esquissé un planning prévisionnel pour nous assurer de la bonne gestion du temps à notre disposition. Cependant, la situation actuellement du confinement nous a compliqué la tâche. Finalement, nous sommes parvenus à respecter les délais et à implémenter les fonctionnalités demandées par l'association cliente, grâce à une bonne **organisation** et la mise en place de canaux de **communication**.

Ainsi, nous avons utilisé l'outil Trello pour la **gestion du temps et l'ordonnancement des fonctionnalités**, ce qui nous a permis d'avoir constamment un idée de l'avancement de chaque tâche, et du projet dans sa globalité.

La répartition du travail au sein du groupe a été comme suit :

— Baptiste Signolle : construction et remplissage de la base de données, controllers d'API
— Jérémi Spiesser : JavaScript, repositories
— Sophie Germain : front end, design, controllers des pages
— Tony Ranini : le framework (core et middleware), composeur de vues

2 Points techniques majeurs

Sans décrire chaque élément intégré au site, comment il l'a été et pourquoi, il nous semble toutefois judicieux d'attirer votre attention sur des spécificités techniques importantes que nous avons tenu à intégrer à notre projet.

2.1 Structure du projet

Tout d'abord, un projet n'est intéressant que s'il est viable sur le long terme. Et un des grands facteurs de cette viabilité et la **maintenabilité** de son code : les développeur à l'origine du projet sont rarement les futurs propriétaire du produit, il faut donc que d'autres développeurs puissent prendre le relais avec une certaine facilité.

L'un des points importants de cette maintenabilité est la **structure** propre du projet. Commençons par souligner que notre projet respecte parfaitement le **modèle MVC** (*Model-View-Controller*). On utilise pleinement la partie Orientée Objet de PHP notamment grâce à un autoloader. En plus des répertoires relatifs à cette architecture, soulignons-en 3 autres d'importance certaine :

- **Repositories :** nous conformant au squelette du projet que l'on nous avait fourni au départ, c'est dans les classes de ce répertoire UNIQUEMENT qu'auront lieu les **interactions avec la base de données**.
- **Middlewares :** comme son nom l'indique, ce répertoire contient toutes les actions qui seront **exécutées avant *et* après** l'exécution de la réponse du controller.
- **Core :** c'est ici que seront définies les classes qui permettront de faire fonctionner l'ensemble du projet, comme le coeur d'un organisme vivant. Il s'agit du **framework que nous avons créé *ex nihilo***, qui contient toute la logique pour simplifier l'utilisation et la compréhension du code.

On soulignera de plus l'utilisation de **Composer**, qui permet de simplifier l'utilisation de l'auto-loader et de librairies PHP (librairies telles que Carbon pour la manipulation du temps).

Enfin, pour assurer la compréhension du code de la part des futurs développeurs en charge de maintenir le site, nous avons mis une **documentation complète** à disposition dans le dossier `./doc`, documentation générée avec Doxygen.

2.2 Front-end

Pour réaliser le front-end, nous avons choisi d'utiliser le moteur de template BladeOne. On entre dès lors dans une logique de factorisation de code, de sorte qu'un même code ne soit jamais dupliqué où que ce soit. La structure pour atteindre ce but est la suivante, dans l'ordre croissant d'inclusion) :

- **les *components* :** il s'agit de morceaux du front-end qui seront **utilisés dans plusieurs vues**, mais jamais tels quels. On peut par exemple citer les interfaces de commentaires : que ce soit pour les vins, les domaines, les types ou les années, l'interface est la même.
- **les *includes* :** le header, ou les notifications (d'erreurs, de bienvenue, ..) en sont de bons exemples. Il s'agit de **morceaux** qui seront **inclus dans tous les *templates*** de pages.
- **les *templates* :** littéralement les patrons, il s'agit de la **structure basique commune** à toutes les pages. Par définition, les *templates* incluent donc tous les *includes*.
- **les pages :** sans nécessiter beaucoup d'explication, il s'agit des pages entières du site. Elle sont donc construites à partir des *templates*, qui incluent les *includes*, et sont autant que possible composées de *components*.

De plus, le thème mis en place est dérivé de Bootstrap, le très connu framework CSS. De par cette utilisation, nous garantissons une site entièrement **responsive** pour s'adapter à toutes les tailles d'écran des futurs utilisateurs.

2.3 API

Lors d'une requête, la logique commence dans le fichier `server.php`, qui détermine si une URL demandée est une ressource existante (comme un fichier CSS, un script JS, une image, ...) ou une page que l'on va devoir traiter.

Ce traitement revient à **l'API**. Il existe dans le Core une **classe Route** définie dans `Route.php`, et qui associe à chaque point d'accès un controller et une méthode. L'ensemble de toutes les routes prévues par l'API sont regroupées dans le fichier `./public/web.php`.

Ces routes sont customisables à plusieurs points de vues :

- chacune des routes a **un nom** qui lui est propre
- on peut choisir d'attribuer **un ou des middleware(s) de route** à celles de notre choix. En effet, il existe deux types de middlewares : les middlewares globaux (qui s'exécutent sur toutes les routes) et les middlewares de route (qui ne s'exécutent que sur les routes auxquelles

on les a attribués).
— leur path supporte le **passage de paramètres dans leur URL**, ce qui est très pratique pour ne créer, par exemple, qu'une seule route `/vin/{id}` pour tous les id de vins existants dans la base.

Ces routes sont utilisables grâce à deux fonctions indispensables définies dans la classe Route :
— la fonction `find`, qui permet de retrouver *une route* à partir de son nom (d'où l'importance de la personnalisation des routes par des noms uniques)
— la fonction `getUrl` qui permet de récupérer *une URL* à partir du nom de la route. Cette fonction est très utilisée dans les vues pour assurer davantage de **robustesse** au code : si l'URL d'une route change pour une raison ou pour une autre, la vue parviendra quand même à la charger.

2.4 Ajout de contenu

Comme précisé précédemment, les Repositories sont les seuls endroits d'où la base de données a le droit d'être manipulée, c'est là que sont formulées toutes les requêtes. Pour faciliter l'implémentation de ce repositoriy, nous avons choisi de créer au préalable une classe `Pgsql.php`, qui nous a permis de simplifier l'utilisation de paramètres dans les requêtes, le traitement des requêtes elles-mêmes, et limiter les erreurs.

La modification et l'ajout de contenu sur le site se fait grâce à la saisie d'information dans des formulaires. La **vérification de la conformité de ces formulaires** est permise par un script **JavaScript** contenu dans `checkForm.js`. Contrairement à de petites vérifications individuelles des formulaires, comme par exemple la vérification de la longueur suffisante du mot de passe pour l'inscription, ce script présente l'énorme avantage de pouvoir être appliqué à **tous les formulaires de notre site**.

En effet, pour l'utiliser, il suffit de rajouter un attribut `data-form-on` sur la balise `form` qui définit le formulaire, puis de mettre un attribut `data-form-field` sur les champs du formulaire auxquels nous souhaitons rajouter des contraintes. Ces contraintes peuvent être :
— la vérification qu'un nombre de caractères minimum (`data-form-min='3'` par exemple pour imposer 3 caractère minimum) ou maximum (de la même façon, `data-form-max='3'`) est saisi
— le contrôle qu'une case obligatoire est bien cochée (`data-form-checked`)
— la comparaison avec une expression régulière pour s'assurer que le texte saisi respecte un pattern donné (`data-form-regex='exprReg'`).

— la comparaison de contenu entre deux champs, tous les champs avec cet attribut (`data-form-same='identifiant'`) et un même identifiant sont dès lors vérifiées pour avoir le même contenu

L'ajout de ces contraintes a deux effets : tout d'abord, le script peut vérifier la conformité d'un élément étant donné son identifiant sur la page ; ensuite grâce à des observateurs de type `onkeyup` sur les champs textes et `onchange` sur les checkboxes, il exécute une fonction qui vérifie que le champs en cours de modification est valide et s'il ne l'est pas, affiche un popover d'alerte.

2.5 Authentification

Il existe trois catégories d'utilisateurs à notre site :

— **les visiteurs :** ce sont les personnes ne possédant pas encore de compte, ou n'étant pas encore connectées. Ces visiteurs n'ont pas de droit particulier et ne peuvent que consulter les informations sur site.

— **les membres :** ce sont les personnes qui possèdent un compte sur le site. Elle peuvent grâce à cela accéder à des fonctionnalités supplémentaires, comme la gestion de son profil avec ses vins favoris, ou l'ajout de vins et commentaires.

— **les administrateurs :** au sein de l'association, il s'agira des membres du bureau. Ce sont celles qui auront tous les droits sur le site, qu'ils pourront modérer à leur guise (modification de contenu, suppression d'utilisateurs, ...).

Pour permettre ces différences dans les droits octroyés à chaque rôle, une système **d'authentification** est nécessaire, et nous utilisons pour cela les sessions PHP. En plus d'avoir créé une classe Session permettant de manipuler les variables de la session, nous avons également créé des variables "flashables", c'est-à-dire à usage unique (utile pour transmettre les erreurs après une redirection par exemple).

Il est important de noter deux fonctions d'Auth.php (le fichier du Core qui gère l'authentification) qui jouent un grand rôle à partir de la connexion :

— `isLogged`, qui teste **si l'utilisateur est connecté**. Cette fonction **conditionne l'accès** à certaines pages restreintes à la seule consultation par les membres, comme celle d'ajout d'un vin

— `loggedUser`, qui permet de **récupérer les informations d'un utilisateur connecté** en retournant une instance de LoggedUser, le modèle php qui étend la table User de la base de données.

La redirection en cas de tentative d'accès à une page non autorisée pour un utilisateur d'un certain type emploie ces fonctions et se fait grâce aux

controllers.

2.6 Sécurité

Au cours du développement de ce site web, nous avons tenté de mettre en place quelques points qui pourrait renforcer la sécurité des données des futurs utilisateurs. Le premier que l'ont peut souligner est l'utilisation de la librairie PHP **Dotenv**, qui permet de **stocker les variables sensibles** de manière simple et sécurisée.

Une menace plutôt répandue pour les utilisateurs (et le site en général si la menace pèse sur un compte ayant des droits d'administration) est l'attaque par fixation de session. Il s'agit pour un attaquant de créer un token de session, puis de créer, indépendamment de notre site, une fausse page web. Cette page web piège aura pour but de définir le cookie de notre site dans le cache du navigateur de la victime avec le token que l'attaquant a généré, puis attendre qu'elle se connecte. Il pourra enfin utiliser le token connu pour se connecter sur notre site à la place du véritable utilisateur.

Pour s'en prévenir, le token de session, en plus de n'être connu que du serveur, va être **réinitialisé à chaque connexion/déconnexion** d'un utilisateur bien qu'en parallèle le Core récupère les autres informations de sessions.

Un autre protection a été mise en place sur le site : la **protection CSRF** (*Cross-Site Request Forgery*). Cette protection consiste en la génération d'un token à chaque chargement de page qu'on stocke dans 2 endroits : un dans la session de l'utilisateur, et l'autre dans un `input` caché de tous les formulaires HTML. C'est à la récupération des valeurs `POST` qu'intervient le middleware `CsrfProtection.php`. Ce middleware fait en sorte qu'avant chaque traitement de la requête `POST`, une comparaison entre le token envoyé et celui stocké dans la session soit effectuée. Si ces **tokens sont différents**, alors la page se réinitialise en affichant une **erreur** plutôt que d'exécuter la requête (une certaine conservation des informations saisies dans les formulaires est permise, de sorte que s'il s'agit simplement d'une erreur de manipulation, l'utilisateur n'ait pas à re-remplir tous les champs manuellement).

Conclusion

En dépit des problèmes d'organisation rencontrés, nous avons été capables de produire un site fonctionnel et répondant parfaitement aux besoins exprimés par les utilisateurs. Une amélioration envisageable pourrait-être, selon les membres du bureau d'ŒnologIIE, de pouvoir envoyer un mail à toute la mailing-list de l'association depuis le site, par exemple. Cela n'était toutefois pas dans les fonctionnalités à implémenter prioritairement, et les contraintes de temps n'étaient pas suffisantes pour que nous le fassions.

De plus, pour les besoins du projet, nous avons géré nous-même l'inscription et l'authentification des membres sur le site. Au long terme, il sera cependant peut-être plus intéressant de lier ce site avec iiens.net, de sorte que la connexion se fasse grâce aux identifiants de la plateforme des élèves.