Rapport de projet de web

Tales of Webseria

Le site dont *vous* êtes le héros

par *La Meute de Crêpes*
Maël Berrou
Nina Buis
Quentin Lecomte
Lana Scravaglieri

May 12, 2020

Contents

1 Introduction

Pour ce projet, nous avons décidé de faire un site permettant de jouer à des jeux du type Les Livres dont vous êtes le héros.

Devant un système de jeu en apparence simple, qui ne nécessite au départ qu'un crayon, une gomme et deux dés à six faces et qui ne demande que des actions simples pour faire progresser le joueur, à savoir choisir le prochain paragraphe au travers de la narration, lancer les dés lorsque cela est requis et effectuer des calculs simples pour déterminer un échec ou une réussite, nous avons rapidement vu qu'il fallait le simplifier encore pour l'automatiser.

C'est ainsi que nous avons réduit la complexité des fonctionnalités à implémenter en faisant un jeu strictement narratif qui garde cependant la possibilité d'être étendu par la suite pour se rapprocher de son modèle.

2 Organisation du travail

Dans un premier temps, nous avons décidé des fonctionnalités que nous voulions voir sur notre site sous la forme de user stories, puis nous avons écarté certaines fonctionnalités que nous avons jugées non-essentielles. Nous avons ensuite classé les fonctionnalités par groupes de priorité, en tenant également en partie compte de la dépendance de certaines fonctionnalités (par exemple, il faut avoir implémenté le login avant de s'inquiéter du logout).

Les users stories que nous avons gardées sont les suivantes (par priorité croissante) :

1

- A guest can create an account anytime they want.
- A guest can log in provided they have an account.
- Everyone can start a story at any time.
- Everyone can choose a path in a story.

2

- A connected user can log out.
- A connected user can leave a comment on a story's summary page.
- Everyone can see a story's summary (with comments).
- Everyone can search through available stories when arriving on the site.

3

- A connected user can save their progression during one of their journeys.
- A connected user can see their saved stories.
- A connected user can load a saved story.
- A connected user can change their account information.
- A connected user can delete their account.
- A connected user can rate a story on the story's summary page.
- An administrator can moderate users and messages.

- An administrator can fetch users' data (achievements, mail adresses... etc).
- An administrator can delete accounts.

Nous avons ensuite défini des tâches pour pouvoir implémenter ces fonctionnalités. Il fallait notamment créer la base de données et le template pour afficher correctement, sur chaque page, un menu et le contenu de façon cohérente et uniforme, tout en gardant une structure claire dans le code.

3 Organisation du code

3.1 Le modèle de Base de Données

Nous avons réfléchi en groupe sur la construction des tables. Voici un diagramme qui montre toutes les tables que nous avons créées ainsi que les liens existants entre elles.

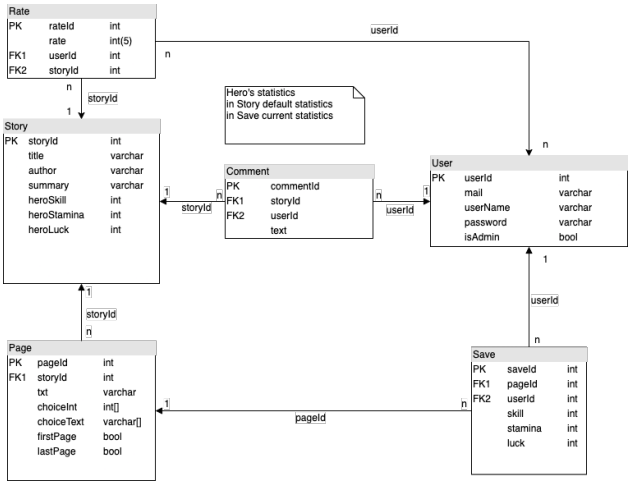

Figure 1: Schéma de la base de données

Pour pouvoir implémenter les différentes fonctionnalités, et notamment tester l'affichage en parallèle de la création des formulaires qui permettent d'ajouter des données dans la base, nous avons peuplé la base avec des entrées factices. En particulier, les histoires ne sont pas complètes, certaines pages sont inexistantes, et la seule histoire qui soit un peu jouable est *La Cité des Voleurs*.

3.2 Template du site et architecture MVC

Pour organiser le code selon l'architecture Modèle-Vue-Contrôleur, nous avons isolé toutes les requêtes à la base de données dans des classes Repository, une par table sauf pour les pages qui sont traitées par le StoryRepository car elles sont intrinsèquement liées à leur histoire. Conformément au design pattern Repository, les données extraites de la base de données sont formatées dans des objets aisément manipulables par le reste de l'application. Ces objets sont définis par les classes dans le dossier Entity. Les Repositories utilisent tous un objet de classe DbAdaper, qui fait la connexion et les requêtes à la base de données à proprement parler.

L'affichage des vues se fait via une fonction `loadView` prenant en argument une chaîne view et un tableau data, et définie dans `template.php`. Cette fonction inclut dans la page un menu défini dans `layout/header.php` et la vue spécifiée par la chaîne view. Chaque vue utilise le tableau data pour afficher du contenu variable dans un format standard.

Les controllers dans le dossier public fonctionnent comme suit. L'inclusion du fichier `autoloader.php` permet d'inclure toutes les classes des dossiers Entity et Repository, la classe DbAdaper ainsi que le fichier template pour pouvoir appeler `loadView`. Ensuite, on crée un objet DbAdaper, les objets Repository dont on a besoin pour la page que l'on veut afficher et un tableau data vide. On effectue alors les vérifications nécessaires, et les requêtes pour mettre dans le tableau data toutes les données utiles à l'affichage ou aux actions que la page permet de faire. Lorsqu'on appelle finalement `loadView` pour l'affichage, le tableau data est transmis comme second argument.

Lors de la construction d'une page, nous pouvions alors réfléchir de la manière suivante : on commence par lister les informations de la base de données nécessaires à l'affichage et aux actions de la page. Si les classes et les Repositories qui correspondent sont manquants ou incomplets, on les construit, puis on fait le controller. Lorsqu'une page nécessite des informations reçues depuis un formulaire, on vérifie leur existence, et on crée sur la page adéquate le formulaire qui transmet les données attendues par notre nouvelle page.

Pour les actions qui ne dirigent pas vers une nouvelle page mais permettent d'interagir directement avec la base de données, des pages spécifiques effectuent les vérifications nécessaires, avant de faire les requêtes via un Repository et de rediriger l'utilisateur vers une page appropriée. Ces fichiers-là portent le suffixe Manager.php pour les derniers que nous avons faits, les premiers avaient un nom descriptif sans norme et nous ne les avons pas renommés.

4 Les utilisateurs

4.1 La connexion

Le client peut se connecter via `login.php` ou créer un compte via `register.php`. Ces deux fichiers ne sont en fait que des formulaires qui appellent ensuite `server.php`. Le fichier `server.php` est un élément clé du projet. C'est lui qui fait le lien entre l'utilisateur et la base de données du site. Il permet donc de créer un compte ou de se connecter à un compte déjà existant. Au début du fichier, on crée un UserRepository qui permettra d'interagir avec la table "users" via un DbAdaper. Ensuite, selon la variable de $_POST qui est initialisée, il va effectuer différentes tâches : créer un utilisateur et l'enregistrer dans la base, ou chercher un utilisateur dans la base et mettre à jour la variable `$_SESSION` en conséquence. C'est également lui qui gère le changement des informations personnelles des utilisateurs. Les requêtes utilisent la fonction `bindParam` pour éviter les injections SQL que les vérifications JavaScript des formulaires auraient laissées passer. Les mots de passe sont encryptés dans la base lors de l'enregistrement d'un nouvel utilisateur, grâce à la fonction crypt. Cela permet d'éviter qu'en cas d'attaque, les utilisateurs voient leur mots de passe révélés. La variable `$_SESSION` permet d'accéder à différentes informations durant toute la session de l'utilisateur. On retrouve notamment son identifiant, son surnom et s'il est administrateur ou non.

4.2 Les pages utilisateur et administrateur

La page d'utilisateur permet à un utilisateur connecté de supprimer son compte et de modifier ses informations personnelles. Pour cela, on utilise une fois encore UserRepository pour pouvoir mettre à jour les informations dans la base, de manière propre, via `server.php`.

La page d'administrateur permet de voir le compte de chaque utilisateur avec son identifiant, son adresse mail... etc, et de supprimer n'importe quel compte non administrateur.

La suppression de compte de ces deux pages est gérée par `deleteUser.php`. Ce fichier a pour but de mettre à jour correctement la base de données pour éviter la violation de contraintes. Par exemple, quand un utilisateur supprime son compte, ses commentaires restent visibles, mais seront alors indiqués comme ayant été écrits par Deleted User . Cet utilisateur particulier ne possède pas d'email ni de mot de passe (il est donc impossible de se connecter via ce compte), et est considéré comme étant administrateur. Ainsi,

on peut conserver les messages écrits et les notes données par quelqu'un ayant supprimé son compte. Les sauvegardes, elles, sont supprimées définitivement.

En plus de cela, un administrateur peut aussi supprimer des commentaires s'il le juge nécessaire, grâce au fichier `commentManager.php`. Nous y reviendrons dans la partie sur les commentaires.

5 Les histoires et le jeu

5.1 La structure de graphe des pages d'histoire

Les pages des histoires sont stockées dans la base de données avec pour informations : un identifiant, un texte narratif, de zéro à trois choix numérotés et leurs textes associés, sachant que ces choix sont des identifiants d'autres pages. La structure de graphe des pages peut donc être schématisée sous forme d'un arbre dont les nœuds seraient les pages, et les successeurs de ces nœuds les pages auxquelles mènent les pages courantes. Une histoire peut ainsi être représentée par un graphe dont les noeuds sont les pages. Les choix possibles à partir d'une page constituent en fait la liste des successeurs du noeud.

L'identifiant d'un successeur est envoyé par la méthode GET lorsque, sur la page `story.php`, le lecteur fait un choix et clique sur le bouton correspondant. La page `story.php` se recharge en affichant cette fois la page suivante, récupérée dans la base de données grâce son identifiant. Ce processus se répète tant que le lecteur avance dans l'histoire, en suivant son chemin dans l'arbre des pages.

Ce graphe est donc représenté par une liste d'adjacence. Ce choix, en plus d'être intuitif au vu de la structure du jeu papier, est parfaitement adapté à un graphe aussi peu dense autant pour le stockage que pour la manipulation.

Figure 2: Page qui permet de lire une histoire et de la jouer

5.2 Le système de lecture

L'affichage des pages d'une histoire est réalisé par `story.php`. Lorsqu'un utilisateur lance une histoire en cliquant sur le bouton "Commencer" au sommaire de l'histoire, l'identifiant de cette dernière est envoyé par la méthode GET à `story.php`. L'histoire et ses pages sont alors récupérées grâce à l'identifiant de l'histoire,

par le biais d'une fonction définie dans `StoryRepository.php`. Les informations récupérées et stockées dans un tableau data sont : l'identifiant et le titre de l'histoire, et la page courante.

Après obtention de toutes les données nécessaires, il est enfin possible d'afficher le titre de l'histoire, le texte narratif de la page et les choix. Ces derniers sont des boutons qui, lorsque le lecteur clique dessus, envoient l'identifiant de la page suivante à `story.php` par la méthode GET. Alors `story.php` se recharge, mais cette fois en affichant la page d'histoire suivante, celle associée au choix que vient de faire lecteur.

Une fonction JavaScript introduite dans `story.php` permet de garder le niveau de défilement du lecteur sur la page `story.php` lorsqu'elle se recharge. Ainsi, l'utilisateur évite de se retrouver au sommet de la page, c'est-à-dire d'être décalé par rapport à la zone de lecture, à chaque fois qu'il effectue un choix. Cela contribue à améliorer l'expérience utilisateur.

6 Les fonctionnalités annexes

6.1 Le système de notation des histoires

Chaque utilisateur peut évaluer chacune des histoires disponibles sous la forme d'une note allant de 1 à 5. Un utilisateur peut noter chaque histoire une, et une seule fois.

Ces notes sont enregistrées dans la base de données, dans la table rate. Lorsque nous accédons à la page de résumé d'une histoire, nous avons la note moyenne de l'histoire. Celle-ci est obtenue à l'aide du RatingRepository de `story_page.php`, qui récupère les informations de la base de données. Nous pouvons aussi voir le nombre de personnes ayant accordé chaque note à l'histoire. De plus, si l'utilisateur est connecté et qu'il n'a pas voté, il a la possibilité de voter pour l'histoire. S'il n'est pas connecté, il lui faudra d'abord le faire. En effet, le bouton n'est pas visible tant que la variable `$_SESSION['id']` n'est pas initialisée (et cela est fait lors de la connexion).

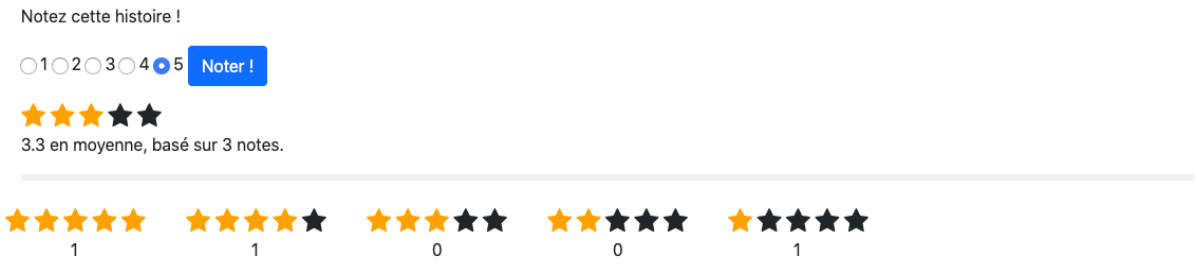

Figure 3: Notation d'une histoire

6.2 Les commentaires

Chaque utilisateur peut commenter autant qu'il le veut une histoire.

Le commentaire tapé dans le formulaire dédié est envoyé par la méthode POST au fichier `commentManager.php` qui l'ajoute à la base de données grâce aux méthodes d'un CommentRepository.

Un administrateur a accès à un autre formulaire pour supprimer les messages jugés inappropriés, qui prend la forme d'un simple bouton permettant de supprimer le commentaire auquel il est associé, en envoyant son id au commentManager.

Ce dernier gère aussi les erreurs, si jamais quelque chose se passe mal lors de l'ajout à la base de données. Le client est alors renvoyé sur la page présentant les différentes histoires avec un message d'erreur.

6.3 La sauvegarde de progression

Lorsque l'on joue une histoire la progression est toujours sauvegardée dans des variables de session ce qui permet de reprendre la dernière histoire visitée à la dernière page consultée grâce au lien du menu "Reprendre". On peut aussi sauvegarder sa progression pour reprendre l'histoire d'une session à l'autre grâce au bouton en bas de la page. Cette fonctionnalité nécessite un compte donc si le visiteur n'est pas connecté, il sera redirigé vers la page de connexion.

Le bouton de sauvegarde appel le fichier `saveManager.php` qui vérifie si l'utilisateur est connecté. Si oui, il vérifie à l'aide d'une instance de SaveRepository si une sauvegarde pour cette histoire et cet utilisateur existe déjà. Cela se fait avec la fonction `existSave` qui effectue une requête avec une jointure entre la table save et la table page pour trouver la paire (`storyid`, `userid`). Si une telle sauvegarde existe on récupère son identifiant `saveid` et on la met à jour avec la fonction `updateSave` dont la requête modifie dans la base la page et les caractéristiques du héros. Sinon on crée une nouvelle sauvegarde avec la fonction `addSave` du repository. L'utilisateur est alors redirigé vers la page qu'il était en train de lire.

Sur la page des parties sauvegardées, un SaveRepository permet de récupérer les sauvegardes d'un utilisateur grâce à son identifiant. Si aucune sauvegarde n'existe on affiche un message le précisant. Sinon on affiche les cartes des histoires avec 2 formulaires consistant en 2 boutons. L'un redirige vers la page de l'histoire avec les identifiants de l'histoire et de la page sauvegardés. L'autre fait appel au `saveManager` qui supprime la sauvegarde à partir de son identifiant avec un SaveRepository. L'utilisateur est ensuite redirigé vers la page des sauvegardes qui est mise à jour.

Figure 4: Page des parties sauvegardées

6.4 La recherche d'histoires

Un client peut chercher une histoire en entrant le titre de celle-ci dans une barre de recherche dans le menu, qui est en fait un formulaire. Ce dernier envoie le titre entré par l'utilisateur à `search.php`, à l'aide de la méthode GET. Le fichier `search.php` fait appel à une fonction définie dans `StoryRepository.php` qui permet de récupérer une histoire dans la base de données à partir de son titre. Si ce titre existe dans la base de données, l'histoire correspondante est récupérée avec comme informations son identifiant, son titre et son auteur. Dans ce cas, la vue affiche le résultat de la recherche : un bouton à l'apparence de lien, présentant l'histoire désirée et son auteur, et qui le redirige vers le sommaire de l'histoire cherchée.

La définition de la fonction qui récupère une histoire à partir du titre de celle-ci est telle que pour obtenir

un résultat de recherche, l'utilisateur doit entrer le titre exact de l'histoire en respectant la casse. S'il tape un titre incomplet ou avec la casse inadéquate, il se retrouvera sans résultat de recherche. Auquel cas, la vue affichera un message expliquant comment rechercher une histoire correctement. Nous n'avons pas eu le temps de faire une recherche plus élaborée avec un titre partiel ou par auteur.

6.5 Les messages d'erreur et de succès

Afin que l'utilisateur reste informé de ses actions sur le site, des messages d'erreur et de succès s'affichent régulièrement.

Pour ce faire, il existe deux champs dans la variable `$_SESSION`, 'errors' et 'success', qui, s'ils sont affectés, sont affichés au début de chaque page (cela est effectué dans le template, en php, après l'inclusion du header). Une fois le message affiché, le champ correspondant est unset, afin qu'il ne soit pas affiché plus tard lorsque l'on change de page. Ces messages sont affectés dans les Managers et dans `server.php` notamment, lors du succès d'une opération pour success ou lors d'un échec pour errors.

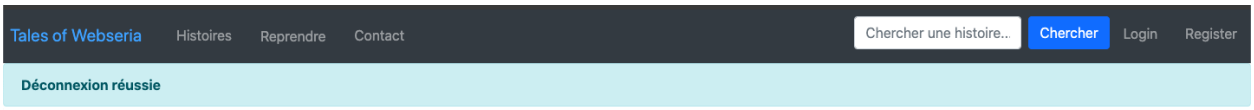

Figure 5: Message de succès suite à une déconnexion

7 Conclusion

Au final, toutes les fonctionnalités essentielles ont été implémentées et fonctionnent correctement. Le site est plaisant à utiliser.

Le site pourrait cependant bénéficier d'une meilleure gestion des erreurs avec une redirection systématique vers la page d'accueil en cas de tentative d'accès à une page non-autorisée ou sans les variables POST et GET correctement assignées.

L'organisation du code est satisfaisante mais si le projet devait être mené plus loin, il serait nécessaire d'isoler les Managers et renommer des fichiers.