Rapport du projet de programmation web

Agenda des associations

Timothé Louzier, Hugo Malard, Laura Scholasch

12 mai 2020

Table des matières

Introduction **2**

Structure du site **2**

Structure de la base de données **3**

Authentification et inscription **3**

Mise en place de l'emploi du temps **5**

Gestion des administrateurs et des utilisateurs **6**

Système de paris **7**

Répartition des rôles **8**

Difficultés rencontrées lors de ce projet **8**

Conclusion **9**

Introduction

Le but de ce projet est de créer un site qui va permettre d'**organiser les réunions** des différentes associations d'une école. Grâce à cet agenda, nous pourrons savoir et gérer qui sera présent lors des réunions.
Nous avons également décidé de rajouter une fonctionnalité à cet agenda : la mise en place d'un **système de paris sur les retards** des participants aux réunions.

Pour accéder au site, effectuez les commandes suivantes :

1) **make db.init** afin d'initialiser la base de données
2) **make start** qui permet de lancer le serveur sur localhost:8080
3) **make db.reset** si vous souhaitez réinitialiser la base

Structure du site

Le site se présente de la manière suivante : la seule page accessible sans connexion est la page **index.php**. C'est sur cette page que l'utilisateur se connecte à son espace membre, il a également la possibilité de créer un compte s'il n'en a pas.

Il y a 3 types d'utilisateurs : les **membres** d'une associations, les **administrateurs** et le **super administrateur.** Chaque type d'utilisateur a accès à des fonctionnalités différentes.

Lorsqu'un utilisateur se connecte il arrive sur une page web avec les **onglets** suivants :

- **Agenda :** c'est ici que l'utilisateur peut voir son emploi du temps avec les réunions de son association. -> **agenda.php**
- **Profil :** l'utilisateur peut voir ou modifier son profil ou encore, s'il est administrateur d'une ou plusieurs associations, peut rentrer le retard des participants -> **profil.php**
- **Réunions :** cet onglet n'est utilisé que par les administrateurs d'associations afin qu'ils puissent créer de nouvelles réunions -> **OrgaReu.php**
- **Paris** : permet de parier sur les différentes réunions -> **bet.php**
- **Déconnexion :** pour se déconnecter

Le super-administrateur dispose d'un onglet supplémentaire, **home_super_admin.php** afin de gérer les utilisateurs et les administrateurs du site. Les utilisateurs sont égalements informés dans la barre d'onglets des points qu'ils possèdent pour les paris ainsi que du nom d'utilisateur du compte sur lequel ils sont connectés.

Nous avons principalement codé le site en **orienté objet**, en créant des fonctions dans des classes qui se trouvent dans des fichiers Repository : par exemple ReunionRepository, AssoRepository…

Structure de la base de données

Nous avons choisi d'utiliser une base de données découpée en 9 tables initialisées dans le fichier **init.sql** :

- ***Membre(****id,username, email,created_at,passwd,points) :* contient tous les utilisateurs et leurs informations personnelles.
- ***Administrateur****(Id_MembreA, Droit) :* contient la liste des administrateurs.
- ***Association****(Id_Assoc, Nom_assoc) :* contient la liste des associations.
- ***Reunion****(Id_Assoc,Id_reu,Date_debut_reu,Date_fin_reu,Id_MembreA,Descriptif)* : contient toutes les informations sur les réunions.
- ***Demandes_user_Superadmin****(username, Nom_assoc)* : table pour stocker les demandes d'élèves qui souhaiteraient devenir administrateur.
- ***Appartenir****(Id_Assoc, Id_Membre, Nom_Assoc, username)* : permet de savoir qui appartient à quelle association.
- ***Administrer****(Id_Assoc, Id_Membre) :* permet de savoir qui est administrateur de quelle association.
- ***Participations****(Id_reu, Id_Membre, statut, retard)* : permet de savoir qui participe à quelle réunion.
- ***Paris****(id_paris, player, id_reu, id_user, retard, mise, date_paris)* : contient les informations sur les paris.

Authentification et inscription

Une des premières étapes de ce projet fût de créer un formulaire d'authentification afin d'accéder à son espace membre. Il a ensuite fallu créer un f**ormulaire d'inscription** lorsqu'un nouvel utilisateur souhaite s'inscrire sur le site.
Nous avons ensuite voulu aider l'utilisateur à corriger les données qu'il saisit. Pour ce faire, nous avons mis en place une **validation des formulaires avec Javascript**, pour tester que l'adresse mail était bien valide, que les champs du formulaire étaient bien remplis ou encore que les deux mots de passes rentrés étaient bien identiques.

La page du profil est composée de **4 parties** :

1. L'édition du profil
2. La demande d'adhésion à une association

3. Le formulaire de demande afin de devenir administrateur d'une association
4. Les informations du profil

Pour l'**édition du profil,** le fait de cliquer sur une modification (nom d'utilisateur, mot de passe, ou suppression du compte) appelle une fonction JS qui va afficher à droite de celle ci un champ de texte s'il y a quelque chose à saisir, puis un bouton pour valider.

En ce qui concerne la **suppression du compte**, nous avons d'abord eu quelques problèmes: nous arrivions à supprimer un individu mais uniquement lorsque celui n'appartenait dans aucune association. Pour régler ce problème, nous avons ajouté la commande "**on delete cascade**" sur les autres tables de la base qui contenaient la clé primaire id_membre. Ainsi, lorsqu'un utilisateur est supprimé, il est également supprimé de toutes les tables auxquelles il appartient.

Pour la **suppression du compte** et le **changement de mot de passe** nous avons choisi de d'appeler un fichier php lorsque l'utilisateur clique sur valider. Celui-ci appelle une fonction de la classe User qui fait les modifications dans la base de données.

Pour la **demande d'adhésion** à une association on utilise une fonction qui recupère tout les noms d'association et les mets dans une liste. Puis, dans un foreach qui va parcourir tous les éléments de cette liste, on va vérifier via une autre fonction qui effectue une requête si l'utilisateur est déjà dans cette association. Auquel cas on n'affiches pas le nom de cette association dans la liste déroulante. Ainsi l'utilisateur ne peut choisir de rejoindre que les associations dans lesquelles il n'est pas déjà.
Une fois qu'il a cliqué sur valider, il est redirigé vers une page php qui appelle une fonction de AssoRepository qui va insérer l'utilisateur dans la table Appartenir avec l'association qu'il a choisi.
Nous avons décidé de ne pas faire de vérifications ici, car nous souhaitions nous rapprocher le plus de la situation à l'école, où il n'y a pas de "sélection" pour faire partie d'une association.

Pour **devenir administrateur d'une association**, nous avons mis un lien cliquable vers la page **Form_demande_admin.php** qui permet, via le même moyen que pour la demande d'adhésion à une association mais à l'inverse, d'afficher dans une liste déroulante les associations desquelles fait partie l'utilisateur. Il n'a donc qu'à sélectionner l'association de laquelle il veut devenir administrateur et à valider. Il est ensuite redirigé vers une page où on lui dit que la demande a bien été envoyée.
Pour qu'il devienne administrateur, il faut ensuite que le super-administrateur valide sa demande. Cette manipulation sera expliquée dans la partie des fonctionnalitées pour le super administrateur.

La partie information du profil est un texte cliquable qui via une fonction JS cache et affiche les noms des associations dont l'utilisateur fait partie.

Mise en place de l'emploi du temps

Nous avons créé un fichier **scheduleTools.php** afin de répertorier toutes les fonctions qui vont permettre le bon affichage de l'agenda comme par exemple une fonction qui va renvoyer true si les réunions se chevauchent et false sinon, ou encore une fonction qui retourne le jour correspondant au n-ième jour du mois.

Toute action de la page **agenda.php** qui recquiert la mise à jour de la base de donnée ou la récupération d'informations depuis cette base est effectuée à travers **transition.php**.

Notre objectif était d'avoir un **affichage propre de l'agenda**. Pour cela, nous avons créé un fichier **schedule.css** spécifique à la mise en page de l'emploi du temps.

Pour qu'un élève soit **notifié d'une nouvelle réunion** créée par l'administrateur d'une association dont il fait partie, la nouvelle réunion apparaît en rouge sur son emploi du temps. Lorsqu'un élève n'a pas encore donné sa réponse, le créneau de la réunion s'affiche en rouge. Lorsqu'il aura donné sa réponse, le créneau deviendra alors bleu.

Une réunion apparaît dans l'agenda si et seulement si il existe une entrée dans la table Participations pour l'utilisateur et cette réunion, d'où l'importance de mettre à jour correctement la base de données lorsque l'on crée une nouvelle réunion par exemple.

Il y a eu deux défis majeurs à gérer pour la mise en place de cette page: l'agencement des réunions (chevauchement des réunions, gestion des dates…), et le statut des réunions.

Pour ce qui est de l'**agencement des réunions**, nous avons du concevoir un algorithme qui vérifie pour chaque réunion si elle se déroule en même temps qu'une autre réunion, puis qui attribue une taille d'affichage et une position d'affichage en fonction de ce nombre de réunions.

Pour la gestion du **statut des réunions** il a fallut différencier les statuts suivants dans la table participation de la base de données:

0. Souhait de participer à la réunion
1. Souhait de ne pas participer à la réunion
2. Souhait non exprimé
3. Réunion passée, retards non enregistrés
4. Réunion passée, retards enregistrés

Nous n'avons pas encore implémenté la gestion des statuts 3 et 4 qui permet de renseigner les retards des élèves pour obtenir des résultats dans le système de paris.
En revanche les statuts 0 à 2 sont correctement gérés, avec l'impossibilité de participer à deux réunions simultanées (on avertit l'utilisateur que confirmer la présence à une réunion annulera sa présence aux réunions qui ont lieu au même moment).

Gestion des administrateurs et des utilisateurs

Il y a donc 3 types d'utilisateurs : les **membres** d'une associations, les **administrateurs** et le **super-administrateur**.

1) **Fonctionnalités spécifiques pour le super-administrateur**

Il y a donc un seul super-administrateur qui a la visibilité sur tous les rôles des utilisateurs du site. Il a donc accès aux listes des membres, et des administrateurs grâce à l'onglet Gestion. Pour accéder facilement à ces listes, nous avons utilisés des fonctions fetch() créées dans **UserRepository.php** qui renvoient les listes de tous les membres et administrateurs.
Le super-administrateur peut également accepter ou refuser la demande d'un membre qui souhaiterait devenir administrateur d'une association. Lorsqu'il accepte la demande d'un membre, celui-ci sera automatiquement ajouté dans la table Admistrateur et Administrer grâce à un INSERT INTO, et supprimé de la table Demandes_user_Superadmin grâce à la commande DELETE.

2) **Fonctionnalités spécifiques pour les administrateurs**

Les administrateurs des associations sont les seuls à pouvoir créer des réunions pour leurs associations. Il a fallu vérifier que l'administrateur puisse créer des réunions uniquement pour les associations dont il sera l'administrateur. Pour ce faire, nous avons créé une fonction fetch_all_Assos_for_Admin, qui, à partir du userid de l'admin nous renvoie la liste des associations dont il est administrateur.

Les administrateurs peuvent également voir qui participe ou non aux réunions, tout comme les membres.

Les administrateurs doivent, après qu'une réunion ait lieu, rentrer le retard des participants. Pour cela, nous avons créé une fonction ajout_retard dans **ParticipationRepository.php** qui permet d'insérer dans la table Participations le retard ajouté par l'administrateur.

3) **Fonctionnalités spécifiques pour les élèves**

Les élèves pourront dire s'ils participent à une réunion. Les statuts possibles des réunions sont : Oui, Non, En attente de réponse.
Ils ont donc accès à leur agenda, tout comme les administrateurs. Ils peuvent, comme dit précédemment, demander à devenir administrateur d'une association, et cela, grâce à un formulaire.

Système de paris

Le système de paris a été principalement codé dans le fichier **bet.php**. Les transitions nécessitant un appel à la base de données sont implémentées dans **transitionParis.php**.

Dans l'onglet gauche de la page, on affiche la liste de toutes les **réunions à venir**, triées par associations et par ordre chronologique.

Après avoir cliqué sur une des réunions, chaque utilisateur peut parier sur le retard potentiel des participants de la réunion (il ne peut bien sûr pas parier sur son propre retard).
Il parie ensuite un certain nombre de points, et recevra son gain s'il a misé juste à 5 minutes prêt lorsque l'administrateur de ladite réunion aura rentré les retards des participants une fois la réunion finie (passage du statut 3 au statut 4 dans la table Participation).

Pour parier, l'utilisateur a accès à la moyenne des retards de chaque participants à la réunion ainsi que la moyenne des retards pour l'association concernée grâce à l'appel de 2 fonctions *getAverageDelay* et *getAverageDelayAsso*.

L'attribut statut de la table Participations nous a été utile : si le statut est inférieur à 3, il s'agit d'une réunion future, si le statut est 3, alors la réunion est terminée mais l'administrateur doit encore rentrer le retard des participants. Si le statut est 4, les résultats du paris sont calculés et disponibles à la consultation.

On discerne alors plusieurs **statuts des paris**, il y en a 5: le statut d'un paris vaut 0 si le résultat est à venir, 1 si la réunion est finie mais les résultats en attentes, 2 si le résultat doit être consulté dans le cas d'une victoire, 3 si le résultat doit être consulté dans le cas d'une défaite, 4 si la victoire a été consultée et 5 si la défaite a été consultée. On peut en déduire un affichage différent pour chaque statut dans l'onglet Mes Paris.

Lorsqu'un utilisateur consulte le résultat d'un paris, si l'utilisateur a remporté son paris, son nombre de points est augmenté en conséquence.

Pour déterminer les gains d'un joueur, il a fallu calculer la **cote associée à son paris**. Nous comptions effectuer un calcul profond à l'aide d'une IA sur une base statistique, mais nous n'avons pas eu le temps de l'intégrer. Nous avons donc dans un premier temps défini la courbe de la côte en fonction du retard parié comme un polynôme du second degré. Le polynôme atteint son minimum en la moyenne du retard moyen et du retard moyen pour l'association (retard estimé le plus probable). La valeur de ce minimum correspond à une cote de 1,1. La courbe atteint également la valeur 2 à + ou - 15 minutes du minimum.

Nous avons rencontré quelques difficultés pour la gestion des dates et des temps de retard sur cette page mais nous n'avons surtout pas eu le temps de corriger les fonctions de calcul des moyennes des retards, ainsi que la fonction permettant à un adminstrateur de rentrer les retards des participants dans la base de donnée pour mettre à jour les statuts. En effet, nous

avons perdu une partie des nos travaux sur ces fonctions lors d'un merge de nos travaux mal effectué. De ce fait nous n'avons pas pu implémenter de fonction calculant le résultat d'un paris une fois le statut des participations mises à jour. Le reste demeure fonctionnel.

Répartition des rôles

Nous avons utilisé Github afin de pouvoir modifier et développer notre projet entre nous 3 assez facilement.
Nous avons travaillé assez régulièrement et nous faisions des points toutes les semaines, pour se montrer l'avancée de nos tâches ou pour se débloquer.
Avant de coder chaque onglet, nous avons rassemblé toutes nos idées sur une feuille de papier afin que la programmation de celui-ci se fasse plus simplement.

La répartition du travail a été principalement faite ainsi:

- **Timothé Louzier :** Mise en place de l'agenda et du système de paris.
- **Hugo Malard :** Edition du profil, CSS, formulaires et validation javascript.
- **Laura Scholasch :** gestion des administrateurs et du super-administrateur, créations de réunions.

Difficultés rencontrées lors de ce projet

- Au début du projet, nous avons passé un peu trop de temps sur la **mise en place de la base de données**. En effet, c'était la première fois que nous manipulions une base de données en Postgresql.
- La **gestion des dates** en php fût assez complexe au début mais les ressources internet nous ont permis de résoudre tous nos problèmes assez facilement. Elle fut tout de même responsable d'une grande perte de temps.
- Les **requêtes sql** étaient assez compliquées à mettre en place dans le sens où nous n'avions pas vraiment trouvé de moyen simple et rapide pour les tester afin de savoir si elles s'exécutaient correctement.

- La **mise en place de l'emploi du temps** et tout le **CSS** ajouté pour un affichage propre de l'agenda a été une tâche très chronophage.

- Nous avons perdu à maintes reprises une partie de nos travaux lors de **problèmes de merge**. En effet, bien que nous travaillons sur des fichiers et branches différents, nous partagions les fichiers Repository et le fichier init.sql.

- Nous nous étions concertés en amont pour construire une base de donnée sur un modèle relationnel SGBD et un diagramme UML. Toutefois, l'ajout de diverses fonctionnalités au site à entraîné des **modifications de la base de données**. A cause des ces modifications, nous avons perdu le caractère optimal de notre base originale ce qui a entraîné de nombreux bugs à corriger et une lourde perte de temps.

- Dans certains cas, le passage du php au javascript a été complexe, car ne pouvant pas utiliser jQuery ou AJAX pour rafraîchir des portions de pages, il nous a fallu jongler entre de nombreuses pages différentes pour actualiser via un script js des informations contenues dans la base de donnée. Puisque le js est compilé côté client et le php côté serveur, ce genre de manipulations ont augmenté la complexité du code.

Conclusion

Grâce à ce projet, nous avons su progresser en html, php, javascript et css, chercher sur internet nos erreurs et mettre en forme nos idées tous ensemble pour aboutir à un site qui nous convenait à tous.
Malgré tout, nous aurions pu améliorer la sécurité de notre site web. En effet, nous n'avons pas trouvé d'autre moyen pour envoyer certaines variables d'un fichier à l'autre que de les transférer via l'url. C'est un danger car un utilisateur peut agir directement sur la base de donnée en affectant des valeurs différentes aux variables de l'url. Ce genre de problèmes auraient toutefois pu être évités en utilisant jQuery ou AJAX qui sont des outils courants particulièrement dans le cas d'appels interactifs à une base de donnée comme ici avec un agenda.
Enfin, nous avons cruellement manqué de temps et nous n'avons pas pu réaliser notre projet de base qui consistait à inclure un script python contenant un réseau de neurones capable de "prédire" les retards auxquels l'utilisateur aurait pu se confronter (en terme de paris) afin d'établir des cotes précises.