

Documentation technique

-

Projet Golf

Contexte

La fédération française de Golf a besoin d'un site web permettant à ses adhérents de voir les classements des principaux tournois, leurs informations et les informations des joueurs professionnels.

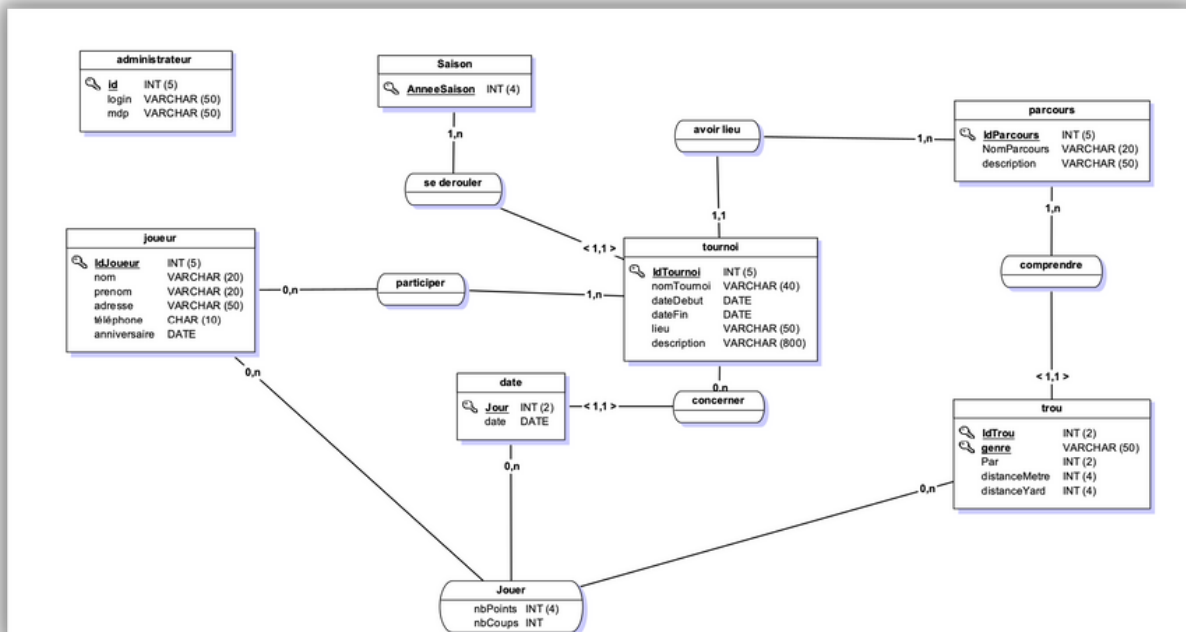
De plus, l'application doit aussi permettre à des administrateurs de créer des nouvelles saisons pour les tournois, y ajouter les scores, ajouter des joueurs et possiblement modifier leurs informations.

Application réalisée

L'application possède les fonctionnalités suivantes :

- du côté des visiteurs :
 - la consultation des informations des joueurs professionnels,
 - la consultation des classements des principaux tournois professionnels (catégorie Homme ou Femme),
 - les informations des tournois, avec leurs parcours et les trous.
- Du côté des administrateurs :
 - la consultation de la liste des joueurs
 - la modification et la suppression de joueurs,
 - la création d'un tournoi pour une nouvelle année,
 - l'ajout des scores d'un tournoi,
 - l'ajout de comptes administrateurs.

Modèle conceptuel de données (MCD) :



Technologies utilisées

Front-end :



React est une bibliothèque JavaScript libre orientée front-end. Elle permet la création d'applications mono-page via la création de composants dépendant d'un état et générant du HTML à chaque changement d'état.



JavaScript est un langage de programmation qui permet d'implémenter des mécanismes complexes sur une page web.

Ce langage peut être utilisé autant pour du **frontend** (animations) que pour du **backend** (NodeJS par exemple).

Back-end :



NodeJS est un environnement d'exécution open-source, multi-plateforme, qui permet aux développeuses et développeurs de créer toutes sortes d'applications et d'outils côté serveur en JavaScript.

Express est le framework le plus populaire dans Node, il permet le traitement de différentes requêtes HTTP répondant à différentes URL (par le biais des routes) principalement.

SOMMAIRE

1 - Arborescence du site

2 - Informations nécessaires à la bonne compréhension

3 - Gestion des données

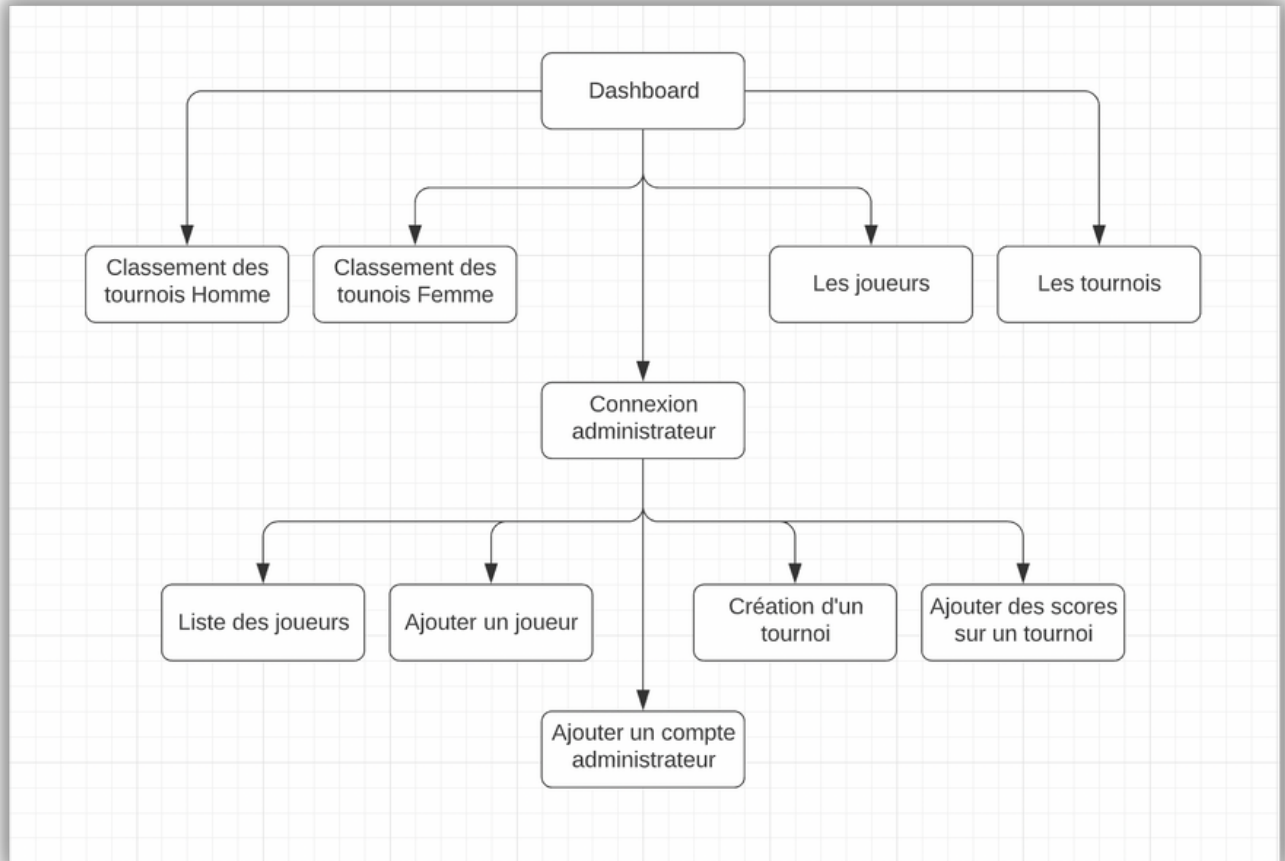
3.1 - Récupération à partir de la base de données

3.2 - Récupération de données envoyées du back dans le front

3.3 - Envoi de données du front au back

3.4 - Envoi de données du back à la base de données

1) Arborescence du site



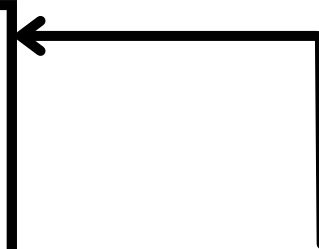
2) Informations nécessaires à la bonne compréhension

Récupération de données à partir de la base de données :

En JavaScript, lorsque nous récupérons des données à partir de la base, nous les récupérons directement sous forme d'objet, chaque ligne est un objet et le tout est contenu dans un tableau.

Exemple :

```
1  {
2
3      "idJoueur": 1,
4      "nom": "Lim",
5      "prenom": "Pascal",
6      "adresse": "8 rue du templier",
7      "telephone": "0678978767",
8      "anniversaire": "2002-10-05T22:00:00.000Z",
9      "genre": "Homme"
10 } ,
11 {
12     "idJoueur": 2,
13     "nom": "Duchemann",
14     "prenom": "Alexis",
15     "adresse": "5 rue du chemin",
16     "telephone": "0612345678",
17     "anniversaire": "2002-10-29T23:00:00.000Z",
18     "genre": "Homme"
19 },
20 {
21     "idJoueur": 3,
22     "nom": "Balde",
23     "prenom": "Alpha",
24     "adresse": "Rue de ménilmontant",
```



Cet objet correspond à une ligne de la table Joueur.

Il n'y a donc aucun intérêt de créer des classes pour chaque table étant donné que tout ce que l'on récupère est déjà un objet.

Connexion à la base de données :

```
42  ⚡ Fonction pour créer la connexion avec la base de données
43  const getConnection = async () => {
44      const db = await mysql.createConnection({
45          host: "localhost",
46          user: "root",
47          password: "root",
48          database: "projet-golf",
49      });
50      return db;
51  };
```

Pour se connecter à la base de données, la fonction de mysql `createConnection()` nous permet de le faire.

Ici, la constante `db` correspond à la connexion à la BDD.

Fonction asynchrone :

Les fonction asynchrone sont des fonctions qui s'exécute de façon asynchrone grâce à la boucle d'évènements en utilisant les promesses (promise) comme valeur de retour.

Une fonction asynchrone peut contenir une expression **await** qui interrompt l'exécution du code tant que la promesse n'est pas résolue. Ensuite, la fonction reprend et renvoie la valeur de résolution.

Sur la fonction au dessus pour se connecter à la base de données, la fonction est **asynchrone** : la constante **db** contenant **await** attend donc une promesse, pendant ce temps la fonction est interrompue, une fois que la promesse est résolue, la fonction reprend et retourne la connexion à la BDD.

Axios :

Axios est une bibliothèque JavaScript fonctionnant comme un client HTTP. Elle permet de communiquer avec des API en utilisant des requêtes. Comme avec les autres clients HTTP, il est possible de créer des requêtes avec la méthode POST.

Bcrypt :

Bcrypt est un algorithme de hachage qui est évolutif en fonction du serveur qui le lance.

Bcrypt utilise l'algorithme de hachage Eksblowfish. Celui-ci est similaire à l'algorithme Blowfish (un algorithme de chiffrement symétrique), excepté que la phase de planification de la clé de Eksblowfish garantit que tout état sous-jacent dépend à la fois du sel et de la clé (le mot de passe encodé). Ainsi aucun état ne peut être précalculé sans connaître à la fois ces deux éléments. A cause de cette différence, bcrypt est un algorithme de hashage **unidirectionnel** : Vous ne pourrez jamais retrouver le mot de passe sans connaître à la fois le grain de sel, la clé et les différentes passes que l'algorithme à utiliser !

Au sein de ce projet, il me permet de hasher les mots de passes insérés dans la base de données.

3) Gestion des données

3.1) Récupération de données à partir de la base de données :

Exemple :

```
283 app.get("/admin/player/list", async(req,res) => {  
284     const db = await getConnection();  
285  
286     try{  
287         const players = await db.query('SELECT * FROM joueur');  
288  
289         res.send(players[0]);  
290     }catch(error){  
291         console.log(error);  
292     }  
293 });
```

Sur ce bloc de code, nous récupérerons tout les joueur de la table joueur.

La constante db correspond à la connexion à la base de données.

Dans le bloc try/catch, nous récupérerons les joueurs dans une constante en envoyant une requête à la base de données, via la fonction .query() :

db.query(" **REQUETE** ").

Ensuite, pour envoyer au front les données récupérée, on utilise la fonction .send() :

res.send(**LES DONNÉES**)

3.2) Récupération des données envoyées du back dans le front

```
12      const datas = await Axios.get('http://localhost:3001/admin/player/list');
```

Grâce à Axios, nous faisons `Axios.get()` pour effectuer une requête HTTP sur l'url :

`http://localhost:3001/admin/player/list`



Le back-end



La route où l'on souhaite récupérer les données

Comme nous pouvons le voir sur le screenshot du back-end (celui avant celui-là), nous avons récupéré dans le front les données correspondant à la route en bleu, dans la constante **datas**.

3.3) Envoi de données du front au back

Exemple avec l'ajout d'un administrateur :

```
32      Axios.post('http://localhost:3001/admin/register', {  
33          login: login,  
34          password: password  
35      });
```

Pour envoyer des données du front au back, encore grâce à Axios, nous avons simplement à faire un **`Axios.post("URL" , {Données au format JSON})`**.

La fonction `.post()` de Axios nous permet d'envoyer des données au back, pour se faire, nous devons obligatoirement envoyé les données au format **JSON**.

3.4) Envoi de données du back à la base de données

```
263 app.post("/admin/register", async(req,res) => {
264   const db = await getConnection();
265
266   const login = req.body.login;
267   const password = req.body.password;
268
269   bcrypt.hash(password, saltRounds, async(err, hash) => {
270     const insertQuery = "INSERT INTO administrateur (login,mdp) VALUES(?,?)";
271
272     try {
273       const result = await db.query(insertQuery, [login, hash]);
274       res.send(result);
275     } catch (error) {
276       res.send({ message: "ERREUR"})
277       console.log(error)
278     }
279   });
280 }
281 });
```

Ici tout d'abord, nous récupérons les données envoyées du front en assignant à une constante ce qui a été envoyé :

Const login = req.body.login;

Const password = req.body.password;

Explication :

- Req : cela signifie request, nous avons fais une **requête** HTTP post.
- body : cela correspond au front.
- login : cela correspond à l'attribut login des données envoyées au format JSON.

Ensuite, vient le hashage du mot de passe envoyé du front et récupéré ici dans la constante password.

Pour hasher le password, la fonction **.hash()** de **bcrypt** est utilisée, voici sa signature :

bcrypt.hash("le mot de passe", "un nombre ", callback());

Ensuite, j'ai écrit la requête dans une constante au préalable, **insertQuery** (ce n'est absolument pas obligatoire mais ça peut clarifier le code) puis dans un bloc try/catch on effectue l'ajout avec la fonction **.query()**, sous une forme différente ici :

db.query("la requête", [les paramètres])

Ici les paramètres dans le insert sont donc le login et le mot de passe, et pour la requête, nous avons simplement à passer la constante **insertQuery** en premier paramètre de la fonction car cela correspond à une string et plus précisément, la requête INSERT.