



NOCTURN

DEVELOPPEUR

WEB

*Projet de formation
Quentin Terzi*

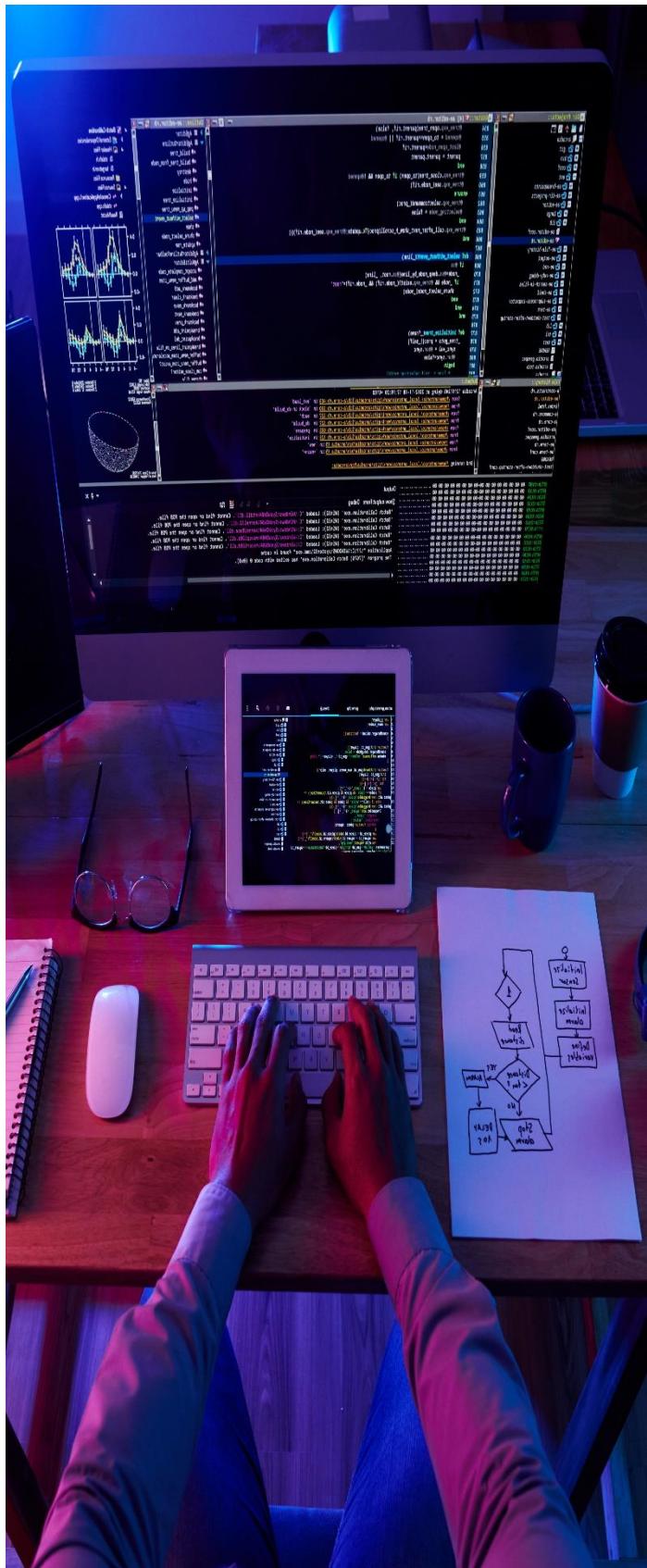


TABLE DES MATIÈRES

remerciments.....	3
Presentation	4
1. Présentation de l'association	5
2. Cahier des charges.....	6
Objectifs.....	6
Les Cibles	6
Les Objectifs Quantitatifs	6
Analyse Marketing	6
Charte Graphique	7
3. outils technique.....	8
Langages utilisés	8
Logiciels utilisés	8
4. Analyse fonctionnelle	9
Maquettage	9
Arborescence.....	12
Diagramme Cas d'utilisations.....	13
Diagramme d'Activité	15
Diagramme de Séquence	16
5. Conception des données	17
Modèle Conceptuel des Données	17
Modèle Logique des Données	18
Création BDD en SQL.....	19
6. Fonctionnalité : Gestion des membres.	20
Introduction	20
Front-end	20
Back-end.....	35
7. Annexes	44
Requêtes modifier et supprimer gestion membres.....	44
Media queries page affichage membre ..	44

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui m'ont accompagnée et soutenue pendant cette formation.

Tout d'abord, merci au corps enseignant d'Adrar pôle numérique pour leurs accompagnement et la transmission de leurs savoirs.

Je souhaite également exprimer ma reconnaissance envers mes collègues de formation pour leur enthousiasme, leur esprit de partage et d'entraide durant cette année.

Je remerci également l'ensemble de l'équipe Firstseller pour leur disponibilité, leur soutien et leur confiance au quotidien durant le stage.

Je n'oublie pas de remercier la Région Occitanie car c'est grâce au soutien qu'elle apporte pour permettre l'égalité des chances que je peux aujourd'hui me présenter au titre professionnel Développeur web et web mobile.

Enfin je remercie Google pour toute l'aide supplémentaire apportée.

PRESENTATION

My name is Quentin, I'm 25 years old, I've been passionate about computer science for 10 years, more precisely about the development of system applications and web applications. I am also passionate about basketball.

I started to learn programming by myself. The first programming languages I learned were "C" and "lua".

I started to learn them in order to code small games on psp in lua and c with a small team of young developers.

I am someone who likes to work with other people. So I think I have developed a good team spirit.

At the level of studies, I did a professional baccalaureate in electronic and digital systems where I acquired good notions of telecommunications networks.

I then tried to do the epitech school where I could deepen my knowledge in C thanks to the system programming under linux, although I failed before the end of the first year.

Then I did some small jobs, one of which was to program excel macros in VBA, then I started the adrar training, where I learn web development.

My ambition is to do an additional year of training in order to become a software designer and I would like, in a few years, to create a small start-up.

1. PRESENTATION DE L'ASSOCIATION

Nocturn est une association évènementielle Toulousaine créé en 2021. Elle a pour but d'organiser des évènements réunissant toutes personnes amatrice de drum and bass et de street culture. Nocturn fait venir différents dj et artistes locaux pour présenter leur art et faire s'ambiancer les festivaliers avec de la musique électronique. Elle propose donc lors de ses soirées différents dj set axé principalement sur la drum and bass, mais pas seulement. L'association ouvrira lors de ses évènements, différents stands, comme par exemple, des stands de vêtements, food truck, tatouages, tableau d'art, graffiti, tous se rapprochant du street art et de la street culture. Ils pourront donc acheter certains biens ou seulement venir observer et danser. D'autres association existent et proposent des soirées du même type mais nous avons en plus ce côté street culture avec les différents stands et style de musical qui permette de nous différencier des autres.

2. CAHIER DES CHARGES

Objectifs

Le projet sera un site internet qui pourra être consultable sur pc et mobile. L'objectif est de gagner en visibilité sur les évènements ainsi que sur l'actualité de l'association. Le site permettra aux gens de connaître l'endroit et le jour des évènements avant la date butoir et pourront acheter leurs billets avant, afin de parer à l'éventualité d'un sold out. Ils seront aussi au courant des différents stands présents lors des évènements et auront la possibilité d'acheter des œuvres exposées lors des évènements.

Les Cibles

Nous visons toutes personnes majeure, amatrice de street culture ainsi que de musiques underground. Les évènements seront ouverts à tous les 18+

Les Objectifs Quantitatifs

Le site web sera composé de 6 pages.

*Actualité : permet de se tenir informer des différentes nouvelles concernant l'association.

*Présentation : présente l'association et ses prestations et raconte l'histoire de l'association

*Membres : Présente les différents membres ainsi que leurs rôles et laisse la possibilité d'accéder au page personnel de chaque membre.

*Evènements : la page donnera des infos sur la date et l'heure ainsi que le lieu des évènements et donnera une description de ces évènements. L'utilisateur pourra acheter les places via cette page.

*Boutique : Page proposant divers articles exposés lors des évènements passé et mis en vente par leurs auteurs laissant la possibilité aux utilisateurs de les acheter

*Connexion/Inscription : Permet aux utilisateurs de s'inscrire et se connecter au site.

Analyse Marketing

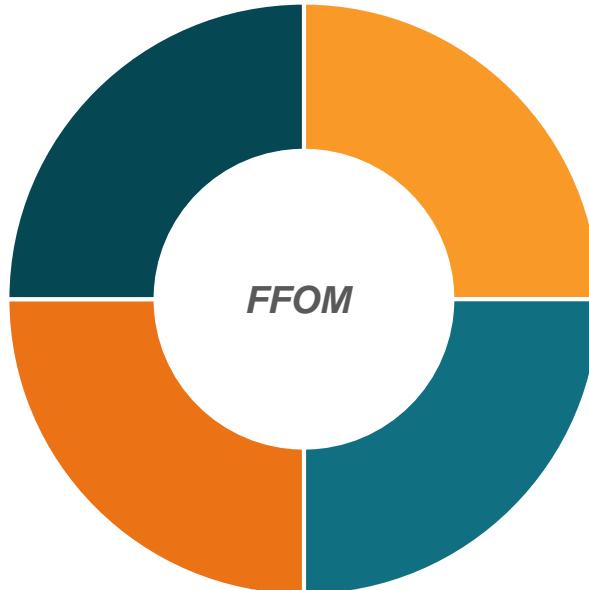
Voici un SWOT représentatif du projet :

FORCES

- Information sur les évènements
- Acheter des billets en ligne
- Stands lors des évènements

OPPORTUNITÉS

- Drum and Bass populaire sur Toulouse
- Culture street-art



FAIBLESSES

- Peu de partenariat avec l'association pour avoir différents stands
- Moindre capacité financière.

MENACES

- D'autres associations centrées sur la DnB
- Maintenir une continuité dans les évènements.

Charte Graphique

Pour le moment aucune contrainte graphique n'est imposé.

Voici le logo :



3. OUTILS TECHNIQUE

Langages utilisés

- **HTML** : Langage permettant la structuration des pages web
- **CSS** : Langage utilisé pour la mise en page et le style des pages web
- **BOOTSTRAP** : Framework CSS permettant de gérer la responsivité et certains styles.
- **JAVASCRIPT (VUE.JS)** : Langage permettant de rendre dynamique les pages web. Je l'ai utilisé avec le framework Vue.js
- **PHP** : Langage de programmation back-end.
- **SQL** : Langage de programmation permettant de mettre en place des requêtes pour communiquer avec la BDD

Logiciels utilisés

- **STAR UML** : Logiciel de modélisation de diagrammes UML
- **BALSAMIQ** : Pour la réalisation du maquettage
- **LOOPING** : Logiciel de conception MLD et MCD
- **VS CODE** : IDE utilisé pour le code
- **GIT** : Logiciel de gestion de version utilisé afin de sauvegarder le code.
- **NODE.JS** : Environnement d'exécution JS, permettant de créer des applications rapides et évolutives côté serveur et en réseau.
- **GLOOMAPS** : Permet de réaliser l'arborescence du projet.
- **XAMP** : Permet la mise en place d'un serveur web local

4. ANALYSE FONCTIONNELLE

Maquettage

Le maquettage est une méthode de conception d'interface qui permet de proposer des interfaces conformes aux attentes et besoins du client.

Il correspond à une esquisse, voire un prototype d'un site internet.

La réalisation des maquettes se concentre principalement sur l'aspect graphique et le design d'un site web et de ses interfaces.

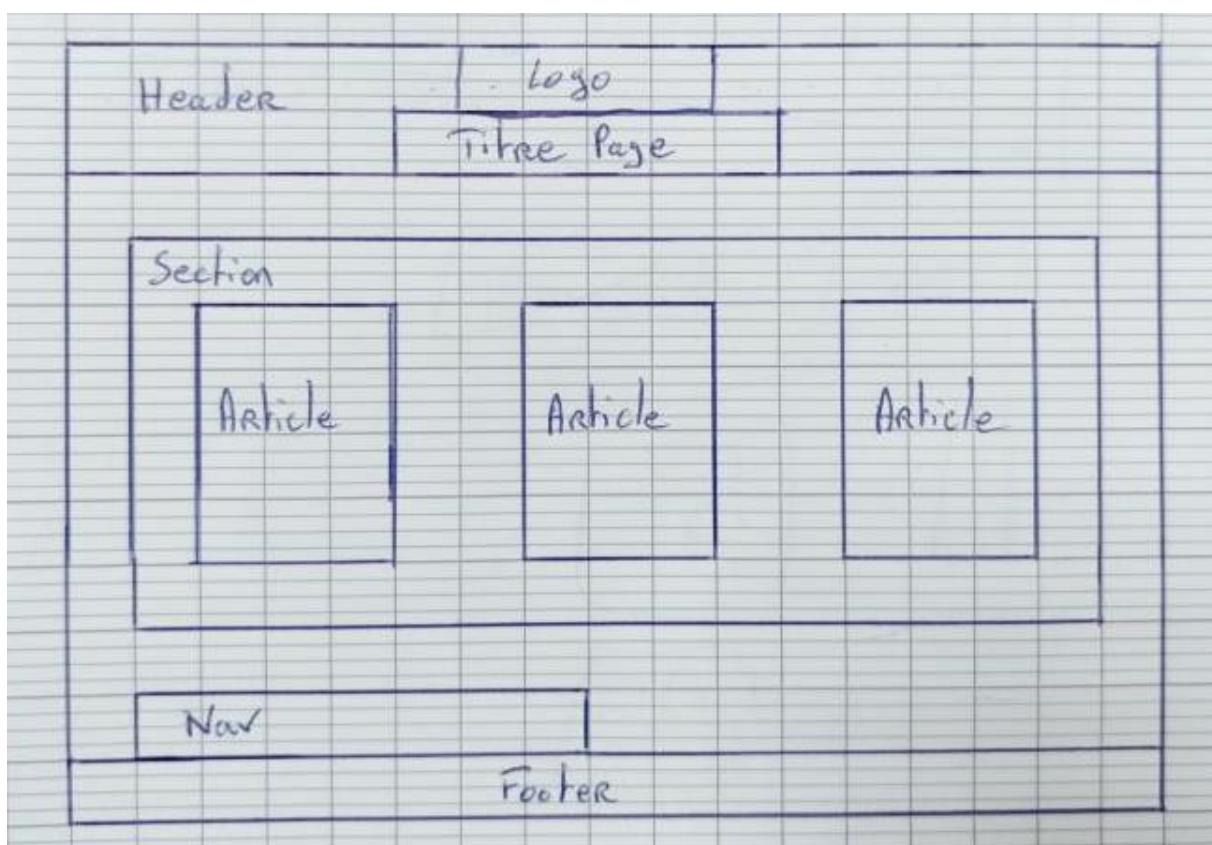
Réaliser les maquettes d'un site internet est donc une phase de réflexion dans l'étape de la conception.

Je vais vous présenter le zoning et le wireframe de la page qui affichera les membres de l'association avec leur rôle et un lien vers leurs réseaux sociaux.

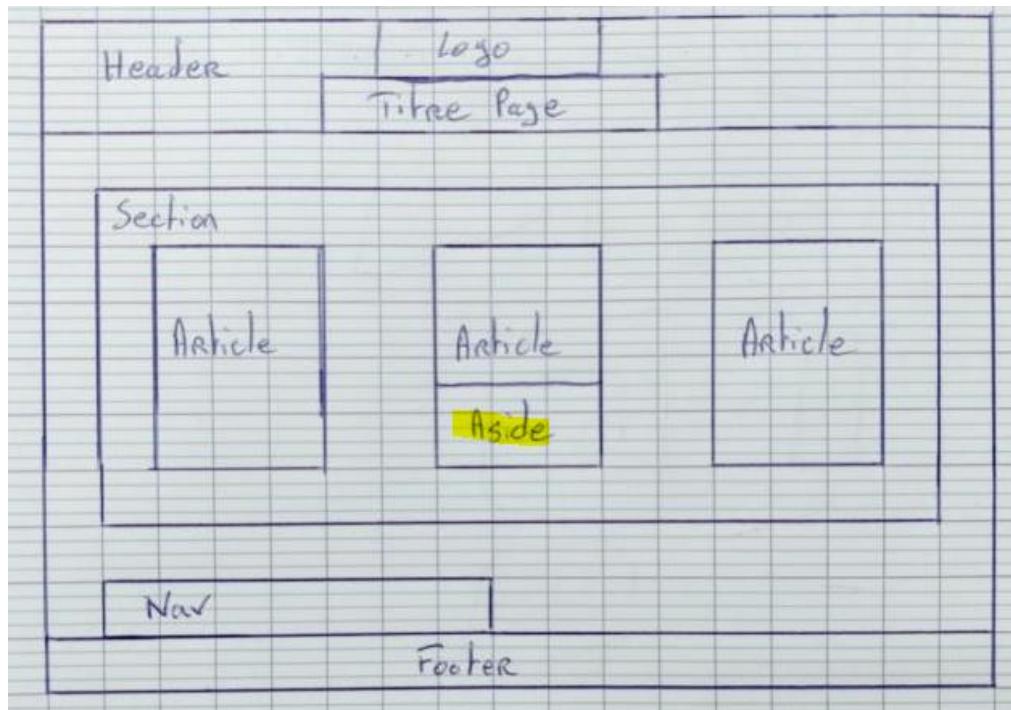
Zoning :

Le zoning permet de penser à la structure du site et à la hiérarchisation des contenus. Il schématise les pages web du site internet à l'aide de boîtes ou de blocs, dans l'optique de montrer les principales zones du contenu et les grandes fonctionnalités.

Elle permet donc de présenter les différents éléments avec et sur lesquels les utilisateurs vont pouvoir réagir.



Lors du survol d'un article, qui affiche la photo du membre, un aside doit apparaître afin d'afficher les liens vers les réseaux sociaux du membre.



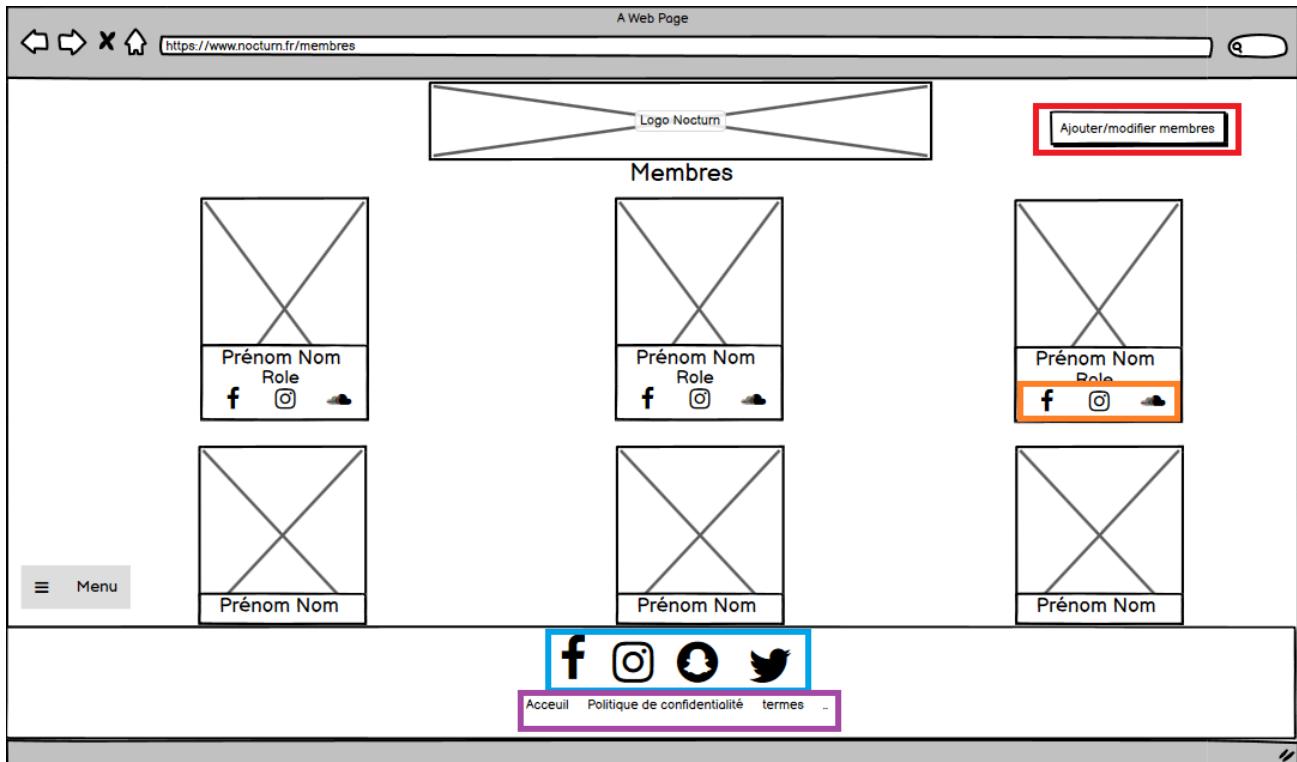
Wireframe :

Le zoning et le wireframe sont deux choses différentes mais complémentaires :

- **Le zoning** permet de visualiser par bloc, les surfaces dédiées à chaque fonction.
- **Le wireframe** contient en plus des éléments d'information.

Le wireframe permet donc la visualisation :

- Des zones de texte,
- L'emplacement des images, des vidéos, des liens,
- Des différents éléments graphiques

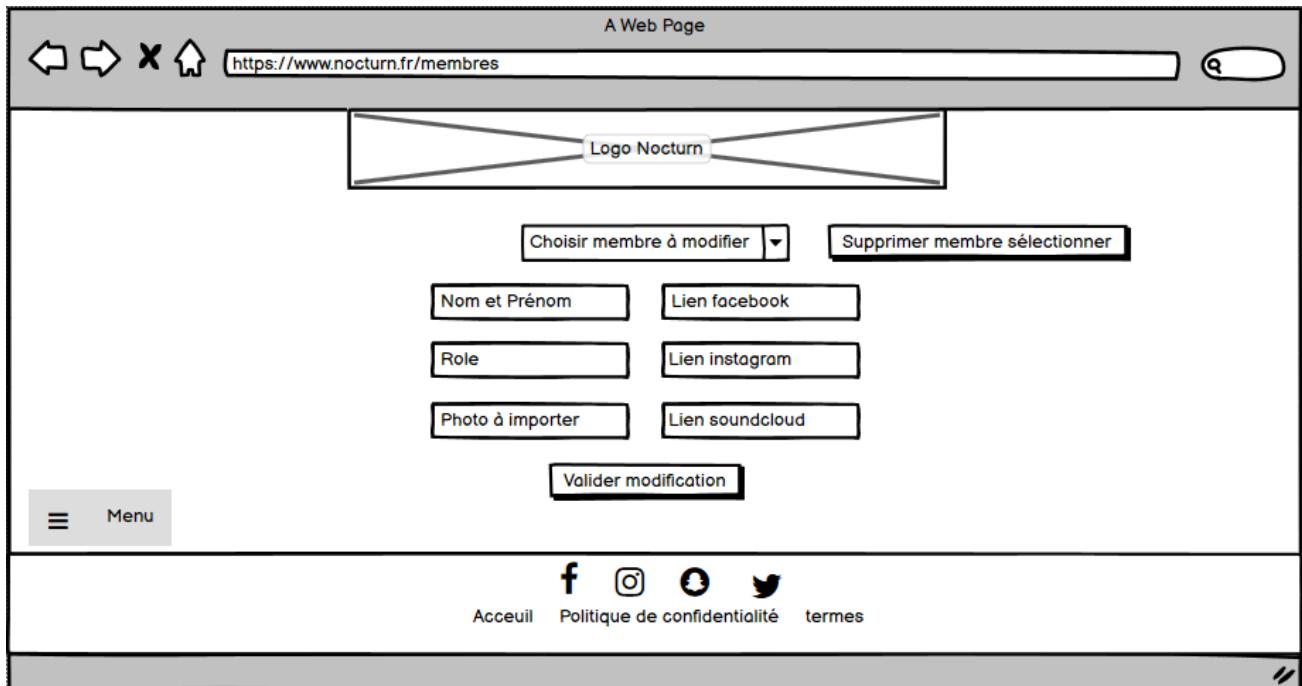


Les membres auront une photo d'eux, et lors du survol d'un des membres nous afficherons son prénom, son nom ainsi que ses réseaux sociaux.

- **En rouge** : Un bouton qui redirige vers la page de gestion des membres (ajouter, modifier ou supprimer un membre). Voir wireframe ci-dessous.
- **En orange** : Des liens redirigeant vers les réseaux sociaux du membre sélectionné
- **En bleu** : Des liens redirigeant vers les réseaux sociaux de l'association.
- **En violet** : Des liens redirigeant vers d'autres pages du site.

Le menu sera un menu déroulant horizontal.

Voici le wireframe de la gestions des membres :



Ici nous avons une Selectbox qui impactera sur les buttons supprimer et valider. En effet si aucun membre n'est sélectionné, le button « Valider modification » sera remplacé par le button « Ajouter membre » et le button « Supprimer membre » sera masqué. En revanche si un membre est sélectionné les buttons « Valider modification » et « Supprimer membre » seront de nouveau affichés.

Mis à part le champ « Photo à importer » qui sera un input de type file, les 5 autres champs seront des input de type texte.

Arborescence

L’arborescence se définit comme un mécanisme de répartition de données et d’informations selon la hiérarchie. L’arborescence d’un site représente ainsi son architecture afin de structurer et d’organiser ses contenus de façon optimale.

Elle permet donc d’optimiser l’ergonomie du site et de faciliter la navigation des utilisateurs sur le site. Une bonne arborescence de site joue un rôle clé dans le positionnement SEO sur les moteurs de recherche.

Elle part de la page d'accueil et est généralement structurée en rubriques, sous rubriques et pages.

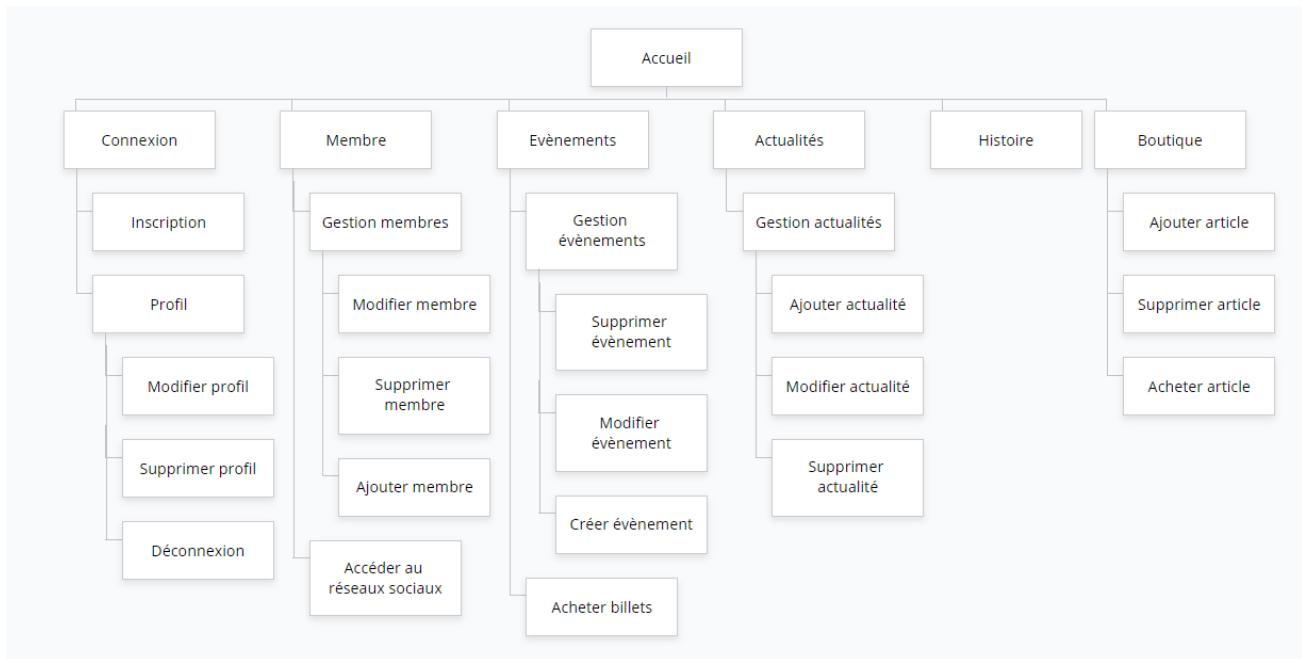


Diagramme Cas d'utilisations

Un cas d'utilisation désigne la manière d'utiliser un système. Les use cases décrivent les fonctions générales et la portée d'un système. Ils décrivent ce que le système fait et comment les acteurs l'utilisent, mais ne montrent pas comment le système fonctionne en interne.

Ils expriment les interactions d'acteurs avec un système et les fonctionnalités ou actions réalisées par ce système sous forme schématique (diagrammes).

Un diagramme use case se compose :

- **Du système** : ensemble de cas d'utilisation, il contient les cas d'utilisation mais pas des acteurs
- **D'acteurs** : Un acteur représente un rôle d'un utilisateur qui interagit avec le système que vous modélisez. L'utilisateur peut être un utilisateur humain, une organisation, une machine ou un autre système externe. Il en existe 4 types différents.
- **De cas d'utilisations** : Un cas d'utilisation décrit une fonction qu'un système exécute pour atteindre l'objectif de l'utilisateur et doit renvoyer un résultat observable qui est utile pour celui utilisant le système.
- **De relations** : Il existe 4 types de relations différentes :
 - 1) *Relations d'association* est un trait simple entre un acteur et un cas d'utilisation.
 - 2) *Relations de généralisation* est une relation dans laquelle un élément de modèle (l'enfant) est basé sur un autre élément de modèle (le parent) et indique que l'enfant reçoit tous les attributs, opérations et relations qui sont définis dans le parent.

- 3) *Relations d'inclusion* est une relation dans laquelle un cas d'utilisation (le cas d'utilisation de base) inclut les fonctionnalités d'un autre cas d'utilisation (le cas d'utilisation inclus).
- 4) *Relations d'extension* spécifie qu'un cas d'utilisation (l'extension) étend le comportement d'un autre cas d'utilisation (la base). Cela révèle des détails sur un système qui sont généralement masqué dans un cas d'utilisation.

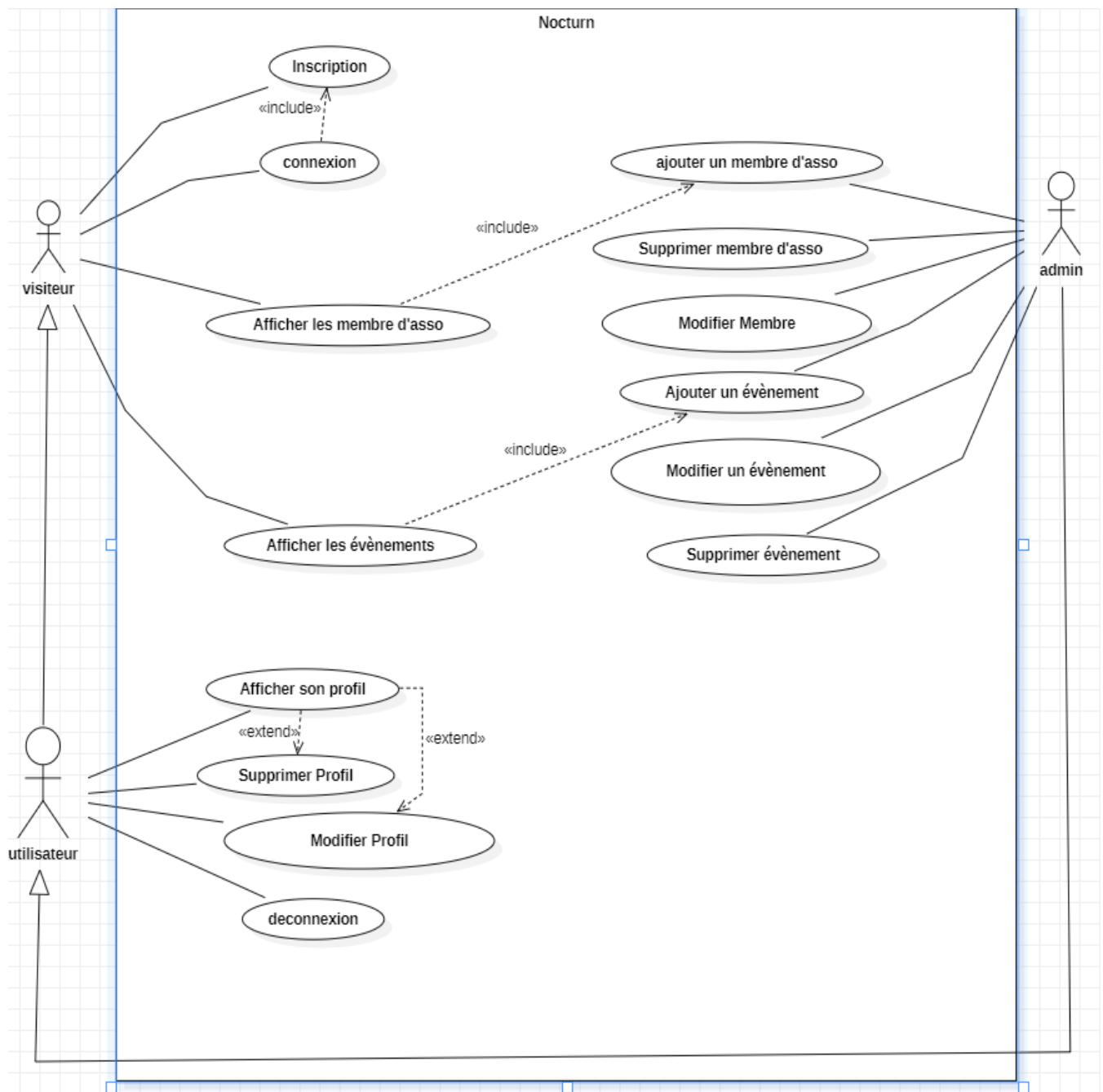


Diagramme d'Activité

Un diagramme d'activité correspond à la traduction algorithmique d'un cas d'utilisation. Ils montrent les flux entre les actions dans une activité et fournit une vue précieuse sur les flux d'information qui sont échangés entre les acteurs.

Un diagramme d'activités peut être divisé en colonnes ("swimlanes") où chaque colonne identifie le domaine de responsabilité d'un acteur.

Un diagramme d'activité est représenté par des formes reliées par des flèches.

Les flèches vont du début à la fin de l'activité et représentent l'ordre séquentiel des activités effectuées.

Les cercles noirs représentent un état initial du flux de travail. Un cercle noir entouré indique un état final.

Les rectangles arrondis représentent les actions effectuées, qui sont décrites par du texte à l'intérieur de chaque rectangle.

Une forme de diamant est utilisée pour représenter une décision, qui est un concept de diagramme d'activité clé.

Ci-dessous le diagramme d'activité traduit la cas d'utilisation « Ajouter un membre de l'association »

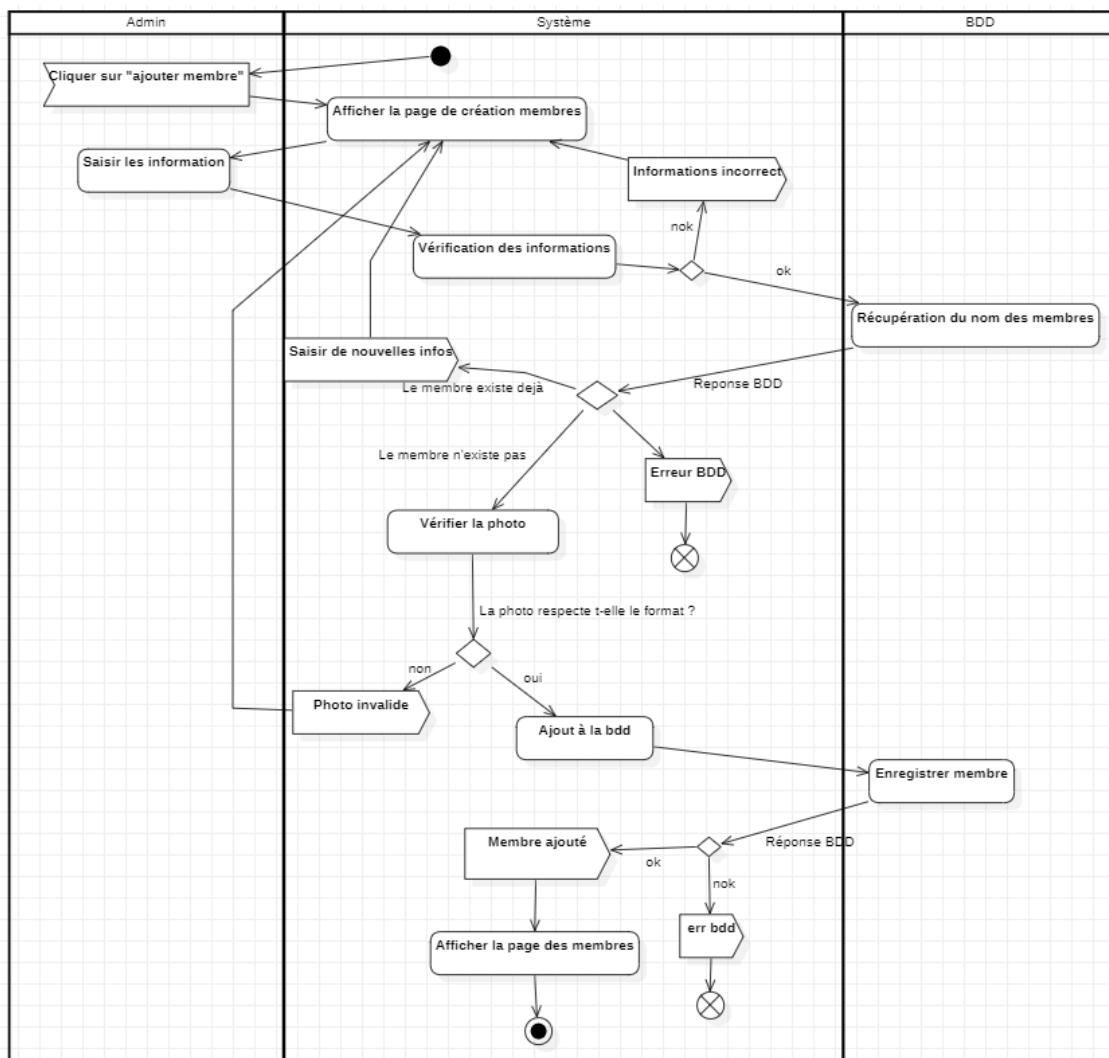


Diagramme de Séquence

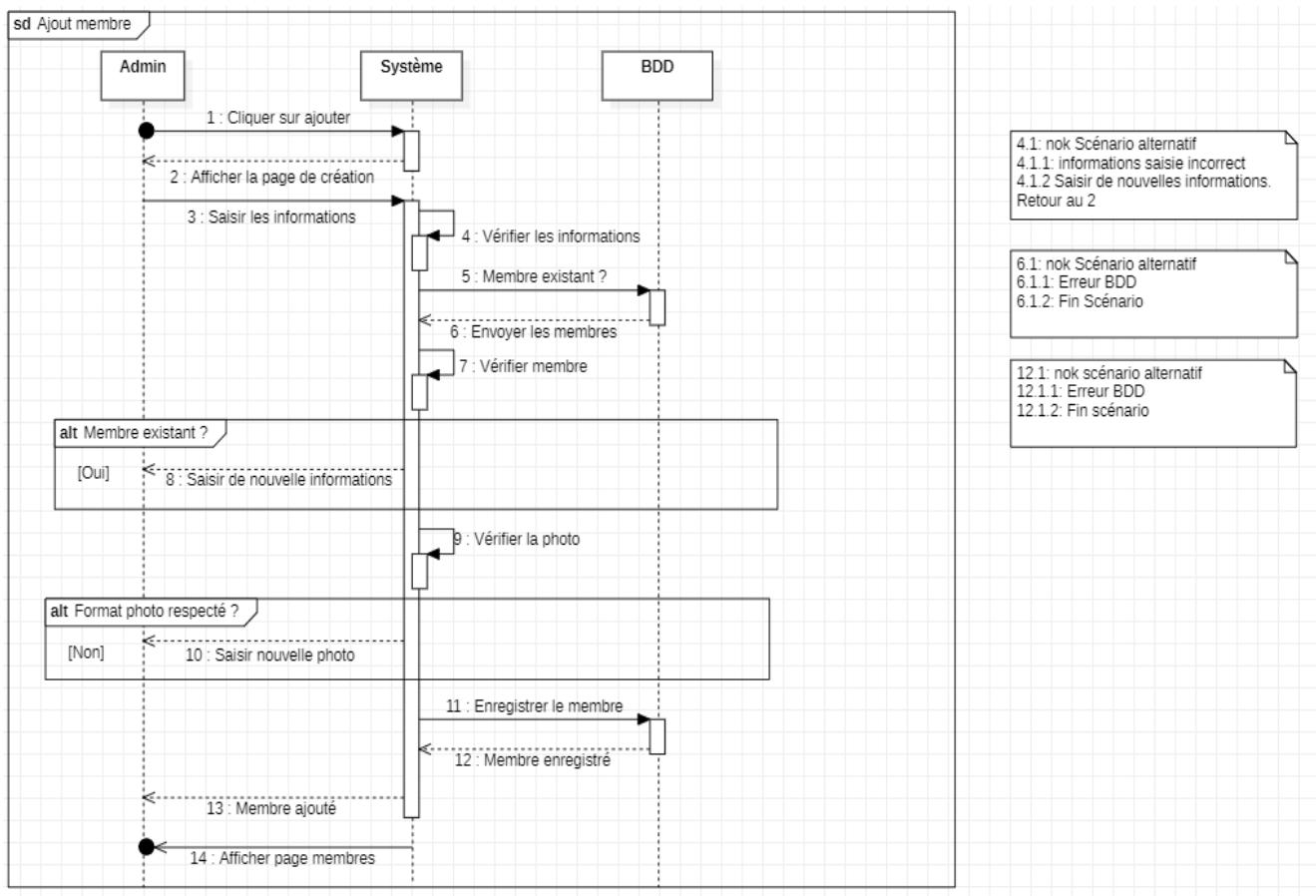
Le diagramme de séquences est une représentation graphique et chronologique modélisant les interactions entre les acteurs et le système dans un cas d'utilisation unique.

Ils illustrent la manière dont les différentes parties d'un système interagissent entre elles pour exécuter une fonction, et l'ordre dans lequel les interactions se produisent lorsqu'un cas d'utilisation particulier est exécuté.

La dimension verticale du diagramme représente le temps, permettant de visualiser l'enchaînement des actions dans le temps. Les périodes d'activité des objets sont symbolisées par des rectangles, et ces objets dialoguent à l'aide de messages.

Un diagramme de séquence comprend un groupe d'objets, représentés par des lignes de vie, et les messages que ces objets échangent lors de l'interaction.

Ci-dessous le diagramme de séquence qui traduit la cas d'utilisation « Ajouter un membre de l'association »



5. CONCEPTION DES DONNEES

Modèle Conceptuel des Données

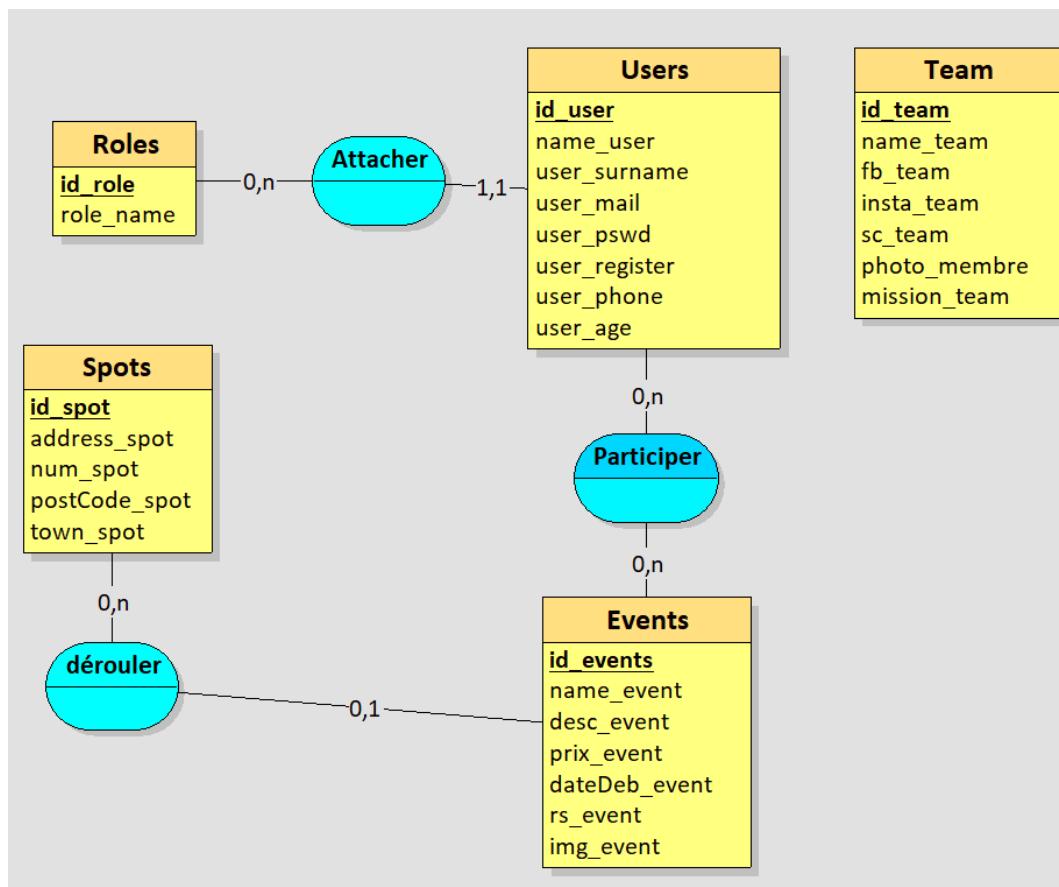
Le MCD permet d'établir une représentation claire des données du système d'information et définit les dépendances fonctionnelles de ces données entre elles.

Il décrit donc les données utilisées par le système d'information et leurs relations.

Les informations sont représentées logiquement en utilisant un ensemble de règles :

- Les entités (1 rectangle = 1 objet)
- Les propriétés (la liste des données de l'entité)
- Les associations qui expliquent et précisent comment les entités sont reliées entre elles (les ovales avec leurs « pattes » qui se rattachent aux entités)
- Les cardinalités (les petits chiffres au dessus des « pattes »).

Ci-dessous le MCD établie pour le site Nocturn :



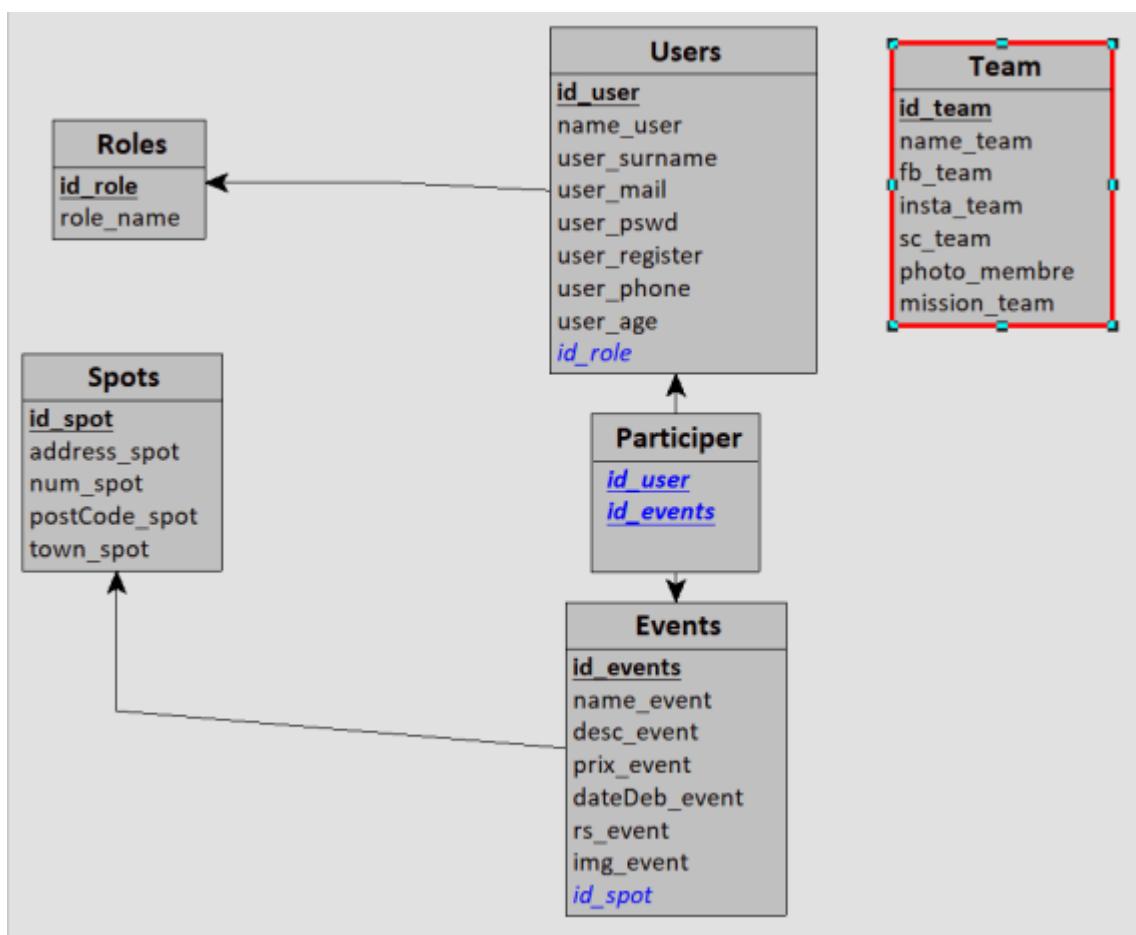
Modèle Logique des Données

Le MLD consiste d'abord à transformer toute entité en table, avec l'identifiant comme clé primaire, puis à observer les valeurs prises par les cardinalités maximums de chaque association pour représenter celle-ci soit (ex : card. max 1 [1-1 ou 0-1]) par l'ajout d'une clé étrangère dans une table existante, soit (ex : card. max n [0-N ou 1-N]) par la création d'une nouvelle table dont la clé primaire est obtenue par concaténation de clés étrangères correspondant aux entités liées.

Il permet de décrire la structure de données utilisée, de préciser le type de données utilisées lors des traitements et de connaître la structure de la base de données

Chaque ligne représente une table

- C'est toujours le nom de la table qui est écrit en premier
- Les champs sont listés entre parenthèses et séparés par des virgules
- Les clés primaires sont soulignées et placées au début de la liste des champs



Création BDD en SQL

```
1  /*DROP DATABASE nocturn;*/  
2  CREATE DATABASE nocturn;  
3  USE nocturn;  
4  CREATE TABLE roles(  
5    id_role INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
6    name_role VARCHAR(20) NOT NULL UNIQUE  
7  );  
8  CREATE TABLE users(  
9    id_user INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
10   name_user VARCHAR(50) NOT NULL,  
11   surname_user VARCHAR(50) NOT NULL,  
12   mail_user VARCHAR(50) NOT NULL UNIQUE,  
13   psswd_user VARCHAR(100) NOT NULL,  
14   phone_user VARCHAR(20) UNIQUE,  
15   age_user DATE,  
16   register_user DATE,  
17   id_role INTEGER,  
18   FOREIGN KEY(id_role) REFERENCES roles(id_role)  
19  );  
20  CREATE TABLE spots(  
21   id_spot INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
22   num_spot VARCHAR(10),  
23   address_spot VARCHAR(75) NOT NULL,  
24   postCode_spot VARCHAR(25),  
25   town_spot VARCHAR(50) NOT NULL  
26  );  
27  CREATE TABLE events(  
28   id_event INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
29   name_event VARCHAR(50) NOT NULL,  
30   desc_event TEXT,  
31   prix_event DECIMAL(3,2) NOT NULL,  
32   dateDeb_event DATE,  
33   rs_event VARCHAR(75) NOT NULL,  
34   img_event VARCHAR(50),  
35   id_spot INTEGER,  
36   FOREIGN KEY(id_spot) REFERENCES spots(id_spot)  
37  );  
38  CREATE TABLE participer(  
39   id_user INTEGER,  
40   id_event INTEGER,  
41   PRIMARY KEY (id_user, id_event),  
42   FOREIGN KEY(id_user) references users(id_user),  
43   FOREIGN KEY(id_event) references events(id_event)  
44  );  
45  CREATE TABLE team(  
46   id_team INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
47   name_team VARCHAR(75) NOT NULL,  
48   fb_team VARCHAR(75),  
49   insta_team VARCHAR(75),  
50   sc_team VARCHAR(75),  
51   photo_team VARCHAR(50) NOT NULL,  
52   mission_team VARCHAR(35)  
53  );
```

6. FONCTIONNALITE : GESTION DES MEMBRES

Introduction

Pour ce projet j'ai utilisé le framework vue js car j'ai apprécié son système de composants qui sont réutilisables et vivent indépendamment les uns des autres ainsi que celui des templates HTML. J'ai également aimé le système de data-binding et celui du rendu déclaratif, ce qui permet de rendre des données au DOM. Les accolades doubles sont utilisées comme espaces réservés pour interpoler les données requises dans le DOM. J'ai également utilisé les librairies Vue Router, Vuex et Axios.

- **Vue routeur** : Va me permettre de synchroniser des URL avec des vues de mon site. Il va me permettre de gérer mes pages, créer des liens et passer des paramètres à mes vues. Les champs sont listés entre parenthèses et séparés par des virgules
- **Vuex** : C'est un gestionnaire d'état. Il sert de zone de stockage de données centralisée et ne change les données que de manière prévisible. Il me permet donc d'avoir accès à des données dans tous les composants du site en étant sûr de leur intégrité et de leur validité.
- **Axios** : 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. Je vais pouvoir faire des requêtes HTTP pour récupérer ou sauvegarder des données en « contactant » mon back-end.

Je vais vous présenter 2 cas d'utilisation :

- Afficher les membres
- ajouter des membres

J'ai donc démarré mon projet en Vue CLI qui est une interface en ligne de commande et qui va me fournir une base standard facilitant le démarrage de projet. Cela va m'assurer que tous les outils dont j'ai besoin fonctionnent ensemble et de faire abstraction de tous les détails de configuration essentiels. Je pourrais également créer des fichiers avec l'extension « .vue ».

Dans un premier temps je vais présenter le Front-end de mon site en vous montrant le composant « Footer ». Je vais montrer comment j'ai intégré ce composant à mes différentes vues. De plus je vais expliquer le routeur que j'ai mis en place, à fin d'accéder à mes vues, en particulier la vue « Membre », grâce à des URL précise. Enfin, une fois l'analyse du template HTML et du CSS terminé, je parlerais du script JS et poursuivrais avec la mise en place de vuex pour gérer les données. J'en viendrais à appeler mes api via Axios.

Donc, dans un second temps, je vais présenter le Back-end en commençant par la structure du MVC. Les différentes étapes entre les controllers et les models, qui permettront de récupérer ou modifier des données en BDD, seront abordées.

Front-end

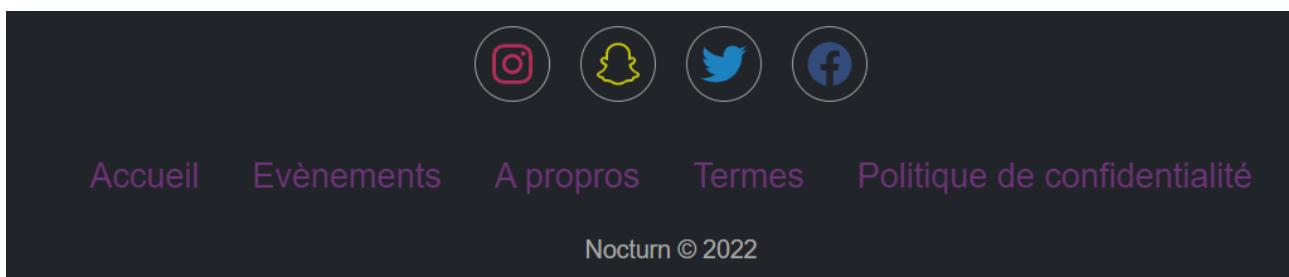
Lorsque je créais un projet Vue CLI celui-ci me créé un dossier de projet avec l'architecture suivante :

```
> node_modules  
> public  
└ src  
    └ assets  
    └ components  
    └ php  
        └ router  
    └ store  
    └ views  
    └ App.vue  
    └ main.js  
    └ .browserslistrc  
    └ .eslintrc.js  
    └ .gitignore  
    └ babel.config.js  
    └ jsconfig.json  
    └ nocturnDB.sql  
    └ package-lock.json  
    └ package.json  
    └ README.md  
    └ vue.config.js
```

- **node_modules** : contient l'ensemble des dépendances qui permettent à mon application de faire tout ce qu'elle fait. Il n'est généralement pas attaché au répertoire mais directement géré par npm.
- **public** : ce répertoire contient le favicon.ico et le fichier index.html de base qui serviront à générer le reste de l'application.
- **src** : ce répertoire est celui dans lequel je passerais 99 % de mon temps, car la quasi-totalité du code y sera située.
- **assets** : il s'agit du répertoire dans lequel je placerais les images et les autres ressources obligatoires
- **router** : Dossier contenant le fichier de configuration du routeur installé avec vue router.
- **store** : dossier contenant le fichier de configuration du gestionnaire d'état vueX.
- **components** : Dossier contenant les composants de l'application.
- **php** : Contiendra le code php en MVC, pour récupérer et envoyer des données en BDD.
- **.gitignore** : ce fichier contient une liste de fichiers et/ou de répertoires qui ne seront pas attachés à un repository.
- **main.js** : c'est ici que sont définies les options de configuration plus high-level de Vue. C'est ici qu'on instancie la nouvelle application vuejs.
- **package.json** : il s'agit du fichier de configuration de base du projet. Il comprend des métadonnées comme le nom et la version du projet, mais il contient également des informations essentielles comme les scripts pouvant être exécutés (comme serve, build, lint) et les dépendances requises pour votre projet.

serve - script qui permet de lancer une environnement de développement en local avec node.js
build - script permet de créer la version finale qui sera livrée à un client ou à l'utilisateur.

Je vais commencer par présenter le composant Footer en montrant son code html puis son CSS. A la fin j'aurais mon composant réutilisable sur mes vues.



Je vais expliquer plus précisément comment j'ai fait la barre des icônes réseaux sociaux.

src
> assets
└ components
└ Actu.vue
└ Event.vue
└ Footer.vue
└ Header.vue
└ HelloWorld.vue
└ Nav.vue
> php
> router
> store
> views

J'ai d'abord créé un fichier Footer.vue dans le dossier components, qui est le dossier qui contiendra tout mes composants, dont la barre de navigation(Nav.vue) par exemple.

Ensuite j'ai pu écrire mon code HTML dans les balises « template » qui sont les balises qui contiennent le code HTML du composant ou de la vue, et qui sera affichés quand ils seront appelés.

Mon footer est composé de 3 blocs principaux. En premier, la ligne qui contient les liens et les icônes réseaux sociaux. En second la ligne qui contient les liens qui redirige vers d'autres vues du site. En dernier, le paragraphe copyright. Dans mes balises footer, j'ai donc mis les liens des réseaux sociaux dans une div pour les séparer de liste « ul » et d'en faire 2 blocs distincts

```
1 <template>
2   <footer class="footer-basic py-2 bg-dark fixed-bottom">
3     <div class="social">
4       <a href="https://www.instagram.com/?hl=fr">
5         <fa :icon="['fab', 'instagram']" style="color: #E1306C"/>
6       </a>
7       <a href="https://www.snapchat.com/l/fr-fr/download">
8         <fa :icon="['fab', 'snapchat']" style="color: #FFFC00"/>
9       </a>
10      <a href="https://twitter.com/?lang=fr">
11        <fa :icon="['fab', 'twitter']" style="color: #1DA1F2"/>
12      </a>
13      <a href="https://fr-fr.facebook.com/">
14        <fa :icon="['fab', 'facebook']" style="color: #3b5998"/>
15      </a>
16    </div>
```

Sur la photo si dessus, on peut voir que j'ai attribué la classe « footer-basic » à la balise « footer »(l.2) à fin de pouvoir facilement donner du style à l'ensemble de mon footer grâce au CSS. Mais j'ai également ajouté du bootstrap dans la classe. Mis à part que bootstrap gère la responsivité, grâce à bootstrap je peux venir ajouter des classes dans des balises html, ce qui

affectera le style du contenu de ces balises.

Par exemple :

py-2 -> padding au top et au bottom de taille 2

bg-dark -> pour donner au footer une couleur noir

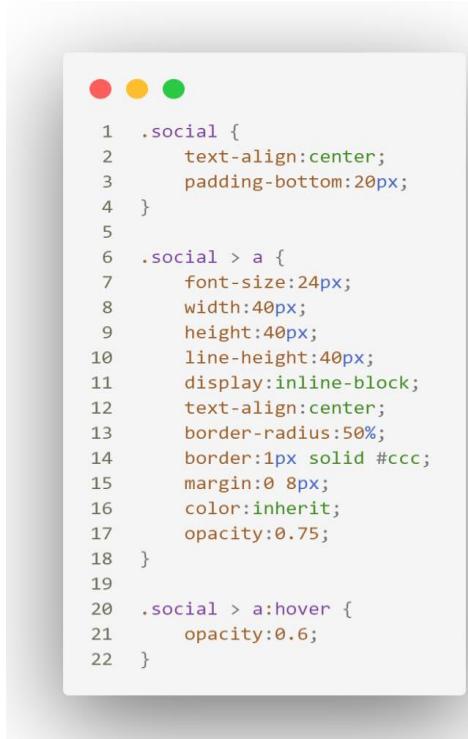
fixed-bottom -> pour fixer le footer au bas de la page.

J'ai aussi donné des noms de classe à « div »(l.3) et « ul »(l.1) non identique pour pouvoir les différencier et appliquer des styles indépendamment l'un de l'autre.

```
1 <ul class="list-inline">
2   <li class="list-inline-item">
3     <a href="#">Accueil</a>
4   </li>
5   <li class="list-inline-item">
6     <a href="#">Événements</a>
7   </li>
8   <li class="list-inline-item">
9     <a href="#">A propos</a>
10  </li>
11  <li class="list-inline-item">
12    <a href="#">Termes</a>
13  </li>
14  <li class="list-inline-item">
15    <a href="#">Politique de confidentialité</a>
16  </li>
17 </ul>
18 <p class="copyright">Nocturn © 2022</p>
19 </footer>
```

A contrario pour les balises « li »(l.2) j'y ai entré le même nom de classe pour que le style s'applique à chaque éléments de la liste.

Grace à la classe « social » dans la div je peux utiliser le sélecteur de classe en CSS(l.1) pour lui mettre un style.



```
1 .social {  
2     text-align:center;  
3     padding-bottom:20px;  
4 }  
5  
6 .social > a {  
7     font-size:24px;  
8     width:40px;  
9     height:40px;  
10    line-height:40px;  
11    display:inline-block;  
12    text-align:center;  
13    border-radius:50%;  
14    border:1px solid #ccc;  
15    margin:0 8px;  
16    color:inherit;  
17    opacity:0.75;  
18 }  
19  
20 .social > a:hover {  
21     opacity:0.6;  
22 }
```

Ici j'ai centré le texte et mis un padding, qui est une marge interieur, de 20px. Cela laissera un espace de 20 px entre mes icônes réseau sociaux et la petite « barre de navigation »(la liste « ul ») .

En plus du selecteur de classe je viens y ajouter un selecteur d'éléments enfants(l.6) afin de donner du style aux balises de liens hypertexte « a » qui sont directement dans la div « .social ». J'ai utilisé la librairie « FontAwsome » pour importer les icônes.

J'ai donc changé la taille de la police pour modifier la taille des icône. J'ai ensuite modifier la hauteur de la ligne ainsi que définir une longueur et largeur et ajouté un « border-radius » afin de faire un cercle qui entoure les icônes.

Ligne 20 j'ai utilisé une pseudo-classe « hover » dans le but de baisser l'opacité des icônes et donc d'ajouter un petit effet lorsqu'on les survols avec la souris.

Maintenant que j'ai réalisé la partie html et css du footer je m'occuper de la partie script. Elle complétera les 2 autres sections qui composera les fichiers composants ou les fichiers vues en vue js :

Template => Qui contiendra le code html

script => Qui contiendras le code JavaScript

Style => Qui contiendras le code CSS



```
1 <script>  
2     export default {  
3         name:'footerRs'  
4     }  
5 </script>
```

Je viens donner un nom(l.3) au composant. C'est avec ce nom que je pourrais rappeler le composant sur mes vues ou mes autres composants.

Nous allons ajouter le composant dans le fichier App.vue. Le fichier App.vue est le premier composant du site qui va contenir tous les autres composants. Il est en quelque sorte le composant racine d'une application de type Single-Page application. Tout est localisé à l'intérieur de ce composant. Si je met mon composant footer dans le app.vue, les composants

seront affiché sur les vues affichés par le routeur selon l'url. Je reviendrai sur le routeur juste-après.

```
1  <template>
2    <router-view/>
3    <navMenu/>
4    <footerRs/>
5  </template>
6  <script>
7    import footerRs from '@/components/Footer.vue'
8    import navMenu from '@/components/Nav.vue'
9    export default {
10      components: {
11        footerRs,
12        navMenu
13      }
14    }
15  </script>
```

Tout d'abord j'importe dans script les composants que je vais vouloir, à savoir le footer. J'indique donc le nom du composant et le chemin d'accès du fichier Footer.vue (l.7). Ensuite je vais indiquer qu'il s'agit d'un composant(l.10-11) afin que je puisse appeler le composant grâce à une balise au nom du composant. Une fois importer je peux donc appeler mon footer grâce à « footerRs », Vue viens importer le contenu du composant (« Footer.vue ») à l'endroit d'appel de la balise.

Pour faire le routeur j'ai utilisé vue router. C'est une librairie qui va me permettre de coder un routeur et donc associer à des URL à des noms de composants. Etant donné que mon projet est fait avec CLI je peux installer vue router avec le gestionnaire de paquets npm.

Il faut, après, importer la librairie dans le fichier main.js, qui est un fichier important dans un projet vuejs. Il est le fichier « racine » de l'application, il sert, en quelque sorte, à configurer le projet. Il précise les composants à utiliser (import App from './App.vue'), initialiser des variables globales et précise où le rendu doit être effectué (\$mount('#app'))).

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import router from './router'
4 import store from './store'
5 import axios from 'axios'
6 import VueAxios from 'vue-axios'
7 import 'bootstrap'
8 import bootstrap from '../node_modules/bootstrap/dist/css/bootstrap.css'
9 import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
10 import { library } from '@fortawesome/fontawesome-svg-core'
11 import { fas } from '@fortawesome/free-solid-svg-icons'
12 import { fab } from '@fortawesome/free-brands-svg-icons'
13
14 library.add(fas, fab);
15 createApp(App).use(store).use(router).use(bootstrap).use(VueAxios, axios).component('fa', FontAwesomeIcon).mount('#app')
```

J'importe l'App.vue, routeur, bootstrap, axios, store(vuex) qui sont des librairies que j'utilise dans mon projet. Un fois fait, je créer une nouvelle instance d'application avec la fonction createApp. L'objet que je passe lui passe en paramètre est le composants « App.vue ». Comme vu plus tôt il sera le « composant racine » qui pourra contenir d'autres composants en tant qu'enfants. J'appelle ensuite la méthode « .use() » pour activer l'utilisation de mes librairies importé plus tôt (« .use(router) » pour activer vue routeur). Enfin, pour que mon instance d'application affiche quelque chose, j'appelle la méthode « mount() » qui attend en argument un « container » (un élément du DOM).

```
● ● ●
1 <body>
2   <noscript>
3     <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled. Please enable it to continue.</strong>
4   </noscript>
5
6   <div id="app"></div>
7   <!-- built files will be auto injected -->
8 </body>
```

Le contenu du composant racine de l'application sera rendu à l'intérieur de l'élément conteneur. Ici le composant App.vue viendra remplacer la div avec l'identifiant « app ».

Maintenant que j'ai importé vue routeur, un dossier « router » s'est créé. Celui-ci contient le fichier « index.js » qui sera le fichier de configuration du routeur et où je vais pouvoir paramétriser mes différentes routes.

```
● ● ●
1 import { createRouter, createWebHistory } from 'vue-router'
2 import actus from '../views/Actualites.vue'
3
4 const routes = [
5   {
6     path: '/',
7     name: 'actusA',
8     component: actus
9   },
```

Pour chaque pages que je vais vouloir afficher, je vais devoir importer le composant correspondant. Comme à la ligne 2, j'importe mon composant Actualités.vue afin de pouvoir afficher la page actus et sera la page d'accueil. Ensuite chaque routes différentes va être un objet différent dans le tableau « route » (l.4). La déclaration « const » permet de créer un tableau routeur constant, accessible uniquement en lecture. Autrement dit la valeur d'une constante ne peut pas être modifiée.

Une route contiendra un path qui fait référence au chemin de l'URL qui appellera le composant et afficher la page, d'un name qui sera le nom du composant appelé, et component qui fera référence au fichier .vue du composant appelé, importé au préalable.

Je vais prendre pour exemple les routes qui vont me mener vers les pages « Afficher les membres »

De la ligne 1 à 5 je créer la route membres, accessible via « /membres » dans l'URL (<http://localhost/membres>). Elle appellera le composants « membresA » qui est dans le fichier « Membres.vue » et qui est importé grâce à une fonction fléché dans « component ».

```
● ○ ●
1  {
2    path: '/membres',
3    name: 'membresA',
4    component: () => import( '../views/Membres.vue')
5  },
6  {
7    path: '/ajouterMembre',
8    name: 'addMembres',
9    component: () => import( '../views/AddMembres.vue')
10 },
```

```
● ○ ●
1 <template>
2   <router-view/>
```

Dans App.vue j'avais fait appel au composant personnalisé « router-view ». Il affichera le composant qui correspond à l'url

```
● ○ ●
1 <li><a href="/membres"></a></li>
```

Dans ma nav bar j'ai mis des liens redirigeant vers mes différents « path » configuré dans mon routeur. Si je clique sur l'ancrage juste au dessus, cela remplacera « router-view » dans App.vue par mon composant Membres.vue et affichera donc la page des membres.

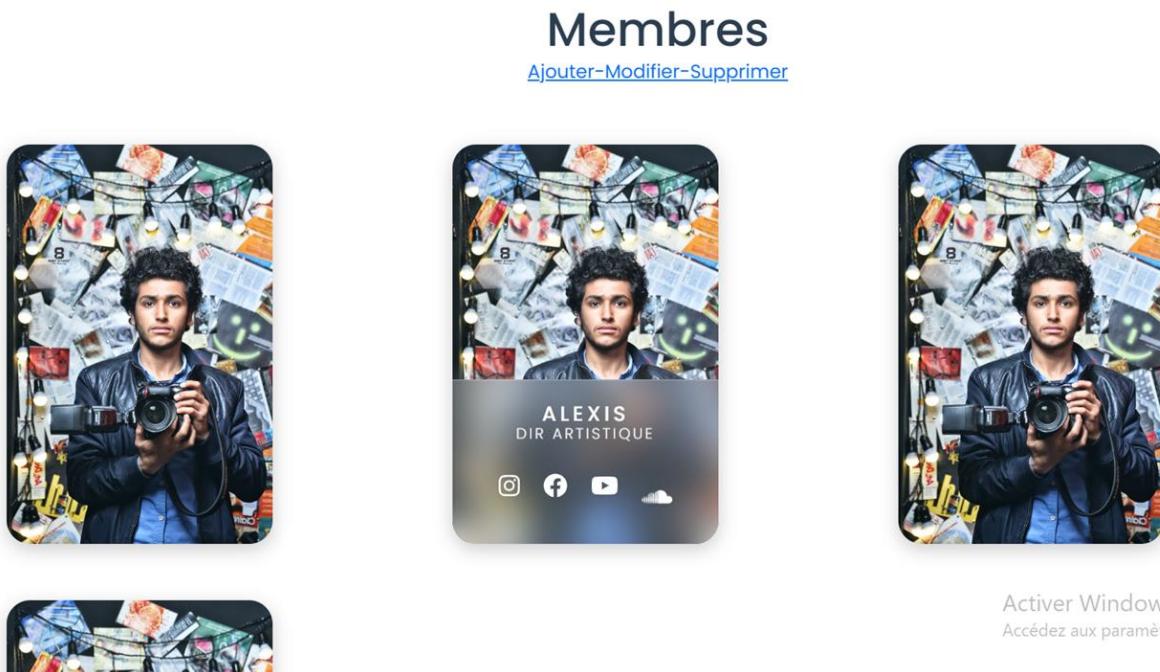
Dans mon composants Membres.vue J'ai ajouté un bouton me dirigeant vers la page « Ajoute membre ». Mais au lieu d'utiliser la balise d'ancrage, j'ai utiliser « router-link ».

```
● ○ ●
1 <router-link to="/ajouterMembre">Ajouter-Modifier-Supprimer</router-link>
```

Au lieu d'utiliser des balises « a », il est mieux d'utiliser le composant personnalisé router-link pour créer des liens. Cela permet à Vue Router de changer d'URL sans recharger la page et de gérer la génération d'URL .

Lorsque je clique sur cet élément, d'après mon routeur, j'appellerais le composant AddMembres.vue qui viendra remplacer « routeur-view » dans le composant App.vue et affichera ma page d'ajout et de modification des membres.

Maintenant que je peux accéder, via ma barre de navigation, à la page d'affichage des membres, j'affiche le composant « membres ».



Nous pouvons voir qu'en bas à gauche et en bas de la page les composants footer et nav sont bien affichés. Ceci grâce aux balises personnalisées appelé dans le App.vue. Il seront donc affiché peut importe l'url.



Dimensions: Samsung Galaxy A51/71 ▾ 412 x 914 50% ▾ Aucune limitation ▾



Pour faire cette affichage j'ai utilisé bootstrap afin de ne pas avoir plus de 3 cartes de membre par ligne et afin de gérer facilement la responsivité. A partir d'un certaine taille d'écran, le site n'affiche qu'une carte par ligne.

```
1 <div class="row">
2   <div class="col-lg-4 mb-5" v-for="(team, index) in this.loadTeam" :key="index">
3     <div class="card p-0">
```

```
1 <div class="card-content d-flex flex-column align-items-center">
2   <h4 class="pt-2">{{team.name_team}}</h4>
3   <h5>{{team.mission_team}}</h5>
4
5   <ul class="social-icons d-flex justify-content-center">
6     <li style="--i:1">
7       <a :href="team.insta_team">
8         <span><fa :icon="['fab', 'instagram']"/></span>
9       </a>
10      </li>
```

Maintenant si je clique sur le lien « Ajouter-Modifier-Supprimer » qui est un routeur-link affichant le composant « addMembre », cela affichera ma page de gestion des membres.

Gestion des membres

--Choisir un membre--

Prénom et Nom

Facebook

Soundcloud

Role

Instagram

Youtube

Photo :

Choisir un fichier Aucun fichier choisi

Ajouter membre

```

● ● ●
1 <div class="row mb-3">
2   <div class="col">
3     <select v-model="id" class="form-select">
4       <option value="">--Choisir un membre--</option>
5       <option
6         v-for="(team, index) in this.loadTeam" :key="index"
7         :value="team.id_team"
8       >
9         {{team.name_team}}
10      </option>
11    </select>
12  </div>
13  <div class="col">
14    <button v-show="id != ''" @click="delMembre()" type="button" class="btn btn-outline-danger">Supprimer le membre sélectionné</button>
15  </div>
16 </div>

```

J'ai créé une div(l.1) avec l'attribut « class » et je lui ai attribué une classe bootstrap pour que le contenu de la div soit en ligne. Le « mb-3 » signifie mettre une marge au bottom de la div, pour avoir l'espace qui sépare les lignes entre elles. J'y ai imbriqué 2 div (l.2 et l.13) avec l'attribut « class=col » qui indique que les éléments de ces div sont les colonnes de la ligne. J'aurais donc ainsi, sur la même ligne, à gauche ma SelectBox et à droite mon bouton « supprimer » qui n'est pour le moment pas visible.

J'ai fait pareil pour les autres ligne du formulaire : La « div row »(l.1) qui va mettre en ligne les colonnes qui contiendront l'« input prénom et nom »(l.3) et l'« input role »(l.6) :

```

● ● ●
1 <div class="row mb-3">
2   <div class="col">
3     <input type="text" v-model="name" class="form-control" placeholder="Prénom et Nom">
4   </div>
5   <div class="col">
6     <input type="text" v-model="role" class="form-control" placeholder="Role">
7   </div>
8 </div>

```

Si on regarde le script du composant, je fais appel à un hooks de cycle de vie « created ». Celui-ci est appellé chaque fois que l'instance a été créée, c'est à dire que le composant a été chargé. La fonction est donc exécuté et le « this.\$store.dispatch » va exécuter une action avec le nom « loadTeam » dans le fichier « /store/index.js » (vueX) et qui aura pour but de récupérer tous les membres en base de données.

```

● ● ●
1 created: function () {
2   this.$store.dispatch('loadTeam');
3 },

```

Maintenant que les membres sont récupérés, le SelectBox va afficher tous les noms des membres.

```

1 <select v-model="id" class="form-select">
2   <option value="">--Choisir un membre--</option>
3   <option
4     v-for="(team, index) in this.loadTeam" :key="index"
5     :value="team.id_team"
6   >
7     {{team.name_team}}
8   </option>
9 </select>

```

Le sélecteur aura autant d'options (qui seront les choix possibles de la sélection) qu'il y a de membres enregistré en BDD. Cela est possible grâce à la directive v-for (I.4) qui permet de faire le rendu de la liste en se basant sur le tableau retourné par «this.loadTeam».

```

1 computed: {
2   loadTeam(){
3     return this.$store.state.membres;
4   }
}

```

La propriété calculée `loadTeam()` va récupérer la valeur retournée par l'action « `dispatch('loadTeam')` » qui aura été enregistré dans le tableau « `membres` » disponible sans le store vueX.

Je vais donc bind l'id du membre à la valeur de l'option afin de pouvoir facilement récupérer l'id du membre sélectionné via le select. Je met comme texte d'option le nom du membre grâce à l'interpolation '{{}}'. Des que la page gestion des membres sera chargé le SelectBox pourra afficher le nom de tous le membres.

Gestion des membres

--Choisir un membre--

Prénom et Nom	Role
Facebook	Instagram
Soundcloud	Youtube
Photo :	
<input type="file"/> Choisir un fichier Aucun fichier choisi	

```

VM1108:1 Another version of Vue Devtools seems to be installed. Please enable only one version at a time.
VM1108:1 [Vue warn]: A plugin runtime-core.esm-bundler.js?d2dd:38 must either be a function or an object with an "install" function.
index.js?68eb:24
▶ (4) [{}]
▶ 0: {0: '41', 1: 'Ot', 2: '', 3: '', 4: '', 5: 'Web desi
▶ 1: {0: '42', 1: 'Alexis', 2: '', 3: '', 4: '', 5: 'Dir
▶ 2: {0: '43', 1: 'Bohl', 2: '', 3: '', 4: '', 5: 'DJ tec
▶ 3: {0: '44', 1: 'Sks', 2: '', 3: '', 4: '', 5: 'comptab
length: 4
[[Prototype]]: Array(0)

```

On voit dans la console que je récupère bien tous mes membres.

On peut également voir dans la console que la data « id » a bien récupéré la valeur de l'id du membre « Alexis ».

Dans la partie script du composant, je créer une fonction data. La propriété du composant data doit être une fonction, afin que chaque instance puisse conserver une copie indépendante de l'objet retourné. J'écris donc toute les variables qu'il me semble nécessaire d'avoir, à savoir le nom, le rôle, les réseaux sociaux et l'id.

```

1  data() {
2      return {
3          id: '',
4          name: '',
5          role: '',
6          photo: '',
7          fb: '',
8          ig: '',
9          sc: '',
10         yt: '',
11     }
12   },

```

Ici, par exemple je vais bind la valeur du select dans la variable « id » et donc récupérer dans cette variable l'id de la personne sélectionné en fonction de sa valeur d'option dans la SelectBox.

```

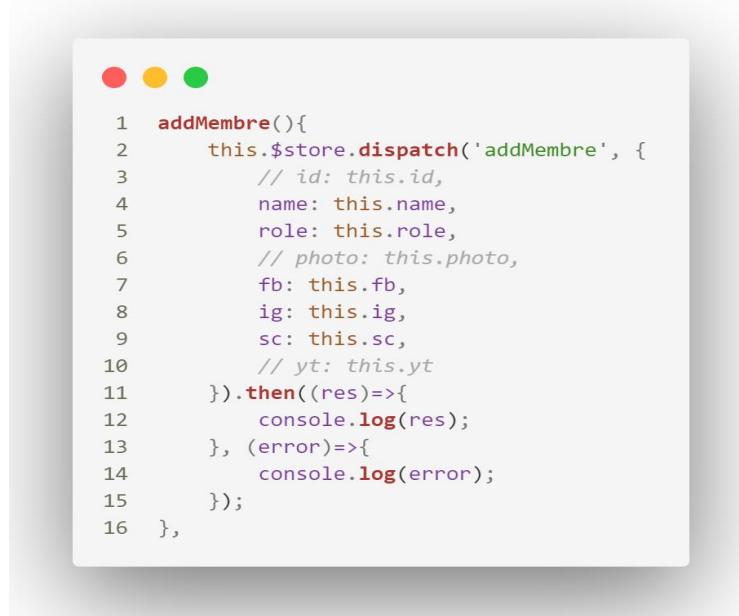
1  <select v-model="id" class="form-select">
2
3      <button type="button" v-show="id == ''" @click="addMembre()" class="btn btn-outline-success mb-3">Ajouter membre</button>
4      <button type="button" v-show="id != ''" @click="modifMembre()" class="btn btn-outline-warning mb-3">Modifier membre</button>

```

Avec la variable « id » je vais afficher ou non les boutons ‘Ajouter Membre’ et ‘Modifier membre’. Cela est possible avec la directive de rendu conditionnel « v-show ». Le buton ‘Ajouter membre’ ne s'affichera que si la variable ‘id’ est vide. Si la variable id a une valeur associé alors ce sera le bouton ‘Modifier membre’ qui s'affichera. De ce fait, quand aucun membre ne sera sélectionné, j'aurais la possibilité d'en ajouter un, mais si, au contraire, un membre est sélectionné, je n'aurais que la possibilité de le modifier.

Il y aussi un écouteur d'évènement qui attend l'évènement du clique (@click="") sur les deux boutons mais qui appelle chacun une méthode différente disponible dans le script.

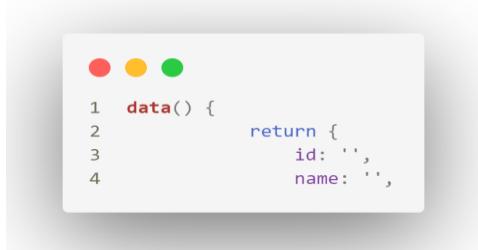
Dès que je clique sur le boutons ‘Ajouter membre’, la méthode « addMembre() » est exécutée. Elle n'a pas besoin de paramètres vu que les données qui nous intéressent sont déjà dans les variables de la fonction « data » et récupérée grâce au binding des « input ».



```
1  addMembre(){
2      this.$store.dispatch('addMembre', {
3          // id: this.id,
4          name: this.name,
5          role: this.role,
6          // photo: this.photo,
7          fb: this.fb,
8          ig: this.ig,
9          sc: this.sc,
10         // yt: this.yt
11     }).then((res)=>{
12         console.log(res);
13     }, (error)=>{
14         console.log(error);
15     });
16 },
```

Pour la méthode addMembre, j'utilise dispatch (l.2) qui va prendre en paramètres l'action vueX que je souhaite appeler avec un nom identique (ici ‘addMembre’). Je lui envoie en paramètre un objet contenant les données qui sont dans data un peu plus haut dans le script. Je leurs donne un nom (qui servira pour que la variable puisse être appelé grâce à \$_POST[‘’] plus tard). Je lui assigne également une valeur. Ici je viens récupérer la valeur qui correspond à l'entrée des input.

« this » est un objet qui fait référence à l' élément qui a reçu l'événement. Par exemple ligne 4, « this.name » fait référence à la data name qu'il y a dans la fonction data.



```
1  data() {
2      return {
3          id: '',
4          name: ''
```

« this.name » retourne la valeur qui est assigné à name, à savoir la valeur dans l'input qui lui est binder.

Le « .then » va permettre de récupérer et afficher dans le console.log la réponse renvoyé par l'api. En cas d'erreur, on l'envoie dans une fonction fléché qui l'affichera dans le console.log.

Je vais parler du gestionnaire d'état vueX. Il permet de centraliser les données entre les différents composants de l'application.



```
store
  index.js
```

J'ouvre le fichier index.js du dossier store qui sera le fichier de configuration de vueX.

```
● ● ●  
1 const axios = require('axios');  
2 const instance = axios.create({  
3   baseURL: "http://localhost/nocturn/src/php/index.php"  
4 });
```

J'importe dans un premier temps axios qui me permettra de faire des requêtes HTTP avec mon back-end. Je créer une instance qui aura une URL de base et qui pourra facilement être appelé plus tard.

```
● ● ●  
1 instance.post("?url=loadTeam")
```

On charge donc l'URL grâce à « instance », j'indique la méthode HTTP utilisé, « get » ou « post ». « post » transmet les informations du formulaire de manière mas

quée mais non cryptée.

«?url=loadTeam » envoie des informations en claire dans l'url et récupérera la valeur d'« url » dans \$_GET['url']. Celle-ci contiendra « loadTeam ». Cela permettra à mon routeur « index.php » de rediriger vers la bonne fonction du controller en back-end.

Une fois l'import et l'instance d'Axios fait, je créais un nouveau store. Je ne peux avoir qu'un seul magasin pour chaque application. L'arbre d'état « State » est un objet qui va stocker les données. Je pourrais récupérer les données à l'intérieur du magasin sur mes composants en renvoyant un état de magasin à partir d'une propriété calculée comme ci-dessous:

```
● ● ●  
1 computed: {  
2   loadTeam(){  
3     return this.$store.state.membres;  
4   }  
}
```

```
● ● ●  
1 export default createStore({  
2   state: {  
3     //gestion des membres  
4     membres: [],
```

Ensuite je vais appeler des actions depuis mes composants vuejs avec dispatch :

Ici j'appelle l'action 'loadTeam' de mon store.

```
● ● ●  
1 this.$store.dispatch('loadTeam');
```

L'action va réaliser une requête HTTP à mon api grâce à axios. Axios est un client HTTP qui permet de communiquer avec des api en utilisant des requêtes. Le chemin d'accès de l'api est défini par l'instance créée plus tôt en plus du paramètre « url » qui sera traité par mon index.php à l'aide de la super global GET. Je vais réalisé la requête avec la méthode HTTP « post ».

Ensuite j'envoie la réponse, en utilisant la destructuration d'argument avec « commit », vers la mutation « loadTeam » du store. En cas d'erreur je l'affiche dans le console.log

Le but de la requête est donc de récupérer la liste de tous le membres enregistré dans la BDD.

```
● ● ●
1 actions: {
2   //gestion des membres
3   loadTeam({commit}){
4     instance.post("?url=loadTeam")
5     .then((res) => {
6       commit('loadTeam', res.data);
7     })
8     .catch((error) => {
9       console.log(error);
10    });
11  },

```

```
● ● ●
1 mutations: {
2   //gestion membres
3   loadTeam(state, membres){
4     console.log(membres);
5     state.membres = membres;
6   },

```

La mutation est La seule façon de réellement changer d'état dans un magasin. Je met l'état comme premier argument (« state »). En second argument je met un 'payload' (objet contenant toutes les données de l'opération à effectuer). Il s'agit de la réponse envoyé par l'action 'loadTeam' à savoir les membres qui sont en BDD. J'ajoute donc dans mon 'state' « membre », qui est un tableau, la liste des membres.

```
● ● ●
1 membres: [],
```

L'état 'membres', qui contient maintenant les membres enregistrés, pourra être appelé sur tous mes composants.

Si je prend l'action 'addMembre', j'ai en plus un argument « membreInfos » qui est un objet qui contient les données envoyé par la méthode du composant.

```
● ● ●
1 addMembre(){
2   this.$store.dispatch('addMembre', {
3     // id: this.id,
4     name: this.name,
5     role: this.role,
```

```
● ● ●
1 addMembre:({commit}, membreInfos) =>{
2   instance.post("?url=addMembre", membreInfos)
```

Cela me permet d'envoyer à mon api des données qui pourront être traité par celle-ci et donc, dans ce cas là, ajouter des membres avec leurs infos en BDD.

Back-end

Plus haut j'ai appelé à l'aide d'axios des api. Dans cette section je vais expliquer comment j'ai organisé mon back-end et donc comment j'ai géré mes api.

` php	●
` controllers	●
` teamController.php	M
` userController.php	M
` models	●
` TeamClass.php	M
` UserClass.php	M
` config.php	
` index.php	

Dans mon dossier php j'ai :

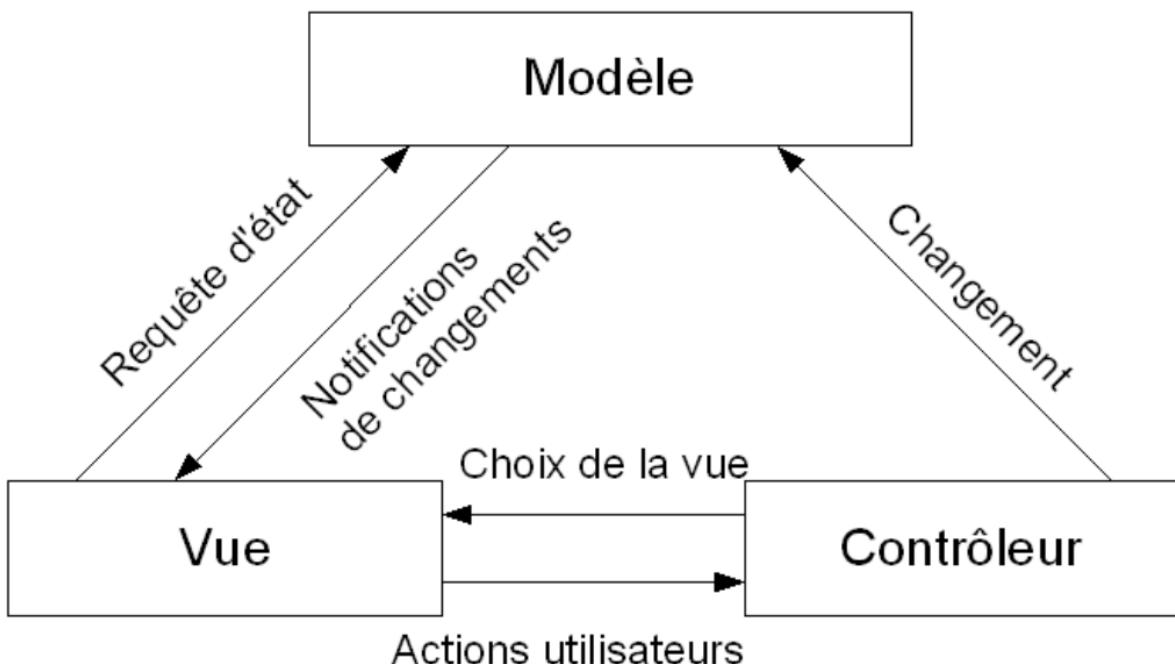
- **index.php** qui va me servir de routeur et me rediriger vers les bonnes fonctions des controller.
- **config.php** qui gère la configuration et la connection à la BDD.
- Le dossier **controllers** qui contiendra mes fichiers controller.
- Le dossier **models** qui contiendra mes fichiers model.

J'ai donc organisé ça avec l'architecture MVC. Elle me permet de mieux gérer la structuration de mon projet. Elle se compose en 3 parties :

Modèle : un noyau de l'application qui gère les données, permet de récupérer les informations dans la base de données, de les organiser pour qu'elles puissent ensuite être traitées par le contrôleur.

Vue : composant graphique de l'interface qui permet de présenter les données du modèle à l'utilisateur. Vue sera gérée avec vueJS.

Contrôleur : composant responsable des prises de décision, gère la logique du code qui prend des décisions, il est l'intermédiaire entre le modèle et la vue.



Je vais utiliser l'index comme routeur.

```
● ● ●  
1 include("controllers/userController.php");  
2 include("controllers/teamController.php");  
3  
4 $userManager = new UserController;  
5 $teamManager = new TeamController;  
6 $url = "";
```

Je vais inclure mes différents contrôleurs. Ici c'est surtout teamController (l.2) qui va nous intéresser. Après, j'instancie un nouvelle objet de la classe « TeamController » (l.5) et je déclare une nouvelle variable « url » vide.

Lorsque j'appelle l'api, je dois indiquer le chemin d'accès de l'api. Si je prend l'exemple avec 'loadTeam', il va aller chercher l'api avec le lien suivant :
«<http://localhost/nocturn/src/php/index.php?url=loadTeam> »

```
● ● ●  
1 if (isset($_GET['url'])) {  
2     $url = $_GET['url'];  
3 }
```

Si \$_GET['url'] n'est pas vide, je vais assigné à la variable \$url, la valeur du paramètre « url » fourni dans l'url de la requete axios. Elle sera récupéré par la super global GET.

Ici, \$_GET['url'] aura comme valeur ce qu'il y a apres le « url= ». La variable « \$url » aura donc comme valeur 'loadTeam'.

```
● ● ●  
1 instance.post("?url=loadTeam")
```

```
● ● ●  
1 switch ($url) {  
2     case 'inscription':  
3         $userManager->register();  
4         break;  
5     case 'connexion':  
6         $userManager->login();  
7         break;  
8     case 'loadTeam':  
9         $teamManager->loadTeam();  
10        break;
```

J'utilise la structure conditionnelle « switch.. case » qui équivaut à une série d'instruction if. En paramètre du switch je vais mettre la variable à comparer et dans le 'case' la valeur qui sera comparé avec le switch.

Le code exécuté sera le code contenu dans le case correspondant au switch.

Etant donnée que \$url vaut 'loadTeam', ca sera la ligne 9 qui sera executé.

Je vais donc appeler la fonction 'loadTeam' de la classe TeamController.

```
1 include('models/TeamClass.php');
2 class TeamController{
3     private $teamManager;
4
5     public function __construct(){
6         $this->teamManager = new Team();
7     }
```

J'inclue, en premier, mon fichier model 'TeamClass.php'. En second je créais la classe 'TeamController' dans laquelle je déclare une nouvelle variable qui a une portée de variable 'private'.

Cela signifie que l'accès à l'élément est uniquement réservé à la classe qui les a définis.

Le constructeur(l.5) va me permettre de créer un objet teamManager de la classe Team à chaque création de nouvelle instance d'objet TeamController.

Plus haut j'ai appelé la méthode 'loadTeam', qui est une fonction public :

```
1 public function loadTeam(){
2     try{
3         $team = $this->teamManager->getFullTeam();
4         echo json_encode($team);
5     }catch (Exception $e) {
6         die('Erreur : ' . $e->getMessage());
7     }
8 }
```

L'instruction try...catch regroupe des instructions à exécuter et définit une réponse si l'une de ces instructions provoque une exception. Il permet de gérer les erreurs. J'entoure le code qui peut potentiellement provoquer une erreur par un bloc try catch, et si une erreur se produit lors de l'exécution du code présent dans le bloc try, PHP va exécuter le code qui se trouve dans le catch.

J'appelle donc la méthode getFullTeam() de la classe Team() grâce à l'objet teamManager. La fonction va me retourner la liste des membres.

La ligne 4 me permet d'encoder la valeur au format JSON et de retourner la réponse grâce à « echo ». La conversion en JSON est importante car axios renvoie des objets JSON.

Avant de parler du model TeamClass je vais parler du fichier de configuration de la base de données.



```

1 abstract class Database {
2     private $host = "localhost";
3     private $database_name = "nocturn";
4     private $username = "root";
5     private $password = "";
6
7     private $conn;

```

Dans ce fichier j'ai un classe « database » avec une portée « abstract », ce qui signifie que c'est une classe qui ne va pas pouvoir être instanciée directement, qu'on ne va pas pouvoir manipuler directement et doit être étendue pour utiliser ses fonctionnalités.

Elle a 5 propriétés qui contiennent toutes des paramètre de la base de données :

- Le chemin d'accès de la base de données, ici « ocalhost » car je travaille en local à l'aide de xamp.

- Le nom de la base de données utilisé
- Le pseudo d'un utilsateur
- Le mot de passe de l'utilisateur
- La propriétés avec lequel je renvoie la connexion à la BDD.

J'ai choisis de faire comme ça pour des raisons de simplicité. En effet si je viens à changer l'adresse de la BDD ou que je souhaite utiliser un autre compte, les donnée à changer seront facilement modifiable.



```

1 protected function getDB(){
2     $this->conn = null;
3     try{
4         $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->database_name, $this->username, $this->password);
5         $this->conn->exec("set names utf8");
6     }catch(PDOException $exception){
7         echo "Database could not be connected: " . $exception->getMessage();
8     }
9     return $this->conn;
10 }

```

La méthode getDB est en protected. Ca signifie qu'elle est limité à la classe elle-même, ainsi qu'aux classes qui en héritent.

Dans le try je créé une instance de la classe PDO qui va activer ma connexion, avec comme paramètres les propriétés qui contiennent les paramètres de connexion à la BDD.

Je précise le charset en utf8. S'il n'y a pas d'erreur une instance de la classe PDO est retourné sinon l'objet PDOException et lancé et attrapé grâce au catch.



```
1 require_once('config.php');
2 class Team extends Database{
3     private $id_team;
4     private $name_team;
5     private $fb_team;
6     private $insta_team;
7     private $sc_team;
8     private $photo_team;
9     private $mission_team;
```

J'importe donc le fichier de config de ma BDD et je vais pouvoir étendre la classe Team avec la classe Database avec le extends.

Il s'agit d'un héritage . La classe enfant (Team) hérite de toutes les méthodes publiques et protégées, propriétés et constantes de la classe parente (Database).

Je créé autant de propriétés qu'il y a de colonne dans ma table Team en BDD, faisant chacune toutes référence à une colonne de la table.

Je viens créer un getter et un setter pour chaque propriété.

Le getter me permet de récupérer la valeur actuelle d'une propriété alors que le setter permet d'attribuer une valeur à une propriété.

Le getter va retourner une valeur alors que le setter aura besoin d'un paramètre lors de l'appel de la méthode.



```
1 public function getName_team(){
2     return $this->name_team;
3 }
4 public function setName_team($name_team){
5     $this->name_team = $name_team;
6 }
```



```
1 public function getFullTeam(){
2     $query = 'SELECT
3             *
4             FROM
5                 team';
6     $req = $this->getDB()->prepare($query);
7     $req-> execute();
8     $data = $req->fetchAll();
9     if(!empty($data)){
10         return $data;
11     }else{
12         return null;
13     }
14 }
```

Dans ma méthode getFullTeam() je prépare une requête sql qui va me sélectionner tous les enregistrements de la table 'team'. Dans la variable \$data (l.8), je récupère un tableau toutes les lignes du jeu d'enregistrements retourné par la requête et je retourne \$data s'il elle n'est pas vide.

```

1 $team = $this->teamManager->getFullTeam();
2 echo json_encode($team);

```

Donc « \$team » récupérera la valeur retourné par la méthode `getFullTeam` appelé par l'objet `teamManager`.

		id_team	name_team	fb_team	insta_team	sc_team	mission_team	
	Éditer		Copier		Supprimer	41	Qt	Web design
	Éditer		Copier		Supprimer	42	Alexis	Dir Artistique
	Éditer		Copier		Supprimer	43	Bohl	DJ techno
	Éditer		Copier		Supprimer	44	Sks	Comptable
	Éditer		Copier		Supprimer	45	Quentin Terzi	aucun
								aucun
								aucun
								Développeur web

▼ Array(5)

```

▶ 0: {0: '41', 1: 'Qt', 2: '', 3: '', 4: '', 5: 'Web design', id_team: '41', name_team: 'Qt', fb_t
▶ 1: {0: '42', 1: 'Alexis', 2: '', 3: '', 4: '', 5: 'Dir Artistique', id_team: '42', name_team: 'A
▶ 2: {0: '43', 1: 'Bohl', 2: '', 3: '', 4: '', 5: 'DJ techno', id_team: '43', name_team: 'Bohl', t
▶ 3: {0: '44', 1: 'sks', 2: '', 3: '', 4: '', 5: 'Comptable', id_team: '44', name_team: 'sks', fb_
▶ 4: {0: '45', 1: 'Quentin Terzi', 2: 'aucun', 3: 'aucun', 4: 'aucun', 5: 'Développeur web ', id_t
length: 5

```

Si je regarde avec l'ajout de membre :

--Choisir un membre--

Adrar Formateur

facebook.com instagram.com

soundcloud Youtube

Photo :

Choisir un fichier Aucun fichier choisi

Ajouter membre

fb: "facebook.com" id: "" ig: "instagram.com" name: "Adrar" photo: "" role: "Formateur" sc: "soundcloud" yt: ""

J'entre les données dans mes inputs et on voit que data les réupère bien.

```

1 <button type="button" v-show="id == ''" @click="addMembre()" class="btn btn-outline-success mb-3">Ajouter membre</button>

```

Quand je clique sur le bouton 'Ajouter membre' cela va m'exécuté la méthode « `addMembre()` ».

```
1 addMembre(){
2     this.$store.dispatch('addMembre', {
3         // id: this.id,
4         name: this.name,
5         role: this.role,
6         // photo: this.photo,
7         fb: this.fb,
8         ig: this.ig,
9         sc: this.sc,
10        // yt: this.yt
11    }).then((res)=>{
12        console.log(res);
13    }, (error)=>{
14        console.log(error);
15    });
}
```

Avec la méthode j'envoie dans l'action du store 'addMembre' les données contenu dans data, celles entrées dans les inputs.

Avec axios, j'envoie l'objet membreInfos à mon index.php avec un paramètre url égale à 'addMembre'.

```
1 addMembre:({commit}, membreInfos) =>{
2     return new Promise((resolve, reject) => {
3         commit;
4         instance.post("?url=addMembre", membreInfos)
5             .then((res) => {
6                 resolve(res);
7             })
8             .catch((error) => {
9                 reject(error);
10            });
11    });
}
```

```
1 case 'addMembre' :
2     $teamController->addMembre();
3     break;
```

Le switch case de mon index.php va me rediriger vers la méthode addMembre() de la classe TeamController dans mon fichier controller.

```

1  public function addMembre(){
2      try{
3          $_POST = json_decode(file_get_contents("php://input"),true);
4          $this->teamManager->setName_team(htmlspecialchars(strip_tags($_POST["name"])));
5          $this->teamManager->setMission_team(htmlspecialchars(strip_tags($_POST["role"])));
6          // $this->teamManager->setPhoto_team(htmlspecialchars(strip_tags($_POST["photo"])));
7          $this->teamManager->setFb_team(htmlspecialchars(strip_tags($_POST["fb"])));
8          $this->teamManager->setInsta_team(htmlspecialchars(strip_tags($_POST["ig"])));
9          $this->teamManager->setSc_team(htmlspecialchars(strip_tags($_POST["sc"])));
10
11         if(!empty($_POST)){
12             $this->teamManager->addTeam();
13         }
14     }catch (Exception $e) {
15         die('Erreur : ' . $e->getMessage());
16     }
17 }

```

Ici je vais décoder le JSON envoyé par axios afin que je puisse assigner les valeurs récupéré plus tot dans mon \$_POST et pouvoir utiliser \$_POST['name'], par exemple, et qui aura la valeur 'Adrar'.

En suite je vais utiliser les setter pour injecter une valeur dans chaque attributs de l'objet teamManager, qui est un objet de la classe TeamModel, les différentes valeurs de la super global \$_POST.

Quand je 'set' les valeur j'utilise htmlspecialchars et strip_tags afin d'améliorer la sécurité et limité les failles XSS et injections sql.

htmlspecialchars va remplacer certains caractères spéciaux et éviter que des données fournies par les utilisateurs contiennent des balises HTML

Strip_tags va supprimer les balises HTML et PHP d'une chaîne de caractère.

Si \$_POST n'est pas vide j'execute la méthode addTeam().

```

1  public function addTeam(){
2      $req = $this->getDB()->prepare('INSERT INTO team (name_team, fb_team, insta_team, sc_team, mission_team) VALUES (:name_team, :fb_team, :insta_team, :sc_team, :mission_team)');
3      $req->execute(array(':name_team'=>$this->name_team,
4                          ':fb_team'=>$this->fb_team,
5                          ':insta_team'=>$this->insta_team,
6                          ':sc_team'=>$this->sc_team,
7                          ':mission_team'=>$this->mission_team));
8  }

```

Je prépare une requete qui va insérer dans la table team et dans certaines colonnes les valeurs entrée par l'utilisateur. Les valeurs vont être attribuées lors de l'execution de la requete dans un tableau array et récupérer les valeurs des propriétés 'set' un peu plus tot.



```
1 Issue: 1
✖ Another version of Vue Devtools seems to be installed. Please enable only one version at a time. VM1920:1
⚠ [Vue warn]: A plugin must either be a runtime-core.esm-bundler.js?d2dd:38 function or an object with an "install" function. index.js?68eb:24

▼ (6) [{...}, {...}, {...}, {...}, {...}, {...}]
  ► 0: {0: '41', 1: 'Qt', 2: '', 3: '', 4: '', 5: 'Web design', id_team: '41', length: 6}
  ► 1: {0: '42', 1: 'Alexis', 2: '', 3: '', 4: '', 5: 'Dir Artistique', id_team: '42', length: 6}
  ► 2: {0: '43', 1: 'Bohl', 2: '', 3: '', 4: '', 5: 'DJ techno', id_team: '43', length: 6}
  ► 3: {0: '44', 1: 'sks', 2: '', 3: '', 4: '', 5: 'Comptable', id_team: '44', length: 6}
  ► 4: {0: '45', 1: 'Quentin Terzi', 2: 'aucun', 3: 'aucun', 4: 'aucun', 5: 'Dé', length: 6}
  ► 5: {0: '46', 1: 'Adrar', 2: 'facebook.com', 3: 'instagram.com', 4: 'soundcl', length: 6}
  ► [[Prototype]]: Array(0)
```

On remarque que la requête à bien ajouté le membre avec les bonnes infos.

J'ai ajouté en annexes les requêtes pour modifier et supprimer les membres afin de compléter le CRUD (Create, Read, Update, Delete) pour la page gestion des membres.

7. ANNEXES

Requêtes modifier et supprimer gestion membres

Supprimer :

```
● ● ●  
1 public function delTeam(){  
2     $req = $this->getDB()->prepare('DELETE FROM team WHERE id_team = '. $this->id_team.'');  
3     $req->execute();  
4 }
```

Modifier :

```
● ● ●  
1 public function modifTeam(){  
2     $req = $this->getDB()->prepare("UPDATE team SET name_team='".$this->name_team . "', fb_team='".$  
3                                         . $this->fb_team . "', insta_team='".$this->insta_team . "' ,sc_team='".$  
4                                         . $this->sc_team . "' , mission_team='".$this->mission_team . "' WHERE id_team='".$  
5                                         . $this->id_team . "'");  
6     $req->execute();  
7 }
```

Media queries page affichage membre

Voici un bout de code pour gérer la responsivité de l'affichage des membres :

```
● ● ●  
1 @media(max-width: 991.5px) {  
2     .container {  
3         margin-top: 20px;  
4     }  
5  
6     .container .row .col-lg-4 {  
7         margin: 20px 0px;  
8     }  
9 }
```

