



WEB – LIGUE 1

[Sous-titre du document]



[DATE]

[NOM DE LA SOCIETE]

[Adresse de la société]

WEB – Ligue 1

0. Sommaire

0.	Sommaire	1
1.	Introduction	2
1.1.	Contexte du projet.....	2
1.2.	Objectifs du développement du site web de la Ligue 1	2
1.3.	Connexion au site pour test.....	3
2.	Conception du site web.....	3
2.1.	Architecture MVC	3
2.1.1.	Définition	3
2.1.2.	Application dans le site web.....	3
2.2.	Routeur	4
2.2.1.	Classe Router	4
2.2.2.	Intégration du Router dans un code HTML.....	5
3.	Présentation des pages du site	6
3.1.	Page Clubs	6
3.2.	Page Articles	7
3.3.	Page Inscription.....	7
3.4.	Page Connexion.....	8
4.	Conception du formulaire d'inscription	8
4.1.	Choix des champs du formulaire	8
4.2.	Ergonomie et design du formulaire	8
4.2.1.	Principes de l'expérience utilisateur	8
4.2.2.	Adaptabilité aux divers dispositifs (responsive design).....	9
5.	Validation du Formulaire d'inscription.....	9
5.1.	Contrôles côté client	9
5.1.1.	Vérifications des champs obligatoires	9
5.1.2.	Formatage des données (adresse email)	9
5.2.	Contrôles côté serveur	10
5.2.1.	Politiques de mot de passe	10
5.2.2.	Stockage sécurisé des mots de passe.....	10
6.	Sécurité contre les Attaques par Injection	11
6.1.	Compréhension des attaques par injection.....	12
6.1.1.	Injections SQL	12
6.1.2.	Cross-Site Scripting (XSS)	12
6.2.	Utilisation de requêtes préparées	12
7.	Intégration d'une API	13

WEB – Ligue 1

7.1.	Test de l'API, liste des communes	13
7.2.	Adaptation au champ Code Postal	15
8.	Connexion de l'utilisateur	15
8.1.	Vérification du formulaire	15
8.2.	Dynamisme des pages	16
8.2.1.	Header du site	16
8.2.2.	Page des articles	17
9.	Conclusion	18
10.	Bibliographie	19

1. Introduction

1.1. Contexte du projet

La réalisation du projet de conception du site web de la Ligue 1 revêt une importance significative au sein du cursus du BTS Services Informatiques aux Organisations (SIO). Au cours de notre deuxième année de formation, notre mission consistait à élaborer un site web dédié à la Ligue 1, en tenant compte des diverses contraintes ajoutées progressivement tout au long de l'année.

Ce projet de développement de site web avait pour objectif d'approfondir nos connaissances dans le domaine du développement web, englobant aussi bien la conception que la cybersécurité.

1.2. Objectifs du développement du site web de la Ligue 1

Le développement du site web de la Ligue 1 s'articule autour de plusieurs objectifs stratégiques. Notre mission principale était de créer une plateforme en ligne complète et fonctionnelle dédiée à la Ligue 1, en intégrant les éléments spécifiques imposés par le contexte du BTS Services Informatiques aux Organisations (SIO). Les objectifs spécifiques de ce projet incluent :

- **Conception Intégrale** : Élaborer une structure de site web complète et intuitive pour offrir une expérience utilisateur optimale tout en respectant les normes de conception modernes.
- **Contraintes Progressives** : S'adapter et intégrer les contraintes additionnelles qui ont été introduites tout au long de l'année, démontrant ainsi notre capacité à réagir et à ajuster le développement en fonction des besoins changeants du projet.
- **Expertise en Développement Web** : Acquérir des compétences approfondies dans le développement web, en se concentrant sur des aspects tels que la programmation, la gestion de bases de données et l'optimisation des performances.
- **Cybersécurité** : Intégrer des mesures de cybersécurité pour assurer la protection des données et la fiabilité du site, mettant ainsi en pratique les connaissances acquises dans le domaine de la sécurité informatique.
- **Apprentissage Continu** : Favoriser un environnement d'apprentissage continu en relevant les défis techniques liés au développement web et en résolvant les problèmes rencontrés, contribuant ainsi à notre formation globale en tant qu'étudiants en BTS SIO.

WEB – Ligue 1

1.3. Connexion au site pour test

Il est possible d'utiliser les logins et mots de passe suivants pour se connecter au site web de Ligue 1.

Login	Mot de passe
dupont.jean@free.fr	p@ssw0rd1dEux3
annmarie.michelle@gmail.com	p@ssw0rd1dEux3
alain.terrieur@yahoo.fr	p@ssw0rd1dEux3

Figure 1. Identifiants de connexion

2. Conception du site web

2.1. Architecture MVC

2.1.1. Définition

Model-View-Controller (MVC) est un modèle d'architecture logicielle d'interface graphique créé par l'ingénieur et informaticien norvégien Trygve Reenskaug¹ en 1978 et très populaire dans les applications Web. Ce modèle se compose de trois types de modules avec trois rôles différents : modèle, vue et contrôleur.

- Modèle : Contient les données à afficher.
- Vue : Contient une représentation de l'interface graphique.
- Contrôleur : Contient la logique des actions effectuées par l'utilisateur.

2.1.2. Application dans le site web

Dans le cadre de la création d'un site web pour la Ligue 1, nous avons adopté une architecture MVC (Modèle-Vue-Contrôleur) pour assurer une structure organisée. Cette architecture se manifeste à travers trois dossiers principaux : « model », « vue » et « controller ». En plus de ceux-ci, nous avons également inclus quelques dossiers supplémentaires pour le style CSS et le code JavaScript. Le code PHP a été structuré de la manière suivante :

- « Model » : Ce dossier contient le fichier PHP responsable de la connexion à la base de données, ainsi que des classes écrites en PHP, telles que la classe « Club ».
- « Vue » : Les fichiers PHP présents dans ce dossier contiennent du code HTML qui sera affiché à l'écran.
- « Controller » : Ce dossier regroupe le code PHP qui sera exécuté côté serveur, permettant ainsi de gérer les différentes actions et fonctionnalités du site.

¹ https://fr.wikipedia.org/wiki/Trygve_Reenskaug

WEB – Ligue 1

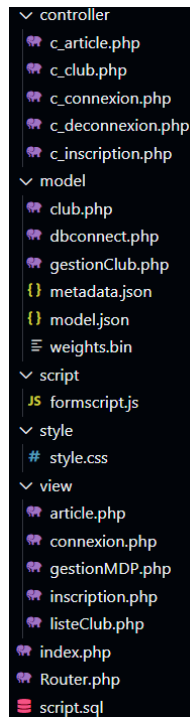


Figure 2. Arborescence du site

2.2. Routeur

Le routeur est une composante essentielle de notre site web de la Ligue 1. Il permet de gérer les différentes requêtes et d'acheminer les utilisateurs vers les bonnes pages en fonction de l'URL demandée. Voici comment le routeur a été mis en place.

2.2.1. Classe Router

Nous avons créé une classe appelée « Router » qui encapsule la logique de routage. Cette classe dispose d'un tableau privé appelé « routes » qui stocke les différentes correspondances entre les URL et les fichiers de contrôleurs associés.

```
public function __construct() {  
    $this->routes=[];  
}
```

Figure 3. Constructeur de la classe Router

Dans le constructeur de la classe, nous initialisons ce tableau vide. Ensuite, nous avons une méthode « addRoute() » qui permet d'ajouter une nouvelle route au routeur. Cette méthode prend en paramètres l'URL à associer et le fichier de contrôleur correspondant. Lorsqu'une demande est faite avec cette URL, le routeur saura quel fichier de contrôleur utiliser.

```
// Ajoute une route au routeur  
public function addRoute($url, $controllerFile) {  
    $this->routes[$url] = $controllerFile;  
}
```

Figure 4. Méthode addRoute()

WEB – Ligue 1

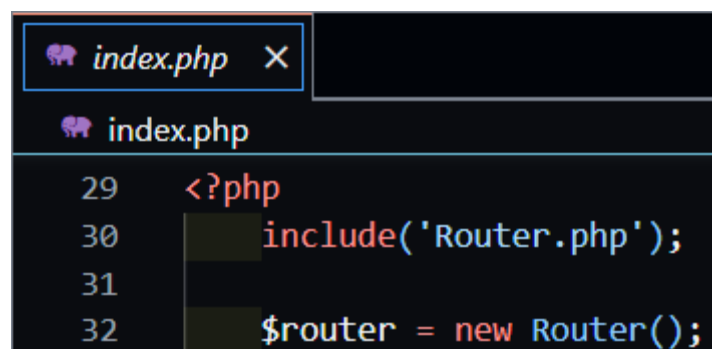
La méthode « `execute()` » est responsable du traitement de la demande actuelle. Elle prend en paramètre l'URL demandée. Si cette URL correspond à l'une des routes enregistrées, le fichier de contrôleur correspondant est inclus. Si le fichier existe, il est inclus à l'exécution. Sinon, une erreur est renvoyée indiquant que le contrôleur n'a pas été trouvé. Si l'URL ne correspond à aucune route, une erreur 404 est renvoyée, indiquant que la page demandée n'a pas été trouvée.

```
// Traite la demande actuelle
public function execute($url) {
    if (array_key_exists($url, $this->routes)) {
        // Si l'URL correspond à une route, incluez le fichier du contrôleur
        $controllerFile = $this->routes[$url];
        if (file_exists($controllerFile)) {
            include_once($controllerFile);
        } else {
            // Gérer les erreurs si le fichier du contrôleur n'existe pas
            echo "Erreur : Contrôleur non trouvé";
        }
    } else {
        // Gérer les erreurs 404 si l'URL n'est pas trouvée
        echo "Page non trouvée (Erreur 404)";
    }
}
```

Figure 5. Méthode `execute()`

2.2.2. Intégration du Router dans un code HTML

Pour intégrer le routeur dans notre site web, nous avons utilisé un fichier « `index.php` ». Dans ce fichier, nous incluons la classe « Router » à partir du fichier « `Router.php` ». Ensuite, nous créons une instance du routeur en utilisant le constructeur.



```
29 <?php
30 include('Router.php');
31
32 $router = new Router();
```

Figure 6. Intégration du routeur

Ensuite, nous ajoutons les différentes routes en utilisant la méthode « `addRoute()` ». Chaque route est associée à un URL spécifique et à un fichier de contrôleur correspondant.

```
$router->addRoute("/accueil", "view/accueil.php");
$router->addRoute("/clubs", "controller/c_club.php");
$router->addRoute("/inscription", "controller/c_inscription.php");
$router->addRoute("/gestionmdp", "view/gestionMDP.php");
$router->addRoute("/article", "controller/c_article.php");
$router->addRoute("/connexion", "controller/c_connexion.php");
$router->addRoute("/deconnexion", "controller/c_deconnexion.php");
```

Figure 7. Ajout des différentes routes

WEB – Ligue 1

Enfin, la variable globale « `$_SERVER["REQUEST_URI"]` » est utilisée pour récupérer l'URL demandée par l'utilisateur. Nous passons ensuite cette URL à la méthode « `execute` » du routeur, qui se charge de traiter la demande et d'inclure le bon fichier de contrôleur. De plus, le code redirige automatiquement vers la page d'accueil dès son exécution, grâce à la condition qui redirige vers la page « `/accueil` », si l'URL actuel est « `/index.php` ».

```
$_SERVER["REQUEST_URI"];

$routeur->execute($_SERVER["REQUEST_URI"]);

$current_url = $_SERVER['REQUEST_URI'];
if ($current_url == '/index.php') {
    header('Location: /accueil');
    exit();
}
```

Figure 8. Fonctionnement des routes

Cela permet de gérer de manière efficace les différentes pages et fonctionnalités du site web de la Ligue 1 en utilisant l'architecture MVC et le routeur.

3. Présentation des pages du site

3.1. Page Clubs

La page Clubs affiche une liste des différents clubs enregistrés dans la base de données, présentée sous la forme d'un tableau en HTML. Chaque club est répertorié avec des informations telles que son nom, et la ligue à laquelle le club appartient (dans le cas où les clubs appartenant à la ligue 2 sont également inscrites en base de données). Cette page permet aux utilisateurs de visualiser facilement tous les clubs associés à la Ligue 1.

ID du club	Nom du club	Ligue du club
1	Paris-SG	L1
2	Lens	L1
3	Lorient	L1
4	Rennes	L1
5	Marseille	L1
6	Lille	L1
7	Monaco	L1
8	Lyon	L1
9	Clermont	L1
10	Toulouse	L1
11	Troyes	L1
12	Nice	L1
13	Montpellier	L1
14	Reims	L1
15	Nantes	L1
16	Strasbourg	L1
17	Brest	L1
18	Auxerre	L1
19	AC Ajaccio	L1
20	Angers	L1

WEB – Ligue 1

Figure 9. Tableau des clubs

3.2. Page Articles

La page Articles est spécialement dédiée aux actualités et aux articles liés à la Ligue 1. Actuellement, un seul article est disponible sur cette page. Les utilisateurs peuvent réagir à cet article en publiant des commentaires. Cela encourage l'interaction et la participation des utilisateurs, leur donnant l'opportunité de partager leurs opinions et leurs réflexions sur les sujets abordés.



Figure 10. Page des articles

3.3. Page Inscription

La page Inscription a pour objectif de permettre aux utilisateurs de créer un compte sur le site. Elle présente un formulaire comprenant des champs tels que le nom, le prénom, l'adresse e-mail, un mot de passe sécurisé, le sexe (homme, femme ou autre), le choix du club qu'ils souhaitent soutenir et sa photo de profil. En remplissant ce formulaire, les utilisateurs peuvent s'inscrire et bénéficier des fonctionnalités personnalisées et de l'interaction sur le site en tant que membre enregistré.

Inscription

Nom

Prénom

Courriel

Mot de passe

Paris-SG

Sexe

☐ Homme ☐ Femme ☐ Autre

Photo de profil

Aucun fichier choisi

Figure 11. Formulaire d'inscription

WEB – Ligue 1

3.4. Page Connexion

Une fois que l'utilisateur a créé son compte avec succès, il est alors nécessaire de lui permettre de se connecter au site à l'aide des identifiants qu'il vient de créer. C'est là que la page de connexion intervient. Elle présente un formulaire avec deux champs obligatoires : l'adresse courriel et le mot de passe.

L'utilisateur doit saisir son adresse courriel dans le premier champ du formulaire. Il est important de noter que l'adresse courriel doit correspondre à celle enregistrée lors de l'inscription. Ensuite, l'utilisateur doit saisir son mot de passe dans le deuxième champ du formulaire. Le mot de passe est une information confidentielle et sécurisée, permettant de garantir l'authentification de l'utilisateur.

Une fois que l'utilisateur a rempli les champs requis, il peut soumettre le formulaire en cliquant sur le bouton de connexion. Le système vérifie alors les informations fournies par l'utilisateur en les comparant aux données enregistrées lors de l'inscription. Si les informations correspondent, l'utilisateur est redirigé vers la page d'accueil du site en tant que membre connecté.

En revanche, si les informations saisies ne correspondent pas aux données enregistrées, un message d'erreur est affiché pour informer l'utilisateur que la connexion a échoué. Dans ce cas, l'utilisateur peut soit vérifier et corriger les informations saisies, soit récupérer son mot de passe en utilisant la fonctionnalité de réinitialisation du mot de passe.

4. Conception du formulaire d'inscription

4.1. Choix des champs du formulaire

Le formulaire d'inscription comprend les champs suivants :

- Nom : champ de texte pour saisir le nom de l'utilisateur.
- Prénom : champ de texte pour saisir le prénom de l'utilisateur.
- Courriel : champ de texte de type email pour saisir l'adresse e-mail de l'utilisateur.
- Mot de passe : champ de texte pour saisir le mot de passe de l'utilisateur.
- Club : Liste déroulante contenant les noms des clubs inscrits en base de données.
- Sexe : Boutons radio permettant à l'utilisateur de sélectionner le sexe auquel il appartient.
- Photo de profil : Bouton permettant à l'utilisateur de sélectionner une image qui sera téléchargée sur le site.

4.2. Ergonomie et design du formulaire

4.2.1. Principes de l'expérience utilisateur

Pour garantir une bonne expérience utilisateur, le formulaire respecte les principes suivants :

- Lisibilité : les champs de texte et les éléments sont affichés de manière claire et facilement identifiables.
- Facilité de saisie : les champs obligatoires sont indiqués comme tels et les types de données attendus (nom, prénom, adresse e-mail) sont spécifiés.
- Feedback visuel : lorsque l'utilisateur remplit correctement un champ, celui-ci est mis en évidence avec une bordure de couleur verte.

WEB – Ligue 1

4.2.2. Adaptabilité aux divers dispositifs (responsive design)

Le formulaire est conçu en utilisant le principe du responsive design, ce qui signifie qu'il s'adapte de manière optimale à différents dispositifs tels que les ordinateurs de bureau, les tablettes et les smartphones. Ainsi, les utilisateurs peuvent remplir le formulaire confortablement, quel que soit le dispositif utilisé.

Pour assurer cette adaptabilité, les règles CSS appropriées sont définies dans le fichier de style « style.css ». Par exemple, les éléments du formulaire sont centrés horizontalement et alignés verticalement pour une présentation équilibrée.

```
.formulaire {  
  font-family: var(--police-ecriture);  
  display: flex;  
  justify-content: space-between; /* Pour aligner à gauche et à droite */  
  align-items: center; /* Pour centrer verticalement */  
  margin-bottom: 10px; /* Ajoute un espace entre les éléments du formulaire */  
}
```

Figure 12. Style CSS basique du formulaire

Ces parties couvrent la conception du formulaire dans le contexte de création du site web de la Ligue 1.

5. Validation du Formulaire d'inscription

Lors de l'inscription sur notre site web, nous avons mis en place un processus de validation du formulaire pour garantir l'intégrité des données fournies par les utilisateurs. Cette validation comprend des contrôles côté client et côté serveur. Voici comment nous avons implémenté ces contrôles.

5.1. Contrôles côté client

Nous avons effectué certains contrôles au niveau du navigateur web de l'utilisateur pour garantir que les champs obligatoires sont remplis et que les données sont correctement formatées.

5.1.1. Vérifications des champs obligatoires

Les champs tels que « Nom », « Prénom », « Courriel » et « Mot de passe » sont marqués comme obligatoires dans le formulaire. Cela signifie que l'utilisateur doit remplir ces champs pour pouvoir soumettre le formulaire. Si l'un de ces champs est laissé vide, une notification s'affichera pour indiquer à l'utilisateur de les compléter.

5.1.2. Formatage des données (adresse email)

Nous avons également effectué une vérification de formatage pour le champ « Courriel » en utilisant le type d'entrée « email ».

WEB – Ligue 1

```
<div class="formulaire">
  <input type="text" id="nom" name="nom" placeholder="Nom" required>
</div>
<div class="formulaire">
  <input type="text" id="prenom" name="prenom" placeholder="Prénom" required>
</div>
<div class="formulaire">
  <input type="email" id="mail" name="mail" placeholder="Courriel" required>
</div>
<div class="formulaire">
  <input type="text" id="mdp" name="mdp" placeholder="Mot de passe" required>
</div>
```

Figure 13. Types d'entrée des champs de texte du formulaire

Cela garantit que l'adresse e-mail fournie par l'utilisateur est dans un format valide. Si l'adresse e-mail n'est pas correctement formatée, une notification s'affiche pour informer l'utilisateur de corriger son adresse e-mail.

Inscription

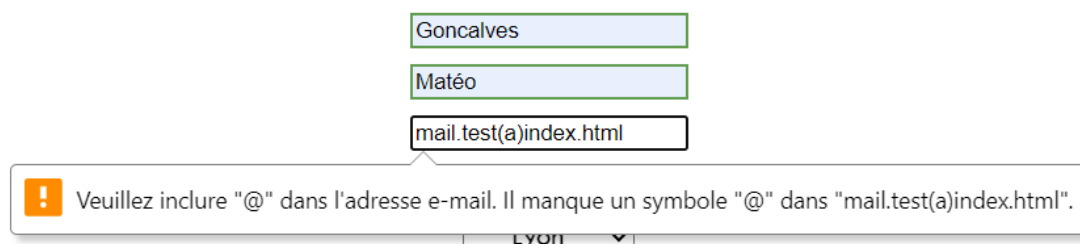


Figure 14. Tentative d'inscription avec une adresse mail invalide

5.2. Contrôles côté serveur

En plus des contrôles côté client, nous avons également effectué des contrôles côté serveur pour garantir la sécurité et la validité des données soumises.

5.2.1. Politiques de mot de passe

Nous avons défini des politiques de mot de passe strictes pour assurer la sécurité des comptes des utilisateurs. Les politiques stipulent que le mot de passe doit avoir une longueur minimale de 12 caractères et doit contenir au moins une lettre majuscule, une lettre minuscule, un chiffre et un caractère spécial (encadré en rouge sur la figure 14). Si l'utilisateur tente de soumettre un mot de passe qui ne respecte pas ces critères, une erreur sera renvoyée, l'informant des exigences de mot de passe.

5.2.2. Stockage sécurisé des mots de passe

Pour garantir la sécurité des informations sensibles des utilisateurs, tels que les mots de passe, nous utilisons des techniques de stockage sécurisé. Dans notre base de données, nous avons une table « UTILISATEUR » avec une colonne « PASSWORD_UTI » qui stocke les mots de passe hachés à l'aide de la méthode « base64_encode() ».

WEB – Ligue 1

```
$mail = isset($_POST['mail']) ? $_POST['mail'] : null;
$mdp = isset($_POST['mdp']) ? $_POST['mdp'] : null;
$list = isset($_POST['list']) ? $_POST['list'] : null;
$sexe = isset($_POST['sexe']) ? $_POST['sexe'] : null;
$image64 = isset($_POST['image']) ? $_POST['image'] : null;

// Traitement du fichier image
$image = $_FILES["image"]["name"];
$target_dir = "../img/"; // Répertoire où le fichier image sera enregistré
$target_file = $target_dir . basename($_FILES["image"]["name"]);

// Vérifie que les clés nécessaires sont définies
if ($nom && $prenom && $mail && $mdp && $list && $sexe && $image) {
    $sql = "INSERT INTO utilisateur (id_club, nom_util, prenom_util, mail_util, password_util, sexe_util, image_util) VALUES
    ($requete = $pdo->prepare($sql);

    $mdp = base64_encode($mdp);
    $image64 = base64_encode($image);

    // Lie les valeurs aux paramètres de la requête.
    $requete->bindParam(':nom', $nom, PDO::PARAM_STR);
    $requete->bindParam(':prenom', $prenom, PDO::PARAM_STR);
    $requete->bindParam(':mail', $mail, PDO::PARAM_STR);
    $requete->bindParam(':mdp', $mdp, PDO::PARAM_STR);
    $requete->bindParam(':idclub', $list, PDO::PARAM_STR);
```

Figure 15. Hashage du mot de passe

Le mot de passe est vérifié lors de l'inscription pour s'assurer qu'il respecte les politiques définies, puis il est haché en base 64 avant d'être stocké dans la base de données. Cela garantit que même en cas d'accès non autorisé à la base de données, les mots de passe des utilisateurs restent sécurisés.

```
CREATE TABLE UTILISATEUR(
    ID_UTI serial NOT NULL,
    ID_CLUB int NOT NULL,
    NOM_UTI varchar(30) NOT NULL,
    PRENOM_UTI varchar(30) NOT NULL,
    SEXE_UTI varchar(15) NOT NULL,
    PASSWORD_UTI varchar(64) NOT NULL CHECK (LENGTH(PASSWORD_UTI) >= 12 AND
    PASSWORD_UTI ~ '[A-Z]' AND
    PASSWORD_UTI ~ '[a-z]' AND
    PASSWORD_UTI ~ '[0-9]' AND
    PASSWORD_UTI ~ '[@#$$%^&*()-=_+{}|;:.,<>?/]',
    DATE_INSCRIPTION date DEFAULT CURRENT_DATE,
    IMAGE_UTI text NULL,
    MAIL_UTI text NULL,
    CONSTRAINT PK_UTILISATEUR PRIMARY KEY (ID_UTI),
    CONSTRAINT UC_UTILISATEUR_MAIL_UTI UNIQUE (MAIL_UTI)
);
```

Figure 16. Conception de la table Utilisateur

Grâce à ces contrôles côté client et côté serveur, nous assurons la validité et la sécurité des données soumises par les utilisateurs lors de leur inscription sur notre site web de la Ligue 1.

6. Sécurité contre les Attaques par Injection

Les attaques par injection sont des techniques d'exploitation courantes visant à compromettre la sécurité d'une application en injectant du code malveillant dans une entrée utilisateur. Ces attaques peuvent être particulièrement dangereuses, car elles permettent à un attaquant d'exécuter du code non autorisé ou d'accéder à des informations sensibles.

WEB – Ligue 1

6.1. Compréhension des attaques par injection

6.1.1. Injections SQL

Une injection SQL est une technique utilisée pour attaquer les applications web en insérant du code SQL malveillant. Cela se produit lorsque les données fournies par l'utilisateur ne sont pas correctement vérifiées avant d'être utilisées dans une requête SQL. L'attaquant peut alors manipuler la requête et exécuter ses propres instructions SQL pour obtenir des informations sensibles, modifier des données ou prendre le contrôle du système. Cela peut se produire en exploitant une faille de sécurité dans l'application. Pour se protéger contre les injections SQL, il est important de valider et filtrer correctement les données utilisateur avant de les utiliser dans une requête SQL.

6.1.2. Cross-Site Scripting (XSS)

Le Cross-Site Scripting (XSS) est une vulnérabilité de sécurité dans les applications web qui permet à un attaquant d'injecter et d'exécuter du code script (généralement du code JavaScript) sur les navigateurs des utilisateurs finaux. Cela peut entraîner des conséquences néfastes telles que le vol d'informations sensibles, la modification du contenu de la page ou les redirections vers des sites malveillants.

6.2. Utilisation de requêtes préparées

Pour renforcer la sécurité de l'application contre les attaques par injection, il est recommandé d'utiliser des requêtes préparées avec des paramètres liés. Les requêtes préparées séparent les instructions SQL des données utilisateur, ce qui empêche les attaquants d'injecter du code malveillant dans les requêtes.

```
// Vérifie si le formulaire de commentaire a été soumis
if (isset($_POST['commentsubmit'])) {

    // Récupère les données du formulaire
    $saisie = isset($_POST['saisie']) ? $_POST['saisie'] : null;

    // Vérifie que les clés nécessaires sont définies
    if ($saisie != null && !empty($saisie)) {
        $id_news = 1;
        $id_utilisateur = 4;

        $sql = "INSERT INTO commentaire (id_news, id_utilisateur, desc_com) VALUES (:id_news, :id_utilisateur, :saisie)";
        $requete = $pdo->prepare($sql);

        // Lie les valeurs aux paramètres de la requête.
        $requete->bindParam(':id_news', $id_news, PDO::PARAM_INT);
        $requete->bindParam(':id_utilisateur', $id_utilisateur, PDO::PARAM_INT);
        $requete->bindParam(':saisie', $saisie, PDO::PARAM_STR);
        $requete->execute();

        $pdo = null;

        // Redirection vers la même page après l'insertion
        header("Location: /article");
        exit();

        include('view/article.php');
    } else {
        include('view/article.php');
        echo "<p class='alert'>Veuillez remplir tous les champs du formulaire.</p>";
    }
}
```

Figure 17. Utilisation d'une requête préparée

Dans cet extrait de code, une requête préparée est utilisée pour insérer un commentaire dans la base de données. Les valeurs des paramètres « (:id_news, :id_utilisateur, :saisie) » sont liées aux variables correspondantes à l'aide de la méthode « bindParam() ». Cela garantit que les valeurs sont correctement échappées et empêche les attaques par injection. En utilisant des requêtes préparées dans l'ensemble de l'application, la sécurité contre les attaques par injection SQL et XSS est renforcée.

WEB – Ligue 1

7. Intégration d'une API

7.1. Test de l'API, liste des communes

Pour ajouter une fonctionnalité permettant de sélectionner une commune en fonction du code postal saisi par l'utilisateur, nous avons décidé d'utiliser l'API "Géo API" fournie par le gouvernement français. Cette API permet de récupérer la liste des communes correspondant à un code postal donné.

Dans un premier temps, ont été ajoutés les champs supplémentaires nécessaires au bon fonctionnement de l'API. À savoir, un champ pour le code postal et une liste déroulante présentant les différentes communes.

```
<div class="formulaire">
  <input type="text" id="nom" name="nom" placeholder="Nom" required>
</div>
<div class="formulaire">
  <input type="text" id="prenom" name="prenom" placeholder="Prénom" required>
</div>
<div class="formulaire">
  <input type="email" id="mail" name="mail" placeholder="Courriel" required>
</div>
<div class="formulaire">
  <input type="text" id="mdp" name="mdp" placeholder="Mot de passe" required>
</div>
<div class="formulaire">
  <input type="text" id="codepostal" name="codepostal" placeholder="Code Postal" required>
</div>

<div class="formulaire">
  <select class="list" id="list" name="list">
    <option>Commune</option>
  </select>
</div>
```

Figure 18. Ajout des champs nécessaires

Ensuite, nous avons testé l'utilisation de cette API en effectuant une requête HTTP GET sur l'URL « <https://geo.api.gouv.fr/communes?codePostal=> » suivie du code postal saisi par l'utilisateur. Cela nous a permis de vérifier que l'API renvoyait bien la liste des communes correspondantes dans un format JSON.

Enfin, nous avons implémenté cette fonctionnalité dans notre site web. Lorsque l'utilisateur saisit un code postal dans le champ prévu à cet effet, une fonction JavaScript est appelée pour effectuer la requête à l'API et mettre à jour la liste déroulante des communes.

WEB – Ligue 1

```
<div class="formulaire">
  <select class="list" id="list" name="list">
    <script>
      xhr = new XMLHttpRequest();
      xhr.responseType = 'json';
      xhr.open('GET', "https://geo.api.gouv.fr/communes?codePostal=26300");
      xhr.send();

      xhr.onload = function () {
        communeJSON = xhr.response;
        var selectElement = document.getElementById("list");
        for (i = 0; i < communeJSON.length; i++) {
          var optionElement = document.createElement("option");
          optionElement.textContent = communeJSON[i]["nom"];
          selectElement.appendChild(optionElement);
        }
      };
    </script>
  </select>
</div>
```

Figure 19. Test de l'API en alimentant la liste déroulante

Le code précédent montre la structure HTML des champs de formulaire ainsi que l'implémentation JavaScript de la fonction « `listerCommunes()` » qui interroge l'API et met à jour la liste déroulante.



The image shows a web form with a blue header bar containing the text "Alixan". Below the header is a list of commune names: Barbières, Beauregard-Baret, Bésayes, Bourg-de-Péage, Charpey, Châteauneuf-sur-Isère, Chatuzange-le-Goubet, Marches, Rochefort-Samson, Jaillans, and Saint-Vincent-la-Commanderie. At the bottom of the list is a dropdown menu with "Alixan" selected and a downward arrow.

Figure 20. Résultat du test

WEB – Ligue 1

7.2. Adaptation au champ Code Postal

L'attribut « oninput » est un événement HTML qui se déclenche lorsque la valeur d'un élément de formulaire est modifiée. Il est utilisé pour exécuter du code JavaScript en réponse à cette modification.

Lorsqu'un utilisateur saisit ou modifie une valeur dans un champ de formulaire, l'événement « oninput » est déclenché. Cela permet de capturer cet événement et d'exécuter une fonction JavaScript spécifiée.

```
<div class="formulaire">
  <input type="text" id="codepostal" name="codepostal" placeholder="Code Postal" oninput="listerCommunes()" required>
</div>
```

Figure 21. Ajout de l'attribut « oninput »

```
<div class="formulaire">
  <select class="list" id="list" name="list">
    <script>
      function listerCommunes() {
        var codepostal = document.getElementById("codepostal").value;
        var xhr = new XMLHttpRequest();
        xhr.responseType = 'json';
        xhr.open('GET', "https://geo.api.gouv.fr/communes?codePostal=" + codepostal);
        xhr.send();

        xhr.onload = function () {
          var communeJSON = xhr.response;
          var selectElement = document.getElementById("list");
          selectElement.innerHTML = ""; // Effacer les options existantes avant de mettre à jour
          for (var i = 0; i < communeJSON.length; i++) {
            var optionElement = document.createElement("option");
            optionElement.textContent = communeJSON[i]["nom"];
            selectElement.appendChild(optionElement);
          }
        };
      }
    </script>
  </select>
</div>
```

Figure 22. Modification du code JavaScript de la liste déroulante

8. Connexion de l'utilisateur

8.1. Vérification du formulaire

Afin de sécuriser le processus de connexion des utilisateurs, nous avons implémenté une vérification rigoureuse du formulaire de connexion. Celui-ci comprend deux champs obligatoires :

Adresse mail :

- Le premier champ du formulaire permet à l'utilisateur de saisir son adresse mail.
- Ce champ est de type « email » et comporte l'attribut « required » pour s'assurer que l'utilisateur renseigne une adresse mail valide.
- Lors de la soumission du formulaire, le système vérifie que l'adresse mail saisie correspond bien à celle enregistrée dans la base de données lors de l'inscription de l'utilisateur.

Mot de passe :

- Le deuxième champ du formulaire permet à l'utilisateur de saisir son mot de passe.

WEB – Ligue 1

- Ce champ est de type « password » et comporte également l'attribut « required » pour garantir que l'utilisateur renseigne bien son mot de passe.
- Lors de la soumission du formulaire, le système compare le mot de passe saisi avec celui enregistré dans la base de données pour vérifier l'authenticité de l'utilisateur.

```
<div class="formulaire">
  <input type="email" id="mail" name="mail" placeholder="Adresse mail" required>
</div>
<div class="formulaire">
  <input type="password" id="password" name="password" placeholder="Mot de passe" required>
</div>
```

Figure 23. Champs du formulaire de connexion

La présence des attributs « required » sur ces deux champs oblige l'utilisateur à les renseigner avant de soumettre le formulaire. Cela permet de s'assurer que toutes les informations nécessaires à la connexion sont bien fournies.

Lors de la validation du formulaire, le système vérifie que les informations saisies correspondent bien à celles enregistrées dans la base de données. Si c'est le cas, l'utilisateur est alors authentifié et redirigé vers la page d'accueil du site. Dans le cas contraire, un message d'erreur est affiché, invitant l'utilisateur à vérifier ses informations de connexion.

Cette vérification rigoureuse du formulaire de connexion contribue à renforcer la sécurité et l'intégrité de notre application web dédiée à la ligue 1.

8.2. Dynamisme des pages

8.2.1. Header du site

Le changement dynamique de l'en-tête en fonction du statut de connexion de l'utilisateur est géré grâce à du code PHP, lequel est encapsulé dans des balises spécifiques. Cette approche permet d'exécuter ce code directement sur le serveur.

Pour un utilisateur non connecté	Site Ligue 1 Accueil Clubs Article Inscription Connexion
Pour un utilisateur connecté	Site Ligue 1 Accueil Clubs Article Déconnexion

Figure 24. Comparaison du header

Déconnexion

- La condition « `if (isset($_SESSION['id']))` » vérifie si l'utilisateur est connecté, c'est-à-dire si la variable de session « `$_SESSION['id']` » est définie.
- Si l'utilisateur est connecté, un lien "Déconnexion" est affiché, pointant vers la page « `/deconnexion` ».
- Lorsque l'utilisateur clique sur ce lien, la page de déconnexion est chargée, permettant de détruire la session en cours et de déconnecter l'utilisateur.

WEB – Ligue 1

Inscription

- Si l'utilisateur n'est pas connecté (c'est-à-dire que la variable de session « \$_SESSION['id'] » n'est pas définie), un lien "Inscription" est affiché, pointant vers la page « /inscription ».
- Lorsque l'utilisateur clique sur ce lien, la page d'inscription est chargée, permettant à l'utilisateur de créer un nouveau compte sur le site.

Connexion

- Toujours dans le cas où l'utilisateur n'est pas connecté, un lien "Connexion" est affiché, pointant vers la page « /connexion ».
- Lorsque l'utilisateur clique sur ce lien, la page de connexion est chargée, permettant à l'utilisateur de s'authentifier sur le site.
- Cette partie du code de l'en-tête permet donc de gérer dynamiquement l'affichage des liens de déconnexion, d'inscription et de connexion en fonction de l'état de connexion de l'utilisateur, améliorant ainsi l'expérience de navigation sur le site web de la Ligue 1.

```
<table border="0" class="header_link">
  <tr>
    <td><a href="/">Accueil</a></td>
    <td><a href="/clubs">Clubs</a></td>
    <td><a href="/article">Article</a></td>
    <?php
      session_start();

      if (isset($_SESSION['id'])) {
        echo "<td><a href='/deconnexion'>Déconnexion</a></td>";
      } else {
        echo "<td><a href='/inscription'>Inscription</a></td>";
        echo "<td><a href='/connexion'>Connexion</a></td>";
      }
    ?>
  </tr>
</table>
```

Figure 25. Intégration du dynamisme du header en cas de connexion

8.2.2. Page des articles

La page affichant les articles dispose d'un formulaire permettant aux utilisateurs connectés de publier des commentaires. Ce formulaire est généré de manière dynamique à l'aide de code PHP.

WEB – Ligue 1

```
<form method="post">
  <?php
  if(isset($_SESSION['id'])) {
    echo '<input type="text" id="saisie" name="saisie" class="commentInput" placeholder="Commentaire" required>';
    echo '<button type="commentsubmit" name="commentsubmit">Publier</input>';
  } else {
    echo "<h2>Veuillez vous connecter pour publier un commentaire.</h2>";
  }
  ?>
</form>
```

Figure 26. Intégration du dynamisme de la page des articles en cas de connexion

Lorsqu'un utilisateur est connecté, le code PHP affiche un champ de saisie pour le commentaire ainsi qu'un bouton « Publier ». Cela permet aux utilisateurs authentifiés de soumettre facilement leurs commentaires sur les articles.

En revanche, si l'utilisateur n'est pas connecté, le code PHP affiche un message l'invitant à se connecter avant de pouvoir publier un commentaire. Cette vérification de l'état de connexion garantit que seuls les utilisateurs autorisés peuvent poster des commentaires, préservant ainsi l'intégrité des échanges sur la plateforme.

Cette fonctionnalité dynamique enrichit l'expérience utilisateur en donnant aux membres connectés la possibilité de participer activement aux discussions, tout en protégeant la page des commentaires des utilisateurs non authentifiés.

Pour un utilisateur non connecté	Terrieur Alain zfulhregherofghierghiergheirg Veuillez vous connecter pour publier un commentaire.
Pour un utilisateur connecté	Terrieur Alain zfulhregherofghierghiergheirg <input type="text" value="Commentaire"/> <input type="button" value="Publier"/>

Figure 27. Comparaison de la page des articles

9. Conclusion

En conclusion, le développement du site web de la Ligue 1 a été abordé en mettant l'accent sur différents aspects importants. La conception du site a été réalisée en suivant l'architecture MVC, ce qui permet une organisation claire du code. Le routeur a été intégré pour gérer les différentes routes du site et diriger les requêtes vers les bonnes actions.

Les pages du site, telles que la page des clubs, des articles et d'inscription, ont été présentées avec une attention particulière portée à leur conception ergonomique et leur design responsive. Un formulaire a été conçu en choisissant les champs appropriés et en appliquant les principes de l'expérience utilisateur.

La validation du formulaire a été traitée en mettant en place des contrôles côté client pour vérifier les champs obligatoires et formater les données de manière adéquate. Des contrôles côté serveur ont également été implémentés pour gérer les erreurs et afficher des messages d'alerte. La gestion des mots de passe a été abordée en définissant des politiques de mot de passe robustes et en assurant un stockage sécurisé des mots de passe dans la base de données.

WEB – Ligue 1

Enfin, la sécurité contre les attaques par injection, telles que les injections SQL et le Cross-Site Scripting (XSS), a été traitée en recommandant l'utilisation de requêtes préparées pour empêcher l'injection de code malveillant dans les requêtes.

10. Bibliographie

Figure 1.	Identifiants de connexion	3
Figure 2.	Arborescence du site	4
Figure 3.	Constructeur de la classe Router.....	4
Figure 4.	Méthode addRoute()	4
Figure 5.	Méthode execute()	5
Figure 6.	Intégration du routeur	5
Figure 7.	Ajout des différentes routes	5
Figure 8.	Fonctionnement des routes.....	6
Figure 9.	Tableau des clubs.....	7
Figure 10.	Page des articles	7
Figure 11.	Formulaire d'inscription	7
Figure 12.	Style CSS basique du formulaire	9
Figure 13.	Types d'entrée des champs de texte du formulaire	10
Figure 14.	Tentative d'inscription avec une adresse mail invalide	10
Figure 15.	Hashage du mot de passe	11
Figure 16.	Conception de la table Utilisateur	11
Figure 17.	Utilisation d'une requête préparée.....	12
Figure 18.	Ajout des champs nécessaires.....	13
Figure 19.	Test de l'API en alimentant la liste déroulante	14
Figure 20.	Résultat du test	14
Figure 21.	Ajout de l'attribut « oninput »	15
Figure 22.	Modification du code JavaScript de la liste déroulante	15
Figure 23.	Champs du formulaire de connexion	16
Figure 24.	Comparaison du header	16
Figure 25.	Intégration du dynamisme du header en cas de connexion	17
Figure 26.	Intégration du dynamisme de la page des articles en cas de connexion	18
Figure 27.	Comparaison de la page des articles	18