

# Dossier Projet Huma Scientio

Titre Professionnel Développeur web et web mobile

Février 2021 - Avril 2022

---

Nicolas SANNA

16 Rue du Bergeret, la Voilerie  
13170 Les Pennes-Mirabeau

## Sommaire

Résumé du projet .....	p. 2
Cahier des charges .....	p. 3
Spécifications techniques .....	p. 4
Réalisations .....	p. 5
Présentation du jeu d'essai .....	p. 36
Veille de sécurité .....	p. 37
Veille technique à partir d'une ressource anglophone .....	p. 42
Extrait de traduction .....	p. 43
Conclusion .....	p. 44

## 1. Compétences du référentiel couvertes par le projet

Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique

Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

## 2. Résumé du projet

Dans le cadre de rapports conservés avec des étudiants et de jeunes professeurs dans le milieu universitaire partout en France avant mon arrivée en formation, diverses ressources relatives aux sciences humaines ont été produites et conservées sur l'application Discord qui sert d'espace de discussion. Ces contenus sont divers : histoire, sociologie, économie, philosophie, art, etc. L'évolution de cet espace et des publications ont rendu nécessaire la création d'une application web permettant une plus grande visibilité ainsi qu'une meilleure gestion des différents contenus. Il s'agit de passer à l'étape supérieure et de rendre accessible et public l'ensemble des productions réalisées depuis plus de 3 ans. Cette application sera utilisée ensuite par les différents membres inscrits, qui pourront publier de manière autonome du contenu. Les membres seront gérés par des administrateurs : plusieurs niveaux d'autorisations sont prévus.

Afin de répondre au cahier des charges de ce projet, j'ai développé les éléments suivants :

### Front-end

- Concevoir la maquette des différentes interfaces utilisateur sur l'ensemble du projet en lien avec la charte graphique établie au préalable avec les personnes concernées.

- A partir de la maquette validée précédemment, produire l'ensemble des codes HTML et CSS pour afficher les interfaces statiques et adaptables comme le back-office d'administration et les formulaires d'ajouts d'articles.
- Mettre en place les différents comportements dynamiques en JavaScript comme l'affichage du menu, la validation des actions des utilisateurs ou encore effectuer une recherche en AJAX parmi les articles.

## Back-end

- Modéliser et créer une base de données directement avec le langage SQL en utilisant le Système de Gestion de Base de Données Relationnelles (SGBDR) MySQL.
- Créer une interface d'administration à l'usage des administrateurs et des auteurs de contenus (articles).
- Créer une interface de gestion de catégories.
- Créer une interface de gestion de comptes utilisateurs.
- Mettre en place un système d'authentification (inscription, connexion), d'autorisation et de droits d'accès pour les commentaires et la publication d'articles.

## 3. Cahier des charges

### 3.1. Objectifs et fonctionnalités attendues

L'objectif du projet est de pouvoir déployer et publier sur une application web l'ensemble des divers contenus actuellement présents sur la plateforme de discussion et permettre ainsi une plus grande accessibilité et visibilité.

Les principales fonctionnalités retenues pour l'application sont les suivantes :

- Un menu de navigation présent sur toutes les pages du site pour :
  - Accéder à tous les articles ;
  - Rechercher un article ;
  - Accéder au back-office d'administration où sont présents les articles en lecture et en écriture en effectuant les quatre opérations du CRUD (*Create, Read, Update, Delete*) pour l'utilisateur connecté ;
  - Accéder à la gestion des utilisateurs, des catégories des commentaires et des articles (uniquement pour les administrateurs du site) ;
  - Modifier ses informations personnelles (email, nom, prénom, pseudo)
- Une page d'accueil avec une présentation des objectifs du site ainsi que la possibilité de consulter les deux derniers articles publiés récemment.

- Une page pour créer un compte utilisateur et un lien de déconnexion placé au niveau du menu visible pour l'utilisateur connecté.
- Une page de connexion (formulaire).

### 3.2. Tableau des fonctionnalités et des droits associés à chaque utilisateur

Droit / Rôle	Visiteur	Nouveau membre	Auteur	Administrateur	Super-Administrateur
Lire les articles					
Publier, modifier, supprimer des articles					
Ajouter des commentaires					
Créer, modifier supprimer des catégories					
Modérer les commentaires					
Modérer les membres					
Modérer les articles					
Nommer un administrateur					

## 4. Spécifications techniques

L'ensemble des interfaces utilisateurs sont alignées sur la charte graphique définie lors du maquetage de l'application. L'affichage doit s'adapter aux différentes tailles d'écrans et permettre un bon rendu de l'ensemble des différentes pages sur différents navigateurs web.

Ce projet est présent sur un dépôt Git et stocké sur mon compte personnel GitHub.

### 4.1.1 Langages utilisés

L'ensemble de l'application a été réalisé à partir de zéro en utilisant les différents langages en natif. Elle possède les caractéristiques techniques suivantes :

- PHP 8.1
- MySQL 8.x
- HTML 5
- CSS 3 (SASS, SCSS)
- JavaScript

Pour la robustesse du code et sa maintenabilité à terme, j'ai choisi de me servir du paradigme de programmation MVC et de l'autochargement des classes de la librairie de dépendances PHP Composer.

#### 4.1.2 Technologies utilisées

- Maquettage et prototypage : Figma
- Environnement de développement (IDE) : Visual Studio Code
- Interface de gestion de bases de données : MySQL Workbench
- Outil de versionning : Git/GitHub

## 5. Réalisations

### 5.1 Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

#### 5.1.1 Maquetter une application

Avant de réaliser l'intégration HTML et CSS des différentes pages du projet, j'ai commencé par la conception de wireframes. Après deux mockups présentés aux personnes participant au cahier des charges, j'ai pu réaliser l'ensemble de la charte graphique en utilisant des couleurs associées aux sciences humaines et en particulier à l'histoire. Je me suis servi des couleurs noires, blanches, orangées (Peru) et grises pour l'ensemble du site avec une teinte de beige pour le fond des pages.

Je suis parvenu à dégager un ensemble cohérent et une structure pour toutes les pages du site notamment pour le header et le footer :

- Pour le header et son bandeau supérieur :
  - Une image de fond ;

- Une bande noire transparente laissant apparaître le titre du site ainsi que le lien menant aux différents articles du site ;
- Le menu de navigation ;
- Pour le footer et son bandeau inférieur :
  - Un bandeau noir contenant la description du site ;
  - Un menu de navigation réduit ;
  - Les droits du créateur ;

Après de nombreuses recherches et de nombreux essais réalisés avec Figma, j'ai finalement retenu une structure spécifique de page avec une vue desktop et une vue mobile.

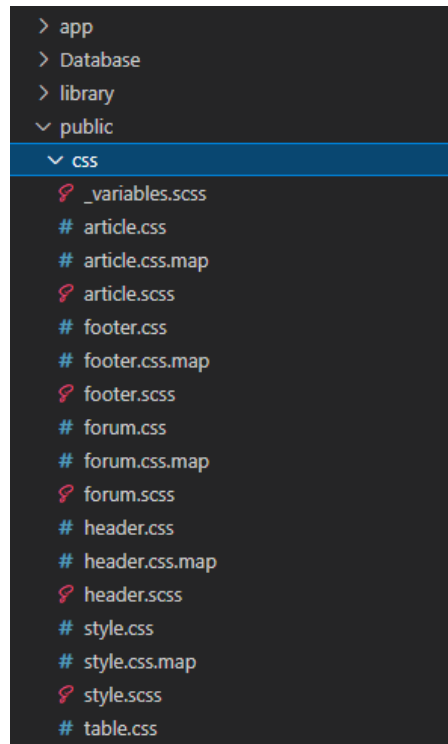
J'ai ensuite procédé au prototypage afin de donner un aperçu virtuel de la navigation et des différentes actions sur les pages de l'application web. Puis j'ai procédé à l'intégration du code en HTML et CSS.

***Les wireframes de la maquette retenue peuvent être consultés en annexe (Annexe 1)***

### 5.1.2 Réaliser une interface utilisateur web statique et adaptable

Afin de réaliser le rendu des différentes pages tant du point de vue du fond (HTML) que de la forme (CSS) et générer les interfaces utilisateurs (UI), je me suis servi du système de templating vu au cours de la formation. Il consiste à se servir d'un fichier .phtml (extension permettant à la fois l'inclusion de code PHP et de HTML) ne servant qu'au rendu et à l'affichage côté client.

J'ai procédé ensuite à la séparation du HTML et du CSS conformément aux normes et en plaçant mes rendus dans le dossier templates et mes feuilles de style dans le dossier public/css.



J'ai ensuite séparé les différentes parties de mes pages en me servant des balises sémantiques de HTML5.

Afin d'écrire le code CSS sur les différentes feuilles de style, je me suis servi du préprocesseur SASS permettant de compiler et d'interpréter l'ensemble du code CSS. Conformément aux normes actuelles, j'ai commencé l'écriture du CSS par le *mobile first*.

Enfin, j'ai inclus les différentes feuilles de style CSS dans le head du base.phtml afin qu'elles soient accessibles sur toutes les pages du site puisque le fichier base.phtml contient les différents templates de pages.

templates/base.phtml

```
<link rel="stylesheet" href="<?=asset('css/style.css');?>">
<link rel="stylesheet" href="<?=asset('css/table.css');?>">
```

### a) CSS général

Sur la feuille de style style.css, j'ai indiqué des propriétés globales pour toute l'application web et dans \_variables.scss les propriétés qui reviennent régulièrement comme la couleur des liens ou encore la police générale de caractères pour tout le site.

public/css/\_variables.scss

```
$colorPeru: peru;
$fontGlobal: 'Open Sans', sans-serif;
```



J'ai appliqué une classe CSS `.Body` sur la balise `body` du `base.phtml` en lui donnant différentes propriétés :

- La couleur du texte par défaut.
- une hauteur et une largeur en précisant que cette propriété passe en priorité par rapport à toutes les autres grâce à `!important`.
- Pas de marges intérieures grâce à la propriété `padding`.
- Pas de marges extérieures grâce à la propriété `margin`.
- J'applique une animation à chaque rafraîchissement et changement de page qui se produit en 0.7 seconde appelée `fadeInFromNone`.

`public/css/style.scss`

```
.Body, html
{
  font-family:$fontGlobal;
  font-size:16px;
  color: rgb(58, 58, 58);
  width: 100% !important;
  height: 100% !important;
  margin: 0;
  padding: 0;
  animation: fadeInFromNone 0.7s ease-out;
  background-color:#fcefd9;

  &-main
  {
    padding: 15px;
    margin-left:20px;
    margin-right:20px;
  }
}
```

L'animation `fadeInFromNone` qui s'exécute en 0.7 seconde consiste à rendre invisible l'intégralité de la page grâce aux propriétés `display:none` et à `opacity:0`. Puis à 1% à faire apparaître la page avec une opacité toujours à zéro, et enfin, une fois à 100% faire apparaître la page grâce à `opacity:1`.

Cela permet une transition plus fluide à l'affichage et une meilleure expérience utilisateur (UX).

`public/css/style.scss`

```
@keyframes fadeInFromNone
{
  0%
  {
    display: none;
    opacity: 0;
  }

  1%
```

```

{
  display: block;
  opacity: 0;
}

100%
{
  display: block;
  opacity: 1;
}
}

```

## b) Menu de navigation

J'ai ensuite codé l'ensemble du menu en HTML. Je me suis servi des balises modernes de HTML5 comme les balises `<nav></nav>` et j'ai utilisé une version personnelle de la méthodologie BEM. Je commence par le nom de la balise puis je nomme les classes plus en profondeur en fonction de leurs spécificités.

templates/header.phtml

```

<nav class="Header-navbarbox">

  <div class="Header-navbarbox-burgerBox">
    <span class="Header-navbarbox-burgerBox-burger" id="burger">≡</span>
    <span class="Header-navbarbox-burgerBox-cross" id="cross">✕</span>
  </div>

  <div class="Header-navbarbox-navbar">
    <ul class="Header-navbarbox-navbar-list">
      <li>
        <a class="Header-navbarbox-navbar-list-a" href="<?=buildUrl('homepage');?>">Accueil</a>
      </li>
      <li>
        <a class="Header-navbarbox-navbar-list-a" href="<?=buildUrl('forum');?>">Forum</a>
      </li>
      <li>
        <a class="Header-navbarbox-navbar-list-a" href="<?=buildUrl('categories');?>">Catégories</a>
      </li>
      <li>
        <a class="Header-navbarbox-navbar-list-a" href="<?= buildUrl('search');?>">Rechercher</a>
      </li>
      [...]
    </ul>
  </div>
</nav>

```

Puis j'ai appliqué un ensemble de règles CSS. Pour le mobile, le menu de navigation est en `display:none` par défaut. Avec la propriété `display:flex`, j'ai aligné à la verticale les différents éléments du menu grâce à la propriété `flex-direction:column`.

public/css/header.scss

```
&-navbar
{
  display:none;

  &-list
  {
    list-style: none;
    display:flex;
    flex-direction:column;
  }
}
```

Au point de rupture à 880px, le menu de navigation est visible en mode desktop grâce à la propriété `display:block`. Enfin, pour que le menu de navigation s'affiche à l'horizontale, je me suis servi de la propriété `flex-direction:row` permettant d'aligner les éléments à l'horizontale.

public/css/header.scss

```
@media screen AND (min-width: 880px)
{
  .Header
  {
    &-navbarbox
    {
      &-burgerBox
      {
        display:none;
      }
      &-navbar
      {
        display:block;

        &-list
        {
          justify-content: center;
          flex-direction:row;

          &-a
          {
            margin-left:5px;
            margin-right:5px;
          }
        }
      }
    }
  }
}
```

```
}
}
```

### c) Formulaires

Pour les formulaires d'inscription, de connexion, de création de catégories ou de gestion utilisateurs, j'ai créé une animation spécifique qui fait apparaître progressivement le formulaire à droite et se place dans son conteneur en revenant à gauche.

J'ai appliqué les classes CSS suivantes sur chacun des formulaires génériques en précisant la durée de l'animation, sa direction et son nom. Une fois ces éléments définis, j'ai créé l'animation en elle-même en précisant que le formulaire devait être à 25% de la gauche de son conteneur et totalement invisible, jusqu'à 0% de la gauche en étant totalement visible.

public/css/style.scss

```
.Form
{
  animation-duration: 2s;
  animation-direction: normal;
  animation-name: glissement;
}
```

public/css/style.scss

```
@keyframes glissement
{
  from
  {
    margin-left: 25%;
    opacity: 0;
  }
  to
  {
    margin-left: 0%;
    opacity: 1;
  }
}
```

***Les résultats visuels du menu de navigation sont consultables en annexe (Annexe 2) en vues mobile et desktop***

### 5.1.3 Développer une interface utilisateur web dynamique

Pour le projet Huma Scientio, j'ai créé un ensemble de quatre éléments dynamiques :

- Un menu burger pour mobile
- Un moteur de recherche en AJAX
- Un aperçu de rendu d'image lors de la création d'un article
- La suppression d'un article en AJAX

Je vais présenter dans un premier temps le menu mobile et dans un second temps le moteur de recherche en AJAX.

### a) Menu mobile

Pour le media query correspondant à la version mobile de l'application web, j'ai créé un menu burger dynamique.

J'ai créé un fichier menu.js placé dans un dossier modules et exporté le module du menu dans main.js placé dans le base.phtml et appelé en bas du template.

public/js/main.js

```
<script type="module" src="<?=asset('js/main.js');?>"></script>
```

Tout d'abord, j'ai récupéré les différents éléments du DOM (Document Object Model) dont le burger, la croix et la barre de navigation grâce aux identifiants (id) et grâce à la classe que je leur ai donnée.

templates/header.phtml

```
<div class="Header-navbarbox-burgerOn">
  <span class="Header-navbarbox-burgerOn-burger" id="burger">☰</span>
  <span class="Header-navbarbox-burgerOn-cross" id="cross">✕</span>
</div>
<div class="Header-navbarbox-navbar">
[...]
```

public/js/modules/menu.js

```
let burger = document.getElementById('burger');
let cross = document.getElementById('cross')
let menu = document.querySelector('.Header-navbarbox-navbar')
```

J'ai créé une classe que j'ai nommée burgerManager elle ne prend pas de paramètre dans le constructeur, en revanche, je lui donne deux événements à écouter grâce à la méthode addEventListener. L'un sur le

burger, l'autre sur la croix. Je donne en premier paramètre l'événement à écouter et en second la méthode à exécuter lorsque l'on clique.

public/js/modules/menu.js

```
export class burgerManager
{
  constructor()
  {
    burger.addEventListener('click', this.burgerEvent);
    cross.addEventListener('click', this.crossEvent);
  }
  [...]
}
```

Au clic sur le bouton burger, on déclenche la méthode `burgerEvent()`. On vérifie si la propriété CSS `display` n'est pas définie ou si sa propriété est en `display:none`. Comme c'est le cas (par défaut le menu mobile est en `display:none`) on entre dans cette condition. On ajoute la propriété CSS `display:block` au menu, de même pour la croix, et l'on fait disparaître le bouton burger en lui donnant la propriété `display:none`.

Enfin, pour une meilleure expérience utilisateur, je précise que cette animation doit durer 2 secondes, sa direction normale, et je lui donne comme animation `glissementMenu` définie dans le CSS en le faisant apparaître progressivement en deux secondes grâce à la propriété `opacity`.

public/js/modules/menu.js

```
burgerEvent ()
{
  if (!menu.style.display || menu.style.display == 'none')
  {
    menu.style.display = 'block';
    cross.style.display = 'block';
    burger.style.display = 'none';
    menu.style.animationDuration = '2s';
    menu.style.animationDirection = 'normal';
    menu.style.animationName = 'glissementMenu';
  }
}
```

De même pour le clic sur la croix. Si le `display` du menu est défini ou égal à `none`, on repasse le menu et la croix en `display:none` et le burger en `display:block` pour le faire réapparaître.

De manière conventionnelle, j'ai créé un dossier JS dans lequel j'ai placé les différents fichiers JavaScript dont `main.js` qui importe l'objet `burgerManager` du dossier `modules/menu.js`. J'instancie l'objet et la classe `burgerManager` dans une constante `burger`.

public/js/main.js

```
import {burgerManager} from './modules/menu.js';
```

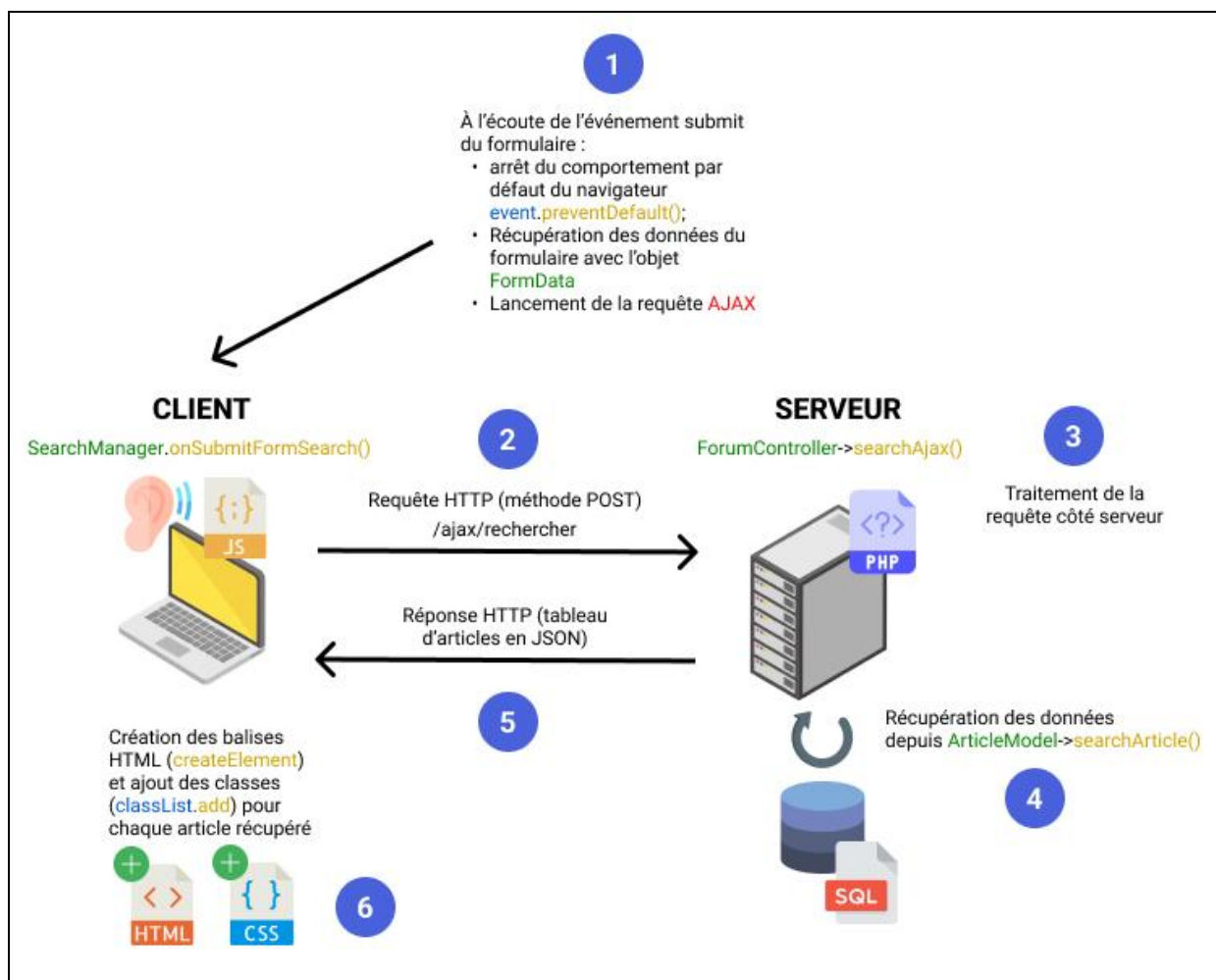
```
const burger = new burgerManager();
```

*Le résultat visuel est consultable en annexe (Annexe 3)*

## b) Moteur de recherche en AJAX

Le projet avait besoin d'un moteur de recherche afin de consulter un ou plusieurs articles. Il était possible de le faire en PHP et en SQL. Mais afin d'éviter de trop solliciter le PHP, j'ai décidé de réaliser ce moteur de recherche en AJAX. De cette manière, c'est le JavaScript qui est à l'initiative de la requête HTTP et va pouvoir communiquer avec le serveur sans recharger la page. Pour des raisons de simplicité et d'optimisation, je me suis servi de l'API fetch, méthode moderne pour effectuer des requêtes asynchrones.

Voici le schéma représentant les différentes étapes du traitement du formulaire de recherche :



Dans le fichier JavaScript `search.js`, je récupère l'objet HTML du formulaire grâce à son id.

`public/js/search.js`

```
const form = document.getElementById('search-form');
```

- 1) Étapes 1 & 2 : Arrêt du comportement par défaut du navigateur, récupération des données du formulaire et lancement de la requête AJAX

A la soumission du formulaire, on pose un écouteur sur l'événement submit et l'on déclenche la méthode `onSubmitFormSearch()`.

`public/js/search.js`

```
form.addEventListener('submit', this.onSubmitFormSearch)
```

On arrête le comportement par défaut du navigateur et on récupère les données grâce à l'objet `FormData`. Si le formulaire de recherche n'est pas vide, et contient bien une chaîne de caractères, alors on peut lancer la requête AJAX.

`public/js/search.js`

```
async onSubmitFormSearch(event)
{
  event.preventDefault();
  const formData = new FormData(form);
  let search = formData.get('search');

  if (search.trim() == '')
  {
    alert("Le champ de recherche est vide");
  }
  [...]
}
```

On récupère dans une constante `URL` l'adresse à interroger, il s'agit de l'attribut `action` du formulaire qui contient le path menant au contrôleur dédié à la requête AJAX.

On stocke dans une constante `options` un tableau associatif contenant différentes données spécifiques à la requête AJAX :

- La méthode : Puisqu'il s'agit d'un formulaire, la méthode est `POST`.
- Le corps ou `body` contient l'objet JavaScript représentant l'élément HTML du formulaire.
- Les headers ou en-têtes afin d'indiquer au serveur qu'il s'agit d'une requête AJAX.

`public/js/search.js`

```
const options =
{
  method: 'post',
```



```

    body: formData,
    headers:
    {
        'X-Requested-With': 'XMLHttpRequest'
    }
}

```

Une fois tous ces paramètres remplis, on peut procéder au lancement de la requête AJAX grâce à l'API fetch qui prend en paramètre l'url à interroger et les différentes options que l'on vient de définir.

`public/js/search.js`

```

const response = await fetch(url, options);
const results = await response.json();

```

## 2) Étape 3 : Traitement de la requête côté serveur en PHP

La méthode `searchAjax()` du contrôleur `ForumController` est chargée de récupérer les données provenant de la requête AJAX. Mais auparavant, il faut vérifier qu'il s'agit bien d'une requête asynchrone. Pour cela, j'ai créé une classe `Server` et une méthode `ajaxIsOkay()` qui redirige vers la page du formulaire de recherche si l'on s'est rendu sur la page de traitement via l'URL (de manière synchrone).

Une fois cette vérification terminée on récupère les données du formulaire dans la variable `$search` depuis la superglobale `$_POST` puis on va pouvoir appeler la méthode du modèle chargée d'aller récupérer le ou les articles recherchés dans la base de données.

`src/Controller/ForumController.php`

```

public function searchAjax()
{
    if(Server::ajaxIsOkay())
    {
        $search = trim(htmlspecialchars($_POST['search']));
        $articleModel = new ArticleModel();
        $searchArticles = $articleModel->searchArticle($search);
        [...]
    }
    else
    {
        return $this->redirect('search');
    }
}

```

## 3) Étape 4 : Récupération des données depuis ArticleModel et sa méthode searchArticle()

Une fois le traitement effectué sur le contrôleur au moment de la réception de la requête AJAX, on appelle le modèle `ArticleModel` et sa méthode `searchArticle()`. On lance la recherche dans la base de données grâce à la procédure stockée `SP_SearchSelect()` qui prend en paramètre le mot ou la chaîne de caractères recherchés. Puis on renvoie le ou les résultats dans le contrôleur.

`src/Model/ArticleModel.php`

```
public function searchArticle(string $searchArticle)
{
    $sql = 'CALL SP_SearchSelect(?)';
    $searchAnArticle = $this->database->getAllResults($sql, [$searchArticle]);
    return $searchAnArticle;
}
```

`Database/3 - Stored Procedures/SELECT/SP_SearchSelect.sql`

```
DELIMITER //
DROP PROCEDURE IF EXISTS SP_SearchSelect //
CREATE PROCEDURE SP_SearchSelect (v_search VARCHAR(512))
BEGIN

    SET v_search = CONCAT('%', v_search, '%');

    SELECT DISTINCT art.id AS id_article, art.title, DATE_FORMAT(art.creation_date, 'Le %d/%m/%Y à %H:%i') AS creation_date, art.user_id, u.pseudo, art.category_id, cat.id AS id_category, cat.category
    FROM articles art
    INNER JOIN categories cat ON art.category_id = cat.id
    INNER JOIN users u ON art.user_id = u.id
    WHERE (content LIKE v_search AND art.status_id = 2)
    OR (art.title LIKE v_search AND art.status_id = 2)
    OR (u.pseudo LIKE v_search AND art.status_id = 2)
    OR (cat.category LIKE v_search AND art.status_id = 2)
    ORDER BY art.creation_date DESC;

END //
```

Une fois que le contrôleur a récupéré le jeu de résultats de la recherche, on encode les données au format JSON (JavaScript Object Notation) et l'on fait un écho du ou des résultats que le JavaScript en AJAX va pouvoir récupérer.

`src/Controller/ForumController.php`

```
$searchArticlesAjax = json_encode($searchArticles);
echo $searchArticlesAjax;
```

#### 4) Étape 5 : Renvoi de la réponse HTTP avec les articles recherchés au format JSON

Une fois le traitement terminé côté PHP, on récupère le résultat de la requête AJAX sous la forme d'un objet promesse (Promise) provenant du serveur dans une constante `response` et l'on transforme le format de la réponse en JSON dans une constante `results`.

public/js/search.js

```
const response = await fetch(url, options);
const results = await response.json();
```

Je récupère dans une constante `container` la div HTML qui va contenir l'ensemble sous forme de liste des différents résultats récupérés dans le template.

Grâce à la propriété `innerHTML`, j'assigne initialement une chaîne de caractères vide de manière à vider la div si on répète une recherche et qu'on relance la méthode afin d'éviter des doublons.

public/js/search.js

```
const container = document.querySelector('.containerSearch');
container.innerHTML = '';
```

Puis j'effectue une boucle de récupération de tous les articles contenus dans la constante `results`. À chaque tour de boucle, je génère l'intégralité du HTML/CSS en créant les balises (grâce à la propriété `createElement`), et en ajoutant les classes CSS (grâce à la propriété `classList.add`). Chaque balise HTML créée reçoit le contenu des résultats de la recherche grâce à la clé associée à la valeur venant du tableau en JSON.

public/js/search.js

```
for (const result of results)
{
    let divSearchList = container.appendChild(document.createElement('div'));
    divSearchList.classList.add('Search-list');

    let divSearchListArticle = divSearchList.appendChild(document.createElement('div'));
    divSearchListArticle.classList.add('Search-list-article');

    let h3 = divSearchListArticle.appendChild(document.createElement('h3'));
    h3.classList.add('Search-list-article-title');
    h3.innerHTML = Tools.htmlspecialcharsJS(result['title']);
    [...]
}
```

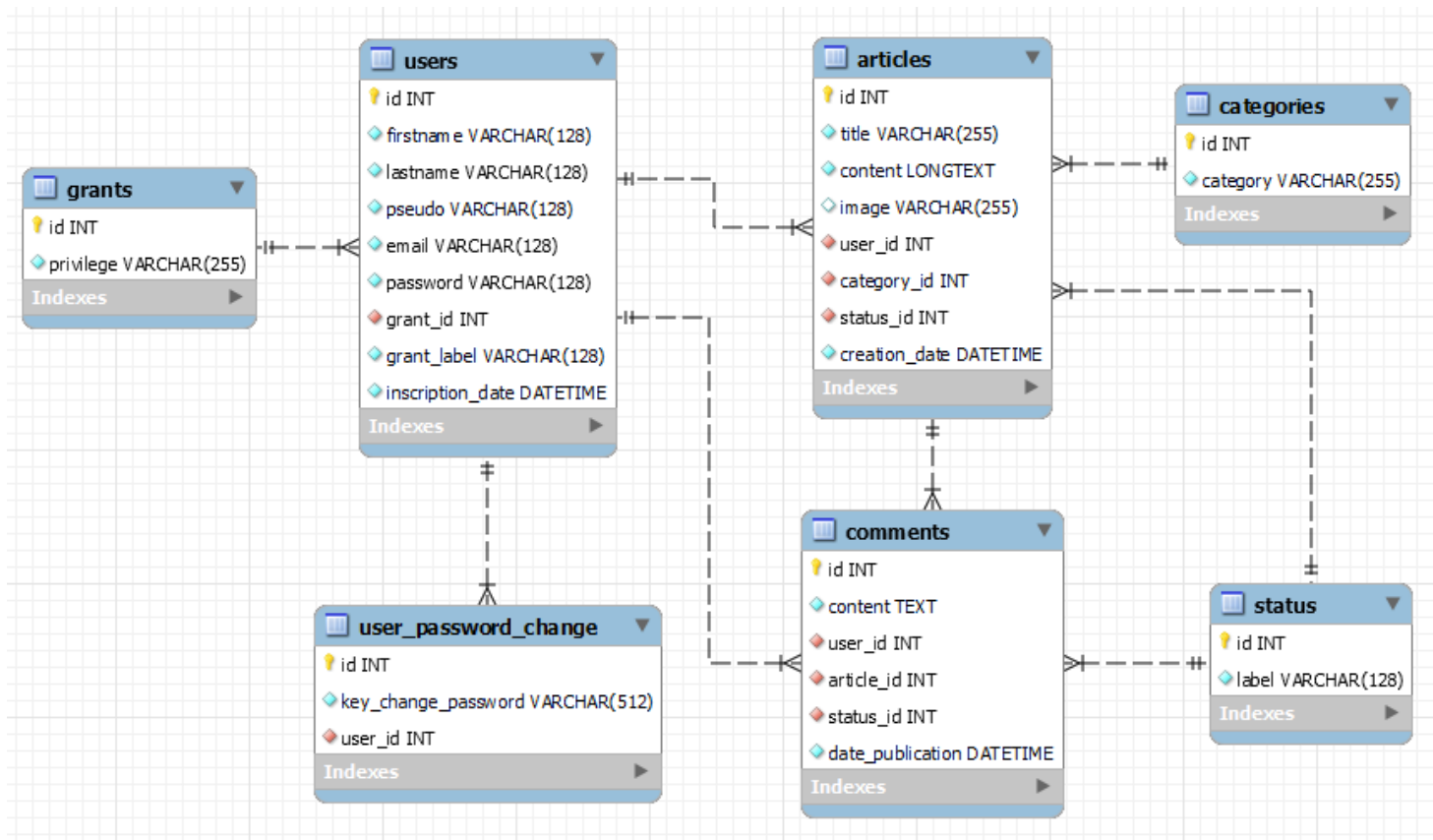
*Le résultat du moteur de recherche en AJAX est consultable en annexe (Annexe 4)*

## 5.2 Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

## 5.2.1 Créer une base de données

### a) Modèle de la base de données

Afin de réaliser ce projet, j'ai eu la nécessité de créer une base de données. Mais auparavant, il a fallu générer le modèle initial de celle-ci afin de connaître les différentes tables dont j'avais besoin, les différentes relations entre elles ainsi que les données à insérer. À la fin de ce travail, cela a donné le résultat suivant :



Dans ce modèle, je n'ai pas eu besoin d'utiliser de tables intermédiaires ayant pour cardinalité n - n mais presque l'ensemble des tables utilisent une cardinalité de type 1 - n.

Une fois le modèle de base de données établi, j'ai pu procéder à la création de la base de données en elle-même, en me servant du SGBDR majoritairement utilisé sur le web à savoir : MySQL.

### b) Création de la base de données et ajout du jeu de caractères

La commande "CREATE DATABASE" comme son nom l'indique, permet d'indiquer à MySQL que l'on souhaite créer une nouvelle base de données. La commande "CHARACTER SET" permet de spécifier l'encodage des

caractères écrits à savoir ici utf8mb4. Cet encodage permet d'inclure les lettres contenant les accents en particulier ceux de l'alphabet occidental ainsi que les nouveaux symboles (les emojis par exemple).

Enfin, la commande "COLLATE", ajoute une spécification supplémentaire en particulier l'ordonnancement et le regroupement des caractères au sein d'un jeu de caractères, permettant de répondre aux problèmes suivants :

- L'ordre dans lequel sont classés les caractères
- Considérer si un A majuscule et un a minuscule ont la même valeur
- La sensibilité aux accents
- Le classement en ordre alphabétique.

Ici, je me suis servi de l'"utf8mb4\_general\_ci", qui possède un interclassement simple : il supprime tous les accents, convertit toutes les lettres en minuscule et procède au tri. Par exemple, il considère les lettres ÄÅÃäã comme étant égales à la lettre A. De cette manière, cet interclassement est insensible à la casse.

Database/1 - Structure/bdd.sql

```
CREATE DATABASE huma_scientio
CHARACTER SET utf8mb4
COLLATE utf8mb4_general_ci;
```

Après la base de données, j'ai procédé à la création de l'utilisateur associé à celle-ci.

### c) Création de l'utilisateur dédié à la base de données et octroi des droits DDL et DML

Avec MySQL, la création d'un utilisateur se fait via la commande CREATE USER qui contient le nom de l'utilisateur ne devant comporter aucun caractère spécial ni accent, suivi du nom de l'hôte avec lequel il se connecte (ici le localhost) et enfin, l'identifiant, c'est-à-dire le mot de passe lui permettant d'accéder à la base de données avec les autorisations accréditées. Pour des raisons de sécurité, le mot de passe n'apparaît pas et il est remplacé ici par un "TODO".

Database/1 - Structure/user.sql

```
CREATE USER '4dm1n1str4teur'@'localhost' IDENTIFIED BY 'TODO';
```

Afin de s'assurer que cette commande a bien été exécutée, on se rend dans la base de données mysql (permettant le bon fonctionnement de toutes les bases de données) et l'on vérifie la présence de l'utilisateur via à la requête SQL suivante :

```
USE mysql;

SELECT User, Host
FROM User
```

```
WHERE User = '4dm1n1str4teur';
```

Le résultat de la requête a bien retourné l'utilisateur dans la base de données `mysql` ainsi que l'hôte correspondant.

	User	Host
▶	4dm1n1str4teur	localhost
*	NULL	NULL

Une fois l'utilisateur créé, je lui ai provisoirement donné tous les droits sur la base de données. Je lui donne tous les privilèges sur toutes ces tables. Je mets à jour et j'enregistre les nouveaux privilèges de mon utilisateur. Cela est provisoire, il ne faut surtout pas donner trop de droits à un utilisateur sur une base de données, il faut lui en donner le moins possible. Ces privilèges seront retirés en production.

Une fois les privilèges accordés à l'utilisateur, je procède aux DDL (Data Definition Language) permettant d'ajouter ou de supprimer une base de données, des utilisateurs, des tables des procédures stockées ou des droits.

En utilisant le script de génération de toutes les tables dont j'ai besoin et que j'ai défini dans le schéma, je crée les différentes tables, j'ajoute les procédures stockées et je réajuste soigneusement les privilèges de mon utilisateur dédié à cette base de données :

- Je lui retire tous les privilèges en persistant le retrait..
- Je lui donne les privilèges DML (Data Manipulation Language) qui sont les suivants : la possibilité de sélectionner, insérer, mettre à jour, supprimer et exécuter les procédures stockées sur toutes les tables de la base de données `huma_scientio`. En résumé, il s'agit du CRUD.
- Je mets à jour enfin les privilèges en persistant les nouveaux droits.

Database/1 - Structure/user.sql

```
REVOKE ALL PRIVILEGES
ON huma_scientio.*
FROM '4dm1n1str4teur'@'localhost';
FLUSH PRIVILEGES ;

GRANT SELECT, INSERT, UPDATE, DELETE, EXECUTE
ON huma_scientio.*
TO '4dm1n1str4teur'@'localhost';
FLUSH PRIVILEGES ;
```

Pour la phase DDL, j'ai créé par exemple la table `articles`.

d) Création des tables : exemple de la table `articles`

Je procède ensuite à la création des tables ainsi que de tous les champs dont j'ai besoin par rapport au schéma. Pour chaque colonne de chaque table, on précise en premier son nom, en second son type (numérique, texte, date, etc.), enfin si elle peut être de type NULL c'est-à-dire si elle peut ne pas recevoir de données. Je précise que la clé primaire est la colonne id qui s'auto incrémente à chaque nouvel enregistrement de ligne.

Je précise ensuite quelles sont les clés étrangères en ayant créé les tables référentes au préalable. Je crée trois contraintes pour deux colonnes étrangères, ici, les colonnes user\_id, categorie\_id et status\_id qui font respectivement référence aux tables users, categories, status et à leurs colonnes id ; enfin, je précise qu'en cas de mise à jour des tables étrangères ces mises à jour tombent en cascade dans la table articles.

Database/1 - Structure/**tables.sql**

```
DROP TABLE IF EXISTS articles;
CREATE TABLE articles
(
    id INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    content LONGTEXT NOT NULL,
    image VARCHAR(255) NULL,
    user_id INT NOT NULL,
    category_id INT NOT NULL,
    status_id INT NOT NULL,
    creation_date DATETIME NOT NULL,
    PRIMARY KEY (id),
    CONSTRAINT fk_user_article
    FOREIGN KEY (user_id)
        REFERENCES users (id) ON UPDATE CASCADE,
    CONSTRAINT fk_category_article
    FOREIGN KEY (category_id)
        REFERENCES categories (id) ON UPDATE CASCADE,
    CONSTRAINT fk_status_id_article
    FOREIGN KEY (status_id)
        REFERENCES status (id) ON UPDATE CASCADE
) ENGINE = InnoDB ;
```

## 5.2.2 Développer les composants d'accès aux données

### a) Création de l'objet PDO et des méthodes d'accès en lecture et en écriture dans la base de données

Après la base de données, j'ai eu besoin de créer les différents composants permettant d'accéder à celle-ci. J'ai créé un fichier PHP Database.php contenant les informations principales relatives à la connexion à la base de données. J'ai créé une méthode getPdoConnection() dans la classe Database permettant de créer une interface entre PHP et la base de données MySQL.

Dans ma variable `$dsn` (pour Data Source Name), j'ai indiqué les différents paramètres nécessaires à la connexion à la base de données. Comme j'en ai fait des constantes dans le fichier `config.php` et que ce fichier est appelé dans mon `index.php`, j'ai accès ici à informations.

Je concatène ces constantes aux différents paramètres de la variable `$dsn`. Je précise quel est le SGBDR (ici MySQL), l'hôte du serveur (`localhost`), le port employé (par défaut, MySQL utilise le 3306), le nom de la base de données à laquelle on souhaite accéder (`huma_scientio`) et le jeu de caractères utilisé à l'affichage des données (`utf8`).

Nous n'avons fait ici que donner des paramètres pour pouvoir nous y connecter. La connexion à celle-ci se fait dans la variable `$pdo`. PDO pour PHP Data Object est un objet qui nous permet de nous interfacer à notre base de données si tous les paramètres requis ont été renseignés et bien sûr s'ils sont tous valides. L'objet PDO reçoit en paramètre la variable `$dsn`, et prend des paramètres supplémentaires :

- Le nom de l'utilisateur de la base de données.
- Le mot de passe de celle-ci.

La méthode `getPdoConnection()` terminée, il ne reste plus qu'à l'appeler à chaque fois que l'on aura besoin d'accéder à la base de données et réaliser des opérations DML à l'intérieur. De cette manière, l'accès à la base de données est encapsulé.

`src/Framework/Database.php`

```
class Database
{
    function getPdoConnection(): PDO
    {
        $dsn = ''.DB_MS.':host='.DB_HOST.':port='.DB_PORT.':dbname='.DB_NAME.':charset='.DB_CHARSET.'';

        $pdo = new PDO($dsn, DB_USER, DB_PASSWORD, [
            PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
            PDO::ATTR_EMULATE_PREPARES => false
        ]);

        return $pdo;
    }
}
```

Pour réaliser des requêtes SQL sécurisées dans un environnement PHP, j'ai eu besoin de créer d'autres méthodes. La méthode `executeQuery()` prend en paramètre une chaîne de caractères (la requête SQL à effectuer) ainsi qu'un tableau de paramètres à transmettre s'il y en a et retourne un objet `PDOStatement` dans la variable `$pdoStatement`.

Les méthodes `prepare()` et `execute()` de l'objet PDO permettent d'éviter les injections SQL, une grave faille de sécurité (nous en parlerons plus loin). Nous préparons la requête dans un premier temps, mais à la



place d'écrire directement les variables transmises nous remplaçons chaque donnée à entrer par un placeholder anonyme (un point d'interrogation) et nous donnons les véritables données en tableau de paramètres à la méthode `execute()`.

src/Framework/Database.php

```
function executeQuery(string $sql, array $params = []): PDOStatement
{
    $pdoStatement = $this->pdo->prepare($sql);
    $pdoStatement->execute($params);
    return $pdoStatement;
}
```

La méthode `executeQuery()` effectue une requête SQL, c'est-à-dire qu'une opération d'exécution s'est déroulée dans la base de données, mais nous n'avons pas récupéré de résultats. C'est le rôle des méthodes suivantes que j'ai créé `getOneResult()` et `getAllResults()` qui sont chargées de remonter un résultat (via la méthode `fetch()`) ou plusieurs résultats (via la méthode `fetchAll()`) de l'objet `PDOStatement` en utilisant la méthode `executeQuery()` au préalable. Puis on retourne les résultats dans `$result` ou `$results`.

src/Framework/Database.php

```
function getAllResults(string $sql, array $params = [])
{
    $pdoStatement = $this->executeQuery($sql, $params);
    $results = $pdoStatement->fetchAll();
    return $results;
}
```

Désormais, que la connexion à la base de données, l'exécution de requêtes SQL et la récupération d'un ou de plusieurs résultats sont réalisées, nous pouvons renvoyer les résultats dans un fichier PHP à l'utilisateur.

## b) Le paradigme de programmation MVC

### 1) Explications générales

Afin de pouvoir accéder aux données, je me suis servi du modèle MVC (Model View Controller). Il s'agit d'un paradigme de programmation et de développement séparant les différentes couches d'une application. Le Modèle contient l'accès en lecture ou en écriture des données (qui doivent être affichées ou envoyées dans une base de données), la Vue le rendu final côté client et le Controller permet de gérer les différents échanges et interactions entre les deux premiers, ou encore la logique des diverses actions réalisées par l'utilisateur (la soumission et la vérification d'un formulaire par exemple).

## 2) Exemple avec la page forum

Afin de représenter le mécanisme de fonctionnement du système MVC, je vais expliquer comment on accède à la page de forum des articles.

Ce paradigme de programmation (employé ici dans le développement web) repose sur un couple de type route / action. En d'autres termes, que doit-il se produire si telle URL - ou route - est adressée à l'application web ?

J'ai en premier lieu défini un tableau associatif contenant le nom de la route, le path, le controller et la méthode associés. En d'autres termes, si le path dans l'adresse URL correspond à /forum, alors on va récupérer le controller Forum et la méthode index de celui-ci.

app/routes.php

```
$routes = [  
    'forum' => [  
        'path' => '/forum',  
        'controller' => 'Forum',  
        'method' => 'index'  
    ],  
];
```

Il faut également prévoir le cas où l'on tape une adresse qui ne correspond à aucune route du tableau et afficher une redirection 404.

app/routes.php

```
'404' => [  
    'path' => $path,  
    'controller' => 'Home',  
    'method' => 'notFound'  
]
```

Dans ce cas précis, le path correspond ici à une variable, c'est-à-dire n'importe quelle entrée de l'utilisateur dans la barre d'adresse qui ne correspond pas à une route du tableau de routes. Elle est placée en dernier si aucune des autres auparavant n'est reconnue.

Une fois que nous avons trouvé notre route, nous allons chercher le contrôleur et la méthode correspondante dans la classe qui les contiennent. Ici, il s'agit de `ForumController` et de la méthode `index`.

src/Controller/ForumController.php

```
class ForumController extends AbstractController  
{
```

```

public function index()
{
    $articlesModel = new ArticleModel();

    $articles = $articlesModel->getAllArticles();

    $pageTitle = 'Forum';

    return $this->render('forum', [
        'articles' => $articles??'',
        'pageTitle' => $pageTitle??''
    ]);
}
}

```

Cette méthode `index` instancie un objet, le modèle `ArticleModel()` chargé d'appeler la méthode `getAllArticles()` permettant la récupération dans la base de données de l'ensemble des articles. La méthode `getAllArticles()` ne prend aucun paramètre. Je crée une variable `$sql` à laquelle j'assigne la requête SQL permettant la récupération de l'ensemble des articles, c'est une procédure stockée MySQL (voir plus loin). Puis je crée une variable `$articles` qui va récupérer les résultats et j'assigne comme paramètre à la méthode `getAllResults()` de l'objet `Database` la variable `$sql`. Enfin, je retourne le traitement de la requête à partir de la méthode.

src/Framework/AbstractModel.php

```

class ArticleModel extends AbstractModel
{
    public function getAllArticles()
    {
        $sql = 'CALL SP_AllArticlesOrderByDateSelect ()';
        $articles = $this->database->getAllResults($sql);
        return $articles;
    }
}

```

Database/3 - Stored Procedures/SELECT/SP\_AllArticlesOrderByDateSelect.sql

```

DELIMITER //
DROP PROCEDURE IF EXISTS SP_AllArticlesOrderByDateSelect //
CREATE PROCEDURE SP_AllArticlesOrderByDateSelect ()
BEGIN

    SELECT art.id, art.title, art.content, art.user_id, art.category_id, DATE_FORMAT(art.creation_date, 'Le
%d/%m/%Y à %H:%i:%s') AS creation_date, u.pseudo, cat.category
    FROM articles art
    INNER JOIN users u ON art.user_id = u.id
    INNER JOIN categories cat ON art.category_id = cat.id
    AND art.status_id = 2

    ORDER BY art.creation_date DESC;

```

END //

Une fois récupérées les données dans `$articles` dans la méthode `index()` du `ForumController`, il ne reste plus qu'à s'occuper du rendu côté client grâce à la méthode `render`. Elle prend en paramètre le nom du template à afficher à l'utilisateur ainsi qu'un tableau associatif de données à insérer dans le template dont le jeu de résultats remontés depuis la base de données.

Dans le template, on vérifie si la variable `articles` n'est pas vide avec la fonction `empty()` de PHP, on effectue une boucle `foreach`. À chaque tour de boucle, on affiche la valeur associée à une clé. On applique aux chaînes de caractères la fonction `htmlspecialchars()` de PHP permettant de nous prémunir contre la faille XSS.

templates/forum.phtml

```
<div class="Forum-list">

    <?php if (!empty($articles)):?>

        <?php foreach ($articles as $article):?>

            <div class="Forum-list-article">
                <h3 class="Forum-list-article-title"><?=htmlspecialchars($article['title']);?></h3>
                <p class="Forum-list-article-pseudo">Écrit par : <?=
htmlspecialchars($article['pseudo']);?></p>
                <p class="Forum-list-article-date">Date de publication : <?= $article['creation_date'];?></p>
                <p class="Forum-list-article-category">Catégorie : <a class="Forum-list-article-category-link"
href="<?= buildUrl('category', ['id'=> intval($article['category_id'])];?>"><?=
htmlspecialchars($article['category']);?></a></p>
                <a class="Forum-list-article-link" href="<?= buildUrl('article', ['id' =>
intval($article['id'])];?>">Lire la suite</a>
            </div>

        <?php endforeach;?>

    <?php else:?>

        <p>Il n'y a encore aucun article.</p>

    <?php endif;?>

</div>
```

Une fois que nous avons notre contrôleur, notre modèle et notre vue associés, il nous faut afficher le contenu à l'utilisateur.

Sur `index.php`, on récupère dans la variable `$path` l'URL de la page courante grâce à la méthode statique `Server::path()`, on inclut le tableau de routes, on instancie un objet `Router` et on appelle la méthode `match()` chargée de vérifier dans le tableau associatif de routes si le path correspond à une méthode et à un

contrôleur. S'il correspond à celui trouvé dans le tableau associatif, on renvoie le contrôleur et la méthode dans index.php.

src/Framework/Router.php

```
class Router
{
    private $routes;
    public function __construct(array $routes)
    {
        $this->routes = $routes;
    }
    public function match(string $path)
    {
        foreach ($this->routes as $route)
        {
            if ($path == $route['path'])
            {
                return [
                    'controller' => $route['controller'],
                    'method' => $route['method']
                ];
            }
        }
        return false;
    }
}
```

Une fois le contrôleur et la méthode associés renvoyés dans index.php qui sert de front controller, on peut récupérer la classe correspondante, instancier celle-ci, appeler la méthode et faire un écho du résultat.

public/index.php

```
require '../vendor/autoload.php';
require '../config.php';
require '../library/functions.php';

use App\Framework\Router;
use App\Framework\Server;

$path = Server::path();

$routes = include '../app/routes.php';

$router = new Router($routes);
$action = $router->match($path);

$class_name = 'App\\Controller\\' . $action['controller'] . 'Controller';
$controller = new $class_name();
$method = $action['method'];

echo $controller->$method();
```

### 5.2.3 Développer la partie back-end d'une application web ou web mobile

Le projet Huma Scientio propose un certain nombre d'interfaces afin d'interagir avec les données de l'application : les utilisateurs peuvent créer un compte, consulter, écrire, modifier ou supprimer des articles, ou encore les administrateurs de leur côté peuvent gérer les commentaires, les catégories ainsi que les membres inscrits. Il s'agit ici de l'ensemble des opérations classiques du CRUD (*Create, Read, Update Delete*).

Je vais présenter les trois principales fonctionnalités du back-office :

- Le tableau d'administration des articles ;
- Le formulaire de création d'articles ;
- Le formulaire d'inscription ;

#### a) Le tableau d'administration des articles

Afin que chaque membre puisse lui-même gérer l'ensemble de ses articles, il a été nécessaire de créer un tableau d'administration.

Pour ce faire, j'ai créé une méthode `myArticles()` dans le contrôleur `ArticleController.php`. Je vérifie grâce à la classe `UserSession` et à deux méthodes statiques `author()` et `administrator()` si la personne connectée peut accéder à cette page, si ce n'est pas le cas, je la redirige vers une page 403. Je récupère l'identifiant en session de l'utilisateur avec la méthode statique `UserSession::getId()`, j'instancie un objet `ArticleModel`, j'appelle une méthode `getMyArticles()` et je donne en paramètre à cette méthode le numéro d'identifiant permettant de retrouver les articles de l'utilisateur connecté.

`src/Model/ArticleModel.php`

```
public function getMyArticles(int $userId)
{
    $sql = 'CALL SP_MyArticlesSelect(?)';
    $getMyArticles = $this->database->getAllResults($sql, [$userId]);
    return $getMyArticles;
}
```

Ce paramètre de la méthode `getMyArticles()` est transmis ensuite à la procédure stockée `SP_MyArticlesSelect()` afin de remonter les articles dont l'auteur est la personne connectée. Pour des raisons de sécurité, j'ai typé le paramètre de la procédure stockée.

Je sélectionne l'id, le titre, l'identifiant l'id de l'utilisateur, l'id de la catégorie, la date sur laquelle j'applique la fonction SQL `DATE_FORMAT` permettant de formater en français l'affichage de la date, l'id et le label de la

catégorie, le nombre de commentaires grâce à la fonction COUNT et le statut de l'article. Je joins la table categories par l'identifiant category\_id comme clé étrangère depuis la table articles, la table users par la clé étrangère user\_id et enfin la table status par la clé étrangère status\_id. Je regroupe les articles par id et je récupère les résultats dans l'ordre descendant de publication (du plus récent au plus ancien) en prenant soin d'indiquer que je récupère les articles dont l'auteur est celui transmis en paramètre dans la clause WHERE.

Database/3 - Stored Procedures/SELECT/SP\_MyArticlesSelect.sql

```
CREATE PROCEDURE SP_MyArticlesSelect (v_id INT)
BEGIN

    SELECT art.id AS articleId, art.title, art.user_id, art.category_id, DATE_FORMAT(art.creation_date,
'%d/%m/%Y') AS creation_date, cat.id, cat.category, COUNT(com.article_id) AS number_comments, stat.label
    FROM articles art
    INNER JOIN categories cat ON art.category_id = cat.id
    INNER JOIN users u ON art.user_id = u.id
    INNER JOIN status stat ON art.status_id = stat.id
    LEFT JOIN comments com ON art.id = com.article_id
    WHERE u.id = v_id
    GROUP BY art.id
    ORDER BY art.creation_date DESC;

END //
```

Enfin, une fois le traitement effectué, je renvoie les données au contrôleur ArticleController et à sa méthode myArticles.

src/Controller/Admin/ArticleController.php

```
public function myArticles()
{
    if (UserSession::author() || UserSession::administrator())
    {
        $id_user = UserSession::getId();
        $articleModel = new ArticleModel();
        $myArticles = $articleModel->getMyarticles($id_user);

        $pageTitle = 'Mes articles';
    }
    else
    {
        $this->redirect('accessRefused');
    }
    return $this->render('admin/article/myarticles', [
        'myArticles' => $myArticles??'',
        'pageTitle' => $pageTitle??''
    ]);
}
```

Sur le template `myarticles.phtml`, j'effectue une boucle `foreach` de récupération de l'ensemble des articles de l'utilisateur connecté transmis depuis le contrôleur avec l'id, le titre permettant d'accéder à l'article publié, la date de création, la catégorie, le nombre de commentaires, le statut et pour chaque article la possibilité soit de le modifier, soit de le supprimer. Ces deux dernières options n'étant possibles qu'au moment de la soumission du formulaire de modification ou lors du clic sur le lien de suppression. Dans les deux cas, le token (chaîne de caractères générée lors de la connexion) doit être identique à celui récupéré comme champ caché dans le formulaire de modification et dans l'URL pour le lien de suppression. C'est une protection contre la faille CSRF.

Pour chaque valeur comme le titre ou encore la catégorie, j'applique une fonction PHP `htmlspecialchars()`. C'est une protection contre la faille XSS qui empêche le navigateur d'interpréter des balises HTML ou JavaScript de manière malveillante.

## b) Le formulaire de création d'articles

L'ensemble du projet Huma Scientio repose sur la diffusion de contenus pédagogiques ayant pour thématique les sciences humaines. C'est pour cela que le formulaire de création d'articles est probablement l'un des plus importants.

Pour cela, j'ai créé un template spécifique et ajouté l'ensemble des balises HTML du formulaire d'ajout d'articles. Je me suis servi de l'éditeur de texte TinyMCE afin de permettre une meilleure mise en forme.

Le contrôleur associé à la page de création d'articles se compose de deux parties :

- L'affichage de l'ensemble du formulaire lorsque l'on arrive sur la page la première fois.
- Le traitement du formulaire une fois soumis.

La première vérification consiste à savoir si l'on est autorisé à accéder à la page de création d'articles : c'est-à-dire si l'on est auteur ou administrateur, sinon on est redirigé vers une page 403. Puis on récupère l'ensemble de toutes les catégories que l'utilisateur aura à choisir à la création de l'article et le nom de la page.

Si le formulaire est soumis, on vérifie que les différents champs ne sont pas vides (le titre, le contenu, la catégorie, l'identifiant de l'utilisateur en train de créer son article, si l'on a ajouté ou non une image et pour finir, le token de sécurité par rapport à la faille CSRF (voir sécurité). En effet, la chaîne de caractères aléatoire du token de sécurité de la personne connectée doit être la même que celle soumise dans le formulaire et le but est de vérifier si les deux sont identiques.

`src/Controller/Admin/ArticleController.php`

```
if (UserSession::author() || UserSession::administrator())
```



```

{
    $categoryModel = new CategoryModel();
    $categories = $categoryModel->getAllCategoriesForArticle();
    $pageTitle = 'Créer un nouvel article';

    if(!empty($_POST))
    {
        $title = trim($_POST['title']);
        $content = str_contains($_POST['content'], '<script>') ? trim(htmlspecialchars($_POST['content'])) :
trim($_POST['content']);
        $category = (int) $_POST['categories'];
        $file = $_FILES['image'];

        if(!$title || !$content || !$category)
        {
            FlashBag::addFlash("Tous les champs n'ont pas été correctement remplis.", 'error');
        }
        if(!empty($file['name']))
        {
            $fileModel = new File();
            $fileName = $fileModel->UploadFileImage($file);
        }
        [...]
    }
}

```

La dernière vérification consiste à savoir si l'on a téléchargé une image et si elle est conforme. C'est le rôle de la classe File.

La classe File possède deux méthodes. La première `createFolderImage()` consiste à vérifier si le dossier chargé de contenir les images d'illustration des articles existe ou non, si ce n'est pas le cas alors on le crée. Cette méthode est appelée lors de l'instanciation de la classe File.

La seconde méthode `uploadFileImage()` va traiter l'image fournie dans le formulaire d'ajout d'article :

- On vérifie qu'il n'y a pas d'erreurs lors de l'ajout du fichier et que la taille de l'image n'excède pas les 2 Mo.
- On vérifie que l'extension du fichier fait partie de celles autorisées et correspond à des images.

Dans tous les cas, s'il y a une erreur, alors la méthode renverra null.

En revanche s'il n'y a aucune erreur on génère un nom aléatoire pour l'image afin que deux images ne portent pas le même nom dans le dossier qui va les contenir, on concatène au nom du fichier original la chaîne aléatoire ainsi que son extension et l'on renvoie le nom de l'image qui sera ensuite entré en base de données dans la table articles.

`src/Framework/File.php`

```

class File
{
    public function __construct()
    {
        $this->createFolder = $this->createFolderImage();
    }

    public function uploadFileImage(array $file, string $imageExist = null)
    {
        [...]
        $fileName = pathinfo($file['name']);
        $fileExtension = $fileName['extension'];

        $validExtension = ['img', 'png', 'jpg', 'jpeg'];

        if(!in_array($fileExtension, $validExtension))
        {
            FlashBag::addFlash("L'extension du fichier n'est pas valide.", 'error');
            return null;
        }
        [...]
    }

    function createFolderImage()
    {
        if (!file_exists(PROJECT_DIR . '/public/img/imgArticle'))
        {
            mkdir(PROJECT_DIR . '/public/img/imgArticle', 0777, true);
        }
    }
}

```

Une fois toutes les vérifications terminées et sans erreur, on peut procéder à l'insertion de l'article. On instancie l'objet `ArticleModel` et on appelle la méthode `insertArticle()`. Elle prend en paramètre les données du formulaire et on donne par défaut la valeur null à `$image` car il est possible qu'un article n'ait pas d'illustration.

`src/Model/ArticleModel.php`

```

public function insertArticle(string $title, string $content, int $category_id, int $user_id, string $image = null)
{
    $sql = 'CALL SP_ArticleInsert(?, ?, ?, ?, ?)';
    $insertArticleId = $this->database->insert($sql, [$title, $content, $category_id, $user_id, $image]);
    return $insertArticleId;
}

```

Enfin, on appelle la procédure stockée `SP_ArticleInsert()` qui va enregistrer le nouvel article en base de données. Pour l'image, on va se servir de la structure de contrôle SQL CASE. Si la valeur de la variable

`v_image` est null, alors on enregistre null dans la colonne image du nouvel article. Sinon, on enregistre le nom de l'image traitée dans le contrôleur et la classe File précédemment.

Database/3 - Stored Procedures/INSERT/SP\_ArticleInsert.sql

```
DELIMITER //
DROP PROCEDURE IF EXISTS SP_ArticleInsert //
CREATE PROCEDURE SP_ArticleInsert (v_title VARCHAR(255), v_content TEXT, v_category_id INT, v_user_id INT,
v_image VARCHAR(255))
BEGIN

    DECLARE v_status SMALLINT;
    SET v_status = 1;

    INSERT INTO articles (title, content, category_id, user_id, status_id, creation_date, image)
    VALUES (v_title, v_content, v_category_id, v_user_id, v_status, NOW(),
        (
            CASE
                WHEN v_image IS NULL THEN NULL
                ELSE v_image
            END
        )
    );

END //
```

### c) Le formulaire d'inscription

Huma Scientio étant une application web chargée de permettre la publication de contenu, il a été nécessaire de créer un certain nombre de formulaires afin d'interagir avec les utilisateurs. Le formulaire principal accessible depuis l'extérieur de l'application et de manière publique est celui d'inscription.

Le contrôleur associé à la page d'inscription se compose de deux parties. La première est l'affichage de l'ensemble du formulaire lorsque l'on arrive dessus pour la première fois, la seconde, le traitement des données une fois que celui-ci a été soumis avec l'affichage du succès ou d'erreurs éventuelles à l'inscription. Compte tenu du fait qu'il s'agit d'un formulaire potentiellement exposé à des attaques de robots provenant de l'extérieur, il a été nécessaire d'installer le protocole de sécurité de Google à savoir, le reCAPTCHA.

Pour cela, il faut placer le script du reCAPTCHA de Google tout en haut du template de page d'inscription en y incluant une clé aléatoire à la création de celui-ci. On ajoute un champ caché dans le formulaire nommé `recaptcha-response` qui sera ensuite envoyé et analysé par Google au moment du traitement du formulaire. Côté PHP, on s'assure que tous les champs ont bien été remplis et ne sont pas vides. D'autre part, je vérifie que les champs remplis ne contiennent pas de caractères susceptibles d'être interprétés par le navigateur. Pour ce faire, j'ai appliqué les fonctions `trim()` et `htmlspecialchars()`. Je vérifie que le mot de passe contient des majuscules, minuscules, des chiffres et un caractère spécial grâce à la fonction `preg_match()`. Je vérifie que le mot de passe contient au moins 8 caractères aussi.

Une fois que l'on a vérifié que le format d'email était correct grâce à la fonction `filter_var()`, on lance une vérification en web API vers Google en donnant dans l'URL la clé secrète fournie par Google ainsi que la valeur du champ caché du recaptcha-response placée dans le formulaire.

Si Google répond par un message de succès, alors on peut poursuivre l'inscription du nouvel utilisateur.

#### src/Controller/AccountController.php

```
$url = "https://www.google.com/recaptcha/api/siteverify?secret=SECRET&response={$recaptchaResponse}";
$response = file_get_contents($url);
if(empty($response) || is_null($response))
{
    FlashBag::addFlash('Erreur à l\'inscription', 'error');
}

$data = json_decode($response);

if(!$data->success)
{
    FlashBag::addFlash('Erreur à l\'inscription', 'error');
}
```

La procédure stockée `SP_inscriptionInsert()` est appelée dans le `UserModel()` une fois les vérifications faites dans le contrôleur (côté Google et côté PHP). Elle prend en paramètre les données des champs du formulaire. Elle procède à un comptage dans la table `users` afin de vérifier si la personne qui souhaite s'inscrire possède le même pseudo ou email qu'un autre membre déjà existant. Si dans l'un, l'autre, ou les deux, le comptage remonte une ligne, cela signifie que c'est le cas, on renvoie alors un message d'erreur à l'utilisateur l'invitant à changer ses futurs identifiants. Si les comptages remontent zéro, alors on peut enregistrer les données dans la table en lui donnant son rôle de nouvel utilisateur.

#### Database/3 - Stored Procedures/INSERT/SP\_InscriptionInsert.sql

```
DELIMITER //
DROP PROCEDURE IF EXISTS SP_InscriptionInsert //
CREATE PROCEDURE SP_InscriptionInsert (v_firstname VARCHAR(255), v_lastname VARCHAR(255), v_pseudo
VARCHAR(255), v_email VARCHAR(255), v_password VARCHAR(255))
BEGIN

    DECLARE message VARCHAR(512);
    DECLARE existEmail SMALLINT;
    DECLARE v_grantId SMALLINT;
    DECLARE existPseudo SMALLINT;
    DECLARE v_grant_label VARCHAR(255);

    SET v_grantId = 3;
    SET v_grant_label = "['ROLE_NEW_USER']";

    SELECT COUNT(id)
    INTO existEmail
```

```

FROM users
WHERE email = LOWER(v_email);

IF (existEmail > 0) THEN

    SET message = "Un autre utilisateur avec cet email existe déjà.";

END IF;

[...]

IF (existEmail = 0 AND existPseudo = 0) THEN

    INSERT INTO users (firstname, lastname, pseudo, email, password, grant_id, grant_label,
inscription_date)
    VALUES
    (v_firstname, v_lastname, v_pseudo, v_email, v_password, v_grantId, v_grant_label, NOW());

    SET message = "Vous êtes bien enregistré, vous pouvez vous connecter.";

END IF;

SELECT message;

END //
```

*La totalité du traitement et des vérifications du formulaire d'inscription sont consultables en annexe (Annexe 5)*

## 6. Présentation du jeu d'essai

### Jeu d'essai back-end

Pour le jeu d'essai back-end, j'ai décidé de me servir du formulaire de connexion. Il se décompose en deux temps :

1. Un utilisateur ne peut pas se connecter s'il entre de mauvais identifiants
2. L'utilisateur peut être mis en session s'il entre les bons identifiants

Scénario -> résultat attendu	Succès du test (Réussi ou Échec)
Je tente de me connecter avec un email ou un mot de passe invalide... -> Le serveur renvoie un message d'erreur indiquant "identifiants incorrects"	Réussi

Je tente de me connecter avec un email et un mot de passe valide ... -> <b>Le serveur renvoie un message de succès "Connexion réussie, bienvenue!"</b>	Réussi
---	--------

## 7. Veille de sécurité

Au cours de notre formation, il nous est apparu clairement que la sécurité faisait partie des éléments essentiels sur lesquels devait porter l'attention d'un développeur. Pendant notre apprentissage ainsi que durant les stages effectués en entreprise de même qu'au cours de recherches et de documentation personnelle, plusieurs axes se sont dessinés :

1. La sécurité d'un site repose sur un ensemble de bonnes pratiques à mettre en place d'entrée de jeu et dans chaque domaine de la conception.
2. **Il ne faut en aucun cas faire confiance aux données provenant de l'utilisateur.**

Le second point est extrêmement important parce que cela signifie que toute donnée, n'importe laquelle provenant d'un utilisateur côté client doit être contrôlée, vérifiée, clarifiée et épurée avant toute utilisation ensuite sur le serveur.

Au cours de mes recherches concernant la sécurité des sites web, je suis tombé sur trois graves failles de sécurité à savoir :

- L'injection SQL ;
- La faille XSS ;
- La faille CSRF ;

### 7.1 L'injection SQL

#### 7.1.1 Définition et explication

En voulant me renseigner sur la faille d'injection SQL j'ai tapé dans le moteur de recherche les mots suivants :

protection contre injection sql

Je suis tombé sur le site suivant :

Source : <https://www.leblogduhacker.fr/se-proteger-de-l-injection-sql/>

J'ai alors compris que la faille injection SQL consistait pour un utilisateur malveillant d'exploiter un des champs du formulaire (par exemple un champ de type texte) d'injecter du code SQL et de détourner la

requête SQL de sa vocation initialement prévue par le développeur. Une attaque de ce genre peut provoquer et obtenir les résultats suivants :

- Usurpation de compte.
- Création de compte avec des privilèges administrateur.
- Accéder à toute la base de données ou à une partie de celle-ci.
- Modifier voire supprimer une partie ou la totalité des données provenant du serveur.

La plupart des frameworks PHP se munissent contre les injections SQL (Symfony, Laravel). Cependant, dans le cas d'un projet monté à partir de zéro, la menace d'injection SQL est bien réelle.

Prenons un exemple afin d'illustrer cela.

Supposons qu'un script PHP possède la requête SQL suivante afin d'authentifier un utilisateur :

```
SELECT *  
FROM users  
WHERE email = 'email@email.com' AND password = '*****';
```

En temps normal, l'authentification pourra se faire, il suffit à l'utilisateur d'entrer ses identifiants et il se connectera.

Mais à présent, supposons qu'un utilisateur malveillant entre dans le champ email ceci :

email@email.com ; --

La requête transmise va alors se transformer ainsi :

```
SELECT *  
FROM users  
WHERE email = 'email@email.com'; -- AND password = '*****';
```

Les caractères -- débutent un commentaire de code SQL. Par conséquent, la requête SQL est transformée et donne ceci :

```
SELECT *  
FROM users  
WHERE email = 'email@email.com';
```

De cette manière, le hacker peut se connecter avec l'email email@email.com et n'importe quel mot de passe. C'est une attaque par injection SQL réussie. Mais il peut y avoir bien pire.

Supposons que dans un formulaire il faille entrer un numéro d'identifiant ce qui donne côté PHP la requête SQL suivante :

```
SELECT *  
FROM users  
WHERE id = "numéro";
```

Mais si dans le formulaire, on entre les données suivantes :

```
3; DROP TABLE comments;
```

Alors, le traitement de la requête donnera ceci :

```
SELECT *
FROM users
WHERE id = "numéro"; DROP TABLE comments;
```

Cette attaque par injection SQL provoquera la suppression de la table comments et de toutes les données qui y sont stockées si les droits DDL accordés à l'utilisateur le permettent.

### 7.1.2 Solutions

Afin de se prémunir contre ce genre d'attaques, j'ai découvert au cours de mes recherches plusieurs solutions :

Tout d'abord, il faut s'assurer que toutes les données transmises par l'utilisateur au sein d'un formulaire par exemple ne puissent en aucun cas changer la nature de la requête SQL.

Il est nécessaire ensuite d'**échapper les caractères**, par exemple, pour les apostrophes avec un antislash..

Une autre manière de s'en prémunir consiste à effectuer des **requêtes préparées**. Pour le projet Huma Scientio, j'ai créé des méthodes au sein de la classe Database afin de préparer et d'exécuter les requêtes SQL de manière sécurisée. En d'autres termes, on écrit la requête SQL d'un côté et l'on met à la place des données à entrer des placeholders anonymes puis l'on transmet les paramètres qui viendront compléter les données en attente de la requête préparée.

Par exemple, on prépare une requête d'insertion dans une table users avec un nom, une adresse et une ville, les données à entrer sont dans VALUES qui possèdent des placeholders anonymes. Les vraies données sont dans la méthode bindParam() qui prend en premier paramètre le placeholder anonyme de VALUES et en second la donnée récupérée en variable depuis un formulaire.

```
$stmt = $dbh->prepare("INSERT INTO users (CustomerName, Address, City)
VALUES (:nam, :add, :cit)");
$stmt->bindParam(':nam', $txtNam);
$stmt->bindParam(':add', $txtAdd);
$stmt->bindParam(':cit', $txtCit);
$stmt->execute();
```

Une autre méthode consiste à se servir de **procédures stockées**. Une procédure stockée est une série d'instructions à exécuter en SQL que l'on déclenche lorsqu'on l'appelle. Sur tout le projet Huma Scientio, je me suis servi de procédures stockées. Il est recommandé de s'en servir pour plusieurs raisons :



- Les performances : lorsqu'on appelle une procédure stockée, le serveur exécute celle-ci et la garde en mémoire sous forme d'exécutable. On évite ainsi d'écrire la requête dans le PHP ce qui va prendre du temps lors de la compilation côté serveur. La procédure stockée est gardée en mémoire sur le serveur qui peut l'exécuter à nouveau plus rapidement lors d'un nouvel appel.
- Errare humanum est : une fois qu'une requête SQL est rédigée selon les besoins, il suffit de l'encapsuler dans une procédure stockée. La requête aura ainsi beaucoup moins de chance de subir une erreur humaine de manipulation.
- **La sécurité** : une procédure stockée est une boîte noire. Elle rend invisibles la structure des tables et l'architecture de la base de données. Elle rend impossible les injections SQL. Pour le projet Huma Scientio, l'utilisateur disposant des droits DML sur la base de données ne peut rien faire d'autre qu'exécuter les procédures stockées. De cette manière aucune requête du CRUD n'est écrite en dur et aucune manipulation des données ne peut se faire en dehors des procédures stockées.

Reprenons l'exemple précédent :

```
DELIMITER //
DROP PROCEDURE IF EXISTS SP_Insert //
CREATE PROCEDURE SP_Insert(v_nom VARCHAR(255), v_adress VARCHAR(255), v_city VARCHAR(255))
BEGIN

    INSERT INTO users (name, adress, city)
    VALUES
    (v_nom, v_adress, v_city);

END //
```

Depuis le PHP, il n'y a plus qu'à faire appel à la procédure stockée. En MySQL, le mot clé est "CALL" :

```
"CALL SP_Insert(?, ?, ?);"
```

Ici, les placeholders anonymes sont des points d'interrogation. Il n'y a plus qu'à les remplacer par les vrais paramètres à transmettre ensuite dans une requête préparée. De cette manière, toute tentative d'injection SQL est impossible. Le grand avantage ici est que l'on maîtrise du début à la fin le traitement de toutes les données.

En règle générale, afin de se prémunir contre ce type de failles, il est vivement recommandé de donner à l'utilisateur ayant accès à la base de données le moins de privilèges possible : retirer au maximum les droits DDL et les droits DML.

## 7.2 La faille CSRF

Durant mes recherches sur le site précédent, j'ai découvert l'existence d'une autre faille, à savoir la faille dite CSRF.

Source : <https://www.leblogduhacker.fr/faille-csrf-explications-contre-mesures/>

Son nom signifie Cross-Site Request Forgery, qui veut dire en français Falsification de Requêtes Inter-Sites.

Il s'agit d'effectuer une action malveillante visant un site précis. C'est l'utilisateur qui la déclenche sans qu'il en ait conscience en exploitant une faille de session. Cette faille de sécurité se produit lorsque l'on est connecté sur un site.

Supposons qu'un utilisateur rédige un article :

<http://monsite.com/create.php?id=1>

Maintenant, un hacker envoie un lien à notre utilisateur :

<http://monsite.com/delete.php?id=1>

Si l'on clique sur ce lien alors que l'on est déconnecté, il y a de fortes chances pour qu'une redirection et un message d'erreur indiquent que l'utilisateur doit être connecté afin de réaliser cette opération.

En revanche, si l'utilisateur clique sur celui-ci pendant qu'il est en session, alors l'action malveillante est exécutée et l'article est supprimé.

Cela signifie que le hacker connaît plus ou moins l'architecture du site pour provoquer ce genre de piratage.

En temps normal, si l'on utilise un CMS comme WordPress, on est protégé contre ce genre de failles, en revanche, si l'on a monté un site à partir de zéro, cette faille est bien réelle.

Une manière de s'en prémunir consiste à utiliser un jeton d'authentification. C'est une chaîne de caractères aléatoirement générée à chaque mise en session de l'utilisateur lors de sa connexion.

De cette manière, il suffit de vérifier si le jeton de session est le même que celui transmis dans un input de type hidden dans un formulaire ou bien dans une URL. Ainsi, il est impossible pour le hacker de deviner à l'avance quelle peut être la chaîne aléatoire, puisqu'elle change à chaque session de connexion.

Pour les opérations délicates du CRUD comme UPDATE ou DELETE, il est recommandé de se servir de cette vérification. Cela permet d'ajouter une couche supplémentaire de sécurité.

Pour le projet Huma Scientio, j'ai créé une méthode `token()` dans la classe `UserSession` qui génère une chaîne de caractères aléatoire à chaque connexion. Pour les formulaires de création et de modification d'un article, je vérifie que la chaîne de caractère transmise dans un input de type hidden correspond bien à celle générée lors de la connexion et dans l'URL pour la procédure de suppression.

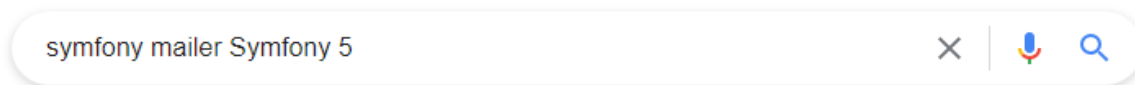
## 7.3 La faille XSS

*Les explications ainsi que les solutions face à la faille XSS se situent en annexe (Annexe 6)*

## 8. Veille technique à partir d'une ressource anglophone

Dans le cadre du projet Huma Scientio, il a été nécessaire de connaître le moyen d'envoyer un email automatiquement à un administrateur afin de le prévenir qu'un nouveau membre s'était inscrit, ou encore à un membre afin de l'avertir de ses nouveaux droits. Pour cela, je me suis servi de la librairie Mailer de Symfony.

Comme nous avons étudié le framework durant la formation, j'ai commencé par faire une recherche simple avec les mots suivants en précisant la version de Symfony :



Parmi les résultats proposés, le premier était un lien direct vers la documentation technique du Mailer de Symfony. Cependant, elle n'offrait pas la possibilité de se servir de cette librairie en dehors du framework. J'ai changé ma recherche car Huma Scientio n'est pas un projet Symfony :



À la suite de cette nouvelle recherche le premier résultat proposé menait vers le site d'un développeur web PHP proposant l'utilisation du Mailer de Symfony sans le framework :

🔗 <https://doeken.org> > Blog > Symfony ▾ [Traduire cette page](#)

### How to use symfony/mailer without the Symfony Framework

9 sept. 2021 — **Swiftmailer** is being replaced by the **symfony/mailer** package. Because of the name, you might think this package can only be used inside a **Symfony** ...

✅ <https://symfony.com> > doc > current ▾ [Traduire cette page](#)

### Sending Emails with Mailer (Symfony Docs)

1 2 3 4 5 6 7 8 \$email = (new Email()) ->getHeaders() // this header tells auto-repliers ("email holiday mode") to **not** // reply to this message because it's ...

[Transport Setup](#) · [Creating & Sending Messages](#) · [Twig: HTML & CSS](#)

Vous avez consulté cette page le 21/02/22.

Il ne me restait plus qu'à suivre les étapes et les extraits de code.

## 9. Extrait de traduction

### 9.1 Source

URL : <https://doeken.org/blog/using-symfony-mailer-without-framework>

On August 19, 2021, Fabien Potencier officially announced [the end of maintenance for Swiftmailer](#). Swiftmailer is being replaced by the `symfony/mailer` package.

Because of the name, you might think this package can only be used inside a Symfony project, but that's not the case. The naming only implies the package is created by Symfony. So let's take a look at this package and how we can use it inside a project without any framework.

#### Introducing the components

To email a recipient you need three things; a mailer service, a transporter and, (of course) a message.

##### Mailer

As you might expect, the `symfony/mailer` package provides a `MailerInterface` with a corresponding `Mailer` service. The `Mailer` service contains the main API and is responsible for sending the message to the appropriate receiver. The interface only has *one* method: `send()`. So the API is very basic and easy to understand. To set up the mailer service a *transporter* is required.

##### Transporter

A transporter is responsible for actually sending a message using a particular protocol. The service must implement the `TransporterInterface`, which also only contains a `send()` method. When you call the `Mailer::send()` method, it will delegate this request to the provided `TransportInterface::send()` method.

Because there are many ways a mail can be sent, there are also many transporters available. By default, this package includes the two most common transporters: `sendmail` and `smtp`. There are however many 3rd party transport services available. A full list of these services can be found on [the documentation site](#).

##### Message

The most important part of sending a message is of course the message itself. Symfony Mailer uses the `symfony/mime` package. This package provides a few handy objects for creating messages that follow the [MIME standard](#). One of these classes is `Email`, which provides a high-level API to quickly create an email message. An `Email` is a data object that contains the message, the recipient, and any other useful headers. This class is also [Serializable](#).

### 9.2 Traduction

Le 19 août 2021, Fabien Potencier a officiellement annoncé la fin de la maintenance pour Swiftmailer. Swiftmailer est remplacé par la librairie `symfony/mailer`.

De par son nom, vous pourriez penser que cette librairie ne peut être utilisée qu'à l'intérieur d'un projet Symfony, mais ce n'est pas le cas. La dénomination implique uniquement que la librairie est créée par Symfony. Examinons donc cette librairie et comment nous pouvons l'utiliser dans un projet sans aucun framework.

#### Présentation des composants

Pour envoyer un e-mail à un destinataire, vous avez besoin de trois choses : un service de courrier, un transporteur et, bien sûr, un message.

#### Messagerie

Comme vous vous en doutez, la librairie `symfony/mailer` fournit un service `MailerInterface` correspondant au `Mailer`. Le service de mail (Mailer) contient l'API principale chargée d'envoyer le

message au destinataire approprié. L'interface n'a qu'une méthode : `send()`. L'API est donc très basique et facile à comprendre. Pour mettre en place le service de messagerie, un transporteur est nécessaire.

## Transporteur

Un transporteur est responsable de l'envoi réel d'un message en utilisant un protocole particulier. Le service doit implémenter le `TransporterInterface` qui ne contient qu'une méthode : `send()`. Lorsque vous appelez la méthode `Mailer::send()`, elle délègue cette demande à la méthode `TransportInterface::send()` fournie.

Parce qu'il existe de nombreuses façons d'envoyer un courrier, il existe également de nombreux transporteurs disponibles. Par défaut, cette librairie inclut les deux transporteurs les plus courants : `sendmail` et `smtp`. Il existe cependant de nombreux services de transports tiers disponibles. Une liste complète de ces services est disponible sur le site de la documentation.

## Un message


La partie la plus importante de l'envoi d'un message est bien sûr le message lui-même. Symfony Mailer utilise la librairie `symfony/mime`. Cette librairie fournit quelques objets pratiques afin de créer des messages qui suivent le standard MIME. L'une de ces classes est `Email` qui fournit une API de haut niveau pour créer rapidement un courrier électronique. Un `Email` est un objet de données qui contient le message, le destinataire et tout autre en-tête utile. Cette classe est également serializable.

[Procédure en PHP du script d'exemple]

## 10. Conclusion

Grâce à la formation "titre professionnel développeur web et web mobile" ainsi que mes expériences durant les stages, j'ai pu mieux comprendre l'ensemble riche, complexe et varié du métier de développeur web. C'est une profession qui possède de nombreuses facettes tout en englobant de multiples compétences transversales comme la gestion de projet, le design, le référencement ou encore l'administration côté serveur.

À titre personnel, je considère que je me sens désormais plus à l'aise sur l'ensemble des parties techniques : l'algorithmique, ou encore la programmation en JavaScript ou PHP bien qu'il ait fallu un certain temps avant que ce soit le cas. J'ai pu également prendre conscience de l'importance de maîtriser certains outils comme Git à titre personnel pour versionner des projets sur lesquels j'ai travaillé seul, d'autres logiciels de versionnings lors des projets d'équipe ou encore du suivi de workflows durant mes stages.



A la suite de cette formation passionnante et extrêmement enrichissante, j'ai l'intention de me tourner vers des postes de développeurs web qui me permettront de monter en compétences notamment sur des frameworks PHP comme Symfony ou Laravel ainsi que sur des frameworks front-end comme Vue.js ou React.js

Je tiens tout particulièrement à remercier l'équipe pédagogique qui m'a accompagné au cours de cette formation : Olivier Meunier, Emmanuel Adinaraynanin ainsi que Sylvain Joyon. Je ne peux pas me permettre bien sûr d'oublier l'équipe administrative ainsi que Delphine Videbien qui m'ont aidé durant mon parcours.

Je tiens également à remercier Jérémy Callejon et Gaël Duflou qui, durant mes stages, m'ont permis de peaufiner ce projet.

Je vous remercie pour votre attention.