

DOSSIER PROJET

Titre Professionnel Développeur Web & Web Mobile



GANDEMÉR Luigi
Session 2021 – 2022

SOMMAIRE

1) Introduction	3
2) Compétences couvertes par le projet	4
3) Abstract	5
4) Cahier des charges	6
A) Présentation du projet	6
B) Contexte et besoin	7
C) Les couleurs	8
D) Les polices	8
5) Analyse Fonctionnelle	9
A) Diagramme cas d'utilisation	9
B) Diagramme de activité	10
C) Diagramme de séquence	12
D) Diagramme de classe	14
E) Arborescence	15
F) Maquettage	16
6) Conception Système d'Information	22
A) MCD	22
B) MLD	23
C) SQL Structure	24
7) Outils techniques	26
A) Langages et choix technologiques	26
B) Architecture MVC	29
8) Fonctionnalités	30
A) Authentification	30
A.1) Inscription	30
A.2) Connexion	32
B) Les articles	35
B.1) Création des articles	35
B.2) Traitement des images	37
B.3) Afficher les articles	38
B.4) Mettre à jour les articles	40
B.5) Suppression des articles	42
C) Les commentaires	44
C.1) Création d'un commentaire	44
C.2) Suppression d'un commentaire	45
D) Gestion des failles	46
D.1) Failles XSS	46
D.2) Injection SQL	47
D.3) Failles CSRF	48
9) Interface Administrateur	49
10) Conclusion	50
11) Annexe	51
A) CSS	51
B) Message d'alerte	52
C) Le JavaScript	53

1) INTRODUCTION

Avant d'entrer en formation à l'ADRAR j'étais en première année de licence AES (Administration Économique et Social) à l'université Paul Valéry de Montpellier, j'ai arrêté au bout de 3 mois car ça ne me plaisait pas.

Je savais que je voulais me diriger vers le domaine de l'informatique j'ai donc fait une formation Projet Pro d'une durée de 8 mois. Pendant cette formation j'ai eu l'opportunité d'effectuer deux stage : un en maintenance informatique mais c'était pas exactement ce que je recherchais.

Je me suis dirigé vers le développement web et j'ai immédiatement accroché. J'ai effectué un stage en tant que développeur web en freelance et j'ai créée un site vitrine avec Bootstrap pour un toiletteur félin et canin.

Une fois ce stage terminé j'ai décidé d'intégrer l'ADRAR pour suivre une formation de développeur web et web mobile.

Au cours de ma formation à l'ADRAR j'ai appris à créer des sites Web avec HTML5, CSS3, JavaScript, PHP et SQL.

Après 7 mois de formation j'ai eu l'opportunité d'effectuer un stage en distanciel dans l'entreprise Enimad, une société de growth marketing digital à Montpellier. J'ai appris à manipuler Elementor Pro et j'ai pu créer 5 sites avec le CMS WordPress dont 3 e-commerce et 1 site de formation en ligne avec le CMS Moodle.

2) COMPÉTENCES DU TITRE COUVERTES PAR LE PROJET

Voici les compétences validées par mon projet pour l'obtention du titre professionnel :

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles	Dossier Projet	DP
1	Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité.	1	Maquetter une application	✓	
		2	Réaliser une interface utilisateur web statique et adaptable	✓	
		3	Développer une interface utilisateur web dynamique	✓	
		4	Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce		✓
2	Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité.	5	Créer une base de données	✓	
		6	Développer les composants d'accès aux données	✓	
		7	Développer la partie back-end d'une application web ou web mobile	✓	
		8	Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce		✓

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles	Dossier Projet	DP
1	Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	1	Maquetter une application	✓	
		2	Développer une interface utilisateur de type desktop	✓	
		3	Développer des composants d'accès aux données	✓	
		4	Développer la partie front-end d'une interface utilisateur web	✓	
		5	Développer la partie back-end d'une interface utilisateur web	✓	

3) ABSTRACT

Presentation type : Reading only

Title : Sneakers Certifiées

Author : Luigi Gandemer

Institutions : ADRAR Formation

Abstract body :

Today I am presenting a web application which is a blog for sneakers enthusiasts, such as people who collect rare or basic sneakers. Many sneaker blogs exist in this day and age but our blog's interface is simpler and easier to use, we don't run ads and our content is very spacious and readable.

Thanks to this application, people will be able to keep up with the brand new and collectible sneakers that are released by brands like Nike, Adidas and others. This web app makes it easier to view content related to the sneakers, such as the colorway, the date of release and additional information regarding the sneakers.

This new app is targeting young and older people who are either passionate about sneakers or even regular people who just want to know a bit more about what they might have on their feet. This application will include a publishing system for the administrator to manage, he will be able to publish and display articles on sneakers with an image and two paragraphs. The administrator will also be able to modify and delete his articles.

This app will be equipped with an administration system to delete any malicious or harmful users. It will also include a sign up and sign in system, the users' id, email address, password, role, city and zipcode will be stored in our database.

For now this app is only available in France and in French but in the future I would like to add every French-speaking country as well as the English language to be able to include every English-speaking country to develop the community on this web app.

4) CAHIER DES CHARGES

A) Présentation du projet

L'objectif de mon projet était de créer un blog avec une interface simple et moderne. Ce blog permet de mettre en avant les informations et l'histoire derrière des paires de chaussures classiques, récentes et exclusifs.

Ce blog a un seul administrateur capable de publier des articles avec des photos et descriptions concernant les paires de chaussures. Les utilisateurs eux auront uniquement accès à ses articles s'ils sont connectés.

Ce secteur est très populaire sur le web donc il y a beaucoup de concurrence, mais «Sneakers Certifiées» se démarque avec un design moderne, aéré et efficace.

Les concurrents existent, pour ne que citer les plus importants :

- <https://www.lesitedelasneaker.com>
- <https://www.sneakers.fr>
- <https://www.sneakers-actus.fr>

Le problème majeur de tous ses sites web est l'interface trop chargé en information.

B) Contexte et besoins

➤ L'**objectif** du site web :

Sneakers Certifiées est une application web et web mobile qui permet à ses utilisateurs de s'informer plus facilement sur les sneakers qu'ils possèdent ou souhaiteraient posséder.

➤ Le **public visé** :

Ce blog vise un public jeune passionné ou tout simplement amateur de sneakers, qui cherche à apprendre un peu plus sur la paire qu'ils ont au pieds ou bien qu'ils souhaitent acheter.

➤ Comment **se démarquer** de la concurrence :

Sneakers Certifiées se démarquera avec son design simple, moderne et aéré. Ce design permettra au utilisateurs d'absorber toutes les informations concernant les sneakers de leur choix.

C) Les couleurs

Les couleurs sélectionnées pour ce projet sont simples et permettent de mettre en place un design moderne.

FFFFFF

White

BBBBBB

Gray X.11 Gray

0F0F0F

Smoky Black

000000

Black

D) Les polices

Les polices sélectionnées pour ce projet sont des polices classiques et simples on évite donc toutes prise de risque.

- **Poppins SemiBold**
- **Montserrat SemiBold**
- Roboto Thin

Titre principal

Sous-titre

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean at magna augue. Aliquam pulvinar luctus justo in dignissim. Maecenas dui odio, sodales ac est et, eleifend vestibulum nisl. In finibus tellus ut ex laoreet molestie. Duis pretium, justo id laoreet lobortis, augue metus.

5) ANALYSE FONCTIONNELLE

A) Diagramme cas d'utilisation

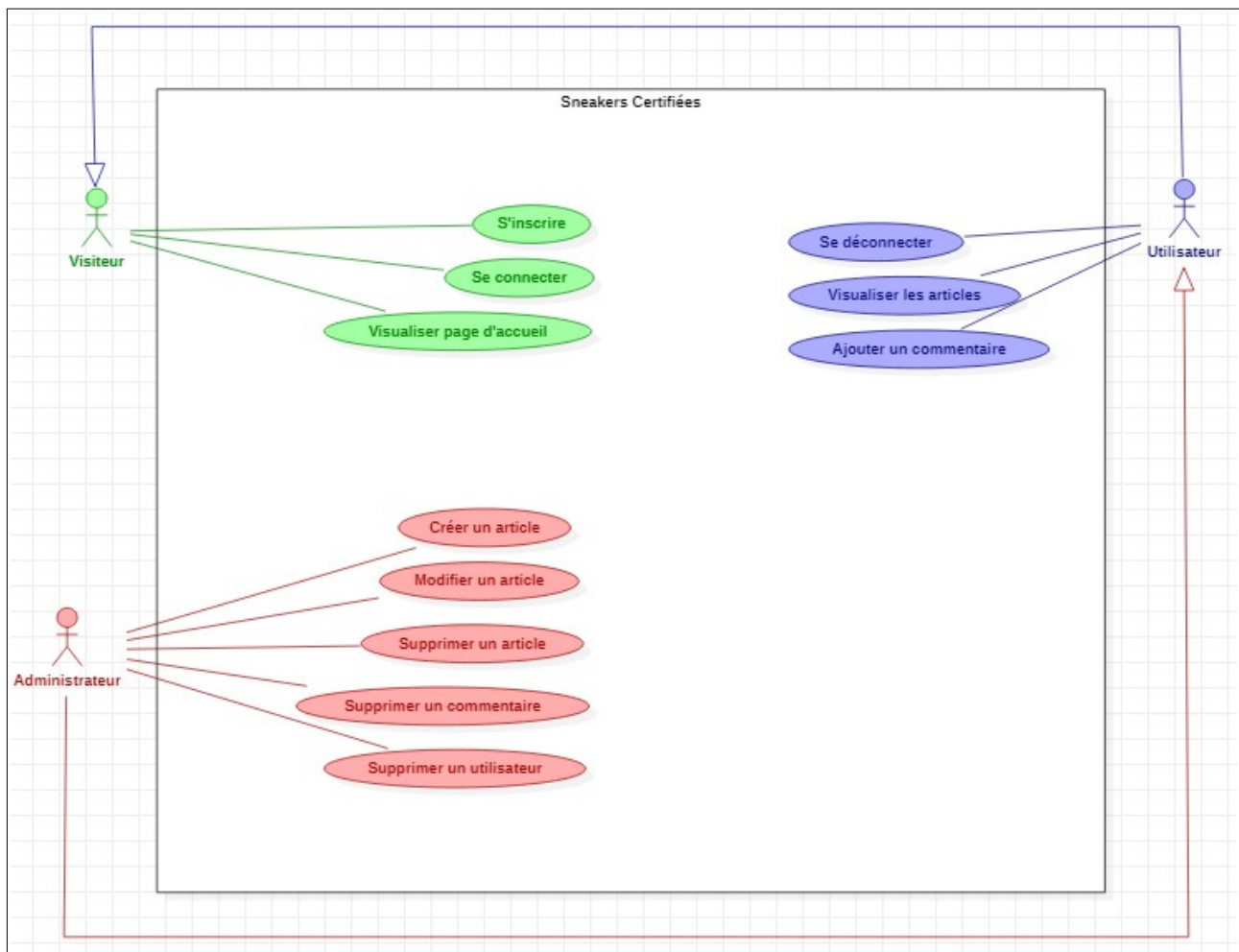
Le diagramme de cas d'utilisation permet de mettre en place les acteurs de notre application web et leurs fonctionnalités.

Notre cas d'utilisation représente 3 **acteurs** :

Les **visiteurs** peuvent se déconnecter, se connecter, s'inscrire et visualiser la page d'accueil.

Les **utilisateurs** peuvent visualiser des articles et ajouter un commentaire. Les utilisateurs **héritent** des fonctions des visiteurs.

Les **administrateurs** peuvent créer , modifier et supprimer des articles. Ils peuvent aussi supprimer des utilisateurs et des commentaires. Les administrateur **héritent** des fonctions des utilisateurs et donc de celles des visiteurs aussi.

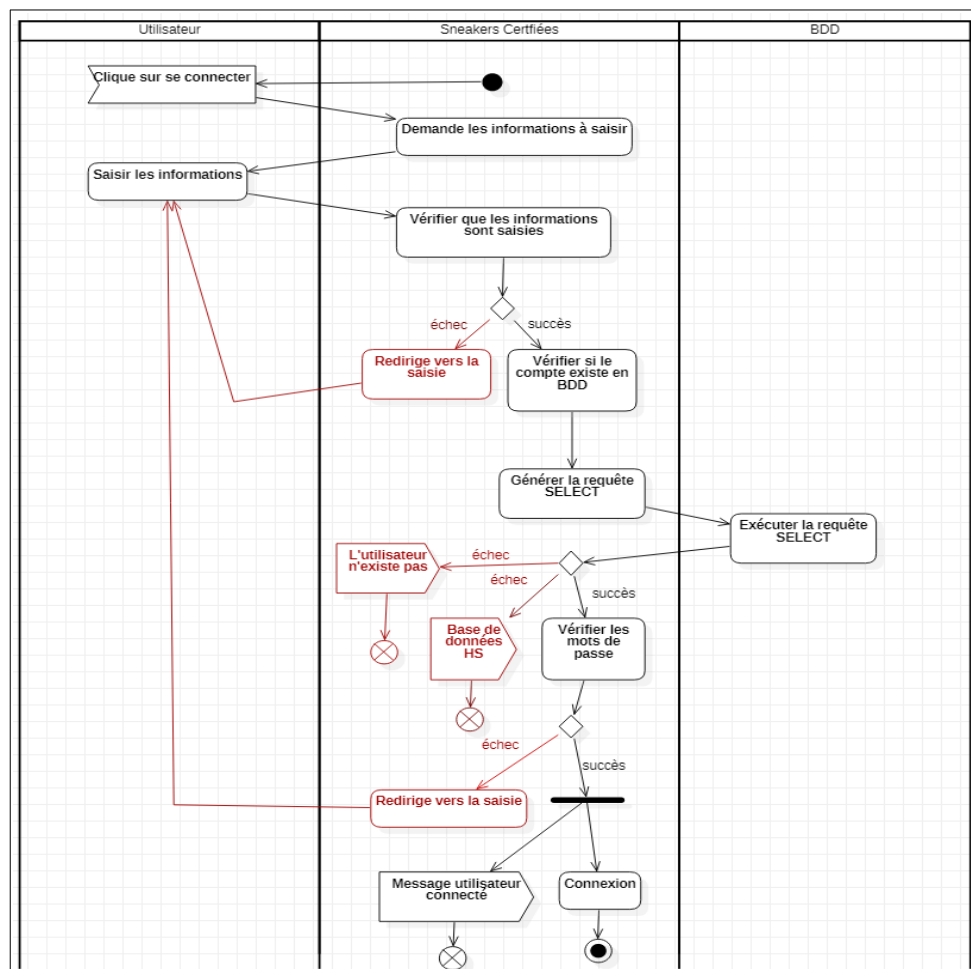


B) Diagramme d'activité

➤ Diagramme d'activité connexion

Ici on peut voir qu'en partant du **nœud initial** le flot de contrôle se dirige vers l'utilisateur qui reçoit un **signal d'acceptation** :

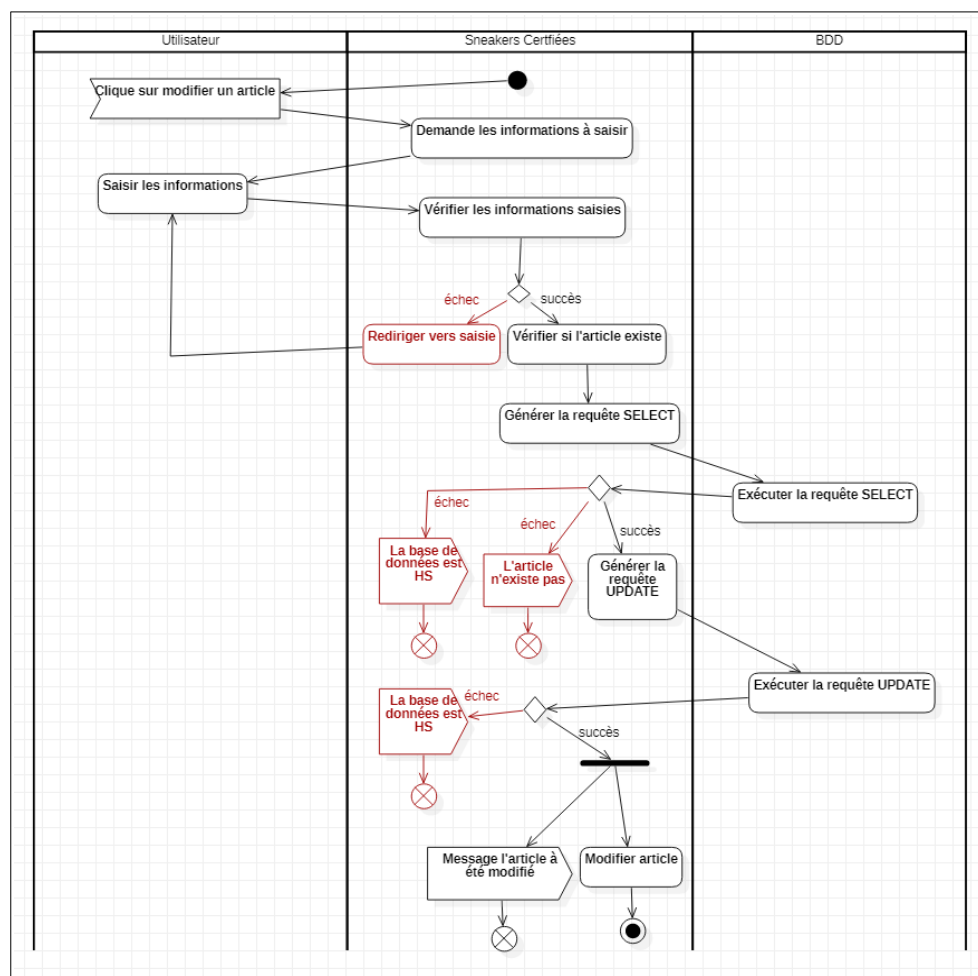
- Il clique d'abord sur se connecter, le flot se dirige vers une **action** qui lui demande de saisir des informations.
- L'utilisateur saisit les informations demandées.
- Le système vérifie les informations. Un **nœud de décision** effectue un choix. Si les informations sont incorrectes, on redirige l'utilisateur vers la saisie, sinon on lance une nouvelle **action** qui vérifie si le compte existe en BDD.
- Le système génère alors une requête SQL. Le flot se dirige vers la BDD qui elle, exécute cette requête.
- Le système vérifie alors les informations, on a 2 cas d'échec ici soit l'utilisateur n'existe pas et on transite vers un **nœud de fin de flot**, ou la BDD est HS et on renvoie un **send signal** puis transite vers un **nœud de fin de flot** une fois de plus.
- Une fois de plus un **nœud de décision** effectue un choix : en cas d'échec on redirige l'utilisateur vers la saisie sinon un **fork** qui connecte l'utilisateur et atteint un **nœud de fin d'activité**, puis renvoie un **send signal** et notifie l'utilisateur de sa réussite et transite vers un **nœud de fin de flot**.



➤ Diagramme de d'activité modifier un article

Ici en partant du **nœud initial** le flot de contrôle se dirige vers l'utilisateur qui reçoit un **signal d'acceptation** :

- Il clique d'abord sur modifier un article, le flot se dirige alors vers une **action** qui lui demande de saisir les informations.
- L'utilisateur saisit les informations demandées.
- Le système vérifie alors les informations saisies avec un **nœud de décision**, en cas d'échec le flot se redirige vers la saisie à nouveau, en cas de succès on vérifie si l'article existe et on génère une requête SQL. Le flot se dirige vers la BDD qui elle exécute cette requête.
- Le système fait des vérifications à l'aide d'un **nœud de décision** : 2 échecs sont possibles ici, soit la base de données est HS, soit l'article n'existe pas, dans les 2 cas on renvoie un **send signal** et on transite vers un **nœud de fin de flot**.
- En cas de succès, on génère une requête, puis le flot transite vers la BDD qui elle exécute une requête.
- Le flot transite vers le système vers un **nœud de décision**, si la BDD est HS le cas est un échec et on renvoie un **send signal** qui transite vers un **nœud de fin de flot**. En cas de succès on transite vers un **fork** qui modifie l'article et transite vers un **nœud de fin d'activité** et renvoie un **send signal** qui indique que l'article a été modifié puis transite vers un **nœud de fin de flot**.



C) Diagramme de séquence

➤ Diagramme de séquence connexion

On peut voir 3 **Lifeline** : L'utilisateur, Le Système d'information et la BDD.

Ici en partant du premier point, l'utilisateur envoie un **message** « Cliquez sur se connecter » vers le SI.

- Le SI envoie un **message de réponse** en demandant à l'utilisateur de saisir les informations.

- L'utilisateur envoie un **message** « Saisir les informations », le SI envoie un **self message** pour vérifier les informations saisies. Ici un **scénario alternatif** s'impose. Si il manque des informations, l'utilisateur est renvoyé à la saisie d'informations.

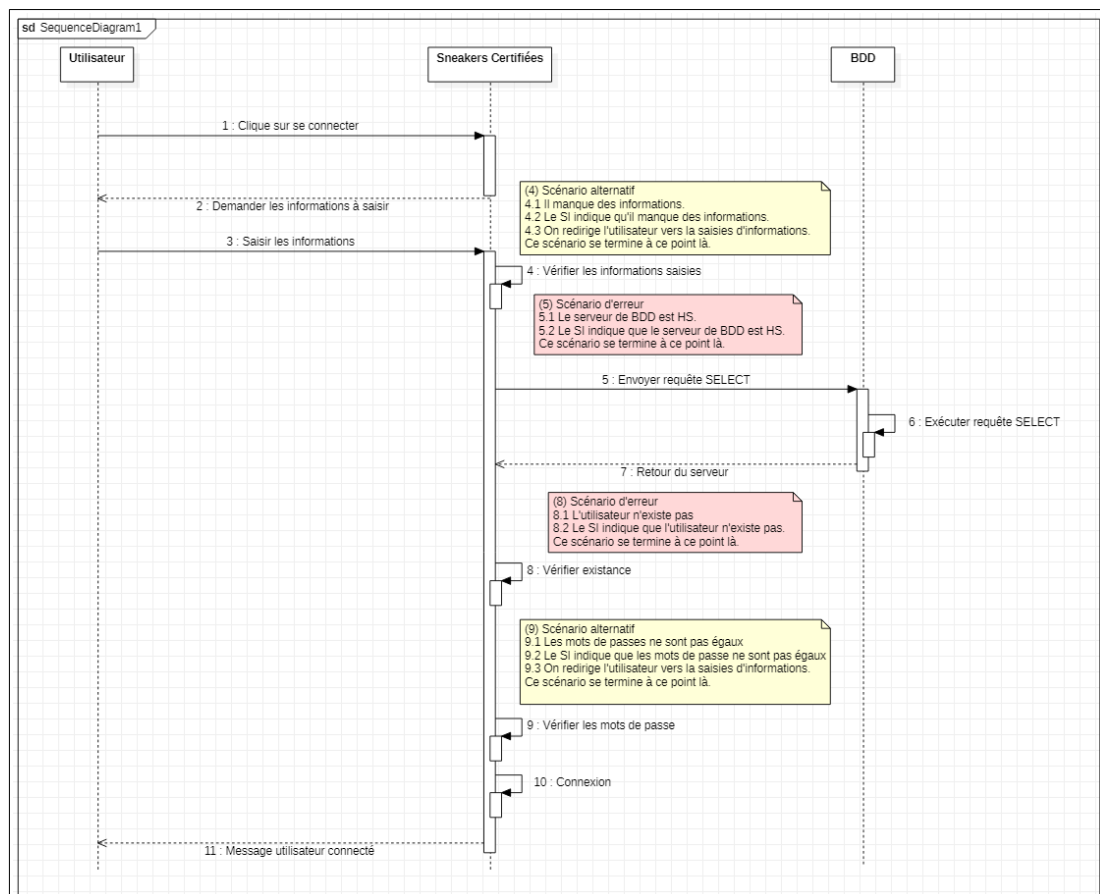
- Sinon le SI envoie un **message** à la BDD et envoie un requête SELECT. Ici un **scénario d'erreur** est possible : La BDD est HS le scénario se termine à ce point.

- Sinon la BDD exécute la requête avec un **self message**. La BDD renvoie la réponse du serveur avec un **message de réponse** vers le SI.

- Le SI vérifie ensuite l'existence du compte avec un **self message**. Un **scénario d'erreur** est possible : L'utilisateur n'existe pas, le scénario se termine ici.

- Puis le SI vérifie si les mots de passes sont égaux avec un **self message**. Ici un **scénario alternatif** s'impose : les mots de passes ne sont pas égaux donc on redirige l'utilisateur vers la saisie d'information.

- Sinon on connecte l'utilisateur avec un **self message** et le SI renvoie un **message de réponse** vers l'utilisateur notifiant que l'utilisateur est connecté.



➤ Diagramme de séquence modifier un article

Ici en partant du premier point, l'utilisateur envoie un **message** « Cliquez sur modifier un article » vers le SI.

- Le SI envoie un **message de réponse** en demandant à l'utilisateur de saisir les informations.

- L'utilisateur envoie un **message** « Saisir les informations », le SI envoie un **self message** pour vérifier les informations saisies. Ici un **scénario alternatif** s'impose, si il manque des informations l'utilisateur est renvoyé à la saisie d'informations.

- Sinon le SI envoie un **message** à la BDD et envoie une requête SELECT. Ici un **scénario d'erreur** est possible : La BDD est HS le scénario se termine à ce point.

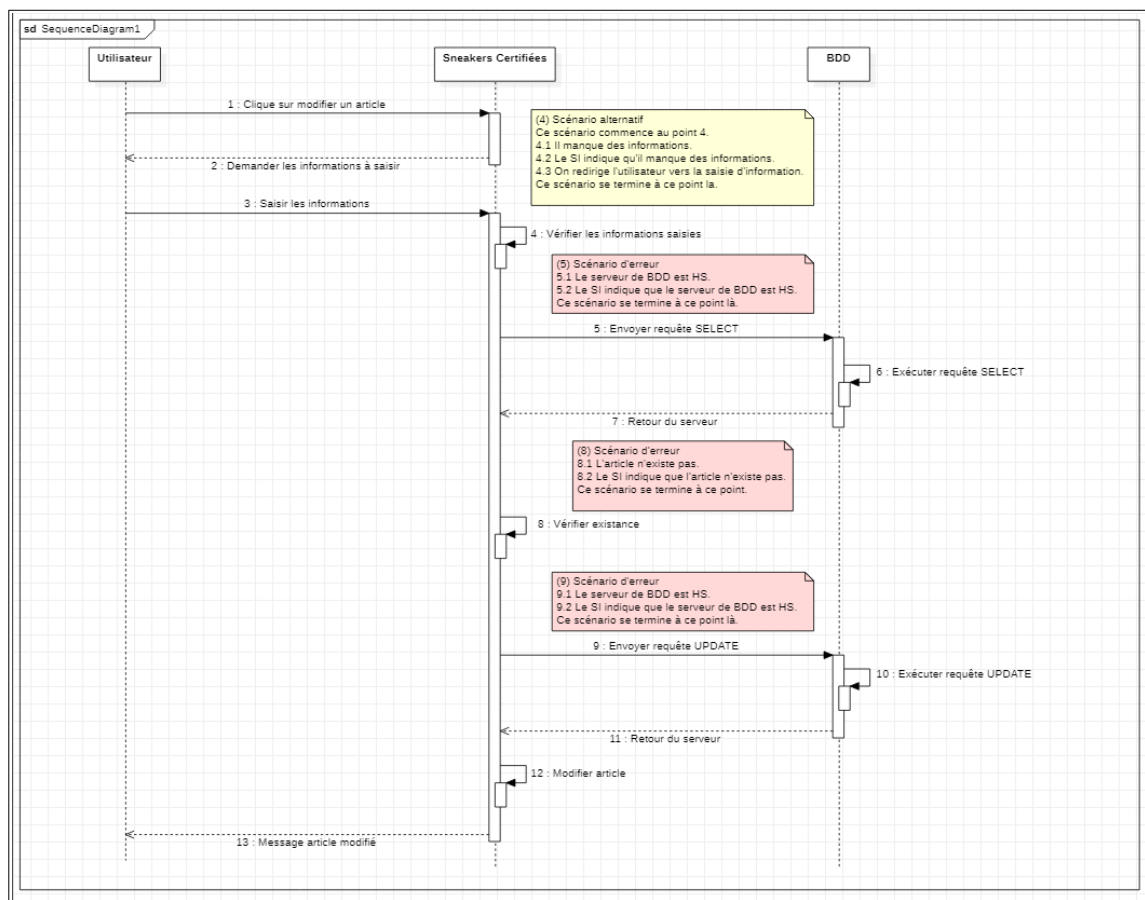
- Sinon la BDD exécute la requête avec un **self message**. La BDD renvoie la réponse du serveur avec un **message de réponse** vers le SI.

- Le SI vérifie ensuite l'existence de l'article avec un **self message**. Un **scénario d'erreur** est possible : L'article n'existe pas, le scénario se termine ici.

- Sinon le SI envoie un **message** à la BDD et envoie une requête UPDATE. Ici un **scénario d'erreur** est possible : La BDD est HS le scénario se termine à ce point.

- Sinon la BDD exécute la requête avec un **self message**. La BDD renvoie la réponse du serveur avec un **message de réponse** vers le SI.

- Sinon le SI met l'article à jour avec un **self message** et ensuite renvoie un **message de réponse** vers l'utilisateur le notifiant que l'article a bien été modifié.



D) Diagramme de classe

Un diagramme de classe permet de spécifier qui intervient à l'intérieur du système. Il spécifie quels liens peuvent entretenir les objets du système.

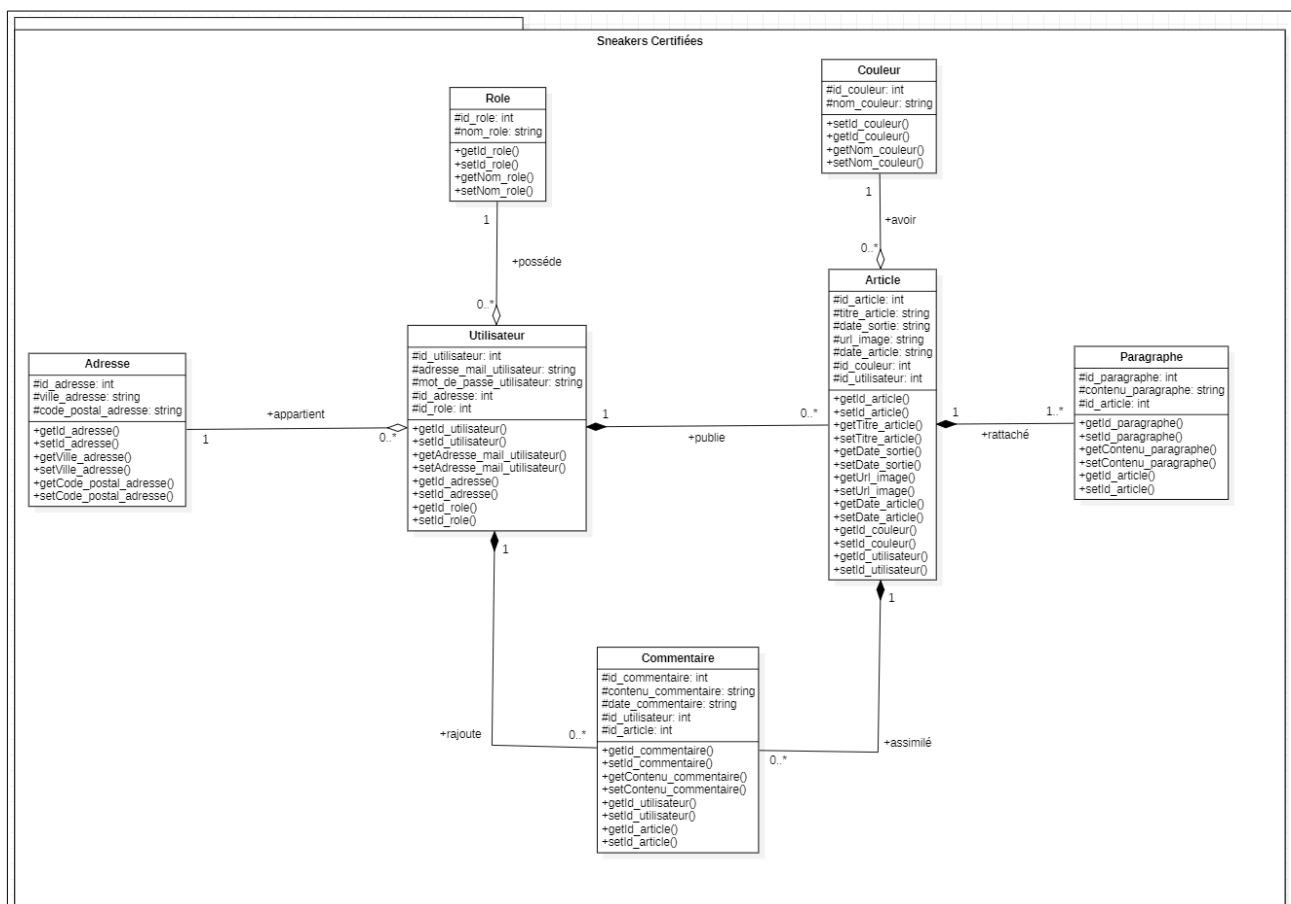
Ici on peut voir que toutes nos classes ont des **attributs** protected, les attributs correspondent aux propriétés de nos classes.

Les **opérations** de nos classes sont des getters et des setters, les opérations correspondent aux méthodes de nos classes.

On peut voir les **agrégations** et les **compositions** qui représentent les associations des classes.

par exemple :

- Une adresse ou un rôle est dans un utilisateur. (agrégation)
- Si on supprime un utilisateur, on supprime ses commentaires et ses articles. (compositions)



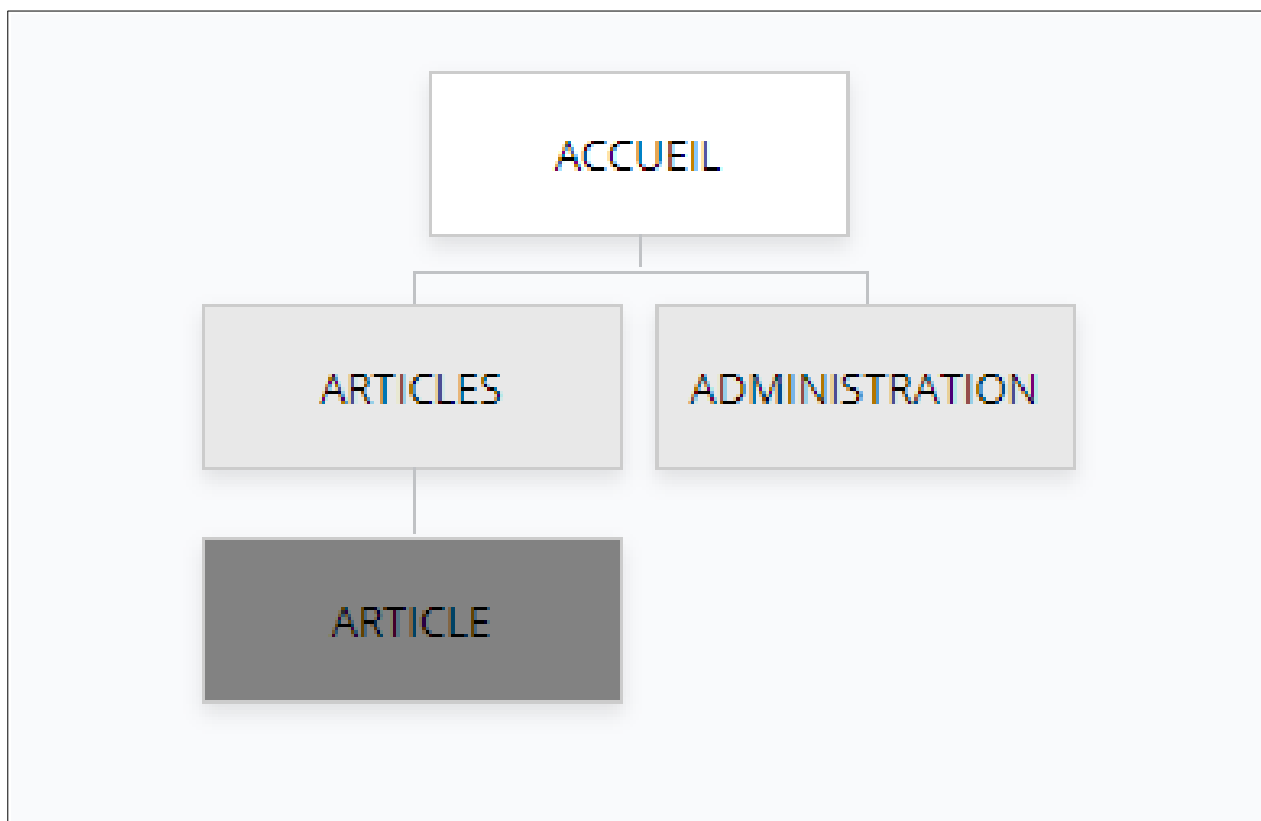
E) Arborescence

L'arborescence du blog « Sneakers Certifiées » est assez légère, en effet l'intégralité des formulaires (inscription, connexion, ajouter un article, modifier un article, supprimer un article, un commentaire, un utilisateur) sont sous formes de **modales pour améliorer l'expérience utilisateur** (moins de chargement de page).

Les **visiteurs** ont uniquement accès à la page d'accueil.

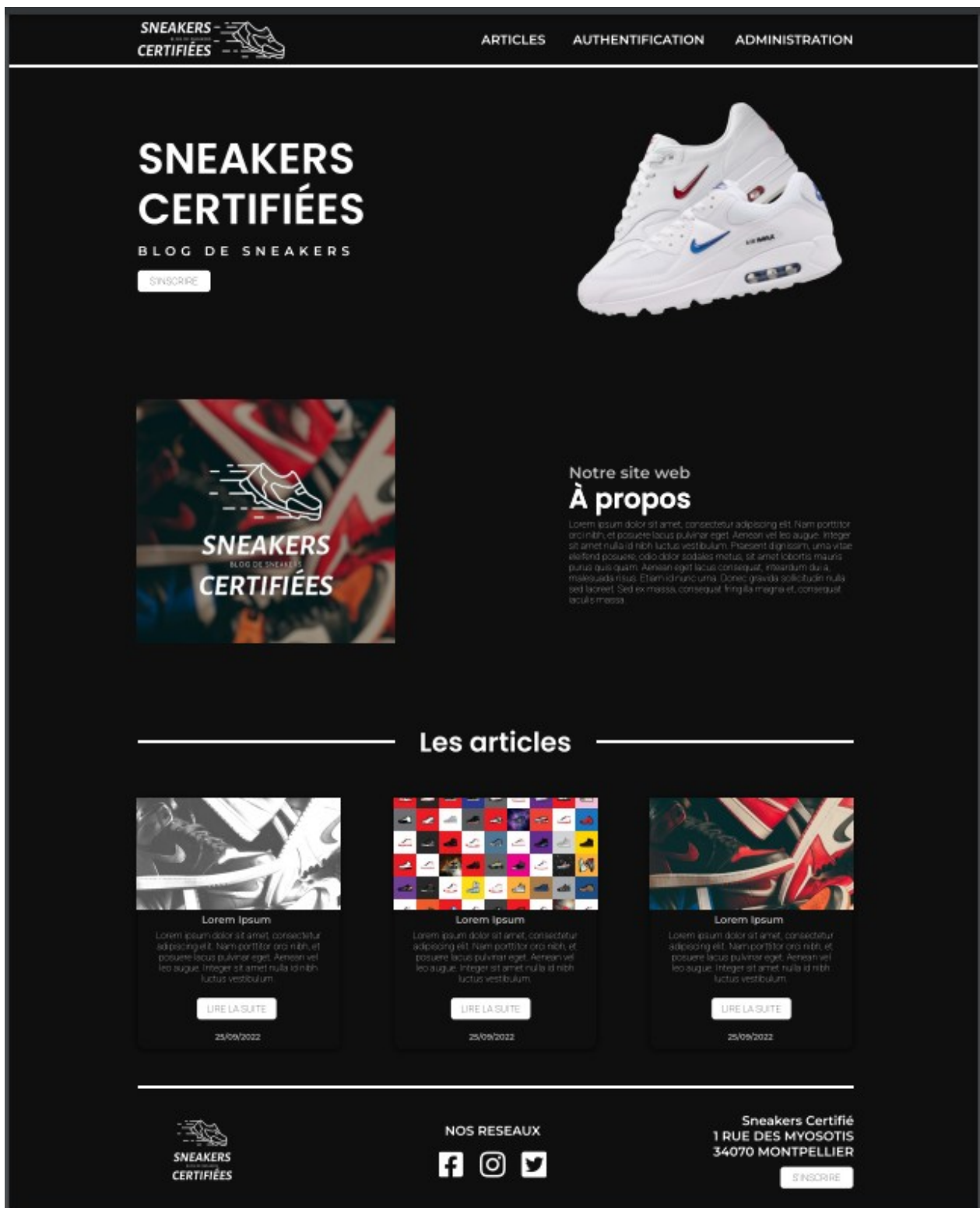
Les **utilisateurs** connectés, eux, ont accès à la page d'accueil, la grille d'articles et les articles uniques.

L'**administrateur** lui a une page de plus, l'administration des utilisateur où il peut supprimer un utilisateur.



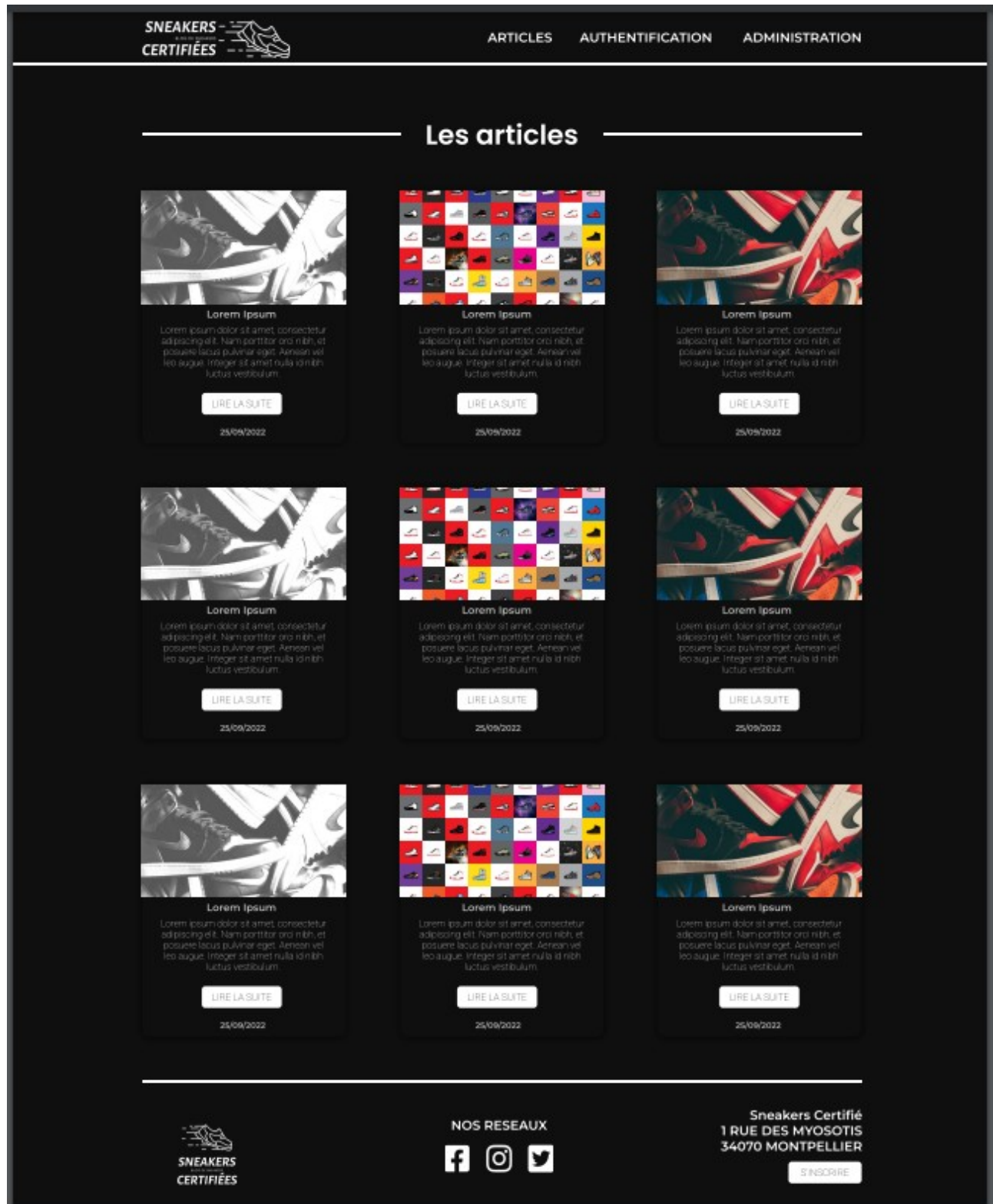
F) Maquettage

Pour la page d'accueil du site web on cherche à produire un design simple et aéré pour que l'utilisateur puisse absorber les informations sans être perturbé par trop d'informations contrairement aux sites concurrents.



Le design pour la grille d'articles cherche lui aussi à être simple, efficace et moderne.

A l'aide d'un **flex-wrap** et d'une **width de 33 %** sur les containers cela est simple à produire.



SNEAKERS
MONSIEUR
CERTIFIÉES

ARTICLES

AUTHENTIFICATION

ADMINISTRATION

Couleur du modèle

Nom de la paire

Date de sortie : 01/01/1940

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum dapibus odio eget tellus cursus bibendum. Curabitur ut ex auctor, ultrices nibh id, pulvinar mi. Morbi non consectetur mauris. Fusce sed volutpat nulla. Quisque et fermentum enim. Curabitur aliquam, lorem quis tristique pretium, elit metus ornare leo, in pretium orci enim vitae nisi. Suspendisse a egestas turpis. Praesent at aliquet est, at porttitor risus. Fusce finibus nisi quis rhoncus elefend.

Interdum et malesuada fames ac ante ipsum primis in faucibus. Donec a urna ut diam luctus hendrerit in id nibh. Nam lobortis interdum ante eget scania. Nunc maximus ligula nec tellus molestie feugiat. Duis nec justo ac magna volutpat semper vel vitae elit. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Praesent a condimentum felis, quis vehicula du. Vivamus dictum nisi vitae felis vehicula condimentum. Nulla facilis. Curabitur sit ante nec ligula pulvinar porta et in massa. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Morbi vitae elit arcu. Donec vitae aliquam augue.

Praesent ligula quam, dapibus quis nisi nec, venenatis volutpat, mi. In efficitur, inodunt ligula, dictum tempus quam, malesuada a. Duis ultricies dolor sit amet dui pellentesque auctor. Pellentesque eu egestas augue. Sed sit amet libero nulla. In hac habitasse platea dictumst. Ut vel tempus nibh, quis dapibus ex. Duis eu accumsan justo. Vivamus cursus sagitta nulla.

Curabitur tristique sodales ipsum sit venenatis. Pellentesque nec vestibulum sapien. Morbi tempor egestas libero, quis viverra elit vestibulum quis. Duis posuere quam magna, id venenatis ex gravida interdum. Sed viverra molestie est, non pharetra enim cursus sed. One lobortis lectus elementum massa venenatis sagitta. Praesent tristique finibus leo et luctus. Suspendisse hendrerit tristique magna ac mollis. Etiam maximus ultrices vulputate. Fusce a elit congue, semper nibh a, tempus nulla. Phasellus venenatis, nulla vel viverra tristique, lectus mi posuere est, id tristique tellus orci eu metus. Vivamus risus ante, elefend ut consectetur vitae, egestas non quam. Curabitur blandit faucibus libero sed interdum. Sed vel accumsan augue, at ultrices nisi.

Aliquam ullamcorper mauris ligula, ac posuere enim convallis sed. Morbi at justo condimentum, aliquet eros a, aliquet massa. Integer pharetra, est dictum fermentum in pharetra, dolor risus condimentum nisl, et sodales nisl arcu vel eros. Sed arcu nunc, vulputate eget lacus eu, porttitor ultrices elit. Phasellus in gravida lectus. Duis ut leo a leo. Aliquam suscipit felis tortor, id pharetra diam tempor nec. Nam odio nunc, tristique vitae tellus sit amet, inodunt ornare metus. Quisque elefend congue nisi, nec tempor erat, auctor vitae. Fusce suscipit augue lorem, ut tempor magna feugiat sit. In pulvinar magna sem, vitae consequat quam varius id. In vulputate arcu nisl, et gravida urna sagittis et. Praesent pulvinar, arcu quis elementum blandit, lacus tellus finibus nibh, et pharetra orci arcu commodo urna. Curabitur imperdiet, massa et malesuada interdum, ipsum magna rutrum neque, fringilla suscipit est neque vitae velit. Quisque vel fringilla est. Aenean non arcu finibus, ultrices tellus et, feugiat nisl.

Date de création article : 01/01/1970

SNEAKERS
MONSIEUR
CERTIFIÉ

NOS RESEAUX

f o t

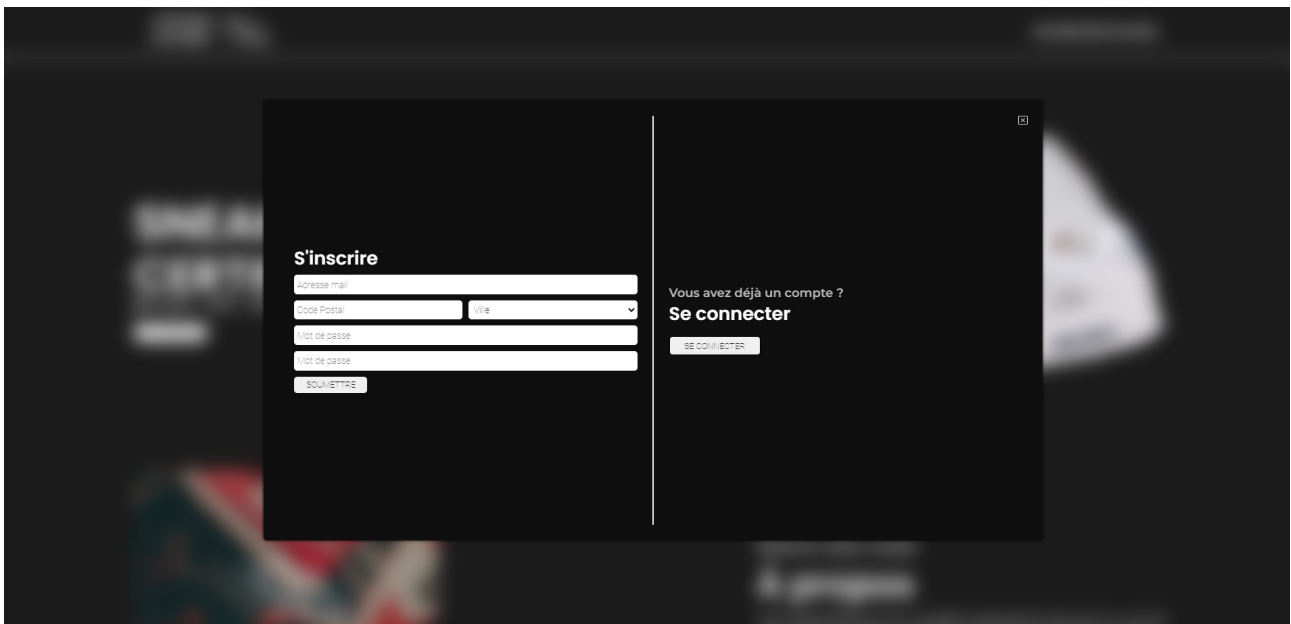
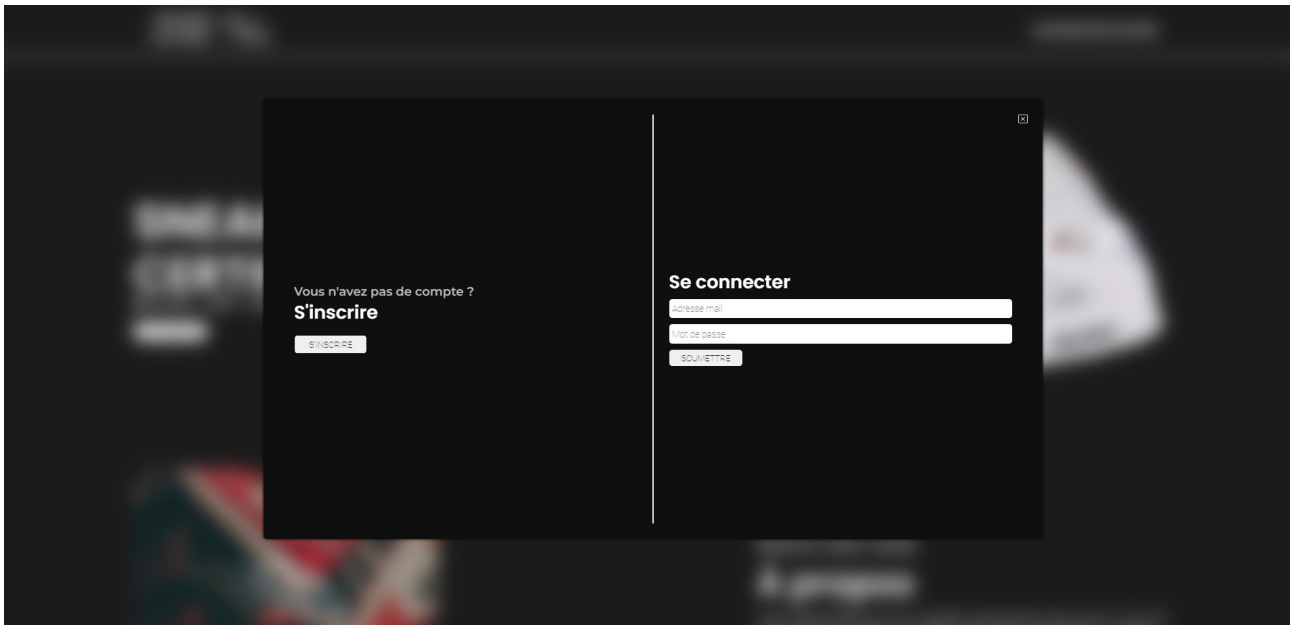
Sneakers Certifié
1 RUE DES MYOSOTIS
34070 MONTPELLIER

S'INSCRIRE

Pour l'authentification, l'utilisation de modals semble nécessaire.

Les modals permettent à l'utilisateur d'avoir une expérience plus agréable sur le site web avec moins de chargement de page et un temps de réactivité plus rapide.

Avec du JavaScript on peut produire cette expérience recherchée.



SNEAKERS

CERTIFIÉES

ARTICLES

AUTHENTIFICATION

ADMINISTRATION

SNEAKERS

CERTIFIÉES

BLOG DE SNEAKERS

23/09/2022



Notre site web

À propos

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam porttitor, orci nisl, et posuere lacus pulvinar eget. Aenean vel leo augue. Integer sit amet nulla id nisl luctus vestibulum. Praesent dignissim, urna vitae eleifend posuere, odio dolor sodales metus, sit amet lobortis mauris purus quis quam. Aenean eget, lacus consequat, interdum dui a, malesuada risus. Etiam id nunc urna. Donec gravida sollicitudin nulla sed laoreet. Sed ex massa, consequat fringilla magna et, consequat iaculis massa.

SNEAKERS

CERTIFIÉES

Les articles



23/09/2022



SNEAKERS

CERTIFIÉES

ARTICLES

AUTHENTIFICATION

ADMINISTRATION

Les articles



23/09/2022





23/09/2022



SNEAKERS

CERTIFIÉES

NOS RESEAUX

f

ig

tw

Sneakers Certifié

1 RUE DES MYOSOTIS

34070 MONTPELLIER

SNEAKERS

CERTIFIÉES

ARTICLES

AUTHENTIFICATION

ADMINISTRATION

Couleur du modèle

Nom du modèle

Date de sortie : 01/01/1940



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam porttitor, orci nisl, et posuere lacus pulvinar eget. Aenean vel leo augue. Integer sit amet nulla id nisl luctus vestibulum. Praesent dignissim, urna vitae eleifend posuere, odio dolor sodales metus, sit amet lobortis mauris purus quis quam. Aenean eget, lacus consequat, interdum dui a, malesuada risus. Etiam id nunc urna. Donec gravida sollicitudin nulla sed laoreet. Sed ex massa, consequat fringilla magna et, consequat iaculis massa.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam porttitor, orci nisl, et posuere lacus pulvinar eget. Aenean vel leo augue. Integer sit amet nulla id nisl luctus vestibulum. Praesent dignissim, urna vitae eleifend posuere, odio dolor sodales metus, sit amet lobortis mauris purus quis quam. Aenean eget, lacus consequat, interdum dui a, malesuada risus. Etiam id nunc urna. Donec gravida sollicitudin nulla sed laoreet. Sed ex massa, consequat fringilla magna et, consequat iaculis massa.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam porttitor, orci nisl, et posuere lacus pulvinar eget. Aenean vel leo augue. Integer sit amet nulla id nisl luctus vestibulum. Praesent dignissim, urna vitae eleifend posuere, odio dolor sodales metus, sit amet lobortis mauris purus quis quam. Aenean eget, lacus consequat, interdum dui a, malesuada risus. Etiam id nunc urna. Donec gravida sollicitudin nulla sed laoreet. Sed ex massa, consequat fringilla magna et, consequat iaculis massa.

Date création article : 01/01/1970

SNEAKERS

CERTIFIÉES

NOS RESEAUX

f

ig

tw

Sneakers Certifié

1 RUE DES MYOSOTIS

34070 MONTPELLIER

INSCRIRE

Pour le responsive du site web on pourrait se servir d'un **flex-direction : column** ; en CSS pour empiler tout les éléments et une fois de plus opter pour un design simple, efficace et moderne.

Page | 20

Pour les modals d'authentification, ils prendront l'intégralité de la largeur de l'écran une fois sur mobile.

On pourrait se servir une fois de plus du **flex-direction : column** ; pour empiler les containers.



A dark-themed mobile modal with a close button (X) in the top right corner. The modal is divided into two sections by a horizontal line. The top section is for registration, with the heading "Vous n'avez pas de compte ? S'inscrire" and a single "S'INSCRIRE" button. The bottom section is for login, with the heading "Se connecter" and two input fields for "Adresse mail" and "Mot de passe", followed by a "SOUMETTRE" button.

✕

Vous n'avez pas de compte ?
S'inscrire

S'INSCRIRE

Se connecter

Adresse mail

Mot de passe

SOUMETTRE



A dark-themed mobile modal with a close button (X) in the top right corner. The modal is divided into two sections by a horizontal line. The top section is for registration, with the heading "S'inscrire" and four input fields: "Adresse mail", "Code Postal", "Ville" (with a dropdown arrow), and "Mot de passe" (repeated twice), followed by a "SOUMETTRE" button. The bottom section is for login, with the heading "Vous avez déjà un compte ? Se connecter" and a single "SE CONNECTER" button.

✕

S'inscrire

Adresse mail

Code Postal

Ville ▾

Mot de passe

Mot de passe

SOUMETTRE

Vous avez déjà un compte ?
Se connecter

SE CONNECTER

6) CONCEPTION DU SYSTÈME D'INFORMATION

A) MCD

Le modèle conceptuel des données a pour but d'écrire de façon formelle les données qui seront utilisées par le système d'information.

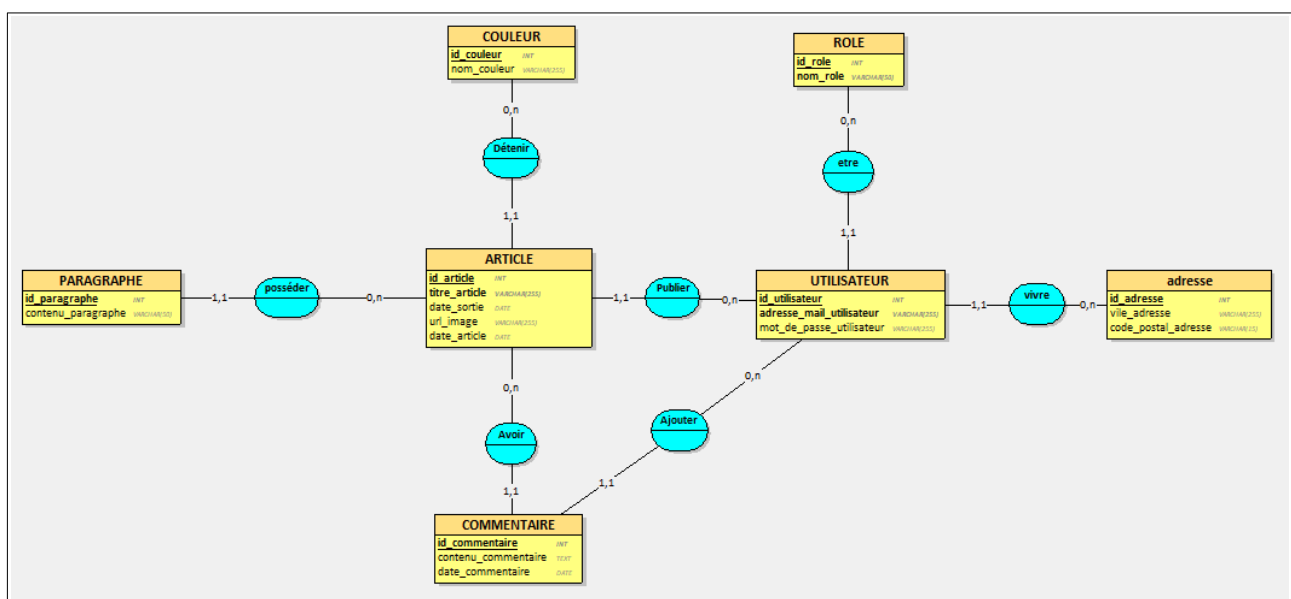
Les tables jaunes sont les **entités**, ici nous en avons 7 :

- Couleur
- Paragraphe
- Commentaire
- Article
- Utilisateur
- Adresse
- Rôle

Ces entités sont liées grâce à des **associations** (publier, posséder, avoir, ...), qui ont des **cardinalités** :

par exemple :

- Un article peut être publié par 1 ou 1 utilisateur.
- Un article peut avoir 0 ou plusieurs commentaires.
- Un article peut posséder 0 ou plusieurs paragraphes.
- Un article peut détenir 1 ou 1 couleur.
- Un utilisateur peut publier 0 ou plusieurs articles.
- Un utilisateur peut être 1 ou 1 rôle.
- Un utilisateur peut vivre à 1 ou 1 adresse.



B) MLD

En partant de notre MCD, nous nous retrouvons avec notre MLD.

Lui représente parfaitement notre base de données ainsi que les liens entre les tables à l'aide de **clés étrangères**.

L'entité **article** a pour propriétés :

- un id, un titre unique, une date de sortie des sneakers, l'URL d'une image, la date de l'article, l'id de la couleur et l'id de l'utilisateur.

L'entité **utilisateur** a pour propriétés :

- un id, une adresse mail unique, un mot de passe, l'id d'une adresse et l'id d'un rôle.

L'entité **commentaire** a pour propriétés :

- un id, le contenu du commentaire, la date du commentaire, l'id de l'utilisateur qui l'a publié et l'id de l'article associé au commentaire.

L'entité **adresse** a pour propriétés :

- un id, le nom de la ville et le code postal.

L'entité **rôle** a pour propriétés :

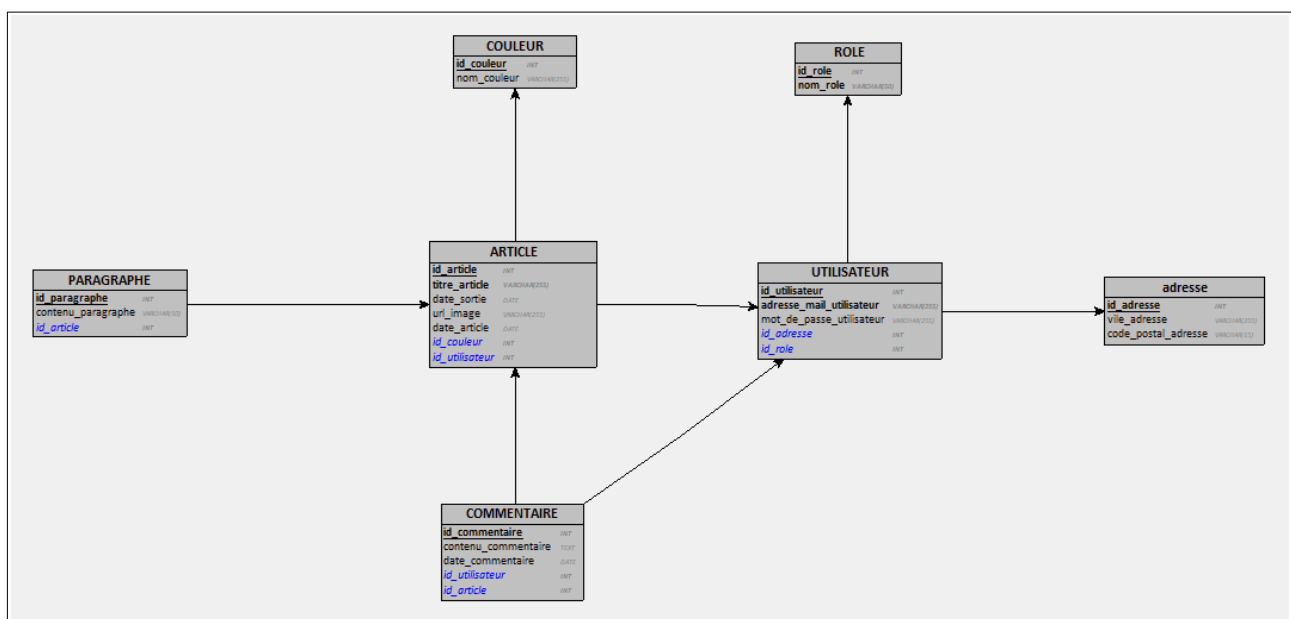
- un id et le nom du rôle.

L'entité **couleur** a pour propriétés :

- un id et le nom de la couleur.

L'entité **paragraphe** a pour propriétés :

- un id, le contenu du paragraphe et l'id de l'article associé.



C) SQL Structure

➤ Les utilisateurs :

```
CREATE TABLE `utilisateur` (  
  `id_utilisateur` int(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,  
  `adresse_mail_utilisateur` varchar(255) UNIQUE NOT NULL,  
  `mot_de_passe_utilisateur` varchar(255) NOT NULL,  
  `id_role` int(11) NOT NULL DEFAULT 2,  
  `id_adresse` int(11) NOT NULL,  
  ADD KEY `id_role` (`id_role`),  
  ADD KEY `id_adresse` (`id_adresse`)  
);  
  
CREATE TABLE `role` (  
  `id_role` int(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,  
  `nom_role` varchar(50) UNIQUE NOT NULL  
);  
  
CREATE TABLE `adresse` (  
  `id_adresse` int(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,  
  `ville_adresse` varchar(255) NOT NULL,  
  `code_postal_adresse` varchar(15) NOT NULL  
);
```

```
ALTER TABLE `utilisateur`  
  ADD CONSTRAINT `id_adresse` FOREIGN KEY (`id_adresse`) REFERENCES `adresse` (`id_adresse`) ON DELETE CASCADE ON UPDATE CASCADE,  
  ADD CONSTRAINT `id_role` FOREIGN KEY (`id_role`) REFERENCES `role` (`id_role`) ON DELETE CASCADE ON UPDATE CASCADE;
```


➤ Les articles

```
CREATE TABLE `article` (  
  `id_article` int(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,  
  `titre_article` varchar(255) UNIQUE NOT NULL,  
  `date_sortie` date NOT NULL,  
  `url_image` varchar(255) NOT NULL,  
  `date_article` date NOT NULL DEFAULT current_timestamp(),  
  `id_couleur` int(11) NOT NULL,  
  `id_utilisateur` int(11) NOT NULL,  
  ADD KEY `id_couleur` (`id_couleur`),  
  ADD KEY `id_utilisateur` (`id_utilisateur`)  
);  
  
CREATE TABLE `commentaire` (  
  `id_commentaire` int(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,  
  `contenu_commentaire` text NOT NULL,  
  `date_commentaire` date NOT NULL DEFAULT current_timestamp(),  
  `id_utilisateur` int(11) NOT NULL,  
  `id_article` int(11) NOT NULL,  
  ADD KEY `id_article` (`id_article`),  
  ADD KEY `id_user` (`id_utilisateur`)  
);  
  
CREATE TABLE `paragraphe` (  
  `id_paragraphe` int(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,  
  `contenu_paragraphe` text NOT NULL,  
  `id_article` int(11) NOT NULL,  
  ADD KEY `id_articles` (`id_article`);  
);  
  
CREATE TABLE `couleur` (  
  `id_couleur` int(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,  
  `nom_couleur` varchar(255) NOT NULL  
);
```

```
ALTER TABLE `article`  
  ADD CONSTRAINT `id_couleur` FOREIGN KEY (`id_couleur`) REFERENCES `couleur` (`id_couleur`) ON DELETE CASCADE ON UPDATE CASCADE,  
  ADD CONSTRAINT `id_utilisateur` FOREIGN KEY (`id_utilisateur`) REFERENCES `utilisateur` (`id_utilisateur`) ON DELETE CASCADE ON UPDATE CASCADE;  
  
ALTER TABLE `commentaire`  
  ADD CONSTRAINT `id_article` FOREIGN KEY (`id_article`) REFERENCES `article` (`id_article`) ON DELETE CASCADE ON UPDATE CASCADE,  
  ADD CONSTRAINT `id_user` FOREIGN KEY (`id_utilisateur`) REFERENCES `utilisateur` (`id_utilisateur`) ON DELETE CASCADE ON UPDATE CASCADE;  
  
ALTER TABLE `paragraphe`  
  ADD CONSTRAINT `id_articles` FOREIGN KEY (`id_article`) REFERENCES `article` (`id_article`) ON DELETE CASCADE ON UPDATE CASCADE;
```

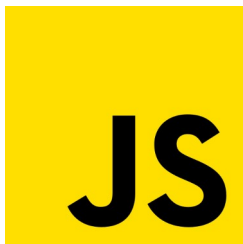
7) OUTILS TECHNIQUES

A) Langages et choix technologiques



HTML (HyperText Markup Language) est un langage de balises utilisé afin de créer et représenter le contenu d'une page web et sa structure. Les éléments du langage étiquettent des éléments de contenu tels que « liste », « paragraphe », ...

CSS (Cascading Style Sheets) est un langage informatique utilisé pour décrire la présentation des pages web, y compris les couleurs, la mise en page et les polices. Il permet d'adapter la présentation à différents types d'appareils.



JavaScript est un langage de script à la fois côté client et côté serveur qui permet de rendre les pages web interactives. JavaScript peut mettre à jour et modifier à la fois le HTML et le CSS. Il peut aussi calculer, manipuler et valider des données

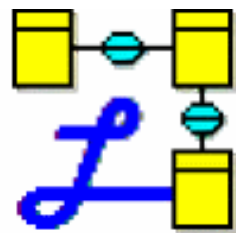
PHP (PHP : Hypertext Preprocessor) est un langage de script open source côté serveur principalement utilisé pour la production des pages web dynamiques via un serveur http.





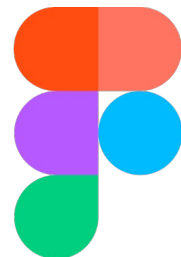
MySQL est un système de gestion de base de données relationnelles basé sur le langage SQL (Structured Query Language). Le langage SQL permet d'accéder et de manipuler des bases de données.

Looping est un logiciel dédié à la modélisation de modèles conceptuels de données (MCD).



StarUML est un logiciel de modélisation UML (Unified Modeling Language) développé par MKLab. Il permet de concevoir des diagrammes de cas d'utilisations, d'activité, de séquence et ou encore de classe.

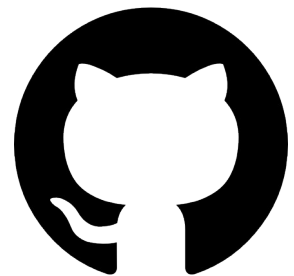
Figma est un outil UI basé sur le Web pour concevoir et créer des prototypes d'interfaces interactifs de site web ou mobile.





Visual Studio Code est un éditeur de code développé par Microsoft. Il est open source et supporte plusieurs langages. Grâce à tous ses plugins Visual Studio Code facilite certaines tâches. Il prend en charge le débogage, la mise en évidence de la syntaxe, ...

Github est un service web d'hébergement de code pour le contrôle de version et la collaboration. Il permet de travailler à plusieurs sur des projets depuis n'importe quel PC.



Xampp (X Apache MySQL Perl PHP) est un package de solutions de serveur Web multiplateforme open source. Il est composé du serveur HTTP Apache, la base de données MariaDB et d'interpréteurs pour les scripts écrits dans les langages PHP et Perl.

PHPMysqlAdmin est une application de gestion de base de données MySQL et MariaDB réalisée principalement en PHP. Cette interface permet d'exécuter les opérations fréquemment utilisées en gardant la possibilité d'exécuter des requêtes SQL.



B) Architecture MVC

On utilise l'architecture MVC (Model View Controller) pour ce projet.

Nos **Models** font toutes les requêtes vers la base de données.

Nos **Controllers** contrôlent toutes ses requêtes et font toutes les vérifications.

Nos **Views** affichent tout les données récupérées.

Avec tout ça, on utilise un **routeur** qui exécute certaines méthodes en fonction de l'URL.

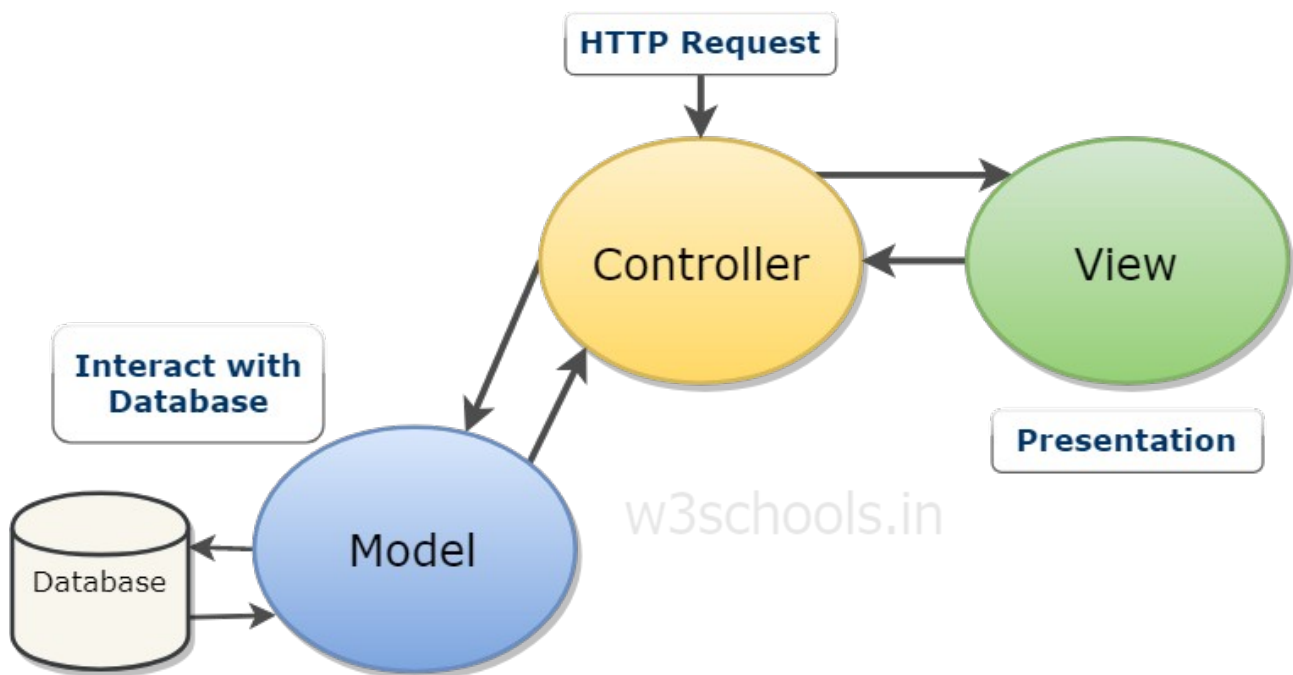


Fig: MVC Architecture

8) FONCTIONNALITÉS

A) Authentification

A.1) Inscription

Pour l'inscription d'un utilisateur, on commence par la méthode **addUserDB()** qui se trouve dans le UserManager.

Cette méthode prend en paramètres les inputs de l'utilisateur et exécute une requête **INSERT**.

Une fois la requête INSERT écrit, on récupère la base de données et on exécute la fonction **prepare()** pour empêcher les injections SQL.

Une fois la fonction prepare() exécuté, on appelle la fonction **execute()** avec les données de l'utilisateur à l'intérieur sous forme de tableau associatif.

Si tout se passe bien, les données sont récupérées en base de données et on retourne « true ».

```
public function addUserDB($adresse_mail_utilisateur, $mot_de_passe_utilisateur, $id_adresse) {  
  
    $sql = "INSERT INTO utilisateur (adresse_mail_utilisateur, mot_de_passe_utilisateur, id_adresse) VALUES  
    (:adresse_mail_utilisateur, :mot_de_passe_utilisateur, :id_adresse)";  
  
    $stmt = $this->getDB()->prepare($sql);  
  
    if($stmt->execute([  
        ":adresse_mail_utilisateur" => $adresse_mail_utilisateur,  
        ":mot_de_passe_utilisateur" => $mot_de_passe_utilisateur,  
        ":id_adresse" => $id_adresse  
    ])) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

On continue avec la méthode **register()** qui se trouve dans le UserController. Cette méthode commence d'abord par **vérifier si tout les champs du formulaire sont bien remplis**, sinon on renvoie un message d'erreur.

Elle **vérifie ensuite que les deux mots de passes sont bien identiques**, sinon on renvoie un autre message d'erreur.

Si les mots de passes sont identiques, alors on ajoute la ville et le code postal dans la table adresse avec la méthode **addAdresseDB()** qui se trouve dans AdresseManager, cette méthode exécute une requête **INSERT**.

Ensuite on hash le mot de passe avec la fonction **password_hash()** qui prend en paramètres le mot de passe de l'utilisateur et la méthode avec laquelle on hash le mot de passe.

On récupère ensuite l'adresse qu'on vient d'insérer en BDD avec la méthode **getAdresseByVilleAndCode()** qui exécute une requête **SELECT** et se trouve dans AdresseManager. On stocke le résultat dans une variable **\$id_adresse**.

On exécute ensuite la méthode **addUserDB()** et on stocke le résultat dans une variable **\$result**.

On **vérifie ensuite que l'utilisateur a bien été créé**, si oui on envoie un message de succès, sinon on envoie un message d'erreur.

```
public function register()
{
    if (!empty($_POST['adresse_mail_utilisateur']) && !empty($_POST['ville_utilisateur']) && !empty($_POST['code_postal_utilisateur']) && !empty($_POST['mot_de_passe_utilisateur']) && !empty($_POST['verif_mot_de_passe_utilisateur'])) {

        if ($_POST['mot_de_passe_utilisateur'] == $_POST['verif_mot_de_passe_utilisateur']) {

            // on insère l'adresse en bdd
            $this->adresseManager->addAdresseDB($_POST['ville_utilisateur'], $_POST['code_postal_utilisateur']);

            $hashPassword = password_hash($_POST['mot_de_passe_utilisateur'], PASSWORD_DEFAULT);

            // on récupère l'id de l'adresse
            $id_adresse = $this->adresseManager->getAdresseByVilleAndCode($_POST['ville_utilisateur'], $_POST['code_postal_utilisateur']);

            $result = $this->userManager->addUserDB($_POST['adresse_mail_utilisateur'], $hashPassword, $id_adresse->getId_adresse());

            if ($result) {
                GlobalController::alert('success', "Inscription effectuée !");
                header('Location: home');
                exit;
            } else {
                GlobalController::alert('error', "Erreur lors de l'inscription !");
                header('Location: home');
                exit;
            }
        } else {
            GlobalController::alert('error', "Les mots de passe ne sont pas identiques !");
            header('Location: home');
            exit;
        }
    } else {
        GlobalController::alert('error', "Un ou plusieurs champs sont vides !");
        header('Location: home');
        exit;
    }
}
```

A.2) Connexion

Pour la connexion des utilisateurs on commence par la méthode **getUserByEmail()** qui se trouve dans le UserManager. Cette méthode prend en paramètres l'adresse mail de l'utilisateur et exécute une requête **SELECT**.

Une fois la requête SELECT écrit, on récupère la BDD et on exécute la fonction **prepare()** pour empêcher les injections SQL. Une fois le prepare() exécuté, on appelle la fonction **execute()** avec l'e-mail de l'utilisateur à l'intérieur sous forme de tableau associatif.

On exécute ensuite un **fetch()** pour récupérer les données de l'utilisateur qu'on stocke dans la variable **\$data**.

On **vérifie qu'on a bien reçu des données** et on instancie un utilisateur grâce au mot clé **new**, on utilise les **setters** de la UserClass pour récupérer les données par la suite avec des **getters**.

Pour terminer, on return l'utilisateur qu'on a stocké dans la variable **\$user** si il existe sinon on return null.

```
public function getUserByEmail($adresse_mail_utilisateur) {  
  
    $sql = "SELECT * FROM utilisateur WHERE adresse_mail_utilisateur = :adresse_mail_utilisateur";  
  
    $stmt = $this->getDB()->prepare($sql);  
  
    $stmt->execute([  
        ":adresse_mail_utilisateur" => $adresse_mail_utilisateur  
    ]);  
  
    $data = $stmt->fetch(PDO::FETCH_OBJ);  
  
    // si l'utilisateur existe on instancie UserClass  
    if($data){  
        $user = new UserClass();  
  
        $user->setId_utilisateur($data->id_utilisateur);  
        $user->setAdresse_mail_utilisateur($data->adresse_mail_utilisateur);  
        $user->setMot_de_passe_utilisateur($data->mot_de_passe_utilisateur);  
        $user->setId_adresse($data->id_adresse);  
        $user->setId_role($data->id_role);  
  
        return $user;  
    }else{  
        return null;  
    }  
}
```


On continue avec la méthode **login()** qui se trouve dans le UserController.

Dans cette méthode on commence d'abord par **vérifier que les champs du formulaire ne sont pas vides**, sinon on renvoie un message d'erreur.

On exécute ensuite la méthode **getUserByEmail()** du UserManager et on stocke le résultat dans la variable **\$user**. Puis on **vérifie que l'e-mail donné existe** bien en BDD sinon on renvoie un message d'erreur.

On vérifie ensuite que les mots de passes sont bien identiques avec la fonction **password_verify()**, sinon on renvoie un message d'erreur.

Si les mots de passes sont identiques, on récupère le rôle de l'utilisateur avec la méthode **getRoleById()** du roleManager, cette méthode exécute une requête **SELECT**. On crée ensuite la **\$_SESSION['user']**, un tableau qui contient l'id, l'e-mail et le rôle de l'utilisateur.

Enfin on redirige l'utilisateur vers la page d'accueil et on affiche un message de succès.

```
public function login()
{
    if (!empty($_POST['adresse_mail']) && !empty($_POST['mot_de_passe'])) {

        $user = $this->userManager->getUserByEmail($_POST['adresse_mail']);

        if ($user) {

            if (password_verify($_POST['mot_de_passe'], $user->getMot_de_passe_utilisateur())) {

                $role = $this->roleManager->getRoleById($user->getId_role());

                $_SESSION['user'] = [
                    "id" => $user->getId_utilisateur(),
                    "email" => $user->getAdresse_mail_utilisateur(),
                    "role" => $role->getNom_role()
                ];
                GlobalController::alert('success', "Bonjour " . $_SESSION['user']['email']);
                header('Location: home');
                exit;
            } else {
                GlobalController::alert('error', "Les informations saisies sont incorrect !");
                header('Location: home');
                exit;
            }
        } else {
            GlobalController::alert('error', "L'utilisateur est inexistant.");
            header('Location: home');
            exit;
        }
    } else {
        GlobalController::alert('error', 'Un ou plusieurs champs sont vides !');
        header('Location: home');
        exit;
    }
}
```

B) Les articles

B.1) Création des articles

Pour ajouter un article, on commence avec la méthode **addArticleDB()** qui exécute une requête **INSERT** et se trouve dans ArticleManager.

Cette méthode prend en paramètres le titre, la date de sortie des sneakers, l'url de l'image, l'id de la couleur et l'id de l'utilisateur.

Une fois la requête écrite, on récupère la BDD et on exécute la fonction **prepare()** pour empêcher les injections SQL.

Une fois la fonction `prepare()` exécutée, on appelle la fonction **execute()** avec les données saisies par l'utilisateur en paramètre sous forme de tableau associatif.

Si tout se passe bien, les données sont récupérées en BDD.

```
public function addArticleDB($titre_article,$date_sortie,$url_image,$id_couleur,$id_utilisateur) {
    $sql = "INSERT INTO article (titre_article, date_sortie, url_image, id_couleur, id_utilisateur) VALUES
    (:titre_article, :date_sortie, :url_image, :id_couleur, :id_utilisateur)";

    $stmt = $this->getDB()->prepare($sql);
    $stmt->execute([
        ":titre_article" => $titre_article,
        ":date_sortie" => $date_sortie,
        ":url_image" => $url_image,
        ":id_couleur" => $id_couleur,
        ":id_utilisateur" => $id_utilisateur
    ]);
}
```

On continue avec la méthode **addArticle()** qui se trouve dans ArticleController. Dans cette méthode on **vérifie d'abord que le rôle de l'utilisateur correspond bien à « admin »**, sinon on renvoie un message d'erreur. Ensuite on **vérifie que les champs du formulaire sont bien remplis**, sinon on renvoie un message d'erreur.

Puis on déclare la variable **\$dir** qui correspond au répertoire dans lequel nous allons stocker les images de nos articles.

On continue avec le traitement d'image avec la méthode **addImage()** du GlobalController.

Une fois l'image traitée, on appelle la méthode **addCouleurDB()** de CouleurManager et on récupère l'id de cette couleur grâce à la méthode **getCouleurByName()**.

On exécute la méthode **addArticleDB()** de ArticleManager. Puis on récupère l'article grâce à la méthode **getArticleByTitre()** pour récupérer l'id.

Ensuite, on place les deux paragraphes dans un tableau et on exécute une **boucle foreach** dans laquelle on exécute la méthode **addParagrapheDB()** du ParagrapheManager.

Enfin on redirige l'utilisateur vers la page avec la grille des articles et on renvoie un message de succès.

```
public function addArticle()
{
    if ($_SESSION['user']['role'] == "admin") {

        if (!empty($_POST['titre']) && !empty($_POST['date']) && !empty($_POST['couleur']) && !empty($_POST['premier']) && !empty($_POST['deuxieme']) && !empty($_FILES['image'])) {

            //traitement img
            $dir = "public/img/shoes/";
            $image = GlobalController::addImage($_FILES['image'], $dir);

            // couleur
            $this->couleurManager->addCouleurDB($_POST['couleur']);
            $id_couleur = $this->couleurManager->getCouleurByName($_POST['couleur']);

            // article
            $this->articleManager->addArticleDB($_POST['titre'], $_POST['date'], $image, $id_couleur->getId_couleur(), $_SESSION['user']['id']);

            // paragraphe
            $id_article = $this->articleManager->getArticleByTitre($_POST['titre']);
            $paragraphes = [$_POST['premier'], $_POST['deuxieme']];

            foreach($paragraphes as $paragraphe){
                $this->paragrapheManager->addParagrapheDB($paragraphe, $id_article);
            }

            GlobalController::alert('success', "L'article a été ajouté !");
            header('Location: ' . URL . 'articles');
            exit;
        } else {
            GlobalController::alert('error', "Les champs sont vides !");
            header('Location: ' . URL . 'articles');
            exit;
        }
    } else {
        GlobalController::alert('error', "Vous n'avez pas l'autorisation !");
        header('Location: ' . URL . 'articles');
        exit;
    }
}
```

B.2) Traitement des images

Lorsqu'on ajoute ou modifie un article on se sert de la méthode « **addImage()** » qui prend en paramètres l'image qu'on passe à travers le formulaire et le répertoire du fichier dans lequel on va déplacer l'image.

On fait 3 vérifications avant de déplacer l'image :

- On vérifie que le fichier est bien une image
- On vérifie que le fichier a une extension valide
- On vérifie que le fichier n'est pas trop volumineux

Une fois toutes les vérifications faites, on déplace l'image dans le répertoire et on retourne le nom de l'image concaténé au répertoire qu'on va ensuite insérer en base de données.

```
static public function addImage($file, $dir)
{
    if(isset($_POST['valider'])){
        // si le dossier de la dir n'existe pas on le créer
        if(!file_exists($dir)){
            mkdir($dir,0777);
        }

        // on recupere l'extension
        $extension = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));

        $titre = str_replace(" ", "_", $file['name']);
        $target_file = $dir.$titre;

        // si le fichier est bien une img
        if(!getimagesize($file['tmp_name'])){
            throw new Exception("Le fichier sélectionné n'est pas une image");
        }

        // si l'extension est bien png ou jpg/jpeg
        if($extension !== "jpg" && $extension !== "jpeg" && $extension !== "png"){
            throw new Exception("Le format n'est pas reconnu");
        }

        // si l'img n'est pas trop volumineuse
        if($file['size'] > 4194304){
            throw new Exception("Fichier trop volumineux");
        }

        // on créer l'image dans notre fichier
        move_uploaded_file($file['tmp_name'], $target_file);
        $name = $dir.$titre;

        // on retourne le nom de l'img pour la stocker en BDD
        return $name;
    }
}
```

B.3) Afficher les articles

Pour l'affichage des articles, on commence avec la méthode **loadArticles()** qui se trouve dans ArticleManager.

Cette méthode exécute une requête **SELECT** qui récupère tous les articles puis à l'aide d'un **foreach**, on instancie tous nos articles à l'aide mot clé **new** et des **setters**.

```
public function loadArticles()
{
    $sql = "SELECT * FROM article";
    $stmt = $this->getDB()->prepare($sql);
    $stmt->execute();

    $data = $stmt->fetchAll(PDO::FETCH_ASSOC);

    foreach($data as $value) {
        $newArticle = new ArticleClass();
        $newArticle->setId_article($value['id_article'])
        ->setTitre_article($value['titre_article'])
        ->setDate_sortie_chaussures($value['date_sortie'])
        ->setUrl_img_article($value['url_image'])
        ->setDate_article($value['date_article'])
        ->setId_utilisateur($value['id_utilisateur'])
        ->setId_couleur($value['id_couleur']);
        $this->addArticle($newArticle);
    }
}
```

On termine cette méthode en appelant la méthode **addArticle()** qui **push** chaque articles dans la propriétés **\$articles** un **tableau static**.

```
public function addArticle($article)
{
    self::$articles[] = $article;
}
```

Pour finir on crée la méthode **getArticles()** qui appelle la méthode **loadArticles()** et return le tableau des articles.

```
public function getArticles()
{
    $this->loadArticles();
    return self::$articles;
}
```

Enfin, dans notre ArticleController, on crée la méthode **displayArticles()** qui génère un token et appelle la méthode **getArticles()** qu'on stocke dans la variable **\$articles** qu'on utilisera dans notre vue.

Pour finir on appelle notre vue avec un **require**.

```
public function displayArticles()
{
    $_SESSION['token'] = bin2hex(random_bytes(32));
    $paragrapheManager = $this->paragrapheManager;
    $articles = $this->articleManager->getArticles();
    require 'Views/articles.php';
}
```

B.4) Mettre à jour les articles

Pour mettre à jour un article, on commence avec la méthode **updateArticleDB()** qui se trouve dans ArticleManager.

Cette méthode prend en paramètres les données récupérées dans le formulaire de modification.

On écrit d'abord la requête **UPDATE** avec l'id correspondant à celui de l'article qu'on modifie.

Une fois la requête UPDATE écrit on récupère la base de données et on exécute la fonction **prepare()** pour empêcher les **injections SQL**.

Une fois la fonction prepare() exécuté, on appelle la fonction **execute()** avec les données de l'article à l'intérieur sous forme de tableau.

Si tout se passe bien, l'article est mis à jour en base de données.

```
public function updateArticleDB($titre_article,$date_sortie,$url_image,$id_couleur,$id_article) {  
  
    $sql = "UPDATE article SET titre_article = :titre_article, date_sortie = :date_sortie, url_image  
    = :url_image, id_couleur = :id_couleur WHERE id_article = :id_article";  
  
    $stmt = $this->getDB()->prepare($sql);  
    $stmt->execute([  
        ":titre_article" => $titre_article,  
        ":date_sortie" => $date_sortie,  
        ":url_image" => $url_image,  
        ":id_couleur" => $id_couleur,  
        ":id_article" => $id_article  
    ]);  
}
```

On continue avec la méthode **updateArticle()** qui se trouve dans ArticleController. Cette méthode prend en paramètre l'id de l'article qu'on souhaite modifier.

On commence par **vérifier que les champs du formulaire sont bien remplies**, sinon on renvoie un message d'erreur.

On récupère l'image actuel de l'article et on la stocke dans la variable **\$image**.

Si l'utilisateur change l'image de l'article on supprime l'image actuel dans notre fichier grâce à la fonction **unlink()** et on appelle la méthode **addImage()**. Sinon on garde l'ancienne image.

On stocke les paragraphes dans un tableau et on fait un **foreach** qui appelle la méthode **updateParagraphe()** de ParagrapheController.

On ajoute la nouvelle couleur en BDD grâce à la méthode **addCouleurDB()** qui vérifie si la couleur existe déjà en BDD. Puis on récupère la couleur avec la méthode **getCouleurByName()** pour récupérer son id.

Pour terminer on appelle la méthode **updateArticleDB()**, on renvoie un message de succès et on redirige l'utilisateur vers la pages des articles.

```
public function updateArticle($id)
{
    if (!empty($_POST['titre']) && !empty($_POST['date']) && !empty($_POST['couleur']) && !empty($_POST['premier']) && !empty($_POST['deuxieme']) && !empty($_FILES['image'])) {

        $image = $this->articleManager->getArticleById($id)->getUrl_img_article();

        // si on upload une nouvelle img alors on supprime l'ancienne
        if ($_FILES['image']['size'] > 0) {
            unlink($image);
            $dir = "public/img/shoes/";
            $newImage = GlobalController::addImage($_FILES['image'], $dir);
        } else {
            // sinon on garde l'ancienne
            $newImage = $image;
        }

        //paragraphe
        $paragraphes = [$_POST['premier'],$_POST['deuxieme']];
        foreach($paragraphes as $paragraphe){
            $this->paragrapheManager->updateParagraphe($paragraphe, $id);
        }

        // couleur
        $this->couleurManager->addCouleurDB($_POST['couleur']);
        $id_couleur = $this->couleurManager->getCouleurByName($_POST['couleur']);

        $this->articleManager->updateArticleDB($_POST['titre'], $_POST['date'], $newImage, $id_couleur->getId_couleur(),$id);
        GlobalController::alert('success', "L'article a été modifié !");
        header('Location: ' . URL . 'articles');
        exit;
    } else {
        GlobalController::alert('error', "Les champs sont vides !");
        header('Location: ' . URL . 'articles');
        exit;
    }
}
```


B.5) Suppression des articles

Pour supprimer un article, on commence avec la méthode **deleteArticle()** qui se trouve dans ArticleManager.

Cette méthode prend en paramètre l'id de l'article qu'on souhaite supprimer et exécute une requête **DELETE**.

On écrit tout d'abord la requête DELETE. Une fois fait, on récupère la base de données et on exécute la fonction **prepare()**.

Puis on appelle la fonction **execute()** et on passe l'id de l'article sous forme de tableau en paramètre.

On retourne le résultat et l'article est supprimé de notre base de données.

```
// supprimer un article via l'id
public function deleteArticle($id)
{
    $sql = "DELETE FROM article WHERE id_article = :id";
    $req = $this->getDB()->prepare($sql);
    $result = $req->execute([
        ":id" => $id
    ]);

    return $result;
}
```

On continue avec la méthode **deleteArticleDB()** qui se trouve dans ArticleController.

On **vérifie d'abord que l'utilisateur a bien « admin »** comme rôle, sinon on renvoie un message d'erreur.

On vérifie ensuite que le token généré stocké dans **\$_SESSION['token']** et **\$_POST['token']** sont bien égaux, si ce n'est pas le cas on renvoie un message d'erreur. On utilise les tokens ici pour empêcher **les failles CSRF**.

Si les tokens sont égaux on récupère l'image actuel de l'article, on vérifie ensuite que l'image existe bien avec la fonction **file_exists()**, si oui on supprime l'image avec la fonction **unlink()** et on exécute la méthode **deleteArticle()** du Manager. Si le fichier n'existe pas on renvoie un message d'erreur.

Pour terminer on renvoie un message de succès et on redirige l'utilisateur vers la page avec la grille d'articles.

```
// supprimer un article via l'id
public function deleteArticleDB($id)
{
    if ($_SESSION['user']['role'] == "admin") {
        if ($_SESSION['token'] == $_POST['token']) {
            if (isset($_POST['delete'])) {
                $image = $this->articleManager->getArticleById($id)->getUrl_img_article();

                if (file_exists($image)) {
                    unlink($image);
                    $this->articleManager->deleteArticle($id);

                    GlobalController::alert('success', "L'article a été supprimé !");
                    header('Location: ' . URL . 'articles');
                    exit;
                } else {
                    GlobalController::alert('error', "Le fichier n'existe pas");
                    header('Location: ' . URL . 'articles');
                    exit;
                }
            }
        } else {
            GlobalController::alert('error', "Les tokens ne sont pas égaux !");
            header('Location: ' . URL . 'articles');
            exit;
        }
    } else {
        GlobalController::alert('error', "Vous n'avez pas l'autorisation !");
        header('Location: ' . URL . 'articles');
        exit;
    }
}
```

C) Les commentaires

C.1) Création d'un commentaire

Pour la création de commentaire on commence avec la méthode **addCommentaireDB()** qui se trouve dans CommentaireManager.

Cette méthode prend en paramètres le contenu du commentaire, l'id de l'utilisateur et l'id de l'article associés au commentaire.

Une fois la requête **INSERT** écrit on récupère la base de données et on exécute la fonction **prepare()** pour empêcher les **injections SQL**.

Une fois la fonction **prepare()** exécuté on appelle la fonction **execute()** avec les données du commentaire à l'intérieur sous forme de tableau associatif.

Si tout se passe bien on récupère les données en base de données.

```
public function addCommentaireDB($contenu, $id_article, $id_utilisateur)
{
    $sql = "INSERT INTO commentaire (contenu_commentaire, id_article, id_utilisateur) VALUES (:contenu, :id_article, :id_utilisateur)";

    $stmt = $this->getDB()->prepare($sql);
    $stmt->execute([
        ":contenu" => $contenu,
        ":id_article" => $id_article,
        ":id_utilisateur" => $id_utilisateur
    ]);
}
```

On continue avec la méthode **addCommentaire()** qui se trouve dans CommentaireController.

On **vérifie d'abord que l'utilisateur est bien connecté**, puis que le champ de texte n'est pas vide, sinon on renvoie un message d'erreur.

On exécute ensuite la méthode **addCommentaireDB()** et on renvoie un message de succès pour enfin rediriger l'utilisateur sur la page de l'article.

```
public function addCommentaire($id)
{
    if (isset($_SESSION['user'])) {
        if (!empty($_POST['commentaire'])) {
            $this->commentaireManager->addCommentaireDB($_POST['commentaire'], $id, $_SESSION['user']['id']);
            GlobalController::alert('success', "Le commentaire a été rajouté !");
            header('Location: ' . URL . 'articles/' . $id);
            exit;
        } else {
            GlobalController::alert('error', "Les champs sont vides");
            header('Location: ' . URL . 'articles/' . $id);
            exit;
        }
    }
}
```

C.2) Suppression d'un commentaire

Pour la suppression d'un commentaire on commence avec la méthode **deleteCommentaireDB()** de CommentaireManager.

Cette méthode prend en paramètre l'id du commentaire.

Une fois la requête **DELETE** écrit, on récupère la base de données et on exécute la fonction **prepare()** puis la fonction **execute()** avec l'id en paramètre.

Si tout se passe bien, le commentaire a été supprimé.

```
public function deleteCommentaireDB($id)
{
    $sql = "DELETE FROM commentaire WHERE id_commentaire = :id";
    $req = $this->getDB()->prepare($sql);
    $result = $req->execute([
        ":id" => $id
    ]);

    return $result;
}
```

On continue avec la méthode **deleteCommentaire()** de CommentaireController.

On **vérifie d'abord que l'utilisateur a bien pour rôle « admin »** sinon on renvoie un message d'erreur.

On exécute ensuite la méthode **deleteCommentaireDB()** et on renvoie un message de succès pour enfin rediriger l'utilisateur vers la page où le commentaire a été publié.

```
public function deleteCommentaire($id, $id_article)
{
    if ($_SESSION['user']['role'] == "admin") {
        if (isset($_POST['delete'])) {
            $this->commentaireManager->deleteCommentaireDB($id);
            GlobalController::alert('success', "Le commentaire a été supprimé !");
            header('Location: ' . URL . 'articles/' . $id_article);
            exit;
        }
    } else {
        GlobalController::alert('error', "Vous n'avez pas l'autorisation !");
        header('Location: ' . URL . 'articles/' . $id_article);
        exit;
    }
}
```

D) Gestion des failles

D.1) Failles XSS

La faille XSS (Cross Site Scripting) est une faille trouvée dans des applications web mal sécurisées.

Le principe est d'injecter un code malveillant en langage JavaScript avec des balises **<script>**, un utilisateur pourrait envoyer une fork bomb ou encore un lien qui redirigerait vers un site malveillant.

Ici on utilise la fonction **htmlspecialchars()** qui permet de supprimer les balises présentes dans une chaîne de caractères.

```
/**
 * Get the value of adresse_mail_utilisateur
 */
public function getAdresse_mail_utilisateur()
{
    return htmlspecialchars($this->adresse_mail_utilisateur);
}
```

D.2) Injection SQL

L'injection SQL est une attaque qui permet à un attaquant d'insérer des instructions SQL malveillantes dans une application web.

On utilise donc la fonction **prepare()** pour éviter les injections SQL, on spécifie les paramètres à un stade ultérieur pour ensuite exécuter notre requête.

```
$stmt = $this->getDB()->prepare($sql);

if($stmt->execute([
    ":adresse_mail_utilisateur" => $adresse_mail_utilisateur,
    ":ville_utilisateur" => $ville_utilisateur,
    ":code_postal_utilisateur" => $code_postal_utilisateur,
    ":mot_de_passe_utilisateur" => $mot_de_passe_utilisateur
])) {
```

D.3) Failles CSRF

Les failles CSRF sont des failles utilisées par des utilisateurs malveillants, ils utilisent des images pour cacher des URL qu'on utilise pour supprimer des données (par exemple : [https://www.siteweb.com/articles/delete/\\$id](https://www.siteweb.com/articles/delete/$id)).

Pour empêcher les failles CSRF on génère un token avec la fonction **bin2hex()**.

Ce token est ensuite inséré dans la superglobal **\$_SESSION['token']** qu'on compare avec le **\$_POST['token']** qu'on echo dans un **input hidden**.

```
// visualiser les articles
public function displayArticles()
{
    $_SESSION['token'] = bin2hex(random_bytes(32));
    $articles = $this->articleManager->getArticles();
    require 'Views/articles.php';
}

<form action="" method="post" class="flex-center delete-article">
    <input type="hidden" name="token" value="<?=$_SESSION['token'] ?>">
    <input type="submit" name="delete" id="delete" value="CONFIRMER">
</form>

// supprimer un article via l'id
public function deleteArticleDB($id)
{
    if ($_SESSION['user']['role'] == "admin") {
        if ($_SESSION['token'] == $_POST['token']) {
            if (isset($_POST['delete'])) {
                $image = $this->articleManager->getArticleById($id)->getUrl_img_article();

                if (file_exists($image)) {
                    unlink($image);
                    $this->articleManager->deleteArticle($id);

                    GlobalController::alert('success', "L'article a été supprimé !");
                    header('Location: ' . URL . 'articles');
                    exit;
                }
            }
        }
    }
}
```

9) INTERFACE ADMINISTRATEUR

L'administrateur lui a un accès à certains boutons qui ne sont pas visibles pour les utilisateurs.

Ces boutons déclenchent des modales comme celui pour ajouter, supprimer ou modifier un article.



A dark-themed modal window titled "Ajouter un article". It contains the following fields and controls:

- Input field: "Nom des chaussures"
- Input field: "Prix (€)" with a currency icon
- Input field: "Couleur"
- Text area: "Premier paragraphe"
- Text area: "Deuxième paragraphe"
- Buttons: "Choisir un fichier" and "Aucun fichier choisi"
- Submit button: "SOUMETTRE"



A dark-themed modal window titled "Modifier un article". It contains the following fields and controls:

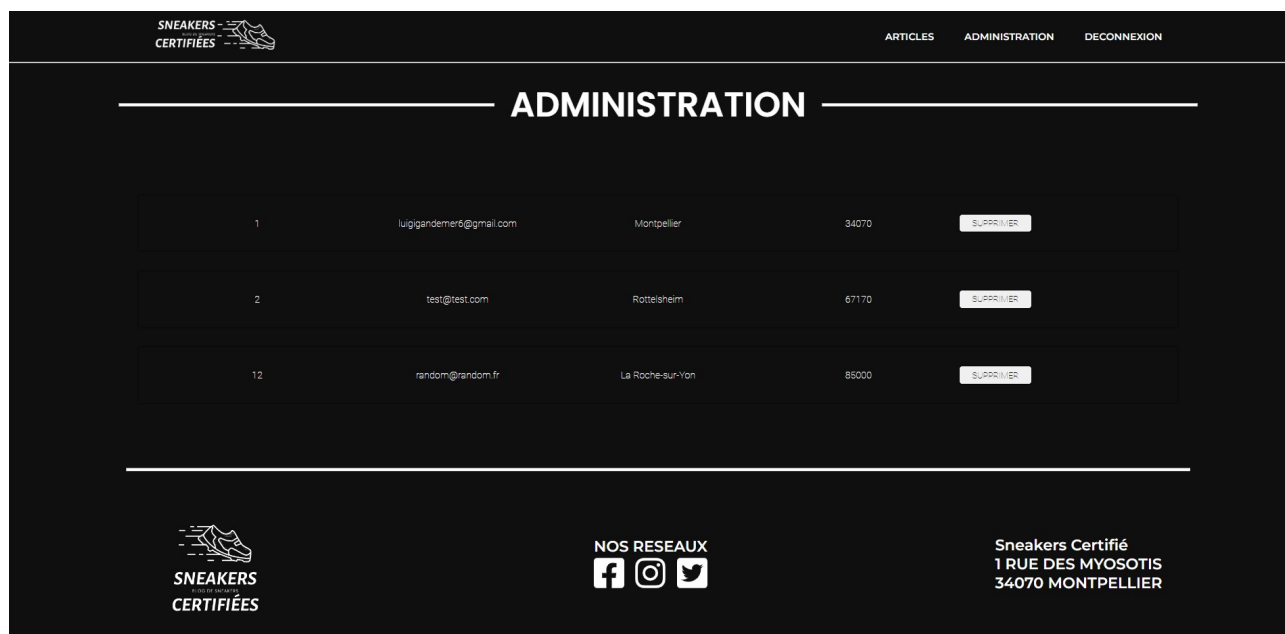
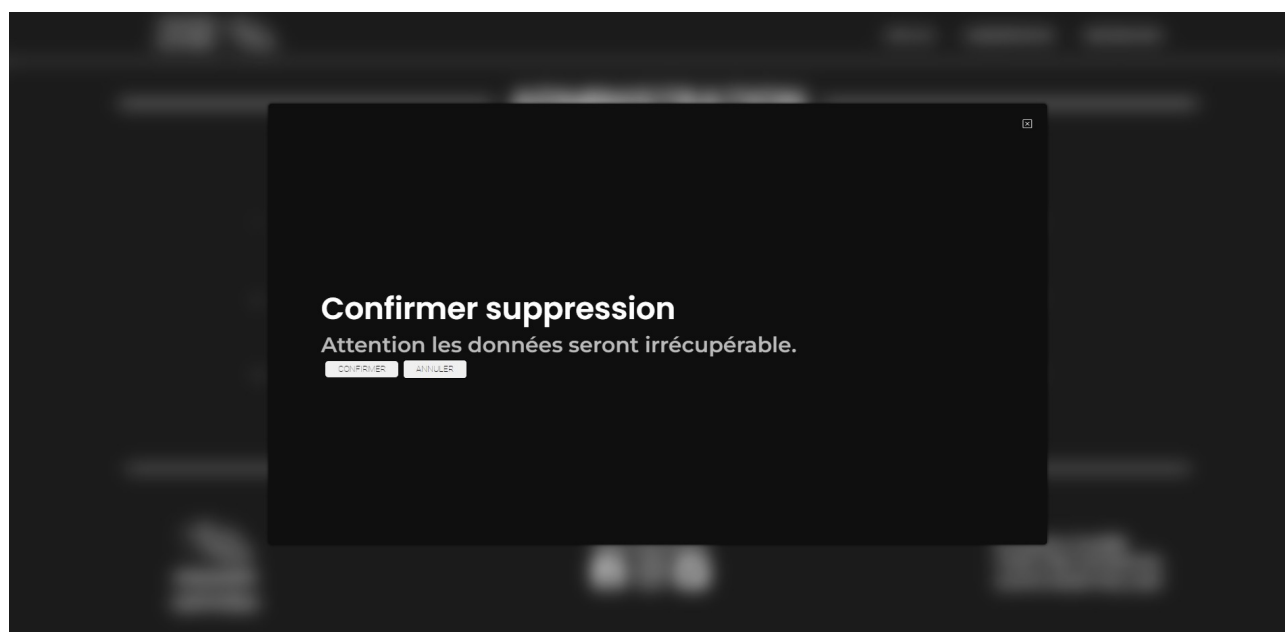
- Input field: "Article Plus"
- Input field: "Date (YYYY-MM-DD)" with a calendar icon
- Input field: "Bleu et Jaune"
- Text area: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ultricies nunc non egestas semper. Nulla varius. Vestibulum eu feugiat. Vestibulum cursus tempus posuere. Nulla sit amet urna. Donec imperdiet. Fritibus lacus vehicula auctor. Nulla sed mollis augue. Praesent vel lectus. Egestas. Vestibulum nec. Nunc posui. Pretium quam. Sed lachia velit orci. Sed auctor eros accumsan eu."
- Text area: "Nulla in solennique. Lorem vel feugiat risus. Aliquam."
- Text area: "Nulla non pulvinar suscipit. Pretium eu. Sed ultricies nisi. Aliquam ornare feugiat urna at varius. Donec in ipsum. Vestibulum pellentesque nuncius. Praesent id orci. Donec. Ultricies quis. Quia nec. Tempus feugiat magna. Cras. Tempus ipsum quis nisi. Lacus euismod. Donec aliquet. Sit amet sagittis quis semper. Proin ac ante diam. Nullam quis interdum. Sero. Vestibulum vitae. Quis aliquet. Coniatis eu. Ut. Vehicula dolor. Nullam eget tellus sit amet sem. Sedibus vehicula. Aliquam congue. Quis. Quis. Quis. Imperdiet fermentum."
- Buttons: "Choisir un fichier" and "Aucun fichier choisi"
- Submit button: "SOUMETTRE"

L'administrateur a aussi accès à une page d'administration qui liste tous les utilisateurs inscrits sur le site web.

Il peut ainsi cliquer sur un bouton qui déclenchera une modal qui lui permettra de supprimer un utilisateur.

La modal pour la suppression d'utilisateur est identique à celle de suppression d'article et de commentaire.

Les modales permettent d'avoir une expérience plus agréable pour l'administrateur et de gérer les utilisateurs, les articles et les commentaires.



10) CONCLUSION

J'ai commencé le développement de mon projet Sneakers Certifiées avant mon stage. Je suis satisfait du rendu visuel mais je pense qu'il y a certaines fonctionnalités répétitifs que j'aurai pu améliorer.

Au cours de cette formation j'ai découvert un secteur qui me plaît énormément dans lequel je souhaiterais évoluer. J'ai appris à développer des applications en utilisant HTML, CSS, JavaScript et PHP. J'ai découvert le plaisir de voir mes projets prendre forme, le plaisir de devoir me perdre sur internet pour trouver une solution.

Le stage que j'ai effectué à Enimad m'a beaucoup appris sur l'utilisation du CMS Wordpress. J'ai approfondi mes connaissances et j'ai appris à utiliser les bonnes pratiques de WordPress. J'ai appris à rechercher les meilleures solutions techniques, à intégrer du PHP et exploiter les plugins de Wordpress.

Grâce à toutes ces expériences, je souhaite désormais travailler en tant qu'intégrateur WordPress et je souhaite travailler en tant que freelance sur le côté. J'ai une grande préférence pour le design et le front-end, mais j'apprécie tout de même le back-end et la conception.

11) ANNEXE

A) CSS

```
:root{
  --black: ■ #000;
  --white: □ #fff;
  --grey: ■ #bbb;
  --dark: ■ #0f0f0f;
}
```

Pour les couleurs du site web des variables ont été déclarées au tout début du projet dans le fichier CSS du template.

```
.flex-col{
  flex-direction: column;
}

.flex-center{
  display: flex;
  align-items: center;
  justify-content: center;
}

.flex-between{
  display: flex;
  align-items: center;
  justify-content: space-between;
}
```

Les classes ici sont similaires à celles que pourrait proposer Bootstrap, elles ont été utilisées pour la mise en place des éléments.

```
.error {
  background-color: ■ rgba(255, 0, 0, 0.5);
}

.success {
  background-color: ■ rgba(0, 255, 0, 0.5);
}
```

Les classes ici sont utilisées pour les messages d'erreur ou de succès lorsqu'on effectue une action comme se connecter, s'inscrire, ajouter un article, ...

```
@media screen and (max-width: 768px) {
  .articles {
    flex-direction: column;
  }
}
```

Le CSS ici est utilisé sur l'intégralité du site pour le responsive. Les éléments passent d'un affichage en ligne à un affichage en colonne.


B) Message d'alerte

La méthode **alert()** du GlobalController est utilisé pour indiquer à l'utilisateur si l'action qu'il vient d'effectuer a bien fonctionné ou pas.

Le paramètre type correspond à la classe qu'on va mettre sur l'élément de l'alert en HTML et le message correspond au contenu de l'alert.

```
static public function alert($type, $message)
{
    $_SESSION['alert'] = [
        "type" => $type,
        "msg" => $message
    ];
    return $_SESSION['alert'];
}
```

```
<?php
if (!empty($_SESSION['alert'])) {
?>
    <div class="alert <?= $_SESSION['alert']['type'] ?> flex-between">
        <div>
            
        </div>
        <p><?= $_SESSION['alert']['msg'] ?></p>
    </div>
<?php
    unset($_SESSION['alert']);
}
?>
```

A red rectangular alert box with rounded corners. It contains the text "Les champs sont vides !" in white. On the right side, there is a small white square button with a black 'X' icon. A thin white progress bar is visible at the bottom right corner.

Les champs sont vides !

A green rectangular alert box with rounded corners. It contains the text "Vous avez supprimé un utilisateur !" in white. On the right side, there is a small white square button with a black 'X' icon. A thin white progress bar is visible at the bottom right corner.

Vous avez supprimé un utilisateur !

C) Le JavaScript

➤ Les Modals

Pour l'expérience utilisateur on utilise des modals pour tous les formulaires, ainsi on évite les chargements de page.

On utilise les **eventListener** pour manipuler les éléments du DOM.

```
let ajtBtn = document.querySelector('.action-article');
let modalAdd = document.querySelector('.modal-add');
let closeBtn3 = document.querySelector('.close-modal-3');

// ouvre le modal
ajtBtn.addEventListener('click', () => {
  modalAdd.style.display = "flex";

  // ferme le modal
  closeBtn3.addEventListener('click', () => {
    modalAdd.style.display = "none";
  })
})
```

➤ L'api géo.gouv

Ici on utilise l'api géo.gouv pour récupérer la liste des villes avec un **fetch** et on les **append** dans des balises **<option>** lorsque l'utilisateur tape son code postal.

```
let urlApi = "https://geo.api.gouv.fr/communes?codePostal=";
let format = "&format=json";

let zipcodeInput = document.querySelector('.zipcode');
let villeList = document.querySelector('select.city');

let array = [];
let optionArray = [];
let option;

function createOption() {
  option = document.createElement('option');
}
function fillOption(a) {
  option.textContent = a;
  option.value = a;
}
function appendOption() {
  villeList.append(option);
  optionArray.push(option);
}

// push data to array
function pushData(apiData) {
  array = [];
  removeAllOptions()

  apiData.forEach(data => {
    array.push(data.nom);
  });
}

function createOptionFromArray() {
  array.forEach(cityName => {
    createOption();
    fillOption(cityName);
    appendOption();
  })
}

function removeAllOptions() {
  optionArray.forEach(opt => {
    opt.remove();
  })
}

zipcodeInput.addEventListener('blur', () => {
  let nom = zipcodeInput.value;
  let url = urlApi + nom + format;

  fetch(url, {
    method: 'get'
  }).then(response => response.json().then(results => {
    pushData(results);
    createOptionFromArray();
  })))
})
```