Projet de Blog Communautaire de Partage d’Emotions Positive

Dev 2022  
Version 1.0

Romain Soulié

Table des matières

[Introduction au projet - 4 -](#_Toc112851165)

[Remerciements : - 5 -](#_Toc112851166)

[Abstract : - 6 -](#_Toc112851167)

[Introduction : - 7 -](#_Toc112851168)

[I - Analyse du besoin - 9 -](#_Toc112851169)

[1 - Présentation de l’entreprise : - 10 -](#_Toc112851170)

[2 - Contexte : - 10 -](#_Toc112851171)

[a) Objectif du site : - 10 -](#_Toc112851172)

[b) Les cibles : - 11 -](#_Toc112851173)

[c) Analyse de la concurrence : - 12 -](#_Toc112851174)

[d) Analyse marketing - 12 -](#_Toc112851175)

[2 - Définition du besoin : - 13 -](#_Toc112851176)

[a) Analyse de l’existant : - 13 -](#_Toc112851177)

[b) Enoncé du besoin : - 13 -](#_Toc112851178)

[c) Les objectifs quantitatifs : - 14 -](#_Toc112851179)

[d) Fonctionnalités du produit : - 14 -](#_Toc112851180)

[5) Contraintes : - 14 -](#_Toc112851181)

[a) Contraintes techniques : - 14 -](#_Toc112851182)

[b) Contraintes légales et réglementaires : - 15 -](#_Toc112851183)

[c) Contraintes de coûts - 15 -](#_Toc112851184)

[d) Contrainte de délais - 15 -](#_Toc112851185)

[II – Spécifications Fonctionnelles - 16 -](#_Toc112851186)

[1 – Les cas d'utilisations (Use Case) - 17 -](#_Toc112851187)

[a- Les fonctionnalités - 18 -](#_Toc112851188)

[b- Les acteurs - 18 -](#_Toc112851189)

[2 - Les diagrammes d'activités - 20 -](#_Toc112851190)

[a - Création d'un article par un utilisateur : - 20 -](#_Toc112851191)

[b - Créer une vidéothèque - 22 -](#_Toc112851192)

[c - Répondre à un Quizz - 23 -](#_Toc112851193)

[3 - Les diagrammes de séquences - 25 -](#_Toc112851194)

[a - Créer un article - 25 -](#_Toc112851195)

[C – Répondre à un Quizz - 26 -](#_Toc112851196)

[III - Maquettage et Arborescence - 27 -](#_Toc112851197)

[1 - L'arborescence - 28 -](#_Toc112851198)

[2 – Le maquettage - 29 -](#_Toc112851199)

[a – Le wireframe - 29 -](#_Toc112851200)

[b – Graphisme & Charte - 30 -](#_Toc112851201)

[c – Le Mock-up (le prototype) - 32 -](#_Toc112851202)

[IV – Conception - 33 -](#_Toc112851203)

[1 – La base de données - 34 -](#_Toc112851204)

[a – MCD - 34 -](#_Toc112851205)

[b - MLD - 36 -](#_Toc112851206)

[V – Réalisation - 41 -](#_Toc112851207)

[1 - Choix des outils techniques - 42 -](#_Toc112851208)

[a - Les langages de programmations - 42 -](#_Toc112851209)

[- 42 -](#_Toc112851210)

[b - Les Frameworks - 42 -](#_Toc112851211)

[- 42 -](#_Toc112851212)

[c – Logiciels et autres outils - 42 -](#_Toc112851213)

[2 – L'arborescence du projet - 43 -](#_Toc112851214)

[a- Django et son architecture - 43 -](#_Toc112851215)

[- 43 -](#_Toc112851216)

[b – Explication de l'arborescence du projet - 44 -](#_Toc112851217)

[3 – Fonctionnalités - 46 -](#_Toc112851218)

[A – L'inscription - 46 -](#_Toc112851219)

[B – Commenter un article - 52 -](#_Toc112851220)

[**"Templates**¶ - 56 -](#_Toc112851221)

[VI - Conclusion - 58 -](#_Toc112851222)

# Introduction au projet

## Remerciements :

Je tenais à remercier le Pôle Numérique de l’Adrar qui m’a donné la chance d’être formé au métier du développement web et d’applications, pendant cette année très enrichissante et ce, sur bien des niveaux. Cela m’a permis de changer la perception que je pouvais avoir du milieu professionnel de l’IT, ainsi que du schéma intellectuel qu’il faut acquérir pour mieux concevoir, élaborer et enfin, coder à proprement parler.  
  
Je remercie tout particulièrement les formateurs qui ont su nous insuffler la motivation, et l’énergie nécessaire dans les moments de doutes.

## Abstract :

The Positivum project is based on the idea that opinions on the Internet are very often the result of a bad experience specific to each person, and most of the time, totally unjustified. It is true that today, before buying a product, going to see a movie in the theatre, eating in a restaurant, but also travelling, we automatically check the reviews on websites such as Tripadvisor, SensCritique and others of the same kind.

Of course, we select according to these opinions, depending on our personality, but also on what we are really looking for as an experience, and what is worth about it. I started from the opposite intellectual path: we all want to have the best possible experience in everything we do that is a bit unusual or exceptional. In facts, what we remenber from an experience comes from the emotions we have lived. So, why not take 27 psychological emotions as a reference point and link them to our experiences, and our desire to share them in a constructive and positive way with others.  
  
The first step is to set the limits of the topics covered. This was to be open-ended in the original project, but the decision was made with the trainers, given the scale of the task, to start with film industry and all that revolves around it. This is already an exciting topic. Later on, i will consider to extend it to other disciplines.   
  
The site was to be presented as a blog, proposing articles that a community of authors would submit to the readers, in order to share their positive constructive criticism of films, actors, theatres and anything else related to the film industry. But, from a blog, I wanted it to transform over time into a real social network that attracts enthusiasts who want or know how to listen to their emotions and share them.  
  
Technically, this is much larger to create than a blog, and the time available was not enough for me to implement all the features, but the project is still in progress and is far from being finished.

## Introduction :

C’est en partant d’un constat que l’on a tous fait, à un moment où un autre au détour d’un article cinglant, ou d’un commentaire injuste sur tout et n’importe quoi : pourquoi le net est-il si méchant !  
  
N’est-il pas plus humain de ne faire que des éloges sur certaines petites choses, qui, par une expérience inoubliable, nous a transporté vers des émotions que nous avons encore du mal à réprimer quant au souvenir de l’œuvre, de l’artiste ou de l’image que cela laisse en nous.  
  
Le projet par de ce postulat, qu’un site dithyrambique quand à son contenu ne pouvais qu’enrichir la toile. Cela peut sembler étrange de par le fait que ce ne sera pas vraiment un site de critique tel que « Senscritique » par exemple.

L’idée de départ n’est pas de partir d’une œuvre afin de la critiquer, mais bien de démarrer sur des émotions ressenties lors d’une expérience vécue. Dès le début du projet, l’idée était de traiter tous les sujets pouvant susciter chez n’importe quel individu une émotion, qu’elle qu’elle soit, et de la partager. Mais, de par la complexité de la tâche, j’ai dû me résoudre à ne traiter que d’œuvres visuelles telle que le cinéma et tout l’univers qui gravite autour, pour le moment. Le concept se prête, ensuite, à tous les domaines qui nous apportent des émotions par l’expérience vécu : restaurants, voyages, visites de lieux, spectacles vivants etc



# I - Analyse du besoin

## 1 - Présentation de l’entreprise :

Ce site internet est basé sur un projet personnel, développé tout le long de l’année, et incrémenté au fur et à mesure, des savoirs et connaissances acquis lors de cette année de formation. Il n’y a donc pas d’études fondées sur une entreprise réelle, et le cahier des charges n’est que le reflet des contraintes techniques et de l’application d’un cursus nouvellement appliqué. Il est à noter, que malgré un planning qui me semblait, pour le moins, assez favorable, je n’ai pas du tout entrevue l’ampleur de la tâche, au vue de mes compétences. De plus, j'ai mis en œuvre des technologies nouvellement apprises et donc l'acquisition a été longue et fastidieuse.

## 2 - Contexte :

### a) Objectif du site :

L’objectif principal de ce site internet est donc de poser objectivement et dans la pratique les fondamentaux de la création en Front-end et Back-end d’un blog communautaire de partage d'expériences positives sur la passion du cinéma, et de tous ses supports numériques.   
  
Le but est de pouvoir partager socialement des émotions positives à travers des passions, comme dans la vie réelle. Le site ne se concentre que sur le ressenti d’un auteur après avoir été marqué par une œuvre, en l'occurrence ici, audiovisuelle et/ou cinématographique, ainsi que par les artisans qui ont aidé à leur conception : que ce soit un réalisateur, metteur en scène, mais tout autant un compositeur, ou bien même un directeur de la photographie.

Pour un utilisateur lambda qui passe sur le site et qui n’est pas contributeur, sa visite lui permettra d’avoir des critiques constructives, sans notation subjective et négative d’une œuvre, afin de ne pas perdre du temps à regarder, et à rechercher une œuvre qui ne lui correspondrait pas. Le visiteur peut rechercher un film ou une série selon un classement des émotions qu’il procure, et non par une note technique. Bien entendu, à termes, il pourra mettre des commentaires, qui seront modérés pour que l'expérience des autres utilisateurs ne soient pas polluer par des retours négatifs, ce qui n'est pas dans la mouvance de ce site.

Cela permet de ne plus mettre au centre d’un art, la technique, et/ou le support, mais bel et bien l'œuvre en elle-même.

En termes quantitatifs, le site permettrait aux utilisateurs de contribuer, et d’enrichir le site par de nouvelles œuvres dont ils ont particulièrement apprécié le ressenti au visionnage. Par de là, proposer au même profil d’utilisateur de créer à la fois des rencontres, et des échanges autours des personnalités au profil identique. Cela dirige, in fine, l'utilisateur vers des œuvres qu’il pourrait apprécier après étude de son profil.

Le projet était, au démarrage prévu, pour partager, autour de toute expérience culturelle, quelle qu’elle soit : cinémas, littératures, musée, voyages, mais aussi sports, gastronomie, et tout ce qui peut composer une expérience enrichissante et nourrissante.

Il a été décidé, vu l’ampleur de la tâche, de, pour le moment, resserrer le sujet sur l’audiovisuel Entertainment (film, série et tout l’univers qui tourne autour.)

### b) Les cibles :

Les visiteurs ciblés sont des personnes, qui, recherche la qualité en premier lieu, dans un film ou une série, et ne veulent plus perdre du temps à regarder des œuvres qui, ne leur procurera aucune émotion. Ce sont des individus qui ont conscience que la répercussion de l’ensemble de leur émotion, crée le vécu, l’expérience ainsi que les souvenirs. Ils peuvent ainsi éviter à la fois les traumatismes, mais aussi enrichir leur palette émotionnelle.

Nous visons donc des adultes, consacrant du temps à la recherche de la qualité, avant la quantité et qui aime partager leur expérience émotionnelle. Ce sera donc principalement des femmes de tous âges ainsi que des hommes majoritairement au-delà de 25 ans. Tous les profils sociaux sont représentés car les œuvres peuvent être aussi bien diffusées gratuitement sur des chaînes de TV, que sortir en salle ou être proposées sur des plateformes de VOD. Par contre, les articles et critiques sur les artisans du cinéma ne vont être créés et partagés que par les amoureux fidèles du cinéma.



### c) Analyse de la concurrence :

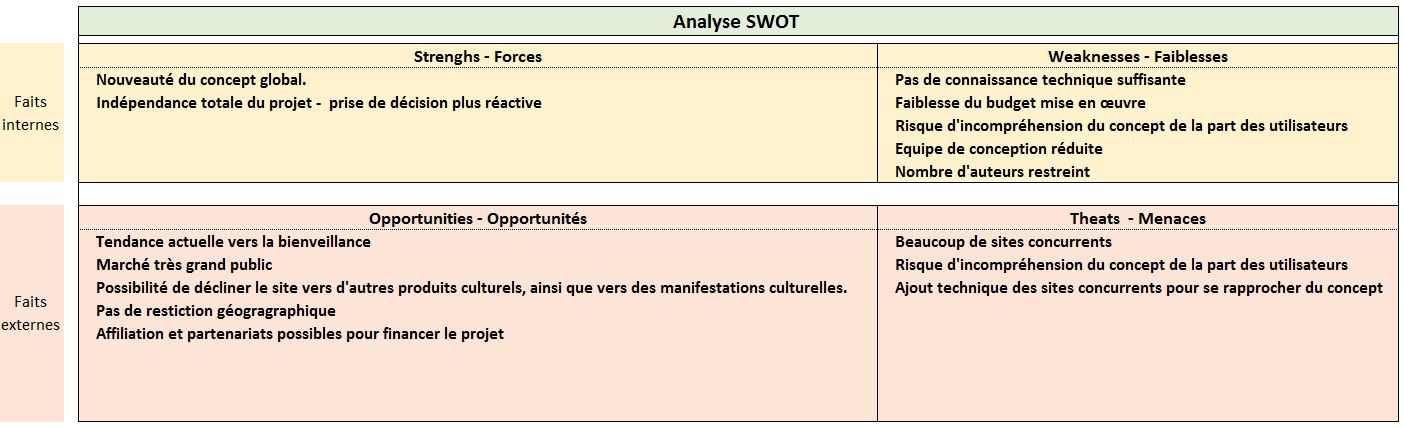
Les sites concurrents sont peu ou prou :

* Senscritique.com : Ce site donne la parole aux internautes sur un large panel de produits culturels (films, JV, etc) afin d’y laisser une note et une critique qu’elle soit négative ou positive.
* Allociné : C’est le site de référence en VF de toute l'actualité du cinéma. C’est aussi une encyclopédie des acteurs du 7ème art . Devant le succès des séries, le site s’est adapté en y ajoutant les informations adéquates.

Les sites qui sont presque identiques mais dans d’autres domaines :

* Babelio : c’est le réseau social des lecteurs. Ce site concentre toutes les informations du monde littéraire. Et rassemble une communauté qui partagent toutes ses informations de lectures, avec possibilité de créer sa propre bibliothèque.
* Tripadvisor : est le site communautaire de critiques en tout genre, mais uniquement sur des lieux de tourisme (restaurants, hôtels etc…)

### d) Analyse marketing

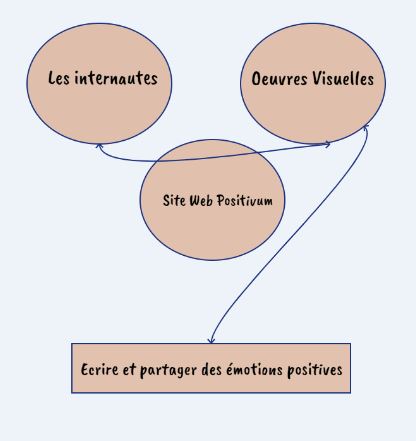


## 2 - Définition du besoin :

### a) Analyse de l’existant :

A la vue du projet en question, il n’y a pas d’existant pour la construction du site internet.

### b) Enoncé du besoin :



Les besoins primordiaux sont :

* Les utilisateurs doivent pouvoir consulter les articles des autres utilisateurs sans se connecter, ainsi que laisser un commentaire.
* Une fois connecté à leur compte, l’utilisateur peut devenir rédacteur du blog, créer une vidéothèque personnalisée des œuvres qu’il a déjà vu, et communiquer avec les autres rédacteurs. Selon son degré d’implication, il peut ensuite devenir à son tour modérateur. Chaque utilisateur connecté peut aussi créer, et/ou participer à un événement.
* Chaque article est rattaché à une fiche culturelle sur l'œuvre en question avec un lien d’achat sur un site marchand, si l'œuvre y est proposée.
* Une section Quizz est présente sur chaque bas de page. Les scores sont comparés entre les utilisateurs enregistrés.
* Les articles, ainsi que leur fiche culturelle doivent pouvoir être classé par catégorie d’émotion qu’il suscite.

### c) Les objectifs quantitatifs :

Dès lors, nous ne pouvons pas préciser les objectifs quantitatifs de ce projet, ni sur le trafic, ni sur le taux de conversion, ni sur le nombre de pages et services qu’il pourra contenir.

### d) Fonctionnalités du produit :

Le site internet ne sera pas bilingue dans un premier temps. Il aura une définition mobile, qui permettra de faire une recherche rapide sur une œuvre, de consulter et de mettre à jour sa vidéothèque (en sortant du cinéma par ex…) et qui limitera ses fonctionnalités à cela. Une version tablette peut permettre à un rédacteur d’écrire un article et de le faire parvenir au modérateur.  
  
La création d'une fiche se rattachant à une critique ne pourra se faire que par un modérateur dans un premier temps. Dans un deuxième temps, il est prévu qu'une API soit mise en place pour lier le tout automatiquement. Quand un utilisateur voudra écrire un article sur un sujet particulier, le sujet en question devra faire l'objet d'une recherche automatisé sur différentes bases de données qui enrichira le site de nouvelles fiches.

## 5) Contraintes :

### a) Contraintes techniques :

Les contraintes techniques sont liées à l’apprentissage de la formation, et ce jusqu’au titre professionnel, ainsi qu'à mon niveau de pratique des différents outils.  
  
Le site internet devra se composer d’un front-end utilisateur présentant les différentes catégories d'articles présentés et d’un back end contenant une base de données assez complète et complexe pour pouvoir rajouter des fonctionnalités à termes.

Dans ce cas présent, le front-end sera composé par l’HTML et le CSS, et un framework Django pour lier le back-end.  
  
Le site web, devra aussi prendre en compte une interface Mobile, afin qu’il soit utilisable sur smartphone.

### b) Contraintes légales et réglementaires :

Le projet devra respecter les lois en vigueur, ainsi que le RGPD pour l’enregistrement des données des utilisateurs.   
  
L’enregistrement du nom de domaine, ainsi que l'intégralité du projet appartiennent à M. Soulié (himslef).

### c) Contraintes de coûts

Pour l'instant aucune étude de coût a été complété pour ce projet. Mais, l'aboutissement et la finalisation fonctionnelle de ce site internet me tenant à cœur, il sera probable qu'un hébergement sur un serveur mutualisé, son déploiement, ainsi que l'enregistrement du nom de domaine, seront des dépenses réalisées à mes frais.

### d) Contrainte de délais

Nous ne pouvons établir de délai précis étape par étape. Nous savons dore et déjà que la date de rendu final est le 6 Septembre 2022.   
  
Ce délai a été ramené au 31 Aout pour des raisons de relectures et de transmission aux personnes compétentes.

# II – Spécifications Fonctionnelles

## 1 – Les cas d'utilisations (Use Case)

### Les fonctionnalités

Le site internet va contenir les informations & les fonctions suivantes :

* Des articles concernant l'univers du cinéma,
* Des billets sur des événements à venir, soit organisé par l'un des utilisateurs de l'application, soit par l'industrie du cinéma : festival, exposition, ouverture d'une nouvelle salle etc,
* Des fiches sur les métiers gravitant autour du monde du spectacle,
* Des fiches sur les personnalités, ou des personnes qui travaillent dans le monde du cinéma,
* Des citations tirées soit d'œuvre cinématographique, soit de personnalités,
* Des informations sur les auteurs des articles et critiques inclues dans leur signature,
* Des articles et des fiches référençant les 27 émotions psychologiques,
* La possibilité de devenir auteur après inscription, et validation de son premier article par le modérateur,
* Faire des Quizz sur tous les thèmes abordés par le site,
* Et enfin la possibilité, de se créer une vidéothèque de ses œuvres préférées se référant à des articles,

### Les acteurs

4 acteurs sont représentés dans ces cas d'utilisation de l'application :

* Le visiteur ou l'utilisateur lambda qui pourra :
  + Se créer un compte,
  + Se connecter,
  + Lire les articles,
  + Parcourir les événements,
  + Lire les profils des blogueurs,
  + Lire des fiches de métiers,
  + Lire une fiche d'un professionnel (acteurs, réalisateurs, techniciens etc..),
  + Ecrire un commentaire,
  + Jouer à un Quizz,
  + Lire des citations accompagnant certains articles,
  + Lire un fiche métier,
  + Lire une fiche ou un article sur l'une des 27 émotions,
  + Editer des commentaires.
* Le blogueur, Auteur, utilisateur héritera des usages du visiteur mais en plus peut :
  + Ecrire un article et le modifier,
  + Créer un événement et le modifier,
  + Se créer une vidéothèque personnelle, la modifier, la consulter et l'effacer au besoin.
  + Editer une citation tiré d'une œuvre, ou que l'on doit à un artiste.
  + Se créer une signature de blogueur qui sera apposée systématiquement à la fin de ses articles, mais aussi la modifier,
  + Afficher et modifier son profil d'utilisateur.
* Le modérateur qui hérite des droits précédents et qui pourra quant à lui :
  + Modifier ou supprimer un événement,
  + Corriger ou supprimer un article,
  + Supprimer un commentaire.
* Et enfin un administrateur du site qui pourra en plus :
  + Créer, modifier et effacer une fiche métier,
  + Effacer un compte utilisateur,
  + Créer, modifier et effacer un Quizz,
  + Modifier et effacer une citation,
  + Créer, modifier et supprimer une fiche d'un professionnel d'un métier du cinéma,
  + Créer, modifier et supprimer une fiche sur les différentes catégories des émotions.

## 2 - Les diagrammes d'activités

### a - Création d'un article par un utilisateur :

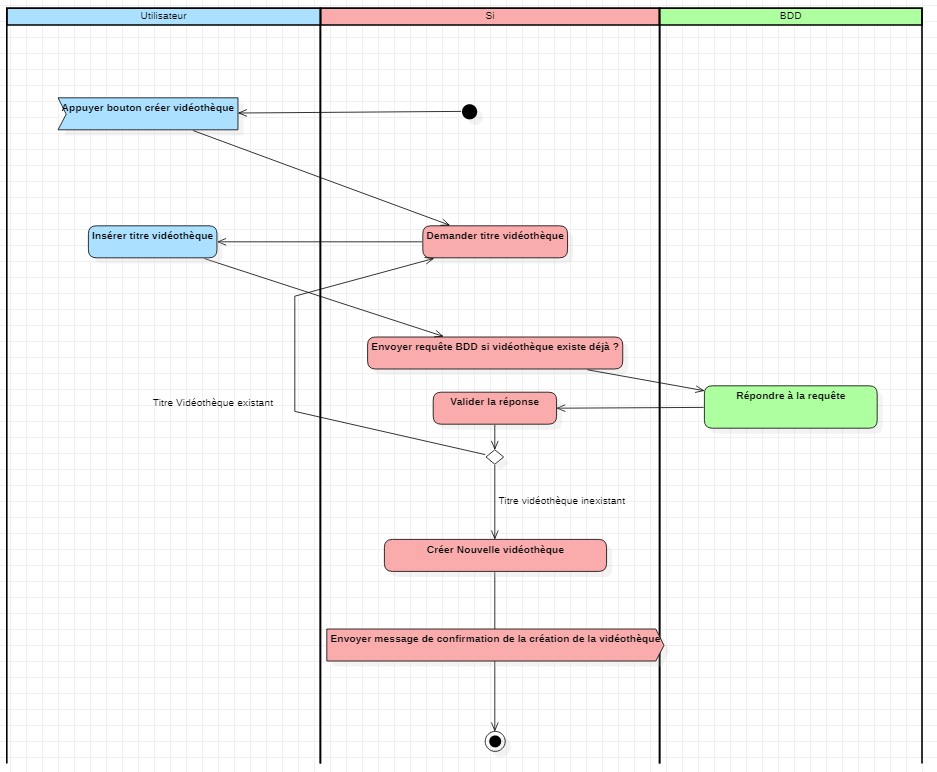
**Acteur(s)** : Auteur/Blogueur  
**Description** : L'auteur doit pouvoir créer un article et le valider  
**Préconditions** : L'utilisateur doit être authentifié en tant que "Blogueur"  
**Démarrage** : Le blogueur est sur la page d'accueil

Le diagramme d'activité nous informe sur les flux de contrôle et les flux de données. Il nous informe par le schéma, du déroulement d'un cas d'utilisation en particulier et un seul.  
  
J'ai pris comme premier exemple la création d'un article de blog par un utilisateur connecté.  
  
Dans un premier temps le système reçoit un **accept signal** de l'utilisateur qui veut créer un article et qui l'a fait savoir par l'action d'un bouton sur l'application.

Par la suite des **actions** décrites dans le tableau suivant vont se suivre entre l'utilisateur et le système. Ces actions sont des **actions atomiques,** c’est-à-dire quelles ne peuvent pas être décomposées. Si elles n'étaient pas atomiques, ce serait donc une activité que nous pouvons décomposer, et pour cela créer un nouveau diagramme d'activités.  
  
J'ai rajouté volontairement la BDD, car à un moment donné du flux, la BDD va être questionnée via une requête du système pour aller chercher une information précise : en l'occurrence si le sujet émis par l'utilisateur rentre bien dans les critères du site internet, et si le sujet peut-être traité.



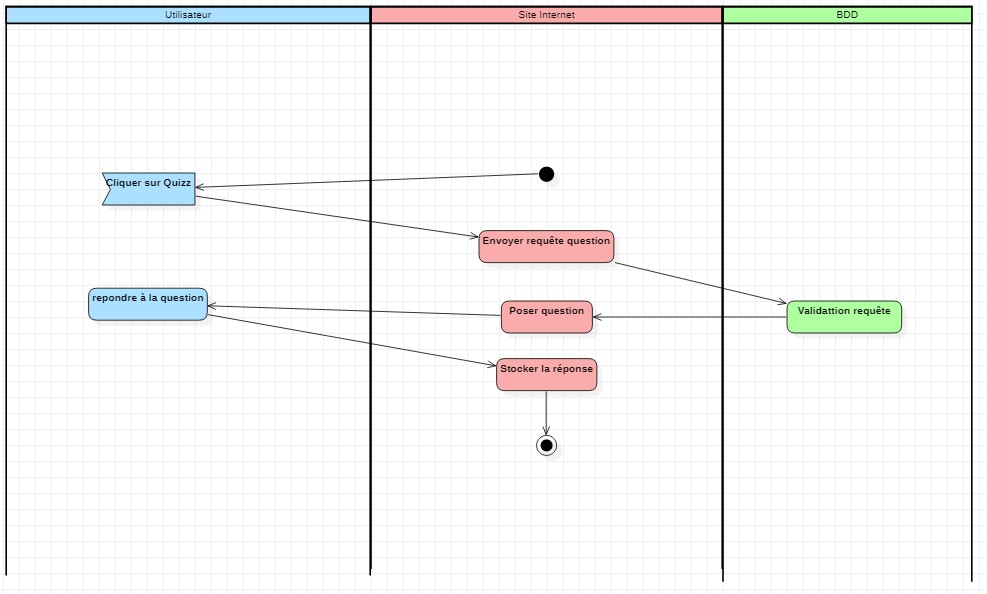
### b - Créer une vidéothèque



**Acteur(s)** : Utilisateur, Blogueur, Auteur  
**Description** : L'utilisateur doit pouvoir se créer une vidéothèque personnelle  
**Préconditions** : Le visiteur doit être authentifié en tant que "Utilisateur"  
**Démarrage** : L'utilisateur est sur sa page de profil



### c - Répondre à un Quizz



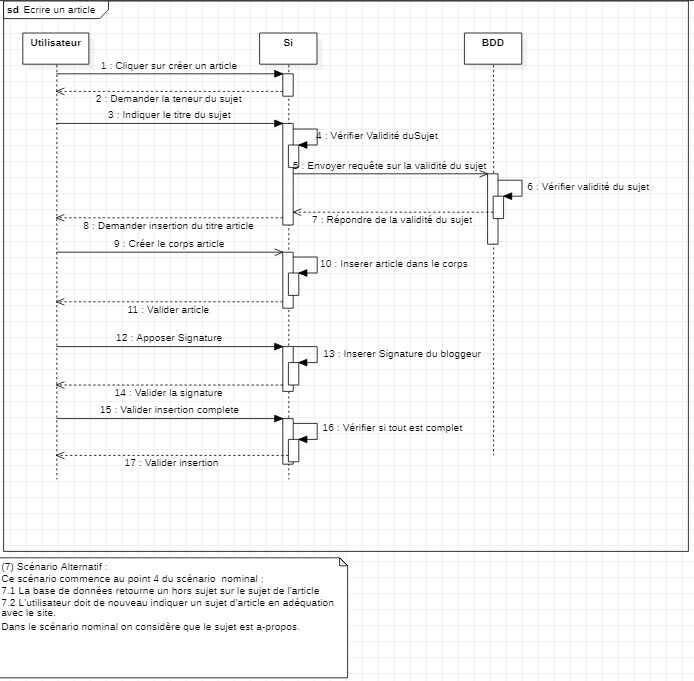
**Acteur(s)** : Tous les utilisateur sans ou avec connexion  
**Description** : Le visiteur peut participer à un quizz   
**Démarrage** : Le visiteur est sur la page d'une œuvre ou d'un article



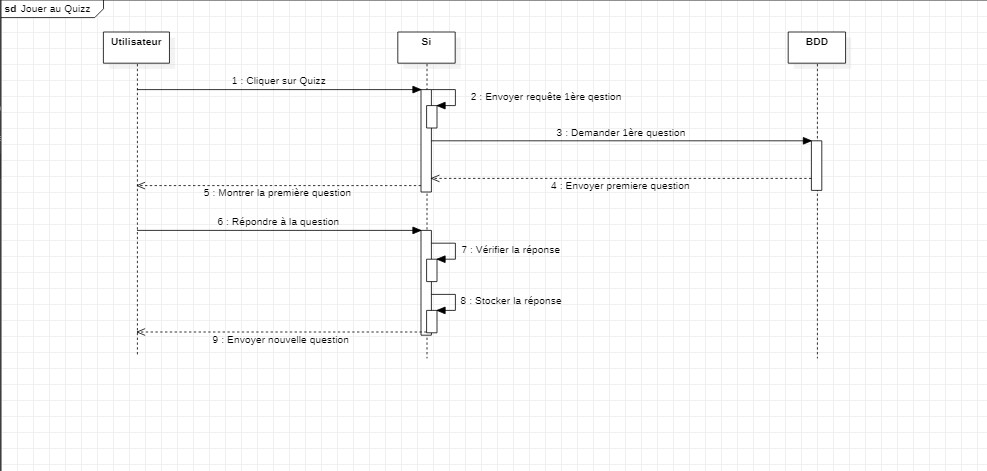
## - Les diagrammes de séquences

### a - Créer un article

Les diagrammes de séquences représentés dans ce document, sont tirés des mêmes exemples que ceux des diagrammes d'activités, qui sont eux même des exemples de cas d'utilisation.  
  
Ici, je reprends le même processus mais en établissant ce schéma avec un scénario nominal, sans cas alternatifs, que je décris plus bas dans le scénario alternatif.  
  
Dans ce diagramme, est représenté, les interactions entre l'utilisateur et le système de manière temporelle.   
On peut voir en 1 un message **synchrone**, car l'utilisateur attend une réponse du système. A l'étape 2, il s'agit de la réponse du système, que l'on appelle un **Reply Message.** De plus, quand le système vérifie une information ou travaille en interne comme à l'étape 4, il s'agit d'un **Self Message** .



### C – Répondre à un Quizz



# III - Maquettage et Arborescence

## 1 - L'arborescence

Le site, dès la page d'accueil, présente un grand nombre de boutons et menus afin d'accéder par n'importe quel point d'entrée, aux articles qui peuvent intéresser le lecteur. La page d'accueil présentera aussi succinctement, des articles récents ainsi que des catégories d'articles mis en avant.

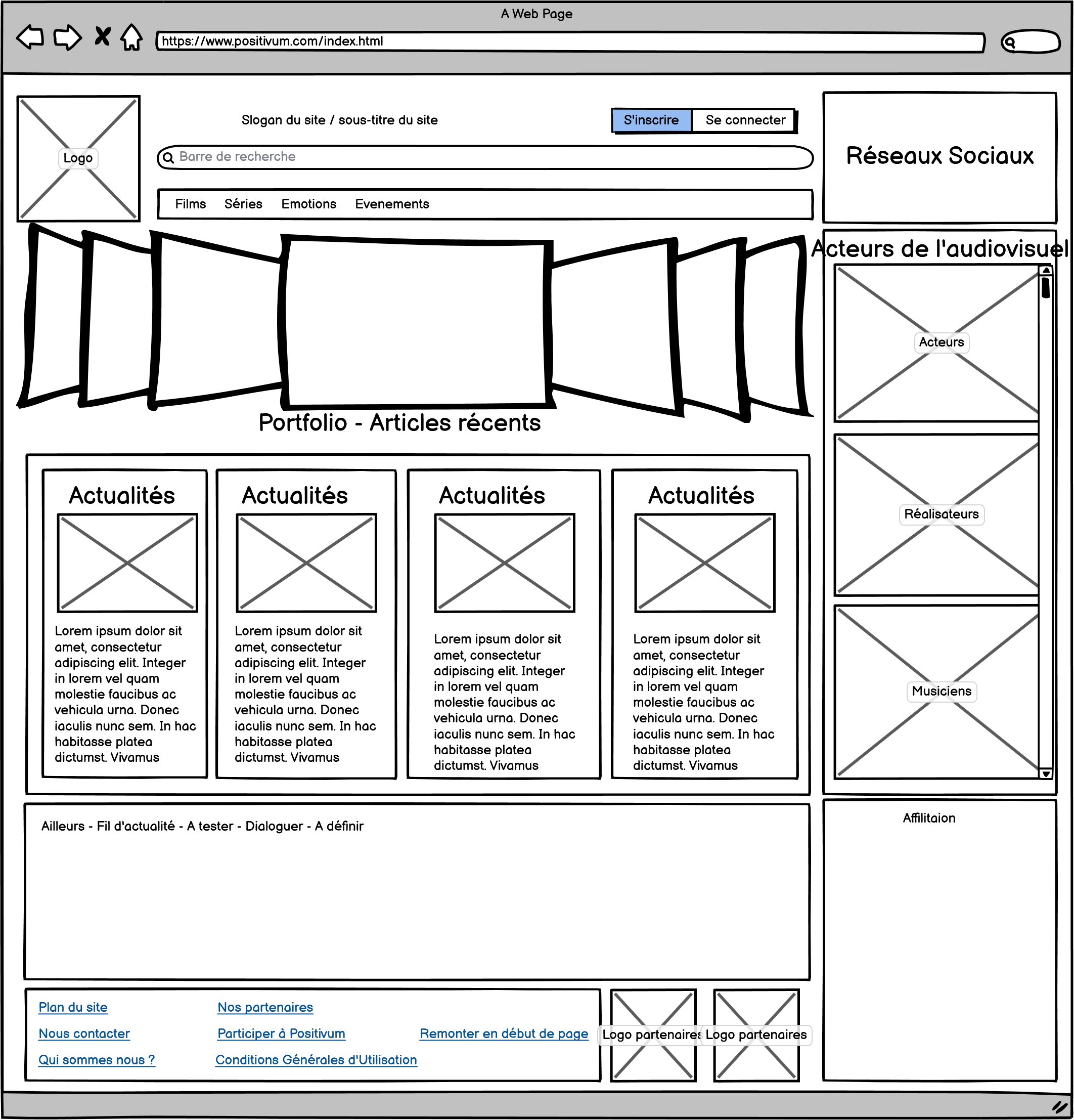
Les pages traitant les émotions reportent sur des articles qui décrivent des œuvres qui ont apportés les émotions définies dans leur catégorie respective.

De la même manière, les fiches traitant des artisans de l'industrie du divertissement, reportent sur tous les articles qui traitent des films dont ils ont participé à l'élaboration.

## 2 – Le maquettage

### a – Le wireframe

Ceci est la page d'accueil créée avec l'application **Balsamiq** – Elle comporte les parties **Header** / **Main** **Content** et **Footer** avec des sous-parties qui seront actualisés de manières dynamique selon les créations des auteurs.

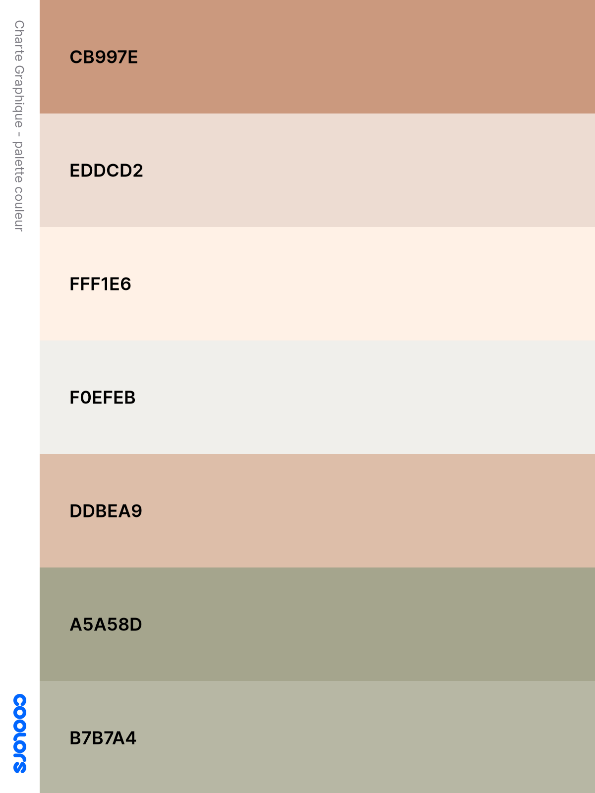


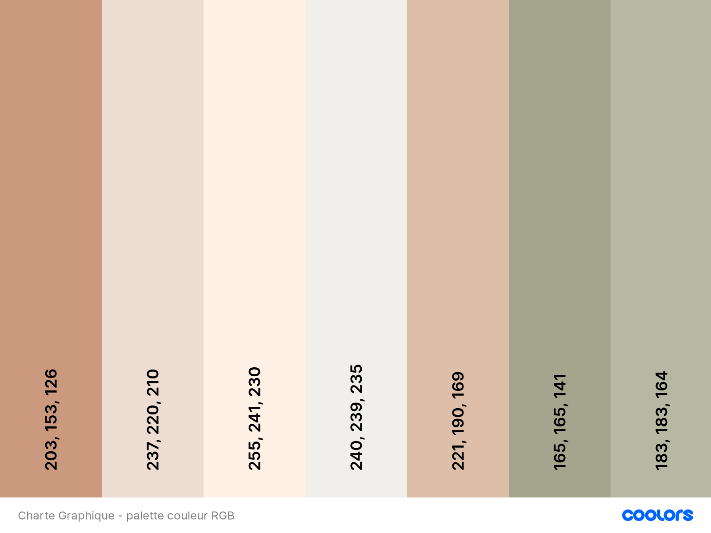
### b – Graphisme & Charte

Le logo a été pensé, pour que le visiteur, d'un seul coup d'œil perçoive le côté positif du concept global. L'œil représente une vision éclairée et bienveillante sur les choses qui nous entoure et que vous pourriez trouver sur le site.   
Le sourire est ce qui symbolise une certaine joie dans l'expérience vécue au travers des différents articles du blog. Les deux représentations de la bouche et de l'œil humanise le logo pour en faire un visage reconnaissable.  
Le choix de la forme géométrique du cercle participe à l'image de bonhomie à l'ensemble du graphisme. Rien ne sous-entend de tension, de force, ou une quelconque violence dans le logo.



La palette de couleurs a aussi été choisie en ce sens, de ne pas faire ressortir de couleur agressive, mais plutôt des nuances faisant référence à la nature. La cohérence apporte de la douceur à l'ensemble du graphisme. Ces deux palettes ont été créer, pour une cohérence entre elles avec le site **Coolors** qui permet, à partir d'une couleur de base, de conceptualisation l'ensemble des nuances cohérentes avec la première. Cela génère ensuite les palettes au format RVB(RGB) ou/et hexadécimal permettant une meilleure intégration au code **CSS**.





En ce qui concerne les polices de caractères, et en restant cohérent avec l'ensemble de l'image que doit apporter le graphisme du site à l'expérience utilisateur, j'ai opté pour du "Dancing Script" pour les menus et boutons et du classique "Arial" pour le contenu. Pour être lisible, la police "Dancing Script" doit toujours être bien détachée de son support avec un contraste bien marqué afin de ne pas nuire à la compréhension.

Dancing Script - Arial

### c – Le Mock-up (le prototype)

# IV – Conception

## 1 – La base de données

### a – MCD

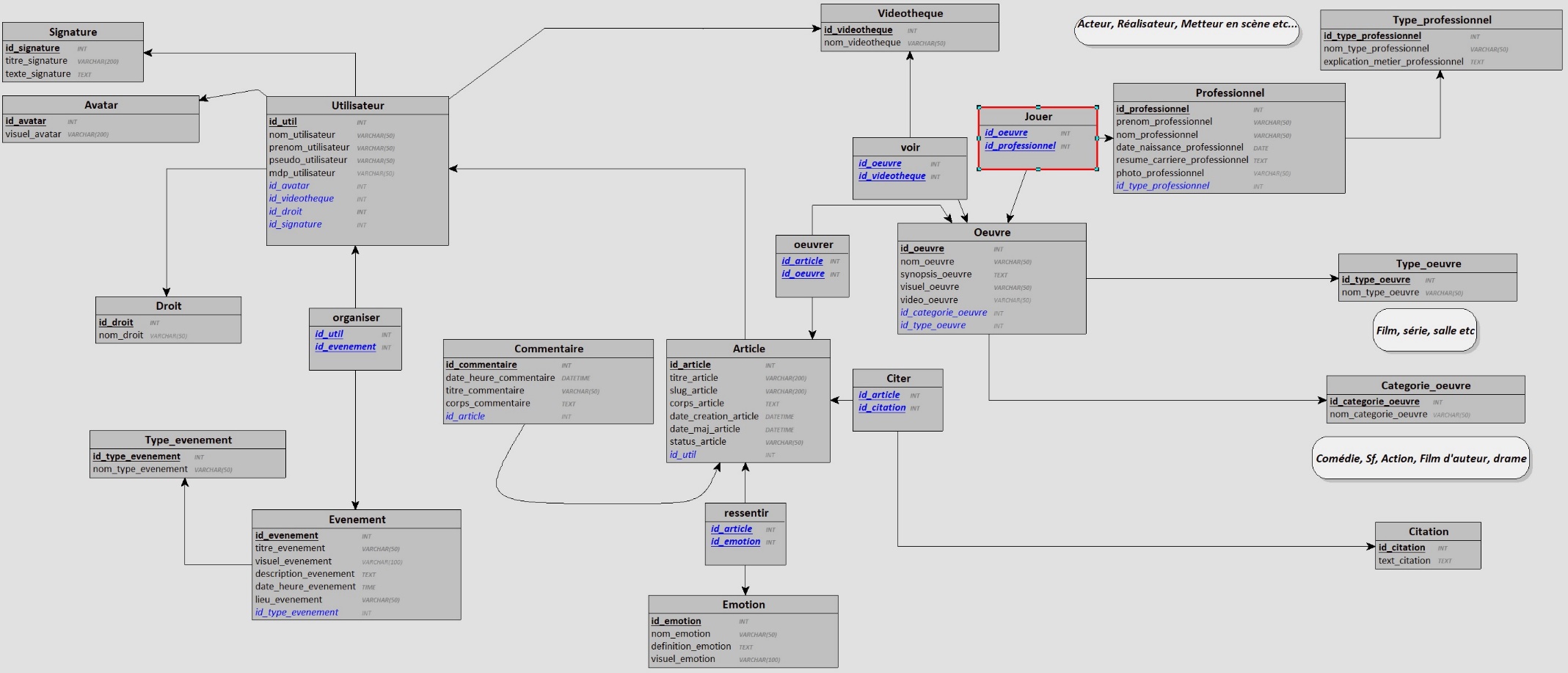
Le modèle conceptuel des données est la représentation graphique qui permet de comprendre comment sont liés les différents éléments que nous avons besoin lors de la création d'une base de données. C'est le schéma de transition pour pouvoir communiquer entre le non-informaticien afin de couvrir les besoins d'une BDD et le concepteur qui devra coder la base de données. C'est donc un plan abstrait et théorique de ce que devra comporter la base de données.   
  
Le MCD est composé d'entités, que l'on peut voir en jaune sur notre graphique, et qui représente un objet. Dans cette entité, la liste des données que nous allons y recueillir se nomment des attributs ou propriétés.

Les liaisons que l'on peut voir en ovale bleu sur le schéma, caractérise les relations qui existent entre les entités se nomment les associations.  
  
Et enfin, les couples de valeurs que l'on retrouve sur les traits des liaisons de tables s'appellent les cardinalités. Elles sont au nombre de 4, dans la plupart des cas, (0,1 / 0,N / 1,1 / 1,N).  
  
Pour la création de la base de données, j'ai utilisé le framework Django avec ses modèles **(models.Model**) qui sont codés comme des objets (class) et dont l'instanciation crée un enregistrement au sein de la base. Le fichier qui est généré est un sqlite3, que l'on peut à tout moment faire migrer vers **Mysql** par exemple où **PostgreSQL** qui aurait été mon choix pour créer la suite de mes fonctionnalités au sein de l'application.



### b - MLD

Le modèle logique des données est la représentation graphique de la structure d'une base de données qui nous permettra de créer le code SQL car ce schéma renferme un nombre suffisant d'informations pour permettre son édition.  
  
Lors du passage du MCD au MLD, les dénominations de chaque élément changent , nous ne parlerons plus ici d'entités mais de tables. Les attributs sont devenus des champs. Et l'ensemble de des champs d'une table est un enregistrement.  
  
Nous pouvons noter aussi, que les associations et les cardinalités ont disparus pour laisser place aux clés étrangères dans les tables, ce qui nous renseigne, et nous permet aussi de savoir, où, l'information doit être récupérer lors de la création de la base de données.



**Exemple du code de la création de la table Article par le framework Django :**

Lors de la création des tables, on peut se rendre compte qu'il n'est pas nécessaire d'indiquer à Django une clef primaire ainsi que de l'auto incrémenter car le framework s'en charge lors de la création des tables.  
  
Le **STATUS\_CHOICES = (** permet de créer un menu déroulant dans l'admin de Django afin de pouvoir modérer l'article et de ne publier que les articles qui ne sont pas en brouillon.  
  
Comme son nom l'indique le terme **models.Foreignkey** attribue à la table une clef étrangère provenant de la table appelée entre parenthèse, en l'occurrence ici la table **User**.  
  
Dans cet exemple on peut voir la création automatiques des tables d'associations par Django : citer, ressentir et œuvrer.   
  
Il est à noter aussi, que j'ai utilisé la super méthode **\_\_str\_\_** pour faire apparaitre le titre de l'article quand un objet article a été instancié. Sans cela, l'affichage de l'admin de Django, lors de la création d'un enregistrement, nous indiquerait uniquement le numéro de l'objet instancié.

class Article(models.Model):

STATUS\_CHOICES = (

('brouillon', 'Brouillon'),

('publié', 'Publié'),

)

titre\_article = models.CharField(max\_length=200)

slug\_article = models.SlugField(max\_length=200, null=True)

visuel\_article = models.ImageField(upload\_to="visuel\_article", blank=True, null=True)

corps\_article = models.TextField(blank=True)

date\_creation\_article = models.DateTimeField(auto\_now\_add=True, null=True)

date\_maj\_article = models.DateTimeField(auto\_now=True, null=True)

status\_article = models.CharField(choices=STATUS\_CHOICES, default='brouillon', max\_length=10)

publish = models.DateTimeField(default=timezone.now())

auteur = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True)

citer = models.ManyToManyField(Citation, blank=True)

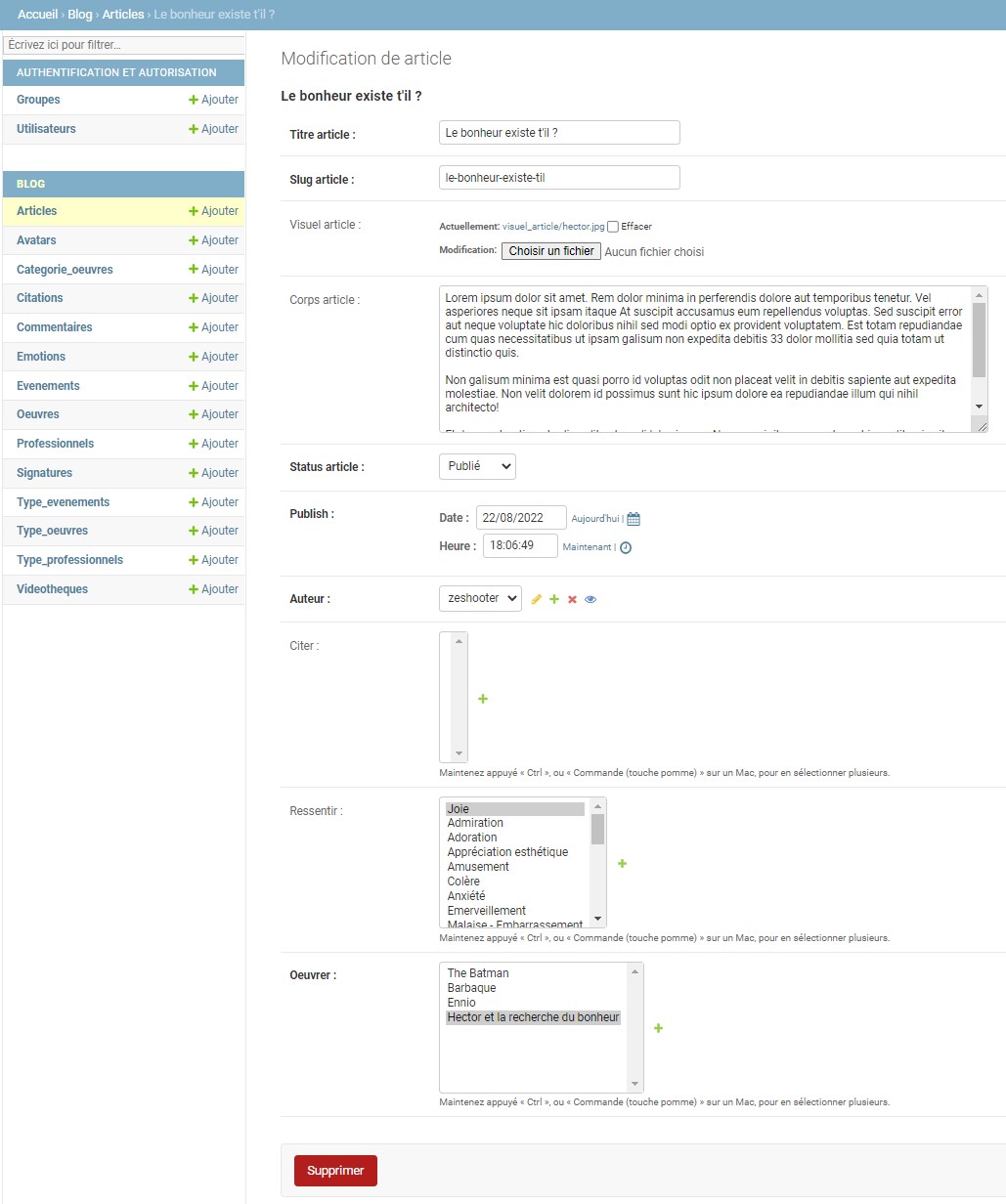
ressentir = models.ManyToManyField(Emotion, blank=True)

oeuvrer = models.ManyToManyField(Oeuvre)

def \_\_str\_\_(self):

return self.titre\_article

**Voici le rendu dans la console d'administration de Django de la table Article :**

****

**Autre exemple du code de la création de la table Commentaire par le framework Django :**

La table commentaire est intéressante car elle comprend un champ date et heure qui s'édite automatiquement avec la date et l'heure actuelle en faisant appel au Timezone.

Mais aussi, on peut voir que la clé étrangère qui provient de la table Article a un attribut de suppression en Cascade. C’est-à-dire dans ce cas précis, que si un article est effacé de la base de données, tous les commentaires affairant à cet article seront aussi supprimés de la dites base de données.

class Commentaire(models.Model):

date\_heure\_commentaire = models.DateTimeField( default=datetime.now, null=True)

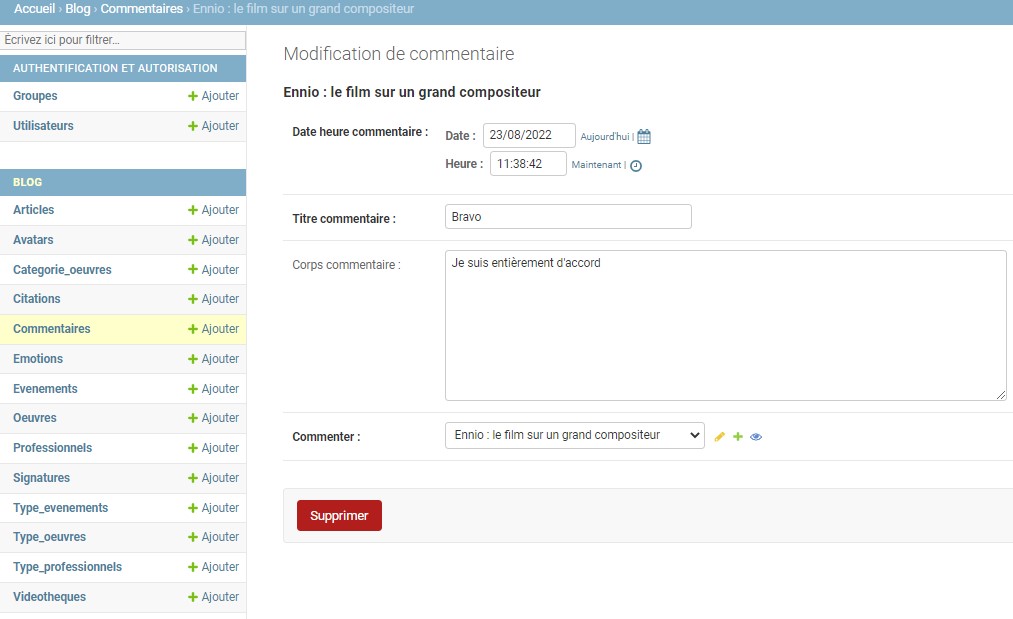
titre\_commentaire = models.CharField(max\_length=50)

corps\_commentaire = models.TextField(blank=True)

commenter = models.ForeignKey(Article, on\_delete=models.CASCADE)

def \_\_str\_\_(self):

return self.commenter.titre\_article

**Voici le rendu dans la console d'administration de Django de la table Commentaire :**

# V – Réalisation

## 1 - Choix des outils techniques

### a - Les langages de programmations

J'ai fait le choix pour la création de ce site de :

* **HTML 5** : pour la structure du front-end et la mise en place du DOM
* **CSS 3** : pour l'habillage et le design
* **Python 3** : pour le codage du Framework django

### 

### b - Les Frameworks

Le site a été créé avec deux frameworks :

* **Django 4**: ce framework repose sur le langage Python, et offre un ensemble de librairie vraiment pratique à l'usage et permettant d'accélérer le traitement de certain usage.
* **Bootstrap 5** : Ce n'est pas à proprement parler un framework, mais un ensemble de librairie CSS, HTML et JS permettant d'embellir ou d'enrichir l'apparence d'un site, tout en étant responsive.

### 

### c – Logiciels et autres outils

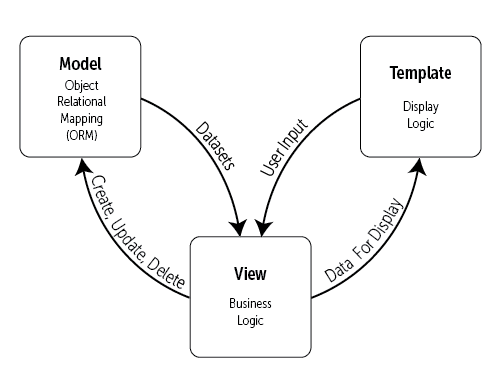
Les outils dont je me suis servis au quotidien sont :

* **VS Code** : L'IDE universel et super polyvalent qui se complète des extensions pour chaque langage afin d'avoir la couleur et l'auto complétion idéale.
* **GitBash**, **GitHub**, et **Github Desktop** : Le logiciel de gestion de version, qui permet aussi, grâce à Github de cloner son dépôt sur internet.

## 2 – L'arborescence du projet

### Django et son architecture

Il faut tout d'abord comprendre, que **Django** arbore sa propre architecture qui n'est pas **MVC** comme peut l'être **PHP**, mais **MVT** : comme Model – View et Template. En effet, on ne peut pas considérer que Django fonctionne avec des contrôleurs.



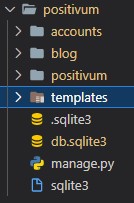
Les **Model** de **Django** sont un **ORM** permettant de créer une base de données, de la manipuler, mais aussi de la migrer au besoin.  
  
Les **View** sont des fonctions **Python** qui vont permettre de générer des pages web ou d'autres contenus de manière dynamique.

Et enfin, les **Templates** sont les pages qui vont être visible par l'utilisateur qui sont en HTML.

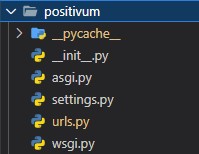
### 

### b – Explication de l'arborescence du projet

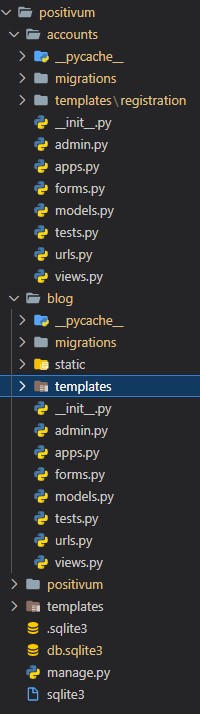
Ici, nous pouvons voir l'arborescence de base d'un projet **Django**. A savoir : le répertoire principal avec le nom du projet **Positivum**, ainsi que les applications qui regroupe la structure **MVT**.

Chaque application gère ses propres fonctions, mais est interconnectée avec les autres applications.   
  
Pour ce projet, j'ai choisi de créer deux applications au sein du même site. Le blog en lui-même qui comporte toutes la structure rédactionnelle sauf les utilisateurs que j'ai décidé de gérer avec l'application **accounts** car j'ai besoin de niveau d'autorisations différentes selon que l'utilisateur puisse être un blogueur, ou un médiateur. **Django** gère les groupes d'autorisations dans sa propre table **User**.

Nous pouvons aussi voir le fichier de la base de données au format **sqlite3**. Et pour finir le fichier **manage.py** qui est créé automatiquement à chaque nouveau projet **Django** et qui sert à lancer des commandes propres à l'administration de **Django** mais en passant directement par le fichier **setting.py** du projet en question.   
  
Et enfin, le répertoire **templates** qui est commun aux deux applications, et dont j'y ai logé le fichier **index.html**.



Le répertoire **positivum** contient tous les fichiers de réglages de tous le projet : les urls des applications, de la page admin de Django, les routes pour les images etc…



L'arborescence du projet est ici déployée afin de montrer les similarités entre les deux applications, mais aussi de détailler plus avant.

Le répertoire **migrations**  renferme la trace de chaque création de table, modification ou suppression de la base de données. En effet, après chaque opération sur la structure de la base de donnée, **Django** demande à ce que l'on fasse la migration des changements.  
  
Le dossier **static** comporte tous les contenus tel que les images ou les fichiers qui ne sont, ou ne seront pas générés par du code dynamiquement, tel que le fichier css.

Ce qui est particulièrement intéressant dans le framework **Django** c'est que la gestion des formulaires est prise en charge par la classe **Form**. Cela fonctionne peu ou prou comme les **Models**  de la base de données et gère une bonne partie du code que nous devrions écrire manuellement et nous le verrons dans une des fonctionnalités. Ici, nous pouvons voir le fichier que j'ai créé pour chaque application qui est **forms.py.**Le fichier **test.py**  comme son nom l'indique permet de faire des test unitaire.   
  
A ce point, je ne réexplique pas les fichiers **views.py, urls.py et models.py**  que j'ai cité plus haut.

## 3 – Fonctionnalités

### A – L'inscription

Comme beaucoup de site internet aujourd'hui, l'inscription est l'étape incontournable pour ensuite, accéder, éventuellement au formulaire de connexion, afin d'avoir plus davantage, et de droits, ainsi que de fonctionnalités.

Pour commencer, j'ai créé le fichier **forms.py**  requis par **Django** afin d'y éditer un formulaire sous forme de **class** (d'objet Python) et d'indiquer par l'instruction **class Meta** : vers quel table je dirige les données saisies par l'utilisateur et à quel champs cela doit correspondre. En l'occurrence ici, nous allons utiliser la table **User** du framework que nous n'avons pas eu besoin de créer par par les objets **Models** .

**Forms.py**

from django import forms

from django.contrib.auth.models import User

class RegistrationForm(forms.ModelForm):

username = forms.CharField(label="Username", max\_length=250, min\_length=5, help\_text='', required=True, widget=forms.TextInput(

attrs={"class": "form-control", "id": "username", "type": "text", "placeholder": "Username", "data-sb-validations": "required"}))

prenom = forms.CharField(label="prenom", max\_length=250, min\_length=3, help\_text='', required=True, widget=forms.TextInput(

attrs={"class": "form-control", "id": "prenom", "type": "text", "placeholder": "Prénom", "data-sb-validations": "required"}))

nom = forms.CharField(label="nom", max\_length=250, help\_text='', required=True, widget=forms.TextInput(

attrs={"class": "form-control", "id": "nom", "type": "text", "placeholder": "Nom", "data-sb-validations": "required"}))

email = forms.EmailField(label="emailAdress", max\_length=250, min\_length=5, help\_text='', required=True, widget=forms.TextInput(

attrs={"class": "form-control", "id": "email", "type": "email", "placeholder": "Adresse Email", "data-sb-validations": "required"}))

password = forms.CharField(label="password", max\_length=250, min\_length=8, help\_text='', required=True, widget=forms.TextInput(

attrs={"class": "form-control", "id": "password", "type": "password", "placeholder": "Mot de passe", "data-sb-validations": "required"}))

confirmation\_mot\_de\_passe = forms.CharField(label="confirmationMotDePasse", max\_length=250, min\_length=8, help\_text='', required=True, widget=forms.TextInput(

attrs={"class": "form-control", "id": "confirmation\_mot\_de\_passe", "type": "password", "placeholder": "Confirmer votre mot de passe", "data-sb-validations": "required"}))

class Meta:

model = User

fields = ('username', 'first\_name', 'last\_name', 'email', 'password')

On peut voir dans le code que je mets comme attributs le contexte qui va remplacer le code HTML dans une balise **<form>.**Pour piloter et faire le lien entre l'objet **RegistrationForm,** j'ai créé une fonction dans le fichier **views.py** qui permet de récupérer en méthode **"POST"** les informations que l'utilisateur va saisir, instancier l'objet **RegistrationForm,** récupérer l'instance dans une variable **new\_user** et sauvegarder cet enregistrement dans la base de données pour ensuite rediriger l'utilisateur vers la page de connexion. J'ai, pour le moment, fait le choix de ne pas faire de connexion automatique à l'enregistrement d'un nouvel utilisateur**.**

**Views.py**

def register\_view(request):

if request.method == 'POST':

user\_form = RegistrationForm(request.POST)

if user\_form.is\_valid():

new\_user = user\_form.save(commit=False)

new\_user.set\_password(user\_form.cleaned\_data['password'])

new\_user.save()

return redirect(login\_view)

else:

return render(request, 'registration/register.html', {'user\_form': user\_form})

else:

user\_form = RegistrationForm()

return render(request, 'registration/register.html', {'user\_form': user\_form})

Voici donc le code HTML du formulaire dont les contextes sont en places que nous avons éditées au préalable. Le bouton final est de type **submit,** ce qui envoi au moment de l'action au clic, les informations saisies pour le traitement que **Django** gère automatiquement.

**Register.html**

<div id="contenu">

<div class="container px-5 my-5">

<form id="contactForm" data-sb-form-api-token="API\_TOKEN" action="." method="post">

{% csrf\_token %}

<div class="form-floating mb-3">

{{user\_form.username}}

<label for="{{user\_form.username.id\_for\_label}}">Username</label>

<div class="invalid-feedback" data-sb-feedback="username:required">Username is required.</div>

</div>

<div class="form-floating mb-3">

{{user\_form.prenom}}

<label for="{{user\_form.prenom.id\_for\_label}}">Prénom</label>

<div class="invalid-feedback" data-sb-feedback="prenom:required">Prénom is required.</div>

</div>

<div class="form-floating mb-3">

{{user\_form.nom}}

<label for="{{user\_form.nom.id\_for\_label}}">Nom</label>

<div class="invalid-feedback" data-sb-feedback="nom:required">Nom is required.</div>

</div>

<div class="form-floating mb-3">

{{user\_form.email}}

<label for="{{user\_form.email.id\_for\_label}}">Adresse Email</label>

<div class="invalid-feedback" data-sb-feedback="emailAddress:required">Email Address is required.</div>

<div class="invalid-feedback" data-sb-feedback="emailAddress:email">Email Address Email is not valid.</div>

</div>

<div class="form-floating mb-3">

{{user\_form.password}}

<label for="{{user\_form.password.id\_for\_label}}">Mot de passe</label>

<div class="invalid-feedback" data-sb-feedback="motDePasse:required">Mot de passe is required.</div>

</div>

<div class="form-floating mb-3">

{{user\_form.confirmation\_mot\_de\_passe}}

<label for="{{user\_form.confirmation\_mot\_de\_passe.id\_for\_label}}">Confirmation mot de passe</label>

<div class="invalid-feedback" data-sb-feedback="confirmationMotDePasse:required">Confirmation mot de passe is required.</div>

</div>

<div class="d-none" id="submitSuccessMessage">

<div class="text-center mb-3">

<div class="fw-bolder">Form submission successful!</div>

<p>To activate this form, sign up at</p>

<a href="https://startbootstrap.com/solution/contact-forms">https://startbootstrap.com/solution/contact-forms</a>

</div>

</div>

<div class="d-none" id="submitErrorMessage">

<div class="text-center text-danger mb-3">Error sending message!</div>

</div>

<div class="d-grid">

<button class="btn btn-primary btn-lg" id="submitButton" type="submit">Submit</button>

</div>

</form>

</div>

</div>

Et pour finir, il faut que **Django** puisse router vers une **URL** pour la page de **Registration** dont nous avons créé une vue avec un contexte.  
  
**Urls.py**

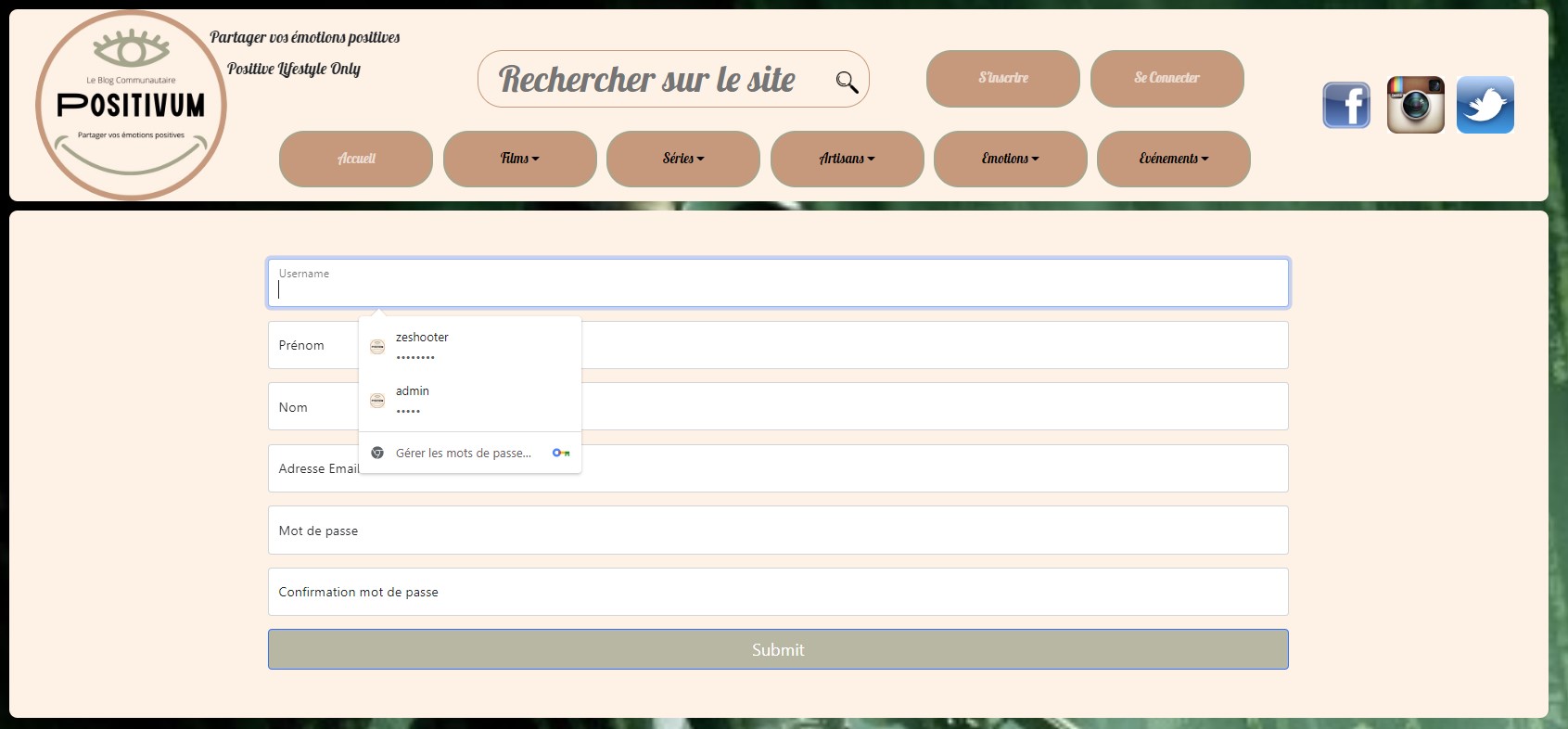
urlpatterns = [

path('login/', views.login\_view, name='login\_view'),

path('logout/', views.logout\_view, name='logout\_view'),

path('register/', views.register\_view, name='register\_view'),

]



### B – Commenter un article

Le site permet aux visiteurs de laisser un commentaire en dessous des articles. Le choix a été fait de ne pas les obliger à se connecter pour cela et peuvent laisser un texte en total anonymat. L'admin et le modérateur peut bien entendu les supprimer si cela ne suit pas la déontologie du site. A termes, si les visiteurs se multiplient, les commentaires seront modérés comme les articles.  
  
Tout comme l'inscription, les commentaires passent par un formulaire éditer dans le fichier **Form.py** et dont la class Meta indique que les informations seront dirigées vers la table **Commentaire**. De plus, je me suis servi des **widgets**  des formulaire Django pour faire apparaitre directement les zones de saisies dans le code HTML.

**Form.py**

class FormulaireCommentaire (forms.ModelForm):

titre\_commentaire = forms.CharField(max\_length=200, widget=forms.TextInput(attrs={'class': 'form-control'}))

corps\_commentaire = forms.CharField(widget=forms.Textarea(attrs={'class': 'form-control', 'rows':3}))

class Meta:

model = Commentaire

fields = ['titre\_commentaire', 'corps\_commentaire']

Ensuite, il faut créer une fonction récupérant ces informations, instancier l'objet **FormulaireCommentaire,**  vérifier si les informations sont valides, et les sauvegarder dans la base de données.

**Views.py**

def article\_detail(request, slug: str):

article = get\_object\_or\_404(Article, slug\_article=slug)

commentaire = Commentaire.objects.filter(commenter=article.id)

nouveau\_commentaire = None

if request.method == 'POST':

formulaire\_commentaire = FormulaireCommentaire(data=request.POST)

if formulaire\_commentaire.is\_valid():

nouveau\_commentaire = formulaire\_commentaire.save(commit=False)

nouveau\_commentaire.commenter = article

nouveau\_commentaire.save()

else:

formulaire\_commentaire = FormulaireCommentaire()

return render(request, 'blog/articles/detail\_article.html', {'article': article, 'commentaire':commentaire, 'nouveau\_commentaire':nouveau\_commentaire, "formulaire\_commentaire":formulaire\_commentaire})

Ensuite on insère un formulaire dans le code HTML en remplaçant toute la structure du formulaire par les **widgets** de **Django.**

**Detail\_article.html**

<section class="mb-5">

<div class="card">

<h2>{% with commentaire.count as total\_commentaire %}

{{total\_commentaire}} commentaire{{total\_commentaire|pluralize}}

{% endwith %}

</h2>

<div class="card-body">

<!-- Comment form-->

{% if nouveau\_commentaire %}

<h3>Votre commentaire a été ajouté</h3>

{% else %}

<h3>Ajouter une nouveau commentaire :</h3>

<form action="" method="post" class="mb-5">

{{formulaire\_commentaire.as\_p}}

{% csrf\_token %}

<button class="btn mb-4"type="submit">Ajouter Commentaire</button>

</form>

{% endif %}

<!-- Single comment-->

{% for commentaire in commentaire %}

<div class="d-flex mb-2">

<div class="flex-shrink-0"><img class="rounded-circle"

src="https://dummyimage.com/50x50/ced4da/6c757d.jpg" alt="..." /></div>

<div class="ms-3">

<div class="fw-bold">#{{forloop.counter}} {{commentaire.titre\_commentaire}} <span class="text-muted">commenté le {{commentaire.date\_heure\_commentaire}}</span></div>

{{commentaire.corps\_commentaire}}

</div>

</div>

{% empty %}

<h2>Il n'y a pas encore de commentaires</h2>

{% endfor %}

</div>

</div>

</section>

Et enfin la redirection via le fichier urls.

**Urls.py**

urlpatterns = [

path ('', views.article\_liste, name='index'),

# path ('liste.html', views.article\_liste, name='liste'),

path ('<slug>/', views.article\_detail, name='article\_detail'),

path ('oeuvrer/<slug:oeuvrer>/', views.article\_liste, name='oeuvre\_article\_liste')

# path('', views.ArticleListeVue.as\_view(), name='article\_liste'),

]



Afin de me documenter sur le fonctionnement du framework Django, je suis souvent tombé sur des sites en anglais, et bien que la documentation de Django en français est plutôt bien réalisé, pour comprendre le fonctionnement des templates, je me suis référé à la documentation en anglais que voici :

**"Templates"**

Being a web framework, Django needs a convenient way to generate HTML dynamically. The most common approach relies on templates. A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted. For a hands-on example of creating HTML pages with templates, see [Tutorial 3](https://docs.djangoproject.com/en/4.1/intro/tutorial03/).

A Django project can be configured with one or several template engines (or even zero if you don’t use templates). Django ships built-in backends for its own template system, creatively called the Django template language (DTL), and for the popular alternative [Jinja2](https://jinja.palletsprojects.com/). Backends for other template languages may be available from third-parties. You can also write your own custom backend, see [Custom template backend](https://docs.djangoproject.com/en/4.1/howto/custom-template-backend/)

Django defines a standard API for loading and rendering templates regardless of the backend. Loading consists of finding the template for a given identifier and preprocessing it, usually compiling it to an in-memory representation. Rendering means interpolating the template with context data and returning the resulting string.

The [Django template language](https://docs.djangoproject.com/en/4.1/ref/templates/language/) is Django’s own template system. Until Django 1.8 it was the only built-in option available. It’s a good template library even though it’s fairly opinionated and sports a few idiosyncrasies. If you don’t have a pressing reason to choose another backend, you should use the DTL, especially if you’re writing a pluggable application and you intend to distribute templates. Django’s contrib apps that include templates, like [django.contrib.admin](https://docs.djangoproject.com/en/4.1/ref/contrib/admin/), use the DTL.

For historical reasons, both the generic support for template engines and the implementation of the Django template language live in the **django.template** namespace."  
  
Dont voici une traduction :  
  
En tant que framework web, Django a besoin d'un moyen pratique pour générer du HTML dynamiquement. L'approche la plus courante repose sur les Templates. Un Template contient les parties statiques du rendu HTML souhaité ainsi qu'une syntaxe spéciale décrivant comment le contenu dynamique doit être inséré. Pour un exemple concret de création de pages HTML à l'aide de templates, voir le Tutoriel 3.

Un projet Django peut être configuré avec un ou plusieurs moteurs de templates (ou même zéro si vous n'utilisez pas de templates). Django fournit des backends intégrés pour son propre système de templates, appelé de manière créative le langage de templates Django (DTL), et pour et sinon il existe populaire Jinja2. Des backends pour d'autres langages de gabarits peuvent être disponibles auprès de "tierces parties". Vous pouvez également écrire votre propre backend personnalisé, voir Backend de template personnalisé.

Django définit une API standard pour le chargement et le rendu des templates, quel que soit le backend. Le chargement consiste à trouver le template pour un identifiant donné et à le prétraiter, généralement en le compilant dans une représentation en mémoire. Le rendu consiste à interpoler le template avec des données contextuelles et à renvoyer la chaîne résultante.

Le langage de gabarits de Django est le système de gabarits propre à Django. Jusqu'à Django 1.8, c'était la seule option intégrée disponible. Il s'agit d'une bonne bibliothèque de templates, bien qu'elle soit assez dogmatique et qu'elle présente quelques particularités. Si vous n'avez pas de raison pressante de choisir un autre backend, vous devriez utiliser la DTL, en particulier si vous écrivez une application pluggable et que vous avez l'intention de distribuer des templates. Les applications contributives de Django qui incluent des templates, comme django.contrib.admin, utilisent la DTL.

Pour des raisons historiques, le support générique des moteurs de modèles et l'implémentation du langage de modèles de Django se trouvent dans django.template.

# VI - Conclusion