Rapport de projet

Intégration de systèmes web et multimédia

Jean Schwickerath

Master 2 en Architecture des Systèmes Informatiques

Année académique **2021-2022**

LIÈGE université

UNIVERSITÉ DE NAMUR

Table des matières

Introduction

Madame Ramonfosse, qui représente notre client, nous demande de créer une application qui permet de diffuser du contenu multimédia sur n'importe quel écran. Cela dans le but de pouvoir afficher des vidéos, images ou documents sur des télévisions qui peuvent se trouver à divers endroits au sein des locaux d'une entreprise.

Cahier des charges

La demande du client est celle-ci : Créer une application web qui permet de créer et diffuser du contenu. La plateforme doit comprendre au minimum :

- Un rôle administrateur et un rôle utilisateur,
- Une page de connexion,
- La possibilité de gérer des fichiers,
- Planifier les diffusions
- Un lecteur récupérant les diffusions.

Le tout doit être facilement maintenable et facile à administrer car l'équipe IT du client n'est pas très grande.

Analyse

Use case

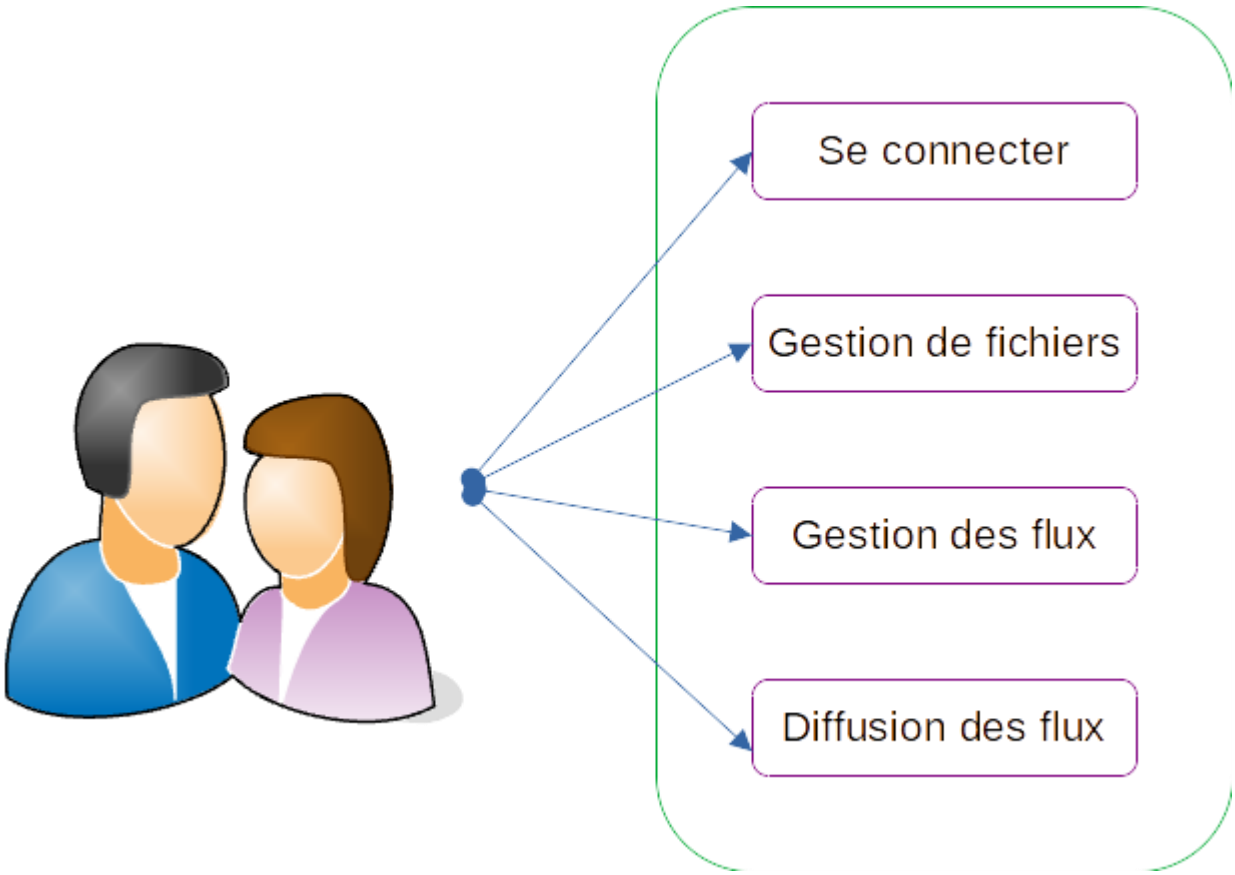

Figure 1 : Use case pour l'utilisateur

Wireframe

Lorsqu'on arrive sur le site et que l'on n'est pas connecté, il redirige sur la page d'index. Sinon, il nous envoie directement sur la page Workstation. Sur la page Index, deux choix sont possible : soit s'inscrire (non implémenté) soit se connecter. Une fois la connexion effectuée on est automatiquement redirigé vers la page Workstation qui permet de travailler les campagnes de diffusion.

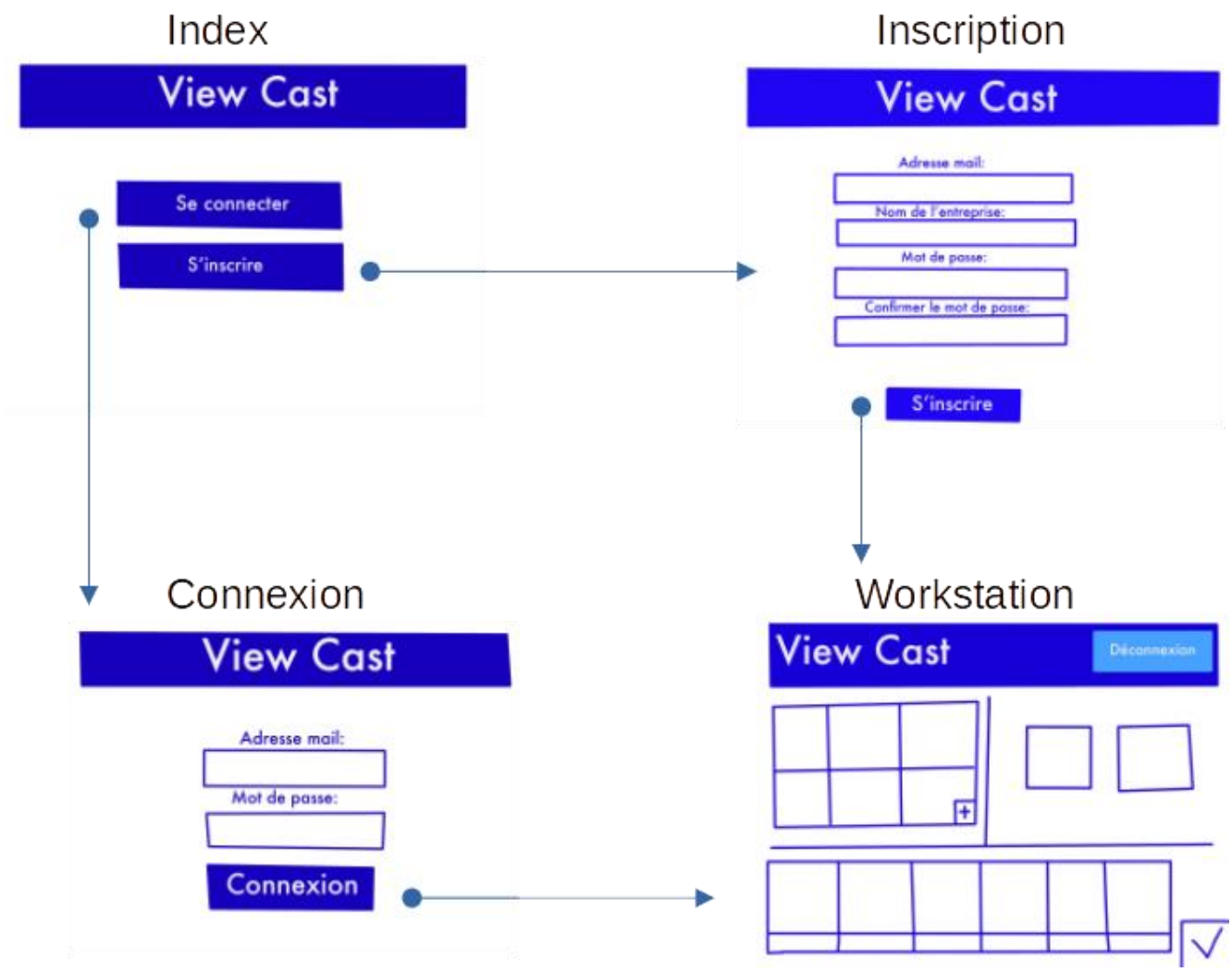

Figure 2 : Wireframe

Choix des technologies

Le projet web est divisé en deux parties distinctes, le frontend qui est la partie vue et usitée par l'utilisateur, et le backend qui est la partie qui permet de récupérer les données et les travailler.

Frontend

Pour une solution web, le frontend est développé en CSS/html. Du javascript est utilisé pour rendre le site dynamique. Bien que d'autre possibilités existent comme le SCSS ou le TypeScript, n'étant pas développeur web, j'ai opté pour cette solution classique mais très bien renseignée et facile à mettre en place. Afin de rendre le projet un minimum responsif, j'ai ajouté Bootstrap au projet.

Bootstrap
Bootstrap 5 est un outil qui permet de rendre les page web responsives grâce à un système de grille flexible. Très facile à mettre en place, il s'agit de fichier CSS et JavaScript a télécharger sur le site officiel et à importer dans notre projet.

Backend

Le backend est développé en python, il s'agit d'un langage multi-plateforme, libre, avec une très grosse communauté qui développe énormément de bibliothèque sous licence MIT. Pour développer un site web en python, deux choix majeurs sont possible, Flask ou Django.

Flask	Django
+ Facile à mettre en place + Léger, + Flexible, – On doit importer toutes les bibliothèques nous-même, – Demande un peu plus de connaissance.	+ Facile à mettre en place, + Importe toutes les bibliothèques dont on a besoin, + Crée automatiquement l'architecture MTV, – Très lourd, – Non flexible.

Après avoir comparé les deux Framework, j'ai décidé d'utiliser Flask, en effet Django n'était pas assez flexible pour moi, et très lourd pour un projet d'une envergure telle que celui-ci.

Flask
Flask est un Framework open source et très léger de développement web écrit en Python. Contrairement à Django il n'intègre pas par défaut de couche d'abstraction de base de données, de système d'authentification, ni d'outil de validation de formulaire, mais comme il est très facile d'en ajouter selon nos besoins, le système est donc très flexible.

Jinja
Flask se base sur le module Jinja, il s'agit d'un moteur de Template pour le langage de programmation Python. Il permet notamment, de rendre les fichiers html dynamique à l'aide de variables, de boucles et autres conditions.

SqlAlchemy
Comme Flask n'intègre pas de système de communication à la base de données, j'ai choisi d'utilise SQLAlchemy. Il s'agit d'une petite bibliothèque open source écrite en Python qui permet le mapping-object-relationnel (ORM). Il utilise du `Pattern Data Mapper` plutôt que de transmettre les requête SQL telles quels. De plus, SQLAlchemy fonctionne avec tous les systèmes de gestion de bases de données SQL.

Mariadb

Pour le système de base de données c'est Mariadb avec une utilisation classique de SQL qui a été retenue, principalement car elle était déjà présente sur mon serveur pour d'autre projet. Mais n'importe quel système peut être utilisé du moment que c'est du SQL, que ce soit SQLite, PostgreSQL ou même Microsoft SQL Server.

Architecture de la base de données

Dans le développement backend, la première étape a été de créer le modèle de données en respectant les 3 formes normales.

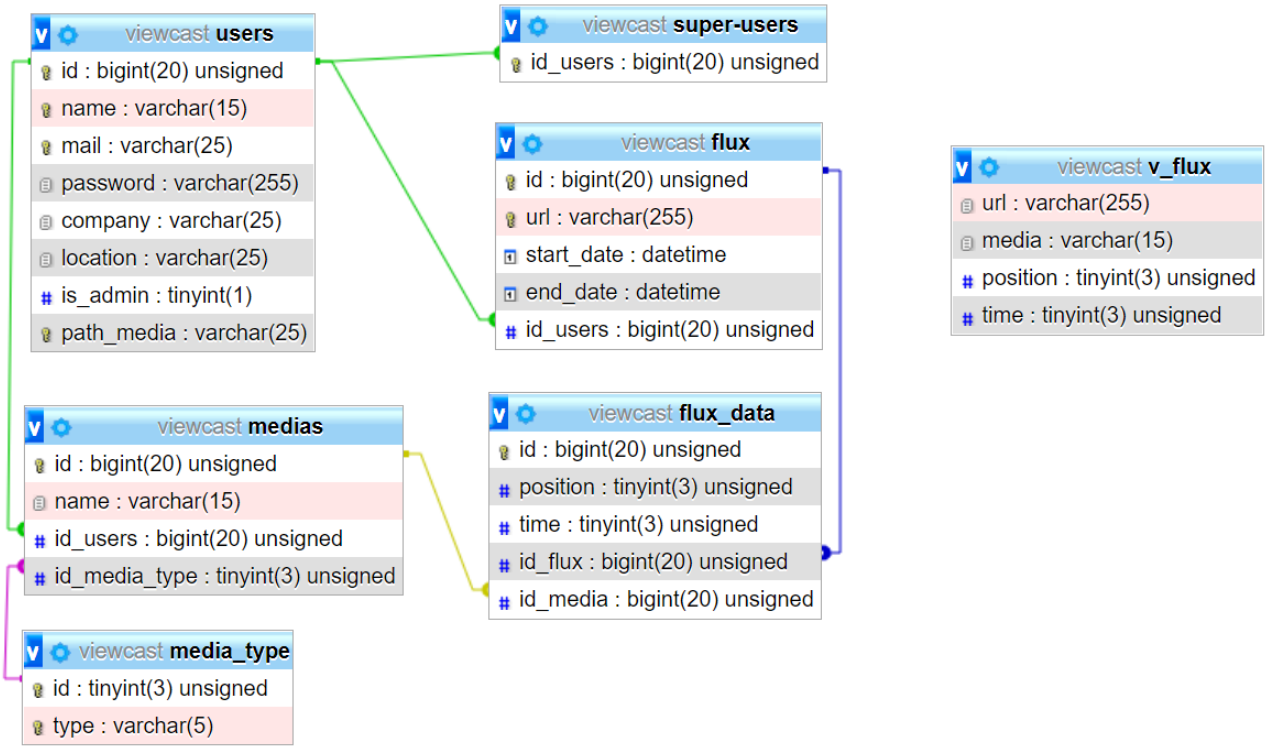

Figure 3 : Architecture de la DB et modèle relationnel

La table `users`, contient toutes les informations de bases. Si un utilisateur est admin, il a la possibilité de créer et planifier les campagnes, sinon il ne peut que les afficher.

La table `medias` contient les informations des médias dont on a besoin pour les campagnes, et la table `media_type` contient la liste des médias autorisés à utiliser sur le site.

La table `flux_data` contient les données que chaque média utilisé dans une campagne à besoin de savoir (comme son positionnement dans la campagne ou le temps d'affichage) et la table `flux` contient les données générales pour chaque flux comme son url (il s'agit de son code unique dont on a besoin pour la diffuser). La vue `v_flux` nous donne toutes les informations détaillées pour chaque média, dans chaque flux.

La table `super-users` est une table qui contient uniquement les id des utilisateurs qui sont aussi administrateur de tout le système. Toute personne ayant ces droits peut modifier / supprimer des utilisateurs / media/ flux.

Architecture du projet

Le projet utilise une architecture MTV. Il s'agit d'une architecture MVC (Modèle – Vue – Contrôleur) mais légèrement réadaptée pour développer en python avec Django ou Flask. Le principe reste le même, mais le Contrôleur prend le nom de Vue et la Vue prend le nom de Template. Le fonctionnement est celui-ci : l'utilisateur fait une requête http à la vue. Cette dernière demande au modèle les données stockées dans la base de données (ou en envoie des nouvelles selon la requête). Une fois les données récupérées, la vue les envoie au Template qui renvoie à l'utilisateur une réponse HTML.

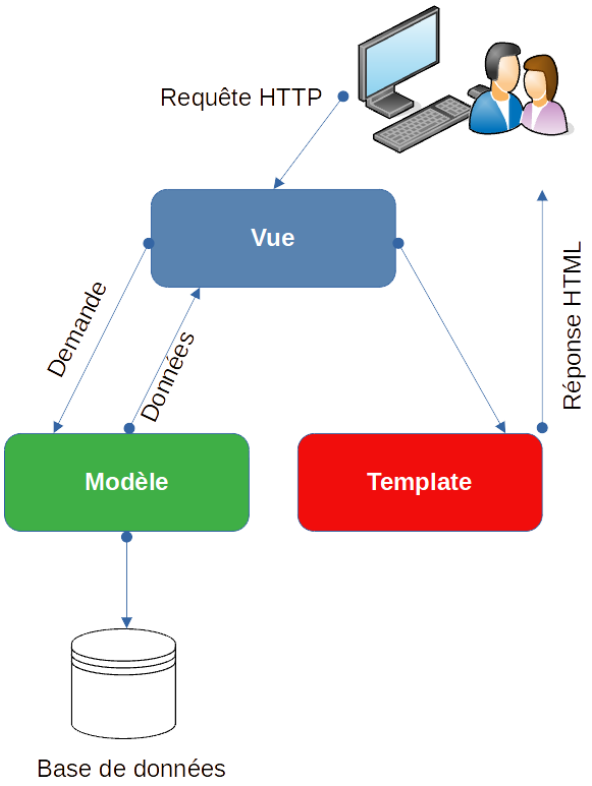

Figure 4 : Modèle MTV

Dans le projet Viewcast, les couches sont découpées de la manière suivante :

- *Modèle* : Cette couche représente la partie des données de l'application. Leur représentation, leur sauvegarde, leur récupération et leur traitement. Dans cette couche on y retrouve les classes suivantes :
    - Les `Repository` : Une par table, elles permettent de communiquer directement avec la base de données. Leur sauvegarde, leur récupération et leur traitement se font par ces classes.
    - Les `Models` : Ces classes nous fournissent la représentation finale des données qui vont être utilisées dans les couches supérieur.
    - Les `Databases` : Ces classes nous donnent les informations sur les structures des tables que l'on doit appeler pour les traiter.
- *Vue* : Il s'agit du contrôleur. Il sert de lien entre la couche Template et la couche Modèle. Il reçoit les requêtes http et appelle les méthodes du modèle et des templates, pour retourner ensuite la page html demandée. Dans cette couche on y retrouve les classes suivantes :
    - `Views` : Cette classe récupère les requêtes http, et les route vers les méthodes qui permet l'affichage de la page html demandée.
    - `Forms` : Ces classes contiennent tous les formulaires du site afin de plus facilement gérer les requêtes.
    - `Services` : C'est dans ces classes que vont être traitées les données et que la liaison avec la couche Modèle se fait.
- *Template* : Il s'agit de la couche qui représente l'interface web. Cette couche contient tous les fichiers html de l'application web.

- *Autre* : En plus des trois couches pour le MTV, deux couches supplémentaires sont implémentées :
    - `Static` : Cette couche contient tous les fichiers statiques relatif au développement web, comme les images et autres ressources, ainsi que les fichier CSS. Elle contient aussi les fichiers javascript.
    - `Tools` : il s'agit d'une bibliothèque de classe que j'utilise dans tous mes projet python. Elle contient différente fonctions et classe générique qui facilite le développement.

Voici-ci-dessous l'architecture du projet avec leurs classes.

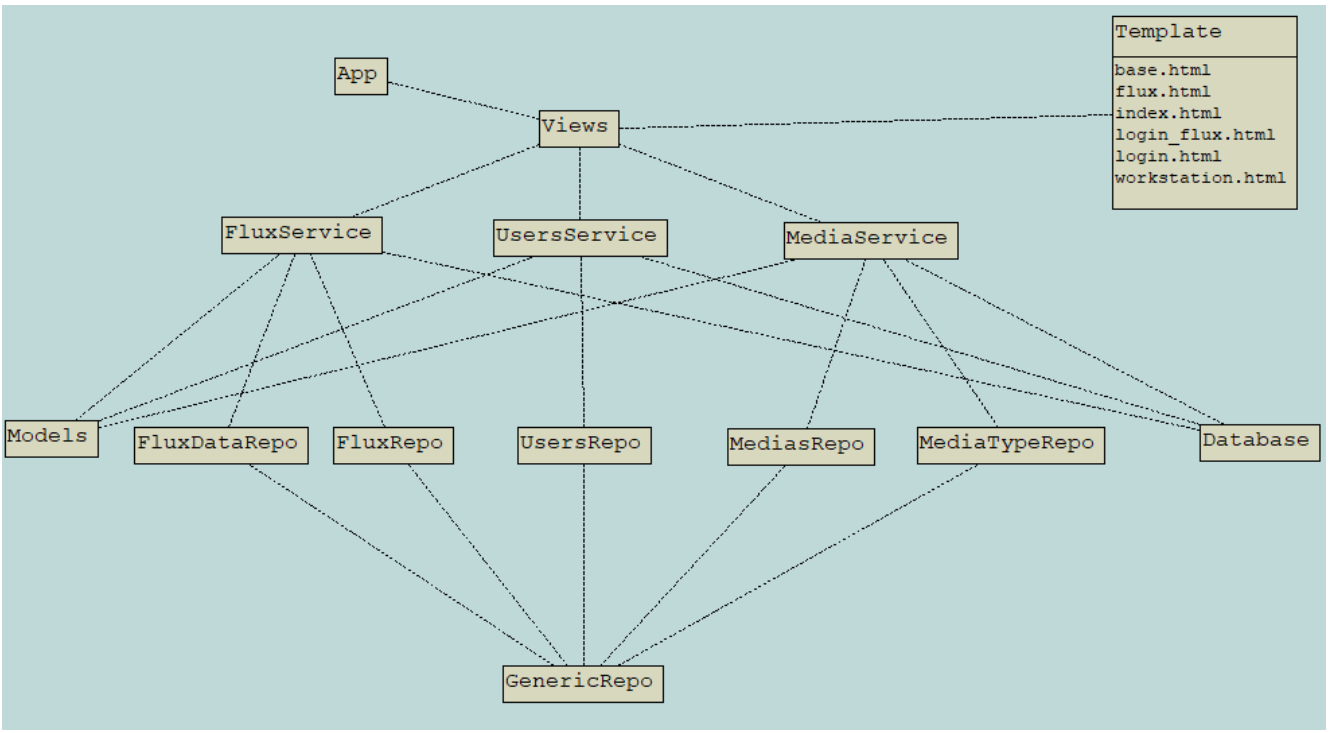

Figure 5: Architecture complet du projet

Rendu final

Interface d'accueil

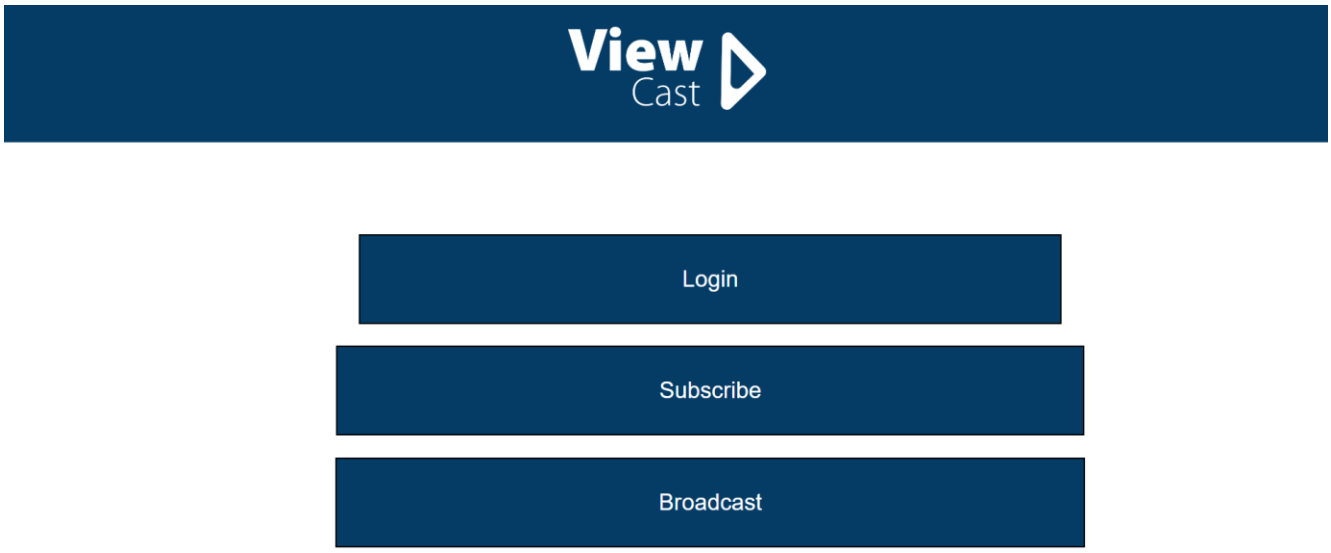

Figure 6 : index.html

Interface de connexion

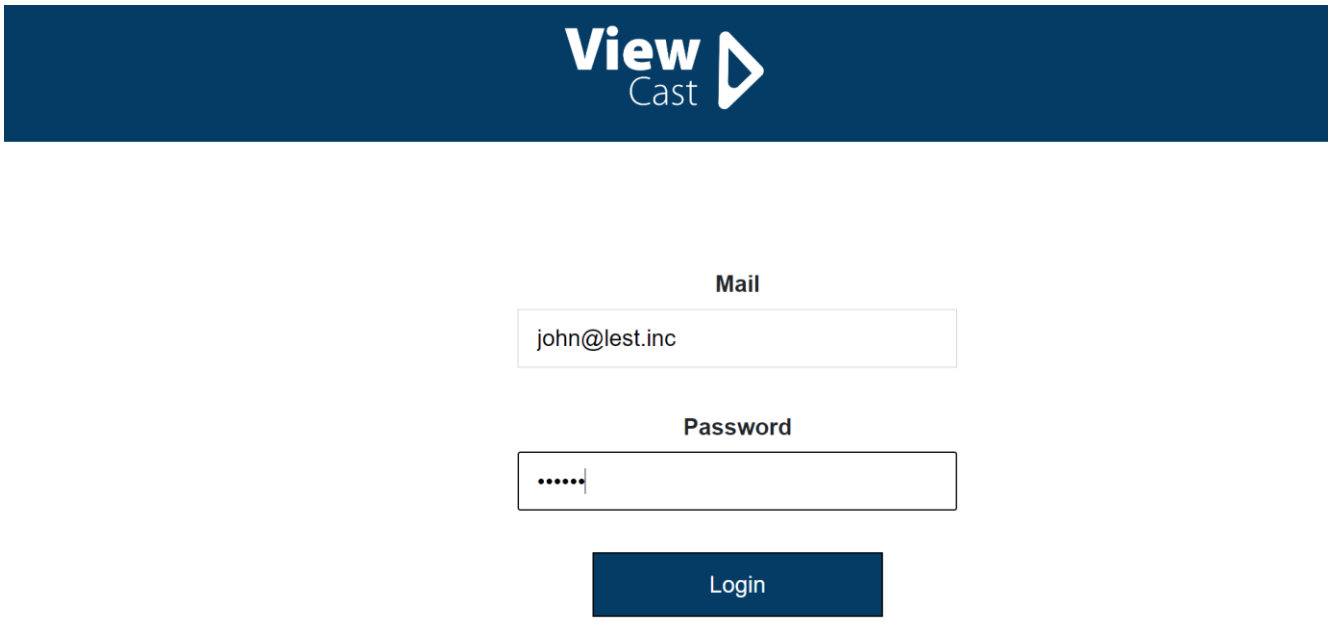

Figure 7 : login.html

Interface de gestion des campagnes

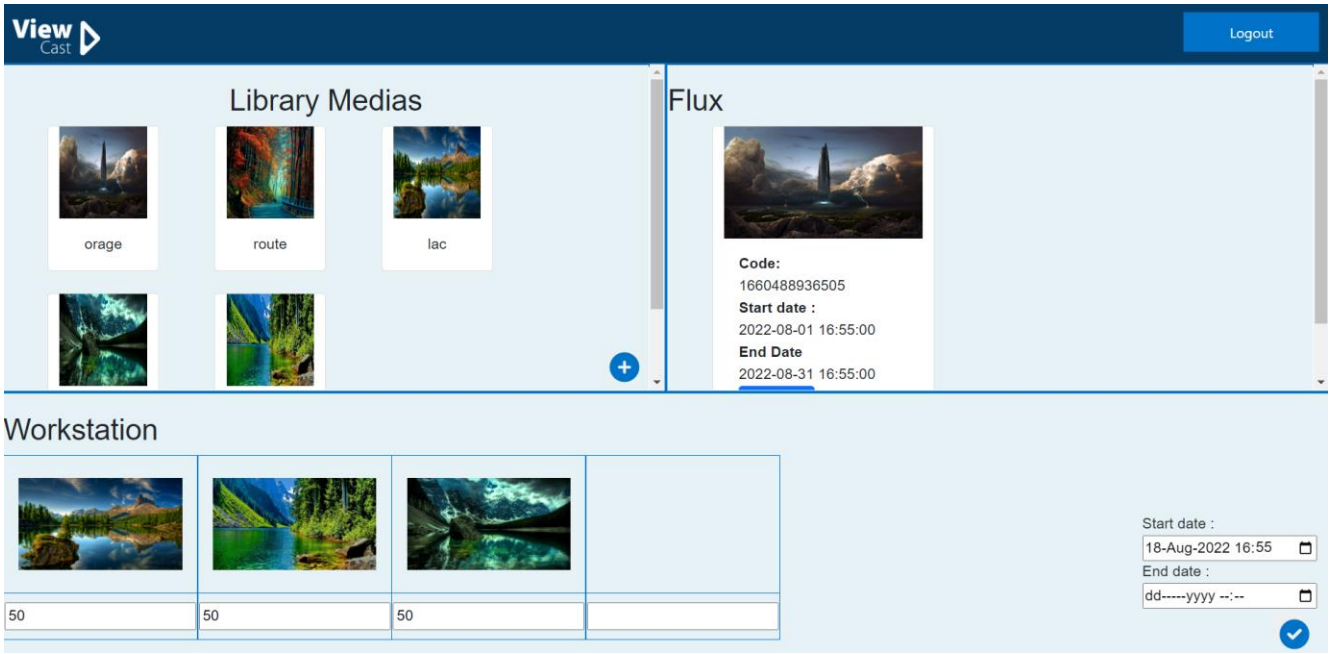

Figure 8 : workstation.html

Interface de connexion pour la diffusion des campagnes

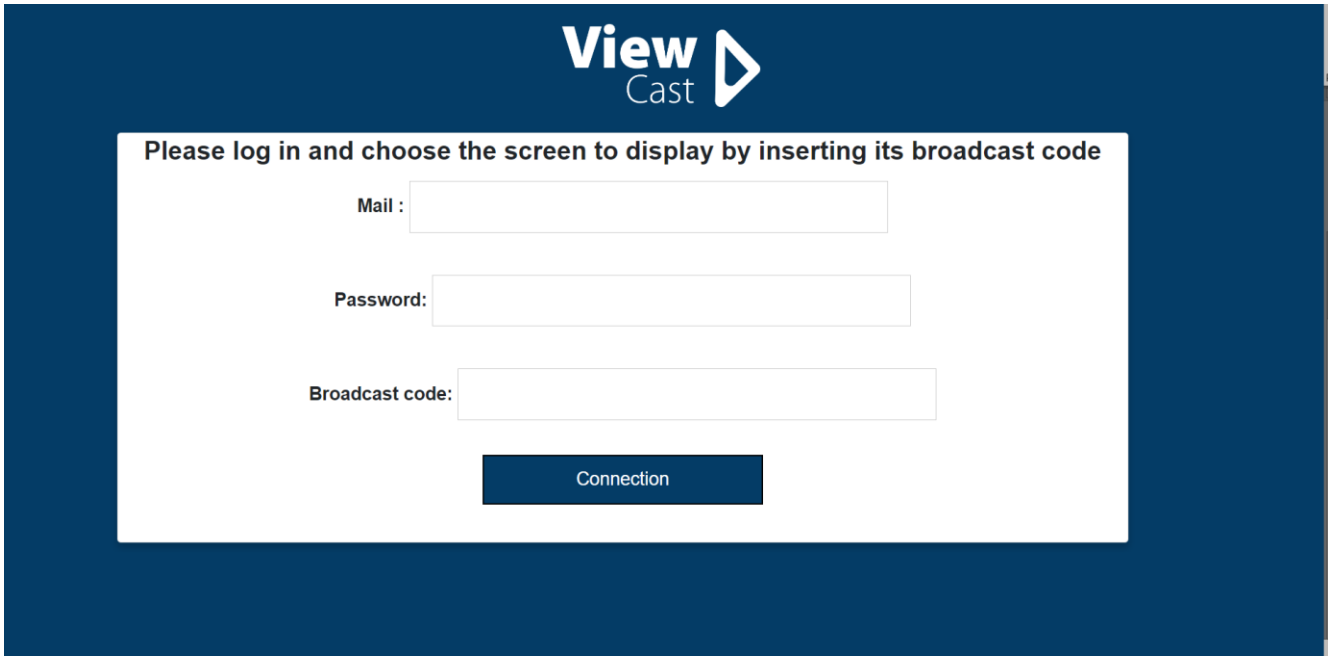

Figure 9 : login_flux.html

Étapes suivantes

Le projet est loin d'être terminé. Il faut encore au minimum :

- Gérer le système d'inscription
- Gérer le système d'administration des utilisateurs.

Ensuite il faut un peu sécuriser le site puisque pour le moment tout fonctionne en http :

- L'ajout d'une communication SSL pour passer en https,
- Chiffrer les mots de passe coté client est le strict minimum.

Certaines fonctionnalités devraient être rajoutées :

- Savoir le nombre de diffusion en temps réelle,
- Pouvoir modifier une campagne,
- Pouvoir diffuser depuis l'interface de gestion des campagnes et non plus devoir se connecter sur chaque écran...

Et le plus important : rendre le projet éco-responsable. Il faut alors réécrire certaines requêtes et certaines parties du code, afin de réduire l'impact environnemental de notre site web.

Mise en production

Comme la technologie utilisée pour développer le site web est de la technologie de base, il est très simple de mettre en production le projet. Si le serveur est un système Microsoft, il faut juste penser à installer Python 3.9 qui n'est pas natif à cet OS (disponible ici : https://www.python.org/downloads/).

Si vous voulez mettre en production via les services docker, un docker-compose est disponible pour mariadb et un Dockerfile pour le projet web.

Pour Mariadb, il suffit de se rendre dans le dossier qui contient le docker-compose et taper la commande suivante : `` `docker-compose -d` ``. Cela va créer un container mariadb sur le port 3306 et un container phpmayadmin sur le port 8081 afin d'avoir une interface graphique. Une fois mariadb installé il suffit d'importer le fichier sql.

Pour le projet en lui-même, il faut d'abord créer l'image à partir du Dockerfile fourni, il suffit donc de taper la commande : `` `docker build -t viewcast_image .` ``, puis la commande `` `docker run -d --name boomcraft_api -p 40100:40100 boomcraft_api_image` `` (vous pouvez bien sûr changer les ports à votre convenance).

Si vous ne souhaitez pas passer par docker, il suffit de créer un environnement virtuel avec la commande `` `python3 -m ./venv/` `` (vous pouvez bien sûr changer le chemin de l'environnement) et ensuite le lancer avec la commande `` `./venv/Script/Activate` ``. Une fois l'environnement lancé, téléchargez toutes les bibliothèques dont le projet a besoin avec `` `pip install -no-cache-dir -upgreade -r ./requierments.txt` `` et pour terminer lancez le projet avec `` `python3 -m flask run -host=0.0.0.0 -port=40100` ``