

Enseignement Technique de Qualification

Technicien(ne) en Informatique

Travail de fin d’étude

*Sauvageland.fr*

**Fontaine Emmerick**

Année scolaire 2021-2022

*Professeurs responsables*

Q. Carpentier et A. Vanovervelt

# Remerciements

Mes remerciements les plus sincères à toutes les personnes qui ont contribuées, de près ou de loin, à l’élaboration de ce rapport, ainsi qu’à la réussite de ma formation. Ce travail de fin d’études n’aurait pas pu voir le jour sans l’aide, au combien précieux, de nombreuses personnes. C’est pourquoi, je tiens à remercier toutes celles et ceux qui m’ont aidé et encouragé tout au long de l’élaboration de ce mémoire.

Je voudrais remercier tout particulièrement :

* Monsieur **CARPENTIER** ainsi que Monsieur **VANOVERVELT**,mes professeurs,qui m’ont beaucoup appris toutes ces années et qui m’ont permis de repousser mes limites.
* Ma famille et mes amis qui m’ont toujours encouragé dans mes choix personnels. Ils ont toujours été là pour moi afin de m’aider à franchir chaque étape de ces années d’études. Ils m’ont également soutenu dans les moments heureux comme dans les moments plus difficiles.
* Monsieur **BRUYERE**,mon maitre de stage, qui m’a permis de découvrir le monde professionnel ainsi que de nouvelles connaissances informatiques qui m’ont apportées un nouveau point de vue.

Table des matières

[Remerciements 2](#_Toc105316856)

[Introduction 5](#_Toc105316857)

[Logiciels utilisés 6](#_Toc105316858)

[Visual Studio Code 6](#_Toc105316859)

[GitHub 8](#_Toc105316860)

[Matériel utilisé 11](#_Toc105316861)

[Raspberry PI 4 B+ 11](#_Toc105316862)

[Analyse des besoins 13](#_Toc105316863)

[Développement 14](#_Toc105316864)

[Langage 14](#_Toc105316865)

[Qu’est-ce que Python ? 14](#_Toc105316866)

[Qu’est-ce que Flask ? 15](#_Toc105316867)

[Qu’est-ce que HTML ? 16](#_Toc105316868)

[Qu’est-ce que CSS ? 18](#_Toc105316869)

[Les croquis 19](#_Toc105316870)

[Mise en place de l’environnement 22](#_Toc105316871)

[Prise en main de Flask 23](#_Toc105316872)

[Initialisation 23](#_Toc105316873)

[Création de la barre de navigation 24](#_Toc105316874)

[Système d’identification 27](#_Toc105316875)

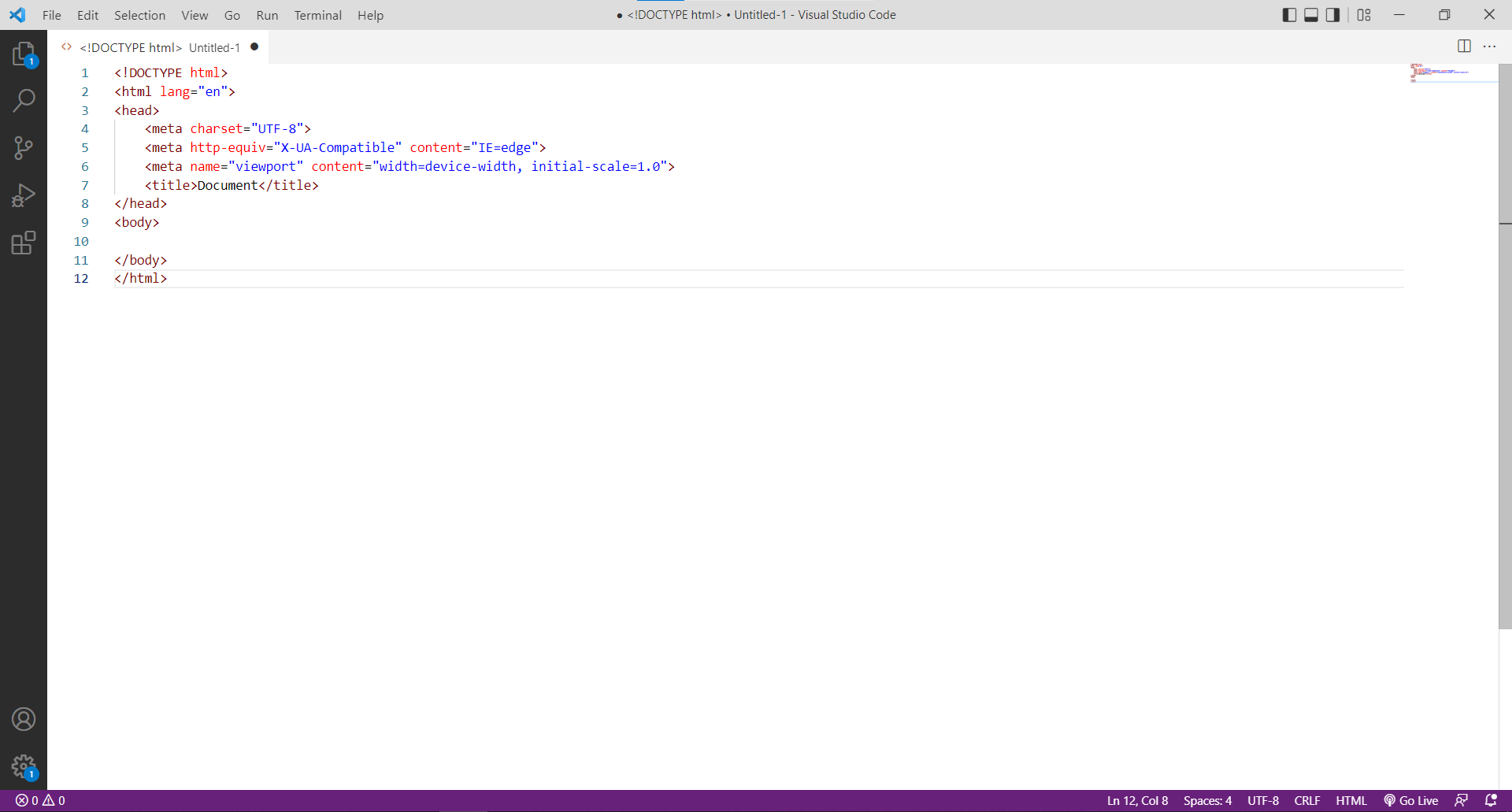
[Les bases données 28](#_Toc105316876)

# Introduction

Dans le but de mes études j’ai réalisé un site web pour un serveur de jeu Minecraft.  
C’est un sujet qui me plait car j’aime la programmation web ainsi que la gestion de serveur et les jeux vidéo. Ce projet me permet de réunir ces trois points.  
Celui-ci n’est pas uniquement un projet pour l’école mais un projet personnel qui tend à évoluer dans le futur. J’ai pour objectif de réaliser un magasin en ligne (shop) ainsi qu’un système de support se basant sur le système de ticket, une page de profil personnalisée en fonction du pseudo Minecraft et un système d’identification, le tout sous python avec l’extension web flask et hébergé temporairement sur un   
raspberry pi.

## Logiciels utilisés

### Visual Studio Code

VS (Visual Studio Code) est un éditeur multi-codes développé par Microsoft, il possède de multiples extensions pour aider aux codages ou des fonctionnalités pratiques comme l’intégration de GitHub et bien d’autres.

1

6

5

4

3

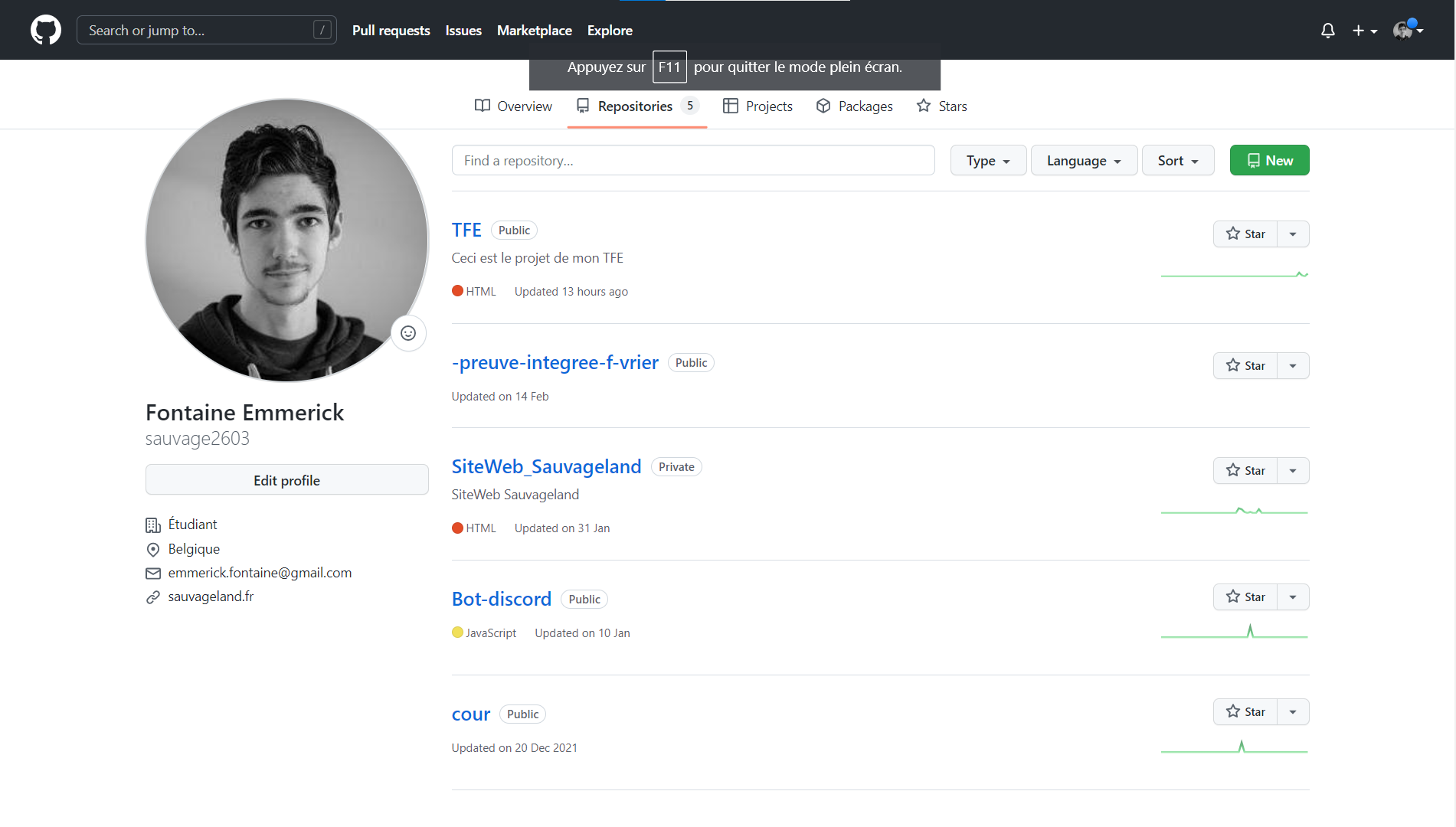
2

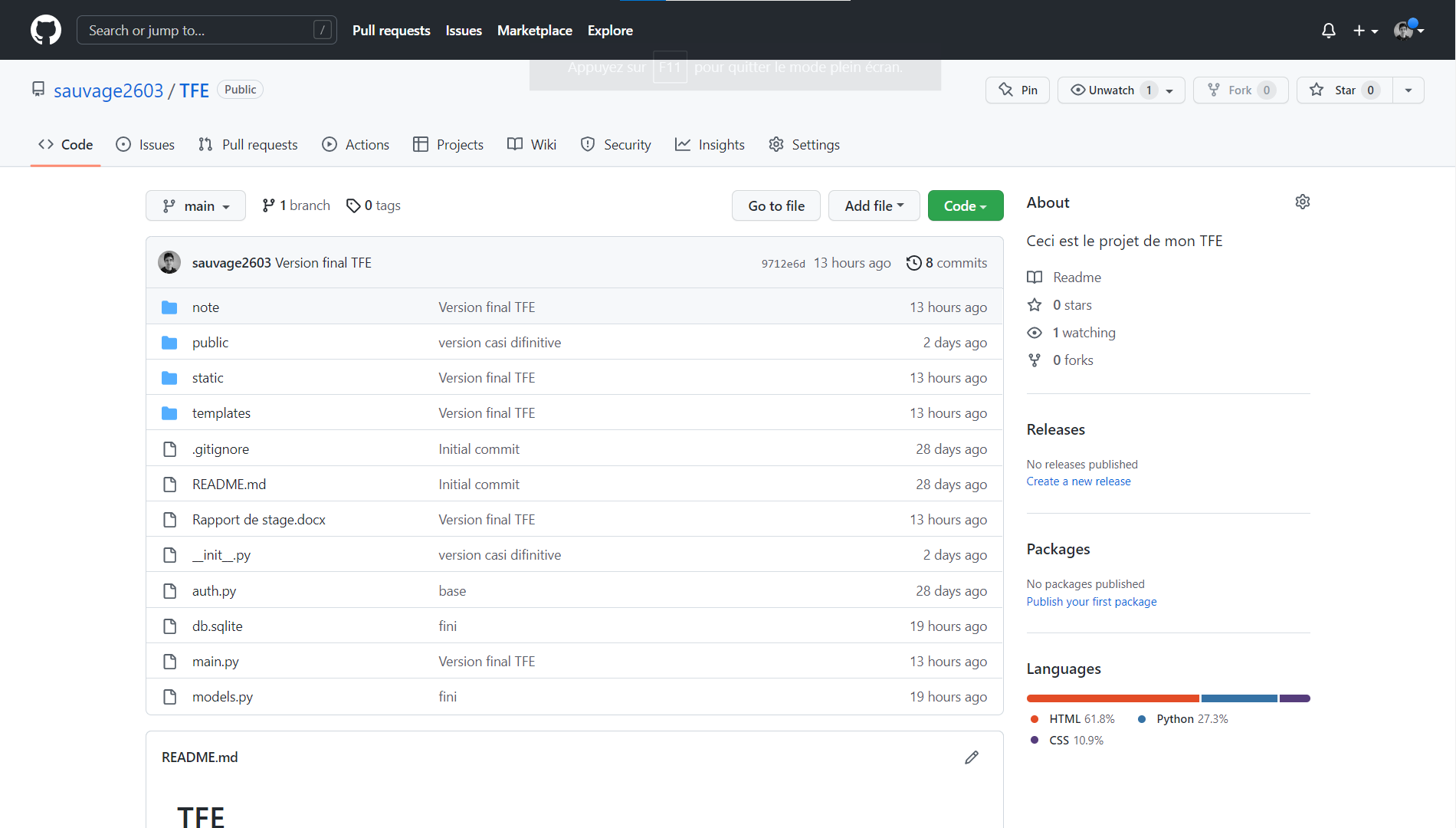
#### Zones de texte, VS intègre emmet (raccourcis qui font gagner du temps lors de la programmation).

1. « Explorateur de fichier » permet de voir les dossiers et fichiers utilisés et facilite ainsi la navigation.
2. « Recherche » permet de rechercher des lettres/mots dans un fichier et de pouvoir les remplacer par d’autre.
3. « Source control » permet de pouvoir utiliser GitHub grâce à un interface graphique (nous allons y revenir plus tard).
4. « Debug menu » permet d’exécuter le code en mode debug ce qui facilite la résolution des erreurs de codage.
5. « Extensions » permet d’installer des multitudes d’extension diverses et variées.

### GitHub

GitHub est un service web qui permet d’héberger des fichiers/dossiers avec un système d’historique et de backup.

Voici ma liste des répertoires personnels :

Et voici l’endroit où est stocké mon TFE :

Pour utiliser GitHub, il faut avant tout le télécharger sur https://desktop.github.com/ pour ensuite ouvrir son terminal et exécuter ces quelques lignes de commande.

Configurer son nom pour l’inscrire dans les fichiers lors de l’exportation des fichiers :

|  |
| --- |
| git config --global user.name "aquelito" |

Configurer son adresse mail :

|  |
| --- |
| git config --global user.email "axel@aquelito.fr" |

Initialiser un répertoire sur l’ordinateur :

|  |
| --- |
| git init <noms du répertoire> |

Ajouter tous les fichiers du répertoire au projet :

|  |
| --- |
| git add \* |

Enregistrer les changements effectués du code dans l’historique git :

|  |
| --- |
| git commit |

Envoyer les commits à votre dépôt distant sur de la branche spécifiée :

|  |
| --- |
| git push <remote> <branche> |

Chercher et merger les modifications d’une branche du serveur git et l’appliquer à la branche sur laquelle vous vous trouvez :

|  |
| --- |
| git pull <branche> |

## Matériel utilisé

### Raspberry PI 4 B+

Le Raspberry pi est un nano-ordinateur de la taille d’une carte de crédit conçu par des professeurs de l’université de Cambridge (école anglaise).   
L’avantage du Raspberry pi est sa faible consommation, son prix attractif (du moins à l’époque) et ces pin GPIO qui ne me seront d’aucune utilité pour ce projet.  
Comme tout ordinateur le Raspberry a besoin d’un OS et pour installer celui-ci, on utilise Pi Imager.





« Operating system » permet de sélectionner l’OS. Dans notre cas, on utilise Raspbian qui est développé pour le Raspberry sous base de linux Debian.  
« SD Card » permet de sélectionner le périphérique sur lequel l’OS sera installé.  
Après avoir tout sélectionné, il ne reste plus qu’à appuyer sur Write.

## Analyse des besoins

* La première chose à penser pour un serveur Minecraft est la facilité de la connexion au serveur. Minecraft utilise un système d’IP pour se connecter au serveur. Cette IP doit être visible dès la première page.
* La deuxième chose est la rentabilité du serveur. Ce qui engendre le plus de fonds est le shop en ligne qui va permettre d’acheter des bonus comme des grades sur Minecraft avec une méthode sécurisée.
* La troisième chose est un support qui permet de résoudre les problèmes des utilisateurs ou de les renseigner sur diverses choses. Je vais donc créer un système de ticket pour que les administrateurs puissent répondre aux demandes.
* La quatrième chose est un système d’identification qui accompagnera le support et le shop.

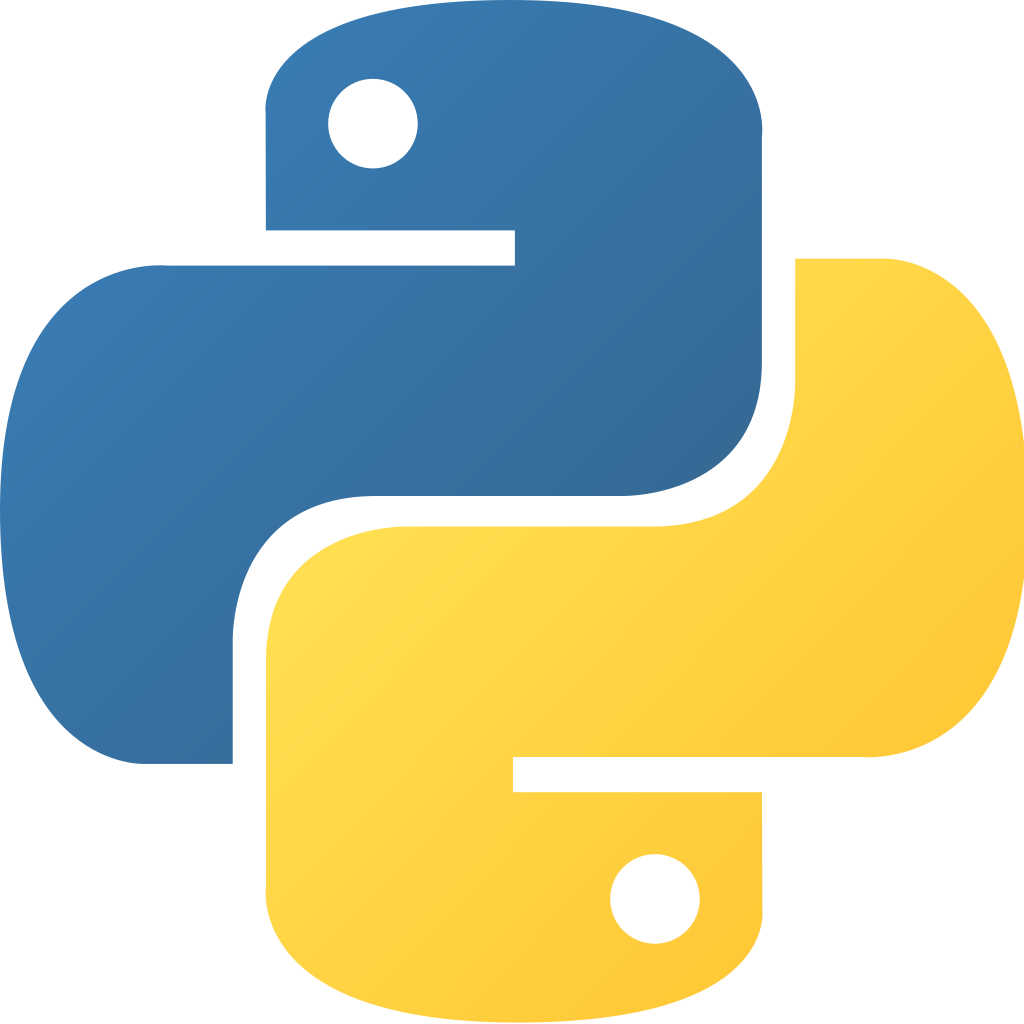
# Développement

## Langage

Pour la création d’un site web cela va de soi l’utilisation d’HTML est primordial ainsi que le CSS pour le designer des pages.

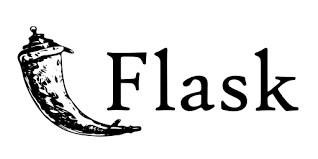
J’ai rajouté à ça l’utilisation de python pour l’exécution des commandes serveur. Pourquoi avoir choisi Python ? J’aurai très bien pu choisir un langage comme PHP mais mon choix s’est porté sur un langage un peu plus moderne.   
J’ai aussi utilisé une extension se nommant Flask afin de découvrir de nouvelles manières de programmer des sites web et améliorer mes connaissances en Python. Ce qui fut le bon compromis.

### Qu’est-ce que Python ?

Python est un langage de programmation créé en 1931, il est multi plates-formes et fonctionne autant sur ordinateur que sur smartphone, que ce soit sur Windows ou Linux. Celui-ci est de plus libres de droits. Il apporte également de nombreuses extensions élargissant les possibilités de création.

Une phrase résume bien Python « il ne faut pas dire que peut-on faire avec Python mais plutôt que ne peut-on pas à faire avec Python ».

### Qu’est-ce que Flask ?

Flask est une extension de python créée en 2010 concurrent PHP. On dit de lui qu’il est un microframework car il est très léger.

Cette extension possède également ses propres librairies, en voici des plus connues et utilisées pour mon TFE :

* « Flask-Login » permet de gérer les inscriptions ainsi que des connexions et de fournir des raccourcis pour la programmation.
* « Flask-mail » permet d’envoyer des e-mails comme indiqué dans son nom. Je n’ai malheureusement pas pu l’utiliser car j’ai rencontré des problèmes liés aux adresses email Google mais il sera implanté dans un futur proche.
* « Flask sqlalchemy » permet d’utiliser un ORM pour la gestion de mes bases de données ce qui facilite grandement l’utilisation de celles-ci. Cette librairie permet également de pouvoir changer de langage de base de données rapidement en quelques lignes de commandes.

Leur utilisation sera expliquée plus en détail plus tard.

### Qu’est-ce que HTML ?

HTML (HyperText Markup Language) a été créé en 1923. Celui-ci permet d’écrire de l’hypertexte (document incorporant un système de renvoi vers un autre document se nommant les liens/URL), de structurer schématiquement la page web ainsi que de créer des formulaires de saisie se nommant « input » qui permet l’acquisition des données que l’utilisateur aura enregistré et pour finir il permet également l’utilisation des multimédias tels que des images et des vidéos.

#### Le fonctionnement de l’HTML

Le code Html fonctionne grâce à un system de balise en voici un exemple   
« <html> – </html> » comme vous pouvez le voir celle-ci sont toujours par deux la première « html » sert à ouvrir la balise la deuxième « /html » commençant par un « / » sert à la fermer. Dans ces balises on peut y rajouter des options comme par exemple « <html class= « Noms\_de\_la\_classe »> – </html>» dans ce cas l’option ce nomme « class » celle-ci permet de créer un groupe qui permet à plusieurs balise de partager le même code CSS (explication en dessous).

Voici une liste des balises les plus utiliser ainsi que leur utilité :

**Balises sectionnantes :**

Ces balises nous servent à construire les éléments de base d’un site web.

* <header> – </header> : en-tête
* <nav> – </nav>: liens principaux de navigation
* <footer> – </footer> : pied de page
* <section> – </section> : section de page
* <article> – </article> : article (contenu autonome)
* <aside> – </aside> : informations complémentaires.

**Balises de tableau :**

Ces balises ont pour but de créer un tableau structuré avec des lignes et des colonnes.

* <table> – </table> : tableau
* <tr> – </tr>: ligne de tableau
* <th> – </th> : cellule d’en-tête
* <td> – </td> : cellule

**Balises de structuration du texte :**

Ces balises permettent de disposer des blocs (container) dans la page contenant du texte, des images ou encore des menus.

* <h1 (à 6)> – <h1> : titre de différents niveaux
* <img /> : image grâce aux attributs src (adresse de l’image) et alt (texte de  
  remplacement).
* <a> – </a> : lien hypertexte grâce à l’attribut href.
* <br /> : retour à la ligne
* <p> – </p>: paragraphe

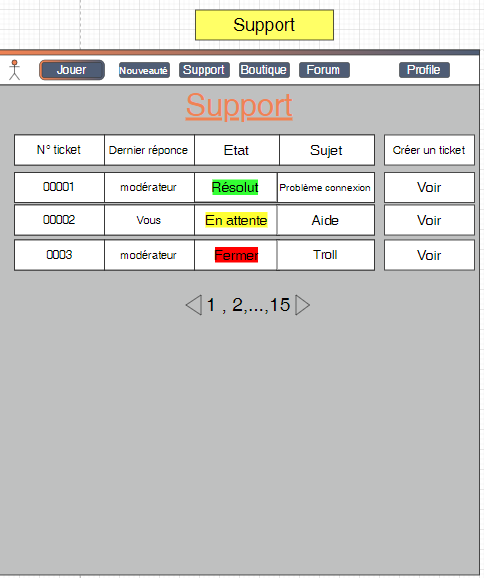
### Qu’est-ce que CSS ?

CSS (Cascading style sheets) est un langage informatique utilisé dans le design créé en 1926. Celui-ci permet d’effectuer des actions comme définir un fond d’écran, changer la police, la taille, voire même la couleur d’un texte mais il permet aussi de définir la taille et la position de tous les éléments affichés sur l’écran. Je ne pourrai pas citer toutes les fonctionnalités de celui-ci tellement il y en a.

## Les croquis

Après avoir fait une analyse des besoins ainsi que la réflexion pour le langage de programmation, nous pouvons nous attaquer au premier croquis du design de notre site web. Pour ce faire, j’ai utilisé le site draw.io.

Les voici :

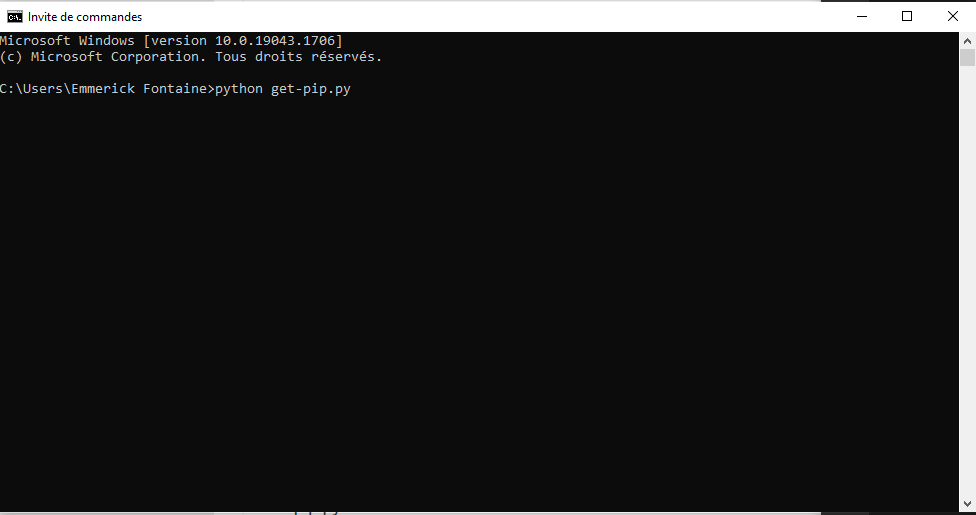


Les croquis m’ont apporté une direction à suivre pour la création et le design des pages et donc ne sont pas définitifs.

## Mise en place de l’environnement

Dans un premier temps, j’installe python en me rendant sur python.org qui est un simple exécutable à lancer.

Une fois python installé, j’implémente une extension se nommant PIP qui me permet d’installer d’autres extensions très rapidement et facilement et s’installe via le cmd (l’inviter de commande) de Windows.



PIP étant installé, je peux maintenant mettre Flask et les extensions utilisées dans le projet grâce à ces commandes aussi utilisées dans le cmd.

|  |
| --- |
| pip install Flask |
| pip install flask-login |
| pip install Flask-SQLAlchemy |
| pip install Flask-Mail |

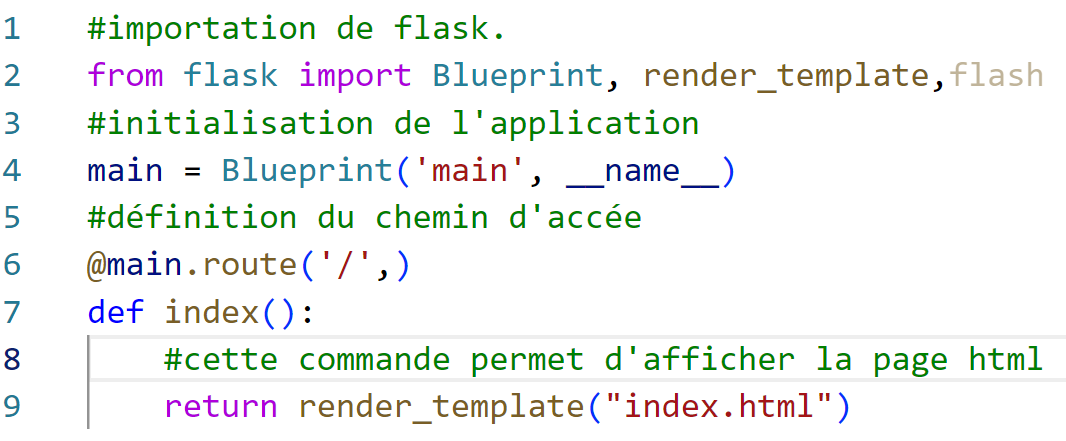
## Création du code

### Initialisation

Maintenant que la mise en place de l’environnement est terminée, nous pouvons nous attaquer au vif du sujet.

Pour m’entraîner, j’ai d’abord créé une première page (index) qui sera ma page principale. Pour ce faire, j’ai créé un dossier se nommant TFE et un fichier du dossier se nommant main.py (fichier python). Celui-ci permettra d’effectuer toutes les principales commandes.

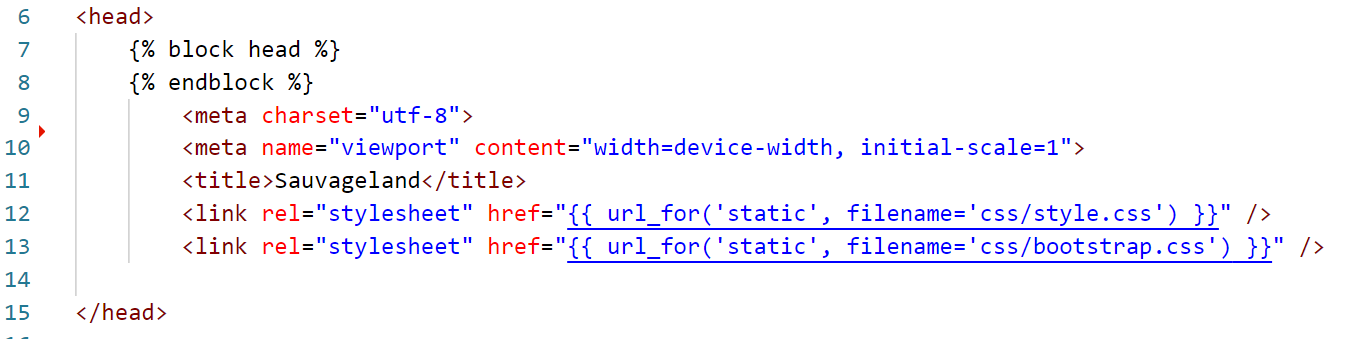
Voici à quoi il ressemble avec l’utilité écrit en commentaire (vert) :



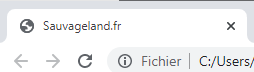
Ces lignes de commande permettent d’initialiser Flask et de rediriger la route principale (URL) vers le fichier index.html.

### Création de la barre de navigation

Dans Flask, la première chose à faire est un fichier de base se nommant base.html. Celui-ci sera incorporé dans toutes les pages en même temps, pour ne pas charger inutilement le code. Ce fichier contrairement aux autres ressemble le plus à un fichier HTML normal.

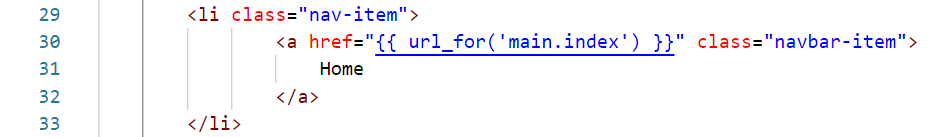
Nous allons d’abord voir la balise « head » en détail. Celle-ci est le code que la page doit charger en premier. La voici :

Je vais vous expliquer l’utilité de chacune de ces lignes :

1. Permet d’ouvrir la balise « head »
2. Permet d’ouvrir une porte se nommant « head » (on y reviendra plus tard)
3. Permet de fermer la porte « head »
4. Permet de dire au navigateur que notre site utilise le langage   
   « utf-8 » (langage le plus utilisé)
5. Permet d’optimiser la page pour un affichage mobile qui a généralement un écran plus petit que celui d’un ordinateur.
6. Permet d’afficher le titre de l’onglet de la page.
7. Permet de faire le lien de connexion entre la page CSS principale de tout le projet.
8. Permet de faire le lien de connexion entre la page CSS de Bootstrap de tout le projet.

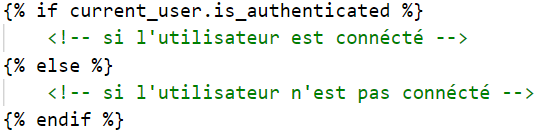
Maintenant que nous avons vu le code de la balise « head », nous allons voir la balise « body » qui contient tout le code qui sera afficher sur la page.

Ce code étant un peu long et répétitif, je ne vais pas vous expliquer ligne par ligne toutes les commandes mais en faire un résumé.

J’ai créé un contenaire contenant une liste de tous mes liens à l’intérieur en y ajoutant des class (vue précédemment) contenue dans le CSS de Bootstrap pour la mise en page.

Voici une partie du code permettant d’être rediriger vers la page « home » :

Comme dans mon site il y a un système d’indentification, certains liens comme l’enregistrement ou la connexion ne doivent plus être visible une fois l’utilisateur connecté ou d’autres comme le shop ou le profile qui doivent être visible une fois l’utilisateur connecté.

Pour ce faire Flask permet d’intégrer des commande python dans le code notamment celle-ci :

Ces lignes de commande servent à regarder si l’utilisateur est connecté.

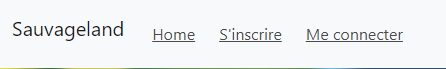
Si oui, les lignes de commande situées à « <!—Si l’utilisateur est connecté 🡪 » seront exécutées. Dans notre cas, l’affichage des pages profile, shop et ticket.

Si non, les lignes de commandes situées à « <!—Si l’utilisateur n’est pas connecté 🡪 » seront exécutées. Dans notre cas, l’affichage des pages de connexion et d’enregistrement.

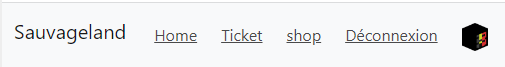
J’ai aussi fait cela pour les administrateurs, sauf qu’on utilisera « if current\_user.admin == « 1 ».

En voici les résultats finaux :

* Si l’utilisateur n’est pas connecté :



* Si l’utilisateur est connecté :



* Si l’utilisateur est connecté et est administrateur :



### Système d’identification

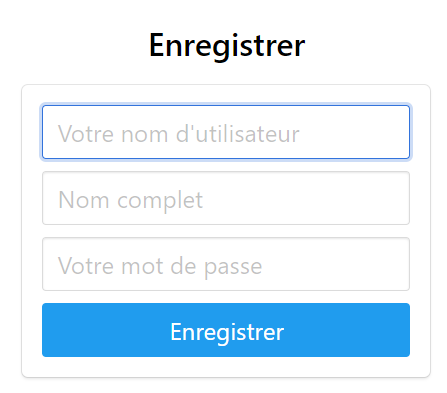
Précédemment, je vous ai parlé de vérifier si l’utilisateur est connecté ou non. Pour cela, nous allons voir le système d’identification ensemble.

Pour commencer, nous aurons besoin de Flask-login ainsi que Flask-sqlalchemy mais je parlerai de celui-ci juste après.

Après quelques recherches sur le sujet, j’ai trouvé sur GitHub   
( <https://github.com/mecmartini/python-flask-login-example> ) un exemple de code pour pouvoir s’enregistrer et se connecter. Je vais l’utiliser comme base du code car celui-ci intègre quasiment tout ce dont j’ai besoin pour ce site.

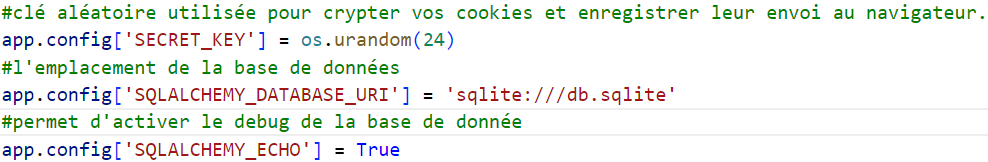
Après analyse du code, je décide de modifier la base de données que je vais expliquer au point suivant et d’y rajouter une donnée se nommant « admin ». Cette donnée sert à définir si l’utilisateur a les permissions d’administrateur ou non (1 ou 0). J’ai également modifié le formulaire d’enregistrement pour qu’il puisse récupérer l’adresse mail de l’utilisateur. De ce fait, j’ai également dû modifier la base de données pour pouvoir accueillir cet email.

En voici le résultat final :



### Les bases données

Les bases de données, dans n’importe quelle application ou site web, sont très importantes et utiles. Elles permettent de stocker toutes sortes de choses comme du texte, des nombres… et donc pouvoir y intégrés des paramètres, des conversations, les données des utilisateurs, …

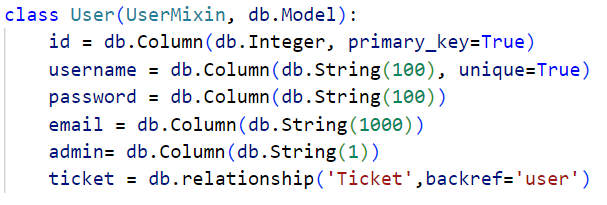
Pour utiliser les bases de données, nous allons utiliser l’extension SQL-Alchemy.  
Dans un premier temps, nous allons initialiser les paramètres dans un fichier se nommant « \_\_init\_\_.py » (fichier de configuration qui se lance en premier).

Voici ses lignes de commande avec leur explication en commentaire (vert) :

Maintenant que la configuration de la base de données est créée, il faut y insérer les tables de données. Pour ceci, il faut créer un fichier « models.py » et y définir le nom et les données des tables.

Voici le code de chaque base de données :

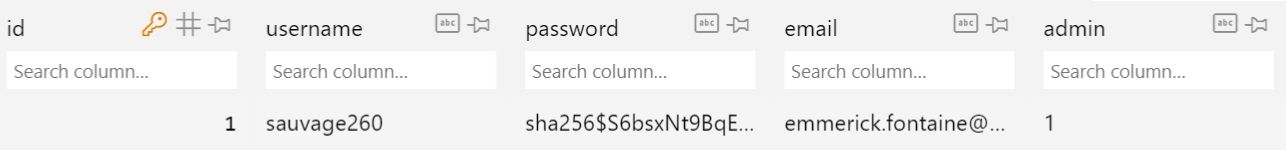
* La table des données d’enregistrement de l’utilisateur :

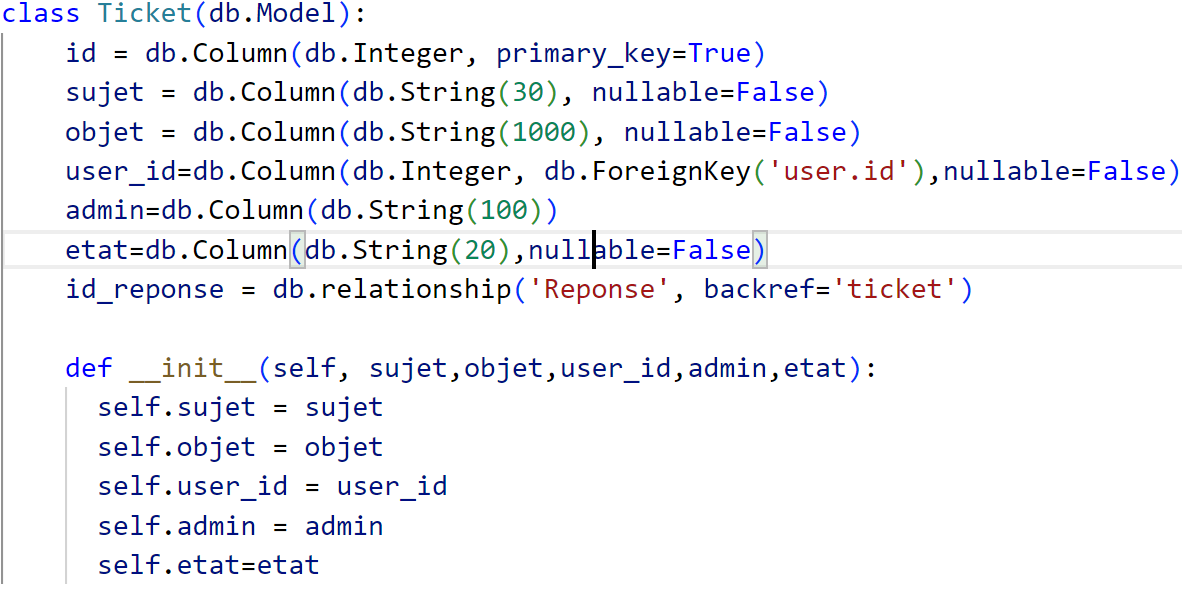


Généralement, nous commençons toujours par un « id » (numéro d’identification) qui est une clé primaire (primary\_key=True) qui permet de rendre ce nombre unique à la table.

Db.Colum(db.String(##)) signifie que la donnée sera de type text avec maximum ## caractère.

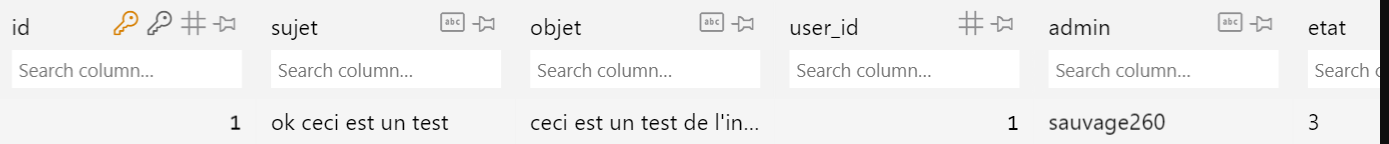
Voici un exemple d’enregistrement de donnée :

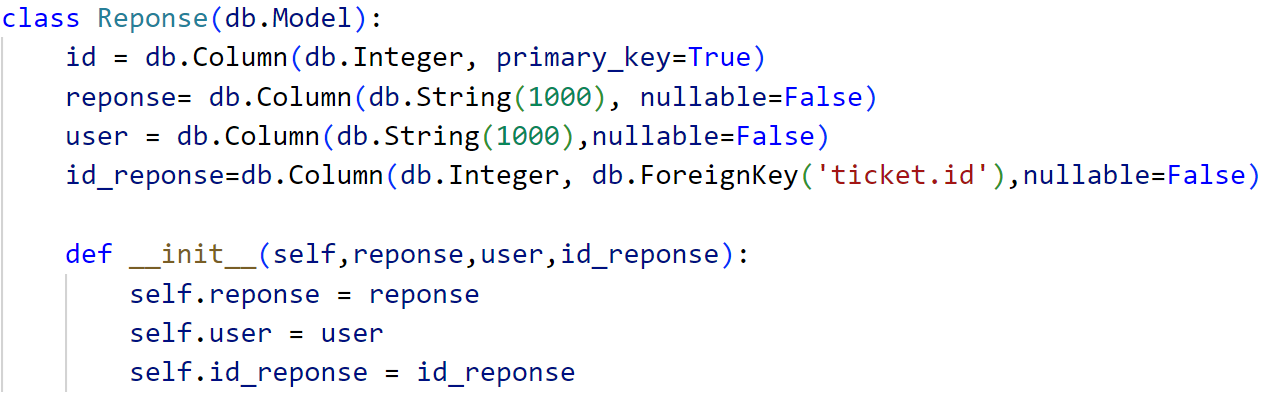


* La table des tickets de-u support :

« Def \_\_init\_\_ » permet de pouvoir enregistrer des données dans la base de données par la suite. Contrairement aux données des utilisateurs, qui est intégré dans le code de base de Flask-login.

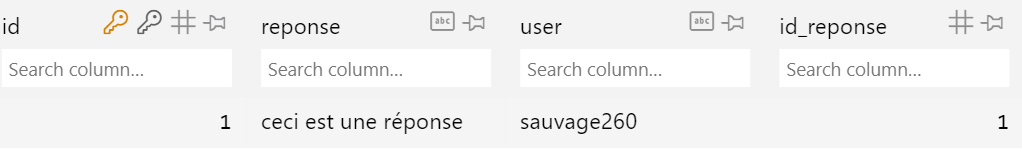
Voici un exemple d’enregistrement de donnée :



* La table des réponses des ticket :

Celle-ci est reliée à la table ticket pour pouvoir joindre les réponses aux demandes.

Voici un exemple d’enregistrement de donnée :



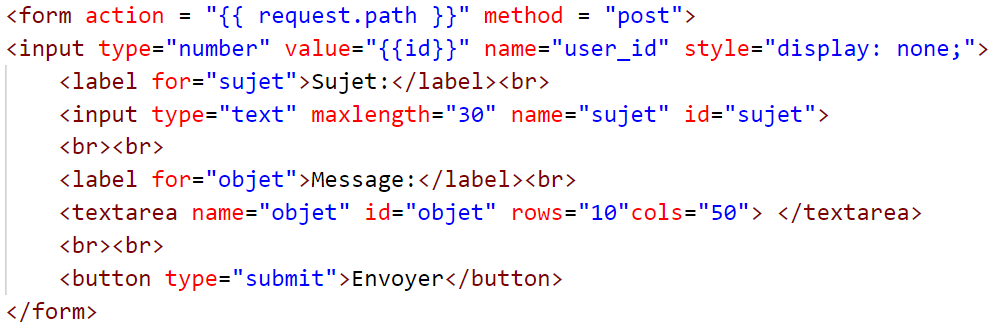
### Les tickets

Maintenant que nous avons vu les bases de données, nous pouvons nous attaquer à la partie la plus complexe et longue du projet : les tickets. Ceux-ci permettent à l’utilisateur de pouvoir contacter rapidement et facilement les administrateurs pour obtenir de l’aide ou des réponses à leurs questions.

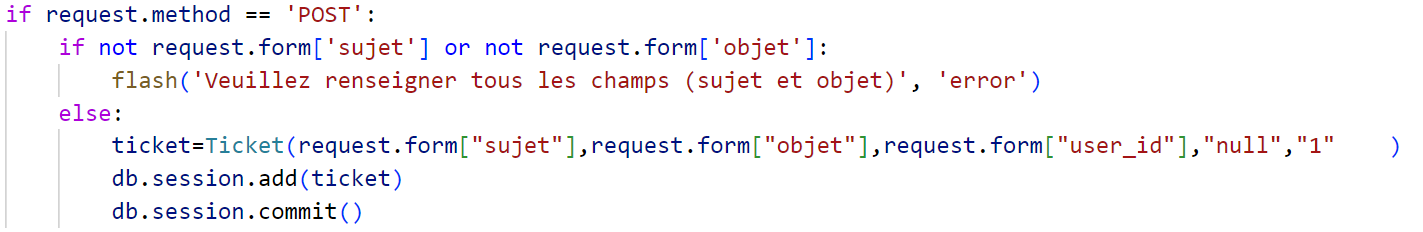
#### Création des tickets

La chose essentielle dans la création du ticket est que l’utilisateur doit savoir facilement le faire.

Dans un premier temps, je mets en place les « input » (formulaires) dans la page web en HTML ainsi que le bouton d’envoie.



Comme vous pouvez le remarquer, j’ai utilisé une balise « form » qui permet de pouvoir récupérer le formulaire de l’utilisateur dans le fichier « main.py » (fichier principale de Flask) à l’aide de ces commandes :



Ces commandes permettent de vérifier si tout le formulaire d’envoie de ticket est complété par l’utilisateur. S’il l’est, les données seront envoyées dans la base de données sinon cela affichera un message d’erreur à l’utilisateur.

#### Affichage des tickets utilisateurs.

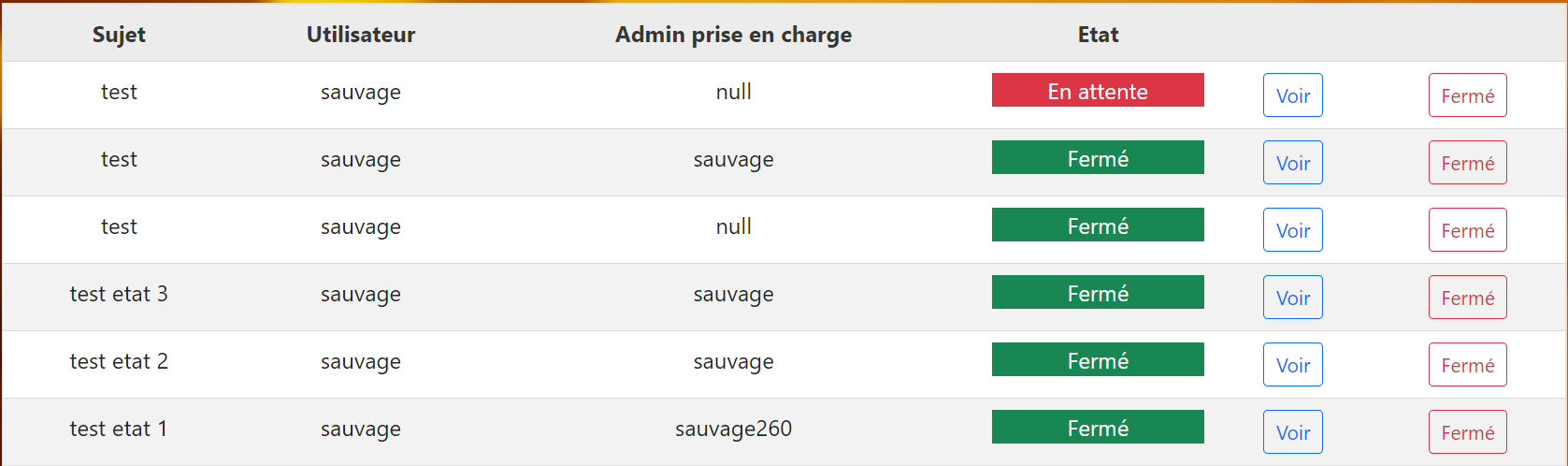
L’utilisateur doit pouvoir voir ses tickets pour y suivre les réponses et le statut de celui-ci. Pour ce faire, j’ai créé une liste de tous les tickets de l’utilisateur en-dessous du formulaire d’envoie.

Pour commencer, j’ai dû effectuer une requête à la base de données ticket :



Cette requête permet d’obtenir les tickets créés par l’utilisateur lui-même et de les trier par ordre décroissant pour pouvoir avoir le dernier ticket créé en premier.

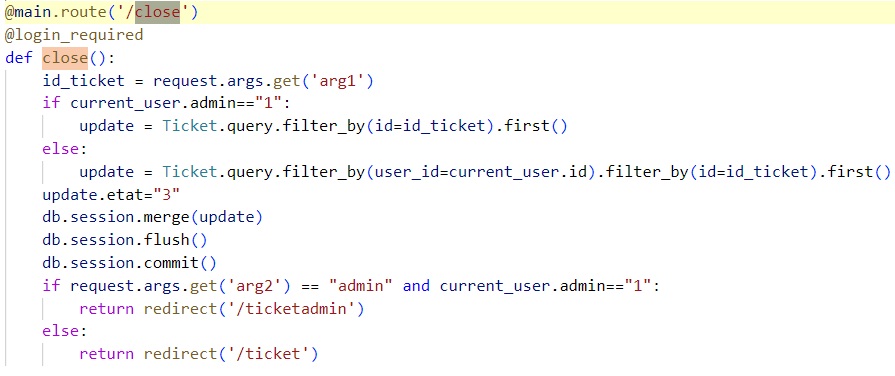
Il me suffit de faire une boucle sur la requête dans l’HTML pour afficher les résultats et je les ai placés dans un simple tableau.



Vous pouvez voir qu’il y a deux boutons « Voir » et « Fermé ».

On va d’abord s’occuper du bouton « Fermé » :

Si l’utilisateur a pu résoudre son problème de lui-même, il a la possibilité de fermé son ticket. Son fonctionnement est très simple, il redirige vers l’URL « /close ?arg1=id\_ticket » pour que le code situé dans « main.py » puisse récupérer l’ID du ticket à fermer et le fermer. Bien entendu en vérifiant que le ticket soit bien à l’utilisateur qui veut le fermer ou que l’utilisateur soit un administrateur.



Pour le bouton « voir », dans le même principe que le bouton « fermé », celui-ci redirige vers une URL contenant un argument avec l’ID du ticket (/voirticket ?arg1=id\_ticket) à la différence qu’il ne ferme pas le ticket mais y montre les détails ainsi que les réponses provenant des administrateurs ou de l’utilisateur lui-même. Mais également de pouvoir répondre au ticket.

