

**Technologies WebII / ProgrammationWeb:**

Rapport de projet Ensisingerie

|  |  |
| --- | --- |
| Antropius Simon  Desmonteix Maxence  Sénéchal Louis | Année universitaire  2023-2024 |

Table des matières

[Introduction 3](#_Toc155492627)

[Organisation des fichiers 3](#_Toc155492628)

[Application ‘users’ 3](#_Toc155492629)

[Réinitialisation des mots de passe : 4](#_Toc155492630)

[Application ‘rooms’ 4](#_Toc155492631)

[Modèles : 4](#_Toc155492632)

[Page de salon : 5](#_Toc155492633)

[Page statistiques : 5](#_Toc155492634)

[Page de gestion des utilisateurs : 5](#_Toc155492635)

[Views : 5](#_Toc155492636)

[Javascript : 6](#_Toc155492637)

[Application ‘main’ 6](#_Toc155492638)

[Dossier static commun à toutes les apps 7](#_Toc155492639)

[Sécurisation du site 7](#_Toc155492640)

# Introduction

Ce rapport a pour but d’expliquer les choix faits lors de la création du site web Ensisingerie, créé par le groupe Antropius/Desmonteix/Sénéchal. Le site a été créé avec les technologies suivantes :

* Django pour la majorité du site
* JavaScript + JQuery pour les actions à effectuer en temps réel, sans recharger la page
* CSS pour le style
* La bilbiothèque de JavaScript Chart.js, uniquement sur la page de statistiques, non demandée dans les consignes. Nous avons bien sûr demandés l’autorisation pour l’utiliser, tant que cela restait pour la page de statistiques.

Le compte administrateur suivant est à votre disposition :  
Pseudo : Admin

Mot de passe : adminpass-singe%Web

# Organisation des fichiers

Notre site web est divisé en 3 applications :

* users : concerne tout ce qui est lié en majorité aux utilisateurs
* rooms : concerne tout ce qui est lié en majorité aux salons
* main : concerne le reste (page d’accueil, d’aide, d’erreurs)

Cela permet de séparer clairement les responsabilités et de rendre chaque app éventuellement réutilisable dans un autre site web bien qu’il existe certaines dépendances entre les apps.  
Dans la même logique, chaque application contient son propre dossier ‘static’, contenant les fichiers CSS, Javascript, éventuellement JSON et images.

Un dossier ‘static’ supplémentaire se trouve au même niveau que les dossiers des apps et contient les fichiers statiques utilisés dans plusieurs apps.  
  
Toujours dans le but de réduire les dépendances entre apps, nous avons décidés que chaque app inclurait son propre fichier base.html dont seront dérivés les autres fichiers HTML de l’app, même si en pratique ils sont tous identiques puisque tout le site utilise la même base.

# Application ‘users’

L’application ‘users’ concerne, comme son nom l’indique, les utilisateurs. Concrètement, elle contient la logique et les pages permettant d’effectuer les actions suivantes :

* Créer un compte
* Se connecter au site
* Se déconnecter
* Changer son mot de passe
* Réinitialiser son mot de passe
* Accéder à profil pour éventuellement y modifier ses informations

L’entièreté de cette app a été faite en s’appuyant sur le modèle User et les différentes pages/formulaires proposés par Django par défaut. Aucun nouveau modèle n’est donc défini dans celle-ci.

Concernant les views, toutes les pages générées le sont en dérivant les classes proposées par Django. Ceci permet de réutiliser un code déjà écrit et de s’assurer que ces pages sont sécurisées et conformes aux pages ‘classiques’ d’un site web.

## Réinitialisation des mots de passe :

Il est possible de réinitialiser son mot de passe en cas d’oubli. Pour cela, nous utilisons la solution Simple Mail Transfer Protocol intégrée de Django couplée au service SMTP de Gmail.

Une adresse mail créée spécialement pour le site envoie alors un mail contenant le lien de réinitialisation.

Attention : notre site n’étant probablement pas ‘standard’ par rapport aux autres, les messages envoyés sont souvent considérés comme étant du courrier indésirable par les boites mails. Vérifiez donc vos spams.

# Application ‘rooms’

L’application ‘rooms’ est la plus grosse des trois. Elle contient le nécessaire pour effectuer les actions suivantes :

* Créer un salon
* Rejoindre un salon
* Accéder à un salon
* Envoyer/supprimer un message
* Gérer les utilisateurs d’un salon (inviter, renvoyer et accepter une demande pour rejoindre le salon)
* Accéder aux statistiques de chaque salon

## Modèles :

Cette app contient 4 modèles, représentant chacun un objet :

* La classe Room représente un salon. Il est caractérisé par son nom, son propriétaire, une liste des utilisateurs présents dans le salon, une liste des utilisateurs ayant déjà été bannis du salon et un attribut ‘private’ représentant le caractère public ou non.
* La classe Message représente un message. Un message est représenté par son auteur, le salon dans lequel il l’a envoyé, sa date de publication, son texte et un attribut ‘is\_deleted’ représentant le fait que le message ait été supprimé.
* La classe JoinRequest représente une demande qu’un utilisateur a faite pour rejoindre un salon. Il contient donc l’émetteur de la demande, le salon en question et un attribut ‘is\_approved’ indiquant si la demande a été approuvée ou non.
* La classe RoomInvitation représente une invitation du propriétaire d’un salon à un autre utilisateur pour rejoindre ce salon. Il contient l’utilisateur invité et le salon en question.

## Page de salon :

Sur une page de salon se trouve un historique des messages envoyés sur celui-ci avec des champs permettant d’envoyer un message et d’y intégrer des émojis.

Le chargement et rafraichissement des messages se fait en Javascript, toutes les 3 secondes afin d’avoir un système de messagerie en temps réel. Un message envoyé par un utilisateur sera donc vu par un autre utilisateur de ce salon sans que ce dernier n’ait à recharger la page.

Un bouton supprimer permet de supprimer un message. Celui-ci n’est en fait pas réellement supprimer mais le ‘Ce message a été supprimé’ apparait à la place, permettant de voir qu’un message qui a été supprimé se trouvait là. Seul le propriétaire du salon peut supprimer n’importe quel message. Un utilisateur lambda ne peut supprimer que ses propres messages.

Des boutons ‘Voir plus’, ‘Voir moins’ et ‘Voir tous’ permettent de contrôler le nombre de messages affichés à l’écran, les derniers étant bien sûrs toujours affichés.

Les émojis sont gérés de la manière suivante :

Une fonction dans les vues django de l’application permet d’ouvrir et lire le contenu d’un fichier json dans les statics de l’app. Il contient des émojis triés par catégorie accompagnés de plusieurs informations comme le nom de l’émoji ainsi que son symbole (c’est cette dernière information qu’on viendra récupérer à l’aide du javascript ainsi que les noms des catégories d’émojis pour les trier). Le fichier json a été récupéré à l’adresse suivante : [unicode-emoji-json/data-by-group.json at main · muan/unicode-emoji-json (github.com)](https://github.com/muan/unicode-emoji-json/blob/main/data-by-group.json). L’inconvénient de cette méthode est que les émojis sont inscrits dans un fichier json d’une certaine manière alors si le nombre d’émojis venait à augmenter en fonction de la version UNICODE on ne pourrait pas les récupérer ou si on voudrait récupérer les émojis d’un fichier json qui ne serait pas formaté de la même manière alors ce serait également impossible sans changer les fonctions javascript.

On trouve également un ou deux boutons permettant d’accéder aux pages ‘Statistiques’ et ‘Gestion des utilisateurs’.

## Page statistiques :

Cette page permet à n’importe quel utilisateur d’un salon de consulter quelques statistiques sur ce salon. Le nombre de messages envoyés par jour est affiché sous la forme d’un graphique utilisant Chart.js. C’est le seul endroit du site où nous utilisons cette bilbiothèque.

## Page de gestion des utilisateurs :

Cette page n’est accessible que par le propriétaire du salon. Il peut y inviter un utilisateur, en renvoyer un du salon et consulter les demandes des utilisateurs pour entrer dans le salon pour éventuellement les accepter.

La recherche d’utilisateurs se fait en Javascript, permettant un affichage dynamique, ce qui rend le site agréable à utiliser.

## Views :

Dans cette app, la majorité des views sont de simples fonctions, permettant un contrôle plus fin sur la logique de l’application. Même si le site est fait de telle façon qu’un utilisateur ne puisse pas avoir directement accès à une fonctionnalité qui lui est interdite (le bouton ‘supprimer’ n’apparait par exemple pas sur les messages dont l’utilisateur n’est pas originaire), des sécurités supplémentaires sont implémentées dans les views :

* La majorité des fonctions sont précédées du décorateurs @login\_required, proposé par Django. Lorsqu’un utilisateur tente d’accéder à cette fonction sans être connecté, il est automatiquement redirigé vers la page de connexion.
* Pour les fonctions censées être appelées par du Javascript, les méthodes @require\_GET et @require\_POST contrôlent que la fonction a bien été appelée avec la méthode prévue.
* De plus, chaque fonction vérifie un certain nombre de condition avant d’effectuer l’action demandée et redirige vers une page d’erreur ou renvoie une réponse JSON en cas de problème, permettant de sécuriser le site. Par exemple, dans la fonction remove\_user, appelée pour éjecter un utilisateur d’un salon et pour quitter un salon, on vérifie d’abord que l’utilisateur appelant la fonction est soit le propriétaire du salon soit la personne à retirer du salon puis on regarde si l’utilisateur à retirer est bien dans le salon et enfin si le l’utilisateur à retirer n’est pas le propriétaire du salon.

## Javascript :

Notre Javascript utilise beaucoup les requêtes AJAX car nous avons souvent besoin de calculer/rechercher une petite partie seulement de la page. Cela permet d’éviter non seulement l’inconfort d’un rechargement complet de page mais également le coût calculatoire que représente ce rechargement.

# Application ‘main’

L’application ‘main’ contient la gestion de la page d’accueil, d’aide et d’administration.

Un utilisateur connecté trouvera sur la page d’accueil la liste de ses salons, des invitations reçues (qu’il pourra accepter ou refuser), des demandes en attentes pour un salon et des demandes qui ont été acceptées. Il peut également y recherche un salon pour le rejoindre ou faire une demande pour le rejoindre et créer son propre salon.

Les différentes listes sont affichées avec Javascript car il faut pouvoir en recharger certaines pendant l’utilisation. Par exemple, si l’utilisateur s’inscrit dans un salon, il faut retirer le salon dans lequel il vient de s’inscrire des résultats de sa recherche et l’ajouter à la liste de ses salons, tout ceci sans recharger la page.

Il est important de noter que bien que le Javascript pour l’application main se trouve dans la même application, le Javascript fait lui appel à des views de l’application ‘rooms’, par exemple pour renvoyer la liste des salons de l’utilisateur. Nous avons en effet décidé de donner cette responsabilité à l’application ‘rooms’ car cela nous semblait plus pertinent.

Sur la page de création de salon, une checkbox permet de définir le caractère publique ou privé du salon.

Enfin, une page d’administration est accessible aux super users. On peut y supprimer complètement un utilisateur, ce qui a pour effet de supprimer tous ses messages, ses invitations et ses demandes et de transférer la propriété de ses salons à l’utilisateur l’ayant supprimé. Pour tester cette fonctionnalité, vous pouvez utiliser le compte administrateur donné dans l’introduction de ce rapport.

Pour ajouter un peu de vie au site, un carrousel codé en Javascript est chargé d’afficher aléatoirement et toutes les 20 secondes une photo d’un singe parmi les 12 disponibles ainsi qu’un court texte explicatif stocké dans un fichier JSON.

# Dossier static commun à toutes les apps

Sur toutes les pages du site se trouve un bouton permettant de passer du thème clair au thème sombre et inversement. Ce changement se fait en Javascript au moyen d’un ‘toggleClass’ sur l’élément HTML ‘body’. Un toggleClass permet d’ajouter une classe à un élément ou à la retirer si elle y est déjà, ce qui permet dans notre cas, à l’aide d’un CSS adapté, de choisir des couleurs différentes dans le cas où l’élément body possède la classe ‘dark-theme’ ou non. De plus, var theme = localStorage.getItem('theme') permet de sauvegarder le thème choisi par l’utilisateur pour que celui-ci soit gardé en mémoire entre les différentes pages du site.

# Gestion du cache des navigateur pour les fichiers static

La ligne :

STATICFILES\_STORAGE = "django.contrib.staticfiles.storage.ManifestStaticFilesStorage" dans settings.py permet d’ajouter un hash MD5 à la fin du nom du fichier (par exemple le fichier **css/styles.css** deviendrait **css/styles.55e7cbb9ba48.css)** et qui change en fonction du contenu du fichier. Le but est d’apporter du cache-busting aux fichiers static.

Ainsi lorsque le contenu du fichier static change alors le hash change également, forçant les navigateurs à télécharger le nouveau fichier plutôt que d’utiliser une version en cache. Cela permet d’optimiser la gestion des fichiers static et améliorer la gestion du cache comme les utilisateurs seront assurés d’avoir toujours la dernière version des fichiers. Il faut cependant pour que cela fonctionne que DEBUG = False dans settings.py et il faut lancer la commande python manage.py collectstatic au moindre changement dans un fichier static.

# Sécurisation du site

Les informations sensibles du fichier settings.py sont sécurisées lors du partage de fichiers sur GitHub grâce à Python Decouple. Ceci permet de stocker les valeurs réelles de certaines variables (SECRET\_KEY et EMAIL\_HOST\_PASSWORD notamment) dans un fichier .env indépendant qui lui n’est pas partagé.

# Adaptabilité du site

Un travail important a été réalisé en CSS pour s’adapter aux différents types de supports, notamment aux smartphones et autres appareils disposant de peu de place horizontalement. N’hésitez pas à tester par vous-mêmes !