## Exercice 2

Table des matières

[Exercice 2 1](#_Toc69098911)

[Technologies 1](#_Toc69098912)

[La page de Connexion : 1](#_Toc69098913)

[Gestion du formulaire : 2](#_Toc69098914)

[Gestion des mots de passes 2](#_Toc69098915)

[Mot de passe sauvegardés en clair 2](#_Toc69098916)

[Mot de passe Haché 3](#_Toc69098917)

[Mot de passe Haché et Salé 6](#_Toc69098918)

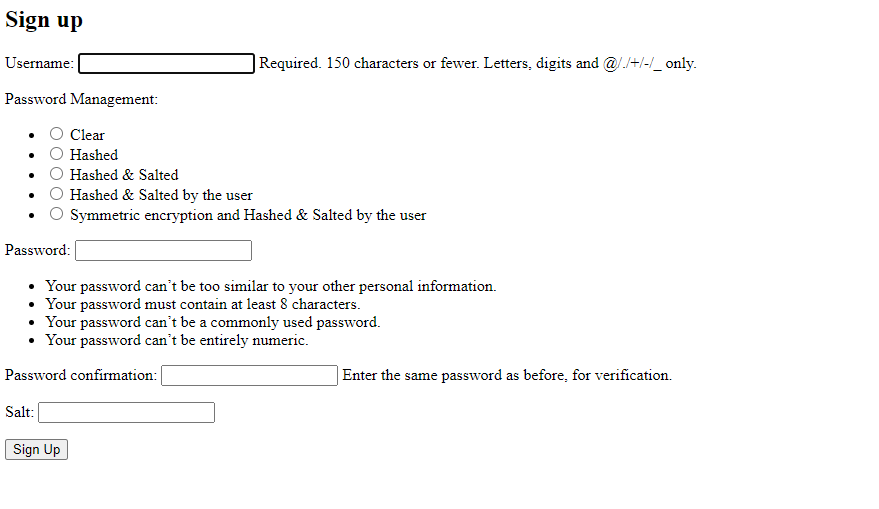
[Salé et Haché par l’utilisateur 7](#_Toc69098919)

## Technologies

Pour faire cet exercice, le Framework Python « Django » a été utilisé pour travailler plus rapidement non seulement pour coder la page de connexion mais aussi pour bénéficier des fonctions de hachage intégrées.

## La page de Connexion :

Une page d’inscription a été codée :



(Tous les champs sont requis même s’ils ne sont pas forcements utilisés (le champ « Salt » par exemple))

L’utilisateur choisit son mode de sauvegarde de mot de passe via les radios, ainsi, une fois le formulaire fournis on sauvegarde le mot de passe de la façon voulues dans la base de données sqlite3 fournies pas Django.

## Gestion du formulaire :

Quand le formulaire est soumis, selon le mode de sauvegarde mot passe choisit, on modifie le mot de passe du nouvel utilisateur de cette manière :

user.password = make\_password(response.POST[‘password2’], ‘Salt’, ‘Algo de hachage’)

user.save(update\_fields=[‘password’])

‘Salt’ correspond à ce qu’il y a dans le champ salt pour les modes où on en a besoin, sinon il est égal à None quand on ne sale pas.

‘Algo de hachage’ est un string qui correspond à l’identifiant des différents algorithmes intégrés à Django, on peut les trouvé dans le fichier hasher.py qu’on trouve dans les fichiers de Django (Accessible sous Windows en suivant le chemin Python39\Lib\site-packages\django\contrib\auth, où Python39 est le dossier où python est installé, version 3.9 ici)

## Gestion des mots de passes

Pour faciliter les tests on a utilisé la fonction de hachage MD5, fonctions déjà implémentée dans Django même si elles sont dépréciées.

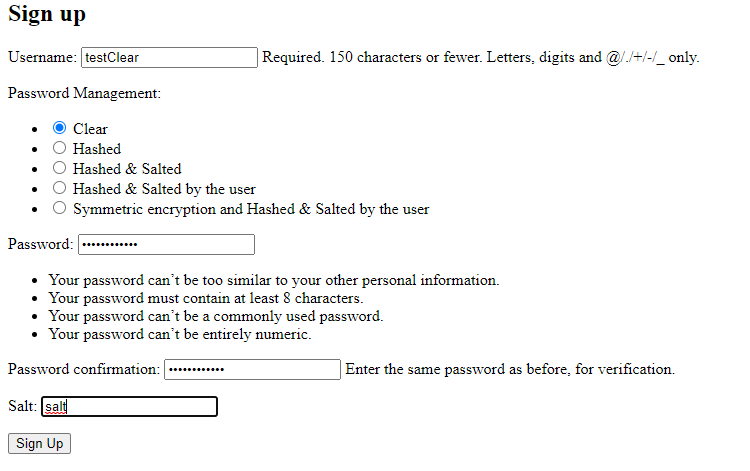
Dans la base de données avec les fonctions de hachage fournies par Django, les mots de passes sont stocké sous la forme « <algorithme>$<iteration>$<salt>$<hash> » ([Doc Django](https://docs.djangoproject.com/fr/3.1/topics/auth/passwords/)) la plupart du temps (ce n’est pas le cas pour le Unsalted MD5 par exemple).

### Mot de passe sauvegardés en clair

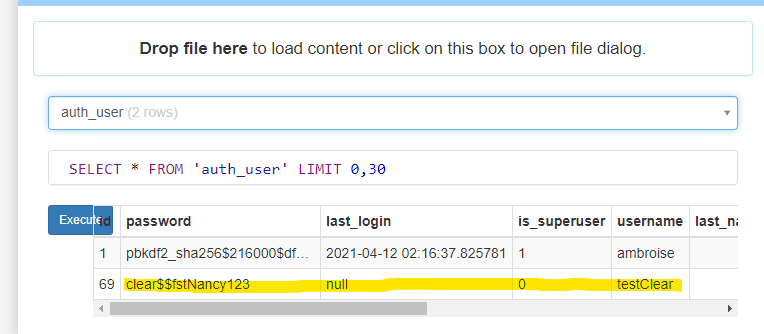
Pour sauvegarder les mots de passes en clair on ajoute une pseudo fonction de hachage pour utiliser make\_password en suivant le modèle des autres fonction prédéfinies :



De cette manière, dans la base de données avec comme mot de passe dans les champs password « fstNancy123 »



On trouve dans la base de données la ligne correspondant au nouvel utilisateur avec le mot de passe stocké en clair :



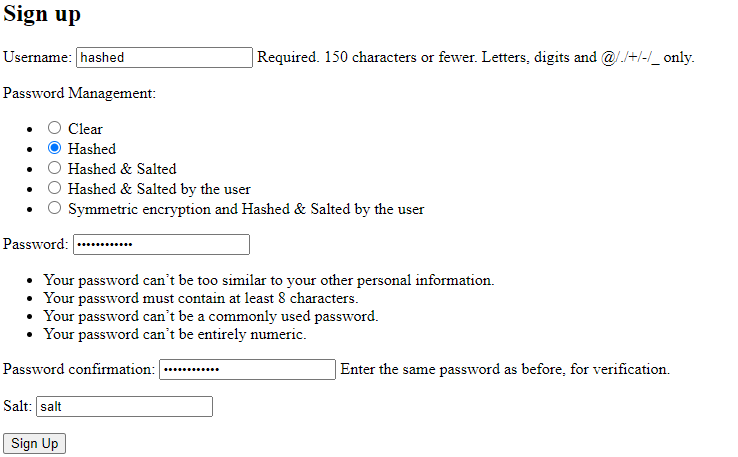
### Mot de passe Haché

Pour hacher le mot de passe, on fait appel à la classe « UnsaltedMD5PasswordHasher » du fichier hasher.py de Django via

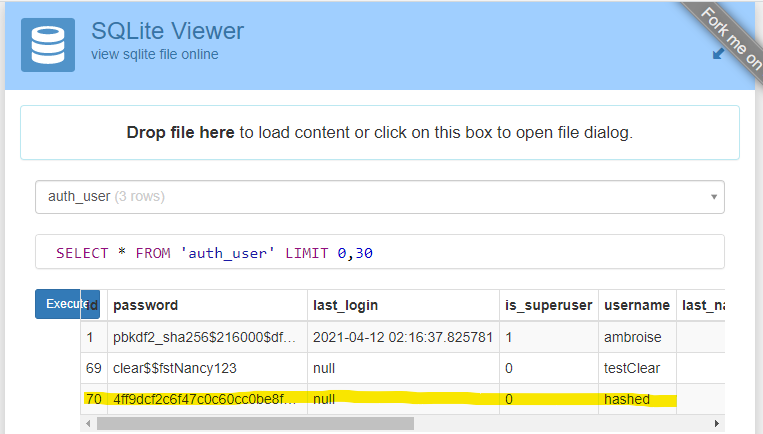


On remplit le formulaire et on précise le bon mode de gestion de mot de passe,

Mot de passe : « fstNancy123 »



La ligne suivante est ajoutée à la base de données :



Le hash est le suivant :

4ff9dcf2c6f47c0c60cc0be8f684d685

Un Hash issu de l’algorithme MD5 si on en croit dcode.fr



### Mot de passe Haché et Salé

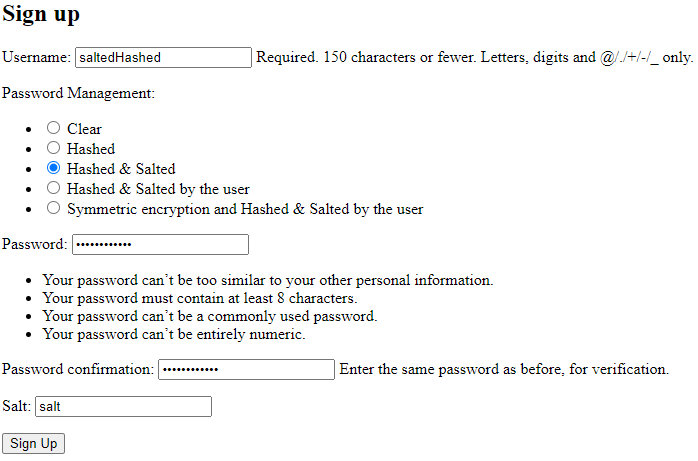
On procède dans les grandes lignes comme le cas précédent, sauf qu’il faut ajouté le salt et utiliser la classe « MD5PasswordHasher ».

On a donc lors de la soumission du formulaire :

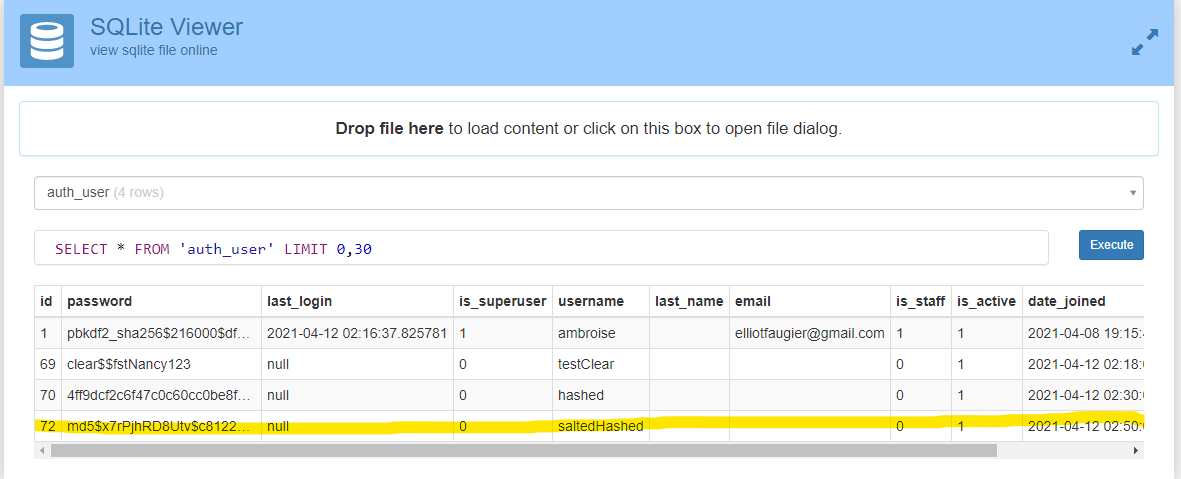


Le salt est choisis par Django et est sauvegardé dans la base de données

On soumet le mot de passe « fstNancy123 »:



On obtient la ligne suivant dans la base de données :



Avec dans le champ password :

md5$x7rPjhRD8Utv$c812233bdb6d1b822bc38027e4663607

(algo $ salt $ hash)

On vérifie que le hash est correct sur dcode.fr :



### Salé et Haché par l’utilisateur

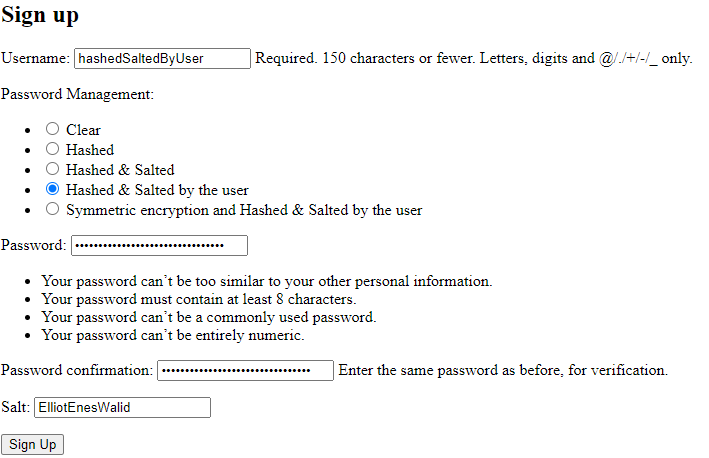
Les mot mots de passes haché et salé sont toujours stocké sous la forme algo$salt$hash, donc on a juste a concaténé correctement les informations fournies par l’utilisateur après la soumission du formulaire :



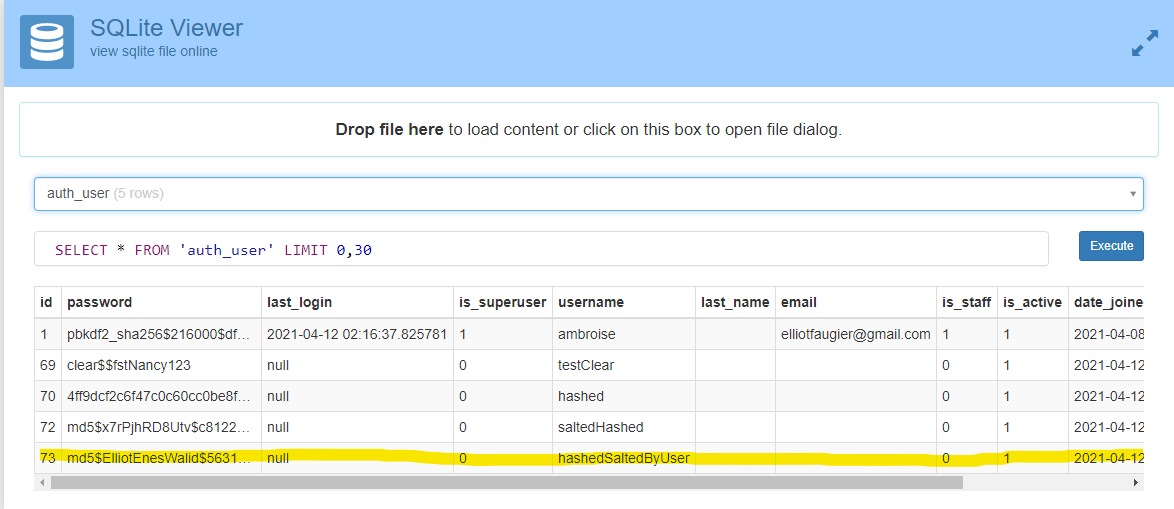
Ici le sel sera « ElliotEnesWalid » et le mot de passe « fstNancy123 », on fournira donc le hash de « ElliotEnesWalidfstNancy123 » qui correspond à « 563125da4ff824c47058347b02eaceab » dans les champs password.



On a alors ce formulaire :



Une fois soumis, cette ligne est ajoutée dans la base de données :

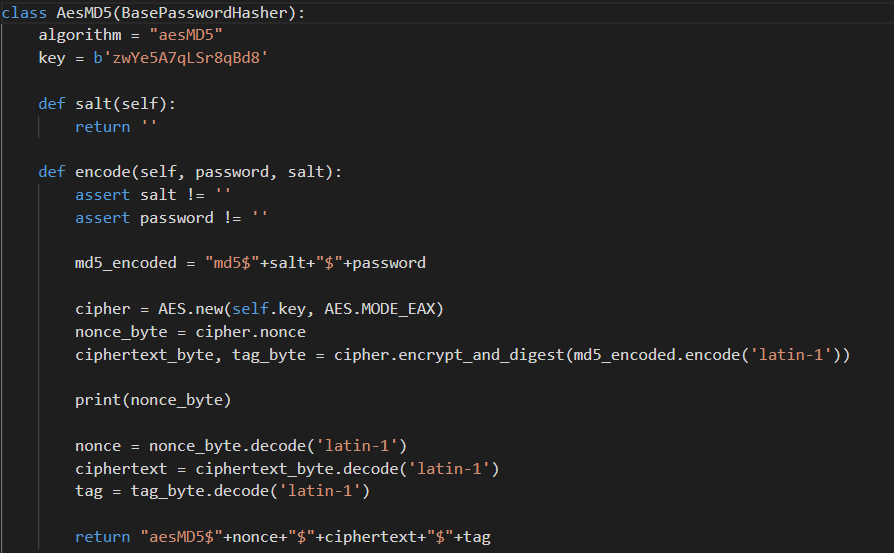


Avec dans le champ password :

md5$ElliotEnesWalid$563125da4ff824c47058347b02eaceab

### Salé, Haché par l’utilisateur et chiffré avec un chiffrement symétrique

Pour chiffrer les mots de passes on ajoute une pseudo fonction de hachage pour quand même utiliser make\_password. Cette classe s’appuie sur la librairie [pycryptodome](https://pycryptodome.readthedocs.io/en/latest/src/introduction.html) pour le chiffracge symétrique.



On chiffre en fait le string « md5 $ salt $ hash » qu’on pourra construire via les informations fournies dans le formulaire d’inscription. Ensuite, on convertit cette chaîne de caractère en objet bytes pour pouvoir le chiffrer (encodée en latin-1). Ces bytes sont ensuite chiffrés en utilisant l’algorithme de chiffrage symétrique [AES](https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html?highlight=aes) fournie par la librairie [pycryptodome](https://pycryptodome.readthedocs.io/en/latest/src/introduction.html). On obtient alors 3 objets bytes « nonce », « tag », et « cyphertext » qu’on convertit en texte pour pouvoir les stocker les 3 ensembles dans la base de données dans le champs password qu’on utilise d’habitude (La base de donnée étant imposée par Django).

On stock une chaîne de caractère « aesMD5$nonce$cipherText$tag » dans la base de données

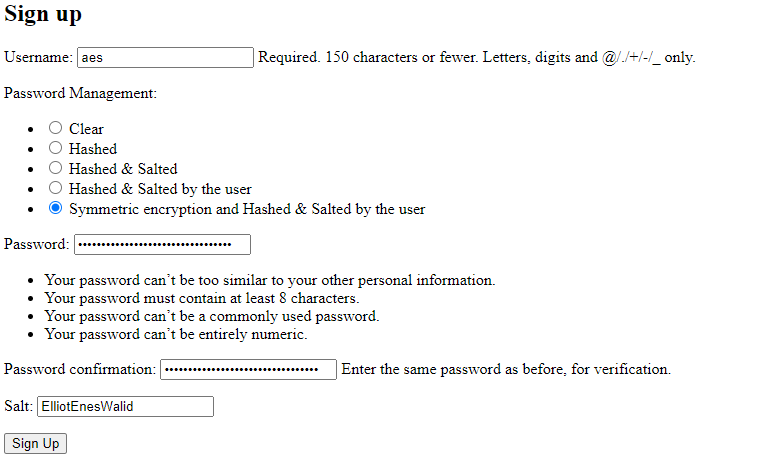


Pour vérifier un mot de passe on déchiffre le message avec la clé de la classe le triplet nonce, cipherText et tag qu’on extrait de la base de données et qu’on convertit préalablement en objet bytes.

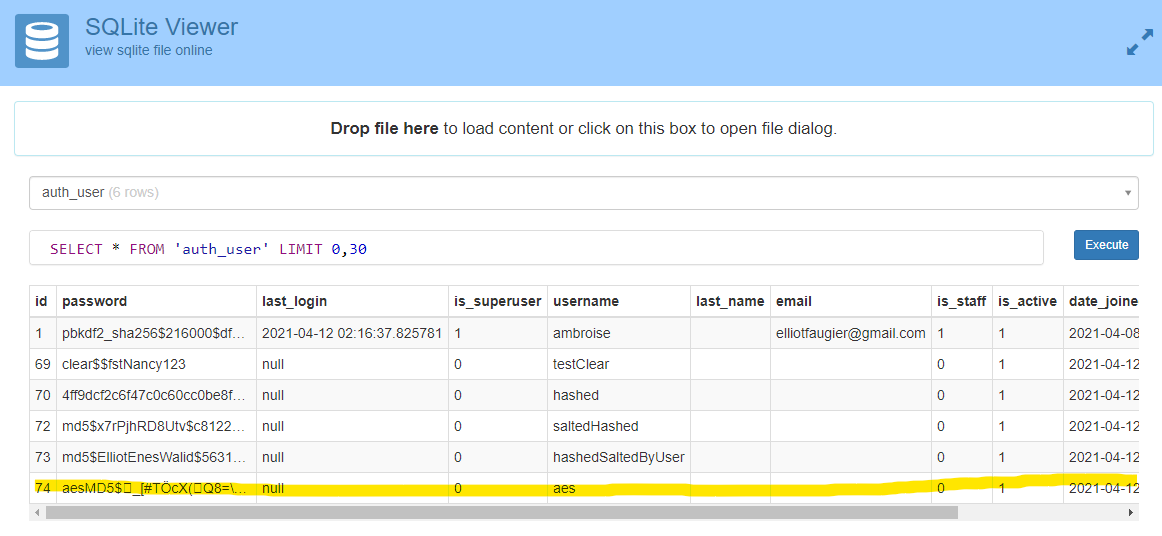
Une fois déchiffrée on obtient une chaîne de caractère « md5 $ salt $ hash » qu’on sait déjà traiter en utilisant la classe MD5PasswordHasher de Django, on a juste a appelé la fonction

verify(passwordAVérifier, « md5 $ salt $ hash »)

On a alors ce formulaire (même hash et salt que dans le cas précédent) :



Dans la base de données on trouve la ligne :



Avec dans le champ password :

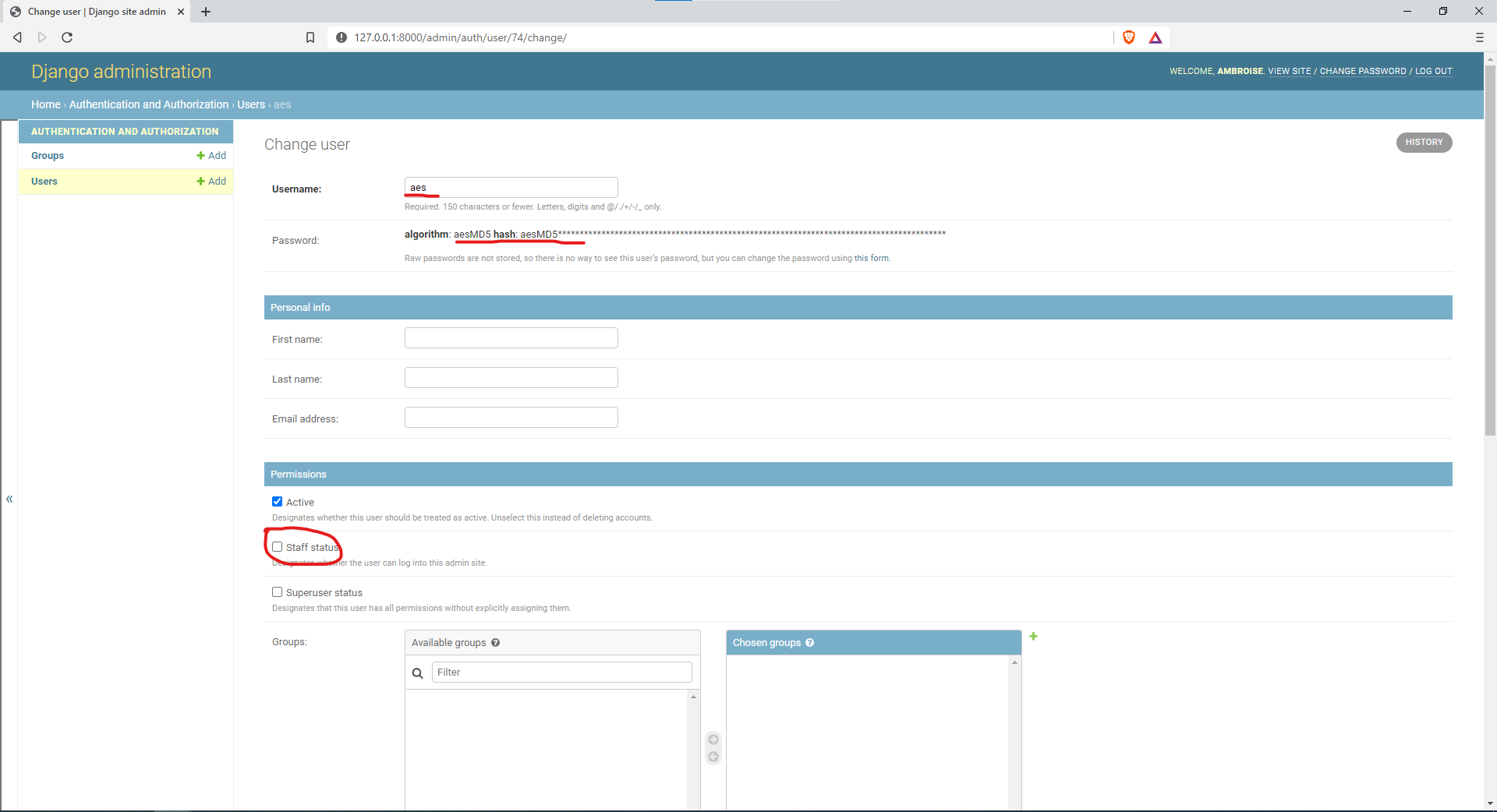
aesMD5**$**\_[#TÖcX(”Q8=\TO**$**¸ˆÉ.ÖÓ°U?v<Zÿ@9âVáUGh¡·Cú^ÌÀ¥¦Ü¼Pé\_„¡‘úHe4d#WHJ}|$É’ƒÅ)Èç`[Û~n­­

## Connexion

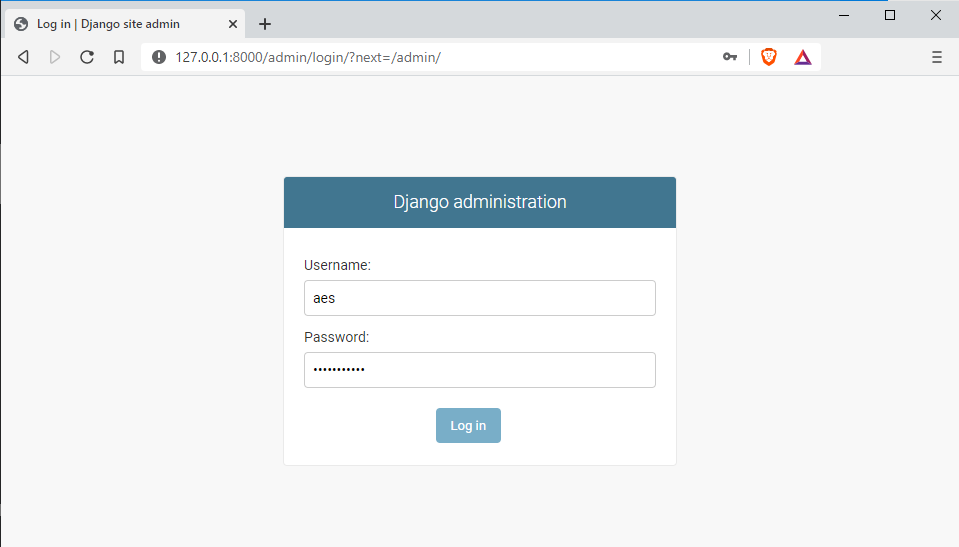
Django intègre une page d’administration /admin, pour y accéder il faut dans un premier temps créer un compte superUser :

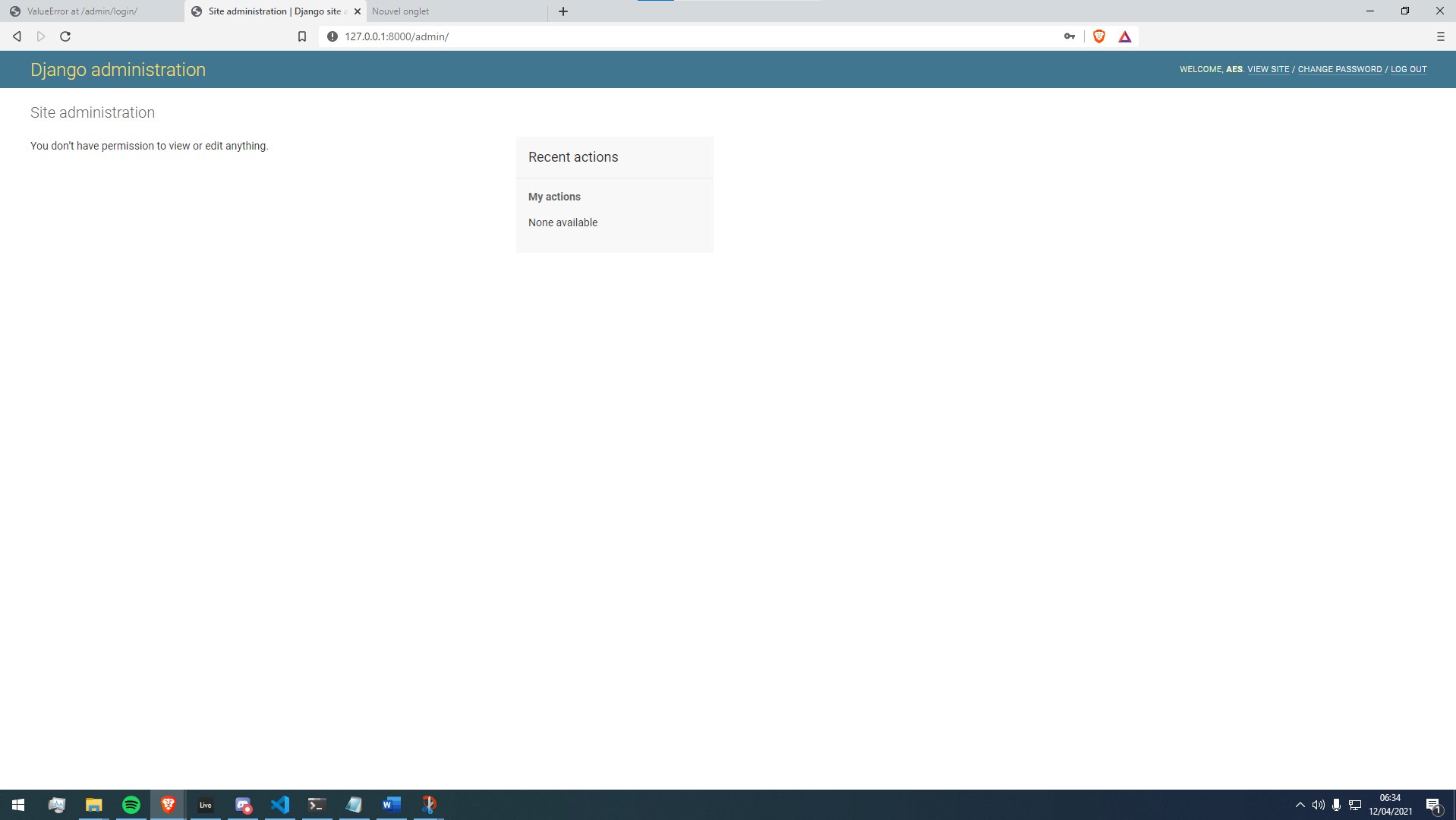


Pour tous les comptes créés on donne le rôle staff qui permet aux à ces utilisateurs de pouvoir se connecter à l’interface admin avec leurs logins.



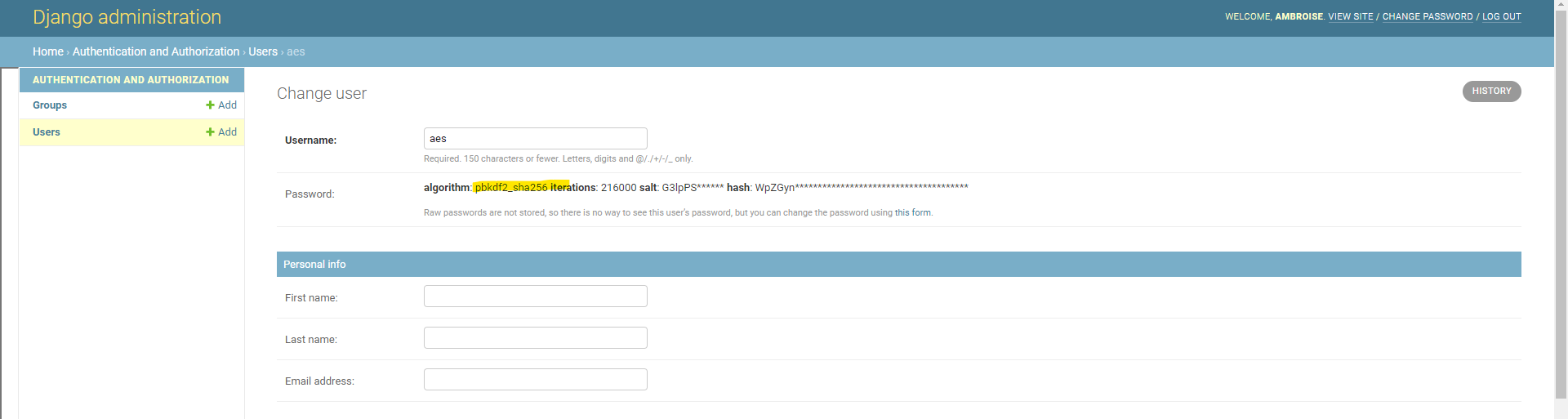
Une fois ceci fait, il sera possible de se connecter aux comptes créés sur /login, ici avec le compte « aes » (le mot de passe haché était « fstNancy123 »)





L’authentification fonctionne de cette manière pour les 5 modes de gestion des mots de passes.

Cependant le hachage choisit sera modifié après connexion :



Django préfère 1 seul mode de hachage qui est par défaut l’algorithme PBKDF2 et change la fonction de hachage qu’on a forcé préalablement pour celui-ci (La connexion fonctionne toujours après ce changement).

## Technologies et Ressources

* [Python](https://www.python.org/) 3.9
* Framework Web Python : [Django](https://www.djangoproject.com/) 3.1.7
* Librairies Python de Cryptographie : [pycryptodome](https://pycryptodome.readthedocs.io/en/latest/index.html)
* [Dcode.fr](https://www.dcode.fr/hash-md5)
* [Sqlite Viewer](https://inloop.github.io/sqlite-viewer/)
* [Github](https://github.com/ElioElioElioElio/demoSecu)