

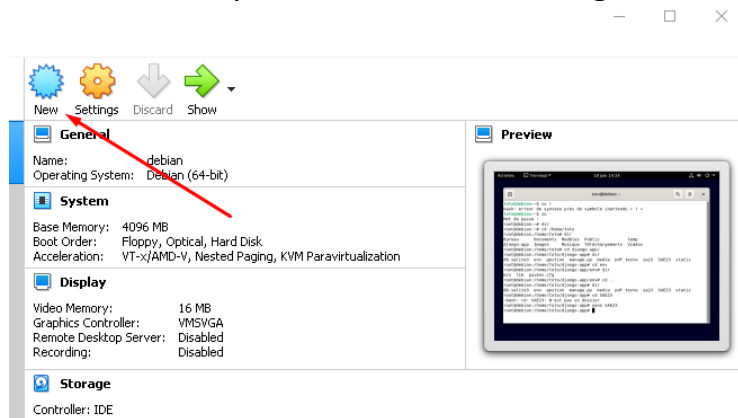
Configuration du serveur Apache pour le projet

I. CONFIGURATION DE LA MACHINE VIRTUELLE

Nous avons commencé la configuration du serveur en téléchargeant Debian 11.3 sur le site officiel du Debian (<https://www.debian.org/download>).

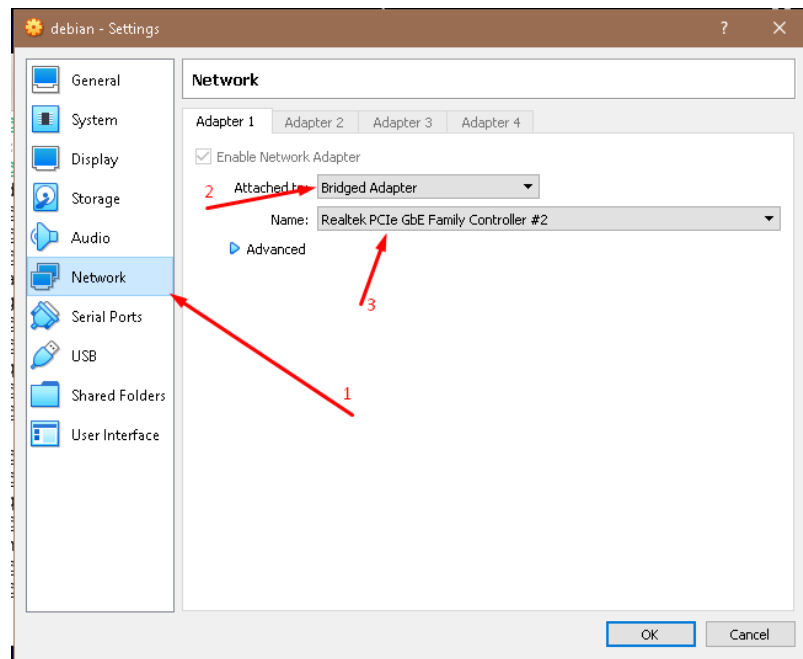
Après nous avons aussi téléchargé Virtualbox qui est un logiciel Open Source proposé par Oracle permettant la virtualisation de système d'exploitation (OS).

Sur Virtualbox nous avons créé une nouvelle machine virtuelle en utilisant le Debian 11.3 que nous avons téléchargé avant.



Nous avons laissé d'abord la configuration réseau en NAT pour avoir accès au Internet pendant l'installation dans lequel nous avons mis en place les configurations nécessaires pour avoir une machine qui tourne en Debian 11.3.

Quand l'installation était finie, nous avons changé la configuration de réseau en accès par pont (Bridged) et nous avons aussi choisi la carte réseau dont notre machine hôte utilise actuellement.



II. Importation du projet dans la machine virtuelle

Maintenant que la machine virtuelle était configurée, nous avons continué avec le processus d'importation du projet qui se trouve sur github vers notre machine.

Pour commencer il fallait savoir que pour configurer un serveur Apache il faut avoir un cheminement très correct des fichiers du projet donc il était nécessaire de faire quelques changements après l'importation sur le cheminement en utilisant la commande mv.

Pour commencer l'importation nous avons d'abord installé le paquet git en utilisant la commande :

apt-get install git

Après nous sommes déplacés vers /home/toto en utilisant la commande :

cd /home/toto

Ici nous avons créé un dossier temporaire qu'on a appelé temp en utilisant la commande :

mkdir temp

Sur le dossier temp nous avons fait l'importation des fichiers du projet en utilisant la commande :

git clone *lien vers le projet*

Donc ici nous avons vu que l'importation était faite en utilisant la commande : « dir » qui nous montre les dossiers et fichiers existant sur le dossier actuel.

```
remote: https
root@debian:/home/toto/temp# git clone https://github.com/Elmirbtj/SAE23
Clonage dans 'SAE23'...
remote: Enumerating objects: 7366, done.
remote: Counting objects: 100% (7366/7366), done.
remote: Compressing objects: 100% (4302/4302), done.
remote: Total 7366 (delta 2038), reused 7282 (delta 1975), pack-reused 0
Réception d'objets: 100% (7366/7366), 9.66 Mio | 8.71 Mio/s, fait.
Résolution des deltas: 100% (2038/2038), fait.
root@debian:/home/toto/temp# dir
SAE23
-
```

Nous sommes ensuite allés dans ce dossier nommé SAE23 jusqu'à ce que nous trouvions le dossier qui contient le fichier manage.py

```
root@debian:/home/toto/temp/SAE23/sa23# dir
db.sqlite3  gestion  manage.py  pdf_teste  sa23
root@debian:/home/toto/temp/SAE23/sa23# █
```

Donc c'était le dossier sa23, après avoir trouvés ça nous sommes revenu vers /home/toto ou nous avons créé un autre dossier qui s'appelle django-app et dans lequel nous avons mis tous les fichiers et dossiers qui se trouvent sur sa23 en utilisant la commande :

mv « chemin de dossier/fichier » /home/toto/django-app

Donc le contenu du dossier django-app va maintenant ressembler à ça :

```
root@debian:/home/toto/django-app# dir
db.sqlite3      gestion  manage.py      pdf_teste  sa23
root@debian:/home/toto/django-app# █
```

III. Installation et configuration d'environnement virtuel et Django

Pour commencer le processus, nous avons téléchargé et installé tous les éléments dont nous avons besoin à partir des référentiels Debian. Cela inclura le serveur Web Apache, le module mod_wsgi utilisé pour s'interfacer avec notre application Django et pip, le gestionnaire de packages Python qui peut être utilisé pour télécharger nos outils liés à Python.

Nous avons commencé par la commande qui va installer apache2, mod_wsgi et pip :

sudo apt-get install python3-pip apache2 libapache2-mod-wsgi-py3

La première étape consiste à créer un environnement virtuel Python afin que notre projet Django soit séparé des outils du système et de tout autre projet Python sur lequel nous pourrions travailler. Nous avons installé la commande virtualenv pour créer ces environnements.

Nous avons obtenu ce paquet en utilisant la commande :

pip3 install virtualenv

Puis, dans le répertoire du projet, nous avons créé un environnement virtuel Python en tapant :

virtualenv env

Cela a créé un répertoire appelé env dans notre répertoire django-app. À l'intérieur, il a installé une version locale de Python et une version locale de pip. Nous pouvons l'utiliser pour installer et configurer un environnement Python isolé pour notre projet.

Avant d'installer les exigences Python de notre projet, nous avons activé l'environnement virtuel en tapant :

source env/bin/activate

Dans l'environnement virtuel nous avons après installer django en tapant la commande :

pip install django

Nous sommes après aller sur le fichier settings.py pour la configurer en utilisant la commande :


nano settings.py

Ici, nous nous concentrerons sur la configuration des hôtes autorisés pour restreindre les domaines auxquels nous répondons et sur la configuration du répertoire des fichiers statiques, où Django placera les fichiers statiques afin que le serveur Web puisse les servir facilement.

On va faire cela en modifiant une ligne et en ajoutant une autre :

```
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True
```

```
ALLOWED_HOSTS = ["*"]
```



Application definition

Donc ici nous avons modifié la ligne ALLOWED_HOSTS qui signifie les hôtes autorisés pour restreindre les domaines auxquels nous répondons et sur la configuration du répertoire des fichiers statiques en ajoutant «*»

```
STATIC_URL = '/static/'  
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

Au bas du fichier, nous avons défini le STATIC_ROOT de Django. Django peut collecter et sortir tous les actifs statiques dans un répertoire connu afin que le serveur Web puisse les servir directement. Nous allons utiliser un peu de Python pour lui dire d'utiliser un répertoire appelé "static" dans le répertoire principal de notre projet.

Maintenant nous pouvons collecter tout le contenu statique dans l'emplacement du répertoire que nous avons défini avec STATIC_ROOT en tapant :

./manage.py collectstatic

Après nous avons ajusté les paramètres de notre pare-feu pour autoriser le trafic vers notre serveur de développement Django, que nous exécuterons sur le port 8000.

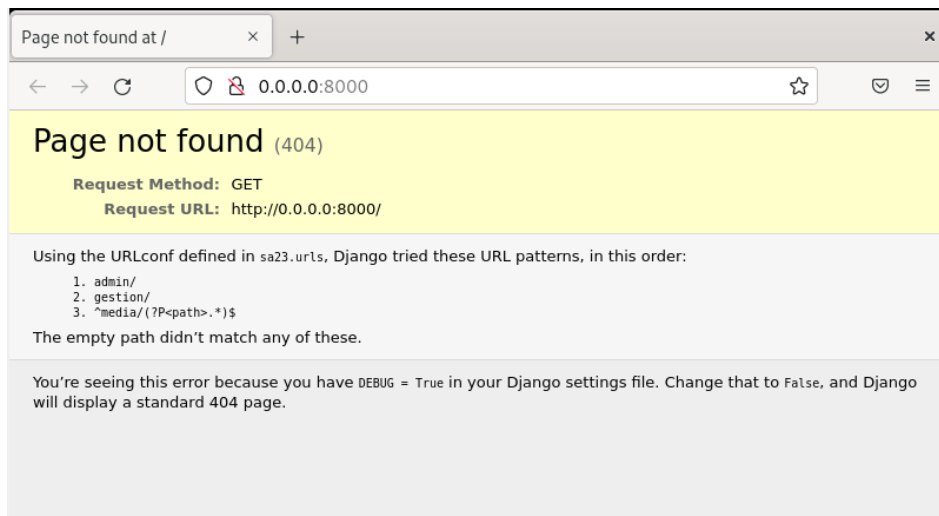
Pour faire cela il faut installer le pare-feu ufw puis autorisés le trafic vers le port 8000 en tapant :

ufw allow 8000

Enfin, nous avons pu tester notre projet en lançant le serveur de développement Django avec cette commande :

./manage.py runserver 0.0.0.0:8000

Et après sur Firefox en mettant comme lien 0.0.0.0 :8000.



Donc maintenant on peut accéder à notre site avec le serveur de développement Django.

IV. Configuration Apache

Nous en avons maintenant fini avec Django pour le moment, nous sommes donc sortis de notre environnement virtuel en tapant :

deactivate

Maintenant que notre projet Django fonctionnait, nous pouvions configurer Apache. Les connexions client qu'il reçoit seront traduites au format WSGI que l'application Django attend en utilisant le module `mod_wsgi`. Cela aurait dû être automatiquement activé lors de l'installation précédente.

Pour configurer le pass WSGI, nous avons dû modifier le fichier d'hôte virtuel par défaut :

nano /etc/apache2/sites-available/000-default.conf

Nous avons conservé les directives qui étaient déjà présentes dans le dossier. Nous venons d'ajouter quelques éléments supplémentaires.

Pour commencer, nous avons configuré les fichiers statiques. Nous avons utilisé un alias pour indiquer à Apache de mapper toutes les requêtes commençant par /static vers le répertoire "static" dans notre dossier de projet. Nous y avons collecté les statiques avant. Nous avons configuré l'alias, puis accordé l'accès au répertoire en question avec un bloc de répertoire.

Ensuite, nous avons accordé l'accès au fichier wsgi.py dans le répertoire du projet de deuxième niveau où le code Django est stocké. Pour ce faire, nous avons dû utiliser une section de répertoire avec une section de fichier à l'intérieur. Nous avons accordé l'accès au fichier à l'intérieur de cette construction imbriquée.

Une fois cela configurait, nous étions prêts à construire la partie du fichier qui gère réellement la passe WSGI. Nous avons utilisé le mode démon pour exécuter le processus WSGI, qui est la configuration recommandée. Nous avons utilisé la directive WSGIDAemonProcess pour configurer cela.

Cette directive prend un nom arbitraire pour le processus. Nous avons utilisé django-app pour rester cohérent. Ensuite, nous configurons la maison Python où Apache peut trouver tous les composants qui peuvent être nécessaires. Comme nous avons utilisé un environnement virtuel, nous l'avons pointé directement vers notre répertoire d'environnement virtuel de base. Ensuite, nous définissons le chemin Python pour qu'il pointe vers la base de notre projet Django.

Ensuite, nous devons spécifier le groupe de processus. Cela devrait pointer vers le même nom que nous avons sélectionné pour la directive WSGIDAemonProcess (django-app dans notre cas). Enfin, nous avons dû définir l'alias du script pour qu'Apache transmette les requêtes du domaine racine au fichier wsgi.py :

```
Alias /static /home/toto/django-app/static
<Directory /home/toto/django-app/static>
    Require all granted
</Directory>

<Directory /home/toto/django-app/sa23>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>

WSGIDAemonProcess django-app python-home=/home/toto/django-app/env python
WSGIProcessGroup django-app
WSGIScriptAlias / /home/toto/django-app/sa23/wsgi.py

:uallHost>
```


Vu qu'on va utiliser la base de données MySQL nous devons autoriser le processus Apache à accéder au fichier db.sqlite3.

Pour ce faire, la première étape consistait à modifier les autorisations afin que le groupe propriétaire de la base de données puisse lire et écrire. Le fichier de base de données s'appelle db.sqlite3 par défaut et se trouvait dans notre répertoire de projet de base :

chmod 664 /django-app/db.sqlite3

chmod 775 /django-app

Ensuite, nous devons donner au groupe sous lequel Apache s'exécute, le groupe www-data, la propriété du groupe du fichier :

chown :www-data /django-app/db.sqlite3

Pour écrire dans le fichier, nous devons également donner au groupe Apache la propriété du répertoire parent de la base de données :

sudo chown :www-data /django-app

Nous devons à nouveau nous adapter à travers notre pare-feu. Nous n'avons plus besoin d'ouvrir le port 8000 puisque nous faisons appel à Apache, nous pouvons donc supprimer cette règle. Nous avons ensuite ajouté une exception pour autoriser le trafic vers le processus Apache :

iptables -D INPUT -p tcp --dport 8000 -j ACCEPT

iptables -I INPUT -p tcp --dport 80 -j ACCEPT

Nous avons ensuite vérifié nos fichiers Apache pour nous assurer que nous n'avons pas faits d'erreurs de syntaxe :

apache2ctl configtest

Tant que la dernière ligne de sortie ressemble à ceci, les fichiers sont en bon état :

```
| Syntax OK
```

Une fois ces étapes effectuées, nous étions prêts à redémarrer le service Apache pour implémenter les modifications que nous avons apportées. Nous avons redémarré Apache en tapant :

systemctl restart apache2

Nous avons donc configuré avec succès le serveur Apache et nous avons pu accéder à notre site Web via l'adresse IP de la machine virtuelle.

Pour trouver l'IP de la machine virtuelle nous avons tapé la commande :

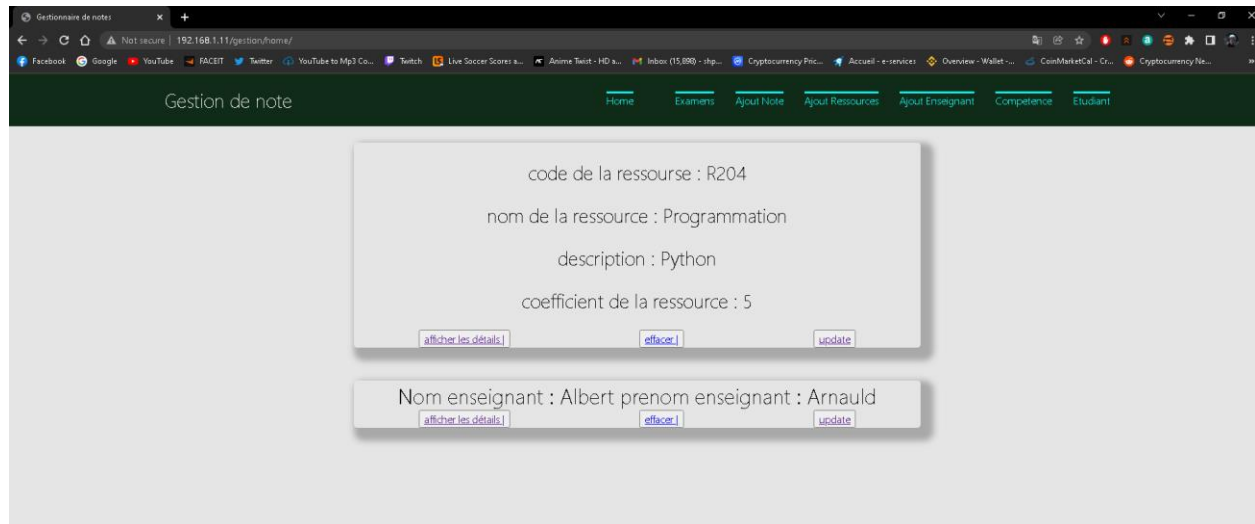
ip a

```
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP  
group default qlen 1000  
    link/ether 08:00:27:99:55:20 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.1.11/24 brd 192.168.1.255 scope global dynamic noprefixroute en  
p0s3  
        valid_lft 35598sec preferred_lft 35598sec  
    inet6 2a01:e0a:28d:4150:45ae:b4f8:c3e5:7c18/64 scope global temporary dynami  
c  
        valid_lft 86020sec preferred_lft 78543sec
```

Donc l'ip est 192.168.1.11 et nous avons après taper cette IP sur la barre de recherche de Firefox pour accéder à notre site web et pour tester si notre serveur web est fonctionnel.



Donc, comme nous pouvons le voir, nous pouvons accéder au site via la machine virtuelle en utilisant son adresse IP mais on va tester le fonctionnement aussi sur la machine hôte en faisant la même chose :



Le site est fonctionnel sur la machine hôte aussi et maintenant nous avons mis en place avec succès un serveur Web pour notre projet.