
Fiche de Procédure Site Web Dynamique Administration Serveur

Table des matières

Introduction	2
Description de L'architecture	2
Prérequis.....	2
Installation et configuration de la base de données	2
Installation d'une instance MySQL	2
Mise en place de la Base de données.....	2
Installation et configuration du site web sous Django.....	5
Mise en place du web dynamique	5
Liaison du sql au projet django	5
Fichier html	6
Fonctionnement du Create :	7
Views et url :	8
Exemple de update	9
Update dans le view:.....	10
Traitement update :	10
Fonctionnement du delete :	11
Configuration de la machine serveur	13
Transfert du fichier Django.....	13
Modifications du fichier settings.py de Django.....	13
Configuration de Gunicorn (fichier gunicorn_config.py)	14
Connexion à la base de données MySQL	15

Introduction

Cette Fiche de procédure a pour but de décrire les étapes de mise en place de l'application web de gestion et administration de serveur de A à Z de la Base de donnée à son hébergement sur une machine serveur

Description de L'architecture

Dans cette application nous utilisons Django afin de pouvoir faire du web dynamique en python et de faire la liaison entre web et Base de données, la base de données est ici hébergée sur une machine différente du serveur

Prérequis

MySQL : Base de données

Nginx : Serveur web

Guinicorn : Serveur applicatif pour exécuter Django

Django et ces libraires : mysqlclient , reportlab

Installation et configuration de la base de données


Installation d'une instance MySQL

Via l'installateur MySQL choisir MySQL Serveur

- Définir la connexion à MySQL sur le port 3306 si ce n'est pas celui par défaut
- Définir un mot de passe pour l'utilisateur Root
- Finaliser l'installation

Mise en place de la Base de données

Utilisation de MySQL Workbench

- Ajouter une connexion a une instance
- Username: root Hostname: 127.0.0.1 (adresse de local host)
- Valider puis entrer le mot de passe défini précédemment
- Ouvrir une fenêtre de requête avec le  logo en haut à gauche

Entrer le code ci-dessous qui permet de crée la base de donnée ainsi que ces tables et ce qu'elle contienne

```
DROP DATABASE IF EXISTS Admin_serveur;
```

```
CREATE DATABASE Admin_serveur;
```

```
USE Admin_serveur;
```

```

CREATE TABLE Type_serveur(
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    type_serveur VARCHAR(40),
    descript VARCHAR(300)
);

CREATE TABLE Serveurs(
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(40),
    nombre_processeur INT UNSIGNED,
    memoire INT UNSIGNED,
    stockage INT UNSIGNED,
    id_type INT UNSIGNED,
    CONSTRAINT fk_type_serveur
        FOREIGN KEY (id_type)
        REFERENCES Type_serveur(id)
);

CREATE TABLE Utilisateur(
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(30),
    prenom VARCHAR(30),
    mail VARCHAR(100)
);

CREATE TABLE Services_disponible(
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(40),
    memoire_necessaire INT NOT NULL
);

CREATE TABLE Services_en_cours(
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    id_service INT UNSIGNED,
    id_serveur INT UNSIGNED,
    date_lancement DATE,
    memoire_utiliser INT,
    CONSTRAINT fk_service
        FOREIGN KEY (id_service)
        REFERENCES Services_disponible(id),
    CONSTRAINT fk_serveur
        FOREIGN KEY (id_serveur)
        REFERENCES Serveurs(id)
);

CREATE TABLE Application_disponible(
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(40),
    logo VARCHAR(100)
);

CREATE TABLE Application_en_cours(
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    id_app INT UNSIGNED,

```

```
id_serveur INT UNSIGNED,  
id_utilisateur INT UNSIGNED,  
CONSTRAINT fk_app  
    FOREIGN KEY (id_app)  
    REFERENCES Application_disponible(id),  
CONSTRAINT fk_serveur_utiliser  
    FOREIGN KEY (id_serveur)  
    REFERENCES Serveurs(id),  
CONSTRAINT fk_utilisateur  
    FOREIGN KEY (id_utilisateur)  
    REFERENCES Utilisateur(id)  
);
```

- Exécuter la Requête avec ce bouton 

La base de donnée a bien été créée et est prête à être utilisée

Installation et configuration du site web sous Django

Mise en place du web dynamique

Afin de faire en sorte que le web dynamique affiche correctement le SQL que nous devons mettre en place. J'ai du dans un premier temps lié via les settings le django au sql.

Liaison du sql au projet django

Voici l'explication du fichier settings

fichier settings :

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'admin_serveur',  
        'USER': 'root',  
        'PASSWORD': 'toto',  
        'HOST': 'localhost',  
        'PORT': '3306',  
    }  
}
```

- 'ENGINE': Cette clé spécifie le moteur de base de données à utiliser. Dans ce cas, le moteur utilisé est "django.db.backends.mysql", ce qui indique que Django va utiliser MySQL pour la base de données.
- 'NAME': C'est le nom de la base de données à laquelle Django se connectera. Dans cet exemple, la base de données s'appelle "admin_serveur".
- 'USER': C'est le nom d'utilisateur utilisé pour se connecter à la base de données. Ici, l'utilisateur est "root".
- 'PASSWORD': C'est le mot de passe associé à l'utilisateur pour l'authentification de la base de données. Dans cet exemple, le mot de passe est "toto".
- 'HOST': C'est l'adresse de l'hôte où se trouve la base de données. Ici, la base de données est hébergée localement sur "localhost".
- 'PORT': C'est le port sur lequel la base de données écoute les connexions. Dans ce cas, le port est défini sur "3306", qui est le port par défaut pour MySQL.

- 'default': C'est le nom de la configuration par défaut de la base de données. Django permet de gérer plusieurs configurations de base de données, mais ici nous n'en avons qu'une seule.

Fichier html

Afin de pouvoir mettre en place les dispositifs qui permettent de créer, modifier et supprimer des choses. Voici un exemple de create :

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <meta charset="UTF-8">
    <title>Lancer Application</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
</head>
<body>
    {% if id is not None %}
        <form method="post" action="/sae/AppCours/traitementupdateAC/{{id}}/">
    {% else %}
        <form method="post" action="/sae/AppCours/confirmationAC/">
    {% endif %}
    {% csrf_token %}
    {{ form.as_ul }} <!-- permet d'afficher tous les champs sous forme d'une liste
-->
    <input type="submit" value="envoyer les données">
</form>
<div class="item-buttons">
<a href="/sae">Retour</a>
</div>
</body>
</html>
```

Fonctionnement du Create :

- `<!DOCTYPE html>` : C'est une déclaration qui indique au navigateur que le document est un fichier HTML.
- `<html lang="en">` : Ceci est la balise d'ouverture de l'élément HTML, et l'attribut `lang` spécifie la langue de la page (en anglais dans cet exemple).
- `<head>` : C'est la balise d'ouverture de l'élément `<head>`, qui contient les métadonnées et les informations de configuration de la page.
- `{% load static %}` : Ceci est une directive Django qui charge les fichiers statiques (CSS, JavaScript, etc.) pour être utilisés dans le modèle.
- `<meta charset="UTF-8">` : C'est une balise qui spécifie le jeu de caractères utilisé pour le document (UTF-8 dans cet exemple).
- `<title>Lancer Application</title>` : C'est la balise qui définit le titre de la page affiché dans la barre de titre du navigateur.
- `<link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">` : C'est une balise qui lie une feuille de style CSS à la page. Le chemin du fichier CSS est spécifié en utilisant la directive Django `{% static %}`.
- `<body>` : C'est la balise d'ouverture de l'élément `<body>`, qui contient le contenu visible de la page.
- `{% if id is not None %}` : Ceci est une directive Django qui vérifie si la variable `id` existe. Si c'est le cas, le formulaire sera soumis à une URL spécifique pour la mise à jour, sinon il sera soumis à une autre URL pour la confirmation.
- `<form method="post" action="/sae/AppCours/traitementupdateAC/{{id}}/">` : C'est la balise `<form>` qui définit un formulaire. L'attribut `method` indique la méthode d'envoi du formulaire (ici, "post"), et l'attribut `action` spécifie l'URL où les données du formulaire seront envoyées. Le `{{id}}` est une variable Django qui sera remplacée par une valeur spécifique.
- `{% csrf_token %}` : Ceci est une directive Django qui ajoute un jeton de sécurité CSRF (Cross-Site Request Forgery) au formulaire, afin de protéger contre les attaques CSRF.
- `{{ form.as_ul }}` : C'est une directive Django qui affiche tous les champs du formulaire sous forme d'une liste non ordonnée (``).
- `<input type="submit" value="envoyer les données">` : C'est un bouton de soumission du formulaire.
- `</form>` : C'est la balise de fermeture du formulaire.
- `<div class="item-buttons">` : Ceci est un conteneur `<div>` avec la classe CSS "item-buttons".
- `Retour` : Ceci est un lien `<a>` qui redirige vers l'URL `"/sae"` (probablement une page de retour).

- `</body>` : C'est la balise de fermeture de l'élément `<body>`.
- `</html>` : C'est la balise de fermeture de l'élément `<html>`.

Views et url :

`def createAC(request):`

`if request.method == "POST":`

`form = ApplicationEnCoursForm(request)`

`if form.is_valid():`

`AC = form.save()`

`return render(request, 'AppCours/confirmationAC.html', {"AC" : AC})`

`else:`

`return render(request, 'AppCours/createAC.html', {"form": form})`

`else :`

`form = ApplicationEnCoursForm()`

`return render(request, 'AppCours/createAC.html', {"form" : form})`

- `def createAC(request):` : Ceci est la définition de la fonction de vue Django nommée "createAC", qui prend la requête en paramètre.
- `if request.method == "POST":` : Cela vérifie si la méthode de la requête est "POST", ce qui indique que le formulaire a été soumis.
- `form = ApplicationEnCoursForm(request)` : Cela crée une instance du formulaire "ApplicationEnCoursForm" en utilisant les données de la requête.
- `if form.is_valid():` : Cela vérifie si les données du formulaire sont valides.
- `AC = form.save()` : Cela enregistre les données du formulaire dans un nouvel objet "AC" de la classe "ApplicationEnCours".
- `return render(request, 'AppCours/confirmationAC.html', {"AC" : AC})` : Cela rend la page de confirmation "confirmationAC.html" en passant l'objet "AC" en contexte.
- `else:` : Cela gère le cas où les données du formulaire ne sont pas valides.
- `return render(request, 'AppCours/createAC.html', {"form": form})` : Cela rend à nouveau la page de création "createAC.html", mais cette fois-ci en passant le formulaire "form" en contexte pour afficher les erreurs de validation.
- `else:` : Cela gère le cas où la méthode de la requête n'est pas "POST" (probablement "GET").
- `form = ApplicationEnCoursForm()` : Cela crée une nouvelle instance du formulaire "ApplicationEnCoursForm" sans données.

- `return render(request, 'AppCours/createAC.html', {"form" : form})` : Cela rend la page de création "createAC.html" en passant le formulaire "form" en contexte pour l'affichage initial du formulaire.

Ce code représente un modèle de vue Django pour créer une nouvelle application en cours. Il vérifie si le formulaire a été soumis, valide les données, enregistre l'objet AC correspondant, puis renvoie la page de confirmation. Sinon, il affiche les erreurs de validation sur la page de création. Si la méthode de la requête n'est pas "POST", il renvoie simplement la page de création avec un formulaire vide.

Exemple de update

```
<!DOCTYPE html>

<html lang="en">

<head>

    {% load static %}

    <meta charset="UTF-8">

    <title>Maj App en Cour</title>

    <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">

</head>

<body>

    <form method="post" action="/sae/AppCours/traitementupdateAC/{{id}}">

        {% csrf_token %}

        {{ form.as_ul }}

        <input type="submit" value="envoyer les données">

    </form>

    <div class="item-buttons">

    <a href="/sae">Retour</a>

    </div>

</body>

</html>
```

- `<!DOCTYPE html>` : C'est une déclaration qui indique au navigateur que le document est un fichier HTML.
- `<html lang="en">` : Ceci est la balise d'ouverture de l'élément HTML, et l'attribut lang spécifie la langue de la page (en anglais dans cet exemple).
- `<head>` : C'est la balise d'ouverture de l'élément `<head>`, qui contient les métadonnées et les informations de configuration de la page.

- `{% load static %}` : Ceci est une directive Django qui charge les fichiers statiques (CSS, JavaScript, etc.) pour être utilisés dans le modèle.
- `<meta charset="UTF-8">` : C'est une balise qui spécifie le jeu de caractères utilisé pour le document (UTF-8 dans cet exemple).
- `<title>Maj App en Cour</title>` : C'est la balise qui définit le titre de la page affiché dans la barre de titre du navigateur.
- `<link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">` : C'est une balise qui lie une feuille de style CSS à la page. Le chemin du fichier CSS est spécifié en utilisant la directive Django `{% static %}`.
- `<body>` : C'est la balise d'ouverture de l'élément `<body>`, qui contient le contenu visible de la page.
- `<form method="post" action="/sae/AppCours/traitementupdateAC/{{id}}">` : C'est la balise `<form>` qui définit un formulaire. L'attribut `method` indique la méthode d'envoi du formulaire (ici, "post"), et l'attribut `action` spécifie l'URL où les données du formulaire seront envoyées. Le `{{id}}` est une variable Django qui sera remplacée par une valeur spécifique.
- `{% csrf_token %}` : Ceci est une directive Django qui ajoute un jeton de sécurité CSRF (Cross-Site Request Forgery) au formulaire, afin de protéger contre les attaques CSRF.
- `{{ form.as_ul }}` : C'est une directive Django qui affiche tous les champs du formulaire sous forme d'une liste non ordonnée (``).
- `<input type="submit" value="envoyer les données">` : C'est un bouton de soumission du formulaire.
- `<div class="item-buttons">` : Ceci est un conteneur `<div>` avec la classe CSS "item-buttons".

Update dans le view:

- `def updateAC(request, id):` : Ceci est la définition de la fonction de vue Django nommée "updateAC" qui prend la requête et l'identifiant (id) en paramètres.
- `AC = models.ApplicationEnCours.objects.get(pk=id)` : Cela récupère l'objet "ApplicationEnCours" correspondant à l'identifiant (id) fourni.
- `form = ApplicationEnCoursForm(AC.dico())` : Cela crée une instance du formulaire "ApplicationEnCoursForm" en utilisant les données de l'objet "AC".
- `return render(request, "AppCours/createAC.html", {"form": form, "id": id})` : Cela rend la page de création "createAC.html" en passant le formulaire "form" et l'identifiant (id) en contexte

Traitement update :

- `def traitementupdateAC(request, id):` : Ceci est la définition de la fonction de vue Django nommée "traitementupdateAC" qui prend la requête et l'identifiant (id) en paramètres.
- `lform = ApplicationEnCoursForm(request.POST)` : Cela crée une instance du formulaire "ApplicationEnCoursForm" en utilisant les données de la requête POST.
- `if lform.is_valid():` : Cela vérifie si les données du formulaire sont valides.
- `ApplicationEnCours = lform.save(commit=False)` : Cela crée une instance de la classe "ApplicationEnCours" à partir des données du formulaire, mais ne l'enregistre pas immédiatement en base de données.
- `ApplicationEnCours.id = id` : Cela attribue l'identifiant (id) fourni à l'instance "ApplicationEnCours".
- `ApplicationEnCours.save()` : Cela enregistre l'instance "ApplicationEnCours" mise à jour en base de données.
- `return HttpResponseRedirect('/sae/')` : Cela redirige l'utilisateur vers l'URL "/sae/" après la mise à jour réussie.
- `else:` : Cela gère le cas où les données du formulaire ne sont pas valides.
- `return render(request, 'AppCours/updateAC.html', {"form": lform, "id": id})` : Cela rend la page de mise à jour "updateAC.html" en passant le formulaire "lform" et l'identifiant (id) en contexte.

Fonctionnement du delete :

`def deleteAC(request, id):` : Ceci est la définition de la fonction de vue Django nommée "deleteAC" qui prend la requête et l'identifiant (id) en paramètres.

- `ApplicationEnCours = models.ApplicationEnCours.objects.get(pk=id)` : Cela récupère l'objet "ApplicationEnCours" correspondant à l'identifiant (id) fourni.
- `ApplicationEnCours.delete()` : Cela supprime l'objet "ApplicationEnCours" de la base de données.
- `return HttpResponseRedirect('/sae/')` : Cela redirige l'utilisateur vers l'URL "/sae/" après la suppression réussie de l'application en cours

Description des fichiers importants :

Models (Modèles) : Le fichier "models.py" définit la structure des données de l'application. Les modèles sont utilisés pour décrire les tables de la base de données et leurs relations. Chaque classe modèle représente une table de la base de données et les attributs de la classe représentent les champs de la table. Les modèles permettent de définir les types de données, les contraintes et les relations entre les différentes tables. Ils jouent un rôle essentiel dans l'interaction avec la base de données et permettent de créer, lire, mettre à jour et supprimer des enregistrements.

Forms (Formulaires) : Le fichier "forms.py" est utilisé pour définir des formulaires dans Django. Les formulaires permettent de créer des interfaces pour collecter des données

auprès des utilisateurs. Les classes de formulaire sont utilisées pour spécifier les champs du formulaire, les types de données attendus, les validations et les messages d'erreur associés. Les formulaires facilitent la collecte, la validation et le traitement des données saisies par les utilisateurs. Ils sont souvent utilisés pour créer, mettre à jour ou filtrer des enregistrements dans la base de données.

URLs (Uniform Resource Locators) : Le fichier "urls.py" est utilisé pour définir les schémas d'URL de l'application. Il associe les URL spécifiées par l'utilisateur aux vues appropriées qui doivent être affichées lorsque ces URL sont demandées. Les schémas d'URL permettent de définir des règles de correspondance entre les URL et les vues correspondantes, en utilisant des expressions régulières et des noms de vues. Les URLs servent à définir la structure de l'application et à spécifier comment les différentes vues sont accessibles via les URL.

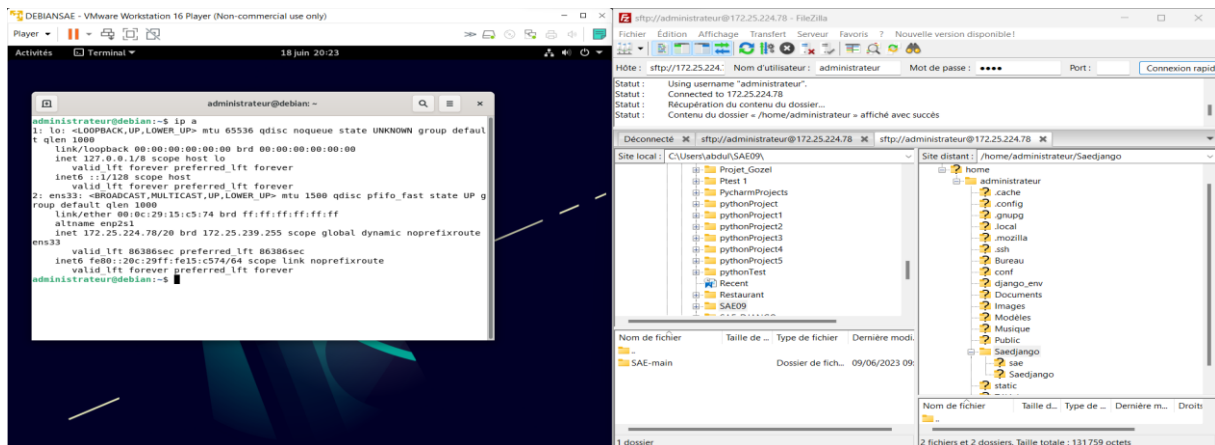
Views (Vues) : Le fichier "views.py" contient les vues de l'application, qui sont les fonctions ou les classes responsables du traitement des requêtes HTTP et de la génération des réponses correspondantes. Les vues définissent la logique métier de l'application et déterminent ce qui doit être affiché à l'utilisateur. Elles reçoivent les requêtes des utilisateurs, interagissent avec les modèles pour récupérer ou modifier les données, puis rendent les templates appropriés pour afficher les résultats. Les vues peuvent également gérer des opérations telles que la validation des formulaires, le traitement des données soumises par les utilisateurs et la gestion des redirections.

Configuration de la machine serveur

Transfert du fichier Django

Le fichier Django, contenant le code source de l'application, a été transféré de la machine physique vers la machine virtuelle à l'aide de FileZilla. On se connecte en mettant les informations de la machine virtuelle avec l'adresse IP, le nom d'utilisateur, le mot de passe et le port 22.

Le fichier a été placé dans le répertoire "/home/administrateur/Saedjango".



Modifications du fichier settings.py de Django

Fichier "Saedjango/Saedjango/settings.py"

```
ALLOWED_HOSTS = ['172.18.36.150']
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'admin_serveur',
        'USER': 'toto',
        'PASSWORD': 'toto',
        'HOST': '172.18.32.1',
        'PORT': '3306',
    }
}
```

```
STATIC_URL = '/static/'
```

```

STATIC_ROOT = '/home/administrateur/Saedjango/sae/static/'
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'sae/media')

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
CSRF_COOKIE_DOMAIN = '172.18.36.150'
CSRF_TRUSTED_ORIGINS = [
    'http://172.18.36.150'
]

```

Les paramètres "ALLOWED_HOSTS" ont été configurés pour accepter les requêtes provenant de l'adresse IP "172.18.36.150". - Les paramètres de connexion à la base de données MySQL ont été configurés, incluant le nom de la base de données, le nom d'utilisateur, le mot de passe, l'hôte et le port. - Les paramètres de fichiers statiques et de médias ont été configurés pour spécifier les répertoires correspondants.

Configuration de Gunicorn (fichier gunicorn_config.py)

Fichier "conf/gunicorn_config.py" :

```

command = '/home/administrateur/django_env/bin/gunicorn'
pythonpath = '/home/administrateur/Saedjango'
bind = '172.18.36.150:8000'
workers = 3

```

Les paramètres de configuration de Gunicorn ont été définis, notamment le nombre de travailleurs et le chemin vers l'application Django.

Configuration de Nginx :

Fichier "/etc/nginx/sites-available/Saedjango" :

```

server {
    listen 80;
    server_name 172.18.36.150;

    location /static/ {
        root /home/administrateur/static;
    }

    location / {
        proxy_pass http://172.18.36.150:8000;
    }
}

```

```
}  
}
```

La configuration de Nginx a été modifiée pour spécifier le serveur web et la gestion des fichiers statiques.

Connexion à la base de données MySQL

- Un utilisateur MySQL avec tous les droits nécessaires a été créé.
- Le fichier SQL contenant les données du projet a été importé dans la base de données "admin_serveur".
- Les informations de connexion à la base de données MySQL ont été configurées dans le fichier "Saedjango/Saedjango/settings.py", incluant le nom de la base de données, le nom d'utilisateur, le mot de passe, l'hôte et le port.