REPUBLIQUE DU CAMEROUN
****
PAIX - TRAVAIL - PATRIE
****
ECOLE NATIONALE SUPERIEURE
POLYTECHNIQUE DE YAOUNDE
****
DEPARTEMENT DU GENIE ELECTRIQUE
ET TELECOMMUNICATIONS

REPUBLIC OF CAMEROON
******
PEACE – WORK - FATHERLAND
****
NATIONAL ADVANCED SCHOOL OF
ENGINEERING YAOUNDE
****
ELECTRICAL AND
TELECOMMUNICATIONS ENGINEERING
DEPARTMENT

Rapport de programmation python

Members du groupe :

Noms	Matricules	Pourcentage en %
TENDJANG NDJAYA ISNEL	22p282	25
EMMANUELLE CINDY	21P367	25
KEDE NGATCHUESSI	22P546	25
MEWALI NNOMO	22p501	25
EFFOUA BEKOLO JUNIOR	22P180	0
YOUWE LAMNA SYLVAIN	21P388	0

***EXAMINATEUR** : Mr. MBIETEU AMOS*

Table des matières

I. INTRODUCTION

Ce rapport présente le fruit du travail collectif réalisé par notre groupe dans le cadre du projet de programmation Python, portant sur le développement d'une application de gestion de tontines à l'aide du framework Django. La tontine, en tant que forme d'épargne communautaire, implique une gestion rigoureuse des contributions, prêts, remboursements et interactions entre membres. Django, grâce à son architecture MVT (Modèle-Vue-Template), s'est avéré être un outil efficace pour structurer l'application selon des principes solides de développement web.

Dans ce document, nous exposons l'architecture générale du projet, la structure des modules (don, prêts, remboursements, tontines, etc.), la configuration des modèles, les vues principales et la logique d'interaction entre les différentes composantes. Une attention particulière a été portée à la clarté du code, la cohérence des données, et à l'expérience utilisateur via les templates HTML

II. CONCEPTION DE LA BASE DE DONNEES MYSQL

Dans cette étape, nous nous sommes concerte et avons décider ce la technologie utilise pour notre base de données. A cette question, nous avons répondu MySQL. Nous avons opté pour une base de données MySQL, car avons juger quelle était plus adapte au projet et au compétences de chacun.

La conception de la base de données a été l'une des étapes les plus délicate du projet, pour ce faire, nous avons utilisé la base de données du professeur comme Template de base.

III. FONCTIONNEMENT DE DJANGO

A. architecture MVC

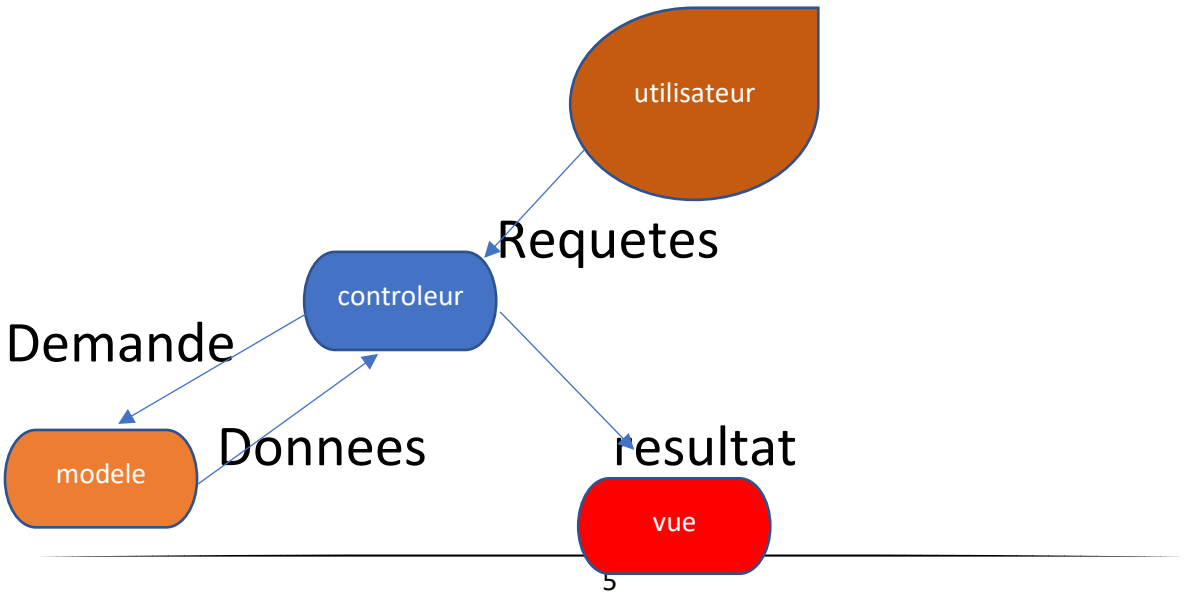
Lorsque nous parlons de frameworks qui fournissent une interface graphique à l'utilisateur (soit une page web, comme ici avec Django), nous parlons souvent de l'architecture MVC. Il s'agit d'un modèle distinguant plusieurs rôles précis d'une application, qui doivent être accomplis. Comme son nom l'indique, l'architecture (ou « patron ») Modèle-Vue-Contrôleur est composé de trois entités distinctes, chacune ayant son propre rôle à remplir.

- **le modèle** représente une information enregistrée quelque part, le plus souvent dans une base de données. Il permet d'accéder à l'information, de la modifier, d'en ajouter une nouvelle, de vérifier que celle-ci correspond bien aux critères (on parle d'intégrité de l'information), de la

mettre à jour, etc. Il s'agit d'une interface supplémentaire entre notre code et la base de données, mais qui simplifie grandement les choses.

- **la vue** qui est, comme son nom l'indique, la visualisation de l'information. C'est la seule chose que l'utilisateur peut voir. Non seulement elle sert à présenter une donnée, mais elle permet aussi de recueillir une éventuelle action de l'utilisateur (un clic sur un lien, ou la soumission d'un formulaire par exemple). Typiquement, un exemple de vue est une page web.
- **le contrôleur** prend en charge tous les événements de l'utilisateur (accès à une page, soumission d'un formulaire, etc.). Ilse charge, en fonction de la requête de l'utilisateur, de récupérer les données voulues dans les modèles. Après un éventuel traitement sur ces données, il transmet ces données à la vue, afin qu'elle s'occupe de les afficher. Lors de l'appel d'une page, c'est le contrôleur qui est chargé en premier, afin de savoir ce qu'il est nécessaire d'afficher.

B. schema de l'architecture MVC



c. specificitee de django : le modele MVT

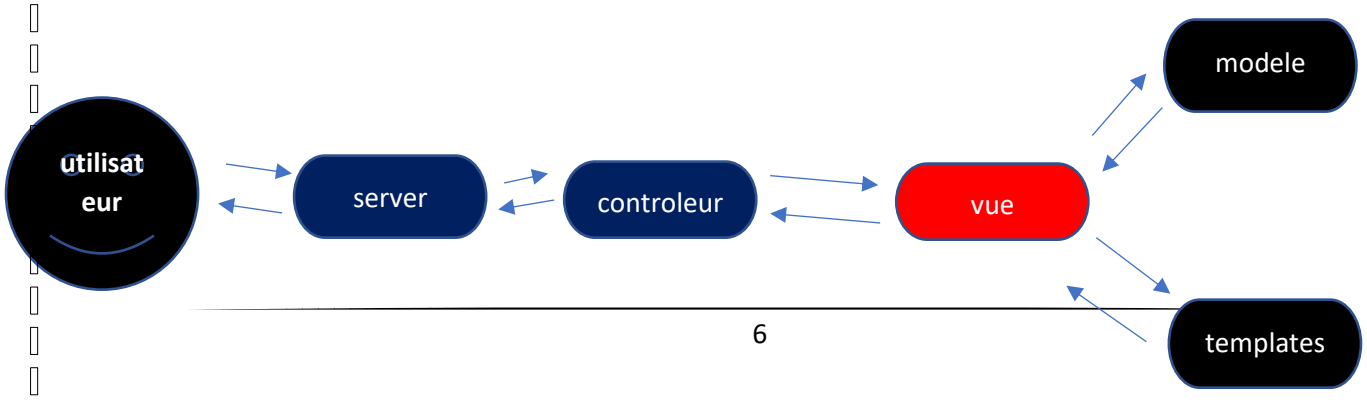
L'architecture utilisée par Django diffère légèrement de l'architecture MVC classique. En effet, la « magie » de Django réside dans le fait qu'il gère lui-même la partie contrôleur (gestion des requêtes du client, des droits sur les actions…). Ainsi, nous parlons plutôt de framework utilisant l'architecture MVT : Modèle-Vue-Template.

Cette architecture reprend les définitions de modèle et de vue que nous avons vues, et en introduit une nouvelle : le template

Un template est un fichier HTML, aussi appelé en français « gabarit ». Il sera récupéré par la vue et envoyé au visiteur ; cependant, avant d'être envoyé, il sera analysé et exécuté par le framework, comme s'il s'agissait d'un fichier avec du code. Django fournit un moteur de templates très utile qui permet, dans le code HTML, d'afficher des variables, d'utiliser des structures conditionnelles (if/else) ou encore des boucles (for), etc.

On en revient donc au modèle MVT. Le développeur se doit de fournir le modèle, la vue et le template. Une fois cela fait, il suffit juste d'assigner la vue à une URL précise, et la page est accessible.

d. Schéma d'exécution d'une requête



e. Projet et applications

En plus de l'architecture MVT, Django introduit le développement d'un site sous forme de projet. Chaque site web conçu avec Django est considéré comme un projet, composé de plusieurs applications. Une application consiste en un dossier contenant plusieurs fichiers de code, chacun étant relatif à une tâche du modèle MVT que nous avons vu. En effet, chaque bloc du site web est isolé dans un dossier avec ses vues, ses modèles et ses schémas d'URL.

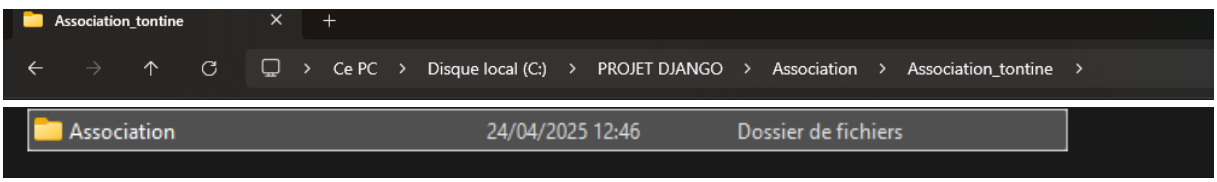
Ce principe de séparation du projet en plusieurs applications possède deux avantages principaux :

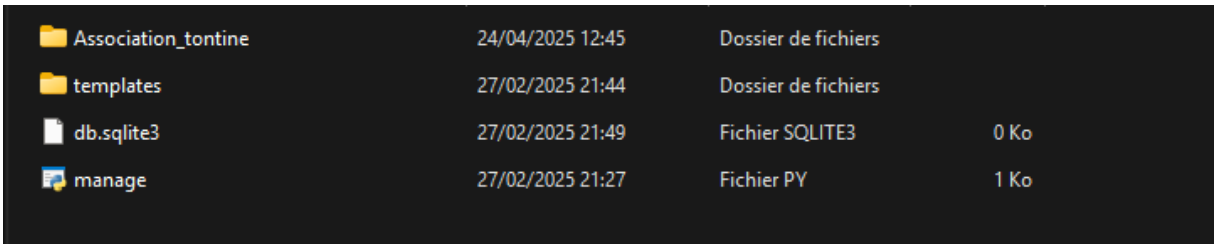
- Le code est beaucoup plus structuré. Les modèles et templates d'une application ne seront que rarement ou jamais utilisés dans une autre, nous gardons donc une séparation nette entre les différentes applications.
- Une application correctement conçue pourra être réutilisée dans d'autres projets très simplement, par un simple copier/coller

IV. GESTION DU PROJET

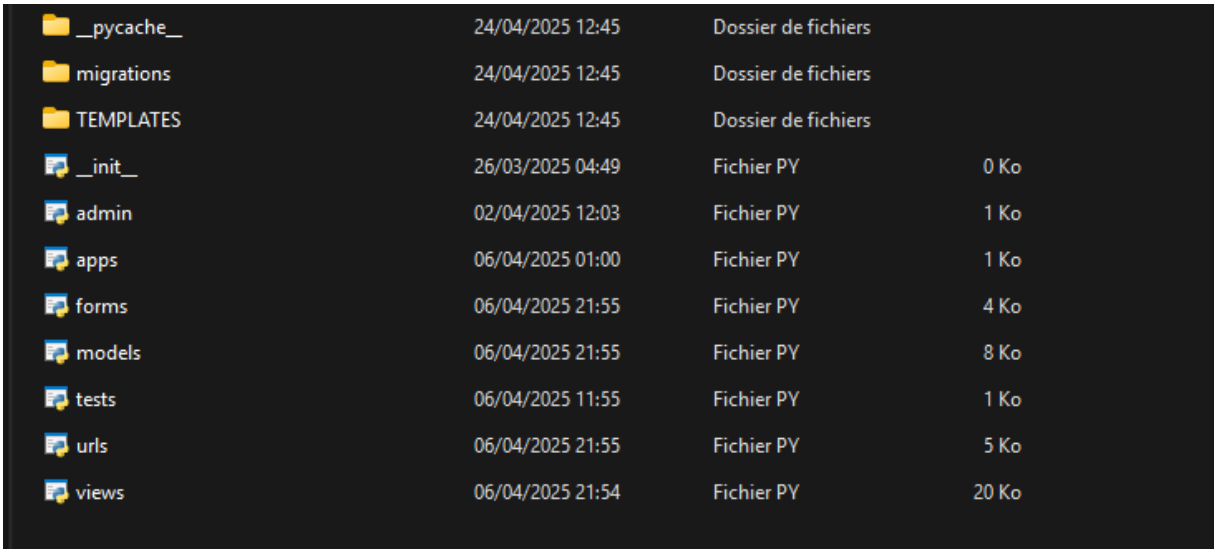
A. Generalisation

Notre projet se nomme **“Association”** et comporte une seule application qui se nomme **“Association_tontine”** structure comme suite :





Association_tontine	24/04/2025 12:45	Dossier de fichiers	
templates	27/02/2025 21:44	Dossier de fichiers	
db.sqlite3	27/02/2025 21:49	Fichier SQLITE3	0 Ko
manage	27/02/2025 21:27	Fichier PY	1 Ko



__pycache__	24/04/2025 12:45	Dossier de fichiers	
migrations	24/04/2025 12:45	Dossier de fichiers	
TEMPLATES	24/04/2025 12:45	Dossier de fichiers	
__init__	26/03/2025 04:49	Fichier PY	0 Ko
admin	02/04/2025 12:03	Fichier PY	1 Ko
apps	06/04/2025 01:00	Fichier PY	1 Ko
forms	06/04/2025 21:55	Fichier PY	4 Ko
models	06/04/2025 21:55	Fichier PY	8 Ko
tests	06/04/2025 11:55	Fichier PY	1 Ko
urls	06/04/2025 21:55	Fichier PY	5 Ko
views	06/04/2025 21:54	Fichier PY	20 Ko

Il faut ajouter cette application au projet. Pour que Django considère le sous-dossier blog comme une application, il faut donc l'ajouter dans la configuration.

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Association_tontine',
]
```

Les noms des fichiers sont relativement évidents:

- **models.py** contiendra vos modèles
- **tests.py** permet la création de tests unitaires
- **views.py** contiendra toutes les vues de votre application.

B. Modeles des modules

Notre applications est subdivisee en 6 modules a savoir :

- Don
- Remboursements
- Prets
- Tontines
- Members
- Tableau de bords

Et chacun de ses modules est constitue de plusieurs modeles leur permettant de stocker leur informations ou donnees.

• Don

Ce module prends en compte les different dons recus et donnes par l'association. Les dons recu par l'association peuvent etre internes (effectues par des members de l'association) et externes (effectues par des externes a l'association).

Et ce module comprends les modeles suivants :

- **Don**
- **Tontines**
- **Membres**

Ce module interagit donc avec les modules : members , tontines et tableaux de bords

• Remboursements

Ce module conserne les informations des membres qui ont eu a faire des prets. Ces informations sont regroupees en champs dans les different modeles de ce module.

Et ce module comprends les modeles suivants:

- **Remboursements**
- **Prets**

Ce module interagit donc avec les modules: prets et tableaux de bords

• Prets

Ce module permet de gerer les prets effectuer par les members de l'association. Il prend en compte le montant du pret, les observations ou motifs du pret, le statut(rembourse, enc ours, non rembourses etc..) , le member qui effectue le pret et a quelle seance cela a ete fait.

Et ce module comprends les modeles suivants:

- **Prets**
- **Membres**
- **Rembourssements**

Ce module interagit donc avec les modules: rembourssements et tableaux de bords

• Tontines

Ce module permet la gestion des differentes tontines de l'association ainsi que des differentes actions qui peuvent etre misent en oeuvres.

Et ce module comprends les modeles suivants:

- **Tontines**
- **Membres**
- **Aide**
- **Cotisation**
- **Epargne**

Ce module interagit donc avec les modules: membres, et tableaux de bords

- **Members**

Ce module permet de gerer les membres de l'association en receuillant leur informations personnelles et les tontines dans lesquelles elles participent.

Et ce module comprends les modeles suivants:

- **Membres**
- **Tontines**

Ce module interagit donc avec les modules: tontines, et tableaux de bords

- **Tableaux de bords**

Cette partie recapitules tous les donnees de l'application et interventions des membres dans l'association et des acteurs externes a l'association.

Ce module fait intervenir tous les modeles de l'application et donc du projet.

Par consequent il interagit avec tous les autres modules.

- **Modele User de Django**

Le modèle **`User`** de Django est utilisé pour représenter les utilisateurs dans une application web. Par défaut, Django fournit un modèle utilisateur prêt à l'emploi, mais il peut aussi être personnalisé.

Il permet entre autre de:

- Ajouter des champs personnalisés
- Changer le champ d'identification
- Adapter les permissions et la logique d'accès
- Utiliser une gestion plus souple des utilisateurs
- Construire un backend API (REST) plus moderne

C.Views de l’application

Chaque vue se doit d'être associée au minimum à une URL. Avec Django, une vue est représentée par une fonction définie dans le fichier views.py. Cette fonction va généralement récupérer des données dans les modèles et appeler le bon template pour générer le rendu HTML adéquat.

Chaque application possède son propre fichier views.py, regroupant l'ensemble de ses fonctions. Comme tout bon blog, le nôtre possèdera plusieurs vues qui rempliront diverses tâches.

Nous avons les vues suivantes:

- **Deconnexion**

La vue de **déconnexion** (logout) sert à mettre fin à la session de l’utilisateur connecté. En d'autres termes, elle déconnecte l'utilisateur de manière sécurisée et propre.

- Supprime la session en cours
- Déconnecte l’utilisateur
- Redirige l’utilisateur

- **envoyer_recu**

la vue **`envoyer_recu`** pourrait avoir un rôle crucial : générer et/ou envoyer un reçu après qu’un membre a effectué un paiement ou une contribution à la tontine.

- Générer un reçu de contribution
- L’enregistrer dans la base de données
- (Optionnel) Envoyer le reçu par email

- **Register**

la vue **`register`** sert à permettre à un nouvel utilisateur (membre) de s'inscrire. cela signifie enregistrer un nouveau participant au système.

- Crée un nouvel utilisateur/membre en base
- Authentifie et connecte (ou redirige) l'utilisateur
- Affiche un formulaire d'inscription

- **Login**

la vue **`login`** permet à un membre déjà inscrit de se connecter à son compte.

- Authentifie l'utilisateur
- Redirige vers la page d'accueil
- Affiche un formulaire de connexion

Et les differentes vues non classiques tells que:

- Membres
- creer_pret
- tontine
- prets
- epargnes
- aides
- versementsols
- remboursements
- ajouter_membre , etc...

D. Le dossier Templates

Le dossier `template` dans un projet Django est essentiel pour gérer la partie visuelle de l'application. C'est là qu'on mets tous les fichiers HTML que Django va rendre dynamiquement pour l'utilisateur.

Ce dossier contient tous les templates HTML et CSS utilisés pour afficher les pages du l'application web : formulaires, tableau de bord, reçus, login/register, etc.

REFERENCES BIBLIOGRAPHIQUES.

- Mathieu, X. (2013). Développez votre site web avec le framework Django. Licence Creative Commons.
- Prolixe (2013). Apprenez à programmer en Python. Licence Creative Commons.

V. CONCLUSION

Au terme de ce projet, nous avons pu consolider nos compétences en programmation Python orientée web, en particulier autour du framework Django. Le développement de l'application de gestion de tontines nous a permis d'aborder des notions clés telles que la gestion des utilisateurs, l'interconnexion entre modèles, la sécurisation des accès, la structuration modulaire d'un projet, et l'affichage dynamique avec les templates.

Ce projet illustre l'importance d'une architecture claire (MVT), de vues bien définies (comme l'enregistrement, la connexion ou l'envoi de reçus), et d'une base de données bien modélisée pour assurer le bon fonctionnement d'une application complexe. Nous sommes fiers du travail accompli, et ce projet constitue une base réutilisable pour toute initiative similaire visant à digitaliser des systèmes de gestion communautaire.