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

Membres du groupe :

NOMS	PRENOMS	POURCENTAGE
AKAM	Patrick Claude	20%
MAATSING SOH	Armanda Helsa	20%
MOHAMOUDOU IBRAHIM	Meissa	20%
NDEFFO MAHORO	David	20%
SOBOTH	Shekinah	20%

Responsable : Mr Amos

Table des matières

I. INTRODUCTION

Ce document synthétise les résultats d'une collaboration collective autour d'un projet Python visant à concevoir une plateforme de gestion de tontines via le Framework Django. Système ancestral d'épargne solidaire, la tontine nécessite une orchestration précise des cotisations, crédits, dettes et échanges entre participants. L'architecture MVT (Modèle-Vue-Template) de Django s'est révélé particulièrement adapté pour organiser l'application selon des bonnes pratiques de développement web.

Nous y décrivons l'écosystème global du projet, l'organisation des différents modules (dons, emprunts, règlements, cycles d'épargne), la modélisation des données, les mécanismes centraux et l'articulation entre les composants clés. Un soin particulier a été apporté à la lisibilité du code source, à la cohérence des informations traitées, ainsi qu'à l'ergonomie via l'implémentation d'interfaces HTML intuitives.

II. CONCEPTION DE LA BASE DE DONNEES MYSQL

Cette phase a mobilisé une réflexion collective pour sélectionner la technologie de base de données la plus pertinente. Après concertation, notre choix s'est porté sur MySQL. Cette décision s'appuie sur son adéquation optimale aux besoins du projet et à l'expertise technique de l'équipe.

La modélisation des données a constitué un défi majeur du développement. Pour structurer notre schéma relationnel, nous nous sommes inspirés de la base existante fournie par l'enseignant, que nous avons adaptée et enrichie pour répondre à nos spécificités.

III. FONCTIONNEMENT DE DJANGO

A. Architecture MVC

Dans le contexte des Framework dotés d'interfaces graphiques (tels que les applications web développées avec Django), l'architecture **MVC** (Modèle-Vue-Contrôleur) constitue un pilier fondamental. Ce paradigme organisationnel segmente une application en trois couches fonctionnelles distinctes, chacune assumant des responsabilités spécifiques :

1. **Le Modèle**
   Il incarne la **représentation structurée des données**, généralement stockées dans une base de données. Véritable gardien de l'intégrité informationnelle, il encapsule les opérations CRUD (Création, Lecture, Mise à jour, Suppression) et valide les contraintes métiers. Agissant comme une couche d'abstraction entre le code applicatif et le stockage physique, il fluidifie les opérations de manipulation des données.

2. **La Vue**
   Cette couche matérialise **l'interface de restitution** visible par l'utilisateur final. Au-delà de sa fonction d'affichage (pages web, formulaires), elle sert de canal bidirectionnel : elle restitue les informations tout en capturant les interactions utilisateurs (clics, saisies). Son rôle se limite strictement à la présentation, sans logique de traitement.
3. **Le Contrôleur**
   Véritable chef d'orchestre, il **orchestre les interactions** entre les autres composants. Lors d'une action utilisateur (navigation, envoi de données), il interprète la requête, sollicite le modèle pour récupérer ou modifier des données, applique si nécessaire des traitements métiers, puis délègue à la vue le soin de générer la réponse visuelle. Il incarne la logique décisionnelle centrale du flux applicatif.

B. Schéma de l'architecture MVC

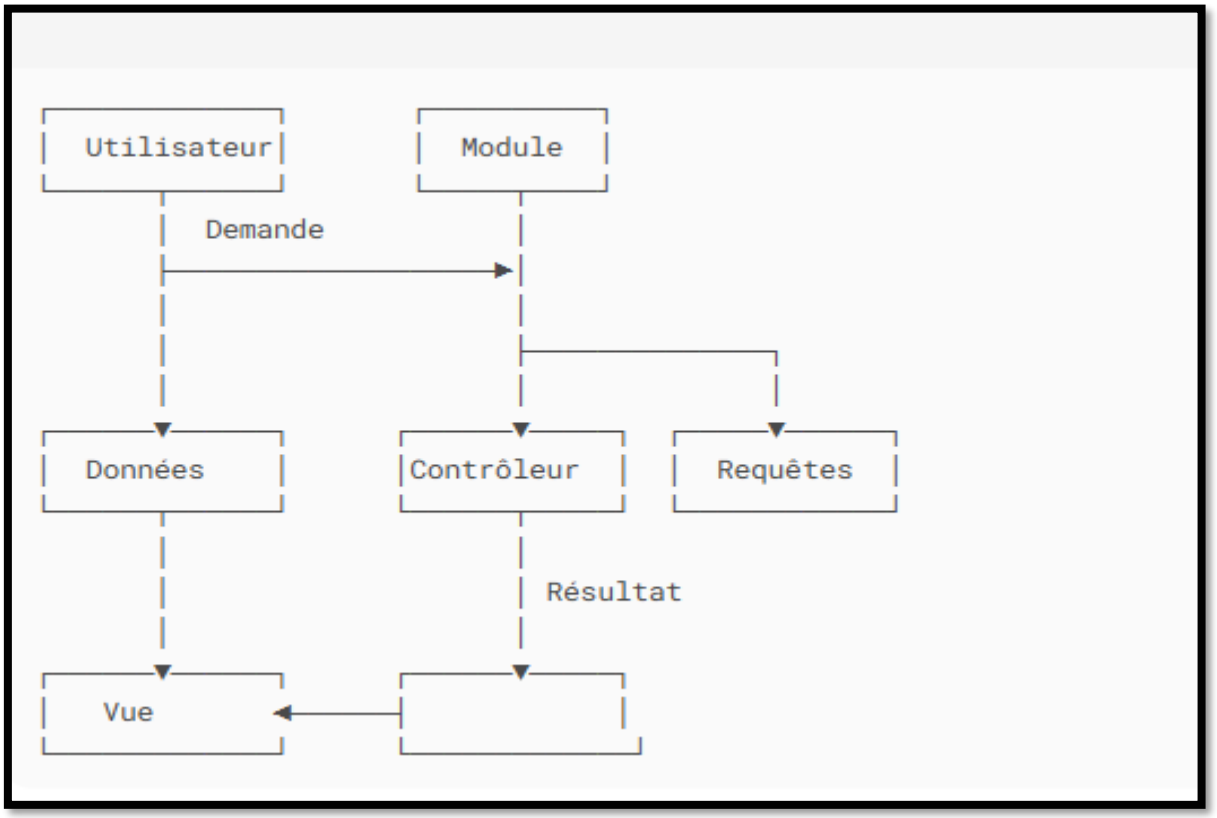


Figure 1 Architecture MVC

c. Spécificités de Django : le modèle MVT

Django s'écarte partiellement du paradigme MVC traditionnel en proposant une **architecture MVT** (Modèle-Vue-Template). Cette particularité réside dans sa capacité à internaliser automatiquement la logique du contrôleur – gestion des requêtes HTTP, validation des permissions, et routage. Le framework opère ainsi une **distinction fonctionnelle** entre trois couches :

1. **Modèle**
   Représente la structure des données et les règles métiers, directement interfacé avec la base de données (opérations CRUD).
2. **Vue**
   Agit comme un **pont logique** : elle récupère les données du modèle, applique des traitements spécifiques, et prépare le contexte pour le template. Contrairement au MVC classique, elle intègre une partie de la logique habituellement dévolue au contrôleur.
3. **Template**
   Squelette HTML dynamique enrichi d'une syntaxe dédiée. Avant d'être servi au client, il est interprété par le moteur de templates de Django, permettant d'y insérer :

   - Variables contextuelles (ex : {{ utilisateur.nom }}),
   - Structures conditionnelles ({% if ... %}),
   - Boucles itératives ({% for ... %}),

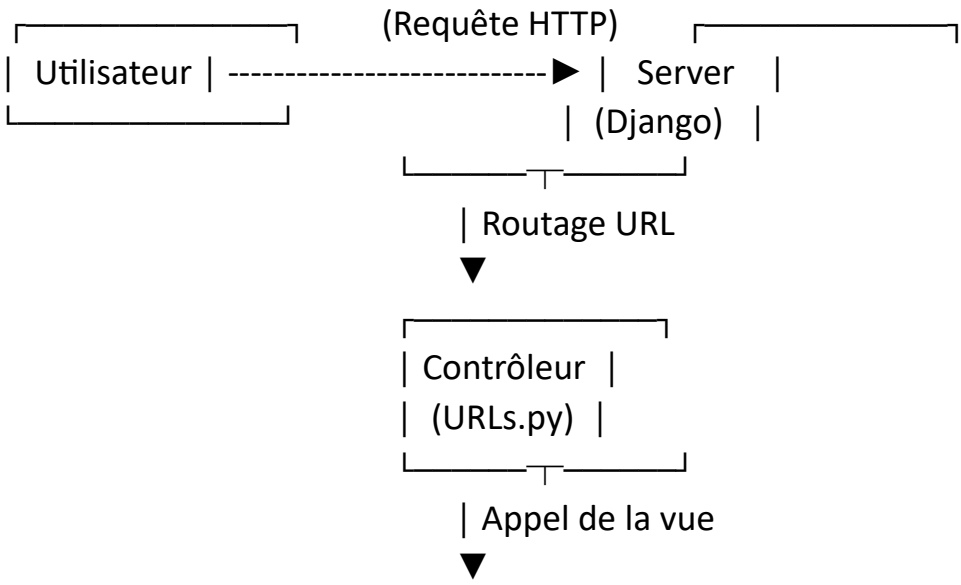
- Héritage de templates ({% extends %}).

Le flux MVT simplifie le développement :

- Le développeur **définit les modèles**, **conçoit les vues** (logique applicative), et **structure les templates** (présentation).
- Une **configuration minimale** (association vue-URL dans urls.py) rend la fonctionnalité accessible.

Cette approche réduit la complexité tout en conservant une séparation claire des responsabilités, caractéristique des architectures modernes.

d. Schéma d'exécution d'une requête



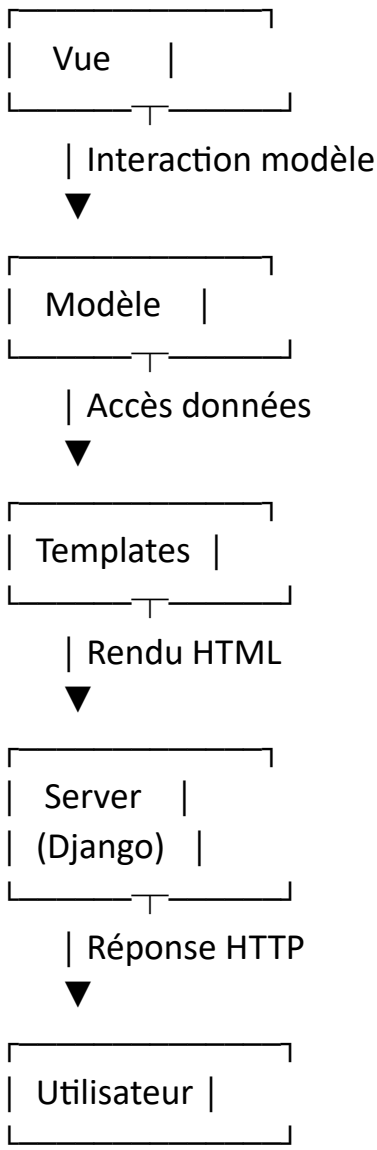


Figure 2 Exécution d'une requête

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.

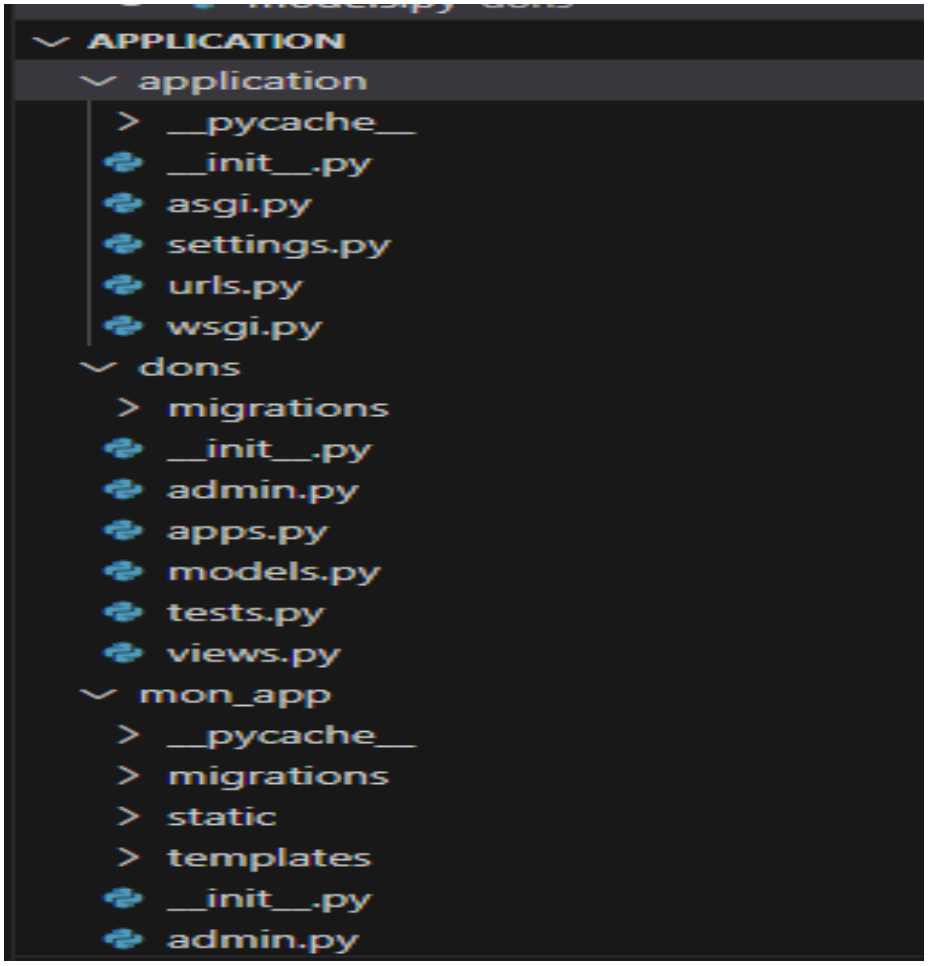
Ce principe de séparation du projet en plusieurs applications possède deux avantages principaux :

- 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. Généralisation

Notre projet se nomme **“Application ”** et comporte une seule application qui se nomme **“application ”** structure comme suit :



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.

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

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. Modèles des modules

Notre application est subdivisée en 6 modules a savoir :

- Don
- Remboursements
- Prêts
- Tontines
- Membres
- Tableau de bords

Et chacun de ses modules est constitué de plusieurs modèles leur permettant de stocker leurs informations ou données.

- **Don**

Ce module prend en compte les différents dons reçus et donnés par l'association. Les dons reçus par l'association peuvent être internes (effectués par des membres de l'association) et externes (effectués par des externes a l'association).

Et ce module comprend les modèles suivants :

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

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

- **Remboursements**

Ce module concerne les informations des membres qui ont eu a faire des prêts. Ces informations sont regroupées en champs dans les différents modèles de ce module.

Et ce module comprend les modèles suivants :

- **Remboursements**
- **Prêts**

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

• Prêts

Ce module permet de gérer les prêts effectuer par les membres de l'association. Il prend en compte le montant du prêt, les observations ou motifs du prêt, le statut (rembourse, en cours, non rembourses etc..), le membre qui effectue le prêt et a quelle séance cela a été fait.

Et ce module comprend les modèles suivants :

- **Prêts**
- **Membres**
- **Remboursements**

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

• Tontines

Ce module permet la gestion des différentes tontines de l'association ainsi que des différentes actions qui peuvent être misent en œuvres.

Et ce module comprend les modèles suivants :

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

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

• Membres

Ce module permet de gérer les membres de l'association en recueillant leurs informations personnelles et les tontines dans lesquelles elles participent.

Et ce module comprend les modèles suivants :

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

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

• Tableaux de bords

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

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

Par conséquent il interagit avec tous les autres modules.

- **Modèle 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 autres 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 :

• **Déconnexion**

La vue de **déconnexion** (log out) 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

• **L**a vue **envoyer reçu**

La vue **envoyer reçu** 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 différentes vues non classiques telles que :

- Membres
- Créer prêt
- Tontine
- Prêts
- Epargnes
- Aides

- Versement sols
- 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 met 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.