RAPPORT DÉTAILLÉ DU PROJET TASKFLOW

Système de Gestion de Tâches avec Django

Abdelkhalek Sirine

11 décembre 2025

Table des matières

Chapitre 1

Introduction

1.1 Présentation

TaskFlow est une application web complète de gestion de tâches (To-Do List) développée avec le framework Django. Cette application permet à plusieurs utilisateurs de créer des comptes personnels et de gérer leurs tâches de manière indépendante et sécurisée.

1.2 Contexte

Dans un contexte où la productivité et l'organisation sont essentielles, disposer d'un outil de gestion de tâches efficace est devenu indispensable. TaskFlow répond à ce besoin en offrant une interface moderne, intuitive et des fonctionnalités avancées.

1.3 Portée du projet

Ce projet couvre :

— L'authentification des utilisateurs (inscription, connexion, déconnexion).

— La gestion complète des tâches (CRUD).

— Un système d'historique avec restauration.

— Un tableau de bord avec statistiques.

— Une interface utilisateur moderne et responsive.

Chapitre 2

Objectifs du projet

2.1 Objectifs fonctionnels

Table 2.1 – Objectifs fonctionnels du projet

Objectif	Description	Statut
Multi-utilisateurs	Chaque utilisateur a son propre espace	Validé
Authentification	Inscription et connexion sécurisées	Validé
Gestion des tâches	Créer, modifier, supprimer des tâches	Validé
Priorités	Trois niveaux de priorité	Validé
Dates d'échéance	Suivi des deadlines	Validé
Historique	Conservation des tâches supprimées	Validé
Restauration	Récupération des tâches supprimées	Validé
Actions en masse	Opérations sur plusieurs tâches	Validé

2.2 Objectifs techniques

— Utiliser l'architecture MVT de Django.

— Implémenter une base de données relationnelle.

— Créer une interface responsive.

— Assurer la sécurité des données.

— Suivre les bonnes pratiques de développement.

Chapitre 3

Technologies utilisées

3.1 Backend

3.1.1 Django 5.x

Django est un framework web Python de haut niveau qui encourage le développement rapide et une conception propre et pragmatique.

Les raisons de ce choix incluent :

— Framework complet avec de nombreux composants intégrés.

— ORM puissant pour la base de données.

— Système d'authentification intégré.

— Protection CSRF automatique.

— Interface d'administration générée automatiquement.

3.1.2 Python 3.x

Le langage de programmation utilisé par Django est Python 3.x, choisi pour sa lisibilité, son écosystème riche et sa large adoption.

3.1.3 SQLite

La base de données utilisée est SQLite, adaptée aux environnements de développement et aux petites applications grâce à sa simplicité de déploiement.

3.2 Frontend

3.2.1 HTML5

HTML5 est utilisé pour la structure sémantique des pages web.

3.2.2 CSS3

CSS3 prend en charge :

— L'utilisation de variables CSS pour la cohérence.

— Les mises en page avec Flexbox et Grid.

— Les animations et transitions.

— Le design responsive.

3.2.3 JavaScript (Vanilla)

Le JavaScript « vanilla » est employé pour :

— Les interactions utilisateur.

— Les actions en masse.

— Le menu mobile.

— La gestion de messages à disparition automatique.

3.3 Ressources externes

3.3.1 Google Fonts

Deux polices sont utilisées :

— DM Sans pour le corps du texte.

— Space Grotesk pour les titres et le logo.

Chapitre 4

Installation et configuration

4.1 Commandes d'installation

Étape 1 : Installer Django

```
pip install django
```

Listing 4.1 – Installation de Django

Étape 2 : Créer le projet

```
django-admin startproject todo_app
```

Listing 4.2 – Création du projet Django

Étape 3 : Naviguer dans le projet

```
cd todo_app
```

Listing 4.3 – Navigation dans le dossier du projet

Étape 4 : Créer l'application `tasks`

```
python manage.py startapp tasks
```

Listing 4.4 – Création de l'application tasks

Étape 5 : Créer l'application `accounts`

```
python manage.py startapp accounts
```

Listing 4.5 – Création de l'application accounts

Étape 6 : Créer les dossiers de templates

```
mkdir templates
mkdir templates/tasks
mkdir templates/accounts
```

Listing 4.6 – Création des dossiers de templates

Étape 7 : Créer les dossiers statiques

```
mkdir static
mkdir static/css
mkdir static/js
```

Listing 4.7 – Création des dossiers statiques

Étape 8 : Créer les migrations

```
python manage.py makemigrations
```

Listing 4.8 – Création des migrations

Étape 9 : Appliquer les migrations

```
python manage.py migrate
```

Listing 4.9 – Application des migrations

Étape 10 : Lancer le serveur

```
python manage.py runserver
```

Listing 4.10 – Lancement du serveur de développement

4.2 Configuration de `settings.py`

4.2.1 Applications installées

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

Listing 4.11 – Configuration des applications installées

4.2.2 Configuration des templates

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        # ...
    },
]
```

Listing 4.12 – Configuration du dossier des templates

4.2.3 Configuration des fichiers statiques

```
STATIC_URL = 'static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
```

Listing 4.13 – Configuration des fichiers statiques

4.2.4 Configuration de l'authentification

```
LOGIN_URL = 'accounts:login'
LOGIN_REDIRECT_URL = 'tasks:dashboard'
LOGOUT_REDIRECT_URL = 'accounts:login'
```

Listing 4.14 – Redirections d'authentification

Chapitre 5

Architecture du projet

5.1 Architecture MVT

L'application suit l'architecture MVT de Django, comprenant les modèles, les vues et les templates, connectés à une base de données relationnelle.

5.2 Structure des fichiers

La structure du projet comprend un module principal de configuration, deux applications (`tasks` et `accounts`), un répertoire de templates et un répertoire pour les fichiers statiques.

Chapitre 6

Modèles de données

6.1 Diagramme entité-relation

Le modèle de données relie les utilisateurs aux tâches via une relation un-à-plusieurs, et ajoute un profil utilisateur via une relation un-à-un.

6.2 Modèle `Task`

Le modèle `Task` contient les champs nécessaires à la gestion des tâches, notamment le propriétaire, le titre, la description, le statut, la priorité, les dates d'échéance et les métadonnées temporelles.

6.3 Modèle `UserProfile`

Le modèle `UserProfile` étend le modèle utilisateur Django avec des informations supplémentaires telles que la biographie, le téléphone et une couleur d'avatar.

Chapitre 7

Application `tasks`

7.1 Configuration des URLs

Les URLs de l'application `tasks` couvrent le tableau de bord, la liste des tâches actives, les tâches complétées, l'historique, la création, l'édition, la complétion, la suppression, la restauration et les actions en masse.

7.2 Vues principales

La vue de tableau de bord calcule des statistiques par utilisateur, liste les tâches récentes et les tâches proches de l'échéance. La vue de création de tâche gère le formulaire de création et l'association de la tâche à l'utilisateur connecté.

7.3 Formulaires

Le formulaire `TaskForm` est basé sur un `ModelForm` et personnalise les champs `title`, `description`, `priority` et `due_date` avec des widgets adaptés à l'interface graphique.

Chapitre 8

Application `accounts`

8.1 Configuration des URLs

L'application `accounts` définit les routes pour l'inscription, la connexion, la déconnexion, la gestion du profil et le changement de mot de passe.

8.2 Vues d'authentification

La vue d'inscription crée un nouvel utilisateur, génère son profil associé et connecte automatiquement l'utilisateur après validation du formulaire. La vue de connexion authentifie les identifiants et redirige vers le tableau de bord ou vers la page initialement demandée.

8.3 Formulaires

Le formulaire d'inscription étend le `UserCreationForm` de Django et ajoute la validation d'unicité de l'adresse e-mail, tandis que d'autres formulaires permettent la mise à jour des informations utilisateur et du profil.

Chapitre 9

Templates et interface

9.1 Template de base

Le template de base définit la structure HTML principale, gère l'état connecté ou non de l'utilisateur, et expose des blocs pour le titre et le contenu afin de permettre l'héritage dans les autres pages.

9.2 Tags de template Django

Les templates utilisent des variables, des structures de contrôle, des tags d'URL et de formulaire (dont le `{% csrf_token %}`) pour générer dynamiquement le contenu en fonction du contexte.

9.3 Design CSS

Le design repose sur des variables CSS, des classes utilitaires pour les boutons, les cartes et les formulaires, ainsi que sur Flexbox et Grid pour une mise en page moderne et responsive.

Chapitre 10

Flux de navigation

10.1 Utilisateur non connecté

Un utilisateur non connecté est redirigé vers la page de connexion ou d'inscription, puis vers le tableau de bord après authentification.

10.2 Gestion des tâches

Le flux de navigation couvre la consultation des tâches actives, complétées et archivées, ainsi que les transitions entre ces états via les actions de complétion, de suppression et de restauration.

10.3 Cycle de vie d'une tâche

Le cycle de vie d'une tâche passe par les états `created`, `completed` et `deleted`, avec la possibilité de restauration et de suppression définitive.

Chapitre 11

Sécurité

11.1 Protection CSRF

La protection CSRF de Django repose sur un jeton unique par session, inséré dans les formulaires et vérifié à chaque soumission.

11.2 Décorateur `@login_required`

Le décorateur `@login_required` restreint l'accès à certaines vues aux seuls utilisateurs authentifiés et redirige les autres vers la page de connexion.

11.3 Isolation des données

Les vues filtrent systématiquement les objets par utilisateur et utilisent des fonctions comme `get_object_or_404` pour s'assurer qu'un utilisateur ne peut pas accéder aux données d'un autre.

11.4 Hashage des mots de passe

Les mots de passe sont stockés sous forme de hachage via l'API d'authentification de Django, qui utilise par défaut un algorithme sûr comme PBKDF2.

11.5 Décorateur `@require_POST`

Le décorateur `@require_POST` est utilisé pour protéger les actions sensibles, empêchant leur déclenchement via des requêtes GET.

Chapitre 12

Conclusion

12.1 Résumé du projet

TaskFlow est une application web complète qui met en avant les capacités du framework Django en termes d'architecture MVT, de gestion de données relationnelles, d'authentification et de sécurité.

Table 12.1 – Synthèse des aspects du projet

Aspect	**Réalisation**
Architecture	MVT avec séparation claire des responsabilités
Base de données	Modèles relationnels via l'ORM Django
Authentification	Système complet avec profils utilisateur
Interface	Design moderne, responsive, thème sombre
Sécurité	CSRF, authentification obligatoire, isolation des données
Expérience utilisateur	Messages flash, actions en masse, recherche

12.2 Compétences acquises

Le projet a permis de renforcer les compétences suivantes :

— Utilisation avancée du framework Django (configuration, routing, ORM, migrations).

— Développement web structuré avec l'architecture MVT, formulaires, sessions et authentification.

— Intégration frontend avec CSS moderne, JavaScript et design responsive orienté UX/UI.

Fin du rapport

Projet réalisé avec Django – Système de Gestion de Tâches TaskFlow