Rapport de projet

Année d’étude 2023/2024

Filière ingénierie informatique et réseaux

**Sous le thème :**

**Application To-Do List avec Authentification**

**Réalisés par : Saad Berrhout -- Jamal Zarqane Encadré par : M. Nafil Khalid**

Année universitaire 2023/2024

# Table des matières

1. **Introduction**
2. **Contexte et objectifs**
3. **Organisation du code et arborescence**
   1. **Dossier racine et virtualenv**
   2. **Applications Django (apps)**
   3. **Fichiers de configuration principaux**
4. **Modélisation des données**
   1. **Le modèle Category**
   2. **Le modèle TodoList**
   3. **Discussion sur les choix de champs et de valeurs par défaut**
5. **Gestion des utilisateurs et authentification**
   1. **Extensions de Django Auth**
   2. **Formulaires d’inscription et de connexion**
   3. **Flux métiers et décorateurs**
6. **Conception « Diagrammes UML »**
   1. **Diagramme de cas d’utilisation**
   2. **Diagramme de classes**
   3. **Diagramme de sequence**
7. **Logique métier et vues**
   1. **Vue principale (my\_todos)**
   2. **CRUD des tâches**
   3. **Gestion des catégories dans les vues**
8. **Templates et interface utilisateur**
   1. **Architecture MVT et héritage des templates**
   2. **Responsive design avec Bootstrap / Tailwind**
   3. **Accessibilité et bonnes pratiques HTML**
9. **Sécurité et bonnes pratiques**
   1. **Protection CSRF et XSS**
   2. **Gestion des permissions et accès utilisateur**
   3. **Nettoyage des données et validation server-side**
10. **Tests, qualité et maintenance**
    1. **Structure des fichiers de tests**
    2. **Recommandations pour tests unitaires et d’intégration**
11. **Déploiement et configuration de production**
    1. **Procfile, runtime.txt et Heroku**
    2. **Gestion des fichiers statiques (WhiteNoise)**
    3. **Exclusion du virtualenv du dépôt**
12. **Axes d’amélioration et perspectives**
13. **Conclusion**

# Introduction

Ce document propose une analyse académique et technique détaillée du projet To-Do List

avec Authentification, développé en Django. L’objectif est de mettre en perspective les choix d’architecture, les bonnes pratiques mises en œuvre, les points d’attention ainsi que les

perspectives d’évolution.

# Contexte et objectifs

Le projet répond au besoin fondamental de gestion personnelle de tâches, tout en garantissant la confidentialité via un système d’authentification. Les objectifs fonctionnels sont :

* Inscription et connexion sécurisées des utilisateurs.
* Création, modification, suppression et marquage de tâches comme « terminées ».
* Filtrage des tâches par état et par catégorie.
* Interface moderne et responsive (Bootstrap ou Tailwind).

# Organisation du code et arborescence

* 1. **Dossier racine et virtualenv**

À la racine (DjangoUnchained/), on trouve :

manage.py requirements.txt runtime.txt Procfile db.sqlite3

.gitignore

Venv/ ← Virtual environment inclus (à exclure via .gitignore en production)

Recommandation : retirer le dossier Venv/ du dépôt et l’ajouter à .gitignore pour alléger et sécuriser le dépôt.

# Applications Django (apps)

Trois apps principales :

* + - login\_registration/ : gestion des utilisateurs
    - category/ : CRUD sur les catégories de tâches
    - todolist/ : logique métier des tâches

Le package projet todoapp/ contient la configuration globale (settings.py, urls.py, wsgi.py, asgi.py).

# Fichiers de configuration principaux

* + - todoapp/settings.py : configuration des apps, base de données, INSTALLED\_APPS, middleware, etc.
    - requirements.txt : liste des dépendances (Django, gunicorn, django-crispy-forms…)
    - runtime.txt : version Python (utile pour Heroku)
    - Procfile : indique à Heroku la commande web: gunicorn todoapp.wsgi

# Modélisation des données

* 1. **Le modèle Category**

```python

# category/models.py

class Category(models.Model):

name = models.CharField(max\_length=100, null=True) class Meta:

verbose\_name = "Category" verbose\_name\_plural = "Categories"

def str (self): return self.name

```

Champ name simple ; on pourrait ajouter un champ slug pour les URLs propres.

# Le modèle TodoList

```python

# todolist/models.py

class TodoList(models.Model):

title = models.CharField(max\_length=250)

content = models.TextField(blank=True, null=True)

created = models.DateField(default=timezone.now.strftime("%Y-%m-%d")) due\_date = models.DateField(default=timezone.now.strftime("%Y-%m-%d")) time = models.TimeField(null=True)

category = models.ForeignKey(Category, default="general", on\_delete=models.CASCADE) user = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True, blank=True)

class Meta:

ordering = ["-created"]

def str (self): return self.title

```

Observation : default=timezone.now.strftime(...) renvoie une chaîne, pas un date. Il serait préférable d’utiliser des callables datetime.

Erreur potentielle : default="general" pour la FK category attend une PK ou un objet Category, pas une chaîne.

# Discussion sur les choix de champs et de valeurs par défaut

* + - Préférer les objets datetime.date aux chaînes
    - Gérer les valeurs par défaut de FK avec un objet ou sa PK existante
    - Ajouter des contraintes (ex. null=False) selon la criticité des champs

# Gestion des utilisateurs et authentification

* 1. **Extensions de Django Auth**

L’app login\_registration utilise :

* + - CustomUserCreationForm (héritant de UserCreationForm)
    - LoginForm (héritant de AuthenticationForm)

# Formulaires d’inscription et de connexion

```python

# login\_registration/views.py def registerUserView(request):

if request.method == 'POST':

form = CustomUserCreationForm(request.POST) if form.is\_valid():

form.save()

user = authenticate( username=form.cleaned\_data['username'], password=form.cleaned\_data['password1']

```

)

login(request, user)

return redirect('todolist:my\_todos')

Flux clair : création, authentification, redirection.

Amélioration : prévoir un message de feedback (messages.success ou messages.error).

# Flux métiers et décorateurs

* + - Toutes les vues de todolist sont protégées par @login\_required
    - La déconnexion appelle logout() puis redirige vers la page principale

# Conception « Diagrammes UML »

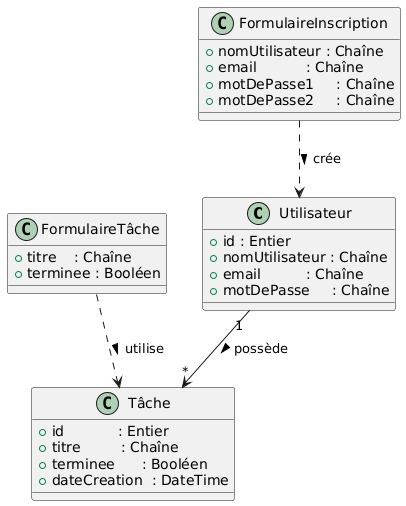
## Diagramme de cas d'utilisation

## 

Ce diagramme montre les différentes interactions possibles entre l'utilisateur et le système.  
L'utilisateur peut :

* **S’inscrire** en remplissant le formulaire d'inscription
* **Se connecter** et **se déconnecter**
* **Voir, créer, modifier, supprimer, valider** des tâches
* **Filtrer les tâches** selon certains critères
* **Valider son inscription**  
  Les cas d’utilisation « Remplir formulaire d’inscription » et « Remplir formulaire de tâche » sont étendus par les cas « S’inscrire » et « Créer une tâche » respectivement.  
  Ce diagramme permet de visualiser les fonctionnalités principales offertes par l'application du point de vue de l'utilisateur.

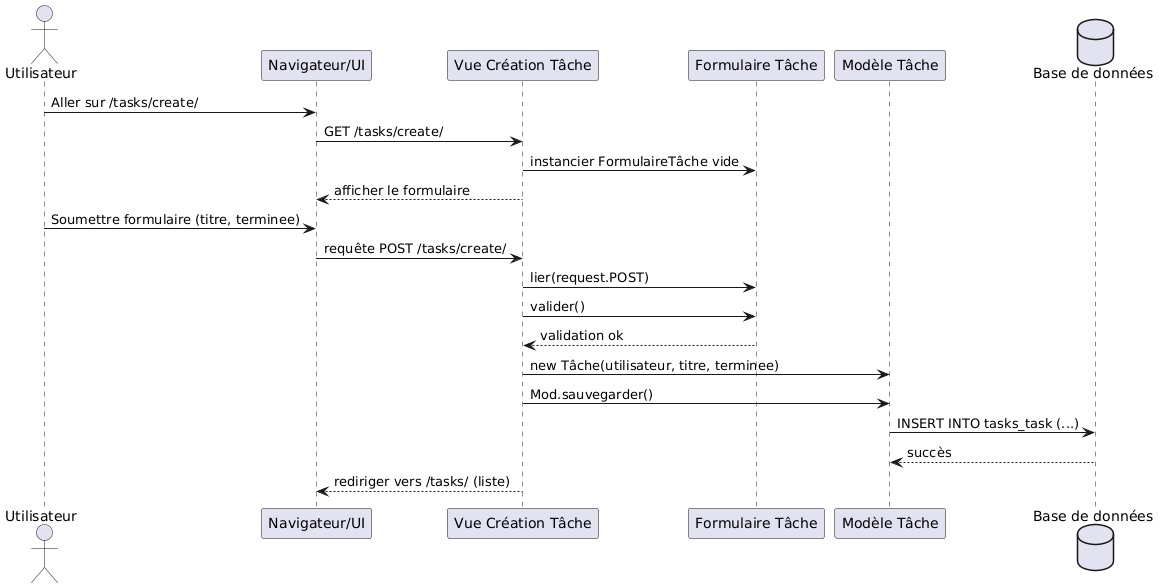
## Diagramme de classes



Ce diagramme présente les classes principales utilisées dans l’application ainsi que leurs relations.

* **FormulaireInscription** permet de collecter les données nécessaires à l’inscription d’un nouvel utilisateur (nom, email, mot de passe).
* **Utilisateur** représente l'utilisateur de l'application avec ses informations personnelles.
* **FormulaireTâche** permet de créer ou modifier une tâche avec deux champs : le titre et l'état (terminée).
* **Tâche** est la classe centrale du système de gestion des tâches, contenant les attributs id, titre, terminée, et dateCreation.  
  Les relations indiquent qu’un utilisateur possède plusieurs tâches, et que le formulaire de tâche est utilisé pour créer ou modifier une instance de la classe Tâche.

## Diagramme de séquence



Ce diagramme illustre le processus de création d'une tâche dans l'application To-Do List, du point de vue de l'interaction entre les différents composants.  
L’utilisateur accède à la page /tasks/create/ via le navigateur. Le contrôleur de la vue de création instancie un formulaire vide (FormulaireTâche) qui est ensuite affiché à l'utilisateur. Ce dernier soumet le formulaire rempli (titre, terminée), ce qui envoie une requête POST au serveur.  
Le formulaire est alors lié aux données reçues, validé, puis utilisé pour créer une nouvelle tâche (liée à l'utilisateur). Le modèle sauvegarde la tâche dans la base de données avec une requête INSERT. Après le succès de l’opération, l'utilisateur est redirigé vers la liste des tâches (/tasks/).

# Logique métier et vues

* 1. **Vue principale (my\_todos)**

```python @login\_required

def my\_todos(request):

user = request.user

now = timezone.now.strftime("%Y-%m-%d") todos = TodoList.objects.filter(user=user) category1 = Category.objects.get(id=1)

# ...

context = { "DateNow": now, "todos": todos,

"category\_i": category1, # autres clés…

```

}

return render(request, "todolist/home.html", context)

Point fort : filtrage par user.

Limitation : récupération manuelle des catégories (id=1, id=2, …) au lieu d’un Category.objects.all().

# CRUD des tâches

Des vues fonctionnelles (add\_task, update\_task, delete\_task) sont déclarées dans todolist/views.py et utilisent ModelForm. Elles suivent le pattern POST/REDIRECT/GET.

# Gestion des catégories dans les vues

* + - Factoriser la liste des catégories via un context\_processor
    - Permettre la gestion des catégories par l’utilisateur

# Templates et interface utilisateur

* 1. **Architecture MVT et héritage des templates**
     + base.html contient le squelette global.
     + Les templates spécifiques héritent et remplissent les blocs content.

# Responsive design avec Bootstrap / Tailwind

* + - Classes Bootstrap (container, row, col-md-6)
    - Grilles flexibles et points de rupture

# Accessibilité et bonnes pratiques HTML

* + - Balises sémantiques (<main>, <section>, <nav>)
    - Attributs aria-label pour les actions

# Sécurité et bonnes pratiques

* 1. **Protection CSRF et XSS**
     + csrf\_token dans tous les formulaires.
     + Échappement automatique des variables.

# Gestion des permissions et accès utilisateur

* + - @login\_required pour restreindre l’accès.
    - UserPassesTestMixin pour contrôle granulaire.

# Nettoyage des données et validation server-side

* + - clean\_<field> dans forms.py.
    - Vérifier la cohérence des dates.

# Tests, qualité et maintenance

* 1. **Structure des fichiers de tests**

Chaque app contient un tests.py, souvent vide. À implémenter : tests unitaires et fonctionnels.

# Recommandations pour tests unitaires et d’intégration

* + - pytest-django pour simplicité.
    - Couverture > 80% et intégration CI.

# Déploiement et configuration de production

## Procfile, runtime.txt et Heroku

* + - Procfile : web: gunicorn todoapp.wsgi
    - runtime.txt pour la version Python

# Gestion des fichiers statiques (WhiteNoise)

* + - STATIC\_ROOT configuré.
    - Middleware WhiteNoise.

# Exclusion du virtualenv du dépôt

* + - Ajouter Venv/ à .gitignore.
    - Utiliser Pipfile ou poetry.lock.

# Axes d’amélioration et perspectives

1. API RESTful via Django REST Framework
2. CI/CD avec GitHub Actions
3. Roles utilisateurs (Admin, Invité)
4. Tri avancé et priorisation
5. Notifications par e-mail ou WebSocket
6. Internationalisation (i18n)

# Conclusion

Le projet To-Do List avec Authentification présente une base solide de bonnes pratiques Django. Les améliorations ciblées concernent la robustesse (tests), la correction de defaults de modèles et l’industrialisation (API, CI/CD).