**Documentation de l’application de**

**Gestion des Tâches Collaboratives**

# BACKEND DJANGO REST

**Vue d'ensemble**

Cette application backend est conçue pour gérer un système de tâches collaboratives à l'ESMT, permettant aux enseignants et étudiants de travailler ensemble sur des projets. L'application est développée en Django avec Django REST Framework pour exposer une API RESTful.

**Principales fonctionnalités**

* Gestion des utilisateurs (étudiants, enseignants, admins)
* Gestion des projets
* Gestion des tâches
* Attribution de tâches aux utilisateurs
* Statistiques de progression
* Calcul des primes pour les enseignants

**Structure du projet**

Le projet est organisé en trois applications principales:

**1. Users**

Gère l'authentification, les profils utilisateurs et les rôles.

**2. Projects**

Gère les projets, leurs propriétaires et membres.

**3. Tasks**

Gère les tâches, leur attribution et leur statut.

**Modèles de données**

**CustomUser (users/models.py)**

class CustomUser(AbstractUser):

ROLE\_CHOICES = [

('admin', 'Admin'),

('enseignant', 'Enseignant'),

('etudiant', 'Étudiant'),

]

nom = models.CharField(max\_length=200)

prenom = models.CharField(max\_length=200)

email = models.EmailField(unique=True)

avatar = models.ImageField(upload\_to='avatars/', blank=True, null=True)

role = models.CharField(max\_length=20, choices=ROLE\_CHOICES, default='etudiant')

is\_active = models.BooleanField(default=True)

**Projet (projects/models.py)**

class Projet(models.Model):

nom = models.CharField(max\_length=200)

description = models.TextField()

date\_creation = models.DateField(auto\_now\_add=True)

date\_fin = models.DateField()

etat = models.CharField(

max\_length=10,

choices=EtatProjet.choices,

default=EtatProjet.EN\_ATTENTE

)

proprietaire = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)

membres = models.ManyToManyField(CustomUser, related\_name='projets')

**Tache (tasks/models.py)**

class Tache(models.Model):

STATUT\_CHOICES = [

('A\_FAIRE', 'À faire'),

('EN\_COURS', 'En cours'),

('TERMINE', 'Terminé'),

]

titre = models.CharField(max\_length=200)

description = models.TextField()

date\_limite = models.DateField()

statut = models.CharField(max\_length=10, choices=STATUT\_CHOICES, default='A\_FAIRE')

projet = models.ForeignKey(Projet, on\_delete=models.CASCADE, related\_name="taches")

assigne\_a = models.ManyToManyField(CustomUser, related\_name="taches\_assignees")

**Explication des modèles de données et des routes API disponibles**

**Modèles de données**

Le système repose sur les modèles de données principaux suivants:

**1. CustomUser (Utilisateur personnalisé)**

Le modèle CustomUser étend AbstractUser de Django pour gérer les différentes catégories d'utilisateurs.

**Champs principaux:**

* nom (CharField): Nom de l'utilisateur
* prenom (CharField): Prénom de l'utilisateur
* email (EmailField): Adresse email unique
* avatar (ImageField): Photo de profil (optionnelle)
* role (CharField): Rôle de l'utilisateur ('admin', 'enseignant', 'etudiant')
* is\_active (BooleanField): Indique si le compte est actif

**2. Projet**

Le modèle Projet représente un projet collaboratif dans le système.

**Champs principaux:**

* nom (CharField): Nom du projet
* description (TextField): Description détaillée du projet
* date\_creation (DateField): Date de création (automatique)
* date\_fin (DateField): Date d'échéance du projet
* etat (CharField): État du projet ('EN\_ATTENTE', 'EN\_COURS', 'TERMINE')
* proprietaire (ForeignKey): Référence à l'utilisateur qui a créé le projet
* membres (ManyToManyField): Utilisateurs assignés au projet

**3. Tache**

Le modèle Tache représente une tâche assignée dans le cadre d'un projet.

**Champs principaux:**

* titre (CharField): Titre de la tâche
* description (TextField): Description détaillée de la tâche
* date\_limite (DateField): Date d'échéance
* statut (CharField): Statut de la tâche ('A\_FAIRE', 'EN\_COURS', 'TERMINE')
* projet (ForeignKey): Référence au projet auquel appartient la tâche
* assigne\_a (ManyToManyField): Utilisateurs assignés à la tâche

**Diagramme des relations**

* Un CustomUser peut être propriétaire de plusieurs Projet
* Un CustomUser peut être membre de plusieurs Projet
* Un Projet a un seul propriétaire mais peut avoir plusieurs membres
* Un Projet peut avoir plusieurs Tache
* Une Tache appartient à un seul Projet
* Une Tache peut être assignée à plusieurs CustomUser

**API Endpoints**

**Documentation API**

Une documentation complète de l'API est disponible via Swagger/OpenAPI à l'adresse suivante après le lancement du serveur:

<http://127.0.0.1:8000/redoc/>

Cette interface permet de:

* Explorer tous les endpoints disponibles
* Tester les requêtes directement depuis le navigateur
* Voir les schémas de données complets
* Comprendre les exigences d'authentification pour chaque endpoint

Et si c’est pour tester l’api directement c’est avec :

http://127.0.0.1:8000/swagger/

**Routes API disponibles**

L'API REST fournit les endpoints suivants:

**Authentification**

| **Méthode** | **Endpoint** | **Description** | **Paramètres** | **Réponse** |
| --- | --- | --- | --- | --- |
| POST | /api/auth/register/ | Inscription | username, email, nom, prenom, password, role, avatar (optionnel) | Détails utilisateur + token JWT |
| POST | /api/auth/login/ | Connexion | username, password | access, refresh tokens |
| POST | /api/auth/refresh/ | Rafraîchir token | refresh token | Nouveau access token |
| GET | /api/auth/profile/ | Récupérer profil | Auth header | Détails du profil |
| PATCH | /api/auth/profile/ | Mettre à jour profil | Auth header + données | Profil mis à jour |

**Utilisateurs**

| **Méthode** | **Endpoint** | **Description** | **Paramètres** | **Réponse** |
| --- | --- | --- | --- | --- |
| GET | /api/users/ | Liste utilisateurs | Auth header | Liste des utilisateurs |
| GET | /api/users/{id}/ | Détails utilisateur | Auth header, ID | Détails de l'utilisateur |

**Projets**

| **Méthode** | **Endpoint** | **Description** | **Paramètres** | **Réponse** |
| --- | --- | --- | --- | --- |
| GET | /api/projets/ | Liste projets | Auth header | Liste des projets |
| POST | /api/projets/ | Créer projet | Auth header, Données projet | Projet créé |
| GET | /api/projets/{id}/ | Détails projet | Auth header, ID | Détails du projet |
| PUT | /api/projets/{id}/ | Modifier projet | Auth header, ID, Données | Projet mis à jour |
| DELETE | /api/projets/{id}/ | Supprimer projet | Auth header, ID | - |

**Tâches**

| **Méthode** | **Endpoint** | **Description** | **Paramètres** | **Réponse** |
| --- | --- | --- | --- | --- |
| GET | /api/taches/ | Liste tâches | Auth header, Filtres (optionnel) | Liste des tâches |
| POST | /api/taches/ | Créer tâche | Auth header, Données tâche | Tâche créée |
| GET | /api/taches/{id}/ | Détails tâche | Auth header, ID | Détails de la tâche |
| PUT | /api/taches/{id}/ | Modifier tâche | Auth header, ID, Données | Tâche mise à jour |
| DELETE | /api/taches/{id}/ | Supprimer tâche | Auth header, ID | - |
| GET | /api/projets/{projet\_id}/taches/ | Tâches par projet | Auth header, projet\_id | Liste des tâches du projet |
| POST | /api/projets/{projet\_id}/taches/ | Ajouter tâche à un projet | Auth header, projet\_id, Données tâche | Tâche créée |

**Statistiques**

| **Méthode** | **Endpoint** | **Description** | **Paramètres** | **Réponse** |
| --- | --- | --- | --- | --- |
| GET | /api/statistiques/ | Stats utilisateurs | Auth header | Stats détaillées |
| GET | /api/tache-stats/ | Stats globales | Auth header, projet (optionnel) | Stats des tâches |
| GET | /api/projets/{projet\_id}/tache-stats/ | Stats par projet | Auth header, projet\_id | Stats des tâches du projet |

**Authentification**

L'application utilise JWT (JSON Web Tokens) pour l'authentification:

1. **Inscription**:
2. POST /api/auth/register/
3. {
4. "username": "exemple",
5. "email": "exemple@esmt.sn",
6. "nom": "Nom",
7. "prenom": "Prenom",
8. "password": "motdepasse",
9. "role": "etudiant"
10. }
11. **Connexion**:
12. POST /api/auth/login/
13. {
14. "username": "exemple",
15. "password": "motdepasse"
16. }

Retourne:

{

"access": "access\_token",

"refresh": "refresh\_token"

}

1. **Utilisation du token**: Pour toutes les requêtes authentifiées, ajoutez l'en-tête:
2. Authorization: Bearer {access\_token}
3. **Rafraîchissement du token**:
4. POST /api/auth/refresh/
5. {
6. "refresh": "refresh\_token"
7. }

**Permissions**

Le système de permissions est basé sur les rôles et les relations:

**Règles générales**

* Tous les utilisateurs authentifiés peuvent voir les projets et les tâches
* Seul le propriétaire d'un projet peut le modifier ou le supprimer
* Seul le propriétaire d'un projet peut ajouter des tâches à son projet
* Les utilisateurs assignés à une tâche peuvent modifier leur statut

**Restrictions par rôle**

* Les étudiants ne peuvent pas assigner des tâches aux enseignants
* Les nouveaux comptes enseignants et admin sont désactivés par défaut
* Un admin doit approuver les comptes enseignants et admin

**Statistiques et reporting**

**Statistiques utilisateurs**

L'endpoint /api/statistiques/ fournit des informations détaillées sur les performances des utilisateurs:

* Nombre total de tâches assignées
* Nombre de tâches terminées
* Taux de complétion des tâches
* Calcul des primes pour les enseignants (100 000$ pour 100% de complétion, 30 000$ pour 90% ou plus)

**Statistiques de tâches**

Les endpoints /api/tache-stats/ et /api/projets/{projet\_id}/tache-stats/ fournissent:

* Nombre total de tâches
* Répartition par statut (À faire, En cours, Terminé)
* Pourcentage de progression global

**Installation et configuration**

**Prérequis**

Voici la liste complète de toutes les technologies utilisées dans mon projet de Gestion des Tâches Collaboratives:

* Backend (Django)
* Python 3.8+ - Langage de programmation principal
* Django 5.1.6 - Framework web backend
* Django REST Framework 3.15.2 - Framework pour créer des API REST
* Django Allauth 65.4.1 - Gestion des comptes et authentification
* Django CORS Headers 4.7.0 - Gestion des requêtes cross-origin
* Django Extensions 3.2.3 - Outils additionnels pour Django
* Django Filter 25.1 - Filtrage des données dans l'API
* djangorestframework\_simplejwt 5.5.0 - Authentification JWT
* drf-yasg 1.21.10 - Documentation automatique de l'API (Swagger/OpenAPI)
* MySQL/mysqlclient 2.2.7 - Base de données et connecteur
* PyMySQL 1.1.1 - Connecteur Python pour MySQL
* Pillow 11.1.0 - Traitement d'images (pour les avatars)
* pytest 8.3.5 - Framework de test
* pytest-django 4.10.0 - Extension pytest pour Django
* Frontend (React)
* Node.js 14+ - Environnement JavaScript
* React 18.2.0 - Bibliothèque UI
* TypeScript 4.9.3 - Langage typé basé sur JavaScript
* Vite 3.2.4 - Outil de build et serveur de développement
* React Router DOM 6.4.3 - Gestion des routes frontend
* Axios - Client HTTP pour les requêtes API
* JWT Decode - Décodage des tokens JWT
* Framer Motion 7.6.12 - Animations et transitions
* Recharts 2.8.0 - Bibliothèque de visualisation pour les graphiques
* Lucide React 0.260.0 - Bibliothèque d'icônes
* Tailwind CSS 3.2.4 - Framework CSS utilitaire
* Outils et technologies supplémentaires
* Git - Système de contrôle de version
* JSON Web Tokens (JWT) - Format pour l'authentification et l'échange d'informations
* CORS (Cross-Origin Resource Sharing) - Mécanisme de sécurité pour les requêtes HTTP
* RESTful API - Architecture pour les services web
* Swagger/OpenAPI - Documentation interactive de l'API
* Autres dépendances listées dans requirements.txt

**Configuration**

1. Cloner le dépôt
2. Installer les dépendances: pip install -r requirements.txt
3. Configurer la base de données dans settings.py
4. Exécuter les migrations: python manage.py migrate
5. Créer un superutilisateur: python manage.py createsuperuser
6. Démarrer le serveur: python manage.py runserver

**Variables d'environnement recommandées**

* SECRET\_KEY: Clé secrète Django
* DEBUG: Mode debug (True/False)
* DATABASE\_URL: URL de la base de données
* USE\_TZ: Réglage de la timezone (False recommandé pour les statistiques)

**Guide d'Installation et Documentation API**

**Instructions pour configurer et lancer le projet**

**Prérequis**

* Python 3.8+
* Gestionnaire de paquets pip
* Node.js et npm (pour le frontend)
* Base de données MySQL
* Git

**Installation du Backend (Django)**

1. **Cloner le dépôt**
2. git clone <URL\_DU\_DEPOT>
3. cd <NOM\_DU\_DOSSIER>
4. **Créer et activer un environnement virtuel**
5. # Création de l'environnement virtuel
6. python -m venv venv
7. # Activation sur Windows
8. venv\Scripts\activate
9. # Activation sur macOS/Linux
10. source venv/bin/activate
11. **Installer les dépendances Python**
12. pip install -r requirements.txt
13. **Configurer la base de données**
14. DATABASES = {
15. 'default': {
16. 'ENGINE': 'django.db.backends.mysql',
17. 'ATOMIC\_REQUESTS' : True,
18. 'NAME': 'gestion\_taches\_collaboratives',
19. 'USER': 'root',
20. 'PASSWORD': '',
21. 'HOST': 'localhost',
22. 'PORT': '3306',
23. 'OPTIONS': {
24. 'charset': 'utf8mb4',
25. 'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'",
26. },
27. }
28. }
29. SITE\_ID = 1

**Appliquer les migrations pour créer la structure de la base de données**

1. cd Gestion\_des\_taches\_collaboratives
2. python manage.py migrate
3. **Créer un superutilisateur pour l'administration**
4. python manage.py createsuperuser
5. # Suivez les instructions pour créer un compte administrateur
6. **Lancer le serveur de développement Django**
7. python manage.py runserver

Le backend sera accessible à l'adresse http://127.0.0.1:8000/

**Installation du Frontend (React)**

1. **Naviguer vers le dossier frontend**
2. cd ../gestion-taches-frontend
3. **Installer les dépendances Node.js**
4. npm install
5. **Lancer le serveur de développement React**
6. npm run dev

Le frontend sera accessible à l'adresse http://localhost:5173/

**Vérification de l'installation**

1. Accédez à l'interface d'administration Django: http://127.0.0.1:8000/admin/
2. Connectez-vous avec les identifiants du superutilisateur créé précédemment
3. Vérifiez que vous pouvez accéder au frontend React à l'adresse http://localhost:5173/
4. Testez la connexion entre le frontend et le backend en vous connectant via l'interface utilisateur

# Frontend (React avec TypeScript)

Le frontend de l'application a été développé avec React et TypeScript, utilisant une approche moderne avec des hooks personnalisés et des composants réutilisables. L'interface utilisateur est construite avec Tailwind CSS pour un design responsive et élégant.

**Technologies utilisées**

* **React** : Bibliothèque UI pour créer des interfaces utilisateur dynamiques
* **TypeScript** : Typage statique pour une meilleure maintenabilité et détection d'erreurs
* **Tailwind CSS** : Framework CSS utilitaire pour un développement rapide et cohérent
* **React Router** : Gestion des routes et navigation
* **Axios** : Client HTTP pour les requêtes API
* **Framer Motion** : Animations fluides et transitions
* **Recharts** : Bibliothèque de visualisation pour les graphiques et statistiques
* **JWT Decode** : Décodage des tokens JWT
* **Lucide React** : Bibliothèque d'icônes

**Structure du projet**

Le projet frontend est organisé en plusieurs dossiers qui séparent les différentes préoccupations :

src/

├── components/ # Composants réutilisables

│ ├── PrivateRoute.tsx

│ ├── StatsBadge.tsx

│ ├── StatsCard.tsx

│ └── ...

├── etatConnexion/ # Contexte d'authentification

│ └── AuthContext.tsx

├── hooks/ # Hooks personnalisés

│ └── useAuth.tsx

├── pagePrincipal/ # Pages principales

│ ├── Calendrier.tsx

│ ├── HomePage.tsx

│ ├── Layout.tsx

│ ├── login.tsx

│ ├── Navbar.tsx

│ ├── ProfilePage.tsx

│ ├── register.tsx

│ └── Statistiques.tsx

├── pagesProjets/ # Pages liées aux projets

│ ├── Dashboard.tsx

│ ├── ProjetDetails.tsx

│ ├── ProjetForm.tsx

│ ├── ProjetsList.tsx

│ └── ...

├── pagesTaches/ # Pages liées aux tâches

├── types/ # Définitions TypeScript

│ └── index.ts

├── axiosConfig.ts # Configuration Axios avec intercepteurs

├── App.tsx # Composant principal avec les routes

└── main.tsx # Point d'entrée de l'application

**Authentification et sécurité**

**Système d'authentification JWT**

L'authentification repose sur JSON Web Tokens (JWT) avec une approche à deux tokens :

1. **Access Token** : Token de courte durée (30 minutes) utilisé pour l'authentification des requêtes API
2. **Refresh Token** : Token de longue durée (24 heures) utilisé pour obtenir un nouveau access token sans reconnexion

**AuthContext**

Le contexte d'authentification (AuthContext.tsx) gère l'état d'authentification de l'utilisateur et fournit des méthodes pour se connecter et se déconnecter :

export const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider = ({ children }: { children: ReactNode }) => {

const [isAuthenticated, setIsAuthenticated] = useState(!!localStorage.getItem("token"));

const [user, setUser] = useState<User | null>(null);

const navigate = useNavigate();

useEffect(() => {

const token = localStorage.getItem("token");

if (token) {

try {

// Décoder le token JWT pour obtenir les informations de l'utilisateur

const decodedToken: any = jwtDecode(token);

// Créer un objet user à partir des infos du token

const userData: User = {

id: decodedToken.user\_id,

username: decodedToken.username,

role: decodedToken.role

};

setUser(userData);

setIsAuthenticated(true);

} catch (error) {

setUser(null);

setIsAuthenticated(false);

localStorage.removeItem("token");

localStorage.removeItem("refresh");

}

} else {

setUser(null);

setIsAuthenticated(false);

}

}, []);

const logout = () => {

localStorage.removeItem("token");

localStorage.removeItem("refresh");

setUser(null);

setIsAuthenticated(false);

navigate("/login");

};

return (

<AuthContext.Provider value={{ isAuthenticated, user, logout }}>

{children}

</AuthContext.Provider>

);

};

**Hook useAuth**

Un hook personnalisé useAuth facilite l'utilisation du contexte d'authentification dans les composants :

export const useAuth = () => {

const context = useContext(AuthContext);

if (context === undefined) {

throw new Error("useAuth doit être utilisé à l'intérieur d'un AuthProvider");

}

return context;

};

**Intercepteurs Axios**

Les intercepteurs Axios gèrent automatiquement l'ajout du token JWT aux requêtes et le rafraîchissement du token en cas d'expiration :

// Intercepteur pour ajouter le token dans les requêtes

api.interceptors.request.use(

(config) => {

const token = localStorage.getItem("token");

if (token) {

config.headers.Authorization = `Bearer ${token}`;

}

return config;

},

(error) => {

return Promise.reject(error);

}

);

// Intercepteur pour gérer le rafraîchissement du token

api.interceptors.response.use(

(response) => response,

async (error) => {

if (error.response && error.response.status === 401) {

try {

const refresh = localStorage.getItem("refresh");

if (!refresh) throw new Error("Aucun refresh token trouvé");

const response = await axios.post("/auth/refresh/", {

refresh,

});

// Sauvegarde le nouveau token

localStorage.setItem("token", response.data.access);

error.config.headers.Authorization = `Bearer ${response.data.access}`;

// Relance la requête avec le nouveau token

return axios(error.config);

} catch {

localStorage.removeItem("token");

localStorage.removeItem("refresh");

window.location.href = "/login";

}

}

return Promise.reject(error);

}

);

**Routes protégées**

Le composant PrivateRoute restreint l'accès aux utilisateurs authentifiés :

const PrivateRoute = () => {

const { isAuthenticated } = useAuth();

return isAuthenticated ? <Outlet /> : <Navigate to="/login" replace />;

};

**Gestion des utilisateurs**

**Inscription (Register)**

Le composant Register permet aux utilisateurs de créer un compte avec les fonctionnalités suivantes :

* Formulaire en deux étapes (informations de connexion, puis informations personnelles)
* Validation des données
* Téléchargement d'avatar
* Sélection du rôle (étudiant ou enseignant)

**Connexion (Login)**

Le composant Login permet l'authentification avec :

* Gestion du stockage des tokens JWT
* Messages d'erreur en cas d'échec d'authentification
* Indicateur de chargement
* Option "Se souvenir de moi"

**Profil utilisateur (ProfilePage)**

Le composant ProfilePage permet la gestion du profil avec :

* Affichage des informations utilisateur
* Modification des données personnelles
* Téléchargement/changement d'avatar
* Animations fluides avec Framer Motion

**Gestion des projets**

**Dashboard**

Le composant Dashboard offre une vue d'ensemble de l'activité de l'utilisateur :

* Statistiques résumées (tâches complétées, projets actifs, etc.)
* Actions rapides (créer un projet, ajouter une tâche)
* Liste des projets récents
* Aperçu des performances et des primes pour les enseignants

**Liste des projets (ProjetsList)**

Le composant ProjetsList présente tous les projets de l'utilisateur avec :

* Filtrage par état (en attente, en cours, terminé)
* Recherche par nom ou description
* Affichage en mode grille ou liste
* Pagination
* Séparation entre "Mes projets" et "Projets auxquels je participe"

**Détails du projet (ProjetDetails)**

Le composant ProjetDetails affiche les informations détaillées d'un projet :

* Informations générales (nom, description, état)
* Statistiques des tâches (à faire, en cours, terminées)
* Liste des membres
* Gestion des tâches (ajout, modification, suppression)
* Permissions basées sur le rôle (propriétaire vs. participant)

**Formulaire de projet (ProjetForm)**

Le composant ProjetForm permet la création et la modification de projets :

* Champs pour les informations du projet
* Sélection des membres
* Différences de comportement selon le rôle de l'utilisateur
* Validation des entrées

**Gestion des tâches**

La gestion des tâches est intégrée dans la vue détaillée des projets :

* Ajout de nouvelles tâches
* Assignation de tâches aux membres du projet
* Définition de dates limites
* Suivi de l'état d'avancement
* Filtrage par statut

**Statistiques et reporting**

Le composant Statistiques offre des visualisations des performances :

* Graphiques de complétion de tâches
* Distribution des primes
* Efficacité par utilisateur
* Filtrage par rôle et autres critères
* Affichage de tendances sur différentes périodes

Exemple d'utilisation de Recharts pour les graphiques :

<ResponsiveContainer width="100%" height="100%">

<PieChart>

<Pie

data={primeDistribution}

cx="50%"

cy="50%"

labelLine={false}

label={({ name, percent }) => `${name} (${(percent \* 100).toFixed(0)}%)`}

outerRadius={80}

fill="#8884d8"

dataKey="value"

>

{primeDistribution.map((entry, index) => (

<Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />

))}

</Pie>

<Tooltip />

<Legend />

</PieChart>

</ResponsiveContainer>

**Interface utilisateur**

**Design System**

L'interface utilisateur utilise un design system cohérent basé sur Tailwind CSS :

* Système de couleurs avec la palette emerald comme couleur principale
* Composants de carte réutilisables
* Badges d'état colorés
* Boutons avec états de survol et de chargement
* Modales et formulaires standardisés
* Icônes de Lucide React

**Responsive Design**

L'interface s'adapte à différentes tailles d'écran :

* Utilisation de grilles flexibles avec Tailwind
* Composants optimisés pour les appareils mobiles et de bureau
* Points d'arrêt (breakpoints) pour différentes tailles d'écran

**Animations et transitions**

L'application utilise Framer Motion pour des animations fluides :

* Transitions entre les états de l'interface
* Animations de chargement
* Effets de survol sur les cartes et boutons
* Animations d'apparition pour les notifications

**Communication avec le backend**

La communication avec le backend Django REST est gérée par Axios :

* Endpoints API pour chaque entité (utilisateurs, projets, tâches)
* Gestion des erreurs et affichage de messages appropriés
* Typage TypeScript pour les réponses API
* Intercepteurs pour la gestion des tokens JWT

**Guide d'utilisation pour les développeurs**

**Installation et configuration**

1. **Prérequis** :
   * Node.js 18+ et npm/yarn
   * Backend Django REST en cours d'exécution
2. **Installation** :
3. git clone <repo-url>
4. cd gestion-taches-frontend
5. npm install
6. **Configuration** :
7. VITE\_API\_URL=http://localhost:8000/api
8. **Lancement en développement** :
9. npm run dev
10. **Construction pour la production** :
11. npm run build

**Diagramme de l'architecture frontend**

┌─────────────────────────────────────────────────┐

│ │

│ React Application │

│ │

└───────────────────────┬─────────────────────────┘

│

┌─────────────────────────────────────────┐

│ │

│ AuthProvider │

│ (gestion de l'authentification) │

│ │

└─────────────────┬───────────────────────┘

│

┌─────────────────────┼─────────────────────────┐

│ │ │

│ ┌─────────────────────────────────────────┐ │

│ │ │ │

│ │ Composants publics (Login, etc.) │ │

│ │ │ │

│ └─────────────────────────────────────────┘ │

│ │

│ ┌─────────────────────────────────────────┐ │

│ │ Routes protégées │ │

│ │ ┌───────────────────────────────┐ │ │

│ │ │ Layout commun │ │ │

│ │ │ ┌─────────────────────────┐ │ │ │

│ │ │ │ Page principale │ │ │ │

│ │ │ │ (Dashboard, etc.) │ │ │ │

│ │ │ └─────────────────────────┘ │ │ │

│ │ │ ┌─────────────────────────┐ │ │ │

│ │ │ │ Pages de projets │ │ │ │

│ │ │ │ (Liste, Détails, etc.) │ │ │ │

│ │ │ └─────────────────────────┘ │ │ │

│ │ │ ┌─────────────────────────┐ │ │ │

│ │ │ │ Pages de statistiques │ │ │ │

│ │ │ └─────────────────────────┘ │ │ │

│ │ └───────────────────────────────┘ │ │

│ └─────────────────────────────────────────┘ │

│ │

└───────────────────┬───────────────────────────┘

│

┌───────────────┴───────────────┐

│ │

│ API Axios │

│ (communication backend) │

│ │

└───────────────┬───────────────┘

│

┌───────────────┴───────────────┐

│ │

│ Backend Django REST │

│ │

└───────────────────────────────┘

**Conclusion et perspectives d'amélioration**

Le frontend React développé pour le Système de Gestion des Tâches Collaboratives ESMT offre une interface utilisateur moderne, responsive et intuitive. L'utilisation de TypeScript, Tailwind CSS et d'autres bibliothèques modernes assure une base solide pour le développement futur.

**Améliorations potentielles**

1. **Tests** : Ajouter des tests unitaires et d'intégration avec Jest et React Testing Library
2. **État global** : Évaluer l'utilisation de Redux ou Zustand pour une gestion d'état plus avancée
3. **Performance** : Optimiser le rendu avec React.memo et useMemo
4. **Accessibilité** : Améliorer la conformité aux normes WCAG
5. **Notifications en temps réel** : Intégrer des websockets pour les mises à jour en temps réel
6. **Mode hors ligne** : Ajouter la prise en charge du mode hors ligne avec Service Workers
7. **Internationalisation** : Ajouter la prise en charge de plusieurs langues

# **FAQ :**

**Q: Comment créer un compte administrateur dans le système?**

**R**: Utilisez la commande python manage.py createsuperuser dans le terminal, puis connectez-vous à l'interface d'administration à l'adresse /admin/.

**Q: Comment ajouter de nouveaux rôles utilisateur?**

**R**: Modifiez la constante ROLE\_CHOICES dans le modèle CustomUser, puis créez et appliquez de nouvelles migrations.

**Q: Comment modifier les critères d'attribution des primes?**

**R**: Modifiez la logique de calcul des primes dans la méthode get\_queryset() de la vue StatistiqueView dans le fichier tasks/views.py.

**Q: Comment réinitialiser le mot de passe d'un utilisateur?**

**R**: Un administrateur peut réinitialiser le mot de passe via l'interface d'administration Django, ou vous pouvez utiliser la commande python manage.py changepassword username.