Plateforme de Gestion de Ligues Sportives Communautaires

Connecter Organisateurs et Joueurs Passionnés

Énoncé du Projet - Hackathon 60h

Architecture Full-Stack Django + React

Durée estimée : 60 heures
Niveau : Intermédiaire
Équipe : 2-3 personnes recommandé

November 21, 2025

Contents

1 Contexte et Objectifs

1.1 Description Générale

Le projet consiste à développer une plateforme web moderne permettant de gérer des ligues sportives communautaires. L'application connecte deux types d'utilisateurs : les **organisateurs** qui créent et gèrent des tournois et équipes, et les **joueurs** qui cherchent à rejoindre des équipes dans leur région.

Vision du Projet

Imaginez un « Meetup » pour les sports amateurs : une plateforme où chacun peut trouver ou créer des opportunités de pratiquer son sport favori, rencontrer de nouveaux coéquipiers, et participer à des compétitions locales.

1.2 Objectifs Principaux

Créer une application web **full-stack** avec séparation claire entre frontend et backend

Implémenter un système d'authentification moderne via Clerk avec gestion de rôles

Gérer le cycle complet des équipes sportives (création, adhésion, matchs)

Développer un système de demandes d'adhésion avec workflow d'approbation

Offrir une expérience utilisateur fluide adaptée à chaque rôle

Intégrer optionnellement un système de paiement via Stripe

1.3 Contraintes Spécifiques

Important - Hackathon 60h

- Concentrez-vous sur le **MVP** (Minimum Viable Product) avant toute fonctionnalité bonus
- Utilisez des outils qui accélèrent le développement (Clerk, templates UI, etc.)
- Testez chaque fonctionnalité avant de passer à la suivante
- Documentez au fur et à mesure pour gagner du temps en fin de projet

2 Architecture Technique

2.1 Stack Technologique

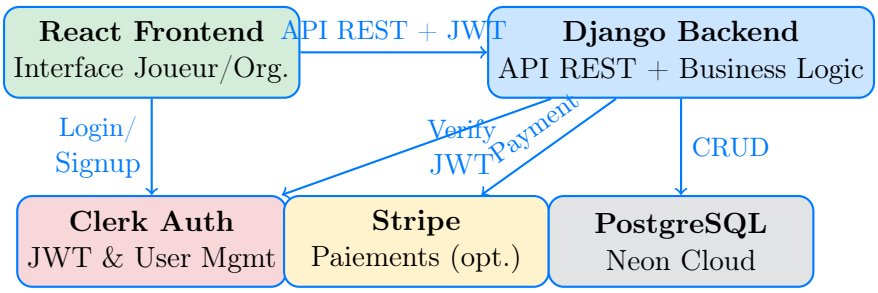
Backend

- **Framework principal :** Django 5.x avec Django REST Framework (DRF)
- **Base de données :** PostgreSQL hébergé sur Neon (cloud)
- **Authentification :** Clerk avec tokens JWT
- **Paiement (optionnel) :** Intégration Stripe API
- **Administration :** Django Admin personnalisé

Frontend

- **Framework :** React 18+ (Vite recommandé pour setup rapide)
- **Authentification UI :** Composants Clerk pré-construits
- **Communication API :** Axios avec intercepteurs pour JWT
- **Styling (suggéré) :** Tailwind CSS ou Material-UI
- **Routing :** React Router v6

2.2 Architecture Système



2.3 Structure des Apps Django

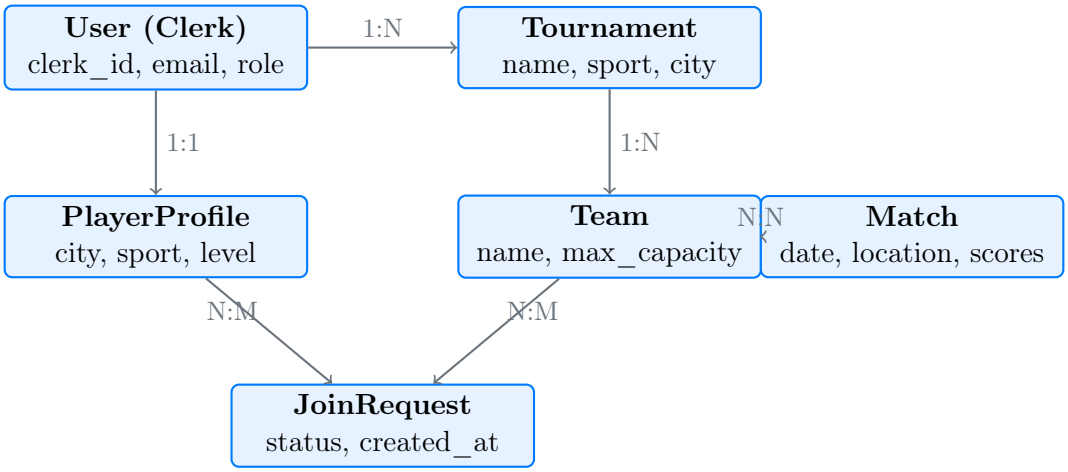
```
sports_platform/
|-- accounts/           # Gestion utilisateurs + Clerk
|   |-- models.py       # User, Profile
|   |-- views.py        # Auth endpoints
|   |-- serializers.py
|
|-- players/            # Profils joueurs
|   |-- models.py       # PlayerProfile
|   |-- views.py
|
|-- tournaments/        # Tournois et equipes
|   |-- models.py       # Tournament, Team
|   |-- views.py
|   |-- serializers.py
|
|-- requests/           # Demandes d'adhesion
```

```
|   |-- models.py       # JoinRequest
|   |-- views.py
|   |-- services.py     # Logique metier
|
|-- matches/            # Matchs
|   |-- models.py       # Match
|   |-- views.py
|
|-- payments/           # Stripe (optionnel)
    |-- views.py
    |-- webhooks.py
```

Listing 1: Organisation recommandée du backend

3 Modèle de Données

3.1 Diagramme Relationnel Simplifié



3.2 Modèles Django Détaillés

3.2.1 User & PlayerProfile

```python
from django.db import models
import uuid

class User(models.Model):
    """Utilisateur synchronise avec Clerk"""
    id = models.UUIDField(primary_key=True, default=uuid.uuid4)
    clerk_id = models.CharField(max_length=255, unique=True)
    email = models.EmailField(unique=True)
    full_name = models.CharField(max_length=255)
    role = models.CharField(max_length=20, choices=[
        ('player', 'Joueur'),
        ('organizer', 'Organisateur')
    ])
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'users'
```

Listing 2: accounts/models.py

```python
class PlayerProfile(models.Model):
    """Profil etendu pour les joueurs"""
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    city = models.CharField(max_length=100)
    favorite_sport = models.CharField(max_length=50)
    level = models.CharField(max_length=20, choices=[
        ('beginner', 'Debutant'),
        ('intermediate', 'Intermediaire'),
        ('advanced', 'Avance')
    ])
    position = models.CharField(max_length=50, blank=True)

    class Meta:
        db_table = 'player_profiles'
```

Listing 3: players/models.py

3.2.2 Tournament & Team

```python
class Tournament(models.Model):
    """Tournoi/Ligue cree par un organisateur"""
    id = models.UUIDField(primary_key=True, default=uuid.uuid4)
    name = models.CharField(max_length=200)
    sport = models.CharField(max_length=50)
    city = models.CharField(max_length=100)
    start_date = models.DateField()
    organizer = models.ForeignKey(User, on_delete=models.CASCADE,
                                  related_name='tournaments')
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'tournaments'

class Team(models.Model):
    """Equipe dans un tournoi"""
    id = models.UUIDField(primary_key=True, default=uuid.uuid4)
    name = models.CharField(max_length=200)
    tournament = models.ForeignKey(Tournament, on_delete=models.CASCADE,
                                   related_name='teams')
    max_capacity = models.IntegerField(default=15)
    current_capacity = models.IntegerField(default=0)
    members = models.ManyToManyField(User, related_name='teams',
                                     blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

    @property
    def available_spots(self):
        return self.max_capacity - self.current_capacity

    @property
    def is_full(self):
        return self.current_capacity >= self.max_capacity

    class Meta:
        db_table = 'teams'
```

Listing 4: tournaments/models.py

3.2.3 JoinRequest (Coeur du Systeme)

```python
class JoinRequest(models.Model):
    """Demande d'adhesion a une equipe"""
    STATUS_CHOICES = [
        ('pending', 'En attente'),
        ('accepted', 'Accepte'),
        ('rejected', 'Refuse')
    ]

    id = models.UUIDField(primary_key=True, default=uuid.uuid4)
    player = models.ForeignKey(User, on_delete=models.CASCADE,
                               related_name='join_requests')
    team = models.ForeignKey(Team, on_delete=models.CASCADE,
                             related_name='join_requests')
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
                              default='pending')
    message = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = 'join_requests'
        unique_together = ['player', 'team']  # Une seule demande par
    joueur/equipe
```

Listing 5: requests/models.py

3.2.4 Match

```python
class Match(models.Model):
    """Match entre deux equipes"""
    id = models.UUIDField(primary_key=True, default=uuid.uuid4)
    team_a = models.ForeignKey(Team, on_delete=models.CASCADE,
                               related_name='matches_as_team_a')
    team_b = models.ForeignKey(Team, on_delete=models.CASCADE,
                               related_name='matches_as_team_b')
    date = models.DateTimeField()
    location = models.CharField(max_length=200)
    score_a = models.IntegerField(null=True, blank=True)
    score_b = models.IntegerField(null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'matches'
```

Listing 6: matches/models.py

4 Fonctionnalités Requises (MVP)

Priorite Absolue

Les fonctionnalités suivantes constituent le **Minimum Viable Product**. Elles doivent être implémentées avant toute fonctionnalité bonus !

4.1 1. Authentification (via Clerk)

Utilisateur : Tous

- Inscription avec sélection du rôle (Joueur / Organisateur)
- Connexion via email/mot de passe ou OAuth (Google, etc.)
- Déconnexion
- Protection des routes selon le rôle
- Redirection automatique vers le bon dashboard

Temps estimé : 4-6 heures
Note : Clerk simplifie énormément cette partie, profitez-en !

4.2 2. Gestion des Profils

Joueur :

- Page « Mon profil » (`/player/profile`)
- Formulaire d'édition avec champs :
  - Nom complet
  - Ville
  - Sport principal
  - Niveau (débutant/intermédiaire/avancé)
  - Poste préféré (optionnel)
- Sauvegarde avec validation

Organisateur :

- Tableau de bord (`/organizer/dashboard`)
- Statistiques rapides :
  - Nombre de tournois créés
  - Nombre total d'équipes
  - Demandes en attente
- Liste des tournois avec actions rapides

Temps estimé : 5-7 heures

4.3 3. Tournois

Organisateur uniquement

- Créer un tournoi (`POST /api/tournaments/`)
  - Nom du tournoi
  - Sport
  - Ville
  - Date de début
- Lister ses tournois (`GET /api/tournaments/mine/`)
- Voir les détails d'un tournoi (`GET /api/tournaments/:id/`)
  - Informations de base
  - Liste des équipes du tournoi
  - Statistiques (nombre d'équipes, joueurs inscrits)
- Modifier/Supprimer un tournoi

Temps estimé : 6-8 heures

4.4 4. Équipes

Organisateur :

- Créer une équipe dans un tournoi (`POST /api/teams/`)
  - Nom de l'équipe
  - Tournoi parent
  - Capacité maximale (ex: 10-15 joueurs)
- Voir la liste des joueurs de l'équipe
- Modifier les informations de l'équipe
- Supprimer une équipe (si aucun joueur inscrit)

Joueur :

- Rechercher des équipes disponibles (`GET /api/teams/available/`)
- Filtres :
  - Par ville
  - Par sport
  - Uniquement équipes avec places disponibles
- Voir les détails d'une équipe :
  - Nom, sport, tournoi
  - Places disponibles : `X / Y`
  - Niveau requis (si spécifié)
  - Liste des membres actuels

Temps estimé : 8-10 heures

4.5 5. Demandes d'Adhésion (Fonctionnalité Centrale)

Coeur du Systeme

Cette fonctionnalité représente la logique métier principale de l'application.

Côté Joueur :

1. **Envoyer une demande**
   - Bouton « Rejoindre l'équipe » sur la page de détails
   - Formulaire avec message optionnel
   - Validation : équipe non pleine, pas de demande existante
   - Endpoint : `POST /api/join-requests/`
2. **Suivre ses demandes** (`/my-requests`)
   - Liste de toutes les demandes avec statuts :
     - En attente - badge jaune
     - Acceptée - badge vert
     - Refusée - badge rouge
   - Date de la demande + Nom de l'équipe et du tournoi
   - Possibilité d'annuler une demande en attente

Côté Organisateur :

1. **Voir les demandes reçues**
   - Page dédiée : `/organizer/requests`
   - Filtre par équipe
   - Pour chaque demande :
     - Nom du joueur + Profil (ville, niveau, sport)
     - Message de motivation + Date de la demande
2. **Accepter une demande**
   - Bouton « Accepter »
   - Endpoint : `PATCH /api/join-requests/:id/` avec `{status: 'accepted'}`
   - Logique backend :
     (a) Vérifier que l'équipe n'est pas pleine
     (b) Ajouter le joueur à `team.members`
     (c) Incrémenter `team.current_capacity`
     (d) Changer le statut de la demande
     (e) (Optionnel) Envoyer une notification au joueur
3. **Refuser une demande**
   - Bouton « Refuser »
   - Changer simplement le statut à `'rejected'`
   - (Optionnel) Raison du refus

Règles métier importantes :

- Un joueur ne peut envoyer qu'une seule demande par équipe
- Si l'équipe est pleine, empêcher les nouvelles demandes
- Une fois accepté, le joueur ne peut plus annuler

Temps estimé : 10-12 heures (fonctionnalité complexe)

4.6 6. Matchs (Version Simplifiée)

Organisateur :

- Créer un match (`POST /api/matches/`)
  - Sélectionner équipe A
  - Sélectionner équipe B (du même tournoi)
  - Date et heure
  - Lieu
- Lister les matchs de son tournoi
- Modifier un match (date, lieu)
- Supprimer un match
- (Optionnel) Saisir les scores après le match

Joueur :

- Voir les matchs de ses équipes (`/matches`)
- Informations affichées :
  - Date et heure
  - Lieu
  - Équipe adverse
  - Score (si saisi)
- Filtrer : matchs à venir / passés

Temps estimé : 6-8 heures

4.7 Récapitulatif des Endpoints API

Méthode	Endpoint	Description
Authentification		
POST	`/api/auth/register/`	Créer un compte
POST	`/api/auth/login/`	Connexion (JWT)
POST	`/api/auth/verify-token/`	Vérifier JWT Clerk
Profils		
GET	`/api/player/profile/`	Mon profil joueur
PATCH	`/api/player/profile/`	Mettre à jour profil
Tournois		
POST	`/api/tournaments/`	Créer un tournoi
GET	`/api/tournaments/mine/`	Mes tournois
GET	`/api/tournaments/:id/`	Détails tournoi
Équipes		
POST	`/api/teams/`	Créer une équipe
GET	`/api/teams/available/`	Équipes disponibles
GET	`/api/teams/:id/`	Détails équipe
GET	`/api/my-teams/`	Mes équipes (joueur)
Demandes d'Adhésion		
POST	`/api/join-requests/`	Envoyer une demande
GET	`/api/join-requests/my/`	Mes demandes
GET	`/api/join-requests/received/`	Demandes reçues (org.)
PATCH	`/api/join-requests/:id/`	Accepter/Refuser
Matchs		
POST	`/api/matches/`	Créer un match
GET	`/api/matches/my/`	Mes matchs
PATCH	`/api/matches/:id/`	Mettre à jour scores

5 Fonctionnalités Bonus (Optionnelles)

N'implémentez ces fonctionnalités que si vous avez terminé le MVP et qu'il vous reste du temps. Elles peuvent améliorer significativement votre note !

5.1 Niveau Amateur (3-5 points bonus)

- **Filtres avancés** : Niveau requis, poste recherché
- **Page publique du tournoi** : Accessible sans connexion pour promouvoir le tournoi
- **Recherche avec auto-complétion** : Recherche intelligente d'équipes
- **Dashboard avec statistiques** : Graphiques simples pour l'organisateur

Temps estimé : 4-6 heures

5.2 Niveau Intermédiaire (6-10 points bonus)

- **Système de classement automatique**
  - Calcul basé sur les victoires/défaites
  - Affichage du tableau de classement
  - Mise à jour automatique après chaque match

- **Notifications visuelles**
  - Badge de nouvelles demandes pour l'organisateur
  - Notification quand une demande est acceptée/refusée
  - Toast messages pour les actions importantes
- **Historique et statistiques**
  - Historique des matchs joués
  - Statistiques personnelles pour les joueurs
  - Rapport d'activité du tournoi

Temps estimé : 8-12 heures

5.3 Niveau Avancé (11-15 points bonus)

- **Intégration Stripe**
  - Frais d'inscription pour rejoindre une équipe
  - Checkout sécurisé
  - Webhooks pour confirmer les paiements
  - Dashboard de revenus pour l'organisateur
- **Upload de logos d'équipe**
  - Stockage sur AWS S3 ou Cloudinary
  - Redimensionnement automatique
  - Affichage dans les listes et détails
- **Système de messagerie**
  - Chat entre joueurs d'une même équipe
  - Messages entre organisateur et joueurs
  - Notifications en temps réel (WebSocket)
- **Multi-sports avec règles spécifiques**
  - Football : 11 joueurs, durée 90 min
  - Basketball : 5 joueurs, 4 quarts-temps
  - Volleyball : 6 joueurs, sets
  - Règles de validation adaptées par sport

Temps estimé : 15-20 heures

6 Exigences Techniques

6.1 Sécurité

Critère Essentiel

- **Authentification JWT via Clerk** : Tous les endpoints sensibles doivent vérifier le token
- **Validation des données** : Utiliser les serializers DRF avec validation stricte
- **Protection CSRF** : Configuration Django appropriée
- **Variables d'environnement** : Pas de secrets dans le code (utiliser .env)
- **Permissions DRF** : Vérifier le rôle (joueur/organisateur) sur chaque endpoint
- **Neon PostgreSQL** : Connexion sécurisée via SSL

6.2 Performance

▷ **Pagination** : Listes d'équipes, demandes, matchs (page size: 20)

▷ **Select related / Prefetch** : Optimiser les requêtes Django ORM

▷ **Indexation base de données** : Index sur les champs fréquemment filtrés

▷ **Lazy loading React** : Code splitting des routes

6.3 Qualité du Code

⋆ **Code propre et commenté** : Docstrings pour les fonctions importantes

⋆ **Gestion d'erreurs** : Try/catch appropriés, messages clairs

⋆ **Git commits** : Messages descriptifs, commits atomiques

⋆ **Structure modulaire** : Séparation des responsabilités (views, models, serializers)

6.4 Tests (Fortement Recommandé)

Backend (Django) :

- Tests unitaires des modèles (validations, méthodes)
- Tests des endpoints API (CRUD complet)
- Tests de la logique métier (acceptation de demandes, etc.)

Frontend (React) - Optionnel :

- Tests des composants principaux (Jest + React Testing Library)
- Tests d'intégration des formulaires

7 Planning Détaillé - 60 Heures

Jour 1 : Setup & Backend Core (20h)

Heures 1-4 : Configuration

- Setup Django + DRF
- Configuration PostgreSQL Neon
- Configuration Clerk (comptes test)
- Setup React + Vite
- Repository Git + structure

Heures 5-12 : Modèles & API Auth

- Création des modèles Django
- Migrations
- Setup Django Admin
- Endpoints authentification + profils
- Tests Postman/Insomnia/RestClient

Heures 13-20 : API Tournois & Équipes

- ViewSets DRF pour tournois
- ViewSets pour équipes
- Serializers avec validations
- Filtres et recherche
- Tests complets des endpoints

Jour 2 : Backend Complet & Frontend Base (20h)

Heures 21-28 : API Demandes & Matchs

- Endpoints demandes d'adhésion
- Logique métier (accepter/refuser)
- Endpoints matchs
- Tests de tous les workflows
- Documentation API (README ou Postman)

Heures 29-36 : Frontend Structure

- Setup routing React Router
- Intégration Clerk (login/signup)
- Axios setup avec intercepteurs JWT
- Layout de base (header, sidebar, footer)
- Pages vides avec navigation

Heures 37-40 : Pages Joueur

- Page profil + formulaire
- Page recherche d'équipes
- Page détails équipe

Jour 3 : Frontend Complet & Polish (20h)

Heures 41-48 : Fonctionnalités Principales

- Système de demandes d'adhésion (joueur)
- Page gestion demandes (organisateur)
- Pages tournois/équipes (organisateur)
- Page matchs

Heures 49-54 : Finitions

- Gestion d'erreurs (messages, fallbacks)
- Loading states et spinners
- Responsive design
- Validations formulaires côté client
- UX polish (transitions, feedback visuel)

Heures 55-60 : Documentation & Tests

- README complet (installation, usage)
- Documentation API
- Tests end-to-end manuels
- Corrections de bugs
- Préparation démo/présentation

8 Django Admin Personnalisé

8.1 Vue d'Administration Requise

```
from django.contrib import admin
from .models import Tournament, Team, JoinRequest

@admin.register(Tournament)
class TournamentAdmin(admin.ModelAdmin):
    list_display = ['name', 'sport', 'city', 'organizer', 'start_date',
    'team_count']
    list_filter = ['sport', 'city', 'start_date']
    search_fields = ['name', 'organizer__full_name']

    def team_count(self, obj):
        return obj.teams.count()
    team_count.short_description = 'Nombre d\'equipes'

@admin.register(Team)
class TeamAdmin(admin.ModelAdmin):
    list_display = ['name', 'tournament', 'current_capacity', '
    max_capacity', 'is_full_display']
```

```
    list_filter = ['tournament__sport']
    filter_horizontal = ['members']

    def is_full_display(self, obj):
        return 'Oui' if obj.is_full else 'Non'
    is_full_display.short_description = 'Equipe pleine'

@admin.register(JoinRequest)
class JoinRequestAdmin(admin.ModelAdmin):
    list_display = ['player', 'team', 'status', 'created_at']
    list_filter = ['status', 'created_at']
    actions = ['accept_requests', 'reject_requests']

    def accept_requests(self, request, queryset):
        for req in queryset:
            if not req.team.is_full:
                req.status = 'accepted'
                req.team.members.add(req.player)
                req.team.current_capacity += 1
                req.team.save()
                req.save()
    accept_requests.short_description = 'Accepter les demandes
   selectionnees'
```

Listing 7: tournaments/admin.py - Exemple

8.2 Page Admin Spéciale : Inscriptions par Cours

Exigence Spécifique

Créer une page d'administration personnalisée affichant :

- Liste de tous les tournois
- Pour chaque tournoi, les équipes associées
- Pour chaque équipe :
  - Liste des joueurs inscrits (membres)
  - Nombre de places restantes
  - Statut des demandes en attente
- Possibilité d'exporter en CSV

9 Livrables Attendus

9.1 1. Code Source

- **Repository GitHub** avec :
  - Structure claire : `backend/` et `frontend/`
  - `.gitignore` approprié
  - Commits réguliers avec messages descriptifs
  - Branches : `main`, `develop`, feature branches

- **Fichiers de configuration** :
  - `backend/requirements.txt` (Python)
  - `frontend/package.json` (Node)
  - `.env.example` avec toutes les variables nécessaires
  - `docker-compose.yml` (optionnel mais recommandé)

9.2 2. Documentation

1. **README.md principal**
   - Description du projet
   - Technologies utilisées
   - Prérequis (Python 3.11+, Node 18+, PostgreSQL)
   - Instructions d'installation pas à pas
   - Variables d'environnement
   - Commandes pour lancer l'application
   - Captures d'écran (optionnel mais apprécié)
2. **Documentation API**
   - Liste des endpoints avec méthodes HTTP
   - Paramètres et body des requêtes
   - Exemples de réponses (JSON)
   - Codes d'erreur possibles
   - Collection Postman exportée (optionnel)
3. **Modèle de données**
   - Diagramme entité-relation (ERD)
   - Format : image PNG/PDF ou fichier draw.io
   - Description des relations

9.3 3. Présentation (5-10 minutes)

Structure de la Présentation

1. **Introduction (1 min)**
   - Présentation de l'équipe
   - Contexte du projet
2. **Démonstration en direct (4-6 min)**
   - Scénario utilisateur complet :
     (a) Organisateur crée un tournoi et une équipe
     (b) Joueur s'inscrit et recherche l'équipe
     (c) Joueur envoie une demande
     (d) Organisateur accepte la demande
     (e) Création d'un match
     (f) Joueur voit le match dans son espace
   - Montrer le Django Admin
3. **Architecture technique (2-3 min)**
   - Schéma de l'architecture
   - Choix technologiques justifiés
   - Points forts de l'implémentation
4. **Difficultés et solutions (1-2 min)**
   - Principaux défis rencontrés
   - Comment vous les avez résolus
   - Ce que vous avez appris

10 Critères d'Évaluation

Critère	Points	Détails
Fonctionnalités MVP	40%	• Auth + Profils : 8% • Tournois + Équipes : 10% • Demandes adhésion : 12% • Matchs : 6% • Admin Django : 4%
Qualité Technique	25%	• Code propre et structuré : 10% • Architecture respectée : 8% • Performance : 4% • Tests : 3%
Sécurité	15%	• Authentification JWT : 7% • Validation des données : 5% • Gestion des erreurs : 3%
Interface Utilisateur	10%	• UX intuitive : 5% • Design cohérent : 3% • Responsive : 2%
Documentation	5%	• README complet : 2% • Documentation API : 2% • Diagramme BDD : 1%
Présentation	5%	• Clarté : 2% • Démo fonctionnelle : 2% • Justifications techniques : 1%
Bonus	+15%	Fonctionnalités optionnelles
TOTAL	**100%**	(+ bonus potentiel)

11 Conseils Pratiques pour Réussir

11.1 Gestion du Temps

- **Planifiez vos sprints** : Utilisez Trello ou GitHub Projects
- **Priorisez impitoyablement** : MVP d'abord, bonus ensuite
- **Timeboxing** : Limitez le temps sur chaque fonctionnalité
- **Commits fréquents** : Au moins toutes les 2-3 heures
- **Tests au fur et à mesure** : Ne pas tout tester à la fin

11.2 Outils pour Accélérer le Développement

- **Backend** :
  - Django Extensions pour shell_plus
  - DRF Spectacular pour documentation API auto

  - Django Debug Toolbar
  - Factory Boy pour fixtures de test

- **Frontend** :
  - Vite au lieu de Create React App (plus rapide)
  - Tailwind UI ou Material-UI (composants prêts)
  - React Hook Form (formulaires simplifiés)
  - React Query (cache et gestion d'état API)

- **DevOps** :
  - Docker Compose pour environnement cohérent
  - Postman Collections pour tester l'API
  - Ngrok pour tester webhooks Stripe localement

11.3 Pièges à Éviter

Attention !

× Vouloir tout rendre parfait dès le début

× Négliger la sécurité (JWT, validations)

× Ne pas tester l'API avant de faire le frontend

× Oublier la gestion d'erreurs (formulaires, API)

× Sous-estimer le temps de déploiement

× Faire du over-engineering sur des détails

× Ne pas lire la documentation officielle

11.4 Checklist Avant Rendu

Toutes les fonctionnalités MVP fonctionnent

README.md complet avec instructions claires

.env.example fourni avec toutes les variables

Code commenté aux endroits critiques

Pas de secrets (API keys) dans le code

Django Admin accessible et fonctionnel

Tests manuels complets effectués

Documentation API à jour

Diagramme de BDD inclus

Présentation préparée (slides ou démo live)

12 Ressources et Références

12.1 Documentation Officielle

- Django Documentation - Guide complet du framework
- Django REST Framework - API REST avec Django
- React Documentation - Guide officiel React
- Clerk Documentation - Authentification moderne
- Neon PostgreSQL - Base de données cloud
- Stripe API - Intégration paiements (bonus)

12.2 Tutoriels Recommandés

- **Django + DRF** : TestDriven.io - Tutoriels Django avancés
- **React + Axios** : React Learn - Tutoriel officiel interactif
- **Clerk + React** : Clerk React Quickstart
- **JWT avec DRF** : Documentation DRF Simple JWT

12.3 Outils Recommandés

IDE et Éditeurs :

- VS Code avec extensions : Python, ESLint, Prettier, GitLens
- PyCharm Professional (gratuit pour étudiants)

API Testing :

- Postman (interface graphique complète)
- Insomnia (alternative légère)
- HTTPie (CLI simple et élégant)
- Rest Client (app de VSCODE)

Version Control :

- Git + GitHub (ou GitLab)
- GitKraken (interface graphique Git)

Project Management :

- Trello (kanban simple)
- GitHub Projects (intégré)
- Notion (documentation + tâches)

Communication :

- Discord (vocal + screen share)
- Slack (professionnel)
- GitHub Issues (tracking bugs)

13 FAQ - Questions Fréquentes

Q : Dois-je déployer l'application en ligne ?
R : Non, ce n'est pas obligatoire. Une démonstration en local est suffisante. Cependant, un déploiement sur Heroku, Render ou Vercel peut être un bonus apprécié.

Q : Comment gérer Clerk avec Django ?
R : Clerk fournit un JWT que vous devez vérifier côté Django. Utilisez la bibliothèque `PyJWT` ou `django-clerk` pour valider les tokens. Stockez le `clerk_id` dans votre modèle User.

Q : Puis-je utiliser d'autres bibliothèques React ?
R : Oui, tant que vous respectez la stack principale (React + Axios). Vous pouvez ajouter React Query, Redux, Zustand, etc.

Q : Dois-je implémenter Stripe ?
R : Non, Stripe est optionnel et considéré comme une fonctionnalité avancée. Concentrez-vous sur le MVP d'abord.

Q : Comment tester les paiements Stripe localement ?
R : Utilisez Stripe CLI avec les webhooks locaux : `stripe listen -forward-to localhost:8000/webhooks/stripe/`

Q : Puis-je travailler seul ou en équipe ?
R : Non. Une équipe de 2-3 personnes est recommandée pour un projet de 60h (répartition frontend/backend).

Q : Que faire si je suis bloqué ?
R : 1) Consultez la documentation officielle, 2) Cherchez sur Stack Overflow, 3) Demandez de l'aide à votre professeur, 4) Regardez des tutoriels vidéo sur YouTube. 5) Utilisez l'IA (ChatGPT - Claude - Gemini) **pour posez vos questions et non pas pour le copier coller**.

Q : Comment gérer les CORS entre Django et React ?
R : Installez `django-cors-headers` et configurez-le dans `settings.py`. Autorisez `localhost:5173` (Vite) ou `localhost:3000` (CRA) en développement.

Q : Quelle base de données utiliser en local pour les tests ?
R : Vous pouvez utiliser SQLite pour les tests rapides, mais Neon PostgreSQL est recommandé dès le début pour éviter les surprises en production.

14 Exemples de Code

14.1 Configuration Clerk Backend

```python
import jwt
from django.conf import settings
from django.http import JsonResponse

class ClerkAuthMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        if request.path.startswith('/api/'):
            auth_header = request.META.get('HTTP_AUTHORIZATION')
            if auth_header:
                try:
                    token = auth_header.split(' ')[1]
                    payload = jwt.decode(
                        token,
                        settings.CLERK_PUBLIC_KEY,
                        algorithms=['RS256']
                    )
                    request.clerk_user_id = payload['sub']
                except jwt.InvalidTokenError:
                    return JsonResponse({'error': 'Invalid token'},
    status=401)

        return self.get_response(request)
```

Listing 8: accounts/middleware.py - Verification JWT

14.2 Configuration Axios Frontend

```javascript
import axios from 'axios';
import { useAuth } from '@clerk/clerk-react';

const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL || 'http://localhost:8000/api',
  headers: {
    'Content-Type': 'application/json',
  },
});

// Intercepteur pour ajouter le token JWT
api.interceptors.request.use(
  async (config) => {
    const { getToken } = useAuth();
    const token = await getToken();

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

    return config;
  },
  (error) => Promise.reject(error)
```

```
);

// Intercepteur pour gerer les erreurs
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      // Rediriger vers login
      window.location.href = '/sign-in';
    }
    return Promise.reject(error);
  }
);

export default api;
```

Listing 9: frontend/src/api/axios.js

Bon développement !

Ce projet constitue une excellente opportunité d'apprentissage des technologies modernes du développement web full-stack.

Prenez le temps de bien comprendre chaque concept, testez fréquemment, et n'hésitez pas à demander de l'aide.

Profitez de ce hackathon pour apprendre et progresser !

Bonne chance à toutes et à tous et surtout AMUSEZ VOUS BIEN !