Municipal Flag NFT Game - Project Requirements and Solutions

Project Overview

Type: Functional Demonstration (Not Production) **Deadline:** December 10th, 2025 **Budget:** USD 512.29
Client Requirement: Private repository, no public sharing

---

1. Core Concept

A web-based NFT game where players collect flags of real municipalities, grouped by country and region. Each flag is an AI-generated NFT with pair acquisition mechanics and social features.

---

2. Requirements and Solutions

2.1 NFT System

Requirement	Solution
Each municipal flag has a pair of identical NFTs	ERC-721 smart contract with pair tracking logic (tokenId mapping to municipality)
First unit: Free, multiple users can request as interest sign	Backend tracks "interested users" list per flag; no blockchain transaction needed for interest
Second unit: Acquired through purchase, completes the pair	Smart contract `purchaseSecondNFT()` function that mints to buyer and marks pair as complete
Completed pair is removed from game	Contract state marks pair as "closed"; frontend filters out closed pairs

2.2 Flag Categories and Discounts

Requirement	Solution
Standard: No discounts	Base price stored in contract/database
Plus: 50% discount on future Standard subscription	Backend tracks user's Plus ownership; applies discount on checkout
Premium: 75% permanent discount on Standard	Backend tracks Premium ownership; applies permanent discount multiplier
Category assignment per flag	Admin panel allows setting category per municipality/flag

2.3 Geographic Hierarchy

Requirement	Solution
Country > Region > Municipality structure	Database schema: `countries` → `regions` → `municipalities` tables with foreign keys
4 Countries	Seed database with 4 configurable countries
1 Region per country (4 total)	Each country has 1 region record
2 Municipalities per region (8 total)	Each region has 2 municipality records
8 Flags per municipality (64 total flags)	Each municipality has 8 flag records
Admin can change displayed countries/regions/municipalities	Admin panel with CRUD operations; toggle visibility flags

2.4 NFT Naming and Metadata

Requirement	Solution
NFT name = coordinates of municipality headquarters	Store lat/long in municipality record; format as NFT name
Various location types (fire station, bakery, etc.)	Add `location_type` field to flag; include in metadata
Metadata storage	JSON metadata uploaded to IPFS; contract stores IPFS URI

2.5 AI Image Generation

Requirement	Solution
AI-generated base image per municipality	Use Stable Diffusion API to generate 1 base image per municipality
AI-generated NFT images from base image	Use img2img Stable Diffusion to create 8 variations per municipality
Pre-generation for demo	Batch generate all 64 flag images before demo; store in IPFS

2.6 Social System

Requirement	Solution
View interested parties by insignia	Database table `flag_interests` with user_id and flag_id
View owners by insignia	Query contract events or backend cache of ownership
Connection between players	`user_connections` table (follower/following relationship)
Off-chain auction for closed flags	`auctions` table with bid history; winner determined by backend logic
Ranking and reputation	Calculate scores based on: flags owned, connections, auction wins

2.7 Blockchain Infrastructure

Requirement	Solution
Polygon network for reduced cost	Deploy to Polygon Amoy testnet (Mumbai deprecated)
ERC-721 standard	OpenZeppelin ERC721 base contract
Pair logic in contract	Custom `flagPairs` mapping tracking first/second mint status
Wallet integration	MetaMask connection via ethers.js or web3.js
Testnet deployment	Hardhat deployment scripts with Polygon Amoy configuration

2.8 Technical Stack (Agreed)

Component	Solution
Backend	Python with FastAPI
Frontend	React (JavaScript)
Smart Contracts	Solidity with Hardhat
AI Generation	Stable Diffusion (local or API)
Image/Metadata Storage	IPFS (Pinata or Infura)
Database	PostgreSQL or SQLite (for demo)

2.9 Admin Panel

Requirement	Solution
Change countries/regions/municipalities	CRUD endpoints + React admin interface
Change displayed NFTs	Edit flag metadata, regenerate images, update IPFS links
Toggle visibility	Boolean `is_visible` field on all geographic entities
Easy to use	Simple forms with dropdowns and toggles

2.10 Documentation Requirements

Requirement	Solution
Technical manual (backend, frontend, contracts)	Markdown documentation covering architecture, API, components
System architecture diagram	Visual diagram showing all components and data flow
Setup instructions	Step-by-step guide for each component
Wallet connection guide	Instructions for MetaMask setup and testnet configuration
Demo script	Written walkthrough for demonstrating all features

Requirement	Solution
Hardhat test documentation	Test descriptions, how to run, expected outputs

3. Demo Scope Boundaries

In Scope

- Functional demonstration of all core features
- 4 countries, 4 regions, 8 municipalities, 64 flags
- Internal social system (interests, ownership, ranking, off-chain auction)
- Admin panel for geographic and NFT management
- Polygon testnet deployment
- Complete documentation

Out of Scope

- Production deployment
- External social network integration
- Real payment processing
- High availability / scaling
- Mobile app

4. Deliverables Summary

1. **Smart Contract** - ERC-721 with pair logic, deployed to Polygon testnet
2. **Backend API** - FastAPI with all endpoints for game logic
3. **Frontend** - React app with navigation, wallet connection, all user flows
4. **Admin Panel** - Interface for owner to manage content
5. **AI-Generated Images** - 64 flag images stored on IPFS
6. **Documentation Package** - Technical manual, architecture diagram, setup guide, demo script
7. **Hardhat Tests** - Unit tests with documentation

5. Key Technical Decisions

Decision	Rationale
Off-chain interest tracking	Avoids gas costs for "free" first unit
Off-chain auctions	Simpler implementation, no complex auction contract needed
Pre-generated images	Ensures demo stability, avoids AI latency during demo
SQLite for demo	Zero configuration, easy setup for demonstration
Polygon Amoy testnet	Mumbai deprecated; Amoy is current Polygon testnet

6. Risk Mitigation

Risk	Mitigation
Stable Diffusion setup complexity	Use cloud API (Replicate, Stability AI) as fallback
IPFS pinning reliability	Use Pinata with paid tier for demo reliability
Testnet faucet issues	Pre-fund test wallets before demo
Time constraints	Pre-generate all images; focus on core flow first