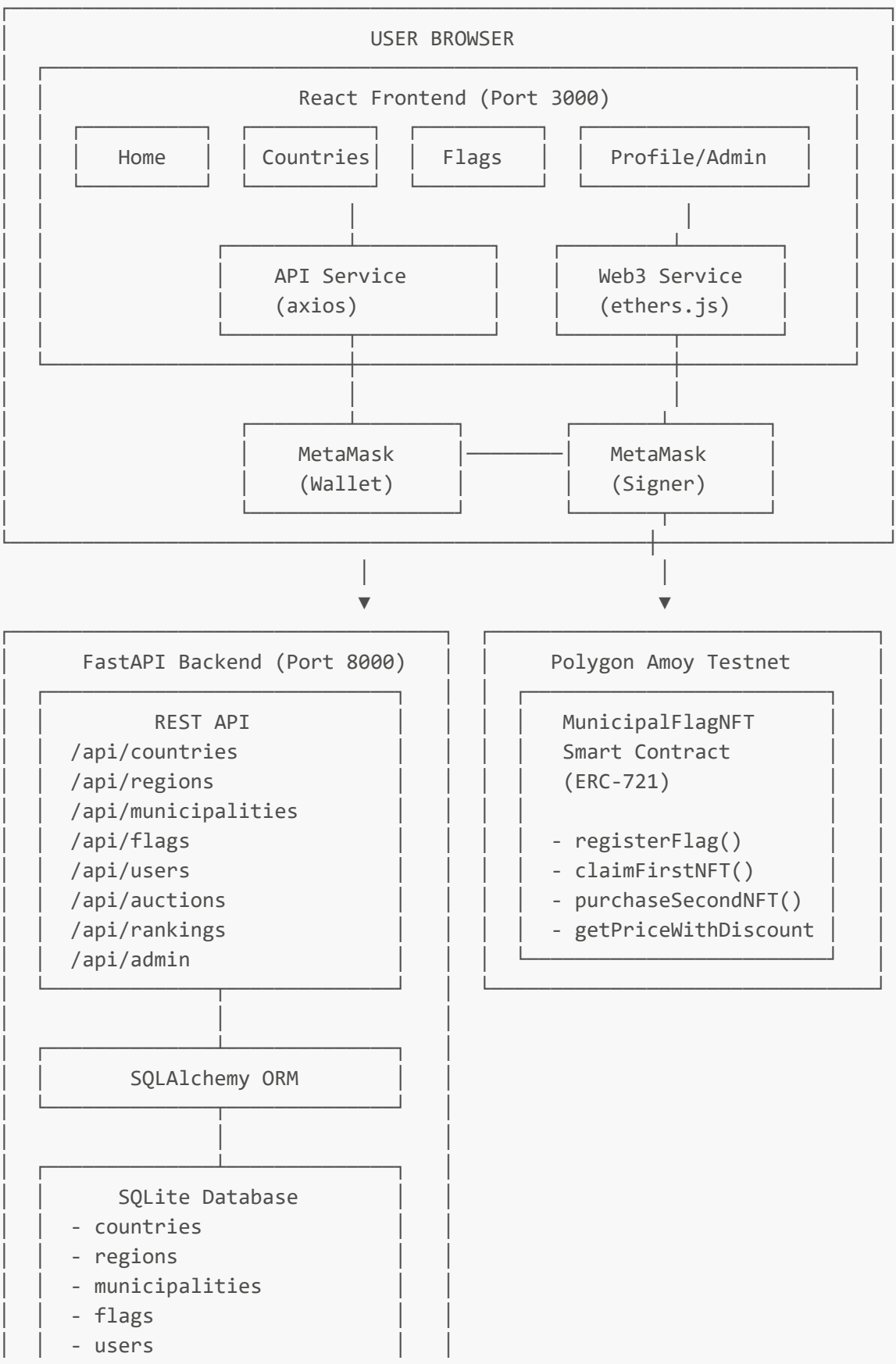
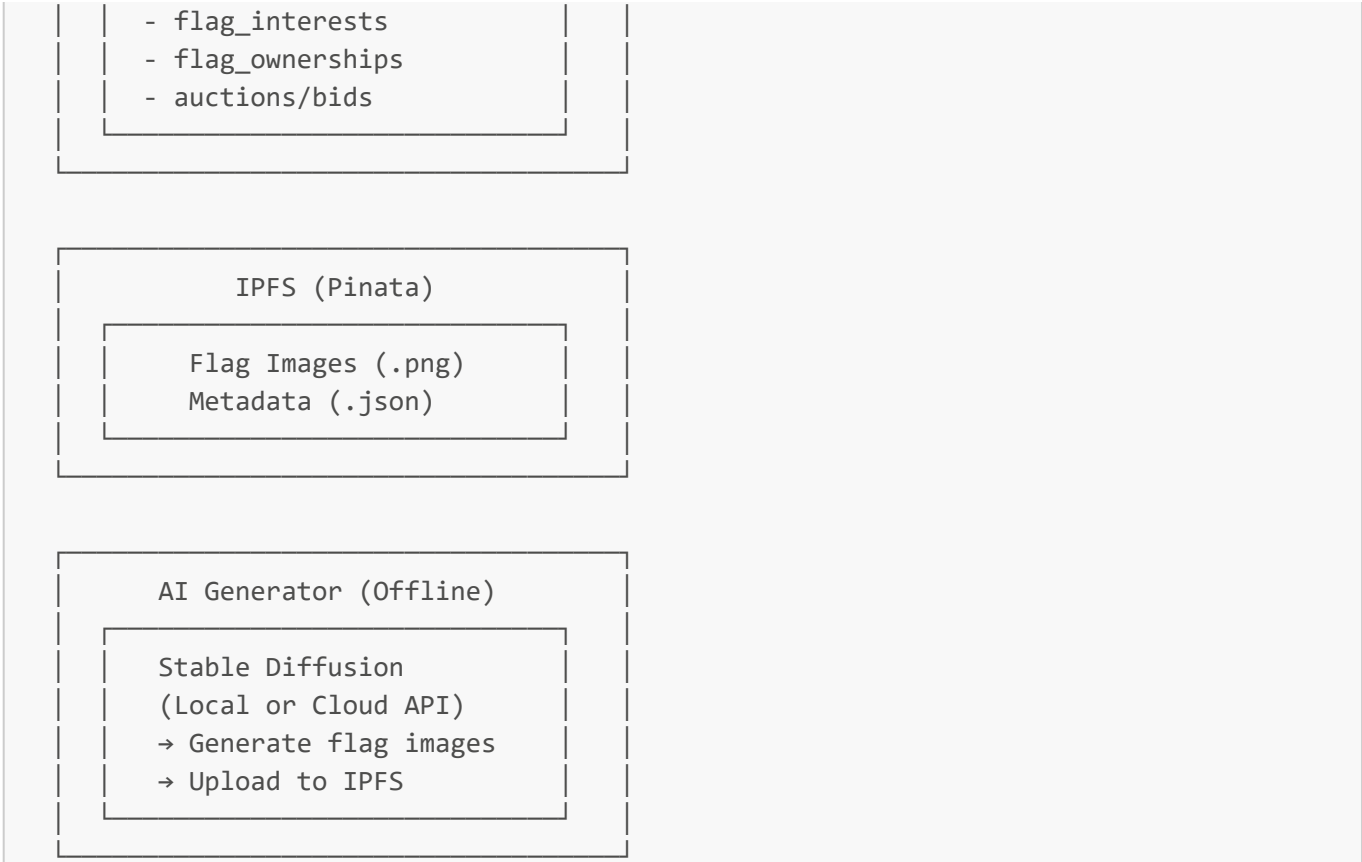
System Architecture - Municipal Flag NFT Game

Overview Diagram





Component Details

Frontend (React)

Technology: React 18, React Router, ethers.js, axios

Key Files:

- `src/App.js` - Main routing and layout
- `src/context/WalletContext.js` - Wallet state management
- `src/services/api.js` - Backend API calls
- `src/services/web3.js` - Blockchain interactions
- `src/pages/*` - Page components
- `src/components/*` - Reusable UI components

Data Flow:

1. User action triggers API call or Web3 transaction
2. API service handles REST requests to backend
3. Web3 service handles contract interactions via MetaMask
4. State updates trigger UI re-renders

Backend (FastAPI)

Technology: Python 3.9+, FastAPI, SQLAlchemy, SQLite

Key Files:

- `main.py` - Application entry point
- `config.py` - Centralized configuration
- `models.py` - Database models
- `schemas.py` - Pydantic validation
- `routers/*.py` - API endpoints
- `database.py` - DB connection

API Structure:

```
/api
├── /countries      # Geographic: countries CRUD
├── /regions        # Geographic: regions CRUD
├── /municipalities # Geographic: municipalities CRUD
├── /flags          # NFT flags + interest/ownership
├── /users          # User profiles + social
├── /auctions       # Off-chain auction system
├── /rankings       # Leaderboards
└── /admin          # Admin functions
```

Smart Contract (Solidity)

Technology: Solidity 0.8.20, OpenZeppelin, Hardhat

Contract: `MunicipalFlagNFT.sol`

- Inherits: ERC721, ERC721Enumerable, ERC721URIStorage, Ownable
- Storage: Flag pairs, token mappings, discount eligibility

Key Functions:

Function	Access	Description
`registerFlag`	Owner	Register new flag with category/price
`claimFirstNFT`	Public	Claim free first NFT
`purchaseSecondNFT`	Public (payable)	Buy second NFT to complete pair
`getPriceWithDiscount`	View	Calculate discounted price
`withdraw`	Owner	Withdraw contract balance

Database Schema

```
countries
├── id (PK)
├── name
├── code (unique)
├── is_visible
└── created_at
```

```
regions
├── id (PK)
├── name
├── country_id (FK → countries)
├── is_visible
└── created_at

municipalities
├── id (PK)
├── name
├── region_id (FK → regions)
├── latitude
├── longitude
├── is_visible
└── created_at

flags
├── id (PK)
├── municipality_id (FK → municipalities)
├── name (coordinates)
├── location_type
├── category (enum)
├── image_ipfs_hash
├── metadata_ipfs_hash
├── token_id
├── price
├── first_nft_status
├── second_nft_status
├── is_pair_complete
└── created_at

users
├── id (PK)
├── wallet_address (unique)
├── username
├── reputation_score
└── created_at

flag_interests
├── id (PK)
├── user_id (FK → users)
├── flag_id (FK → flags)
└── created_at

flag_ownerships
├── id (PK)
├── user_id (FK → users)
├── flag_id (FK → flags)
├── ownership_type (enum)
├── transaction_hash
└── created_at

user_connections
```

```
├── id (PK)
├── follower_id (FK → users)
├── following_id (FK → users)
└── created_at

auctions
├── id (PK)
├── flag_id (FK → flags)
├── seller_id (FK → users)
├── starting_price
├── current_highest_bid
├── highest_bidder_id (FK → users)
├── status (enum)
├── ends_at
└── created_at

bids
├── id (PK)
├── auction_id (FK → auctions)
├── bidder_id (FK → users)
├── amount
└── created_at
```

Data Flow Examples

1. Claiming First NFT

```
User clicks "Claim First NFT"
        │
        ▼
Frontend calls web3.claimFirstNFT(flagId)
        │
        ▼
MetaMask prompts for transaction
        │
        ▼
Transaction sent to Polygon Amoy
        │
        ▼
Contract.claimFirstNFT() executes
  - Mints NFT to user
  - Emits FirstNFTClaimed event
        │
        ▼
Frontend receives transaction hash
        │
        ▼
Frontend calls API POST /flags/{id}/claim
  - Records ownership in database
  - Updates flag status
        │
```

```
        ▼
UI refreshes with new status
```

2. Geographic Navigation

```
User visits /countries
        │
        ▼
Frontend calls api.getCountries()
        │
        ▼
Backend queries: SELECT * FROM countries WHERE is_visible = true
        │
        ▼
Returns JSON list of countries
        │
        ▼
User clicks country → /countries/{id}
        │
        ▼
Frontend calls api.getCountry(id)
        │
        ▼
Backend queries country with regions
        │
        ▼
User clicks region → municipality → flags
```

Security Considerations

1. **Smart Contract**:
    - Only owner can register flags
    - Price validation before purchase
    - Reentrancy protection via OpenZeppelin
2. **Backend**:
    - Admin endpoints require API key header
    - Wallet address validation on all user endpoints
    - SQL injection prevention via SQLAlchemy ORM
3. **Frontend**:
    - No sensitive data stored in browser
    - All transactions require MetaMask approval
    - Environment variables for configuration

Scaling Considerations

For production deployment:

1. **Database**: Migrate from SQLite to PostgreSQL
2. **Caching**: Add Redis for rankings and frequently accessed data
3. **CDN**: Serve IPFS images through CDN
4. **Load Balancing**: Multiple backend instances
5. **Monitoring**: Add logging and metrics collection