Webopoly: Game API Design

A Multiplayer Turn-Based Property Game

CSC 667 - Milestone 3

Game Overview

What is Webopoly?

A turn-based, Monopoly-style multiplayer board game where players:

- **Roll dice** and move around a shared board
- **Buy properties** strategically to build their portfolio
- **Pay rent** when landing on opponent-owned spaces
- **Manage money** to avoid bankruptcy

Win Condition

The last player standing (not bankrupt) wins!

Game Overview

Key Features

- 2-6 players per game
- Real-time updates via Socket.io
- Server-validated gameplay (no cheating!)

API Endpoints Overview

Game Management (4 endpoints)

`POST /api/games` - Create new game lobby

`POST /api/games/:game_id/join` - Join existing game

`POST /api/games/:game_id/start` - Start the game

`GET /api/games/:game_id/state` - Get current game state

API Endpoints Overview

Game Actions (4 endpoints)

`POST /api/games/:game_id/turn/roll` - Roll dice & move

`POST /api/games/:game_id/properties/:property_id/buy` - Buy property

`POST /api/games/:game_id/properties/:property_id/pay-rent` - Pay rent

`POST /api/games/:game_id/turn/end` - End current turn

Complex Endpoint #1 - Roll Dice & Move

Route: `POST /api/games/:game_id/turn/roll`

Purpose

Rolls dice, moves player, updates position, and triggers tile-specific effects (property purchase, rent payment, etc.)

Authorization Checks

- User must be authenticated
- User must be a player in this game
- Must be the current player's turn
- Game state must be "playing"
- Player cannot have pending actions

Complex Endpoint #1 - Roll Dice Edge Cases

Route: `POST /api/games/:game_id/turn/roll`

Edge Cases

- **Double bankruptcy**: Player lands on rent they can't afford
- **Pass GO on exact landing**: Still collect $200
- **Network delay**: Use database transactions to prevent duplicate rolls
- **Player disconnect**: Game pauses until reconnection or timeout

Complex Endpoint #1 - Roll Dice Events

Route: `POST /api/games/:game_id/turn/roll`

Socket.io Events Triggered

- **game:state:update**
  - All players (new position, updated board)
- **game:player:options**
  - Current player (buy/skip/pay decision)
- **game:player:balance:update**
  - Current player (if passed GO)

Complex Endpoint #1 - Roll Dice Events

Route: `POST /api/games/:game_id/turn/roll`

Response (202 Accepted)

```
{ "dice": [3, 4], "new_position": 17, "pending_action": { "type": "buy_property",
"property_id": 15 } }
```

Complex Endpoint #2 - Buy Property

Route: `POST /api/games/:game_id/properties/:property_id/buy`

Purpose: Allows current player to purchase an unowned property after landing on it.

Authorization Checks:

- User must be authenticated
- User must be the current player
- Must have a valid `pending_action_id` for this purchase

Request Body:

```
{ "pending_action_id": 42 }
```

Buy Property Validation

Route: `POST /api/games/:game_id/properties/:property_id/buy`

Validation Checks

- Property exists and is on the game board
- Property is not already owned
- Player has sufficient balance (balance >= property_cost)
- Pending action matches this purchase
- Player is on this property's position
- Game state is "playing"

Buy Property Logic

Route: `POST /api/games/:game_id/properties/:property_id/buy`

State Updates

- Deduct property cost from player balance
- Update `property_ownership` table: set owner_id
- Delete pending action from database
- Check if player can continue or turn ends

Buy Property Edge Cases

Route: `POST /api/games/:game_id/properties/:property_id/buy`

Edge Cases

- **Simultaneous purchase attempts**: Use database locks on property row
- **Balance changed before purchase**: Re-validate balance
- **Player goes bankrupt from purchase**: Invalid (validated before)
- **Property was bought by another player**: Return 409 Conflict

Buy Property Events

Route: `POST /api/games/:game_id/properties/:property_id/buy`

Socket.io Events Triggered

- **game:state:update**
  - All players (board shows new owner)
- **game:player:balance:update**
  - Purchasing player (reduced balance)
- **game:player:options**
  - Current player (what's next?)

Response (202 Accepted)

```
{ "success": true, "new_balance": 1300, "property_id": 15 }
```

Socket.io Events Summary

Public Events (Broadcast to all players)

Event	Trigger	Data
game:state:update	Any public state change	Full game state
game:turn:changed	Turn advances to next player	New current_player_id
game:player:joined	Player joins lobby	New player info
game:ended	Game concludes	Winner & final standings

Socket.io Events Summary

Private Events (Sent to specific player)

Event	Trigger	Data
game:player:options	Player must make decision	Available actions
game:player:balance:update	Player balance changes	New balance amount

Key Architecture Principles

1. **Server as Single Source of Truth**

All game logic validated server-side. Never trust the client.

2. **Atomic Operations**

Use database transactions to prevent race conditions.

Key Architecture Principles

3. **Public vs Private State:**

- **Public**: Board, positions, turn order, property ownership
- **Private**: Individual balances, pending actions

4. **Async Event Flow:**

1. HTTP Request
2. Validate
3. Return 202
4. Update DB
5. Emit Events

Questions?