Final Project Submission

Kwame Attrams

Department of Humanities and Social Sciences, Ashesi University

CS_341_A: Web Technologies

David Sampah

December 18, 2025

Link to public url

Link to GitHub repository

Link to YouTube video

Geekerz Application Documentation

1. Project Overview

Geekerz is a web-based platform designed to centralize multiplayer gaming within a persistent environment. Unlike transient browser games, this application maintains state across sessions, allowing users to pause and resume matches asynchronously. The system relies on a three-tier architecture comprising a client-side **Game Manager**, a server-side **Score Submission System**, and a relational **Tournament Structure**.

This documentation details the technical implementation of these components, focusing on data flow and state management.

2. Client-Side State Management: game_manager.js

The GameManager object is the primary interface between the user's browser and the backend infrastructure. It is responsible for initializing game contexts, verifying user identity, and synchronizing the game board with the server.

Initialization and Identity Verification

The entry point for any game session is the GameManager.init(gameSlug, scoringType) function. This sets the internal configuration for the specific game type (e.g., 'win' for Tic-Tac-Toe or 'score' for PacMan).

Crucially, the manager enforces access control through the loadMatchState(callback) function. This function performs an AJAX request to get_match_state.php to retrieve the current match context.

- **Input:** It requires a valid match_id.
- **Output:** It receives a JSON object containing the player1_id, player2_id, status, and the serialized board_state.

Upon receiving this data, the manager compares the PHP session's user_id against the match's player IDs. If the current user matches player1_id, the client unlocks Player 1's controls; otherwise, inputs are disabled. This client-side validation prevents users from manipulating the game state when it is not their turn.

State Synchronization

To support asynchronous play, the application does not rely on real-time sockets but rather on state serialization. When loadMatchState returns the board_state string (a serialized JSON object), the specific game engine (e.g., GameWorld.js) parses this string to reconstruct the visual scene—placing balls on the pool table or characters on the grid exactly as they were left.

3. Server-Side Persistence: submit_score.php

The submit_score.php script handles data persistence. It functions as an API endpoint that accepts raw JSON payloads via POST requests.

Data Payload Structure

The script expects a JSON object with the following schema:

```
{
  "game": "8ball",
  "type": "turn_update", // or "win"
  "match_id": 101,
  "score": 0,
  "board_state": "[...serialized arrays...]"
}
```

Processing Logic

The script processes requests based on the type parameter:

1. **Type: turn_update**
    - This indicates an ongoing match. The script executes an SQL UPDATE query on the matches table, overwriting the board_state column with the new serialized data.
    - **State Toggling:** It automatically toggles the match status field. If the current status implies it was Player 1's turn, the system updates it to allow Player 2 to move next. This ensures strict turn alternation.
2. **Type: win**

- This indicates match conclusion. The script marks the match status as 'completed' and sets the winner_id.
- **Leaderboard Aggregation:** Simultaneously, it triggers an INSERT ... ON DUPLICATE KEY UPDATE query on the leaderboard table. This increments the highscore (win count) for the winning user, ensuring real-time ranking updates without requiring a separate calculation process.

4. Tournament Architecture

The tournament system is implemented through a relational database structure rather than a separate codebase. It organizes individual matches into a hierarchical progression system.

Database Schema Relationships

The system uses a one-to-many relationship between the tournaments and matches tables:

- **tournaments Table:** Defines the event entity (ID, Name, Creator).
- **matches Table:** Contains a tournament_id foreign key. This allows the application to query all matches associated with a specific event.

Round Progression Logic

Tournaments follow a bracket system managed via the round column in the matches table.

- **Initialization:** The system generates initial pairings labeled as 'Round 1'.
- **Progression:** As submit_score.php marks Round 1 matches as 'completed', the system identifies the winners. These user IDs are then programmatically seeded into new match entries labeled 'Round 2'.

This data-driven approach allows the platform to scale from simple 1v1 challenges to complex elimination brackets using the same underlying match execution logic described above.