CSE 4540 - Web Programming (Group B)

January 2, 2026

You are required to work with the code provided in the classroom and make the necessary modifications. At first download the file, unzip and open it in the VS code. In the IDE terminal, run *npm install*, it'll install all the necessary files attached to the code. Modify the database URI with your ID and password. Finally, you can simply type *npm run dev* in the terminal to run your code.

For each schema, insert at least **five** sample input records. After completing the implementation, export all Postman API collections as a JSON file. Organize all required project files (along with the exported json file) into a single folder, excluding the `node_modules` directory. Create a zip folder with the last 3 digits of your ID and submit it in the classroom.

1 Introduction

This document outlines the extension of the current Gaming CRUD application into a comprehensive Game Management System. The system introduces three intercon- nected models: Player, Developer, and Game.

2 Model Specifications & Basic CRUD

2.1 Player Model (playerModel.js)

Represents a game player/user account.

Fields:

- name (String, required)
- email (String, required, unique)
- age (Number, min: 12, max: 100)
- membershipLevel (String, enum: ['free', 'premium', 'elite'], default: 'free')
- joinDate (Date, default: Date.now)

- active (Boolean, default: true)

CRUD Operations:

- Create new player
- Get all players (with optional filters: membership level, active status)
- Get single player by ID
- Update player (change membership, deactivate account)
- Delete player

2.2 Developer Model (developerModel.js)

Represents a game developer.

Fields:

- name (String, required)
- email (String, required, unique)
- specializations (Array of Strings, enum: [‘RPG’, ‘FPS’, ‘Puzzle’, ‘Strategy’, ‘Simulation’])
- experienceYears (Number, min: 1)
- hourlyRate (Number, required, min: 10)
- available (Boolean, default: true)
- certifications (Array of Strings)

CRUD Operations:

- Create new developer
- Get all developers (with optional filters: specialization, availability)
- Get single developer by ID
- Update developer (change hourly rate, availability, add certifications)
- Delete developer

2.3 Game Model (gameModel.js) [Provided]

The Game model is already implemented in the base project and should be used as-is.

Fields:

- title (String, required)
- genre (String, required, enum: ['RPG', 'FPS', 'Puzzle', 'Strategy', 'Simulation'])
- rating (Number, min: 0, max: 10)
- multiplayer (Boolean, default: false)

CRUD Operations (for reference only):

- Create new game
- Get all games (with optional filters: genre, rating)
- Get single game by ID
- Update game details
- Delete game

3 Complex Functionalities

You can add/modify the schema if needed.

3.1 Advanced Player-Developer Collaboration System

Implement a feature where players can request custom game modifications or content from developers. You can create multiple API endpoints to complete the feature.

Process Flow:

- Player submits a customization request for a specific game
- System validates player membership (premium/elite required)
- Developer can view, accept, or reject requests
- If accepted, system creates a collaboration record with:
  - Player ID
  - Developer ID
  - Game ID
  - Request description

- Automatic cost calculation based on developer's hourly rate and estimated hours
- Status: 'accepted', 'in-progress', 'completed', or 'cancelled'
- Timeline and milestones