**Milestone 2 Report: GameScape – A Personalized Platform for Video Game Discovery and Journaling**

**Team Members:**

Xinchen Luo (a1932592)

Bowen Zhu (a1909116)

Jie Li (a1939094)

Xiyuan Zhang (a1932152)

**1. Project Overview**

GameScape is a web application designed to assist users in discovering new games and organizing their personal gaming experiences. It provides a centralized platform for game information and personal inventory management, allowing users to explore, rate, collect, and track gameplay status, with future plans for personal reviews and notes.

Following Milestone 1, we refined our focus based on the RAWG API's comprehensive data for PC and console platforms (Steam, Xbox, PlayStation, Switch, PC). Consequently, we prioritized these platforms and tailored our target audience to mid-to-heavy gamers, initially excluding mobile gaming due to data limitations and platform constraints.

For Milestone 2, we successfully developed the core frontend and backend systems, integrated the RAWG API with our database, and completed three key functional modules: User Login and Registration, Game Search Functionality, and User Game Collection management.

**2. Progress Summary**

2.1 Implemented Features and Current Status

• User Authentication & Session Management: Successfully implemented user registration and login using Node.js/Express.js, password hashing (bcryptjs), and persistent storage in an SQLite database via Sequelize. JWTs are issued at login and persisted in localStorage for session management. Frontend HTML pages (Register.html, Login.html) provide the space for these functionalities.

• RAWG API Integration: Backend proxy endpoints (/api/games/search, /api/games/:id) are created, retrieving game search results and detailed game data from the RAWG API. The frontend makes use of these endpoints for dynamic content display.

• Database and Collection Management: A database is set up with Sequelize ORM using SQLite. User and CollectedGame models—game ID, title, image, rating, and user link—are established and synced. Authorized users can add games to their personalized collections and see their collections with the data in the database.

• Core Frontend Functionality: The main search page (search-page.html) facilitates searching of games, shows search results, displays game detail views, and handles add/view collections. It also loads the featured games on page load.

2.2 Obstacles Encountered & Solutions:

• Initial Setup & Connectivity: Resolved CORS issues between backend and frontend using the cors middleware. Rectified and corrected frontend fetch API call URLs to communicate properly with backend endpoints.

• Database Persistence: Resolved an issue in which collection data was lost upon server restart by refining the Sequelize sync strategy from { alter: true } to { alter: false } in development after schema initial setup, so the data is persisted.

• Development Workflow: Gained practical experience in managing async/await for API calls and database queries and understood the importance of secure API key management.

**3. Technical Details**

3.1 Code and Architecture

The GameScape application follows a client-server architecture:

• Frontend (Client-Side):

* + Built with HTML, CSS, and vanilla JavaScript.
  + Handles user interaction, form submissions (registration, login, search), and dynamic display of game data.
  + Makes asynchronous API calls (fetch) to the backend to retrieve game information and manage user-specific data.
  + Key files: search-page.html, login.html, register.html, and their associated JavaScript (assets/js/search.js) and CSS (assets/css/).

• Backend (Server-Side):

* + Developed using Node.js and the Express.js framework.
  + Serves as an API provider for the frontend.
  + Handles business logic: user authentication (registration, login with JWT), proxying requests to the external RAWG API, and managing user game collections.
  + Interacts with the database via the Sequelize ORM.
  + Key file: server.js (currently monolithic, with plans to potentially refactor into routes/ for better organization if complexity increases significantly).

• Database:

* + SQLite is used as the relational database for persistent storage.
  + The database file is gamescape\_database.sqlite located in the Backend directory.
  + SQL schema definitions also serve as a useful reference for the overall relational design."

• Key Technologies:

* + HTML5, CSS3, JavaScript (ES6+)
  + Node.js, Express.js
  + Sequelize (ORM for Node.js)
  + SQLite3
  + SQL
  + axios (for backend HTTP requests to RAWG API)
  + bcryptjs (for password hashing)
  + jsonwebtoken (for JWT creation and verification)
  + cors (for handling Cross-Origin Resource Sharing)
  + dotenv (available for managing environment variables, though API keys are currently hardcoded for simplicity in this development phase).

3.2 API Integration (RAWG Video Games Database API)

• Status: Successfully integrated.

• Implementation: The backend server.js includes two primary proxy endpoints:

* + GET /api/games/search?query=<term>: Takes a search term, calls the RAWG API's game search endpoint, and returns the results (list of games with basic info like name, ID, background image, rating).
  + GET /api/games/:id: Takes a RAWG game ID, calls the RAWG API's specific game details endpoint, and returns comprehensive details for that game (description, platforms, genres, developers, publishers, website, etc.).

• Information Transmission: These backend APIs are called by the frontend module, search-app.js. The backend then actually requests RAWG using a server-side API key and relays its response back to the frontend.

• Challenges Encountered: The primary challenges experienced were making sure that the API key was properly inserted into backend requests to RAWG, specifically regarding the game details endpoint after its addition. The issue was fixed by ensuring that the apiKey variable was properly declared and passed as a part of the axios parameters for all applicable routes.

3.3 Database Implementation

• Progress: Database design and implementation for core features are complete and functional.

• Technology: SQLite with Sequelize ORM.

• Schema:

A screenshot of a computer

AI-generated content may be incorrect.

• Data Storage:

* User registration data (hashed passwords) is stored in the “users” table.
* When a user adds a game to their collection, an entry is created in collected\_games linking the rawgGameId and other game info to the userId.

**4. Team Contributions**

The team has worked collaboratively, leveraging individual strengths to progress on GameScape.

4.1 Current Roles and Responsibilities

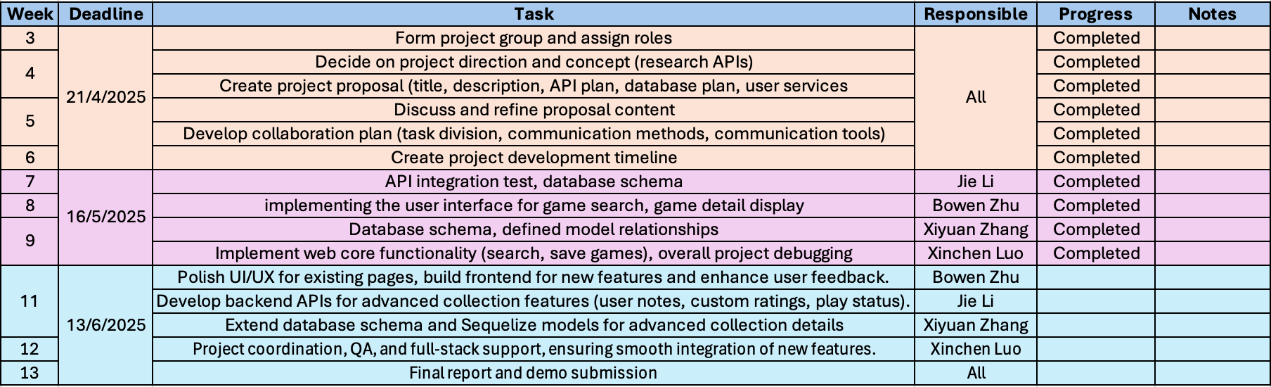
• Bowen Zhu (Frontend Developer): Responsible for designing and implementing the user interface for game search, game detail display, and the user dashboard/collection views. This includes HTML structure, CSS styling, and client-side JavaScript for dynamic content and user interaction. Creation and updating of the project timeline, editing of the project report.

• Jie Li (Backend Developer): Focused on the server-side logic, including the integration of the external RAWG API (search and game details), and developing the API endpoints for user game data management (adding to, viewing, and removing from collections). Editing the Project report Schema diagram.

• Xiyuan Zhang (Database Engineer): Led the database schema design using Sequelize, defined model relationships (User, CollectedGame), and implemented the backend logic for user authentication (registration with password hashing, login with JWT generation and verification). Editing of the project report.

• Xinchen Luo (Full Stack Developer): Oversees project coordination, facilitates weekly check-ins, manages task distribution, conducts quality assurance testing on implemented features, and contributes to full-stack development tasks as needed, ensuring overall project coherence, debugging, and is responsible for the current progress video demonstration. Editing of the project report.

**5. Development Timeline Update**

****

Overall, our project has progressed steadily, aligning with the initial timeline. Key tasks such as API integration (Week 7), game interface development (Week 8), and core backend functionalities for game search and saving (Week 10) were completed as scheduled.

To enhance modularization, allow for more thorough testing, and balance workload, some frontend refinements (navigation bar, footer) and the development of advanced backend features (user notes, personal ratings, play status) were strategically rescheduled from Week 10 to Weeks 11–12.

To maintain this trajectory and address any issues promptly, we utilize daily WeChat standups, a shared Trello task board, and weekly Thursday Zoom sync-ups. We are confident in completing all planned development tasks with the anticipated quality and scope.

**6. Video Demonstration of Progress**

<https://www.youtube.com/watch?v=kvDqvpcdaro>

**7. Risks and Mitigation**

7.1 Risk 1: Complexity of advanced functions

• Problem description: To apply high-end features needs coordination among frontend, backend, and database levels, with threats of excessive workload and complicated debugging.

• Mitigation measures:

* Prioritize essential logic: Begin with fundamental functions (e.g., authentication of users, game searching, favorites addition), then build on (e.g., labels, notes).
* Use timeboxing: If the task takes more than 2 days, reduce scope or employ a fallback
* Use existing UI kits: Leverage tools like Material UI to accelerate development.

7.2 Risk 2: Team coordination and time management

• Problem statement: Ineffective task delegation or incompatible schedules could reduce overall efficiency.

• Mitigation measures:

* Use Trello to divide and assign specific subtasks with deadlines.
* WeChat daily updates: yesterday's progress, today's goals, blockers.
* Hold weekly Zoom code reviews to match logic and style.
* Use Git feature branches and PRs to prevent merge conflicts.

7.3 Risk 3: Insufficient time for UI/UX improvement

• Problem description: Well-polished UI/UX requires a lot of time and iteration.

• Mitigation measures:

* Finalize core Motiff mockups early to reduce redesign.
* Use component-based libraries (shadcn/ui, TailwindCSS) for faster builds.
* Ship UI in stages: essentials first, embellishments afterward.
* Use standardized design specifications (colors, spacing, fonts) throughout.

**8. Feedback and Questions**

• Does our current feature set (auth, API use, DB collections with CRUD & ratings) meet Milestone 2's core database and API requirements?

• For enhancing the 'game journal' aspect of GameScape, what specific features do you believe would add the most significant value to the user experience at this stage?

• What 1-2 key UI/UX areas in search-page.html (search, details, collection views) needing immediate improvement?

• Considering our current stack (Node.js/Express, Sequelize/SQLite, vanilla JavaScript frontend), what are one or two common architectural patterns or backend/frontend best practices we should prioritize learning or implementing next to improve the application's robustness or prepare for more complex features?