Complete Updated VGB Database Design Based on Instructor Feedback (Final Version)

Key Changes:

1. **PLATFORM entity removed** → becomes a simple attribute in GAME
2. **GAME_PLATFORM associative entity removed**
3. **USER_Type composite attribute** → becomes simple attributes (Guest, Registered, Administrator)
4. **Status composite attribute in GAME** → becomes simple attributes (Upcoming, Released)
5. **COMMENT entity replaced with REVIEW entity** (includes Text, Rating, Date_Time_Posted)
6. **Administrator role** → focused solely on content management (no Guest/Registered operations)
7. **Search and filtering tools** → advanced multi-criteria filtering
8. **Specs attribute removed from GAME** → replaced with **Genre attribute**
9. **Tech stack updated** → Using actual implementation stack (HTML5, CSS3, Vanilla JS, Node.js, Express.js, Firebase)
10. **Image attribute added to GAME** → for game cover/thumbnail images

---

📘 UPDATED CONCEPTUAL DATA MODEL (CDM)

Entities and Attributes

Entity	Attributes (Underline = PK)	Description
USER	User_ID, Username, Password, Email, Guest, Registered, Administrator, Registration_Date	Represents the users of the system (Guest, Registered, or Administrator) with their authentication and role information

GAME	Game_ID, Title, Description, Release_Date, Platform, Genre, Image, Upcoming, Released	Represents video games in the database with complete information including release details, platforms, genres, and cover images that users can browse, search, filter, view, review, and favorite
REVIEW	Review_ID, Text, Rating, Date_Time_Posted	Represents reviews and ratings made by registered users on specific games
FAVORITE	User_ID, Game_ID, Date_Added	Represents the many-to-many relationship between users and their favorited games; associative entity resolving M:N between USER and GAME

Relationships, Cardinality, and Participation

Relationship	Type	Cardinality	Participation	Description
USER – REVIEW (Posts)	1:M	USER (0..M) → REVIEW (1..1)	USER optional, REVIEW mandatory	A user may post zero or many reviews; each review is made by exactly one user
GAME – REVIEW (Has)	1:M	GAME (0..M) → REVIEW (1..1)	GAME optional, REVIEW mandatory	A game may have zero or many reviews; each review refers to exactly one game
USER – GAME (Favorites)	M:N	Resolved through FAVORITE	Both optional	Represents users marking games as favorites

---

📗 UPDATED LOGICAL DATA MODEL (LDM)

USER Table

Field	Data Type	Key	Description
User_ID	INT	PK	Unique identifier for each user
Username	VARCHAR(50)		User's chosen display name
Password	VARCHAR(100)		User's hashed password
Email	VARCHAR(100)		User's email address

Guest	BOOLEAN	Indicates if user has guest privileges
Registered	BOOLEAN	Indicates if user is a registered user
Administrator	BOOLEAN	Indicates if user has admin privileges
Registration_Date	DATE	Date the user registered

GAME Table

Field	Data Type	Key	Description
Game_ID	INT	PK	Unique identifier for each game
Title	VARCHAR(100)		Title of the game
Description	TEXT		Description of the game
Release_Date	DATE		Date of release
Platform	VARCHAR(100)		Gaming platform(s) (e.g., PC, PS5, Xbox)
Genre	VARCHAR(100)		Game genre(s) (e.g., Action, RPG, Strategy, Horror)
Image	VARCHAR(255)		URL or path to game cover/thumbnail image
Upcoming	BOOLEAN		Indicates if game is upcoming
Released	BOOLEAN		Indicates if game is released

REVIEW Table

Field	Data Type	Key	Description
Review_ID	INT	PK	Unique identifier for each review
User_ID	INT	FK → USER(User_ID)	User who posted the review
Game_ID	INT	FK → GAME(Game_ID)	Game the review refers to

Text	TEXT	Review content/comment (can be NULL for rating-only reviews)
Rating	INT	Star rating (1-5 scale, can be NULL for comment-only reviews)
Date_Time_Posted	DATETIME	Date and time review was posted

Constraints:

- Rating must be between 1 and 5 (or NULL)
- At least one of Text or Rating must be NOT NULL (user must provide either text, rating, or both)

FAVORITE Table

Field	Data Type	Key	Description
User_ID	INT	PK, FK → USER(User_ID)	User who marked the game as favorite
Game_ID	INT	PK, FK → GAME(Game_ID)	Game marked as favorite
Date_Added	DATE		Date when the favorite was added

📙 UPDATED DATA DICTIONARY

Table 1: Data Dictionary of Video Game Bulletin (VGB)

Table Name	Attribute Name	Data Type	Key Type	Description / Purpose
FAVORITE	Date_Added	DATE		Date the game was added to favorites
FAVORITE	Game_ID	INT	PK, FK	Game that was favorited
FAVORITE	User_ID	INT	PK, FK	User who favorited the game

GAME	Description	TEXT		Description of the game
GAME	Game_ID	INT	PK	Unique identifier for each game
GAME	Genre	VARCHAR(100)		Game genre(s) such as Action, RPG, Strategy, Horror, Racing
GAME	Image	VARCHAR(255)		URL or path to game cover/thumbnail image
GAME	Platform	VARCHAR(100)		Gaming platform(s) for the game
GAME	Release_Date	DATE		Game release date
GAME	Released	BOOLEAN		Indicates if game is released
GAME	Title	VARCHAR(100)		Game title
GAME	Upcoming	BOOLEAN		Indicates if game is upcoming
REVIEW	Date_Time_Posted	DATETIME		Date and time the review was posted
REVIEW	Game_ID	INT	FK	Refers to the game being reviewed
REVIEW	Rating	INT		Star rating from 1 to 5 (can be NULL)
REVIEW	Review_ID	INT	PK	Unique identifier for each review
REVIEW	Text	TEXT		The review content/comment (can be NULL)
REVIEW	User_ID	INT	FK	Refers to the user who posted the review
USER	Administrator	BOOLEAN		Indicates if user has admin privileges
USER	Email	VARCHAR(100)		User's email address
USER	Guest	BOOLEAN		Indicates if user has guest privileges
USER	Password	VARCHAR(100)		User's hashed password

USER	Registered	BOOLEAN		Indicates if user is a registered user
USER	Registration_Date	DATE		Date the user registered
USER	User_ID	INT	PK	Unique identifier for each user
USER	Username	VARCHAR(50)		User's login or display name

PART I: SYSTEM SELECTION AND DEFINITION

1.1 Background of the Organization

Description of the Organization: The Video Game Bulletin (VGB) is a hypothetical organization addressing the challenge game enthusiasts face in tracking new releases amidst scattered and overwhelming digital information. The platform aims to be a streamlined hub for accurate and relevant release information with community-driven reviews and ratings.

Logo: [As shown in VGB_groupactivity1]

Mission Statement: To provide a centralized, interactive, and easy-to-use platform for game enthusiasts to stay updated on all upcoming and newly released games while fostering a community through shared reviews and ratings.

Vision Statement: To become the go-to resource for accurate and timely video game release information, fostering a community of informed and connected gamers who share their experiences through reviews and ratings.

Nature of Business: The VGB operates as a web-based information service, providing a searchable database of video game releases and related news with integrated community review and rating system.

Products and Services: The main product is a web application featuring a release calendar, search and filtering tools, and detailed game pages with cover images, community reviews and ratings. Services include real-time database updates, user account creation, and interactive features like reviewing, rating, and favoriting games.

Functional Areas of the Organization:

Departments:

1. **Information Technology (IT) and Development:** Manages the creation, maintenance, and updates of the web platform and its database

2. **Content Management:** Responsible for sourcing, verifying, and inputting accurate data on game releases including game cover images
3. **User Support/Community Management:** Handles user inquiries and moderates community interactions like reviews and ratings

Key Positions/People:

- **Project Lead:** Responsible for overall project direction, coordination, and ensuring objectives are met
- **Front-End Developer:** Builds the user-facing side of the application, including UI components, image display, review/rating interface, and user experience
- **Back-End Developer:** Manages back-end development, including database setup, data models, image storage/retrieval, and review/rating logic
- **Quality Assurance (QA) Tester:** Tests the application to identify bugs and refine UI/UX, including review/rating functionality and image display
- **Content Manager:** Gathers and verifies game information including cover images to ensure database accuracy; monitors reviews for inappropriate content

Figure 1: Organizational Chart of Video Game Bulletin [Insert the organizational chart from your document in a box]

1.2 PROPOSED DATABASE APPLICATION

Brief Description

The Video Game Bulletin (VGB) database application is a web-based platform designed to centralize video game release information. The system uses Firebase (Realtime Database and Firestore) to store and manage data about games (including cover images), users, reviews with ratings, and user favorites. The database supports real-time updates, allowing users to access the most current information about game releases, platforms, genres, visual content, and community reviews. The application features advanced search and filtering capabilities, enabling users to find games by multiple criteria including title, platform, release date, status, genre, and user ratings.

Scope and Boundaries

The application's scope includes a release calendar with game images, search and filtering tools, game detail pages with cover images, reviews and ratings, and user interaction functionalities. Key features include account creation, favoriting games, and a comprehensive review and rating system. The database's boundaries are defined by the data it stores and manages, which includes information on games (with integrated platform, genre, and image data), release dates, user profiles, reviews with ratings, and favorites.

Key Features:

- **Visual Game Display:** Game cover images displayed throughout the platform (calendar, search results, detail pages)
- **Advanced Search and Filtering Tools:** Users can search by title or keywords and apply multiple filters simultaneously
- **Filter by Platform:** Narrow results to specific gaming platforms (PC, PS5, Xbox Series X, Nintendo Switch, etc.)
- **Filter by Genre:** Find games by genre categories (Action, RPG, Strategy, Horror, Racing, Adventure, etc.)
- **Filter by Status:** View only upcoming or already released games
- **Filter by Release Date:** Find games within specific date ranges
- **Filter by User Rating:** View games sorted by average user rating
- **Multi-Criteria Filtering:** Combine multiple filters for precise game discovery (e.g., PS5 + Action genre + upcoming + highly rated)
- **Review and Rating System:** Users can write text reviews, provide star ratings (1-5), or both

Major Application Areas and User Groups

Major Application Areas

The system's main functionalities are divided into three areas:

1. **User-Facing Information and Discovery:** This includes the display of a real-time release calendar with game images, an advanced search and filtering system for finding games by multiple criteria (title, keywords, platform, genre, status, release date, rating), and a detailed view for individual game information with cover images, reviews and ratings.

2. **Interactive User Features:** This covers user account creation, login, favoriting games, and the review and rating system.

3. **Administrative Content Management:** This area includes specialized tools exclusively for managing, updating, and monitoring the game information database (adding, updating, and removing game information with platform, genre, and image data), and moderating reviews by viewing and removing user reviews.

User Groups

Guest Users (General Public): Individuals who browse the site to check releases for games, search and filter games by various criteria (platform, genre, status, date range, rating), view game details (platform, genre, description, cover image), and **read reviews and ratings** without needing an account. They can also create an account to access interactive features.

Registered Users: Individuals with an account who can perform all actions available to Guest Users, plus log in, favorite games, and **write reviews and provide ratings** on games.

Administrator/Content Manager: Specialized role exclusively responsible for **administrative content management**. This includes monitoring databases, ensuring the accuracy of game information by adding, updating, and removing game information (including platform, genre, and image data), and **moderating reviews** by viewing all registered users' reviews and removing inappropriate, spam, or any user reviews as needed. Administrators do **not** perform Guest or Registered user actions as their role is purely administrative.

Business Processes and Use Cases

Business Processes

1. **Content Acquisition:** Gathering and verifying raw information on new games including platform availability, genre classifications, and cover images from various sources. This is the initial input process that precedes all administrative data management tasks.

2. **Data Management:** This represents the core administrative task of maintaining the database. It includes the continuous entering, updating, removing, and monitoring of all game details (including platform, genre, and image information) to ensure accuracy and consistency.

3. **User Interaction:** Handling all user-driven behaviors in the system. This covers the essential behaviors of all user roles: Guest Users (who can check releases with game images, search and filter games by multiple criteria including genre, view detailed information with cover images, read reviews, and create an account); Registered Users (who perform all Guest actions, plus log in, favorite games, write reviews, and provide ratings); and Administrators (who focus exclusively on managing the database content by adding, updating, removing, and monitoring game information including images, as well as viewing and moderating all user reviews).

Use Cases

1. **Search for a game:** A user enters a game title or keyword to find its release information. (Applies to Guest and Registered)

2. **Filter games by platform:** A user selects specific platform(s) to view only games available on those platforms. (Applies to Guest and Registered)

3. **Filter games by genre:** A user selects specific genre(s) to view games within those categories (e.g., Action, RPG, Strategy, Horror). (Applies to Guest and Registered)

4. **Filter games by status:** A user filters to view only upcoming games or only released games. (Applies to Guest and Registered)

5. **Filter games by release date:** A user specifies a date range to find games releasing within that period. (Applies to Guest and Registered)

6. **Filter games by rating:** A user filters to view games with a minimum average rating (e.g., 4+ stars). (Applies to Guest and Registered)

7. **Apply multi-criteria filters:** A user combines multiple filters (e.g., PS5 platform + Action genre + upcoming status + Q1 2025 release date + 4+ rating) for precise results. (Applies to Guest and Registered)

8. **Check releases for games:** A user views the main calendar or list of upcoming and newly released games with cover images. (Applies to Guest and Registered)

9. **View game details:** A user clicks on a game to see its cover image, description, platform information, genre, release date, average rating, and reviews. (Applies to Guest and Registered)

10. **Read reviews:** A user reads text reviews and ratings from other users on a game's detail page. (Applies to Guest and Registered)

11. **Creating an account:** A user registers for an account to gain access to interactive features. (Applies to Guest)

12. **Favoriting games:** A registered user adds a game to their personal list for future reference. (Applies to Registered only)

13. **Write review:** A registered user posts a text review on a game to share their experience. (Applies to Registered only)

14. **Provide rating:** A registered user gives a star rating (1-5) on a game. (Applies to Registered only)

15. **Edit review:** A registered user modifies their previously posted review or rating. (Applies to Registered only)

16. **Delete review:** A registered user removes their previously posted review. (Applies to Registered only)

17. **Add game info:** An administrator adds new game information to the database, including platform, genre, and cover image. (Applies only to Administrator)

18. **Update game info:** An administrator modifies existing game information in the database, including platform availability, genre, and image. (Applies only to Administrator)

19. **Remove game info:** An administrator deletes game information from the database. (Applies only to Administrator)

20. **Monitor databases:** An administrator monitors the database for status and consistency. (Applies only to Administrator)

21. **View all user reviews:** An administrator views all reviews posted by registered users across the platform. (Applies only to Administrator)

22. **Moderate reviews:** An administrator removes inappropriate, spam, or any registered user's reviews from the platform. (Applies only to Administrator)

Functional and Non-Functional Requirements

Functional Requirements

- The system must display a real-time release calendar with game cover images
- Guest Users must be able to:
    - Check releases for games (displaying a real-time release calendar with images)
    - Perform advanced search and filtering (by title, keywords, platform, genre, status, release date, and rating)
    - Apply single or multiple filter criteria simultaneously
    - View detailed game information (description, platform, genre, release date, cover image)
    - Read reviews and ratings from other users
    - Create an account
- The platform must provide detailed game information, including description, platform(s), genre(s), cover image, release date, and average rating
- The system must calculate and display average ratings for each game based on user reviews
- The system must store and display game cover images
- Registered Users must be able to:
    - Perform all actions available to Guest Users
    - Log in to their account
    - Favorite games (add to personal list)
    - Write text reviews on games
    - Provide star ratings (1-5) on games
    - Edit or delete their own reviews and ratings
- Administrators/Content Managers must be able to:

    - Manage the database by adding, updating, and removing game information (including platform, genre, and image data)
    - Utilize tools to monitor the database for consistency
    - View all reviews posted by registered users
    - Moderate user reviews by removing inappropriate, spam, or any user reviews
- Administrators do **NOT** perform Guest or Registered user actions (purely administrative role)

Non-Functional Requirements

- **Data Consistency:** The database must be updated in real time to ensure the accuracy of release information, images, and average ratings
- **Scalability:** The system should be able to handle an increasing number of users, games, images, and reviews without performance degradation
- **User-Friendliness:** The platform should have a simple and organized interface with intuitive search and filtering controls, including genre selection and visual game displays
- **Performance:** The application should load quickly and efficiently, with filter results and images displaying in real-time
- **Image Loading:** Game images should be optimized for fast loading and display
- **Filter Responsiveness:** Multi-criteria filtering (including genre and rating filters) should return results within 2 seconds
- **Search Accuracy:** Search and filter operations should return relevant and accurate results based on genre, platform, and other criteria
- **Rating Calculation:** Average ratings should be recalculated immediately when new reviews are added or existing reviews are modified or deleted
- **Review Moderation:** All user reviews should be viewable and removable by administrators

Preliminary Methodology

The VGB project follows an iterative development approach using a full-stack JavaScript architecture. The project is broadly divided into four key phases: Planning & Design, Core Development, Feature Implementation & Integration, and Testing & Deployment.

Technology Stack

Frontend:

- **HTML5** - Semantic markup and page structure
- **CSS3** - Styling and responsive design (style.css) for image display and layout
- **Vanilla JavaScript (ES6+)** - Client-side logic, interactivity, and image handling
- **ES6 Modules** - Modern JavaScript module system for code organization

Backend:

- **Node.js** - Server-side runtime environment
- **Express.js v4.18.2** - Web application framework for RESTful API
- **CORS** - Cross-Origin Resource Sharing middleware
- **Body-Parser** - Request body parsing middleware
- **Axios v1.4.0** - HTTP client for API requests

Database & Authentication:

- **Firebase Realtime Database** - Real-time data storage for games (including image URLs) and user data
- **Firebase Firestore** - Document database for reviews and complex queries
- **Firebase Storage** - Cloud storage for game cover images
- **Firebase Authentication** - User authentication and authorization
- **Firebase Admin SDK v11.10.0** - Backend Firebase operations
- **Firebase Client SDK v12.6.0** - Frontend Firebase integration

Development Tools:

- **npm** - Package manager for dependency management
- **Visual Studio Code** - Recommended IDE for development

API Architecture:

- **RESTful API** design pattern
- **HTTP Methods:** GET, POST, PUT, DELETE
- **JSON** data format for client-server communication

Key Steps

1. **Planning & Design (September 2025):** This phase involves project planning, requirement analysis, and finalizing the project scope. UI/UX design, including wireframes and mockups for the search and filtering interface (with genre and rating filters) and image display layouts, will also be completed. Database schema design for Firebase Realtime Database, Firestore collections, and Firebase Storage organization will be finalized.

2. **Core Development (October 2025):**

    - **Frontend:** Set up HTML5 structure, CSS3 styling for image galleries and cards, and core UI components including search bar, filter controls (platform, genre, status, date, rating), image display components, and review/rating interface using Vanilla JavaScript
    - **Backend:** Configure Node.js and Express.js server, implement RESTful API endpoints with CORS and Body-Parser middleware, set up Firebase Admin SDK for server-side operations, configure Firebase Storage for image uploads

    - **Database:** Initialize Firebase Realtime Database for games (with image URLs) and user profiles, set up Firestore for reviews, configure Firebase Authentication, set up Firebase Storage buckets for game images, and define data models and indexing strategy for efficient filtering by genre and rating
3. **Feature Implementation & Integration (November 2025):**
    - Implement core features including release calendar with image thumbnails, search functionality, and multi-criteria filtering system (platform, genre, status, date, rating filters)
    - Develop review and rating system with Firestore integration
    - Integrate Firebase Authentication for user account management
    - Build average rating calculation logic
    - Implement favorite games functionality
    - Develop image upload and display system with Firebase Storage
    - Create admin dashboard for game management (including image uploads), review moderation with ability to view and remove all user reviews
    - Optimize filter performance, database queries, and image loading
    - Implement client-side routing with ES6 modules
4. **Testing & Deployment (December 2025):**
    - Conduct comprehensive testing including unit tests, integration tests, and end-to-end testing
    - Test search and filter accuracy and performance (especially genre and rating filters)
    - Validate review submission, editing, and deletion functionality
    - Test average rating calculation accuracy
    - Test image upload, storage, and retrieval functionality
    - Verify admin moderation features including viewing all reviews and removal capabilities
    - Perform security audits on Firebase Authentication, Storage, and API endpoints
    - Test image loading performance and optimization
    - Refine system based on user feedback
    - Deploy backend to production server
    - Deploy frontend to web hosting service
    - Configure Firebase production database rules and Storage security rules
    - Complete project documentation including API documentation and user guides

Development Dependencies

```
{
  "dependencies": {
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "body-parser": "^1.20.2",
    "firebase": "^12.6.0",
```

```
    "firebase-admin": "^11.10.0",
    "axios": "^1.4.0",
    "multer": "^1.4.5"
  }
}
```

1.3 Objectives of the Proposed Database Application

1. **Centralize Game Information:** Provide a single source of truth for video game release data with comprehensive details including platforms, genres, and visual content
2. **Enable Visual Game Discovery:** Display game cover images throughout the platform to enhance user experience and facilitate quick game identification
3. **Enable Advanced Search and Filtering:** Allow users to find games through multiple filter criteria (platform, genre, release date, status, rating, title)
4. **Foster Community Engagement:** Enable registered users to share experiences through reviews and ratings, creating a collaborative gaming community
5. **Enable User Interaction:** Allow users to create accounts, provide reviews and ratings, and maintain personalized favorite lists
6. **Support Real-Time Updates:** Ensure release calendar, game information, images, and average ratings are always current and accurate
7. **Facilitate Content Management:** Enable administrators to efficiently manage game data (including cover images) and moderate user-generated content
8. **Improve Information Discovery:** Provide robust search and filtering functionality with visual aids for precise game discovery based on multiple criteria

---

1.4 Significance of the Proposed Database Application

Who will benefit from the proposed database application?

1. Game Enthusiasts (Guest Users):

- Access accurate, centralized release information with visual game displays without searching multiple sources
- Quickly identify games through cover images and detailed information
- Make informed decisions based on community reviews and ratings
- Discover games through advanced filtering by platform, genre, status, and ratings
- Browse upcoming and released games with comprehensive visual and textual information

2. Registered Users:

- Personalize their experience through favorites and community engagement via reviews and ratings
- Quickly find games matching specific criteria with visual browsing experience
- Contribute to the gaming community by sharing experiences through text reviews and star ratings
- Build and maintain a personal favorite games collection with images
- Edit or remove their own reviews to keep their feedback current and accurate

3. Content Managers/Administrators:

- Efficiently maintain and update game database with specialized administrative tools including image upload and management
- Upload and manage game cover images to enhance visual presentation
- Moderate user reviews to maintain content quality and community standards
- View all registered users' reviews with game images for comprehensive oversight
- Remove inappropriate, spam, or any user reviews to ensure platform integrity
- Monitor database consistency and health for optimal system performance

4. Gaming Community:

- Foster connection through shared information, reviews, and ratings
- Discover games through sophisticated filtering and visual browsing
- Build trust through transparent user reviews and community-driven ratings
- Access visually organized game information for better decision-making
- Participate in a moderated environment that maintains quality standards

5. The Organization (VGB):

- Establish VGB as a trusted resource in the gaming information space with superior search capabilities and visual presentation
- Build a platform with community-driven content and user engagement
- Maintain content quality through effective administrative review moderation
- Provide a visually appealing and user-friendly experience that differentiates VGB from competitors
- Create a scalable platform that can grow with increasing user base and game catalog

---

1.5 Scope and Limitations of the Proposed Database Application

Scope:

Core Functionality:

- Real-time release calendar display with game cover images
- Advanced search and filtering tools (by title, keywords, platform, genre, status, release date, and rating)
- Multi-criteria filtering for precise game discovery
- Visual game display with cover images throughout the platform (calendar, search results, detail pages)
- Detailed game information pages (description, platform, genre, release date, cover image)

Review and Rating System:

- Review and rating system (text reviews, 1-5 star ratings, or both)
- Average rating calculation for each game
- User review history with game images

User Features:

- User account management (registration, login, profile via Firebase Authentication)
- Interactive features (favoriting games, writing reviews, providing ratings)
- Personal favorite lists with game images and details
- Edit and delete own reviews functionality for registered users

Administrative Features:

- Administrative content management tools (add, update, remove, monitor game data)
- **Image management system** (upload, update, and delete game cover images via Firebase Storage)
- View all registered users' reviews with user and game information including images
- Review moderation capabilities (remove inappropriate, spam, or any user reviews)
- Moderation logging for accountability and tracking
- Database monitoring tools for consistency checks

Filtering and Discovery:

- Filter by platform (PC, PS5, Xbox Series X, Nintendo Switch, etc.)
- Filter by genre (Action, RPG, Strategy, Horror, Racing, Adventure, etc.)
- Filter by status (Upcoming, Released)
- Filter by release date range
- Filter/sort by average user rating
- Visual browsing with game cover images

Technical Features:

- Support for multiple user types (Guest, Registered, Administrator with distinct roles)
- Real-time data synchronization via Firebase
- Cloud-based image storage with Firebase Storage

- RESTful API architecture for data operations

Limitations:

Content Scope:

- Focus on game release information, cover images, and user reviews only (no gameplay videos, trailers, or professional critic reviews from external sources)
- Does not include e-commerce functionality (no game purchases or transactions)
- Does not track user gaming activity, achievements, or playtime statistics
- Community features limited to reviews and ratings (no forums, messaging, or social networking features)

Platform and Access:

- Limited to web-based access (no native mobile applications in initial release)
- No integration with gaming platforms, digital storefronts, or game launchers (Steam, Epic Games, PlayStation Network, Xbox Live)
- Cannot sync with user's gaming libraries or accounts on other platforms

Data Structure:

- Platform information stored as attributes rather than separate entities (simplified model)
- Genre information stored as comma-separated values rather than separate genre entity
- Filtering limited to predefined categories (platform, genre, status, release date, rating, title)

User Limitations:

- Each user's review capability is unlimited per game (or can be limited to one review per user per game - specify based on your implementation)
- Guest users cannot write reviews or favorite games (must register for interactive features)

Administrative Limitations:

- Review moderation is manual (no automatic content filtering or AI-based moderation)
- Administrators have purely administrative role (cannot browse as Guest or log in as regular user)
- Rating scale fixed at 1-5 stars (not customizable)
- No automated game data import from external APIs (manual entry required)

Technical Limitations:

- Image file size limited to 5MB for uploads
- Image formats limited to common types (JPEG, PNG)

- No video content support (only static images)
- No advanced image editing features within the platform
- Search limited to title and description text matching (no semantic or AI-powered search)

Scalability Considerations:

- Performance optimization required for large image catalogs
- Client-side filtering may become slower with very large datasets
- Firebase Storage costs scale with image storage volume

Key Changes Summary:

- **Removed Specs attribute** from GAME entity
- **Added Genre attribute** to GAME entity for game categorization
- **Added Image attribute** to GAME entity for game cover/thumbnail images
- Replaced "general search tools" with "search and filtering tools" throughout
- Added **Filter by Genre** as a key feature and use case
- Updated functional requirements to include genre filtering and image display capabilities
- Enhanced multi-criteria filtering to include genre as a filter option
- **Updated technology stack** to reflect actual implementation (HTML5, CSS3, Vanilla JS, Node.js, Express.js, Firebase) with Firebase Storage for images
- **Updated methodology** to include specific technologies, image handling, and development workflow
- Maintained all review and rating system features
- **Administrator role strictly limited** to: manage database (including images), monitor consistency, view all user reviews, moderate reviews by removing any user reviews (NO Guest/Registered operations)
- **Updated entity descriptions** for clarity and completeness
- **Added Firebase Storage** for cloud-based image management

---

PART II: REQUIREMENTS ANALYSIS AND SPECIFICATIONS

2.1 General Description

The Video Game Bulletin (VGB) is a web-based platform built with HTML5, CSS3, Vanilla JavaScript frontend, and Node.js/Express.js backend with Firebase (Realtime Database, Firestore, and Storage) at its core, designed to solve fragmented information challenges for game enthusiasts. The system serves as a centralized hub for tracking and discovering new game releases with advanced search and filtering capabilities (including genre-based

discovery), visual game displays with cover images, complemented by a community-driven review and rating system.

Major Functions:

1. **Information Display:** Real-time release calendar with game images and detailed game information with cover images and community reviews
2. **Search and Filtering Capability:** Advanced search by title, keywords, and multi-criteria filtering (platform, genre, release date, status, rating)
3. **Genre-Based Discovery:** Filter games by genre categories (Action, RPG, Strategy, Horror, Racing, Adventure, etc.)
4. **Visual Game Display:** Display game cover images throughout the platform for better user experience
5. **Review and Rating System:** Users can write text reviews, provide star ratings, or both; view average ratings
6. **User Management:** Account creation, authentication via Firebase Authentication, and profile management
7. **Interactive Features:** Game favoriting, review writing, and rating system
8. **Content Management:** Administrative tools for database maintenance (including image management), viewing all user reviews, and review moderation

2.2 Basic Operation

Operations Supported by User Type:

Guest Users:

- Check releases for games (view real-time calendar/list with game images)
- Search and filter games (by title, keywords, platform, genre, release date, status, rating)
- View game details (description, platform, genre, release date, cover image)
- Read reviews and ratings from other users
- View average rating for each game
- Create an account

Registered Users (all Guest operations plus):

- Log in to account (via Firebase Authentication)
- Favorite games (add to personal list)
- Write text reviews on games
- Provide star ratings (1-5) on games
- Edit or delete their own reviews

Administrators (administrative operations only):

- Add game information (title, description, platform, genre, release date, status, cover image)

- Update game information (modify existing game data including genre and image)
- Remove game information (delete games from database)
- Monitor database status and consistency
- View all reviews posted by registered users
- Moderate reviews (remove inappropriate, spam, or any user reviews)

2.3 Information Needs

Data Requirements:

1. **USER Data:**
    - User_ID (Primary Key)
    - Username, Password (hashed via Firebase Authentication), Email
    - Guest, Registered, Administrator (boolean flags)
    - Registration_Date
2. **GAME Data:**
    - Game_ID (Primary Key)
    - Title, Description, Release_Date
    - Platform (e.g., PC, PS5, Xbox Series X, Nintendo Switch)
    - Genre (e.g., Action, RPG, Strategy, Horror, Racing, Adventure)
    - Image (URL or path to cover/thumbnail image stored in Firebase Storage)
    - Upcoming, Released (boolean status flags)
3. **REVIEW Data (stored in Firestore):**
    - Review_ID (Primary Key)
    - Text (nullable - for text review content)
    - Rating (nullable - integer 1-5 for star rating)
    - Date_Time_Posted
    - User_ID (Foreign Key), Game_ID (Foreign Key)
    - **Note:** At least one of Text or Rating must be provided
4. **FAVORITE Data:**
    - User_ID (Foreign Key, part of composite PK)
    - Game_ID (Foreign Key, part of composite PK)
    - Date_Added

Transaction Requirements:

1. **Search Games:** Retrieve games matching keyword in Title or Description
2. **Filter Games by Platform:** Retrieve games matching specific platform
3. **Filter Games by Genre:** Retrieve games matching specific genre(s)
4. **Filter Games by Status:** Retrieve games based on Upcoming or Released status
5. **Filter Games by Release Date:** Retrieve games within a specified date range

6. **Filter Games by Rating:** Retrieve games sorted by average user rating
7. **Multi-Criteria Filter:** Retrieve games matching multiple filter conditions simultaneously (platform, genre, status, date, rating)
8. **Check Releases:** Retrieve games ordered by Release_Date with images
9. **View Game Details:** Retrieve single game with all details including genre, image, and reviews
10. **Calculate Average Rating:** Compute average rating for a game from all user ratings
11. **Upload Game Image:** Store game cover image in Firebase Storage and save URL (Admin only)
12. **Create Account:** Insert new USER record via Firebase Authentication
13. **Favoriting Games:** Insert FAVORITE record linking User to Game
14. **Write Review:** Insert REVIEW document in Firestore with text, rating, or both (Registered users only)
15. **Edit Review:** Update existing REVIEW document in Firestore (user can only edit their own reviews)
16. **Delete Review:** Remove REVIEW document from Firestore (user can delete their own review or admin can moderate)
17. **View User's Reviews:** Retrieve all reviews posted by a specific user from Firestore
18. **View Game's Reviews:** Retrieve all reviews for a specific game from Firestore
19. **Add Game Info:** Insert new GAME record in Firebase Realtime Database with genre and image URL (Admin only)
20. **Update Game Info:** Update existing GAME record including genre and image (Admin only)
21. **Remove Game Info:** Delete GAME and related records including image from Storage (Admin only)
22. **Monitor Database:** Retrieve database status information via Firebase Admin SDK (Admin only)
23. **View All User Reviews:** Retrieve all reviews from all registered users (Admin only)
24. **Moderate Review:** Remove any user's REVIEW document from Firestore (Admin only)

Output Requirements:

1. **Release Calendar/List:** Game Title, Cover Image, Status flags, Release_Date, Platform, Genre, Average Rating
2. **Search Results:** List of Game Titles with cover images, description snippets, genre, and average rating
3. **Filtered Results:** List of games with cover images matching selected filter criteria (including genre) with average rating display
4. **Game Detail Page:** Full game information including Cover Image, Title, Description, Platform, Genre, Release_Date, Average Rating, List of Reviews, Favorite status
5. **User's Favorite List:** Games favorited by user with cover images, Release_Date, Platform, Genre, and Average Rating
6. **User's Review History:** List of all reviews posted by the user with game titles, images, and dates

7. **Review Display:** Individual review showing username, text content, star rating, date posted
8. **Average Rating Display:** Calculated average rating (e.g., 4.3/5.0) with total number of ratings
9. **Administrative Dashboard:** Game management interface (including image upload), database status, review moderation queue
10. **All User Reviews List:** Complete list of all reviews from registered users with reviewer information (Admin view)
11. **Filter Options Display:** Available platforms, genres, status options, date range selectors, and rating ranges

---

PART III: CONCEPTUAL DATABASE DESIGN

3.1 User's Requirements Specification

The VGB system must support three primary user groups with distinct requirements:

Guest Users require:

- Ability to browse game releases without authentication
- Visual display of game cover images throughout the platform
- Advanced search and filtering functionality to find specific games by multiple criteria
- Filter by platform to see games for specific gaming systems
- **Filter by genre** to discover games in preferred categories (Action, RPG, Strategy, Horror, etc.)
- Filter by status to view only upcoming or already released games
- Filter by release date range to find games within specific timeframes
- Filter/sort by user rating to find highly-rated games
- Access to detailed game information including genre and cover image
- Read reviews and ratings from the community
- View average ratings for each game
- Option to create an account for enhanced features

Registered Users require:

- All Guest User capabilities
- Secure login mechanism via Firebase Authentication
- Ability to mark games as favorites for quick access
- Write text reviews on games to share experiences
- Provide star ratings (1-5) on games
- Edit or delete their own reviews
- View their review history

- Personal favorite list management with game images

Administrators require:

- **Exclusive focus on administrative content management** (do not perform Guest/Registered operations)
- Tools to add new game entries to database with complete information including genre and cover image
- Image upload functionality to Firebase Storage for game covers
- Functionality to update existing game information (including platform, genre, and image)
- Ability to remove outdated or incorrect game data including associated images
- Dashboard to monitor database health and consistency
- View all reviews posted by registered users across the platform
- Moderate user reviews by removing inappropriate, spam, or any user reviews
- **Cannot browse as Guest, cannot log in as regular user, cannot write reviews or rate games** (purely administrative role)

3.2 ERD

Figure 2: ERD of Video Game Bulletin (VGB)

[Create a new ERD with these specifications:]

Entities (rectangles):

1. **USER** (with attributes: User_ID [underlined], Username, Password, Email, Guest, Registered, Administrator, Registration_Date)
2. **GAME** (with attributes: Game_ID [underlined], Title, Description, Release_Date, Platform, Genre, Image, Upcoming, Released)
3. **REVIEW** (with attributes: Review_ID [underlined], Text, Rating, Date_Time_Posted)
4. **FAVORITE** (with attributes: User_ID [underlined], Game_ID [underlined], Date_Added)

Relationships (diamonds):

1. USER "Posts" REVIEW (1:M, USER optional, REVIEW mandatory)
2. GAME "Has" REVIEW (1:M, GAME optional, REVIEW mandatory)
3. USER "Favorites" GAME (M:N, resolved through FAVORITE entity)

Notation:

- Solid lines for mandatory participation
- Dashed lines for optional participation
- Crow's foot notation for "many" side
- Both User_ID and Game_ID underlined in FAVORITE (composite PK)
- Review_ID underlined in REVIEW (PK)

3.3 Data Dictionary

[See the complete alphabetically ordered data dictionary table above in the "Updated Data Dictionary" section - label it as Table 1: Data Dictionary of Video Game Bulletin (VGB)]

PART IV: LOGICAL DATABASE DESIGN

4.1 ERD to Relational Mapping

Figure 3: ERD to Relational Mapping

[Create a mapping diagram showing:]

USER (User_ID [PK], Username, Password, Email, Guest, Registered, Administrator, Registration_Date) ↓ (1:M Posts) **REVIEW** (Review_ID [PK], User_ID [FK], Game_ID [FK], Text, Rating, Date_Time_Posted) ↑ (M:1 Has) **GAME** (Game_ID [PK], Title, Description, Release_Date, Platform, Genre, Image, Upcoming, Released)

FAVORITE (User_ID [PK, FK], Game_ID [PK, FK], Date_Added)

- Connects USER and GAME (M:N relationship)

Constraints:

- REVIEW.Rating: CHECK (Rating BETWEEN 1 AND 5 OR Rating IS NULL)
- REVIEW: CHECK (Text IS NOT NULL OR Rating IS NOT NULL)
- GAME.Image: VARCHAR(255) for storing Firebase Storage URL

4.2 The Relations

Table 2: USER Relation

User_ID	Username	Password	Email	Guest	Registered	Administrator	Registration_Date
1	gamer_pro	hashed_pw1	gamerpro@email.com	TRUE	TRUE	FALSE	2024-01-15
2	admin_vgb	hashed_pw2	admin@vgb.com	FALSE	FALSE	TRUE	2024-01-10
3	casual_player	hashed_pw3	casual@email.com	TRUE	TRUE	FALSE	2024-02-20

4	fps_master	hashed_pw4	fpsmaster@email.com	TRUE	TRUE	FALSE	2024-03-05
5	rpg_enthusiast	hashed_pw5	rpgfan@email.com	TRUE	TRUE	FALSE	2024-03-12
6	indie_lover	hashed_pw6	indiegames@email.com	TRUE	TRUE	FALSE	2024-04-01
7	esports_fan	hashed_pw7	esports@email.com	TRUE	TRUE	FALSE	2024-04-15
8	retro_gamer	hashed_pw8	retro@email.com	TRUE	TRUE	FALSE	2024-05-01
9	mod_team	hashed_pw9	moderator@vgb.com	FALSE	FALSE	TRUE	2024-01-10
10	game_reviewer	hashed_pw10	reviewer@email.com	TRUE	TRUE	FALSE	2024-05-20

Table 3: GAME Relation

Game_ID	Title	Description	Release_Date	Platform	Genre	Image	Upcoming	Released
1	Cyber Warriors 2077	Futuristic action RPG	2025-06-15	PC, PS5, Xbox Series X	Action, RPG	https://storage.firebase.com/game1.jpg	TRUE	FALSE
2	Fantasy Realm Online	MMO RPG adventure	2024-11-20	PC, PS5	MMO RPG, Fantasy	https://storage.firebase.com/game2.jpg	FALSE	TRUE
3	Speed Racers Ultimate	Racing simulation	2025-03-10	PC, PS5, Xbox Series X, Nintendo	Racing, Simulation	https://storage.firebase.com/game3.jpg	TRUE	FALSE

				Switch				
4	Zombie Survival HD	Horror survival game	2024-10-31	PC, Xbox Series X	Horror, Survival	https://storage.firebase.com/game4.jpg	FALSE	TRUE
5	Strategy Empire IV	Turn-based strategy	2025-01-25	PC	Strategy, Turn-Based	https://storage.firebase.com/game5.jpg	TRUE	FALSE
6	Battle Royale Pro	Multiplayer FPS	2024-08-15	PC, PS5, Xbox Series X	Action, FPS, Multiplayer	https://storage.firebase.com/game6.jpg	FALSE	TRUE
7	Indie Pixel Quest	Retro platformer	2025-02-14	PC, Nintendo Switch	Platformer, Indie	https://storage.firebase.com/game7.jpg	TRUE	FALSE
8	Space Explorer VR	VR adventure	2025-04-20	PC (VR)	Adventure, VR, Sci-Fi	https://storage.firebase.com/game8.jpg	TRUE	FALSE
9	Fighting Champions	Fighting game	2024-12-01	PS5, Xbox Series X	Fighting, Competitive	https://storage.firebase.com/game9.jpg	FALSE	TRUE
10	Mystery Detective	Puzzle adventure	2025-05-30	PC, PS5, Nintendo Switch	Puzzle, Adventure, Mystery	https://storage.firebase.com/game10.jpg	TRUE	FALSE

Table 4: REVIEW Relation

Review_ID	User_ID	Game_ID	Text	Rating	Date_Time_Posted
1	1	1	Can't wait for this release! The gameplay trailer looks incredible.	5	2024-11-15 14:30:00
2	3	2	Best MMORPG I've played! Great story and community.	5	2024-11-21 10:15:00
3	4	6	Graphics are amazing and gameplay is smooth. Highly recommend!	5	2024-08-20 16:45:00
4	5	4	Too scary for me but great game for horror fans. Well made.	4	2024-11-01 20:00:00
5	6	7	Love the retro style. Brings back childhood memories.	5	2024-11-25 12:30:00
6	1	3	Best racing game coming. Can't wait to play multiplayer.	NULL	2024-11-28 09:00:00
7	7	6	Competitive scene is growing. Great for esports!	4	2024-09-10 18:20:00
8	8	9	Reminds me of classic fighters. Character roster is diverse.	4	2024-12-02 15:10:00
9	10	2	NULL	4	2024-11-22 11:00:00
10	5	1	The RPG mechanics look deep and engaging. Day one purchase!	5	2024-11-30 13:45:00

Table 5: FAVORITE Relation

User_ID	Game_ID	Date_Added
1	1	2024-11-15
1	3	2024-11-28

3	2	2024-11-21
4	6	2024-08-20
5	1	2024-11-30
5	4	2024-11-01
6	7	2024-11-25
7	6	2024-09-10
8	9	2024-12-02
10	2	2024-11-22

PART V: NoSQL (FIREBASE IMPLEMENTATION)

5.1 Database

Create a Firebase project named "video-game-bulletin" or "vgb-database"

Firebase Configuration:

- **Firebase Realtime Database:** For games and user data
- **Firebase Firestore:** For reviews (better querying capabilities)
- **Firebase Storage:** For game cover images
- **Firebase Authentication:** For user management
- **Firebase Admin SDK:** For backend operations
- **Firebase Client SDK v12.6.0:** For frontend integration

Figure 4: Firebase Database [Screenshot showing your Firebase project dashboard with Realtime Database, Firestore, and Storage enabled]

5.2 Collections

Firebase will have:

- **Realtime Database nodes:** users, games, favorites
- **Firestore collections:** reviews
- **Firebase Storage buckets:** game-images

Figure 5: Users Collection (Realtime Database) [Screenshot of Users node in Firebase Realtime Database]

Figure 6: Games Collection (Realtime Database) [Screenshot of Games node in Firebase Realtime Database with image URLs]

Figure 7: Reviews Collection (Firestore) [Screenshot of Reviews collection in Firebase Firestore]

Figure 8: Favorites Collection (Realtime Database) [Screenshot of Favorites node in Firebase Realtime Database]

Figure 9: Game Images Storage (Firebase Storage) [Screenshot of game-images bucket in Firebase Storage]

5.3 Documents

Sample document structures:

Figure 10: Sample User Document (Realtime Database)


```
{
  "users": {
    "user1": {
      "userId": "1",
      "username": "gamer_pro",
      "email": "gamerpro@email.com",
      "guest": true,
      "registered": true,
      "administrator": false,
      "registrationDate": "2024-01-15"
    }
  }
}
```


Figure 11: Sample Game Document (Realtime Database)


```
{
  "games": {
    "game1": {
      "gameId": "1",
      "title": "Cyber Warriors 2077",
      "description": "Futuristic action RPG",
      "releaseDate": "2025-06-15",
      "platform": "PC, PS5, Xbox Series X",
      "genre": "Action, RPG",
```

```
      "image":
"https://firebasestorage.googleapis.com/v0/b/vgb-database.appspot.com/o/game-images%2Fga
me1.jpg?alt=media",
      "upcoming": true,
      "released": false,
      "averageRating": 5.0,
      "totalRatings": 2
    }
  }
}
```

Figure 12: Sample Review Document (Firestore)

```
{
  "reviewId": "1",
  "userId": "1",
  "gameId": "1",
  "text": "Can't wait for this release! The gameplay trailer looks incredible.",
  "rating": 5,
  "dateTimePosted": "2024-11-15T14:30:00"
}
```

Figure 13: Sample Favorite Document (Realtime Database)

```
{
  "favorites": {
    "user1_game1": {
      "userId": "1",
      "gameId": "1",
      "dateAdded": "2024-11-15"
    }
  }
}
```

5.4 CRUD Operations

Figure 14: Create Operation (Add Game with Image) - Backend

```
// server.js - Express.js endpoint to add a new game with image (Admin only)
const admin = require('firebase-admin');
const multer = require('multer');
const db = admin.database();
```

```
const bucket = admin.storage().bucket();

// Configure multer for image upload
const upload = multer({
  storage: multer.memoryStorage(),
  limits: {
    fileSize: 5 * 1024 * 1024 // 5MB limit
  }
});

app.post('/api/games', upload.single('image'), async (req, res) => {
  try {
    const { gameId, title, description, releaseDate, platform, genre, upcoming, released } =
req.body;

    let imageUrl = '';

    // Upload image to Firebase Storage if provided
    if (req.file) {
      const blob = bucket.file(`game-images/${gameId}.jpg`);
      const blobStream = blob.createWriteStream({
        metadata: {
          contentType: req.file.mimetype
        }
      });

      await new Promise((resolve, reject) => {
        blobStream.on('error', reject);
        blobStream.on('finish', async () => {
          // Get public URL
          await blob.makePublic();
          imageUrl = `https://storage.googleapis.com/${bucket.name}/${blob.name}`;
          resolve();
        });
        blobStream.end(req.file.buffer);
      });
    }

    const gameRef = db.ref(`games/${gameId}`);
    await gameRef.set({
      gameId,
      title,
      description,
      releaseDate,
```

```
      platform,
      genre,
      image: imageUrl,
      upcoming,
      released,
      averageRating: 0,
      totalRatings: 0
    });

    res.status(201).json({ message: 'Game added successfully', gameId, imageUrl });
  } catch (error) {
    console.error('Error adding game:', error);
    res.status(500).json({ error: 'Failed to add game' });
  }
});
```

Figure 15: Read Operation (Get Game Details with Reviews and Image) - Backend

```
// server.js - Express.js endpoint to get game with reviews
const firestore = admin.firestore();

app.get('/api/games/:gameId', async (req, res) => {
  try {
    const { gameId } = req.params;

    // Get game data from Realtime Database
    const gameSnapshot = await db.ref(`games/${gameId}`).once('value');
    const gameData = gameSnapshot.val();

    if (!gameData) {
      return res.status(404).json({ error: 'Game not found' });
    }

    // Get reviews from Firestore
    const reviewsSnapshot = await firestore.collection('reviews')
      .where('gameId', '==', gameId)
      .orderBy('dateTimePosted', 'desc')
      .get();

    const reviews = [];
    reviewsSnapshot.forEach(doc => {
      reviews.push({ reviewId: doc.id, ...doc.data() });
    });
```

```
    res.json({ game: gameData, reviews });
  } catch (error) {
    console.error('Error getting game:', error);
    res.status(500).json({ error: 'Failed to get game' });
  }
});
```

Figure 16: Update Operation (Update Game Info with Optional New Image) - Backend

```
// server.js - Express.js endpoint to update game (Admin only)
app.put('/api/games/:gameId', upload.single('image'), async (req, res) => {
  try {
    const { gameId } = req.params;
    const { title, description, releaseDate, platform, genre, upcoming, released } = req.body;

    const updateData = {
      title,
      description,
      releaseDate,
      platform,
      genre,
      upcoming,
      released
    };

    // Upload new image if provided
    if (req.file) {
      const blob = bucket.file(`game-images/${gameId}.jpg`);
      const blobStream = blob.createWriteStream({
        metadata: {
          contentType: req.file.mimetype
        }
      });

      await new Promise((resolve, reject) => {
        blobStream.on('error', reject);
        blobStream.on('finish', async () => {
          await blob.makePublic();
          updateData.image = `https://storage.googleapis.com/${bucket.name}/${blob.name}`;
          resolve();
        });
        blobStream.end(req.file.buffer);
      });
    }
```

```
    const gameRef = db.ref(`games/${gameId}`);
    await gameRef.update(updateData);

    res.json({ message: 'Game updated successfully' });
  } catch (error) {
    console.error('Error updating game:', error);
    res.status(500).json({ error: 'Failed to update game' });
  }
});
```

Figure 17: Delete Operation (Remove Game and Image) - Backend

```
// server.js - Express.js endpoint to delete game (Admin only)
app.delete('/api/games/:gameId', async (req, res) => {
  try {
    const { gameId } = req.params;

    // Get game data to retrieve image URL
    const gameSnapshot = await db.ref(`games/${gameId}`).once('value');
    const gameData = gameSnapshot.val();

    // Delete image from Firebase Storage
    if (gameData && gameData.image) {
      try {
        const imageFile = bucket.file(`game-images/${gameId}.jpg`);
        await imageFile.delete();
      } catch (error) {
        console.log('Image not found or already deleted');
      }
    }

    // Delete all reviews for this game from Firestore
    const reviewsSnapshot = await firestore.collection('reviews')
      .where('gameId', '==', gameId)
      .get();

    const batch = firestore.batch();
    reviewsSnapshot.forEach(doc => {
      batch.delete(doc.ref);
    });
    await batch.commit();

    // Delete favorites from Realtime Database
```

```
    const favoritesSnapshot = await db.ref('favorites')
      .orderByChild('gameId')
      .equalTo(gameId)
      .once('value');

    const favoriteUpdates = {};
    favoritesSnapshot.forEach(childSnapshot => {
      favoriteUpdates[childSnapshot.key] = null;
    });
    await db.ref('favorites').update(favoriteUpdates);

    // Delete the game
    await db.ref(`games/${gameId}`).remove();

    res.json({ message: 'Game, image, and related data deleted successfully' });
  } catch (error) {
    console.error('Error deleting game:', error);
    res.status(500).json({ error: 'Failed to delete game' });
  }
});
```

Figure 18: Create Review Operation - Backend

```
// server.js - Express.js endpoint to add review
app.post('/api/reviews', async (req, res) => {
  try {
    const { userId, gameId, text, rating } = req.body;

    // Validate input
    if (!text && !rating) {
      return res.status(400).json({ error: 'Must provide either text, rating, or both' });
    }

    if (rating && (rating < 1 || rating > 5)) {
      return res.status(400).json({ error: 'Rating must be between 1 and 5' });
    }

    // Add review to Firestore
    const reviewRef = await firestore.collection('reviews').add({
      userId,
      gameId,
      text: text || null,
      rating: rating || null,
      dateTimePosted: new Date().toISOString()
```

```
    });

    // Update game's average rating if rating was provided
    if (rating) {
      await updateGameRating(gameId);
    }

    res.status(201).json({ message: 'Review added successfully', reviewId: reviewRef.id });
  } catch (error) {
    console.error('Error adding review:', error);
    res.status(500).json({ error: 'Failed to add review' });
  }
});

// Helper function to update game rating
async function updateGameRating(gameId) {
  try {
    const reviewsSnapshot = await firestore.collection('reviews')
      .where('gameId', '==', gameId)
      .where('rating', '!=', null)
      .get();

    let totalRating = 0;
    let count = 0;

    reviewsSnapshot.forEach(doc => {
      const review = doc.data();
      if (review.rating) {
        totalRating += review.rating;
        count++;
      }
    });

    const averageRating = count > 0 ? parseFloat((totalRating / count).toFixed(1)) : 0;

    await db.ref(`games/${gameId}`).update({
      averageRating,
      totalRatings: count
    });
  } catch (error) {
    console.error('Error updating game rating:', error);
  }
}
```

Figure 19: Update Review Operation - Backend

```
// server.js - Express.js endpoint to update review
app.put('/api/reviews/:reviewId', async (req, res) => {
  try {
    const { reviewId } = req.params;
    const { userId, text, rating } = req.body;

    // Get review to verify ownership
    const reviewDoc = await firestore.collection('reviews').doc(reviewId).get();

    if (!reviewDoc.exists) {
      return res.status(404).json({ error: 'Review not found' });
    }

    const reviewData = reviewDoc.data();

    // Check ownership
    if (reviewData.userId !== userId) {
      return res.status(403).json({ error: 'User can only edit their own reviews' });
    }

    // Validate input
    if (!text && !rating) {
      return res.status(400).json({ error: 'Must provide either text, rating, or both' });
    }

    if (rating && (rating < 1 || rating > 5)) {
      return res.status(400).json({ error: 'Rating must be between 1 and 5' });
    }

    // Update review
    await firestore.collection('reviews').doc(reviewId).update({
      text: text || null,
      rating: rating || null
    });

    // Recalculate rating if changed
    if (rating !== reviewData.rating) {
      await updateGameRating(reviewData.gameId);
    }

    res.json({ message: 'Review updated successfully' });
  } catch (error) {
```

```
    console.error('Error updating review:', error);
    res.status(500).json({ error: 'Failed to update review' });
  }
});
```

Figure 20: Delete Review Operation - Backend

```
// server.js - Express.js endpoint to delete review
app.delete('/api/reviews/:reviewId', async (req, res) => {
  try {
    const { reviewId } = req.params;
    const { userId, isAdmin } = req.body;

    const reviewDoc = await firestore.collection('reviews').doc(reviewId).get();

    if (!reviewDoc.exists) {
      return res.status(404).json({ error: 'Review not found' });
    }

    const reviewData = reviewDoc.data();

    // Check permissions
    if (!isAdmin && reviewData.userId !== userId) {
      return res.status(403).json({ error: 'User can only delete their own reviews' });
    }

    const gameId = reviewData.gameId;
    const hadRating = reviewData.rating !== null;

    // Delete review
    await firestore.collection('reviews').doc(reviewId).delete();

    // Update game rating if review had a rating
    if (hadRating) {
      await updateGameRating(gameId);
    }

    res.json({ message: 'Review deleted successfully' });
  } catch (error) {
    console.error('Error deleting review:', error);
    res.status(500).json({ error: 'Failed to delete review' });
  }
});
```

Figure 21: Display Games with Images - Frontend

```
// app.js - Vanilla JavaScript to display games with images
function displayGamesWithImages(games) {
  const gamesContainer = document.getElementById('games-container');
  gamesContainer.innerHTML = '';

  games.forEach(game => {
    const gameCard = document.createElement('div');
    gameCard.className = 'game-card';
    gameCard.innerHTML = `
      <div class="game-image">
        <img src="${game.image || '/images/placeholder.jpg'}" alt="${game.title}"
            onerror="this.src='/images/placeholder.jpg'">
      </div>
      <div class="game-info">
        <h3>${game.title}</h3>
        <p class="game-genre">${game.genre}</p>
        <p class="game-platform">${game.platform}</p>
        <div class="game-rating">
          <span class="stars">⭐ ${game.averageRating}/5</span>
          <span class="review-count">(${game.totalRatings} reviews)</span>
        </div>
        <p class="game-release">Release: ${new
Date(game.releaseDate).toLocaleDateString()}</p>
      </div>
    `;
    gameCard.addEventListener('click', () => viewGameDetails(game.gameId));
    gamesContainer.appendChild(gameCard);
  });
}
```

Figure 22: Search Games by Title - Frontend

```
// app.js - Vanilla JavaScript to search games
async function searchGamesByTitle(searchTerm) {
  try {
    const response = await
fetch(`/api/games/search?title=${encodeURIComponent(searchTerm)}`);
    const games = await response.json();

    displayGamesWithImages(games);
  } catch (error) {
    console.error('Error searching games:', error);
```

```
    showError('Failed to search games');
  }
}
```

Figure 23: Filter by Genre - Frontend

```
// app.js - Vanilla JavaScript to filter by genre
async function filterByGenre(genre) {
  try {
    const response = await fetch(`/api/games/filter?genre=${encodeURIComponent(genre)}`);
    const games = await response.json();

    displayGamesWithImages(games);
  } catch (error) {
    console.error('Error filtering games:', error);
    showError('Failed to filter games');
  }
}

// Initialize genre filter dropdown
document.addEventListener('DOMContentLoaded', () => {
  const genreFilter = document.getElementById('genre-filter');
  const genres = ['Action', 'RPG', 'Strategy', 'Horror', 'Racing', 'Adventure', 'Platformer', 'FPS'];

  genres.forEach(genre => {
    const option = document.createElement('option');
    option.value = genre;
    option.textContent = genre;
    genreFilter.appendChild(option);
  });

  genreFilter.addEventListener('change', (e) => {
    if (e.target.value) {
      filterByGenre(e.target.value);
    }
  });
});
```

Figure 24: Multi-Criteria Filter - Backend

```
// server.js - Express.js endpoint for multi-criteria filtering
app.get('/api/games/filter-multi', async (req, res) => {
  try {
```

```
    const { platform, genre, status, minRating, startDate, endDate } = req.query;

    // Get all games from Realtime Database
    const gamesSnapshot = await db.ref('games').once('value');
    let games = [];

    gamesSnapshot.forEach(childSnapshot => {
      games.push(childSnapshot.val());
    });

    // Apply filters
    if (platform) {
      games = games.filter(game => game.platform.includes(platform));
    }

    if (genre) {
      games = games.filter(game => game.genre.includes(genre));
    }

    if (status === 'upcoming') {
      games = games.filter(game => game.upcoming === true);
    } else if (status === 'released') {
      games = games.filter(game => game.released === true);
    }

    if (minRating) {
      games = games.filter(game => game.averageRating >= parseFloat(minRating));
    }

    if (startDate && endDate) {
      games = games.filter(game =>
        game.releaseDate >= startDate && game.releaseDate <= endDate
      );
    }

    res.json(games);
  } catch (error) {
    console.error('Error filtering games:', error);
    res.status(500).json({ error: 'Failed to filter games' });
  }
});
```

Figure 25: Add Favorite - Frontend

```
// app.js - Vanilla JavaScript to add favorite
async function addFavorite(userId, gameId) {
  try {
    const response = await fetch('/api/favorites', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ userId, gameId })
    });

    const data = await response.json();

    if (response.ok) {
      showMessage('Game added to favorites!');
      updateFavoriteButton(gameId, true);
    } else {
      showError(data.error || 'Failed to add favorite');
    }
  } catch (error) {
    console.error('Error adding favorite:', error);
    showError('Failed to add favorite');
  }
}

function updateFavoriteButton(gameId, isFavorited) {
  const button = document.getElementById(`favorite-btn-${gameId}`);
  if (button) {
    button.textContent = isF
```