Web Programming

Game Management & Distribution MIS

By: Abdul Rafy

1. Introduction

GameHub is a web-based platform for discovering, purchasing, and managing digital games. It is developed in C# using the .NET Framework 4.8 and ASP.NET MVC, with Entity Framework (Database- First) and SQL Server as the data layer. The system provides two main experiences:

1. A user storefront where players browse games, manage accounts, buy titles, and view their personal game library.
2. An admin console where operators manage the catalog, discounts, users, transactions, and sales reports.

The project demonstrates a complete end- to- end web application: UI, business logic, data persistence, security, and reporting, implemented with modern web technologies (Bootstrap 5, jQuery, Razor, HTML/CSS, JavaScript).

2. Problem Statement and Objectives

Digital game marketplaces require more than just a product list. They must track users, purchases, ownership, discounts, and reports for decision making. Many student projects show only part of this flow (e.g., a catalog without real purchases or admin analytics).

GameHub aims to:

- Provide a simple but realistic game storefront with secure- style checkout and personal libraries.
- Include an admin back office with dashboards, reporting, and management tools.
- Demonstrate a coherent software architecture using ASP.NET MVC, Entity Framework, and SQL Server.
- Serve as a solid base for future enhancements such as real payment gateways and stronger security.

Key objectives:

- Implement core user features: browsing, searching, registration, login, wishlist-style behavior, purchasing, and library.
- Implement admin capabilities: game catalog CRUD, user management, discount codes, transaction history, and revenue reports.
- Use a clean architecture and consistent UI so that new developers and non- technical stakeholders can easily understand how the system works.

3. Scope

In scope:

- Web UI for users and admins (desktop- first, responsive layout).
- Session- based authentication and simple role handling (user vs admin).
- Recording purchases as transactions and reflecting ownership in a user library.
- Basic discount system and reports (top sellers, revenue summaries).
- Dark themed Bootstrap interface with validation and some AJAX interactions.

Out of scope (future work):

- Real payment gateway integration and PCI compliance.
- Advanced recommendation engine and personalization.
- Full role- based authorization model and multi- tenant support.
- Mobile apps or a complete SPA front end.

4. System Overview

GameHub follows the classic MVC pattern:

- Model (M) – Entity Framework entity classes representing database tables: **Game**, **User**, **Transaction**, **UserLibrary**, **Discount**, **GameFeedback**, etc.
- View (V) – Razor **.cshtml** files under **Views/*** using Bootstrap 5 and custom CSS to generate HTML pages.
- Controller (C) – C# classes in **Controllers** that process HTTP requests, run business rules, and select views.

The solution is a single ASP.NET MVC project that can be hosted on IIS / IIS Express and connects to a SQL Server database.

Two main user roles:

- Guest/User – Browses catalog, registers, logs in, purchases games, and manages the personal library.
- Admin – Manages games, users, discounts, and views analytics and reports.

The System (server) is responsible for authentication, authorization, pricing logic, discount validation, recording transactions, and running reporting queries.

5. Functional Requirements

5.1 User-Facing Features

- Browse the game catalog by genre and keyword.
- View detailed information for each game: title, description, price, release date, and rating.
- Register for a new account and log in with email and password.
- Maintain a personal profile and account status.
- Purchase games via a checkout page, with optional discount codes.
- View a My Library section listing owned games and activation codes.
- Submit simple ratings/feedback for games.

5.2 Admin and Operational Features

- Access an Admin Dashboard with:
    1. Total counts for games, users, and transactions.
    2. Revenue totals and “today/this month” statistics.
    3. Top-selling games.
- Manage games (create, view, update, delete).
- Manage users (view list, edit, activate/deactivate, change roles).
- Manage transactions (view/edit/delete with basic audit information).
- Create and manage discount codes and attach them to games.
- Generate basic reports:
    1. Sales summary by date range.
    2. Per-game revenue.
    3. Top sellers and trends.

- Trigger or monitor backups / logs (conceptual operations, recorded in UI and DB).

6. Non-Functional Requirements

- **Usability:** Clean, consistent dark interface with clear navigation and Bootstrap components.
- **Performance:** Suitable for a small to medium catalog and user base; LINQ queries are optimized with eager loading where needed..
- Manage transactions (view/edit/delete with basic audit information).
- Create and manage discount codes and attach them to games.
- **Security (current level):**

  1. Session- based authentication.
  2. Admin areas protected by a custom authorization filter.
  3. Passwords currently stored as simple hashes.

- **Reliability:**: Database-backed persistence; purchase operations use transactional logic to ensure consistency between Transactions and UserLibrary.
- **Maintainability:** Clear MVC project structure, descriptive controller and view names, and EF models scaffolded from the database.

7. Technology Stack and Their Roles

- **C# (.NET Framework 4.8)** – Main programming language and runtime for controllers, models, filters, and helpers.
- **ASP.NET MVC (Razor)** – Application framework that handles routing, controller dispatch, and server-side HTML rendering.
- **Entity Framework (Database-First)** – ORM layer mapping SQL Server tables to C# entity classes. Used for querying and persisting data with LINQ.

- **SQL Server** – Relational database storing all persistent data (games, users, transactions, discounts, libraries, feedback).
- **Bootstrap 5 – Front-end CSS** framework providing responsive layout, grid system, and pre-built UI components.
- **Reliability:**: Database-backed persistence; purchase operations use transactional logic to ensure consistency between Transactions and UserLibrary.
- **Maintainability:** Clear MVC project structure, descriptive controller and view names, and EF models scaffolded from the database.
- **jQuery** – Simplifies DOM manipulation, AJAX calls, and client- side validation wiring.
- **HTML/CSS** – Markup and styling of all pages, including the dark theme and card- based layout.
- **JavaScript** – Custom client scripts for search, discount validation, and minor UX enhancements.

8. System Design

8.1 Logical Architecture

The logical architecture can be viewed in four layers:

1. Presentation Layer

- Views, layout, and static assets.
- Form inputs, validation messages, and interactive elements.

2. Controller Layer

- Embedded in controllers via helper methods.
- Rules for pricing, discount eligibility, and whether a user may purchase a game.
- Aggregation logic for dashboards and reports.

3. Domain / Business Logic

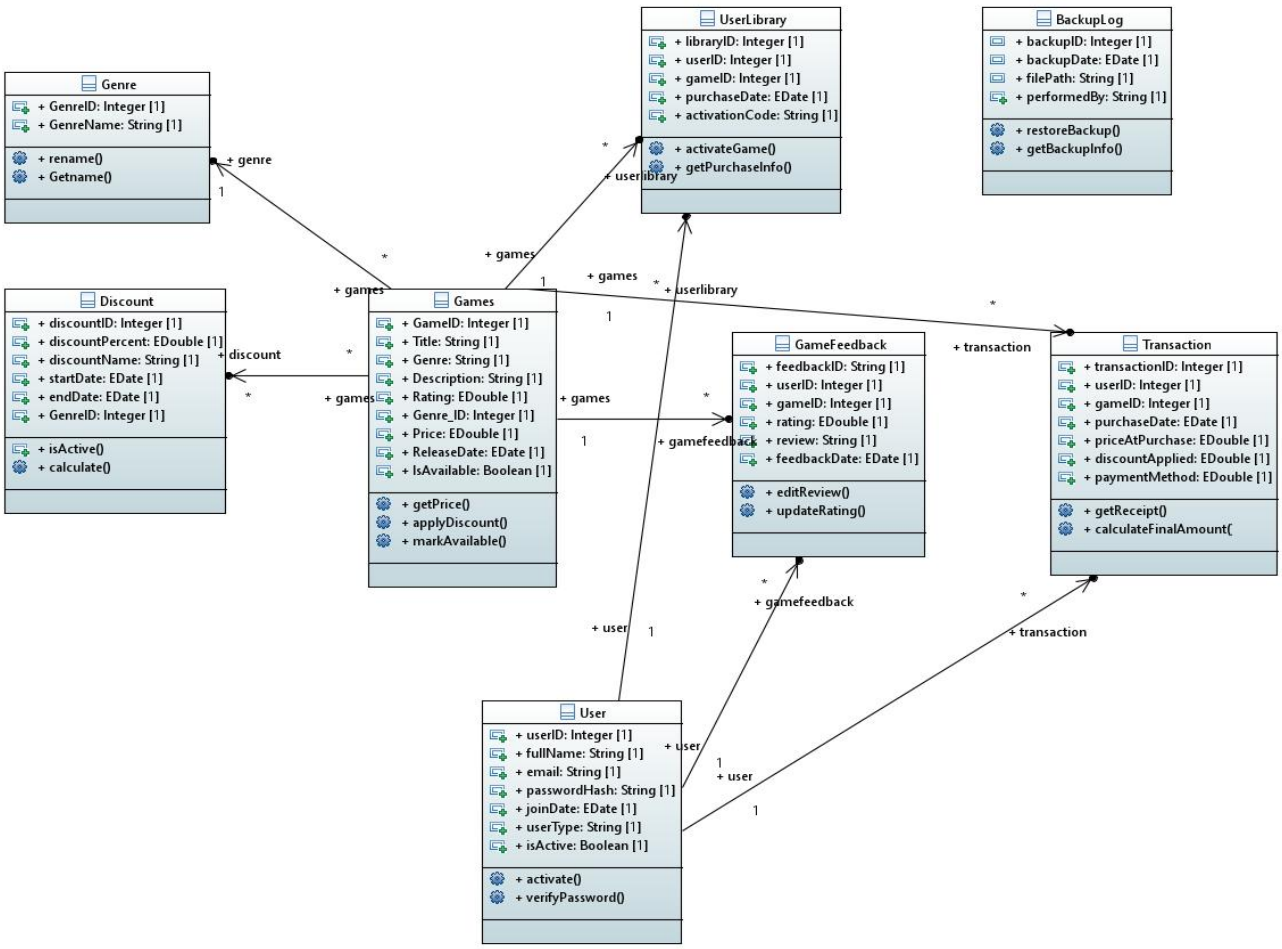
- Entity Framework context and entity classes.
- SQL Server as physical storage.

4. Domain / Business Logic

- Entity Framework context and entity classes.
- SQL Server as physical storage.

82 Data Models

Class Diagram

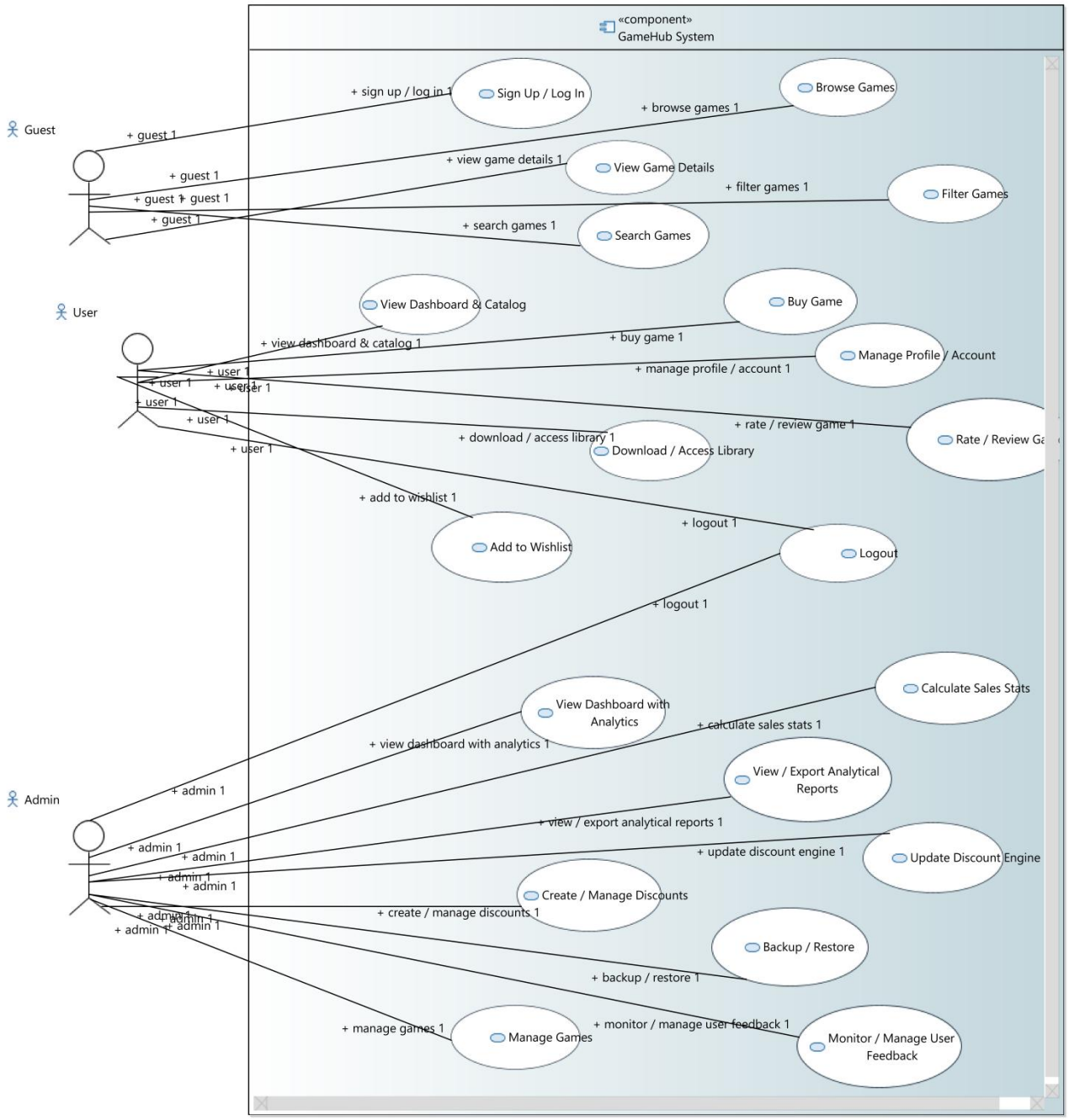


The class diagram shows the main data model for GameHub and how the entities relate.

- User stores player/admin details and links to their library, transactions, and feedback.
- Games and Genre model game information and category.
- Discount holds promotional rules applied to games.
- UserLibrary and Transaction represent purchased/owned games and payment records.
- GameFeedback captures user ratings and reviews.
- BackupLog records system backup operations.

Together, these classes form the static structure that all other diagrams (use case, sequence, etc.) use.

Use Case Diagram



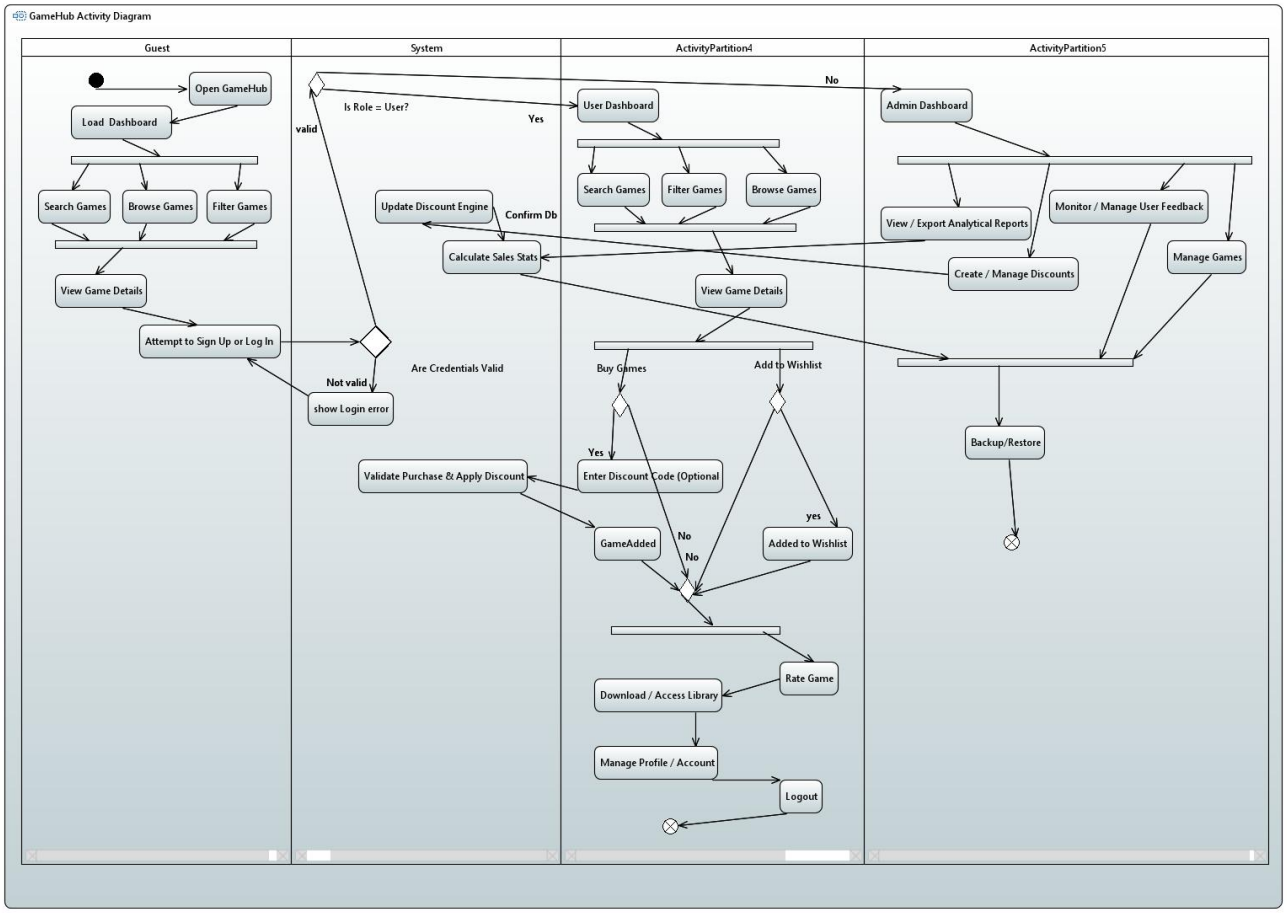
The use case diagram shows how different actors interact with the GameHub system:

- Guest can browse and search games, view details, and sign up/log in.
- User (logged- in) can view the catalog dashboard, filter/search games, buy games, download/access their library, add to wishlist, manage profile, rate/review games, and log out.

- Admin can view analytics dashboard, calculate sales stats, view/export reports, create/manage discounts, update the discount engine, manage games, monitor user feedback, and perform backup/restore.

It summarizes all main functional requirements from each user's perspective.

Activity Diagram



This activity diagram shows the main workflows of GameHub across Guest, System, User, and Admin lanes.

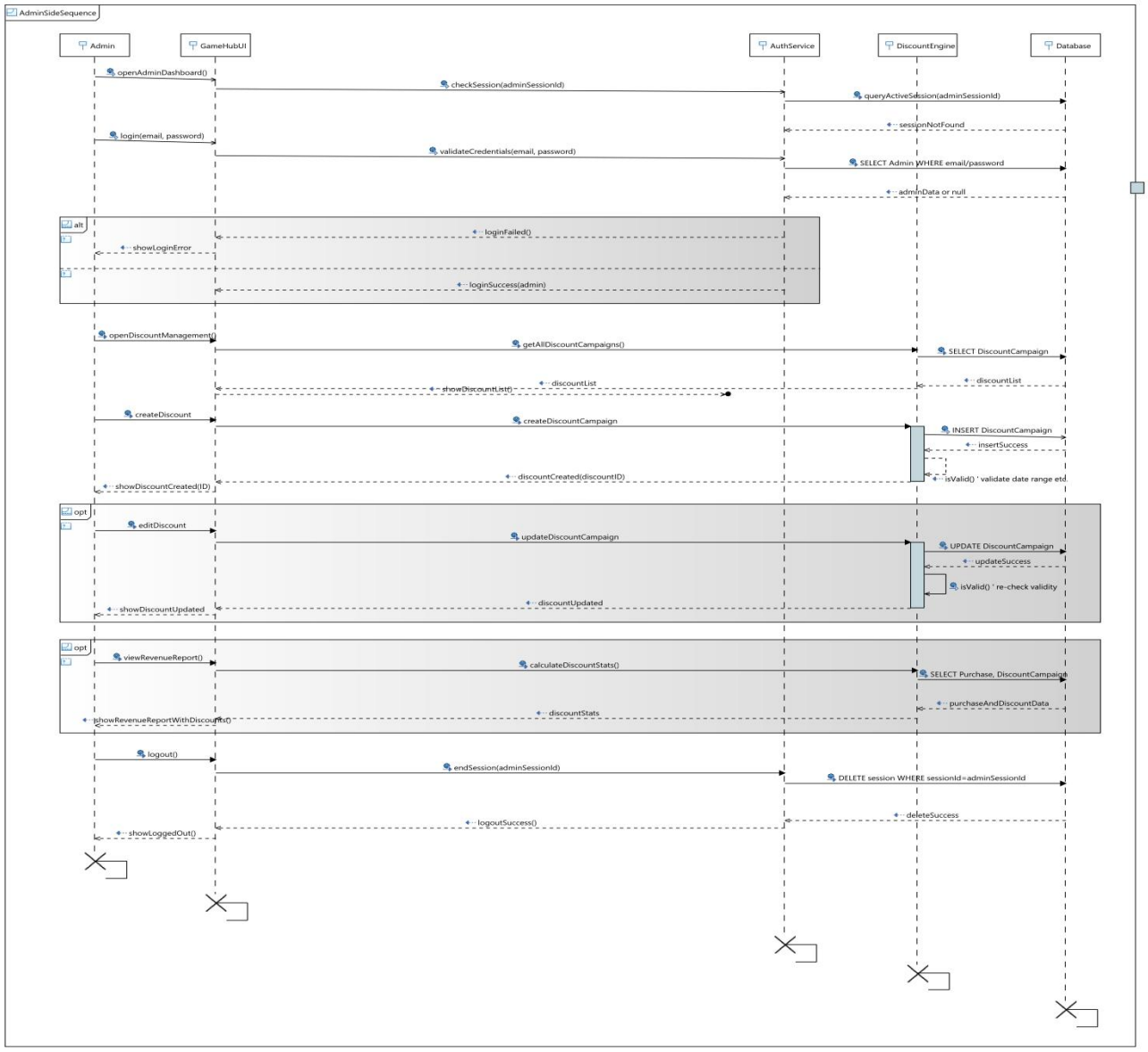
- A guest opens GameHub, loads the dashboard, searches/browses/filters games, views details, and attempts to sign up or log in.
- The system validates credentials and user role. If valid and role = User, the user dashboard is shown; otherwise a login error is displayed.
- A user can search/filter/browse games, view details, buy games (optionally entering a

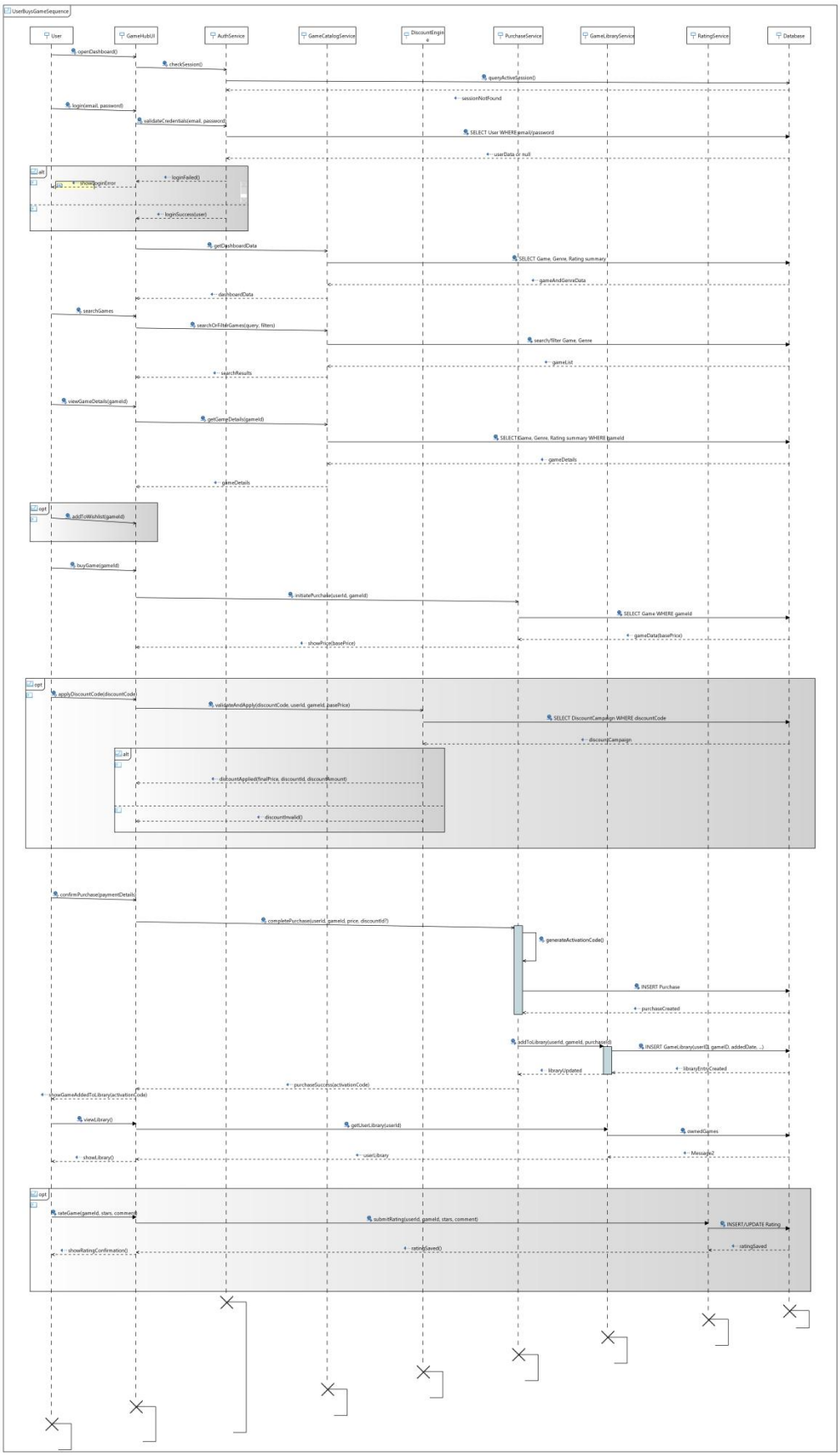
discount code), add games to wishlist, download/access their library, rate games, manage profile, and log out.

- An admin accesses the admin dashboard to view/export analytical reports, monitor/manage feedback, create/manage discounts, manage games, and trigger backup/restore, which ends the flow.

This diagram summarizes the end-to-end behavior from opening GameHub to user actions and admin maintenance.

Sequence Diagram




UserBuysGameSequence
User
GameHubUI
AuthService
GameCatalogService
DiscountEngine
PurchaseService
GameLibraryService
RatingService
Database
openDashboard()
checkSession()
queryActiveSession()
sessionNotFound
login(email, password)
validateCredentials(email, password)
SELECT User WHERE email/password
userData or null
alt
loginFailed()
showLoginError
loginSuccess(user)
getDashboardData
SELECT Game, Genre, Rating summary
gameAndGenreData
dashboardData
searchGames
searchOrFilterGames(query, filters)
search/filter Game, Genre
gameList
searchResults
viewGameDetails(gameId)
getGameDetails(gameId)
SELECTGame, Genre, Rating summary WHERE gameId
gameDetails
opt
addToWishlist(gameId)
buyGame(gameId)
initiatePurchase(userId, gameId)
SELECT Game WHERE gameId
gameData(basePrice)
showPrice(basePrice)
applyDiscountCode(discountCode)
validateAndApply(discountCode, userId, gameId, basePrice)
SELECT DiscountCampaign WHERE discountCode
discountCampaign
discountApplied(finalPrice, discountId, discountAmount)
discountInvalid()
confirmPurchase(paymentDetails)
completePurchase(userId, gameId, price, discountId?)
generateActivationCode()
INSERT Purchase
purchaseCreated
addToLibrary(userId, gameId, purchaseId)
INSERT GameLibrary(userID, gameID, addedDate, ...)
libraryUpdated
libraryEntryCreated
purchaseSuccess(activationCode)
showGameAddedToLibrary(activationCode)
viewLibrary()
getUserLibrary(userId)
ownedGames
showLibrary()
userLibrary
Message2
rateGame(gameId, stars, comment)
submitRating(userId, gameId, stars, comment)
INSERT/UPDATE Rating
ratingSaved
showRatingConfirmation()
rateAdded()

User-Side Sequence Diagram – Game Purchase Flow

This sequence diagram shows how a normal user interacts with GameHub to buy a game. The user signs up or logs in through the GameHub UI, which calls the Auth Service and Database to validate credentials. After login, the user searches and views a game via the Game Catalog Service. When the user starts a purchase (optionally with a discount code), the Purchase Service coordinates with the Discount Engine, Game Library, and Database to validate the discount, calculate the final price, store the purchase, and add the game to the user's library. Finally, the UI confirms the successful purchase to the user.

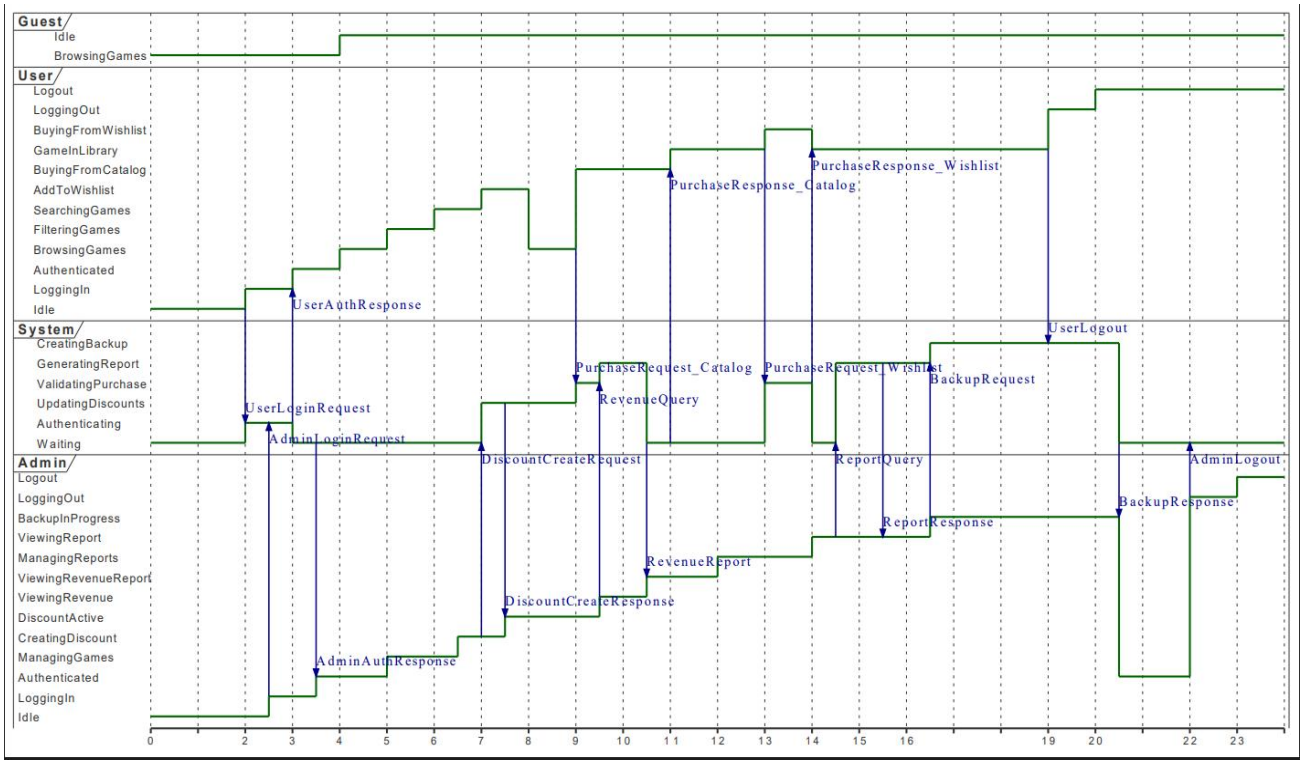
Admin-Side Sequence Diagram – Discount & Reporting Flow

This sequence diagram describes how an admin manages discounts and views revenue. The admin logs in via the GameHub UI; Auth Service checks credentials against the Database. After successful login, the admin opens the discount management screen. The Discount Engine and Database are used to create or update discount campaigns. Later, the admin opens the revenue report screen; Reporting logic queries purchase and discount data from the Database and returns aggregated statistics. The admin can then log out, and the Auth Service ends the session.

Timing Diagram

This timing diagram shows how the states of Guest, User, System, and Admin change over time and how their interactions are ordered.

- The User logs in, browses, filters/searches games, buys from catalog and wishlist, then logs out.
- The Admin logs in, creates discounts, views revenue and reports, triggers backup, then logs out.
- The System switches between waiting, authenticating, updating discounts, validating purchases, generating reports, and creating backups.
- Vertical messages (login requests, purchase requests, report/backup requests and responses) show when each actor and the system communicate during the scenario.

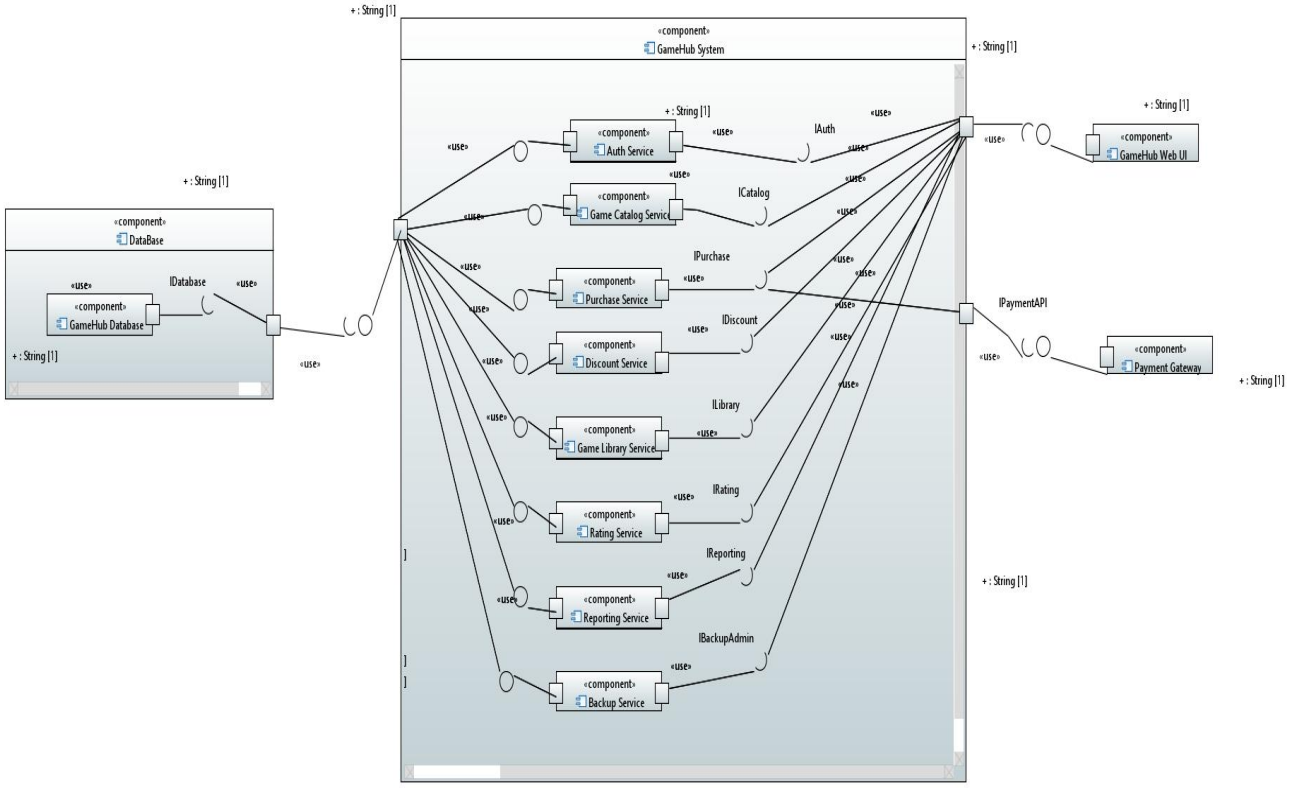


Component Diagram

The component diagram shows the high- level architecture of GameHub and how components communicate through interfaces.

- A main GameHub System component contains backend services: Auth, Game Catalog, Purchase, Discount, Game Library, Rating, Reporting, and Backup Service.
- A separate DataBase component contains the GameHub Database and provides the IDatabase interface used by all backend services.
- The GameHub Web UI component, outside the system, requires interfaces IAuth, ICatalog, IPurchase, ILibrary, IRating, IReporting, and IBackupAdmin from the internal services.
- The Payment Gateway component provides the IPaymentAPI interface, required only by the Purchase Service.

Together, this diagram explains how frontend, services, database, and external payment system are organized and connected.



Deployment Diagram

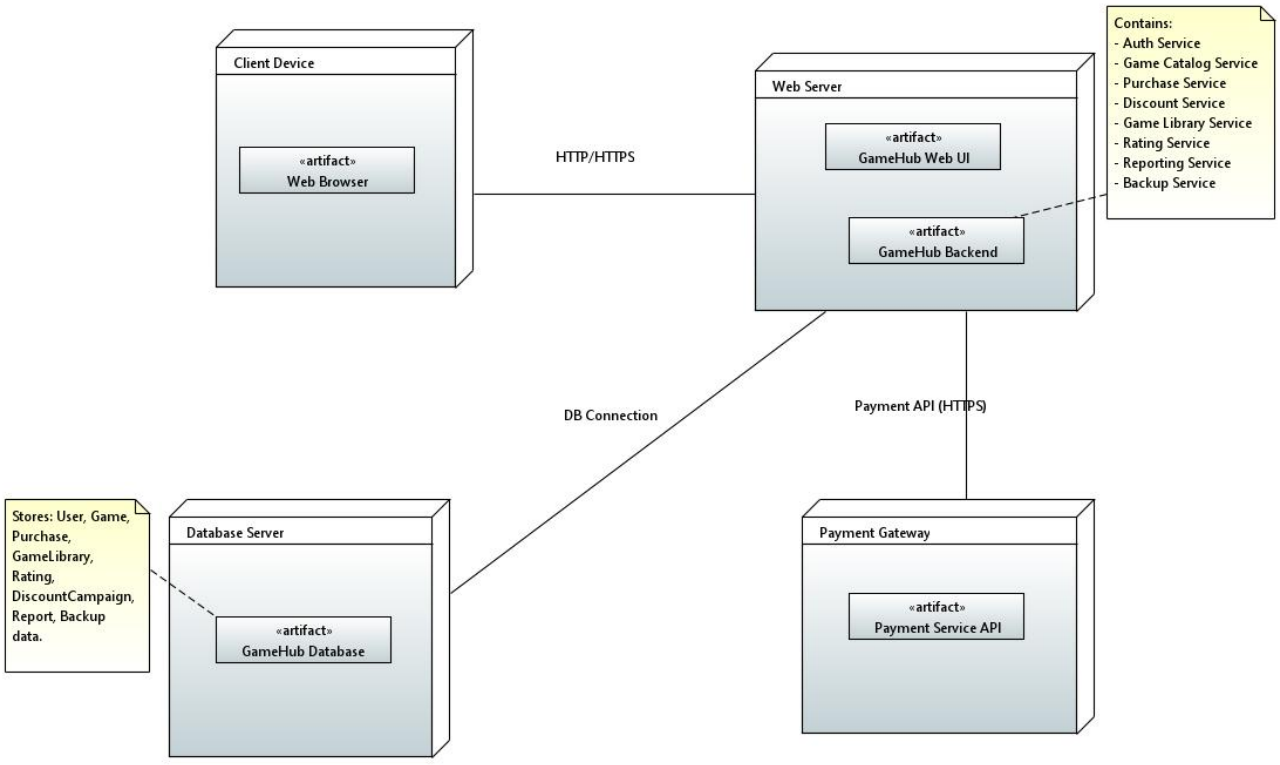
The deployment diagram shows how GameHub is physically deployed:

- A Client Device node hosts the Web Browser artifact, which users use to access GameHub.
- A Web Server node hosts GameHub Web UI and GameHub Backend, which contain all backend services (Auth, Catalog, Purchase, Discount, Library, Rating, Reporting, Backup).
- A Database Server node hosts the GameHub Database artifact, storing all core data (User, Game, Purchase, GameLibrary, Rating, DiscountCampaign, Report, Backup data).
- A Payment Gateway node hosts the external Payment Service API.

Communication paths show:

- Client Device ↔ Web Server via HTTP/HTTPS.
- Web Server ↔ Database Server via a DB connection.
- Web Server ↔ Payment Gateway via a Payment API (HTTPS).

This diagram links the logical architecture to the actual runtime environment.



9. Detailed Module Descriptions

9.1 User Catalog and Game Details

- Controller: **GamesController**
- Main actions:
  - **Index** – Loads games, supports filtering by genre, search keyword, and possibly sorting by popularity or price.

- **Details(id)** – Fetches a single game plus its feedback and any discount information. Also determines if the current user already owns the game.

The associated views present the data as game cards, hero banners, and responsive layouts. Searches may call dedicated actions or return JSON for partial updates.

92 Account Management

- Controller: **AccountController**
- Features:
  - **Register** – Creates a new **User** record with base profile data.
  - **Login** – Validates credentials, sets session values (**UserID**, **UserName**, **UserType**, **IsAuthenticated**).
  - **Logout** – Clears session, returning the user to guest mode.

Session keys are later used for personalization (e.g., showing "My Library") and for enforcing admin authorization.

93 Purchase and Checkout

- Controller: **TransactionsController**
- Flow:
  - **Checkout** – User navigates to checkout for a chosen game. The controller loads the game and any applicable discount, building a view model with original and discounted prices.
  - **Discount Validation** – Optional AJAX call to **ValidateDiscount**; the controller checks code validity and returns discount details or an error.
  - **Purchase** – On form submit, the controller:
    1. Ensures the logged- in user and game exist and that the user does not already own the game.
    2. Finds active discounts; calculates final price.
    3. Begins a database transaction.
    4. Creates a **Transaction** row and modifies **UserLibrary**, adding or updating an activation code.

5. Commits or rolls back if an error occurs.

The result is a consistent record of the purchase in both transaction history and the user's library.

9.4 User Library

The User Library interface shows all games a user owns. It is driven by:

- Data in **UserLibrary** joined with **Game** for display fields.
- Controller actions that enforce login and load only the current user's records.

This module ensures users can always see which titles they purchased and their activation codes.

9.5 Admin Dashboard and Management

- Controller: **AdminController** plus supporting controllers (**GamesController**, **UsersController**, **TransactionsController**, **ReportsController**).

Dashboard:

- Loads totals for games, users, and transactions.
- Computes revenue and shows today/this month stats.
- Lists top sellers based on grouped transaction data.

Game Management:

- Associates games with discounts.
- Admin views for create/edit/delete.

User Management:

- Listing all users.
- Editing roles, setting IsActive, and basic account controls.

Transaction & Discount Management:

- Viewing and editing transaction records for correction or review.
- Creating and updating discount codes and linking them to games.

Reports:

- Synchronizes date filters from the UI with EF queries.
- Generates sales summaries and per- game revenue for decision making.

The AdminAuthorizeAttribute filter ensures these actions are only accessible to authenticated admins.

10. Implementation Overview (Without Code)

This section summarizes how the implementation is organized without showing source code:

- **Controllers** directory contains C# classes for each module. Each action method:
  1. Accepts parameters (route values, form posts).
  2. Uses the EF context for data access.
  3. Applies business rules and populates view models.
  4. Returns a view or JSON result.

- **Models** directory and EDMX define the schema and generated entities. Entity classes represent rows, while navigation properties represent relationships.
- **Views** directory is split by controller, with each **.cshtml** view tied to a specific feature page. Shared layout and partial views encapsulate repeated UI (navigation, headers, messages).
- **Content** holds CSS and theme files; **Scripts** holds jQuery and validation libraries.
- **Filters** implement cross-cutting behaviour like admin-only routes.

11. Limitations and Future Enhancement

Known Limitations

- Passwords are stored in a simplistic way, not suitable for production.
- Payment handling is simulated; no real payment gateway is integrated.
- Business logic is mainly inside controllers, which complicates unit testing.
- Logging is minimal and uses debug outputs.
- Timezone handling is basic and assumes a single primary region.

Proposed Improvements

- Integrate ASP.NET Identity with hashed/salted passwords and full account management.
- Use a payment provider (e.g., Stripe, PayPal) for real transactions and PCI compliance.
- Extract services (e.g., PurchaseService, ReportingService) and repositories to separate business logic and improve testability.
- Introduce structured logging and centralized monitoring.
- Enhance reports (trend graphs, cohort analysis, retention).
- Add recommendation features (similar games, "customers also bought").

12. Conclusion

GameHub is a complete demonstration of a modern ASP.NET MVC application that covers the full lifecycle of a digital product sale:

- Guests and users browse and search games.
- Registered users purchase games and manage their libraries.
- Admins control catalog, users, discounts, and can see revenue and performance.

Behind the scenes, a well- structured stack—C#, ASP.NET MVC, Entity Framework, SQL Server, Bootstrap, and jQuery—works together to deliver a coherent system. Although some aspects (password security, payments, logging) remain simplified for educational purposes, the project already reflects the essential patterns and components used in real- world web applications, and provides a strong foundation for further enhancement or deployment.