Documentation

My Game Journal

Contents:

- Problem
  - Case Study
  - Existing Solutions
- Solution
  - Features
  - Resources
  - Technologies
  - Innovations
  - Limits
- Conclusion

Problem

Case Study

Gamers use multiple platforms in order to play games such as Steam for CS:GO, Riot Games's Client for League of Legends and Epic Games for Fortnite. Because of this, players find it difficult to share their progression in a game. Some platforms lack features for sharing screenshots or personal thoughts, forcing users to rely on third-party alternatives. Even if it was possible for a game to be shared, it might not be seen by friends who use other platforms.

In addition, this makes it hard for users to be able to invite their friends to play a game with them in real life or online.

Existing Solutions

An existing platform that is known for solving most of the problems is Steam, which is a digital distribution platform where you can buy games, share your screenshots and thoughts with other Steam users and even invite your Steam Friends to play with you. Because of its large gaming library and player base, most of the users prefer to use this platform, making it more possible to reach that friend through Steam.

However, major titles such as Valorant, Fortnite or Genshin Impact, are not available through Steam as they are absent from Steam due to publisher exclusivity (such as Epic Games or the Riot Games Client) .

There are also games that are exclusive for consoles, for example Nintendo Games which can be only played on Nintendo's own consoles (Switch, DS, WII, etc). For these cases, the console manufacturers offer their own social platform where you can invite your friends to play with you and share your gaming journey, but it's mostly only for that specific console and can be frustrating for other console or PC users to contact them.

Solution

As a solution, we created "My Game Journal", a platform where the users have their own gaming library and can manage their "Games Wishlist", "Ongoing Games" and "Completed Games" categories. The user can share their thoughts about those games and share the media of their journey through it. In addition, they can connect with other players and invite them to online or LAN gaming sessions.

Features

Our application contains features such as:

- Create a personal gaming library where you can add games in the following categories: wishlist, ongoing and finished
- Add media to games for showing the user's achievements or journey
- Add your thoughts about the game
- See other user's libraries, with their added media and thoughts
- Search users based on their stats or location
- Invite other users to play a specific game, either online or offline

Resources

Our application has the following resources: User, Game, Media, Challenge.

User:

It represents an individual registered account on the platform. It holds the identity, preferences, and social connections of the gamer. It has a User ID, Username, Email, Password, Location and Stats. The User interacts with all the other resources as it follows:

- Game: A User has a library where they can add Games and categorise it by the state of the game (if it's on a wishlist, it's ongoing or it's finished).
- Media: A User can add media to a Game, such as a screenshot or video gameplay so they can register their journey.
- Challenge: A User can invite other Users to a Challenge, either online or offline, for a certain Game (example: User A invites User B to an online session of FIFA).

Game:

It represents a specific video game title available in the application's global database given by the RAWG API. It has a Game ID, Title, Genre, Cover Image, Platforms and Price. The Game interacts only with User and Media:

- User: A Game is added to a User's Library in order for the User to record their journey with it.
- Media: A game can have multiple Media attachments that represent the User's journey through that specific Game.

Media:

It represents the visual proof of a User's journey (screenshots, clips, or achievement captures) uploaded to cloud storage. It has Media ID, File, Media Type (if it's a video or an image), Timestamp and its Associated Description. Media interacts with User and Game:

- User: A Media entry is linked to an User.
- Game: A Media entry is linked to a Game.

Challenge:

It represents the invitation system between two Users. It has a Challenge ID, Host User ID, Invited User ID, Game ID, Session Type and Status. Challenge interacts with User and Game:

- User: UserA decides to use the Challenge feature to invite UserB to an online/offline game session.
- Game: When UserA invites UserB, Challenge also gives the UserB the Game Name that it was chosen by UserA.

Technologies

The development of the game library application is centered on a Cloud-based architecture, utilizing Google Cloud components.

Backend & Infrastructure

- **Python & FastAPI:** Chosen as the core backend framework due to its high performance, native support for asynchronous programming

and automatic OpenAPI/Swagger documentation generation. This ensures fast API responses when fetching data from external gaming networks.

- **Docker:** Used to containerize the application, ensuring that the development environment matches the production environment.
- **Google Cloud Run:** A serverless platform used to deploy the Docker containers. It was chosen because it automatically scales up or down based on traffic, using the exact compute resources.
- **Google Cloud SQL:** Houses our relational database (MySQL). It provides automated backups, scaling, and high availability for storing complex user libraries, relationships, and challenge statuses.
- **Google API Gateway:** Acts as the single entry point for the frontend, handling routing, protecting backend services, and managing API traffic.
- **Google Secret Manager:** Securely stores sensitive data such as database credentials, session tokens, and external API keys, ensuring they are never hardcoded into the source code.
- **Firebase Authentication & Storage:** Firebase is utilized for quick, secure user authentication (OAuth/Email sign-in) and Firebase Storage acts as our cloud bucket to host user-uploaded screenshots and media.
- **Google Cloud Monitoring & reCAPTCHA:** Cloud Monitoring provides real-time logs and error tracking, while reCAPTCHA protects our registration and login endpoints from automated bot attacks.

Frontend Ecosystem

- **Node.js & Vite:** Vite is used as the frontend build tool because of its Hot Module Replacement during development and optimized bundling for production.
- **JavaScript:** The core language used to build a dynamic, responsive Single Page Application (SPA) that allows smooth navigation without page reloads.

External APIs Integration

- **RAWG API:** Used as the database to fetch up-to-date game titles, cover images, genres, and supported platforms.
- **CheapShark API:** Integrated to fetch live pricing and deals for games, adding monetization context to the user's wishlist.
- **Frankfurter API:** Since CheapShark provides prices primarily in USD, the Frankfurter API is utilized to perform real-time currency conversion into EUR and RON, providing a localized experience.

Innovations

While standard gaming applications focus solely on data tracking, My Game Journal introduces innovation across social, application-level, and infrastructure levels.

Social Innovation: Decentralized "Couch Play" & Physical Session Orchestration

Most modern gaming platforms focus entirely on isolated online matchmaking. The social innovation of our application lies in "couch co-op" and LAN parties. By mapping the platform's Location metrics and the offline capabilities of the /api/challenges endpoint, the app allows users to host LAN parties and shared-device sessions. This shifts the focus from purely online matchmaking to real-world, centralized community gaming setups.

Performance Innovation: Media Upload Pipeline

To support the `/api/games/{gameId}/media/{mediaUuid}` endpoint without bottlenecking backend traffic, the platform eliminates traditional server-side file handling through a hybrid approach:

- **Direct Cloud Injection:** Utilizing the Signed Upload URLs workflow specified in the Storage Architecture, the client bypasses the FastAPI backend entirely, uploading data directly to Firebase/Google Cloud Storage. This drastically reduces server-side CPU utilization and eliminates data transit costs on the main backend.

Architectural Innovation: Unified API Gateway & Resource Aggregation

Instead of implementing a traditional, complex Load Balancer, the system utilizes a streamlined Google API Gateway as a reverse proxy and single entry point. The innovation lies in its role as a secure traffic aggregator. It enforces HTTPS, checks authentication metadata, controls rate limiting, and handles third-party data streams (RAWG, CheapShark, Frankfurter API) simultaneously. This prevents the frontend from making multiple fragmented API calls, ensuring optimal performance on low-spec client devices.

Infrastructure Resilience: Read-Replica Dependency (No-Cache)

To maintain structural simplicity during early deployment stages, the architecture intentionally excludes a Redis caching layer. To compensate for this without sacrificing performance during traffic spikes, the platform relies on an optimized Primary-Replica database architecture via Google Cloud SQL:

- The system routes data writes (such as creating users via `/api/users` or launching challenges) strictly to the Primary node.
- All high-frequency, read-heavy operations (browsing other users' journals, searching games, or verifying stats) are dynamically offloaded to multiple Read-Replicas. This ensures high availability and low latency without the architectural overhead of an external memory cache.

Limits

While the architecture is designed to be highly scalable in the future, the current implementation operates under several logical, functional, and physical constraints due to early-stage development and budgetary limitations:

- The application is strictly designed as a country-wide solution, rather than an international platform. It relies on localized features, such as the Frankfurter API for converting game prices specifically into RON and EUR, and location-based matchmaking tailored to regional coordinates. Because the infrastructure lacks a global Content Delivery Network (CDN) and multi-region database deployment, users accessing the app from outside the target region will experience high network latency.
- The platform explicitly does not support live video streaming or rich video hosting features. Allowing users to broadcast live gameplay or host uncompressed video clips requires immense network bandwidth, dedicated media transcoders, and high-cost video distribution infrastructure. To keep cloud costs sustainable, media features are strictly limited to static images (screenshots) and short, highly compressed clips handled asynchronously via Signed URLs.
- As a project developed under strict student budget constraints, the physical deployment layer is deliberately minimal. The application is deployed on a single server node without a dedicated external Load Balancer or an enterprise-grade global infrastructure. Consequently, the maximum throughput of the system is capped, meaning the platform can reliably handle a maximum of a few thousand concurrent requests. Exceeding this traffic threshold without scaling up the server node or adding memory caching could lead to response delays or temporary service interruptions.

Conclusion

My Game Journal successfully addresses the challenge of gaming ecosystem fragmentation by offering a space to preserve gaming memories and foster local communities.

Architecturally, the project demonstrates that enterprise-grade cloud paradigms, such as direct asset uploads via Signed URLs and Primary-Replica database separation, can be engineered efficiently on a limited student budget. By shifting heavy workloads to client-side compression and managed cloud storage, the system achieves excellent resilience and low latency using a single server node.

Business Model

Business Model

Customers (Customer Profile)

- Gamers looking for an alternative to Steam
- People who buy games online
- Developers who want to sell games without paying high commissions

Customer Assumptions

- Gamers are willing to try a new platform if it offers a simpler, faster, and more stable experience.
- Developers are frustrated by the high commissions charged in the industry and are willing to move at least part of their portfolio to a more advantageous platform.
- Cloud saves are considered a basic necessity by users, and the lack of this feature would be a reason for abandoning the platform.
- Gaming streamers have a strong influence and can convince their audience to use a new platform through recommendations.

What We Offer (Value Proposition)

- We provide a simple platform for purchasing and downloading games.
- We offer ratings and reviews for every game.
- We include cloud saves so players never lose their progress.
- We charge sellers only a 3% commission.
- We provide clear analytics and statistics for game sellers.

Partners

- Game developers
- Cloud storage service providers
- Payment processors such as Stripe
- Video platforms such as Twitch and YouTube

Daily Activities

- Improving the platform
- Reviewing newly uploaded games
- Maintaining servers and ensuring uptime

Customer Relationships

- Fast technical support

- Community support through Discord
- Regular news and updates

Key Resources

- Servers and cloud storage space
- Development team
- Game catalog

Channels

- Desktop application
- Website
- Gaming streamers and YouTubers

Main Expenses

- Servers and internet infrastructure (largest cost)
- Team salaries
- Marketing
- Payment processing fees

Revenue Streams

- 3% commission from every game sale
- 4 free games to upload then above 4 there is a fee
- Monthly subscription for players
- Paid Ads

System Requirements

System Requirements

This document defines the technical architecture, infrastructure/scalability requirements, and operational design for the gaming platform. The platform is designed to encourage multiplayer physical gaming sessions, minimizing hardware requirements for participants at group level.

The architecture covers:

- Client-facing frontend
- Backend services
- Scalability and load distribution
- Database replication and storage
- Monitoring and operational concerns
- Security and Authentication
- Exposed APIs

Customer Requirements

- Minimal device with internet access.
- Web browser with HTTPS support.
- Users must own or create a gaming account.
- A minimum of one gaming-capable device is required per gaming session.

Software Requirements

- Backend code
- Gateway code
- Frontend interface

Scalability Requirements

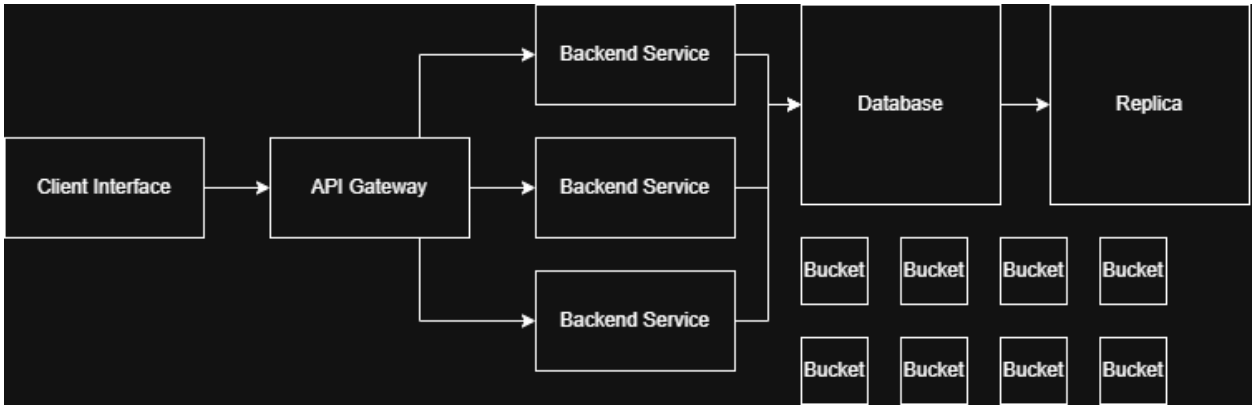
CAP Theorem:

In case of a network disruption high availability is preferred. Consistency is a manner of very sensitive and important data. In our case we manage personal game libraries. Syncs between nodes can be delayed with no bad consequences on the user's experience.

- High Availability:

    - Multiple backend nodes
    - Load balancing
- Fault Tolerance:
    - Database replication
- Performance:
    - API Gateway SSL termination:
        - Backend Service doesn't have to
    - Signed upload URLs:
        - Allow secure direct uploads
        - Reduce backend traffic
        - Minimize server-side bandwidth usage
- Cache Strategy:
    - Current architecture intentionally excludes Redis caching.
    - Application responses do not yet justify dedicated caching complexity
    - Simpler infrastructure management during early stages

Architecture



Client Interface

The frontend acts as the client-facing layer responsible for:

- User authentication

- Session management
- Game interaction and media uploading
- API consumption

API Gateway

The API Gateway serves as the central entry point. Acts as a Load Balancer, redistributing traffic using RR algorithms. In future implementation, this model can be changed to request rerouting based on the HTTP method/verb.

Responsibilities:

- Request routing
- Authentication validation
- SSL termination
- Rate limiting:
    - DDOS mitigation support
- Monitoring:
    - Server health
    - API performance
    - Error tracking
- Metrics
    - CPU usage
    - Memory usage
    - Latency
    - Throughput
    - Logs

Enables **horizontal scaling**, easy to add one new node.

- No CDN - country wise solution
- No Redis cache - app doesn’t respond with large payloads

Backend Service

The backend is responsible for actually handling the requests forwarded by the gateway. Its primary concerns are:

- User management
- Authentication and Authorization
- Matchmaking
- Game library and media synchronization

Database

Scaling methods:

- Read replicas
- Query optimization
- Connection pooling

The database layer stores:

- User profiles
- Challenge history
- Game entries and media urls

Database replication:

- Improves availability
- Increases read throughput
- Provides failover support
- Reduces downtime

Replication strategy:

- Primary-write node
- Multiple read replicas
- Automated failover handling
- Designated cronjob for database sync

Buckets

Cloud storage buckets are used for:

- Media uploads
- Replay storage
- Game assets
- Temporary session files

Security

API Security

- HTTPS enforcement
- Request validation
- Input sanitization
- Rate limiting

Infrastructure Security

- Firewall policies
- IAM role management
- Private networking
- Secret management

User Security

- Password hashing

APIs

Our clients benefit from clearly documented REST API provided below. All system resources define the CRUD lifecycle operations. Each one is mapped to a specific HTTP verb according to:

- Create - POST
- Read - GET
- Update - PUT

- Delete - DELETE

Resource wise:

- Account/Profile Management:
    - Endpoint: /api/users
    - Methods: POST, GET, DELETE
- User Challenges:
    - Endpoint: /api/challenges
    - Methods: POST, GET, PUT, DELETE
- Game Entry:
    - Endpoint: /api/games
    - Methods: POST, GET, PUT, DELETE
- Game Media:
    - Endpoint: /api/games/{gameId}/media/{mediaUuid}
    - Method: POST, GET, PUT, DELETE

References

References

Course page:https://edu.info.uaic.ro/cloud-computing/coursepractical-works.html
Project requirements:
https://docs.google.com/document/d/17TAtmLlngE9MXDXY2xGp8Sacho4cR5-6
Project Organization:
https://docs.google.com/document/d/1atpfPoseAm0uaYea5z0wizxk-481fz4-kgW8IyJyHks/edit?tab=t.yow97k1a1otc
Personal:
https://www.google.com/url?q=https://github.com/InAndOut-Stack/.github/blob/main/doc/solution.md
Database schema: https://dbdiagram.io/d/Cloud-Homework-69c933dc78c6c4bc7a92a5fd