TQS: Product specification report

Filipe Viseu [119192], Duarte Branco [119253], Samuel Vinhas [119405], André Ferreira [119061]
v2025-12-14

1 Introduction

1.1 Overview of the project

This document specifies the requirements and architecture of our project, GameRent, a peer-to-peer rental platform designed to challenge the traditional consumption in the gaming sector. Developed within the scope of the Software Quality Systems (TQS) course, this project aims to implement a medium-sized software solution while applying Software Quality Assurance (SQA) principles, DevOps practices (CI/CD), and Agile methodologies.
GameRent allows users to list consoles, games, and accessories and book items from other users for specific periods at affordable rates. The solution creates a "win-win" ecosystem for gamers and hardware owners.

1.2 Project limitations and know issues

Payment Processing: The platform integrates with a payment gateway structure; however, actual financial transactions are simulated using Stripe Test Mode.

External Data Dependency: The catalog relies on the IGDB API for game metadata. Availability and rate limits of this external service may impact the "Item Discovery" features.

Geographic Scope: For the MVP, we didn't use a location system, we could have made it but we would have to sacrifice other things that were, in our opinion, more important.

1.3 References and resources

IGDB API: Used for fetching game and console metadata.
Stripe: Used to simulate payments
Spring Boot: The core backend framework used for service implementation.
React: Used for the frontend user interface implementation.
PostgreSQL: Relational database for persistent storage.
Testing Tools: Cucumber (BDD), Playwright (E2E), K6 (Load Testing), JUnit (Unit), and SonarQube (Static Analysis).

2 Product concept and requirements

2.1 Vision statement

GameRent, as we said, is a digital marketplace for peer-to-peer equipment rentals, specifically tailored for the gaming community. The platform solves the inefficiency of expensive gaming hardware sitting idle by connecting owners with gamers who need temporary access.

Key features include seamless item discovery via the IGDB catalog, secure booking management, a dual-dashboard system for users who act as both renters and owners, and a reputation system to build trust. Unlike generic marketplaces (ex.: OLX), GameRent manages the specific scheduling and availability logic required for rentals.

Everyone in the group liked gaming and we thought, why not do the project on this, since we understand about the theme and we can easily think about interesting things that we can add.

2.2 Personas and scenarios

The following personas represent the key user segments for GameRent:

Persona	Description	Goals	Pain Points
Miguel, 21 (Renter)	University student who loves gaming but can't afford to buy all consoles and games.	Easily find available consoles and games, rent safely and affordably, track rental dates.	Difficult to find trustworthy rentals; unclear item conditions; fear of losing money.
Liliana, 28 (Owner)	Game enthusiast with multiple consoles and accessories collecting dust.	Earn extra money by renting out her PlayStations and games, manage bookings, and communicate with renters.	Managing schedules manually, uncertainty about item condition after rental.
João, 32 (Platform Admin)	Works for GameRent, ensuring smooth operations, trust, and compliance.	Monitor transactions, users, and booking metrics; manage user verification and resolve disputes.	Lack of visibility on system metrics and user issues.
Andreia, 25 (Hybrid User)	Casual gamer who sometimes rents and sometimes lends her own games.	Flexibly switch roles between renter and owner, manage both dashboards.	Confusing role separation in typical marketplaces.

Scenarios:

1. Search and Book (Miguel): Miguel searches for "PS5" using distance filters. He reviews item photos and owner ratings, selects rental dates for the weekend, and confirms the booking.

2. Listing Management (Liliana): Liliana registers her PS4 Pro, sets the daily price, and manages incoming requests via the Dashboard. She wants an app that can easily detect if her item is already rented on a specific date and that doesn't allow other users to rent it on that date.

3. Dispute Resolution (João): A renter returns a damaged console. Liliana reports the issue. João (Admin) reviews the evidence and chat history to mediate the dispute and decide on compensation.

2.3 Project epics and priorities

The development is divided into the following Epics to deliver the MVP:

Item Discovery & Catalog: browse and search items, filter by category, view detailed item descriptions with photos, (integration with IGDB).

Booking & Scheduling: book items for specific dates, prevent double bookings, manage booking requests, calendar integration

Item Management: CRUD operations, register new items with details and photos, manage maintenance schedules, track item condition.

Payment and Transactions: simulated secure payments.

Reviews and Ratings: rate and review items, rate and review users, build reputation scores, report issues.

Platform administration: monitor platform operations such as monthly earnings, total users, active reports, etc.

3 Domain model

The domain model of **GameRent** facilitates a Peer-to-Peer rental ecosystem. It is centered around the interaction between **Users** (Renters/Owners) and **Items**, managed through **Bookings**, and supported by communication (**Chats/Messages**) and quality assurance (**Reviews**, **Disputes**).

User: Represents the actors in the system (Renters, Owners, Admins). A User can possess multiple Items and make multiple BookingRequests.

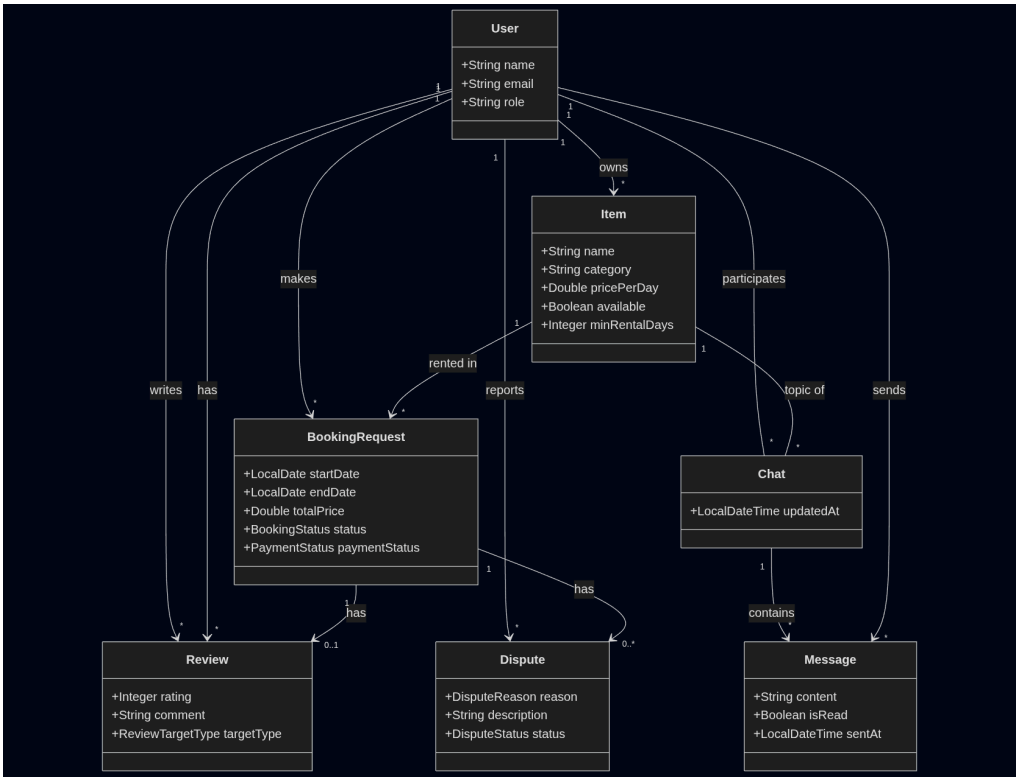
Item: A physical good (Console, Game) listed by an Owner. It contains pricing, description, and availability status.

BookingRequest: Represents a transaction where a User rents an Item for a specific date range. It holds the state of the rental (e.g., PENDING, APPROVED, CANCELED).

Review: A qualitative assessment linked to a Booking. Users can review the Item they rented, providing a rating (1-5 stars) and a comment.

Chat & Message: Facilitates communication. A Chat is unique to a Renter-Item pair, containing multiple Messages exchanged between the Renter and the Item's Owner.

Dispute: A formal complaint linked to a specific Booking, raised by a User regarding issues like damage or non-delivery, requiring Admin intervention.



4 Architecture notebook

4.1 Key requirements and constrains

Tech Stack: We are required to use Spring Boot (Java) for the backend and React for the frontend, communicating via a REST API.

Data Persistence: PostgreSQL is required for relational data storage.

External Integration: Our system relies on IGDB for game data and Stripe for payments. Since we don't control these services, our architecture needs to handle them carefully. We isolate these integrations into specific services so that if the external API is slow or down, it doesn't break our core application logic.

SQA & DevOps: The architecture must support TDD/BDD, automated CI/CD pipelines.

Deployment: All components must be containerized via Docker for consistent deployment across environments.

4.2 Architecture view

Layered Architecture utilizing a RESTful API communication pattern:

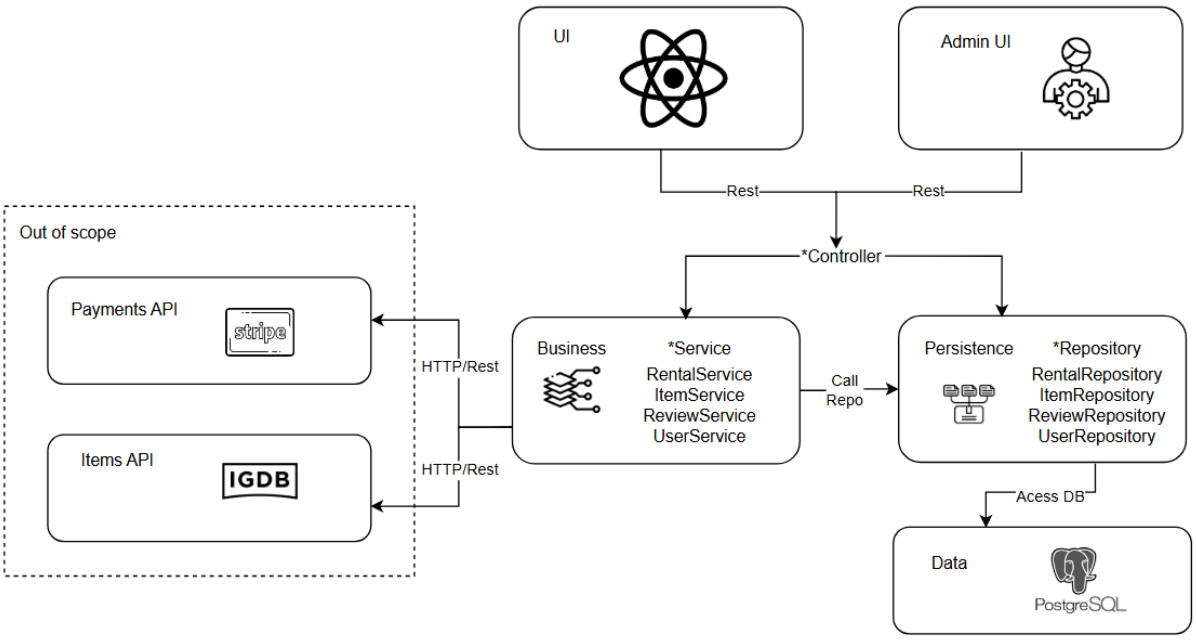
Presentation Layer (UI): A React-based Single Page Application (SPA) that consumes the backend API.

Controller Layer: Spring Boot REST Controllers that handle HTTP requests and map them to business logic.

Business Logic Layer (Service): Contains the core domain logic (e.g., preventing double bookings, calculating prices). It interacts with external services (IGDB, Stripe) via HTTP clients.

Persistence Layer (Repository): Uses Spring Data JPA to interact with the PostgreSQL database.

Data Layer (PostgreSQL): Stores all the system data



4.3 Deployment view (production configuration)

The production environment is containerized using Docker.

Database Container: Runs the PostgreSQL image.
Backend Container: Runs the Spring Boot application.
Frontend Container: Serves the React build.
Orchestration: Docker Compose is used to manage the multi-container setup, ensuring network communication between the backend and the database

5 API for developers

The API is structured into the following primary resource collections:

Authentication (/api/auth)
Manages user identity and session lifecycle. Unlike standard resources, this collection handles the actions of registration, login, and logout. It also provides the /me endpoint for the client to retrieve the current security context.

Items (/api/items)
Represents the catalog of rentable items.
Supports retrieving paginated lists (/catalog), searching by category or keywords, and CRUD operations for Owners to manage their inventory.

Bookings (/api/bookings)
Allow creating new requests, filtering bookings by specific Item or User, and updating the booking status (e.g., APPROVED, CANCELLED).

Users (/api/users)
Provides access to public user profiles and user management. This is distinct from /auth as it deals with the User entity data rather than the session state.

Chats & Messages (/api/chats)
Facilitates communication resources. A Chat resource aggregates Message sub-resources. Clients can retrieve conversation history or post new messages to specific chat IDs.

Reviews (/api/reviews)
Manages the feedback ecosystem. Reviews are linked to specific Bookings but can be retrieved based on the target Item or User profile.

Disputes (/api/disputes)
Resources for handling conflict resolution. Allows users to submit claims regarding specific bookings and allows Admins to intervene.

Payments (/api/payments)
Handles the creation of checkout sessions and payment confirmation. This resource acts as a bridge between the core domain and the Stripe payment gateway.

Admin (/api/admin)
A restricted resource collection providing system-wide metrics (User count, Booking stats, Revenue) for dashboard visualization.

Integrations (/api/igdb)
A utility resource that proxies requests to the IGDB (Internet Game Database) API, allowing users to search for standardized game metadata when listing items.