TQS: Product specification report

Daniel Martins [115868],
Tiago Mendes [119773],
Catia Lopes[119087],
Diogo Nascimento[120031]
v2025-12-19

1 Introduction

1.1 Overview of the project

BitSwap is a videogame rental marketplace that connect renters who want to save money and have short-term access to videogames and item owners who want to monetize from renting their videogames. The platform provides search, booking, payment and dashboards tailored for renters, owners and administrators.

1.2 Project limitations and know issues

Key risks and suggested mitigations:

- Booking conflicts (technical): strong calendar validation, optimistic locking and conflict detection in the booking module.

- Payment flow failures: use sandbox testing, idempotent operations and server-side verification of webhook events.
- Fraud / low listing quality: owner verification steps, photo-based QA and admin moderation workflows.
- Scalability if adoption grows: design stateless services, containerised deployment, and CDN for assets.

Operational risks: late returns, disputes — mitigate with clear policies, deposit or hold mechanisms, and an escalation workflow.

Market risks: competition from subscription services; mitigate by focusing on niche markets (retro games, collectors, local community) and value-added UX.

1.3 References and resources

- User stories: `./UserStories.md`
- Personas: `../It0/Personas.md`
- System architecture: `./SystemArchitecture.md`
- CI and pipeline: `./CIPipeline.md`
- SQA and tools: `./SQETools.md`

2 Product concept and requirements

2.1 Vision statement

Core features — highest priority:

- Renter services: search & filter, detailed game pages, booking for dates, simulated payments, renter dashboard for active/past rentals.
- Owner services: register & list games, manage availability/pricing, approve/reject bookings, owner dashboard with rentals & revenue.
- User management: register, login, role-based access (renter / owner / admin).
- Admin backoffice: manage users/listings, monitor suspicious activity, platform KPIs and basic moderation tools.

Extended features — lower priority:

- Recommendations engine (AI-powered)
- Real-time messaging between renters & owners
- Push / email notifications
- Photo-based QA checklists

Technology stack:

- Frontend: *HTML, JavaScript, CSS*
- Backend:

Language: JavaScript
Framework: Spring
Build Tool: Maven
- Database: *PostgreSQL*

2.2 Personas and scenarios

Representative personas are kept in `../It0/Personas.md`.

- Hannah Wilson — Product Owner
- Tom Schmidt — Renter
- Ronan Connsworth — Renter
- Dawan Houtcheques — Admin

Representative scenarios are kept in `../It0/Main Scenarios.md`.

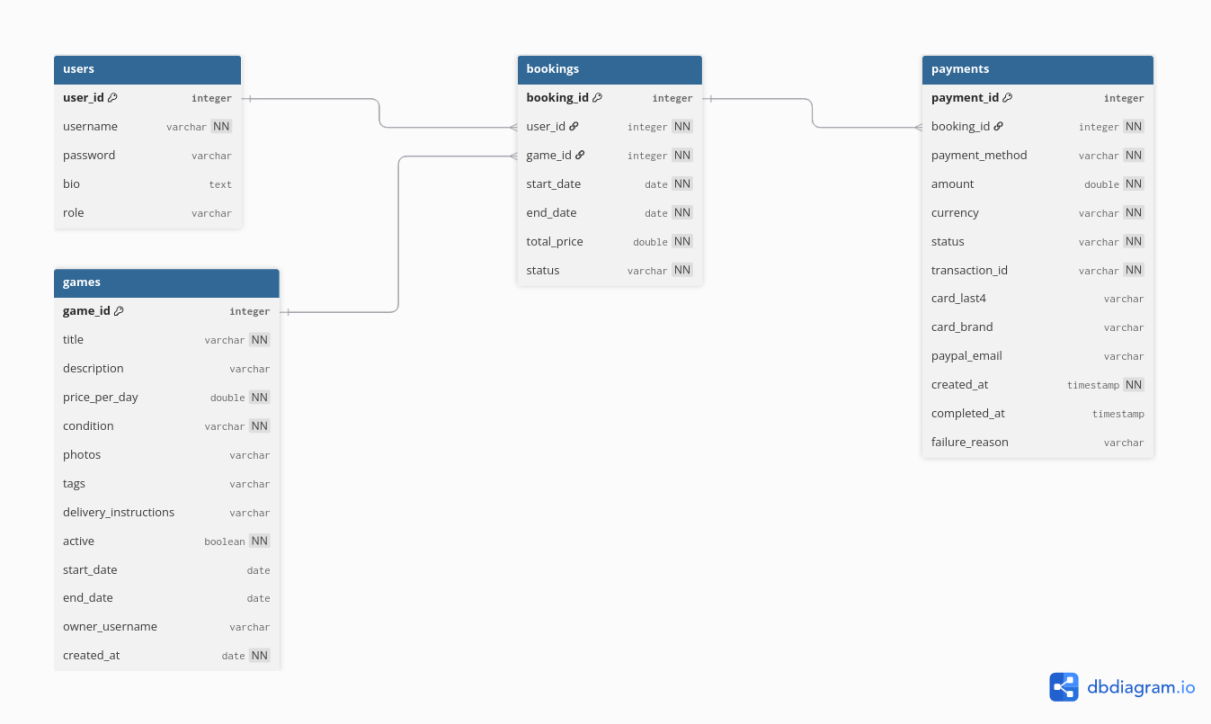
The full set of user stories used to drive development is maintained in `./UserStories.md`. Below are a few high-priority examples (extracted):

• User Story 1.1 - Add a videogame listing
As a user, I want to be able to add a listing for a videogame (including the title, a description, the platform, its condition and some images) so that other users can see it and rent it.
• User Story 2.4 - View purchase history as a User
As a user, I want to be able to view my rental and payment history, so I can keep track of my past transactions and expenses.
• User Story 3.1 - Game search
As a user, I want to be able to search for game listings and filter them to my needs (by platform, genre, price range, owner and age rating).
• User Story 6.5 - Manage flagged content
As an admin, I want to be able to see and take action on flagged listings, reviews, or messages, so I can remove harmful or inappropriate material.

2.3 Project epics and priorities

Representative epics are kept in `../It0/Epics.md`.

3 Domain model

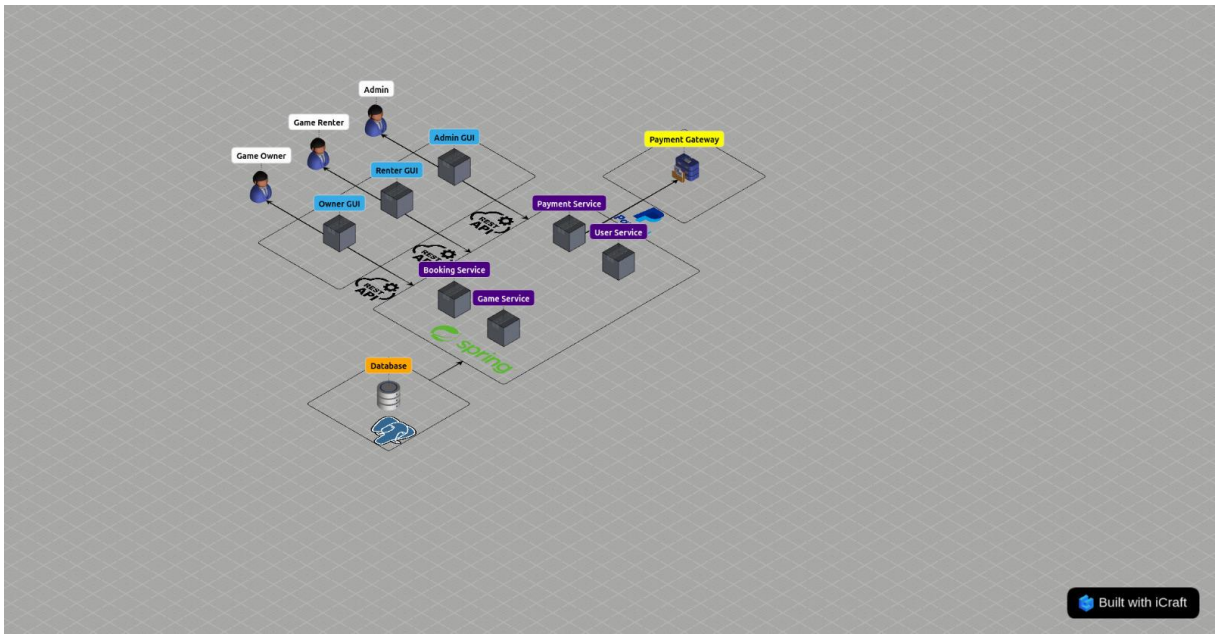


4 Architecture notebook

4.1 Key requirements and constrains

System Constraints

- **Web-based access:** The system must be available primarily through a modern web browser.
- **Database dependency:** The system relies on a relational database (PostgreSQL), which must support transactional consistency for bookings and payments.
- **External integrations:**
  - Payment systems (PayPal).
- **High reliability:** Rental and payment workflows must function correctly even under moderate to high user load.



4.2 Architecture view

Architecture style: microservices (service-oriented)

- **API Gateway** (HTTP/REST)
  - Single entry point for web clients — routes requests to services, handles API versioning.
- **Web App (GUI)**
  - Frontend that interacts with the API Gateway. Split into 3 sub-categories: renter,admin and owner.
- **Payment Service**
  - Service managing payment processing, retrieval, and refunds for bookings.
- **Game Service**
  - Service handling creation, retrieval, updating, and deletion of game listings.
- **User Service**
  - Service managing user creation, retrieval, update, deletion, and password validation.
- **Booking Service**
  - Service handling creation, retrieval, and status management of game bookings.
- **Payment Gateway**
  - Controller handling payment processing, retrieval, refunds, and transaction management for bookings.

4.3 Deployment view (production configuration)

The BitSwap platform is deployed using a **container-based, cloud-ready architecture**, designed to support scalability, reliability, and operational simplicity.

- **Production Environment Overview**

The production environment consists of multiple containerized services deployed in an isolated network, following the architecture defined in the Architecture Notebook.

- **Web Application (Frontend)**
  - Deployed as a containerized web application

    - Communicates exclusively with the API Gateway over HTTPS
    - Static assets are cached and served efficiently
- **API Gateway**
    - Single entry point for all client requests
    - Routes requests to backend services
    - Enforces authentication, authorization, and request validation
    - Handles API versioning and basic rate limiting
- **Backend Microservices**
    - Each core service (User & Listing, Booking, Payment, Search) runs in its own container
    - Services are stateless, enabling horizontal scaling
    - Communication is performed over RESTful HTTP APIs
- **Databases**
    - Primary relational database (MySQL/PostgreSQL) deployed as a managed service
    - Ensures transactional consistency for booking and payment operations
    - Backups and replication are configured to guarantee data durability
- **Message Broker**
    - Handles asynchronous communication
    - Improves system resilience and responsiveness
- **External Services**
    - Payment gateways operate in sandbox mode
    - Email and SMS providers are integrated via secure APIs
- **Deployment Characteristics**
- **Container Orchestration:** Services are deployed using container orchestration mechanisms (e.g., Docker Compose).
- **Environment Configuration:** Sensitive data (API keys, database credentials) is injected through environment variables and secrets management.
- **Scalability:** Stateless services allow independent scaling based on load.
- **Observability:** Logging and metrics are centralized for monitoring and troubleshooting.

This deployment view ensures that the production configuration remains consistent, reproducible, and aligned with non-functional requirements such as availability, performance, and security.

5 API for developers

BitSwap exposes a **RESTful API** that supports the web frontend and internal services.

- **Key Characteristics:**
    - JSON-based, stateless communication
    - Role-Based Access Control (Renter, Owner, Admin)
    - Secure hashed authentication
- **Main API Domains:**
    - **Authentication & Users:** registration, login, profile management
    - **Listings:** create, update, view, and manage videogame listings
    - **Search:** keyword and filtered search (platform, genre, price, rating)
    - **Bookings & Rentals:** booking lifecycle, availability checks, rental history
    - **Payments:** payment initiation, confirmation, and transaction records
    - **Admin & Moderation:** user/listing management and flagged content handling
- **Documentation & Testing:**
    - API documentation is generated using **OpenAPI/Swagger**
    - Automated API and integration tests are executed in the CI pipeline to ensure contract stability