

2 TQS: Product specification report

Ana Ribeiro [119876], Gonçalo Almeida [119792], João Barreira [120054], Rodrigo Santos [119198]
 v2025-12-16

1	Introduction	1
1.1	Overview of the project	1
1.2	Known limitations	2
2	Product concept and requirements	2
2.1	Vision statement	2
2.2	Personas and scenarios	3
2.3	Project epics and priorities	8
3	Domain model	12
4	Architecture notebook	13
4.1	Key requirements and constraints	13
4.2	Architecture view	14
4.3	Deployment view	15
5	API for developers	17

1 Introduction

1.1 Overview of the project

This project assignment for the Teste e Qualidade de Software (TQS) course serves as a practical application of the knowledge and concepts acquired throughout the semester, with emphasis on Agile development methods, Software Quality Assurance (SQA), Continuous Integration and Continuous Delivery (CI/CD) approaches.

Party Share aims to solve the problem of expensive, wasteful, and inefficient access to party equipment by offering a convenient peer-to-peer rental marketplace. For owners, it offers a simple way to generate extra income from their stored equipment; for renters, it provides an intuitive discovery process to find exactly what they need.

Ultimately, the high-level goal of Party Share is to make the rental process smooth, trustworthy, and transparent, promoting a more sustainable model that reduces waste and increases access to party resources within a unified platform.

1.2 Project limitations and known issues

The development of PartyShare followed an Agile methodology, prioritizing a Minimum Viable Product (MVP) that could be thoroughly tested and deployed within the timeframe given. Consequently, certain features originally envisioned in the domain model were deprioritized or simplified to ensure the robustness of the core rental workflow.

1.2.1 Functional Limitation

Due to time constraints and the pedagogical focus on Quality Assurance (QA) rather than feature quantity, the following functionalities were not fully implemented:

- **Real-time Notifications:** While the first plans featured a notification system, the implementation of WebSockets for real-time alerts (e.g., "New Booking Request" popping up instantly) was postponed. Currently, users must refresh the dashboard to see status updates.

1.2.2 Known Issues

The following minor issues have been identified during the testing phase and are documented:

- **Mobile Responsiveness:** While the application is web-based, the UI optimization focused primarily on Desktop viewports. Some layouts may not be fully optimized for mobile screens.

2 Product concept and requirements

2.1 Vision statement

Party Share aims to solve the problem of expensive, wasteful, and inefficient access to party equipment by offering a peer-to-peer rental marketplace dedicated to event items.

The system focus is making it easy for renters to discover and rent needed items, while decreasing the usability friction for owners, to make them generate income from equipment they already have.

The high-level goal of Party Share is to reduce waste, increase access and bring transparency to the rental process.

To achieve this, the following key features are:

- **Item Discover:** A searchable catalog of party equipment, with filters and favorites;
- **Booking & Availability Management:** Booking requests with conflict-free reservations;
- **Private Communication:** Message chat between Renter and Owner
- **Role-Based Access:** Distinct dashboards for Owners, Renters and Admins
- **Administration Tools:** User & Categories management, Platform Monitoring

The solution differentiates itself by focusing specifically on event/party equipment, offering useful structured item categories to facilitate the event planners' and casual party organizers' job.

2.2 Personas and scenarios

2.2.1 Personas

Paulo, 42 - Accountant [Casual Renter]

Paulo lives in a quiet suburban neighborhood and enjoys hosting casual gatherings, family barbecues, and birthday parties in his backyard. He likes making these moments special, but as someone very mindful of budgeting, he avoids buying equipment that he knows he will use only once or twice a year. Although comfortable with technology, Paulo loses patience with platforms that feel confusing, unreliable, or that bury essential information behind too many steps.

Pain Points

- Buying party items feels financially wasteful due to low usage frequency.
- Local stores often have limited stock and offer little item variety.
- Comparing prices and availability across different websites / stores is time-consuming.
- Worries about renting damaged items or dealing with unreliable owners.

Motivations

- Spend less by renting instead of purchasing rarely-used items.
- Find all equipment for a party in a single, well-organized platform.
- Easily compare items based on price, distance, availability, and category.
- Plan events quickly thanks to intuitive search and filtering.

Sofia, 25 - Pastry Artist & Event Caterer [Frequent Renter]

Sofia is a talented pastry artist who turned her passion for baking into a small business. She creates custom desserts for birthdays, weddings, and corporate events, and also organizes baking workshops. Although highly creative and skilled, she lacks the storage space and financial capacity to own all the professional furniture and equipment she needs for different events.

Pain Points

- She frequently needs the same (or similar) items she cannot justify buying.
- Seasonal demand means equipment bought would often stay unused for long periods.
- Renting through traditional companies is expensive and rigid (minimum rents, long-term commitments).
- Last-minute client requests make it difficult to plan with fixed or rigid providers.

Motivations

- Access to professional equipment whenever needed, without long-term commitments.
- Quickly verify item availability for specific event dates to ensure smooth planning.
- Save frequently rented items to a favorites list for faster repeat bookings.
- Rely on transparent pricing, trustworthy ratings and detailed item photos.
- Communicate directly with item owners through in-app messaging.

Ricardo, 38 - Amateur DJ & Music Producer [Casual Owner]

Ricardo is a music enthusiast who has invested significantly in high-quality sound equipment for his own projects and occasional gigs. However, as he works full-time in HR, his expensive gear (mixing deck, speakers) spends the majority of the month stored in his garage. He wants to monetize the investment he made, however he is hesitant to lend his equipment to strangers, fearing damages to its collection.

Pain Points

- He is frustrated by the fact that the equipment is idle 90% of the time, feeling it represents a financial loss.
- Lack of time to manage phone calls and / or emails to coordinate rentals.
- Difficulty in advertising his gear beyond his family and friends circle.

Motivations

- Generate passive income to amortize the cost of expensive equipment.
- Avoid the exhaustion of negotiating prices typical of general classified ad sites.
- Manage equipment availability to ensure he can still use it for his own gigs.

Helena, 47 - Small Business Owner [Professional Owner]

Helena runs a small event decoration business. She owns a considerable inventory of chairs, tables and decorative arches. While she is very busy during peak season (summer weddings), there are months when her stock sits completely still in the warehouse. She looks for a way to rent out her materials during off-peak times for smaller private events without increasing her advertising or sales team costs.

Pain Points

- High warehouse costs, even when materials are not generating revenue.
- Lack of time to manage phone calls and / or emails to coordinate rentals.
- Difficulty in advertising his gear beyond his family and friends circle.

Motivations

- Maximize inventory utilization, reducing seasonal phases without use.
- Direct communication with clients to avoid misunderstandings.
- Difficulty reaching the "small home party" market share that doesn't usually hire full service decorators.

2.2.2 Scenarios

2.2.2.1 Scenario 1: Paulo Rents an Item for a Weekend Barbecue

Paulo is organizing a small barbecue for his family and needs a portable speaker to set the mood. He opens Party Share, searches for sound equipment, and compares items based on price, ratings, and distance. After reading reviews, he messages the owner to confirm pickup time and place. Satisfied with the details, Paulo proceeds with the booking, choosing the preferred pickup and return dates. He makes a lower price offer, which the owner replies shortly after. This response can either accept the offer or propose a counteroffer. Satisfied with the details and the new price, Paulo proceeds with the booking.

Actions:

- Opens the Party Share platform
- Searches for sound equipment
- Applies filters to narrow down options
- Identifies a suitable speaker
- Checks item details and photos
- Checks ratings and reads reviews
- Inputs preferred rental dates
- Sends a message to the item owner inquiring about pickup details
- Makes a different price offer (and waits for the owner's response)
- Books the item

2.2.2.2 Scenario 2 – Paulo Rates the Item and the Owner After Returning It

After the barbecue, Paulo returns the speaker and is satisfied with both the quality of the item and the responsiveness of the owner. Wanting to help other renters, he logs in to PartyShare, visits his past bookings, and leaves a rating for both the item and the owner. He adds a short comment highlighting the good condition of the speaker and the smooth communication. With this, Paulo closes the renting cycle feeling confident to use the platform again.

Actions:

- Logs into PartyShare
- Opens the “Past Rentals” section
- Selects the completed booking
- Rates the item
- Rates the owner
- Submits a written comment

2.2.2.3 Scenario 3 – Sofia Rents an Item from Her Favorites List

Sofia is preparing a dessert workshop and needs extra folding tables. She opens PartyShare and navigates to her favorites list, where she keeps the items she rents most often. She selects the folding tables she trusts, since she already knows the owner and the item's quality. She checks the availability for the workshop date and confirms the rental.

Actions:

- Opens PartyShare on her phone
- Navigates to the favorites tab
- Selects the frequently rented item
- Checks date availability
- Confirms the booking

2.2.2.4 Scenario 4 – Sofia Cancels a Rental Due to Weather Issues

Sofia has an outdoor catering event scheduled, but unexpected heavy rain forces her client to cancel. To avoid unnecessary expenses, she opens her upcoming rentals in the app, selects the rented lighting equipment for that event, and submits a cancellation request, after reviewing the cancellation policy. She may also contact the owner through the chat to apologize for the inconvenience and explain the situation.

Actions:

- Opens the upcoming rentals section
- Selects the affected booking
- Reviews the cancellation policy
- Submits the cancellation request
- Sends a message to the owner to explain the situation

2.2.2.5 Scenario 5 – Ricardo Lists a New Item for Renting

Ricardo owns a sound system in perfect condition, but rarely utilized. Consequently, he decides to list it on Party Share to start earning money with it. To ensure clarity and avoid misunderstandings for potential renters, he describes the item with the most detail possible, sets a competitive renting price and includes some visual media of the product.

Actions:

- Creates an Owner account on PartyShare
- Navigates to Owner Dashboard
- Selects the “Add new Item” option
- Describes the product
- Sets the daily renting price
- Select the most appropriate item category
- Uploads a photo of the product
- Submits the listing form
- Verifies if the item appears under “My Rental Items” on the Owner Dashboard

2.2.2.6 Scenario 6 – Ricardo Accepts a Booking Request and Coordinates Pickup

Ricardo verifies that someone wants to rent his speakers for the weekend. Ricardo checks his personal calendar to ensure he doesn't need them. He refuses the price offer proposed and makes a reasonable counter-offer, which is accepted by the future renter. After that, he uses the integrated chat feature to agree on the time and location for the item pickup.

Actions:

- Opens the new booking requests section
- Checks the request conditions specified by the renter
- Sends a counter offer for the requested price (and awaits acceptance)
- Use the chat feature to message the renter and confirm pickup availability details.

2.2.2.7 Scenario 7 – Helena Disables an Item Listing for a Season

Anticipating a heavy workload over the summer, Helena has decided to temporarily disable the listing for an item she won't be able to rent during that period. This action removes the listing from search results, preventing new bookings. The listing can be quickly reactivated with a single click when she is ready to rent the item again.

Actions:

- Navigates to the Owner Dashboard
- Locates the specific listing within the “My Items” list
- Selects the “disable” command

2.2.2.8 Scenario 8 – Helena Reports an Issue with a Returned Item

After a rental, a customer returns a table with a notorious bump and several scratches that weren't there before. She accesses the completed booking, reports the incident through the platform, and leaves an honest review for the user, mentioning the damage and giving him a low rating.

Actions:

- Navigates to the Owner Dashboard
- Locates the specific listing within the “Past Rentals” list
- Submits the “Report Damage” form, explaining the situation
- Selects the “Rate Renter” option and gives the respective rating and comment

Even though we had not defined a persona for the ‘admin’ actor, we listed some scenarios for its context.

2.2.2.9 Scenario 9 – Admin Monitors System Metrics and Expands Catalog

The System Admin logs into the administration area to check the platform's metrics and analytics. He reviews the main dashboard to see key metrics like total users, total bookings, total revenue and booking states. Wanting to spread the range of items available, the admin decides to add a new category to the system.

Actions:

- Views the Admin Dashboard to analyze system statistics
- Navigates to the Categories Management section
- Inputs the new category name and clicks on “Add Category”

2.2.2.10 Scenario 10 – Admin Resolves a Report and Deactivates a User

The System Admin checks the report page, and spotted a new user violation report. The Admin marked it as “In Progress” to show he started working on resolving the issue. After confirming the problem was legit, the Admin found the user in the database and went ahead and deactivated their platform account. Once that was done, the Admin closes the ticket.

Actions:

- Navigates to the Reports page and opens a reported marked as “New”
- Reads the report made by the user and updates the report state to “In Progress”
- Navigates to User Management and uses Search/Filter to locate the user involved
- Deactivates the user account (sets “Is Active” to false)
- Returns to the report and updates the state to “Closed”

2.3 Project epics and priorities

This subsection outlines the project plan for incremental implementation of the solution, detailing the functionalities to be delivered per epic and user stories across several iterations.

2.3.1 EPIC 1: Renter Experience

Goal: Enable users to discover items, manage bookings and interact with the platform as renters.

User Story 1.1 - Search and Filter Items:

As a renter I want to view the catalogue of items available to rent before deciding what to rent.

Acceptance Criteria:

- The user can apply filters for Category (e.g., "Lighting"), Price Range (e.g., €50–€200), Minimum Rating, and Location.
- The system displays a list of items matching all active filters.
- The search results update dynamically when filters are changed.

User Story 1.2 - View Item Details:

As a Renter, I want to view full item details before deciding to rent, so that I know exactly what I am booking.

Acceptance Criteria:

- Clicking an item opens a detailed view showing photos, description, daily price, and location.
- The details page displays specific rental rules and current availability.
- Reviews and ratings associated with the item are visible.

User Story 1.3 - Rent an Item:

As a Renter, I want to rent an item for specific dates so that I can secure it for my event.

Acceptance Criteria:

- The user can select valid start and end dates for the rental.
- The system prevents selection of dates that are already booked or blocked.
- The user can proceed to payment and receive confirmation upon success.

User Story 1.4 - View and Manage Bookings:

As a Renter, I want to view my active and past bookings so that I can track my scheduled rentals or cancel if necessary.

Acceptance Criteria:

- The user can navigate to a "Bookings" section to see a list of Active and Past rentals.
- The user can cancel a booking within the allowed policy timeframe, triggering a refund if applicable.
- Attempting to cancel outside the policy results in an error message.

User Story 1.5 - Save Favorite Items:

As a Renter, I want to save items as favorites so that I can easily revisit them later.

Acceptance Criteria:

- The user can click "Add to Favorites" to store an item.
- The user can click "Remove from Favorites" to delete an item from the list.
- The favorites list persists across user sessions.

User Story 1.6 - Rate Owner / Item:

As a Renter, I want to rate the owner or the item after using it so that I can share my experience with the community.

Acceptance Criteria:

- The user can submit a rating and comment for the Item or Owner after a rental is completed.
- The submitted rating is stored and updates the average score displayed on the platform.

User Story 1.7 - Message Owner:

As a Renter, I want to message the owner so that I can ask questions and discuss details before renting.

Acceptance Criteria:

- The user can compose and send a message to the item owner.
- The message is delivered and stored in a conversation history.

User Story 1.8 - Report Problems:

As a Renter, I want to report an issue related to my rental experience so that support can intervene.

Acceptance Criteria:

- The user can select a specific issue type (Item or Owner related) and submit a report.
- The system logs the report and creates a support ticket.

User Story 1.9 - User Access (Login/Register):

As a User, I want to register and log in so that I can access the platform features.

Acceptance Criteria:

- New users can register with email and password.
- Existing users can authenticate to access their account.
- Users can deactivate their own account if desired.

2.3.2 EPIC 2: Item Owner Experience

Goal: Enable owners to monetize their equipment by managing listings, booking requests, and protecting their assets.

User Story 2.1 - Add New Item:

As an Item Owner, I want to add new items to the platform so that renters can find and book my equipment for their events.

Acceptance Criteria:

- The user can select a specific issue type (Item or Owner related) and submit a report.
- The system logs the report and creates a support ticket.
- Authenticated owners can access the "Add New Item" form.
- The form validates mandatory fields - name, description, daily price, category, and location.
- Uploading at least one image is mandatory.
- The item is created with an "ACTIVE" status by default and appears in search results.

User Story 2.2 - Edit and Manage Inventory:

As an Item Owner, I want to edit, activate, or deactivate my items so that I can keep information accurate and control availability.

Acceptance Criteria:

- The item owner can edit item details; changes are saved immediately.
- The item owner can deactivate an item, hiding it from search results without deleting booking history.
- The item owner can reactivate a deactivated item.
- Only the owner of the item can perform these actions.

User Story 2.3 - View and Manage Booking Requests:

As an Item Owner, I want to view and manage rental requests so that I can accept, decline, or negotiate with renters.

Acceptance Criteria:

- Owner sees a list of pending requests with renter details, dates, and price.
- The item owner can Accept (status changes to ACCEPTED) or Decline (status changes to DECLINED) a request.
- The item owner can make a Counter-offer with a new price (status changes to COUNTER_OFFER and notifies renter).

User Story 2.4 - Owner Dashboard:

As an Owner, I want to see all my upcoming and past rentals so I can manage my activity.

Acceptance Criteria:

- Dashboard displays a summary of listed products and their status.
- Dashboard shows upcoming and past booking information.
- The item owner can view reviews and ratings received on their products.

User Story 2.5 - Rate Renter:

As an Owner, I want to rate the renter's behavior after a completed booking so that I can

warn or recommend them to other owners.

Acceptance Criteria:

- The item owner can submit a rating for a renter only after the booking status is completed.
- The renter's average score is updated upon submission.

User Story 2.6 - Item Returned Damaged:

As an Owner, I want to report damage when an item is returned in poor condition so that I can claim compensation or warn admins.

Acceptance Criteria:

- The item owner can submit a "Damage Report" only after the booking end date.
- The system stores the report and notifies the admin.

User Story 2.7 - Message Renter:

As an Owner, I want to message renters so that I can answer questions and negotiate details.

Acceptance Criteria:

- The item owner can reply to renter messages or initiate messages regarding a booking.
- Message history is maintained within the session context.

2.3.3 EPIC 3: Admin & Platform Management

Goal: Provide the necessary tools for system administrators to monitor platform health, manage users, and enforce rules.

User Story 3.1 - Search and Filter Users:

As an Admin, I want to search and filter users by registration date, name, or email so that I can quickly find specific accounts.

Acceptance Criteria:

- The system admin can input text to search for users by name or email.
- The system admin can apply filters (e.g., registration date) to the user list.
- The system displays users matching the criteria or a message if no user is found.

User Story 3.2 - View Detailed User Profiles:

As an Admin, I want to view detailed user profiles including rental history and verification status so that I can assess user trustworthiness.

Acceptance Criteria:

- Clicking a user in the list loads their full profile.
- The profile displays history as both Renter and Owner, along with ratings and verification status.
- The system admin can deactivate a user account from this view.

User Story 3.3 - View and Manage Reports:

As an Admin, I want to view and administrate report claims so that I can resolve disputes and enforce rules.

Acceptance Criteria:

- The system admin can view a list of reported claims.
- The system admin can filter reports by state (New, In Progress, Closed).
- The system admin can update the state of a report as it moves through the resolution workflow.

User Story 3.4 - Add New Categories:

As an Admin, I want to add new item categories so that owners can better classify their equipment.

Acceptance Criteria:

- The system admin can access a form to input a new category name.
- Upon submission, the category is added to the system and becomes selectable for Owners.

User Story 3.5 - View Platform Statistics:

As an Admin, I want to view platform statistics so that I can monitor the system's growth.

Acceptance Criteria:

- Admin dashboard displays relevant metrics such as total users, active items, and total bookings.

3 Domain model

The following domain model illustrates the logical structure of the PartyShare platform. It defines the essential business entities, their attributes, and the relationships necessary to support the core functionality of our peer-to-peer rental marketplace.

3.1 Domain Diagram

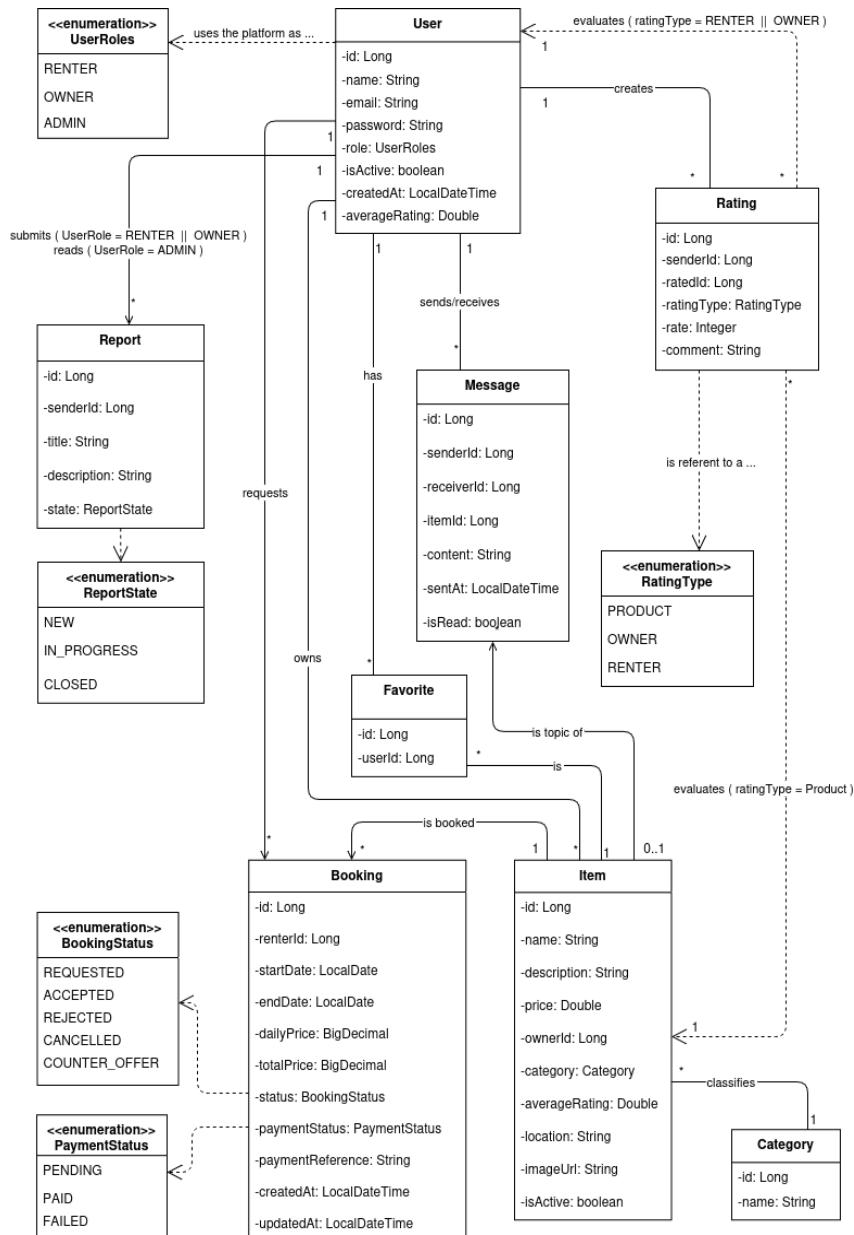


Image 1: Domain Diagram

3.2 Concept Definitions

The domain could be structured into four key areas:

User: A unified entity representing all system actors. The use of a UserRoles enumeration instead of separate subclasses simplifies the logic for shared services (authentication, profile management).

Inventory (Item, Category, Favorites): Manages the assets. Items are categorized and contain pricing/location logic. The Favorite entity links users to items, enabling a wishlist feature for future booking planning.

Booking: The transactional heart of the system. It links a renter to an item and uses strict state machines (BookingStatus, PaymentStatus) to track the rental progress from request to finalization.

Communication & Trust: Groups the features that build trust. Messages allow for pre-booking negotiation. Reports enable content moderation. Ratings provide a polymorphic review system (for both Items and Users) to establish community trust scores.

4 Architecture notebook

4.1 Key requirements and constraints

The architecture of PartyShare is shaped by functional expectations, quality attributes, and external system interactions. The platform must support a smooth rental experience while ensuring security, scalability, and reliable communication between users.

We recognized several essential architectural drivers and characteristics that need to be carefully addressed to ensure the system is designed and implemented effectively.

Through integration with external payment providers ([Stripe](#)), the application ensures secure and reliable financial transactions between users, isolating the complexity of sensitive data handling from the core platform.

Role-based access control (RBAC) will be implemented to manage permissions for different user roles, specifically Renters, Item Owners, and Administrators.

To ensure consistent deployment, the system's infrastructure is orchestrated using Docker Compose. The configuration packages the Spring Boot backend and the PostgreSQL database into separate, lightweight containers connected via a dedicated bridge network. This setup guarantees that the application logic and data persistence layers interact securely and run identically across development, CI/CD pipelines, and production environments, eliminating configuration drift.

4.2 Architecture view

4.2.1 Overview

PartyShare follows a Layered Architecture (N-Tier Architecture), a widely adopted architectural style that promotes separation of concerns by organizing the application into distinct horizontal layers. Each layer has specific responsibilities and communicates only with adjacent layers, ensuring loose coupling and high cohesion.

The application is designed as a monolithic server-side rendered web application using Spring Boot with Thymeleaf templates, complemented by a RESTful API for programmatic access.

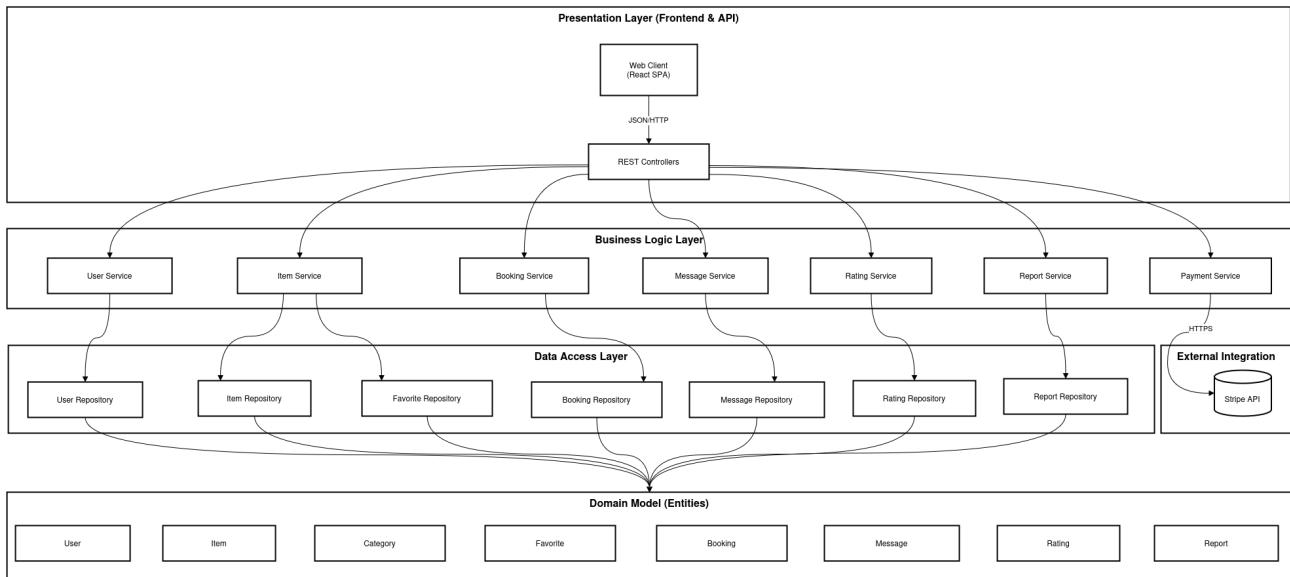


Image 2: Architecture Diagram

4.2.2 Layer Responsibilities

The table below lists every architecture layer's purpose on the project.

Layer	Responsibility	Components
Presentation	Handles HTTP requests, renders views, exposes REST endpoints	Controllers (Web + REST), Thymeleaf Templates
Business Logic	Implements business rules, orchestrates operations, enforces validation	Service classes
Data Access	Abstraction database operations, provides CRUD functionality	Repository interfaces (Spring Data JPA)
Domain Model	Defines core business entities and their relationships	Entity classes

Table 1: Architecture Layers, its Responsibilities and Components

4.2.3 Layer Interactions

As example of the layers / modules interaction, below there is a sequence diagram of the main system workflow: the “Renting an Item” process:

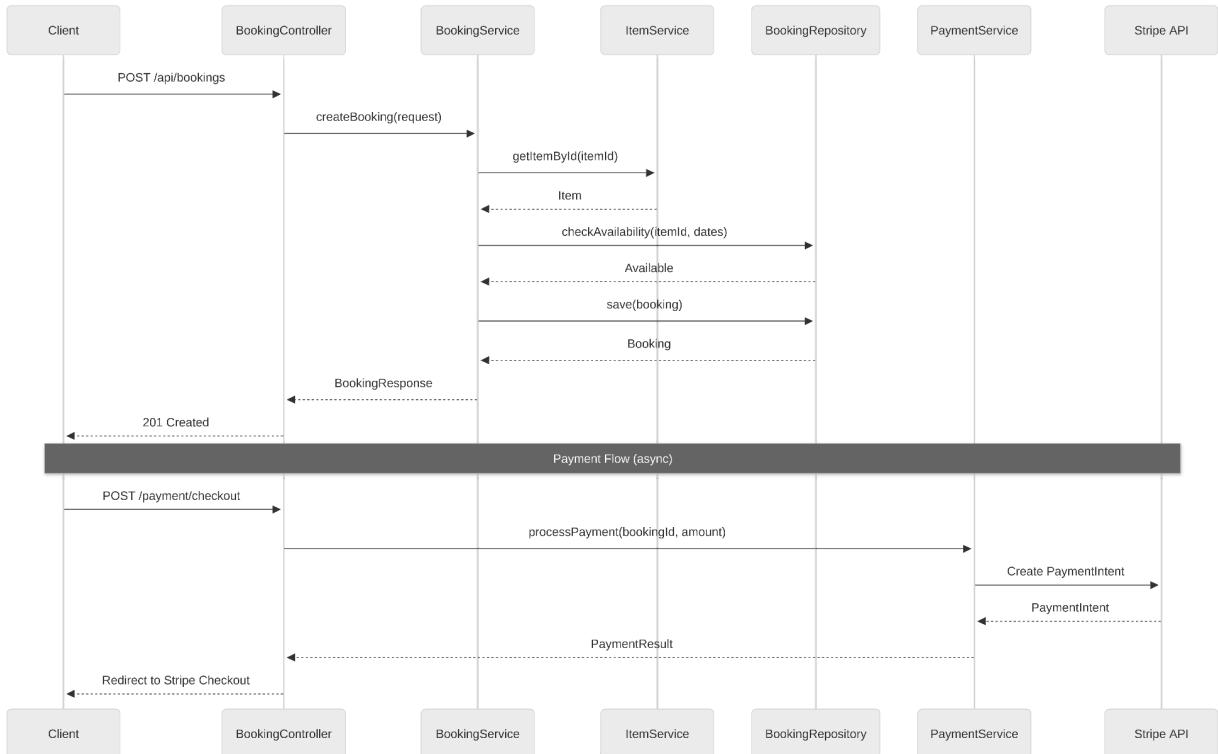


Image 3: Sequence Diagram showing up the Layer Interactions on the “Renting an Item” workflow

4.2.4 Integration with External Services

4.2.4.1 Payment Processing (Stripe)

The application integrates with Stripe for secure payment processing.

Integration Details:

- **Protocol:** HTTPS/REST with Stripe API
- **Authentication:** API keys via environment variables
- **PCI Compliance:** Stripe-hosted checkout (no card data on app servers)
- **Data Sync:** Local state updated on success callback;

4.3 Deployment view (production configuration)

To maintain consistency across development, testing, and production environments, the system is deployed using Docker containers, each encapsulating a distinct component of the application. These containers run on a virtual machine and communicate through an internal Docker network.

4.3.1 Backend Container (Spring Boot)

Purpose: Hosts the Spring Boot application with Thymeleaf templates.

Functionality: Processes all business logic, serves the web UI, communicates with the PostgreSQL database, and integrates with external APIs (Stripe for payments).

Endpoints: Exposes RESTful API and web pages over HTTP.

Port: Runs internally on port 8080, exposed externally to user access on port 80.

Health Check: Monitored via Spring Actuator (/actuator/health).

4.3.2 Database Container (PostgreSQL)

Purpose: Runs the PostgreSQL 15 database instance.

Functionality: Stores all persistent application data, including users, items, bookings, messages, wallets, and transactions.

Access: Not exposed externally — only accessible by the backend container for security.

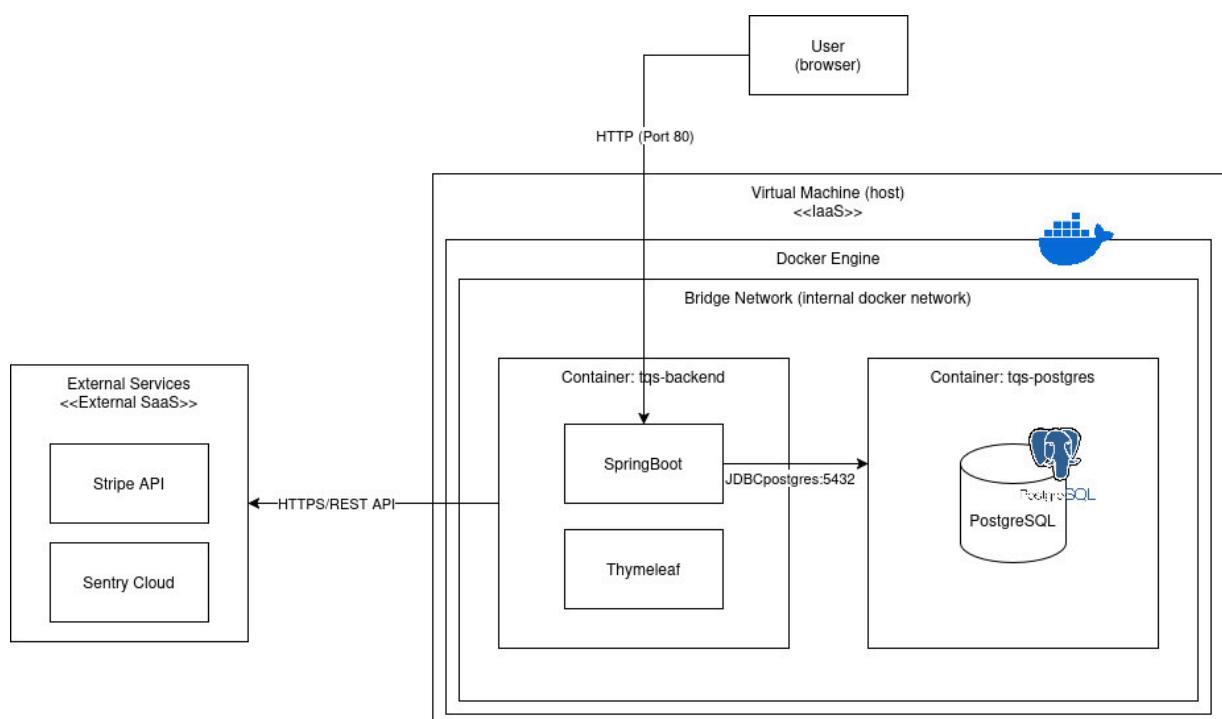
Data Persistence: Uses a Docker volume (postgres_data) to persist data across container restarts.

Migrations: Schema managed via Flyway migrations.

In the production environment, user requests arrive at port 80, are handled by the backend container, which queries the PostgreSQL database as needed and communicates with Stripe for payment operations. All internal communication uses the Docker bridge network, while external API calls use standard HTTPS.

Environment variables control all sensitive configuration, including database credentials (POSTGRES_USER, POSTGRES_PASSWORD), Stripe API keys, and the administrator password. These are loaded from a .env file and injected at container startup.

Together, these containers provide a clean separation of concerns, ensure environment security,



and simplify deployment and maintenance of the application.

Image 4: Deployment Diagram

5 API for developers

The PartyShare backend follows a modular RESTful architecture. Each controller manages a specific resource domain (e.g., Items, Bookings, Users), exposing endpoints for standard CRUD operations via HTTP methods (GET, POST, PUT, DELETE).

API documentation is automatically generated using Swagger (OpenAPI), ensuring synchronization with the source code and providing an interactive interface for testing endpoints directly within the application.

The design strictly adheres to REST best practices:

- **Semantic Paths:** Use of plural nouns for resources (e.g., /api/items, /api/bookings).
- **Standard Verbs:** HTTP methods clearly reflect the action performed.
- **Clear Responses:** Usage of standard HTTP status codes and consistent JSON structures.

Available on: <http://deti-tqs-08.ua.pt/swagger-ui/index.html>

Method	Endpoint	Action
User and Authentication		
POST	/api/users/register	Register User
POST	/api/users/login	Login User
GET	/api/users/search	Search Users
GET	/api/users/id/{id}	Get User By ID
GET	/api/users/email/{email}	Get User By Email
Administration		
POST	/api/admin/users/{id}/deactivate	Deactivate User
POST	/api/admin/users/{id}/activate	Activate User
GET	/api/admin/stats	Get Global Stats
GET	/api/admin/reports	Get All Reports
GET	/api/admin/dashboard-stats	Get Dashboard Stats
Owner Dashboard		
GET	/api/owner/dashboard/items	Get Owner Items
DELETE	/api/owner/dashboard/items/{id}	Delete Item
PATCH	/api/owner/dashboard/items/{id}/deactivate	Deactivate Item
PATCH	/api/owner/dashboard/items/{id}/activate	Activate Item
GET	/api/owner/dashboard/items/{id}/ratings	Get Item Ratings
POST	/api/owner/dashboard/bookings/{bookingId}/rate-renter	Rate Renter
GET	/api/owner/dashboard/bookings/{bookingId}/renter-rating	Get Renter Rating
POST	/api/owner/dashboard/bookings/{bookingId}/damage-report	Create Damage Report
Reports, Favourites and Rating		

POST	/api/reports/new	Create Report
GET	/api/reports/{id}	Get Report By ID
PUT	/api/reports/{id}/state	Update Report State
GET	/api/reports/state/{state}	Get Reports By State
GET	/api/reports/sender/{senderId}	Get Reports By Sender
GET	/api/favorites/{userId}	Get Favorites
POST	/api/favorites/{userId}/{itemId}	Add Favorite
DELETE	/api/favorites/{userId}/{itemId}	Remove Favorite
POST	/api/ratings/new	Create Rating
GET	/api/ratings/{id}	Get Rating By ID
DELETE	/api/ratings/{id}	Delete Rating
GET	/api/ratings/search	Search Ratings
Items / Categories / Bookings / Payments / Wallet		
GET	/api/items/search	Search Items
GET	/api/categories	Get All Categories
POST	/api/bookings	Create Booking
GET	/api/bookings/{id}	Get Booking
POST	/api/bookings/{id}/cancel	Cancel Booking
GET	/api/bookings/unavailable/{itemId}	Get Unavailable Dates
POST	/payment/create-checkout-session/{bookingId}	Create Checkout Session
POST	/api/wallet/withdraw	Withdraw Wallet Funds
GET	/api/wallet	Get all Wallets Info
GET	/api/wallet/transactions	Get all Wallets Transactions
Messaging		
GET	/api/messages	Get Messages
POST	/api/messages	Send Message
GET	/api/messages/conversations	Get Conversations List
GET	/api/messages/conversation/{otherUserId}	Get Conversation

Table 2: API endpoints list, divided into usage purposes.