TQS: Product specification report

Miguel Alexandre Moura Neto [119302]
Alexandre Filipe da Paula Andrade [119279]
Thiago Aurélio Pires Barbosa Vicente [121497]
João Pedro Silva Pereira [120010]

v2025-11-25

[This report is the main source of technical documentation on the project, clarifying the functional scope and architectural choices. Provide concise, but informative content, **allowing other software engineers to understand the product**.
Tips on the expected content placed along the document are meant to be removed!
You may use English or Portuguese; do not mix.]

1 Introduction

1.1 Overview of the project

<contextualize the objectives of this project assignment in the scope of the TQS course>
<introduce your application/product: brief overview of the solution concept. What is it good for? For whom? Introduce the name of the product if it has one>

The project assignment given in the TQS course is devised to develop a peer-to-peer equipment and gear rentals marketplace platform. Item owners will register their products for renters to browse and book, in order to give usage, thus providing passive income from otherwise idle assets.

In terms of the project's scope of equipment and gear is comprised of **musical instruments** available for users to rent and use for whichever context they are required, ranging from string instruments to wind instruments and percussion, as well as **physical, registered sheet music samples** which are used for variably-sized groups of musicians (that is, *ensembles*) and different music styles.

Such a solution allows peers to no longer resort to several social media and similar platforms for renting and sharing idle assets, which became a nuisance due to having to stay up to date with multiple sources to ensure income or find the desired asset. Other advantages inherently include feedback among peers, using reputation scores and asset quality assurance, as well as controlling *renting* (booking will be considered more appropriate for the remaining report, due to the nature of the project's guidelines and concept overall). Differentiating features to ease discovery, booking, payments and so on are key factors for the platform.

Those are the core concepts for the product, which was called OLS (OLSheets), paraphrasing the known platform OLX, known for the online exchange of various products, focusing on instruments and sheet music in this case.

1.2 Project limitations and know issues

<explain the known limitations, especially the **features that were planned/expected but not implemented** (and why…)

To be reviewed and completed by the end of the project >
<Indeed has to be reviewed by the end as mentioned above.>

Despite great effort to implement as much as possible for the sake of a seamless experience for the end users, there had to be scrapped off features like favorites/wishlists, for the sake of simplicity and to stay on the scope of the core evaluation parameters taken in the TQS class, which revolve especially around testing, Quality Assurance, and overall the DevOps lifecycle.

1.3 References and resources

<document the key components (e.g.: libraries, web services) or key references (e.g.: blog post) used that were really helpful and certainly would help other students pursuing a similar work>
<Include references to easypay api, and textbee…>
<This topic will be expanded on future iterations>

2 Product concept and requirements

2.1 Vision statement

<functional (black-box) description of the application: Which is the high-level/business problem being solved by your system? Which are the key features you promise to address it?>
<if needed, clarify what was planned/expected to be included but was changed to a different approach/concept >
<optional: how is your system different or similar to other well-known products?>
<optional: additional details on the process for the requirements gathering and selection (how did we developed the concept? Who helped us with the requirements? etc)>

2.2 Personas and scenarios

<"Personas are fictional people. They have names, likenesses, clothes, occupations, families, friends, pets, possessions, and so forth. They have age, gender, ethnicity, educational achievement, and socioeconomic status. They have life stories, goals and tasks. Scenarios can be constructed around personas, but the personas come first. They are not 'agents' or 'actors' in a script, they are people. Photographs of the personas and their workplaces are created and displayed. […] It is to obtain a more powerful level of identification and engagement that enable design, development, and testing to move forward more effectively". Adapted from Grudin, J. and Pruitt, J., 2002, June. Personas, participatory design and product development: An infrastructure for engagement. In Proc. PDC (Vol. 2).
Sample personas: secção 4.1, neste artigo (open access)] >

<Develop one or more representative scenarios for each persona. You don't need to include all possible details. Pick the main scenarios, related to the core value of the system.>
<The scenarios tell the story of the Personas in their lives, doing their daily/professional activities that are relevant to find the points of contact with the system under specification.
Scenarios are somewhat similar to use cases (they have a goal and tell a story), but, unlike use cases, they capture a larger process, with activities that may not use the software. Scenarios don't require a "template", like the usual use cases description.>
Sample: secção 4.2 neste artigo (open access)] >

<meter muuuuito mais yap. 50% dos nomes são adaptados de casos reais>

Simão Rio (Instrument Renter): Wants to rent an instrument

Tiago Rosário (Instruments owner) : Has some instruments at home that are not regularly used. He doesn't want to sell them, but is open to renting them

Alcino Batuta (Music sheets owner) : Has some music sheets and wants to make some money off of them

Alberto Jacinto (Sheets renter): Wants to rent some music sheets for some musics he wants to play/learn

Amílcar Almeida (System Administrator): System administrator

2.3 Project epics and priorities

[Apresentar um plano indicativo para a implementação incremental da solução ao longo de várias iterações/releases, explicando as funcionalidades a atingir por *epics*]

<Desenvolver em iterações futuras sobre a sequência planeada de desenvolvimento conforme os épicos criados>

List of epics:

- **Item Discovery & Catalog**

User Stories

As an instrument renter, I want to search for instruments by name so that I can quickly find what I want to rent.	As an instrument renter, I want to filter instruments by instrument family (wood, metal, strings, percussion) so that I can browse related items.
As a sheet renter, I want to search for music sheets by music title so that I can find the specific sheet I want to learn	As a sheet renter, I want to filter music sheets by category (e.g., classical, rock, jazz) so that I can find styles I like.
As an instrument renter, I want to filter instruments by type (e.g., guitar, violin) so that I can narrow down my options.	As a sheet renter, I want to view details(and photos maybe) for music sheets so that I know exactly what I'm renting
As an instrument renter, I want to view detailed descriptions and photos of instruments so that I can evaluate their condition and suitability	

- **Item Management**

As an instrument owner, I want to register my instruments with photos and descriptions so that renters can find them	As a sheet owner, I want to upload my music sheets with details so I can rent them out.
As any owner, I want to update availability so that renters know when items can be booked.	As any owner, I want to update pricing so that I can adjust rental rates.

- **Booking & Scheduling**

As an instrument renter, I want to book an instrument for specific dates so that I can reserve it.	As a sheet renter, I want to book music sheets for specific dates so that I can plan my learning schedule.
As any owner, I want to approve or reject booking requests so that I stay in control of instruments/sheet rentals.	As any owner, I want calendar integration so that I can see upcoming rentals easily.

- **Payment & Transactions**

As any renter, I want to pay securely for an instrument rental so that I can complete the transaction safely.

- **Platform Management & Administration**

As an admin, I want to monitor platform KPI's so I can understand performance and usage.	As an admin, I want to oversee bookings, payments, and user behavior so I can ensure everything runs smoothly.
As an admin, I want to monitor reported issues, So I can ensure that the users' problems are being taken care of.	

- **User Profiles & History**

As any renter, I want to view my past rentals so that I can track what I've used.	As any renter, I want to see my active bookings so I know what I currently have rented.
As a sheet renter, I want to save my favorite sheets so I can rent them later.	As any owner, I want to monitor my earnings so that I can track my income.

- **Reviews & Trust System**

As any renter, I want to rate an instrument/sheet after renting it so that I help others make decisions.	As any owner, I want to rate renters so that I build trust with future customers.
As any user, I want to report issues so that the platform can resolve problems.	As any renter, I want the reputation scores of owners to help me decide whether to rent

3 Domain model

<which information concepts will be managed in this domain? How are they related?>
<use a logical model (UML classes) to explain the concepts of the domain and their attributes, not a entity-relationship relational database model>

4 Architecture notebook

4.1 Key requirements and constrains

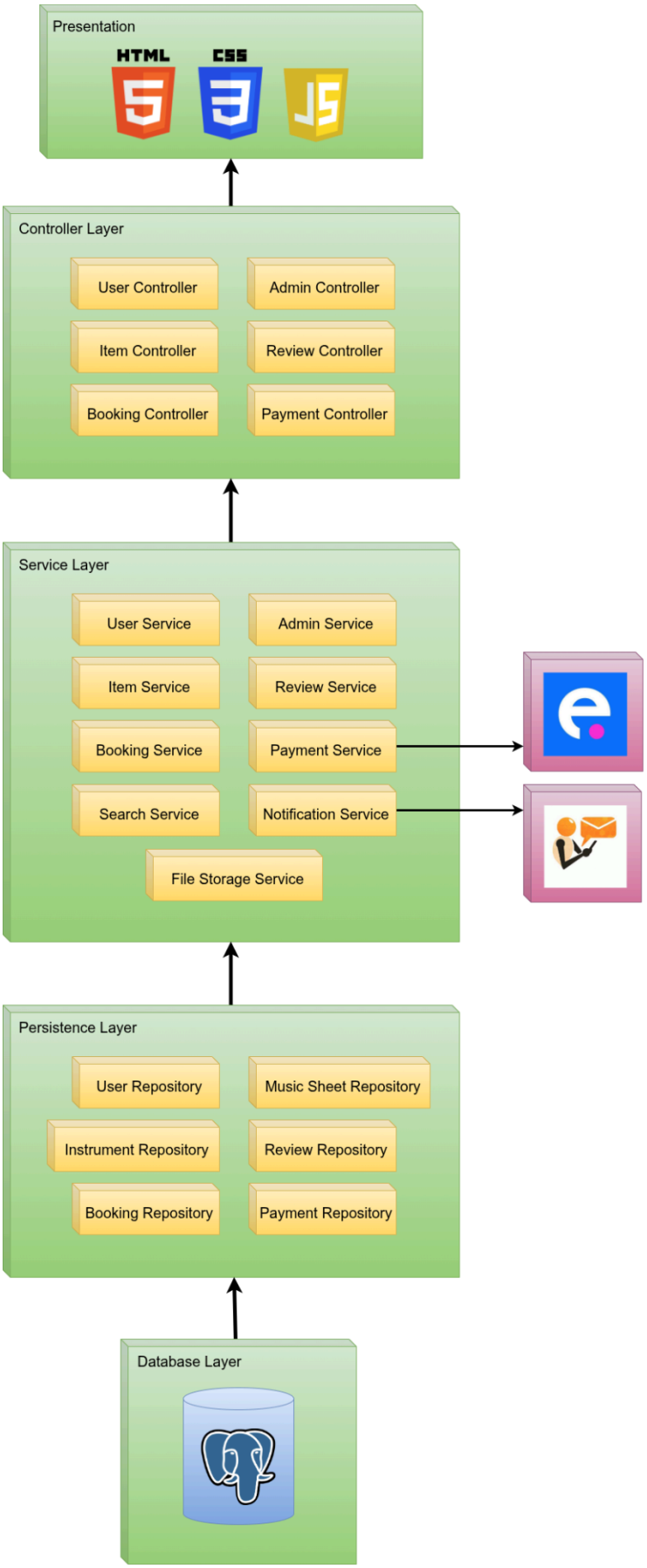
<**Identify issues that will drive the choices for the architecture** such as: Are there hardware dependencies that should be isolated from the rest of the system? Does the system need to function efficiently under unusual conditions? Are there integrations with external systems? Is the system to be offered in different user-interfacing platforms (web, mobile devices, big screens,…)?
For a more systematical approach:

- Note the collection of Architectural Characteristics the software architect should be aware
- Identify architectural characteristics that are relevant for your project (will drive the key design decisions). Note the case study and the explicit characteristics related to users and extensibility. This will support later non-functional tests.

4.2 Architecture view

→ Discuss architecture planned for the software solution: what are the main building blocks? include a diagram (a logical view, such as a package or block diagram). Avoid implementation technology or deployment references, but protocols/standards can be included.
→ refer to the architecture style applied, if any

 explain how the identified modules will interact. Use a sequence diagram to clarify the interactions along time, when needed
→ discuss more advanced app design issues: integration with Internet-based external services, data synchronization strategy, distributed workflows, push notifications mechanism, distribution of updates to distributed devices, etc.>


Presentation
HTML
CSS
Controller Layer
User Controller
Admin Controller
Item Controller
Review Controller
Booking Controller
Payment Controller
Service Layer
User Service
Admin Service
Item Service
Review Service
Booking Service
Payment Service
Search Service
Notification Service
File Storage Service
Persistence Layer
User Repository
Music Sheet Repository
Instrument Repository
Review Repository
Booking Repository
Payment Repository
Database Layer

4.3 Deployment view (production configuration)

[Explain the planned organization of the solution in terms of production configuration (deployment). Note the implementation technologies in the diagram, e.g.: place the PostgreSQL symbol in the Database, etc.]. Indicate the existence of containers (Docker), IP addresses, and ports, etc.
This part will be completed when deployments are actually in place.

5 API for developers

[Explain the API organization and main collections in general terms. Details/documentation of methods should be included in a hosted API documentation solution, such as Swagger, Postman documentation, or included in the development itself (e.g., Maven site).
 Be sure to use best practices for REST Api design. Keep minda REST API applies a resource-oriented design (APIs should be designed around resources, which are the key entities your application exposes, not actions)