TQS: Product specification report

Ana Alexandra Antunes [876543], Bruno Bernardes [876543]
v2025-04-25

[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>

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 >

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>

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

Commented [IO1]: or, in alternative, actors and use cases

<"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)] >

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*]

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.>

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)