This document provides a template for the Requirements Document. We have made good experience to actually write it in markdown within the gitlab Web-IDE (and then to use pandoc or similar to convert it to PDF). You are also welcome to use LaTeX or any other collaborative writing tool of your choosing.

We provide some guidelines on which content to provide. Yet, you are in many cases welcome to explore alternative ways to provide the appropriate content. If in doubt, discuss with your supervisor.

Title

Include the following information on Page 1 - Authors, email address - Date, Version history - Which author contributed to which parts (move to appendix if too long)

1. High-level description

This part of the requirments document scopes the project and provides the key ideas. It is hard to imagine any project that would not benefit from making explicit the requirements on this level.

1.1 Goal and scope

Describe the overall goal of the software. Include a context diagram to show the scope and describe it in text.

1.2 Business case and stakeholder map

Describe who will be using the software, which other stakeholders exist, and how the software will be worth its development - Describe how this project will affect the business, which business goals are relevant. This information provides critical guidance to scoping decisions. - Use *goal domain tracing* to derive a first draft of the stakeholder map. - Elaborate then about different stakeholder groups and their relation to the project.

use a table such as in L2, consider a graphical portfolio

Name	Relationship	Representative	Sentiment/Power
STAKEHOLDER NAME	RELATION TO System under Construction	Name and contact info of one person	How they feel about this project and whether they can influence it

1.3 Core functionality

Describe the core functions that the software will provide - Include a *use case diagram* or *goal model* that provides an overview of key actors and key functions. - Use IDs to trace between elements of the model and detailed use cases or task descriptions (to be found later in the document).

Consider using user story format here: As a USER I want FEATER so that BENEFIT, connecting the benefit to your business cases, if possible. Use a use case diagram to give an overview of core functionality. Make sure this is consistent with context diagram and content of Section 1.1

1.4 Performance Requirements, Specific Quality Requirements, Constraints

Describe the most important non-functional requirements in relation to the business goals. Please revisit the definition of functional and non-functional requirements (that is also why we choose the verbose title here). This section offers opportunity to adjust it to your specific project context: - It is fine to replace heading with "Non-Functional Requirements" but make sure to cover all of them. - Consider writing quality scenarios or user stories to link the attributes to stakeholder groups (this can complement PLanguage in Section 2.3, not replace it - still sufficient detail should be given on how quality can be measured). - Keep the content in this section on a high-level, related to the overall success of the project from a customer/business point of view. Move any operationalization and relation to functions to Section 2.

Provide some prioritization and rationale about which attributes are more or less important than usual. Use for example the *quality grid*.

2. User Requirements Specification

In this part, more detailed requirements are provided from the perspective of users. The goal is to provide problem-based requirements, i.e. to not foreclude any design decisions.

Most projects in real life do benefit from a concise description of this content. Agile projects in particular spread out this information or leave it for the agile team to discover them during the sprint. A large agile project would be well advised to document such content explicitly. Our company partners have started to manage this content as part of the source code repository, together with software and tests. Then, it is the responsibility of the agile teams to maintain the information.

It can be good to re-organize this section in a different way. - Especially for complex products, services, projects, you can consider to split it down by workarea or component. Then, the following headings are one level deeper, under each component.
- It can be good to change the order, e.g. start with functional requirements, then data, etc. - If in doubt, discuss with your supervisor

2.1 Data requirements

Provide a class or ER (or: class) diagram, a data dictionary, and requirements as needed. Keep them on a high level though, so that you can rely on these data items in your description of functional requirements. Make sure to explain all diagrams in text, e.g. through data dictionary. If you are tempted to add more detail to the data, move those details to Section 3.3 instead.

2.2 Functional requirements

Provide a list of features or task descriptions that give detail to the core functionality in Part 1 (High-Level Description). Make sure that each functional requirement relates to business value and its importance to particular stakeholders as well as project success. You could consider user stories here, but the section should provide more detail than Section 1.3. Perhaps use the user stories in Section 1.3 as the Task Name of your task descriptions and then add the additional information for a task description, task and support, or even screens and prototypes (but then, perhaps start working on your UI Prototype in Section 3.2 and rather provide strong tracing to the screens in that prototype instead to these task descriptions). Rely on L4 content to shape the content here.

2.3 Detailed Performance Requirements, Specific Quality Requirements, Constraints

Based on your prioritization in Section 1.4, Use *PLanguage* or similar to describe (only) the important quality attributes (or: non-functional requirements) in detail. Depending on your project and the rest of this document, there are a lot of good options: - Consider tracing to or from functional requirements or data requirements to describe quality in context. - Consider moving some detailed quality requirements to a new Section in Section 3, especially if they relate to internal qualities directly connected to the system.

2.4 Proposed prioritization

Make explicit how functional requirements were prioritized. Use the subsections here to provide guidance to the supplier about priorities.

Do not hestitate to also reveal here the analysis (e.g. $100 method) and its results.

2.4.1 Next release

Describe which requirements to cover in the next release of the system under construction (i.e. the one you defined in your project mission and that you are writing requirements for, NOT R1, R2, R3 in this course).

2.4.2 Second release

Describe which requirements to cover in the second release. This is a best practice, if you ever have to schedule tasks: Always also plan for the iteration/release/increment after the one that comes next. It makes it easier to focus on the most important things in your next iteration.

2.4.3 Future releases (optional)

Describe which requirements not to cover in the first two releases. Note, that these could still be important requirements, but perhaps, based on interdependencies and relation to business goals / priorities of certain stakeholder groups, they might not be urgent. Make sure that a reader can understand your decisions.

3. System Requirements

In this part, more detail is provided and requirements are specified from the perspective of the system. Requirements are much closer to specific solution ideas or design decisions.

Modern agile projects may rely on the actual artifact instead, e.g. early version of the software itself instead of prototypes, the data base schema instead of data requirements, and automated tests of some sort. Such projects often suffer from a lack of overview and fail to properly include UX experts in the development or to maintain their data schemas or effective test suites. However, in a typical, fast-paced agile project, this content may get outdated too quickly. Especially low level functional requirements then become cumbersome to maintain. For a supplier, this level of requirements can be crucial to maintain: How are the customer requirements thought to be covered?

For the scope of this course, it is our goal to show some awareness for this perspective, especially to avoid unrealistic requirements in Part 1 and 2. You may focus and provide such detailed requirements only for certain parts.

3.1 System requirements

Detailed requirements from the perspective of a system. This could be feature requirements ("The system shall…"), consider EARS notation (https://alistairmavin.com/ears/) If no more detailed requirements can be reasonably added, consider providing acceptance tests instead (Section 3.4)

3.2 UI Prototype

Sketch the most important UI elements of the project, e.g. screens. Arrange them to show how core functionality and critical attributes are supported. An idea that might work for you is to rely on an activity diagram syntax and replace the activities with prototype screens.

3.3 Detailed Data Requirements

Specify constraints and detailed properties of data as needed.

3.4 Acceptance Tests (Optional)

Describe how to determine whether the product is acceptable with respect to core functionality and critical attributes. This is not a focus within this course, but should be a concern in a real world project. This can be a useful alternative, if no more detailed requirements can be reasonably added in Section 3.1