**Requirements Management App, July – August 2025**

Antoan Krasimirov Ivaylov

Documentation

Table of contents

[1. Project idea 3](#_Toc203723144)

[2. Requirements 4](#_Toc203723145)

[2.1 Domain overview 4](#_Toc203723146)

[2.2 List of functional requirements (Extracted from the domain overview and other systems) 5](#_Toc203723147)

[2.3 SoftUni requirements for the project 5](#_Toc203723148)

[2.4 Non-functional requirements for the system 5](#_Toc203723149)

[3. Solution 6](#_Toc203723150)

[3.1 Entities and E/R Model 6](#_Toc203723151)

[3.2 Object oriented model 6](#_Toc203723152)

[3.3 Testing 6](#_Toc203723153)

[3.4 How are concrete requirements handled? 6](#_Toc203723154)

[3.4.1 List of functional 6](#_Toc203723155)

[3.4.2 SoftUni 6](#_Toc203723156)

[3.4.3 Non-functional 6](#_Toc203723157)

# Project idea

The idea for the project came from a compulsory course “Analysis of software requirements” in Sofia University “Saint Kliment Ohridski” during the 2023-2024 summer semester. Then I together with a few colleagues had to develop a plan to capture, document, validate requirements from stakeholders on a project of our choice according to the methodology of strict requirement analysis, which intakes user stories and translates them to fully developed and documented requirements, which can be used as fundamentals for contract negotiations. Everything seemed fine, until the project had to be implemented partly inside a **requirements management system. None of the available at the time were free (GNU GPL) or gave a student’s license** to us as students at Sofia University. We had to settle for a 2 weeks free trial of a system instead of the regular 3 months for developing the project, in which we had to translate large Word files to the corresponding software, which was no fun at all.

Then came the “Web technologies” course in the same semester in which I took part in a group project, which had as a goal to target the missing system. Along the course of the semester, gradually the team dropped interest in it due to different interests of the group members, until of course the final 2 weeks came. The project, which can be [found at GitHub](https://github.com/mikaelar/Web-Project), is still available, but was / is a disaster non the less. We had to write it in PHP (which is learned at the time of the course + the whole HTML5 family), which didn’t even have a debugger included from the get go. In another words, the whole debugging process became an ‘echo’ printing fiesta on web steroids. Another critical point was the prohibition of libraries, most notably an ORM, which at the time I didn’t know was such a big feature for keeping up the relational model with the object-oriented one. After switching up the Database model 2 times, together with the controllers for it, which took me nearly 2 weeks and about 5000+ lines of refactored code, the deadline came and of course the project failed due to changing requirements and unmanageable code.  
  
I am determined to rewrite this project the right way – using a top-notch language as C#, cover the code with unit tests, rely on Object relational mapper to keep the E/R and OO models synchronized, and use REST API together with HTML5 to produce a full stack end-to-end web application, which is free and available to the public using a GNU GPL license.

# Requirements

## 2.1 Domain overview

User stories and requirements are the fundamentals of software architecture by which modern software systems are developed. The process is cyclic since there is a problem inside a given domain, which requires a solution (software) and since the solution changes the domain, a new requirements analysis is needed afterwards. Basically, a system which handles software requirements should encapsulate the knowledge of organizations’ projects to the extent that the information written inside requirements can be used as a common ground for contract between **stakeholders**. The **requirements** are either classified as **functional** (what does the software do), **non-functional** (constraints on functionalities - how does the software achieve the desired actions) and **domain** ones (strictly related to the area for which the software is developed). Since requirements are documented knowledge, to achieve that, one must apply a software elicitation technique to extract knowledge (mostly from stakeholders, but other sources still do apply) and write it in the form of **user stories** in the form “Me as a <person\_who\_uses\_the\_system>, would like it to implement functionality <some\_system\_functionality>, so that / because <reason\_of\_the\_person>”. **From each user story should a set of requirements should be generated and linked directly to it**. For thoroughly thought analysis, the software requirements engineer should use a number of techniques, together with communicating with a sheer variety of stakeholders – each one should represent a target group. Key point in software requirements analysis is **prioritization of requirements**, since not all wishes of the clients can be achieved or more precisely are rational to be implemented. For that a **scale of 1-5** is often used with higher number meaning a requirement of bigger importance for the given group of stakeholders and therefore the current project. Key attribute of understanding requirements is that each one factors out a single characteristic of the system, so that a **set of requirements defines the system as a whole, meaning they do not live in isolation one from another**. **For effectively handling the relationship between requirements a scale of [-5,5]** is also used between linked ones, where -5 means crucial negative impact, whilst 5 stands for perfect symbiotic relationship. Here another benefit of requirement’s prioritization can be seen – the ability to effectively tackle the problem of **conflicting requirements**. Each one inside the violating set may have pluses and minuses, but according to the stakeholders, which have made an informed decision, the right requirement of a conflicting group can be composed together, even if it means that compromises must be made in the final form of the requirement. Since descriptions of the system can vary from pure abstract ones to extremely concrete others and according to the rule of thumb that each requirement must be at most 3, 4 sentences long while targeting only 1 specific goal, **subrequirements can be used to extend the parent requirement**. Thus we can assume that the **requirements may follow a hierarchical (tree-like) structure**. Lets go back to the beginning of the paragraph and the statement that the domain changes and so must the analysis. Each requirement must be treated as a subject of change, that is valid until the domain changes – new product is released, feedback from prior version, defects found in development stages, shifts in organizations culture and etc. – in the perfect system there should be a way to keep track of the changes and if possible, go back to a prior stage of the requirement.

Another extra feature of the system may be **comment** sections, which can be appended for each requirement since the system could provide a room for discussion of the project inside it.   
On the same note of extra features, I propose an **email notifier with a concrete event and a subscription mechanism to concrete type of events** (Observer pattern) can be implemented so that users can get a heads up when their ideas get commented, liked or the project gets injected with new information from the other employees who are involved in it (new requirements, deleted ones, new priorities / links between).

## 2.2 List of functional requirements (Extracted from the domain overview and other systems)

1. The system allows for registration of a new project by any user. The project has a name with optional parameters being description and image.
2. The owner of the project can invite other people to join the project.
   1. If a user accepts to join, he is eligible to \**contribute to the project*.
3. A project partaker can contribute to a project by creating new user stories, comments or requirements to the given one, modifying existing requirement priorities or creating (removing) links among requirements inside the given project.
4. A comment is a text section, which can be appended to either a user story or a requirement. It supports Unicode and also contains meta information – author, date and time of writing.
   1. Comments should be provided with deletion functionality, but their contents should stay hidden inside the system if legal questions arise in the future.
5. The system should support user stories, which can be defined as an Entity having a title, persona, action / desire, argumentation, creation date, modified date and author.
6. Requirements must be implemented inside the system.
   1. A requirement has a heading, description, priority (for the individual project!).
   2. If a requirement is a non-functional one, it should also support an acceptance Criteria in the form of text and a metric – pair of metric name and expected value.
   3. Requirements must provide an interface for binding them to other requirements (with impact coefficients), user stories and for appending comments onto them.
   4. Requirements should also support the inverse operation of linking to other entitites
7. Deletion of requirements, user stories and projects must be implemented as a soft delete, since requirements may be used for foundation of contracts.
8. The software should provide a mechanism of rollbacking to a prior version of every domain entity – comment, user story, requirement, project.
   1. Memento pattern should be used – either by storing the transformation and building the inverse one of it, to apply to the modified data or just by keeping track of previous records of the data.
9. When changes inside a project, which the user is partaker occur, he should receive a notification by a channel of his choice.
   1. The default channel of communication is notifications within the App and by email.
   2. A user can reconfigure his channels of communication during system runtime.
   3. A user can reconfigure the events he is subscribed to during system runtime. That includes getting notifications for comments, requirements, user stories and the different actions with them according to points 3, 4, 5 and 6.
10. The system must be able to generate requirement dependencies matrices on a project level.
    1. The generated matrix may be configured, so that it shows only a subset of all requirements
    2. The generated image must be stored in the <FORMAT> and be eligible for download
    3. The generated image must be displayable on the web interface of the App

## 2.3 SoftUni requirements for the project

## 2.4 Non-functional requirements for the system

# Solution

## 3.1 Entities and E/R Model

More than 50% of requirements can be reused (Source: “Lectures of software requirements analysis)”

## 3.2 Object oriented model

## 3.3 Testing

## 3.4 How are concrete requirements handled?

### 3.4.1 List of functional

### 3.4.2 SoftUni

### 3.4.3 Non-functional