BookLoop

Version 1.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 18 Mar 2021 | 1.0 | The initial description of the system. | Stancu Gabriel - Iulian |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

2. Non-functional Requirements 4

2.1 Availability 4

2.2 Performance 4

2.3 Security 4

2.4 Testability 5

2.5 Usability 5

3. Design Constraints 5

# Introduction

The purpose of this document is to define requirements of the BookLoop project. This Supplementary Specification lists the requirements that are not readily captured in the use cases of the use-case model. The Supplementary Specifications and the use-case model together capture a complete set of requirements on the system. We will include here quality attributes, such as the usability, reliability, performance and supportability of the application. Lastly but not least, we will include some other technical features, such as the compatible operating systems for this project, the supporting environments, compatibility requirements and design constraints.

# Non-functional Requirements

The non – functional requirements of a system describe the system’s operation capabilities and constraints to enhance its functionality. In what follows, we will discuss and document the following aspects: the availability, the performance, the security, the testability and the usability of the overall system. We will provide both the expectations from each such aspect, but also the solution employed for reaching this convenience and provide the final users (humans, book readers more specific) with all the functionalities provided by this project development.

## Availability

The system provided by this project will be available 24 hours a day, 7 days a week. Here we take into consideration the server, first of all, which should be online at every moment in time, but also the possibility of the clients to connect to the server itself, through a network connection. As some issues might still appear (updates on the server functionality, for instance), the down time of the server will not exceed more than half an hour.

## Performance

When talking about performance, we take into consideration several factors. First of all, scalable factors such as the number of registered users, posts etc., but also the number of simultaneous connections to the server, will increase over time. Depending on the approach taken (physical server / cloud server), upgrades should be performed over time: improve the physical components of the server or achieve a better hosting plan from the provider respectively.

Now, considering lower rate scalable factors, the database access response time will be reduced to a 10 second latency. If no active connection to the database can be established, a timeout will be reached. Otherwise, for situations such as posts and articles retrieving, the time required for the database operations should be even shorter. From the messages point of view, live messages will be redirected to the receiver almost instantly, as this is the principle that stands for the “instant messaging” concept. Older messages retrieve from the database might take slightly longer.

## Security

Considering the security of the application, we have to provide a safe way for managing users’ information. As most of the information is stored in a database, this is the first concern we address. Because an object – relational mapping framework will be employed, all database queries are handled by the framework itself (and their security included: SQL injection protection, for instance).

When employing a physical server, although not recommended, several layers of protection should be designed, such as firewalls, proxies etc. When going for a cloud server, however, these aspects are handled by the provider of the service (such as Azure). Furthermore, the connection to the database will be secured, needing a registered user and an associated password.

Finally, for user information safety, their personal data will be saved in encrypted format (data such as their password), so that not even the database administrator would have access to this sensible data. Moreover, the messages will be encrypted once, before they leave the client sender’s application, they will reach the server and they will be sent in this format to the receiver client, to be decrypted, displayed etc., but before being stored in the database, they will be encrypted once more, thus simulating a two – factor authentication approach for the security of the messages themselves.

## Testability

The testability of the project will be ensured by the design decisions taken for building it. Both the server and the client application will be designed in a layered architecture, so that ultimately the only part of the project not available for direct testing should be the user interfaces. Providing automated testing for the entire application (all layers except the UI: database access layer, business logic layer etc.) will fasten the process of later improvements introduction in the application. Tests will be written as the project is developed, in a test – driven development manner.

## Usability

For increasing the usability of the whole project, portable technologies were employed (they will be discussed in chapter 3 of this paper). The server component may be hosted on any popular operating system, which is Windows, MacOS, Linux. However, it can also be deployed to cloud service providers, such as Azure. The client component also provides high portability, as the application can be deployed to both Android and iOS devices. Future releases might include UWP devices as well.

# Design Constraints

The project comes with several design constraints. Because we are targeting several platforms with the components we design (the server and the client, but also the database could be taken into consideration here), we will use a multi – platform, portable programming language. For this matter, the C# programming language will be used.

As for the development tools, Visual Studio (or the lightweight version, Visual Studio Code) will be used for most of the project development. For managing the server, applications such as cPanel or ownCloud might be needed. For database design, access and overall management, SQL Management Studio will be used for handling an SQL Server structure.

In order to provide high portability for the server component, the ASP.Net Core framework, provided by Microsoft, will be used. As previously mentioned, this decision will allow the server to be deployed on the most popular OSes, such as Windows, MacOS or Linux. Coming from Microsoft, this also provides access to the Azure hosting services.

For the client application, considering the intention to target as many platforms as possible and the need for a simple connection and communication with the server, another .NET framework will be employed for developing the phone application that will act as a client: the Xamarin Forms framework. This will provide a simple way to develop simultaneously the application for both Android and iOS devices with just one codebase.

For database access from the server component, a .NET compatible object - relational mapping technology will be needed. Considering this, the Entity Framework Core package is the best approach that could be implemented.

For the messaging functionality, another .NET compatible framework will be used: the SignalR library. It will be injected in both the server and the client application, mapping itself on the client – server protocol-based communication system.

Finally, we take into consideration the architecture of both the client and the server component. For the client component (the phone application), the MVVM architectural pattern will be employed, making use of the underlying structure of a Xamarin Forms project, similar with a WPF one. The server will be divided by the MVC pattern, the view component being replaced by a server API in this case.