Fridgify

For Fridge Content Tracking

Version 0.1

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 3

1.1 Purpose 3

1.2 Scope 3

1.3 Definitions, Acronyms, and Abbreviations 3

1.4 References 3

1.5 Overview 3

2. Overall Description 3

3. Specific Requirements 3

3.1 Functionality 3

3.1.1 <Functional Requirement One> 3

3.2 Usability 3

3.2.1 <Usability Requirement One> 3

3.3 Reliability 3

3.3.1 <Reliability Requirement One> 3

3.4 Performance 3

3.4.1 <Performance Requirement One> 3

3.5 Supportability 3

3.5.1 <Supportability Requirement One> 3

3.6 Design Constraints 3

3.6.1 <Design Constraint One> 3

3.7 On-line User Documentation and Help System Requirements 3

3.8 Purchased Components 3

3.9 Interfaces 3

3.9.1 User Interfaces 3

3.9.2 Hardware Interfaces 3

3.9.3 Software Interfaces 3

3.9.4 Communications Interfaces 3

3.10 Licensing Requirements 3

3.11 Legal, Copyright, and Other Notices 3

3.12 Applicable Standards 3

4. Supporting Information 3

# Introduction

[The introduction of the **Software Requirements Specification (SRS)** provides an overview of the entire **SRS**. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the **SRS**.]

[Note: The **SRS** document captures the complete software requirements for the system, or a portion of the system. Following is a typical **SRS** outline for a project using only traditional, natural-language style requirements—with **no use-case modeling.** It captures all requirements in a single document, with applicable sections inserted from the Supplementary Specifications (which would no longer be needed). For a template of an **SRS** using use-case modeling, which consists of a package containing Use Cases of the use-case model and applicable Supplementary Specifications and other supporting information, see rup\_srsuc.dot.]

[Many different arrangements of an **SRS** are possible. Refer to [IEEE830-1998] for further elaboration of these explanations, as well as other options for **SRS** organization.]

## Purpose

The **Fridgify SRS** provides a general overview over the project as well as a detailed description. This document is going to delve into the general vision, **Fridgify’s** purpose and its features. System specifications, interfaces and constraints of the product will be illustrated in this **SRS**.

## Scope

**Fridgify** is a mobile application, designed to help people keep track of the contents of their fridges. This is achieved by scanning the barcode of a product and its due date, which will be kept track in a database.  
The application should be free to download in **Apple’s App Store** as well as **Google’s Play Store**.  
  
Furthermore, **Fridgify** requires an internet connection to fetch and display results stored in our database. All information related to the system, user and contents are maintained in a database, which is located on a root server. The mobile app is going to interact with a Python backend, which provides an API interface to retrieve, insert and process data from the database. By using a centralized backend, the user can synchronize his fridge on multiple devices alone and with different users.

## Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| User | Someone who interacts with the mobile phone application |
| Device | Device, which allows users to keep track of their fridge contents (e.g. mobile phone, Raspberry Pi) |
| REST | **RE**presentational **S**tate **T**ransfer is an architectural style for distributed hypermedia systems [1] |
| API | Application Programming Interface connects to *client* and *server*. [2] |
| Application Store | A mobile application store, where users can get the application (e.g. App Store, Play Store) |
| OS | Operating System |
| Android | Google’s OS for mobile phones [3] |
| iOS | Apple’s OS for mobile phones [4] |

## References

[1] “What is REST”, <https://restfulapi.net/>  
[2] Braunstein, Mark L., „Health Informatics on FHIR: How HL7’s New API is Transforming Healthcare”. Springer, 2018  
[3] “About the platform”, <https://developer.android.com/about>  
[4] “iOS 13 – Apple Developer”, <https://developer.apple.com/ios/>

Here are documents and links which could be helpful to understand **Fridgify**:

[A] Fridgify Blog: <https://fridgify.donkz.dev/>  
[B] Fridgify GitHub: <https://github.com/DonkeyCo/Fridgify>  
[C] UML-Diagram:

## Overview

[This subsection describes what the rest of the **SRS** contains and explains how the document is organized.]

The remainder of the document includes three chapters. The second one provides an overview of the system functionality and system interaction with other systems.

The third chapter provides the requirement specification in detailed terms and a description of the different system interfaces.

In chapter four, extra information is provided, such as appendixes, user stories, etc.

# Overall Description

**Product Perspective**

This system is going to consist of two parts: one mobile application and one Web-API. The mobile application is used to keep track of contents inside the fridge as well as registering and unregistering items. The Web-API is used to store, retrieve and process data provided by the backend.

The mobile application needs to communicate with a Web-API. The Web-API, designed as a REST API, provides necessary data for the client.

Since this is a data centric product, a database is required to store data. For a client to access data, he has to communicate with the REST API, which in conclusion works with the database. No direct access to the database is required. To communicate with the REST API, the client needs to authenticate himself, otherwise operations for data retrieval as well as data storing is prohibited.

**Product Functions**

With the mobile app of Fridgify, the user is able to look into the items his fridge contains. By registering, via scanning a barcode or manual input, a user can add items to the tracking system. Via the mobile app a user is able to unregister an item by removing the specific item from the list by manual input (in later versions maybe by scanning?)

Scanning a barcode produces an article identifier provided by the code. This code is used to store the product inside of a database or retrieve information for the product. When scanning the barcode, the user can scan the due date as well to keep track of it.

Messages are managed by the backend. By this, a user can be notified if an item is expired, an item is out of stock or a reminder to buy new items.

The Web API enables the client application to add and retrieve data via GET and POST Requests. An OAuth 2.0 Authentication is needed to communicate with the backend.

**User Characteristics**

There is one only one user group interacting with the application. Each user has the ability to manage his or her own fridge, they also have the opportunity to join groups. Joining groups allows them to keep track and manage their fridges with multiple users, which is helpful for families or people sharing an apartment. Every user is able to register, unregister and list items of a fridge.

**Constraints**

The mobile application is constrained by the capacity of the database. Since multiple users can request items, requests could be possibly queued.

Requests to the backend by the mobile app are constrained by the server capacities, high traffic could possibly lead to slower times.

An internet connection is recommended, because otherwise synchronization with the database is not possible. Data is being stored in a local database, if possible.

**Assumptions and dependencies**

One assumption about the product is that it will be used on mobile devices, which have the necessary computing power. If the phone does not have enough hardware resources available for the application, there may be scenarios where the application is not working properly.

# Specific Requirements

[This section of the **SRS** contains all software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. When using use-case modeling, these requirements are captured in the Use Cases and the applicable supplementary specifications. If use-case modeling is not used, the outline for supplementary specifications may be inserted directly into this section, as shown below.]

## Functionality

[This section describes the functional requirements of the system for those requirements that are expressed in the natural language style. For many applications, this may constitute the bulk of the **SRS** package and thought should be given to the organization of this section. This section is typically organized by feature, but alternative organization methods may also be appropriate; for example, organization by user or organization by subsystem. Functional requirements may include feature sets, capabilities, and security.

Where application development tools, such as requirements tools, modeling tools, and the like, are employed to capture the functionality, this section of the document would refer to the availability of that data, indicating the location and name of the tool used to capture the data.]

### <Functional Requirement One>

[The requirement description.]

## Usability

[This section includes all those requirements that affect usability. For example,

* specify the required training time for a normal users and a power user to become productive at particular operations
* specify measurable task times for typical tasks or base the new system’s usability requirements on other systems that the users know and like
* specify requirement to conform to common usability standards, such as IBM’s CUA standards Microsoft’s GUI standards]

### <Usability Requirement One>

[The requirement description goes here.]

## Reliability

[Requirements for reliability of the system should be specified here. Some suggestions follow:

* Availability—specify the percentage of time available ( xx.xx%), hours of use, maintenance access, degraded mode operations, and so on.
* Mean Time Between Failures (MTBF) — this is usually specified in hours, but it could also be specified in terms of days, months or years.
* Mean Time To Repair (MTTR)—how long is the system allowed to be out of operation after it has failed?
* Accuracy—specifies precision (resolution) and accuracy (by some known standard) that is required in the system’s output.
* Maximum Bugs or Defect Rate—usually expressed in terms of bugs per thousand lines of code (bugs/KLOC) or bugs per function-point( bugs/function-point).
* Bugs or Defect Rate—categorized in terms of minor, significant, and critical bugs: the requirement(s) must define what is meant by a “critical” bug; for example, complete loss of data or a complete inability to use certain parts of the system’s functionality.]

### <Reliability Requirement One>

[The requirement description.]

## Performance

[The system’s performance characteristics are outlined in this section. Include specific response times. Where applicable, reference related Use Cases by name.

* Response time for a transaction (average, maximum)
* Throughput, for example, transactions per second
* Capacity, for example, the number of customers or transactions the system can accommodate
* Degradation modes (what is the acceptable mode of operation when the system has been degraded in some manner)
* Resource utilization, such as memory, disk, communications, and so forth.

### <Performance Requirement One>

[The requirement description goes here.]

## Supportability

[This section indicates any requirements that will enhance the supportability or maintainability of the system being built, including coding standards, naming conventions, class libraries, maintenance access, and maintenance utilities.]

### <Supportability Requirement One>

[The requirement description goes here.]

## Design Constraints

[This section indicates any design constraints on the system being built. Design constraints represent design decisions that have been mandated and must be adhered to. Examples include software languages, software process requirements, prescribed use of developmental tools, architectural and design constraints, purchased components, class libraries, and so on.]

### <Design Constraint One>

[The requirement description goes here.]

## On-line User Documentation and Help System Requirements

[Describes the requirements, if any, for o-line user documentation, help systems, help about notices, and so forth.]

## Purchased Components

[This section describes any purchased components to be used with the system, any applicable licensing or usage restrictions, and any associated compatibility and interoperability or interface standards.]

## Interfaces

[This section defines the interfaces that must be supported by the application. It should contain adequate specificity, protocols, ports and logical addresses, and the like, so that the software can be developed and verified against the interface requirements.]

### User Interfaces

[Describe the user interfaces that are to be implemented by the software.]

### Hardware Interfaces

[This section defines any hardware interfaces that are to be supported by the software, including logical structure, physical addresses, expected behavior, and so on.]

### Software Interfaces

[This section describes software interfaces to other components of the software system. These may be purchased components, components reused from another application or components being developed for subsystems outside of the scope of this **SRS** but with which this software application must interact.]

### Communications Interfaces

[Describe any communications interfaces to other systems or devices such as local area networks, remote serial devices, and so forth.]

## Licensing Requirements

[Defines any licensing enforcement requirements or other usage restriction requirements that are to be exhibited by the software.]

## Legal, Copyright, and Other Notices

[This section describes any necessary legal disclaimers, warranties, copyright notices, patent notices, wordmark, trademark, or logo compliance issues for the software.]

## Applicable Standards

[This section describes by reference any applicable standard and the specific sections of any such standards which apply to the system being described. For example, this could include legal, quality and regulatory standards, industry standards for usability, interoperability, internationalization, operating system compliance, and so forth.]

# Supporting Information

[The supporting information makes the **SRS** easier to use. It includes:

* Table of contents
* Index
* Appendices

These may include use-case storyboards or user-interface prototypes. When appendices are included, the **SRS** should explicitly state whether or not the appendices are to be considered part of the requirements.]