The Cool Company

Fridgify
Software Requirements Specification
For Fridge Content Tracking

Version 1.0

Revision History

Date	Version	Description	Author
19.10.19	1.0	Nearly completed draft	Duc Vo Ngoc

Table of Contents

Software Requirements Specification

1. Introduction

1.1 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**.

1.2 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.

1.3 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]

1.4 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:

1.5 Overview

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.

2. 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.

3. Specific Requirements

3.1 Functionality

This section contains all the functional and quality requirements of the system. It gives a detailed description of the system and all its features.

3.1.1 User Interface

The user interface should be easily accessible. Users can log in and register on a dedicated view. Users can access a list, showing every item registered in the fridge. Inside that view, users can start a scanning process to register items, register them manually or unregister them.

3.1.2 Scanning

Via the mobile application users should be able to scan barcodes of items, which in turn registers them in the application. Scanning the item automatically adds all information to a local or central database for the user. Item information are stored in the database with the barcode.
Users can also scan expiration dates to keep track of their expiry.

3.1.3 REST API

The REST API provides information for each user. Via an OAuth 2.0 Authentication model, a mobile application can communicate with the backend and retrieve data. Calling the API endpoints allows applications to retrieve raw and processed data as well as adding data.

3.1.4 Notification Service

The backend should send notifications to individual users. The notification service notifies users when an item is expiring, updating users of their current fridge status or empty items.

3.2 Usability

3.2.1 Ease of use

Users should be able to easily keep track of their fridge.
Scanning an item should only require the user a maximum of 3 to 4 clicks.
Unregistering an item should only require the user a maximum of 3 to 4 clicks.
Looking at the list of items should only require a maximum of 2 to 3 clicks.

3.2.2 Fridges

Users should be able to change a fridge easily. They should be able to change registered fridges with a maximum of 3 clicks.

3.3 Reliability

3.3.1 Code Coverage

Via Unit Tests a code coverage of a minimum of 95% should be reached. Reaching such a code coverage allows a very good reliability of both the frontend and backend.

3.3.2 Server Reliability

The percentage of the backends time availability should be around 94%. Restart of the backend should be automatically. Maintenance should not interfere with the backends availability. The backend's maximum

downtime should have a maximum of 5 hours.

3.4 Performance

3.4.1 Registering Items

Response time for registering items should have an average processing time of 3 to 5 seconds and a maximum processing time of 10 seconds.

3.4.2 Unregistering Items

Unregistering items should have an average processing time of 3 to 5 seconds and a maximum processing time of 8 to 10 seconds.

3.5 Supportability

3.5.1 Coding Standards

The python source code of the application must follow the PEP 8 Style Guide to guarantee readability and one style of code written by multiple authors.

3.5.2 Coding Conventions

3.6 Design Constraints

3.6.1 MVC

Backend as well as frontend are being developed with MVC architecture.

3.6.2 Programming Languages

This application uses two different programming languages, Python 3.7 and Dart. A PostgreSQL Database is used to store data.

3.7 On-line User Documentation and Help System Requirements

Documentation for the API will be available soon on a dedicated web page. To be up to date with the newest development status, you can follow the Fridgify Blog.

3.8 Purchased Components

The following components were purchased:
- Root Server (60GB SSD, 16GB RAM, Intel Xeon Gold 6140 / 6320)

3.9 Interfaces

3.9.1 User Interfaces

To be decided.

3.9.2 Hardware Interfaces

The backend can be used by a Raspberry Pi to register items. This could be implemented in the future.

3.9.3 Software Interfaces

The application should be accessible from:
· Android Devices
· iOS Devices
· (Raspberry Pi [Microcontrollers in general])

3.9.4 Communications Interfaces

The backend is accessible through HTTPS on port 443. Any unencrypted connection over HTTP on port 80 are not supported and will be redirected to HTTPS.

3.10 Licensing Requirements

To be decided.

3.11 Legal, Copyright, and Other Notices

This document makes use of the generic he for reasons of readability. Any terms containing the words "he", "himself", "his", etc. are meant to include both women and men, unless explicitly stated otherwise.

The Fridgify Team will not take any responsibility for forgotten items, over purchases, etc.

3.12 Applicable Standards

Additionally, the application must be developed to provide the best possible security. Released code needs to be checked regarding the following types of attack:

SQL injections

Cross-site scripting (XSS)

Cross-site request forgery (CSRF)

Manual security tests needs to be performed regulary.

4. Supporting Information

Fridgify - Blog – The Fridgify Blog with all news regarding Fridgify.

Fridgify - GitHub – The Fridgify GitHub contains the source code.

Fridgify - Project Management – Fridgify uses YouTrack for project management.