**Software Requirements Specification**

**Healthy Bytes**

Version 1.0

April 4th, 2018

**Lead Software Engineer:**

Alexis Arriola

**Project Team:**

Kayla Buentello

Luis Gonzalez

Ruben Olivares

Matthew Rodriguez

Prepared for

Software Engineering

Instructor: MK Quweider, Ph.D.

Spring 2018

**Table of Contents**

[1. Introduction 3](#_30j0zll)

[1.1 Purpose 3](#_1fob9te)

[1.2 Scope 3](#_2et92p0)

[1.3 Definitions, Acronyms, and Abbreviations 3](#_1t3h5sf)

[1.4 References 4](#_3rdcrjn)

[1.5 Overview 4](#_qsh70q)

[2. General Description 5](#_1pxezwc)

[2.1 Product Perspective 5](#_49x2ik5)

[2.2 Product Functions 5](#_147n2zr)

[2.3 User Characteristics 5](#_1v1yuxt)

[2.4 General Constraints 5](#_2u6wntf)

[2.5 Assumptions and Dependencies 5](#_111kx3o)

[3. Specific Requirements 6](#_3ygebqi)

[3.1 External Interface Requirements 6](#_2dlolyb)

[3.1.1 User Interfaces 6](#_sqyw64)

[3.1.2 Hardware Interfaces 6](#_3cqmetx)

[3.1.3 Software Interfaces 6](#_1rvwp1q)

[3.1.4 Communications Interfaces 6](#_4bvk7pj)

[3.2 Functional Requirements 6](#_2r0uhxc)

[3.2.1 Store User Login Credentials 6](#_1664s55)

[3.2.2 Store Recipes into User’s My Recipes List 7](#_kgcv8k)

[3.2.3 Remove Recipes from User’s My Recipes List 7](#_1jlao46)

[3.2.4 Save the My Recipes List to a Data/Text File 8](#_2iq8gzs)

[3.2.5 Send User Recipe List to a contact via Email 8](#_4h042r0)

[3.3 Use Cases 8](#_2afmg28)

[3.3.1 Create a Healthy Bytes Account 8](#_pkwqa1)

[3.3.2 User Search for Recipe 9](#_39kk8xu)

[3.3.3 Adding a Recipe to Favorites 9](#_1opuj5n)

[3.3.4 Generate Shopping List 9](#_48pi1tg)

[3.3.5 Create Text/Data Output File 9](#_2nusc19)

[3.4 Classes / Objects 9](#_1302m92)

[3.4.1 RB\_MainApp 9](#_3mzq4wv)

[3.4.2 Recipe\_Book 9](#_1gf8i83)

[3.4.3 RB\_DatabaseManager 9](#_40ew0vw)

[3.4.4 RB\_View 9](#_2fk6b3p)

[3.4.5 RB\_Controller 9](#_upglbi)

[3.4.6 RB\_Model 9](#_3ep43zb)

[3.5 Non-Functional Requirements 9](#_4du1wux)

[3.5.1 Performance 9](#_2szc72q)

[3.5.2 Reliability 9](#_184mhaj)

[3.5.3 Availability 9](#_3s49zyc)

[3.5.4 Security 10](#_279ka65)

[3.5.5 Maintainability 10](#_meukdy)

[3.5.6 Portability 10](#_36ei31r)

[3.6 Inverse Requirements 10](#_1ljsd9k)

[3.7 Design Constraints 10](#_2koq656)

[3.8 Logical Database Requirements 10](#_3jtnz0s)

[3.9 Other Requirements 10](#_4iylrwe)

[4. Analysis Models 10](#_1d96cc0)

[4.1 Sequence Diagrams 11](#_2ce457m)

[4.3 Data Flow Diagrams (DFD) 11](#_rjefff)

[4.2 State-Transition Diagrams (STD) 11](#_3bj1y38)

[5. Change Management Process 11](#_1qoc8b1)

[A. Appendices 12](#_1idq7dh)

[A.1 Appendix 1 12](#_42ddq1a)

[Document Approval 12](#_2hio093)

[A.2 Appendix 2 12](#_wnyagw)

[Revision History 12](#_3gnlt4p)

[A.3 Appendix 3 13](#_1vsw3ci)

# 1. Introduction

## 1.1 Purpose

The purpose of the Software Requirements Specifications is to provide a detailed description of the Healthy Bytes application. The intention of the SRS is to examine the purpose and features of the application along with the user and external interfaces, constraints, functional requirements, dependencies, and attributes. This artifact will provide the design and implementation of the software and explain the description of the application to the customer. This artifact is intended to be viewed by client, users, and developers.

## 1.2 Scope

The software application described within this SRS document is the Healthy Bytes application. This is a free desktop application that can used by any user, but it is primarily designed for individuals who are interested in cooking healthy meals or creating shareable recipes.

By using a Graphical User Interface (GUI), registered users can view, save and share recipes from a list of recipes stored in an existing MySQL database. The user will be able to navigate through recipes by first selecting a protein which then displays the names of dishes you can make with it. When a recipe is selected you can view more information about it, add it to your favorites, share the recipes with other users via email, and optionally generate a shopping list of ingredients from it as a text message, PDF, or text document. Additionally, users can also create their own recipes using a fillable form which will get added to the database.

## 1.3 Definitions, Acronyms, and Abbreviations

The following terms, acronyms, and abbreviations are used throughout the document which are presented in the table.

|  |  |
| --- | --- |
| **Term** | **Definition** |
| SRS | Software Requirements Specifications |
| GUI | Graphical User Interface. A visual interface as a means for a software user to manipulate controls, commands, and features of a software application. |
| Database | A collection of structured data that can be conveniently accessed by the software. |
| IDE | Integrated Development Environment; software that provides tools for the development and organization of programming code. |
| Git | A version control system for the development of software. |
| GitHub | A web-based repository provided for software development teams. |
| Java Virtual Machine | Provides the necessary links allowing a java program to run on a machine using a particular operating system. |
| Java Runtime Environment | Including the Java Virtual Machine, all necessary components for a system to establish the environment in which Java programs will run. |
| SQL | Structured Query Language; the standard relational database query language. |
| JDBC | Java Database Connectivity; an Java API developed by Oracle Corporation which provides methods for querying and updated a database. |

## 1.4 References

Git - <https://git-scm.com/>

GitHub - <https://github.com/>

Java Virtual Machine - <https://java.com/en/download/>

Java Runtime Environment - <http://www.oracle.com/technetwork/java/javase/downloads/jre8downloads-2133155.html>

JavaFX - <http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>

MySQL - <http://dev.mysql.com/downloads/mysql/>

NetBeans - <https://netbeans.org/>

## 1.5 Overview

The remaining content will go over the details of the application in the following separate sections: General Description, Specific Requirements, Analysis Models, Change Management Process, and Appendices. The General Description is created to have the Healthy Bytes application easier to understand for users from a high level point of view. Specific Requirements will discuss the requirements in detail giving the technical information needed by the developers. The Analysis Models will give a list of all the models that were used for creating the specific requirements. This section might see the most change since most of the requirements won’t be met right away and the SRS might see many changes during its lifetime. Change Management Process will give an outline of the procedures we must follow when attempting to make any changes to the Healthy Bytes application including the SRS document. Appendices will hold any documentation or diagrams that the developers may use. Some of these documents may or may not be used for requirement definitions, but further details pertaining to this will be specified for each of them.

# 2. General Description

## 2.1 Product Perspective

The Healthy Bytes is independent in that it does not attempt to improve upon any existing products. However, Healthy Bytes does make use of MySQL for its database capabilities. Connectivity to an existing MySQL server is included within the software. Healthy Bytes database will store preexisting recipes, in addition to storing recipes that users can optionally input which provides users with a centralized location to access a large variety of different dishes instead of scouring the internet. Healthy Bytes intends to replace products that charge users for access to healthy recipes, such as paperback Healthy Bytess and online subscriptions.

## 2.2 Product Functions

The Healthy Bytes application will have these functions:

* Store user login credentials in the database
* Store preexisting recipes in the database.
* Store user submitted recipes to the database.
* Provide users the ability to browse through a variety of recipe selections.
* Provide users the ability to share their own recipes by email.
* Generate a description of the recipe.
* Display an image of the recipe.
* Maintain a list of favorite recipes.
* Generate a text file or text message of the ingredients.

## 2.3 User Characteristics

Healthy Bytes will be primarily used by individuals who like cooking, want to share recipes they have created, or want to learn healthy recipes. Additionally, Healthy Bytes users could simply want to know a recipes nutritional information or use the recipe creator to send a quick shopping list.

## 2.4 General Constraints

The Healthy Bytes application will have the following constraints:

* Healthy Bytes must have a JavaFX-based GUI.
* Healthy Bytes must be developed in Netbeans IDE.
* Healthy Bytes must be MySQL database-driven.
* Healthy Bytes must be desktop-based.
* Choice of being Local or remote database server must be present.
* Git and GitHub must be used for version control.

## 2.5 Assumptions and Dependencies

The Healthy Bytes application will have the following assumptions and dependencies:

* It is assumed that Healthy Bytes will run on a system that will have an operating system or program that is capable to run a Java Virtual Machine and up to date Java Runtime environment.
* It is assumed that Healthy Bytes will run on or connect to a system that is using a MySQL server.
* It is assumed that in the event of remote MySQL server connectivity, Healthy Bytes will run network capabilities to be connected to said database.

# 3. Specific Requirements

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

Like most modern applications, The Healthy Bytes program will have a GUI that will allow the user to accomplish all of the goals necessary to create new accounts, save recipe lists, and allow them to send their recipes to another user via email or save it as a data file.

### 3.1.2 Hardware Interfaces

The hardware interfaces needed for this program are a standard keyboard and mouse.

### 3.1.3 Software Interfaces

The Healthy Bytes is a JavaFX program which interfaces with the Java Runtime Environment and Java Virtual Machine for whichever platform it runs on. Healthy Bytes must operate with a MySQL Server. Connectivity to the MySQL server is included with the software.

### 3.1.4 Communications Interfaces

Healthy Bytes can connect to a remote MySQL database which may require internet connectivity. Healthy Bytes can also send information to the database which will require internet connectivity. The application communicates with the MySQL server via a driver that is embedded in the software and uses SQL standards.

## 3.2 Functional Requirements

### 3.2.1 Store User Login Credentials

3.2.1.1 Introduction

The Healthy Bytes stores the user’s email and password into the database.

3.2.1.2 Inputs

The user will be asked to enter an email address and password of their choosing.

3.2.1.3 Processing

Once the credentials are entered they will be stored within the ‘accounts’ database table.

3.2.1.4 Outputs

Upon successfully entering the credentials, the user will be allowed access to the main menu screen.

3.2.1.5 Error Handling

The Healthy Bytes will validate if the email address entered contains the proper format. If not the login screen will reset with a warning message.

### 3.2.2 Store Recipes into User’s My Recipes List

3.2.2.1 Introduction

Healthy Bytes provides a sequential process to gather user information and recipes and store them in a database.

3.2.2.2 Inputs

The user provides their full name and username as well as recipe information and is stored in the database as a string text.

3.2.2.3 Processing

The user stores the collected data in the ‘user\_info’ and ‘user\_recipes’ tables of the database.

3.2.2.4 Outputs

A message will appear when item has been successful added.

3.2.2.5 Error Handling

If the recipe has already been added to the My Recipe list, the current state will refresh with a message warning the user the recipe cannot be added again.

### 3.2.3 Remove Recipes from User’s My Recipes List

3.2.3.1 Introduction

Removes currently selected recipe from the user’s My Recipe list.

3.2.3.2 Inputs

The user must push the ‘Remove Current Recipe’ button in the My Recipe’s state.

3.2.3.3 Processing

Once pressed the currently selected recipe will be deleted from the user’s My Recipes list and the database.

3.2.3.4 Outputs

A message will appear when recipe has been successful deleted.

3.2.3.5 Error Handling

If there are no recipes in the user’s My Recipe’s list then the current state will be refreshed with a message warning the user the list is empty.

### 3.2.4 Save the My Recipes List to a Data/Text File

3.2.4.1 Introduction

Takes the user’s current recipe list, and stores the information into a text/data file.

3.2.4.2 Inputs

User will be prompted to give a name of their file.

3.2.4.3 Processing

Once the name is entered and button has been pressed, the system will read through the user’s recipe list line by line and output the data to a text file.

3.2.4.4 Outputs

The text/data file will be created under the directory of the application folder.

3.2.4.5 Error Handling

The user will be prompted to re-enter the name of the file if the name previously entered includes special characters that cannot be entered as part of the file name.

### 3.2.5 Send User Recipe List to a contact via Email

3.2.5.1 Introduction

Grabs the user’s My Recipe list and send it to a target email.

3.2.5.2 Inputs

The recipe information from the database as a string text. User must provide an email address to send to target location.

3.2.5.3 Processing

The user must push the ‘Share’ button to send the recipe information.

3.2.5.4 Outputs

A message stating that the email has been successfully sent.

3.2.4.5 Error Handling

The Healthy Bytes will validate if the email address entered contains the proper format. A warning message will appear if invalid.

## 3.3 Use Cases

### 3.3.1 Create a Healthy Bytes Account

Goal: User creates an account for the Healthy Bytes application

Input: Email/Password

Output: User is granted access to the main menu of the application.

Main Scenario: User creates an account to store user login credentials in User database table

Precondition: The Healthy Bytes application needs to be running.

### 3.3.2 User Searches for Recipe

Goal: User selects a predetermined Recipe from an array of Recipes

Input: None

Output: Displays the Ingredients used for the recipe, and the steps to make the recipe.

Main Scenario: User wants to look at the recipes already generated by the application.

Pre Condition: User must have a registered account, and Healthy Bytes application must be running.

### 3.3.3 Adding a Recipe to Favorites

Goal: Saves the recipe into the User’s My Recipes list

Input: User pushes ‘Add To MyRecipes‘ button

Output: Adds the recipe and displays message it was added.

Main Scenario: User wants to save the recipe to email friends and family.

Pre Condition: User must have a registered account, and Healthy Bytes application must be running

### 3.3.4 Creating a Personal Recipe

Goal: Creates a new template for a new Recipe object to be added to the User’s My Recipe List

Input: User inputs the Ingredients needed for the recipe, and a message explaining the steps of how to create the recipe.

Output: Adds the recipe into the User’s My Recipes List.

Main Scenario: User would like to create a recipe that doesn’t exist within the Database.

Pre Condition: User must have a registered account, and Healthy Bytes application must be running

### 3.3.5 Send Recipes to an Email

Goal: Sends an email to a target recipient with the recipe/recipes attached to it.

Input: User inputs the target’s email address, subject of email, and message.

Output: None

Main Scenario: User wants to send the recipes (in text document form) saved on the directory of the application to a person via email.

Pre Condition: User must have a registered account, must have at least one or more text documents/files of recipe information, and Healthy Bytes Application must be running.

## 3.4 Classes / Objects

### 3.4.1 RB\_MainApp

3.4.1.1 Attributes

3.4.1.2 Functions

<Reference to functional requirements and/or use cases>

<< **THE FOLLOWING SPACE FOR THE REST OF SECTION 3.4 ARE RESERVED FOR POTENTIAL CLASSES THAT CAN BE USED FOR THIS PROJECT. The descriptions and details for these classes and objects will be updated accordingly during the design process >>**

### 3.4.2 Recipe\_Book

### 3.4.3 RB\_DatabaseManager

### 3.4.4 RB\_View

### 3.4.5 RB\_Controller

### 3.4.6 RB\_Model

### 

## 3.5 Non-Functional Requirements

### 3.5.1 Performance

There are no performance requirements during this time of the development of Healthy Bytes. The end product will be able to meet the performance at it’s typical standards.

### 3.5.2 Reliability

There are no reliability requirements during this time of the development of Healthy Bytes. The developers do plan on making the application as reliable as possible.

### 3.5.3 Availability

The information from the database will be available to use or view upon the request of the users. Should there be no issues with internet connectivity or the database server being inactive for the user.

### 3.5.4 Security

The following system will be implemented to avoid any compromise the user’s information. Users of Healthy Bytes must log in with their credentials (user name and password). The password must be hashed before being stored in the database. The user may request to change their password and will receive an email that they must verify to confirm the password change.

### 3.5.5 Maintainability

The development team will make use of specified coding standards aimed to help with codebase readability and maintenance. The coding standards for the development team can be found in Appendix 3.

### 3.5.6 Portability

Although, the requirement of the application Healthy Bytes must be in the Java programming language, it implies that it can be portable to any machine with an operating system. Ones that correspond to the Java Runtime Environment with the corresponding Java Virtual Machine that exists.

## 3.6 Inverse Requirements

The Healthy Bytes application will not function without the connection of the MySQL database. In case there is no connectivity to the database, the user will be notified and the application will shutdown.

## 3.7 Design Constraints

The Healthy Bytes application must be implemented in a sequential process, it will rely heavily on GUI elements to view and create recipes.

## 3.8 Logical Database Requirements

Healthy Bytes must be able to access and store data in a MySQL database. The only requirements for the types of data and capabilities are that they align with the completion of a working product that meet the requirements. Security must be implemented which is outlined in section 3.5.4 of this document.

## 3.9 Other Requirements

No other requirements are available at this point of development.

# 4. Analysis Models

There are no models currently available to be viewed at this stage of development for the application.

## 4.1 Sequence Diagrams

There are no sequence diagram available at this time that meet the requirements of the application. This section will be updated soon.

## 4.3 Data Flow Diagrams (DFD)

There are no data flow diagrams available at this time that meet the requirements of the application. This section will be updated soon.

## 4.2 State-Transition Diagrams (STD)

There are not state-transition diagrams available at this time that meet the requirements of the application. This section will be updated soon.

# 5. Change Management Process

The following procedure are required when changing this SRS document:

* Any sort of changes that are presented by the development team must be done independently by branching off from the original project on GitHub.
* The project manager must sign off and approve any changes that is agreed upon and will attach said document to the updated SRS.
* If changes are indicated by the customer, whether written or verbally, the project manager must create an updated copy of the SRS and attach the signed document.
* The project manager must present the updated SRS to the customer for final approval and the customer will then sign off on it.
* When both the project manager and customer sign approve the updates to the SRS, the project manager will then merge the official SRS with the updated SRS via Git.
* The project manager completes the procedure by recording the update in Appendix 2.

# A. Appendices

## A.1 Appendix 1

Required form for the approval of changes to this SRS document:

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
|  | <Your Name> | Lead Software Eng. |  |
|  | Dr. M.K. Quweider | Instructor, CSCI-3340 |  |
|  |  |  |  |

## A.2 Appendix 2

Required form must be attached to the end of this document if there are any changes after its initial completion:

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## A.3 Appendix 3

The following are the coding standards used by the development team. Following these standards is a self-imposed requirement agreed upon by the team members.

**CodingStandards:** This file is subject to change throughout the course of the project. If there needs to be a change or something added, contact the project manager so that we can discuss it.

##############################################################################

JDK: Java: 1.8.0\_162

##############################################################################

**FORMATTING:**

For our bracket formatting we are choosing to use new lines. So for example:

private void function()

{

//some code

}

As we know that netbeans formats the brackets differently we can change the settings by doing the following.

1. Navigate to Tools -> Options -> Editor
2. Navigate to Editor -> Formatting
3. Select the following
   1. Language: Java
   2. Category: Braces
4. In “Class Declaration”, “Method Declaration”, “Other”
   1. Braces Placement: New Line

To avoid any problems, before saving and especially before commits in Git, do Ctrl+A to select all of the code in the .java file and then do Alt+Shift+F

This will not only place brackets where they should be according to our new settings for NetBeans, it will format just about everything such as spaces on each side of an operator.

##############################################################################

**COMMENTS:**

We will use the javadoc comments (/\*\*\*Enter) for all classes and functions for easability.

For other comments, we will add them where they seem appropriate.

##############################################################################

**ERROR HANDLING:**

Use exception handling as needed to ensure we don’t miss any of them. Make sure that you use the proper log method that we decide on as a team for catches.

We will more than likely create our own logging method for errors soon. The details of this log class will be updated here once it has been completely designed along with examples on how to use it. In hindsight we do know that we will be using some form of code to dictate where the error is happening and what type of error it may be. The log might be stored in a text or dat file within the project directory.

If there is an error that should just be ignored, catch the exception and print it to the console, but allow the program to continue execution. This is needed sometimes. An example is when you traverse directories. In windows, some directories visible by java are actually junctions. isHidden() will return false, isDirectory() will return true, and canRead() will return true... however, when you have java attempt to list the files within the directory, it will cause an exception. Since a try and catch is the only way to test for junctions, the exception must simply be ignored so that the program can continue on.

##############################################################################

**INSTANTIATION AND INITIALIZATION**

All instantiation and initialization should be at the top of functions. An exception to this is when you need to create only temporary variables and objects used within an inner block of code, such as a loop.

##############################################################################

**NAMING CONVENTION**

##############################################################################

**CLASS FILE ORGANIZATION**

data variables

constructors

setters and getters (properties)

public functions

protected functions

private functions

subclasses

##############################################################################