# *Software Engineering*
# *Software Requirements Specification (SRS) Document*

**NutriLog: Smart Nutrition**

**February 12, 2024**

**0.1.0**

**By: Carlos Villarreal, Michael DeHaan, and Salvador Macias**

**In Adherence to the UNCG Academic Integrity Policy**

# Table of Contents

# 1. Introduction

## 1.1. Purpose

The goal of NutriLog: Smart Nutrition is to create a smart-food tracking application that allows users to see information about the foods they consume and provide nutritional education. The app will advise users on the nutritional values of the foods they consume so that they can make informed decisions on the foods they eat, ultimately leading to better health outcomes.

## 1.2. Document Conventions

The purpose of this Software Requirements Document (SDR) is to describe the client-view and developer-view requirements for the Nutrilog: Smart Nutrition web application. We will describe the system to a user and a software developer in it. Client-oriented requirements will detail the system from the perspective of a regular user. The user types will be administrator, nutrition specialist, and user. The requirements also include a description of the user types. Developer-oriented requirements describe the system from the perspective of a software developer. The software requirements include detailed information about functionality, data, performance, and other requirements.

## 1.3. Definitions, Acronyms, and Abbreviations

| | |
|---|---|
| Java | A programming language originally developed by James Gosling at Sun Microsystems. We will be using this language to build NutriLog. |
| HTML | Hypertext Markup Language. This is the code that will be used to structure and design the web application and its content. |
| CSS | Cascading Style Sheets. This is the code that will be used to style the content of the web application. |
| SpringBoot | An open-source Java-based framework used to create a micro Service. This will be used to create and run our application. |
| MVC | Model-View-Controller. This is the architectural pattern that will be used to implement our system. |
| Spring Web | Will be used to build our web application by using Spring MVC. This is one of the dependencies of our system. |
| Thymeleaf | A modern server-side Java template engine for our web environment. This is one of the dependencies of our system. |
| IntelliJ IDEA | An integrated development environment (IDE) for Java. This is where our system will be created. |
| API | Application Programming Interface. |
| FDC API | An API that provides access from the USDA Global Branded Foods Database. This is where the core backend functionality will come from. |

## 1.4. Intended Audience

The roles of stakeholders, developers, and project managers are all shared by the developers who created NutriLog: Carlos Villarreal, Michael DeHaan, and Salvador Macias. The user role belongs to anyone who wishes to use the service.

- Parts 1 and 2: Offer high-level overviews of the product that all roles may benefit from viewing.
- Part 3: Offers more detailed views for the developers and project managers.
- Parts 4 and 5: Offers views created based on user/customer specifications but within the SRS are meant for developers and project managers to implement these user-defined high-level requirements.

## 1.5. Project Scope

The goal of NutriLog is to offer individuals a comprehensive and easy-to-use platform to take control of their daily nutrition. This aligns with the goal of promoting health and well-being among users by enabling them to make informed food choices based on their personal goals.

The benefits of the project to individuals include:

- Enhancing food safety by flagging recipes that contain foods a user may be allergic to.
- Promoting overall well-being by ensuring users have a more nutritionally complete diet.
- Improving nutritional education by providing specialist services to users who need additional assistance.

## 1.6. Technology Challenges

The challenges from this project will be from frontend implementation and API integration. Every developer team member is familiar with Java and backend logic but not UX and UI development. The new technologies will be SpringBoot and integrating the FDC API with it.

## 1.7. References

Academic integrity policy. Student Affairs. (2023, November 3). https://sa.uncg.edu/division-of-student-affairs/students/academic-resources/student-policy-handbook/academic-integrity-policy/

# 2. General Description

## 2.1. Product Features

The product features allow users to find recipes and save them and flag recipes they search for any potential ingredients they may be allergic to. The users are also able to request for specialist assistance allowing them to obtain help in creating a more healthy and balanced diet. The specialist can then create a meal plan based on the specific nutritional needs of the user. The admin can create new accounts and delete ones, generate reports, and view metadata.

## 2.2. User Class and Characteristics

Our application only expects our users to have prior knowledge of computers from using a web browser. NutriLog is intended for users with any background in nutrition, ranging from no prior nutritional knowledge to expert knowledge. A basic understanding of dietary components such as micro and macronutrients may be beneficial but is optional.

## 2.3. Operating Environment

This application is designed to operate on any modern web browser regardless of the device.

## 2.4. Constraints

Due to safety considerations and project scope, contact between specialists and users will not be instant.

## 2.5. Assumptions and Dependencies

The software will be dependent on Spring Boot, HTML, CSS, and Java. The software will also use the FDC API for searching and displaying recipes, and food items. The software will be developed using IntelliJ IDEA.

# 3. Functional Requirements

## 3.1. Primary

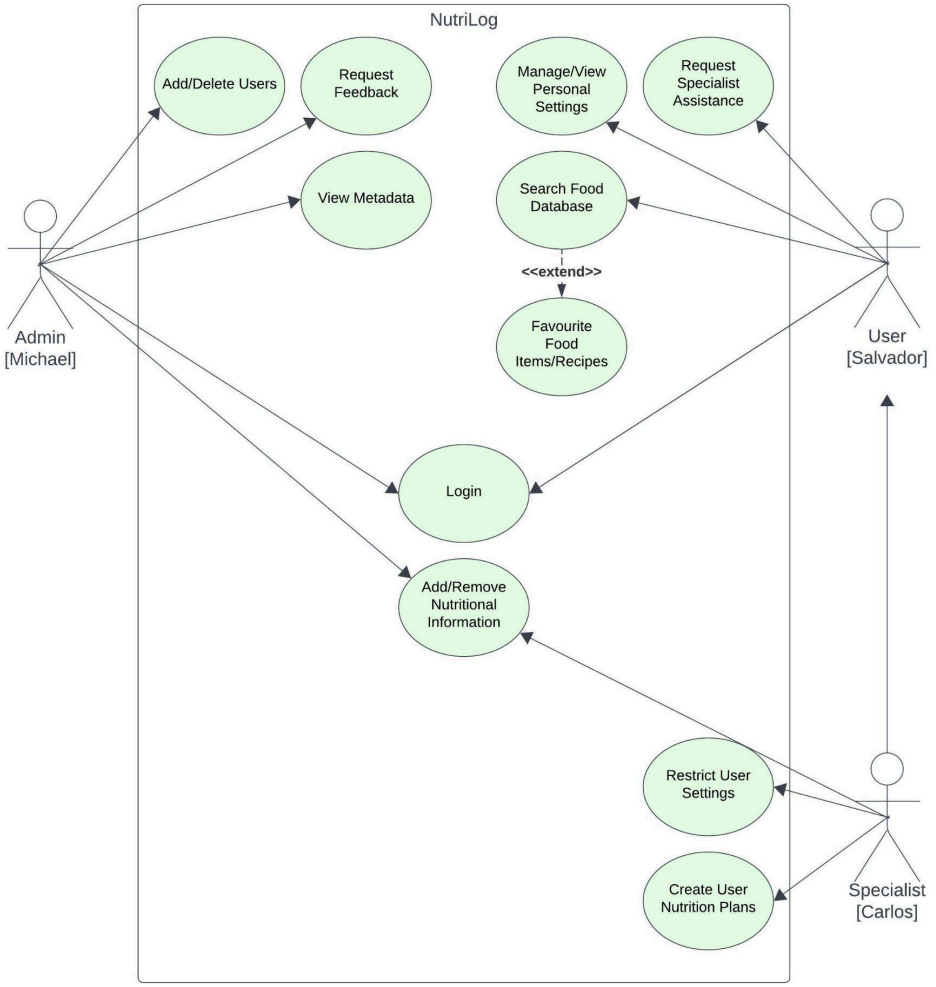
- FR0: The system will allow users to input dietary settings such as allergies. These settings will be used as filters and flags when searching for food items and recipes.
- FR1: The system will allow the user to request specialist assistance. This will alert specialists that a user is requesting assistance.
- FR2: The system will allow specialists to modify nutritional information tailored to users' settings.
- FR3: The system will allow administrators to delete user and specialist accounts described by their IDs.

## 3.2. Secondary

- All user types have a unique ID and password.
- Users are only able to view their own data.

## 3.3. Use-Case Model

### 3.3.1. Use-Case Model Diagram



### 3.3.2. Use-Case Model Descriptions

#### 3.3.2.1. Actor: Admin (Michael DeHaan)

- **Add/Delete User**: The admin can create or delete a user based on the user's ID.
- **Request Feedback:** The admin can send a feedback request to other users to gain insight on their experience using the app.
- **View Metadata**: The admin can view analytic data about users to see how they are using the app.

#### 3.3.2.2. Actor: Specialist (Carlos Villarreal)

- **Create User Nutrition Plans**: The specialist will be able to create a Nutrition plan for a user who requests a nutrition plan.
- **Restrict User Settings**: The specialist can restrict the settings of a user based on certain criteria, such as their age. Children users will have less features than adult users.
- **Search/Save Recipe and Food Items:** The specialist will have the ability to search and save recipes that they find so that they can later make recommendations to users. The same ability will be granted to food items.

#### 3.3.2.3. Actor: User (Salvador Macias)

- **Manage personal settings**: The user will be able to enter his name and allergy information along with any other relevant data to create a food plan.
- **Request special assistance**: The user will be able to ask for assistance from a specialist.
- **Search/Save Recipe and Food Items**: The user will be able to search for and save recipes that they would like to try based of the relevant information they gave such as allergies.

### 3.3.3. Use-Case Model Scenarios

#### 3.3.3.1. Actor: Admin (Michael DeHaan)

- **Use-Case Name**: Add/Delete User
    - **Initial Assumption**: The admin has access to the app and the admin role.
    - **Normal**: The admin will enter the user's ID to an add/delete function.
    - **What Can Go Wrong**: The admin enters the wrong ID.
    - **Other Activities**: n/a
    - **System State on Completion**: The user will either be added or no longer be registered. The ID will be either set to active or inactive.
- **Use-Case Name**: Request Feedback
    - **Initial Assumption**: The admin has access to the app and the admin role.
    - **Normal**: The admin will enter the user's ID to a feedback function.
    - **What Can Go Wrong**: The admin enters an invalid ID.
    - **Other Activities**: The admin can specify a feedback message.
    - **System State on Completion**: The user will be notified of a feedback request.
- **Use-Case Name**: View Metadata
    - **Initial Assumption**: The admin has access to the app and the admin role.
    - **Normal**: The admin will select a view to see various metadata about the app and its users.

- **What Can Go Wrong**: Changes to the software could make the metadata invalid.
- **Other Activities**: The admin can create custom reports based on the metadata.
- **System State on Completion**: No state change, view only.

#### 3.3.3.2. Actor: Specialist (Carlos Villarreal)

- **Use-Case Name**: Create User Nutrition Plans
  - **Initial Assumption**: The specialist has access to the web application and has the specialist role.
  - **Normal**: The specialist will be able to create custom nutrition plans for users based on the feedback users give for what their dietary needs are.
  - **What Can Go Wrong**: User's dietary needs are not given to the specialist because they did not fill out a form of some type to give to the specialist. The Specialist should have an "other" option.
    The user is unwilling to give health and dietary information necessary to create a custom nutrition plan. The specialist will then inform the user somehow that they are limited with the options they can give in a nutrition plan or that they are unable to create a suitable nutrition plan.
  - **Other Activities**: (Placeholder, details can be added later)
  - **System State on Completion**: The new nutrition plan is created and shown to the user somehow.
- **Use-Case Name**: Restrict User Settings
  - **Initial Assumption**: The specialist has access to the web application and has the specialist role.
  - **Normal**: The specialist is able to restrict the settings that a user can have based on certain user information, such as their age.
  - **What Can Go Wrong**: The specialist accidentally restricts the setting of an adult user. The specialist should have the ability to remove restrictions.
  - **Other Activities**: (Placeholder, details can be added later)
  - **System State on Completion**: The child user no longer has access to certain settings.
- **Use-Case Name**: Search/Save Recipes and Food Items
  - **Initial Assumption**: The specialist has access to the web application and has the specialist role.
  - **Normal**: The specialist has the ability to save recipes and foods that they find somewhere so that they can refer to them later when they make recommendations to users.
  - **What Can Go Wrong**: n/a
  - **Other Activities**: (Placeholder, details can be added later)
  - **System State on Completion**: The saved recipes/food items are saved somewhere in the system so that the specialist can easily find them later.

#### 3.3.3.3. Actor: User (Salvador Macias)

- **Use-Case Name**: Manage personal settings
    - **Initial Assumption**: The user has access to the web browser and is logged in.
    - **Normal**: The user accesses and can modify his personal settings and information..
    - **What Can Go Wrong**:
    - **Other Activities**:
    - **System State on Completion**:
- **Use-Case Name**:
    - **Initial Assumption**:
    - **Normal**:
    - **What Can Go Wrong**:
    - **Other Activities**:
    - **System State on Completion**:

# 4. Technical Requirements

## 4.1. Interface Requirements

### 4.1.1. User Interfaces

There will be three different interfaces based on the three user types: Admin, Specialist, and User. The app aims to be user-friendly and easily accessible. Upon logging in, each user will be directed to their homepage. Each page will have a navigation bar at the top of the viewport that can direct them to the subpages of their user type by using links and buttons. Input will be specified by using various HTML tags such as select, checkbox, etc. Styling will be done primarily through CSS and CSS templates.

### 4.1.2. Hardware Interfaces

The app will run locally on any modern web browser that is connected to the internet and has the ability to interact with web page components. This includes but is not limited to, smartphones, tablets, desktop computers, and laptops.

### 4.1.3. Communications Interfaces

It must be connected to the internet to connect to the FDC API. Local communication between specialists and users does not require an internet connection, but they must be on the same local network or machine. Communication between users, the FDC API, and specialists will all be done via HTTP.

### 4.1.4. Software Interfaces

We will use Spring Boot and ThymeLeaf to build the front end and Microsoft Excel for the backend database functionality. We will also use Spring Boot with Java to connect the front end to the back end.

# 5. Non-Functional Requirements

## 5.1. Performance Requirements

## 5.2. Safety Requirements

## 5.3. Security Requirements

## 5.4. Software Quality Attributes

### 5.4.1. Availability

### 5.4.2. Correctness

### 5.4.3. Maintainability

### 5.4.4. Reusability

### 5.4.5. Portability

## 5.5. Process Requirements

### 5.5.1. Development Process Used

### 5.5.2. Time Constraints

### 5.5.3. Cost and Delivery Date

## 5.6. Other Requirements