Software Requirements Specification

For

Personal Food Log App

Version 1.0

Prepared by Group 6: Longxuan Zhao 301385113
Junchen Li 301385486
Jianan Xu 301368689
Weilong Xu 301385645
Zikun Zhang 301386607

April 10, 2022

Content

1. Introduction **1**

1.1 Purpose 2

1.2 Scope 2

1.3 Product overview 2

1.3.1 Product perspective 2

1.3.2 Product functions 2

1.3.3 User characteristics 2

1.3.4 Limitations 2

1.4 Definitions 3

2. References 4

3. System requirements 4

3.1 Functions 4

3.2 Performance requirements 5

3.3 Usability requirements 6

3.4 Interface requirements 6

3.5 Logical database requirements 6

3.6 Design constraints 7

3.7 Software system attributes 7

3.8 Supporting information 9

4. Verification 9

5. Appendices 11

5.1 Acronyms and abbreviations 11

1. Introduction

1.1 Purpose

The SRS document aims to outline the Personal Food Log Application Software requirements. It also helps users to understand the functions of the system, and how the website can serve them. Therefore, we provide some explanation and proof of main system requirements about this software. In this document, we will describe the product overview as well as the categorized requirements. Verification process for each requirement will then be detailed.

1.2 Scope

The Personal Food Log software allows users to input and track simply using their mobile device's camera. This application will host this collected data in a database on a remote server where it can be delivered to users when requested or used to improve the underlying algorithm. Once the algorithm has identified the foods and weights in the photo it will calculate the macronutrients and calories of the food based on a conversion table stored online. The application will allow users to visualize their historical data analysis by pulling the data from the database.

1.3 Product overview

1.3.1 Product perspective

The main application homepage/home screen is simple and descriptive. It clearly gives the user the ability to select between the different functions of the application.

1.3.2 Product functions

The application allows the user to take photos through the embedded device camera through a user interface. It also allows users to look through their user history in terms of scanned images and the nutritional data which concerns them.

1.3.3 User characteristics

Expected user characteristics are not restricted. The application can predict and analyze users with any characteristics.

1.3.4 Limitations

The supplier in this system is Amazon. The product of Amazon Cloud Service cannot expose data, so it is necessary to keep data well. It is better to have backup data in case of data loss. And the Cloud services should bot have in data import and output parallelly, such as confusing customer data. Moreover, the services need to protect from hackers like DDos attack.

1.4 Definitions

(a) big data:

Sets of information that are too large or too complex to handle, analyze or use with standard methods.

(b) Database:

An organized set of data that is stored in a computer and can be looked at and used in various ways.

(c) Software administrator:

Manage all aspects of the software, ensure the safety of the software operation, audit each user and the information they release, and update the database. This type of user has the highest authority over the software system.

(d) Server:

A computer or computer program which manages access to a centralized resource or service in a network.

(e) Machine Learning:

The use and development of computer systems that are able to learn and adapt by using algorithms and statistical models to analyze and draw inferences from patterns in data.

(f) AI:

This new technology uses AI to recognize character features in the same way a human brain does.

(g) Algorithm:

A process or set of rules that is followed in calculations or other problem-solving operations, especially by a computer.

(h) User:

A person who uses or operates the application.

(i) Nutrition:

The process by which living things receive the food necessary for them to group and be healthy.

(j) Calorie:

A unit for measuring how much energy food will produce.

2. References

1. IEEE 29148-2018 Requirements Engineering. (2018, November).
2. *Oxford Learner's dictionaries: Find definitions, translations, and grammar explanations at Oxford Learner's dictionaries*. Oxford Learner's Dictionaries | Find definitions, translations, and grammar explanations at Oxford Learner's Dictionaries. (n.d.). Retrieved March 26, 2022, from https://www.oxfordlearnersdictionaries.com/us/

3. System requirements

3.1 Functions

3.1.1. Validity checks:

A security layer is required on the back end of the system, especially close to the database layer. The main purpose of validating input is to prevent malicious network attacks, i.e., hackers trying to break into the database. Although our database contains sensitive information of our customers, we need to ensure that the database cannot be compromised. There are several common types of malicious attacks that require our attention, such as SQL injection, DDoS attacks, or cross-site scripting attacks. A software component needs to be specially developed to handle malicious attacks and should run when the server is up and running. Another input check we need to do is to make sure that the data the user enters for the system is in the correct format. Unlike how we handle malicious attacks, this can be done on the front end, where the script can check the user input format. For example, when a user uploads an image, a validity check needs to be done to ensure the image is of the correct size and focus.

3.1.2. Exact sequence:

The user enters the information to register, checks whether the user's information already exists, sends the user registration confirmation email, the user types in the username and password, the database retrieves the user information based on the value of the user information, the user starts a food classification session, and the back-end code retrieves the appropriate from the database. The pre-trained model, the scanned image is sent to the database that feeds the model, the input image data is fed into the convolutional neural network model to classify the image content, the content is sent back to the user's device, the user is asked if they are satisfied with the results, and the user types in the correct label , send the corrected labels back to the backend to retrain the model.

3.1.3. Responses to abnormal situations

3.1.3.1. Overflow:

Always conduct a workload test such as JMeter to test out the server workload capability. During an abnormal overflow situation, an immediate action needs to take place to store the current state of the server and the database.

3.1.3.2. Communication Facilities:

When a problem is caused by insufficient communication, the cause of communication needs to be discussed and analyzed for how to avoid it in the future.

3.1.3.3. Hardware failures:

Before a possible abnormal hardware failure happens, the maintenance engineers need to ensure the availability and parallelism of the system, to ensure when a disaster occurs, there is another set of hardware's can take the workflow immediately.

3.1.3.4. Error handling:

the current state of the system and the database need to be stored consistently. When an error occurs, a certain diagnosis procedure needs to be performed automatically and take action to revert to the previous state.

3.1.4. Hyperparameters:

During the model training process, it is important to be careful of setting the hyperparameters, such as number of epochs or hidden layer dimensions, because these parameters can easily overload the server.

3.1.5. CNN Model:

For the CNN model, the output is the predicted or classified label of the input images. To convert back to the corresponding images, all the inputs and outputs need to be stored in a relational type of datastore.

3.2 Performance requirements

3.2.1. Dynamic numerical requirements:

- 80 % of the failover recovery should be conducted within 10 minutes
- 95 % of the transactions shall be processed in less than 0.5s
- 90 % of the AWS EC2 instances should be up within 5 minutes

3.2.2. Static numerical requirements:

- The number of concurrent AWS EC2 instances to support availability is: 20
- The number of simultaneous users to be supported:300
- The amount of software version update per months is: 1

3.3 Usability requirements

1. The down time caused by software version update should be quick that won't exceed a day to avoid inconvenience of the users.

2. The UI of the homepage needs to be clear and distinctive, so that users can easily navigate through all the available functionalities.

3. The statistical analysis of user's daily calories assumptions need to be informative and tailored down to specific user needs.

4. The overall app use experience should be smooth; users shouldn't experience any noticeable lag.

5. The system should be able to support different user device types.

6. The database should be able to prevent typical malicious attacks such as SQL Injection, DDoS attack, or Cross-Site Scripting.

3.4 Interface requirements

1. The color of the interface needs to conform to the public's aesthetic.

2. The responsiveness of clicking a button should be quick without noticeable processing time.

3. The scan functionality should be smooth when a user is trying to open the camera.

4. There should always be a "back" button for users to get back to the previous screen anytime they desire.

5. The front size displayed on the interface needs to match the system font size.

6. The panels of the interface need to be more suitable, and the priority of important content.

3.5 Logical database requirements

3.5.1. Types of information used by various functions:

i. The user information including age, sex, weight, height etc. should be stored as a relational datastore type.

ii. The user calorie consumption history should also be stored as a relational datastore type.

iii. The history of user requests should be stored as non-relational datastore types.

iv. The pre-train models should be stored as bytes datastore type.

3.5.2. Frequency of use:

All the different variations of databases should be able to access frequently as the number of users increase. They should be highly available to access.

3.5.3. Accessing capabilities:

The assessing capability should be both quick and restrictive. Being quick ensures better user experience and synchronous updates. Being restrictive ensures only transactions and administrator behaviors may access the database, which are stored with sensitive user data.

3.5.4. Data entities and their relationships:

Entities: User, Account, Model, History Profile, Food Profile. Users is a one-to-one relationship with Account, Model is a one-to-many relationship with Food Profile, Account is a one-to-many relationship with History Profile.

3.5.5. Integrity constraints:

All updates to the database should follow a specific set of integrity constraints. A successful calorie input session should update both entity User and entity History Profile. The user entity should contain username, password upon creation before any other input attributes. A specific numerical index should be set for each entity for quick access and uniqueness.

3.5.6. Security:

The database should be able to prevent typical malicious attacks such as SQL Injection, DDoS attack, or Cross-Site Scripting. Information should only be sent to users who currently logged in using the exact same username and password in the current session.

3.5.7. Data retention requirements:

In general, data retention should meet both the system requirements and most importantly, the Data Protection Act. User information should only be kept if the user is still active or has not yet deleted their account. Other information such as food profiles, models, or session related information should be kept if there is specific functionality using them. These data should be archived regularly to save spaces.

3.6 Design constraints

There are different types of design constraints in this project: hardware, software, UI, and database. The visual design constraints are normally specified by users using a specialized tool to render each screen they want to appear. The hardware constraints in this project are the usage of AWS cloud computing instances. Due to the limited project budgets, we need to restrict the amount of simultaneously running AWS services should not exceed a certain amount so that we don't get charged by AWS for exceeding the budget. Every unused AWS service instance should be closed to reduce cost. The database constraint is that we need to purchase a datastore which is relatively fit to our data flow size. A bigger datastore has a higher cost. We need to design the type of datastore and the capacity of the datastore. When picking a datastore, we need to take the database scalability into consideration.

3.7 Software system attributes

3.7.1. Reliability:

At the time of delivery, every component of the system that is required to run the software should be up and ready to take requests. In specific, the front-end servers need to be up and ready to take http connections, they need to be able to transfer and send the request to assigned back-end servers and map the applications to assigned ports. The back-end server needs to be ready to communicate with the database server to update and retrieve information in real time. The distributed database servers need to be able to load the pre-trained models to the back-end server to make prediction or classification. Before the time of delivery, both manual testing and automation testing need to be finished in several rounds to ensure the quality of delivery. A DevOps process should be ready to monitor the current states of the servers.

3.7.2. Availability:

The system states including AWS EC2 instance states and database server state are needed to be stored consistently. This is to ensure during a disaster or unpredicted event happens, we can get back to the previous state of the whole system safely.

3.7.3. Security:

To ensure data security, The system should be able to prevent typical malicious attacks s.t SQL Injection. This is mainly done in the back-end server code for accepting user inputs. It is required to keep the log and history of the current state of the system for future reference. The sensitive user data should be stored in a separate database, which is designed specifically for security even with some

level of latency towards the whole system. This special tasked database server needs to only accept connections from only one source to restrict communication from other areas that could potentially increase the risk.

3.7.4. Maintainability:

Developers should ensure the modularity of the code, to ensure the code is reusable and easy to maintain. Code should be split reasonably into different modules or classes. APIs should be encapsulated into classes with some hierarchy. Then, each major component of the code should be packaged separately. In-code documentation is required to specify use cases, inputs, and outputs of certain modules or functions. Detailed overall system documentation needs to be tailored for each use case, including how to start the system from scratch, what to do during unexpected events, and how to verify the current state of the system, etc.

3.7.5. Portability:

Delegated DevOps teams should package the components of the system. Specifically, Docker can be used to encapsulate components such as front-end servers, back-end servers, and SQL databases. This way, the system does not depend on existing code on existing servers. Components can be easily migrated to other servers or platforms. The percentage of elements containing host-dependent code should be less than 50%. The percentage of host-dependent code should be less than 30%. Code should be developed in a commonly used language. In this project, we want to be able to deploy the backend code to a Linux based environment. Also, both IOS and Android platforms need to be supported. Therefore, two sets of parallel front-end application development are required.

3.8 Supporting information

Send a survey to registered users every month. The survey will be based on multiple choice questions and paired text answers. The multiple-choice results are stored in a relational table and further analyzed as the average satisfaction level of the user. The input questionnaire will be fed into a pre-trained model for analysis using deep learning. Negative comments will be singled out for review by human employees and summarized into a more structured set of feedback.

4. Verification

4.1. Functions

4.1.1. Validity checks:

Firstly, Simulate some malicious attacks s.t SQL Injection. Then, input some incorrect format of data. Finally, check the request whether be rejected.

4.1.2. Exact sequence:

Check the logs to see if the sequence of operations of operations is working as expected.

4.1.3. Responses to abnormal situations

i. Overflow: Use JMeter to test server workload capabilities. Simulate an exception overflow situation and check that the current state of the server and database is preserved.
ii. Communication Facilities: If a problem is caused by insufficient communication, the cause of communication needs to be discussed and analyzed for how to avoid it in the future.
iii. Hardware failures: The availability and parallelism of the system should be checked biweekly.
iv. Error handling: Simulate an error, and check if it resumes the previous state data.

4.1.4. Effect of parameters:

The hyperparameters should be checked if it causes the server to be overloaded and if it could meet accuracy standards.

4.1.5. relationship of outputs to inputs:

Check all the inputs and outputs of the CNN stored in a relational type of datastore.

4.2. Performance requirements

4.2.1. Static numerical requirements:

i. Check the number of concurrent AWS EC2 instances to support availability is 20.
ii. Check the number of simultaneous users to be supported is 300.
iii. Check the amount of software version updates is once a month.

4.2.2. Dynamic numerical requirements:

i. 80 % of the failover recovery should be conducted within 30 minutes
ii. 95 % of the transactions shall be processed in less than 0.5 s
iii. 90 % of the AWS EC2 instances should be up within 5 minutes

4.3. Usability requirements

1) Make sure the down time of software update must be under one day before proceeding.
2) Before publishing each UI pattern, the UI of the homepage needs to be clear and distinctive.
3) The statistical analysis of user's daily calories assumptions needs to be informative and tailored down to specific user needs.
4) Make sure the software can run on both iOS and Android.
5) Simulate SQL Injection or DDoS attack to check the security of the database.
6) Check if the system can revert to a previous state after an abnormal failover.

4.4. Interface requirements

1) Check there is no noticeable processing time of clicking

any button.

2) The scan functionality should be smooth when a user is trying to open the camera.
3) Check there is always a way for the user to go back.
4) Check the font, panels and the color are suitable for the users by conducting market research.

4.5. Logical database requirements

- For every update on the database, the requirements on the database should be reviewed before release.

4.6. Design constraints

- For the update on design of hardware, software, UI, and database, the requirements on the design should be reviewed before each release

5. Appendices

5.1 Acronyms and abbreviations

Acronym/Abbreviation	**Definition**
Healthy eating plan	A healthy eating plan gives your body the nutrients it needs every day while staying within your daily calorie goal for weight loss. A healthy eating plan also will lower your risk for heart disease and other health conditions.
CNN	Convolution Neural Network, a class of artificial neural network (ANN), most commonly applied to analyze visual imagery.