Software Requirements Specification

for

EurekaEats

Version 1.1.1 approved

Prepared by Dana Mendoza, David Tabor, Derek Tan, Devin Chang, Iqra Irfan

CSULA

10/21/2023

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Derek T.	10/22/23	Update application message notes.	1.1.1

1. Introduction

The software requirement specification(SRS) is a document that describes what the software will do and how it will be expected to perform. The main audience for the SRS are developers, testers, and project managers..

1.1 Purpose

The purpose of this document is to plan and decide what specifications will be included in the document. The software engineers involved in this project will better understand the requirements, the breakdown of the project, and individual software related to this project.

1.2 Intended Audience and Reading Suggestions

This document is intended for developers and testers to understand the requirements and purpose of the EurekaEats web application. Sections one and two give an overview of the project, sections three to five are specific functional requirements, and section 6 covers other concerns within the scope of this document. The developers should be familiar with the entirety of this document. Software testers should be familiar with the first four sections.

1.3 Product Scope

Software Product: EurekaEats

This software is a website that combines the specific food preferences of a user and finds a local, best-fit restaurant by those factors. Website users can write reviews, rate reviews, and find restaurants. The benefit of the software after release consists of a new foodie base community. The main use of EurekaEats website by any user is to select the best dining location for themselves rigorously. Another primary use of the product by any user is to assist other users in choosing optimal dining locations through reviews.

1.4 Definitions, Acronyms, and Abbreviations

TBD means “to be determined”.

1.5 References

- Software Requirements Specification document with example - Krazytech
- Server Size Guidelines – Logi Analytics

2. Overall Description

The website will allow users to create a profile, search for restaurants, and leave reviews. The search will be able to find restaurants for users based on their eating preferences.

2.1 System Analysis

The most important goals of the project are to offer a more fitting service to find and select the best restaurants for any user and to serve as an informal reference to inform users on which restaurants are most desirable or not.

The technical hurdles of this project include designing an algorithm to find the best restaurant within constraints, creating a database schema, and creating a user-friendly, accessible UI.

Our software development team will research and design a UI, database schema, and have multiple coding reviews to fit the requirement.

2.2 Product Perspective

EurekaEats is comparable to Yelp. EurekaEats is different from Yelp in that users are able to define more specific eating requirements when searching for restaurants.

User's information:

User information includes the user's eating preferences(cuisine, vegan, protein, keto, allergies, and more). This information is used to find a restaurant for the user.

Restaurant information:

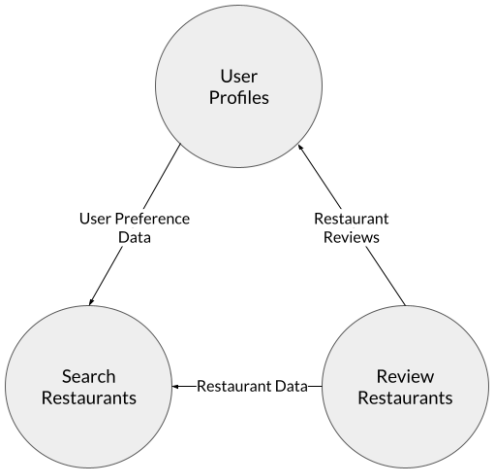
Restaurant information includes restaurant address, type of cuisine, price range, ambiance, Wi-Fi accessibility, and more. This information is used for users to see what type of restaurant it is.

? DIAGRAM HERE ?

2.3 Product Functions

Users should have the ability to use these functions on our website.

- User Profiles - Users can create an account where their eating preferences, restaurant reviews, and favorites will be saved.
- Search Restaurants - Users can search for restaurants based on the preferences voluntarily submitted to their profile.
- Review Restaurants - Users can review a restaurant on the website to let other users be more aware of the foods they serve and the level of service.



2.4 User Classes and Characteristics

- Restaurant Owner - Wants to claim their restaurant and list their food
- Power Reviewer (foodie type/yelp critic) - Reviews restaurants often.

- Casual User - Mostly searches for restaurants and sometimes may review them.

2.5 Operating Environment

The technologies used on this platform will be platform-independent. Specifically, any desktop or server-oriented operating system should be able to run high-level program languages that are already independent of any hardware or OS..

2.6 Design and Implementation Constraints

Items that can influence the ability of the software developers to implement the product would be system overloading due to the amount of users using the website.

Give a general description of any items that will influence the ability of the software developers to implement the product. These can include things such as:

- Interfaces with other applications
    - The website will use 3rd party APIs such as the Yelp API for restaurant data and the Google Maps API for location data.
- Reliability requirements:
    - The software must be reliable in performance and uptime. Page operations should take at most 3 seconds, and the website runs should not crash more than 10% of the time.
- Safety and Security Considerations:
    - Since user-inputted data is unverified, the application must validate it based on the data format, the accepted data values, and the message format. For formatting, poorly formatted messages and data sent to or from the website could cause errors or crashes if not handled. Also, validating data for appropriate values ensures that the data fits realistic constraints: star ratings cannot be negative, etc.
    - Since passwords are crucial for users to access their profiles, storing them as plain text will allow any database breach to expose sensitive data. If attackers somehow access the database and steal passwords, they could hijack user accounts or expose their personal information. Thus, we will consider using string hashes to protect passwords.
- Memory Constraints
    - Other programs may be running on test machines for *localhost* software versions, so the software must not take excessive RAM. A possible constraint can be that the software should not have an out-of-memory error.

2.7 User Documentation

TBD

2.8 Assumptions and Dependencies

The cost of API calls can change and free products can become unavailable. If a service we use (such as Google Places) changes pricing, we may need to change our strategy.

3. External Interface Requirements

3.1 User Interfaces

Design for user interfaces will be designed in figma. Once design is completed, our team will upload the design to match our website. The user interface will be set up as a website and we will store user inputs into our database. Once inputs are collected into our database, we can use those data to run functions for our website. Website font and size has to be clear for the users to see and understand.

3.2 Hardware Interfaces

This software does not have hardware interface requirements.

3.3 Software Interfaces

- Browsers & Versions
    - Chrome V117.0
        - https://www.google.com/chrome/
    - Firefox V118.0
        - https://www.mozilla.org/en-US/firefox/new/
- APIs
    - Yelp Fusion (?)
        - https://fusion.yelp.com/
    - Google Places (?)
        - https://developers.google.com/maps/documentation/places/web-service/overview
- Database:
    - MongoDB 7.0
        - https://www.mongodb.com/try/download/community-edition
- Programming Languages:
    - Python 3.12
        - https://www.python.org/downloads/release/python-3120/?ref=upstract.com
        - Replaced our original choice of Java by group consent.
    - Javascript: ES6+
        - TBD

3.4 Communications Interfaces

Network protocols:

- Application: HTTP/1.1 (common on local test servers)
- Transport: TCP (almost everywhere) & IP addressing (everywhere)

Message Action Codes: Denotes application API message tasks like “verbs”.

- **NOTE: The *application* API Codes are organized by groups of up to 16 possible actions per part: restaurants, reviews, and users. This is to anticipate future changes.**
- 0: Search for restaurants.
- 1 to 15: **Unspecified for future use.**
- 16: Search for reviews.
- 17: Create a review for a valid user.
- 18: Update an existing review for only the author.
- 19: Delete an existing review for only the author.
- 20 to 31: **Unspecified for future use.**
- 32: Get user information.
    - Private Info Arguments: session cookie and user password hash for authorizing a complete fetch.
    - Public Info Arguments: both or any of the private info arguments are `null`. Only gets the username and user summary.
- 33: Update user profile.
    - Arguments: A JSON object with keys of profile setting names to the new setting values. Values include username, email, password, and summary.
- 34: Update user food preferences.
    - Arguments: A JSON object with keys of preference names to values of the new preference settings.
- 35 - 46: **Unspecified for future use. These codes may become admin actions to moderate user content.**
- 47: **Dummy API call for testing. Does nothing but say “Hello World!” and needs no arguments.**

Message Payload Codes: Means the data type of the response to any API call.

- 0: number
- 1: string

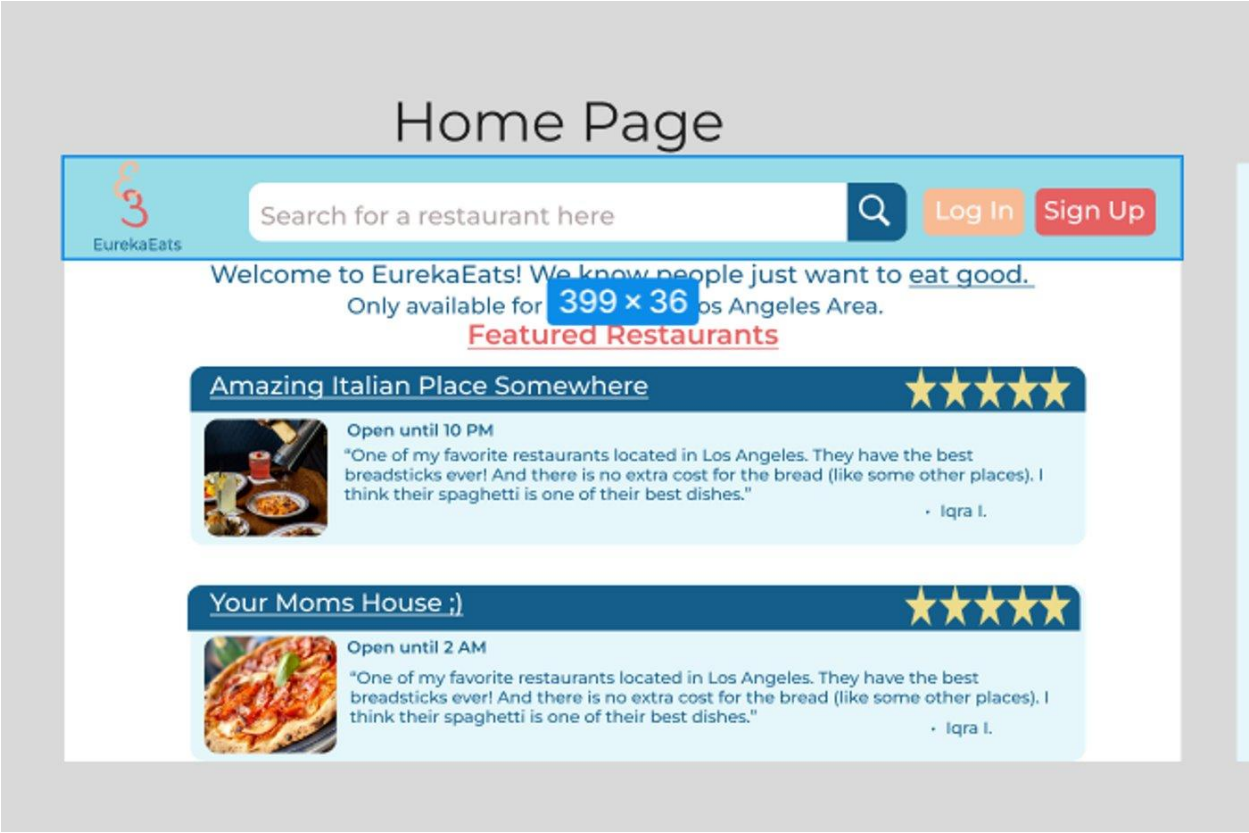
- 2: object
- 3: boolean

Application Message Data:

- Format: JSON text
- App Operations: (usually for API requests to the EurekaEats backend)
    - `{ "action": number, "args": <object | array>}`
- App Data: (usually a response to an EurekaEats API call)
    - `{ "payload": string, "data": <object | array>}`

4. Requirements Specification

The main core of the project is that it will be a website. With this fact, we will have many functions to collect user information from mouse clicks, mouse scroll, and keyboard entries. To know the requirements of our website, we will split it between main pages for our website.



Homepage:

- Contains our EurekaEats logo
- Search bar to enter a restaurant, a certain cuisine, or a type of food.
    - Search items should have a filter to guess that you are looking for.
    - Search bar should have the functionality of sending the word by pressing the ‘enter’ key or by clicking the search button
- Login button for users to log in. Mouse click for input
- Sign up button for users to sign up for an account. Mouse click for input
- Short welcome phrase for our users with our catch phrase
- Featured Restaurant based on popularity.

- Contains a review about the restaurant
- Contain the opening hours till closing
- Users should be able to click on the restaurant name to to sent to the restaurant's page
- Contains the star value of the restaurant

This section contains all of the necessary software requirements with enough detail to allow designers to accurately design the software to satisfy those requirements, and to allow testers of the software to verify that all requirements have been satisfied. The requirements should include a description of every input to the system, every output, and all functions performed by the system in response to an input or output.

The biggest thing to remember is that this section is for the software developers (technical people) while the previous sections were for the customers / non-technical people.

Also remember that this is not HOW things will be implemented, but WHAT will be implemented.

Requirements should be written according to the following:

- Specific requirements should be correct, unambiguous, complete, consistent, ranked for importance and / or stability, verifiable, modifiable, and traceable.
- Specific requirements should be cross-referenced to earlier documents that are relevant.
- All requirements should be uniquely identifiable using a consistent numbering system, i.e. 1.1, 1.2, 1.1.2, and so on.
- Requirements should be organized in a logical manner to provide the most readability.

Use the following format for each requirement:

- The system shall... (this means this requirement is mandatory).
- The system should... (this means a desired feature, but may be delayed until later).
- This system may... (A optional, nice-to-have feature that might not be implemented).

Remember to number each requirement for traceability. Use a system such as 1.1, 1.1.1, 1.1.2.1, and so on. Each requirement need to be testable. Avoid statements that are general and vague such as "The system shall be easy to use." or "The system shall be developed using good software engineering practices."

Do not include examples. Remember that this is a specification and the designer should be able to read this and build the system without having to bother the customer again. Every minute detail must be documented here.

EVERYTHING in section 4 must be written following the above guidelines.

4.1 Functional Requirements

Functional requirements define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as "shall" statements starting with "The system shall…"

These include:

- Validity checks on the inputs
- Exact sequence of operations
- Responses to abnormal situation, including
    - Overflow
    - Communication facilities
    - Error handling and recovery
- Effect of parameters

- Relationship of outputs to inputs, including
  - Input/Output sequences
  - Formulas for input to output conversion

It may be appropriate to partition the functional requirements into sub-functions or sub-processes. This does not imply that the software design will also be partitioned that way.

This section should be as detailed as possible, again, listing WHAT your software is going to do, not HOW you are going to accomplish it.

4.2 External Interface Requirements

This contains a detailed description of all inputs into and outputs from the software system. It complements the interface descriptions in section 3 but does not repeat information there. Remember section 3 presents information oriented to the customer/user while section 4 is oriented to the developer.

It contains both content and format as follows:

- Name of item
- Description of purpose
- Source of input or destination of output
- Valid range, accuracy and/or tolerance
- Units of measure
- Timing
- Relationships to other inputs/outputs
- Screen formats/organization
- Window formats/organization
- Data formats
- Command formats
- End messages

4.3 Logical Database Requirements

This section specifies the logical requirements for any information that is to be placed into a database.

This may include:

- Types of information used by various functions
- Frequency of use
- Accessing capabilities
- Data entities and their relationships
- Integrity constraints
- Data retention requirements

If the customer provided you with data models, those can be presented here. ER diagrams (or static class diagrams) can be useful here to show complex data relationships. Remember a diagram is worth a thousand words of confusing text.

4.4 Design Constraints

Specify design constraints that can be imposed by other standards, hardware limitations, etc. This should be a more technical description of the overview given in section 2.5.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The number of simultaneous users will be up to 1000 which will require

Memory: 16 GB CPU: 16 core Disk: 100 GB

5.2 Safety Requirements

- Possible loss, damage, or harm: EurekaEats will include safeguards to prevent data/security leaks using tools already available via MongoDB, this is to ensure personal data remains confidential. Mechanisms to prevent data loss include regular scheduled data backups in the case of a system failure.

5.3 Security Requirements

- User identity authentication requirements: Strong password policy (must be over 10 characters with 1 special character), captcha not a robot check.
- EurekaEats will follow security guidelines for products released in California.

5.4 Software Quality Attributes

- Straightforward UI for usability, prevents the user from re-learning how to navigate through the product. Emphasis on ease of use over ease of learning. This will be done by having basic functionalities/buttons the user has seen before in other applications. In addition, the layout of the screens will be pleasing to the eye so that users can find what they are looking for.
- Well-maintained code: EurekaEats code will be documented thoroughly so that any updates to the software will not result in system failure, in addition to having easily readable code (variable, function, and class names are clearly named). Having well-maintained code leads to a more reliable product.

5.5 Business Rules

- Main engineers (group members) are considered admins of the product. They will have access to the database (CRUD functions) and code for the product.
- Communication between users of the product and the admins can be done with a report button through the product.

6. Legal and Ethical Considerations

We will reference code sources related to this project not developed by us.

Appendix A: Glossary

See section 1.4 for all abbreviations or special terms.

Appendix B: Analysis Models

TBD

Appendix C: To Be Determined List

TBD