WASHINGTON STATE UNIVERSITY
VANCOUVER

Software Requirements Specification

for

Journaling Software

Version 1.0

Prepared by

Group Name: Dream Team

Malcolm Anderson	**011707961**	**malcolm.i.anderson@wsu.edu**
Tony Maniscalco	**011467861**	**antonio.maniscalco@wsu.edu**
Chase Jamieson	**11732855**	**chase.jamieson@wsu.edu**

Date: 6 November 2020

1 REVISIONS **1**

2 INTRODUCTION **2**

2.1 Document Purpose 2
2.2 Product Scope 2
2.3 Intended Audience and Document Overview 2
2.4 Definitions, Acronyms and Abbreviations 3
2.5 Document Conventions 4
2.6 References and Acknowledgments 4

3 OVERALL DESCRIPTION **5**

3.1 Product Perspective 5
3.2 Product Functionality 5
3.3 Users and Characteristics 5
3.4 Operating Environment 6
3.5 Design and Implementation Constraints 6
3.6 User Documentation 6
3.7 Assumptions and Dependencies 6

4 SPECIFIC REQUIREMENTS **7**

4.1 External Interface Requirements 7
4.2 Functional Requirements 8
4.3 Behavior Requirements 9

5 OTHER NON-FUNCTIONAL REQUIREMENTS **10**

5.1 Performance Requirements 10
5.2 Safety and Security Requirements 10
5.3 Software Quality Attributes 10

6 APPENDIX A – DATA DICTIONARY **12**

1.1 Meteor Code Variables 12
1.2 Necessary Functions and Variables for Software 12

7 APPENDIX B – GROUP LOG **14**

1 Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.0	Malcolm Anderson Chase Jamieson Tony Maniscalco	Initial draft of the SRS document, describing the product.	11/06/20

2 Introduction

2.1 Document Purpose

The purpose of this document is to describe and define specifications for the development and deployment of a journaling software user application. Such specifications will include:

- The subsystems created to handle different aspects of the application's operation,
- The manners in which these subsystems interact,
- The manners in which the users interface with the software,
- The interface, hardware, and software requirements for user interaction, and
- Non-functional requirements and performance thresholds required for the software.

2.2 Product Scope

The software will allow users to write private journal entries and store them, sorted by date. Users may choose to make a journal entry public, such that anyone with the entry's URL can read it. Multiple users will be able to log into the software website simultaneously. While logged in, they will be able to manage or compose journal entries. Using the software, users can keep track of important events in their lives and look back through them later on and share especially important moments with friends and family.

For the first testing iteration of the software, only one or two users may be permitted to log on simultaneously (since it will be hosted on a computer locally, not on a server). This first iteration may not be effective at handling a large number of data requests from users simultaneously as a result. Addressing this issue will be a priority in later versions.

2.3 Intended Audience and Document Overview

This document is intended to be read by the client and developers who will be building the software. Language will be restricted to that which is understandable to the "average person". Technical terms that may not be well-known will be listed and defined in the following subsection, "1.4 Definitions, Acronyms, and Abbreviations", as well as in the section "Appendix A - Data Dictionary" at the end of this document.

It is recommended to read this document in order, from start to finish, in order to best understand what is required for the software. The current section provides the reader with a basic understanding of the journaling software's goal, as well as important background information regarding its development process.

The second section of this document, "Overall Description", will define and discuss the journaling software in more detail. Such details include:

- Specific functions the software will be able to perform,
- The software's target audience,
- The environment in which the software will be expected to be used,
- The level of documentation that will be required for the target audience, and
- Important assumptions that developers will need to operate on.

The third and fourth sections of this document, "Specific Requirements" and "Other Non-Functional Requirements", will break the discussed specifications for the journaling software down into more specific requirements, including:

- The hardware and software required to run and use the software,
- Any specific protocols or procedures that will need to be adhered to when developing and running the software, and
- Any performance thresholds that need to be met in order for the software to be deployed.

2.4 Definitions, Acronyms and Abbreviations

This is a list of some of the more common technical terms that may be found in this document. More terms may be found in the section "Appendix A - Data Dictionary" of this document.

- **Backend** - The processes that run on internal servers, completely separated from user input. Secure database information, such as journal entries and user authentication information, may also be stored here.
- **Collection** - A related group of data in a database. For example, the group of all journal entries stored in the database would be considered one collection, as would the group of all user login information [2].
- **Facebook** - A social media platform. An owner of a Facebook account may use it to sign into other services.
- **Frontend** - The processes that run on the user's computer. Processing here is limited to that which the user is authenticated to see by the backend, since in theory the user may be able to modify the code run by the frontend.
- **GitHub** - A computer code hosting platform. An owner of a GitHub account may use it to sign into other services.
- **Google** - A popular online software suite. An owner of a Google Account may use it to sign into other services.
- **Identification (ID) Number** - A number/string meant to uniquely identify a piece of data.
- **JavaScript** - A programming language used for creating interactive web applications.
- **Meetup** - A social media and group-forming platform. An owner of a Meetup account may use it to sign into other services.
- **Meteor** - A JavaScript framework for building web applications with a semi-unified frontend and backend code base [3].
- **Model-View-Controller (MVC)** - A design pattern for linking user interfaces with internal program data. This allows the user interface and program data to respond to valid user input while preventing the user from modifying program data in unintended or malicious ways.
- **OAuth** - A standardized procedure for protecting a user's passwords and sensitive information when logging into a web service or making requests.
- **React** - A JavaScript framework for creating responsive user interfaces.
- **Request** - The process by which the frontend asks the backend to take a specific action or provide a specific piece of information.
- **Response** - The process by which the backend responds to a request from the frontend.
- **Twitter** - A social media platform. An owner of a Twitter account may use it to sign into other services.
- **Uniform Resource Locator (URL)** - An address that points to a specific location/page in the software. For example, "https://www.journal.app/my_journals" may point to a list of the currently logged in user's journals, and "https://www.journal.app/entry/0123/edit" may point to a page for editing a journal entry with the ID number 0123.

2.5 Document Conventions

This document should follow the IEEE 830-1998 standard for formatting software requirements specifications documents (https://standards.ieee.org/standard/830-1998.html). Most of the text in this document will use the sans-serif Arial font, but when computer code is discussed, it will be written in the monospace Courier New font. Key words may be bolded in order to emphasize their importance. Lists may be used to make collections of content easier to read. If the list items are numbered, then their order is important; if they are bulleted, the items have no particular order.

2.6 References and Acknowledgments

[1] "Accounts," *Meteor API Docs.* [Online]. Available: https://docs.meteor.com/api/accounts.html. [Accessed: 07-Nov-2020].

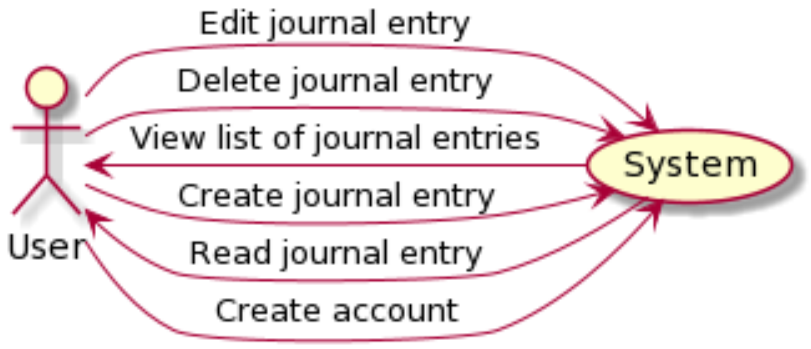
[2] "Collections and Schemas," *Meteor Guide.* [Online]. Available: https://guide.meteor.com/collections.html. [Accessed: 07-Nov-2020].

[3] *Meteor API Docs.* [Online]. Available: https://docs.meteor.com/. [Accessed: 07-Nov-2020].

3 Overall Description

3.1 Product Perspective

The journaling software is a self-contained product (i.e. it is not based on an existing product). There are two components to the software: a frontend, which will provide the user with an interface with which to retrieve and enter data, and a backend, which will store and release data according to the user's permissions.



These operations will be completed via a Model-View-Controller (MVC) user interface, which will permit the user to only perform operations and view information if they have permission to do so.

3.2 Product Functionality

The user will be able to interact with the software via a web site. The following functions will be permitted:

- The user may create or log into a personal account.
- When logged into their account, the user will be able to view a list of all journal entries they have access to, sorted by date.
- The user will be able to write a new journal entry, which they will be automatically granted access to.
- If desired, the user will be able to grant other users access to a specific journal entry, either for editing or for reading only.

3.3 Users and Characteristics

The software is not intended for specialized use; rather, the goal is for it to be easily usable by members of the general public, with a wide variety of technical expertise and educational level. As a result, the interface must be accessible to people without much technical experience. Because this will be a public product, there will be no need to differentiate users based on any security or privilege level, including those with limited experience using computer interfaces.

3.4 Operating Environment

The user will access the software via an internet browser, such as Safari, Microsoft Edge, Google Chrome, or Firefox. The minimum platform requirements for the user's system are thus the requirements for their internet browser of choice. The aforementioned internet browsers are the most commonly used ones in the world, so they should be expected to be able to interact with the website with little to no issues.

The software itself will run on a headless server. During development, however, it may be run on a local device instead for quicker testing. The server will likely use the Meteor framework for frontend and backend management.

3.5 Design and Implementation Constraints

The following are potential design and implementation constraints:

- The server will need a database(s) to keep track of users' journal entries and accounts.
- Users will expect that information they provide to the software is safe and secure from other users. As a result, strong security for user accounts will be important.
- The Meteor framework may be used for developing the software. It uses the JavaScript programming language, as well as one of several web user interface frameworks (such as React) [3]. As a result, developer knowledge of the JavaScript programming language, as well as these frameworks, will be necessary.

3.6 User Documentation

As stated in section 2.3, the Software has a wide target audience, and extensive prior technical experience is not necessarily expected. User documentation that covers the following topics would be important:

- How to create an account
- How to sign into an existing account
- How to read a journal entry
- How to draft a new journal entry
- How to share a journal entry with someone else

3.7 Assumptions and Dependencies

The following assumptions and dependencies will be important to consider while developing the software:

- It is likely that multiple users will interact with the Software simultaneously (e.g. requesting entries, drafting entries, etc.). As a result, the Backend will need to be able to handle requests from multiple users at once.
- The developers may be using different platforms, such as macOS, Windows, or Linux, to develop the software. This means that frameworks and libraries will need to be cross-platform.
- Open-source frameworks and libraries used by the software may come with different licenses. When deciding which ones to use, considering the license terms and their implications will be important to ensure that the software as a whole remains in compliance with them.

4 Specific Requirements

4.1 External Interface Requirements

4.1.1 User Interfaces

When logged in, the user's experience with the software will take place across four user interfaces:

- The **Entry List View** will provide the user with a list of journal entries that they own, sorted by date. The user may scroll through this list and select a journal entry to view it in the Entry Reading View. There may also be a button to create a new journal entry that the user owns, and open it in the Entry Editing View.
- The **Entry Reading View** will display the full text of a journal entry for the user to read. If the user is the owner of the current journal entry, there may also be a button here to navigate to the Entry Editing View for this entry.
- The **Entry Editing View** will allow the user to edit the contents of a specific journal entry. Support for rich-text formatting (such as bolded, italicized, underlined, or colored text) may be added if time and resources permit.

There will also be user interfaces for when a user is not logged in:

- The **Landing Page View** should display when a non-logged-in user attempts to visit the software's main web page. Basic information about the software, as well as a link to the Login View, should be displayed here.
- The **Login View** should display when a non-logged-in user attempts to log in or view a protected page (i.e. a private journal entry). This screen should allow the user to input a username and password to log in. If they are properly authenticated, they may be redirected to either the Entry List View or the Entry Reading View for the entry they were attempting to view (assuming they have privileges to access that entry).

4.1.2 Hardware Interfaces

The journaling software will interface with the user via standard input and output methods. The user may use a computer mouse or trackpad to navigate between user interfaces in the software, and a keyboard to enter text to the Entry Editing View for journal entries. The user interfaces will be displayed to the user via a computer monitor.

At this time, there are no plans for further or more complex hardware interfacing. Entries will only consist of text, so the computer will not have a need for audio speakers.

4.1.3 Software Interfaces

The Software system requires the use of Google Chrome, Microsoft Edge, Firefox, or Safari as a web browser. It is recommended that the current system meets the requirements of the suggested browsers.

The application server will be implemented with the full-stack JavaScript platform Meteor. Meteor stores persistent data as Collections, which can be cached on compatible databases such as MongoDB [2].

4.1.4 Communications Interfaces

At this time, the Meteor framework is being considered to base the software prototype upon. This framework provides built-in support for user accounts. Since the initial version of the software will be a prototype, where security is not as important, these should suffice.

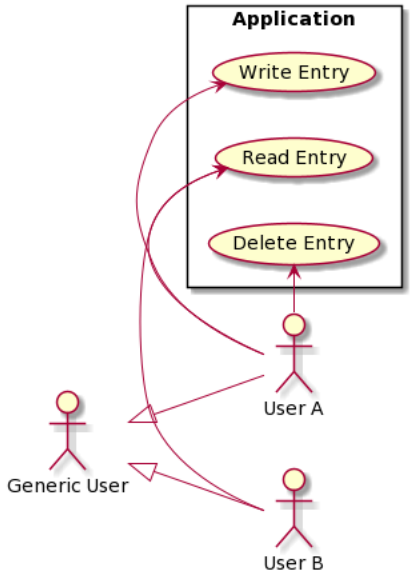
The initial prototype of the software will run on a local machine, not on a server, so encryption will not be necessary.

4.2 Functional Requirements

1. **Create and log into a personal account** - Using the core packages and OAuth support Meteor actively maintains, users may be able to have Facebook or Google login within the website. Additional packages for GitHub, Twitter, and Meetup can also be implemented. The collection is stored under the `Meteor.users()` method [2].
2. **Access data when logged into their account** - When a user is logged into our app with the one of the methods above, Meteor has on the Client the `Meteor.userId()` global method which returns the ID of the currently logged in user [1]. The server has a `Meteor.userID()` global method which returns the user ID depending on which method it is called from [1]. Using these User IDs as keys for each entry gives a user logged into said accounts access to the data under their ID.
3. **User will be able to write a journal entry** - Users are by default allowed their own profile field with a call to `Accounts.createUser()` [1]. Information on their account can be modified with `Meteor.users.update()`. With `update()`, users can append new journal entries to their existing list.
4. **The user will be able to grant other users access to a specific journal entry, either for editing or reading only** - New Journal entries are automatically given permission to the creator, using a `UserCanEdit(entryId, userId)` function, involving the `editableBy(userId)` Meteor method, which will throw an error unless the author has added another user ID to their list of permissions to other users [1]. Users will have a uniquely named top level field which will be a list of user IDs that have an editing or a reading permission.

4.3 Behavior Requirements

4.3.1 Use Case View



The above image is a use case diagram demonstrating how two users may interact with the software. User A may write an entry, giving themselves ownership of the entry in the process. As the owner of the entry, they may also read and delete it. User B, who is not an owner of the journal entry, may not delete it, nor write entries corresponding to User A. However, if User A grants User B permission to read a journal entry, they may be allowed to do so.

5 Other Non-functional Requirements

5.1 Performance Requirements

The following are performance requirements for the software:

- Adding new journal entries to profiles will process in under 5 seconds
- Users can grant other users access to a given journal entry in under 5 seconds.
- Retrieving past journal entries for editing, making public, etc. will process in under 5 seconds.

5.2 Safety and Security Requirements

The following are important safety and security requirements:

- Users may not access data that they are unauthorized to view. For example, if a user does not have reading permissions for a journal entry, they should not be allowed to view the contents of the journal entry at its URL.
- User data must be securely stored within the software database. Sensitive information, such as passwords, must be encrypted such that they would not be recoverable if the system were hacked (i.e. they must not be stored in plaintext).

In later iterations of the software, the following additional requirements may also be added:

- Users may choose to use two-factor authentication (2FA) when logging into their account.
- Upon creating a new account, users will need to confirm their email address before gaining access to the software.

5.3 Software Quality Attributes

5.3.1 Maintainability

The code base should be well organized and commented for the developers to edit and develop further without wasting time trying to understand the code. The system's code base and commit history will be available on GitHub alongside necessary documentation to explain planned and added features.

5.3.2 Portability

The system will be compatible with Windows, Linux, and macOS platforms. The client application should be developed on Windows and macOS to cover the largest set of users. The browser interface will support newer versions of popular browsers such as Chrome, Edge, Firefox, and Safari. The full-stack framework being used, Meteor, is supported by most platforms.

5.3.3 Reliability

The software system should be subject to load- and feature-testing prior to release and development of additional features. Testing would be two parts: trying to overload the various

methods and features for a bug, and verifying the correct data is being inputted and outputted across the various functions.

5.3.4 Testability

The system will provide functionalities to run unit tests. The construction and running of each unit test should be fast, since fast test building allows quick feedback. The tests must be built into our own codebase to run without extra tools on our platform. This is to achieve fast unit tests with quick adjustment and feedback to the developer.

5.3.5 Usability

The system will provide functionalities to run unit tests. The construction and running of each unit test should be fast, since fast test building allows quick feedback. The tests must be built into our own codebase to run without extra tools on our platform. This is to achieve fast unit tests with quick adjustment and feedback to the developer.

6 Appendix A – Data Dictionary

1.1 Meteor Code Variables

Name	Type	Description
`Meteor.userId`	Variable	Provides the software with the ID for the currently logged in user, if they are logged in [1].
`Meteor.users`	Collection	Contains information for all of the users registered to use the software [2].
`Accounts.createUser()`	Function	Creates a new user object and adds it to the `Meteor.users` collection [1].

1.2 Necessary Functions and Variables for Software

Note that the underlying structure for the pseudocode given for these functions/processes may change as development progresses to better fit JavaScript/Meteor/React design patterns better.

Name	Pseudocode Representation	Description
Log In	`LogIn(username, password)`	Log in the user with a given `username` and `password`.
Is Logged In	`IsLoggedIn`	Return a Boolean (true/false value) determining whether or not a user is logged in.
Create New Entry	`CreateEntry()`	If a user is logged in (see `IsLoggedIn`), create a new journal entry and assign it to the currently logged in user.
Get Entry Contents	`GetEntry(entryId, userId)`	If the currently logged in user has permission to view the entry with ID `entryId` (see `UserCanView(entryId, UserId)`), return the text content of the entry with ID `entryId`.
Set Entry Contents	`SetContents(entryId, userId, contents)`	If the currently logged in user has permission to edit entry with given ID (see `UserCanEdit(entryId, userId)`), set the text of

		entry with ID `entryId` to given `contents`.
Delete Entry	`Delete(entryId, userId)`	If the currently logged in user has permission to edit the given entry (see `UserCanEdit(entryId, userId)`), delete the entry with ID `entryId`.
Get List of Entries for User	`GetEntries()`	If a user is logged in (see `IsLoggedIn`), return a list of the entries the currently logged in user has permission to view or edit.
Get List of View-Only Entries for User	`GetViewableEntries()`	Sorts output from `GetEntries()` and returns only the entries the currently logged in user has permission to view but not edit. Use the `UserCanView(entryId, userId)` and `UserCanEdit(entryId, userId)` functions to perform the sorting.
Get User Can View Entry	`UserCanView(entryId, userId)`	Returns a Boolean (true/false value) determining whether or not the user with ID `userId` can view an entry with ID `entryId`.
Get User Can Edit Entry	`UserCanEdit(entryId, userId)`	Returns a Boolean (true/false value) determining whether or not the user with ID `userId` can edit an entry with ID `entryId`.

7 Appendix B – Group Log

Note: Most of the work on the SRS document was done asynchronously via Google Docs, in order to more easily fit team members' different schedules. The work from Google Docs was transferred into a Word document by the project leader, which is why the Git commits in the project repository all appear to come from a single source. The Google Docs document is available here: https://docs.google.com/document/d/1SwgCps6V2aUJR65t5KcWBAx5-9ZcBakg8vY2dFQZXgY/edit?usp=sharing

Date	Description

28 September 2020	• Setting up group
	• Discussing project ideas
	  ○ Trivia game
	  ○ Survey software
	  ○ Web scraper
	  ○ "Tinder for DnD players"
	  ○ Project idea finder

2 October 2020	• Exchanging information • Deciding how to structure project group • Deciding when to hold biweekly meeting times ○ Decided on Tues/Thurs at 3 PM, when there are matters to discuss
5 October 2020	• Finalizing team hierarchy • Discussing project ideas ○ Journaling software • Finalizing Team Formation Agreement
8 October 2020	• Setting up Google Docs document for collaborating on SRS
11 October 2020	• SRS template sections and hints transferred to Google Docs document
13 October 2020	• Brief meeting • Work on section 1 of SRS started
18 October 2020	• First work on section 2 of SRS completed
24 October 2020	• Quick check-in with other members
27 October 2020	• Quick check-in with other members

30 October 2020	• Quick check-in with other members
	• Functional requirements added
	• Non-functional requirements added
	  ○ Performance
	  ○ Safety and security

4 November 2020	• Started transferring writing from Google Docs document to SRS Word document template • Requested student ID numbers for document
5 November 2020	• Jamieson's student ID number added to document
6 November 2020	• GitHub repository created with standard Meteor project and SRS Word document