Software Requirements Specification

For

TapWater

Version 1.0 approved
October 12, 2014

Prepared by: Jonathan Hooper, Cody Rogers, Kon-Kon Chao, David Fontana

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a detailed description of the functionalities of the Tap Water system. This document will explain the purpose and features of the system, as well as a look into the system's User Interface (UI). The document will cover hardware, software, and various other technical dependencies and use cases. This document is intended for both the stakeholders and the developers of the system.

1.2 Document Conventions

This document will use some terminology that the audience may be unfamiliar with. Please see Appendix A (Glossary) for a list of these terms and definitions.

1.3 Intended Audience and Reading Suggestions

This document is intended for all individuals participating in the development or management of the TapWater system. Individuals interested in a summary of the product should continue reading the rest of Section 1 (Introduction) and Section 2 (Overall Description). These sections give an overview of each piece of the system.

Individuals who wish to learn more about the technical aspects of TapWater should read Sections 1 and 2 as well as continue on to Section 3 (External Interface Requirements). Section 3 gives you a detailed look into the technical details, including information on the user interface as well as the hardware and software platforms on which the system is run on. After Section 3, it is also recommended to move on to Section 4 (System Features), which details each feature the system offers.

Individuals who are interested in the non-technical aspects of the project should read Section 5, which covers performance, safety and security requirements as well as software quality attributes and business rules.

1.4 Product Scope

The TapWater system is composed of two main components: a client facing application that will run on iOS and Android devices as well as a centralized server to synchronize data between multiple devices. The system is designed to help users track, save and view a history of their water intake. The system will also dynamically generate reminders on the user's device based on how much water they have recorded for the day. Registration is not required but doing so will synchronize your history between multiple devices.

1.5 References

"Android Developers." *Android Developers*. N.p., n.d. Web.

"Heroku Dev Center." *Heroku Dev Center*. N.p., n.d. Web.

"IOS Developer Library." *IOS Developer Library*. N.p., n.d. Web.

"Web Development That Doesn't Hurt." *Ruby on Rails*. N.p., n.d. Web.

2. Overall Description

2.1 Product Perspective

The TapWater development project works towards creating a new, self-contained software product, namely in the form of a joint-OS (Android and iOS) mobile app. Thus, the primary focus is development of these two co-existing applications. There will also exist a server-side component of development to facilitate synchronization across multiple platforms and devices. In this way, the project requires and will make use of server- and client-side technologies.

2.2 Product Functions

The following is a (non-exhaustive) list of core features, along with a brief description without diving into great detail of the technology supporting them. All features listed are categorized as "core" or "main" features, meaning they are either essential to the applications operation or promised functionality which created the demand for such a product.

Features:

1. Unique User Registration and Setup
    - Encompasses a “first-time only” account setup to register the user as a unique Client.
    - Subsequent logins (including those on additional devices) require only a typical username/password submission.
    - Setup provides the User the opportunity to customize a few additional settings, including toggling notifications for the device (critical).
2. Adding Drinks
    - From a simple menu, the User is able to add drinks from a list of pre-configured amounts (typical amounts, such as 4, 8, and 16fl oz.)
3. Tracking Drinks
    - A simple view of “drink history” throughout the day, including the times they were logged into the application.
    - It is also possible to view previous days to compare progress.
4. Drink Notifications
    - From a tracking algorithm, we can push notifications to the User’s device(s) to remind them to drink.

2.3 User Classes and Characteristics

As a fairly simple, straightforward application, the User Classes are somewhat trivial to lay out in detail. The TapWater application is intended to offer ease-of-use in an intuitive and unobtrusive interface. Part of this intention is fulfilled by refraining from adding “embellishment” features and menus that are rather unnecessary to fulfill the intended use. Moreover, as a cross-platform application, these interfaces should mirror each other as much as possible. For the purposes of User Classes, we will define the following:

Simple User – A User who utilizes TapWater on only a singular device **Complex User** – A User who utilizes TapWater across multiple devices and, possibly, multiple platforms.

Simple User

Critical Functions:

- Log water intake
- Track progress towards daily goals

Requirements:

- Methods for logging water intake, including fixed and customized amounts
- Simplistic history-view of intake for each day
- A visible daily progress bar for added clarity

Complex User

Critical Functions:

- All functions identified in the first User Class
- Synchronization functions to update progress and drink-history on all client devices

Requirements:

- Synchronization method for pushing new data to server
- Synchronization method for pulling updated data from server

The TapWater development team recognizes that both User Classes are critical to the success of the project. Therefore, all identified requirements are classified as “critical”, in that they must function, and function reliably, to minimize frustration and promote user-base growth.

2.4 Operating Environment

As the core functionality of the TapWater development project is focused within the mobile applications, the intended operating environments are the Android and iOS mobile operating systems. Due to the intentional simplicity of the interface and functionality, the system is highly unlikely to overload the hardware of any popular mobile device. Server-side functionality will be implemented through Heroku, eliminating the need for development and maintenance of complex server technologies.

2.5 Design and Implementation Constraints

From a design standpoint, it will be important to remember the functional limitations of mobile devices. By keeping the interface minimalistic, screen size and resolution are less of a concern. However, menu hierarchy will need to be carefully managed to avoid a frustrating user-experience in navigating the application. The action of "logging" a water intake should not be a hardware intensive action, either. The primary concern, then, lies with synchronization of data to the server, which can (potentially) be somewhat more processor-intensive as well as more overall complex.

2.6 User Documentation

Again, we are at an advantage here due to the simplistic nature of the application's function and design. Because we fully intend there to be very little learning curve to use this application (which is, for the most part, a critical requirement of all mobile apps) there is also little need for complex documentation. A very brief tutorial will be provided in the app-store descriptions to clarify any ambiguous features or menu options. An in-app help menu will also identify the function of all screen elements on each page.

2.7 Assumptions and Dependencies

Hardware Dependencies

Again, all server functionality will be implemented through Heroku, reducing complexity of the implementation. However, this also increases external dependency for the project, because the application will only be able to sync when the server functions properly. Client-side, the application will be dependent upon the mobile device's mobile and Wi-Fi antennas to accomplish synchronization tasks. The mobile applications will also be dependent on other integral technologies, such as the touch-screen hardware.

Software Dependencies

With the Android operating system evolving so rapidly over the past several years, we must be sure to develop only using functions available several iterations previous to the current OS version. Many mobile phone users do not update their devices in a timely manner. With Android, we will also be implementing functions from the native Android push notification library. With the iOS operating system, this is less of a problem, but we must still be sure to develop several versions back to support older devices, which cannot update to the latest iOS platform.

3. External Interface Requirements

3.1 User Interfaces

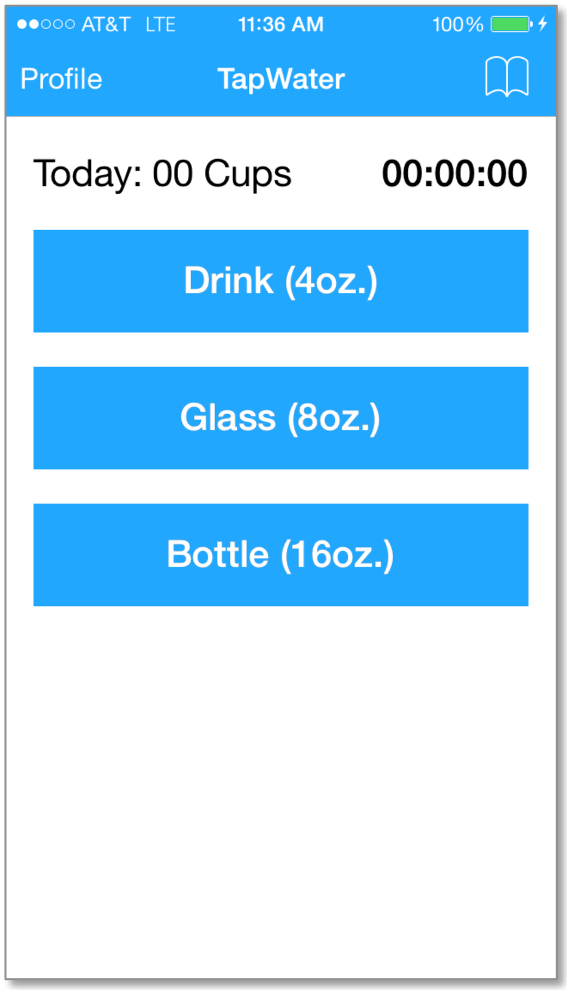
3.1.1 General Guidelines

The layouts for both the iOS and Android systems will adhere to the current design and layout guidelines published by Apple and Google. This will streamline the user experience and keep it fluid within each respective mobile operating system.

3.1.2 Welcome Screen

Welcome screen differs based on user registration. If this is the first time the system is run, the user will be presented with a screen to create an account or skip account creation. When account information is filled out, it will be sent to the server and all drink history will be synchronized with the account across every device it’s logged into. A quick tutorial option is also present for the user to select.

3.1.3 Home Screen

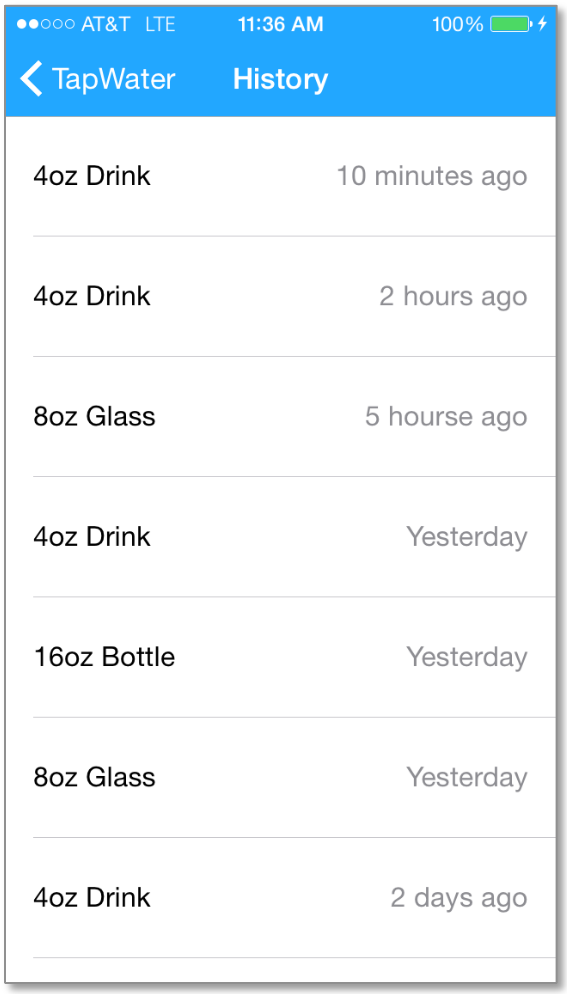


The main screen displays three predefined and common amounts of water a user can select. Tapping on "Profile" will bring the user to the profile screen with history, summary of drinks, amount of water needed to meet goal and next notification with drink amount recommendations.

3.1.4 Profile Screen

The profile screen shows the user name and summary of current drinks recorded. A "History" button will take the user to a more detailed view showing dates and amounts of each drink recorded. The user can also set customized notifications by tapping on "Notifications". If user didn't create an account, a button will be present for the user to register and push their local history to the server. Tapping the "TapWater" button will bring the user back to home screen.

3.1.5 Drink History Screen



4oz Drink	10 minutes ago
4oz Drink	2 hours ago
8oz Glass	5 hourse ago
4oz Drink	Yesterday
16oz Bottle	Yesterday
8oz Glass	Yesterday
4oz Drink	2 days ago

The history screen lists the user's drink history in chronological order. Details include time elapsed since each drink based on user's current time and how much the user drank. The user will also be able to use this screen to delete, duplicate or accidental drinks. Tapping on "TapWater" will bring you back to the home screen.

3.1.6 Drink Notification Schedule Screen

This screen allows users to schedule notifications to remind them to drink water. There are fields where the user can set a goal for the number and amount of drinks they want to reach for the day. The user will then select a beginning/end time from within the 24-hour day. An enable/disable notifications toggle will control notifications for the device.

3.2 Hardware Interfaces

TapWater is being developed and deployed as a mobile system for iOS and Android platforms. This includes mobile phones and tablets running supported versions of iOS (7+) and Android (4.0+). User accounts and drink history will be synchronized across each device hosted on a server through Heroku. The notification service is a locally run service and not handled by the server.

TapWater is only being developed for iOS 7+ and Android 4.0+ and updated periodically to add features, squash bugs and adhere to each company's design guidelines. Network communication and hardware interface methods are provided by the iOS and Android platforms.

3.3 Software Interfaces

The TapWater system developed for Android operating systems will be using the Java development kit and the Android software development kit tools. TapWater for iOS will be developed in swift with the iOS software development kit.

TapWater has outgoing data consisting of user account creation, drinks and authentication requests to the server. Incoming data consists of drink history retrieval and authentication verification. The server is built in Ruby with Ruby on Rails web framework and features an system programming interface with the following capabilities: Creating and authenticating users, saving drinks associated with an authenticated user, and downloading an authenticated user's drink history. TapWater's drink reminder notifications are handled locally and communication with the server is not needed for that function.

3.4 Communications Interfaces

The TapWater system will be connected to a server hosted on Heroku built with in Ruby with the Ruby on Rails web framework. The server's purpose is to store and retrieve information to and from the database and pushes them to each of the user's devices. The Ruby on Rails framework contains an system programming interface for the three interactions the system will be making with the server and databases.

4. System Features

4.1 Log Drinks

Users will record the drinks they consume throughout the day.

4.1.1 Description and Priority

Users will record their consumption of water throughout the day manually by choosing from three different categories: a "drink", a "glass", or a "bottle". A drink constitutes 4 oz of water, a glass constitutes 8 oz of water, and a bottle constitutes 16 oz of water. This feature is critical to the overall usability of the system, so it is of high priority.

4.1.2 Stimulus/Response Sequences

1. From the main screen, the three TapWater water consumption sizes are presented and selectable.
2. The user will select one of the options when they have drunk a matching amount of water.
3. The total amount of water consumed in that day will increment on the screen
4. A new drink with be added to the device's database

4.1.3 Functional Requirements

- REQ 1.1: The system should provide 3 different options for recording drinks.
- REQ 1.2: The system should create a drink object with a unique UUID for the created drink, the category of the drink selected, and the current date as the drink date.
- REQ 1.3: The system should save the created drink into the device's database and be ready to accept another drink before the selection's button animation completes.

4.2 View Drink History

Users will view the drinks they have recorded.

4.2.1 Description and Priority

Users will have a list of all their previously recorded drinks. This list will contain the size of the drink logged as well as the time it was logged. This feature is a high priority.

4.2.2 Stimulus/Response Sequences

1. The user will select an icon on the main screen, which will lead them to their drink history.
2. Users will see a list of all the previous drinks they have recorded as well as information regarding the size of the drink and time it was recorded.

4.2.3 Functional Requirements

- REQ 2.1: The system should provide a button, which will open a screen to view the user's logged drinks.
- REQ 2.2: The system should list each drink the user has logged, with each entry displaying the category of the drink and the date the drink was logged.

4.3 Schedule Drink Notifications

Users will be able to schedule the frequency in which they are reminded by the system to drink water.

4.3.1 Description and Priority

The system will allow for both manual, unassisted drink entry as well as remind the users that it is time for them to consume another drink of a certain size in order to meet their self-defined goal. This goal is user defined and the user will specify how often (or how many times per 24 hour period) they would like to be reminded to drink. The priority of this feature is medium.

4.3.2 Stimulus/Response Sequences

1. The user will tap a button to take them to a notification settings screen.
2. The user will enter a goal for the amount of water they wish to drink in a day.
3. The user will use a date picker to pick a start and end time to specify the period in which they would like to be reminded to drink.
4. The user will toggle notifications on/off.
5. The application will notify the user to drink an 8 ounce cup of water at even intervals throughout the day until they've met their goal.

4.3.3 Functional Requirements

- REQ 3.1: The system should provide a "settings" icon, which provides access to the notification scheduler.
- REQ 3.2: The notification scheduler should provide fields to edit the number of ounces the user wishes to drink daily, as well as how often the system will notify them.
- REQ 3.3: The system will push notifications to the user's device when the specified time periods are met including the number of remaining ounces to drink as calculated by the system.

4.4 User Registration/Login/Log Out

Users will register a unique account using their user-defined username and password, and log in and out of this account as they wish.

4.4.1 Description and Priority

Users will have a list of all their previously recorded drinks. This list will contain the size of the drink logged as well as the time it was logged. This feature is a medium priority.

4.4.2 Stimulus/Response Sequences

Registration

1. From the main screen, the unauthenticated user will tap a button to navigate to the user settings screen.
2. The user will enter a username, password, and password confirmation
3. If the password and the password confirmation match, the application will try to register the user with the rails server. If there is an error the registration form will reset and an error message will be displayed.
4. If the account creation is successful the server will send the device a device token, which it will use to identify itself and the user in future requests.
5. The registered user will be logged in as the current user for the mobile application.

4.4.3.1 Functional Requirements

- REQ 4.1: The system should provide a registration button if there is no user is logged in.
- REQ 4.2: The system provides fields for the user to enter a username, password, and confirm their password.
- REQ 4.3: The system will provide a submit button to create a user and log them in.

Log In

1. From the main screen, the unauthenticated user will tap a button to navigate to the user settings screen.
2. The user will enter a username and password.
3. The application will send the username and password to the server.
4. If the authentication information is valid, the server will respond with a device token. Otherwise, it will respond with an error and the mobile application will reset the log in form and display an error message.
5. The device token will be saved to be used in future requests.

6. The registered user will be logged in as the current user for the mobile application.

4.4.3.2 Functional Requirements

- REQ 4.4: The system should provide a log in button to authenticate a previously created user.
- REQ 4.5: The system should return a 300 code if the log in isn't valid, and return a device token if the log in is valid.

Log Out

1. From the main screen, the authenticated user will tap a button to navigate to the user settings screen.
2. The user will press a log out button and the device token will be deleted and the user will be logged out of the mobile application

4.5.3 Functional Requirements

- REQ 5.1: The system should provide the functionality to pull down on the screen from the drink history screen and trigger synchronization.
- REQ 5.2: The system should send a POST request to the server containing all the drinks in the device's local database and receive from the server a list of the current drinks between devices for the user.

4.5 Synchronization

Users will sync their data with the system's server in order to be up-to-date between devices.

4.5.1 Description and Priority

Users will be able to pull down on the history screen, which will synchronize their drink history with the history recorded for the user on the server. This feature is of low importance.

4.5.2 Stimulus/Response Sequences

1. User will view their drink history.
2. User will pull down on the screen in order to trigger the synchronization feature.
3. The system will post the drinks in the history to the server, and the server will respond with the current history between devices for that user.

4.5.3 Functional Requirements

- REQ 5.1: The system should provide the functionality to pull down on the screen from the drink history screen and trigger synchronization.
- REQ 5.2: The system should send a POST request to the server containing all the drinks in the device's local database and receive from the server a list of the current drinks between devices for the user.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Performance issues are not expected form TapWater. Simplicity is a fundamental element of the design.

Performance will vary depending upon device and system version.

Changing between screens and processing button presses will be handled by the built in features of the SDK for the given mobile device and should perform at device expectations. Synchronization time may vary depending on network connectivity. The data exchanged should be optimized to assure the fastest synchronization possible. A local cache of drinks will be kept by the system so that it does not have to load data from the server repeatedly.

5.2 Safety Requirements

Use of the system should not present any harm to the user's device.

TapWater should not be used while operating a motor vehicle or any other kind of dangerous machinery.

Users should not use TapWater to consume more than a healthy amount of water. What is considered a "healthy" amount of water varies from person to person. If a user is uncertain they should seek the advice of a doctor.

Users should not use TapWater if they have a condition in which water consumption may damage their health.

5.3 Security Requirements

The drink data is not sensitive information. It is still private user information and should be secured using standard database security protocols.

The system communicates usernames, passwords, and device information over the network. This communication should be encrypted with an HTTPS connection. Additionally, the database should not store plain text user passwords.

Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.

5.4 Software Quality Attributes

The user interfaces should be appealing to the eye. The system should follow a consistent design style with a consistent color scheme.

The mobile applications should be available on their respective software marketplaces. The iOS app should be available on the Apple App store. The Android app should be available on the Google Play store.

The data synchronized to each of a user's devices should be consistent with the data on the other devices.

6. Other Requirements

The database on the iOS and Android apps will be an SQLite database. The schema will look like the following:

Drinks

Field Name	Description
uuid	A unique identifier for the drink.
category	The type of drink. Possible values are "drink", "glass", and "bottle"
drink_date	The date/time the drink was logged

The database on the Server will have the following schema:

Users

Field Name	Description
username	The user's username
password_digest	The password hash and salt for the user's password

Devices

Field Name	Description
device_token	The token this device will use to authenticate API calls
device_type	The type and system version of the device

Field Name	**Description**
user_id	Foreign key for the user associated with the device

Drinks

Field Name	**Description**
uuid	A unique identifier for the drink.
category	The type of drink. Possible values are "drink", "glass", and "bottle"
drink_date	The date/time the drink was logged
user_id	Foreign key for the user associated with the drink