Poly Brewers

Software Project Plan

Gregory Jans, Nicholas Porter, Tomas Uribe

Version 2.0
3/23/2023

Document Version	Date	Authors	Comments
1.0	2/6/2023	Gregory Jans, Nicholas Porter, Tomas Uribe	Meeting requirements for Deliverable 1 assignment
2.0	3/23/2023	Gregory Jans, Nicholas Porter, Tomas Uribe	Updated to reflect changes to planning, scheduling, and testing for Deliverable 2 assignment. Added use case diagram and split section 1.2 into functional and nonfunctional requirements. Refactoring of Data model and UML diagram also applied. Short description of Provider tool also introduced.

Table of Contents

1 Introduction

1.1 Identification

(Greg: 100%)
This document applies to the development of the Poly Brewers system version 1.0. Poly Brewers is developed for Florida Poly Course CEN3033 and seeks to solve the lack of recipe sharing sites for homebrewers.The software will be delivered in the form of a website hosted through Google Firebase as well as publicly available source code by the deadline specified by the professor.

1.2 Scope (Updated)

1.2.1 System Features

(Greg: 80% Tomas 20%)
The system will remain a free and open website and database for recipes. Any user with or without an account will be able to search for and view homebrewing recipes, as well as filter search results. Users who create an account will be able to save recipes for later, post their own, and possibly comment on other recipes. Inexperienced users will also be provided a page explaining the basics of homebrewing.

1.2.2 Other Nonfunctional Requirements

(Greg: 80% Tomas 20%)
The website should be able to retrieve recipes from searches and load each webpage within several seconds assuming a decent internet signal. The database through Firebase should be kept secure, especially in terms of password protection. All software will be developed through Visual Studio Code and use Github for its version control.

1.3 Document Overview

(Greg: 100%)
This Software Development Plan serves to guide the development of Poly Brewers. Software Quality Assurance practices will be outlined in Appendix A. The Software Configuration Management Plan, as designed by all group members will be outlined in Appendix B. Appendix C will outline the security and risk management plans for the system, with a general overview and explanation of the project security techniques used. Appendix D will briefly describe any project maintenance plans regarding changes after delivering the final Poly Brewers system.

2 Acronyms and Definitions

2.1 Definitions

(Nicholas: 60% Greg: 40%)

Recipe	A detailed explanation of the materials, equipment, and processes needed to brew a given home brew
Monday	A website used to maintain a shared activity board and project timeline
Flutter	User Interface framework built with the Dart programming language.
Cloud Firestore	Google Firebase’s NoSQL database located in Google Cloud Services. Uses folder based entities and document based records to store data
Cloud Storage	Google Firebase’s storage section for user generated data

2.2 Acronyms and Abbreviations

(Greg: 100%)

MAU	Monthly Active Users
SRS	Software Requirements Specification

3 Overview (Updated)

(Greg: 100%)
The project schedule and description for Poly Brewers was initiated January 24 with a target completion for late April. Incremental project deliveries may be requested, with two currently scheduled - the first being an updated SRS document and the second being a breakdown of the system architecture at both high and low levels. The delivery requirements will be the successful hosting of the Poly Brewers system and will be delivered and presented on a date specified by the professor. At this time, there will be no user manuals produced as part of the deliverable system. Detailed requirement and design specifications will be provided in the following document sections.

3.1 External Interface Requirements (Renamed)

(Greg: 100%)
The system's main external relationship will be its integration into the Firebase system, and as such will need to be configured to allow for communication to and from Google's servers. New brews entered by the users will need to be sent to the Firestore database, while all brews must be able to be retrieved from the database and displayed throughout the website. Additionally, the system will need to interface with Google's cloud services for web hosting. Finally, through Firebase, users will be able to login directly from their Google account and as such, the system will need to interface directly with Google authentication.

3.2 Source Code

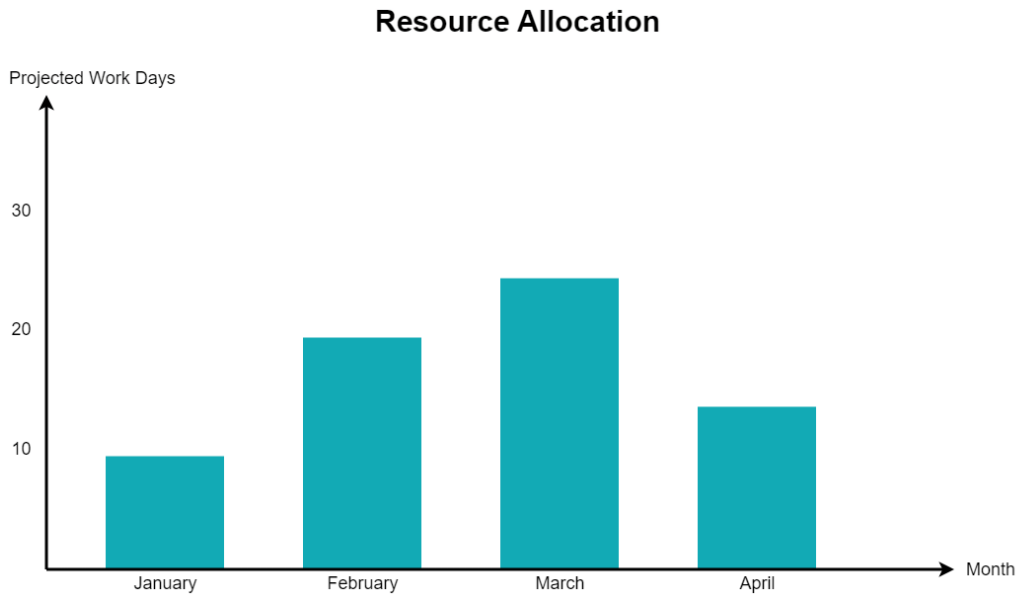
(Greg: 70% Tomas 30%)
The source code will be located in a public git repository on the Github website. The repository link will be https://github.com/GregJans/poly_brewers and will remain public throughout the project timeline.

3.3 Documentation and Training

(Greg: 100%)
For beginning homebrewers, Poly Brewers will contain a training page dedicated to explanations of processes and terminology. This will serve to mitigate any potential confusion involved with the homebrewing process. Apart from this page, there will not be any external training documents or manuals created regarding the use of the website itself. Internal project deliverables requested by the professor will be produced as needed by all team members in order to meet set deadlines.

3.4 Project Resources

(Greg: 100%)
The project team will not have a hierarchy, as each member will have an equal say and share equal responsibility. Gregory will be in charge of the UI developed in Flutter, Nicholas will be in charge of the backend through Firebase, and Tomas will be in charge of project deliverables and unit testing. Time spent on the project should be kept consistent through the months of February and March, however it is likely that additional time will be required in March in order to remain on schedule. As the final deliverable is due in April, the number of days worked will need to be shorter.



3.5 Project Constraints

(Nicholas: 80% Tomas 20%)

The software associated with the project will be constrained to the Dart language with the Flutter framework for UI and implementation, all database and backed management will be constrained to the Firebase platform, and all task distributions will be limited to being posted on the shared Monday board.

Schedule constraints must also be considered, due to having to balance the project along with any other class work and work schedules. This may limit the free time available to work on the project considerably, in addition to only having four months to fully develop the project.

Firebase related constraints will primarily involve cost in relation to monthly active users (MAU) and data usage. Initially, the system will be restricted to incurring no additional costs. If the system is successful, some costs may be incurred to accommodate additional scaling and data flow through Firebase. Initial limitations will be as follows:

Cloud Firestore:

- Allowed to store 1 GiB of data before Google cloud pricing of $0.18 per GiB per month
- Allowed 20k database writes per day before pricing at $0.18 per 100,000 documents.
- Allowed 20k database reads per day before pricing at $0.06 per 100,000 documents.
- Allowed 20k database deletes per day before pricing at $0.02 per 100,000 documents.

Authentication:

- Allowed 50k MAU until we are charged

Cloud Storage:

- Allowed up to 5 GB of stored user generated data, then we are charged $0.026 per GB
- Allowed up to 1 GB of user downloads per day, then we are $0.12 per GB
- Users allowed to upload 20k items a day, then we are charged $0.05 per 10k uploads
- Users allowed to download 50k items a day, then we are charged $0.004 per 10k uploads

Hosting:

- Allowed 10GB of website storage, then charged $0.026 for each additional GiB
- Allowed 360 MB/day of data transfer, then $0.15 for each additional GiB
- Custom domain name and including multiple sites per project always come at no cost.

4 Software Process

4.1 Software Development Process

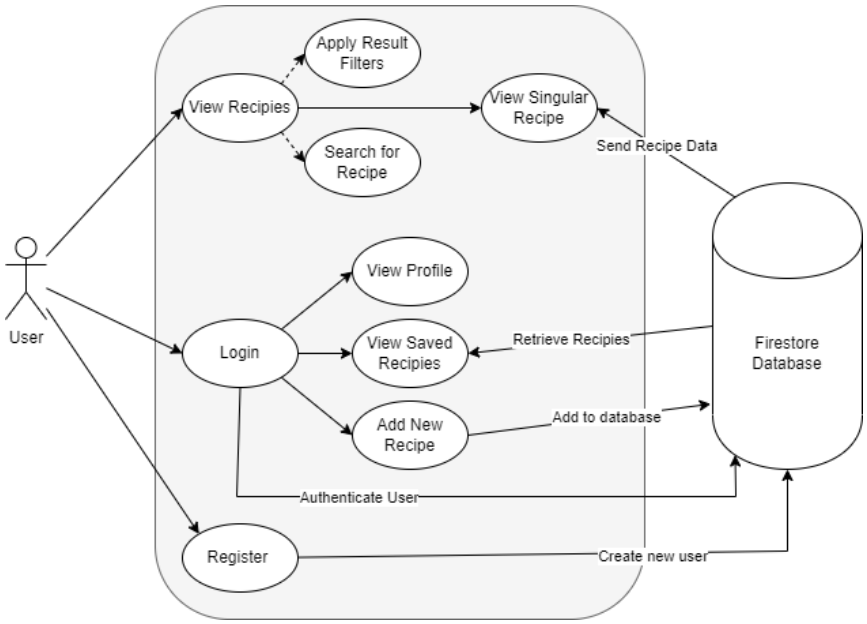
4.1.1 Life Cycle Model

(Greg: 90% Tomas 10%)
We will be following a Scrum development lifecycle with week long sprints. These weekly sprints will allow for each team member to have a flexible schedule, while the meetings will serve as a way to ensure progress is being made and that each member is made aware of any changes either to the system or the schedule. All team members are expected to participate in the weekly sprints and to contribute to project meetings.

4.2 Software Engineering Activities

4.2.1 System Use Cases (Added)

(Greg: 100%)



4.2.2 Handling of Critical Requirements

(Greg: 100%)
Critical requirements will be managed by team members throughout the development process. New requirements may be added as seen fit by the team, however any functionality described in this document must be implemented by the final deliverable.

4.2.3 Recording Rationale

(Greg: 100%)
Design and implementation decisions will be recorded through a shared board on Monday, which will list the tasks assigned to each team member. Any changes to the design of the system will be reflected in the tasks and subtasks that are assigned. It is up to each team member to decide how functionality assigned for them to complete will be implemented into the system.

4.2.4 Data Management Plan (Updated)

(Nicholas: 100%)
User data will be stored and managed through Firestore which is part of the Firebase platform. The initial recipe database will also be stored on Firebase and will contain some sample recipes at time of release, with additional

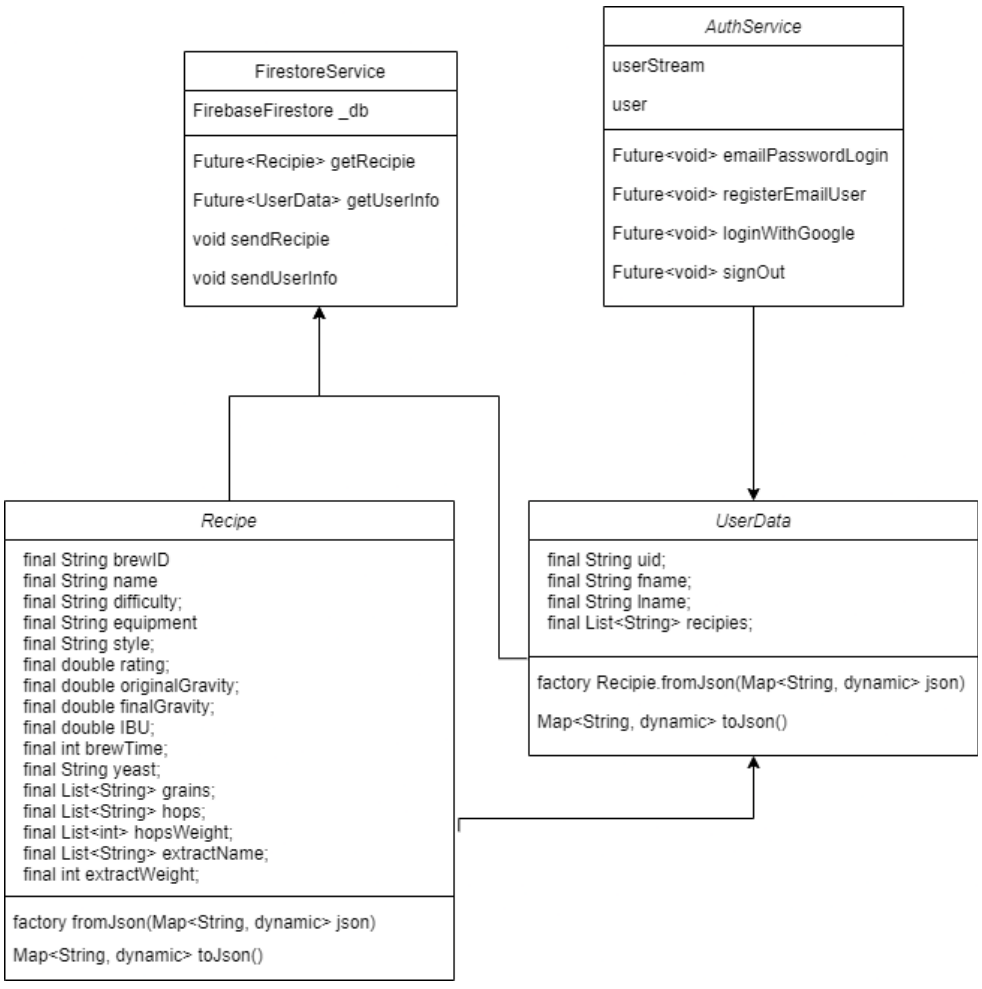
recipes being gathered over time through user recipe submissions.

Primary storage in the Firestore database will follow a NoSQL model wherein there are no direct relationships between entities. Any entity relationships will be created using the equivalent of primary and foreign key pairs. Each entity will be stored in a folder in a partition of Google Cloud services, and within each folder will be several files of which attributes are stored in a json-like format. The format of the folder can be modeled as such:

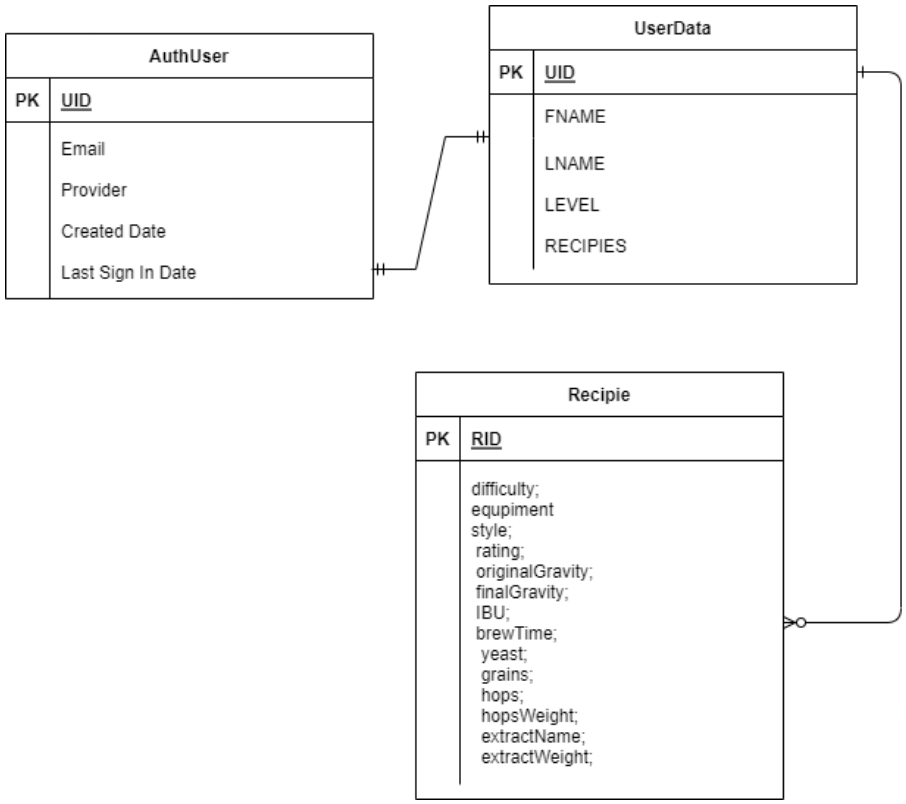
Folder name: User
File: User1

UID: 11123456
FNAME: Harrison
LNAME: Bergeron
HEIGHT_IN_FEET: 7

Data may also be placed under a header if there are many types of an attribute, such as a list of hop types. All queries from reads to writes as well as major security considerations such as type checking will be handled via the code in the flutter framework.The essence of the data's structure of the application revolves primarily around the user and the recipes they create. Identifying each user is handled by Firebase's built in authentication system, which generates a user ID and keeps the user email address. Extra user data such as user's name and a dynamic array of recipes owned by the user will be stored in a Firestore collection known as UserData. Then for each recipe there will be a "BrewID" assigned to each Recipe. Both types of data are then accessed and written via the Firestore Service class methods which parses data into a Map data structure and is passed in the backend as JSON. Style guidelines for recipes will follow the publicly available recommendations on the Brewers Association[1]. Below is a UML diagram focused upon how the Objects interact with each other in this regard, as well as a Entity Relationship model that relates how each major datapoint share their relations.



[1] The Brewers Association guidelines can be found here:
https://www.brewersassociation.org/edu/brewers-association-beer-style-guidelines/



Data piping throughout the project is handled primarily by the Provider library. This tool allows us to define a list of data instances we desire to have access to across the app by defining the provider at the head of the app in the main function and calling upon the provided data wherever needed.

4.2.5 Computer Hardware Resource Utilization

(Greg: 100%)
Hardware resources such as server storage and hosting will be managed entirely through Google as part of their cloud platform, rather than through the Poly Brewers system. This service will be managed through a free account belonging to one of the group members.

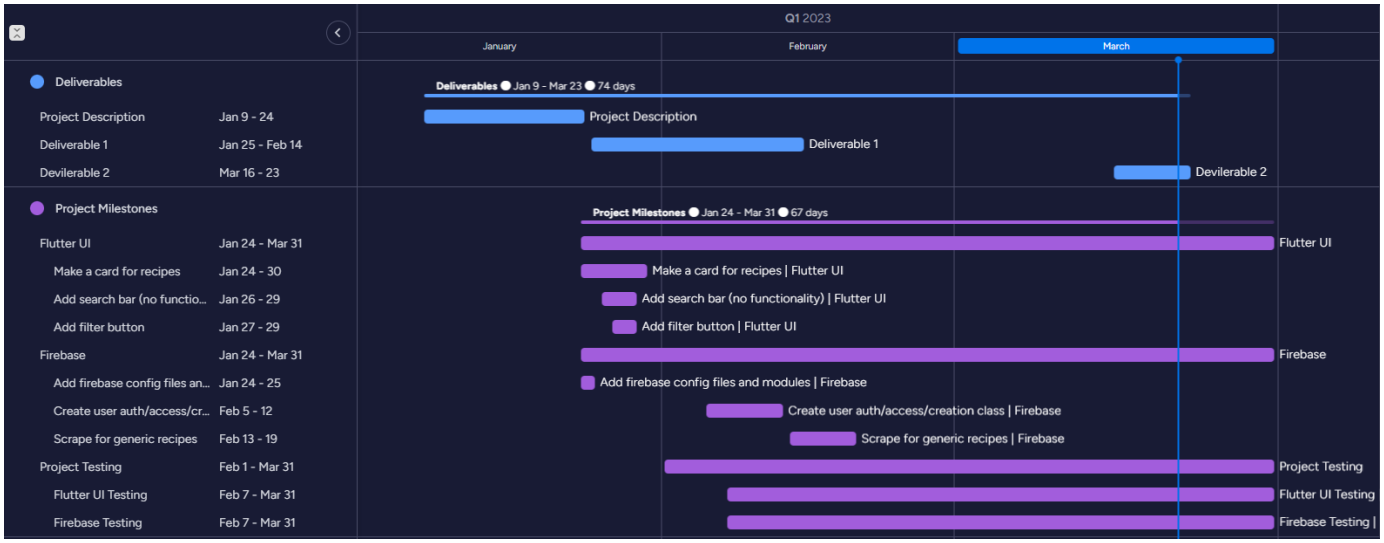
4.2.6 Software Testing (Updated)

(Tomas: 80%, Greg: 20%)
Unit tests are conducted using Mockito and are used to test individual functions, methods, or classes. These tests are used to verify the correctness of logic under a variety of conditions. Additionally, widget tests are conducted using the built in testing tools in flutter, which will test an individual widget to ensure the UI looks as expected and has the correct functionality. Widgets being tested should be able to receive and respond to user actions and events as well as instantiate child widgets. Finally, Integration tests will be used to test the entire system or large portions of it, and will ensure the widgets are interacting with the backend as intended.

5 Schedule

5.1 Timeline (Updated)

(Tomas: 40%, Greg: 30%, Nicholas: 30%)



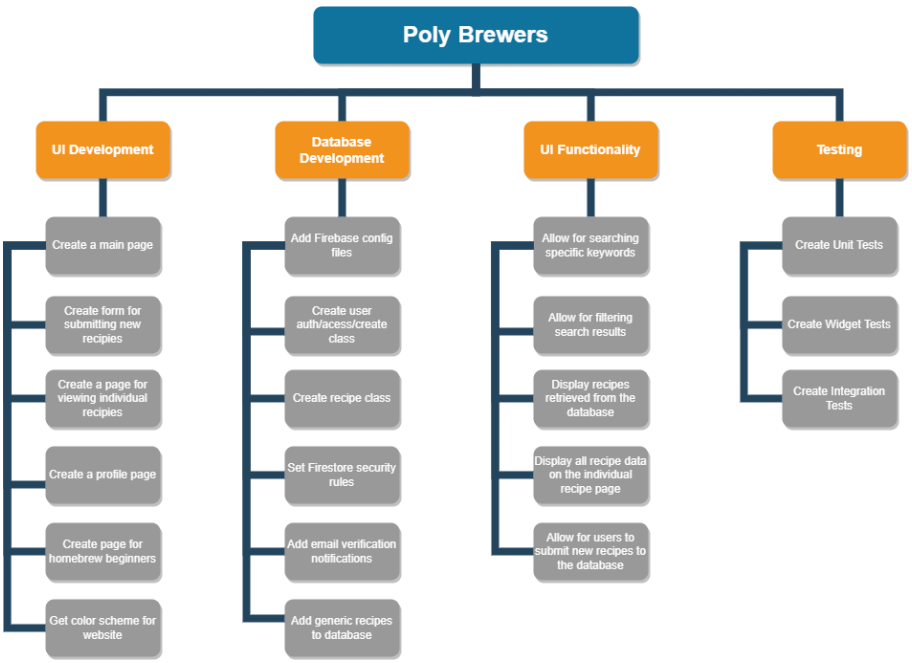
5.2 Activity Graph

(Tomas: 40%, Greg: 30%, Nicholas: 30%)

Refer to the Gantt chart for scheduling and task breakdowns.

5.3 Work Breakdown Structure

(Greg: 100%)



Appendix A Software Quality Assurance

Evaluations And Corrective Action

(Greg: 100%)
Developed software will be tested for quality assurance as per the testing plan outlined in section 4.2.5, with integration tests serving as the main evaluation for Quality Assurance. Integration tests will serve to ensure that the system functions properly in any use case and has successfully met all requirements outlined in this document. Tests will be run periodically throughout the project's development in order to catch issues as early in development as possible, with the entire system also undergoing extensive testing between the development and deployment phases to ensure all requirements have been met. Any issues or unmet requirements found during integration testing, either before or after development, must be reported to the entire team. The responsibility for fixing the issue may fall to either the team member whose role is most closely related to the issue or the team member with the most available time to fix said issue.

Targeted User Profile

(Greg: 100%)
The targeted user is defined as anyone with an interest in homebrewing. The user can be of any skill level, although the system may be more easily understood and utilized by users with a greater knowledge of homebrewing procedures and terminology. Users should not require any additional technical or outside expertise in order to utilize the system. While the website will be available to users of all kinds, this profile outlines those who will likely find the most use in the system.

Appendix B Software Configuration Management

Source Code Configuration

(Greg: 100%)
The system will consist of shared files on the specified git repository. As such, all files will be shared among all team members. File names will not follow additional naming conventions apart from those put in place by the dart language, however file names should not be changed unless necessary. The repository will consist of one main branch which will be used for deployment and hosting. Each team member will have a personal branch following the format of "dev-[member first name]" where they will push any code changes and where each push consists of a descriptive title and any additional notes. After changes are pushed to the individual branch, they may be merged into the main branch only if there are no merge conflicts. All developers are expected to pull any changes from the main branch into their personal branch before modifying files in order to mitigate potential merge conflicts. Any conflicts should be managed by the developer who initiated the merge. The system will be continually developed throughout the timeframe provided, but will not follow a versioning scheme.

Additional Deliverable Configuration

(Greg: 100%)
Additional documents and deliverables will be handled in the same manner as source code, and as such will remain as one file to be worked on by each team member. Documents will be created through a shared Google Doc to allow for work to be done by multiple team members synchronously and will utilize the built-in versioning functionality to monitor changes and revert to previous versions if needed.

Appendix C Security and Risk Management

Security Plan

(Nicholas: 100%)
Data security and password management will be handled through Firebase Which offers many methods of authentication via email and third party accounts. Primarily, however, the Google Identity platform on Google's Cloud server will be implemented as a Firebase method utility. The identity platform will provide access to utilities such as "signInWithEmailAndPassword", which will asynchronously await for user input in the two fields and will then verify that the email and password combination is within the database. Additional security rules further defined by the Firebase platform may be added, such as allowing read/write access to certain parts of the database for each user. Additionally, user input will be validated via Flutter's built-in type checker.

Project Security Description

Security Requirements

(Nicholas: 100%)
Data security should be primarily focused on the authentication of users. Users should be authenticated before making changes to the database, either through creating a new recipe or altering previously posted recipes. The system domain, as deployed through Firebase, should also be kept secure and needs to be protected from take-down attacks. Data security on the recipes themselves will be kept to a lower priority, as it will not contain any personally identifiable information.

Access Control and Authentication

(Nicholas: 100%)
Users will have access to using email and password as well as their google account for authentication methods. Likewise, two factor authentication will be introduced into the system as a means of further securing authentication. Any password resets or verifications will take place via email to the user from the Firebase manager. Additional security rules for the Firestore database will be used to control the read and write privileges of users for the database from the front end.

Secure Design by Default

(Nicholas: 100%)
Users entering the site will only be allowed access to view already posted recipes. Any additional functionality from the system will require the user to log in or create an account, which will be verified through two factor authentication.

Possible Component Vulnerability

(Nicholas: 100%)
Given that Firebase is relied upon heavily for data storage and website hosting, it may pose a vulnerability to the Poly Brewers system. Firebase has had a vulnerability in the past known as "HospitalGown"[2] which caused the service to leak many databases being stored on the platform. As such, Firebase may be subject to future vulnerabilities which will need to be prevented if possible and handled when vulnerabilities are not properly prevented.

Secure Logging and Monitoring

(Nicholas: 100%)
User activity will be monitored via the Firebase dashboard console. This console will generate logs of user actions and log-in sessions. Alerts to potential attacks such as high traffic may also be set up in the cloud's firestore and hosting components.

[2] More information about HospitalGown can be found here: https://gbhackers.com/firebase-vulnerability/

Input Validation

(Nicholas: 100%)
User input will be type checked to match the data set for components of the Firestore database. For example, original gravity and final gravity will be set as float types (will never exceed three decimal values). Likewise the name of a recipe must be of type string lest an exception be thrown and handled by creating a user error message and allow for another chance at input for a certain number of times.

Risk Management Plan

(Nicholas: 100%)
The primary point of risk within the scope of the project will be the Firebase integration. The aforementioned security vulnerabilities in using a backend framework would require a large amount of trust in the providers of the framework, however, some some risk management can be done, namely:

- Ensuring that Firebase is up to date by scripting the various package managers available to the system whether it be pub get or the flutter-fire in the backend
- Making sure the admin password (i.e. the Google account password connected to Firebase) is highly secure and changes every quarter
- Following the security best practices delineated in the security plan above

Beyond technical risks, human factors must also be considered. Scheduling and timeframe issues may affect development time, as team members may need to focus on additional projects and work. Efficiency will also not always be guaranteed, due to varied workloads and personal commitments. This is the primary reason a more agile SDLC has been chosen for this project, allowing for team members to have more leeway in how often and to what extent project phases should be accounted for. In the event that an element should not be completed in a reasonable amount of time, it is ultimately up to the team member to communicate difficulties and seek out assistance should the situation become dire enough.

Finally, there is the question of legality of our chosen problem domain. Some jurisdictions might not be friendly to the idea of a website which teaches people how to make alcohol should the consumption be illegal in said area. It has been concluded for now that the website may be blocked in those areas, and should it not be, a disclaimer will likely be placed in the about section. It should also be noted that no homebrewing supply website displays these facts prominently, though it is certainly mentioned.

Appendix D Project Maintenance

(Greg: 100%)
There will not be much in terms of a maintenance plan, given that the hardware will be handled by Google’s cloud services and there will be no additional training materials created for explaining the overall use of the website. Some changes to the code may be allowed, however, in order to maintain or update the system in the future. These changes will be made only if a significant bug or error is found within the website and will not be based on a set update schedule.