Short-Term Memory System (STeMS)
Programmer Guide
By AlphaSoft

Team Members

Awasthi, Aaditya

Blavat, Oleksiy

Bond, Matthew

Carter, Mackenzie

Mahbobi, Sayed shah

McAllister, Charlie

McCool, Max

McLaughlin, Taylor

Powers, Michael

Weaver, Daniel

Revision History

Date	Version	Description	Author
22 July 2023	1.0	Initial submission to Client	AlphaSoft
5 August 2023	2.0	Updated to match current state of project.	AlphaSoft

Table of Contents

Figures & Tables

1. Introduction

1.1. Purpose

This Programmer Guide (PG) describes why the architecture and system design of the Short-Term Memory System (STeMS) is structured the way it is. The full system architecture, including both functional and non-functional requirements, dependencies, workflow, and risk assessments is provided in the Technical Design Document (TDD) and will only be summarized here. This document, instead, explains the rationale behind the TDD. Not only why some decisions were made, but also why others were rejected.

1.2. Intended Audience

This document is intended to be used by current members of the AlphaSoft Team and future programmers providing support to STeMS. It is a reference for the architecture and design decisions made for STeMS during initial development. Senior and junior developers at AlphaSoft will find this document useful when enhancing the product to make sure that future functionality does not conflict with existing rational. This makes for a more consistent product and project, both of which are signs of professionalism.

1.3. Project Documents

This Programmer Guide will be included in the documentation portion of the project deliverables. All documentation created throughout the project lifecycle is done so with the intent to assist in the understanding, implementation, and maintenance of both the mobile application and browser extension.

The following documents will be included as part of the project deliverables.

Document	Version	Date
Project Plan (PP)	4.0	5 August 2023
Software Requirements Specification (SRS)	4.0	5 August 2023
Technical Design Document (TDD)	3.0	5 August 2023
Software Test Plan (STP)	3.0	5 August 2023
Programmer Guide (PG)	2.0	5 August 2023
Deployment and Operations Guide	2.0	5 August 2023
Software Test Report (STR)	1.0	5 August 2023
User Guide (UG)	1.0	5 August 2023
Traceability Matrix (TM)	1.0	5 August 2023

Table 1: List of project documents.

1.4. Acronyms, Definitions, and Abbreviations

Throughout this PG, a variety of terms and acronyms specific to the proposed application are used. For clarity, their definitions are provided below:

Term	Definition
AI	Artificial Intelligence
API	Application Programming Interface, or a way that difference applications interact with each other
App	A program that is included on the App User's mobile device
BESie	Back End Service
Browser Extension	An extension to either the Chrome or Safari browser, which works with the App to provide specific functionality
Conversation	A recording the App User made and saved within the app
ConvoBuddy	The app under development by this project
Diarization	The ability of STT (speech to text) programs to be able to distinguish one speaker from another.
DOM	Document Object Model
EULA	End User License Agreement
Flutter	A software framework for developing cross-platform mobile applications
HTTP	Hypertext Transfer Protocol
iOS	iPhone's Operating System
JSON	JavaScript Object Notation
Mobile Device	A smart phone, tablet, or some other portable computer with either the iOS or Android operating system
LLM	Large Language Model, a type of AI algorithm
OS	Operation System, like Android or iOS
PG	Programmer Guide
PP	Project Plan
Deployment and Operations Guide	A document that explains how to deploy the program
SRS	Software Requirements Specification
STeMS	Short-Term Memory System, the name of this project
STP	Software Test Plan
STR	Software Test Report
STT	Speech to Text

Term	Definition
System	The complete STeMS system, which includes the App, the Brower Extension, and the Browser Extension Service
TDD	Technical Design Document
TM	Traceability Matrix. A document that traces defects and test cases back to their requirement.
TMGR	Transmogrifier
UG	User Guide
UI/UX	User Interface / User Experience
URL	Uniform Resource Locator, typically a path to files on a computer

Table 2: Acronyms, Definitions and Abbreviations

1.5. Organization of Document

This document has five sections: Introduction, Technical Summary, Rationales, Program Code Details, and Appendix. The Introduction gives an overview of this document, who it was written for, and other helpful information to understand the rest of the document. The Technical Summary section is a brief overview of information in the TDD so that readers can understand the necessary technical details in the rest of this document without having to go through the entire TDD. This is a convenience to the reader and the TDD should be referred to for explicit details and clarifications if the reader has any doubts.

The next two sections are explanations for why decisions were made with various aspects of the STeMS project. They include supported operations systems, languages used, the architecture (both high-level and low-level), entity relationships, designs, cross-cutting concerns, testing tools, and specific complex code algorithms. Rationales and details are provided for all three parts of STeMS: the app, the web service, and the extension. Every program has bugs, but many do not impact functionality enough to prevent a release of the product. The reasoning for why all known bugs were not fixed is also included.

The last section, Appendix, covers all remaining items that need to be explained but do not fit well elsewhere. The sub-section Code Files explains how the code files are organized. The Bug Tracker section explains how the bug tracking system works and why these procedures must be followed.

2. Technical Summary

This section provides minimal information to the reader so they can understand the larger context of the details provided in later sections. Please see the Technical Design Document (TDD) for more details of the technical design.

2.1. Business Case

The client has contracted with AlphaSoft to develop software that will assist people with short-term memory issues. The software needs to run on a mobile device, record conversations, and use AI to assist the user.

2.2. Architecture

There are three major components of STeMS: the ConvoBuddy app, the web service BESie (Back End Service), and the browser extension. The browser extension and BESie are tightly coupled. The app is the most user-facing part of STeMS. The app also works with the AI and stores all the data. It uses a process called Transmogrifying to convert the recording into useable information for the user via ChaptGPT.

A brief overview of how the browser extension and BESie work with the app follows. The browser extension will scrape the DOM of a web page and send that payload to BESie. BESie is a pass-through service, forwarding the payload to the correct instance of the ConvoBuddy app. The app will interact with the user to process the request and send the results back to BESie. BESie will again pass the results to the browser extension, which will update the DOM.

The ConvoBuddy app has three main screens: The Conversation List screen, the Conversation Details screen, and the Information screen. The Conversation List screen shows all saved conversations and allows the user to search and sort them. From here, the user can select a conversation to see its details. The Conversation Details screen allows the user to edit or delete the conversation. They can also play the recording back and see the results of different transmogrifiers.

The number of screens was kept to a minimum in an attempt to maximize the intuitiveness of the program. By keeping the design simple, users will spend less time having to search for what they seek.

2.3. Typical Use Scenarios

The client laid out several scenarios as examples of what the app should be able to do.

- Convert the recorded conversation to text using diarization and display the text of the conversation.
- Create reminders based on the conversation.
- Parse the food orders for different people.
- Provide a summary of the conversation.

The AlphaSoft team has considered other scenarios where STeMS that might be useful:

- Lawyers talking to clients will have the exact length of their conversation for billing and the text of the conversation to make sure they do not forget any details.
- People that are hard of hearing could use the STT feature to read the parts of the conversation they miss.

2.4. Licensing

At this time, the ConvoBuddy app and browser extension are being offered free of charge to consumers. However, because the ConvoBuddy app records conversations, there is the potential for

illegal use of the app. Therefore, a EULA (End User License Agreement) will be put in place for all users of the app.

There has been some consideration of a free and a paid version of the program, but that has not yet reached the stage of development's concern.

3. Rationales

3.1. Operating Systems

The app is being designed to work with mobile phones. The most popular of these modern phones use Android and iOS operating systems. By choosing the most popular operating systems (OS) the app can effectively help the most users.

Despite Windows being the most populate OS, the app is not being targeted for Windows specifically. This is because the Windows OS is used only for less portable computers, and the ConvoBuddy app needs to be very portable to be the most effective for the user.

The browser extension works within any Chromium (R16 or higher) based browser regardless of the OS that is running the browser.

3.2. Development Languages and Frameworks

3.2.1. ConvoBuddy App

DART 3.0.6 is the primary programming language used to develop the ConvoBuddy app. Leveraging the Flutter 3.10.6 framework, DART allows the development of platform-agnostic mobile applications, ensuring a consistent user interface between iOS and Android devices. The selection of Flutter can be attributed to its robust networking libraries, superior performance, and ease of development.

3.2.2. BESie

BESie is built using Java 20 and the Spring Boot framework (v 3.1.1). Spring Boot's flexibility enables BESie's deployment across various operating systems, cloud services, or on-premise servers, adding to its robustness. Its ease of use and the development team's expertise further influenced this choice. Spring Boot allows for efficient and concise coding, simplifying the creation of server endpoints, REST controllers, and WebSocket controllers.

3.2.3. Browser Extension

The browser extension is primarily developed using HTML 5 and JavaScript (ES13). SockJS and STOMP JS are the libraries used for WebSocket connections and STOMP client. Bootstrap is used for out-of-the-box modern UI components like text inputs and buttons. jQuery 3.7.0 is used for advanced DOM manipulation.

These languages and libraries provide flexibility and compatibility across any browser that supports web API calls. The decision to use these foundational web technologies was driven by the development team's expertise, the need for broad browser compatibility, and the functionality they provide that makes coding this area easier and faster.

3.3. Architecture

3.3.1. Overview of Stems Architecture

The STeMS architecture comprises several components, meticulously designed to perform in harmony for an optimized user experience. The primary component is the user-facing mobile app, which is built with the Flutter framework and the Dart language. Supporting components include two services and a browser extension.

3.3.2. Key Supporting Components

The mobile app is backed by three elements:

- The Transcription Service/AI Component
- The Browser Extension Component
- BESie

3.3.3. BESie: The Intermediary Component

BESie, a core part of this architecture, serves as a broker between the app and its supporting elements. It handles the critical task of message passing, ensuring smooth communication between the app, the browser extension, and the supporting artificial intelligence services. Having a central broker facilitates setup and configuration of the browser extension's ability to interact with ConvoBuddy. With BESie, this setup and configuration can be done without any user intervention.

3.3.4. The Concept of Transmogrifiers

In our context, "transmogrifiers" refer to any operation performed on a recorded conversation. Depending on their functionalities, transmogrifications might involve all or some of the components within the architecture. All of the transmogrifiers require internet access to communicate with either of the services. Transmogrifiers provide a common language for talking about all of the ways that the ConvoBuddy app can process a recording.

3.3.5. Main Application Functionality

The mobile app's main function is to transcribe audio conversations from speech to text. It leverages Amazon Web Services (AWS) for diarization and speech-to-text services. Once the transcription is completed, the app forwards it to OpenAI's ChatGPT for analysis and pertinent information extraction.

AWS was chosen for STT because it provides diarization services. Other services that offer diarization either were not as robust, cost more, or were more limited in their capabilities. As this project scales up, attention should be paid to external services used in case other options become cheaper or more applicable.

3.3.6. Form Filling with ConvoBuddy

Form filling is the process of a user interacting with ConvoBuddy to populate a webpage. Upon encountering a web form, the user can activate the ConvoBuddy browser extension, which prompts the user to enter a unique code. The user submits the code and then receives a notification in ConvoBuddy. From the app, the user selects a conversation. Back on the web form, the form is filled out with information in the conversation.

This code is used as a form of authentication. The ConvoBuddy app generates a unique code on first launch that is then registered with BESie along with enough information for BESie to communicate later with the app. When the user submits the code in the browser extension, it is sent to BESie with the web form's information. This code allows BESie to know which instance of the app to send the request to. This seamlessly sets up BESie without the user doing anything extra, allowing BESie to broker all communication between the extension and the app.

3.3.7. Interaction with OpenAI's API

Any time the user starts a transmogrification, the app checks to see if a result already exists. If so, then it skips the call to the API and processes this saved result for display. If a result is not saved, then it sends a request to OpenAI's API. This request includes the transcript of the conversation and prompts for the AI to process the conversation. The returned data is then stored with the selected conversation.

This process flow separates out the areas of storage, API calls, and UI. By compartmentalizing the code, each class focuses on one thing and does it well. Also, by storing every response from the API, there is no need to re-do the transmogrification. ChatGPT's AI has a randomizer; therefore, this process prevents the user from getting two different results for the same transmogrification.

3.3.8. Information Exchange and Communication

Both the app instances and the browser extension subscribe to a topic to exchange information. BESie also facilitates other forms of communication, such as with AWS and between the browser extension and the mobile app.

This architecture was chosen because having a central broker allows for some efficiencies and a decoupling of the code. As a central broker, changes can be deployed at any time, allowing fixes and improvements to not be delayed. Also, version management is easier as the browser extension has a separate version from the app. Either one can change without a required change in the other.

Additionally, topics are used because they are scalable, efficient, and the general standard for sending notifications in a distributed system. There are also many libraries to support this code pattern.

3.3.9. BESie as a REST Controller

BESie also functions as a REST controller, providing a base for future functionality expansion. This forward-looking approach ensures the system remains flexible and adaptable to future technological advancements or additional requirements.

3.4. UX Design

The UX design combines ease of use, a lot of functionality, and scalability. Each of these will be detailed in turn. The target audience of the ConvoBuddy app are people with short-term memory issues, either because of physical disability or work-related distractions. The former group is overwhelmingly older than average. Studies have shown that building for the most disabled makes products better for everyone. Knowing that UX design does not have fixed rules, these ideas were our guiding principles.

3.4.1. Ease of Use

Ease of use is composed of multiple areas: fonts, colors, size of controls, process flow, and intuitiveness. The app must work for those with poor eyesight because of the skew towards older individuals. This means fonts must be bigger than normal and colors should have high contrast. Color blind people should also not have any issues when using the app. The size of the controls (buttons, textboxes, etc.) must not be small either.

Process flow is about how the user progresses through different features. From the first screen that displays (discounting first-time use prompts), all features can be accessed within three taps. This means that a user does not need to remember a lot of steps to get through a process. Sufferers of short-term memory loss should find this much easier to progress through a feature. A competing trait of a short process tree is usually a wide selection of options at each step. The more options presented to a person, the more likely they will choose incorrectly. Our design overcomes this by keeping the app simple and organizing it into cohesive units.

3.4.2. Lots of Functionality

Although there are only four main screens, there is a lot of functionality packed into them by proper organization.

The opening screen is the Conversation List screen. This includes a list of all saved conversations; therefore, all options to sort and filter the list are here. Because this is the home screen, it must also

provide access to all other features. There are only 3 other things that can be done at this time: go to the Information screen, view a conversation's details, or start a recording.

The Conversation Details screen has all information and functionality related to a single conversation. Editing the title, deleting it, playback of the recording, and viewing and running transmogrifiers all can be done from this area. The area is organized in a top-to-bottom approach. The meta data for the selected conversation is at the top, including the title, date of recording, and length of recording. The delete button is in the top right corner so that it is away from the most used area of a phone (the bottom corners) to reduce accidental deletion of the conversation. Below the meta data is the list of possible transmogrifiers. Selecting a transmogrifier shows the results of that process below it. By only showing the selected transmogrifier's results, the screen stays clean and focused on the last action the user took. The selected transmogrifier will show in a different color as well to help remind the user which one they are viewing. At the bottom is the recording playback button.

The Recording screen only does one thing – it controls the recording. There are controls that display what is going on with the recording in real time to assist the user with decision making. There are also only two options presented at a time. First, pause and stop are available. If pause is pressed, then only resume and stop are available. By pressing the Stop button, all other functionality happens automatically, saving the user from having to deal with additional decisions after Stop is pressed, and prevents any delays in starting another recording. This order of operation was decided on to make it easiest for a user to start another recording so as not to miss any crucial conversation.

The last of the main screens is the Information screen. This is an odd collection of things that did not warrant a full screen individually. By consolidating them, the user can see all of these things at a glance instead of digging through screen after screen to hunt them down. This screen contains the browser extension's code, a button to show the EULA, and a button to launch the guided tour. The EULA and guided tour are minor screens with very limited functionality.

3.4.3. Scalability

There are a few places where scalability influenced the design of the Conversation List and the Conversation Details screens.

In the Conversation List screen, the list of conversations is effectively limited only by the size of the phone's storage. With the large number of conversations, there had to be a way for the user to find a conversation faster than scrolling through a list. That is why the sort and filter controls were added.

In the Conversation Details screen, the list of transmogrifiers was put on a carousal (sideways scrolling list). This serves two purposes. First, as more transmogrifiers are added, the UI design does not need to change to accommodate them. Second, it allows more room for viewing the results, which could be lengthy. By showing the results on only one selected transmogrifier at a time, it is easier for the user to determine which result is being shown and prevents the need for additional sort and filter options to find the result one needs.

3.5. Security

Native device security is used to keep the app's files secure. The architecture was designed to minimize exposure of conversation recordings and details, and local storage will be used to help in this regard. When exposing data outside the app, such as with the form filler function, a unique number identifier will be used to secure the transaction between the browser extension and the application.

To secure the interactions between BESie and OpenAI, a secure WebSocket has been created to manage those communications.

At this time no user verification is required, but it may be an additional feature in future versions.

3.6. Communication Protocols

Constructed using Java Spring Boot, BESie operates as a WebSocket server with a bidirectional pipe: one direction for app-to-browser extension communication, and the other for extension-to-app communication.

The WebSocket protocol fits into the context of STEMS because it provides a seamless experience for the user. Streaming complex data could be accomplished with TCP or UDP, but the WebSocket protocol has less of an implementation overhead.

3.7. Testing Tools

The ConvoBuddy app portion of STeMS will be tested on both Android (version 11 and higher) and iOS (version 12 and higher) systems to ensure multi-platform compatibility. Android testing will be conducted using Android Studios (Flamingo, version 2022.2.1). Android Studios provides an Android device emulator that allows for multiple architectures to be used. The Android emulator will also allow for the AlphaSoft QA team to consolidate their testing environment to a single device. iOS testing will be conducted using an available mac laptop running the latest iOS version. This method will again allow for the consolidation of test environments to a single device while ensuring the app is operational for all Apple products.

Testing for the browser extension portion of STeMS will be conducted within the chrome web browser. This allows for easy installation of custom browser extensions and is available on all current operating systems. The ConvoBuddy application and browser extension will both be installed on the same local host, allowing for the consolidation of QA testing environments.

The BESie portion of STeMS will be deployed on testing environments using the most recent stable versions of VS Code, NetBeans, or Eclipse. Any of these IDEs are acceptable as they can run Java and deploy BESie.

STeMS will utilize both manual and automated test suites. This will allow for optimal time for both testing and documentation efforts. All manual test cases have been stored within Microsoft Excel to allow for ease of traceability between testing steps and their associated requirements. Results from these testing efforts will be documented within the Software Test Report (STR).

For automated testing of ConvoBuddy, we are using APPIUM 2.0 as an automation tool, as this is free and provides the flexibility to emulate the mobile app in different devices. This tool also supports Android and iOS devices. Another advantage of this tool is the flexibility for testing different languages such as Java, JavaScript, Python, Ruby, and C#. While we are using Java now, this will allow us to easily accommodate new languages to the project and easily test them when scaling up.

4. Program Code Details

4.1. Entities & Classes

4.1.1. ConvoBuddy Entities

ConvoBuddy depends on one core model – the Conversation class. Each Conversation is required to have a unique ID, the file path to the recording audio, the date of the recorded audio, and the duration of the recorded audio. The two most important fields are the ID and file path. These two fields will be passed to the backend to initiate all the transmogrifier processes.

There are also four fields that are instantiated with default values but each are expected to be populated by transmogrifier processes when they are completed. The fields are:

- Title: Simply the title of the conversation. This defaults to "NewConversation".
- Transcript: Holds the value of the Speech-To-Text result. This field defaults to an empty string.
- CustomDescription: A custom description of the conversation. This field defaults to an empty string.
- GptDescription: ChatGPT's summary of the conversation.

4.1.2. ConvoBuddy Custom Widget Classes

Two custom, stateless widgets were created to display the conversations and their details:

- ConversationListItem: This widget consumes a conversation and outputs important information about the conversation in an easy-to-read format in a list view. The ConversationListScreen maps each conversation to a ConversationListItem to display them in a list.
- TransmogListItem: This widget receives information about a transmogrifier process, such as the title of the process and an icon and constructs a pressable UI element that will show the user the results of the transmogrifier process.

4.2. Algorithms

4.2.1. Filtering and Sorting Conversations in ConvoBuddy

Filtering and sorting conversations are closely connected because there is always a filter and sort applied to the displayed conversations, even if that filter is an empty string. The code is constructed to always filter the conversations first, and then sort them. This ensures that the list of filtered conversations will only have their order affected by the sorting, not additions/removals of conversations from the list.

The filtering works by first accepting a String that is passed via user input. This filter String is converted to lowercase and trimmed. The filter String is then passed into Flutter's built-in "where" predicate test available on "Iterable" objects; in this case, the list of conversations. The "where" clause will convert each conversation's "title" field to lowercase and then compare it to the filter String. If the conversation's title and the filter String are equal, the conversation will pass the "where" predicate test and be added to the filtered conversations list.

The filtered list of conversations will then be passed through a sorting function. Flutter has built-in sorting functions that cover ConvoBuddy's sorting use cases by comparing each field of an object and creating a new list of objects based on those comparisons. In the case of ConvoBuddy, the *title*, *duration*, and *recordedDate* fields can be passed into Flutter's sort functions. It was decided that when sorting by title, the title should first be converted to lowercase so a true alphabetical sort could be

implemented. Otherwise, the same letter but different casing would be considered different in the sorting function and give the user an unwanted view of the conversations.

4.2.1.1. Possible issue and solution – Performance Decline over time

Filtering and sorting are almost instantaneous with a small number of conversations, but the performance can and will start to decrease as the number of conversations increases. Realistically, the users will not notice until they have 100+ conversations, but a solution is to filter and sort conversations in batches. The user will only be able to view a certain number of conversations at a time, so the batching could be executed as the user scrolls down the list instead of all at once when the screen loads.

4.2.2. Form Field Identification

There is a finite number of fillable field components, and based on those tags, we can identify what type of information is expected in those fields. Fillable form fields are identified by searching the DOM for “input”, “textarea”, and “select” tags. The name attribute is used as the ID, or name, of the field. The data type of information expected in a field is determined from the field's type attribute. All the field names and data type key/value pairs are stored in an array and added to the form fill request JSON payload.

4.2.3. Form Field Populating

BESie will send a JSON payload back to the browser extension containing the field name (same as the field’s name attribute) and the value to be filled into the field. Once the browser extension receives the payload, it will parse the JSON payload and convert it into a JavaScript object. The object is a key/value pair of the field names and field values. It will iterate through the key/value pair list and call the JavaScript method document.getElementByName for each key and set the value property of the field DOM object to the corresponding key/value pair’s value.

4.3. Tests

During this stage of testing, mainly functional testing was conducted. Functional testing was used to ensure:

- That all requirements have been met.
- That all application features operate as expected.
- To ensure that the user experience is easy and intuitive.

Functional testing is also helpful in ensuring that all possible interactions and scenarios were tested fully to detect inconsistent behavior and broken features, which will minimize the risk of encountering these issues during production.

23 test cases have been created and documented to test all aspects of the application such as recording a conversation, pausing a recording, the guided tour, and filling in a form via the web browser extension. All documented test cases have the following information:

- Test Number – Unique Identification number for the test case, used for tracking.
- Requirement ID – Number that refers to which requirement the test case applies to.
- Test Type – The type of testing done.
- Description - Describes which feature is being tested.
- Prerequisite - Any assumptions and/or conditions that need to be met in order to perform a specific test case.

- Steps - Describes in detail how the test cases are performed. This helps with being able to reproduce any defects found and helps the QA team understand and follow the specific testing procedure to obtain the expected output.
- Expected Output - Describes the expected results of the test case.

If any test cases had defects, the following information is captured:

- Defect ID – Unique Identification number for the defect.
- Defect Description – Describes the defect.
- Defect Date Found – Date defect was discovered.

Having properly documented test cases ensures that all functionality of the application is covered and that all members of the team understand why a test case was needed and what the objectives of the test cases are.

When testing begins, all test cases are written as cards on a Trello list called “In QA” and the cards were assigned to various members of the QA team. When the QA team member successfully completes testing, they are then required to move the card to a list called “Done”. If a defect was discovered, the QA team member is required to create a new card for the defect (which was linked to the test case) and move the card to the “In Dev” list to let the development team know that there was an issue that needed attention. The defects are also captured in a spreadsheet.

This is done to help the QA and development team collaborate with each other. By tagging the development team in a card, Trello will automatically notify users when an assigned card in a Trello board was moved so QA did not have to manually notify the development team of any issues.

4.4. Known Bugs

All known bugs identified throughout the development period will be documented within Trello. This will allow for the linkage of defect cards back to their associated store and requirement. Developers and testers can use this feature to easily reference the expected and actual behaviors of a known bug. Defect cards will be moved to the “QA” section when completed, like all other work items. This allows QA to consolidate their list of upcoming work into a single area.

5. Appendix

5.1. Code Files

All code files are stored in GitHub for source control. The URL is https://github.com/umgc/summer2023.

Contact the class professor for an invite to UMGC's GitHub projects. The professor will need your student email address and GitHub username.

5.2. Bug Tracker

All bugs are tracked in Trello for refinement and sprint planning. The URL for the sprint backlog is https://trello.com/b/VCgyrE8t/sprint-0-backlog. To gain access, contact the Trello administrator for AlphaSoft, which is currently Matt Bond (mbond5@student.umgc.edu).