Speech-to-Meme

Business Proposal

Prepared by: Team 7 - *ICBINGO*

Teja Dasari, Aaron Quashnock, June Schober, Caleb Short, & Jonah Uellenberg

Prepared for: Professor Morteza Chini

CSS 370 Section C

Date: June 8th, 2025

Making comedy gold accessible to all

Table of Contents

Executive Summary

Discord users tend to send memes in high quantities to fellow users, but rarely while playing video games. As Discord is designed for people who play video games, it is a platform for these users to communicate with like-minded people in their preferred ways. However, while Discord is a very capable platform in many ways, it is lacking in facilitating powerful and accessible rich text communication while its users are in-game; a stark contrast to the core app experience.

This lack of fast-paced in-game communication hampers the mutual enjoyment of games. Often, when humorous events happen in a game, the emotions experienced as a result are often felt by one or two players. When these happenings are explained using words, the full picture is largely absent: jokes simply cannot be explained by words alone. For instance, consider the following scenario:

> Discord user Jack Cortes is playing Team Fortress 2. After a brilliant play, he considers posting that one inside joke again to his friends on a Discord server. He comes up with a brand-new take of the meme and is excited to post it. However, the El Paso summer is too hot, hot enough that using Discord overlay for a silly joke could crash his computer. He doesn't want to leave his team down a person but really wants to post his joke with a golden idea before he forgets it.

Cortes, in this case, would have no choice but to forget sharing the experience with his friends. The best he could hope for would be to describe the situation later through spoken or written words; however, this is essentially taking the humor out of the moment and the joke.

We propose a voice command-based meme posting feature for Discord that can allow users to share memes rapidly and with minimal input. Using pre-cataloged images and voice commands, Discord could allow users to rapidly post memes without exiting their games. With already built-in voice filtering technologies, voice commands could be readily filtered out from background noise to be processed by a server, before selecting the correct image based on the voice command and posting it into a text channel. Nitro users could even create more extensive catalogs of images and utilize other perks, further enhancing the value of their subscriptions.

Detailed Project Proposal

The following explains what Speech-to-Meme will offer to Discord users as a product, while also discussing its viability as a business venture.

Statement of Need

Discord users tend to use the platform for playing video games and sharing memes, both in very social contexts. However, we have identified a potential unmet need for users: in-the-moment meme sharing while playing video games. Discord currently has no features that can create this functionality. As a result, its users generally lack the ability to express their in-game experiences to fellow server members in a way that captures the emotions involved. Memes that reflect what video game players feel during the act of playing may facilitate emotional communications between users playing and other users that are not in-game. This will encourage users within given servers to bond over game experiences that otherwise could not be expressed organically. Therefore, we propose our solution, Speech-to-Meme.

Speech-to-Meme Architectural Overview

Speech-to-Meme's architecture is unique as we are able to take advantage of many of Discord's existing infrastructure features. Consequently, the architecture does not have clear 1:1 pairing to typical architecture patterns. In Workshop #5, our group examined differing types of potential architecture. In consequence of that discussion we produced Figure 1.

Figure 1: Architectural overview of Speech-to-Meme

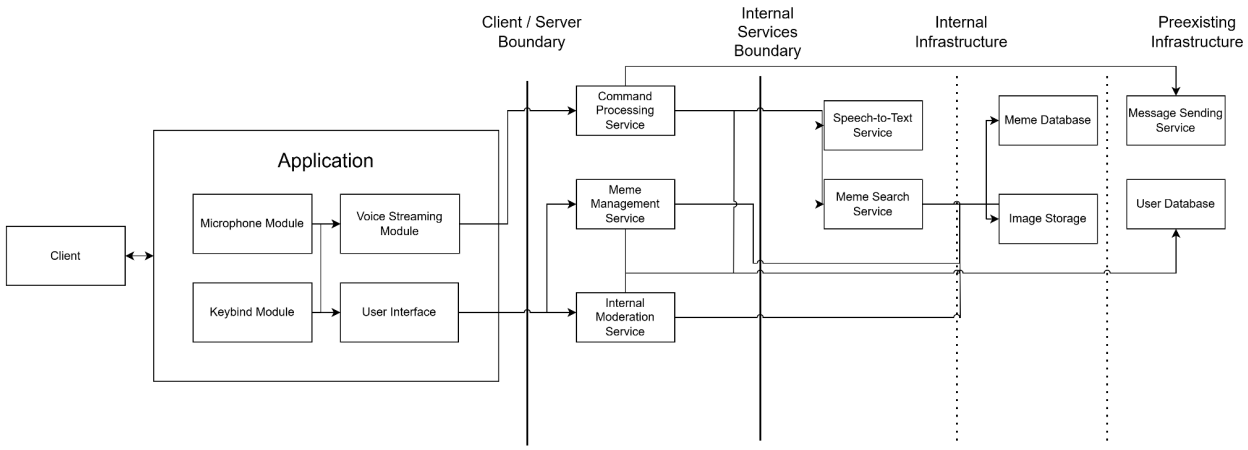


Figure 1 broadly illustrates Speech-to-Meme's architecture; we found Speech-to-Meme resembles best a Microservice architecture embedded within Discord's MVC-esque system.

Our feature is built on several self-contained services that utilize the abstraction already provided by Discord's infrastructure including:

- Voice commands
- Voice input filtering
- Custom media uploading
- Moderation tools

Project Plan

This subsection describes the broad implementation plan for Speech-to-Meme.

General Overview

When we identified key business risks related to Speech-to-Meme, detailed later in this document, we identified the need to *minimize* the cost of failure. Consequently, we plan to develop this feature in two phases: a Minimum Viable Product (MVP) prototype, and a significantly larger system wide integration. The feedback we gather by presenting the MVP shall guide the development and refinement of Speech-to-Meme's design.

MVP Plan

This subsection discusses the plan to implement a Minimum Viable Product (MVP) for Speech-to-Meme.

Overview

The Minimum Viable Product (MVP) for Speech-to-Meme should be quickly and cheaply implemented as a prototype. Fortunately, Discord's modular architecture and exposed API allows an independent team to design and build such a prototype. To build a Minimum Viable Product (MVP) for Speech-to-Meme, we (ICBINGO) will create a Discord bot that can optionally be incorporated into any Discord server. This bot will have the following functionalities:

- Store memes and corresponding voice commands to be posted by users.
- Allow users and server moderators to add memes and their voice commands to a meme storage database.
- Allow users to instruct the bot to enter the user's current voice channel and respond to voice commands.

- Use text-to-speech to listen to commands from the user to post a meme.

This MVP requires the following components be implemented:

- A database-storage component, which can be used to add and list memes from the database and retrieve a user's channel preference.
- A search component, which can be used to search for memes in the database given the meme name, the user's ID, and the server's ID.
- A speech-to-text component.
- A Discord text command component to provide the UI for adding memes and setting the meme channel.
- A voice command component, which can join channels, listen to voice commands, and respond to them.

From this, we shall commence additional customer research to determine final user reception to our proposal. We shall identify any points of issue that must be addressed before organizing a team to implement the feature for production.

Estimated Time

Our goal is to implement this MVP before the end of CSS 370. For this reason, we have provided a breakdown of the estimated time for each component to be implemented in Table 1.

Table 1: Estimated hours for each MVP component.

Component Name	Hours (est.)
Database-storage	2
Search	2
Speech-to-text	3
Discord command	2
Voice command	3

This brings us to a total of 11 hours to complete the project. Note that, as an MVP, this does not include many of the security or scalability requirements which have guided the rest of this

document. A production-grade implementation would require significantly more time to build, for which we discuss later in this document.

Proposed Tools

The following tools, libraries, frameworks, languages, and programs will be used to build the MVP:

- **TypeScript (JavaScript superset):** Fast language suitable for building robust applications with its type-safety. Use of this language should allow the MVP to be built the fastest compared to other languages. JavaScript, a superset of TypeScript, also includes many features for web-based and asynchronous programming, which are very useful in this type of software. JavaScript and TypeScript also include a rich ecosystem of libraries that increase reusability and decrease time-to-deployment.
- **NodeJS:** JavaScript runtime, needed to run, test, and deploy the MVP.
- **Discord.js:** The most popular library for building Discord bots in JavaScript. Discord.js includes all the Discord integration features we will require for this MVP, including Discord command handling, sending messages, and listening to voice channels [1].
- **ESLint and Prettier:** Tools used to maintain high-quality JavaScript/TypeScript codebases. Their use will make implementation of the MVP smoother, especially when developing with multiple contributors.
- **Git:** Maintains version history in case of mistakes and further simplifies multi-user collaboration.
- **GitHub:** Used to host our Git repository and simplify collaboration.
- **TypeScript-capable IDE (e.g., Intellij IDEA, VSCode)**: Quality of life features included in TypeScript-capable IDEs such as autocomplete and IntelliSense in Visual Studio can accelerate the development process by suggesting valid and safe fields based on existing types.
- **Vosk and vosk-koffi:** Speech-to-text library used to convert audio streams into text for searching, along with NodeJS bindings to connect it to the existing codebase.
- **Fuse.js:** Fuzzy-search library that can be used to approximate meme names from speech-to-text output when they don’t perfectly match up.

ICBINGO Internal Assignments

When the planning work for the MVP Assignment was complete, many team members were over encumbered by other classes at the time to meaningfully contribute to development tasks. As per our Team Manifesto, we recognized and balanced these constraints effectively to make sure we could make a high quality proposal. Throughout the early planning phases, we incorporated the MVP as a consideration for this proposal.

Jonah, as the primary proponent of the project, had experience with many of the technologies that would be used and ultimately contributed to the project the most. Teja also assisted with testing the feature and wrote Discord text commands that users would use to interact with the Discord bot. These contributions are recorded in Table 2.

Table 2: Team members assigned for MVP component implementation.

Component Name	Team Member
Database-storage	Jonah
Search	Jonah
Speech-to-text	Jonah
Discord command	Teja
Voice command	Jonah

While other members of the team were busy with other tasks during implementation time, their work on the planning and presentation of the MVP were still meaningful contributions towards its completion. As a result of the thorough documentation created by the team, implementing the feature as a proof of concept required fewer developer hours.

In addition, the MVP could be further expanded to match the overall proposed project better, such as allowing the Discord bot to join multiple channels in a server at a time and further refining the accuracy of text-to-speech. These expansions can be developed by team members during the summer quarter when there is more time to contribute towards the feature.

MVP System Architecture

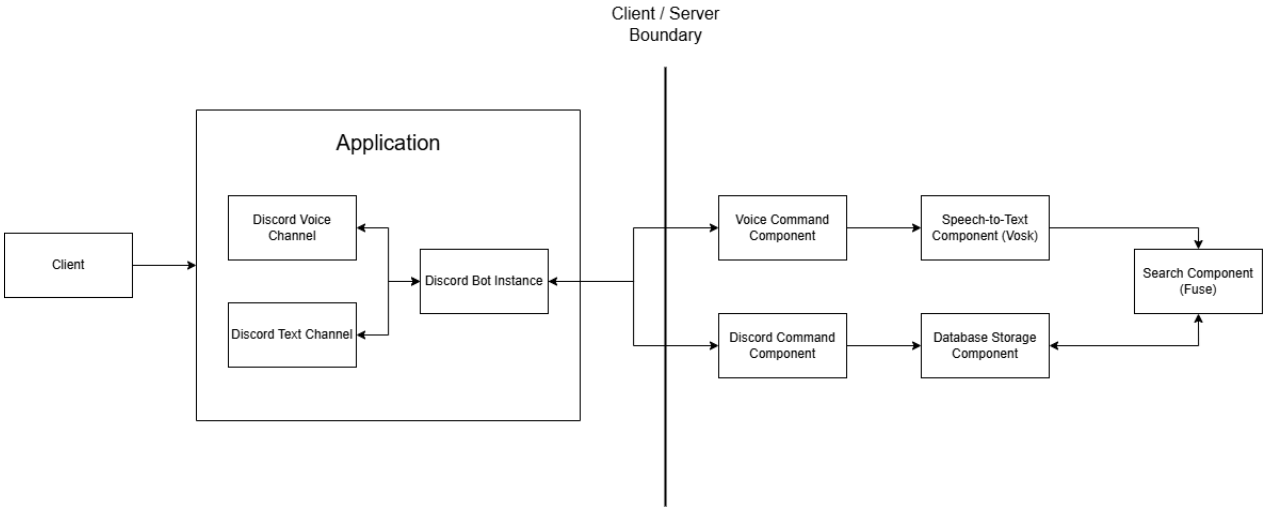
In the final MVP system architecture, the client interfaces with the Speech-to-Meme system through the Discord Bot Instance in both the Discord Text Channel the user sends commands in and the Discord Voice Channel that the user speaks meme names in. The Discord Bot Instance processes these interactions internally through its Discord Command Component and Voice Command Component respectively.

The Discord Command Component takes user commands from the Discord Text Channel and uses them to add and remove meme messages from the Database Storage Component. Whenever a meme message is added to the database, a corresponding entry containing the meme message's name is added to the Search Component's library for later use when querying the database for a meme to post.

The Voice Command Component pipes audio data to the Speech-to-Text Component that uses Vosk speech recognition to convert the audio data into a text format. The text data is then entered into a fuzzy search using Fuse.js that recognizes searches approximately equal to existing entries in the database. This process is used to resolve potential speech recognition errors deriving from low quality audio streams. The resulting keyword is then queried in the Database Storage Component for its corresponding image that is finally posted into the Discord Text Channel the Discord Bot Instance has joined.

The relationships between each component and interface in the Speech-to-Meme system are diagrammed in Figure 2.

Figure 2: MVP system architecture diagram



MVP Discord Bot Commands

The MVP uses various Discord Commands inputted into a server text channel to accomplish its various functionalities. Table 3 includes these commands along with their respective functionalities.

Table 3: MVP Discord bot commands and their functionalities.

Command	Functionality
/join	Has the Discord Bot Instance join the user's voice channel and listen for commands

/add-server-meme [name] [link]	Adds a meme message to the database with [name] as the meme keyword to be spoken in a voice command and [link] as a live link to the meme image or GIF.
/remove-server-meme [name]	Removes the meme message with a corresponding [name] from the database.
/list-server-memes	Lists all of the database's memes in the order in which they were added to the database. They are listed with their name, the live link used to access them, and finally the full meme image.

Production Plan

This subsection discusses how Speech-to-Meme should be implemented for production.

Overview

In consequence of the MVP, we will revise our production-grade artifacts according to user feedback. Then we will assemble a lightweight team for developing the feature for production, which would be best suited for this sort of project due to the integration of Speech-to-Meme's required microservices for production.

Team Organization

We need our team to be highly aware and in touch with their respective sector to ensure the complete experience is congruent with the rest of Discord's business. This can include business strategy, but also technical aspects such as server backend architecture.

A lightweight team would allow each member to retain as much contextual information as possible and communicate with their departments of the nature of these changes. The risks associated with lightweight teams outweigh the benefits, especially for implementation time, as it is essential that this feature is efficient, thoroughly accepted, and understood by each involved department. The team shall still have a product manager, being a lightweight team. So that the complete end-to-end picture is manifested by a tangible stakeholder.

Customer Feedback Process

Customer feedback, being vitally important to improving upon the MVP for the final product, will of course be collected and analyzed following the deployment of the MVP. The following will describe the three steps of collection, analysis and incorporation of user feedback into Speech-to-Meme's production.

The feedback collection process will include both anonymous and in-person testing and interviews. For anonymous feedback, selected users will be allowed to test prototypes of

Speech-to-Meme and fill out forms indicating their satisfaction and suggestions for future iterations. These forms will include self-reported scores for ease-of-use and general satisfaction and text-based response prompts. In-person testing will involve monitoring users as they interact with Discord in general and Speech-to-Meme in particular, recording actions taken and reactions experienced by users. Discord personnel will then interview these testers to derive further comments. It is important to note that users from a wide range of backgrounds, ages and other demographics should be encouraged to test Speech-to-Meme in order to gather the most far-reaching data.

In order to be useful, collected feedback must be analyzed. Firstly, as more iterations of Speech-to-Meme are built and tested, numerical data in the form of self-reported scores can be examined for trends in overall satisfaction and ease-of-use. To supplement this, written feedback can be categorized by positives and negatives, and further subcategorized by more and more specific details mentioned. These categories can then be weighed by frequency and impact before being compiled into reports suggesting action in future iterations. In-person feedback can be analyzed similarly by recording and comparing actions, reactions and comments. The sum of all feedback and its insights can be assembled into a single document, a “comprehensive feedback document” for discussion by the development team.

The final step in each iteration is to take action based on customer feedback. From the “comprehensive feedback document”, new Agile user stories can be created, or existing ones modified. If requirements have changed between iterations, this must also be reflected in the upcoming iteration. Once user stories and requirements have been updated, the development team can then update backlog items and begin work on the next prototype, before preparing for the next cycle of customer feedback.

Backlog

The details in this section and the MVP Plan section have been used to draft an initial project backlog in Table 3. Difficulty is rated on a Fibonacci scale, and large difficulty numbers mean highest difficulty. Priority is on a linear scale, and small priority numbers mean highest priority.

Table 3: Project backlog (MVP to final deployment)

Name	Description	Difficulty	Priority
Conduct Feature Surveys	Determine a list of wanted and unwanted features / requirements by conducting feature surveys. These will be used to inform future development.	2	1
Document MVP	Make and document design decisions for	5	1

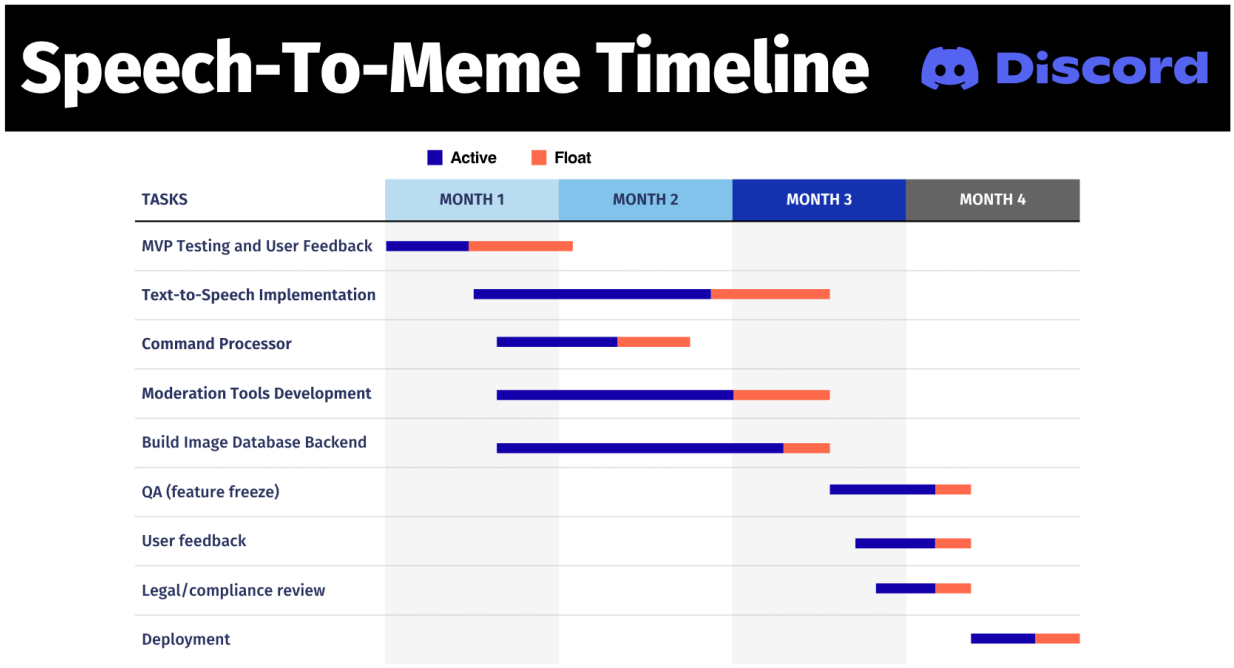
Design	the MVP (if they differ from those in this document), including any necessary research.		
Implement MVP Database Storage	Build MVP components necessary for saving/loading data, including meme data and user preferences.	3	2
Implement MVP Discord Commands	Build MVP components necessary for handling and responding to the MVP's Discord commands. Does not include full handling for the /join command.	3	2
Implement MVP Meme Search	Build MVP components necessary for searching for memes using a user, server, and keyword. This functionality is internal to the MVP and is used by other parts of it (i.e., voice commands), and must not be directly accessible by users.	5	3
Implement MVP Voice Commands	Encompasses handling receiving audio, speech-to-text, and subsequent meme search based on the user's query. This isn't broken down into smaller tasks because it's likely to be tightly coupled in the MVP (at least, it was in our implementation).	5	3
Deploy MVP and Gather User Feedback	Use the MVP to learn more about what works and what needs to be changed, and to gather data for the real product.	5	4
Design Final Product	Create a design for the production implementation of Speech-to-Meme, using data gathered by the MVP to inform its development. This design must be scalable such that it supports the number of users Discord expects to use the feature. Individual backlog items haven't been broken down for the final product because they depend heavily on the design which is chosen by the development team, which in turn is	8	5

	based on data gathered by the MVP — data which we do not have at this stage.		
Implement Final Product	Implement the design for the production version of Speech-to-Meme.	13	6
Begin Feedback Cycle	Deploy the feature (using a staged rollout / alpha test), gather feedback, use that feedback to make design changes and fix bugs, implement those design changes, then repeat.	8	7

Project Timeline & Estimated Time to Ship

Speech-to-Meme, being designed as a lightweight addition to Discord, should be implemented relatively rapidly. As the MVP is improved upon, additional considerations such as centralized data centers and legal concerns become more prominent for the main release. In total, we expect this release to take no less than three months, though this time could easily be increased. Therefore, we shall assume that Speech-to-Meme will require approximately four months to complete, following the development of the MVP.

Figure 3: Gantt chart of the timeline of Speech-to-Meme's development and deployment



The time to complete development of Speech-to-Meme begins with testing the MVP with Discord users, then collecting and analyzing feedback. After feedback is processed, the development team can begin work on text-to-speech implementation. This must be started before the other features of Speech-to-Meme as they are largely dependent on text-to-speech.

Once text-to-speech is underway in the first month, design and implementation of the other features can begin. Command processing, moderation tools and the image database backend can be implemented concurrently as they are loosely coupled microservices. It is anticipated that the database backend will require the most time, but all three of these primary features should be completed about halfway through month three.

During the rest of month three and into month four are Quality Assurance (QA), user testing and feedback, legal and compliance reviews and finally deployment. During this phase of the project, no new features will be added. A final round of feedback will be collected prior to deployment. By the end of the fourth month, Speech-to-Meme should be fully deployed and operational.

Project Justification

This section explains how Speech-to-Meme fulfills a niche within the Discord ecosystem and business strategy that is yet to be covered.

Justifying the Project

Speech-to-Meme provides users the ability to optimize their gaming experience through seamless voice-activated messages. Our feature provides a new way to build on the social, cultural, and intellectual capital emergent from our user base, from our Meme users to our Moderators.

We believe that Speech-to-Meme is a delighter we can implement at low cost to users, both on backend and also client resource requirements. Our customer research has indicated that users would use Speech-to-Meme in a number of new desirable ways (see “Scenarios”) that Discord has not yet provided. The smooth end-to-end experience can generate more interest into Discord, especially as the company is preparing to initiate an Initial Public Offering (IPO) within the next few years.

Feasibility

From a technical standpoint, this feature is essentially speech-to-text piped into a text search, with some integration to allow message sending and filtering memes based on server and sender. Significant effort has been placed into optimizing text search over the past decades, and Discord already supports it on a much larger scale than would be required for memes (Discord allows searching through all sent messages). Therefore, the main technical hurdle we may face would be with the speech-to-text.

As it stands, our design offloads speech-to-text onto Discord's own servers, rather than processing locally on users' devices. The reason why is because one of our target demographics represents users with less powerful computers, and introducing additional processing requirements may make the feature unusable for them. However, this decision will place strain on Discord's infrastructure and cost money, which may make scaling difficult. These issues are addressed under the "Viability" sub-section.

If we make targeting these users a priority, then it will not be feasible for us to process speech-to-text locally. Doing so online, however, is feasible, and there is existing precedent for it (for example, Google's voice search). As this is the only technical aspect whose feasibility may be in question, the project as a whole is feasible.

Desirability

We found that we have an opportunity to engage our users in a new way that would delight. During the course of our customer research we examined multiple groups of customers that would highly benefit from this new means of communication. In particular, Speech-to-Meme would be good at enabling accessibility to both our core audience and our audience with physical disabilities.

Viability

With all of the aforementioned solutions and benefits that Speech-to-Meme provides, we have an opportunity to provide even more for users that pay to use Discord, e.g., Discord Nitro subscribers. We propose several perks for current and future Nitro users: increased upload sizes, custom image frames and personal meme collections are just a few options. These perks and others will entice prospective subscribers of Discord Nitro and satisfy existing subscribers, ensuring that the feature is funded and it is financially viable.

Quantifying "cost" and "benefit"

Below, we discuss the quantifiable costs of the pre-existing alternative to Speech-to-Meme, and the measurable benefits that Speech-to-Meme offers, to highlight the impact it has on users.

During voice communication, users cannot easily send images as a response to a conversation without first switching out of their current application or diverting the focus on their screen. No matter the choice a user makes, this dilemma means that the final outcome is always negative: either they interrupt their current activity (which may be time-sensitive or inflexible towards interruption), or they fail to respond to the conversation in the way that most benefits them or is the most effective in scope of the conversation. Either one of these choices leads to frustration.

The benefit to the user can be seen in two aspects: the capital gained by responding to the conversation, and the capital gained by continuing their current activity uninterrupted. We can break this down into the following types of capital, as defined by Ethan Roland's 8 Forms of Capital [2], where the capital gained by responding to the conversation can be classified as social capital by improving social cohesion in the prompt response that is sent in the same context as the message being responded to and where the capital gained by continuing their current activity uninterrupted can be classified as intellectual capital or experiential capital, for being able to continue the workflow that you were in the middle of before attempting to send a message. A further breakdown of the tangible and intangible benefits gained through this feature can be seen in Table 4.

Table 4: Tangible versus Intangible Benefits

Tangible	Intangible
• Premium features for Discord Nitro users encourages existing Discord users to purchase a subscription, resulting in direct monetary profit. • Premium features for Boosted Servers encourage users to purchase additional tiers for their servers, resulting in direct monetary profit.	• Increases the convenience and usability for one of Discord's common use case scenarios. • Server dedicated meme message assets promote community engagement with users. • Innovation from the feature maintains Discord as a top choice in online communication. • Maintains consistency of updates and new features that uphold brand loyalty from users. • Voice-only features allows Discord to meet the accessibility needs of a wider audience. Publicity from the feature can attract new or returning users to Discord. The monetary profit from this can be measured through exposure to ads and partner integrations for new members. • Successful feature launches can increase positive user feedback and app adoption, resulting in better ratings for sponsor and partner negotiations

As for quantification, there are metrics for determining social cohesion and how swiftly responses can be sent out. Most projects must be justified based on an economic analysis and as such requires tangible benefits. For example, we can:

- Count of images sent over a set, defined period time (day, week, month, year, etc.).
- Count of time required to send a meme while operating without focus on the Discord application.
- Count the number of game crashes or losses due to users switching focus to Discord.

These metrics, and our estimates based on them, can be seen in Figure 4, Table 4, and Table 5.

Figure 4: Hassle map before and after Speech-to-Meme.

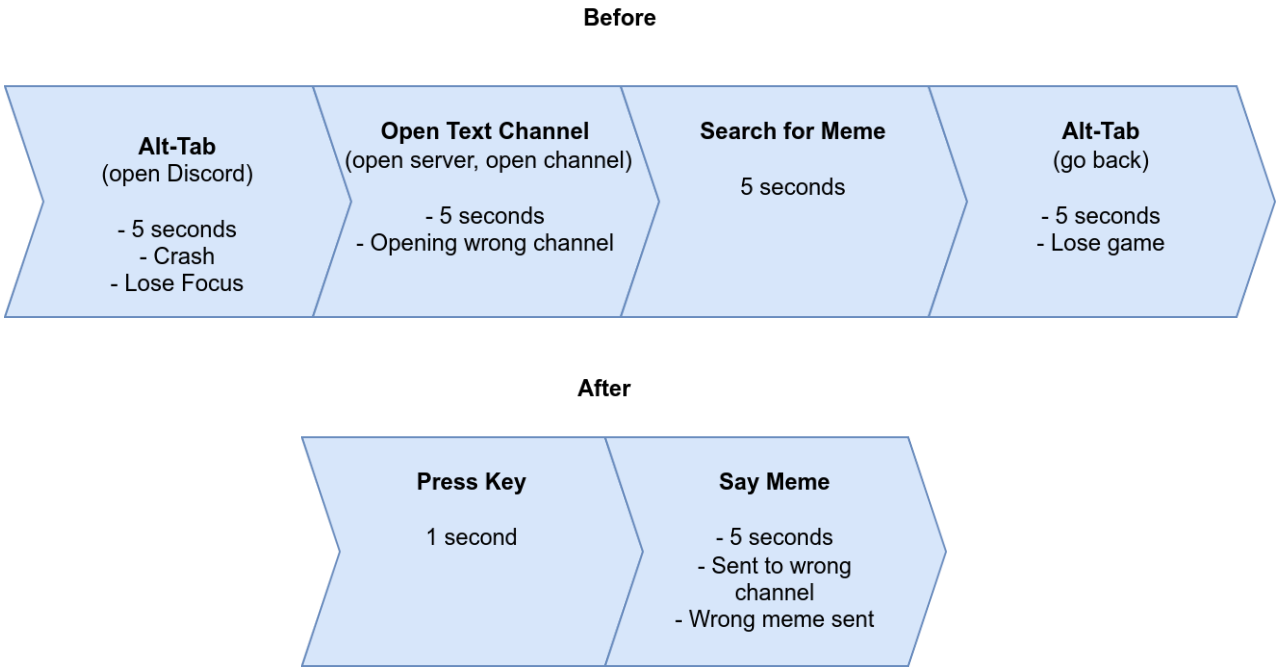


Figure 4 estimates the costs of posting images to Discord while playing a video game, both before and after the introduction of Speech-to-Meme. This demonstrates the amount of time savings users can expect for the same task by using Speech-to-Meme. It becomes readily apparent that using Speech-to-Meme costs a fraction of the normal time, approximately 6 seconds as opposed to 20 seconds, plus additional risks such as losing focus on a game application.

Table 5: Estimates of the cost of hassles to users, based on uncertainty in hassle cost.

Hassle	Expected Cost (time)	Expected Cost (crash / lose focus chance)	Low Cost (time)	Low Cost (crash / lose focus chance)	High Cost (time)	High Cost (crash / lose focus chance)
BEFORE						

Alt-Tab (open Discord)	5	0.05	3	0.005	8	0.1
Open Text Channel	5		0		10	
Search for Meme	5		3		10	
Alt-Tab (go back)	5		3		8	
SUM	20	0.05	9	0.005	36	0.1
10,000 Uses	200000	500	90000	50	360000	1000
AFTER						
Press Key	1		0.5		2	
Say Meme	5		3		8	
SUM	6	0	3.5	0	10	0
10,000 Uses	60000	0	35000	0	100000	0

Table 5 quantifies the costs of posting images while accounting for the potential of low-frequency but high-impact incidents such as losing focus on the video game application. It is further broken down into expected, low and high costs. Expected costs are the average expected costs of the given task, with low and high bounds. The table sums the total cost per 10000 uses. The expected cost per 10000 uses is 60000 seconds, with lower and upper bounds of 35000 and 100000 seconds respectively. This comes to an expected cost of six seconds per use, with a bounds of 3.5 seconds to 10 seconds.

Table 6: Estimates of the cost of hassles to users, based on uncertainty in the number of users.

	per meme	Impact (annual)			
Hassles		**low**	**estimate**	**high**	**actual (2025)**
# of yearly users (millions)		300	513	600	560
Alt-Tab (to switch to an active Discord application window) (sec.)	5	1500000000	2565000000	3000000000	2800000000

Crash / Lose Focus Chance (%)	0.05 ± 0.05	15 ± 15	25.65 ± 25.65	30 ± 30	28 ± 28
Open Text Channel (sec.)	5	1500000000	2565000000	3000000000	2800000000
Search for Memes (sec.)	5	1500000000	2565000000	3000000000	2800000000
Alt-Tab (to switch back to a previous window) (sec.)	5	1500000000	2565000000	3000000000	2800000000
SUM (sec.)	20	6000000000	10260000000	12000000000	11200000000
SUM (%)	0.05 ± 0.05	15 ± 15	25.65 ± 25.65	30 ± 30	28 ± 28
Benefits					
Press Key (sec.)	1	300,000,000	513,000,000	600,000,000	560,000,000
Say Meme (sec.)	5	1,500,000,000	2,565,000,000	3,000,000,000	2,800,000,000
SUM	6	1,800,000,000	3,078,000,000	3,600,000,000	3,360,000,000

Table 6 accounts for the uncertainty of the number of active users Discord has and calculates the overall impact accordingly. If we estimate that Discord has 300 million users who send one meme in a voice channel while Discord isn't focused, that causes 6,000,000,000 seconds, or 190 years, to be spent per year. If instead these users used Speech-to-Meme, they would in aggregate spend 1,800,000,000 seconds, or 57 years. This is 30% of the original time spent, and under this estimate, users save around 133 years per year using Speech-to-Meme.

Formal Economic Cost/Benefit Analysis

This section discusses the cost and benefit models we have built for the project, in order to analyze its profitability.

Costs Estimate

Our estimated development costs are based on the following estimates for implementation time. Table 7 discusses how many months each major Speech-to-Meme component is estimated to take in Work-Months. These estimates are based on the time spent on each

component in the MVP while taking into consideration additional time for components determined to have a high potential for security threats.

Table 7: Estimated implementation time for each project feature

Component Name	Work-Months (est.)
Speech-to-Text: recognizing voice commands from an audio stream	1
Image Storage: a way to store and retrieve saved meme images	2
Internal Moderation Service: the functionality needed to allow Discord's own moderators to find and remove unacceptable memes	2
Voice Streaming Module: the functionality needed to send and receive an audio stream	2
User Interface: modifications to the user-facing part of Discord's application	1
Meme Management Service: handles uploading memes with images and keywords	2

This ends up being 10 months of total work if completed sequentially. To determine the cost, we found the average cost of a software engineer in Discord's San Francisco and Amsterdam offices ($160,000 and $100,000 (converted from EUR[€] to USD[$]) respectively), found the average between them ($130,000), converted that to a monthly salary ($10,800 per month), then rounded it up for a more conservative estimate ($11,000 per month)[3][4].

From this, we estimate that 5 engineers can complete the work concurrently in 2 months. For safety, we've increased that estimate to 2.5 months in case of unforeseen issues during development. Combining the time, number of developers, and estimated salary, we expect development to cost $27,500.

After development, maintenance will likely require fewer developers. We estimate one full-time and one half-time (1.5) developers will be required, which is approximately $17,000 per month, using the salary estimate for software engineers. Given our time estimate, we expect to transition from development to maintenance after the second month, which is reflected in our cost estimates for the third month as an interpolation of the costs of the previous and next months.

Finally, marketing the feature will contribute a cost to the overall cash flow. In the past, Discord has marketed their features primarily using short animated teaser trailers that they share on their various platforms, including inside the Discord application itself. According to

an article by Advids, a business content creation service, well-crafted 2D animated graphics videos can achieve impactful results within a budget of $1,200 - $3000. Discord's past animated teaser videos, while short and created in-house, often involve complex animations [5]. Given these considerations, we have placed an estimate for the marketing cost of the feature at $2000.

Cash Flow Analysis

Table 8 estimates the cash flow of the development, deployment and maintenance of Speech-to-Meme over its first few months.

Table 8: Cash flow estimate by month

Category	Month 1	Month 2	Month 3	Month 4	Month 5
Revenue	0	0	100,000	100,000	100,000
Software Development	11,000	11,000	5,500	1,000	1,000
Maintenance	0	0	8,500	17,000	17,000
Marketing	0	0	2,000	0	0
Total Costs	11,000	11,000	16,000	18,000	18,000
Cash Flow	-11,000	-11,000	84,000	82,000	82,000
Cumulative Cash Flow	-11,000	-22,000	62,000	144,000	226,000

Return on Investment Analysis

Table 9 demonstrates the Return on Investment (ROI) of Speech-to-Meme over the first year of deployment and active use.

Table 9: ROI Over 1 Year

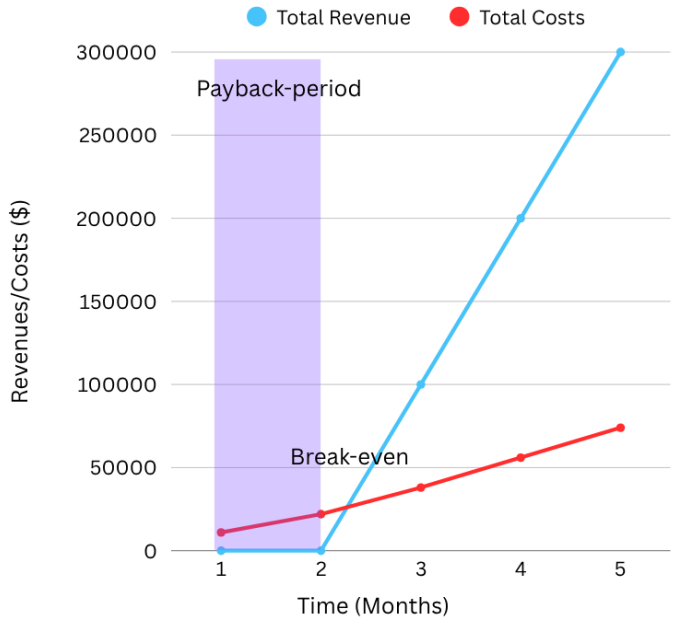
Quarter (3 Months)	Total Costs	Total Benefits	ROI
1	$38,000	$100,000	1.632
2	$92,000	$400,000	3.348

3	$146,000	$700,000	3.795
4	$200,000	$1,000,000	4.000

Payback Analysis

Using the data from the Cash-Flow Analysis in Table 8, we can determine the break-even point according to the graph in Figure 5. This point is an estimation of when total benefits outweigh total costs. It should be noted that our projections break even just after two months have passed with the feature fully implemented.

Figure 5: Payback analysis graph based on cash-flow analysis



Benefits Estimate

According to our customer research, we anticipate that following the deployment of Speech-to-Meme, a larger percentage of users will be encouraged to purchase Discord Nitro subscriptions due to previously described perks associated with the feature. This is due to the addition to Discord Nitro better fitting the needs and capitals of our identified user personas and their scenarios.

Discord currently has around one million Nitro subscribers. If our feature increases that number by 1%, then we gain an additional 10,000 Discord Nitro users. As each user is paying $10 per month, this works out to a revenue of $100,000 per month.

Justification

According to the above analyses, Speech-to-Meme can be a financially viable product within a fiscal quarter. While there is negative cash flow during the first two months, this is because development takes place with no Minimum Viable Product. After development, expected revenues quickly overtake cumulative costs and generate a positive net cash flow. Over a longer period of time, the Return-On-Investment makes it clear that Speech-to-Meme will remain viable, though the total ROI decreases over time. The analyses clearly demonstrate the financial potential of Speech-to-Meme as a venture for Discord.

Risk Management: Business

This section describes the business risks of publishing Speech-to-Meme (i.e., how deploying it might decrease Discord's competitive position) and how it can be mitigated.

Description of the Business Risk Paradigm

Business risks involve consequences that may arise from Speech-to-Meme being incorporated into Discord, resulting in decreases in users, profitability, and costs. These risks are disconnected from the technical implementation, so bugs and security issues aren't covered in this section.

Metrics of Risk

Here, we quantify risks informally by rating its severity in the following categories, on a scale of low, medium, and high: brand impact, customer retention, feature cost, and feature usage. This is summarized in Table 10.

Table 10: Description of each metric of risk to be evaluated

Risk Metric	Definitions/Key Questions
Brand impact	How much does this risk leak into Discord's *wider brand image* and cause it to deteriorate? How does it contribute to building a negative perception of Discord in users' minds?
Customer retention	How much does this risk lead to customers *leaving* Discord?
Cost	How much does this risk cause the cost of developing and operating Speech-to-Meme to *increase* (or, conversely, the profit gained from it to decrease)?

Usage Impact	How much does this risk make users *less likely* to use Speech-to-Meme?
Probability	How likely this risk could become true during the development and/or operation of Speech-to-Meme?

Discord's Risk Appetite

Discord's main concern is likely to be the brand impact of risks, followed second by customer retention. Risks causing excess costs or less-than-expected usage of the feature can be mitigated much more easily than those which damage Discord's reputation or cause users to leave. Discord likely won't tolerate risks with high brand or customer retention impacts.

Risk Analysis

This section includes an analysis of the different business risks associated with Speech-to-Meme that we have identified.

Risk Identification

We have identified business risks by examining areas where Speech-to-Meme can lead to dissatisfaction for the customer. This is generally where risks impacting customer retention or Discord's brand have been identified. In addition, we have focused on factors which may lead to excess usage of Speech-to-Meme, thus increasing costs. These findings have been summarized in Table 11.

Table 11: Speech-to-Meme Business Risks, Ranked by Severity

Feature	Description	Brand Impact	Usage Impact	Customer Retention	Cost Impact	Probability
Slow Client	The Discord client slows down or experiences crashing. This leads to a worse experience for the user, which is likely to outweigh Speech-to-Meme's benefits.	High	Medium	High	Low	*Medium*
Private Conversation Leaked	Speech-to-Meme activates and records a user's audio when they don't intend it. If this	Medium	High	High	Low	*High*

	audio or a transcript from it is made public, it can lead to privacy issues for the user, reducing trust in Discord. Accidental activation is very common with voice-activated software.					
Accidental Usage	Similar to Private Conversation Leaked, accidentally activating the feature can lead to more usage than projected, potentially making the feature unprofitable. This may also overload the servers responsible, leading to a worse feature.	Low	Medium	Low	High	*Medium*
Poorly Received Feature	Discord's users don't view Speech-to-Meme positively and it makes them view Discord negatively.	High	Low	Medium	Low	*Medium*
Unintentional Image Sent	Either a user sends an image they didn't mean to, or an existing image is changed to a different one. In either situation, this leads to embarrassment for the user, which may lead them to abandon the feature.	Medium	Low	Medium	Low	*Medium*

Risk Exposure

Risks of this nature will almost always be present in any possible implementation of Speech-to-Meme, and may not always have a clear path to mitigation. One of the most straightforward and broadly applicable ways to reduce Discord's exposure to these risks is through a staged rollout. By keeping the feature contained and limited by resources and number of users who can access it, there is a fixed ceiling placed on how much of an impact

any risks will elicit. In addition, a staged rollout allows us to gather insights without needing to affect Discord's entire user base. For some risks whose mitigations involve analytics (e.g., Accidental Usage), having access to this data will prove invaluable for making the feature ready for Discord's wider audience.

Risk Mitigation

This subsection describes how we intend to mitigate certain business risks with Speech-to-Meme.

Overview

Alongside a staged rollout, we have identified mitigations for each of the risks listed in Table 11, which provide ways to reduce the scope, probability, and impact of each risk in order to reduce its overall impact on Discord as a whole.

Slow Client Risk

Benchmarks should be taken before and after implementation of the feature, and on a wide variety of devices, to ensure that Discord's performance standards are maintained. Furthermore, performance must be a high priority and serve as a constant target for the development team during implementation.

This ensures that performance is a target metric throughout the entire development process, and that development goals are aligned with the customers'. It also requires that the feature can only be deployed if it meets performance targets.

Private Conversation Leak Risk

All uses of audio data and data derived from it must be tightly controlled, and reviewed for its potential exposure and the impact thereof. Any time this data is used in a way that exposes it (or data derived from it) publicly, it must be documented, justified, and reviewed.

This ensures that any areas of implementation which could lead to privacy are tightly controlled, reducing the risk of a mistake leading to data exposure.

Accidental Usage Risk

Usage of Speech-to-Meme should be recorded, alongside unsuccessful invocations. The ratio between successful and unsuccessful invocations must be carefully monitored. In an ideal world, this would be 0 (i.e., the feature will always be activated and used successfully), but there will be instances where users either accidentally activate the feature or use it incorrectly, leading to wasted resources. If this ratio becomes too high (e.g., 3 unsuccessful

attempts per successful one), then it should be investigated, and the feature modified to reduce it.

This allows Discord to carefully monitor for any outcomes that don't align with estimates, and swiftly take action to remedy them.

Poorly Received Feature Risk

Discord should send out interest surveys before implementation, both to guide the success of the feature and to ensure that it won't result in an overwhelmingly negative outcome for its users. These surveys should include questions measuring users' preference for Speech-to-Meme, and their preferences surrounding different sub-features of it (e.g., whether to have a configurable keybind for activating it). The results of these surveys should be used to determine how and what to implement, and even whether to cancel Speech-to-Meme entirely (if attitudes towards it are very bad).

After implementation is complete, Discord should undergo alpha testing so that it can ensure Speech-to-Meme actually works for customers, and to determine any changes that need to be made after this stage before it can be released to the public.

Sending out surveys ensures that Discord can gather preliminary data to guide the requirements of the feature, although this data may not necessarily align with reality, which is why alpha tests are critical as well. Users may give responses in the survey that don't align with reality, and which will be discovered during alpha testing.

Unintentional Image Sent Risk

This is one of the most difficult risks to mitigate, because many mitigation strategies will likely come at the expense of a good user experience. In almost all of Discord's features which involve user-generated content that can be sent by other users (e.g., emojis and stickers), users can see what they're sending; in fact, there's no way for users to send them without seeing a preview first. This is not the case with Speech-to-Meme, where the point of the feature is to allow users to send images without needing to interact with any visual interfaces.

That being said, Discord does have some precedent with similar issues. In their Soundboard feature, which allows users to play pre-recorded audio clips into a voice channel, the audio that will be played isn't immediately obvious to the user until they play it. There is an option to preview it, but unlike emojis and stickers, the user must go out of their way to do so. And in addition, this doesn't mitigate issues of the sound clip being swapped out at a later date. A user could preview the clip and find it acceptable, then if someone swaps out the clip for a different sound that the user might not find acceptable, the user will likely play it by accident because they were unaware it was changed.

So, it should be acceptable if Speech-to-Meme were implemented in a similar way as the Soundboard feature. We should have a way for users to view a list of memes that exist on the server, so that users know what each keyword maps to. The other major consideration is that there's no way to see what Soundboard clips users have played in the past, so the scope of damage caused by the user playing one by accident is more limited. This isn't the case with Speech-to-Meme, where images are sent as messages that exist until deleted. Therefore, it's critical that we provide a way for users to delete memes they have sent in the past.

Risk Management: Cybersecurity/Tech

This section describes the cybersecurity and technical risks of implementing Speech-to-Meme (i.e., what might go wrong during implementation) and how it can be mitigated.

Description of the Cybersecurity/Tech Paradigm

There are technical risks that have been identified across each component of Speech-to-Meme's architecture. Each of these risks are purely concerned with the inner workings of Speech-to-Meme from interactions to services.

Risk Analysis Approaches

To analyze technical risks, we have identified the DREAD methodology (Damage, Reproducibility, Explosibility, Affected Users, Discoverability) as the metrics to determine how to deal with each risk.

Table 12: DREAD Criteria Estimates for Identified Threats

Vulnerability	Damage	Reproducibility	Exploitability	Affected Users	Discoverability	Total	Decision
1. Tampering with the meme database	4	6	7	8	7	32	Mitigate
2. Moderator spoofing	3	8	7	6	9	33	Mitigate
3. Discord Admin spoofing	5	4	6	7	5	27	Mitigate

4. Spoofing as another user	4	8	3	2	7	24	Transfer
5. Accessing internal servers	10	1	3	10	2	26	Avoid
6. Overloading our systems	8	3	3	10	1	25	Mitigate/ Avoid
7. Changing meme images	5	5	3	0	1	10	Accept

Risk Identification Process

The method of identifying technical risks came from identifying trust boundaries in Data Flow Diagrams that we individually made in Sprint 8 and then combined our insights in our Workshop 7 document, and then used the STRIDE methodology (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privileges) to brainstorm and pinpoint possible exploitable points in our feature, the finding of which can be seen in Table 12.

Please see the section “System Description Artifacts” for Figure 31: Data Flow Diagram 1, which contains the following Trust Boundaries:

- **Standard User Trust**

  This trust boundary contains the entities, processes, and data flow concerned with the general Discord user, such as hitting the keybind and recording your audio command. It covers everything except additional manipulation of the user’s store of memes, such as deleting and recording voice commands, which is still okay.

- **Discord Admin Trust**

  This trust boundary contains the entities, processes, and data flow concerned with Discord Admins, such as checking the appropriateness of the image. It covers more cases than it should, as it encompasses the command conversion and sending; the only cases Discord Admins or employees at Discord would have is trust and safety checks of the user’s message contents.

- **Server Moderator Trust**

  Tools and data included in this trust are permitted to standard users with a Moderator status in a server to help them moderate and control the flow of data from users. Discord moderators in this trust boundary have authority over the server settings of

their server and can modify the server's store of assets along with user permissions regarding the feature in their server.

Please see the section "System Description Artifacts" for Figure 32: Data Flow Diagram 2, which contains the following Trust Boundaries:

- **Client/Server**

  This trust boundary designates the separation between processes that a user can access and manipulate on their client-side application through their interfaces with Discord, and the processes offered server-side, like processing commands, etc. It covers all use-cases within the scope of a general end user.

- **Internal Services**

  This boundary explains the relationship between the Discord server and its internal workings. Key data that moves through the boundary includes memes, voice commands and the moderation queue.

- **Internal Infrastructure**

  This trust boundary describes the connections between the speech-to-text implementation and the data that the feature accesses. Data that moves through this boundary includes memes and images in separate databases.

- **Pre-existing Infrastructure**

  The final boundary shows how the rest of the feature will interact with text messages. Data that is passed to Discord users as messages include messages that are generated with commands and arguments.

Risk Appetite

There is a consensus as to what risks would be tolerated and what risks would be addressed through development, whether that would be prior to delivery or after. As a part of Discord, Speech-to-Meme must not risk the quality of a large user population's experience casually using Discord or put personal information at risk; they will not be tolerated in any way.

It would be ideal that Speech-to-Meme could deliver that quality to individual users, however it should be recognized that no software system is completely without risks and that with enough effort, users may be targeted with malicious intent. The developing team can only reduce the probability of these risks by introducing additional security measures.

Risk Probability

To assess the probability of each identified risk, the reproducibility, exploitability, and discoverability, should be taken into account. Each score in those columns gives a generic scale of each risk's likelihood of occurrence. In prioritization, we would take the sums of the scores of these columns for each risk, resulting in Table 13.

Table 13: Modified DREAD Criteria Estimates for Identified Threats (Probability)

Vulnerability	Reproducibility	Exploitability	Discoverability	Total	Decision
Moderator spoofing	8	7	9	24	*Mitigate*
Tampering with the meme database	6	7	7	20	*Mitigate*
Spoofing as another user	8	3	7	18	*Transfer*
Discord Admin spoofing	4	6	5	15	*Mitigate*
Changing meme images	5	3	1	9	*Accept*
Overloading our systems	3	3	1	7	*Mitigate/ Avoid*
Accessing internal servers	1	3	2	6	*Avoid*

Risk Impact

To assess the impact of each identified risk, the damage potential and the possible affected users should be taken into account. Each score in those columns gives a generic scale of what the result would be if the risk occurs. In prioritization, we would take the sums of the scores of these columns for each risk, resulting in Table 14.

Table 14: Modified DREAD Criteria Estimates for Identified Threats (Impact)

Vulnerability	Damage	Affected Users	Total	Decision
Accessing internal servers	10	10	20	*Avoid*

Overloading our systems	8	10	18	*Mitigate/ Avoid*
Discord Admin spoofing	5	7	12	*Mitigate*
Tampering with the meme database	4	8	12	*Mitigate*
Moderator spoofing	3	6	9	*Mitigate*
Spoofing as another user	4	2	6	*Transfer*
Changing meme images	5	0	5	*Accept*

Going further, the impact of these risks can be further described in terms of effects on development schedule and costs, as well as feature quality, failure, and satisfaction:

1. **Tampering with the meme database** - directly modifying data in the meme database.
    a. *Development Schedule and Costs:* This could possibly incur a lot of damage were it to happen and were it to adjust a large amount of data.
    b. *Feature Quality, Failure, Satisfaction:* It would adjust user experience for what could be many different users, as their expectations when interacting with Speech-to-Meme would be very different.
2. **Moderator spoofing** - acting as moderators, adding memes that wouldn't be compliant with server policies:
    a. *Development Schedule and Costs:* As it wouldn't occur often and we cannot plan for that, we would only have basic security measures in place upon initial release and delivery.
    b. *Feature Quality, Failure, Satisfaction:* This would result in the unintended use of our feature for malicious actions to small communities, which would result in the failure of the feature, in that scope.
3. **Discord Admin spoofing** - acting as Discord Admin, adding memes that wouldn't be compliant with server policies:
    a. *Development Schedule and Costs*: As it wouldn't occur often and we cannot plan for that, we would only have basic security measures in place upon initial release and delivery.
    b. *Feature Quality, Failure, Satisfaction*: This would result in the unintended use of our feature for malicious actions to multiple small communities, which would result in the failure of the feature, in that scope.
4. **Spoofing as another user** - acting as another user, sending memes that weren't intended:

   a. *Development Schedule and Costs*: As it wouldn't occur often and we cannot plan for that, we would only have basic security measures in place upon initial release and delivery.
   b. *Feature Quality, Failure, Satisfaction*: This would result in the unintended use of our feature for malicious actions to a single small community, which would result in the failure of the feature, in that scope.
5. **Accessing internal servers** - being able to modify the front-end or back-end:
   a. Development Schedule and Costs: This would require security measures that validates only those on the development team and would be done before initial release.
   b. *Feature Quality, Failure, Satisfaction*: This may result in immediate feature failure being able to directly modify data on the feature, though it would most likely reduce the quality of the feature heavily.
6. **Overloading our systems** - not allowing users to send memes immediately do to reaching concurrency limits:
   a. *Development Schedule and Costs*: This would be something that needs to be scaled after release to what would be satisfying for the population of users who decide to use our feature.
   b. *Feature Quality, Failure, Satisfaction*: This would reduce satisfaction, as there may be a longer queue for each user's request to the server.
7. **Changing meme images** - through the front- or back-end:
   a. *Development Schedule and Costs*: As it wouldn't occur often and we cannot plan for that, we would only have basic security measures in place upon initial release and delivery.
   b. *Feature Quality, Failure, Satisfaction*: Would make user unintentionally send images they wouldn't know about, lowering satisfaction.

Risk Exposure

There will be strategies that we would implement for risks that were determined to be avoided, such as accessing internal servers and overloading our systems. However, those that have determined that will be mitigated, transferred or accepted, that will carry into further development of Speech-to-Meme.

Risk Mitigation

The decisions were made to mitigate certain risks identified such as tampering with the meme database, moderator and Discord Admin spoofing, and overloading our systems.

To mitigate the risk of tampering with the meme database, we could implement encryption methods, strong access controls, as well as create "append-only" databases.

To mitigate the risk of moderator and Discord Admin spoofing, we could make it so that the app might only be added to servers where moderators have two-factor authentication enabled, and just generally add extra verification for special permission Discord accounts.

To mitigate the risk of overloading our systems, we would consider rate and concurrency limiting so that requests to our database could be adequately managed.

Alternatives Considered

The following will describe considered and rejected ideas, and rationales for why they were not pursued. These descriptions shall illustrate our due diligence in investigating multiple potential design and implementation avenues.

Architecture Pattern

The primary decision that was initially disagreed upon for Speech-to-Meme's design was its architectural pattern. Proposed patterns included layered and microservices. Each had their own potential benefits, but it was of the utmost importance that the most ideal pattern was chosen. Below is a summary of the team's reasons for each proposed pattern.

A layered architecture was one of the patterns that was proposed for Speech-to-Meme's architecture. This pattern features several distinct layers that each handle their own designated responsibilities, and can only communicate with adjacent layers. The primary rationale for using this pattern would be clear communication between the client, voice parsing modules, and meme databases.

The second pattern that was discussed was the microservices pattern. This pattern is structured as a collection of small services that each serve a specific function. In particular, these services are only loosely coupled for independent development and deployment. Key benefits of this pattern included modularity between clients, servers and services, separation of implementation concerns and functional reliability.

We decided upon using the microservices pattern because the MVP could then rely upon existing services for rapid and effective implementation. It was also agreed that modularity was highly desirable to improve both implementation and user experience.

Keybinds

The second major discussion our group had was whether or not to use keybinds. This was an important point of clarification: this would affect the complexity of operation for users and

the level of control they would have. This discussion happened during the creation of our architecture. Since some team members assumed the presence of keybinds, and others the absence thereof, discrepancies occurred between proposed architecture models. Below are rationales for and against each possibility.

If keybinds were to be used, the primary considerations were how mobile users would interact with Speech-to-Meme, and how "hands-on" we wished the feature to be. Another consideration was ease of implementation; if the Discord client could simply wait for a designated key to be pressed, then filtering out irrelevant voice input would be much simpler. However, requiring additional input would needlessly complicate the relationship between user and service. Another consideration we realized was that mobile users would not have a clear way to activate the feature without a significant architecture overhaul.

On the other hand, if keybinds were not to be used, a key factor would be tracking voice commands. The main benefit of this decision would be a streamlined user experience due to minimizing the input required. However, without keybinds, the only way users could interact with the feature would be constantly monitoring voice input, which could be expensive from a computational or financial perspective, or both.

Ultimately, we decided that reducing the amount of "hand-on" input for the feature was more desirable to the user. We determined that it was more valuable to create a smooth experience for users than saving computational power.

Speech-to-Text Paradigm

A third key design decision for Speech-to-Meme was how to handle speech-to-text for command parsing. One option was supporting it client-side, while the other was outsourcing the logic to server-side. Each of these decisions were implied by different team members when outlining potential architectures, which necessitated a new discussion.

Hosting speech-to-text on the Discord client would provide its own advantages and disadvantages. One benefit of this decision would be rapid processing of voice input, with zero latency between client and server. This could in theory improve the responsiveness of the feature. A potential issue with this paradigm, however, would be client overhead. Each client would have to install and execute its own instance of speech-to-text, which would involve significant overhead and potentially harm overall performance of the Discord client.

Server-side processing of voice input would present its own unique characteristics. A possible benefit of this paradigm would be reduced client overhead. This would be particularly useful in older and less powerful devices operated by users, since the Discord client would retain a lower performance profile. A potential drawback would necessarily be increased server-side

processing. This could require Discord to acquire additional or more powerful servers to handle the increased data traffic.

Eventually, the team decided to use a server-side paradigm in the architecture. It was agreed upon that it would be more desirable to maintain a lower performance requirement for users on the client side to preserve Discord's extensive accessibility. Additionally, since data for Speech-to-Meme must be passed between clients and servers anyways this decision simplifies implementation further.

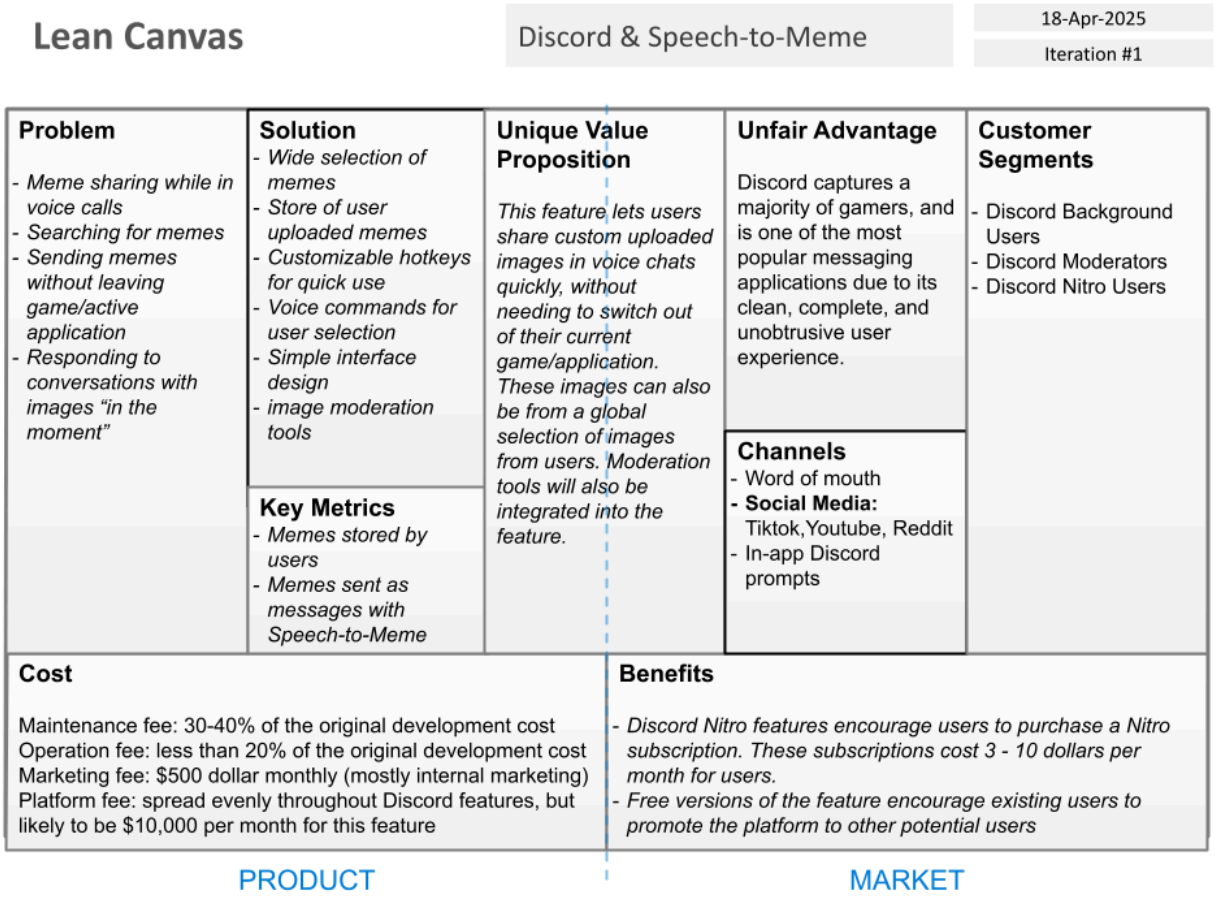
System Description Artifacts

This section contains a non-exhaustive collection of key system/business description artifacts produced for this proposal.

Lean Canvas

A lean canvas is used as a mechanism to iterate, evolve, and grow a business model by identifying and making a sole definition for aspects of our product's design. It also answers whether or not to pivot from an initial product concept, customer segment, etc.

Figure 6: Lean Canvas for Speech-to-Meme

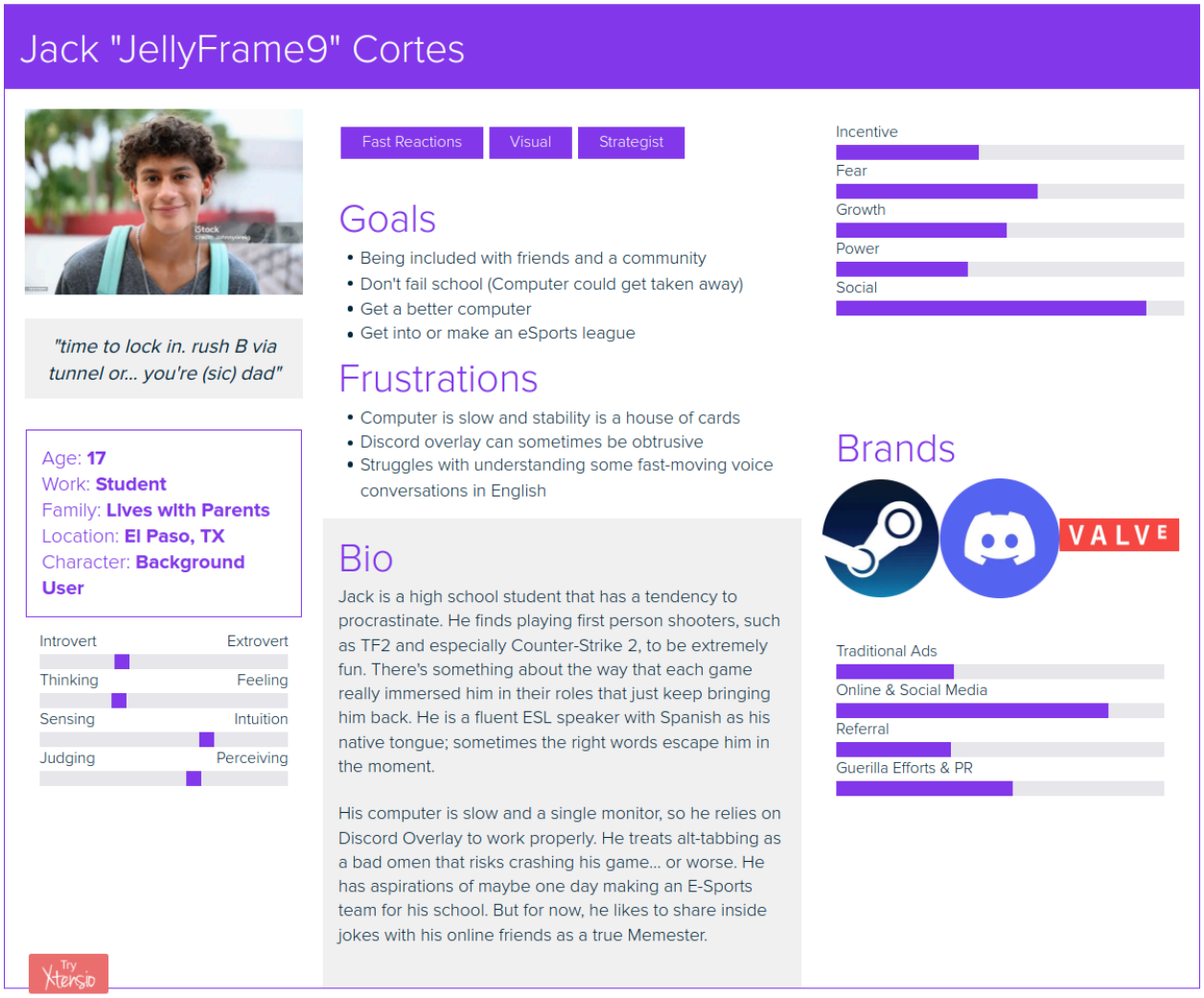


We created Figure 6 during our second sprint while conceptualizing our product idea. Initially, we were given the choice of developing a feature for either Discord, an IDE (Integrated Development Environment) of our choice, or a popular social media platform, and to brainstorm ideas using the lateral thinking methodology. From there, we chose the single best one and proceeded to define it further, filling in the sections of the Lean Canvas.

Customer Personas

Customer personas are meant to represent the customer segments that we had identified in our Lean Canvas by personifying them. This allows us, the developers of our feature to go further and identify how to prioritize customer segment needs, what customer pain points are worth solving, etc.

Figure 8: Jack Cortes Persona

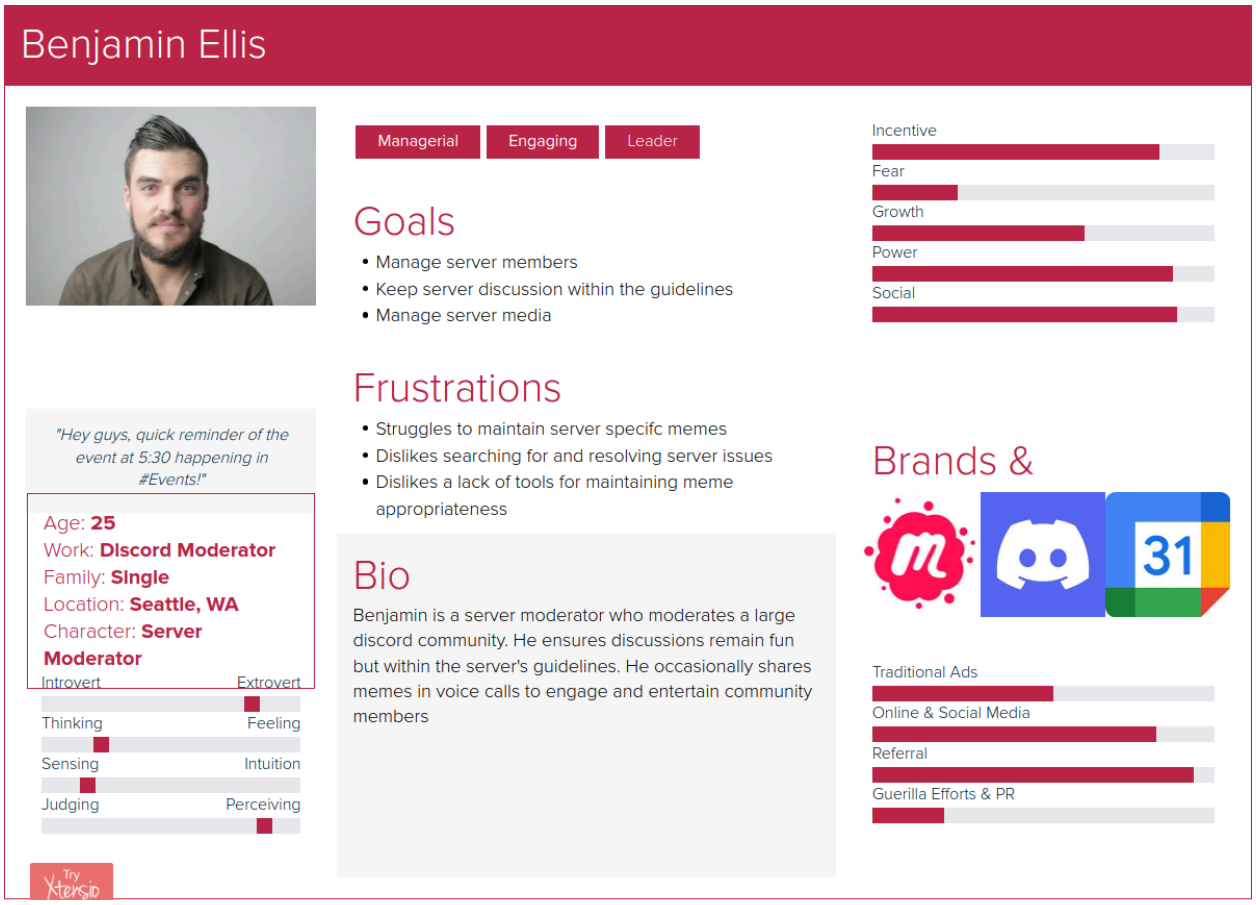


A member of our team, June, created the customer persona contained in Figure 6, which we later designated as the best one to represent our generic customer, our MPP (Most Power Persona); it focused on the Background User customer segment, focuses on performance, keeps feature grounded in being performant, and is aligned with Discord's goals of being a fast, convenient messaging app.

Figure 7: John Bogner Persona

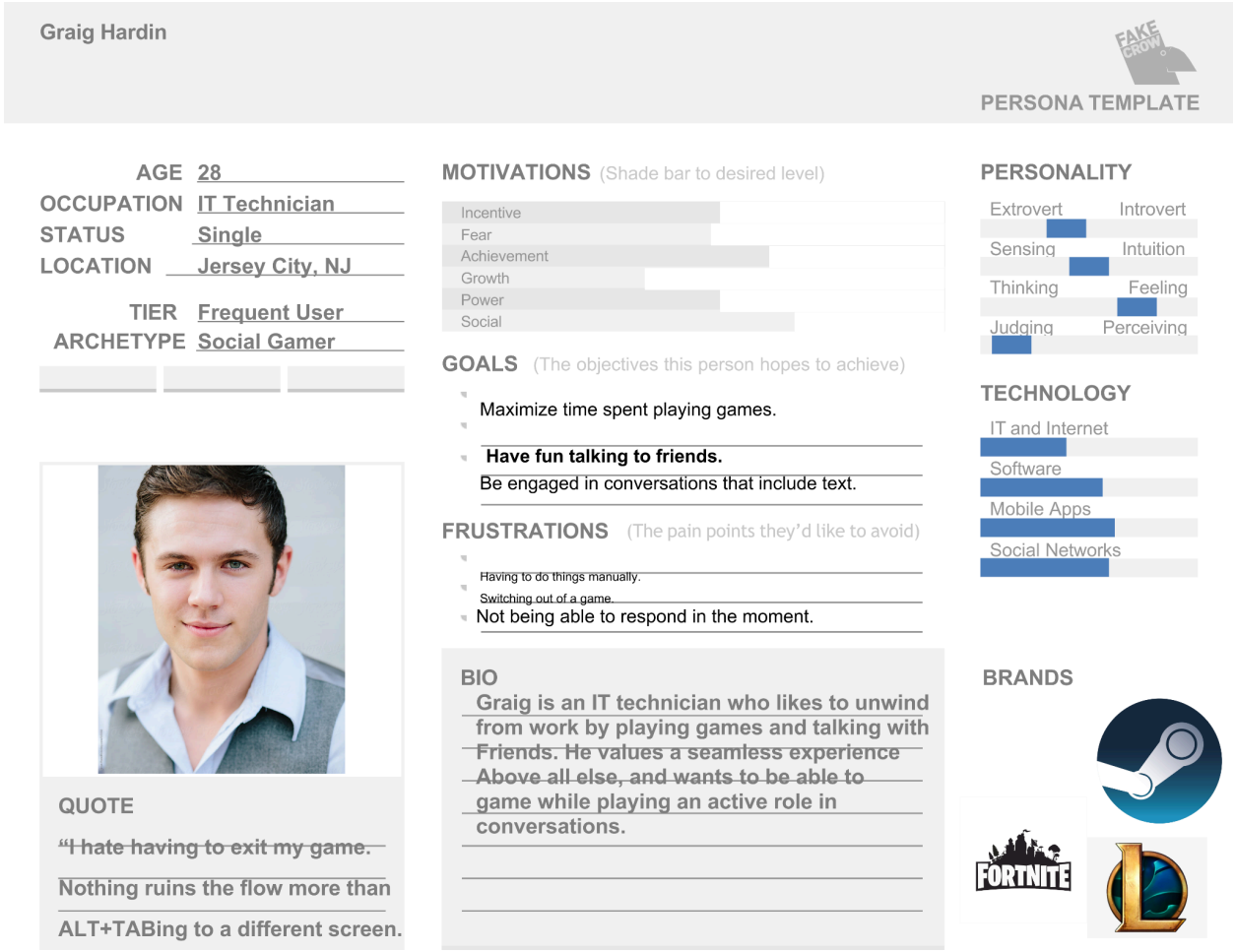
A member of our team, Caleb, created the customer persona contained in Figure 7 based on the Community Manager customer segment we had identified in our Lean Canvas. It was a candidate for MPP due to representing a prominent role as a manager of a large community and had a focus on community engagement.

Figure 8: Benjamin Ellis Persona



A member of our team, Aaron, created the customer persona contained in Figure 8 based on the Moderator customer segment we had identified in our Lean Canvas. It was a candidate for MPP due to focusing on control of the feature and may provide insight to necessary features.

Figure 9: Graig Hardin Persona



A member of our team, Jonah, created this the customer persona contained in Figure 9 on the Background User customer segment we had identified in our Lean Canvas. It was a candidate for MPP due to its strong alignment with Background User archetype.

Figure 10: Alex Mitchell

A member of our team, Teja, created the customer persona contained in Figure 10 based on the Community Manager customer segment we had identified in our Lean Canvas. It was a candidate for MPP due to its focus on accessibility due to the conceptualized user being disabled.

SPICIER Customer Scenarios

This subsection covers the SPICIER customer scenarios that guided Speech-to-Meme's creation.

Overview

SPICIER Customer Scenarios, sourced from *Scenario-Focused Engineering*, are an extended analysis of use case scenarios that measure them against metrics such as whether it tells a story or includes personal details.

Customer scenarios describe potential interactions between a potential user and a product, outlining the intended customer flow of experiences and specific circumstances that contextualize the interaction. Each customer scenario gets "filled out" as the use case scenario is completed, as detailed later in this document.

Scenario 1 - Caleb Short

This Customer Scenario describes the sending memes use case, one of our core use cases. This scenario in particular describes the outcome of the use of the feature that could be rationalized into one of the capitals

Version 2 Marco, a 15-year-old Discord user, finds himself playing a fast-paced video game with his friends in Discord after school. He has been playing this game with his friends for a while and has frequently encountered funny occurrences while playing but can't fully convey the humor of his situation to his friends through simply talking or typing. Explaining funny happenings with words just feels insufficient, unsatisfying and unfunny to his friends. One night while playing a round with his friends in a voice call, Marco experienced a hilarious event within the game. Bewildered and excited, he wants to send a meme to convey the absurdity of what just happened. However, the game being fast-paced means tabbing over to Discord and searching for an adequate meme could mean losing the round for his team. **<magic happens>** Using Discord's feature Speech-to-Meme, he posts a meme perfectly fitting the situation to the games channel in his friend's Discord server, all while staying focused on the game and bringing his team to victory. Various members of Marco's Discord server reacted humorously, both in the text channel the meme was posted in and on the voice channel Marco spoke in. Satisfied with his victory and his humorous meme that entertained all his friends, he logged out of the game and Discord for the night.
Team Member's name: Caleb Short
S (tells a story): The scenario follows a storyline that has a beginning, middle and end.
P (includes personal detail): 15-year old, plays video games, often encounters funny moments while gaming.
I (is implementation-free): The scenario mentions the feature, but does not strictly mention how it is implemented.

C (customer's story not the product's story): The scenario has a heavy focus on the user's experience, and only briefly mentions the experience as a mechanism to make that experience complete.
I (reveals deep insight about customer needs): Needs further elaboration
E (includes emotions and environment): Lots of mentions of humor and the implication of satisfaction in sharing common feelings, but no mention of the concrete emotional benefit to the user. Accurately captures the environment as the user experiences it.
R (based on research): It's loosely based on Caleb's personal experience.

Scenario 2 - June Schober

This Customer Scenario describes the sending memes use case, one of our core use cases. In particular, this Customer Scenario includes many specific details of the emotions and environment of the main actor.

Version 1

Jack is playing Team Fortress 2 on the map "pl_upward" on a PUG (pick-up game) server. This server is currently in Highlander mode, which means that there is one team member for each of the nine classes the game offers. Playing as the Spy on the RED team, he just managed to pull 4 chain stabs on BLU after correctly determining a tiny opening on the opposing team's onslaught. A feat that is incredibly difficult to pull off reliably, but for Jack it was Tuesday afternoon.

Of course, as with the cycle of the game, his luck does run out and his character gets shot to death. As he waits to respond, he considers posting that one inside joke again. There is a specific channel (#spy-moment) on a small server where Jack's friends post the same screencap of a Red Spy Ragdoll in a rather peculiar position. The image is usually captioned as "SPY JUST ___ MY ____ 💯💯🤯🤯😱🗿", where the spaces are usually ad-libbed with something random or funny that each person in the server thinks of.

He comes up with a brand-new take of the meme and is excited to post it. However, the El Paso summer is hot, too hot. The AC is struggling to keep up with the weather, and his computer fans are whirring loudly. The computer begins to drop the occasional frame as thermal throttling turns down the computer's power. Cortes is aware that even using just the Discord overlay now for a silly joke could very well crash the game. He doesn't want to leave

his team down a person or risk getting banned for disconnecting (via crashing) but really wants to post his silly joke with a golden idea before he forgets it. **<magic happens>** Using Discord's new Speech-to-Meme Feature, Jack is able to post the inside joke without having to switch out of the game, or even type anything! He's able to attach an equivalent voice clip of him saying the caption, creatively ad-libing out the emoji in a noticeable Hispanic accent: "SPY JUST CHUCKLEDDUNKED MY CHUCKLENUTS! NO MANCHES! Bruh" This gets multiple Discord reactions from the other server members, to which Jack finds out after finishing the game.
Team Member's name: June Schober
S (tells a story): Very elaborate and detailed. The story is told as a narrative that can stand on its own as something separate from the product itself, and includes many details to help the reader empathize with the reader.
P (includes personal detail): Includes many personal details about the actor's motivation and emotions with each event.
I (is implementation-free): There is an explicit "<magic happens>" part, to make distinct the details from the actual use case.
C (customer's story not the product's story): Very little mention of the product, other than introducing it. The actual invocation and usage of the product is not mentioned in the scenario, with the main focus being placed on what the user gets out of using it. However, it would be good to put some focus on the product's story, as using the product is part of the user experience and that experience isn't discussed.
I (reveals deep insight about customer needs): Yes. It addresses the fear of Discord Overlay shutting down his game, and of the product supporting different accents (or even languages).
E (includes emotions and environment): Yes; fear of getting banned or disconnected; excitement in sharing the meme. Notes weather in El Paso
R (based on research): Research was based on a current persona, but discussion with someone closer to the persona's demographic could be needed.

Scenario 3 - Teja Dasari

This Customer Scenario describes the setup of the Speech-to-Meme feature as a use case. It describes an actor's use of the setup process with uploading images and configuring voice recognition.

Version 1 Sitr is a Background User, going to school and at home, studying and staying on top of his messages, and he's just found out about our product. He is downloading our product for the first time and eager to try it out, see if it's any use. Sitr's problem is that he must setup our product for the first time, the problem isn't very complex, but it is a prerequisite to use our product. Sitr may be keeping track of schoolwork and any other external obligations, but the problem is simple enough to be done in a few minutes. **<magic happens>** Sitr has accomplished a part of the setup process towards replying easier and faster, and to measure the success of that outcome is to see whether or not the he has sent their first meme, since we will be keeping track of user who had our product and are using that product. The solution needs to be able to help in the setup process for Sitr to be delighted, and they would still mostly be neutral about it, with the satisfaction that they had finished their setup process.
Team Member's name: Teja Dasari
S (tells a story): The scenario is written as a narrative, but would benefit by having more details and mentioning some of the actions the user will do.
P (includes personal detail): Rather limited; mentions home and school but these are very general. Not too connected to the user themselves.
I (is implementation-free): "keeping track of user who had our product and are using that product. The solution needs to be able to help in the setup process"
C (customer's story not the product's story): discusses the product almost as much as the customer themselves. The customer's story arc could be made more clear.
I (reveals deep insight about customer needs): doesn't really explain why the customer wants to use the product and focuses on setup

E (includes emotions and environment): Their emotions and environment can be derived, but they could be made clearer in-text.
R (based on research): Based on personal experience.

Scenario 4 - Aaron Quashnock

This Customer Scenario describes a moderation workflow use case. In particular, it describes a case where a server moderator wants to use their server's tracing tools to track down a user who uploaded an image to take action.

Version 2

Elliot is the owner of a Discord server for a school badminton club that they are in. One of the chats in the server is dedicated to announcing when the next meeting is going to be, and another is for socializing with other members of the badminton club.

Elliot and their friends in the club like to share memes and images relating to badminton and their latest games together. In one of the latest meetings, a student member slipped when trying to hit a fly birdie, causing them to lose the round. The student was not injured and had fun playing for the rest of the meeting. However, when making the announcement for the next badminton meeting, Elliot noticed that one of their friends in the server had created a meme making fun of the student that slipped, and many of the server members are sharing it in the socializing chat. It was apparently dubbed, “dude’s failing”, and students often mention it. Elliot wants everyone in the server to stop posting the image and to confront the image’s creator. However, given their history with the friend, Elliot knows that they likely didn’t mean to offend the student who slipped and so would not like to kick them out of the badminton club or server.

<magic happens>

Elliot was able to simply get rid of the uses of the newly created meme as well as prevent other students from posting it. He also prevented their friend from sending the image and timed him out quickly to avoid escalating the situation and causing further impact as well as DM them about it, and all of this was done in no time flat. Elliot is relieved that it didn’t

spread very far, the friend who initially posted it was very quick to apologize, and the student was able to comfortably stay in their club.
Team Member's name: Aaron Quashnock
S (tells a story): Tells a complete story about the customer (Elliot).
P (includes personal detail): Explores Elliot's experience in the Badminton game, goes in-depth on the personal interaction between the two.
I (is implementation-free): Speech-to-Meme was not directly mentioned.
C (customer's story not the product's story): Discusses the customer story, with an ending and where the feature is used and involved.
I (reveals deep insight about customer needs): Discusses the user's end goal, which is reducing the spread of a certain image and disciplining the user who created it. The story does not entirely reveal insight on how Speech-to-Meme would mold around these needs.
E (includes emotions and environment): Emotions are largely absent, though there are some mentions. Explains the badminton game, the players within it, and the Discord server they use to communicate as the environment.
R (based on research): Based on a persona generated by the group.

Scenario 5 - Jonah Uellenberg

This Customer Scenario describes the sending memes use case, one of our core use cases. This scenario in particular describes a motivation for the main actor, Jeff, wanting to send memes without interrupting their game.

Version 1

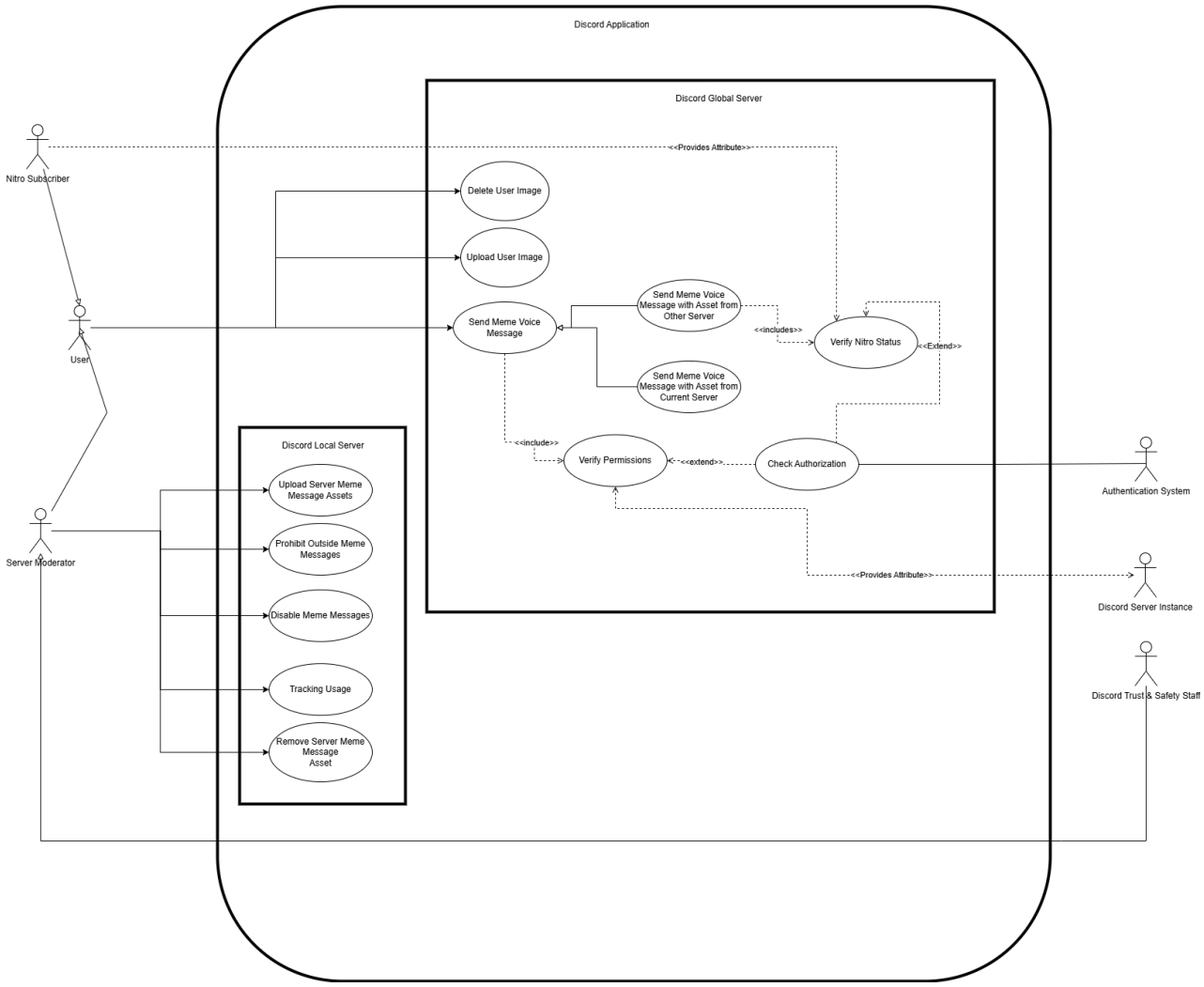
Jeff is an avid gamer who likes to spend time with his friends in voice calls while playing video games. Oftentimes during conversations, Jeff wants to send an image or meme in response to something one of his friends sent, but he can't easily navigate to Discord to do so. Pausing the game to open Discord might cause him to lose his match or break the flow of his game, which can be frustrating, and if he waits until he's ready, then the conversation will probably have already moved on and sending it won't make sense anymore, losing him the satisfaction of having his friends respond to his image.

With the Speech-to-Meme feature, Jeff says the name of his intended image out loud, and the feature sends it to his friends, who can see and respond to it as a part of the conversation. Because he never has to open Discord, Jeff can continue his game uninterrupted, while still being able to engage in conversations that are deeper than just voice.
Team Member's name: Jonah Uellenberg
S (tells a story): The story seems quite generalized.
P (includes personal detail): Next-to-no identifying characteristics that makes Jeff unique, but has enough to contextualize the scenario.
I (is implementation-free): Saying the name of the meme could be considered implementation; a good substitute would be "...is able to get their meme posted with only using their voice".
C (customer's story not the product's story): The customer was the primary focus of the scenario
I (reveals deep insight about customer needs): Jeff's need for the feature is directly outlined and fulfilled.
E (includes emotions and environment): Jeff's emotions and environment are established in key moments.
R (based on research): Based on personal experience.

Use Case Diagram

Use Case Diagrams are high-level UML models of the usage requirements for a system, showing system scope and providing a high-level description of expected functionality, by visually showing how users and external systems interact with the system and what those interactions are.

Figure 11: Speech-to-Meme Use Case Diagram - Version 1



We created this Use Case Diagram in Figure 11 as a combination of individual Use Case Diagrams during Workshop 3, and encompasses any unique use case that other members of our team have not thought up during brainstorming.

Use Case Scenarios

This subsection covers the Use Case Scenarios created for Speech-to-Meme's development

Overview

Use Case scenarios describe *specific* interactions between a potential user and a product, outlining the intended, step-by-step, customer actions and system reactions. These are used to delineate specific scenarios to be implemented by a systematic approach. They are not to be confused with customer scenarios, as these begin to involve implementation unlike their counterpart.

Use Case Scenario #1 - Caleb Short

This subsection describes the Use Case Scenario created by Caleb Short, implementing Customer Scenario #1.

Scenario Table - v1.1

Table 15 implements Customer Scenario #1 by defining how messages should be posted. This specifically defines how the feature will allow users to post memes in text chats quickly and without interruption.

Table 15: Scenario table describing posting meme a Meme Message as a Use Case Scenario

Use-case:	Posting a Meme in Text Chat
Primary actor:	Generic User
Goal in context:	Post a meme in text chat with minimal effort
Preconditions:	1. The server they are using the feature on has at least one voice channel and one text channel 2. The user has an unmuted microphone
Trigger:	1. User decides that they want to post a meme into a specific text channel
Scenario:	1. User enters a Discord server 2. User enters a voice channel within the server 3. User says the keyword(s) associated with the meme of their choice 4. The Discord server checks the keywords and attempts to match them to a meme in either the server's database or the universal database 5. If the spoken keywords match with a meme in either database, the meme is retrieved from the database 6. The meme is then posted into the active text channel
Exceptions:	1. Microphone input does not match with a meme

Implemented Customer Scenario - v1.1

The following describes a semi-fictional scenario that a character, Marco, has involving Discord and Speech-to-Meme. It is based on the above Scenario Table and helps illustrate Use Case #1, Version 2.

> Marco, a 15-year-old Discord user, finds himself playing a fast-paced video game with his friends in Discord after school. He has been playing this game with his friends for a while and has frequently encountered funny occurrences while playing but can't fully convey the humor of his situation to his friends through simply talking or typing. Explaining funny happenings with words just feels insufficient, unsatisfying and unfunny to his friends. One night while playing a round with his friends in a voice call, Marco experienced a hilarious event within the game. Bewildered and excited, he wants to send a meme to convey the absurdity of what just happened. However, the game being fast-paced means tabbing over to Discord and searching for an adequate meme could mean losing the round for his team.

1. While in game, Marco presses a preset key on his keyboard and speaks a voice command: "funny ragdoll".
2. Discord interprets the voice input and retrieves an image from the server's collection
3. The retrieved image is posted as a text message in the channel "ingame-memes", which was determined by the server owner.

> Using Discord's feature Speech-to-Meme, he posts a meme perfectly fitting the situation to the games channel in his friends' Discord server, all while staying focused on the game and bringing his team to victory. Various members of Marco's Discord server reacted humorously, both in the text channel the meme was posted in and on the voice channel Marco spoke in. Satisfied with his victory and his humorous meme that entertained all his friends, he logged out of the game and Discord for the night.

Use Case Scenario #2 - June Schober

This subsection describes the Use Case Scenario created by June Schober, implementing Customer Scenario #2

Scenario Table - v1

Table 16 implements Customer Scenario #2 by defining how messages should be posted. This specifically defines how the feature will be hands-free, and how Discord will process the data.

Table 16: Scenario Table describing sending a Meme Message as a Use Case Scenario

Use-case:	Send Meme Voice Message with Asset From Current Server
Primary actor:	User
Goal in context:	To post a Meme Message

Preconditions:	1. The User has registered a meme to a trigger (hotkey/voice command) 2. The asset is from the current server for which the message will be posted to. 3. A desired channel is specified by some means (could be via presently opened channel, specific setting, etc)
Trigger:	The User decides to send a Meme Message to a desired channel
Scenario:	1. The User uses their bound message trigger 2. Discord begins taking a short sample of audio 3. After the recording concludes, Discord combines the audio and Meme asset, then posts to the user's desired channel 4. Users who can see the channel, including the User who posted, will be able to see the message in the User's desired channel
Exceptions:	User has no write permissions to the desired channel User has no microphone

Implemented Customer Scenario - v1

We've defined the <*magic happens*> in our respective Customer Scenario using the defined Use Case Scenario Table in Figure X. Now with everything, the Scenario can be paired with a possible implementation. Please note, that the steps in the scenario are broken up throughout the scenario into its relevant spot in the narrative.

> Jack is playing Team Fortress 2 on the map "pl_upward" on a PUG (pick-up game) server. This server is currently in Highlander mode, which means that there is one team member for each of the nine classes the game offers. Playing as the Spy on the RED team, he just managed to pull 4 chain stabs on BLU after correctly determining a tiny opening on the opposing team's onslaught. A feat that is incredibly difficult to pull off reliably, but for Jack it was Tuesday afternoon.
>
> Of course, as with the cycle of the game, his luck does run out and his character gets shot to death. As he waits to respond, he considers posting that one inside joke again. There is a specific channel (#spy-moment) on a small server where Jack's friends post the same screencap of a Red Spy Ragdoll in a rather peculiar position. The image is usually captioned as "SPY JUST ___ MY ____ 💯💯🤯🤯😱🗿", where the spaces are usually ad-libbed with something random or funny that each person in the server thinks of.
>
> He comes up with a brand-new take of the meme and is excited to post it. However, the El Paso summer is hot, too hot. The AC is struggling to keep up with the weather,

> and his computer fans are whirring loudly. The computer begins to drop the occasional frame as thermal throttling turns down the computer's power. Cortes is aware that even using just the Discord overlay now for a silly joke could very well crash the game. He doesn't want to leave his team down a person or risk getting banned for disconnecting (via crashing) but really wants to post his silly joke with a golden idea before he forgets it.

1. Jack says "Discord, clip that Spy Moment"
    a. Discord begins "listening" upon hearing through Jack's microphone "Discord, clip that"
    b. Discord then also catches "Spy Moment", Jack's registered trigger for this meme
2. Discord plays an audio cue just shortly after Jack says the meme to let Jack know it's recording for the clip.

> Using Discord's new Speech-to-Meme Feature, Jack is able to post the inside joke without having to switch out of the game, or even type anything! He's able to attach an equivalent voice clip of him saying the caption, creatively ad-libing out the emoji in a noticeable Hispanic accent: "SPY JUST CHUCKLEDDUNKED MY CHUCKLENUTS! NO MANCHES! Bruh" This gets multiple Discord reactions from the other server members, to which Jack finds out after finishing the game.

3. Jack stops talking.
4. Discord detects the silence and discontinues recording to begin processing the meme.
5. Discord then posts the processed meme message to the intended channel for this message (#spy-moment) that was configured before this scenario.

Use Case Scenario #3 - Teja Dasari

This subsection describes the Use Case Scenario created by Teja Dasari, implementing Customer Scenario #3.

Scenario Table - v2

Table 17 implements Customer Scenario #3 by describing how to set up a Speech-to-Meme feature as a use case. It describes an actor's use of the setup process with uploading images and configuring voice recognition.

Table 17: Scenario table describing setting up a keybind for Speech-to-Meme

Use-case:	Set up key binding

Primary actor:	End-Users
Goal in context:	To set up the key binding that will activate the feature
Preconditions:	1. Must have already downloaded out product or enabled it 2. Must be on a computer
Trigger:	Users must set up a way to turn on the voice command prompt manually.
Scenario:	1. Our product is downloaded on User's computer. 2. User may need to manually integrate our product with Discord (it not built in). 3. User may need to boot our product or launch it in some other way. 4. Upon launch sequence, it will require a user to make a keybind before allowing them to use our product. 5. When the User, with their keyboard, successfully sets a keybind, they will be allowed to use our product. 6. The User press that key with Discord open in the background. 7. Says or makes the sound associated with a certain meme. 8. Will fill that meme into the text chat and will send it automatically.
Exceptions:	If the meme doesn't exist or isn't associated with any auditorial sound.

Implemented Customer Scenario - v2

We've defined the <*magic happens*> in our respective Customer Scenario using the defined Use Case Scenario Table in Figure X. Now with everything, the Scenario can be paired with a possible implementation. Please note, that the steps in the scenario are broken up throughout the scenario into its relevant spot in the narrative.

> Mark is a Background User, going to school and at home, studying and staying on top of his messages, and he's just found out about our product. He is downloading our product for the first time and eager to try it out, see if it's any use. Mark's problem is that he must setup our product for the first time, the problem isn't very complex, but it is a prerequisite to use our product. Mark may be keeping track of schoolwork and any other external obligations, but the problem is simple enough to be done in a few minutes.

6. The end-user downloads our software using an installer/setup wizard, be enabled from Discord's server settings, or installed as a Discord app.

7. The end-user configures their audio input device and set up voice recognition, which will allow them to use speech-to-text features.
    a. The user will need to select their audio input device from a menu.
    b. The user will be expected to test their microphone to see if the voice recognition is able to accurately interpret their voice.
8. The user will be prompted to make a keybind.
    a. The user will see a pop-up window that will ask them to press any key, ideally convenient for quickly hitting.
    b. Upon pressing the key, the pop-up window will disappear, thus setting the keybind.
9. The user will set up a custom keybind which will activate the voice command prompt.
10. The user will go through a small tutorial to set up the command for just one GIF or image, which will be put into a “store”, whether that’s a file directory or an online database.
11. The user will be redirected/open Discord.

> Mark has accomplished a part of the setup process towards replying easier and faster, and to measure the success of that outcome is to see whether or not the he has sent their first meme, since we will be keeping track of user who had our product and are using that product. The solution needs to be able to help in the setup process for Mark to be delighted, and they would still mostly be neutral about it, with the satisfaction that they had finished their setup process.

Use Case Scenario #4 - Aaron Quashnock

This subsection describes the Use Case Scenario created by Aaron Quashnock, implementing Customer Scenario #4

Scenario Table

Table 18 implements Customer Scenario #2 by defining how meme messages should be moderated. This specifically defines how the feature will allow server moderators to moderate their server’s meme store.

Table 18: Scenario table describing moderating a server’s meme store

Use-case:	Moderate Meme Store

Primary Actor:	Server Owner
Goal in context:	View memes that have been uploaded to the server and delete ones that violate server guidelines
Preconditions:	1. The user has created and now owns a Discord server 2. Another user on the Discord server has uploaded images to the server image store.
Trigger:	The server owner decides to review the current store of images in the server
Scenario:	1. The server owner logins into Discord 2. The server owner opens the server they own 3. The server owner navigates to the server settings tab 4. The server owner enters the image store interface for the server The server owner looks through all the images uploaded by users 5. The server owner spots an image they wish to remove and presses the delete button 6. Discord presents them with a prompt to warn the user that uploaded the image, report the user, ban the user, or do nothing. 7. The server owner selects the option to do nothing, and the image is deleted from the server store 8. The image is deleted from the global store, but some tracking data for the image still exists for Discord Staff Admins to see
Exceptions:	The user that uploaded an image has since left the server.

Implemented Customer Scenario

The following describes a semi-fictional scenario that a character, Elliot, has involving Discord and Speech-to-Meme. It is based on Table 18 and helps illustrate Use Case #4, Version 2.

> Elliot is the owner of a Discord server for a school badminton club that they are in. One of the chats in the server is dedicated to announcing when the next meeting is going to be, and another is for socializing with other members of the badminton club. Elliot and their friends in the club like to share memes and images relating to badminton and their latest games together. In one of the latest meetings, a student member slipped when trying to hit a fly birdie, causing them to lose the round. The student was not injured and had fun playing for the rest of the meeting.

> However, when making the announcement for the next badminton meeting, Elliot noticed that one of their friends in the server had created a meme making fun of the student that slipped, and many of the server members are sharing it in the socializing chat. It was apparently dubbed, “dude’s failing”, and students often mention it. Elliot wants everyone in the server to stop posting the image and to confront the image’s creator. However, given their history with the friend, Elliot knows that they likely didn’t mean to offend the student who slipped and so would not like to kick them out of the badminton club or server.

1. The server owner logins into Discord
2. The server owner opens the server they own
3. The server owner navigates to the server settings tab
4. The server owner enters the image store interface for the server The server owner looks through all the images uploaded by users
5. The server owner spots an image they wish to remove and presses the delete button
6. Discord presents them with a prompt to warn the user that uploaded the image, report the user, ban the user, or do nothing.
7. The server owner selects the option to do nothing, and the image is deleted from the server store

> The image is deleted from the global store, but some tracking data for the image still exists for Discord Staff Admins to see Elliot was able to simply get rid of the uses of the newly created meme as well as prevent other students from posting it. He also prevented their friend from sending the image and timed him out quickly to avoid escalating the situation and causing further impact as well as DM them about it, and all of this was done in no time flat. Elliot is relieved that it didn’t spread very far, the friend who initially posted it was very quick to apologize, and the student was able to comfortably stay in their club.

Use Case Scenario #5 - Jonah Uellenberg

This subsection describes the Use Case Scenario created by Jonah Uellenberg, implementing Customer Scenario #5.

Scenario Table - Version 1.1

Table 19 implements Customer Scenario #5 by defining how Speech-to-Meme messages should be posted via a keybind. This specifically defines how the feature will respond to

Speech-to-Meme in a background setting. For the final document, we have ensured to include an “Exceptions” category in the table.

Table 19: Scenario Table describing sending a Meme Message, initiated by a hotkey

Use-case:	Post Image via Voice Command
Primary actor:	Normal User
Goal in context:	To send an image without needing to open Discord
Preconditions:	1. User has enabled Speech-to-Meme feature. 2. User is in a voice channel. 3. Images have been configured for the user or for the server they’re in. 4. User has a specific preconfigured image in mind and knows its name.
Trigger:	User wants to send an image to add to a conversation but doesn’t want to exit their application.
Scenario:	1. User holds down the Speech-to-Meme hotkey. 2. User says the name of the meme into their microphone. 3. User lets go of the Speech-to-Meme hotkey. 4. The application searches the database of images registered to the user and to the server for the name. 5. If an image is found, it is sent to the configured text channel under the user’s name. 6. Otherwise, the user is informed that an error has occurred.
Exceptions:	No matching meme is found The user’s speech exceeds the maximum length/size The user’s query is ambiguous (multiple equally valid options exist) No hotkey is configured No microphone is configured

Implemented Customer Scenario - Version 1.1

We’ve defined the <*magic happens*> in our respective Customer Scenario using the defined Use Case Scenario Table in Table 19. Now with everything, the Scenario can be paired with a possible implementation. For the final document, we have ensured to include formatting for the specific steps involved, in order:

> Jeff is an avid gamer who likes to spend time with his friends in voice calls while playing video games. Oftentimes during conversations, Jeff wants to send an image or meme in response to something one of his friends sent, but he can't easily navigate to Discord to do so. Pausing the game to open Discord might cause him to lose his match or break the flow of his game, which can be frustrating, and if he waits until he's ready, then the conversation will probably have already moved on and sending it won't make sense anymore, losing him the satisfaction of having his friends respond to his image.

1. Jeff presses the configured keybind on his keyboard and says the name of his intended image out loud.
2. This audio is converted to text, which is then given to the database to find the corresponding image.
3. The user's personal images are searched, and if that gives no results, the server's images are searched.
4. The feature sends the found image to his friends as a message in the configured channel.
5. His friends can now see and respond to it as a part of the conversation.

> Because he never has to open Discord, Jeff can continue his game uninterrupted, while still being able to engage in conversations that are deeper than just voice.
>
> If the image is not found, the feature sends the user an error message, including the search term that was recorded. Finally, a success/failure is recorded in the database, in order to monitor the success of the feature and to determine if issues occur.

SRS Requirements (func/non-func)

Software Requirement Specifications (SRS) requirements are statements of what a proposed software system should do and how it should perform, in mutual agreement between stakeholders and development teams.

Functional Requirements

These functional requirements, described in Table 20, outline Speech-to-Memes features that attract users to use it, along with the full Discord platform. These requirements were chosen from a pool created by the full group using *MoSCoW* and iterated on to focus on the delighting features of Speech-to-Meme.

Table 20: Speech-to-Meme Functional Requirements - Version 1

Req. ID	Requirement
FR-1	The system must provide a way for users to post images using voice commands into text channels on Discord.
FR-2	The system must provide tools for moderators to effectively control the images used by the system.
FR-3	The system must be able to be activated by hitting a user-specified hotkey.
FR-4	The system must be able to connect to a voice input device.
FR-5	The system must allow users to select the image/media asset they wish to use with the feature
FR-6	The system must accept custom media from users
FR-7	The system should allow Nitro users to bypass or reduce certain established limits including but not limited to image asset limits, cool downs, and posting a S2M message from one server to any server.
FR-8	The system should filter voice input to eliminate background noise from triggering the feature.
FR-9	The system should be able to support multiple languages.
FR-10	The system could allow Server Moderators to track meme usage over the past 14 days within their server store.
FR-11	The system could provide additional decorations (such as borders) to be displayed around images as a Nitro perk.
FR-12	The system could have default assets globally available, similar to Emoji and the sticker system.
FR-13	The system will not provide on-the-fly AI image generation.
FR-14	The system won't allow Boosted Servers to store more meme message assets at a time than non-boosted server
FR-15	The system will not provide on-the-fly AI image generation.

Non-functional Requirements

This section covers non-functional requirements, split into two main categories: Performance and Safety.

Performance Requirements

The non-functional requirements, contained in Table 21, outline the performance qualities of the Speech-to-Meme feature that allow it to be used seamlessly by a variety of users. These requirements were chosen from a pool created by the full group using MOSCOW and iterated on to focus on the performance of the feature.

Table 21: Speech-to-Meme Non-Functional Performance Requirements - Version 1

Req. ID	Requirement	Rationale
NFR-1	The system must not conflict with Discord's Terms of Use.	The requirement is essential for the existence of the product. Without following Discord's Terms of Use, the product will not be possible on the platform.
NFR-2	The system should have a fast end-to-end turnaround time including: • The system must post images within 1 second of the voice command being completed. • The system must respond within 5 seconds of a spoken user command. (Performance) • Speech-to-Meme must begin recording for the message within 500 milliseconds of the trigger being activated/said/	This requirement is important in achieving the feature's main goal of allowing users to use it without breaking focus because the input and response time has an effect on the seamlessness of the feature's use. If this performance is not upheld then users may not be likely to use the feature.
NFR-3	The system should be interoperable with other Discord features related to images.	The main goal of this feature is to provide users with a seamless experience. Therefore, in order for it to feel natural, the difference between using it and sending images directly should be minimized as much as possible.

NFR-4	The system must be able to handle a high load of concurrent users. (and other performance limits)	Discord has a high number of users, and all features must be able to handle that demand. If they can't, then they won't work at all, there will therefore provide no value to users.
NFR-5	Speech-to-Meme should be lightweight on resources; not adding significant extra overhead to the Discord client	This requirement is to ensure that users do not have an unsatisfactory, end-to-end experience, as extra overhead may cause delays in messages or lag when it comes to other software applications running alongside Discord.
NFR-6	The system must be reliably usable without tabbing out another software or typing. (accessibility)	This feature is intended to allow users to send images without exiting their current application. This is therefore a core requirement of the feature, in order for it to provide value for users. After all, if using this feature requires users to tab out of their current software, why wouldn't they use the already existing features to send images?
NFR-7	The system could allow users to seamlessly use different devices.	Discord users often use multiple devices. If this feature doesn't work seamlessly on all of them, then it will frustrate the user, potentially causing them to not use the feature or only use it on certain devices.

Safety Requirements

The non-functional requirements, described in Table 22, outline the safety qualities of the system that serve to protect users using the Speech-to-Meme feature. These requirements were chosen from a pool created by the full group using MOSCOW and iterated on to focus on the user's safety.

Table 22: Speech-to-Meme Non-Functional Safety Requirements - Version 1

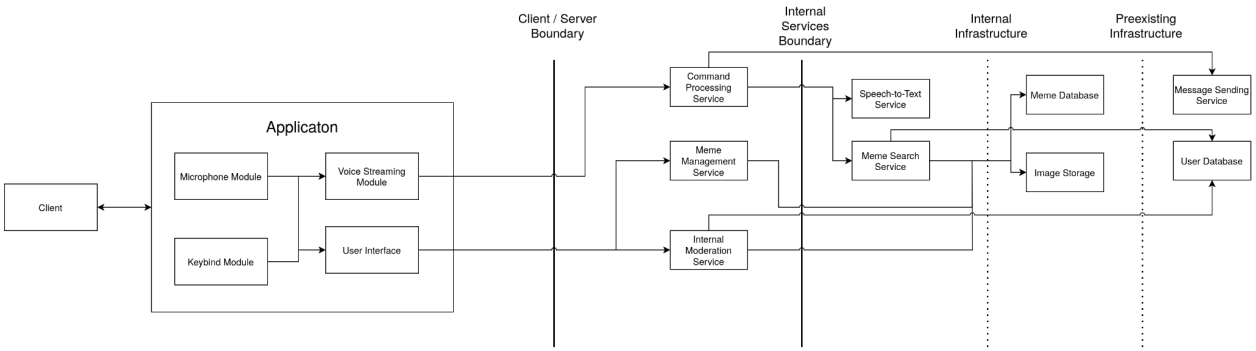
Req. ID	Requirement	Rationale

NFR-7	The system must not pose any security issues for Discord beyond the platform's baseline risk tolerance • The system must be secure, and limit access to creating/deleting memes to certain users. • The system must not expose endpoints that add and delete meme message assets from the server store images to Discord's API (Application Programming Interface) • The system must not be able to access the content of other users' messages.	Ensuring that Discord maintains security is important to avoid compromising data for both Discord and its customers. If security is not upheld, then it can cause serious issues for Discord's users, directly affecting them and their trust with Discord.
NFR-8	The system must not store sensitive user data, like authentication details.	Storing such data would pose an unnecessary security risk for users.
NFR-9	The system should integrate with existing hateful/excessively vulgar image filters to prevent such imagery from being posted	This would make sure that Discord cannot be used as a distributor of exploitative imagery.
NFR-10	The system must be moderatable by external server moderators and internal staff.	The servers in which the system is used to send messages may need to have restrictions placed on content or use.

System Architectural Description (Block Diagram)

A System Architectural Diagram is meant to represent the structure and components within a system, as a high-level blueprint regarding the components of the system and their physical-virtual location, allowing the system architecture to be defined.

Figure 12: Speech-to-Meme System Architecture Diagram - Version 1



This system architecture in Figure 12 was created as a part of Workshop 6 as a means of visually representing our Microservice architecture pattern, consisting of several services that together achieve the goal of sending images to text channels via voice input.

Domain Diagrams

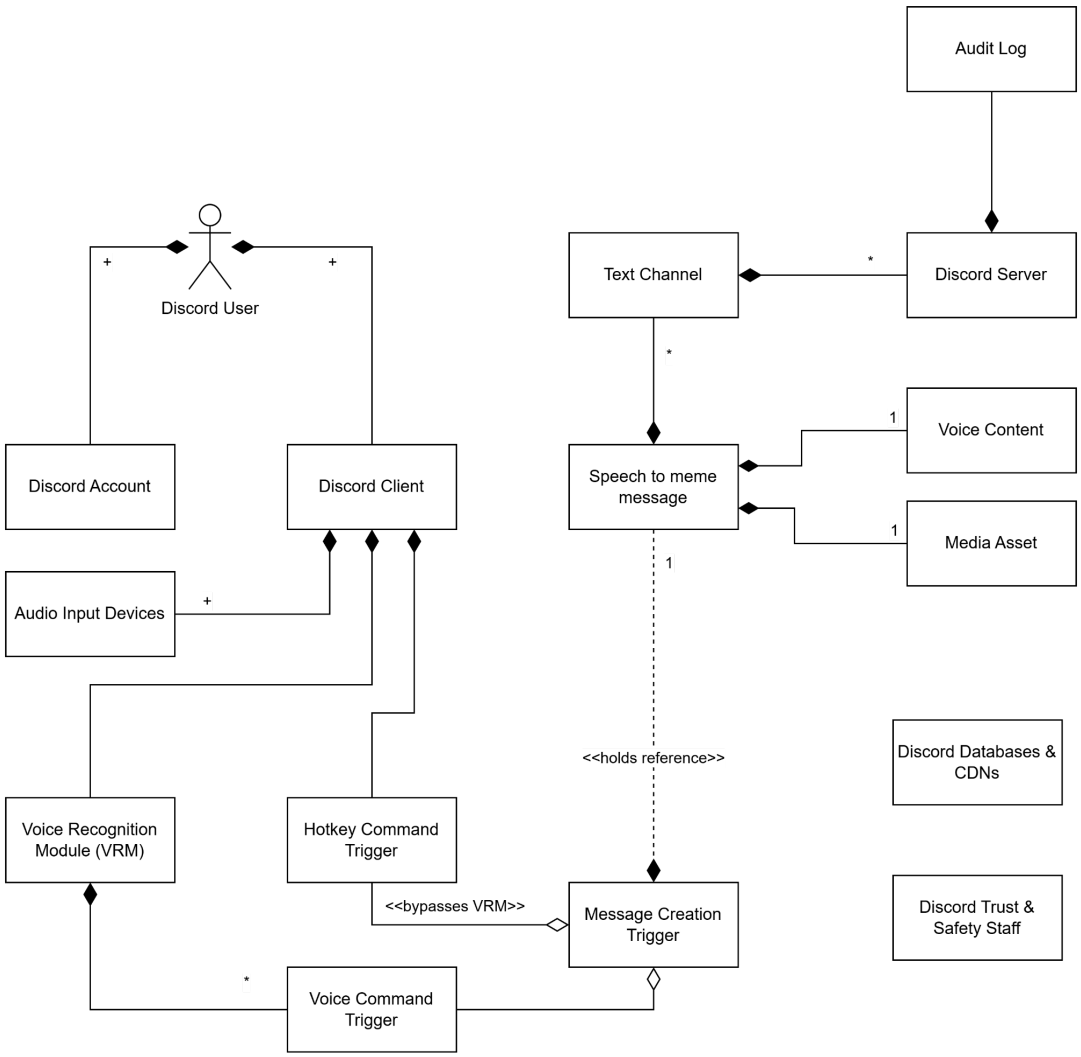
This subsection discusses Domain Diagrams in Speech-to-Meme, which displays the components and their interactions with each other.

Primary Domain Diagram

Figure 13 serves as the main building blocks comprising Speech-to-Meme. This feature consists of multiple microservices which contain small packets of data (or references) thereof. This forms the backbone of a cheap to implement and cheap to scale. Discord databases and staff are kept in mind with this diagram because they must be considered or accessed from for Speech-to-Meme to work. We also accommodate the fact that many Discord users have multiple devices and Discord accounts. In particular, this helps address the following non-functional requirements:

- NFR-2 (The system should have a fast end-to-end turnaround time).
- NFR-4 (The system must be able to handle a high load of concurrent users).
- NFR-7 (The system could allow users to seamlessly use different devices).

Figure 13: Speech-to-Meme Logical View Domain Diagram - Version 1 by June Schober

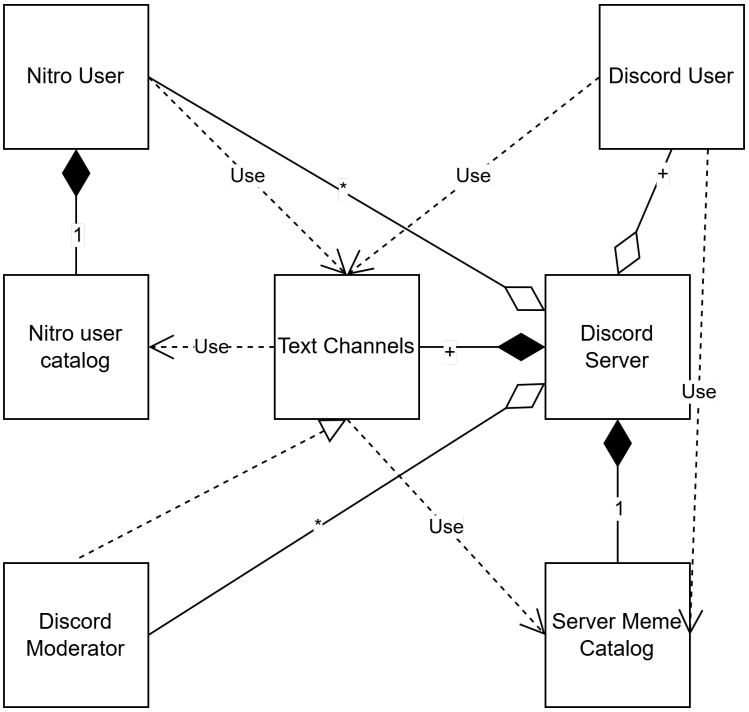


Other Domain Diagrams Considered

Figure 14 outlines an alternative structure to Speech-to-Meme. Like the above figure, this diagram consists of several interconnected services that implement the functionality of Speech-to-Meme.

Figure 14 was not chosen as our final Domain Diagram because the previous one was considered to be more complete and was better able to fit the requirements. However, we were able to get a better understanding of Speech-to-Meme as a collection of Microservices through a more abstract lens of the system.

Figure 14: Alternative Logical View Domain Diagram - Version 1 by Teja Dasari



Activity Diagram

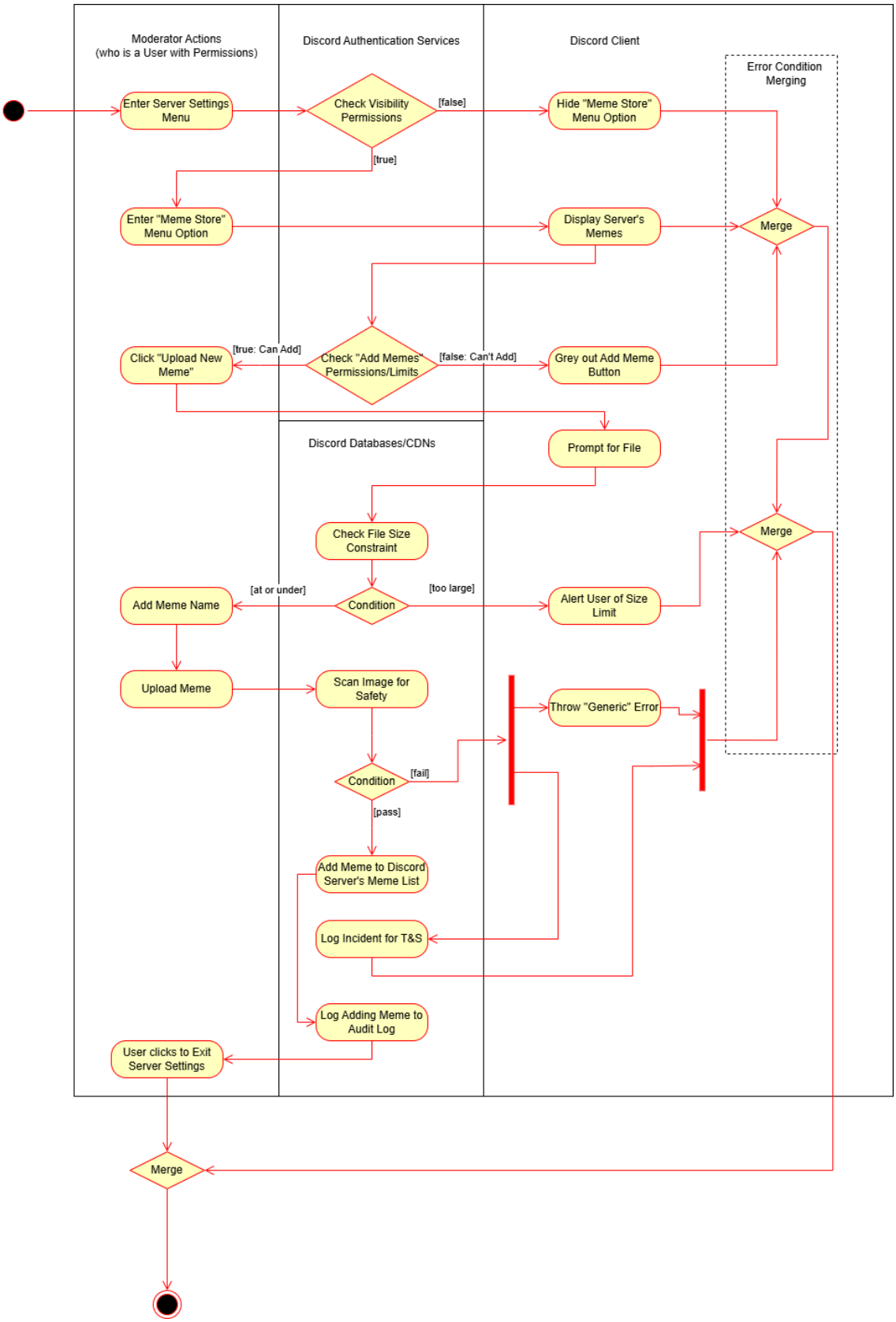
Activity diagrams are used to understand the workflow of the processes or activities in a system, moving from one state to the next.

The activity diagram, as Figure 15, serves our system design by highlighting the workflow for one use case that our product encompasses, uploading memes for a server's usage, and showing how that may work with other components that the user would interact with, that in turn, would interact with Discord. The diagram helps reinforce the implementation of the microservices architecture, as specified in Part #1, as it cleanly separates concerns across multiple services. This allows the service to easily scale to handle the demands of Discord's vast user base. This diagram is important because it can visually showcase the services pipeline that an image goes through in order to be added to a server's meme store. It also addresses the following non-functional requirements that we have elicited and documented:

- NFR-7: The system must not pose any security issues for Discord beyond the platform's baseline risk tolerance
- NFR-9: The system should integrate with existing hateful/excessively vulgar image filters to prevent such imagery from being posted
- NFR-10: The system must be moderatable by external server moderators and internal staff.

It also addresses stakeholder expectations, primarily focusing on Server Moderators, showcasing the use cases where they are interacting with Speech-to-Meme that aligns with their responsibilities in managing a server; being able to control and manage the content that is associated with the server.

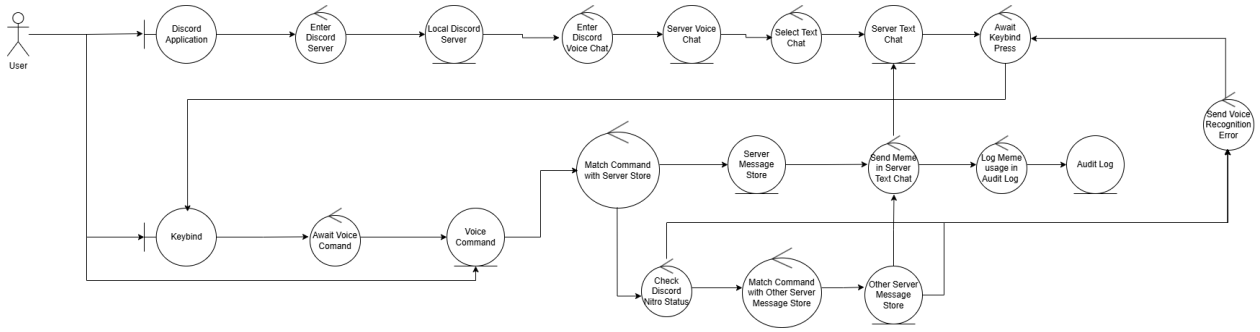
Figure 15: Uploading Meme Process View Activity Diagram - Version 1 by June Schober



Robustness Diagrams

Robustness Diagrams are used to identify different defined objects in a single use case, such as interfaces and processes. Figure 16 is a robustness diagram that outlines the process of sending memes using Speech-to-Meme. The diagram outlines end-to-end use, starting with logging into the Discord application and ending with producing an audit log after successfully sending a meme. It also includes a flow for Nitro users that send memes from other libraries.

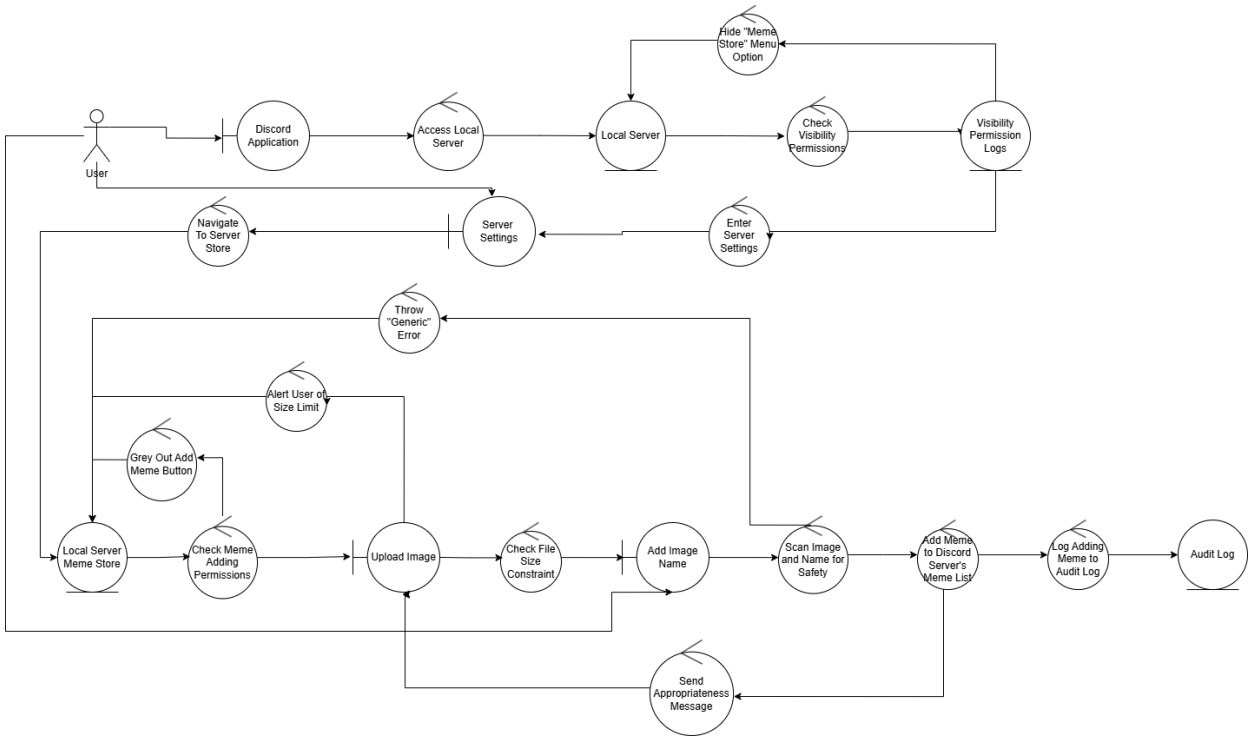
Figure 16: Final Sending Memes Robustness Diagram - Version 1 by Aaron Quashnock



We utilized the ICONIX process to progressively iterate our system description models. The primary purpose of this robustness diagram was to inform the creation of the corresponding sequence diagram, detailed later in this document under Figure 18. As we produced said sequence diagram, we revised our robustness diagram as more information was discovered.

This robustness diagram, illustrated by Figure 17, outlines the process of uploading memes to the Speech-to-Meme feature for later use. The diagram outlines end-to-end use, starting with logging into the Discord application and ending with producing an audit log after successfully uploading a meme to the server store. It also includes various flows for error handling with the file upload functionality to accommodate Discord and server permissions. The primary purpose of this robustness diagram was to inform the creation of its corresponding sequence diagram, Figure 19.

Figure 17: Final Uploading Memes Robustness Diagram - Version 2 by Aaron Quashnock

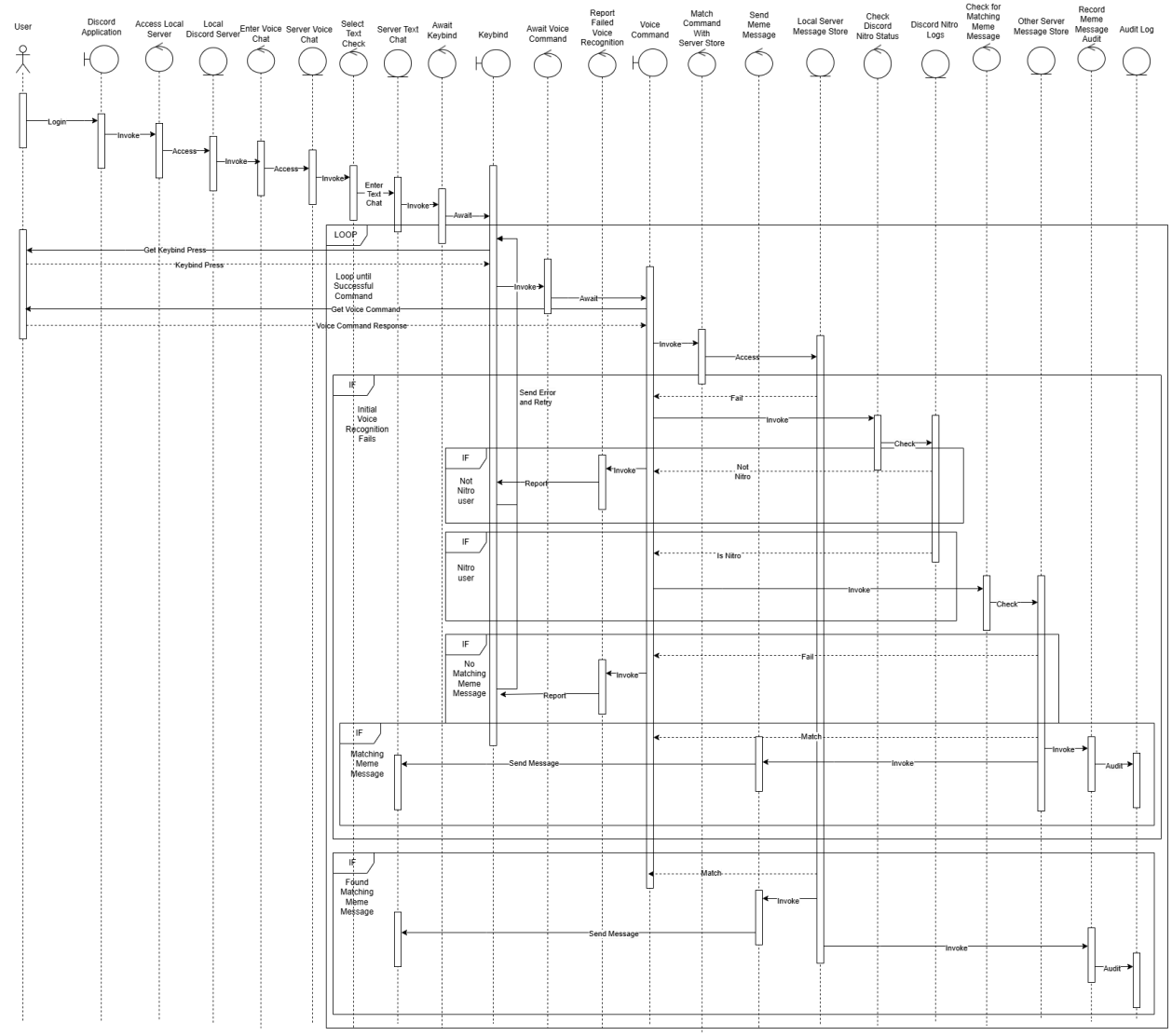


Sequence Diagrams

Sequence diagrams are an extension of robustness diagrams that represent the interactions between different objects over time.

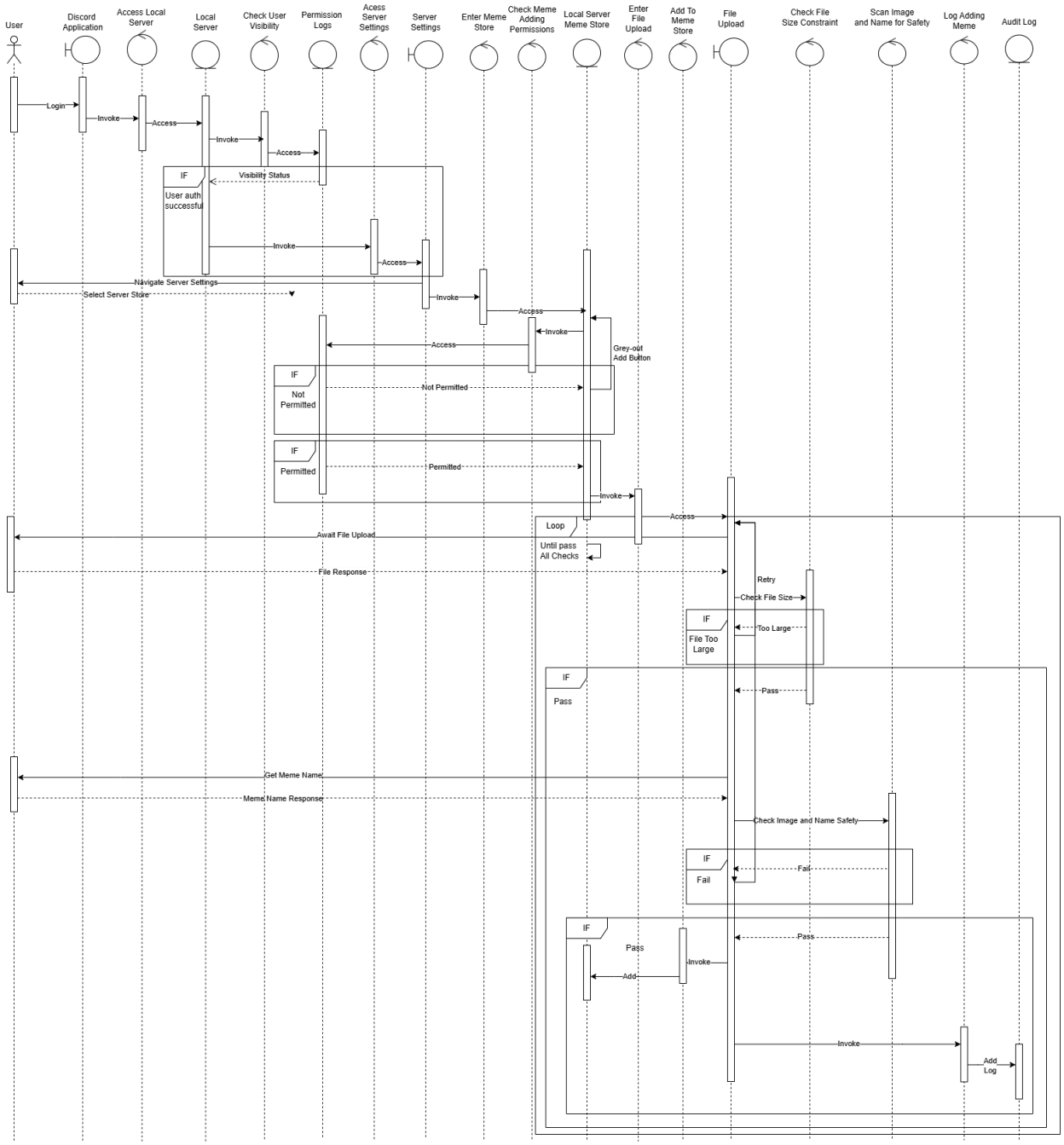
This sequence diagram, illustrated by Figure 18, outlines the process for sending memes using the Speech-to-Meme feature specified in the robustness diagram, Figure 16, established earlier in this document. Similar to the robustness diagram, it features an end-to-end flow and a flow for Nitro users sending memes from other servers. The purpose of this diagram is to specify the most common user flow with the Speech-to-Meme feature.

Figure 18: Final Sending Memes Sequence Diagram - Version 1 by Aaron Quashnock



After a revision of the robustness diagram, described earlier in this document in Figure 17, we revised our sequence diagram. This new sequence diagram, illustrated by Figure 19 outlines the process for uploading memes to the Speech-to-Meme feature specified in the robustness diagram, Figure 17. Similar to the robustness diagram, it features an end-to-end flow and various flows for handling errors with the file uploading system. The purpose of this diagram is to specify one of the most common use cases with the Speech-to-Meme feature.

Figure 19: Final Uploading Memes Sequence Diagram - Version 2 by Aaron Quashnock



User Interface/Wireframe

The following artifacts provide examples of what users will see while using Speech-to-Meme.

Figure 20 shows the command panel of Speech-to-Meme. These commands can then be executed by users to connect the bot to voice and text channels, add memes and list all memes in the server.

Figure 20: Speech-to-Meme command interface - Version 1 by Teja Desari

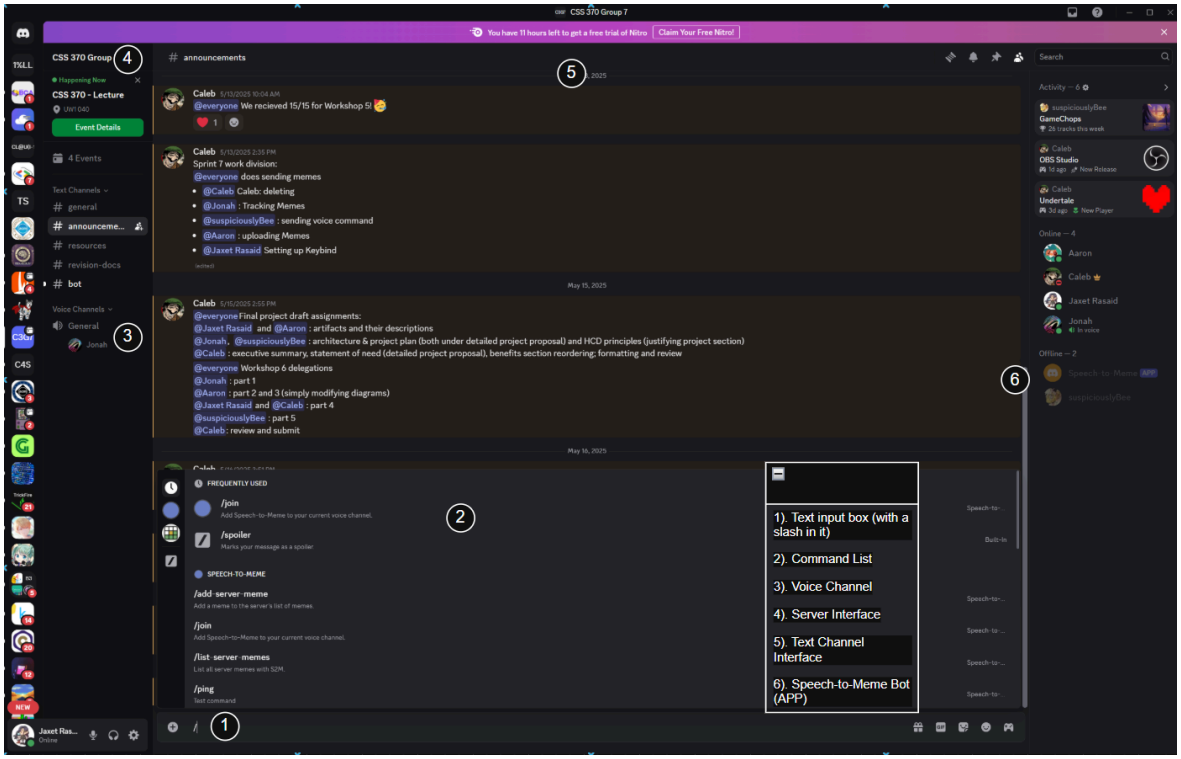
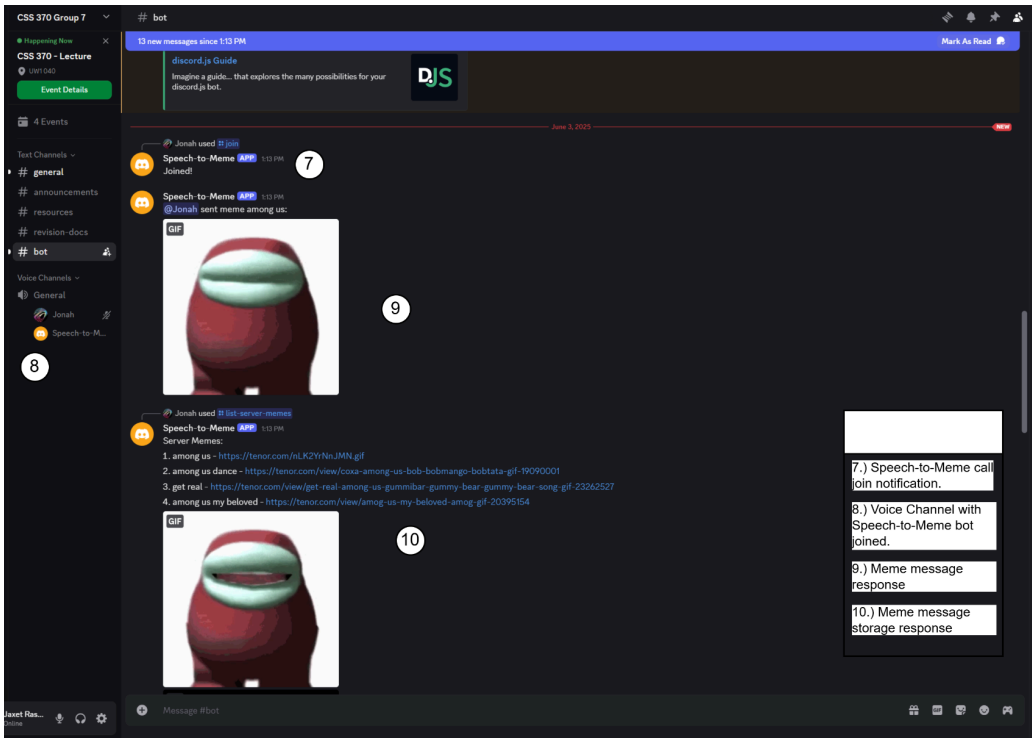


Figure 21 shows two successful executions of the “Discord, send meme Among Us” command, as the output would appear to users in text channels on Discord servers.

Figure 21: Speech-to-Meme command output - Version 1 by Teja Desari

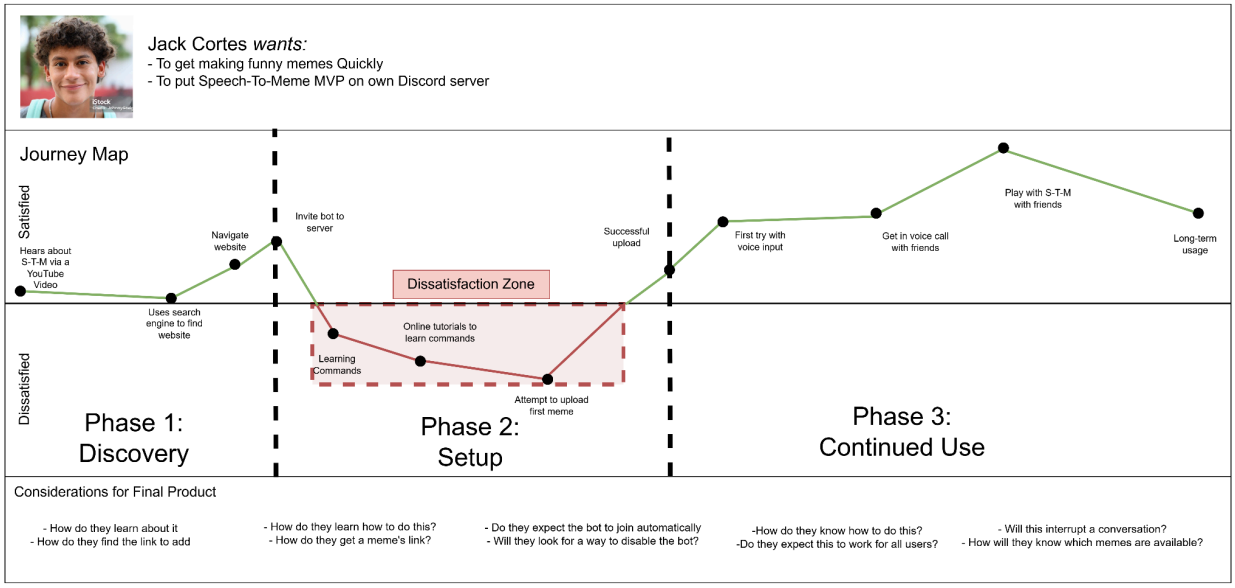


Server Owner Journey Map

Figure 22 is a proposed map of a user's journey in discovering, preparing and using Speech-to-Meme. It also tracks the expected satisfaction of the user during each phase of usage. In Phase One is discovery, in which the user finds Speech-to-Meme and prepares to use it. Phase Two is the "setup" phase, in which the user does the preliminary work to begin using Speech-to-Meme. Finally in Phase Three, the user begins and continues their usage of the feature.

Per the MVP->Production pipeline discussed in the Project Plan, This map is based primarily on the MVP because the concept must be tested and refined at the MVP level. The MVP and Production will, by definition, have user experience overlap at key points of our use cases. Ensuring that the feature can be satisfying, even at a prototype stage is crucial for the success of Speech-to-Meme.

Figure 22: Proposed map of user journey - Version 1 by ICBINGO collectively



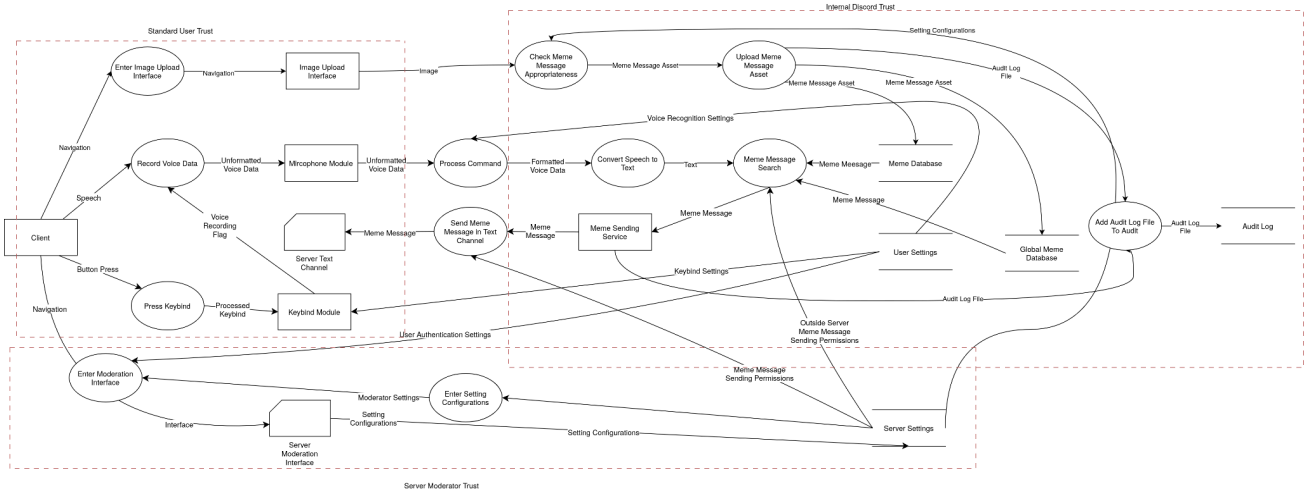
Threat Model Analysis

This subsection describes the threat model we created for the project, more details can be found in the "Risk Management: Cybersecurity/Tech" section of this document.

Data Flow Diagram 1

Figure 23 outlines the flow of information across the Speech-to-Meme system. It then shows "trust boundaries", where trust is required and therefore can be violated by malicious parties. The Data Flow Diagram is important for understanding potential "entry points" where attackers can cause damage to the system.

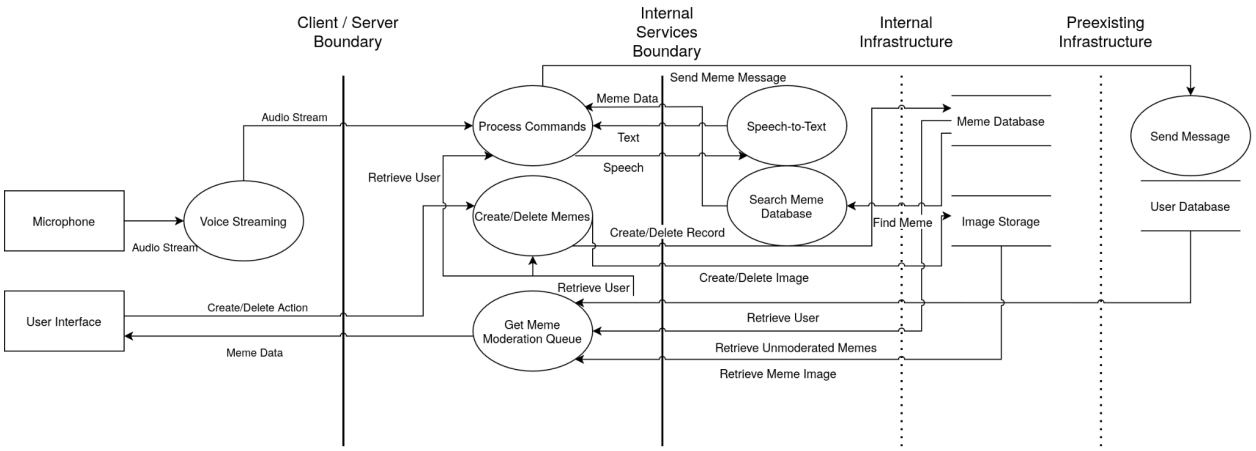
Figure 23: Data Flow Diagram 1



Data Flow Diagram 2

Figure 24 outlines the flow of information across the Speech-to-Meme system, similar to Figure 24: Data Flow Diagram 1, however it shows it more so across areas of architecture. It then shows “trust boundaries”, where trust is required and therefore can be violated by malicious parties. The Data Flow Diagram is important for understanding potential “entry points” where attackers can cause damage to the system.

Figure 24: Data Flow Diagram 2



References

[1] Voice Recorder, https://discord.js.org/docs/packages/voice/main/VoiceReceiver:Class#subscribe, (accessed Jun. 8, 2025)

[2] E. Roland, Eight forms of capital, http://www.appleseedpermaculture.com/wp-content/uploads/2011/04/8_Forms_of_Capital_PM68.pdf (accessed Apr. 29, 2025).

[3] Full stack developer salary in San Francisco, CA, https://www.indeed.com/career/full-stack-developer/salaries/San-Francisco--CA, (accessed Jun. 8, 2025).

[4] How much does a Software Developer make in Amsterdam, Netherlands, https://www.glassdoor.com/Salaries/amsterdam-netherlands-software-developer-salary-SRCH_IL.0,21_IM1112_KO22,40.htm, (accessed Jun. 8, 2025)

[5] How Much Does Teaser Animation Video Production Cost In 2025, https://advids.co/pricing/how-much-teaser-animation-video-creation-cost#:~:text=Overall%20average%20cost%20of%20teaser,ranging%20from%20$1%2C000%20to%20$16%2C000.&text=We%20identified%20a%20'sweet%20spot,quality%2C%20customization%2C%20and%20value.&text=For%20startups%20and%20SMB%20businesses,%25%20to%2034%25%20on%20average (accessed Jun. 8, 2025).