– Independent Work Report Fall 2023 –

Tigerverse: An AI-powered Q&A System for Decentralized Organizations

Aabid Ismail

Adviser: Prof. Robert Dondero

Abstract

Q&A systems are popular systems in place for communities to help each other with questions and answers that each other may know. There have been many forms of Q&A systems such as Quora, Reddit and Stackoverflow. However, these systems often suffer from limitations such as the need for users to indefinitely wait for a period of time until individuals with the specific subject expertise respond to their queries. This paper addresses these shortcomings by proposing the development of an AI-powered Q&A system tailored for the Princeton community. Our solution emphasizes two key features: authenticated user-anonymity and the integration of dedicated AI-powered chatbots for each organization within the community. These innovations aim to enhance the overall user experience and efficiency of knowledge exchange while putting minimum burden on the administrators of organizations at Princeton.

1. Introduction

In academic institutions like Princeton University, a plethora of information is managed by smaller sub-organizations such as student groups, academic departments, sports teams, and eating clubs. These decentralized organizations are responsible for maintaining and updating information within their specific domains. The nature of this information is that they change quite frequently. For example, the date for trials, deadline for applications, first day of practices, dates for events, are all information that are different from semester to semester.

The challenge arises in the dissemination of this information, which is of significant interest to a key demographic - students. Students, particularly new students seeking to engage with new clubs, sports, or eating clubs, require specific, updated information about these sub organizations. For example, a student interested in joining the Princeton Debate Panel would be interested in knowing when their trials are, how much of a time commitment is it, and how many tournaments do they have each year. However, the distribution of this information is often highly inefficient, as these decentralized sub-organizations at Princeton University do not always effectively present their information to the students. This inefficiency leads to a reliance on less efficient communication methods, such as email or phone calls, to obtain information. There is, however, a reasonable explanation as to why many such sub-organizations do not effectively present their information to students.

Historically, information from these sub-organizations was centralized on the internet through web pages. This model is effective for organizations with IT support capable of regularly updating these pages. Yet, many sub-organizations at Princeton, like student organizations and eating clubs, lack the resources to continuously update web pages with fresh information. The task of maintaining these pages is often too burdensome and time consuming for the student officers of these sub-organizations who are often busy being full-time students. As a result, many student organizations either do not have a dedicated web page or only provide basic, unchanging information, such as an introduction to their organization through their web pages. This limitation forces students to directly contact these sub-organizations for specific inquiries, which might include questions about meeting times, openness to beginners, or membership eligibility for upperclassmen.

This direct contact method imposes significant overhead for both the students seeking information and the administrators responding to inquiries. Moreover, the need to reveal one's identity when making phone calls or sending emails, as opposed to the anonymity of web browsing, can be a huge deterrent for students seeking information. This paper refers to the mental effort and time invested in this communication process – ensuring professionalism, error-free communication, and the exposure of one's identity – as 'overhead' or 'mental overhead'.

To address this problem, this paper proposes a novel solution combining an authenticated, anonymous Q&A system coupled with a chatbot ecosystem. This approach aims to reduce the time and effort required for administrators to respond to queries and for users to seek updated information from these decentralized organizations at Princeton University.

2. Background and Related Work

The inspiration for this project stems from a web application, Real Talk Princeton(RTP), that I have used immensely during my First year at Princeton to navigate specific issues such as how to file taxes as an international student to how to go about selecting classes for a semester. While Real Talk Princeton was a great resource, it has many obvious shortcomings.

Firstly, it is hosted on an external platform, Tumblr, which inhibits real talk princeton to build anything bigger than a simple Q&A website. This also constrains RTP to create a good user experience. For example, currently, users are required to fill an external Google form that is linked to the RTP website description in order to submit a question. Being diverted to a different website and having to log onto an external system again (Google Forms) is not the best user experience. Furthermore, Real Talk Princeton only allows a dedicated team of admins to answer questions. This means that if a question falls outside the scope of the admins, the question ends up not being answered. The reason for having a dedicated set of admins is mainly to ensure that questions are sanitized before being uploaded to the web page. While this process of sanitizing is important, there are modern tools that can be used to automatically filter them out as opposed to being reliant on human filtration. Beyond Real Talk Princeton, there are other platforms students commonly use for different types of student discourse. Some of them are discussed in the section below.

2.1 Sidechat and Fizz

Sidechat and Fizz are two mobile applications that provide college students the ability to post public text messages, images anonymously. The key differentiation between our project and these platforms is that both Sidechat and Fizz are used mainly as social media platforms for students to send playful jokes, memes and other attention-grabbing media. [1] Notably, these platforms do not recognize any organizations within a college and are more of an open playground where all users can say anything.

2.2 Tiger Confessions and Social Media

Tiger Confessions is very similar to both Sidechat and Fizz with the main difference being that it is hosted on Facebook. This means that users do not need to download a new app and can use it while browsing other things in Facebook. However Tiger Confessions face the same problem of Real Talk Princeton as it relies on a Facebook page, having to use Google Forms for authenticated user queries and having a singular admin for sanitizing user queries.

2.3 Email

Email has been the oldest form of student communication and it provides the ability for students and anyone affiliated with Princeton to send emails to many students at once via listservs. For example, using Hoagiemail students can send emails to all residential college listservs at once. While this method is quick and easy, it has the obvious shortcoming of having a reduced scope. For example, emailing is not the best option for questions that require anonymity, as emails do not provide user anonymity. Furthermore, emails are rarely used as a platform for student discourse as it is mainly used for one-way communications such as notices or events.

2.4 Edstem.org

Edstem.org provides students the option to send anonymous messages. However, even when students opt to send an anonymous message, the messages are not completely anonymous as course staff can see who sent the messages. Furthermore, Edstem.org categorizes students based on classes and it is meant to be a platform for questions related to courses.

Most of the platforms mentioned above serve well for some use cases. However, it is clear that we currently do not have a platform that is meant for discourse between students and student organizations and especially a platform that is not reliant on an external platform or on admins to filter messages before being published on the platform. Furthermore, as discussed in the abstract and introduction sections, all of the above platforms have an intrinsic delay in obtaining answers to questions as they rely on someone with specific subject or domain expertise to log onto the platform and answer questions.

3. Approach

I approached the project by designing a system that achieves two goals: a) Reducing the overhead for users to ask questions and seek information and b) Reducing the overhead for administrators to ensure that interested users have timely, updated information about their organization.

3.1 Reducing the overhead for users to ask questions and seek information

One point of overhead related to seeking specific information is the process of finding who is responsible to answer a question. For example, if you need to know whether as an international student you need to file your taxes twice a year or just once a year, you may be unsure of whether to direct that question to Global Financial Services, Davis IC, or your Residential College Advisor. An easier way to gather that information is to ask an upperclassman international student who had to file taxes in previous years. However, you may not know an upperclassman international student. A structure that could help bridge this gap is a public Q&A forum, much like Stackoverflow. In this structure, everyone could contribute and benefit off of the platform by creating and answering questions.

At the moment our system looks like the following:

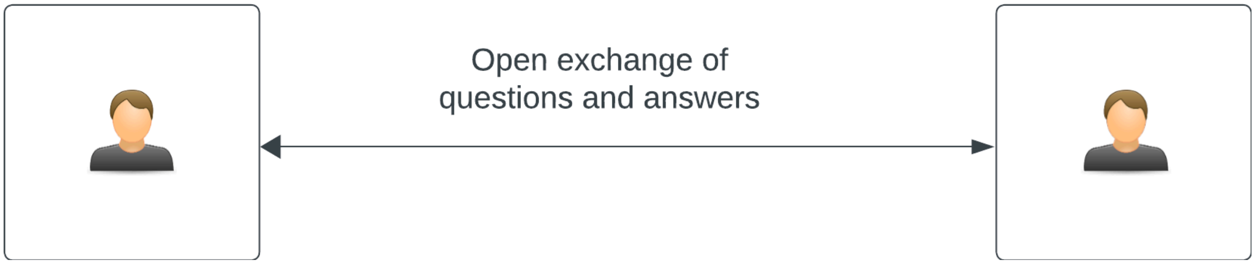


Diagram 1: System Design illustrates the concept of an open exchange of questions and answers among all users.

Now that we have a public Q&A forum to tackle the dilemma of finding the correct person to present a question, we can move onto the next pertinent point of overhead: revealing your identity.

Revealing one’s identity is a point of overhead that depends on the user’s personality and the sensitivity of the question at hand. Most users would like the option for anonymity when it comes to asking questions related to sensitive topics. For others, even when it comes to asking generic questions, they would much rather not reveal their identity as they are uncomfortable with the potential for public judgment. For these reasons, allowing users the option for anonymous questions is an important feature that the system should have. This can be easily implemented by hiding the name of the person who asked the question whenever they prompt for an anonymous question. The same can be done for anonymous answers. However, with anonymity comes other problems - how do we keep the community safe from trollers. At the moment, our system looks like the following:

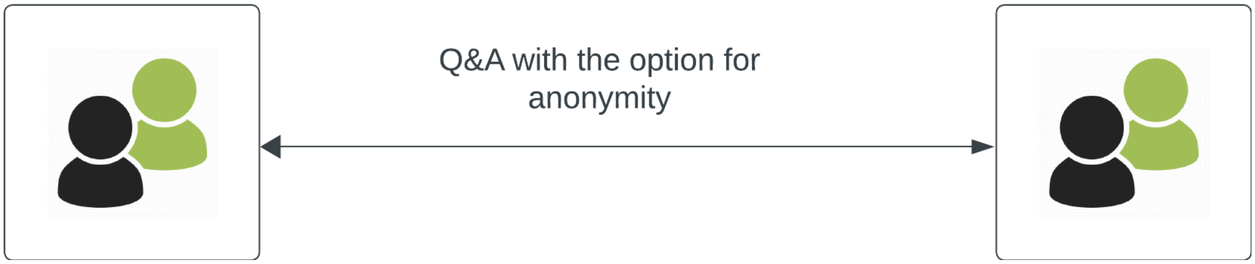


Diagram 2: System Design illustrates the concept of an open exchange of questions and answers among all users with the option for anonymity amongst users.

There are two forms of concerns when it comes to anonymous trollers. One form of concern is not knowing where they come from. For this, we need to ensure that all users of the application come from the Princeton ecosystem. With this protection, we know that all anonymous questions and answers come from within the Princeton ecosystem. The next concern falls under the usage of foul words, toxic language, usage of politically insensitive phrases and words. To not allow questions or answers that fall under these categories we use the Perspective API, which uses state-of-the-art machine learning models to disallow input that contains inappropriate language.

With these updates, our system looks like the following:

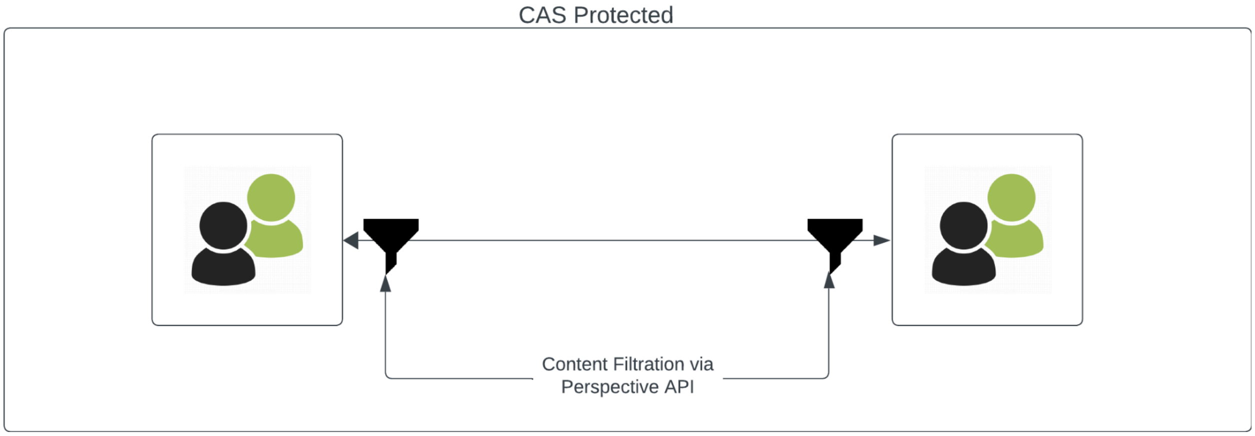


Diagram 3 illustrates a CAS-protected Q&A system with user anonymity and content protection via the Perspective API.

At the moment, our system looks great for users. Users can ask questions with the option for anonymity alongside safeguards for abuse. However, a user still has to wait until another person with the required subject expertise logs into the platform and answers their question. Additionally, what if it is a question that your peers simply do not have the answer for? In this case, there needs to be a way to direct the question to the decentralized institution that is responsible for the question, directly.

3.2 Reducing the overhead for administrators of organizations

Having a system where peers can guide each other for simple queries is great for administrators as it will reduce the overall input of questions that end up in their inbox. For example, a student organization will now receive less emails because the students interested in their club would be asking a question on Tigerverse first as opposed to sending an email to them. However, we still do not solve the fundamental problem of having a system to update and control new information for institutions without having to maintain their own websites.

To bridge this gap, we introduce chatbots for each decentralized organization. Each organization would have their own chatbot and each chatbot would have a knowledge base that holds all the information related to an organization. Whenever an organization needs to make a change to their existing information, such as weekly practice times for a sport or weekly office hours, they can simply edit that entry within the knowledge base of the chatbot.

With the introduction of the chatbot to our system, users now have two ways to go about getting answers to their questions. If they know which organization their question is directed towards, they can find the chatbot of that organization and query the chatbot for an answer. If the chatbot has the appropriate knowledge surrounding the user's question, the chatbot will intelligently parse the data from the knowledge base and respond to the user's query. If the question is beyond the scope of the knowledge base that the chatbot has access to, the chatbot will ask the user to direct the query to the organization via their email address. The user can then either send an email, post a question on Tigerverse so that any of their peers who have the answer can answer or do both. Technical details on how the chatbot works will be addressed in the next section of the paper.

Our overall system now looks like this:

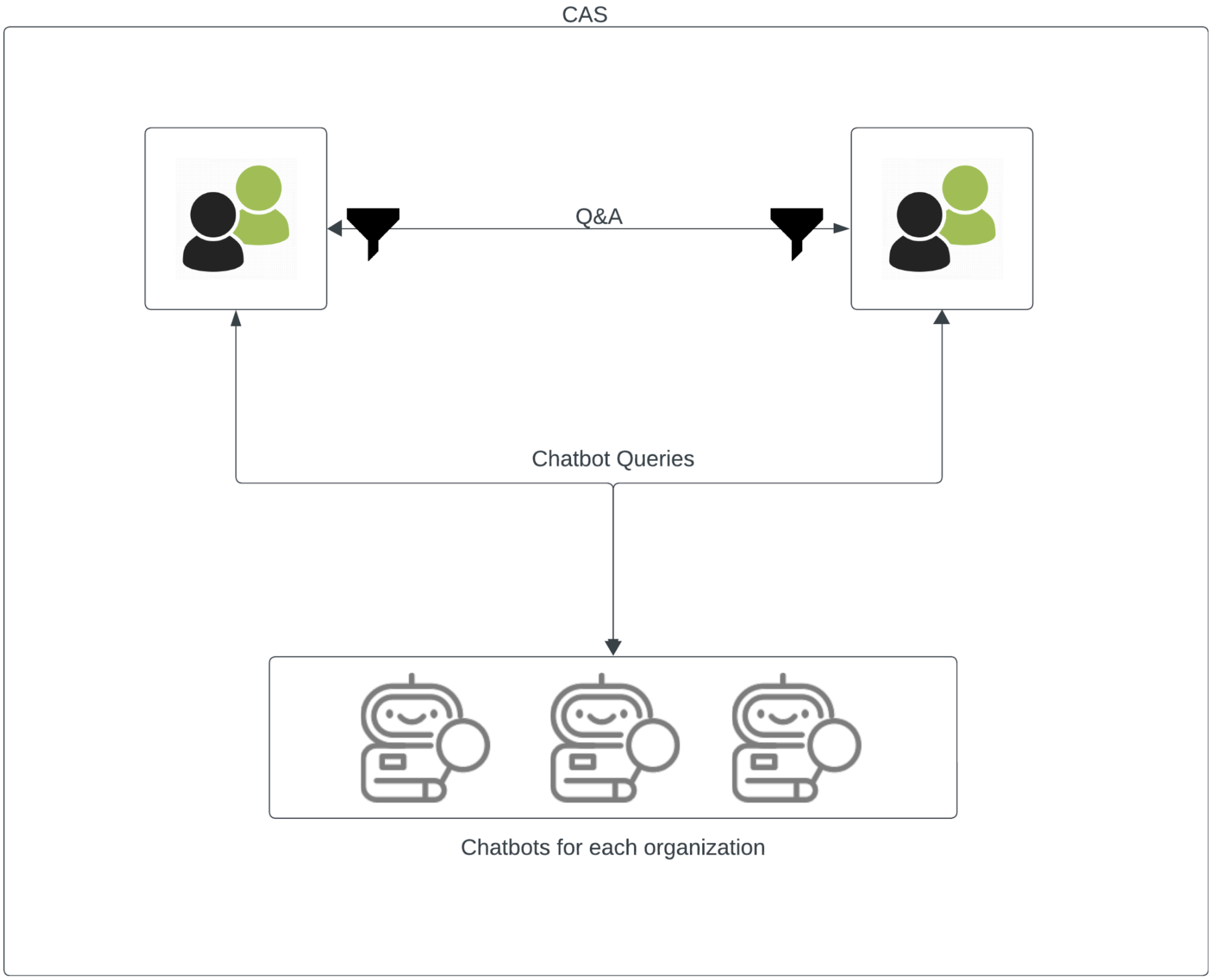


Diagram 4 illustrates a CAS-protected Q&A system with user anonymity, content protection, and organization-specific chatbots.

4. Technical Implementation

This section discusses the system architecture of Tigerverse and how each component of the system works. We will discuss what the technical components are and how they interact with each other. Similar to the previous section, we will break down the system architecture into four stages. In each stage, we will introduce and integrate additional components, building upon the previous ones.

Technical System Design 1: Q&A System

Our first system design involved an open Q&A exchange between users. We used React.js(React) for the frontend of the Q&A system while using Express.js(Express) for the main backend server. There were several advantages in using React and Express as the main stack. One main advantage was that both frameworks use Javascript which allows a single developer to easily switch back and forth while developing. The next advantage was the ability to use Typescript and the ability to define types and interfaces that can persist between both the backend and frontend. For example, the “question” type can be used within both the frontend and backend. Furthermore, React and Express are very mature frameworks that are extensively used in industry. This helps in searching for help online during debugging.

For our initial Q&A system to work we needed to build API endpoints to create questions, get questions, create answers, and get answers by question IDs. I used the REST architecture for the APIs as it is the most commonly used architecture in practice. Other possible architectures include GraphQL introduced by Facebook. The frontend allows users to view the questions in the order of the most recent to least recent. It also allows users to create a new question and answer questions that have been created. Beyond these basic capabilities, a feature that we implemented that is novel in industry and increasingly common in consumer web applications is infinite scrolling.

4.1 Infinite Scrolling

The traditional method of presenting very large lists of data to users involved using page numbers. In this approach, there was a fixed window height, and at the bottom of the window, page numbers were displayed.

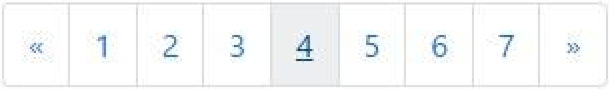


Diagram 5 shows a picture of page numbers bar used as the method for pagination in traditional web applications

However, in modern web applications, there is a growing trend towards implementing infinite scrolling to load new content. For example all modern web applications such as Facebook, Instagram, Twitter (Now, X.com) all use infinite scrolling as opposed to page numbers. From the user's point of view, as they scroll down a screen that consists of a list of data, new information loads at the bottom of the window, creating the illusion of an infinite scrolling experience.

To enable infinite scrolling in Tigerverse, we had to make use of the Intersection Observer API of Javascript. The MDN web doc introduces it as “a way to asynchronously observe changes in the intersection of a target element with an ancestor element or with a top-level document's viewport.” [3] For a detailed explanation of how to use the Intersection Observer API for infinite scrolling and how to use the Intersection Observer API specifically in React.js head over to Appendix A.

Technical System Design 2: Adding the option for Anonymity

The improvement we make during this stage is the addition of the option for anonymous questions and answers. The option for anonymity can be achieved by modifying the database schema for questions and answers. At the moment when a question or answer is created, we record the user id of the question creator inside the question model that is created. As such, when we retrieve a list of questions and answers we need to follow the user id and retrieve the netid of the user who created the q&a and display it alongside the question or answer. We need to modify this workflow to enable anonymous questions and answers.

One way to achieve this is to have one boolean value per question and answer created. Whenever a user decides to create an anonymous q&a, the boolean "isAnonymous" will be marked true. Afterwards, whenever we retrieve a list of questions and answers, for each question and answer we can check whether its "isAnonymous" field is marked true. If it is set to true, instead of returning the netid of the user who created it we can return "anonymous". While this method works to achieve user anonymity, it unfortunately does not work very efficiently as it requires a comparison operation to be done in the order of O(n) for every piece of question and answer that is retrieved.

An important observation is that in Q&A systems, the number of GET requests to retrieve questions and answers are significantly higher than the number of POST requests to create questions and answers. The infamous 1% rule states that if you let 100 users online then only one will create a new piece of content while 89 will just view content. [2] We can use this observation to put most of the burden of handling anonymity when a question or answer is created as opposed to when retrieving them. Following that train of throught, we can have a new string field called “displayName" for each question and answer created. We will populate the “displayName” whenever a new question or answer is created. We default the displayName to the user’s netid. However, if a user elects their question or answer to be anonymous, we replace the field, displayName with “anonymous” instead. This way, whenever we retrieve a list of questions or answers we do not need to do a foreign-key lookup to search for the user who created it and nor do we need to check all isAnonymous booleans per question or answer - we simply retrieve the list of questions and answers and the list of displayNames associated with the questions and answers. Therefore, we have now achieved a more efficient and scalable version of a Q&A system with the option for anonymity.

At this point, our technical system architecture looks like the following:

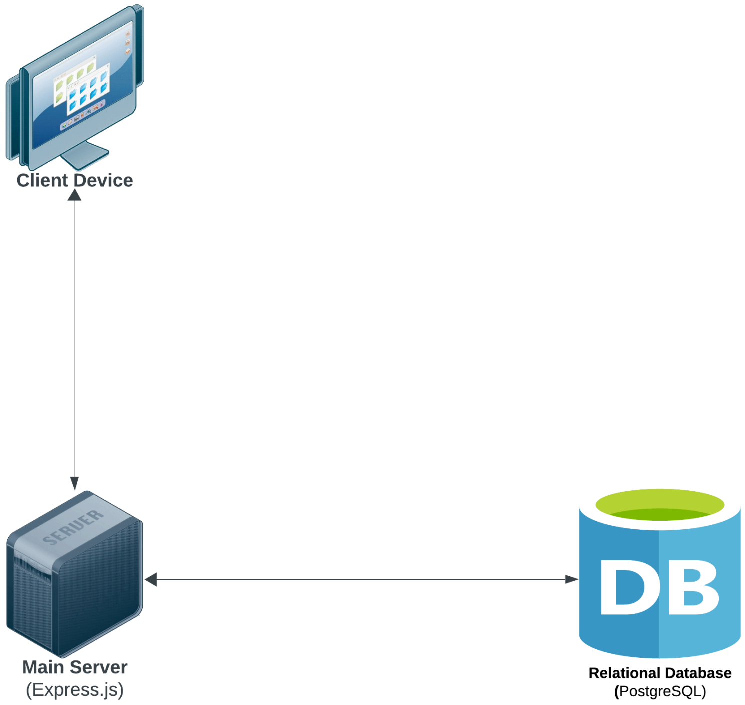


Diagram 6 illustrates a technical system architecture design for a Q&A system

Technical System Design 3: Adding Authentication and Content Moderation

We have now implemented the basic functionality of our Q&A system. However, we still have not added authentication and content moderation to it. In terms of Authentication, we use the Central Authentication System to authenticate a user during Login. For authorizing each subsequent request, we use JSON web tokens (JWT). [4]

4.2 Stateful vs Stateless Authorization

JWT-based authorization is a relatively new concept. The way industry used to perform authorization requests was through session tables that store each user session on the server. The problem with the use of session tables was that it a) takes up memory or database storage to store the session tables and b) it becomes increasingly difficult to implement microservices as every microservice will now need to copy the session table in order to authorize requests to its server.

JWTs eliminate this problem by shifting the burden of authorization to the client. Whenever a user is first authenticated(via CAS), the server will issue a JWT that is signed by a secret key located safely in the server. The signed JWT has information about the user such as the netid, and when the token expires. The client will then store the JWT securely via an http-only cookie or via local storage. Whenever a client makes any request (other than the initial login request), the client needs to attach the token to the Authorization Header field. Whenever a JWT expires or is close to expiring, the server prompts the user to authenticate themselves with CAS again to issue a new token. If the user already has a valid CAS session, this request will automatically be fulfilled. There are several other mechanisms to prolong user sessions securely via refresh tokens, however, that is outside the scope of this project.

Notably, the usage of JWTs eliminates the necessity to maintain an additional sessions database. Furthermore, it is faster than session tables as a database lookup is slower than decrypting a token. Furthermore, if we need to horizontally scale the server by adding more server instances or if we need to add a microservice, we can simply share the secret token with the microservice and allow the microservice to do its own authorization of requests.

4.3 Content Moderation

Content moderation for answers and questions is done via the Perspective API, an external service that provides a scoring system for content based on several parameters such as toxicity, identity theft, profanity and many more. Whenever the server receives a piece of content such as a question or an answer, it passes the piece of content through the Perspective API and if it returns a high score for either of their parameters we return a 400-level error. (specifically, a 422 error for unprocessable content).

At this point, our technical system architecture looks like the following:

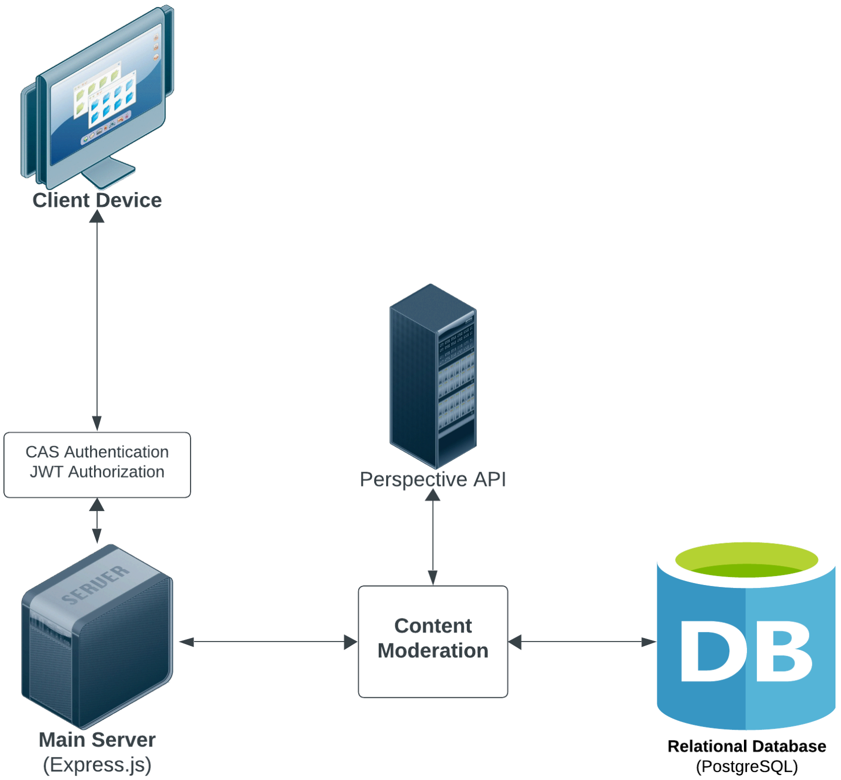


Diagram 7 illustrates a CAS-protected Q&A system with content moderation

Technical System Design 4: Chatbots for Organizations

Our technical architecture looks robust for a question and answering system. In many levels, the system we have designed surpasses alternate Q&A systems that currently exist for Princeton students such as Real-Talk Princeton and Tiger Confessions. However, an important extension to our system is the addition of chatbots to help with students getting instantaneous answers based on the available knowledge uploaded by organizations. Before we explore how these chatbots can be connected to our existing system, we need to take a quick detour to explain how chatbots work with transformer models and how we can add a knowledge base to extend the capabilities of a chatbot. As the main focus of this paper is the extension made to the project via transformer models and not the transformer models itself, the following topics will be overly simplified.

4.4 Large Language Models (LLMs)

A large language model learns to predict the next word of a sequence of words by exposing its large neural network to a large volume of training data. Let's say we pass the sentence "The dog barked at the tree" to the neural network. The neural network actively tries to guess the next word of the sentence and compares its guess with the actual next word of the training data and then adjusts the parameters of its network so that it guesses the correct word for scenarios like that in the future. For example, at the end of "The dog", the neural network might predict the word "smiled". But after seeing the word "barked" from the training data, it will adjust its parameters such that the next time someone inputs "The dog", it will predict the next word as "barked".

Doing this at a humongous scale allows Large Language Models to generate text with human-like precision. For example GPT-4 published by OpenAI reportedly has close to 1.76 Trillion parameters and took more than $100 million dollars of compute power to be trained. [5] However, it is not just a neural network that allows a LLM to spout human-like content. For LLMs to work well it needs to understand the context surrounding its input data as well.

4.5 Transformers and Generative Pretrained Transformers (GPT)

Transformers enable Large Language Models (LLMs) to identify interdependencies between words and sentences, playing a crucial role in understanding context and generating coherent responses. Unlike traditional neural networks, transformers use a mechanism known as "attention" to weigh the importance of different parts of the input data. This attention mechanism allows transformers to focus on relevant parts of the text when making predictions, making them highly effective for tasks involving natural language understanding and generation. For example in the sentence:

“The dog barked at the tree because it was starving”, attention is what enables transformers to learn that the word “it” references the dog and not the tree.

Generative Pre-trained Transformer models (GPTs), are a type of transformer that has been specifically designed for generating text. The 'pre-trained' aspect refers to the initial training phase, where the model is exposed to a large corpus of text data. The immense scale of this pre-training, as exemplified by GPT-4's exposure to approximately 1.76 trillion parameters, equips the model with a broad understanding of language nuances and contexts that can be used to power chatbots for many use cases.

Companies like OpenAI have published their GPT models which allow other developers to build on top of them. For example, if someone wants to build a chatbot for the medical industry they have the option to fine-tune OpenAI’s GPT model on their own training dataset which would consist of medicine-related books, articles, videos, etc. Another way to build on top of GPT models for a specific use case is through Retrieval Augmented Generation (RAG), which is what we use in our project to power the chatbots for organizations.

4.6 Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation is a process that allows developers to reference their own data while using powerful transformer models such as GPTs. While GPTs have the ability to understand the nuances and contexts of language, we might be interested in asking GPTs to reference our own dataset to reason against a user query. For this to work, we create an intermediate step before feeding the user query to the transformer model. This intermediate step involves searching through our own dataset to find data that matches the user query and augmenting these pieces of data to the user query before sending it to the transformer model. This is best explained by comparing and contrasting an example that directly queries the GPT model against an example that uses RAG before reaching the GPT model.

Let's first look at what happens when we query a pre-trained generative transformer model directly with a typical student query related to a student organization.

Case 1: User asks the query "How can I join the Princeton Debate Panel?" directly from a GPT model

User
SERVER
GPT Transformer

Diagram 8 illustrates a user directly communicating with a GPT Transformer Model

After feeding the query to the GPT-3.5 model, it responded to the query with the following answer:

Output: "To join the Princeton Debate Panel, you should follow these steps. Research: Start by researching the Princeton Debate Panel and their requirements. Visit their official website or contact them directly to gather information about their application process, eligibility criteria, and any specific guidelines they have. Prepare: Once you have a clear understanding of their requirements, prepare your application. This may include filling out an application form, submitting your academic records, and providing any necessary recommendations or essays. Attend Meetings: If the Princeton Debate Panel holds regular meetings or events, attend them to get a sense of the group's dynamics and culture. This will also allow you to connect with current members and express your interest in joining. Showcase Your Skills: Highlight your debate or public speaking skills, if you have any. Prepare a persuasive argument or presentation that demonstrates your abilities. This can be useful during interviews or auditions. Apply: Submit your application following their specific instructions. Make sure to meet all deadlines and provide all required documents. Interview/Audition: Some debate panels may require an interview or audition as part of the selection process. Be prepared to discuss your interest in debate and your relevant skills. Stay Engaged: While waiting for a response, stay engaged with the debate community at Princeton. Attend events, join discussions, and continue to show your passion for debate. Follow Up: If you don't hear back within a reasonable timeframe, consider following up with the panel to express your continued interest and inquire about the status of your application. Remember to check the Princeton Debate Panel's specific requirements and procedures, as they may vary from year to year. Good luck with your application! If you need more detailed information or specific guidance, feel free to ask."

As you can notice, the output of the GPT is grammatically correct and contextually relevant. However, it spouts incredibly wordy and vague information such as calling the user to "research the club", and also provides misleading facts such as "prepare a presentation that showcases your talents". This phenomenon is known as AI Hallucination. IBM defines AI Hallucinations as "a phenomenon wherein a large language model (LLM)—often a generative AI chatbot or computer vision tool—perceives patterns or objects that are nonexistent or imperceptible to human observers, creating outputs that are nonsensical or altogether inaccurate." [6] The main reason behind hallucinations is that GPT models were not exposed to knowledge that provides answers to specific queries such as how do I join the Princeton Debate

Panel during its training phase. In fact, it is likely that GPT models in the future will not be able to answer such queries because information such as “how do I join the Princeton Debate Panel” is often not public data and it also changes from semester to semester.

Case 2: User asks the query “How can I join the Princeton Debate Panel?” from a GPT model via RAG

In this case, there are several things that happen once a user types the query and clicks enter.

Step 1: Embedding the user query into a vector via an embedding model.
Step 2: Comparing the user vector against all vectors in the knowledge base and retrieving the words or sentences associated with the closest vectors in the knowledge base.
Step 3: Attaching the words or sentences associated with the closest vectors to the user query via prompt engineering.
Step 4: Querying the GPT transformer model with the augmented query.

We will walk through each of these steps in the context of our example query. A diagram that represents the workflow of these steps is depicted below:

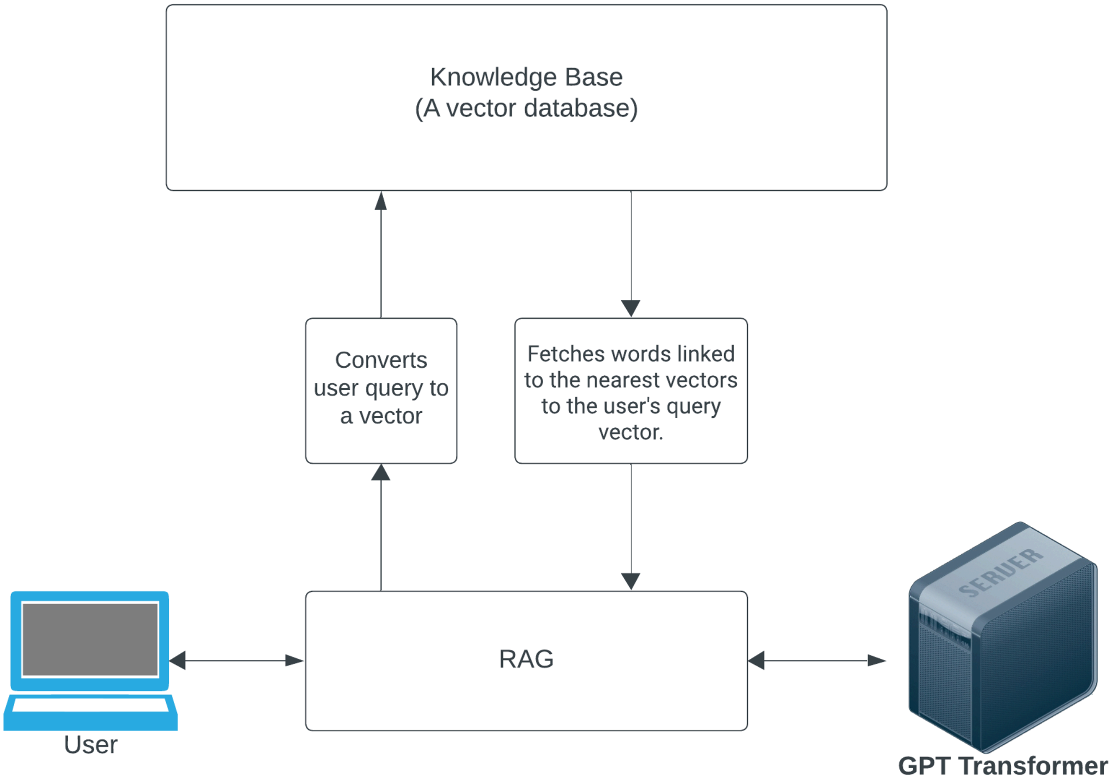

Knowledge Base
(A vector database)
Converts user query to a vector
Fetches words linked to the nearest vectors to the user's query vector.
RAG
User
SERVER
GPT Transformer

Diagram 9 illustrates a user communicating with a GPT Transformer Model via Retrieval Augmented Generation

Using our previous example, let's assume that a user sends the query "How can I join the Princeton Debate Panel (PDP)?". Let's also assume that the PDP administrators have entered the sentence "We hold trials during the first weekend of each semester at the Oak Lounge at Whig Hall" into the knowledge base of the chatbot.

Step 1: Embedding the user query into a vector via an embedding model

During the initial step, the sentence "How can I join the Princeton Debate Panel?" will be converted to a vector via an embedding model. If we use the BAAI-base embedding model, the output vector will have 767 dimensions. We will call the converted vector, the "user query vector".

Step 2: Comparing the user query vector against all vectors in the knowledge base

We will now take the user query vector and find the closest vectors to it from the knowledge base. We can use either the cosine distance or the dot product to measure the distances. In our case, if we had the sentence “We hold trials during the first weekend of each semester at the Oak Lounge at Whig Hall” encoded as a vector in our knowledge base (with the same embedding model), then it would be ranked first amongst the closest vectors. We will call the piece of text retrieved from the knowledge base as “context data”.

Step 3: Attaching the words or sentences associated with the closest vectors to the user query via prompt engineering

We will now create a new prompt to send to the GPT transformer model. We cannot simply attach the context data to the user query as we need to provide explicit instructions to the GPT model on what to do with the context data. For example, we need to tell the model what to do in the case that the context data does not answer the user query. The following prompt template was used for our project:

“

You are a representative for the [Organization Name]. I will provide a user query to you delimited by triple quotes ```. Your task is to answer the user query using the context data delimited by triple exclamation marks !!!. Do not make up new information. If the user query is outside the scope of the context data, respond by saying “Your question is outside the scope of my knowledge at the moment. Please contact the organization administrators at [organization_email]”.

``` [user query] ```

!!! [context data] !!!

”

Step 4: Querying the GPT transformer model with the augmented query.

Now that we have our new prompt, we can send this to the GPT model to seek an answer. Using GPT-3.5 we received the following answer from the chatbot:

Output: “To join the Princeton Debate Panel, you can participate in the trials that are held during the first weekend of each semester at the Oak Lounge at Whig Hall. This is where you can showcase your debating skills and potentially become a member of the panel. Good luck with your application! If you have any more specific questions about the process, feel free to ask.”

This answer is clearly more accurate, and concise compared to the previous example, and it provides clear instructions on what the student has to do next.

4.7 Vector Databases

A vector database is a database where we store the knowledge base of an organization. Unlike traditional database systems, vector databases do not have schemas. Whenever an organization adds new sentences about that organization, the text that was entered gets embedded into vectors and those vectors are stored in a vector database so that they can be compared against user query vectors. We used the Pinecone vector database for our project. Pinecone provides the option to add metadata to each vector, and also the ability to search vectors amongst namespaces. Namespaces are isolated subgroups within a vector database. This fits our project very well as we can define a namespace per organization so that we only search for vectors within a given namespace/organization per user query.

4.8 Connecting RAG Chatbots to Tigerverse

An important step before adding chatbots to Tigerverse is uploading the knowledge base for each organization. In a real world setting, if an organization wants to list their chatbot on Tigerverse they would have to upload data to their knowledge base (Data in this case simply refers to sentences that describe information relevant to their organization such as when do they convene weekly, how can students join, what kind of events they have, etc.). Assuming that the organizations complete this step to set up their knowledge bases, users can search for chatbots that they want to interact with and carry on a conversation. Our final technical system architecture design looks like the following:

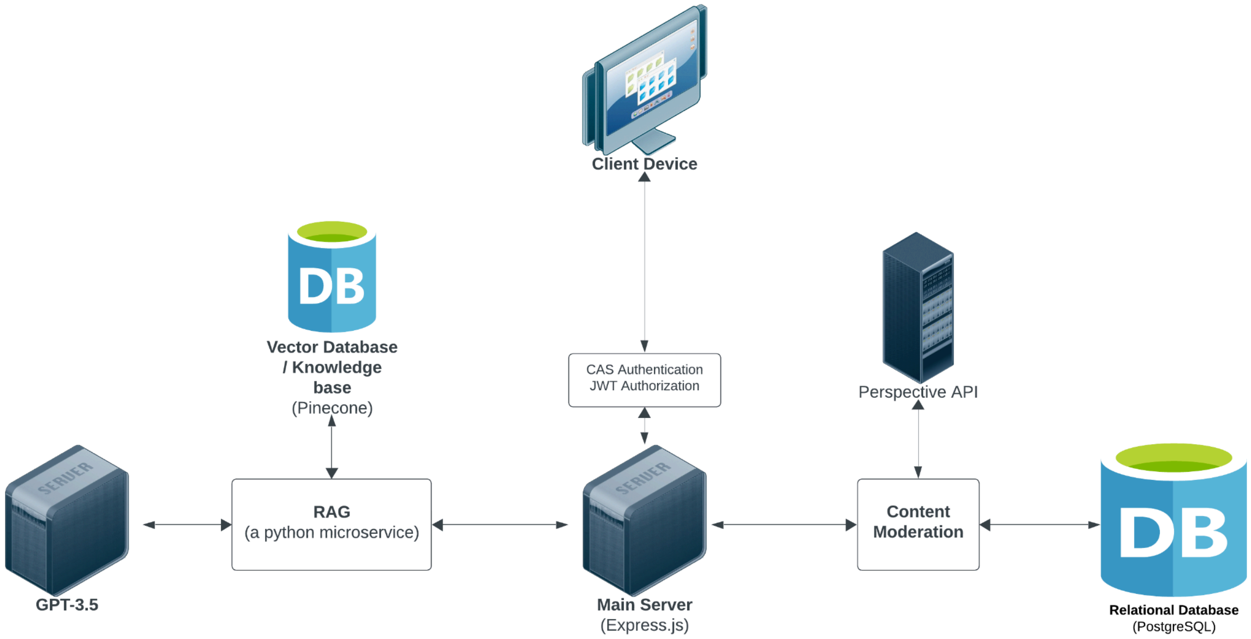


Diagram 10 illustrates the final technical system architecture of a Q&A system coupled with a chatbot ecosystem for organizations

4.9 Additional components not shown in the final system architecture diagram

There are several other components that were omitted from the final system architecture above in order to preserve its simplicity. Firstly, there are two other user interfaces - namely, the super admin interface and the chatbot admin interface.

The super admin interface is a client that has privileges to delete any question or answer and add chatbot accounts to Tigerverse. The super admin interface directly interacts with the main Express.js server.

The chatbot admin interface is the interface where organization admins login to manage their chatbot. They can edit existing data of the chatbot, or add new data to the chat bot. The chatbot admin interface is created using Next.js, a powerful framework made on top of React.js. The chatbot admin interface too directly interacts with the main Express.js server. However, the actions performed by the chatbot admin interface such as adding, updating or deleting data from their knowledge base causes the main server to perform those operations by invoking API endpoints of the python microservice which has access to the Pinecone vector database where the knowledge base for organizations live.

The reason to use a different microsystem written in Python is two fold: a) For better system isolation, and b) The Python ecosystem is more mature and capable at handling ML-related functions such as vector computations, and tokenizations.

5. Evaluation

Evaluation for this project was done in two stages. One was a pre-development evaluation to assess the need for a platform like Tigerverse and the second was a post-development evaluation to test the platform and receive user feedback.

5.1 Pre-development User Survey

A survey was sent out via Hoagie Mail to all residential college listservs with a short survey on what students think of a new platform for student discourse and activism. Highlights of the survey are mentioned below:

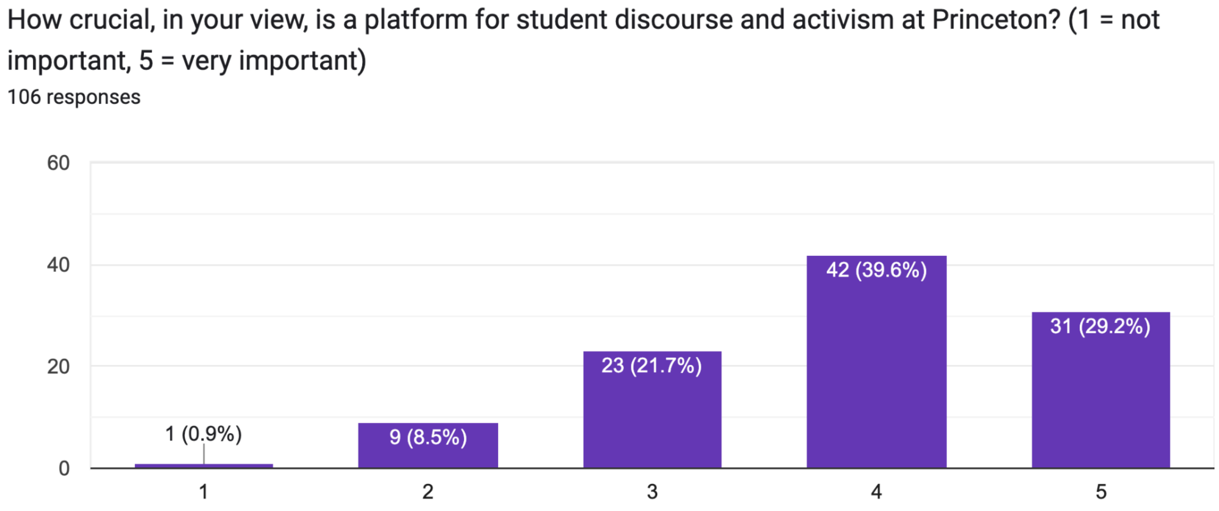


Diagram 11 illustrates that most students believe that a new platform for student discourse and activism is crucial at Princeton

The student survey also inquired about students' usage of Real Talk Princeton (RTP) and Tiger Confessions. According to the results, a significant 71.7% of students indicated that they had never heard of Real Talk Princeton.Further, a staggering 90.6% of students reported that they had never asked a question on RTP. Similar findings were observed for Tiger Confessions where 90.6% of students admitted to never having asked a question through the platform. You can find additional charts related to the survey in Appendix C.

5.2 Post-development User Interviews

We conducted five user interviews so that users can test out Tigerverse and describe their user experience. Four of the five students were Computer Science majors from Princeton University, while the other user was a student at Columbia University. The user interviews were qualitative in nature. They were given a task list (see Appendix B), and were asked to speak out loud while completing the tasks.

Overall, the users provided positive feedback. Especially, navigating through the task list was very straightforward according to most of them. The process of setting up the chatbot was also described as “very easy and straightforward”.

On the more suggestive side, some users mentioned that it was not clear whether a user should post a question on Tigerverse or interact with a chatbot to get their answers. Providing some guidance on the home page on when to post a question and when to use a chatbot would have been helpful, according to them. Another user mentioned that having the ability to directly email administrators of the chatbot via Tigerverse would add to the user experience. Most of the feedback users provided were valid and I hope to add these improvements before deploying Tigerverse to the campus community.

6. Future Work and Conclusion

Based on the feedback I received from user interviews, the immediate next step is to work on improving the user experience. Having a simple user guide upon the initial login should be a necessity as navigating Tigerverse is non-trivial. Furthermore, I plan to migrate the Tigerverse client from React.js to Next.js. Relying on the authentication, routing, and server-side rendering infrastructure that Next.js provides will make Tigerverse much more stable and easier to extend in the future.

On the subject of chatbots, it would be great to have the functionality for users to save their chatbot conversations so that they can reference them later. Furthermore, it would be great for admins of organizations to also have a history of past chatbot conversations so that they can assess how the chatbot performs, and especially, see if there are questions to which the chatbot does not have the knowledge to answer. Furthermore, having the ability for administrators to add data in the form of uploading pdf documents, web links and videos would make the process of uploading data much more smoother.

Overall, the AI-powered Q&A system we have built is largely successful. The AI-powered Q&A system we developed could be expanded to many other organizations that have sub-organizations under it. For example, a big corporation could use it so that new employees could navigate the different departments under the big corporation easily.

7. Acknowledgments

I would like to thank my adviser, Professor Robert Dondero for his advice, commitment and support throughout, the staff of the Independent Work Program at Princeton, and my fellow classmates, preceptors and professors that I have interacted with in my previous Computer Science classes. This independent work project provided me the platform to fearlessly combine new technology and test my

software engineering and system design skills. I have had a highly positive experience during my Independent Work project.

8. Bibliography

[1] “Anonymous apps Fizz, Sidechat compete for students’ attention,” The Princetonian. Accessed: Jan. 10, 2024. [Online]. Available: https://www.dailyprincetonian.com/article/2022/09/fizz-sidechat-compete-students-attention-anonymous

[2] C. Arthur, “What is the 1% rule?,” *The Guardian*, Jul. 20, 2006. Accessed: Jan. 11, 2024. [Online]. Available: https://www.theguardian.com/technology/2006/jul/20/guardianweeklytechnologysection2

[3] “Intersection Observer API - Web APIs | MDN.” Accessed: Jan. 11, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API

[4] M. B. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” Internet Engineering Task Force, Request for Comments RFC 7519, May 2015. doi: 10.17487/RFC7519.

[5] “Microsoft pushes the boundaries of small AI models with big breakthrough | Semafor.” Accessed: Jan. 11, 2024. [Online]. Available: https://www.semafor.com/article/11/01/2023/microsoft-pushes-the-boundaries-of-small-ai-models

[6] “What are AI hallucinations? | IBM.” Accessed: Jan. 11, 2024. [Online]. Available: https://www.ibm.com/topics/ai-hallucinations

[7] “Learn Next.js | Next.js.” Accessed: Jan. 11, 2024. [Online]. Available: https://nextjs.org/learn/dashboard-app

9. Appendices

9.1 Appendix A: Infinite Scrolling via Intersection Observer API

Infinite scrolling using HTML, CSS and Javascript

HTML

```
<div id="content">
  <!-- initial content goes here -->
</div>
<div id="load-more-trigger"></div>
```

CSS

```
#content {
  overflow: hidden;
}

#load-more-trigger {
  height: 20px; /* Height of the space that triggers a load */
  background-color: #f0f0f0; /* Set it to the color of viewport*/
}
```

Javascript

```
// Function to load more content
function loadMoreContent() {
  // logic to fetch and append more content
  const moreContent = document.createElement("div");
  moreContent.textContent = "Additional Content";
  document.getElementById("content").appendChild(moreContent);
}

// Intersection Observer configuration
const options = {
  root: null, // Use the viewport as the root
  rootMargin: "0px",
  threshold: 0.1, // When 10% of the load-more-content div is visible
};

const observer = new IntersectionObserver((entries) => {
  entries.forEach((entry) => {
    if (entry.isIntersecting) {
           loadMoreContent();
    }
  });
}, options);

// Start observing the trigger element
observer.observe(document.getElementById("load-more-trigger"));
```

How it works

We insert a <div> element with the id of “load-more-trigger” at the bottom of our data feed. In our example, the feed is contained by a <div> element with the id of “content”. Whenever the <div> with the id “load-more-trigger” appears visible in the viewport of the client, an intersection event is triggered. This causes the loadMoreContent() function to be invoked, which populates more content within the <div> element with the id “content”. When the user again reaches the bottom of the updated content feed, an intersection will be detected again and the loadMoreContent() function will be loaded which causes more content to be added to the <div> element holding all the content. This happens repeatedly until there is no more content provided by the server. Note that the code above currently has no mechanism to store the state. For example, we need to keep track of the number of items loaded on the client-side so that we can send a page number to the loadMoreContent() function. This could be done via global variables, by instantiating a counter() object, using local storage, etc. For a practical example of how infinite scrolling can be achieved look at the following implementation in React.js.

Infinite scrolling using React.js

Implementing infinite scrolling in react.js is non-trivial. A background knowledge of React and the useEffect hook, useCallback hook and the useRef hook is recommended.

```
import { useCallback, useEffect, useRef } from "react";
import { useQnaContext } from "./Contexts/qna";

  const {
    questions,
    loadMoreQuestions,
  } = useQnaContext();

  // ref to the target component to be observed for an intersection with the viewport
  const observerRef = useRef(null);

  // true = we are within a window where we should not send a new fetch request.
  const loadingBuffer = useRef(false);

//----------------------------------
// Function: Loads more qna items while maintaining a setTimeout
```

```
//-------------------------------------------------------------------
  const loadMoreItems = useCallback(async () => {
    if (loadingBuffer.current) {
      return;
    } else {
      loadingBuffer.current = true;
      // questions, and loadMoreQuestions are a list of questions and
a function to load more questions. These are provided by a Context
Provider. We access these two state variables via the useQnaContext
hook.
      await loadMoreQuestions(Math.ceil(questions.length /
10).toString());
      // Timeout to avoid multiple calls between a 200ms window
      setTimeout(() => {
        loadingBuffer.current = false;
      }, 200);
    }
  }, [loadingBuffer, loadMoreQuestions, questions]);

  useEffect(() => {
    // Initialize the Intersection Observer
    const options = {
      root: null, // The viewport
      rootMargin: "0px",
      threshold: 0.1, // Trigger when at least 10% of the target is
visible
    };

    const observer = new IntersectionObserver(([entry]) => {
      if (entry.isIntersecting) {
        // Target is in the viewport, load more items
        loadMoreItems();
      }
    }, options);

    // Start observing the intersection
    if (observerRef.current) {
      observer.observe(observerRef.current);
    }
    // Clean up and disconnect the observer when the component
unmounts
    return () => observer.disconnect();
  }, [questions]);

   ….

   return(
    <>
      <div id="content">
      <div ref={observerRef}>
    </>
   )
```

How it works

Similar to the example with HTML, CSS and Javascript, we declare an intersection observer. Instead of passing down the id of a div to the Observer.observe() function, we pass a ref. To declare a ref we use the useRef() hook, and we pass the result of useRef() to the component we want to reference. In our case, we passed the observerRef to the final <div> element that is below the content feed. Whenever an intersection is observed between this <div> element with the observerReg and the viewport, the loadMoreItems() function is executed. The loadMoreItems() function is wrapped under a useCallback() hook to prevent redefining the function whenever rerenders occur. Furthermore, whenever we call loadMoreItems(), it executes the function loadMoreQuestions() and sets a timeout until which the loadingBuffer ref is going to hold the value True. If loadMoreItems() is called again while the loadingBugger is true, the loadMoreQuestions() function is not executed. This is done to prevent, users from scrolling too fast and to avoid cases with browser malfunctions where the scrollbar automatically scrolls to the bottom.

9.2 Appendix B: Task List for User Interviewers

Task 1: You are a student wanting to know whether international students have to file taxes on their own or whether the university files on behalf of them.

Step 1: Log into Tigerverse with your netid.

Step 2: Click on the button “Ask a Question”.

Step 3: Type out your query in the title section and optionally, add context in the content section.

Step 4: Click submit.

Step 5: Observe that your new question was automatically added to the feed of Questions in your home page.

Task 2: You are another student who has filed taxes before as an international student. You log in to Tigerverse and see that a student has asked a question concerning taxes to which you know the answer. However, you want your answer to be anonymous.

Step 1: Log into Tigerverse with your netid.

Step 2:Click on the expand button on the right side of the question card to expand the question.

Step 3: Type out your answer on the text box dedicated for answers.

Step 4: On the left side of the text box, there is a selection box. The default option of the selection is “not anonymous”. Click on it and toggle the option to “anonymous”.

Step 5: Click submit or press enter.

Step 6: Observe that your new answer was added as an answer under the question. Your netid is not visible on top of your answer. Instead, it says “anonymous”.

Task 3: You are a student founder who wants to raise money for your student startup. You have heard of the Princeton Entrepreneurship Council (PEC) but you have no idea how they can help you with your startup.

Step 1: Log into Tigerverse with your netid.

Step 2:Click on the discover menu item on the navbar on top of the webpage.

Step 3: Click on the PEC Chatbot icon to enter the PEC chatbot.

Step 4: You can see the chatbot interface for PEC. Type “I am a student founder. How can the PEC help me?’ on the textbox visible.

Step 5: Wait for the chatbot to answer, and follow up the conversation with a response of your choice!

Task 4: You are the president of the Princeton Debate Panel (PDP). You have been authorized by the administrators of Tigerverse to set up a chatbot for PDP. You have been given a username and a password for the PDP chatbot.

Step 1: Log into the Tigerverse admin interface using the username and password given to you via the superadmin.

Step 2: You will initially see the dashboard page which should say that you have zero projects uploaded to your chatbot. A project is a name that holds a piece of data about your organization. For example, "How to join the club" could be the name of a project and you would have sentences describing the process to join the club as data under the project.

Step 3: Go to the manage datastore page by clicking on that tab from the sidebar, and click on the button "Add Data".

Step 4: Type the project name: "how to join PDP", a link to a website or a blog if it exists and the data of your project: "We hold trials at the start of every semester. An email will be sent to all listservs at the start of each semester including the details of the trials and a Google sheet to register for trials. Stay tuned for our Spring term trials.".

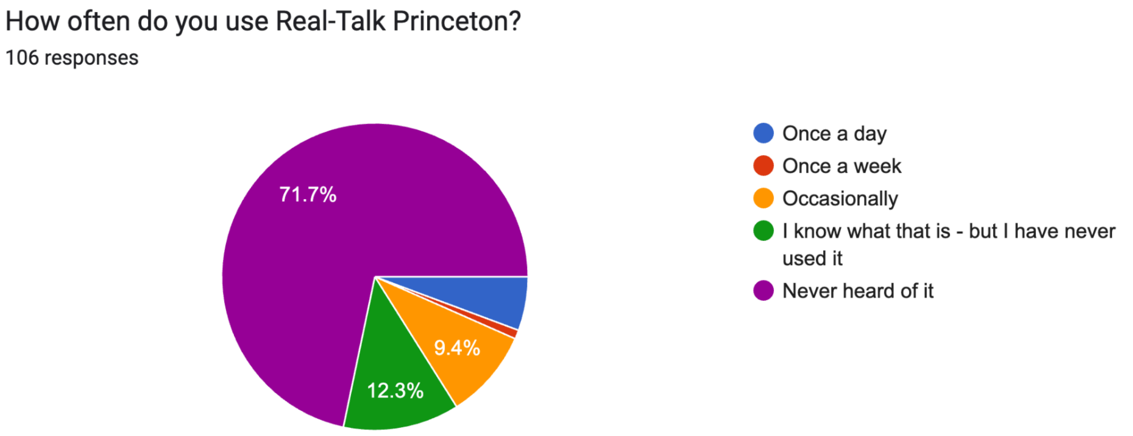
Step 5: Click on submit.

Step 6: Log back on to Tigerverse.

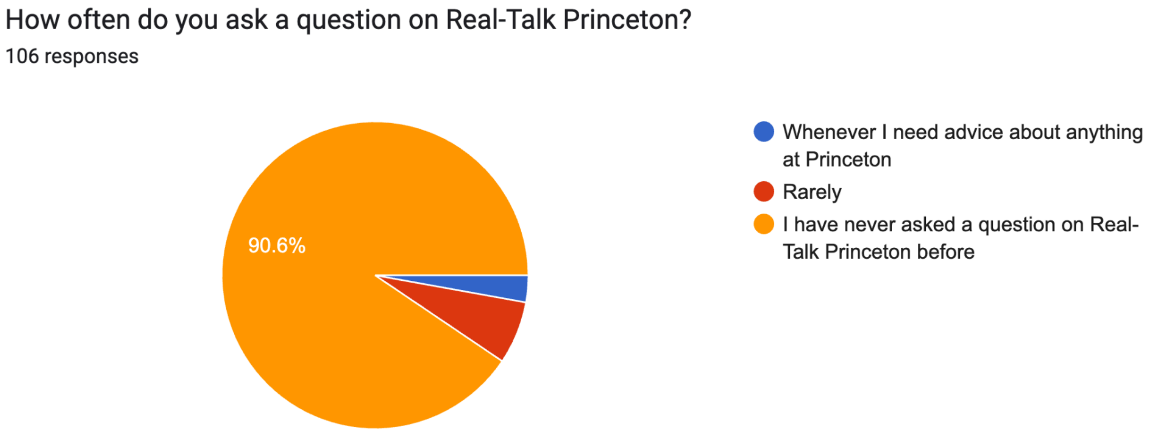
Step 7: Go to the discover section, and click on the PDP chatbot.

Step 8: Ask the question: "How do I join PDP".

9.3 Appendix C: Charts Related to the Pre-development Survey


How often do you use Real-Talk Princeton?

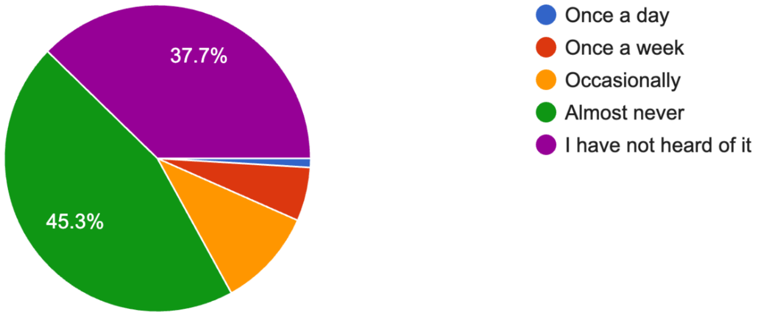
106 responses


How often do you ask a question on Real-Talk Princeton?

106 responses

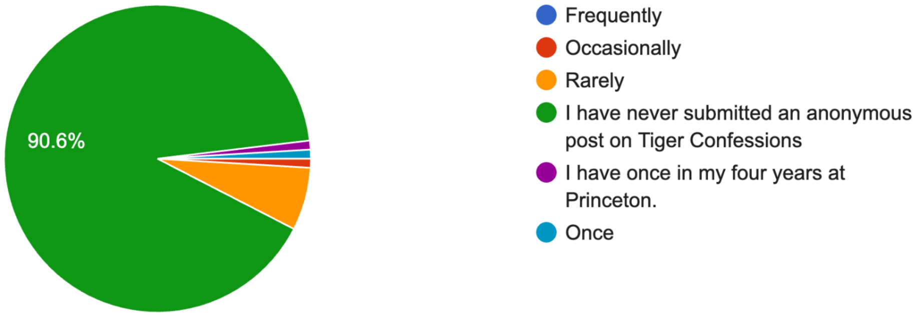
How often do you use Tiger Confessions?

106 responses



How often do you submit an anonymous post on Tiger Confessions?

106 responses


Frequently
Occasionally
Rarely
I have never submitted an anonymous post on Tiger Confessions
I have once in my four years at Princeton.
Once