Virtual Companionship Chatbot - Indy 13

Final Report

Course Number: CS4850

Semester: Spring 2023

Professor: Ms. Perry

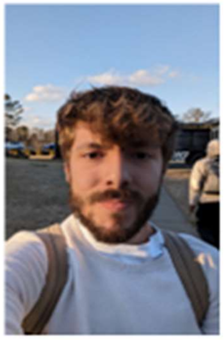
Github Link: https://github.com/NoahDClark/VirtualCompanionChatbot

Website Link: https://virtualcompanionchatbot.wordpress.com/

Submission Date: April 2024

Fabritzzio Cabrejos

Rakshak Nath Gurung



Noah Clark

Cesar Lucena

Number of Lines of code in the Project: 1360

Number of Components in the Project: 28

Contents

Introduction

In today’s rapidly changing digital world, an intriguing paradox has surfaced: as technology plays a bigger role in human connections, feelings of loneliness and isolation are also rising. According to recent studies, such as one by Gitnuss, there is a worrying trend: by their senior year, almost 77% of college students report feeling lonely and about 61.5% report having extreme anxiety. This emphasizes the critical need for creative solutions that improve human interactions while also bridging gaps.

The Virtual companionship chatbot project is our answer to this challenge; it goes beyond conventional AI chatbots. This mobile app uses sophisticated sentiment analysis to comprehend and relate to users, so it’s not just about conversations. Our chatbot is not like other chatbots; instead, it’s meant to be a dynamic emotional outlet that provides users with individualized sympathetic interactions that help them feel understood and appreciated. The chatbot creates a real human connection by customizing its responses based on a thorough analysis of the context of conversations. With this project, we hope to create a future in which technology does more than just link us; it also comprehends and supports us, strengthening our emotional connections at every turn.

Project Goals

The Electronic Partnership In the creation of mobile applications that enable meaningful human-machine interaction, chatbots mark a revolutionary leap. The central component of our breakthrough is our AI-powered chatbot, which has been painstakingly designed to offer comfort and company to those who are lonely. Below, we outline our main project objectives with the intention of optimizing the effectiveness and influence of this application:

1) Increasing User Engagement with Intuitive Design:
    a) Create a mobile application with an intuitive UI that makes interacting with the AI chatbot easier. The design will place a high value on simplicity of use and minimalist design concepts to promote ongoing interaction without tiring out the user.
2) High Levels of Emotional Intelligence
    a) Incorporate advanced algorithms for sentiment analysis and natural language processing (NLP). This will enable the chatbot to recognize and adjust to the emotional context of user messages, allowing for pertinent and sympathetic responses. The goal is to establish a more intimate relationship between the user and the chatbot by making every interaction feel sympathetic and personal.
3) Customization of User Experience:
    a) Make efficient use of data to customize interactions for specific users. Over time, the chatbot will adjust its responses to better reflect the user's preferences and

emotional state by storing a limited amount of data, such as conversation history and sentiment trends. To maintain a continuous, personalized experience, users will need to log in, guaranteeing that their interaction history is safe yet easily accessible.

4) Strong Security Procedures:
    a) To safeguard user data, strict security procedures and encryption are put in place. Strong access controls will be a part of this to guard against unwanted data access and guarantee the privacy and security of user interactions.
5) Thorough Testing for Performance and Reliability:
    a) To make sure the application is dependable and performant, thoroughly test it on a variety of hardware and operating systems. The application will undergo stress tests, usability tests, and performance evaluations to ensure that it can withstand different loads and offer a consistent user experience on all platforms.

Design Constraints and Considerations

The key factors and constraints that affected the creation of the Virtual Companionship Chatbot are described in this section. Comprehending these elements is essential for evaluating the functionality and extent of the application.

User Characteristics

Make sure it is user-friendly and accessible to all.

- Accessibility: The application is designed to be user-friendly and suitable for users of all ages and technological skill levels. The design philosophy of the app is centered around this inclusivity.
- User Interface Design: By placing a strong emphasis on user-friendly design and intuitive navigation, users can interact with the app with ease. Each interaction is made more relevant and personalized by the AI-driven interface, which adjusts responses in response to user inputs.
- Versatility: The app can produce engaging and contextually appropriate responses by utilizing AI's adaptive capabilities, which will increase user satisfaction and engagement.

System

Offer a stable, adaptable, and dependable platform.

- Adaptive Learning: The system uses a step-by-step learning process for users who are unfamiliar with the app. Over time, personalization is improved by using the context created by the initial interactions to inform subsequent responses.

- Cross-Platform Compatibility: The software, which was created with React Native, functions flawlessly on both the iOS and Android operating systems, guaranteeing that a large user base can access and utilize it.
- Data management: The app's ability to effectively personalize user experiences and preserve conversation continuity is made possible by a sophisticated database system that securely stores user data.

Assumptions and Constraints

Recognize and address any possible constraints.

- Network Dependency: A reliable internet connection is required for the application to function properly. To retrieve user-specific data from the database and to retrieve responses from the server, there must be continuous connectivity.
- Performance Sensitivity: The app's capacity to interpret and react to user inputs may be severely hindered in situations where the network is erratic or unavailable. The app's operational planning and user experience strategy both take this dependency into account.

Architectural Strategies

There are many design decisions that we made for overall standing architecture which affects the overall system organization and structure. Listed below are the different strategies and decisions we made for our project:

a. **Programming Language:** While other programming languages such as C++ may provide a more efficient and powerful coding environment, the machine learning libraries which we intended to employ throughout our project were primarily available in Python programming. Consequently, the AI chatbot was implemented and was coded and trained in python. However, the UI and other functionalities were coded in ReactJS.

b. **Machine Learning Libraries:** As stated in the previous point, python is rich in machine learning libraries, hence our project implemented NLP algorithms from libraries such as NTLK, spaCey and tensorflow.

c. **Database Management System:** For the development of our machine learning system, FireBase was used. While initially we were planning to work with MongoDB, FireBase provided easier access as well as reliability to work on.

d. **Memory Management:** Since the overall system was centralized given the role of the server in our project, memory management wasn’t as important. However, maximizing system efficiency was still important for increasing the speed and response time of the overall system. Therefore, efficient retrieval of information and storing of information in the cache through lazy loading was used for minimizing startup latency.

e. **Network Communications:** Our project is divided into 2 servers primarily, the east and the west. This was so that if the server on the east side lost connection, the west side server could cover for the clients on the east side and vice versa. Initiallly, we approached the project with a simple client-server protocol, but we slowly moved onto a new stateless protocol which can handle multiple clients concurrently. While it is still in development, most of the stateless protocol has been implemented in the project we submitted.

System Architecture

Our mobile application's system architecture, which includes AI chatbot and sentiment analysis functionality, is designed to allow for seamless and productive interaction between users and the chatbot while allowing for scalability, flexibility and maintainability. Since it is a high-level architecture, it was divided into multiple key components/subsystems of which each of them oversees a specific functionality or interaction with the application. The sub-system architecture are as follows:

a. **User Interface Component:** The User Interface Component serves as the application's front-end, presenting the chatbot interface to users in an intuitive and visually appealing manner. It includes features like logging in, talking with the chatbot, looking at past conversations. Since it is implemented with ReactJS, it is compatible for both IOS and Android platforms. There is easy access and flow in the chatbot, but also to improve it, we had ideas like voice commands and screen reader to improve the usability for people with disabilities (but this is still in work).

b. **Chatbot Engine:** The chatbot engine is the main component that handles the user message, implements natural language input and then generates appropriate responses. It uses advanced natural language processing (NLP) algorithms such as tokenization, part-of-speech tagging, and named entity recognition to extract meaning from user input. Machine learning models, such as recurrent neural networks (RNNs) and transformers, are used to analyze user sentiment and customize responses. The chatbot's knowledge base, which includes pre-defined response templates and conversational rules, is constantly updated and improved based on user interactions and feedback.

c. **Backend Server Component:** The Backend Server Component serves as a bridge between the client application and external services, managing data storage, retrieval, and communication. It manages user accounts, stores conversation history, and securely saves application settings. RESTful APIs are used to facilitate communication between the client application and the backend server, resulting in seamless data exchange. The backend server uses strong security measures such as encryption, authentication, and access control to protect sensitive user data and ensure compliance with privacy regulations.

d. **External service integration:** To build up on our already packed up project, the external service integration component in our project communicates with external services such as cloud-based NLP APIs and sentiment analysis libraries which only help to improve the chatbot, which makes it more scalable and flexible for the upcoming users. Furthermore, techniques such as asynchronous processing techniques like message queues and webhooks were used to make better communication with external services.

e. **System Collaboration:** Everything inside of the application is connected to each other which makes it so that everything is collaborating with each other due to which the app runs smoothly and efficiently. The Chatbot Engine communicates with the Backend Server Component to retrieve user data, update conversation history, and save application settings permanently. The External Services Integration component works alongside the Backend Server Component to securely access external services and retrieve additional data or resources as needed.

f. **Rationale:** The system was decomposed into these components/subsystems to achieve a clear separation of concerns while also promoting modularity and maintainability. Each component is intended to encapsulate specific functionalities, enabling independent development, testing, and deployment. This architecture allows for scalability and flexibility, as each component can be scaled horizontally or vertically based on demand, and new features can be added or modified without disrupting existing functionality.

Requirements

Functional Requirements:

The following specifications outline the actions and qualities that the Virtual Companionship Chatbot needs to possess:

User Interaction Management:

- Capability: The program must be able to manage user interactions effectively, such as messaging chatbots and safely storing login information.
  Details: To handle ongoing interactions without necessitating repeated logins, ensure secure input fields for user authentication and implement session management.

Natural Language Processing:

- Capability: Make accurate use of NLP methods to decipher the user's intentions from their messages.
- Details: To create responses that are logical and appropriate for the given context, use sophisticated natural language processing (NLP) algorithms that can comprehend sentiment, context, and user intent.

Response Generation:

- Capability: To provide prompt and pertinent answers to user inputs.
- Details: Make use of trained models to guarantee that responses are timely and tailored to the user's preferences and ongoing conversation history.

Personalization:

- Capability: Offer individualized options for user interaction depending on user activity and previous exchanges.
- Details: To improve user engagement, use data analytics to monitor user preferences and adjust the chat interface and responses appropriately.

Session management and authentication:

- Capability: Confirm user identity to preserve session integrity and offer a smooth experience over several sessions.
- Details: To guarantee that users can safely continue interacting, use encrypted session tokens and multi-factor authentication.

Non-Functional Requirements:

The qualities that the system needs to fulfill are specified by these requirements:

Security:

- Goal: Uphold the highest standard of security when transferring and storing data.
- Implementation: To transmit data securely, use TLS/SSL protocols. Employ OAuth for safe user authentication and implement strong encryption for data that is stored.
- Compliance: Verify that every security measure complies with applicable laws and industry standards, including the GDPR for users in Europe.

Scalability and Capacity:

- Goal: Support numerous users at once without sacrificing performance.

- Implementation: When user load rises, add more servers or instances by designing the backend to scale horizontally. For elastic scalability and storage expansion, make use of cloud services.

Usability:

- Goal: Make sure that a wide range of users, regardless of technical proficiency, can easily access and utilize the application.
- Implementation: Pay close attention to a responsive design, clear navigation, and a minimalist user interface. To improve usability and get feedback, test your product with actual users.

Compliance and upkeep:

- Goal: Comply with relevant privacy laws and make sure the program is kept up to date to satisfy changing user requirements and security requirements.
- Implementation: Update the program frequently to fix security flaws, enhance functionality, and add new features in response to user feedback. By adding the required privacy features and conducting frequent audits, you can maintain compliance with privacy laws like the CCPA and GDPR.

Software Development Life Cycle (Narrative discussion)

The Software Development Life Cycle (SDLC) is a rigorously structured process that guides the development of the Virtual Companionship Chatbot and guarantees methodical progress through multiple stages, from conception to deployment. The special difficulties presented by fusing cutting-edge AI technology with user-centric mobile application design have been considered in the creation of this SDLC.

Phase 1: Requirement Analysis

We began the project by conducting a thorough analysis of the real needs that our stakeholders and potential users had for our chatbot. We obtained the hard data and anecdotes to fully understand their expectations through in-person interviews, and engaging group discussions. Everything we discovered was meticulously recorded, and we verified that it all matched our goals for the project. The foundation for our system's design was created during this stage.

Phase 2: System Design

After gaining a firm understanding of our needs, we entered the design stage and started shaping the software's architecture. Our group decided on the database schema, worked out the finer points of the design, and created a high-level structure. Choosing the appropriate tools and technologies to meet all user and technical requirements while bringing our chatbot to life was the focus of this stage.

Phase 3: Implementation

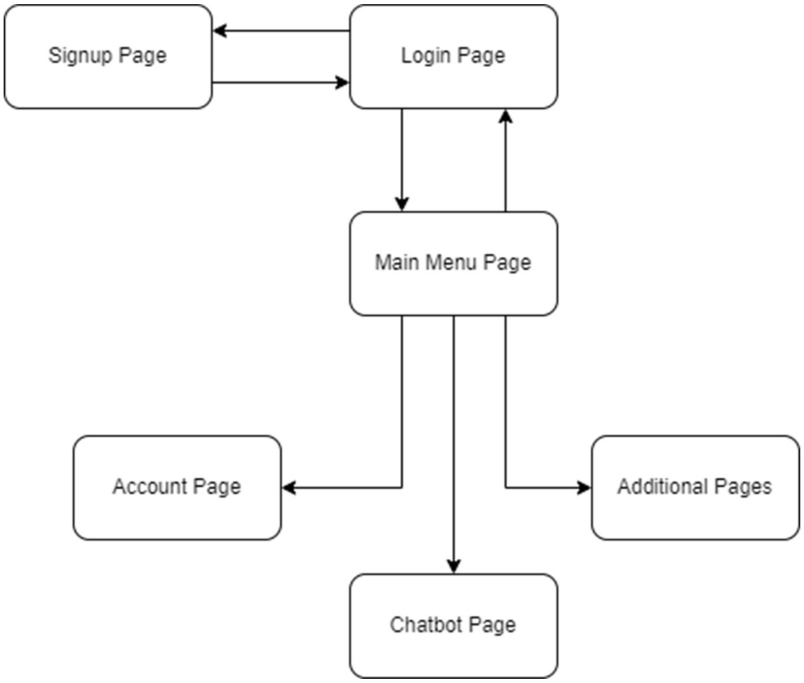
Subsequently, we executed our designs in the phase of implementation. Our developers created code based on documents and diagrams in sprints. New features and advancements were added with every cycle, and frequent review meetings kept everything on course. Throughout this process, we made a point of adhering to our design philosophies and quality standards.

Phase 4: Testing

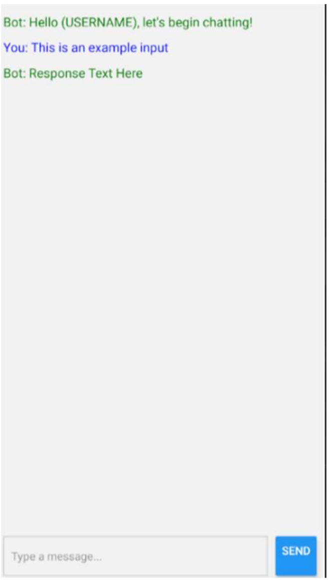
We thoroughly tested the chatbot before launching it to make sure it performed as intended. This phase was all about doing the right things and crossing the right ts: unit testing each component, checking that all the pieces fit together perfectly in integration tests, and lastly making sure actual users could use it without any problems in acceptance tests.

Software Development Life Cycle (UI & DBMS)

For the UI, it was determined early on that the most effective method of developing the software would be a software development life cycle which consisted of creating new pages, performing testing on existing and new elements, and repeating until the majority of the functionalities of the app are complete. This would produce a steady development of new features while also ensuring high product quality with minimal bugs. However, before beginning, we made a diagram depicting the core pages which our app will need to have minimal functionality.



With this diagram in mind, we started this software development life cycle by building the UI around the chatbot. As we are creating a screen for the chatbot to function and building the rest of the app around it, the first UI page created consisted of developing a chatbox which the user could interact with, and the AI could reply to. Since the large language model was not complete at this point in the project, temporary methods were implemented as placeholders until integration with the rest of the project was initiated. Pictured below is the first prototype of the page which the user will be using to interact with the chatbot.

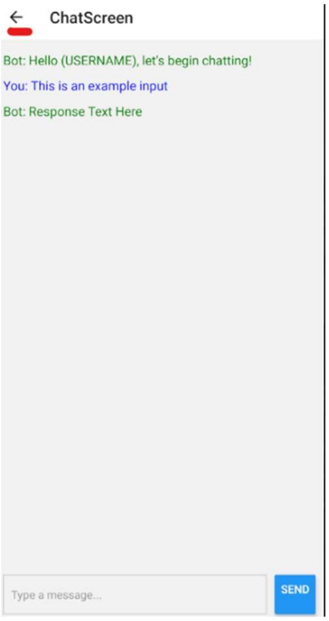


Bot: Hello (USERNAME), let's begin chatting!

You: This is an example input

Bot: Response Text Here

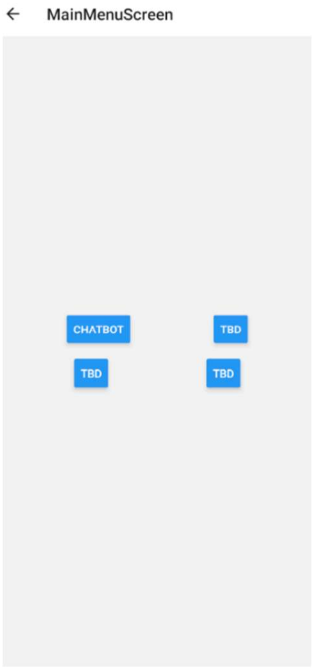
After the chatbox screen, we began developing the additional functionalities by implementing a main menu screen. Since this is still the prototype product, we created a barebones main menu with four buttons as place holders which will later connect to other pages. Furthermore, a back button was implemented through using the navigation feature included in react native. This would allow the app to jump from page to page and keep track of which pages the user has navigated to.



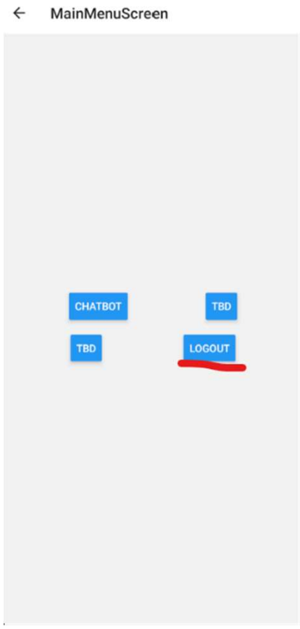
Bot: Hello (USERNAME), let's begin chatting!

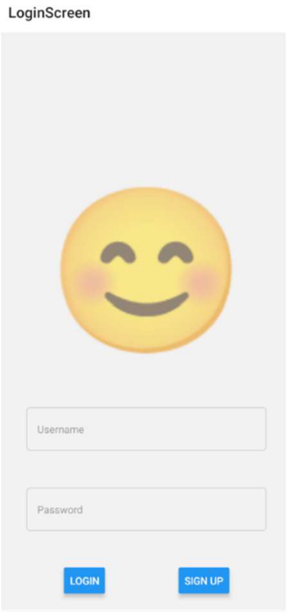
You: This is an example input

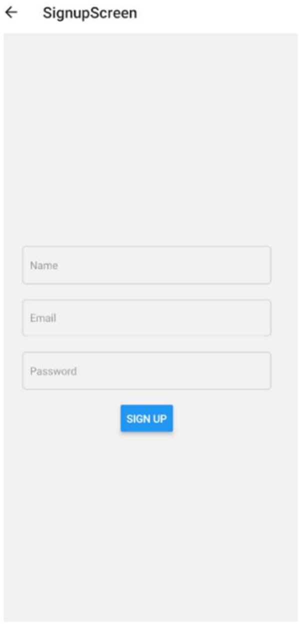
Bot: Response Text Here



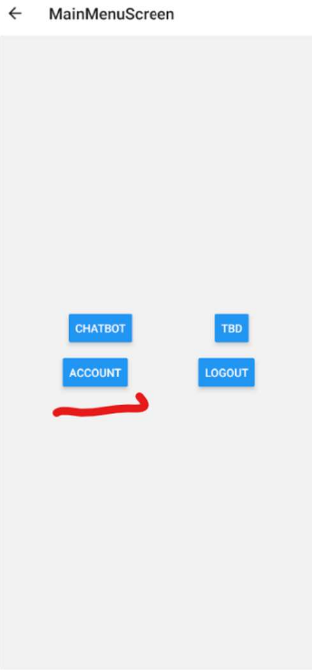
From there, we determined the next step was the development of the user login page so that we would have a basis to begin setting up the database system for managing user logins. Therefore, a login page was set up which would be the start page and show up prior to the mage menu screen. The establishment of a login page also necessitated the creation of a signup page, which was also created at this point in the cycle. As previously stated, performed testing to ensure that the new features (signup and login page) functioned as intended and we also ensured that old features, like the main menu page, correctly linked to the newly established pages as pictured below.

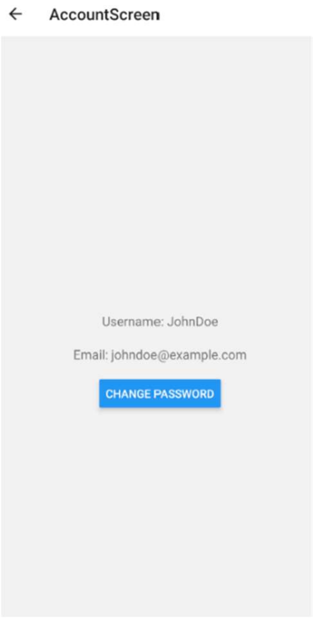






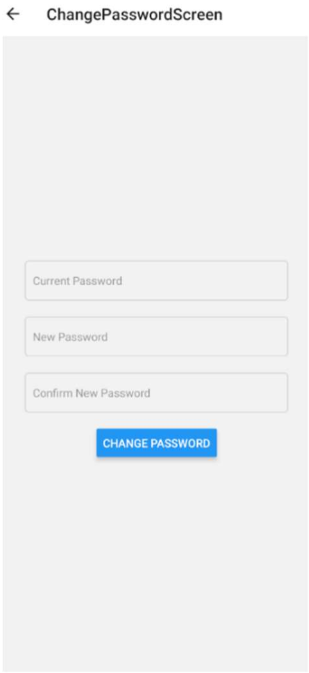
From here, we continued with working on the other aspects of the project which will require the DBMS. However, since the DBMS has not yet been established, the functions which are in place for the buttons are still just placeholders. So, for example, the login and signup button don't yet verify or create an account. With this in mind, we created an account page which contains user info and a button to reset password. This button will later be added to the login page when functionality is fully established.



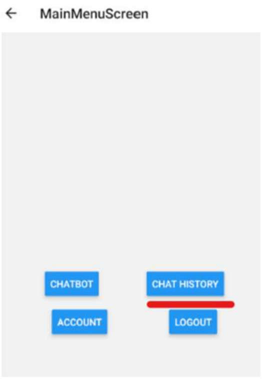


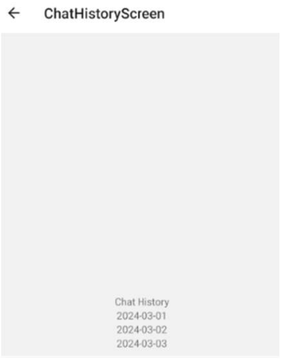
Username: JohnDoe

Email: johndoe@example.com



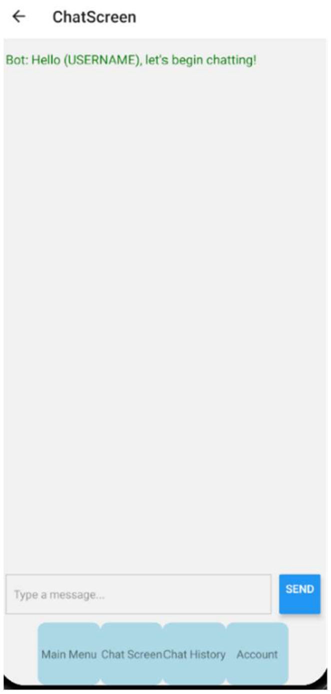
After this page, we discussed with the team if the fourth button available on the main menu screen was needed and determined that we would like to have a chat history page. Ideally, we would like this page to feature sentiment analysis that can provide the user into insights of their chats with the bot. However, for now, it will function more literally and just show chat history and dates. As previously mentioned, the DBMS is still not setup at this point, so the page just uses placeholders.



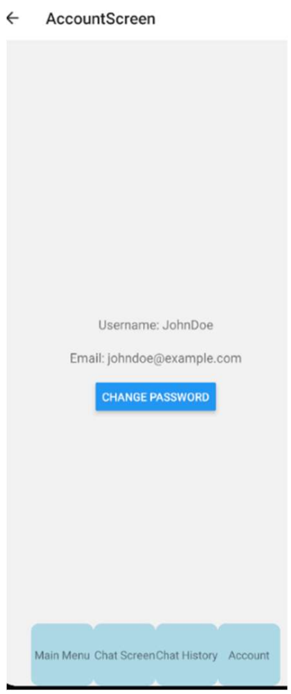


Chat History
2024-03-01
2024-03-02
2024-03-03

At this point, we branched out our development cycle into two: 1. Working on the DBMS since all the features which require it have been established and 2. Continuing to work on the UI and make it look cleaner for the user experience. However, before making aesthetic changes to existing buttons, one last addition has been made to improve user functionality. This is the addition of page buttons at the bottom of every page.



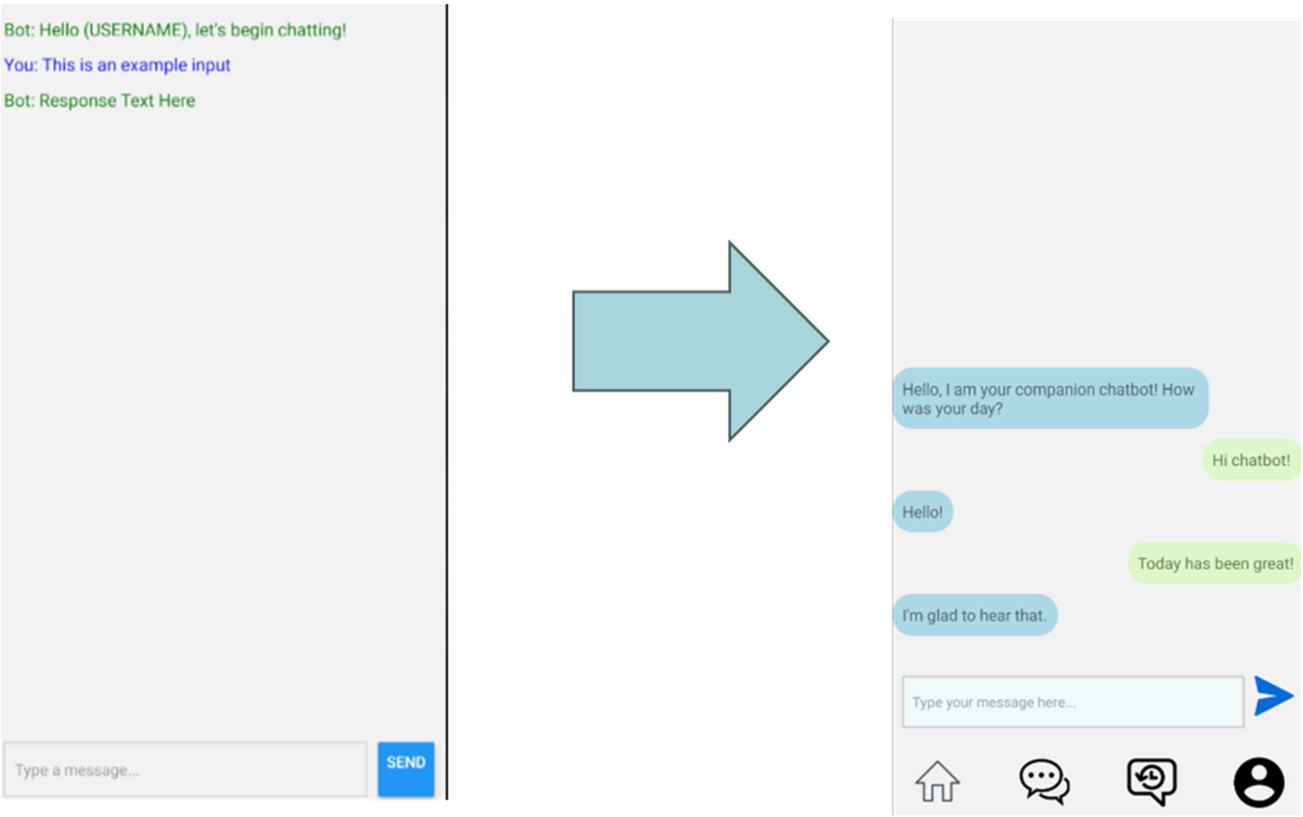
Bot: Hello (USERNAME), let's begin chatting!



Username: JohnDoe

Email: johndoe@example.com

From here, the rest of development prioritized cleaning up the UI and making the experience more friendly and appealing to the user. So for this we started with the chat page since the user will be spending much of their time there. This meant redesigning the chat messages as bubbles which appeared at the bottom of the screen along with a redesign of the buttons and color scheme.



Bot: Hello (USERNAME), let's begin chatting!

You: This is an example input

Bot: Response Text Here

Hello, I am your companion chatbot! How was your day?

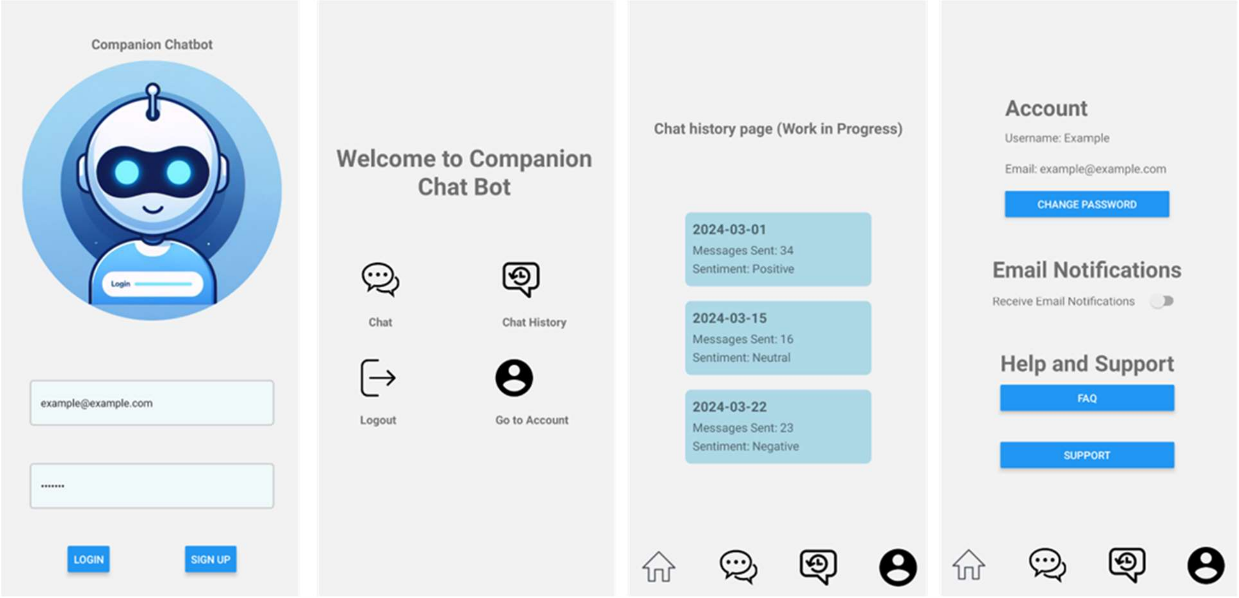
Hi chatbot!

Hello!

Today has been great!

I'm glad to hear that.

From here, we enhanced the rest of the user interface through fixing the color scheme and adding icons similar to the improved chat screen shown above. In addition to this newly developed scheme, we made minor functional modifications by adding an FAQ, SUPPORT, and Receive email notifications button to the account page.



example@example.com

•••••••

Welcome to Companion Chat Bot

Chat history page (Work in Progress)

2024-03-01
Messages Sent: 34
Sentiment: Positive

2024-03-15
Messages Sent: 16
Sentiment: Neutral

2024-03-22
Messages Sent: 23
Sentiment: Negative

Account

Username: Example

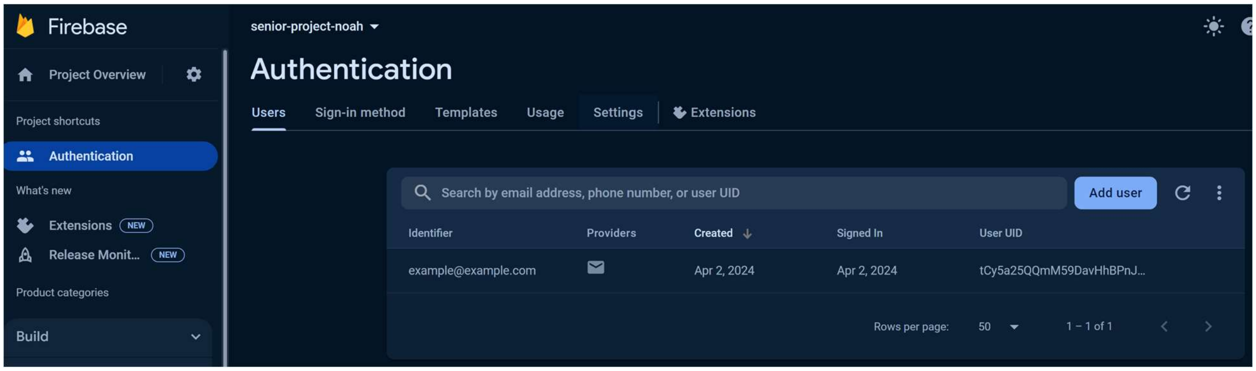
Email: example@example.com

Email Notifications

Receive Email Notifications

Help and Support

This essentially concludes the development of the user interface, and any additional changes made to the UI was done so to accommodate either the LLM or DBMS. However for the most part, neither required any considerable modifications. For example, the DBMS only required an additional box to account for the creation of new accounts. This is shown below in the firebase dashboard below.

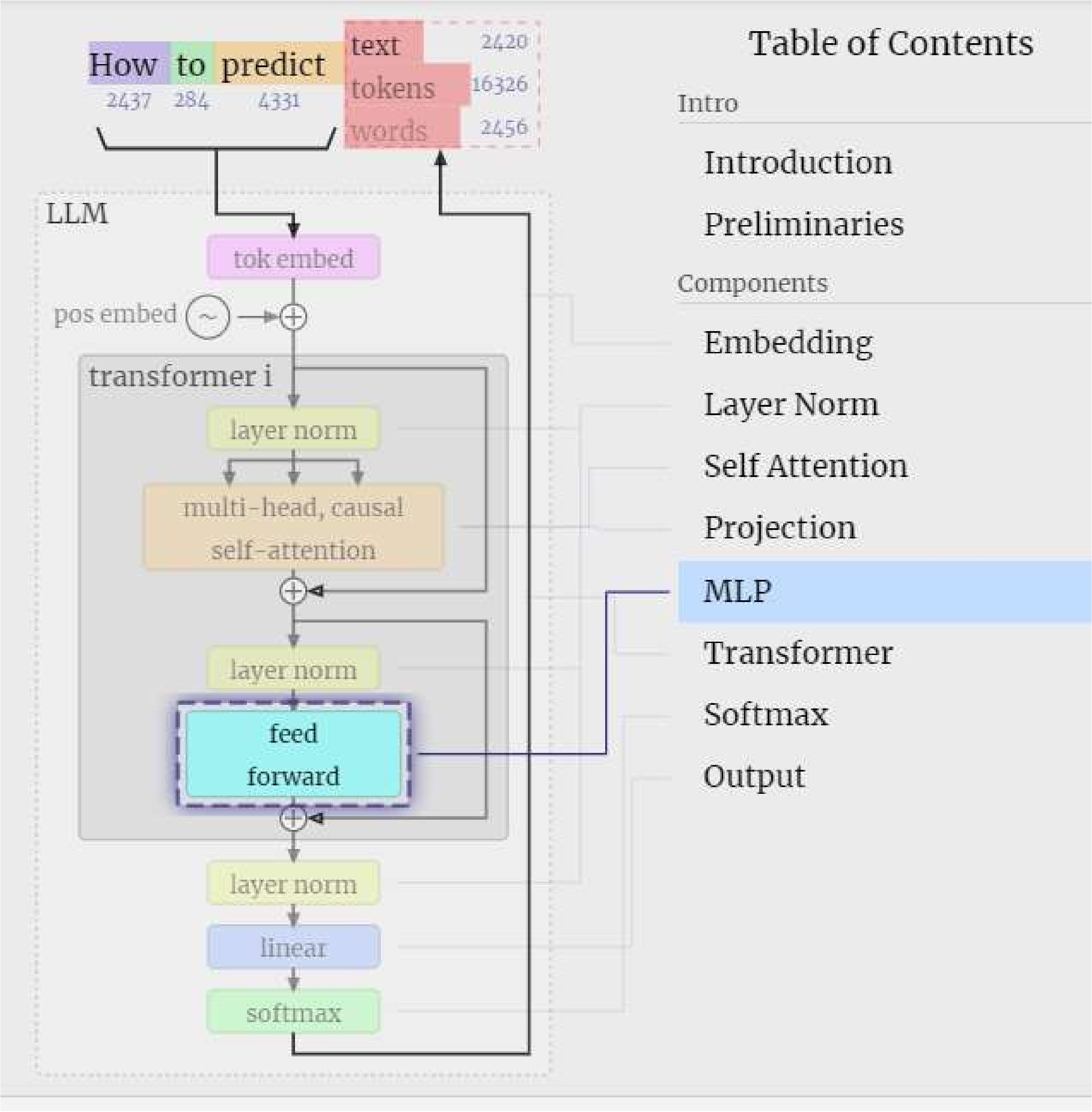


Identifier	Providers	Created	Signed In	User UID
example@example.com	✉	Apr 2, 2024	Apr 2, 2024	tCy5a25QQmM59DavHhBPnJ...

Setting up the DBMS was fairly straightforward, as the primary purpose of it was to setup an authentication system to verify user login. We designed this system to contain and store user chat information for a short period of time to provide the LLM with contextual information. However, given computational and resource constraints, this portion of the project was withheld from the final functioning product.

Software Development Life Cycle (LLM)

Recognizing the richness of interaction that the Large Language Model (LLM) can provide requires an understanding of it. Let's go over the parts and procedures that are essential to processing and reacting to user input, as shown in the attached image.



- Token embedding: In this first step, the input text is divided into tokens (which you can think of as words or subwords) that the model can comprehend. The semantic meaning of the text is then captured by an embedding, a numerical representation of each token.
- Positional Embedding (pos embed): To provide the model with context regarding word order, positional embeddings are added in addition to token embeddings. This is important because a sentence's meaning can be entirely altered by its order.

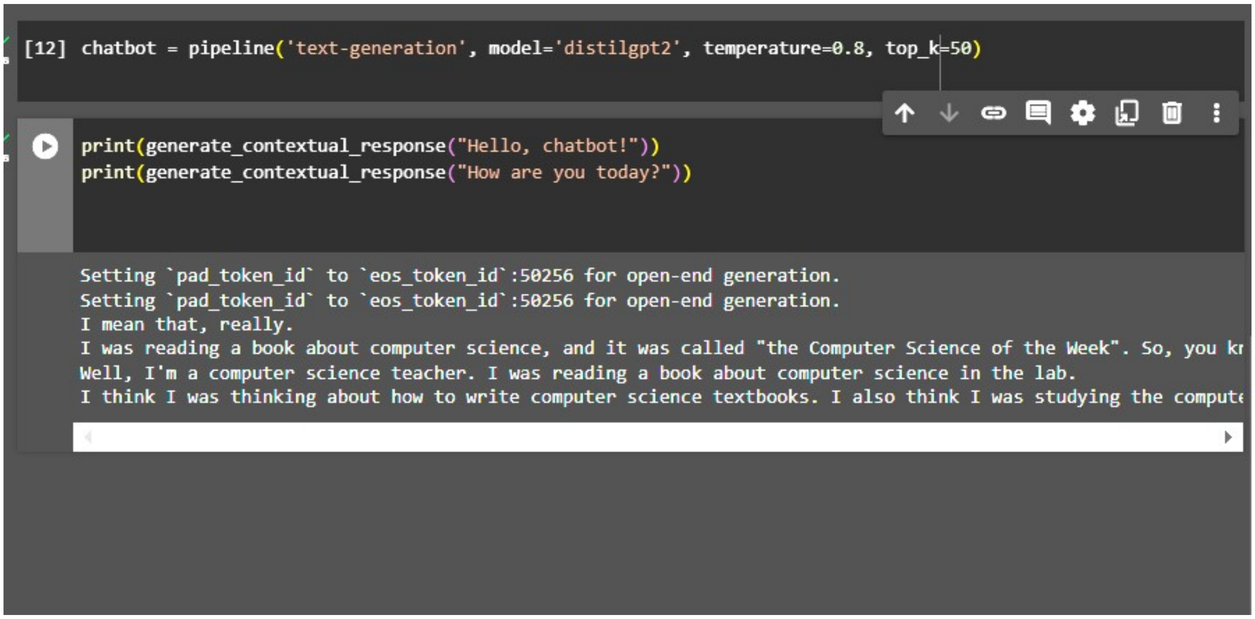
- Transformer Layer (transformer l): The transformer layers comprise the LLM's core. These are complex structures that go through several sub-layers to process the embeddings, such as:
    - The layer norm, or layer normalization: This sub-step helps to stabilize and expedite the learning process by standardizing the data within each layer and guaranteeing a consistent number scale.
    - Causal Self-Attention in Multiple Heads: In this case, the model looks at various segments of the input sequence to identify the words (or tokens) that are most pertinent to the ones it is presently processing. The 'causal' component makes sure the model only examines tokens that have already been entered, not ones that it hasn't seen yet in the future.
- Feed-Forward Neural Network: Following the attention mechanism, a feed-forward neural network processes the data. This network predicts what will happen in the text next by using weights that it learned during training.
- Output Generation: Lastly, the last layer norm normalizes the data before sending it to a linear layer, following several transformer layers (the number may vary based on the model's complexity).
    - A softmax function that transforms the numerical data back into probabilities for every potential token comes after the linear layer. In essence, the output is a prediction of the next word or text by selecting the token with the highest probability.

Like a gardener sowing seeds in a rich field of digital soil, we embarked on our LLM journey together. Much learning and listening took place in the early going, much like when you prepare the ground for planting. We sought to comprehend the kinds of conversations our users longed for, the kind words they would whisper to a friend, and the consoling answers they would receive. Stories and technical requirements were gathered like precious gems during this phase of our requirement analysis.

We went to the drawing board armed with a basket of insights. Our canvas was System Design, on which we meticulously mapped out the neural pathways necessary to convert text into meaningful dialogue, thus creating the framework of our LLM. We looked at the architecture here, which was intended to make our LLM not just smart but emotionally intelligent.

Developing Capabilities: Our LLM's role grew as the project did. By adding sentiment analysis capabilities, we were able to upgrade it from a basic conversationalist to a perceptive friend that could recognize and react to the emotional overtones in user messages. This change represented a significant milestone in the chatbot's lifecycle, extending its capabilities beyond information retrieval to include emotional support.

The Implementation phase then arrived, during which our plans began to take shape. As if they were composers at the piano, our team members' fingers danced over keyboards, coding away to produce a symphony of algorithms that would enable our chatbot to communicate with grace and empathy.

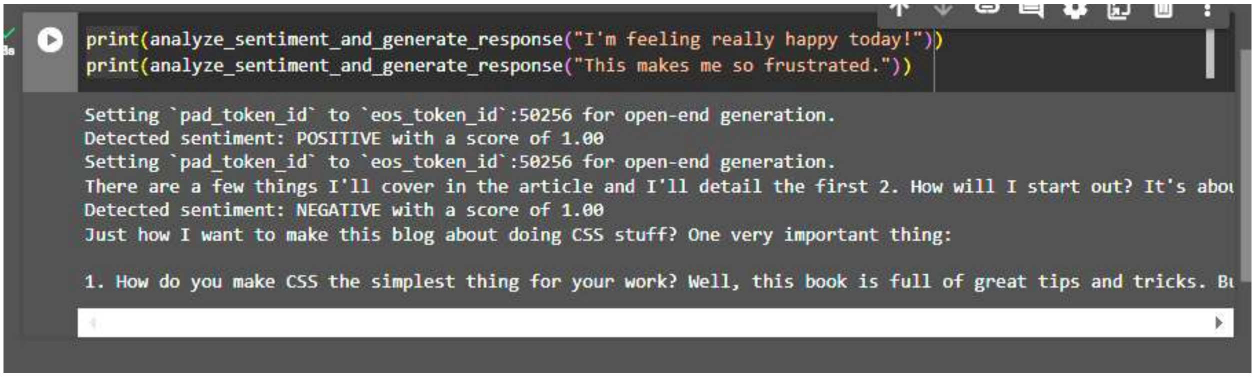


```
[12] chatbot = pipeline('text-generation', model='distilgpt2', temperature=0.8, top_k=50)
```

```
print(generate_contextual_response("Hello, chatbot!"))
print(generate_contextual_response("How are you today?"))
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
I mean that, really.
I was reading a book about computer science, and it was called "the Computer Science of the Week". So, you kr
Well, I'm a computer science teacher. I was reading a book about computer science in the lab.
I think I was thinking about how to write computer science textbooks. I also think I was studying the compute
```

Sentiment analysis implementation: The first step in the update was adding a sentiment analysis module that was designed to identify user emotions based on textual cues. This was integrated with the LLM so that the chatbot could now determine the emotion in a user's message—be it happiness, sorrow, or annoyance—and adjust its responses to offer consolation or acknowledge accomplishments appropriately.



```
print(analyze_sentiment_and_generate_response("I'm feeling really happy today!"))
print(analyze_sentiment_and_generate_response("This makes me so frustrated."))
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Detected sentiment: POSITIVE with a score of 1.00
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
There are a few things I'll cover in the article and I'll detail the first 2. How will I start out? It's abou
Detected sentiment: NEGATIVE with a score of 1.00
Just how I want to make this blog about doing CSS stuff? One very important thing:

1. How do you make CSS the simplest thing for your work? Well, this book is full of great tips and tricks. Bu
```

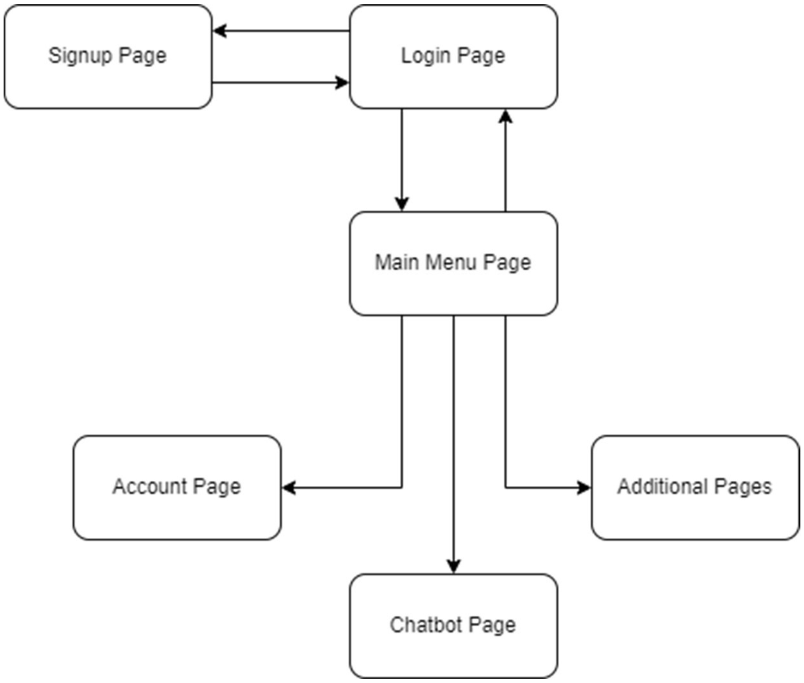
However, what would a creation be without some testing? Our testing process was thorough, exacting, and essential. We put our inexperienced chatbot through every possible situation, pushing it to comprehend and react with the same complexity and warmth as a human. This was about fostering growth, not just about locating bugs.

At last, we launched with a mixture of nervous excitement and anticipation. Not only was deployment a technical procedure, but it also marked the time we unveiled our virtual friend to the world and watched as it began to form bonds and kindle conversations.

Version Control

We opted to utilize GitHub for version control due to its robust collaborative features, however it wasn’t integrated thoroughly until the later stages of our project. Reason being, we focused on developing the LLM and UI components in parallel, with the intention of integrating them after a sufficient level of completion had been reached. However, due to time constraints and our focus on reaching project deadlines, we over-relied on local version control systems and a meticulous system of manual backups to manage our codebase. Although we realized it from the start, the project quickly progressed to the point where our need for collaboration and version tracking became much greater, which is when we made the decision to switch to GitHub. After this point, we leveraged the abilities of concurrent work to simplify our version control processes. This enriched our software development workflow and increased project management efficiency.

Screen Mockups



This screen mockup essentially depicts the flow of information within the app. So for example, the login page acts as the gate keeper which ensures that the user is authenticated or signed up before entering the app. Afterwards, the main menu page acts as the gateway to the rest of the application, such as the chatbot page, account page, and so on.

Server Architecture

In developing the architecture, it was decided to focus on ensuring that there would always be a high likelihood that the application would be available to retain users because outages tend to drive users to other platforms. As a result, for production deployment of this application we planned for a high availability (HA) architecture. For this we would deploy the app across two Azure cloud servers with one strategically being on the east coast and the other being on the west coast. Doing this will reduce latency across the U.S, where our main target audience is located, and increase reliability in the event of regional outages or natural disasters.

The servers planned for hosting are the NC6s_v3 Azure Cloud servers. These have the memory and GPU computational resources that would be required for hosting the Python server hosting the LLM used by our application for the chatbot functionality. These Python servers will integrate with two Firebase Realtime databases configured for data replication to prevent the complete data loss of the data necessary to customize the individual user experience.

It is important to avoid focusing too heavily on the technical requirements of the architecture design because each deploying applications in the real world always have monetary costs associated with scaling the application. Our deployment strategy has also taken finances into consideration. By using Microsoft’s freely available Azure Pricing Calculator we were able to determine the monthly cost of running the dual server setup would be $4870.56. Additionally using Firebases equivalent cost estimator, the Database (DB) servers would cost an additional $200. This brings our total monthly costs to $5070.56. This deployment would allow us to handle a capacity of approximately 2,400 requests per minute to support approximately 22,000 users signed up for the application bases on our usage frequency estimates.

While this would be the ideal setup assuming we would be able to grow the application's user base rapidly through some advertising campaigns. This could be an unlikely scenario. This is why we have a second architecture planned for the early stage of the application where we may have a very small user base which the first architecture would be too much. We would deploy one azure cloud server of the same type with a single DB on the east coast of the U.S. This would have reduced latency compared to self-hosting the server as well as halving our costs compared to the dual server setup. The downside being it would reduce the capacity to 1,200 request per minute only support 10,000 users. This would be an efficient cost saving measure which would not pose major issues due to Azure's ability to quickly deploy cloud servers if we notice the user base approaching 10,000 users.

Test Plan and Report

Using the backend tool for testing the LLM before it was connected to the UI, we were able to use the following data set to test if it was able to detect the correct emotion for the prompt associated with the emotion. The process we followed was to feed it a prompt on the backend and then we ignored the response the LLM gave but took note of the emotions it detected in the prompt it was given. This methodology was chosen since widely testing an LLM can be costly as shown by the current leaders in the space OpenAI.

Evaluating the quality of responses is typically done at-scale by employing a large team of humans. If we wanted to test the LLM on a larger scale, and were provided with the appropriate budget, we would leverage Amazon's Mechanical Turk service. This is a service which offers a low-cost solution to getting a very large number of very small tasks completed by real humans compared to employing a workforce of employees yourself. This method would allow us to further train and reinforce the current model we have trained. Additionally with further time and resources there are several training benchmarks we could employ for further testing the LLMs ability to read emotions.

Data set:
Joy: "I just got promoted at work and I'm ecstatic about the new opportunities!"

Sadness: "I lost my favorite necklace today, and it's left me feeling really down."

Anger: "Someone just cut me off in traffic, and it made me incredibly angry!"

Fear: "I heard a noise downstairs in the middle of the night and felt terrified."

Surprise: "I can't believe they threw me a surprise party for my birthday!"

Disgust: "There was mold on the bread I almost ate; it was disgusting."

Trust: "My best friend always supports me, which makes me feel deeply trusted and secure."

Anticipation: "I'm looking forward to the concert next month; I've been excited for weeks!"

Sadness: "Watching the end of that movie always brings tears to my eyes."

Joy: "Winning the championship was the happiest moment of my life."

Conclusion

As we conclude the final report, we have demonstrated that our project has made progress in addressing the growing issue of loneliness among college students. Through our project we created a sophisticated chatbot that is not only able to respond to interactions but through sentiment analysis is able to categorize the emotions being used in the conversations it is having. This allows our chatbot to augment its responses to account for the emotional nuances of user's messages.

With 1360 lines of code we took our project and transformed it from a concept thought up during the first few weeks of the semester and transformed it into several components which each operated separately before finally having our completed project where all the individual components operate together seamlessly.

Not only have we successfully developed a functioning react native mobile application, but we have also developed a mobile application that is designed to scale into the future. We have designed our application in a way that we can add new features to improve experience, take advantage of modern cloud computing to hyperscale our application, and scale the fine tuning of the LLM's responses along with the resources available to us.

In conclusion, our project, Virtual Championship Chatbot, has produced an innovative way for people to interact with technology for their benefit. Our project has not only met the initial goals of the initial goals but has been setup to have a foundation for future enhancements that can continue to improve the benefits that people could receive from our application.

React Native Training Proof

This section certifies that the undersigned individuals have successfully completed the designated training in React Native.

Participant Names and Signatures:

1. Fabritzzio Cabrejos: __________Fabritzzio Cabrejos_____________
2. Cesar Lucena: ___________Cesar Lucena ____________
3. Rakshak Nath Gurung: ____________Rakshak Nath Gurung ___________
4. Noah Clark: ___________Noah Clark ____________

Date: _________4/27/2024______________