Kennesaw State University

CS 4850, Section 02, Spring 2024

SP-15: Chat Bot

Final Report

Team members:

David Chavarro

Ethan Byrd

Aimi Tran

Matthew Fincher

Professor: Sharon Perry

April 27, 2024

Total Lines of Code: 33,375

Total Components: 42

Website link: https://aletheianomous-ai.github.io/

GitHub link: https://github.com/Alethianomous-AI

Table of Contents

1.0. INTRODUCTION

1.1. Overview

The Artificial Intelligence (AI) Chatbot project aims to create an intelligent chatbot that is accessible through web interfaces. The chatbot is designed to provide accurate information ethically, aligning with human values, and may offer citations to support its responses.

1.2. Project Goals

Develop an intelligent chatbot capable of engaging users in text-based conversations through a web interface. Ensure the chatbot provides accurate and relevant information in response to user queries, promoting a reliable and informative user experience. Implement algorithms and guidelines to ensure the chatbot responds in an ethical manner and promotes positive interactions. Incorporate a feature for the chatbot to provide citations when delivering information, enhancing transparency and credibility.

1.3. Definitions and Acronyms

AI: Artificial Intelligence
Chatbot: A computer program designed to simulate conversation with human users, especially over the internet.
FR: Functional Requirement
NFR: Non-Functional Requirement
MVC: Model-View-Controller
ER: Entity-Relationship
GUI: Graphical User Interface
RESTful APIs: Representational State Transfer Application Programming Interfaces
SQL Database: Structured Query Language Database
IDE: Integrated Development Environment
ML: Machine Learning
HTTPS: Hypertext Transfer Protocol Secure
GPU: Graphical Processing Unit
APIs: Application Programming Interfaces
Git: Version Control System

1.4. Assumptions

User Engagement: It is assumed that users will actively engage with the chatbot and provide relevant queries to initiate conversations.

Internet Connectivity: Users are assumed to have access to stable internet connections to interact with the chatbot via the web interface.

Ethical Behavior: The chatbot is expected to behave ethically based on the guidelines and algorithms implemented during development.

Citation Availability: It is assumed that relevant information sources with citations are readily accessible to the chatbot to support its responses.

2.0. PROJECT REQUIREMENTS

2.1. Functional Requirements

FR.1.0 Home Page (Phase 1)

The system shall display a home page providing product name, relevant information and navigation to Login or Signup page.

FR.1.1 Navigation Bar: The bar should be on the top of web page with links to final report, video presentation, and the GitHub site of the project.

FR.1.2 Login button: The login button shall redirect users to the login page, enabling them to access the chatbot using their login credentials.

FR.1.3 Signup button: The signup button shall redirect users to the signup page, where users can register a new account using their email address and password.

FR.1.4 Headshots of Contributors: The home page must contain a table containing headshots, names, and description of team members who worked on the AI chatbot project.

FR.1.5 Logo & Name: The home page must contain the logo, name of the AI chatbot product, and project ID.

FR.1.6 Project Overview: The homepage must contain project overview, semester & course information.

FR.1.7 Content size for web clients: All requirements FR1.0 to FR.1.6 must be visible on the web client without requiring the user to scroll through the content.

FR.2.0 Login Page

The system shall provide a secure login mechanism requiring an email and password for users to access personalized features.

FR.2.1 Login with email and password (Phase 1): The system shall provide a window for users to enter their email address and password to login. Those fields are required and should not be left empty.

FR.2.2 Login Button (Phase 1): Upon user input, the login button shall validate credentials. If valid, the system shall log in the user and redirect to the main chat window. If the user has set their account to use multi-factor authentication, clicking the button will re-direct the user to the multi-factor authentication screen for further verification.

FR.2.3 Link to Signup Page (Phase 1): The login window shall provide a link to the signup page for users without an existing account associated with the chatbot.

FR.2.4 Multi-factor Authentication Screen (Phase 1): If the user has set their account to use multi-factor authentication, the page will ask the user to enter their six-digit authentication code.

FR.3.0 Signup Page (Phase 1)

FR.3.1 Sign-up with email and password: The system shall provide a window for users to enter their email address and password to register for a new account. Those fields are required and should not be left empty.

FR.3.2 User information: The system shall provide fields for users to enter their information (name, age)

FR.3.3 Signup button: The signup button shall validate and process user registration upon submission of the required information.

FR.3.4 Email verification: The system shall send an email verification code to the users' email address. The signup process will not be completed until users validate their email.

FR.3.5 Link to Login Page: The signup page shall include a link to the login page for users that already have an existing account.

FR.4.0 Chat Window

The chatbot will feature a user-friendly chat window that allows users to input questions or text and receive accurate responses. The chat window shall support a conversational interface with the ability to display citations if applicable.

FR.4.1 Profile Button (Phase 1): The profile button should be in the top right corner of the screen. Clicking on it should display links titled “settings” and “log out”.

FR.4.2 Chat title (Phase 1): The title of the chat should be on top of the chat history (FR.4.4) but below the profile button (FR.4.2) that indicates the chat number (ex: “Chat Number #1”) which the user is interacting with the AI.

FR.4.3 Choose a new topic (Phase 1): The button should be right of the chat title (FR.4.2). Clicking on it should clear the chat history and re-initialize the AI model.

FR.4.4 Chat history (Phase 1): This feature should show the history of the conversation with the chatbot. Each message should have a title mentioning the sender of message (User titled as “You” and the AI titled as its name). The user’s message must have a right alignment with the AI’s response aligned left. The chat history should be scrollable.

FR.4.5 Citation Preview Card (Phase 2): If the chat bot attaches a link to its cited page, a card below the chat bot’s message should appear, where it shows an image preview of the page, the page’s title, and a preview of the text contained in the cited page.

FR.4.6 Text Box (Phase 1): The text box should be positioned below the chat history (FR.4.4) and allow the user to type a message to the chat bot.

FR.4.7 Text Box Send Button (Phase 1): The send button should appear once the text box field (FR.4.6) is not empty. Clicking on it will send the message to the chat bot where it will generate a response to the user.

FR.5.0 Settings Page

The settings page should allow the user to manage configurations, including account, security, and privacy settings. Each option should provide the option name and a brief description of the option.

FR.5.1 Account & Security Settings (Phase 1): This category should allow the user to change the e-mail address and password and disable/disable multi-factor authentication.

FR.5.2 Privacy Settings (Phase 1): This category should allow the user to decide whether to use the chat history to train new models.

FR 5.3 Appearance Settings (Phase 2): This category should allow the user to adjust font size of the chat messages, and whether the color themes of the web client should follow the “light” theme, “dark” theme, or should be chosen automatically based on their client’s operating system color settings.

FR.6.0 AI Model (Phase 1)

The AI model should be able to parse user messages and respond to the user while meeting requirement NFR.4.1.

FR.6.1 Search Query Generator: There should be a backend framework that allows the chatbot to perform an online search and retrieve results from the search, including the entry’s URL.

2.2. Non-Functional Requirements

NFR.1.0 Security

NFR.1.1 Data Encryption: All user data, including login credentials and chat interactions, shall be encrypted during transmission to ensure data integrity and confidentiality.

NFR.1.2 User Authentication Security: The authentication process shall implement multi-factor authentication (verification code via SMS or email) to protect user accounts from unauthorized access.

NFR.2.0 Capacity

NFR.2.1 Concurrent Users: The system shall support a specified number of concurrent users interacting with the chatbot simultaneously.

NFR.3.0 Usability

NFR.3.1 Intuitive User Interface: The user interface, including the chat window, shall be designed to be intuitive and user-friendly, requiring minimal training for users to interact effectively.

NFR.4.0 Other

NFR.4.1 Ethical Response Behavior: The chatbot's responses shall align with ethical standards and human values, promoting positive and respectful interactions. These values are considered ethical and align with human values if they promote peace and do not cause harm to other users. For instance, the chat bot should not offer advice on activities that are illegal and promote violence.

2.3. External Interface Requirements

2.3.1. User Interface Requirements

The user interfaces for the AI Chatbot will be developed for the web client and iOS & Android platforms for mobile clients. These interfaces will adhere to GUI standards and styles specified by the project, featuring responsive designs suitable for various screen sizes. Users will interact with the chatbot through a chat window that allows input of questions or text. The interface will be designed to be simple and intuitive, providing clear instructions on how to use the chatbot. Standard buttons, functions, and error message display standards will be implemented for a seamless user experience.

2.4.2. Hardware Interface Requirements

The hardware components for the AI Chatbot include a computer or server with sufficient processing power and memory to handle the chatbot's workload. This system should be connected to the internet to facilitate real-time interactions with users. Specific hardware requirements, such as supported device types, graphical processing units (GPUs), communication protocols, and input/output formats, will be determined based on the technology stack's demands.

2.4.3. Software Interface Requirements

The AI Chatbot interacts with various software components, leveraging Python and Flask for the backend and ReactJS for frontend. The front-end interfaces communicate with the backend using RESTful APIs. Microsoft Azure SQL Database will store and retrieve data, ensuring efficient data management. Git will serve as the version control system, allowing collaborative development. PyTorch will be employed for any machine learning aspects. Visual Studio Code will be the integrated development environment (IDE) for

coding and testing. Docker will be used for creating images with software dependencies included. This will better facilitate the deployment of ML models.

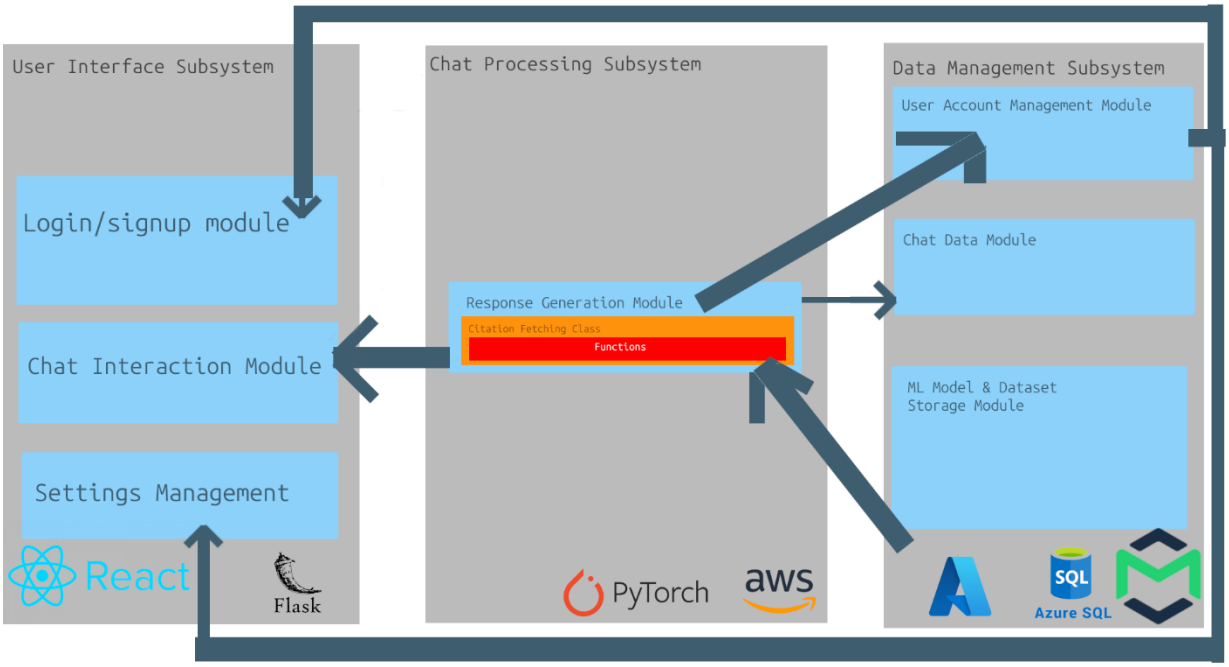
2.4.4. Communications Interface Requirements

The product will utilize Microsoft Azure as the web hosting service, ensuring reliable and scalable hosting. Communication functions will involve standard web protocols for web and mobile clients, including HTTPS for secure data transmission. The development team will use Git for version control, fostering efficient collaboration. Any communication with external services, databases, or APIs will be secured, and data transfer rates will be optimized for responsive user experiences.

3.0. DESIGN

3.1. Classification

For this AI Chatbot project, the classification of components is structured around its capacity to engage users, process queries, and provide ethical, accurate information across web and mobile interfaces. The figure below shows the diagram of the program's classification.



Specifically:

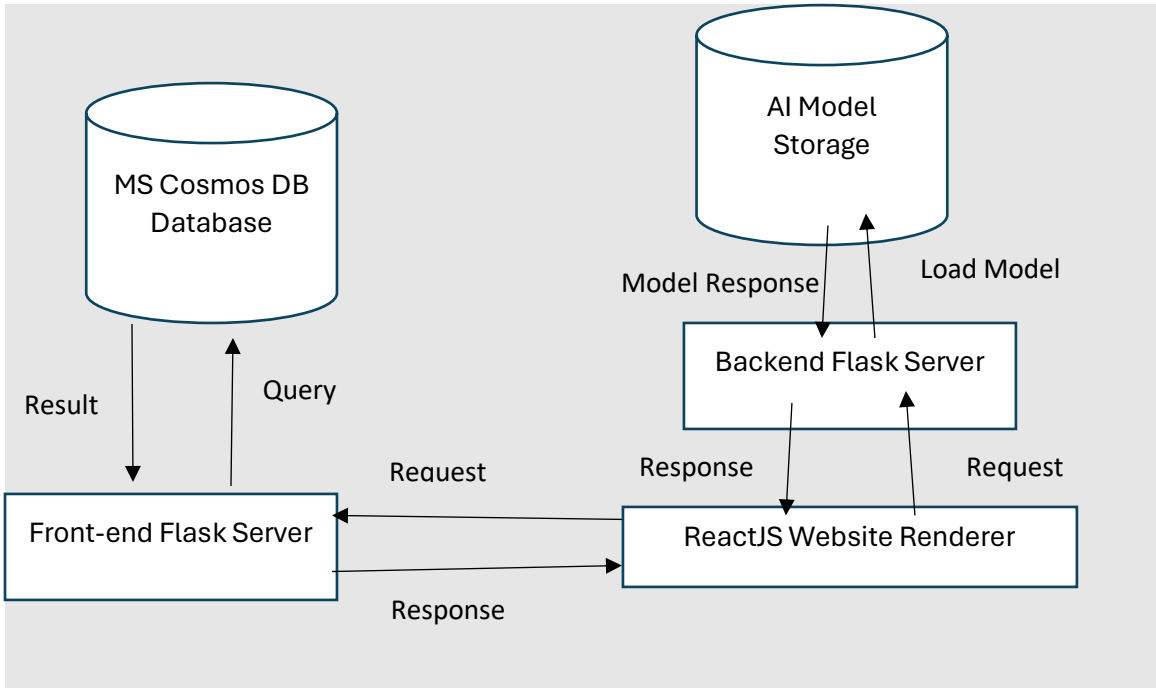
- **Subsystem Classification:** The project is divided into the User Interface Subsystem, ensuring accessibility and ease of use for diverse user groups across platforms such as iOS, Android, Windows, and macOS; the Chat Processing Subsystem, which incorporates advanced NLP techniques to understand and generate responses, along with loading the saved ML models; and the Data Management Subsystem, which handles the storage and retrieval of data, including user queries and system-generated citations, utilizing technologies like Microsoft Azure SQL Database for operational efficiency, along with using MailTrap to send users multi-factor authentication.

- **Module Classification:** Within the Chat Processing Subsystem, modules include the Query Parsing Module, leveraging Python and potentially AI libraries like PyTorch for processing user inputs, and the Response Generation Module, tasked with formulating answers and fetching citations. The User Interface Subsystem contains modules for Login/Signup Processes, Chat Interaction, and Settings Management, developed with ReactJS to facilitate a responsive and intuitive user experience. Last of all, the Data Management Subsystem contains the User Account Management Module, Chat Data Module, and the ML Model & Dataset Storage Module.

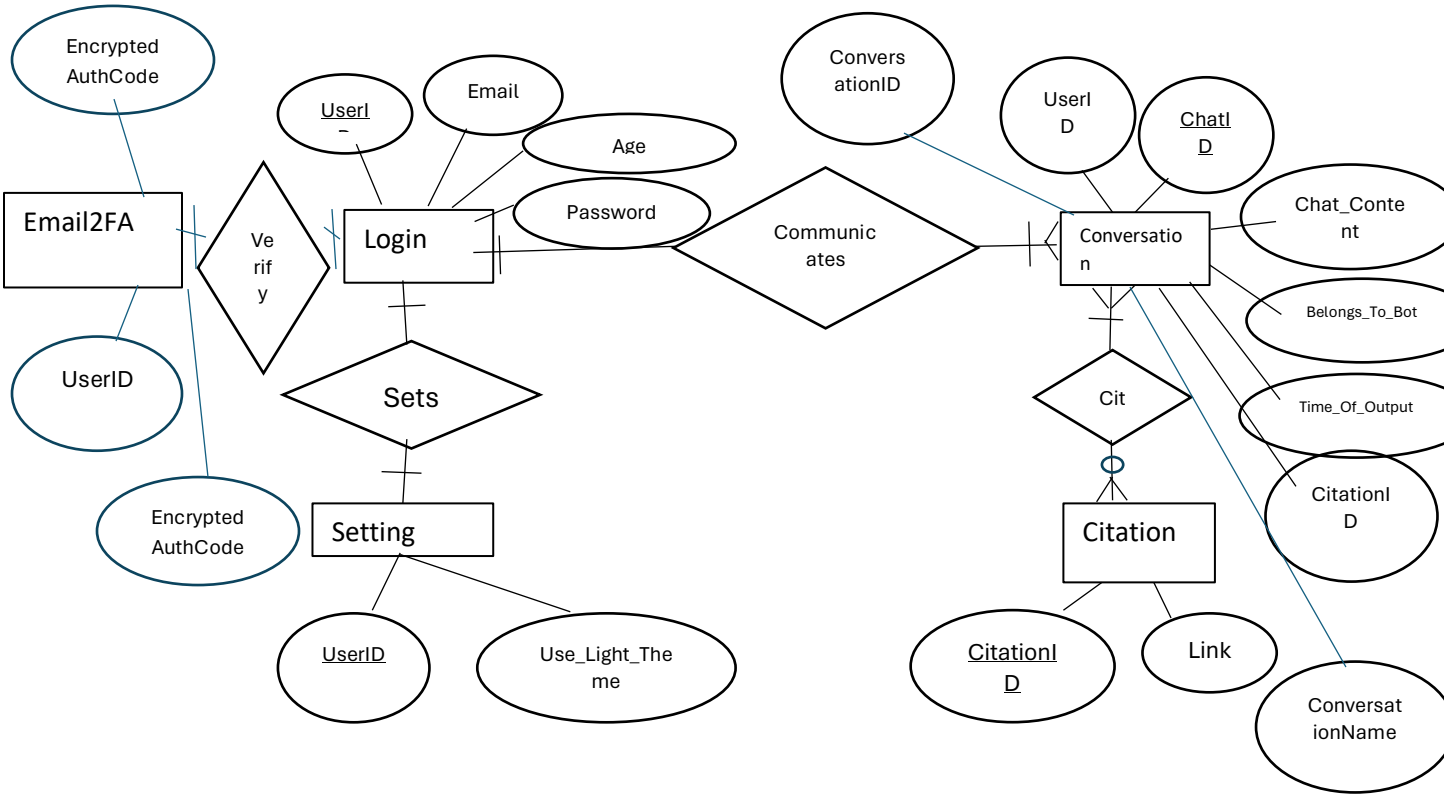
- **Class and Function Classification:** Each module is further decomposed into classes that encapsulate specific functionalities. For instance, the Query Parsing Module might have classes for Intent Recognition and Entity Extraction, each containing functions to execute their respective tasks, such as extracting key phrases from user inputs or identifying the intent behind a query. The Response Generation Module includes a Citation Fetching Class, with functions designed to retrieve and format citations from external sources, enhancing the credibility and transparency of provided information.

This hierarchical classification ensures a clear delineation of responsibilities within the AI Chatbot system, facilitating a modular approach to development and maintenance. It supports the project's overarching goals of delivering a reliable, informative, and ethically aligned user experience, while accommodating the diverse technical environments and user characteristics outlined in the project specifications.

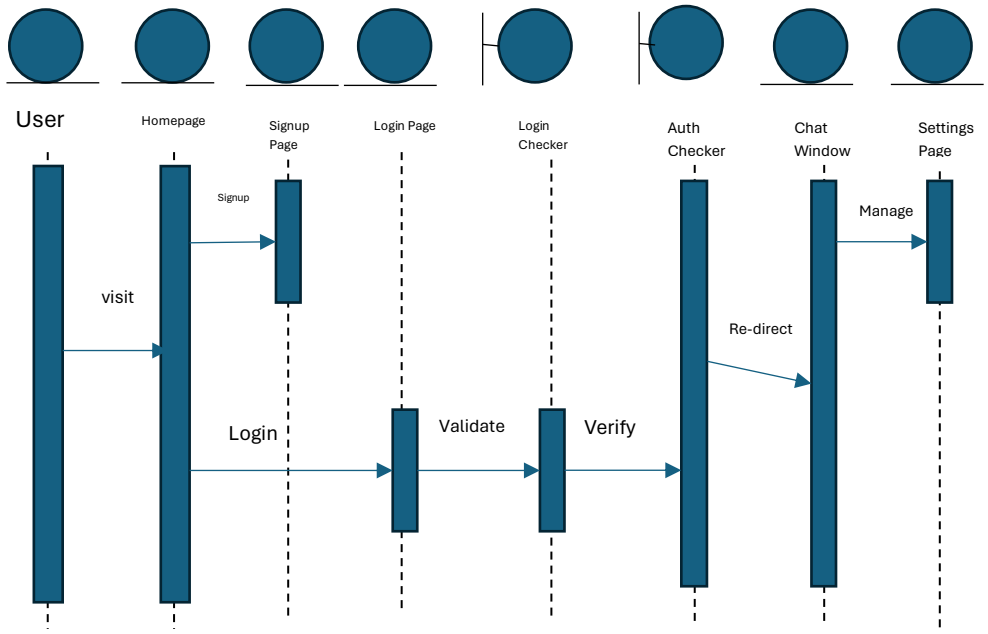
3.2. MVC Class Diagram



3.3. ER Diagram

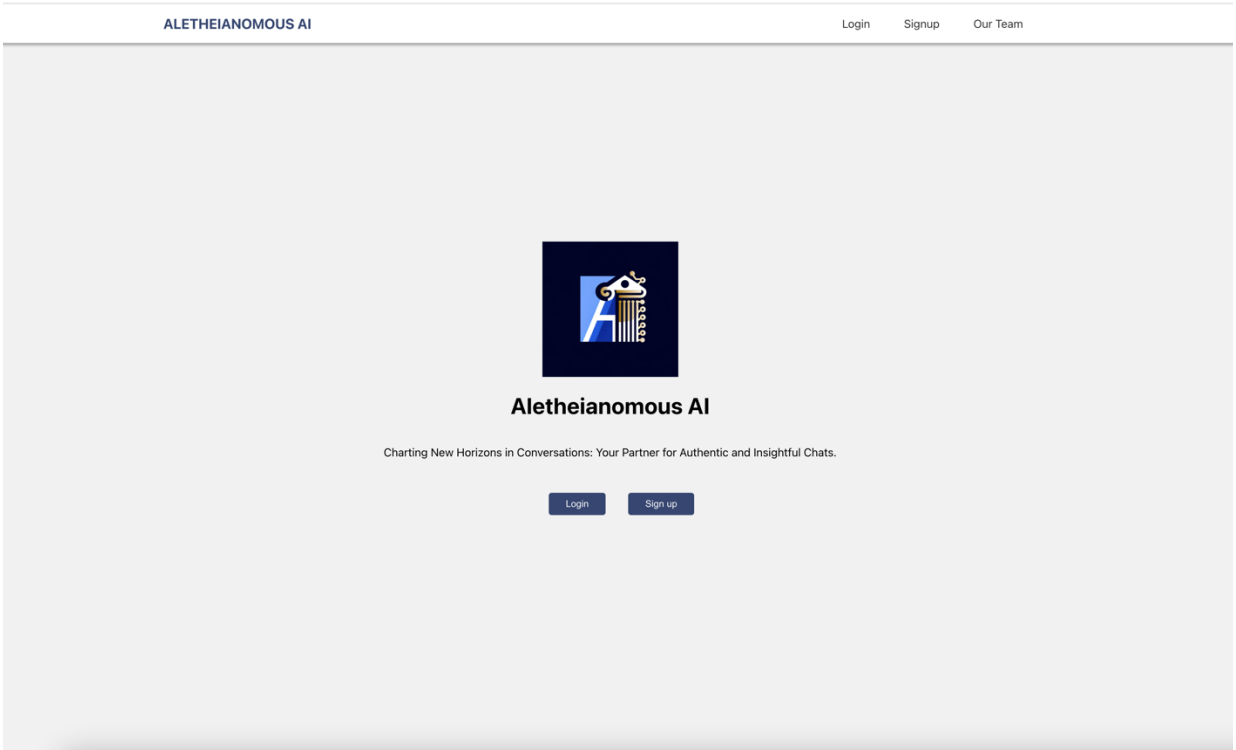


3.4. Sequence Diagram

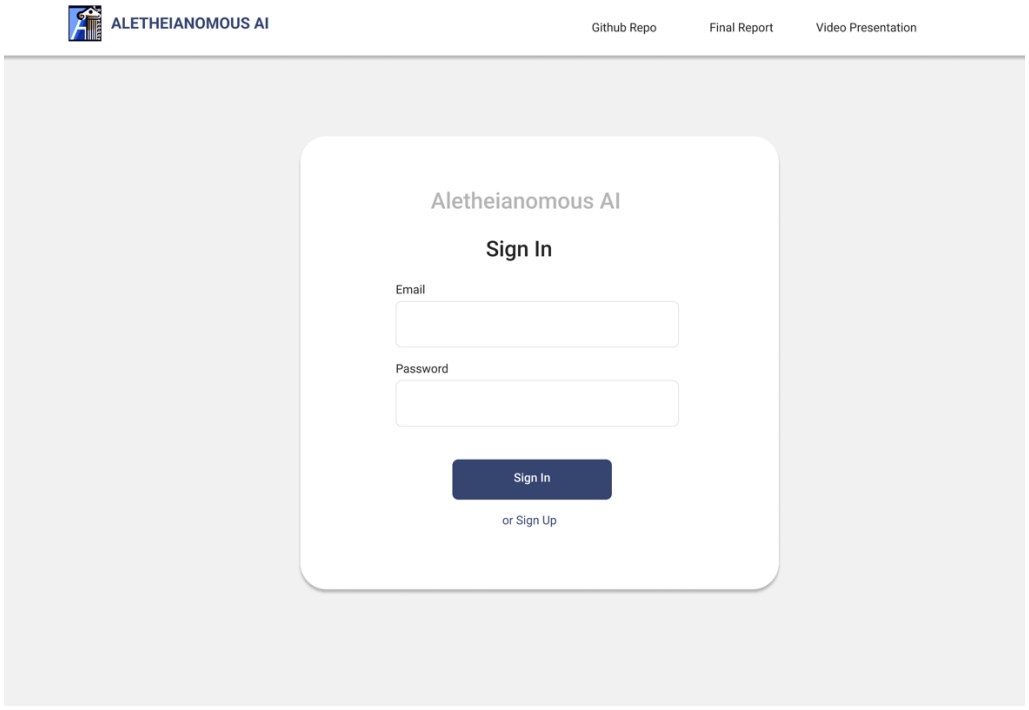


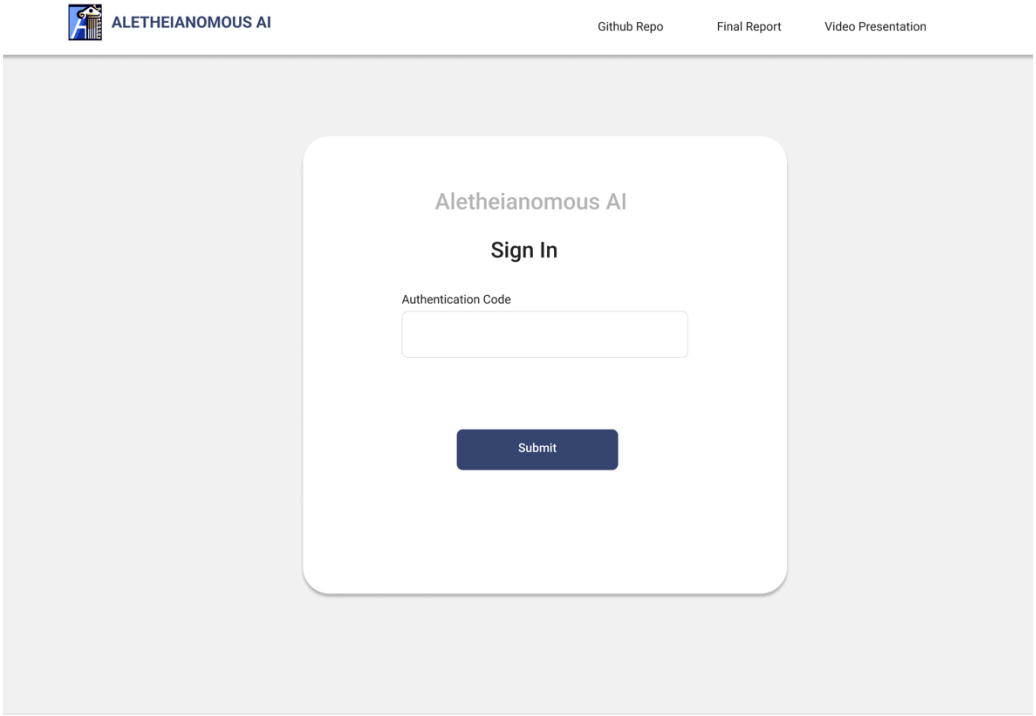
3.5. User Interface Mockup

Home Page

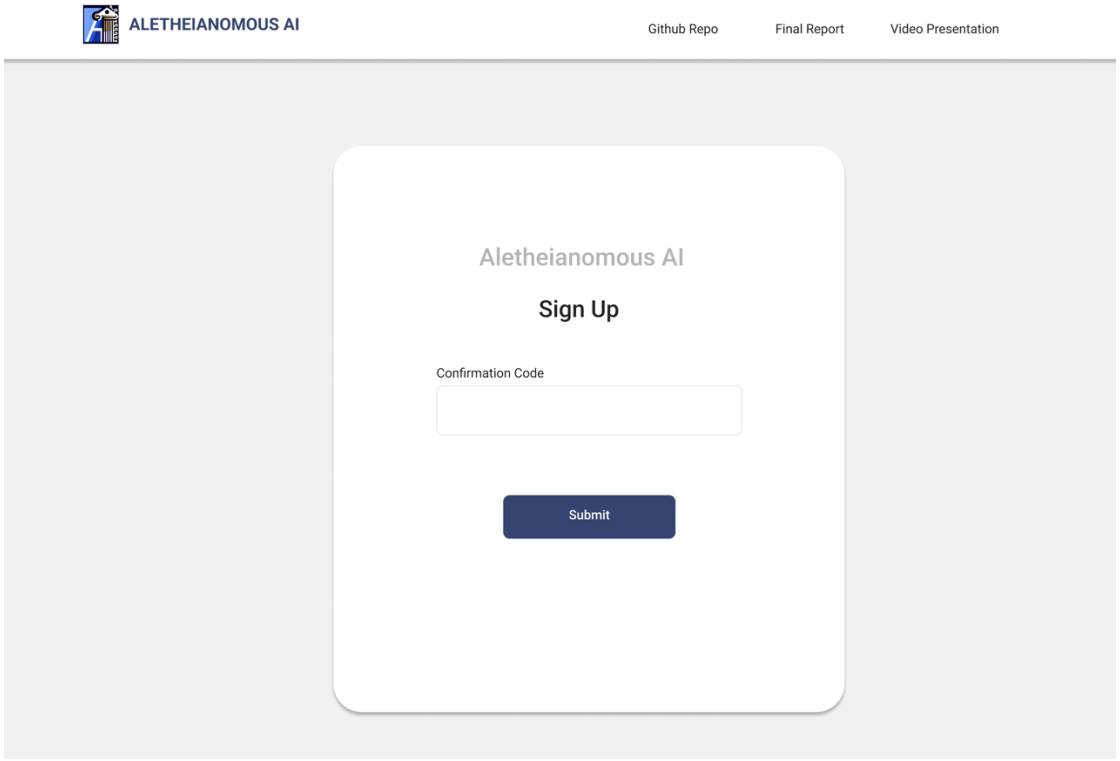


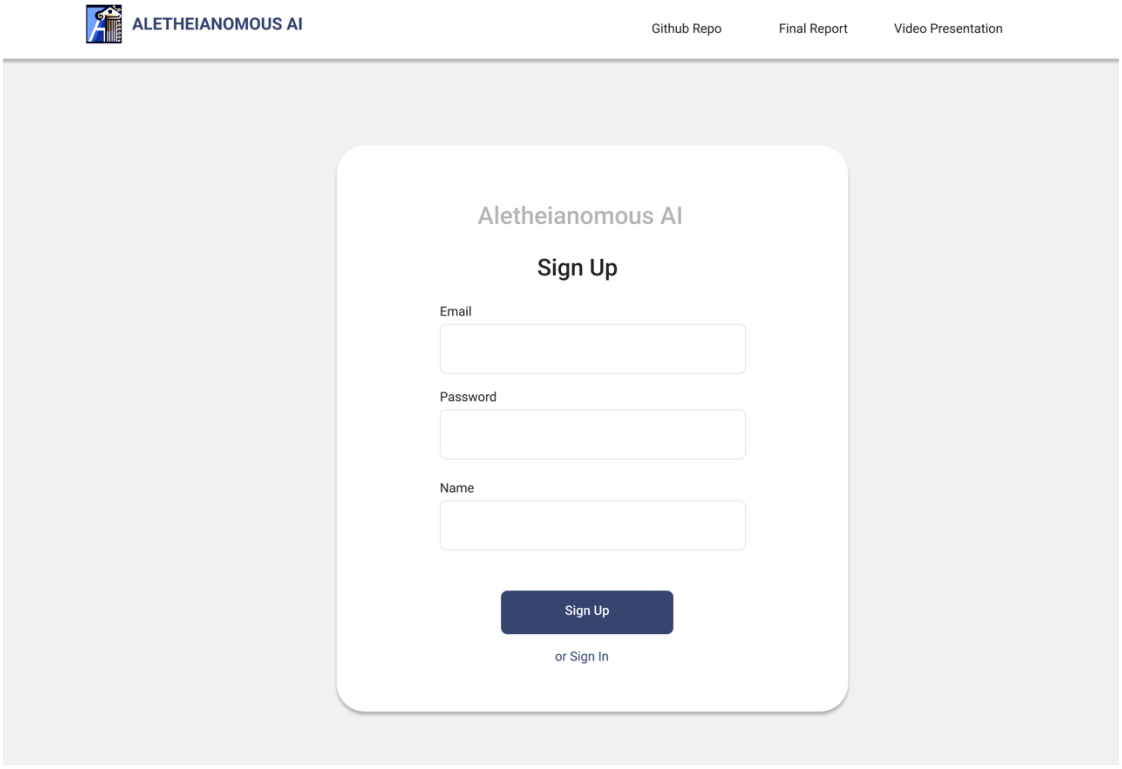
Sign-in Page


ALETHEIANOMOUS AI
Github Repo
Final Report
Video Presentation
Aletheianomous AI
Sign In
Email
Password
Sign In
or Sign Up

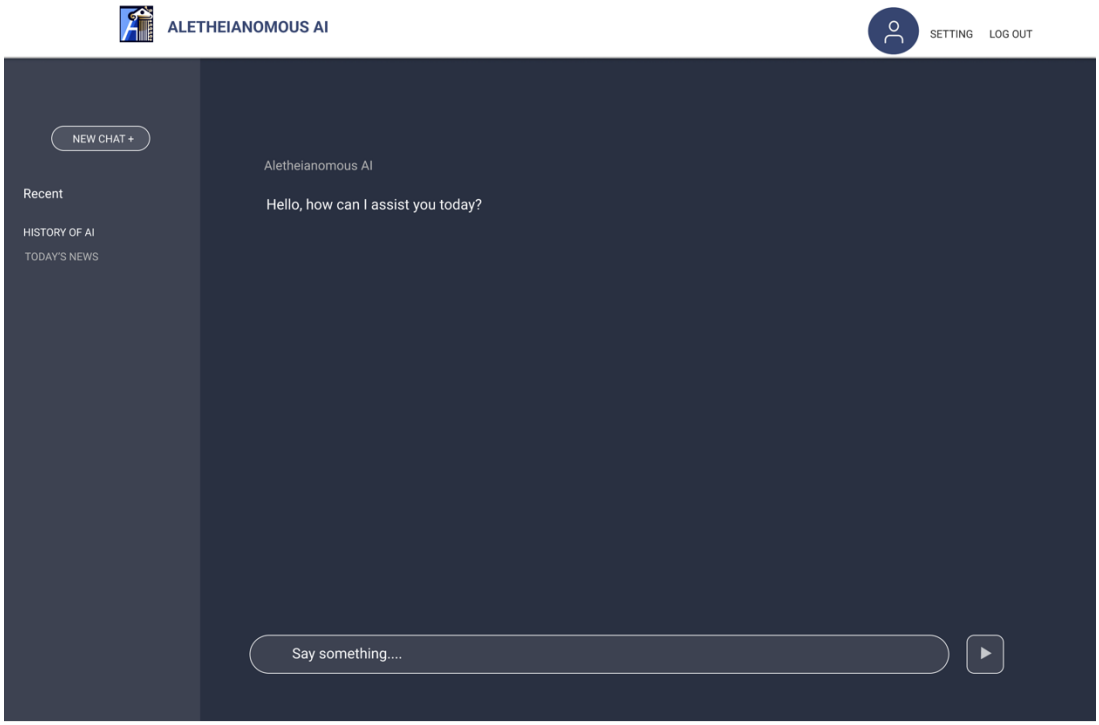

ALETHEIANOMOUS AI
Github Repo
Final Report
Video Presentation
Aletheianomous AI
Sign In
Authentication Code
Submit

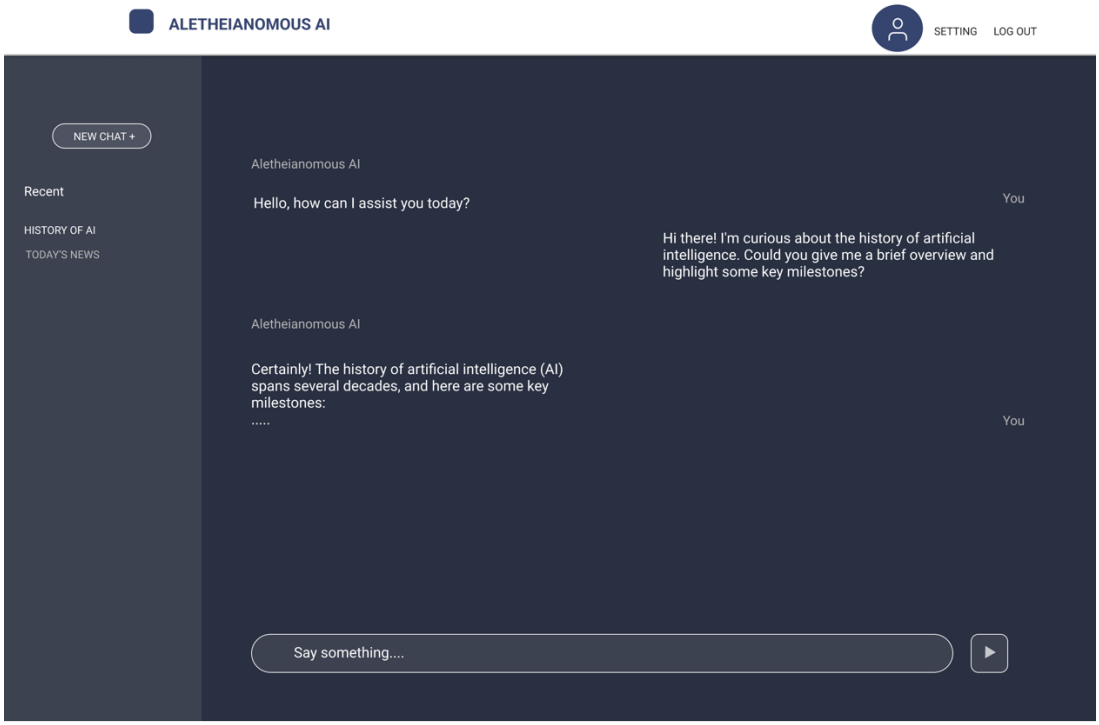
Sign-up Page


ALETHEIANOMOUS AI
Github Repo
Final Report
Video Presentation
Aletheianomous AI
Sign Up
Confirmation Code
Submit

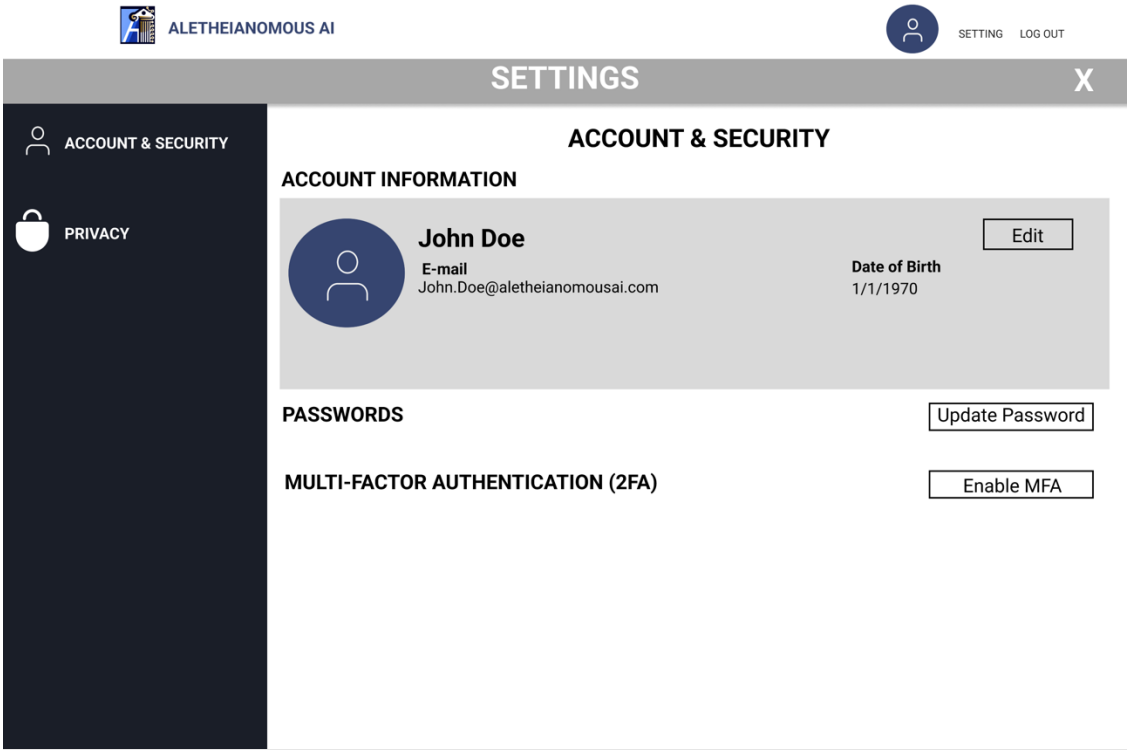

ALETHEIANOMOUS AI
Github Repo
Final Report
Video Presentation
Aletheianomous AI
Sign Up
Email
Password
Name
Sign Up
or Sign In

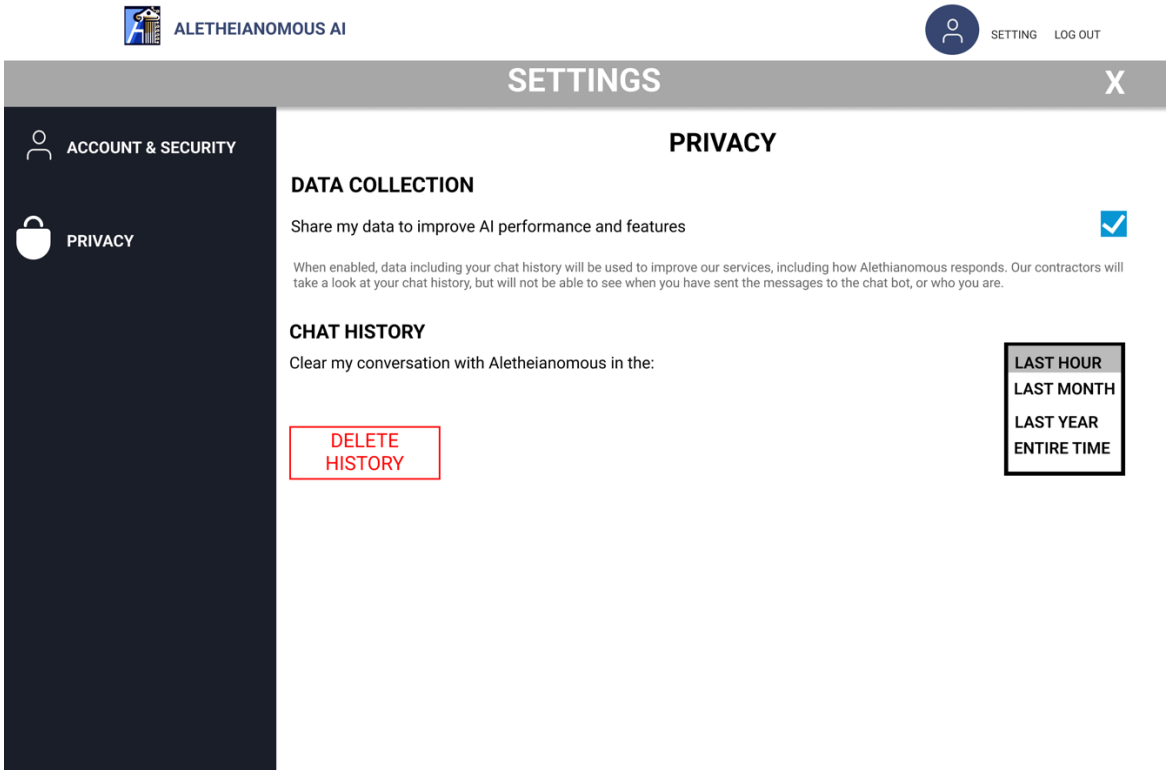
AI Chat Page


ALETHEIANOMOUS AI
SETTING
LOG OUT
NEW CHAT +
Recent
HISTORY OF AI
TODAY'S NEWS
Aletheianomous AI
Hello, how can I assist you today?
Say something....


ALETHEIANOMOUS AI
SETTING
LOG OUT
NEW CHAT +
Recent
HISTORY OF AI
TODAY'S NEWS
Aletheianomous AI
Hello, how can I assist you today?
You
Hi there! I'm curious about the history of artificial intelligence. Could you give me a brief overview and highlight some key milestones?
Aletheianomous AI
Certainly! The history of artificial intelligence (AI) spans several decades, and here are some key milestones:
.....
You
Say something....

Settings Page





4.0. DEVELOPMENTS & CHALLENGES

4.1. Frontend Development

Technology Stack

For the frontend development of our chatbot project, we utilized ReactJS, CSS, and JavaScript. We chose these technologies primarily because our project focuses on web application development. ReactJS offers a component-based architecture, which facilitates modular development and reusability of code. CSS provides styling capabilities to enhance the visual appeal and user experience of the web application, while JavaScript enables dynamic functionality and interactivity.

Development Process

Figma Design: Before diving into development, we collaborated on designing the user interface using Figma. This allowed us to create wireframes and prototypes for each page of the web application, ensuring a clear understanding of the design requirements and user interactions before implementation.

React App Creation and Project Structure Setup: We initiated the project by creating a React application using the create-react-app command. We established the general structure of the web application, including components for the navigation bar and main pages. This allowed us to create a cohesive user interface with consistent navigation and layout across different pages.

Homepage Development: The homepage serves as the entry point for users and provides essential information about the chatbot project. We utilized React Router DOM to enable seamless navigation between different sections of the application. We employed the `useNavigate` hook to handle navigation actions, ensuring smooth transitions without full-page reloads. Buttons were implemented using the `useState` hook to manage their state, allowing for dynamic interaction and feedback.

Login and Signup Pages: These pages provide users with authentication and registration functionalities, allowing them to access the features of the chatbot. We integrated `useState` hooks to manage form input fields such as email and password. We implemented user-friendly interfaces for inputting login credentials and registration information, ensuring a seamless user experience.

Chat Page Implementation: The chat page is where users interact with the chatbot. The `useState` hook facilitated dynamic updates to the message input field, enabling users to type and send messages in real-time. The chat page also includes features such as message history and input validation.

Challenges Encountered

Responsive Design: Ensuring that the web application is responsive across various devices and screen sizes posed a challenge. We addressed this by implementing responsive design principles and testing the application using Inspect features in Chrome for different device screen sizes.

4.2. Backend Development

Technology Stack

For the backend development, Flask was used to handle the REST API calls from the front-end server that rendered the web pages. The backend contained two types of servers: the front-end REST API handler server, and the backend server (which runs the AI models). The servers were virtual machines running Ubuntu 22.04, where the front-end REST API handler was deployed in Microsoft Azure, and the back-end server was deployed in AWS. Microsoft Azure was used since it contained a low hourly price for running the VM. Following this, AWS was used to deploy the backend server, since it required access to the GPU for running the AI models. Microsoft Azure does not provide servers with GPU for free, and it has a higher hourly price compared to AWS. The backend server communicated with the database servers using the MS SQL 18 Drivers for Linux, and the API was integrated by using the PYODBC Python library.

Development Process

Front-end REST API Handler server: This server is deployed in Microsoft Azure and handles REST API calls from the web client. The server contains Python Files that comprises the data management subsystem, which are responsible for retrieving and uploading chat data. To handle REST API calls, a Python file named app.py was created, where it contains each function that is executed depending on the API call the server retrieves. The functions in app.py incorporate the classes from the Data Management Subsystem and handles requests to retrieve chat history for a specific user, upload chat data to the database server, and to request the backend server to make the response generator reply to user input (See Back-end Server).

Back-end REST API Handler server: This server was used as a VM in AWS to handle requests from the front-end server to generate output by the response generator model. Similarly to the front-end server, this server contains Python files related to the response generator class and citation fetcher class. The app.py file only handles requests from the front-end server which only contains the input message. This message is then passed to the response generator class, which then returns the model's output and citations (if applicable). Following this, the server returns this data to the front-end server as the JSON file.

Challenges Encountered

Reliability: The challenge was ensuring that both servers could handle REST APIs requests that were valid. To continue, if a server error occurs while processing the request, it must handle it. This was addressed by committing extensive unit testing on the REST APIs calls.

4.3. Database Management

Technology Stack

In our project, we have fully integrated Microsoft Azure as the backbone of our database management tech stack, harnessing the powerful capabilities of Azure SQL Database. This cloud-based service has enabled us to scale our operations efficiently, ensuring robust performance and reliability across our applications. By utilizing Azure SQL Database, we benefit from built-in high availability, automated backups, and dynamic scalability options which significantly reduce our overhead for hardware management and maintenance.

Development Process

During our project's development, the development process played a crucial role in shaping our database schema and functionalities, leveraging the robust capabilities of Microsoft Azure SQL Database. Our approach began with the construction of an initial schema focused on the core components such as the Login, Chat History, Citation, and various linking tables to support the different relationships between data points. For instance, we identified the need to securely manage user authentication and preferences, leading to the creation of the Login and Setting tables. We tried to find best practices for data security by deciding to encrypt passwords, a necessity we realized during the analysis phase where user data protection emerged as a paramount requirement. Azure SQL Database's security features, including its built-in encryption capabilities, provided us with the necessary tools to implement this securely and efficiently without too much

back breaking. Furthermore, as our system needed to manage citations linked with chat interactions, the development of the Citation and Citation-Chat History tables was guided by the necessity for potential future data retrieval methods.

Challenges Encountered

While there were no major challenges, the biggest faced was simply the learning curve, as while we are accustomed to different technologies and techniques, none of us had ever set up anything on Azure, so that did require some solid training through articles and videos, and experimentation. The complexity of Azure such as managing and scaling databases, the thorough security settings, and different analytics were overwhelming and difficult to adapt to at first.

The 2nd challenge was the advancement past Azure. We wanted our system to be nice and robust from the start, we found that finding ways to enhance it would be difficult and potentially out of our scope and budget. So, we started to explore third-party encryption libraries, but it is difficult to evaluate which ones could potentially be better compared to Azure. We found that the best way for this was to add MFA and oher security measures at the application level.

4.4. AI Model

Technology Stack

For training the model, we used frameworks that were related to machine learning models. JupyterLab was used as a type of IDE for machine learning, as it allows the developer to run a cell of code and see the output in graphical and terminal format. In addition, PyTorch and HuggingFace were used as Python libraries, as HuggingFace allows the AI engineer to download pre-trained ML models, and PyTorch can be used to re-train the model on new data. The NLTK library was used as the sentence separation model, which can separate user input into sentences. To determine whether the user input could be queryable, a custom LSTM model was trained on both the SquAD 2.0 and UltraChat datasets. Also, the re-trained Zephyr-7-alpha was used to serve as the search query extractor model, since it was pre-trained on the input sentence-keyword pairs. For the response generator model, the original Zephyr-7b-Alpha model was used, since it accepts system prompts, which specify how the model should respond to user input. Last of all, the DuckDuckGo-Search Python library was used to execute the search query and forward the search results to the response generator model. The models were tested using a local Lambda server containing an NVIDIA RTX 2080 GPU. Once the model's

development was completed, it was deployed to an AWS server that contained access to an NVIDIA GPU with 16 GB of VRAM.

Development Process

Response Generator Class: The response generator class was created as a JupyterLab file. In this class, the search query extractor model, the sentence separator models, and response generator models were defined. With this class, the keyword extractor model was tested by sending test input data to the model. Following this, the response generator model's response was tested by sending test questions to it. The criteria for the model's accuracy depends on whether the model has answered the user's question, and if the content provided by the chatbot was correct.

Citation Fetcher Class: The citation fetcher class was created as a separate Jupyter Lab file. It was tested by executing the keywords search query using the DuckDuckGoSearch API, and evaluating the output dictionary that the citation fetcher class has returned.

Challenges Encountered

Real-time Model Response: The challenge was to use a set of models that can be executed on the server. Their memory usage had to be sufficient that it is within the range of the server's capacity. In addition, the latency for the model to respond to user input must be in real-time. This was resolved by researching the server hardware that could support the model, and by applying memory management techniques.

Information Accuracy: The information presented by the model to the user had to be accurate and correct. In addition, the model must have explainability with regards to the sources it used. To resolve this, we decided to use a pipeline where the search results from the citation fetcher class were forwarded to the model's input. In addition, the model was evaluated by sending test prompts to it and verifying the accuracy of the model's output.

4.5. Backend-Frontend Integration

Technology Stack

Our technology stack encompasses GitHub, Microsoft Azure, AWS, JavaScript, MailTrap, and Flask. GitHub facilitated version control and collaboration,

enabling concurrent work and efficient code management. Microsoft Azure provided scalable hosting services, ensuring high availability and resource management. AWS offered a suite of cloud computing services, optimizing data storage and retrieval processes. JavaScript empowered dynamic and interactive frontend development, enhancing user experiences. Flask served as our backend framework, facilitating rapid development of RESTful APIs. Last of all, MailTrap provided the means to message users the six-digit authentication code.

Development Process

Our integration process involved the following steps:

RESTful APIs: We established RESTful APIs on the backend to expose endpoints for frontend communication. These APIs have facilitated data exchange and interaction between the frontend and backend components.

Data Exchange: Through RESTful API endpoints, data was exchanged between the frontend and backend layers. This included sending requests from the frontend to fetch data from the backend, as well as sending data from the frontend to the backend for processing and storage.

Authentication and Authorization: We implemented authentication and authorization mechanisms to secure communication between the frontend and backend. This ensured that only authorized users could access protected resources and perform specific actions within the application.

Error Handling: Robust error handling mechanisms were implemented to handle exceptions and errors that may occur during communication between frontend and backend. This included providing meaningful error messages to users and logging errors for debugging and troubleshooting purposes.

Challenges Encountered

Cross-Origin Resource Sharing (CORS): Configuring CORS policies to allow communication between frontend and backend servers posed a challenge, especially when deploying the application to different environments. We addressed this by configuring CORS headers on the backend server to allow requests from specific origins.

5.0. APPLICATION TESTING

SP-15 ChatBot Application Testing Report		
	Pass	Fail
Have AI model produce output in response to user message	Yes	
Have AI model add references to information it has cited	Yes	
Have Chat Data Module upload chat logs to SQL server	Yes	
Have Chat Data Module retrieve chat logs to SQL server	Yes	
Have the program authenticate user based on email and password	Yes	
Have front-end server communicate with backend-server on request to run model	Yes	
Have web client communicate with frontend-server	Yes	

6.0. VERSION CONTROL PLAN

We have chosen Git as our primary version control system due to its widespread adoption, robust feature set, and flexibility. Git's distributed nature allows each team member to work offline and independently, while still providing powerful collaboration capabilities when connected to a central repository.

Our branching strategy follows the Gitflow model, consisting of main branches (master and develop) and supporting branches for feature development, release preparation, and bugfixes. Feature branches will be created for each new feature or enhancement, providing isolation for development and ease of integration. All code changes must be made in feature branches and submitted via pull requests for review. Code reviews are mandatory for all pull requests to ensure quality, adherence to coding standards, and knowledge sharing.

7.0. FINAL SUMMARY

To summarize, the final report document describes the entire development process of the AI chatbot. This includes information from the project plan, requirements, and design sections. The premise of this project is to provide a web interface where users can communicate with the AI chatbot. Each user is authenticated by sending a REST API post request to the front-end server, which contains username and password data. If this data matches on the SQL database server, access is granted to the user. The user can then access the chat history and send new messages to the chatbot, where it can respond to the user in real-time. From the web client, the user can quickly edit settings, including the appearance of the chat window, which data could be collected from the user, and changing the email address and password.

The project plan describes how this project was conceptualized with our goals defined. Non-functional and functional requirements are described in the requirements section, while the design of the hardware and software programs, along with the frameworks, are described in the design section. This includes figures regarding the relationships between database tables, the model view controller diagram, and along with the subsystems used in our program. Lastly, the test plan describes how our program implementation was rigorously tested.

APPENDIX A – Training

I, Aimi Tran, completed the ReactJS tutorial at React Full Course

Signed by: Aimi Tran

I, David Chavarro, completed the MailTrap tutorial at the MailTrap course.

Signed by David Chavarro

APPENDIX B – Project Plan

PROJECT PLAN

Project Overview

Having the right information is important to make impactful decisions. Users utilize a search engine to acquire that information. However, the results presented may not always be reliable. This would require the user to read the source article to test for accuracy. As a result, productivity is reduced. Some chat bots summarize but may fail to output correct information, especially regarding mathematical fields.

For this project, an interactive AI chatbot will be developed that interacts with the user in text format. The user can provide questions to the AI chatbot through a web and mobile client interface, which the chatbot can reply by providing accurate information. Note that the mobile client interface will be on iOS & Android. In addition, the chatbot may provide citations to its provided information, if applicable. The chatbot should respond in an ethical manner that aligns with human values.

Deliverables

- Final Report
- Web & Mobile client where user can communicate with AI chatbot.
- Website that will describe the AI chatbot project.
- GitHub link where software documents and source code for the AI chatbot will be housed in the repo.
- Final Video Presentation – A presentation describing how the AI chatbot is developed (40% of video) and showing a live demo of the chatbot (60% of video)
- Team/Project Selection Document

- Project Plan
- Gantt Chart
- SRS
- SDD
- Requirements Document
- Final Product Name

Milestone Events

#1 – By 2/11/2024: Project Requirements & Design Complete

- Team/Project Selection Document
- Project Plan
- Gantt Chart
- SRS
- SDD
- Requirements Document
- Final Product Name

#2 - By 3/17/2024: Finish Prototype Development

- Research AI Model type, and Train & Deploy a trained AI Chatbot
- Code review for frontend & backend development.
- STP
- Web & Mobile client
- Develop & Test Prototype

#3 – By 4/21/2024: Have Presentations & Deliverables Ready

- Prototype Presentation for Peer Review
- Peer reviews
- Final Report Draft
- Video Demo
- Website displaying the project.
- C-Day Package
- Final Report Package

Weekly Activity Reports (WARs – Individual Assignment; weekly from 2/9/2024 to 4/20/2024)

Meeting Schedule Date/Time

Starting on February 7th, 2024, the team meeting will be held via Microsoft Teams on Wednesdays from 10:00 AM to 10:30 AM (Eastern Time zone).

Collaboration and Communication Plan

Communication

All team members will communicate with the team leader and advisor using Microsoft Teams. This relies on a channel that has been set up by the advisor. Members should always check the posts on Microsoft Teams and reply to inbound communication within 48 hours or 2 days. In addition, the team leader will host a weekly standup meeting (as mentioned above). All team members are expected to attend the meeting (excluding the project advisor). If they believe that they will not be able to attend, they should send a Teams message to the team leader at least one day in advance. During the meeting, Aimi Tran will write down effective notes regarding the meeting. In addition, with the consent of all attending members, the meeting will be recorded.

Collaboration

Team members will collaborate with the working draft of the documentation files using Microsoft OneDrive under their KSU organizational account. On the other hand, software files (including, but not limited to Python files, Jupyter notebook files, etc.) will be uploaded to the organizational GitHub account. In addition, the finalized versions of the documentation files will be saved into the main branch under the “documentation” folder. This account is different from an individual GitHub account (see the “Version Control Plan” section).

Project Schedule and Task Planning (Applies to All)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	**Project Name:**	SP-15 AI Chatbot																	
2	**Report Date:**	1/30/2024																	
3						*Milestone #1*				*Milestone #2*				*Milestone #3*					
4	**Phase**	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	**01/23**	**01/30**	**02/06**	**02/13**	**02/20**	**02/27**	**03/05**	**03/12**	**03/19**	**03/26**	**04/02**	**04/09**	**04/16**	**04/23**
5	Requirements	Team Project Selection Doc	100%	Complete	All	3													
6		Project Plan	100%	Complete	All	2	2												
7		Gantt Chat	100%	Complete	Ethan		2												
8		SRSS	0%				2	2											
9		SDD	0%				3	3											
10		Requirements Document	0%				1	5											
11		Final Product Name	0%					4											
12	Finish Proto Develop	Research AI Model Type and Train	0%					5	5	5									
13		Code Review for Front and Backend	0%						4	4	4								
14		STP	0%						2	2	1	1							
15		Web and Mobile Client	0%						5	5	5	5							
16		Develop working prototype	0%								10	10							
17		Test prototype	0%								7								
18	Development	Prototype Present for Peer Review	0%								2	2	2	2	2	2	2		
19		Peer Reviews	0%								1	1	1	1	1	1			
20		Final Report Draft	0%											10	10				
21		Video Demo	0%												2	2	1		
22	Final report	Website Showcasing Product	0%						2	2	2	2	2	2	2	2	2	2	2
23		C-Day Package	0%													2	1	1	1
24		Final report submission to D2L and project owner	0%														10	10	10
25																			
26				**Total work hours**	**211**	**5**	**10**	**19**	**18**	**18**	**32**	**21**	**5**	**15**	**17**	**9**	**16**	**13**	**13**
27																			
28		* formally define how you will develop this project including source code management																	
29																			
30	*Legend*																		
31	Planned																		
32	Delayed																		
33	Number	Work: man hours																	

Risk Assessment

Risk 1: Third-Party Dependencies

- Likelihood: Low
- Impact: Moderate
- Description: Reliance on external libraries or APIs for AI functionalities may introduce vulnerabilities or service disruptions.
- Mitigation: Regularly update and monitor third-party dependencies. Have fallback mechanisms in place.

Risk 2: Infrastructure Scaling

- Likelihood: Low
- Impact: High
- Description: Inadequate infrastructure scaling leading to poor performance during peak usage.
- Mitigation: Monitor server loads, implement auto-scaling, and regularly assess and upgrade infrastructure as needed.

Risk 3: Infrastructure Outage

- Likelihood: Low
- Impact: High
- Description: Servers handling AI chatbot's model user input/output requests might be unavailable

- Mitigation: Create multiple backup copies of user (chatbot history, user preferences) & AI model data. Deploy model on backup server, which will handle requests.

Risk 4: Model's accuracy

- Likelihood: Moderate
- Impact: moderate
- Description: Model may output invalid information or have low performance.
- Mitigation: Have a monitoring dashboard that shows performance metrics; analyze performance; re-train the model on new dataset.