Episodic Future Thinking Chatbot

Tyler Buxton, Aaron Gomez, Jared Melini, and Patrick Johnson

Client: Sareh Ahmadi

CS4624 Multimedia, Hypertext, and Information Access

Professor: Dr. Edward A. Fox

Virginia Tech, Blacksburg, VA 24061

9 May 2024

Table of Contents

Table of Figures

Table of Tables

Abstract

This project focuses on enhancing the user interface (UI) and usability of an artificial intelligence (AI) chatbot designed to implement episodic future thinking (EFT) cue text generation for the treatment of obesity and type 2 diabetes. Episodic future thinking, a technique facilitating guided visualization of personal futures, has shown promise in reducing delay discounting and promoting lifestyle changes including weight loss. The project aims to overcome existing limitations, such as personnel costs and scheduling constraints, by leveraging AI to streamline EFT delivery -- improving scalability, reach, and potential efficacy.

The proposed system involves the development and testing of a website running on a local Computer Science (CS) virtual machine at Virginia Tech. The website supports user profile creation and management, password protection, administering of surveys, and personalized EFT cue text generation through interactions with an AI chatbot accessible via API given by the client. The entire interaction sequence, including interim and final chatbot-generated EFT cue texts, will be recorded for analysis.

Key functionalities include user account creation, login for EFT cue text generation sessions, chatbot interactions with feedback mechanisms, and a usability assessment using established metrics like the System Usability Scale. The project's impact is expected to greatly improve user experience, research outcomes, and implementation efficacy, ultimately contributing to enhanced health outcomes in the long term for the users.

Introduction

Problem

Episodic future thinking (EFT) is a cognitive process fundamental to human behavior, involving the imaginative process of projecting oneself into future scenarios. This ability plays an important role in shaping behavior by facilitating goal setting, informed decision-making, and strategic planning. It empowers individuals to resist immediate gratification, increasing their capacity for delayed rewards. EFT serves as a strong motivator, providing a vision of positive outcomes that sustains determination and effort. Beyond individual benefits, this cognitive skill contributes to emotional regulation, adaptive behavior, and social cognition, enhancing interactions with others and themselves. As part of personal identity and narrative construction, EFT creates future scenarios into the situation of one's life story, resulting in a sense of continuity and purpose. Interview-guided EFT cue generation is a method where researchers conduct interviews with participants to help them construct positive and realistic future event descriptions [1] Moreover, this process aids in problem-solving by engaging in pre-emptive thinking about potential challenges that they could encounter. Episodic future thinking significantly influences various aspects of human functioning, from individual decision-making to broader social dynamics. [2, 3, 4, 5]

At the start of the project there was no working public interface with the chatbot. The only way to interact with the bot was to be on a computer with access to the EFT trained model and running everything locally, connecting with OpenAI's GPT-4. The absence of a user interface (UI) holds significance in various contexts, particularly when prioritizing efficiency, automation, and resource optimization. A UI provides the users with comfortability knowing that they won't have to work with a complex system but instead only handle their own thoughts, hence improving the overall experience that they have interacting with the chatbot. In other words, the presence of a UI that reduces the complexity of the interaction frees the user to be able to express their thoughts more freely than if there wasn't one.

The absence of usability support within a system can dramatically impact user experience and overall system effectiveness. Users may grapple with frustration and inefficiency, leading to a diminished perception of a product or service. A lack of usable interfaces and effective workflows can result in an increased learning curve, hindering both new and experienced users. Higher error rates may surface due to unclear instructions, potentially compromising data integrity and user confidence. Poor usability may contribute to reduced user adoption and negatively impact a brand's reputation as dissatisfied users share their experiences. This can also lead to heightened support and training costs, as users seek help navigating usability challenges. Moreover, without attention to usability, accessibility for all users, including those with disabilities, may be compromised. In essence, prioritizing usability is paramount for creating positive user experiences, fostering user loyalty, and ensuring the success and reputation of a product or service [9].

Motivation

A refined and supportive user experience serves as a powerful motivator for several compelling reasons, enhancing overall user satisfaction through usable interfaces, streamlined workflows, and clear navigation. This not only reduces frustration but also significantly improves efficiency, enabling users to accomplish tasks more quickly and with greater ease. Additionally, a positive user experience fosters a strong emotional connection between users and a product or service, leading to increased engagement and retention. Furthermore, by aiding episodic future thinking through effective mental simulations and envisioning potential outcomes, the user experience strengthens goal setting, decision-making, and cognitive processes associated with personal growth and adaptability. In essence, prioritizing a better user experience directly benefits users while also contributing to a product's reputation and long-term success [6].

Team Roles

Tyler Buxton serves as the project manager, overseeing task delegation and acting as the primary contact for Dr. Fox, our client Sareh Ahmadi, and the graduate teaching assistant, Satvik Chekuri. In addition to his managerial role, Tyler actively contributes to front-end development, ensuring the seamless operation of the completed front-end and back-end components.

Aaron Gomez assumes the role of lead architect, steering technical decisions and supporting fellow developers in code writing and design. He has reviewed existing codebases, assisted in environment setup, and established the back-end.

Patrick Johnson takes charge of report management, handling writing and layout organization. Responsible for setting and meeting deadlines, Patrick ensures the timely completion of the report. His involvement extends to front-end UI development, focusing on the Login and Signup pages, and sourcing an alternative anonymizer code for safeguarding user privacy. Patrick manages the tasks and ensures timely completion of code deliverables and report submissions.

Jared Melini, a versatile developer, contributes to both front-end and back-end tasks under the guidance of Tyler or Aaron. His responsibilities include developing the usability survey page and assisting in database construction after completing the front-end. Jared also specializes as our resident button expert and has established the messaging flow. Jared has contributed to debugging and finalizing the front-end.

Meeting Notes

25 January: Met Sareh for the first time and got an overview of the project, clarified details about the project that we were not sure of, such as the layout of the site.

1 February: First scheduled meeting with the client, got some additional guidance with the client and got an alternative to the anonymizer code that currently isn’t working because it seems to be missing dependencies that may have been updated since it last run correctly. We attempted to troubleshoot but are going to talk to Satvik about possible solutions.

8 February: Met with the client and showed her what we have gotten up to this point. Tyler showed placeholders for the chatbot interaction page, complete with chat bar and history placeholders. A note from the client on this page explains that she wants an additional button

underneath the chat bar to toggle other EFT cues. Aaron showed the client the back-end work that had been completed so far. Patrick showed the output from the scrubadub Python library and shared it with Tyler on Discord [11]. Sareh showed the team the chatbot, displaying how it works and what it is supposed to accomplish. The current state of the interactions with the EFT chatbot is nonexistent because of a lack of endpoint connections between the site and chatbot, looking to connect a local chatbot for testing.

13 February: Class meeting. Going over the progress of the team. Front-end: working placeholders for the front-end. Back-end: still waiting on chatbot endpoint to connect the back-end to the chatbot endpoint. Report: rough idea for the abstract and introduction sections. Make a GPT-4 call on Azure, Qualtrics runs as an iframe; it does its thing and comes back.

15 February: Weekly meeting with client: showed the survey page, blank at the time. The client wanted the team to figure out how to get the Qualtrics page, for the user survey, connected to the site. Scheduled another meeting with someone who could help. Got the page to work, and for the time being placed the Qualtrics URL directly on the page.

20 February: In class discussion: Went over the feedback from Presentation 1, Barton is tasked with fixing the common issues with the slides. Aaron wanted to ditch React, but it's too late to make any major changes like that, so we're going to continue with what we have. Talked about how to store the user and chatbot conversations, decided to store everything as a string in the database. We also talked about the progress on the rough draft of the report.

22 February: Weekly meeting: Got clarification on changes Dr. Fox discussed with the team on Tuesday, nothing changed contrary to what he had mentioned during class on Tuesday. We scheduled another meeting with the Qualtrics expert, Dr. Allison Tegge. Talked about getting the unique ID from Qualtrics from the User to store those responses along with storing the users' data.

29 February: Weekly meeting: Talked about the report and getting the UI deployed to the server the site is supposed to run on. Need to have Ollama and Docker installed on that server for everything to run [14, 13]. Showed Sareh the current page, but everything we showed was not on the server that Sareh wants it on; she hoped to have that completed before Spring Break, but it must be pushed to during break. However, we had it working on a local server that Aaron set up to test how the site works on a container like EFT server.

9 March: Meeting before the presentation, talked about what needs to be added to the slides, mainly the work accomplished by Tyler and Aaron over break. They were able to get the site running on the Virginia Tech EFT server.

14 March: Weekly team meeting, issues with signup page but determined it was because of multiple users already in the system. Learned that again things changed with the project, need to look at the updated UML that Dr. Fox made to see the changes. Need to look at the stuff that Sareh sent but again it might change again so standby for the moment. Asked the team to do some research on Azure and Kubernetes [18, 19].

21 March: Dr. Alison Tegge, our Qualtrics expert, attends the weekly meeting. Aaron showed Dr. Tegge what was going on when we ran the curl command that she gave us. Sareh is asking if we need to store the unique ID, which we already are. Talked about using the unique identifier as their username for the website, instead of passing a variable like this, contact="someUniqueID", in the URL of the page, so we can just grab the unique survey ID when the user makes an account.

28 March: We explained what was completed this week between the front-end and back-end to Sareh. There was an issue with the signup page and we're working on fixing that. We agreed that the page needs to be redone to make it more useable.

4 April: Tyler shared the updates with the signup page and the ability to hide the side bar stuff. He also let her know that the site is now more readable on mobile if the user wants to access the site on their cell phone. Aaron showed the back-end where he fixed the timestamp issues; before it was giving the timestamp in different format. He also showed a lot of comments within the code itself, hopefully making understanding the code easier. He also explained the error output; he simplified it so the errors would be more helpful to find and explain what's going on. Sareh shared her chatbot; we haven't had a chance to integrate it with the site yet. She explained some of the prompts she provided to the chatbot and explained the LangChain library she used to concatenate strings [12]. We discussed potential issues if the site goes down and what we would do to fix it, e.g., error message 500 would be an OpenAI issue. We would have to either simply reload the page or might have to wait on whatever OpenAI's issues are. One solution that Aaron came up with to deal with this is to regenerate the previous chat only if the site gets disconnected using the chat template. Sareh asked us to go back to Mistral to test a more powerful model again.

11 April: Showing the client how the site looks with Ollama still. Informed her that a development and deployment mode has been set up; the major difference between them is hiding the menu bar in the header to prevent the user from freely clicking around the site. Aaron is showing the client how the extracting will look but we haven't run it on the site because of cost concerns but Sareh informed us that cost wouldn't be an issue, just to run it. Now we're going to swap testing the OpenAI chatbot (the client's) on the site. She said not to worry but make sure that the EFT cue is being sent to Qualtrics and wants the user to see the EFTs generated. She wants to make sure that the user can respond to the EFT cues properly. She wants us to add functionality to ensure that if the user types in something inappropriate or outside the scope of the usage to say nothing. Gave us the openai-cookbook to create a template to prevent the user from giving an inappropriate response [15].

18 April: Attempting to get any last second details completed before the end of the semester. Primarily going over the report with Sareh and what she would like to see changed in the Developer's manual and addressing any of the outdated links and commands that are there.

25 April: We are still experiencing issues with the EFT server when the client is on campus. I've given Sareh the command to SSH into the server using the server's ports: "ssh -L 5000:localhost:5000 -L 3000:localhost:3000 eft.cs.vt.edu". However, if these ports are already in

use by another developer, the .config file needs to be updated to ensure the ports bind properly. When in developer mode, remove the 'static' from the URL, as 'static' is used for loading the page. We discussed the contents of the README, particularly the “Building” section. Aaron is currently updating this section to make it clearer. Sareh wanted to observe the extracted cues working from her directory to understand how they are being stored. Aaron interacted with the chatbot to demonstrate what the storage JSONs contain. We are still facing some challenges in capturing user responses at the end of sessions. Each user rates events (or questions) on a scale of 1 to 5. After generating an EFT cue, Aaron instructed the chatbot not to ask the questions immediately, which successfully generated the cue. Sareh now intends to extract this cue and send it to Qualtrics. We plan to make a change so that once the chatbot has generated seven EFT cues, we will add a message in the textbox on the chatbot page stating, “Thank you for using the EFT Chatbot. Please review your generated EFTs and click Continue when you’re ready to proceed.” The textbox will be disabled to prevent further user interaction with the chatbot. We are also trying to handle edge cases where the user refreshes the page and are still figuring out the best approach. Before next week, we need to address several items: change the name of the chatbot from “Assistant” to “EFTeacher”, check the resend function, the continue-left message, extract cues at the end of conversations, navigate to the next page at the end, check the event timeline, log errors, send data to Qualtrics, and fix the cue regex.

30 April: Fixed the SSH issue encountered when signing onto the campus internet and updated the README and developer’s manual to include this fix. We discussed eliminating the signup page altogether due to the new method we're using to get users to sign into the site. However, we decided it would be better to keep it for development purposes. Added a generator script for automatically creating user accounts and passwords before they enter the site. We may add two different commands to the developer’s manual, allowing the client to make changes while the site is live, to avoid this issue. The production ports are 443 for HTTPS and 80 for HTTP, the latter being used first when the site is deployed to the EFT server.

Requirements

This section will outline the required technologies used to implement the project. A quick description of each required page will be provided.

Front-end
Our front-end development relies on *React JS,* known for its component-based structure and efficient state management. Using React helps us create modular UI components, making code more manageable and reusable. The virtual DOM ensures a responsive interface for a seamless interaction with the website [7].

For visual aesthetics and responsiveness, we use *Cascading Style Sheets (CSS)* to style and layout the webpages [10]. Responsive design, achieved through media queries, ensures a consistent user experience on different devices. We also use the CSS preprocessor and framework, Material-UI to streamline styling, resulting in a visually cohesive front-end. We use Material-UI primarily because it was part of the existing codebase, so it was easier to continue with it [20].

The client requested five front-end pages to be completed as part of the project's UI; signup, login, user survey, chatbot interaction, and usability survey. These are listed below:

Signup Page
For the signup page, it checks if the username that the user puts in doesn't already exist in the system and that the user doesn't already have an account in general. They must provide a name, email address, and password for their account before they can gain access to the site. This page is only for developer testing.

Login Page
The login page enforces a password for the associated username. The usernames are from the pre-generated username and password comma-separated value (CSV) file. In development mode, the usernames can be the user's email.

Chatbot Page
The chatbot interaction page has several different components. From left to right on the page. On the far left of the page, the user's status is displayed. This will allow the user to fully understand the tasks required to finish their interaction with our website. The top center of the page will hold the current chat that the user is having with the bot; the user will be able to scroll up and down through this chat. The bottom center of the page will be where the user can interact with the chatbot; the user will be able to type in a response to the chatbot. Above the input box will be a space with the ability to scroll through the generated EFTs.

Usability Assessment Page
The last page, the usability survey, has the system usability survey. The questions come from John Brooke's usability scale [21] After completion of this page the user and EFT cues are sent to Qualtrics.

Back-end

Python was selected as the back-end programming language due to its readability, extensive library support, and ease of integration.

The back-end is tasked with managing connecting logic to the Qualtrics API, data processing, communication with the database, and connecting the chatbot and APIs. APIs, designed to facilitate seamless interaction between the front-end and back-end, serve as a crucial component. Python's versatility enables integration with external services and APIs, enriching the applications functionality.

The separation of concerns between the front-end and back-end, with a well-defined API layer, ensures parallel development and promotes collaboration among team members. This approach enhances the scalability, maintainability, and overall efficiency of the project, aligning with modern best practices in web development [24].

Ollama is a tool that allows us to run open-source large language models (LLMs) locally. The team debated on two of the models, Mistral and Llama2 [22, 23]. Ollama was used as a temporary fix to the delay in getting access to OpenAI.

The *Clients Chatbot* runs off *OpenAI's* GPT-4. It generates EFT cues based on the prompt provided by the client.

User Manual

This section outlines the intended flow of the website. This is the current version of the website that will change based on the studies requirements.

EFT Chatbot URL

The site can be found at: https://eft.cs.vt.edu/

Login Page

The user will first land on the login page, Figure 1. They will need to provide the username and password given to them when they sign up at Qualtrics and consent to participate in the study. The username and password they are provided will already be in the database. This page can be skipped entirely when sent from Qualtrics. It is intended to only be used if the user leaves the session to continue it at another time.

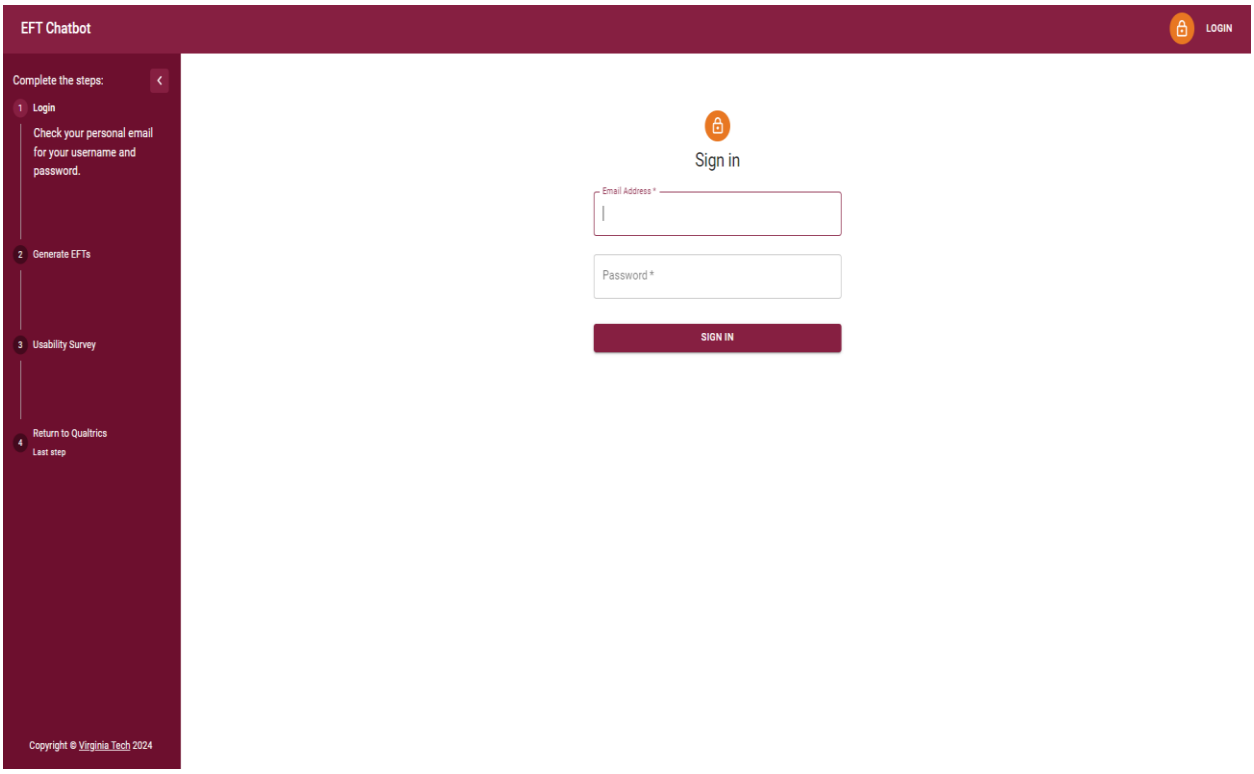

Figure 1: Login page

Home Page

After the user has finished with the Qualtrics survey they will be sent to the home page of the website. This page is where the user can start the interaction by pressing the start message. The user should answer the questions provided by the client's prompt. Figure 2 shows an interaction with the chatbot. The box above the input stores the EFT cues. This allows the user to easily access their EFT cues. After the set number of EFT cues have been generated, the input box will become disabled and a continue button will appear. The user will be redirected to the usability survey once this button is pressed. This can be seen in Figure 2.

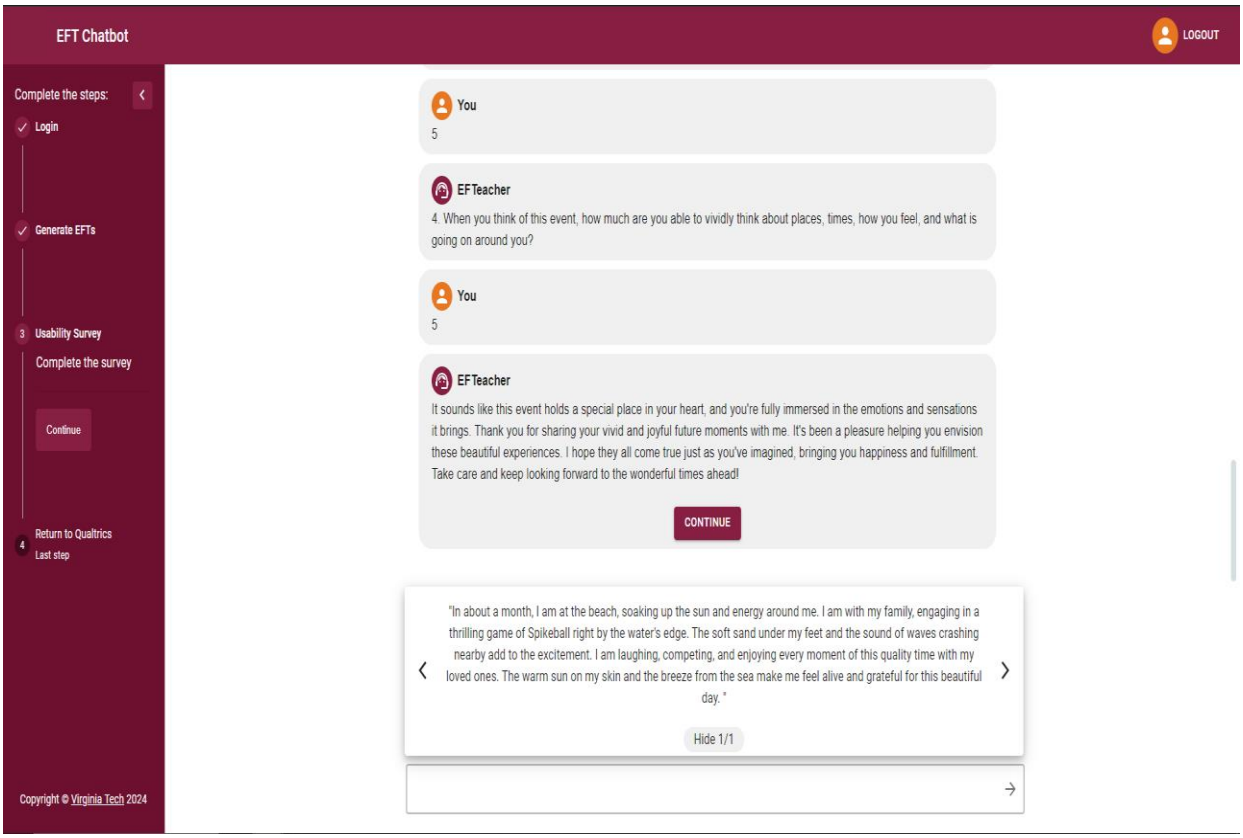

Figure 2: Chatbot Interaction Page

Usability Assessment Page

The usability assessment page is landed on after the user is finished with the chatbot. Figure 3 shows the survey consisting of five of the ten Likert styled questions and a submission button. Once the submission button is pressed, the user is redirected back to Qualtrics, and the extracted EFT cues are sent.

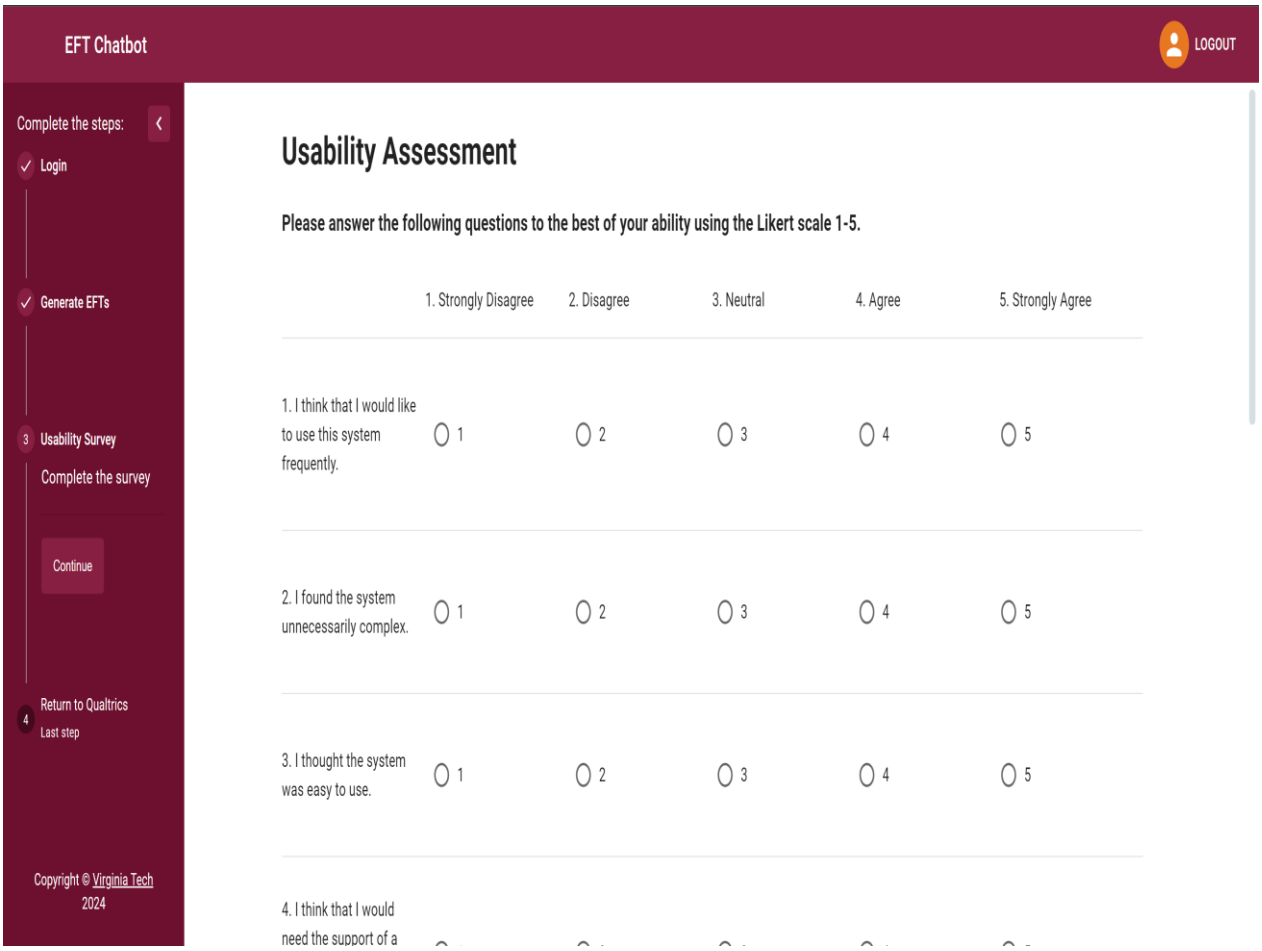

Figure 3: Usability Assessment

Testing

Testing will primarily be in the form of use case tests, that will show the functionality of both the front-end and back-end of the site. Each page will have specific criteria to pass the tests to ensure that the page is functioning properly. For the signup page, to pass the test the site needs to check the database to see if there is a matching username in the database, meaning they have an account. After the user creates the account, we show that the user successfully created an account. We then need to create unit tests to make sure that its functionality is working for future development.

The login page needs to enforce the password provided with the given username tests would include functionality and unit tests, providing both correct and incorrect usernames and passwords. In the case of the correct username and password combination then access would be granted and if not, a message will tell the user that an incorrect username/password was given and offer either to put in another username/password or a password reset option. To ensure that the username and password is correct unit tests would confirm that the back-end is correct, because the site will tell the user that it could be the username or password that is incorrect. Unit tests would include both correct and incorrect username and passwords combinations but only the correct combination would give access.

The chatbot interaction will require the most amount of testing because it has a lot of important components, the main chat window in the middle, the text box at the bottom, and EFT cues. The main window will display both the user responses and the response of the chatbot. The text box down below will accept user input but will not autofill or predict the next words. The user will then be able to hit either enter on the keyboard to submit a prompt to the chatbot or click the button inside the box to submit. Testing the EFT cue extraction will be important to ensure that the end of conversation is handled correctly. After the set number of cues is generated, the user shouldn’t be allowed to send anymore messages and only have the option to continue.

The last page will be the usability survey, the boxes that the user selects as their choice must highlight and after the survey all the answers must be recorded and sent to the back-end. The extracted cues and suer should be redirected and sent to Qualtrics.

Testing the back-end will require only unit testing, first ensuring that the endpoints between the chatbot and the back-end are connected. The tests would also include whether the user exists in the back-end portion and whether they have a valid username and password. To further test this we would supply an incorrect password to an account in the database, our system should be able to find the username and check that the passwords do not match and deny access. When the cues are generated, the back-end needs to ensure that there are only seven generated, this can be done by generating the cues and simply counting how many are stored in the back-end and ending the conversation with the user. Along with that the back-end needs to ensure that they’re stored as they get generated and can be exported to the client for research purposes.

Developer's Manual

The code for this project can be divided into front-end and back-end. The code can be found on the private repository at: https://git.cs.vt.edu/aargmz02/ai-chatbot-website

Unified Modeling Language

Figure 4 shows the Unified Modeling Language, UML, for this project and the workflow. This UML is for the current version and might differ depending on the studies requirements.

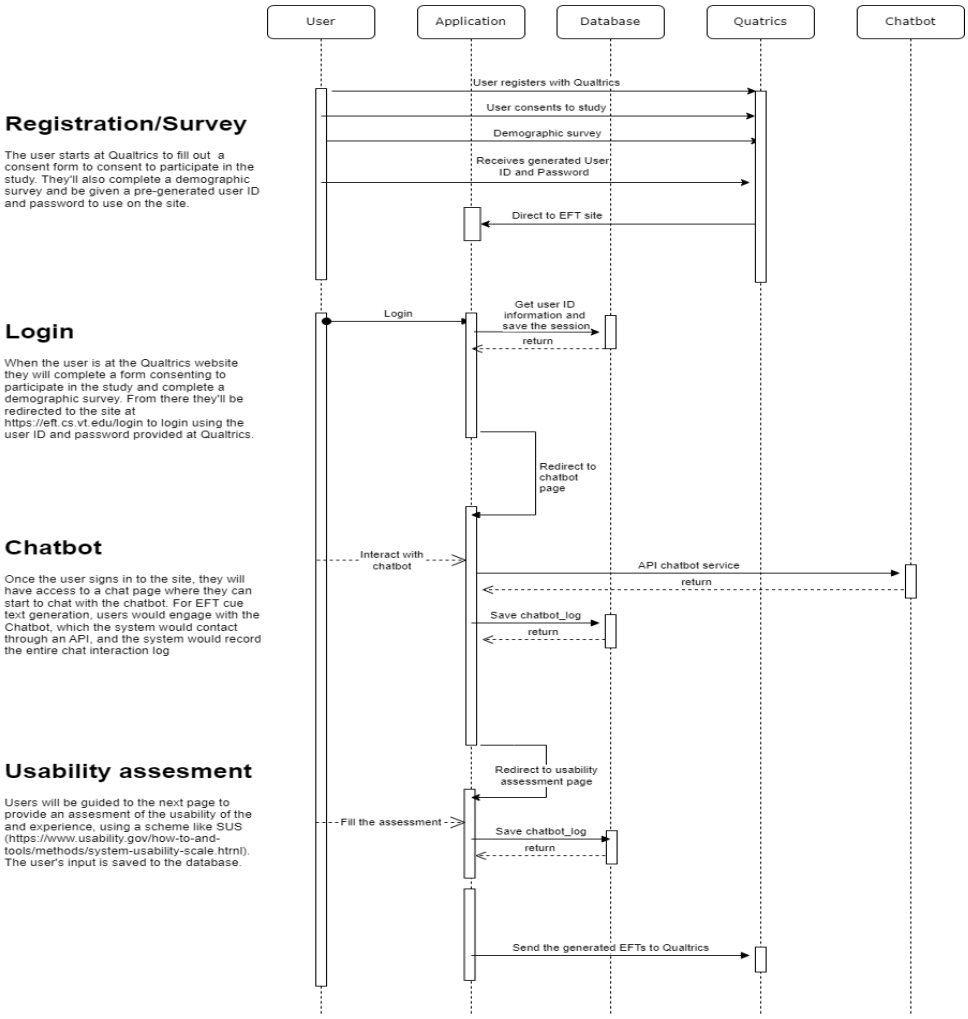

Figure 4: Project UML and flow

Front-end

The front-end is created in ReactJS. The main pages and components with a description can be found in Table 1.

No.	File Name	Purpose
1	Footer.js	Displays footer content on all other pages.
2	Header.js	The Header contains Menu, links, and Login Icon.
3	Home.js - components	Logic behind the Home Page displays chatbot conversations. Handles resend message if message fails to send.
4	Message.js	Core components of a single message, used to also display message errors.
5	Sidebar.js	Contains the code for the sidebar and calls the VerticalLinearStepper component.
6	UsabilityQuestion.js	Contains the code for an individual question in the usability survey.
7	UserSurvey.js	Contains the logic for the Qualtrics consent.
8	VerticalLinearStepper.js	Keeps the user going throughout the site as intended, if they missed something they'll be redirected based on the stepper value.
9	About.js	Provides an introduction and privacy information.
10	Home.js - pages	Calls the home component and header.
11	Login.js	This is the entry to the system, when users are authenticated, they can interact with the system.
12	QualtricsSurvey.js	File contains the sidebar, header, and UserSurvey component.
13	Signup.js	Stores the user's first and last name, the email address, and password (that has to be confirmed) into the database
14	UsabilityAssessment.js	Holds the questions for the usability assessment, assigns numbers to rate their experience
15	App.js	This file renders all pages to display on the browser.
16	Constants.js	The constants file contains API links.
17	Index.js	Renders the theme for the site (Virginia Tech colors).
18	Theme.js	Sets the colors of Virginia Tech.
19	__init__.py	Initializes all the back-end and contains the API endpoints.
20	Authenticator.py	Checks if an account already exists or if either the username or password doesn't match anything in the system.
21	Database.py	Stores the information provided by the user, and the communication between the chatbot
22	llm_clients.py	Connects the LLMs to the site, both Ollama and Client's chatbot.

Table 1: Front-end files

The front-end has a .env file that contains the lines:

```
REACT_APP_DEVELOPMENT=Boolean

REACT_APP_EFT_CUES_NUM=Integer
```

The Boolean value should be true when developing and false when deploying. This allows the developer to bypass the automatic redirects and navigate freely.

The Integer determines the number of expected EFT cues to be generated. For easier testing, make the number smaller. For deployment, the number should be set to 7.

Back-end

The back-end has four database tables.

Table 2 shows the values associated with a User. Messages stores the list of messages associated with the user. Eft_cues store the extracted EFT cues. The usability_assessment stores a string associated with the responses to the system usability scale survey.

User	
id	String (primary key)
password_hash	String
survey_id	String
messages	List[Message]
eft_cues	List[Cue]
usability_assessment	String

Table 2: User data class

Table 3 shows the Message data class. Message is a class that stores the required information for a message. The user_id associates a Message with a User id. The role specifies if the Message was from the user or bot.

Message	
id	String (primary key)
user_id	String (foreign key)
content	String
role	String
timestamp	Integer

Table 3: Message data class

Table 4 contains the information associated with a Cue. The user_id associates the Cue with a User. It also has a timeframe value that stores what timeframe the cue is from. This is part of the EFT cue generation.

Cue	
id	String (primary key)
user_id	String (foreign key)
content	String
timeframe	String

Table 4: Cue data class

Table 5 demonstrates the Event class. The Event class is used to store all events that happen on the website. The user_id associates the Event with a User. The string Action stores the said action. This might include signing out, completing Qualtrics survey, or sending a message.

Event	
id	String (primary key)
user_id	String (foreign key)
action	String
timestamp	Datetime

Table 5: Event data class

The back-end has a .env file in the root of the back-end directory. The values in the .env file are in Table 6.

Variable name	Description
STORAGE_ROOT	Path to the backend server's local storage
DATABASE_URL	URL of the SQL database to use
EFT_CUE_TARGET	The amount of EFT cues to generate for users
LLM_CLIENT	Name of the LLM client to use, either `ollama` or `azure-openai`
OLLAMA_ENDPOINT	URL of the Ollama API endpoint to use (subdirectories unsupported, see Issue #113)
OLLAMA_MODEL	Name of the Ollama model to use, also see Ollama model library
AZURE_OPENAI_ENDPOINT	URL of the Azure OpenAI REST API endpoint to use
AZURE_OPENAI_API_KEY	API key for accessing the Azure OpenAI API
AZURE_OPENAI_API_VERSION	Version of the OpenAI API to use, also see REST API versioning
AZURE_OPENAI_MODEL	Name of the Azure OpenAI model to use
QUALTRICS_ENDPOINT	URL of the Qualtrics API endpoint to use
QUALTRICS_API_KEY	API key for accessing the Qualtrics API
QUALTRICS_DIRECTORY_ID	ID of the directory/pool for the mailing list used on Qualtrics
QUALTRICS_MAILING_LIST_ID	ID of the mailing list to store contacts in on Qualtrics

Table 6: Back-end .env setup

Development

These instructions are to get a copy of the project running on your machine for development/testing or to build the Docker image from source.

1. **Start by cloning the project**

```
git clone https://git.cs.vt.edu/aargmz02/ai-chatbot-website.git
```

2. **Check for dependencies**

- Python 3.9+

```
python --version
```

- Poetry 1.71+
    - Using the official installer script is recommended, as this will install the latest version of Poetry.
    - https://python-poetry.org/docs/#installing-with-the-official-installer
    - Depending on your system configuration, it may be necessary to add $HOME/.local/bin to your $PATH, which can typically be done in your shell profile script. As an example, Bash users can add the line export PATH=$PATH:$HOME/.local/bin to the file ~/.bash_profile.

```
poetry --version
```

- Node.js 20+
    - Using the v20.X.X (LTS) release is recommended.

```
node -v
```

3. **Install dependencies**

I. Navigate into the back-end directory.
II. Install the Quart server's required dependencies:

```
poetry install
```

III. Navigate into the front-end directory.
IV. Install the React app's required dependencies:

```
npm install
```

4. **Running the development servers**

Before running the development servers, be sure all necessary configuration is complete.

I. Quart: Run the below command in the back-end directory.

```
poetry run start
```

II. React: Run the below command in the front-end directory.

```
npm run start
```

Building

Once the appropriate dependencies have been installed, the project can be built using Make. Several targets are available for building the project and can be run with the command `make <target-name>`.

The provided Makefile includes:

- `all`: (Default) Builds the `eft-website` Docker image and the necessary front-end/back-end files.
- `Back-end`: Generates `requirements.txt` from the back-end's Poetry dependencies.
- `front-end`: Builds static React files for the website front-end.
- `docker-build`: Builds the `eft-website` Docker image, without automatically building the front-end/back-end.
- `docker-export`: Exports the saved `eft-website` Docker image to `eft-website.tar`, deleting the existing file first if necessary.
- `clean`: Removes `requirements.txt`, React build files, and `eft-website.tar`.

If you wish to use the exported Docker image on another machine, it can be imported with the command:

```
docker image load -i </path/to/eft-website.tar>
```

Signup Page

As seen in Figure 5, the signup page is designed exclusively for developer access and is not available to general users. This dedicated page enables system testing by allowing developers to establish a unique username and password, thus bypassing the standard pre-generated credentials distributed to study participants via Qualtrics. Developers are required to provide their name and an email address, which functions as the username. They must also create a password and confirm it in a separate field to ensure accuracy.

Figure 5 displays the password creation process, highlighting requirements that have been met with a green check mark alongside those yet to be fulfilled, which are indicated by a red 'x'. The text detailing the password requirements will appear in green once all are satisfied. However, any unmet requirement will continue to display in red until the appropriate characters are included. Moreover, if the entered passwords do not match, the system will deny access, preventing the creation of duplicate developer accounts. It is important to note that the system currently does not support a password recovery option, underscoring the importance of precise password entry.

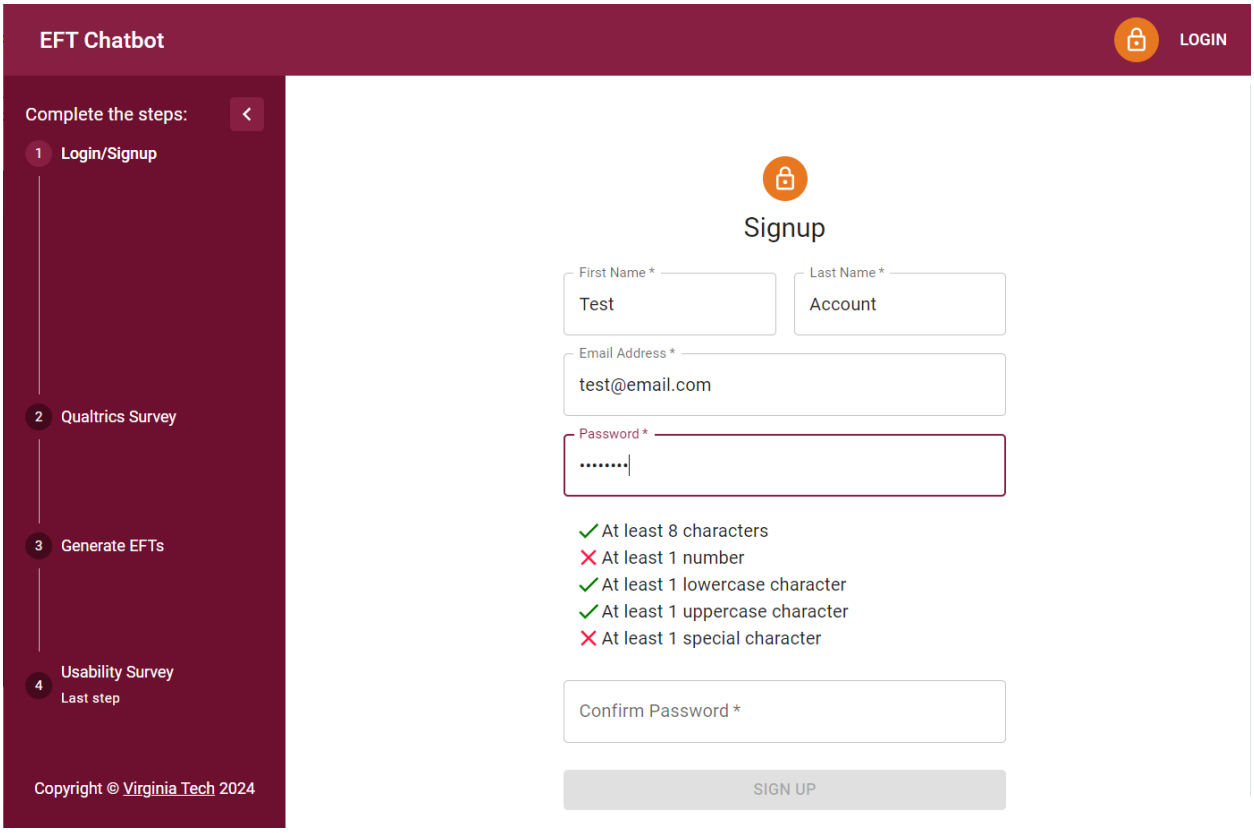

Figure 5: Signup Page for Development

Deployment

This section can be broken into general deployment and specific deployment on the eft.cs.vt.edu.

General Deployment

Docker Compose is the preferred way to deploy this app, as it encapsulates everything needed to serve the website over HTTPS, optionally including a container to use Ollama language models.

A sample `docker-compose.yml` is provided, with placeholders for sensitive values like API keys. This configuration includes a container for a local Ollama instance and uses nginx with Certbot to serve requests over HTTPS. Note that Certbot must first be configured to set up SSL certificates and auto-renewal. See Configuration: Certbot for instructions.

When deployed with a configuration using the provided defaults, the app's directory structure should be similar to Figure 6.

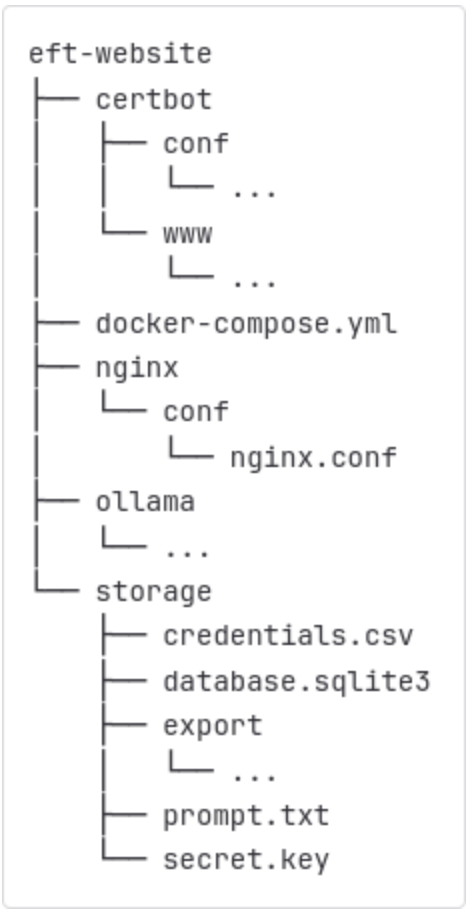

Figure 6: App's directory structure

Specific to eft.cs.vt.edu server
The below steps do not all need to be completed every time. The steps are a general flow to deploy new changes and more specific details can be found in the Readme or above instructions.

1. Establish a connection with sareh@eft.cs.vt.edu. This will require a password that can be provided by the client.

```
ssh sareh@eft.cs.vt.edu
```

2. Navigate to the development directory.

```
cd path/development/ai-chatbot-website/
```

3. Pull the most recent changes.

```
git pull
```

4. Rebuild the image.

```
make all
```

5. Navigate to the production directory.

```
cd ../../production/
```

6. Ensure the image is not running.

```
sudo docker compose down
```

7. Restart all processes.

```
sudo docker compose up -d
```

Certbot

HTTPS is likely desired when running in production, which requires an SSL certificate. Certbot is included and can be used to obtain free SSL certificates from *Let's Encrypt*. Start with a dry run, as *Let's Encrypt* has a limit on the number of free certificates available per month. Be sure to replace 'example.domain.org' in the following command with the desired domain name:

```
docker compose run –rm certbot certonly –webroot –webroot-path
/var/www/certbot/ -d example.domain.org -v –dry-run
```

If the dry run is successful, remove the '--dry-run' flag and run the command again to obtain a real certificate. Once Certbot obtains an SSL certificate, edit the 'nginx/conf/nginx.conf' configuration file and uncomment the 'server' block responsible for HTTPS. If the nginx container is still running, it may be necessary to restart it using 'docker compose restart nginx' (root privileges may be required).

Uncomplicated Firewall

When using an uncomplicated firewall (UFW) as a system firewall, connections to Docker containers are often not properly protected as the iptables Docker manages are fundamentally incompatible with those managed by UFW. This allows incoming connections to bypass UFW's rules and will inadvertently expose services running inside Docker containers to the outside network, an especially dangerous thing for Internet-facing machines. Common solutions include disabling Docker's iptables functionality, though this also removes Docker's ability to provide network management and isolation for containers.

The ufw-docker project solves these issues by modifying one of UFW's configuration files to allow it to correctly protect Docker containers. The ufw-docker project provides a script for modifying UFW's rules, which can be installed with the following commands (root privileges may be required)[26]:

```
sudo wget -O /usr/local/bin/ufw-docker
https://github.com/chaifeng/ufw-docker/raw/master/ufw-docker

sudo chmod +x /usr/local/bin/ufw-docker
```

After installing the script, update UFW's rules by running `sudo ufw-docker install`, then run `sudo ufw reload` to load the changes (root privileges may be required). The nginx container used when running this project in production exposes the website on TCP ports 80 and 443 (HTTP and HTTPS). These ports can be exposed for any running service with (root privileges may be required):

```
sudo ufw allow http
sudo ufw allow https
```

If it is instead preferred to expose these ports only when they are bound to the running nginx container, a different command provided by ufw-docker can be used (root privileges may be required):

```
sudo ufw-docker allow eft-website-nginx-1 80/tcp
sudo ufw-docker allow eft-website-nginx-1 443/tcp
```

Note that this UFW rule will only work if the nginx container keeps the same IP address, which often is not the case if the container is recreated, or the Docker daemon restarted. The Docker Compose configuration this project uses assigns the static IP `10.1.56.56` to the nginx container by default, so no extra work needs to be done.

Networking

Docker comes pre-configured to assign addresses from the 172.17.0.0/12 subnet to its containers. On a machine connected to a network that also uses IP addresses in the same subnet (such as Virginia Tech's network), the system will ignore incoming external requests from these addresses, which may prevent access for some users. To avoid this, configure Docker to assign container IP addresses in an unused subnet by creating/editing the file `/etc/docker/daemon.json` with the following contents:

```
{
  "default-address-pools": [
    {"base": "10.1.0.0/16", "size": 24}
  ]
}
```

This configures Docker to assign IP addresses from the `10.1.0.0/16` subnet, which should no longer conflict with external network addresses in the `172.17.0.0/12` range. After editing the configuration file, make sure to reload the Docker daemon, which can be done with `sudo systemctl restart docker` on machines using systemd.

Once configured, the site's services can be started/stopped with Docker Compose commands shown in Table 7 (root privileges may be required).

Command	Description
`docker compose up -d`	Start services in background
`docker compose up`	Start services in foreground (press Ctrl+C to exit and stop)
`docker compose down`	Stop services
`Docker compose restart <service-name>`	Restart a specific service defined in 'docker-compose.yml'

Table 7: Docker Compose Commands

When services are running, the website should be available on the standard HTTP/HTTPS ports. By default, 'nginx' is configured to serve the website over HTTPS and will redirect insecure HTTP connections when needed. If you are unable to access the website and have confirmed it is running, check that your server/network's firewall allows incoming traffic to TCP ports 80 and 443 (HTTP and HTTPS). If the system is using UFW as the firewall, check the UFW section for instructions.

When changes are made to `docker-compose.yml` or a new container image is available after building or importing, the currently running service containers can be recreated by running Docker Compose's 'up' command again. If changes are made to other configuration files or the SQLite database file, it may be faster to restart a specific service's container instead.

Exporting

The SQL database's contents can be exported to files in JSON format, which should be helpful if additional parsing/processing of user data is necessary.

When running the project in a development environment, the database can be exported with the command ‘`poetry run export`’.
In a production environment using Docker Compose for deployment, Poetry is not available, and a different command can be used instead:

```
docker compose exec eft-website python3 . export
```

Files are stored within an `export` folder located in the app's storage directory. A unique folder is created for each registered user. Each user-specific folder contains four JSON files, the details of which are provided in Table 8. The structure of these files are depicted in Figure 7.

JSON file name	**Description**
`account.json`	Conversation-unrelated user info such as the login and survey IDs
`eft_cues.json`	Extracted EFT cues from a user's chatbot interaction
`events.json`	List of each logged event from this user
`messages.json`	All messages from a user's conversation with the chatbot
`ratings.json`	Rating question responses extracted from a user's conversation
`usability.json`	Usability assessment responses

Table 8: JSON files and Descriptions

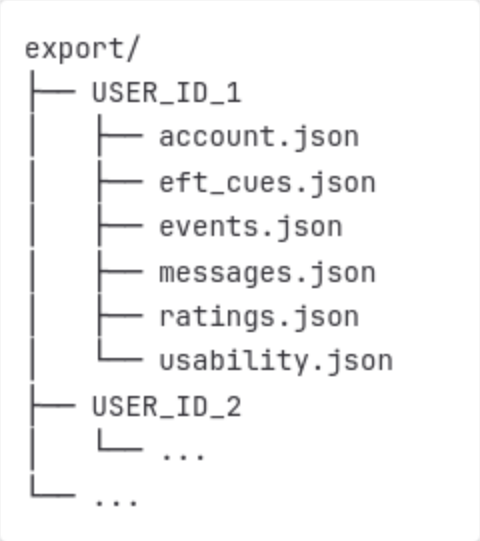

Figure 7: Exported File Structure

Viewing Exported Data

After exporting the data, it may be necessary to view the raw data of a specific user. Raw JSON data can be difficult to work with directly, so a few convenience scripts are provided in the 'scripts' directory:

Script file name	Description
`extract_eft_cues.py`	Manually extract EFT cues from a conversation using the same method as the Quart server. Takes the path to a source `messages.json` file and the path to a destination `eft cues.json` file as arguments.
`view_eft_cues.py`	View EFT cues extracted from a conversation, displaying timeframes and cue contents. Takes the path to an `eft_cues.json` file as an argument.
`view_messages.py`	View an exported conversation, displaying roles, timestamps, and message contents. Takes the path to a `messages.json` file as an argument.

Table 9: Data viewing scripts and descriptions

Creating Passwords for Qualtrics

Find GenerateEFTPasswords.py in the back-end and run the file with:

```
python \GenerateEFTPasswords.py <name> <min> <max> <output-file>
```

This will generate a CSV with Username and Password pairs.

- <name> is the username for the new users.
- <min> is the lower bound inclusive for number of users.
- <max> is the upper bound inclusive.
- <output-file> is the name of the output CSV file.

To preload the file into the database, add the file generated credentials.csv into the storage directory. At the start of the server, it will be preloaded.

Lessons Learned

Timeline

January:

Pick the project and meet with the client and get clarifications about the project.

February:

1st: Look at the anonymizer code that the GTA sent (this will be to hide personally identifiable information that the user shares).

3rd: Ensure the team has the environment and dependencies installed on VisualStudio Code.

11th: Have the prototype of the site completed can just be very basic functionalities.

22nd: Have the functionality tests completed for the site, including the signup page creating accounts (may or may not be able to check if the user already has an account). The sign-up page forces a password to sign in. User survey redirects the user to the Qualtrics site, the third-party survey. The chatbot can interact with the chatbot interaction page. The usability survey page has something on it.

March:

1st: Rough draft of the report is completed.

2nd: The front-end UI works as intended but doesn't have to look very nice.

15th: Decided not to include anonymizer in the project.

April:

1st: The final draft of the report is sent to the client to ensure things are not missed.

4th: Try to connect clients chatbot to the website.

12th: The final draft of the report is checked by the group.

15th: Final draft due to VTechWorks.

22nd: Final presentation

May:

3rd: Final adjustments made to the UI completed.

Problems and Solutions

Qualtrics is a third-party site that allows users to complete surveys on a variety of topics, and because of that versatility our client has decided to use them to perform the user survey portion of the site. To begin with the team was having trouble directing the user to the Qualtrics page and getting them back to the site with the completed survey. The first solution was to just put the link within the components of the React framework, direct them to the page and allow them to come back. However, the issue was that there was no way to determine if the user completed the survey and sent that information back to the site.

Dr. Fox, the advisor of our client, suggested that we use an iframe to solve this issue. An iframe, or inline frame, is an HTML element that loads another HTML page within a document. It's essentially a webpage within a parent page that allows us to implant the survey directly into the site and on the back-end, we only need to store the ID of the survey within the database, so we know that the user has completed the survey.

Using an iframe were the initial fix that allowed us to continue developing the website. We were then put in contact with Dr. Allison Tegge, a Qualtrics expert for the team, and she created a URL that allowed us to directly put the form into the site using an inline anchor, *href*, which ended up being getting the demographic survey to the site [25].

Within the last few weeks of the project, the requirement for Qualtrics had changed again. The website needed to receive the user from Qualtrics with their given Username and Password. At the end of the user's interaction, they would then be sent back to Qualtrics alongside all the extracted EFT cues. This is currently the solution to solve this problem.

Future Work

Forgot password

The current system has no way to recover or reset the password. Currently our solution is that the user would have to contact a systems administrator to retrieve their password that's in the system. This might hurt the possibility of users wanting to continue to use the service that the chatbot provides.

Testing with real users

To enhance the quality and usability of the site, securing access to unbiased testers is important. Testing exclusively within the development team can often miss critical flaws, as developers may overlook issues that are not immediately obvious to them. Engaging with real users from our target demographic will provide insights and uncover potential problems that could affect user experience. This approach will ensure that the platform is robust and tailored to meet the actual needs of the users.

Continue Maintenance

Further use of the website might expose possible issues that have not surfaced. The documentation in this paper and repository should allow for future bug fixes and changes to be completed by other developers. Testing the use of the chatbot from the beginning of Qualtrics to the end of Qualtrics could lead to possible overlooked errors that might need addressed.

Acknowledgments

Client: Sareh Ahmadi PhD Student

Sareh is the client for this project and is a Ph.D. student at Virginia Tech in the Department of Computer Science. She developed the chatbot prompt that guides the creation of the Episodic Future Thinking cues and manages conversations with the users.

saraahmadi@vt.edu

Professor: Dr. Edward A. Fox

Advisor for Sareh and instructor of CS4624: Multimedia, Hypertext, and Information Access. He's a researcher, Director of the Digital Library Research Laboratory, and Executive Director of the Networked Digital Library of Theses and Dissertations.

fox@vt.edu

Graduate Teaching Assistant: Satvik Chekuri

Satvik Chekuri is a Ph.D. student in the Department of Computer Science. His advisor is Edward Fox. Chekuri's research focuses on developing knowledge-graphs for scholarly documents such as ETDs (electronic versions of theses and dissertations) to aid in downstream tasks including question-answering, semantic querying, and recommendations.

satvikchekuri@vt.edu

References

[1] Jeremiah Michael Brown and Jeffrey Scott Stein. Putting prospection into practice: Methodological considerations in the use of episodic future thinking to reduce delay discounting and maladaptive health behaviors. Frontiers in Public Health, 10:1020171, 2022. Accessed 5 Mar. 2024.

[2] Madison Milne-Ives, Caroline de Cock, Ernest Lim, Melissa Harper Shehadeh, Nick de Pennington, Guy Mole, et al. The effectiveness of artificial intelligence conversational agents in health care: systematic review. Journal of medical Internet research, 22(10):e20346, 2020. Accessed 6 Mar. 2024.

[3] Tilman Dingler, Dominika Kwasnicka, Jing Wei, Enying Gong, and Brian Oldenburg. The use and promise of conversational agents in digital health. Yearbook of Medical Informatics, 30(01):191–199, 2021. Accessed 7 Mar. 2024.

[4] Cristina M Atance and Daniela K O'Neill. Episodic future thinking. Trends in cognitive sciences, 5(12):533–539, 2001. Accessed 7 Mar. 2024.

[5] Abhishek Aggarwal, Cheuk Chi Tam, Dezhi Wu, Xiaoming Li, and Shan Qiao. Artificial intelligence–based chatbots for promoting health behavioral changes: Systematic review. Journal of Medical Internet Research, 25:e40789, 2023. Accessed 7 Mar. 2024.

[6] Hassenzahl, M., & Tractinsky, N. (2006). User experience - A research agenda. Behaviour & Information Technology, 25(2), 91–97. https://doi.org/10.1080/01449290500330331. Accessed 7 Mar. 2024.

[7] Chen, Songtao, Upendar Rao Thaduri, and Venkata Koteswara Rao Ballamudi. "Front-End Development in React: An Overview." *Engineering International* 7.2 (2019): 117-126. Accessed 8 Mar. 2024.

[8] McGranahan, M. K., & Ubur, S. (2023). (rep.). (S. Ahmadi & E. A. Fox, Eds.) *Final Report CS 5604: Information Storage and Retrieval*. Blacksburg, VA: Virginia Polytechnic Institute and State University. Accessed 9 Apr. 2024.

[9] Tamimi, Arin. *Chatting With Confidence: A Review On The Impact Of User Interface, Trust, And User Experience In Chatbots, And A Proposal Of A Redesigned Prototype*, North Dakota State University of Agriculture and Applied Science, Apr. 2023, https://library.ndsu.edu/ir/handle/10365/33240. Accessed 9 Apr. 2024.

[10] Perrier, J.-Y., & Willee, H., *et al.* (2024). *CSS: Cascading style sheets: MDN*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/CSS. Accessed 10 Apr. 2024.

[11] Malmgren, Dean. *Scrubadub*, 2021, scrubadub.readthedocs.io/en/stable/. Accessed 10 Apr. 2024.

[12] Chase, H., & Friis, E., *et al.* (2024). "Introduction." *Langchain*, 2024, https://python.langchain.com/docs/get_started/introduction/. Accessed 10 Apr. 2024.

[13] Tiigi, T., Chadwell, J., *et al. Build*. Docker Build. (2024, March 7). https://www.docker.com/#build, Accessed 10 Apr. 2024.

[14] Morgan, J., & Yang, M., *et al*. "Ollama/Ollama: Get up and Running with Llama 2, Mistral, Gemma, and Other Large Language Models." *GitHub*, github.com/ollama/ollama. Accessed 11 Apr. 2024.

[15] Pratico, Krista. "OpenAI-Cookbook/Examples/Azure/Archive/Chat.Ipynb at Main · OpenAI/OpenAI-Cookbook." *GitHub*, github.com/openai/openai-cookbook/blob/main/examples/azure/archive/chat.ipynb. Accessed 11 Apr. 2024.

[16] Gillis, Alexander S. "What Is a Docker Image? Introduction and Use Cases." *IT Operations*, TechTarget, 14 May 2021, www.techtarget.com/searchitoperations/definition/Docker-image. Accessed 21 Apr. 2024.

[17] Ahmadi, Sareh, and Edward A Fox. "AI Chatbot for Generating Episodic Future Thinking (EFT) Cue Texts for Health." *arXiv Preprint arXiv:2311.06300*, 2023. Accessed 22 Apr. 2024.

[18] Buck, A., & Liebermann, J., *et al*. (2024). *What is azure devops? - azure DevOps*. Azure DevOps | Microsoft Learn. https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops. Accessed 23 Apr. 2024.

[19] Bannister, T., & Teng, Q., *et al. Kubernetes Documentation*. Kubernetes. (2024). https://kubernetes.io/docs/home/. Accessed 23 Apr. 2024.

[20] Tassinari, O., Kovalenko, D., *et al. Material UI: REACT components that implement Material Design*. Material UI: React components that implement Material Design. (2024). https://mui.com/material-ui/. Accessed 23 Apr. 2024.

[21] Brooke, J. (1995, November). (PDF) SUS: A quick and dirty usability scale. https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale. Accessed 24 Apr. 2024.

[22] Morgan, J., & Yang, M., *et al.*, (2024). *Mistral*. Ollama. https://ollama.com/library/mistral. Accessed 24 Apr. 2024.

[23] Morgan, J., & Yang, M., *et al.*, (2024). *LLAMA2*. Ollama. (2024). https://ollama.com/library/llama2. Accessed 24 Apr. 2024.

[24] Achhab, A. (2023, November 5). *Empowering development teams: Achieving seamless backend and frontend collaboration*. Medium. https://alaedev.medium.com/empowering-

development-teams-achieving-seamless-backend-and-frontend-collaboration-fbb1dd864443 Accessed 24 Apr. 2024.

[25] Perrier, J.-Y., & Willee, H., *et al. URL: HREF property - web APIs: MDN*. MDN Web Docs. (2024). https://developer.mozilla.org/en-US/docs/Web/API/URL/href. Accessed 30 Apr. 2024.

[26] Feng, C., & Peshne, A. (2023). *Chaifeng/UFW-docker: To fix the Docker and UFW security flaw without disabling iptables*. GitHub. https://github.com/chaifeng/ufw-docker. Accessed 30 Apr. 2024.