

Florida International University
Knight Foundation School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

Project Title: STARS GPT (AI-Based Tutoring Chat Agent)

Team Members: Vincent Carrancho, Robin Obregon, Nicholas Belschner, Stephanie Torres, Garvee Valcourt

Product Owner(s): Patricia McDermott-Wells

Mentor(s): Andres Lopez

Instructor: Masoud Sadjadi

The MIT License (MIT)

Copyright (c) 2016 *Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Abstract

This web application is a cutting-edge tool designed to enhance the tutoring experience for students/tutors by providing personalized assistance and support. Leveraging the latest advancements in Generative Artificial intelligence, the platform enables students to receive tailored guidance and explanations across various subjects in the Knight Foundation of Computing and Information Sciences. With a user-friendly interface, students can easily seek help with homework assignments, prepare for exams, and clarify concepts. By combining innovative technology with educational expertise, our tutoring web application aims to revolutionize the way students access academic support and resources, ultimately fostering greater academic success and confidence.

Table of Contents

Introduction.....	6
Current System.....	6
Purpose of New System.....	6
User Stories.....	7
Implemented User Stories/Work Items.....	7
Pending User Stories.....	8
Project Plan.....	8
Software Resources.....	8
Sprints Plan.....	9
Sprint 1.....	9
Sprint 2.....	9
Sprint 3.....	9
Sprint 4.....	10
Sprint 5.....	10
Sprint 6.....	10
Sprint 7.....	11
System Design.....	11
Architectural Patterns.....	11
System and Subsystem Decomposition.....	11
Deployment Diagram.....	14
Design Patterns.....	15
System Validation.....	15
Glossary.....	17
Appendix.....	17
Appendix A - UML Diagrams.....	17
Appendix B - User Interface Design.....	18

Appendix C - Sprint Review Reports.....	18
Sprint 1.....	18
Sprint 2.....	19
Sprint 3.....	19
Sprint 4.....	19
Sprint 5.....	19
Sprint 6.....	19
Sprint 7.....	19
Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents.....	19
Installation:.....	19
Frontend: Pre-reqs: Must have Node.js version >= 20 installed.....	19
Backend: Pre-reqs must have Python 3.10+ installed. Highly recommended using a virtual environment.....	19
Developing on Frontend.....	20
Shortcomings and Wishlists.....	20
References.....	21

INTRODUCTION

The application is a chat application that is tailored to FIU KFSCIS course offerings—more specifically, this catered to FIU STARS Tutoring spearheaded by Dr. Patricia McDermott-Wells. The functionality of the app is to help students and STARS tutors alike so that on top of tutoring a student may receive, they are also supplemented by a ChatBot that can help hammer home those concepts.

Current System

As it stands now, there is no currently deployed system. The closest thing to this application's use case is ChatGPT—however, we have found that ChatGPT, more often than not, gives

inaccurate responses for student prompts. Additionally, since ChatGPT is geared towards general intelligence and knowledge, some of the responses, we found, were too broad.

Purpose of New System

This tutoring web application, customized for FIU KFSCIS courses and specifically tailored to FIU STARS Tutoring under the guidance of Dr. Patricia McDermott-Wells, serves a dual purpose of supplementing traditional tutoring methods with AI-driven assistance. This application solves two main problems that was observed by the STARS Tutoring group:

1. Students are using LLMs (Large Language Models) such as ChatGPT and Google's Gemini (formerly known as Bard), in order to give them direct answers to a query they might have. We have found that these queries stem from assignments.
 - a. Students usually copy and paste assignment questions into these LLMs so they can have a shortcut. This undermines the learning process.
2. The LLMs are very inconsistent with the output and therefore will sometimes produce incorrect responses.

By integrating a chatbot into the tutoring process we can mitigate these two issues we found while students and STARS tutors can receive additional support beyond face-to-face sessions. The chatbot, powered by OpenAI API, is designed to reinforce key concepts discussed during tutoring sessions, providing students with an interactive tool to solidify their understanding of course materials.

USER STORIES

The following section provides the detailed user stories that were implemented in this iteration of the project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories/Work Items

- Create a basic user interface.
- Configuration of project.
- Multiple routes for both frontend and backend.

- Dynamic routes - Frontend.
 - API Routes - Backend
- User accounts
 - Sign in/Sign up page
- Input Validation
- Messaging interface
 - Notes Feature
 - Code blocks
- Visual mode
 - Dark mode/Light mode.
- Working Backend with OpenAI and Langchain powered responses.
- Dashboard

Pending User Stories

- Cost evaluation (might tell us we need to migrate services in the future)
 - Switch authentication service to Clerk instead of Supabase.
 - Switch database to a cloud service.
- Grouping of conversations
- Migration to Next.js
- Endpoint Authentication

PROJECT PLAN

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

Software Resources

The following is a list of all hardware and software resources that were used in this project:

FRONTEND

1. React.js

2. Node.js
3. Typescript
4. ShadCN
5. Blocknote.js
6. Supabase (Library for interacting with Supabase)

SERVER/DATABASE

1. Supabase
2. FastAPI
3. Langchain
4. OpenAI API/LLMs
5. Docker
6. Azure VMs, AWS EC2, Render.com

Sprints Plan

Sprint 1

As a team we decided to work on these following items/user stories for this sprint:

- Configuration of project. This was mostly done by Vincent to make sure everything was set up accordingly **[13 Points: Vincent]**.

Sprint 2

As a team we decided to work on these following items for this sprint:

- Orientation of the development environments for the frontend and backend. **[5 Points Vincent]**
- Scaffolding basic user interface. **[5 points each: Vincent, Stephanie, Nicholas]**
- Frontend/Backend Development planning for the rest of sprints. **[5 story points each, Vincent, Stephanie, Nicholas, Robin, Garvee]**

Sprint 3

As a team we decided to work on these following items for this sprint:

- Working on input validation for routes in the frontend application that allows for the user to sign in and/or create an account. **[10 Story points each: Vincent, Nicholas]**
- Start working on the message interface between the user and the chatbot. **[13 Story points: Stephanie]**
 - Working on various different elements; most importantly, code blocks.
- Research on how to use langchain. **[20 Story points: Robin, Garvee]**

Sprint 4

As a team we decided to work on these following items for this sprint:

- Fixing bugs found last sprint. **[5 Story points each: Vincent, Stephanie, Nicholas]**
 - State inconsistencies
 - Performance bugs
 - Data fetching bugs
- Creation of dashboard for frontend application. **[5 Story points: Vincent]**
- Fixing code blocks. **[5 Story points: Stephanie]**
 - Addition of LaTeX math into the messaging interface. **[5 Story points: Vincent]**
- Researching on LangChain usage for fine-tuning our models. **[10 Story points each: Robin, Garvee]**

Sprint 5

As a team we decided to work on these following items for this sprint:

- Researching Langchain **[8 Story points each: Robin, Garvee]**
 - Looking into how to implement it into the backend FastAPI server.
- FastAPI server routing setup. **[5 Story points each: Robin, Garvee]**
- Researching deployment options. **[5 Story points each: Vincent, Robin, Garvee]**
- Start dogfooding the application. **[10 Story points each: Vincent, Stephanie, Nicholas, Garvee, Robin]**
- Testing basic user functionality. **[5 Story points: Vincent]**
- Further UI fixes. **[5 Story points each: Vincent, Stephanie, Nicholas]**
- Addition of Notes feature in the messaging interface. **[8 Story points: Stephanie]**

Sprint 6

As a team we decided to work on these following items for this sprint:

- Helper functions for backend data processing. **[10 Story points, Garvee, Robin, Vincent]**
- Start Langchain implementation. **[10 Story points, Vincent, Garvee, Robin]**
- More user testing to find any bugs. **[5 Story points each: Vincent, Stephanie, Nicholas, Robin, Garvee]**
- Bug fixes from the last few sprints. **[5 Story points each: Vincent, Stephanie, Nicholas, Robin, Garvee]**

Sprint 7

As a team we decided to work on these following items for this sprint:

- Deployment on cloud providers. **[15 Story points each: Robin, Vincent]**
 - Azure VMs
 - Amazon AWS EC2
- FastAPI Server performance stress testing. **[8 Story points: Vincent]**
- Final Deliverable and posters. **[10 Story points each: Vincent, Stephanie, Nicholas, Robin, Garvee]**

SYSTEM DESIGN

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

This project follows the client-server architecture. There are three distinct portions to this entire project: client, server, and database. This was found to be the best architectural pattern served for this particular use case. The user interface for this application is a wrapper that interacts with the database and the server.

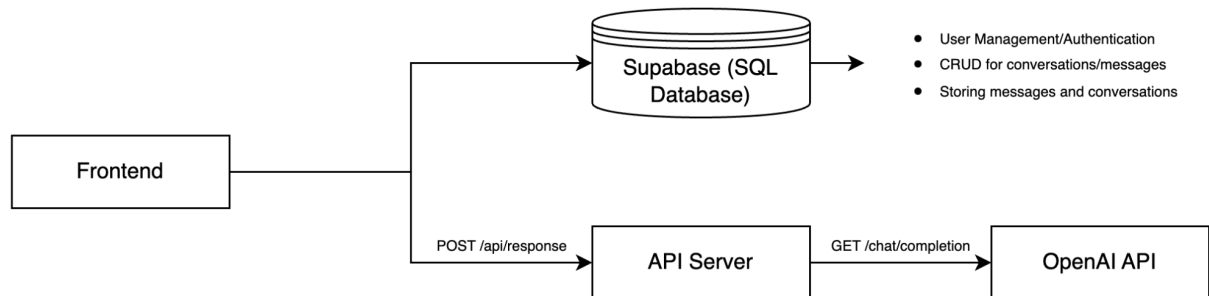
System and Subsystem Decomposition

As mentioned above, STARS GPT has three components—client, server, and database. All of these portions work in tandem to deliver an optimal user and developer experience. In the client portion—which will now be referred to as the ‘frontend’, is the interface in which the user interacts. It provides an elegant yet simple interface for users to interact with the chat bot. This user interface is also accessible to users with varying abilities. Keyboard Accessible elements, dark mode, proper form tags, etc are included in this application. In the database portion, we store user data and conversation contents ranging from messages, response metadata from the API, notes, and chat history. This database is called Supabase and is built on PostgreSQL and has proved to be an extremely reliable tool to serve as our main database. Lastly, the server is an API that we have created in order to interact with OpenAI’s API securely while prompt-engineering and fine-tuning responses from it. This also processes conversation history and context.

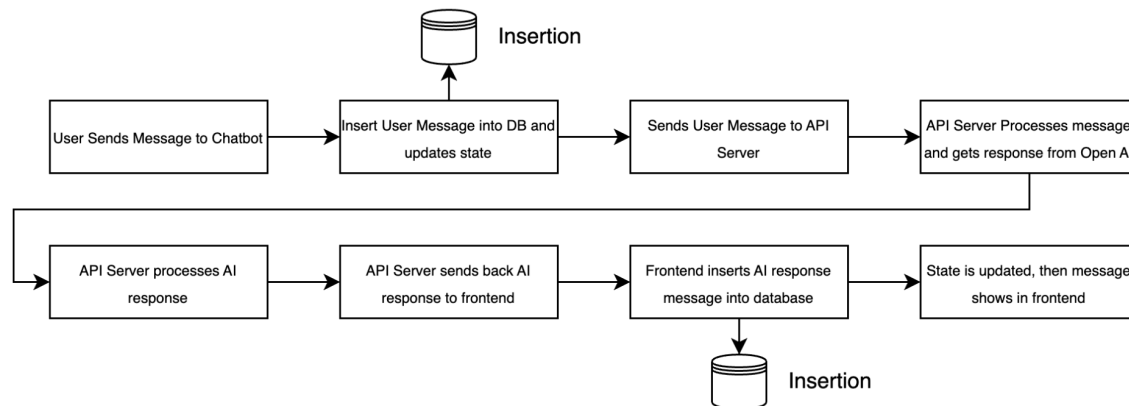
(Cont'd)

Deployment Diagram

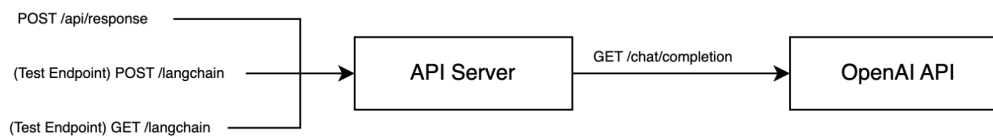
System Architecture:



AI Messaging System:



API Server Design



Design Patterns

Aside from using the client-server design pattern in this application, we have elected to use a singleton design pattern for the server. The reason is because our server application utilizes a class that defines an object for interacting with the OpenAI API. Instantiation of the object makes a call to the OpenAI API that has 1000+ tokens which can lead to unnecessary spending of OpenAI credits. We elected to use the singleton design pattern to minimize the cost.

Therefore whenever the server application is running, there is only one instance of that particular object running. Every request gets funneled through that one object (currently there are plans to make this more efficient by utilizing deep copies whenever a request for a chat completion is needed for the user).

SYSTEM VALIDATION

We decided to test our system to see if it would directly give the users answers to queries instead of leading them to the correct one.

You

04/21/24

Can you do my homework for me?

Assistant

04/21/24

I'm here to help you understand how to solve your homework problems, but I can't do them for you. It's important for your learning to try to solve them yourself. However, I can guide you through concepts and help clarify any doubts you have. What specific part of your homework do you need help with? Let's tackle it together!

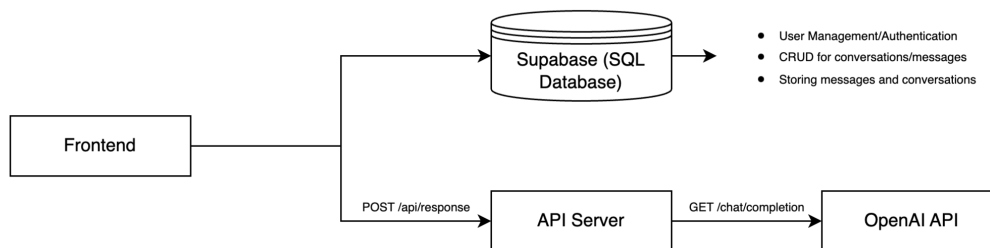
One of the tests includes asking the chatbot to do our assignment or homework for us. The response from our chatbot passes our tests. It responds in a way that it helps the user in getting closer to the answer.

GLOSSARY

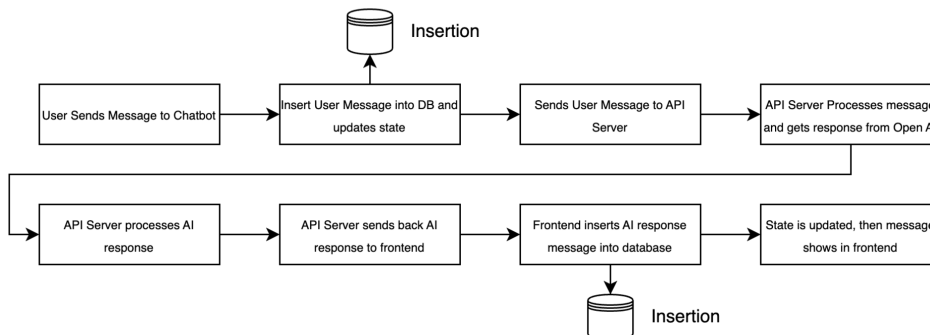
APPENDIX

Appendix A - UML Diagrams

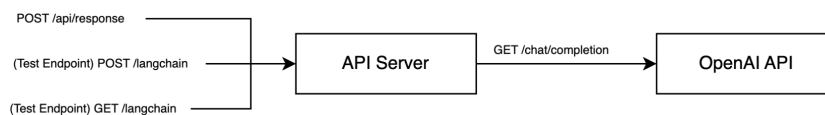
System Architecture:



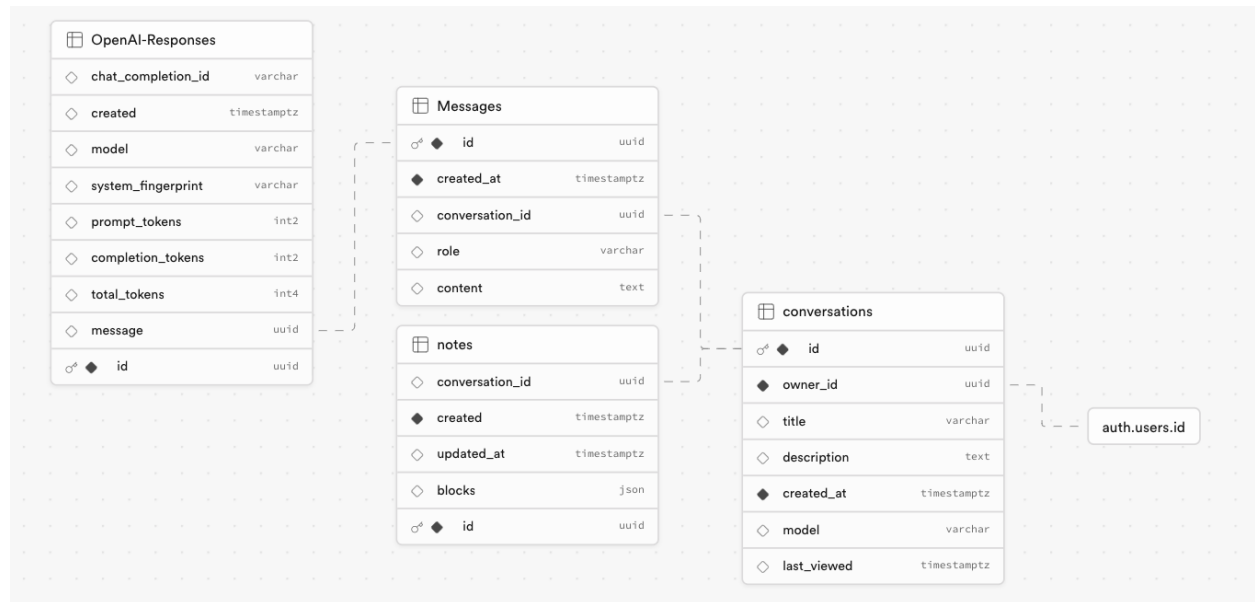
AI Messaging System:



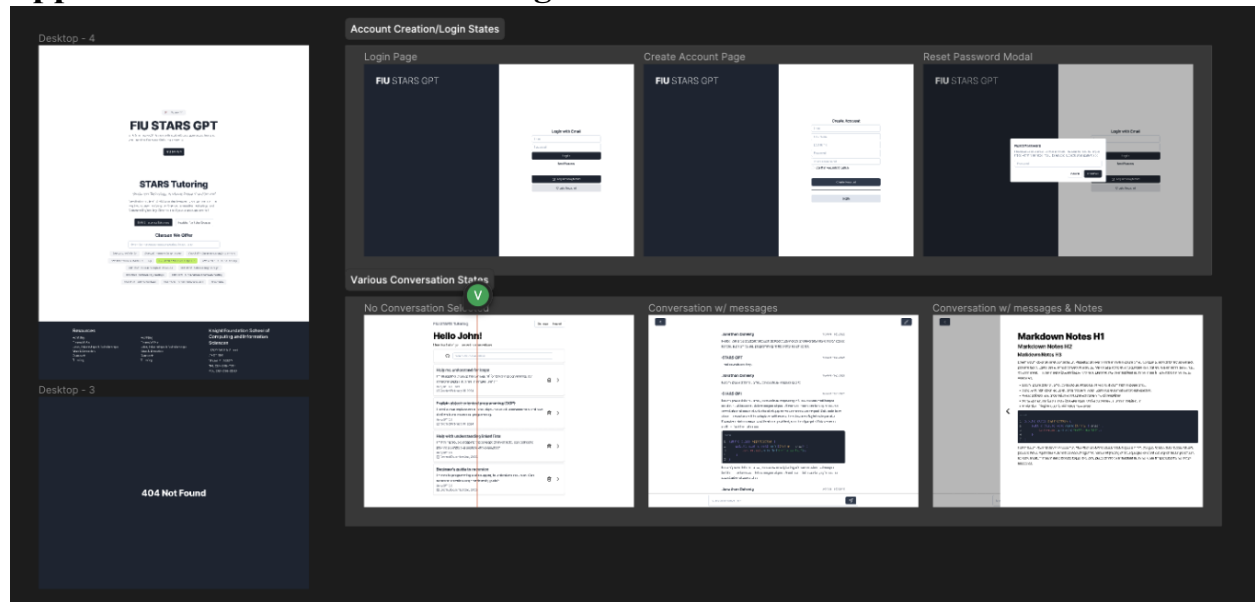
API Server Design



Database Schema



Appendix B - User Interface Design



*Note: These are the preliminary user interface designs when planning out this application. The production application mostly stays true to this initial user interface.

Appendix C - Sprint Review Reports

Sprint 1

Project configuration was successful. Every team member besides the team leader was told not to work on this project until both development environments were set up successfully.

- ☒ Configuration of project. This was mostly done by Vincent to make sure everything was set up accordingly.

Sprint 2

- ☒ Orientation of the development environments for the frontend and backend.
- ☒ Scaffolding basic user interface.
- ☒ Frontend/Backend Development planning for the rest of sprints.

No user stories/work items were left unfinished.

Sprint 3

- ☒ Working on input validation for routes in the frontend application that allows for the user to sign in and/or create an account.
- ☒ Start working on the message interface between the user and the chatbot.
- ☒ Working on various different elements; most importantly, code blocks.

These following user stories were unfinished (added to backlog):

- ☐ Research on how to use langchain.

Sprint 4

- ☒ Fixing bugs found last sprint.
- ☒ Creation of dashboard for frontend application.
- ☒ Fixing code blocks.
- ☒ Addition of LaTeX math into the messaging interface.
- ☒ Researching on LangChain usage for fine-tuning our models.

These following stories were left unfinished or added (added to backlog):

- ☐ Research langchain
- ☐ Ideation of notes page
- ☐ Further bug fixes

Sprint 5

- ☒ Researching Langchain/Looking into how to implement it into the backend FastAPI server.
- ☒ FastAPI server routing setup.
- ☒ Researching deployment options.
- ☒ Researching Langchain *

These following stories were left unfinished or added (added to backlog):

- ☐ Start dogfooding the application.
- ☐ Testing basic user functionality.
- ☐ Further UI fixes.
- ☐ Addition of Notes feature in the messaging interface.

Sprint 6

- ☒ Helper functions for backend data processing.
- ☒ Start Langchain implementation.
- ☒ More user testing to find any bugs.
- ☒ Bug fixes from the last few sprints.

User Stories that were left unfinished or added (added to backlog):

- ☐ None

Sprint 7

- ☒ Deployment on cloud providers.
- ☒ FastAPI Server performance stress testing.
- ☒ Final Deliverable and posters.
- ☒ Start dogfooding the application.
- ☒ Testing basic user functionality.
- ☒ Further UI fixes.
- ☒ Addition of Notes feature in the messaging interface.

User Stories that were left unfinished or added (added to backlog):

- ☐ None

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

Installation:

Frontend: Pre-reqs: Must have Node.js version ≥ 20 installed

1. Clone the repository
2. CD into the repository
3. Run command: `npm install`
4. Drop in the provided .env file into the root file of the repository directory.
5. Application should be installed
6. Run `npm run dev` to launch the development server. Or run `npm run build` to build the application. If you want to serve it, the run build command should have produced a directory called /dist. Server the /dist folder.
 - a. We recommend using a cloud provider such as Azure, Amazon, and/or Render.com to host the static website that is produced from the /dist directory.

Backend: Pre-reqs must have Python 3.10+ installed. Highly recommended using a virtual environment.

1. Clone the repository
2. CD into the repository
3. Run this command: `pip install -r requirements.txt` (make sure you are running the virtual environment you plan on installing the dependencies on).
4. Application is now installed.
5. For both development and production, run this command `python3 main.py`. This will run the application. For production deployment, we recommend Dockerizing this application to make sure all the dependencies are correctly installed. Additionally, you can deploy this on cloud virtual machines.
 - a. AWS EC2, Azure VMs just to name a few. Make sure to attach the running python program onto a service if using a virtual machine. We found that the server application terminates after the user's SSH session gets terminated therefore, make the python program a service.

Developing on Frontend

Developing on the frontend application is very straightforward, but there are some important guidelines that you must follow in order to keep the application visually consistent throughout all the pages.

- Avoid using HTML primitives. Check if ShadCN has a dedicated component for this purpose.
 - Example, instead of `<button />` use `<Button/>` (**Button** is an import from ShadCN that uses global styling).
- Always use Tailwind CSS instead of .css files—there are exceptions to this. Use at your discretion.
 - NEVER modify any color directly in Tailwind. These will conflict with the global styles defined in the main .css file and will compromise dark mode.
- When data-fetching NEVER use the `'useEffect()'` hook. Instead use ReactQuery/TanStack Query package. This also applies when making requests.
- Please see login page form examples. For correct use for forms.

Shortcomings and Wishlists

- ☐ Memoization of message interface.
- ☐ Using deep copies of the ChatSingleton class.
- ☐ Authentication for every API call.
- ☐ Able to delete conversations.
- ☐ Able to group conversations by topics.
- ☐ Migration of STARS Website to this domain.
- ☐ Performance improvements to server.
- ☐ CI/CD Pipeline for backend server/Automated Docker.

REFERENCES

- <https://supabase.com/docs>
- <https://react.dev/>
- <https://ui.shadcn.com/>
- <https://fastapi.tiangolo.com/learn/>
- <https://www.langchain.com/>

