1. **Video Walkthrough**
2. **Project Overview**

This project aims to create an AI chatbot specifically designed to support instructors in teaching introductory computer science topics, with potential adaptability to other teaching contexts. Leveraging the OpenAI API, particularly ChatGPT models, this chatbot operates according to predefined instructions provided by instructors. Its primary role involves engaging with students, providing guidance, and explanations within the framework set by the instructor. Notably, it's tailored to avoid default responses that simply offer direct answers, creating a more interactive and personalized learning experience for students. Essentially, it serves as a virtual teaching assistant, enhancing the educational process through dynamic interaction and support.

1. **User Groups and Scenarios**

**Registered User - Instructors**

**Scenario**: Josh, a professor at UBC, teaches computer science. Aware of the increasing popularity of AI assistance software, he recognizes that his students also use them. While Josh encourages his students to utilize the new technology, he prefers that AI doesn't offer direct answers. Instead, he wants it to provide guidance, helping students understand the fundamental concepts.

**Registered User - Students**

**Scenario**: John, a first-year computer science student, attends an introductory Java programming class. He frequently encounters assignments in his labs that leave him feeling uncertain. John seeks clarification on certain concepts to better solve these problems.

1. **Software Requirements**

**User Authentication**: The application must support user registration, login, and logout functionalities.

**Question Insertion / Deletion**: The application should allow instructor user groups to add or delete new questions to the system.

**Tag Insertion**: The application should allow instructor user group to assign tags to newly inserted questions.

**Question Filter**: The application should allow all user groups to filter questions based on tags on the homepage.

**Instruction Insertion/Deletion**: The instructor user group needs to be able to add and delete instructions.

**Assign Instruction to Question**: The instructor needs to be able to assign a specific instruction to a question. So, when the chatbot answers the question it will keep the instruction in mind.

**Chatbot Interaction**: All user groups will be able to interact with the chatbot where they can communicate with the OpenAI ChatGPT model.

**Feedback Collection**: The application will have a page to asks for user experience.

**File Upload**: The instructor user group should be able to upload lecture notes or slides and assign questions to certain files, so the chatbot could answer questions having these uploaded files as reference.

**Statistic Display**: The instructor user group should be able to view user feedback statics.

**Usability**: The application needs to be easy to use, with an intuitive interface that makes it easy for the user to navigate and perform actions.

**Performance**: The system should load pages and process user request as quickly as possible to ensure a smooth user experience

**Security**: Encryption for sensitive information and compliance with relevant data protection guidelines.

**Reliability**: The chatbot should be able to handle user requests simultaneously without crashing the server.

1. **Current Software Status**

Throughout the development phase, ten functionality requirements were either set up initially or rose as the project grew; as of April 25, 2024, eight out of ten requirements have been fully delivered. The file upload requirement and statistic display requirement have been partially delivered that future work or refinement will be needed.

|  |  |  |
| --- | --- | --- |
| Requirement ID | Requirement Description | Implemented? |
| 1 | Registration - Users should be able to register in the system | Yes |
| 2 | Login - Registered users should be able to log in to the system using their credentials | Yes |
| 3 | Question Insertion / Deletion - The instructor user groups should be able to add/delete questions. | Yes |
| 4 | Instruction Insertion / Deletion - The instructors should be able to create or delete instructions | Yes |
| 5 | Assign Tags to Questions - The instructors should be able to assign tags to newly created questions | Yes |
| 6 | Question Filter - All users should be able to filter homepage questions based on selected filters | Yes |
| 7 | Chatbot Interaction - All user groups should be able to interact with the OpenAI ChatGPT model | Yes |
| 8 | Feedback Collection - All user groups should have access of a feedback collection page | Yes |
| 9 | View Statistics - The instructor group should be able to view user feedback statistics | No |
| 10 | File Upload - The instructor group should be able to upload lecture slides and assign questions to selected files | No |

Most of the requirements were successfully captured and delivered, enabling core functionalities such as assign instructions to questions and interaction with the chatbot.

An experiment was designed and conducted by another student involving the usage of this tool in first-year computer science labs. It appears that the initial requirements generally captured the essential details needed for the project. From limited verbal feedback from the students involved, the tool also served its purpose which is to assist students in their learning journey.

1. **Architecture Overview**

The application utilizes Remix as its main framework, and typescript as its programming language. Remix is a framework which is built on top of the popular React library. It allows a project to keep its frontend and backend at one place without switching language. More specifically, the Remix Blues stack was used for an easier developing process. The stack integrate many popular tools including the tailwind, docker, prisma, fly.io, cypress, prettier and ESLint. For database, the project uses prisma to run PostgreSQL.

1. **Installation Details**

**Prerequisites**

Before initiating the project setup, ensure the following essential tools and

applications are installed on your system:

**Web Browsers**

Google Chrome, Mozilla Firefox, or Microsoft Edge for testing and debugging.

**Integrated Development Environment (IDE)**

Visual Studio Code is recommended.

**Version Control System**

Git is necessary for source code management and team collaboration.

**Command Line Tools**

Terminal (macOS/Linux) or Command Prompt/PowerShell (Windows) for running.

scripts and managing the project.

**Installation Steps**

**Install Node.js and npm**

Download and install Node.js from the official Node.js website, which includes npm.

The latest version can be found [here](https://nodejs.org/en/download/current).

**Install Docker**

The latest version of docker can be found [here](https://www.docker.com/products/docker-desktop/).

**Create Prisma Migration**

*npx prisma migrate dev --name <migration-name>*

**Apply Migration**

*npx prisma migrate dev*

**Generate Prisma Client**

*npx prisma generate*

**Install Project Dependencies**

Navigate to the project directory and execute *npm install* to install all necessary

dependencies as listed in the package.json file.

**Run the Project**

Start the application by executing *npm run dev* in the command line, which launches

the development server and opens the application in your default web browser,

typically accessible at <http://localhost:3000>

**Dependency Table**

|  |  |
| --- | --- |
| @emotion/react | Provides React components for Emotion CSS-in-JS library, allowing styling of React components with Emotion. |
| @emotion/styled | Provides styled components for Emotion CSS-in-JS library, enabling creation of styled components using Emotion. |
| @heroicons/react | Provides a set of React components for Heroicons, offering a library of customizable SVG icons. |
| @isaacs/express-prometheus-middleware | Middleware for Express.js to expose Prometheus metrics for monitoring Express applications. |
| @mui/icons-material | Provides Material-UI icons as React components for easy integration with Material-UI designs. |
| @mui/material | Provides Material-UI components for React, offering a library of pre-styled React components following Material Design guidelines. |
| @prisma/client | Prisma client library for Node.js applications, facilitating database access and manipulation for applications using Prisma ORM. |
| @remix-run/css-bundle | Bundle of CSS styles for Remix-run applications, optimizing CSS delivery for performance. |
| @remix-run/express | Integration for using Remix-run with Express.js, allowing Remix-run applications to be served via Express. |
| @remix-run/node | Node.js integration for Remix-run, enabling server-side rendering and routing for Remix-run applications. |
| @remix-run/react | React integration for Remix-run, providing hooks and utilities for building React components within Remix-run applications. |
| bcryptjs | Library for hashing passwords securely using bcrypt algorithm, commonly used for user authentication. |
| chokidar | File watching library for Node.js, used for watching file changes during development. |
| cloudinary | SDK for integrating with Cloudinary media management platform, offering tools for managing and delivering images and videos in web applications. |
| compression | Middleware for Express.js to enable gzip compression for HTTP responses, reducing the size of responses for improved performance. |
| cross-env | Utility for setting environment variables across different platforms in Node.js scripts. |
| express | Web application framework for Node.js, providing a robust set of features for building web servers and APIs. |
| formidable | Library for parsing form data in Node.js applications, particularly useful for handling file uploads. |
| highlight.js | Syntax highlighting library for web development, providing tools for highlighting code syntax in various programming languages. |
| isbot | Utility library for detecting bot user agents in web applications, useful for distinguishing between human users and bots. |
| morgan | HTTP request logger middleware for Express.js, logging request details for debugging and monitoring purposes. |
| multer | Middleware for Express.js for handling file uploads, particularly for processing multipart/form-data requests. |
| openai | SDK for integrating with OpenAI API, providing tools for accessing and using OpenAI's language models and other AI capabilities. |
| prismjs | Lightweight syntax highlighting library for web development, offering support for highlighting code syntax in various programming languages. |
| prom-client | Library for instrumenting Node.js applications with Prometheus metrics, allowing monitoring of various application metrics. |
| react | JavaScript library for building user interfaces, providing tools for creating reusable UI components. |
| react-beautiful-dnd | React library for creating drag-and-drop interfaces, offering a simple and powerful API for implementing drag-and-drop functionality in React applications. |
| react-dom | React library for working with the DOM, providing tools for rendering React components into the browser. |
| react-masonry-css | React library for creating responsive masonry layouts, allowing for dynamic arrangement of elements in a grid layout. |
| react-syntax-highlighter | React library for syntax highlighting, offering components for rendering code with syntax highlighting in React applications. |
| source-map-support | Library for enhancing stack traces in Node.js applications with source maps, improving error debugging and logging. |
| tiny-invariant | Tiny invariant assertion library for JavaScript, providing a lightweight way to enforce conditions in code. |

1. **Known Bugs and Future Improvements**
2. **Reflections**