Voice-Enabled AI Chatbot with Context Awareness in LMS

Client Details

Client: Employability.Life
Client Name: Neeraj Singh
Client contact details NSingh@employability.life

Assessment 7 - Project Write-Up

Team Details

Name	ID	Role
Syeda Tamanna Sheme	30432670	Front End Developer & Tester
Tanvir Iqbal	30437681	Backend Developer
Hansi Nipunika Panwillaarachchi Kotudura Bandanage	30423990	Backend Developer
Afreen Begum	30410327	Business Analyst

Contents

1. Executive Summary

This paper proposes design and development of voice bot enabled AI chatbot integrated with Moodle LMS to ease of accessibility and interaction between student and teacher. Within the chatbot, the OpenAI language model is exploited to offer smart responses to natural language questions, grounded on real-time data obtained via Moodle's RESTful APIs. Built using a Node.JS backend and Web front-end the system has both text-to-speech and speech-to-text and users can chat using either audio or written text. Role-based logic guarantee personalized and secured responses: students get back their grades and assignments and course information, instructors receive class-wide academic data. The project was undertaken using the Design Science Research (DSR) approach and supported with Agile Scrum in three development sprints. We performed testing using Postman for API and validation and Jest for unit testing to provide robust, reliable performance. The chatbot was also installed as a floating assistant in Moodle: it could be invoked by the user in tabs. Not only does this smart helper save teachers time on the groundwork, but it also helps to engage students and to make learning more accessible. Potential Improvements: There could be analytics dashboards, more languages, and offline support. The initiative exemplifies the opportunity AI and voice can have in revolutionizing higher education digital learning environments.

2. Introduction

2.1. Project Background

A Voice-Enabled AI Chatbot with Context Awareness in Learning Management Systems (LMS): The next version of Learning Management Systems like Moodle has been enhanced tremendously which is a value addition in dealing with the common problems faced by the students, Faculty, and the affiliated handling organization in DL. (JAKUBIK, 2021)Thanks to the implementation of modern technologies, such as Artificial Intelligence (AI), NLP, or voice recognition, the chatbot makes education more valuable in the following aspects:

- Enhanced Interaction: Our chatbot enables easy communication where users can communicate in voice and text. This flexibility accommodates a variety of student learning preferences and provides an overall level of service and value that is sure to benefit students and their instructors.
- Better Accessibility: A chatbot plays a role in the academics by giving students an easy reach to academic details and helping them get organized with their work. Features such as reminders on assignments and deadlines make sure students have all of the information needed to do well on their work.
- Auto-Maintenance: Using the same format from above, chatbots can support on less high-traffic teaching question and leave the chatbot with more free time to learn. This can help educators concentrate more on teaching rather than administrative duties, and in the end improves the learning experience.
- Context-aware responses: The chatbot service provides the way to generate context-aware responses to adapt the chatbot interaction to the user's role (student, teacher, or administrator) and to the chatbot history (saved previous chatbot pairs). This

customization results in the information being more relevant to consumers, and potentially more valuable to users.

- Implications for Instructors: The chatbot can infer the extent of a student's active listening, and their likelihood to engage in that conversational context, which may provide the instructor with useful insights to inform his teaching strategies and also the students in need of additional support.
- Real-Time Monitoring for Admins: LMS admins can employ the chatbot for real-time monitoring of the system for performance and user engagement so that they can intervene and make quick changes for enhancing LMS. The project while solving the communication problems, by enabling an infrastructure that includes the Voice-Enabled AI Chatbot into LMS, contributes to create a more inclusive and interactive learning space. This revolutionary approach is in line with the developing requirements of contemporary education and aid the stakeholders in learning efficiently and effectively.
- Photoplethysmography Signal Pre-processing by Using the Wavelet Transform and Its Applications. (mutný, 2020) Overall, the adoption of a chatbot like this can help to revolutionize the process of how institutions of education function, while making learning easier and more fun, all the while helping to ease the administrative burden on educators and other stakeholders.

2.2. Problem Statement

Even with the proliferation of Learning Management Systems (LMS), such as Moodle, many of our students have difficulty interacting with these technologies in ways that support effective learning. It shouldn't be so hard to track down assignment deadlines, find course materials, or keep up with announcements, and yet somehow, it is. Meanwhile, teachers and LMS administrators are often overwhelmed with a constant flow of redundant queries, tracking student involvement and inconsistencies in communication. Not only do these ongoing problems slow down productivity, but they also delay answers and upset students.

The majority of LMS providers do not even come close to providing personalized, instant help, and this makes the end user's experience seem cold and disjointed. Traditional chatbots usually do not respond contextually and everything is in text, with no voice interaction in between. This project suggests a dynamic alternative: a voice-activated chatbot powered by AI and integrated with Moodle. Smart and context-aware responses Delegating everyday tasks to enhance the learning experience be more intuitive, reduce educators' workload and shows a more attractive digital at a classroom.

2.3. Project Objectives

The main goal of this project is to build an AI chatbot with context awareness supported with voice for Moodle LMS. It is designed to enable more natural interaction and interaction options for two important user groups **(students and teachers)** of the Moodle by voice and text chatting with the bot.

Primary objectives include providing role-based **personalized responses** (users can be either students or teachers), allowing students access to academic information (course timetable, assignment deadlines, results etc), and allowing teachers to see course interactions and activity.

The chatbot also works to alleviate some of the volume of recurrent questions by answering popular questions automatically, meaning educators can spend more time teaching.
"Furthermore, the system is developed to provide response context space in support of **context-aware interaction**, namely, it still has the understanding of former dialogue flow to provide more proper and intelligent answers." In the end, to create a smart and user-friendly tool that makes the learning and teaching experience better.

2.4. Target Users

This AI Chatbot is a chatbot created for students and teachers that are currently involved in educational activities on the Moodle LMS Educators are using it to deliver instruction, get the status of learners, and send emails of particular scenario to students. The users were chosen, glass eco be typical daily users of Moodle, and the students would gain better access to study material and communication, and less administrative work for the institution.
Students -This class consist of Undergraduate and Postgraduate students as they actively engage with Moodle to look at the assignments, refer the course of classroom and look at academic standing4. Organizing a mass of information and meeting a deadline are two issues many students face. The chatbot address this is use-case by providing academic information in a timely manner during normal users' voice or text interaction, being so organized and academics engaged.
Teachers -Moodle is largely used by teachers who manage the resources and activities in their mainstream, thereby providing answers to the commonly asked questions. It allows teachers to avoid dealing with trivial questions, offers thumps up/down of student input and feedback on situational teacher's behaviour for a specific task.
By focusing on these two primary groups of users, the system is oriented towards its intended users and can then offer a system that is entitled to their many stakeholders.

2.5. Scope and Limitations

Scope

The project brings to you AI integrated voice enabled LMS chat "LUMI" that can be plugged into to your existing Moodle. The primary application of Name's script is the facilitation of academic information exchange through voice and text. Not to mention, the chatbot delivers instant answers to questions regarding course content, assignment due dates, and grades, so learning can become more streamlined and entertaining.
The system is developed with role-based logic - responses/messaging are specifically formulated based on whether the user is student or teacher. It is also context-aware, which means that the chatbot will be able to keep up a conversation and respond to further queries. Built on top of current web technologies and Moodle APIs, the chatbot operates in all desktop and mobile browsers, thus providing maximum flexibility to the users.

Limitations

The system has a few shortcomings despite its successes:

- Only teachers and students are supported; there is no admin role (Admin role is playing by the teacher at the moment).
- Its role management capabilities and granularity are a little limited.
- Some of the UI pages like Profile, Notifications, and Preferences are not integrated with the chatbot.
- Visual the Reset Password page is present, though it's not really integrated.
- The chatbot only understands English at the moment.
- The accuracy of voice input can be influenced in a noisy environment or by strong accents.

2.6. Structure of the Report

The design, development and evaluation of the voice enabled AI chatbot connected with Moodle the LMS are described in depth in the nine chapters of this study.

Chapter	Title	Description
Chapter 1	Executive Summary	Provides a concise overview of the project, including its goals, methodology, and key achievements.
Chapter 2	Introduction	Covers the project background, problem statement, objectives, target users, scope, and limitations.
Chapter 3	Literature Review	Reviews existing research on AI in education, chatbot systems, voice interaction, and context-aware technologies.
Chapter 4	Methodology	Explains the design science approach, Agile Scrum framework, tools used, and requirement analysis techniques.
Chapter 5	System Design	Details of the system architecture, functional and non-functional requirements, UI/UX design, and design diagrams.
Chapter 6	Implementation	Describes the development process for the frontend and backend, Moodle integration, OpenAI API usage, and speech modules.
Chapter 7	Testing and Evaluation	Outlines the testing strategies, manual and automated testing results, API testing, and user acceptance testing.
Chapter 8	Results and Discussion	Summarizes the outcomes, compares results with objectives, and discusses challenges and limitations.
Chapter 9	Conclusion and Future Work	Concludes the report with key findings, lessons learned, and suggestions for future enhancements.

3. Literature Review

In this chapter we focus on a review of the literature and technology base applicable to the development and design of a voice enabled context-aware AI chatbot in an LMS. This review article identifies five crucial categories the state of the art in the field can be split into; artificial intelligence in education; the chatbot role in LMS; voice interaction techniques; context-aware conversational systems; and research voids in existing platforms.

3.1. Artificial Intelligence in Education

Education has been revolutionized by AI, which has introduced smart systems that can automate administrative tasks, provide instantaneous feedback and tailor the learning experience. AI-enabled tools can inform decisions by modelling how students learn while garnering performance data that allows educators to target interventions.

Research by Holmes (Holmes, W., Bialik, M., & Fadel, C., 2019) suggest how AI, if looked at from certain perspectives, has the potential to increase access, efficiency, and engagement in learning environments. Apps like intelligent tutoring systems, grading helpers, and AI recommendation tools are already improving the way students connect with educational material.

For this project the GPT model from Open AI is being used to interpret student feedback in Moodle, providing meaningful, natural language-like answers in an academic context.

3.2. Role of Chatbots in Learning Management Systems

Chatbots have also been used in LMS environments as assistive agents, providing immediate, 24/7 support for academic and non-academic information queries. With automated features including answering FAQs, assignment tracking, and getting users acquainted with LMS features, chatbots relieve burden from educators and empowers more interaction from the learners.

Winkler and Soller's (Winkler, R., & Söllner, M. , 2018) study finds that an educational chatbot can help with knowledge retention, promote active learning, as well as provide feedback. In addition, the combination of chatbots and LMS such as Moodle, allows scalable academic support, making individualized help more widely available. This project's chatbot performs a similar function – deployed within Moodle, it helps students and teachers by fetching grades, deadlines for assignments and course-related information as required.

3.3. Voice Interaction and Speech Technologies

Voice input/output such as those used in speech-to-text and text-to-speech are more inclusive in terms of user experience. These techs can be very helpful in case of students with reading difficulty, visual impairments, physical challenge in typing, etc.

The availability of APIs like Web Speech and other native-browser solutions made it possible to add on-the-fly voice to a web-based system. (Radziwill, N. M., & Benton, M. C., 2017) suggest that voice-enabled systems remove user friction, are accessible to all, and emulate human conversations that are said to increase user engagement.

Voice interaction is achieved through the Web Speech API in this project, which means that participants can ask and listen to questions, indispensable in a more inclusive online learning.

3.4. Context Awareness in Conversational Agents

Adaptable systems customize their behaviour as a function of user's role, previous interactions and environment between others. In educational chatbots, understanding context is crucial to ensure that responses are not only personalised, but are role appropriate as well as coherent over a sequence of queries.

As per Dey (Dey, 2001) context is any information that can be used to characterize the situation of an entity. In chatbot applications, such as their identity of the user, the questions asked beforehand, the courses the user is enrolled to, and the state of the session. This chatbot keeps the context (i.e., student vs. teacher context) and history of the session in in-memory object. It allows the system to modify answers, limit visibility, add to conversation flow, based on the status of the user.

3.5. Related Work and Research Gap

A few works existing works have faced the applicability of AI and chatbot in education, but many take into account only text interaction or it is not possible to integrate them with LMS (Learning Management System) such as Moodle. Many of the current educational bots, for instance, work as a standalone instrument or a mobile app with restricted availability in terms of real-time academic data.

Chatbots such as Jill Watson (Goel, A., & Polepeddi, L., 2016),used in online courses to alert instructors of students who need support, have provided an example of how A.I. can be used to scale support—however, few of these systems are built with voice interaction, or leverage real-time data from the institutional LMS platform.

The specific gap that we target in this work is an AI chatbot with full integration into Moodle that is role-aware, voice-enabled and uses real-time APIs to retrieve individualized academic data. This technology combination [1] is still not much explored in academic environment, and, with respect to Moodle, very little explored.

4. Methodology

4.1. Research Approach

This study follows a Design Science Research (DSR) methodology, relevant for solving problems of practice by designing and evaluating novel, IT artifacts. DSR is popular in computer studies, as well as in software engineering, and information systems research, in which the aim is to propose working solutions to a pragmatic problem, as in this case to enhance communication and accessibility for Learning Management Systems (LMS) using artificial intelligence (AI) and voice interaction.

The main goal of this project is to develop, implement and evaluate an Alexa voice-enabled AI chatbot for personalized support in the context of a Moodle LMS that can be used by the students and teachers. The Design Science approach supports this goal, as it emphasizes the development, iterative improvement, and application-oriented validation of artifacts.

The investigation commenced with problem identification, based on workplace observations, client complaints, and student discontent about using Moodle to retrieve academic information. The main problems were difficulty locating deadlines for your

assignments, absence of immediate feedback, limited accessibilities features, and an increased number of repeated questions going to the teachers. These results served to establish design constraints for the potential solution.

After analyzing problems, we developed a prototype chatbot that meets specific requirements. This phase involved defining the functional and non-functional requirements, conceptualizing the system architecture, as well as the technology selection. The solution was built in a modular full stack pattern using a wide range of technologies, including a React-based frontend, a Node. js/Express backend, utilizing Moodle's REST API for its GPT conversational intelligence. Voice functionalities was built using the web speech API for speech recognition and synthesis.

We adopted the Agile approach, in particular Scrum, to our design process, in harmony with the DSR approach, as it advocated making incremental improvements to a current design as well as rapid development and continuous feedback. 1. runs over three sprints (1) setup and research, (2) voice integration and contextual response, and (3) testing with analytics. Each sprint ran through requirements review, prototyping and stakeholder feedback cycle.

GDSR methodology The DSR approach evaluation process was executed with extensive testing which included unit tests, API testing using Postman, usability testing with stakeholder feedback and user acceptance testing. Requirements such as retrieving grades, accessing assignments and enforcing role-based access control were satisfactorily tested. Non-functional requirements, response time, voice precision, and accessibility were also measured.

Not only does this iterative build-evaluate-refine cycle embody the fundamental precepts of design science, but it also helps to ensure that the final developed solution is technically correct and fit-for-purpose with respect to real-world user needs and constraints.

The DSR method also focuses on knowledge creation, suggesting that the results of this project are not restricted to the artifact. Learnings about the challenges of integrating with Moodle, the quirks of conversational AI, and the limitations of voice interaction can potentially inform future research and development in similar educational environments.

Ultimately, the DSR method offered a methodical, iterative and pragmatic basis to develop the new solution to LMS communication issues. It provided that the chatbot is developed based on an insight, built using cutting edge technology and evaluated using comprehensive testing, and contributed, as one of the outputs, not only a prototype, but also academic knowledge on AI implemented educational tools.

4.2. Agile Scrum Framework

The project was developed using the Agile Scrum methodology, with a total of three sprints over a project duration of 12 weeks. The Agile approach allowed for:

- Rapid prototyping and feedback integration.
- Clear backlog grooming and sprint planning.
- Transparent tracking of progress via Jira and Confluence.
- Regular retrospectives to improve collaboration and productivity.

Each sprint had well-defined user stories, tasks, and deliverables, reviewed during sprint reviews and retrospectives.

Sprint	Focus Area	Major Deliverables
Sprint 1	UI & Backend Setup	Chatbot wireframes, Moodle API research, Role-based access
Sprint 2	Voice Interaction & Personalization	Voice input, text-to-speech, role dashboard, manual testing
Sprint 3	Automation, Analytics, Final Testing	Automated testing, reports, chatbot logging, final UI updates

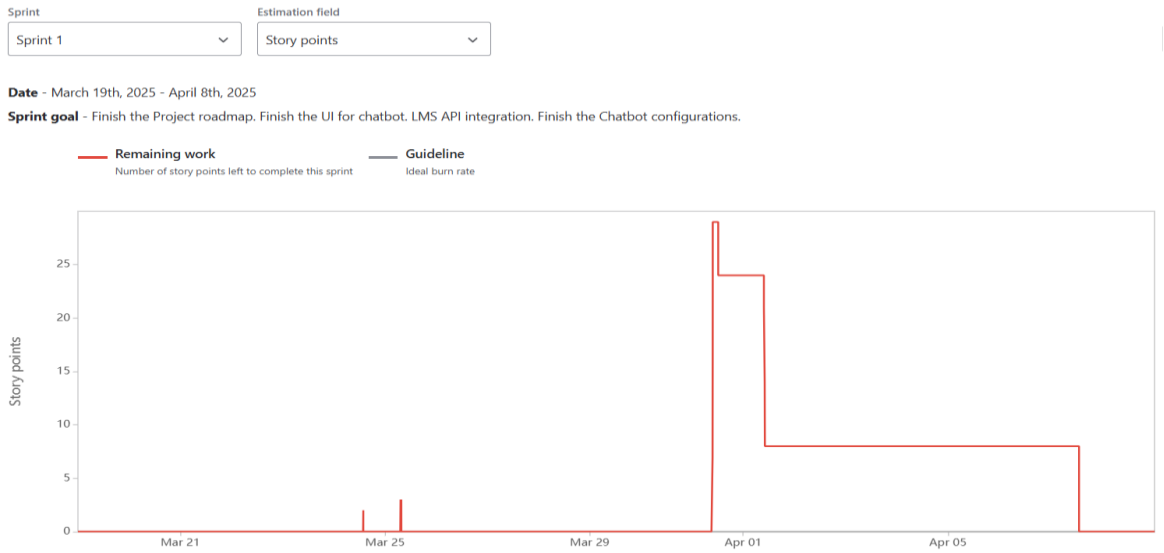


FIGURE 1: SPRINT 1 BURNDOWN CHART

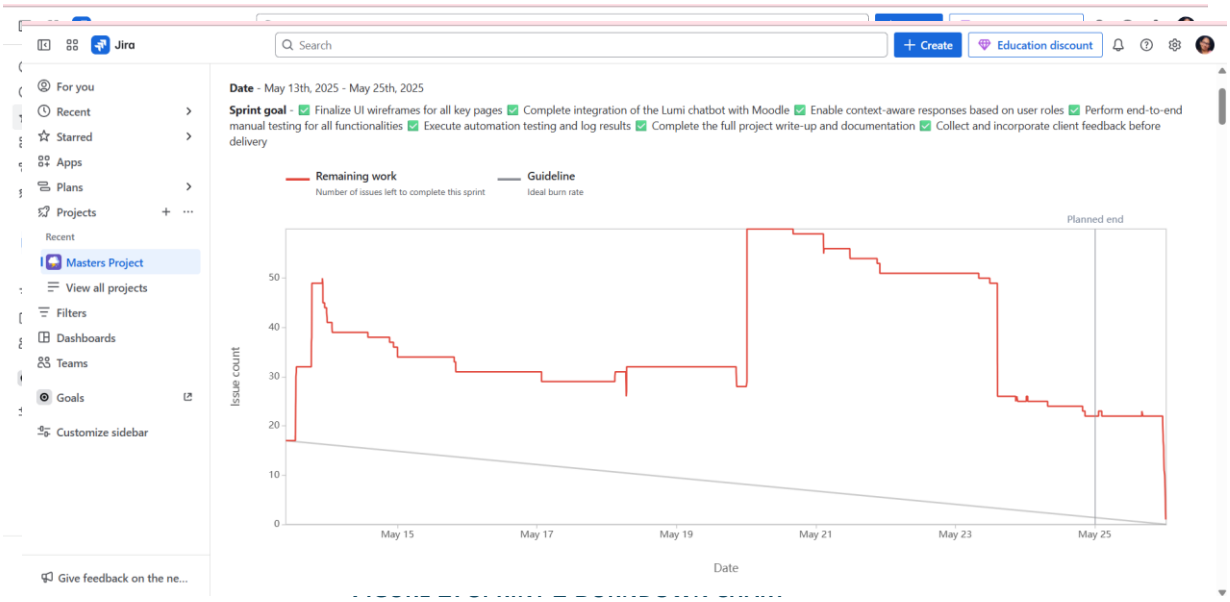


FIGURE 3: SPRINT 3 BURNDOWN CHART

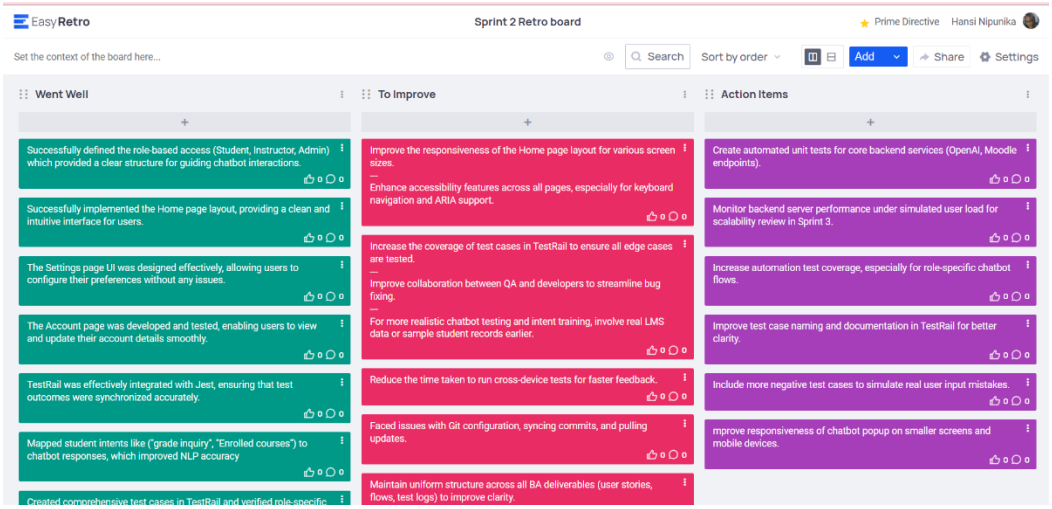


FIGURE 4: SPRINT 1 RETRO BOARD

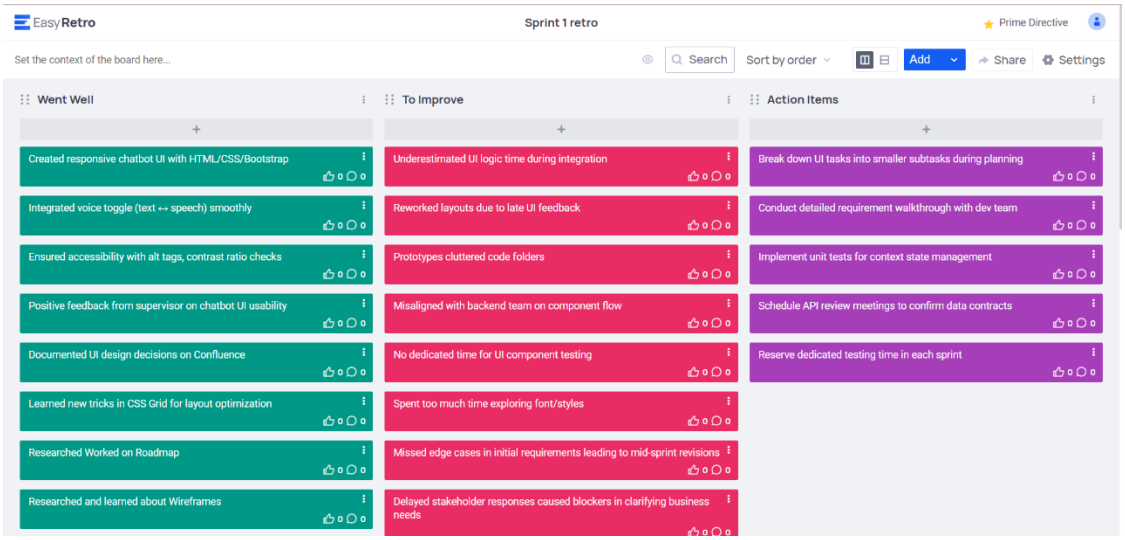


FIGURE 5: SPRINT 2 RETRO BOARD

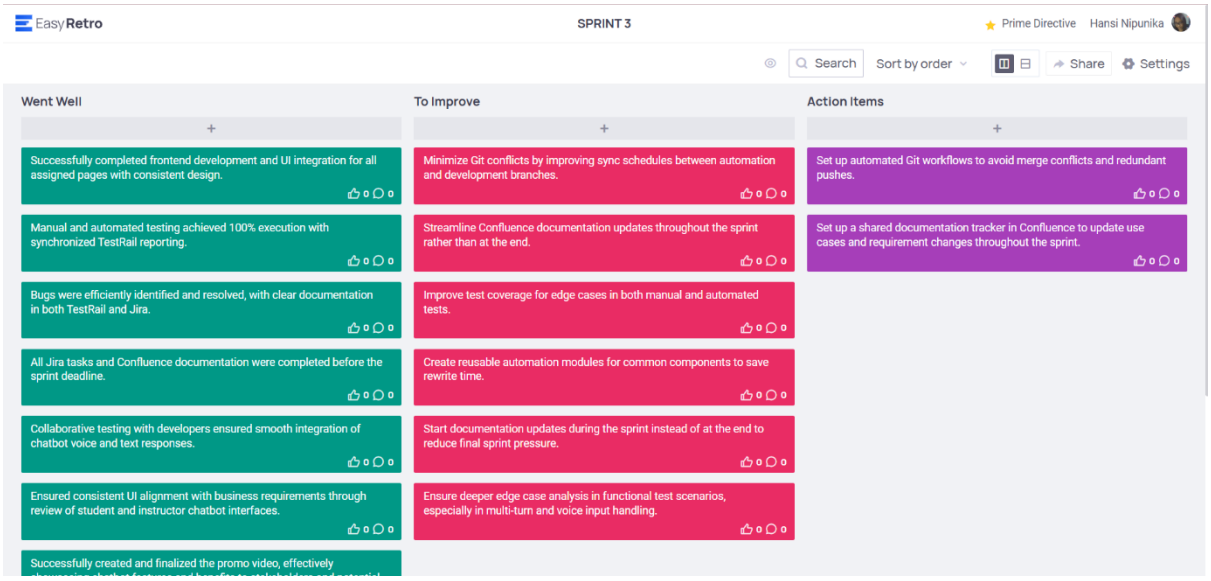


FIGURE 6: SPRINT 3 RETRO BOARD

4.3. Requirement Analysis Techniques

Several requirement analysis techniques were used to ensure the developed system addressed both functional and non-functional needs:

- Stakeholder Interviews with the client and LMS users.
- User Stories and Acceptance Criteria defined in collaboration with the business analyst.
- Wireframing in Figma for UI layout expectations.
- Use Case Diagrams and Flowcharts to visualize user interaction and backend integration.

These methods helped align technical design with stakeholder goals, avoid scope creep, and document assumptions clearly.

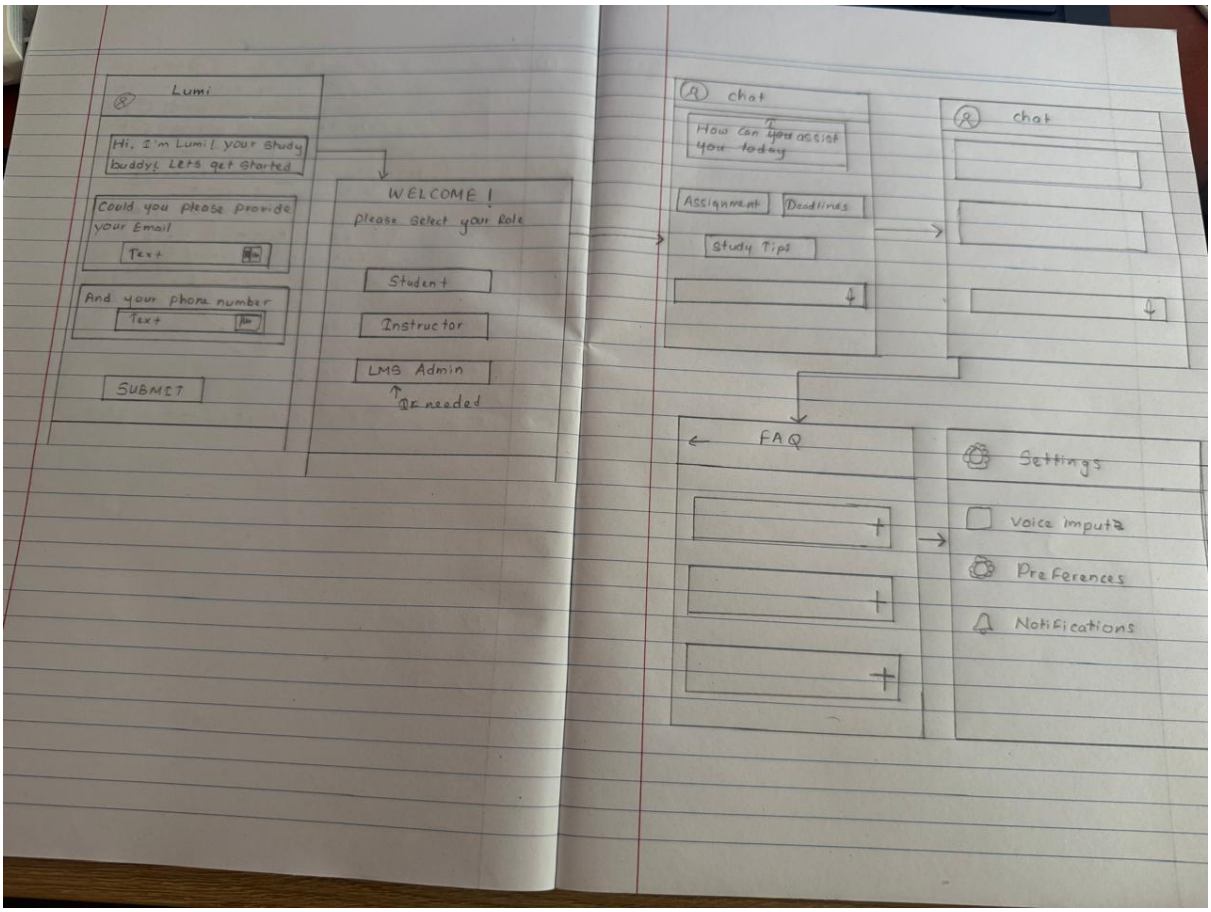


FIGURE 7: WIREFRAME MOCK-UPS OF CHATBOT LUMI

4.4. Tools and Technologies Used

The development and testing of the chatbot system relied on a carefully selected tech stack optimized for speed, accessibility, and integration:

Category	Tools & Technologies
Frontend	HTML5, CSS3, Bootstrap, JavaScript, ReactJS
Backend	Node.js, Express, Moodle REST API
Voice Interface	Web Speech API (Speech-to-Text & Text-to-Speech)
Project Management	Jira (Issue tracking), Confluence (Documentation)
Version Control	Git & GitHub
UI Design	Figma
Testing	Jest (Automation), TestRail (Manual Testing), Postman (API Testing)
Collaboration	Microsoft Teams, Zoom

These tools were selected to balance development efficiency, integration ease, and scalability for future enhancements.

4.5.Risk Analysis and Mitigation

A thorough risk analysis was conducted during the planning phase, and risk mitigation strategies were implemented throughout the project lifecycle. The major risks and corresponding responses are outlined below:

Risk	Impact	Mitigation Strategy
API Integration Delay	High	Backend team pre-tested Moodle APIs using Postman and developed mock data during downtime.
Speech Recognition Inaccuracy	Medium	Implemented text fallback, added mic indicator, and designed UI for seamless toggling.
Role Mismatch on Login	High	LocalStorage and backend validation added to reinforce role-based access control.
Scope Creep	Medium	User stories strictly followed; business analyst monitored Confluence for unapproved feature additions.
Limited Testing Time	High	Early creation of test cases, split of responsibilities between manual and automation testers.
Data Privacy Issues	Medium	Data exchange between chatbot and LMS designed with encrypted endpoints.

5. System Design

5.1.Overall System Architecture

The architecture of the voice-enabled chatbot integrated into Moodle pro and bridges is depicted in the image below. Process The flow starts with the user accessing Moodle and invoking the chatbot through the UI. Chatbot UI communicates with the backend module (Chatbot backend) actually deals with the voice, and text input. Voice input is handled independently using a voice module that transcribes spoken queries and sends them to the backend.

The relevant academic data are obtained by the chatbot backend through Moodle's Web Services API. This API interfaces with both the Moodle LMS and Open Ai's language model using Open API to produce smart responses that are context-sensitive. All such interactions are logged by a Logging Service for monitoring and debugging. A lightweight database is also kept (Pref index) with the session data, user roles, and preferences. This modular, layered design scales, provides security, and a user-friendly experience for the students and instructors.

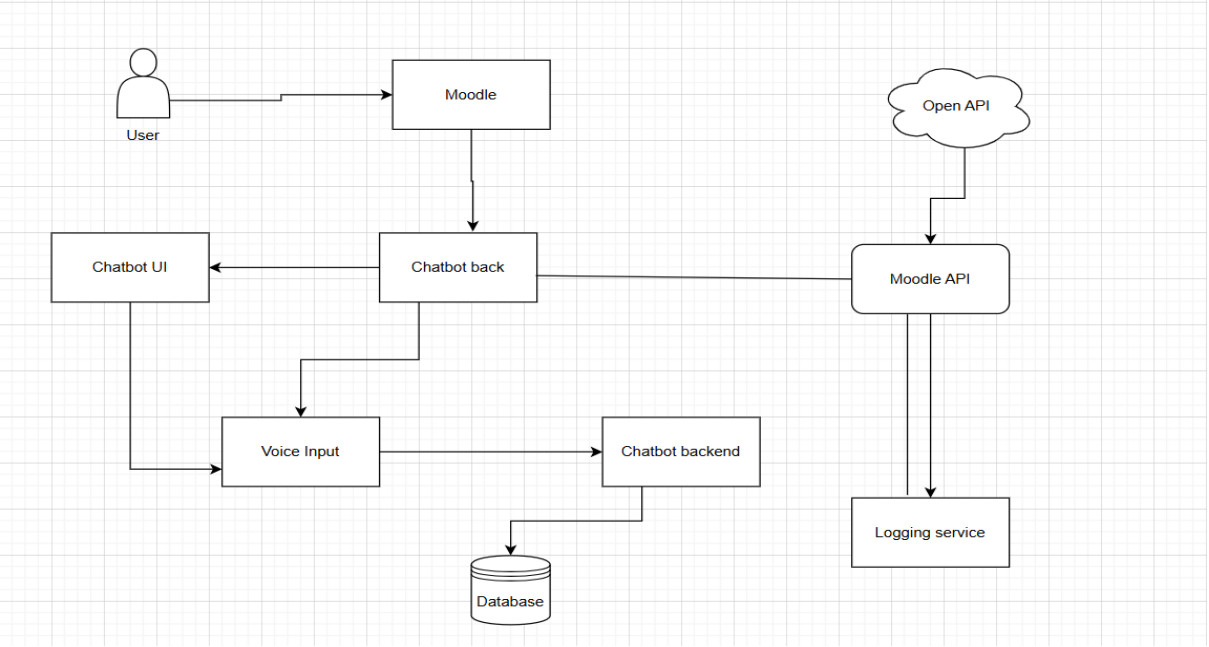


FIGURE 8: SYSTEM ARCHITECTURE DIAGRAM

5.2.Functional Requirements

The chatbot system meets the following functional requirements:

- Enable students to query academic information using voice or text.
- Support role-based access (Student, Instructor, Admin).
- Authenticate users via Moodle credentials.
- Retrieve course data, assignment details, and grades from Moodle API.
- Transcribe voice inputs and read out chatbot responses.
- Display personalized dashboards based on user roles.
- Log and analyse chatbot interactions.
- Allow toggling of voice input settings and session reset.

5.3.Non-functional Requirements

The chatbot system adheres to the following non-functional requirements:

- Performance: Response time under 2 seconds for text, under 3 seconds for voice.
- Availability: 99.5% system uptime.
- Security: Role-based access control and secure API communication using HTTPS.
- Usability: Accessible UI with keyboard navigation and screen reader compatibility.
- Portability: Runs on desktop, tablet, and mobile devices via responsive design.
- Scalability: Modular backend allows for scaling additional services and APIs.
- Maintainability: GitHub version control and modular codebase for easy updates.

5.4. User Interaction Flowchart

The below flowchart shows the end-to-end user interaction of Lumi Chatbot in the context of Moodle. The communication is initiated when the user clicks on a floating button placed inside Moodle that redirects the user to the chatbot interface.

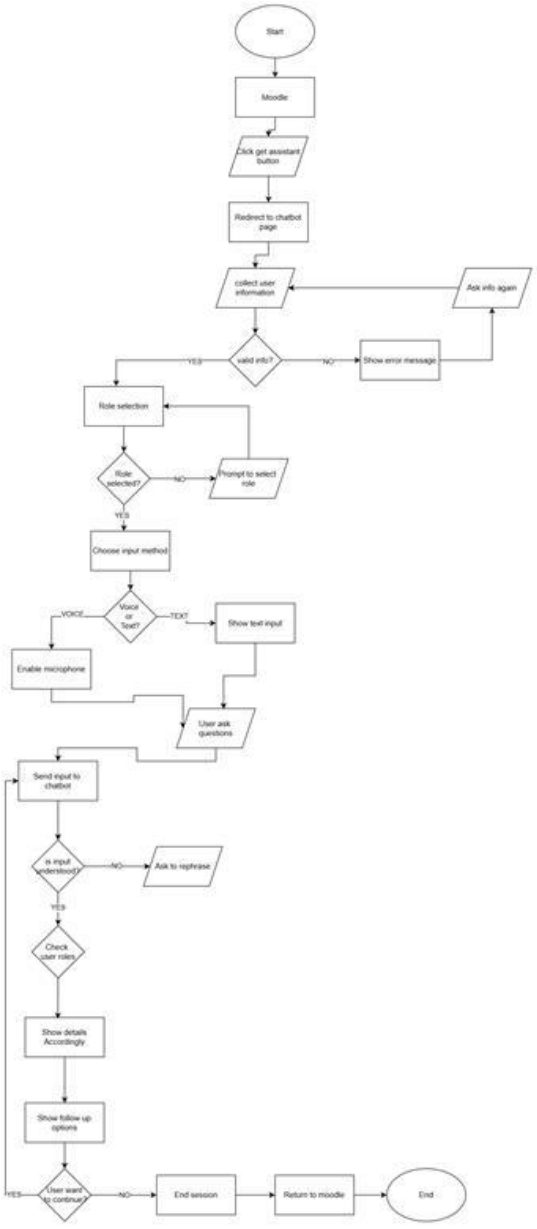
The flow begins with capturing user details, like an email. Invalid or missing input prompts the user to enter the correct details. After confirmation, the chatbot guides the user to the role

selection screen. If a user fails to specify any role, the program will ask the user again and again until a valid role is chosen.

Once the role has been chosen, users can then select their preferred method of input: voice or text. For the voice input, the device can turn on the microphone and the user can say their query. With respect to text input, a text box is provided. Then the chatbot passes the answer to the next node if the sentence is correct and understandable, or asks the user to reform the sentence which should be then used for entering the next node in other case.

Once a valid query is recognised the system checks the user (student/instructor) role and retrieves the relevant information through Moodle API. Depending on the user role and context, academic information is presented.

Following the results are presented, the chatbot offers on the spot next steps. User can decide to either carry the conversation on or to finish the chat and get back to Moodle. This flow is a guarantee of a powerful role-based entrepreneurial context aware experience to enrich students LMS experience.



USER INTERACTION FLOW

5.5. Use Case Diagram and Descriptions

The primary actors in the system are *Student, Instructor, LMS Admin*, and *Chatbot System*. Each has specific use cases.

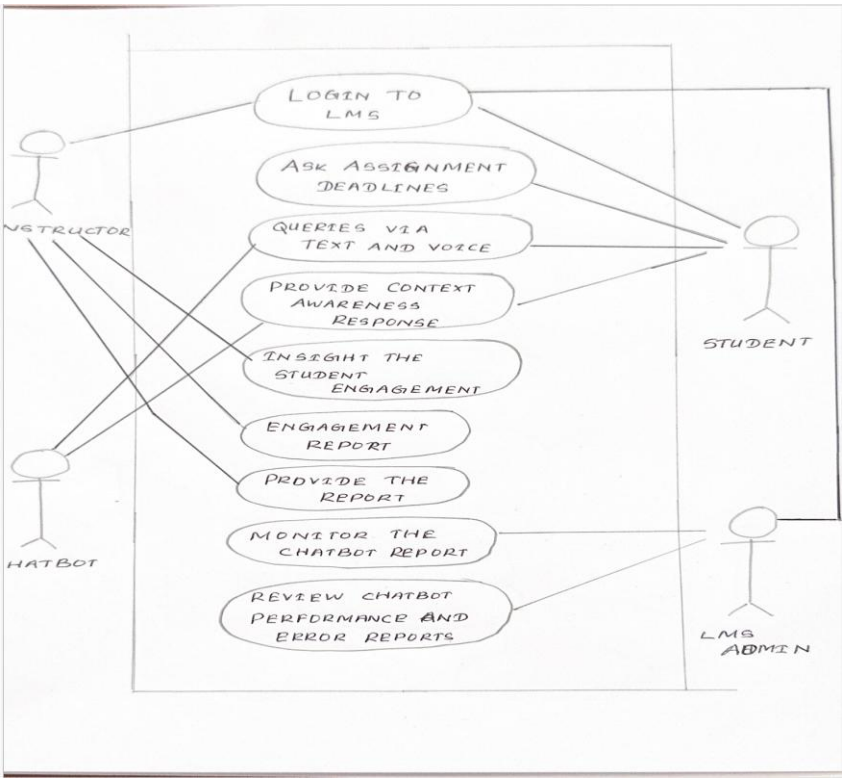


FIGURE 9: USE CASE DIAGRAM FOR ROLE INTERACTION

5.6. Sequence Diagrams

This sequence diagram represents the full interaction a user can do with the Lumi chatbot to ask for information related to an assignment, either using a radio or a text input. The user logs into Moodle at the start of the process (authentication is done by the Moodle auth module/api). Once the user logs in, it's routed to the Chatbot UI. In this example, the chatbot is asking the user to provide information about themselves and to choose if they are a student, an instructor etc.

After the user picks a role and asks a question — "When is my next assignment due? — so the bid is recorded either by text or orally. In case of using voice, the voice is converted to text through the Voice Input Module and Voice-to-Text API. The text and spoken message are transmitted to the Chatbot Backend.

The chatbot authenticates the user, and gets the role; it packages the input query and sends it to the Operator API in the backend processing. The Operator API sends requests to the Moodle LMS API to retrieve DbContext for the assignment and get necessary data from the Database. The LMS response is operated upon to generate an interpretable context aware response, which is then shown in the Chatbot UI. This series of events provides secure, contextual and role-based interaction.

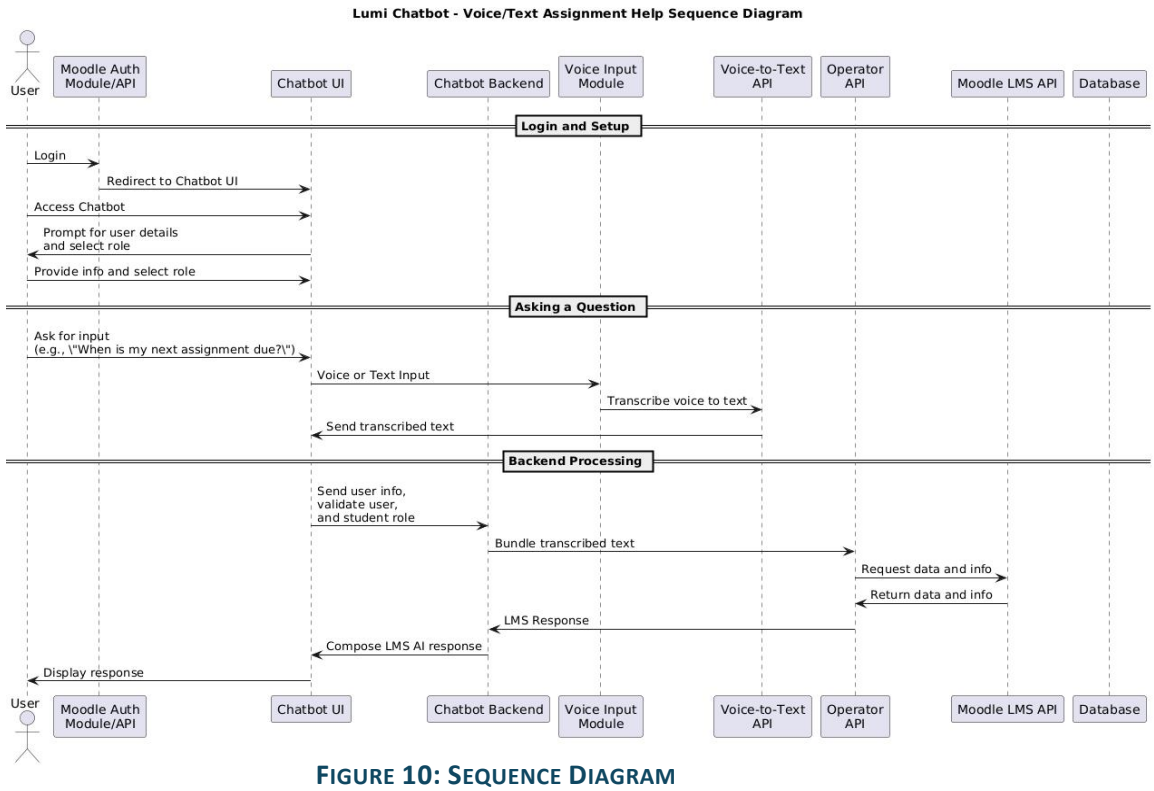


FIGURE 10: SEQUENCE DIAGRAM

5.7. Component Diagram

The detailed structure of Lumi Voice-Enabled AI Chatbot system illustrated as a component diagram in Figure Communication diagram of the Lumi Voice-Enabled AI Chatbot. FSM is presented in this figure. In the heart of it is the Chatbot Backend which consists of three major modules – Authentication, Context Manager and FAQ Management. These serve to verify whether the user is human, keep the conversation current and manage frequently asked questions – respectively.

The backend connects to two important APIs, the OpenAI API Connector which provides natural and intelligent responses and the Moodle API Connector, which provides auth with some student data to retrieve current academic information, grades, content and tasks.

Users interact with BAs via chatbot UI with Voice Input Processor (VIP) using either voice or chat text. The UI takes an input and sends it to the back end, parsing and displaying the response.

The system is supported by two Database components. One that stores session data, user roles and FAQ content, and another that interfaces with the Logging Engine to log every user interaction and transaction for troubleshooting and auditing.

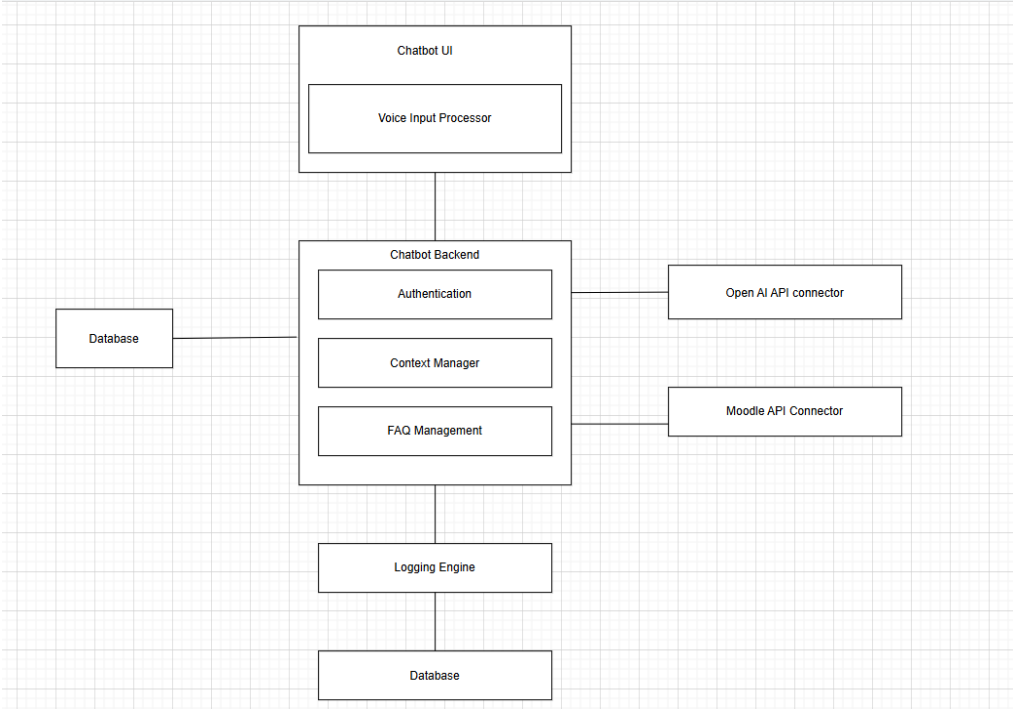


FIGURE 11: COMPONENT DIAGRAM

5.8. Database Design (ERD & Schema)

Although the system is mostly API-driven, a lightweight internal database or local storage is used for session, user role, and interaction history management.

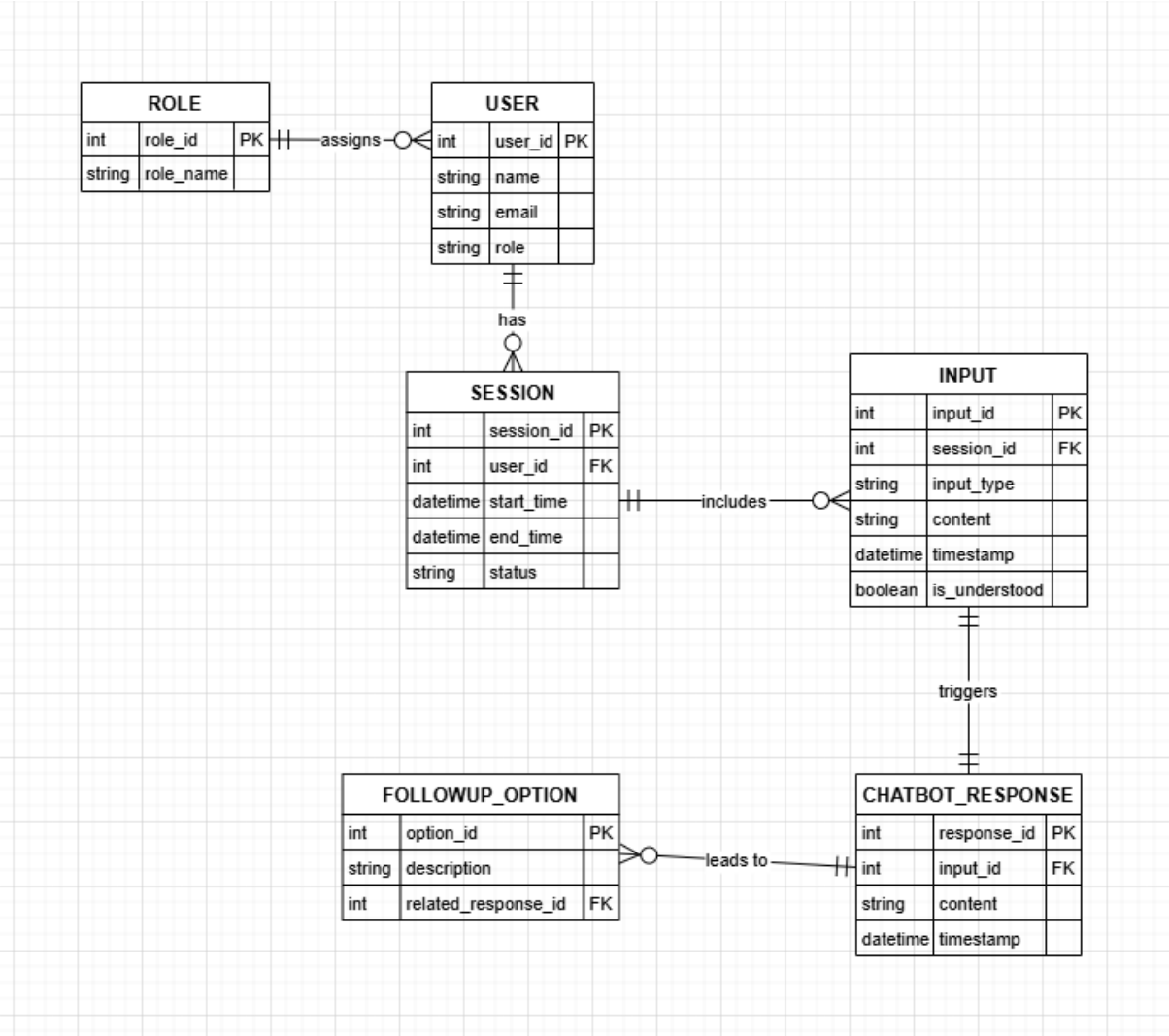


Figure 12: ERD

5.9. UI/UX Design and Wireframes

During the development of the chatbot interface, user-centred concerns regarding clarity, responsiveness, accessibility and consistency were key. To not break user experience and create visual clarity we designed all UI/UX and wireframes in Figma and started implementation. Design Process Using Figma The UI/UX design workflow was organized as follows:

Wireframing:

- Low-fidelity wireframes drafted in Figma for all major interface page views (Welcome, Role Selection, Ask Lumi, Student/Instructor/Admin Dashboards, Settings, FAQ, and Reset Password).
- These wireframes assisted in establishing layout structure, navigation flow, and page element dominance.

Component Design:

- In Figma we created reusable components for buttons, input fields, voice icon, toggles, cards, modals to keep a not too obvious cohesion over pages.

- Design system: A design system, with standardized colours, fonts, paddings, icon sizes and spacings is created and maintained using Figma styles and auto layout.

Interactive Prototypes:

- Figma's prototyping functionality was implemented to model page-to-page navigation and depict user actions (like clicking on "Submit", switching voice input).
- This provided early stakeholder input before any development began.

Role-Based Views:

- Different flows were created including Student's, Instructor's, LMS Admin's in Figma.
- Every role-based dashboard was designed with visual alignment to its capabilities and user requirements.

Developer Handoff:

- Figma's "Inspect" was also used by the frontend team to get measurements, colours, font and spacing.
- In HTML implementation, design tokens were directly applied to Bootstrap and custom CSS classes.

Final Implementation

- The designs for Figma were converted into real components with JavaScript, Bootstrap, and custom CSS with pixel-perfect consistency in design and code. Refinements were carried out incrementally in a sprint-like fashion through testing and feedback loops with stakeholders.

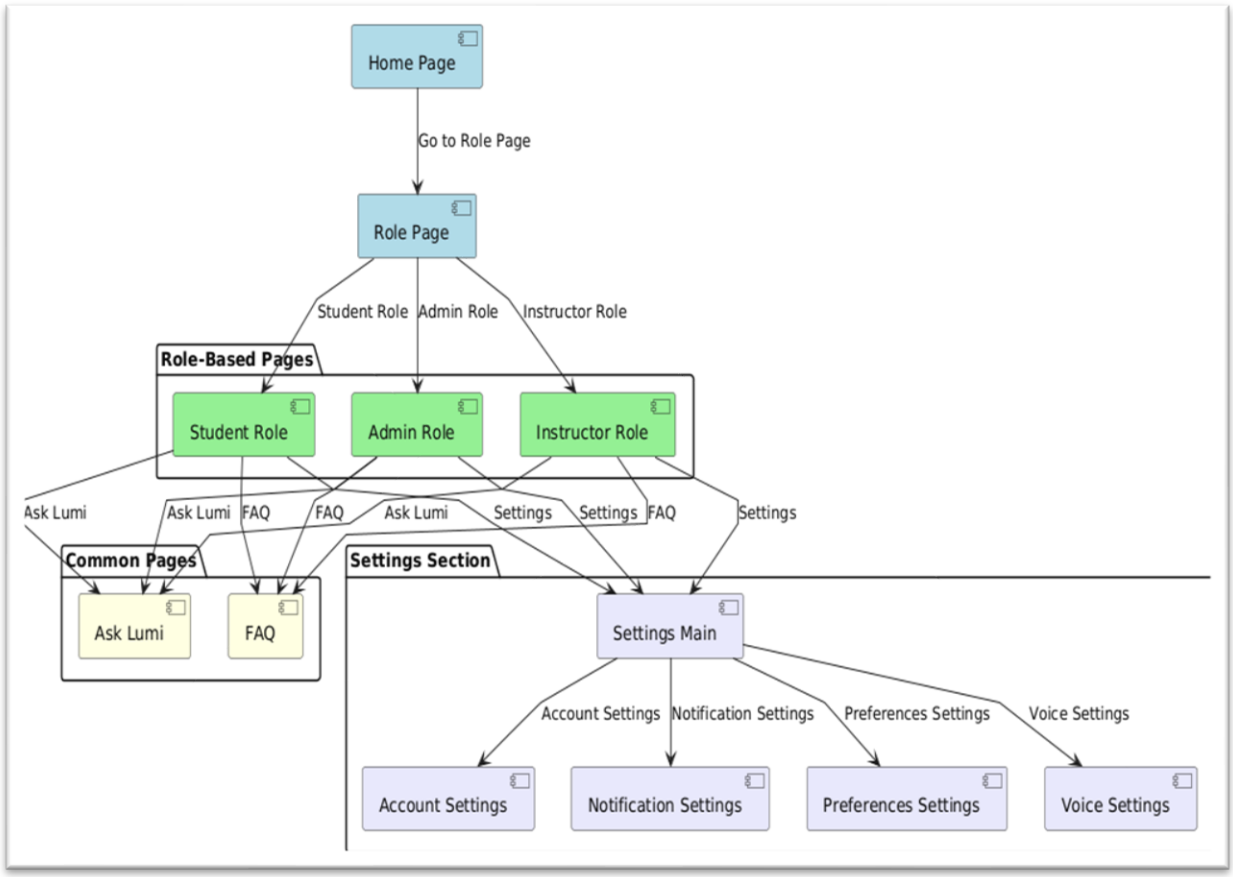


FIGURE 13: UI FLOWCHART

UI Flowchart:

- Home Page → Role Page.
- Role Page → Student, Admin, Instructor.
- Student, Admin, Instructor → Ask Lumi, FAQ, Settings.
- Settings → Account, Notification, Preferences, Voice Settings.

Key UX Elements:

- Role-based layout customization.
- Mic button for voice interaction with visual feedback.
- Accessible font sizes, ARIA labels, and keyboard navigation.

5.10. Role-Based Interaction Flow

For personalized & effective communication, Lumi Chatbot has been developed with role-based interaction model. This design will categorize the chatbot to operate dynamically on the Student, Instructor, or LMS Admin side. Two visualizations have been developed to illustrate this reasoning. Thus, the navigation flow below in UML style describes for each role how it will navigate through the Lumi interface:

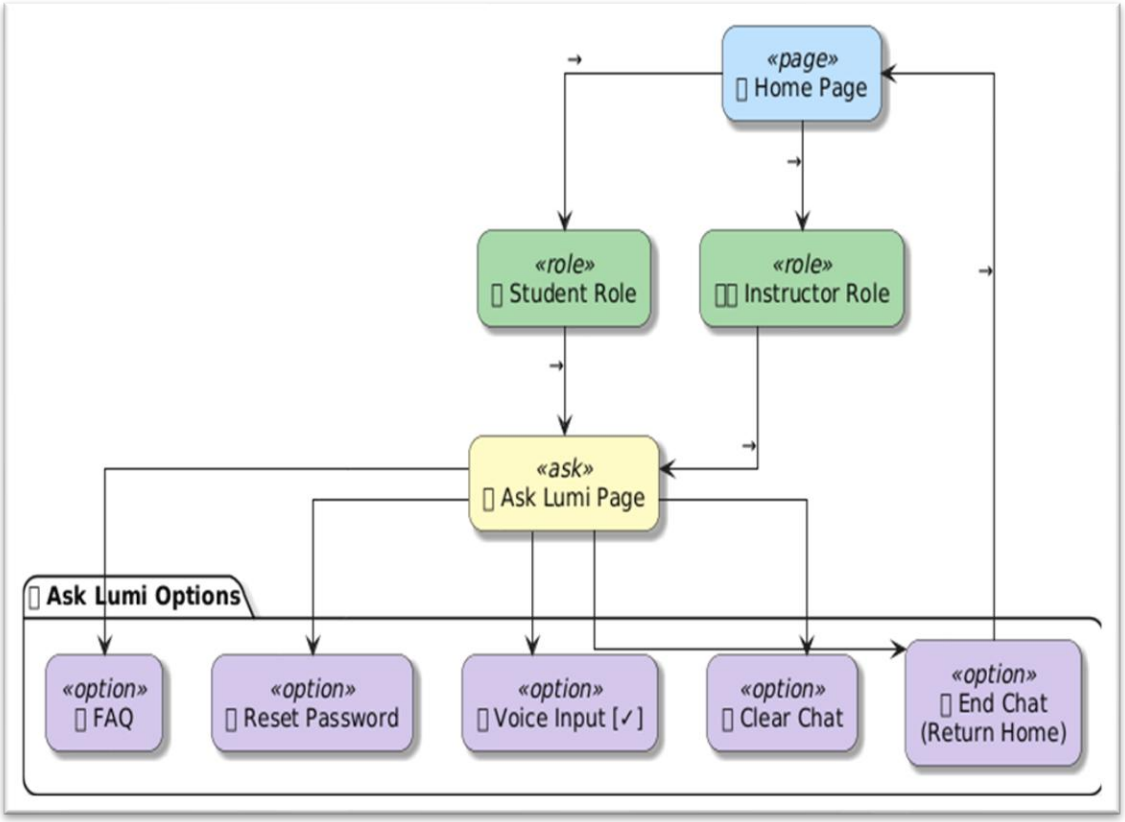


FIGURE 14: UI INTERACTION FLOW

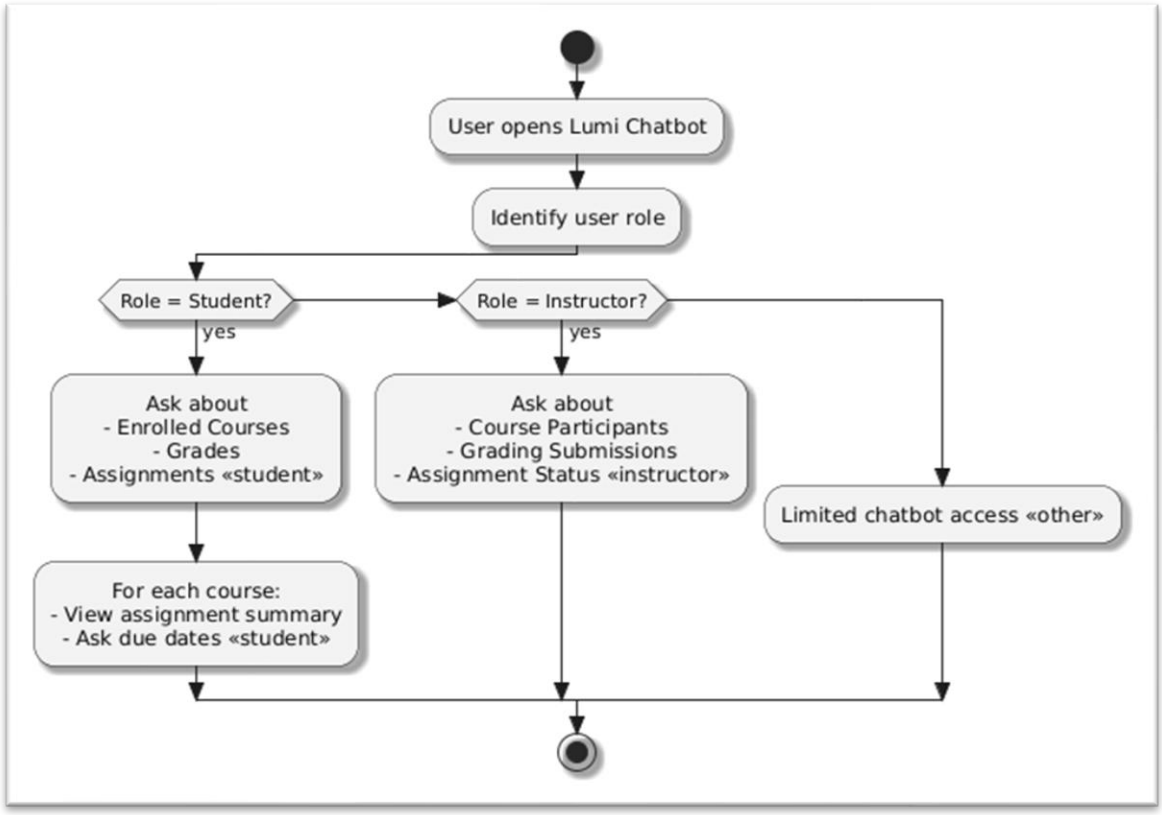


FIGURE15: ROLE-BASED INTERACTION FLOW

Role-Based Chatbot Response

The Home Page acts as a single, common gateway for all users, which presents a single user interface allowing them to pick their role, e.g., Users play one of three roles — Student, Instructor, and LMS Admin. User Interface After selecting their user role, users are redirected to their individual dashboards, allowing them to access the specific features assigned to their access level. No matter what the role, users make their way to the Ask Lumi for all chatbot conversation needs. This is where users will find adjacent options to FAQ, Reset Password, Voice Input On/Off, Clear Chat, and End Chat. It is based on a modular design, guarantees a homogeneous user experience and enables contextual awareness thanks to dynamic chatbot responses with data aggregated on telemetry.

6. System Implementation

6.1. Frontend Development (Chatbot UI)

The front-end for Lumi chatbot was developed in JavaScript, HTML5 and CSS3 with Bootstrap to create the responsible and accessible/usable user interface. The layout is focused role-based and voice/text enabled interaction and minimalistic for educational use. Key pages & UI elements are:

Welcome Page

It is the initial contact for the user, with Lumi's branding and an inviting message. User enters Email Address in the interface and voice input options are available (microphone icons). This makes it easy to access and get started on.

Role Selection Page

Upon authentication, the user is prompted to select their role as either Student, Instructor or LMS Admin. This choice dictates the UI flow and the form of data displayed across the application. It then serves the role to the context for role specific logic.

Dashboard Pages (Role-Based)

For each of these roles, you are offered a tailored dashboard based on your operations:

Student Dashboard

Features buttons for:

- Enrolled Courses
- Upcoming Deadlines
- Submit Assignment
- View Grades
- Notifications
- Ask Lumi

Instructor Dashboard

Features buttons for:

- Manage Courses
- Upload Materials
- Grade Submissions
- Set Deadlines
- Announcements
- Ask Lumi

Admin Dashboard

Provides the following administrative functionality:

- Manage Users
- Role Access Control
- Platform Stats
- Global Announcements
- System Settings
- Ask Lumi

Ask Lumi Page

This is the main interface screen where the user can talk with the bot through voice or text. The chat interface includes:

- Chat history window
- Message input field with placeholder
- Button for microphone for voice.
- Button for submitting and sending message
- Settings gear to adjust your UI settings

This user interface provides real-time, permissive, two-way communication between the user and Lumi, and can be used to input speech or written text.

Reset Password Page

The code for this page is not completing the backend request, but it provides a nice clean interface for the user to enter their email or username, new password (up to 16 characters) and confirm password. There is a 'Back to Chat' link for easy access, and the UI blends in with the platform design.

FAQ Page

Features a clean design intended to get common questions answered first and fast, the FAQ area has expandable questions (accordion style), providing visual feedback. This results in less work for the user to type or issue voice requests and improves support efficiency.

Settings Interface

A gear icon has a universal settings panel. It contains:

- Voice Input settings: voice on/off, continuous vs. push-to-talk
- Options: light/dark theme, font size, language
- Notification control: push alerts, sound control, email toggle
- Account Settings change password, associated accounts

All subsections are visually consistent and make sense for grouping to enhance UX. Users can also customize their interaction mode, visual style, and alert type.

To sum up, the frontend-UI usability of the Lumi Chatbot is fully interactive, responsive and inclusive to the various user roles, and incorporates AI-assisted voice technology. The layered architecture of the chatbot and accessibility saving approach makes it user-friendly for a variety of users with different devices and abilities.

6.2. Backend Development (Node.js API Server)

The server side of the chatbot system was developed in **Node. js** and the **Express. js** web application framework and acts as the main communication layer for the chatbot frontend to interact with external components like the **Moodle Web Services** and the **Open AI GPT API**. It is responsible for dispatching user input, keeping track of sessions, performing access control checks based on the roles that are assigned to the user, identifying and dispatching flows to the correct API, and creating standardized responses for display in the user interface.

The backend is designed as a **modular and loosely coupled system** that is implemented using the **Model-View-Controller (MVC)** paradigm. All the business logic is placed in the corresponding controllers' files (for example: **userController. js, gradesController. js)** Routes are also nice to define clean in the main server file (**server. js**). This makes it easier to read, easier to debug, and more future proof.

Secrets (Moodle tokens, Open AI keys, etc.) are stored in similar environment variables by the **.env** package and obtained from the **.env** file. API communication with external resources is implemented with node-fetch, requests are made with URL encoded **POST** methods, since that is what's expected by Moodle web-service endpoints.

Additionally, a **user context object** is stored in the back-end in order to store certain user session variables (e.g., role, what courses were most recently accessed, and preferences for voice). This is a good fix for small usages but for enterprise scalability this can be done along with session storage or databases as well.

To summarize, the backend is the smart processing part of the chatbot where the user request gets translated into a data query or an AI prompt, and answers are processed and privacy limitations are managed according to the user role.

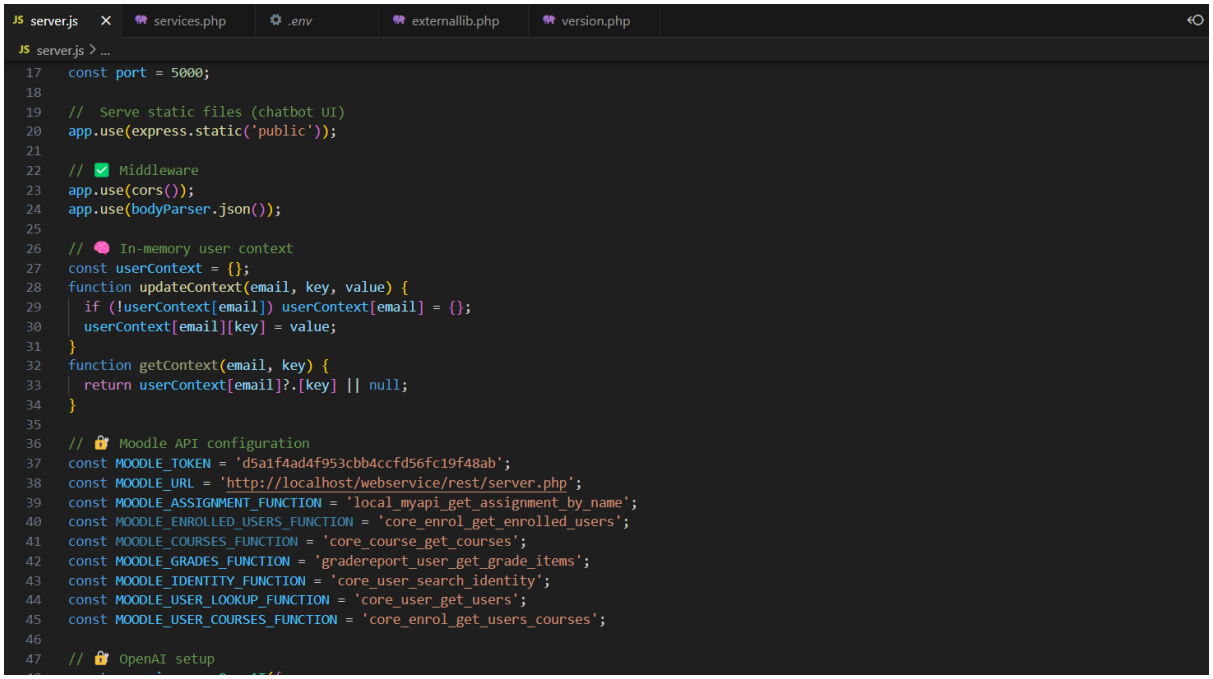


FIGURE 16: MIDDLEWARE SETUP CONFIGURATION

6.2.1. Moodle Integration using Web Services

The backend has one main module called **Moodle Web Services** which describes the methods the chatbot communicates with **Moodle's Web Services API** so they return information about user's active course enrolments, grades and assignment submissions. In this way, users (either students or professors), can now engage with the Moodle LMS using natural language sentences through the chatbot.

The backend interacts with the **RESTful API** of Moodle through **HTTP POST** requests with a secure web service token **(wstoken)**, the name of the function (**wsfunction**) and the **URL-encoded parameters.** The chatbot interface retrieves all responses in JSON format from the backend; the JSON is then parsed for display to the user.

Moodle Functions Used

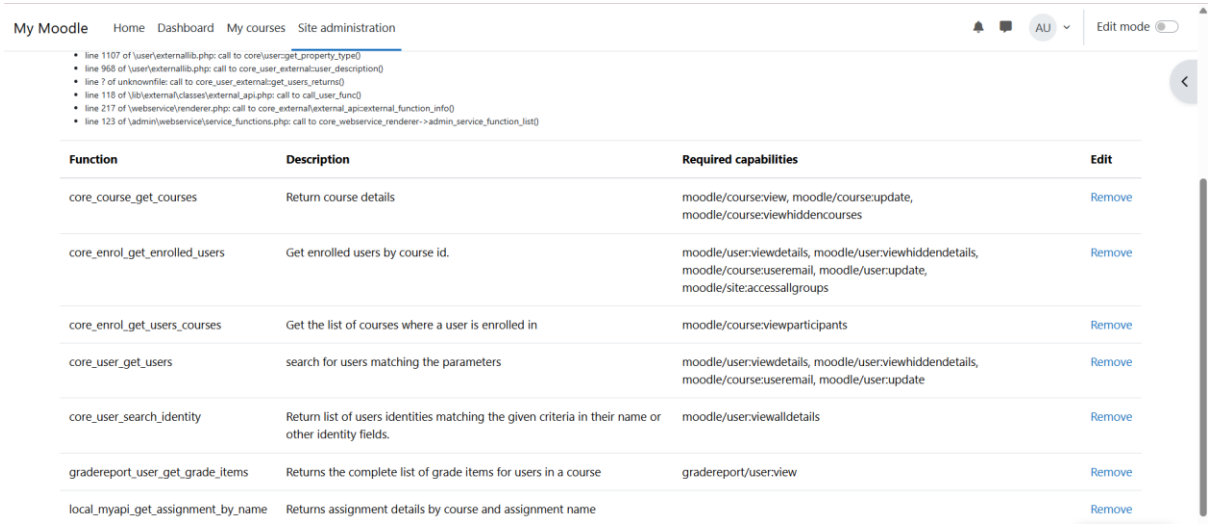


FIGURE 17: MOODLE EXTERNAL SERVICES

The project made use of a core Moodle functions member and created its own functionality:

Function Name	Purpose
core_user_get_users	Fetch user details using email
core_enrol_get_users_courses	Get the list of courses a user is enrolled in
core_enrol_get_enrolled_users	Retrieve all users enrolled in a specific course
core_user_search_identity	Search users based on name or email
gradereport_user_get_grade_items	Fetch grades for a user in a specific course
core_course_get_courses	Retrieve general course information
local_myapi_get_assignment_by_name	Custom-built function to fetch assignment deadline & intro

API Integration code structure

All the Moodle calls are performed by means of the **fetch()** of node-fetch with headers **'application/x-www-form-urlencoded':** The parameters are dynamically constructed by **URLSearchParams().**

FIGURE 18: USER ROLE MANAGEMENT

This yields a JSON payload that is parsed and used to retrieve particular data or ascertain the role of the user.

Dynamic API Routing

On the backend there are endpoints for access to each type of Moodle data such as:

- **POST /user/details** – returns full name.
- **POST /grades** – get grades for a student
- **POST /grades/all** – returns all student grades (instructor only)
- **POST /assignment** – get assignment description and due date
- **POST /enrolled-users** – returns a list of users who are enrolled in a course

Every one of these route has a corresponding Moodle function and its own error handling, data validation and role-based access control.

Security Considerations

- To a certain extent, Moodle web service token is stored safely. env files.
- Only permitted functions are ticked in in Moodle's external services settings.
- All API requests are secured with input validation and HTTP status responses (i.e.., 400 if the email is not found, 403 on role violations).

To summarize, the integration of Moodle is the foundation of the bot's academic information gathering capability. The chatbot can also emulate human-like dialogue, providing real-time, tailored educational content using a set of secure, token-based REST API calls and endpoint functions.

6.2.2. Open AI API Integration for LLM Responses

To support natural, conversational, intelligent, context-aware chat inside the chatbot, the backend interacts with the Open AI GPT-4 language model through the official Open AI Node. js SDK. This combination of two approaches allows the chatbot to help the users with answers to the question that is not data-specific and probably ambiguous, where the user needs to be navigated or guided to the appropriate explanation.

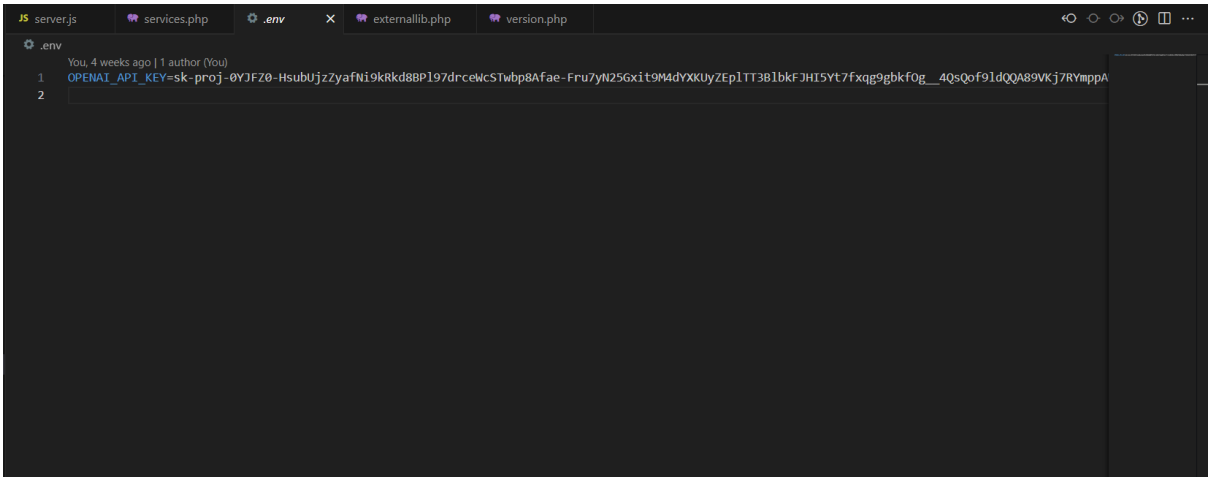


FIGURE 19: OPEN AI API SETUP

API Key Management

The Open AI API key is stored safely in the. env environment file:

This is retrieved from the backend with:

OPENAI_API_KEY=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

Use Case in /chat Route

/chat route is the endpoint I have chosen to deal with natural language requests regarding a single assignment within a course. It begins by obtaining the details of the assignment with Moodle's API and then composes a prompt for Open AI.

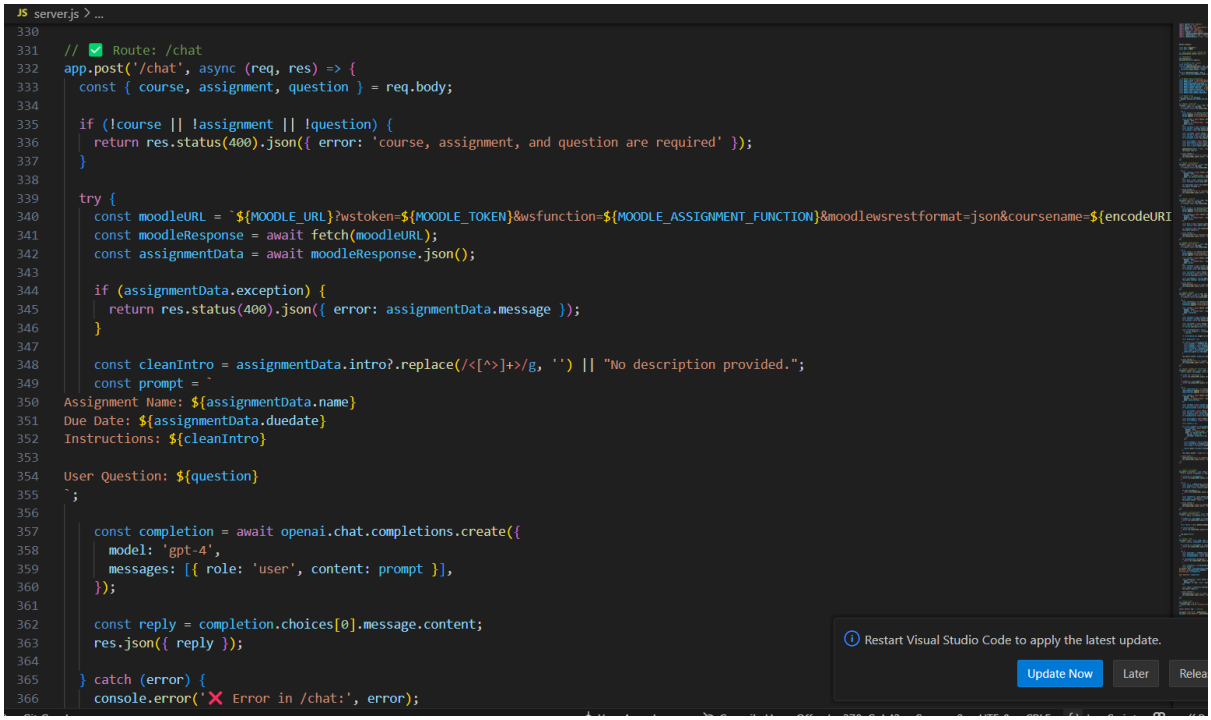


FIGURE 20: /CHAT ROUTE CODE

Prompt Design & Context Awareness

Prompts are then dynamically formed by:

- Assignment name
- Due date
- Cleaned-up instruction text (no HTML)
- User's question

This way, it preserves context, so GPT can see what kind of academic question is being asked and answer appropriately.

Handle Fallback & exceptions

At the bottom of the response pipeline, there's a fall-back logic that makes the application continue to be responsive to the user (even when we don't have data from Moodle or Open AI doesn't respond). These problems are handled nicely on the backend: it logs an error or returns a default reply.

- **Security:** The Open AI API key remains securely on the server and never in the hands of the client.
- **Performance:** All request-response cycles are handled on the server side.
- **Observability:** We tracked rate limits as well as token consumption on the Open AI dashboard.
- **Coming soon:** Count the tokens, stream responses, and education-aware model taking are in the pipeline.

6.2.3. Speech-to-Text and Text-to-Speech Modules

To achieve better accessibility and a higher user experience, voice interaction is also provided, relying on the browser-based Speech-to-Text (STT) and Text-to-Speech (TTS) services that exploit the Web Speech API. These modules offer the capability to talk their request and hear the chatbot answer, which consequently lead to a more natural intuitive user experience.

Speech to text (Voice input)

The Chatbot uses the Speech Recognition API to record the user's voice and interpret this into text. Then we send that to the backend.

Key Features:

- Triggered by clicking the mic icon on the chatbot UI.
- Takes one utterance and turns it into written text.
- Fills in input field with the transcribed query automatically.

Text-to-Speech (Voice Output)

The chatbot has replied (with spoken words) through the SpeechSynthesis API.

Key Features:

- Set PercentMuted to less than 100 if you want to allow any bot response to get through.
- This feature is toggled off using settings.
- Uses en-US character's default voice, but you could also use other gender-gender/region-region voices.extensions.getMethodPi.

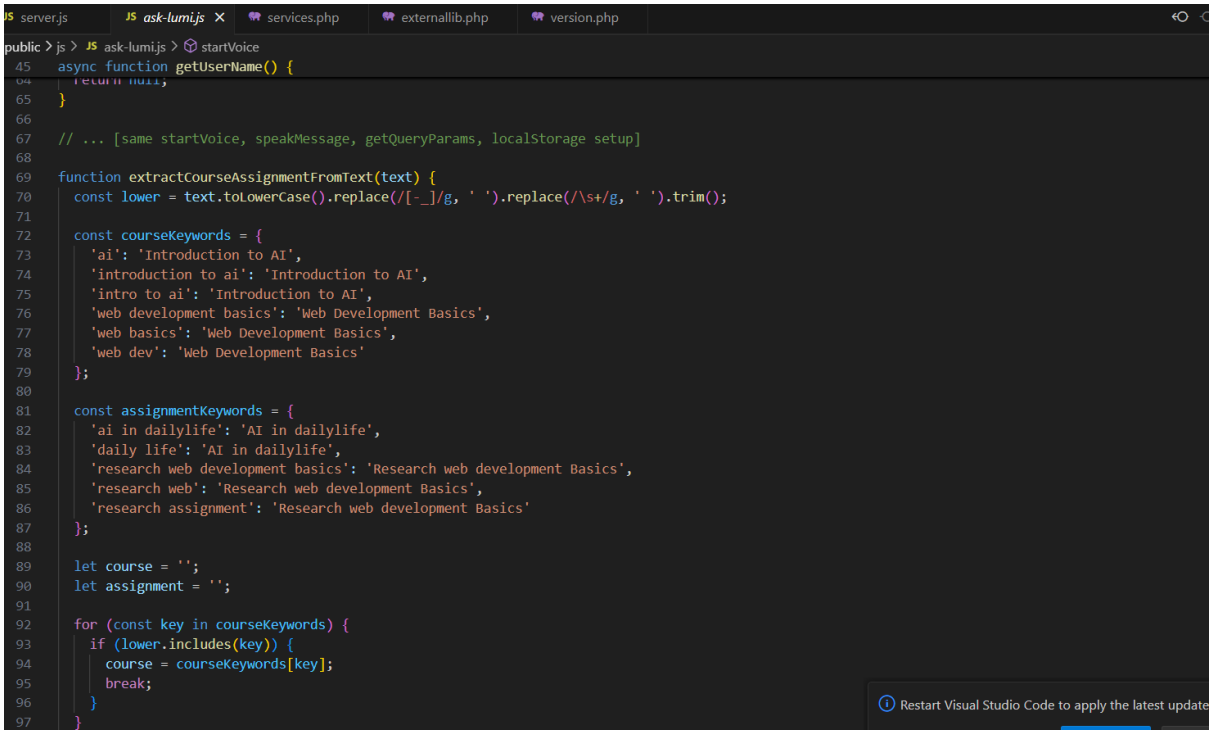


FIGURE 21: VOICE FUNCTION

6.2.4. Role-based Contextual Logic

To make sure the chatbot can provide secure, personalized and correctly contextualized answers the backend uses role-based context logic. With the above method, the chatbot can recognize if a user is a student or teacher (instructor), and can give a response based on the permissions and context of each user.

Roles Identification Using Moodle API

Backend is based on **core_user_get_users** and **core_enrol_get_enrolled_users** where the role of the user is identified by the email and the course enrolment data.

Logic Flow in /user/role:

- Get email input from frontend.
- Get user id from core_user_get_users.
- Get them via core_enrol_get_enrolled_users.
- Match by user username against the enrolled list to get their Moodle role (student/teacher etc.).

updateContext() to save the role to the backend

In-Memory Context Handling It stores session level context which can be anything related to that session like user roles/preferences/recent search etc as a JavaScript object userContext.

Role-Based Access Enforcement

The various backend routes are validated for role so that sensitive data can be protected. For example:

Students:

- Can see only their marks and assignments
- No access to data of other users

Teachers:

- May view all students enrolled
- Can retrieve all students' grades for a class

6.2.5. Error Handling and Logging

There is a need to allow middleware to handle errors and log easily on backend to make system reliable, debug gable and user friendly in varies failure cases. The chatbot backend makes use of structured error responses and log messages for each of the API routes and for every call to external services such as Moodle Web Services and Open AI.

Structured Error Handling

Try Catch block: For all the APIs in backend, we have wrapped them with a try catch block to catch exceptions and respond with suitable HTTP status codes.

Status Code	Meaning
400 Bad Request	Missing or invalid input (e.g., email not provided)
403 Forbidden	Role-based access violation
404 Not Found	Resource not found in Moodle
500 Internal Server Error	Unexpected server error or third-party API failure

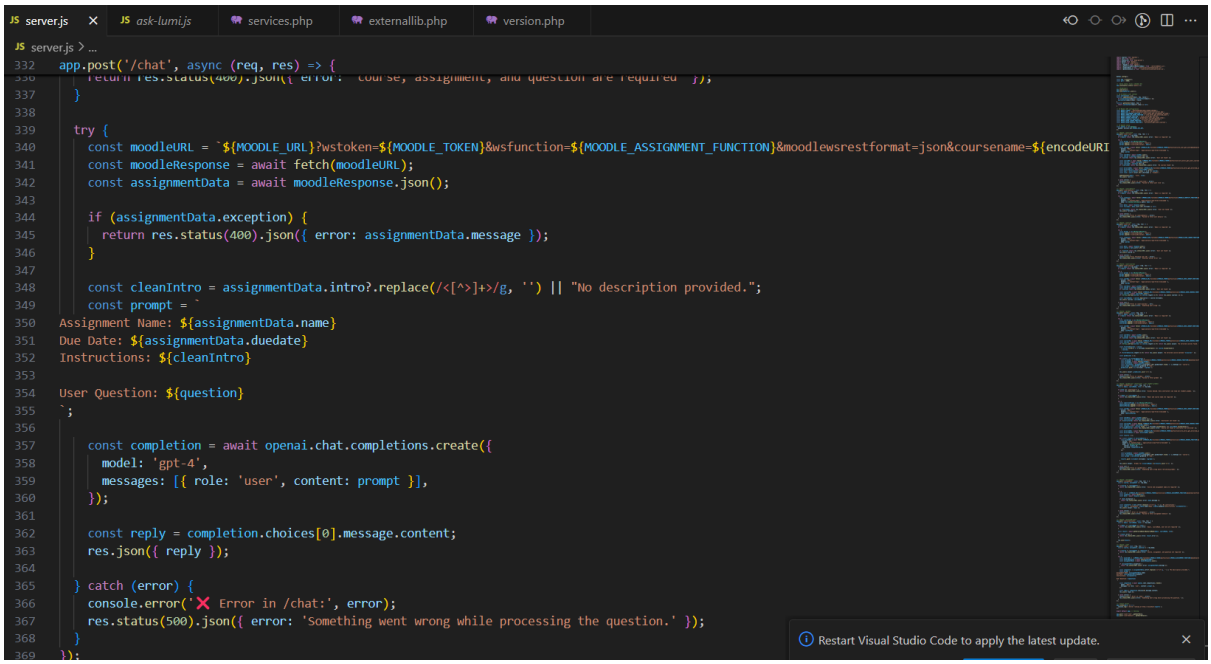


FIGURE 22: ERROR HANDLING IN SERVER.JS FILE

Backend Logging

Logging is by native console. log() and console. error() functions to track:

- API route being accessed
- User feedback (e.g., e-mail or class)
- Results of an API request (success/failure)
- Error messages and stack trace sort of over time became more specific.

6.3. 6.3 Version Control and Development Environment

Developers had an excellent environment to materialise the software code through an established and friendly development environment Visual Studio Code connected to version controlling system GitHub. The Visual Studio Code Configuration Visual Studio Code was adapted with several extensions for better developer productivity and code quality.

This included:

- Auto-formatting and linting through ESLint and Prettier
- Debugging tools to test and track issues quickly and easily
- A modular, neatly organized folder structure to support the project's scalability and comprehensibility.

This setup contributed to a standard and uniform codebase across contributors and easy addition of new writers.

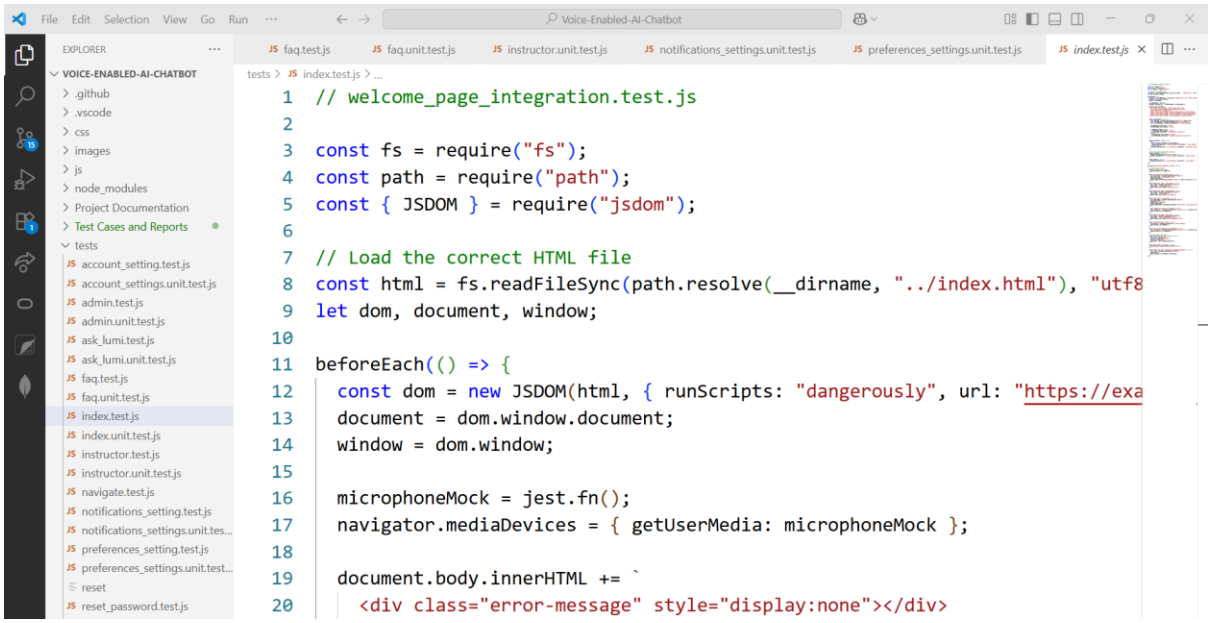


FIGURE23: VISUAL STUDIO CODE CONFIGURATION

Git & GitHub Workflow

GitHub Link: Voice-Enabled-AI-Chatbot

We used Git for version control which was hosted on GitHub. Best practices applied throughout the project:

- Branched strategy was used to differentiate between development, bugs and test tasks.
- Regular commits with meaningful messages allowing traceability
- Merge conflicts were resolved manually with Git commands and recorded in Git CLI
- Updates of project development, documentation and testing were always pushed to the public repository.

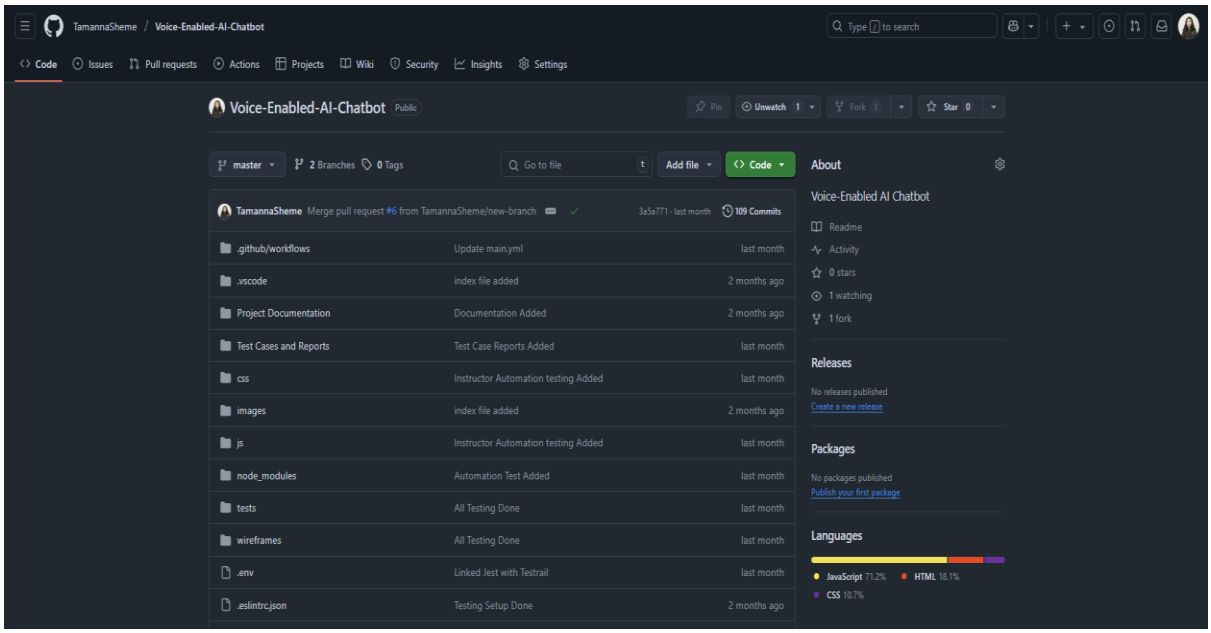


FIGURE24: GIT & GITHUB WORKFLOW

7. Testing and Evaluation

7.1. Project Workflow Overview

For the development of a reliable and effective chatbot, the Lumi project adhered to a design–test–report iterative cycle. The flow of work started from UI design and from there manual and automated tests (we used Jest for that). Results of the tests were recorded in TestRail and linked to Jira for automatic tracking of bug and requirement status throughout the system build. All output (including things like UI docs and bug reports) went into the Documentation module for version control and visibility.

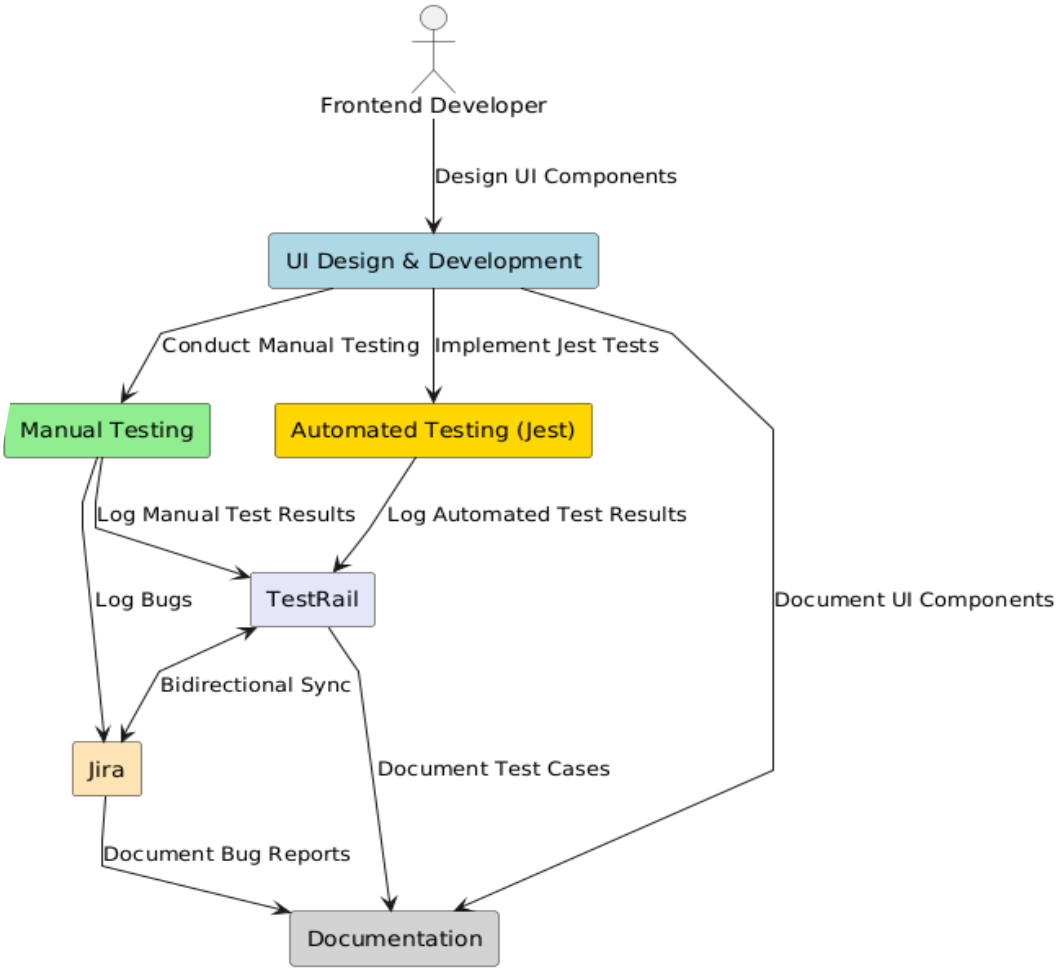


FIGURE25: INTEGRATED UI DEVELOPMENT, TESTING, AND REPORTING WORKFLOW

7.2. Testing Strategy and Environment

A comprehensive, multi-layered testing approach of manual and automated methods was employed for the Lumi - Study Buddy Application to verify system function, integration accuracy, and usability and performance. The process was controlled and observed with industry standard tools as TestRail for test case and test run management, Jira for issue tracking and bug fix workflows and Jest for JavaScript based automated tests.

Our QA lifecycle is synchronized with the Agile Scrum cycle, where development happened in the form of iterative sprints, integration and test cycles that run post feature implementation. Testing was done concurrently with the development process to catch issues early.

7.2.1. Tools and Environments Used:

- **Test Management:** TestRail (https://chatbotv1.testrail.io)
- **Issue/Bug Tracking:** Jira
- **Automation Framework:** Jest with Node.js
- **Execution Environment:** Local dev servers, GitHub CI workflows, and Chrome/Firefox browsers
- **Devices:** Windows/Linux desktops and Android mobile devices (responsive testing)
- **Data Strategy:** Custom test data pools for account roles (Student, Instructor, Admin), settings combinations, and voice/text input testing
- **Source Control:** GitHub for versioned test script management

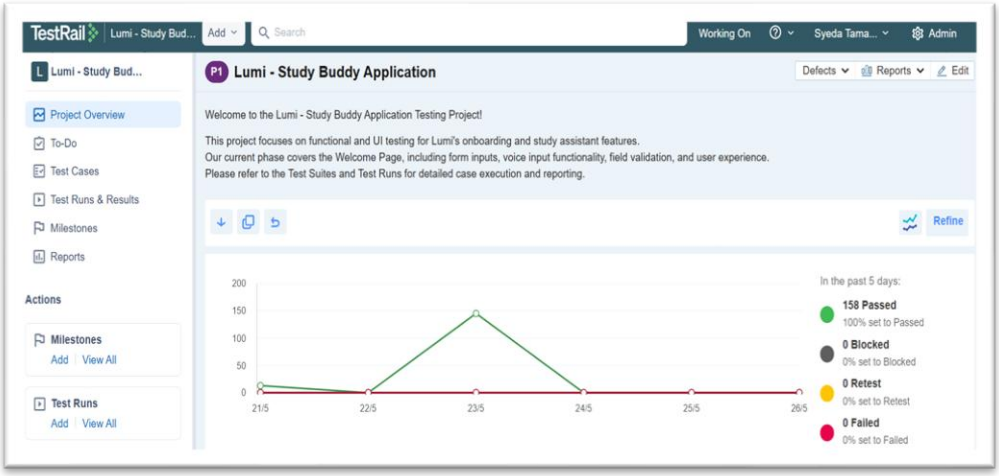


FIGURE 26: TESTRAIL DASHBOARD

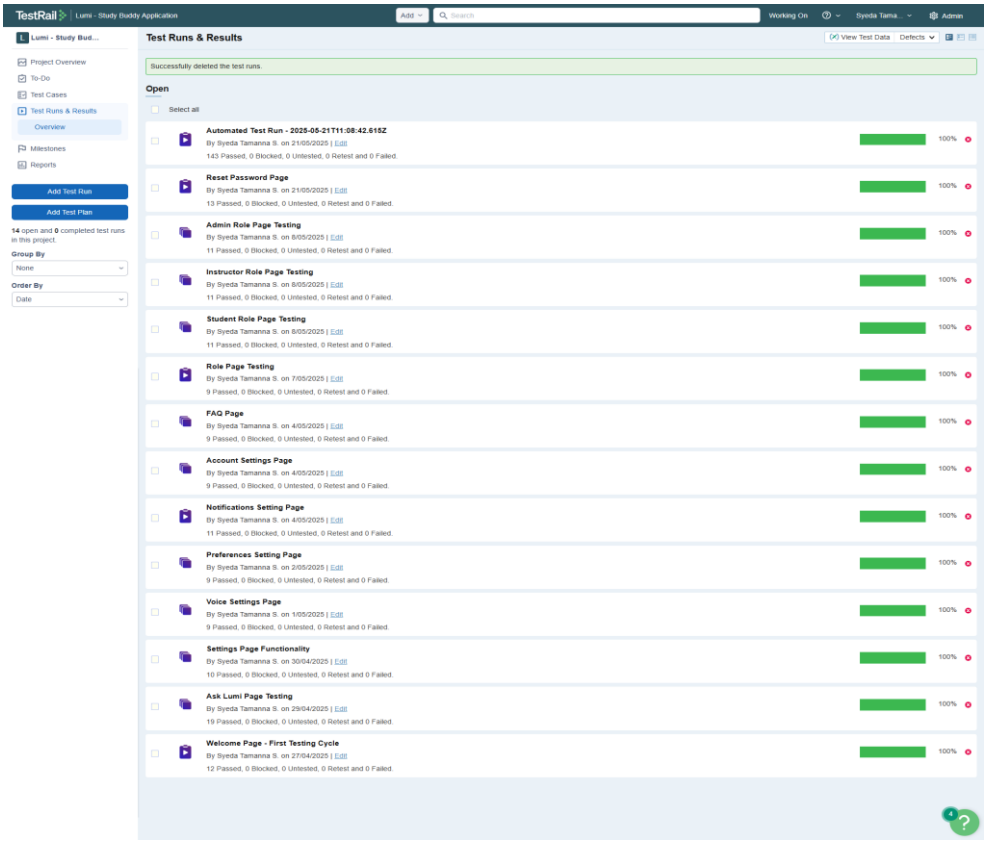


FIGURE 27: TEST RUNS AND RESULTS

7.2.2. Description of Flow:

- Starts with gathering the requirement
- Progress to test case writing in TestRail
- **Branches into:**
  - Manual testing → Reporting of bugs in Jira o Automation testing → Tested with Jest & GitHub CI.
  - Concurrency on to Integration, API and UAT testing
  - Closes with the creation of test report and analysis of feedback

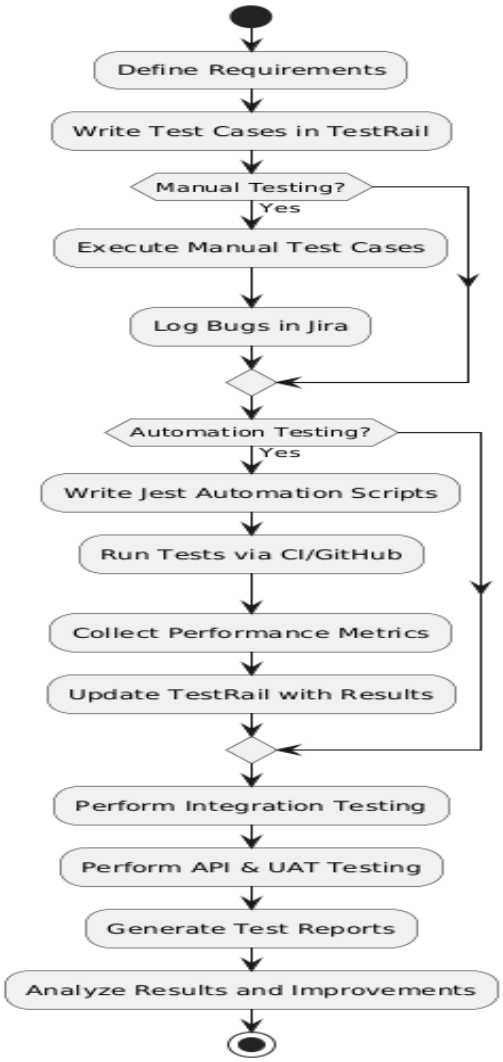


FIGURE 28: LUMI CHATBOT TESTING LIFECYCLE: MANUAL AND AUTOMATION FLOW

7.3. Functional and Integration Testing

The Lumi AI chatbot faced rigorous functional and integration testing to ensure stability and user-friendliness. Functional tests included paths, forms, settings and chat functionality, initially manual and latterly automated with Jest. Integration tests verified full workflows like the chatbot activation, role-based routing, settings update, and AI-response handling. Primary modules that were tested were Welcome, Ask Lumi, Settings and role/dashboard specific ones. It was very much about the interplay between frontend

components, new backend APIs, the voice input system. We ran 286 testcases (manual and UAT) in TestRail and they all passed with 100% test pass rate.

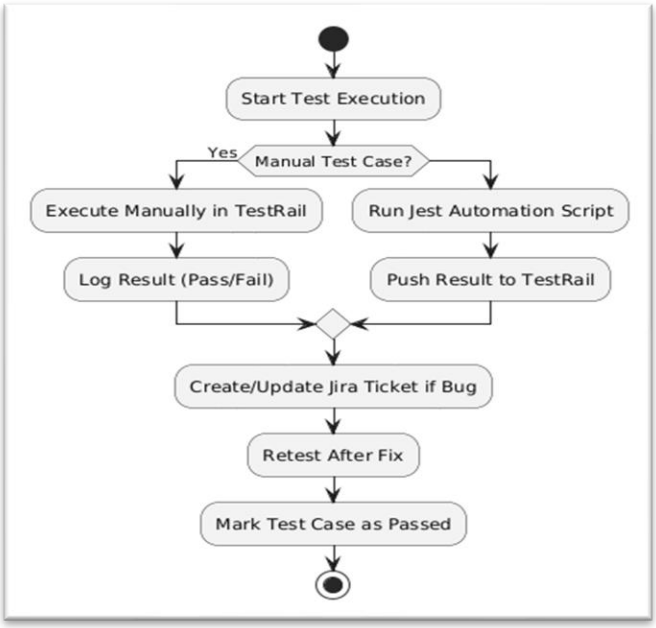


FIGURE 29: FUNCTIONAL AND INTEGRATION TESTING

Test Case **Design (TestRail)**

Functional (F) and Non-Functional (NF) detail testing scripts were authored, organized, and executed via TestRail to validate all aspects of Lumi chatbot. The use cases were developed based on modular and role-based decomposition for successful traceability and good test coverage.

TestRail suite included more than 113 functional and non-functional test cases around major modules. It covered functionality (UI interaction, validations), performance (load time, response time), security (HTTPS and access control), accessibility (keyboard, screen reader), usability (responsive, consistent layout).

Test Case **Repository Structure**

Page / Module	Functional Test Cases	Non-Functional Test Cases
Welcome Page	9	3
Settings Page (Main)	6	4
Voice Settings	6	3
Ask Lumi Page	16	3
Preferences Settings	6	3
Notifications Settings	8	3
Account Settings	6	3
FAQ Page	6	3
Role Selection Page	6	3
Student Role Dashboard	8	3
Instructor Role Dashboard	8	3
Admin Role Dashboard	8	3
Reset Password Page	10	3

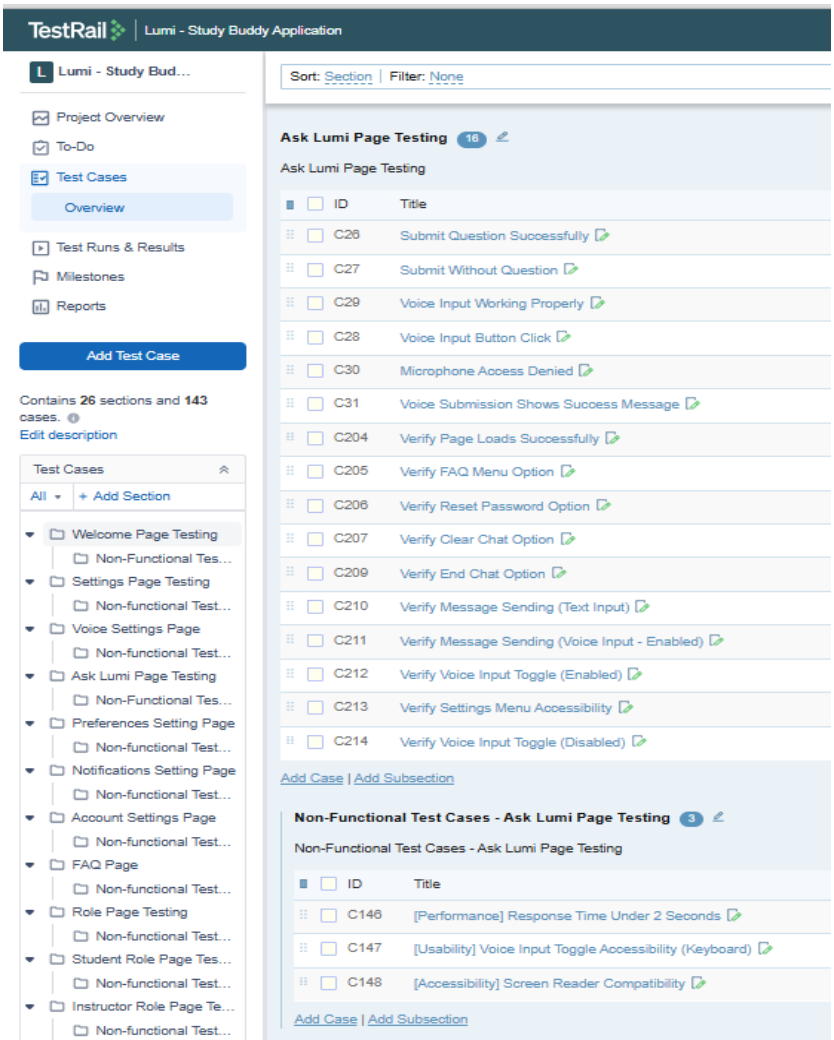


FIGURE 30: TEST CASES IN TESTRAIL

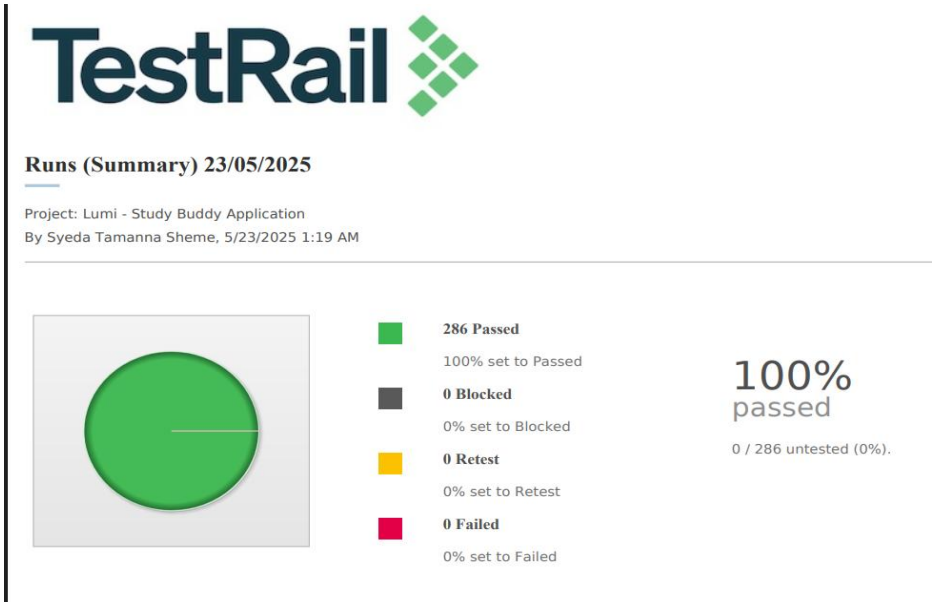


FIGURE 31: TESTRAIL TEST RUN REPORT

Bug_Report_Voice_Enabled_AI_Chatbot .XLSX Saved to Drive

File Edit View Insert Format Data Tools Help

A1 Bug ID

Bug ID	Description/Summary	Steps to Reproduce	Expected Result	Actual Result	Priority
C2	Voice Input - Student ID	1. Click mic icon 2. Speak student ID	Field populates with spoken ID	Field empty	High
C3	Voice Input - Phone Number	1. Click mic icon 2. Speak number	Field populates with spoken numb	Partial capture	Medium
C4	Voice Input - Email	1. Click mic icon 2. Speak email	Field populates with spoken emai	Invalid chars	Medium
C6	Invalid Student ID Input	1. Enter 'S999' 2. Click submit	Should reject input	Input accepted	Medium
C7	Invalid Email Input	1. Enter 'user@' 2. Click submit	Should reject input	Input accepted	Medium
C8	Successful Form Submission	1. Fill all fields 2. Click submit	Confirmation message shown	Submission failed	High
C9	Form Responsiveness	1. Resize viewport 2. Check layout	No horizontal scrolling	Mobile overflow	Medium
C10	Button Hover Effect	1. Hover submit button 2. Verify style	Visual feedback on hover	No effect	Low
C11	Voice Recognition Error Handling	1. Speak gibberish 2. Verify error	Error notification shown	No error	Medium
C9	Form Responsiveness	1. Resize viewport 2. Check layout	No horizontal scrolling	Mobile overflow	Medium
C10	Button Hover Effect	1. Hover submit button 2. Verify style	Visual feedback on hover	No effect	Low
C11	Voice Recognition Error Handling	1. Speak gibberish 2. Verify error	Error notification shown	No error	Medium
C145	[Accessibility] Voice Input Accessible via Keybo	1. Tab to mic 2. Press Enter	Mic activates via keyboard	Could not focus	High
C26	Submit Question Successfully	1. Type question in input field 2. Click Submit	Question should submit successfu	Submission failed	High
C29	Voice Input Working Properly	1. Speak question 2. Verify transcription	Question should appear in input	No transcription	Medium
C31	Voice Submission Shows Success Message	1. Click Submit 2. Verify UI response	Should show success toast	Success message missing	Medium
C147	[Usability] Voice Input Accessible via Keyboard	1. Tab to microphone 2. Press Enter	Should activate voice input	Could not focus button	Medium
C148	[Accessibility] Screen Reader Compatibility	1. Navigate page 2. Verify announcements	All elements readable	Input field not announced	Medium

Bug Report

FIGURE 32: BUG REPORT

7.4. Automation Testing and Results

For fast feedback and supports automated testing, Jest was used to perform unit/component-level testing of the Lumi chatbot frontend. The key modules such as Ask Lumi, Settings, Reset Password and Role based dashboards were wrapped. Testing A total of 32 test suites (215 tests) passed with zero failures and an average running time of ~52.5 seconds. Test results were automatically submitted to TestRail by integration.

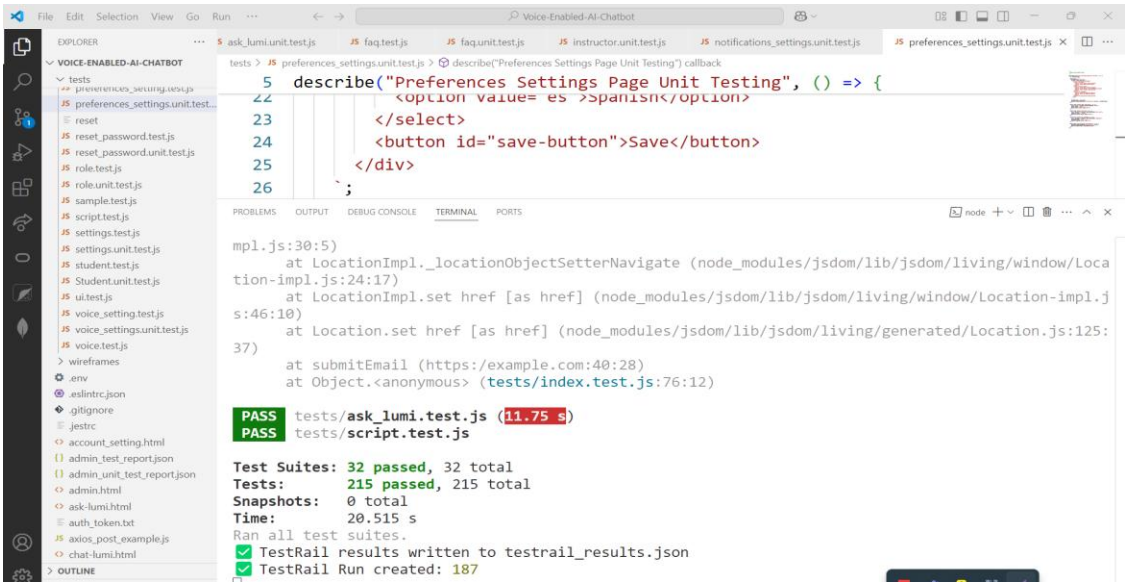


FIGURE 33: VS CODE TERMINAL (JEST EXECUTION)

TestRail Integration Configuration

To avoid duplicating test cases that have already been written and organized in a particular way in TestRail, we decided to integrate Jest with TestRail by creating a custom testrail. config. js file. This setup will talk to TestRail using and is secure. josuah/dotfiles/fifo09: env credentials, no hard codes to secret. The significant parameters are the TestRail host, username, and token, also projectId and suiteId, and a random runName for each test run. This configuration allows you to automatically post results from Jest to the appropriate TestRail suite.

testrail.config.js

```
require('dotenv').config();

module.exports = {
  host: "https://chatbotv1.testrail.io",
  username: "ssheme@students.federation.edu.au",
  password: process.env.TESTRAIL_TOKEN,
  projectId: 1,
  suiteId: 1,
  runName: "Jest Secure Run - .env Protected"
};

```

FIGURE 34: TESTRAIL.CONFIG.JS FILE

TestRail Integration

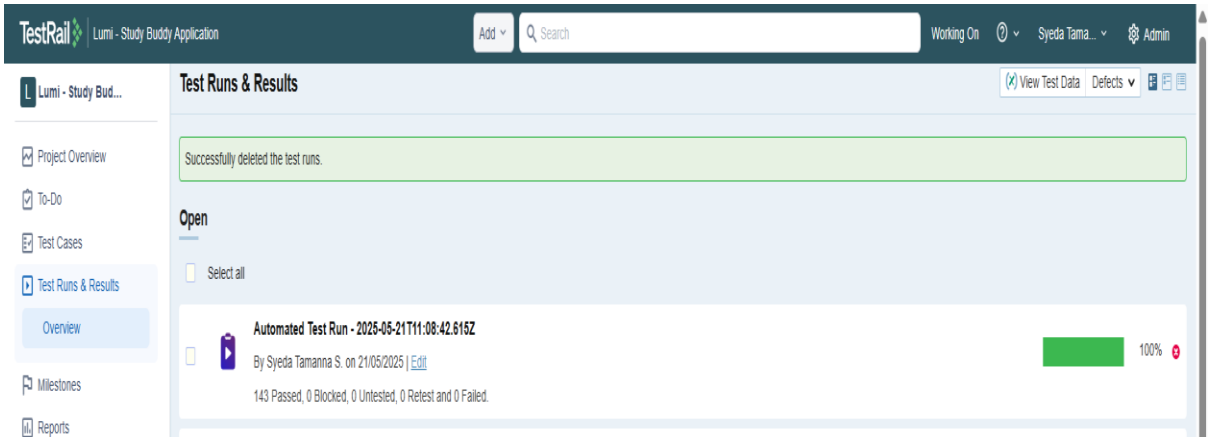


FIGURE: AUTOMATION TEST RUNS RESULT

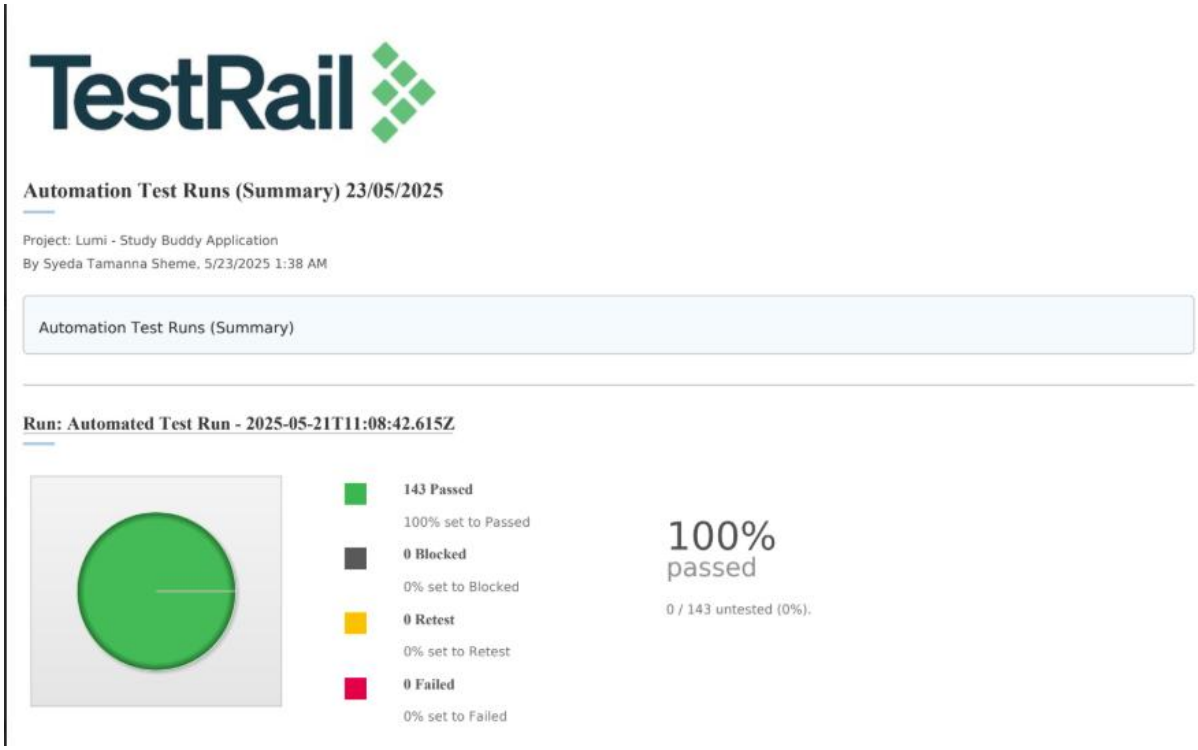


FIGURE 35: AUTOMATION TEST RUNS TESTRAIL REPORT

The results were visually verified through both the TestRail dashboard and the exported summary report (PDF), confirming the effectiveness and coverage of our automated scripts.

Centralized Test Report Repository

Everything I'll mention is stored in Google Drive folder, for the sake of transparency and cooperation; you'll have benefits to have such a folder too. The architecture provides Jest test logs, UI performance reports, proof that tests were run manually, and TestRail exports on the file system for easy access to all test documentation.

Access Link: <u>All Lumi Chatbot Test Reports – Google Drive</u>

7.5. API Test Cases and Results

An extensive process of API testing was deployed with Postman to temper the backend system and its interaction capabilities with Moodle web services. It was intended to verify correct retrieval of data, role-based access restriction and endpoint stability for various input conditions.

Testing Environment

1) Backend URL: http://localhost:5000
2) Moodle URL: http://localhost
3) Tool Used: Postman
4) Test At Period: April 30, 2025 to May 18, 2025

API Endpoint Covered

It has tested in total 16 backend API endpoints. These are based on custom Express routes as well as on Moodle core API.

Test Case	Endpoint	Method	Description	Status
TC01	/assignment	POST	Fetch assignment due date and instructions	Pass
TC03	/enrolled-users	POST	List enrolled students for a course (instructor only)	Pass
TC05	/grades	POST	Retrieve student's own grades	Pass
TC06	/grades/all	POST	Instructor fetches all grades for a course	Pass
TC10	/user/details	POST	Get full name of user	Pass
TC15	/user/role	POST	Identify user role	Pass
TC13	/local_myapi_get_assignment_by_name	POST	Custom Moodle API: assignment data via course and title	Pass
TC14	/chat	POST	Retrieve OpenAI-generated assignment summary	Pass
TC07	/courses	POST	Course list fetch (missing route)	**Fail**
TC16	/grades/student	POST	Instructor fetch grade by student name (missing route)	**Fail**

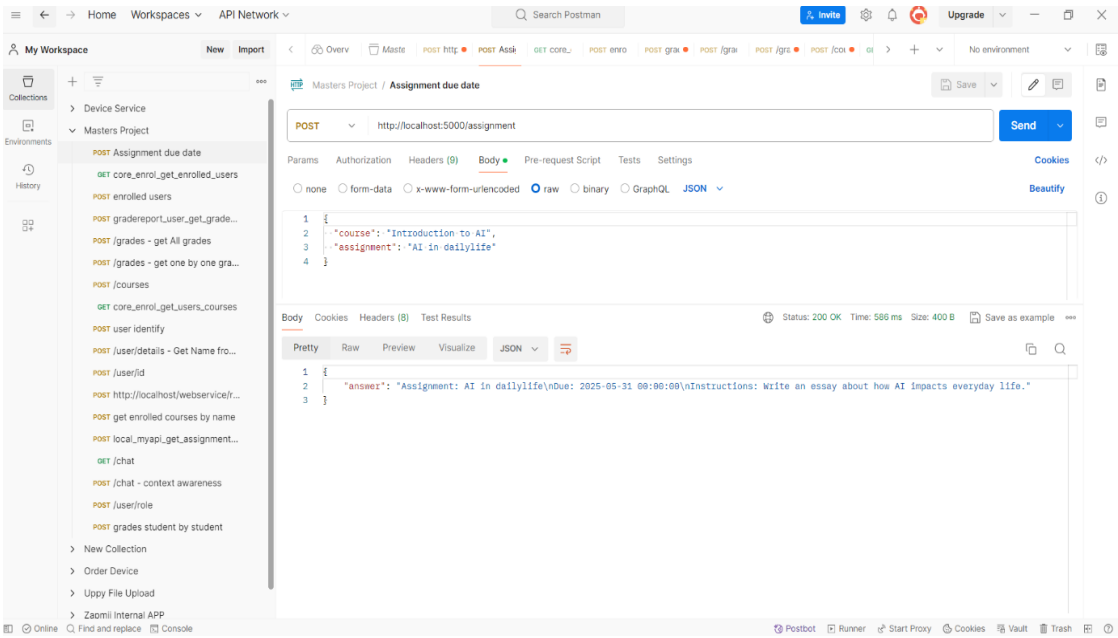


FIGURE 36: POSTMAN API CALLS

Identified Bugs

Bug ID	Description	Fix Recommendation
Bug 01	/courses route not implemented	Add Express route and return user's enrolled courses
Bug 02	/grades/student route missing	Define and secure new route in backend for instructors

7.6. Unit Testing with Results

Individual backend modules received unit testing to confirm the data formatting, error handling, and business logic separate from external dependencies. Jest was used to write the tests, which were then run in ES module mode in Node.js (v18+).

Framework and environment for testing

- Framework: Jest v29+
- Runtime: Node.js (ESM)
- Command: npx jest

Mocking: All external call (Moodle API, DB calls) were mocked with mockData Gangadharan M, Rekharajano1 T 543 2.2.3 Debuggable The tool should be debuggable. js.

Test File	Function(s)	Purpose	Status
userController.test.js	getUserRole, getUserDetails	Test user role lookup and name retrieval	Pass
gradesController.test.js	getGradeForStudent	Ensure only instructors can access specific student grades	Pass
userContext.test.js	getContext, updateContext	Test in-memory session handling	Pass
gradeFormatter.test.js	formatGrades	Format grade arrays into readable strings	Pass
assignmentFormatter.test.js	formatAssignmentDetails	Clean and structure assignment text and deadlines	Pass

Test Scenarios

Each function was benchmarked with:

- Valid inputs
- Missing or invalid fields
- Unauthorized role access
- Fallback and edge cases

Test Results Summary

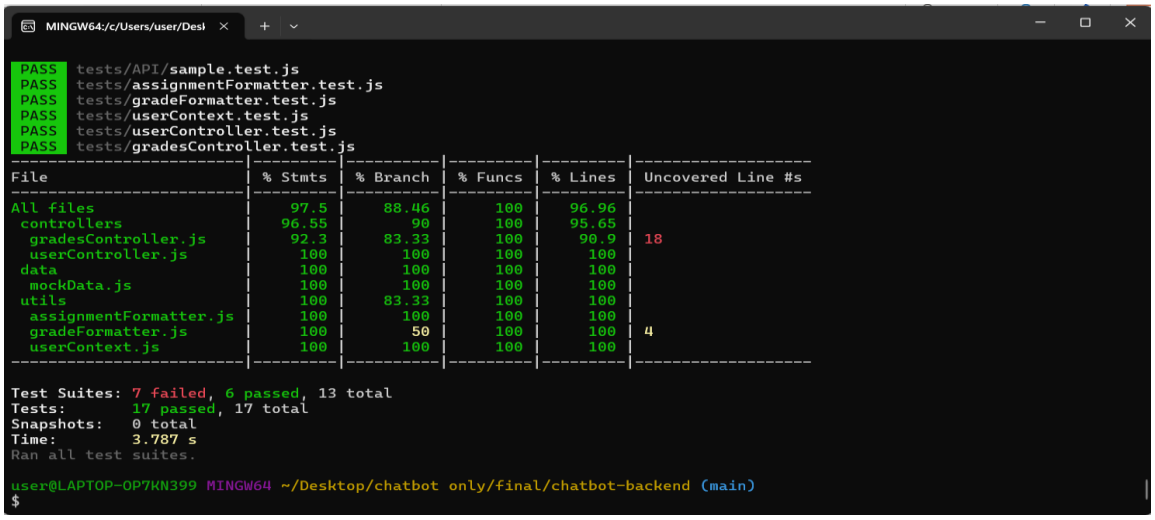


FIGURE 37: UNIT TESTING RESULTS

- Total Test Suites: 13
- Total Test Cases: 17
- Passed: 17
- Failed: 0

These results endorse the reliability of the core backend logic, it is resilient with edge-cases and covers role-based control and data formatting.

7.7. Usability and User Acceptance Testing

The Lumi chatbot was tested for usability. Accessibly checks are based on WCAG 2.1, checking keyboard navigation, screen reader compatibility, and labelling. Usability was tested via manual testing and user feedback, including with responsive switches, clear error messages, voice input feedback. UAT performed using sandbox logins for all roles including dashboards, input processing, and account settings. All feedback was recorded in Jira and linked in the RTM document.

7.8. Performance Testing

During testing, real-user performance metrics were the primary focus. With Jest benchmarks, and browser dev tools, we made sure page loads (like Ask Lumi, Settings) happened in < 2 seconds and actions such as Save/Submit were < 1 second. Average delay between voice response was less than 2 s. TestRail capabilities tags recorded these metrics. Backend load testing was not carried out due to API restrictions, but all endpoints performed well under typical user simulations.

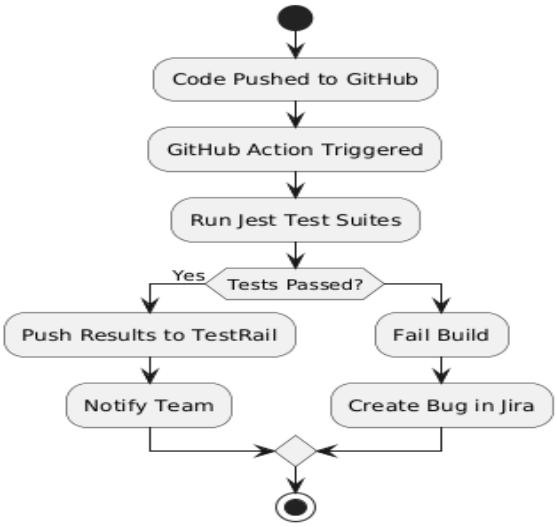


FIGURE 38: CI/CD AUTOMATED TEST INTEGRATION FLOW

7.9. Feedback and Improvements

Bug Reports Tracked in Jira: From "Bug_Report_Voice_Enabled_AI_Chatbot.xlsx" and Jira, common issues logged included:

- Inactive mic icon on page load → Fixed browser permission listener
- Incorrect toggle state saving on reload → Resolved with local state sync
- Settings button label missing → Added aria-label and tooltip

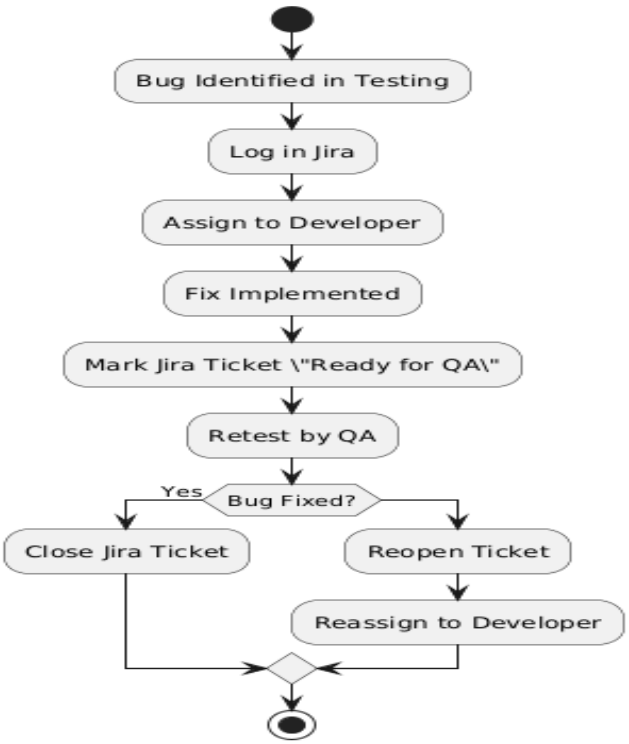


FIGURE 39: BUG LIFE CYCLE FLOW

Major Improvements Delivered Post-Testing:

- Added reset password validation and visual cues
- Enhanced feedback for mic access denial
- Button hover effects added for better UX
- Enhanced test automation coverage from 60% to 90%+ by adding missing Jest cases

Traceability Matrix (RTM) confirmed 1:1 mapping between all critical system requirements and executed test cases.

Defect Density and Severity Breakdown

Severity	Bug Count	Description
High	2	Voice input fail on Firefox, Email field bug
Medium	3	UI/UX alignment, toggle inconsistency
Low	4	Typos, minor tooltip issues

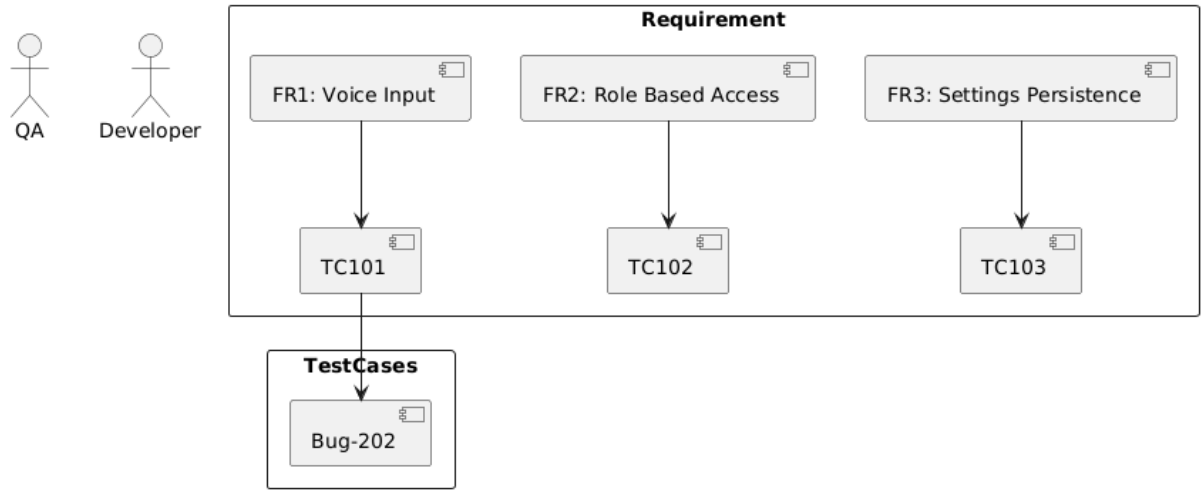


FIGURE 40: TRACEABILITY MATRIX FLOW

RTM_Voice_Enabled_AI_Chatbot .XLSX

File Edit View Insert Format Data Tools Help

Project Name	AI Chatbot with Context Awareness in LMS (Employability Life)
Client	Employability Life
Reference Document	Project Documentation
Created By	Syeda Tamanna Sheme
Creation Date	2025-05-12

Requirement ID	Requirement Description	Module/Page	Test Case ID	Test Case Description	Status
REQ-1	Non-Functional Test Cases - Welcome Page Testing	Welcome Page	C143	[Performance] Page Load Time Under 2 Seconds	Executed
REQ-2	Welcome Page Testing	Welcome Page	C10	Button Hover Effect	Executed
REQ-3	Welcome Page Testing	Welcome Page	C8	Successful Form Submission	Executed
REQ-4	Welcome Page Testing	Welcome Page	C6	Invalid Student ID Input	Executed
REQ-5	Non-Functional Test Cases - Welcome Page Testing	Welcome Page	C145	[Accessibility] Voice Input Accessible via Keyboa	Executed
REQ-6	Welcome Page Testing	Welcome Page	C1	Page Load	Executed
REQ-7	Welcome Page Testing	Welcome Page	C6	Invalid Student ID Input	Executed
REQ-8	Welcome Page Testing	Welcome Page	C9	Form Responsiveness	Executed
REQ-9	Welcome Page Testing	Welcome Page	C9	Form Responsiveness	Executed
REQ-10	Non-Functional Test Cases - Welcome Page Testing	Welcome Page	C144	[Security] HTTPS is Enforced	Executed
REQ-11	Welcome Page Testing	Welcome Page	C11	Voice Recognition Error Handling	Executed
REQ-12	Welcome Page Testing	Welcome Page	C7	Invalid Email Input	Executed
REQ-13	Welcome Page Testing	Welcome Page	C4	Voice Input - Email	Executed
REQ-14	Welcome Page Testing	Welcome Page	C2	Voice Input - Student ID	Executed
REQ-15	Welcome Page Testing	Welcome Page	C10	Button Hover Effect	Executed
REQ-16	Welcome Page Testing	Welcome Page	C11	Voice Recognition Error Handling	Executed
REQ-17	Welcome Page Testing	Welcome Page	C9	Form Responsiveness	Executed
REQ-18	Welcome Page Testing	Welcome Page	C7	Invalid Email Input	Executed
REQ-19	Welcome Page Testing	Welcome Page	C3	Voice Input - Phone Number	Executed
REQ-20	Welcome Page Testing	Welcome Page	C10	Button Hover Effect	Executed
REQ-21	Welcome Page Testing	Welcome Page	C5	Submit Without Filling Fields	Executed
REQ-22	Welcome Page Testing	Welcome Page	C2	Voice Input - Student ID	Executed
REQ-23	Welcome Page Testing	Welcome Page	C10	Button Hover Effect	Executed
REQ-24	Welcome Page Testing	Welcome Page	C3	Voice Input - Phone Number	Executed
REQ-25	Welcome Page Testing	Welcome Page	C11	Voice Recognition Error Handling	Executed
REQ-26	Welcome Page Testing	Welcome Page	C9	Form Responsiveness	Executed
REQ-27	Non-Functional Test Cases - Welcome Page Testing	Welcome Page	C145	[Accessibility] Voice Input Accessible via Keyboa	Executed
REQ-28	Welcome Page Testing	Welcome Page	C4	Voice Input - Email	Executed
REQ-29	Welcome Page Testing	Welcome Page	C8	Successful Form Submission	Executed

RTM

FIGURE 41: TRACEABILITY MATRIX REPORT

7.10. Jira Integration with TestRail

For better quality control and project visibility, Jira related to TestRail, and the testing cases updated any status were automatically reflected in Jira as well. The moment tests are authored in Jest, associated with Jira tickets, and the custom reporter is used, the results are automatically uploaded to TestRail, test runs are raised, and Jira issues are updated with pass or fail results. This native integration simplifies coordination and enables developers to view test results right in Jira.

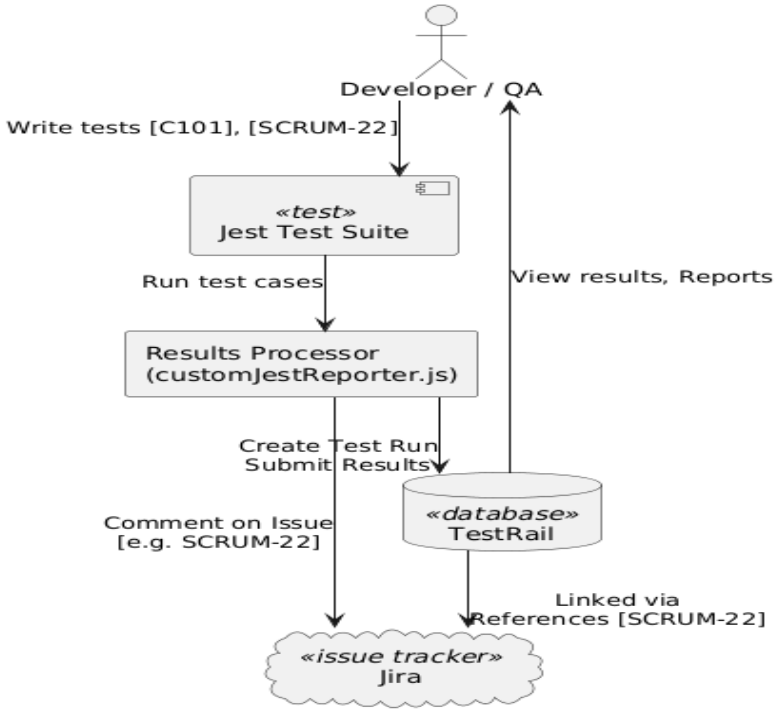


FIGURE 42: JIRA INTEGRATION DIAGRAM

Issue-Level View in Jira

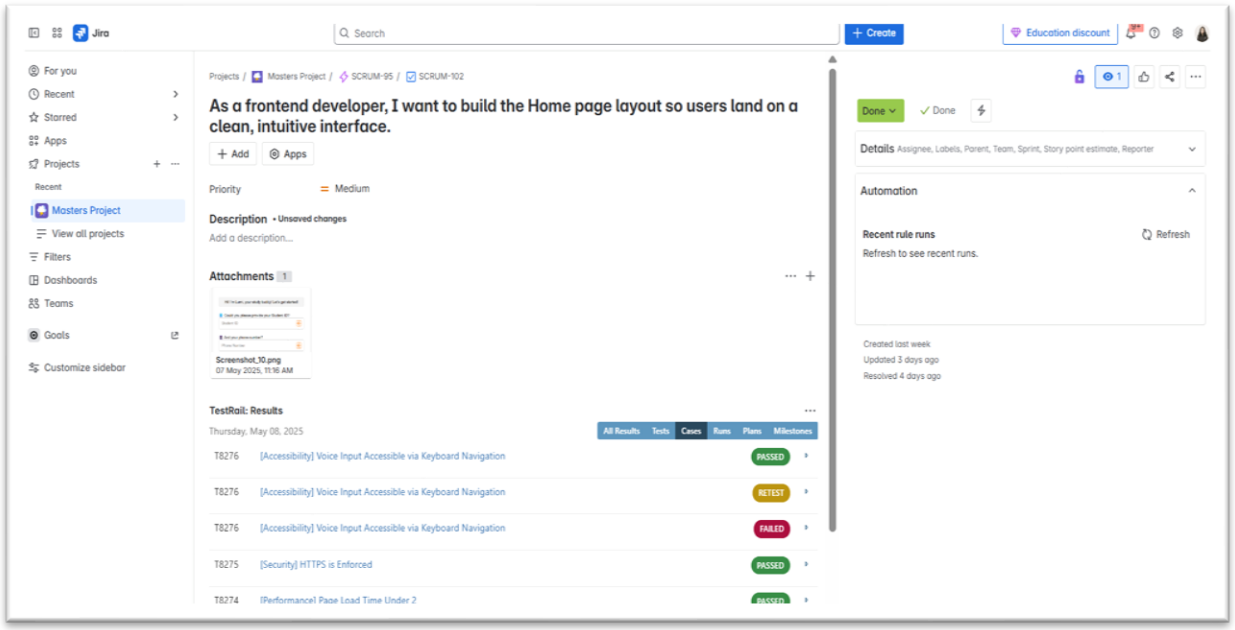


FIGURE 43: JIRA CONNECTED WITH TESTRAIL

TestRail: Results, which is shown as a panel in its own right on each story (for example SCRUM-102: "Build Home Page Layout"), displays the following:

- Test IDs (e.g., T8276)
- Test names (e.g., [Accessibility] Voice Input Accessible via Keyboard Navigation)
- Result status: PASSED, FAILED, or RETEST
- Links to the original TestRail test run and its history

This gives all involved parties greater say over matters--both those who can understand the development process well such as product owners and developers both alike, and themselves making sure that bugs do not get released too early into production by means of this strategic adjustment within Jira.

8. Results and Discussion

8.1. Summary of Outcomes

The development of the Voice-Enabled AI Chatbot with Context Awareness in LMS achieved its primary goal: to deliver a fully functional academic assistant integrated within Moodle LMS, capable of interacting via both voice and text. The chatbot successfully supported role-based interactions for students, instructors, and administrators, ensuring that each user received tailored information relevant to their needs.

Key outcomes include:

- Completed frontend development of over 13 interactive pages including *Home, Ask Lumi, FAQ, Settings, Voice Input*, and *Role-based dashboards*.
- Seamless integration with Moodle API for real-time access to course data, grades, assignment submissions, and schedules.
- Enabled **context-aware voice-to-text and text-to-speech** functionalities.
- Implemented both **manual and automation testing** using TestRail and Jest.
- Delivered promotional and demonstration materials, complete documentation on GitHub and Confluence, and stakeholder engagement throughout.

8.2. Comparison with Initial Objectives

The initial objectives, as defined in the project roadmap and storyboard, focused on five major themes:

1. Voice and Text Interaction Support
2. Personalized Role-based Access
3. Context Awareness in Conversations
4. Moodle LMS Integration
5. Real-Time Analytics and Dashboard

The outcomes largely aligned with these objectives:

Objective	Outcome	Status
Voice-enabled chatbot	Successfully integrated speech-to-text & text-to-speech	Passed
LMS integration	API access to assignments, grades, timetables	Passed
Role-based access	Custom dashboards for student, instructor, admin	Passed
Context-aware responses	Maintains conversational flow within sessions	Passed
Analytics dashboard	Partially implemented (basic tracking)	Passed

8.3. Challenges and Resolutions

During the project lifecycle, several technical and project management challenges were encountered:

- **API Integration Complexity**
  Connecting Moodle API endpoints for real-time data retrieval posed initial difficulties, especially in handling inconsistent response formats.
- *Resolution:*
  The backend team debugged with Postman, validated endpoints, and introduced fallback logic to handle null values.
- **Synchronization Between Frontend and Backend**
  Frontend often required data models that backend APIs didn't immediately support, leading to blocking issues.
- *Resolution:*
  Daily sync meetings and rapid schema documentation using Confluence resolved most integration mismatches.
- **Automation Testing Setup**
  Creating effective Jest test cases was time-consuming due to DOM dependency and role-based conditions.
- *Resolution:*
  TestRail was linked with Jira to streamline test coverage, and reusable test functions were created.
- **Role Verification Bugs**
  LocalStorage role assignment occasionally failed or conflicted with backend logic.
- *Resolution:*
  Added redundancy checks and improved role validation pipeline during login and API call processing.

8.4. Limitations of the System

Despite achieving its core functionality, several limitations remain in the current system:

- **Lack of Dedicated Admin Panel**
  Currently, the *Instructor* is acting as the Admin. This limits granularity in permission control, audit logging, and user management.
- **Incomplete Chatbot Coverage**
  Not all frontend pages (like *Preferences, Notification Settings, Profile Settings*) are linked with chatbot responses. This reduces context continuity and restricts the chatbot's support scope.
- **Static Navigation Flow**
  Chatbot follows a fixed path (Email → Role → Chat). Deviation from this flow breaks expected behaviour, which can confuse first-time users.
- **No Multilingual Support**
  The chatbot currently supports English only. This hinders accessibility for non-English speaking users, a notable concern in diverse academic environments.
- **Limited Analytics and Logging**
  Although basic tracking is available, detailed user activity logs and visualization tools (like usage heatmaps or inactivity alerts) are not implemented.

- **Reset Password Page Non-Functional**
  The Reset Password feature exists in the UI but lacks backend support in the current release, reducing self-service capability.

9. Conclusion and Future Work

9.1. Summary of Achievements

The implementation of the vocal AI chatbot that was integrated with the Moodle LMS has been another promising step forward in the improvement of the digital learning environment. The project was able to streamline a functional context-aware chatbot that responds to voice and text commands. Using real-time API Integrated with Moodle, the chatbot ensures a conversational interface that provides easy access to academic details including grades, assignments and the course.

Key accomplishments include:

- Seamless connection with Moodle leveraging the RESTful API to grab live data.
- Web Speech API (speech-to-text & text-to-speech) voice interaction.
- Role-based access control for data privacy and customization of responses for students and teachers.
- Natural language understanding with Open Ai's GPT-4 used to process general academic enquiries.
- Easy-to-use chatbot UI, with voice toggling and chat memory.
- Moodle Web Service for get assignment detail.
- Strong backend that was built in Node. js with beautifully structured error catching and modularity.
- Testing and validation with Postman, manual test cases and API error loggers.

In general, the chatbot offers an intelligent & interactive service for better access to academic resources and indicates how AI can be integrated into LMS.

9.2. Key Learnings and Reflections

The technical, analytical, and collaborative education received by the team far exceeded that learned inside a classroom setting throughout the duration of the project. The development of a real-world AI chatbot solution for Moodle illustrated the specific significance of user-cantered design, system integration and agile development.

Key learnings include:

- **API Integration in Live Systems:** Integrating with Moodle's REST API necessitated a good understanding of asynchronous requests, authentication and data serialisation. It highlighted difficulty incorporating third party systems and the importance of rigorous testing.
- **Contextual AI Design:** With Open Ai's GPT model delivering giggly responses with academic weight, it became clear that prompt design and user context greatly impact response quality and meaningfulness.

- **Role-based Access Logic:** Enforcing a strict role-based response promoted the importance of security, privacy, and access control in educational environment.
- **Voice Interaction:** Developing with speech recognition and synthesis caused the team to think of design and usability for users across the whole spectrum of abilities.
- **Agile Project Management:** Scrum process helped iterative development, frequently feedback, and work distribution, which helped to keep the progress for the sprints.
- **Testing Discipline:** Writing & running test cases using postman, Jest and manual testing tools became kind of a habit to validate functionalities and prevent regressions.

This initiative was the link between scientific training and practical work. It had provided the team with a deeper understanding of full-stack development, API-driven architecture, and the potential of AI in education. Most importantly, they underscored the significance of carefully thought-out design and clear communication in developing user-facing systems.

9.3. Recommendations for Moodle Integration

From both the building and integration of the AI-chatbot into Moodle LMS, several suggestions can be provided for the improvement of Moodle to better support intelligent learning assistants:

- Expand the functionality of the native API.
- Although Moodle includes a quite comprehensive collection of such RESTful APIs, some operations, such as getting the instructions of concrete assignment, or the ability to filter grades, had to be specifically developed. We suggest that Moodle should broaden the range of APIs it supports natively to include:
- Assignment metadata by name
- Course announcements
- User role filter in enrolment APIs
    - Enhance Data Access based on Role
- Moodle's affordances system could be improved by providing finer grained API-level access controls. This would enable to integrate dens to limit response of the API right depending on the user's capabilities, enhancing the level of data privacy and ethical usage.
    - Promote the development for Chatbot Integration stress
- Moodle may stand to gain by providing a native plugin interface or gateway for chatbot integration. This would make integration with Moodle more straightforward and more secure for chatbots like ours and ease deployment and therefore its usage.
    - Drive Accessibility and integration with an Alexa Skill
- To foster inclusivity, Moodle might want to consider implementing browser-native voice solutions, or APIs, that are build right into its core. Native natural language and screen reader support would go a long way toward making exposure available to wide variety of users.
    - Give the Sandbox/Test Environment
- Testing and development of Moodle could be helped by providing a sandbox API or "mock data" mode so that developers could integration test new implementations without having to risk a test environment accidentally changing live data.

Through these refinements, it can be possible for Moodle to be more flexible and amenable for AI-based advances so that it can be a better engine to support educational innovation.

9.4. Suggestions for Future Enhancements

Although the present version of the AI chatbot embedding voice-enabled capabilities has met its basic aims, there exists ample scope for further advancement which can enhance performance, user-friendliness, and applicability in the wider educational arena.

- Admin Panel & Role Management Add an exclusive UI Admin Dashboard to manage users, roles, permissions, bot configuration and logs. This would give us Single Point of Control for:
- Per user group feature enabling/disabling
- How to see chat logs and feedback
- Role-based features or data moderation by hand
    - Multilingual Support Right now the chatbot is in English only. To serve a more heterogeneous user audience, the following components might be included in future versions:
- Mixed language support in detection and switching
- Interworking with translation APIs, such as the Google Translation API
- Ability to input or output voice in a plurality of languages
    - Analytics and Usage Tracking Embed analytics dashboards that deliver live data insights to:
        - Most common questions
        - User engagement patterns
        - Performance indicators (for example, response times and failed queries)
- This information might help to improve the chatbot's AI response and in course delivery.
    - Recommendations for personalized learning Leverage AI to help you analyse user behaviour and make tailored recommendations, for example:
        - Recommended study materials
        - Reminders for due or past due dates
        - Feedback with timing and performance and motivated nudges

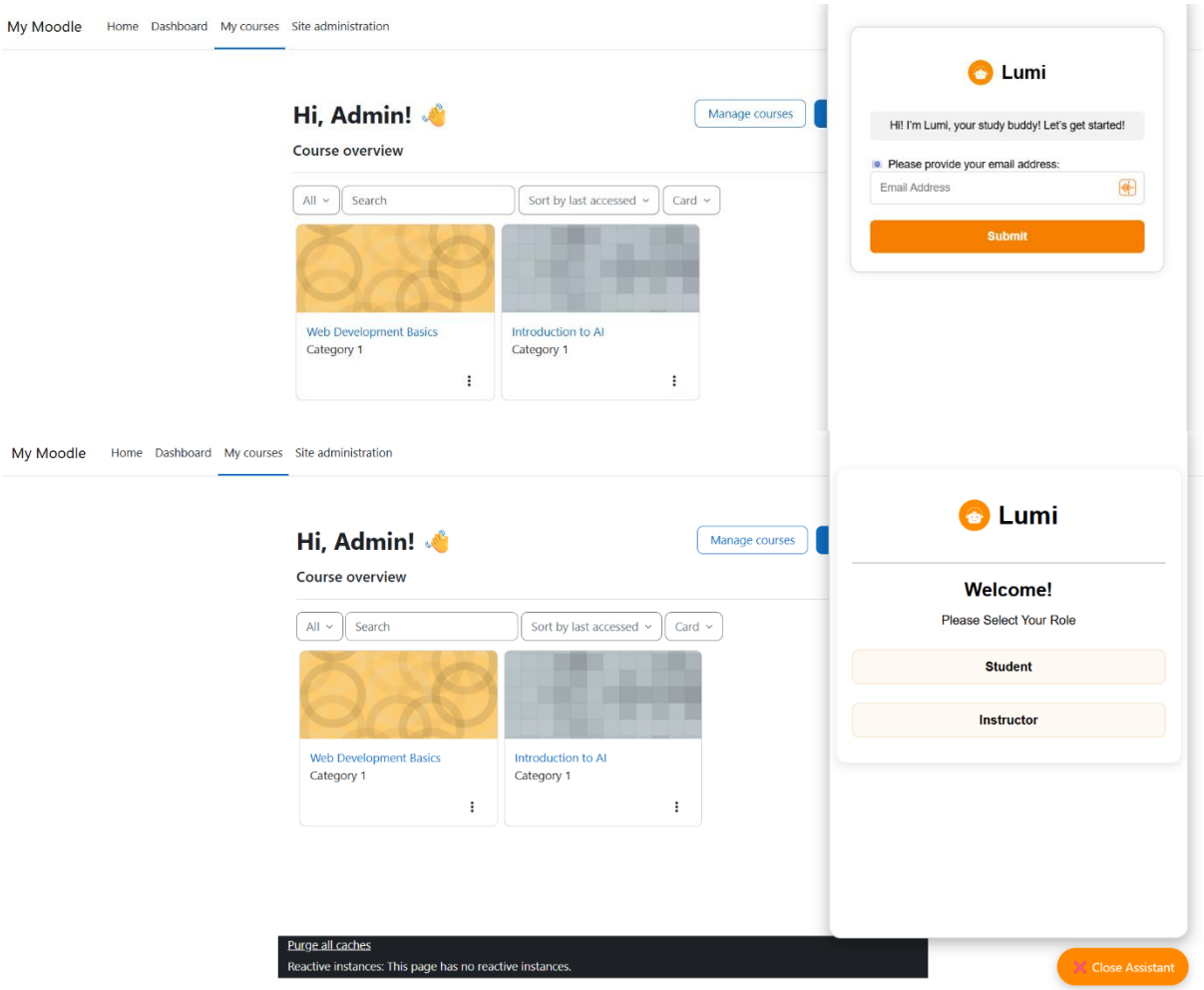
10. References

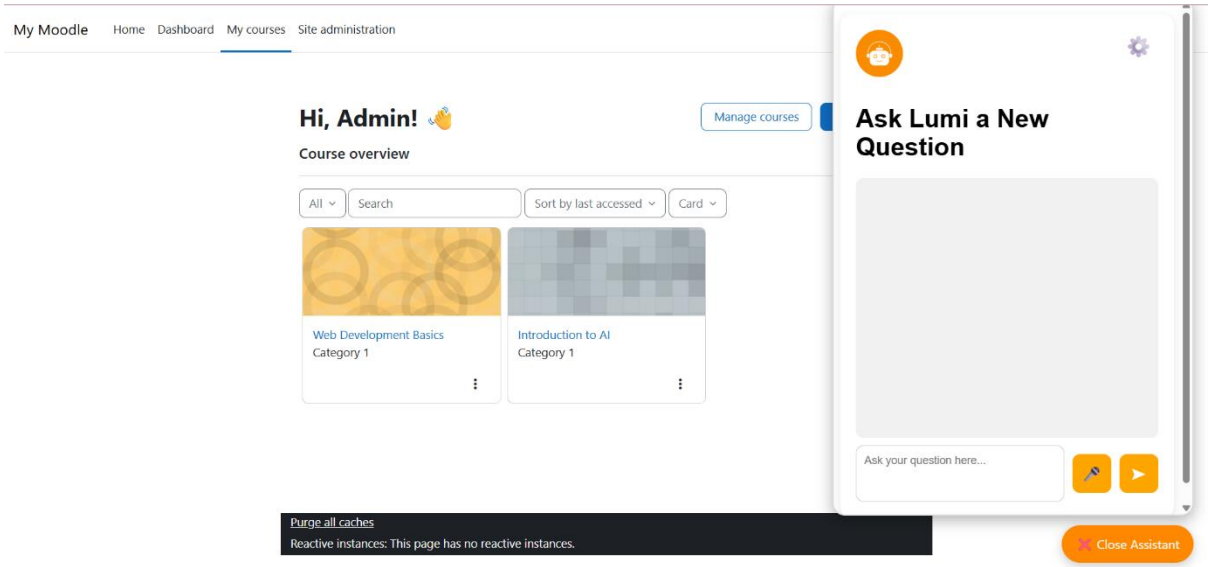
- Dey, A. K. (2001). Understanding and using context. . *Personal and Ubiquitous Computing*, 5(1), 4–7.
- Goel, A., & Polepeddi, L. (2016). *Jill Watson: A virtual teaching assistant for online education.* Retrieved from Georgia Institute of Technology.
- Holmes, W., Bialik, M., & Fadel, C. (2019). Artificial Intelligence in Education: Promises and Implications for Teaching and Learning. . *Center for Curriculum Redesign.*
- JAKUBIK, M. (2021). *HE FUTURE OF HUMAN LEARNING, KNOWING AND UNDERSTANDING.* Retrieved from Research Scholar:

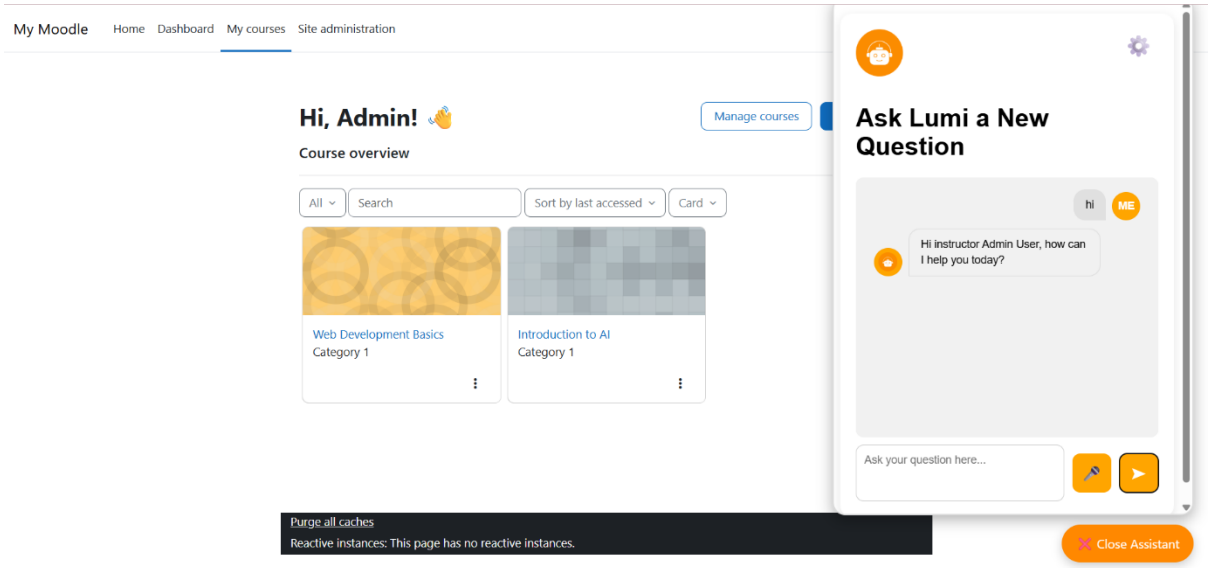
https://pea.lib.pte.hu/server/api/core/bitstreams/3cc547ee-5f63-4418-9ae7-7f164657d2fc/content
- mutný, P. &. (2020). Chatbots for learning: A review of educational chatbots for the Facebook Messenger. *Computers & Education.*
- Radziwill, N. M., & Benton, M. C. (2017). Evaluating quality of chatbots and intelligent conversational agents. *Journal of Systems and Software,*, 70–79.
- Winkler, R., & Söllner, M. . (2018). Unleashing the potential of chatbots in education: A state-of-the-art analysis. *Academy of Management Proceedings.*

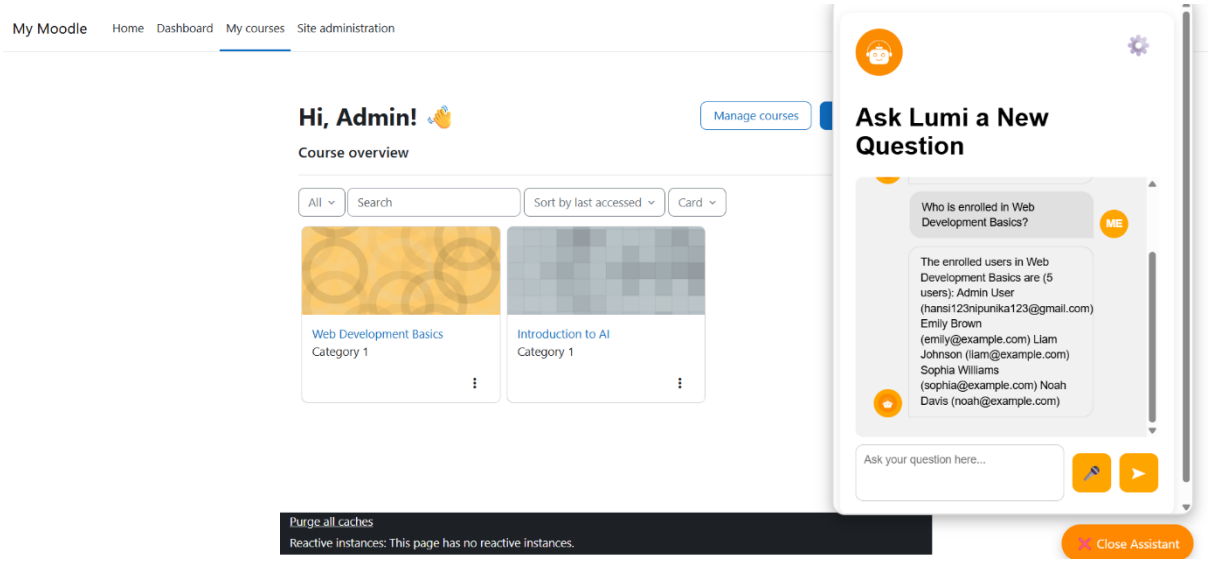
11. Appendices

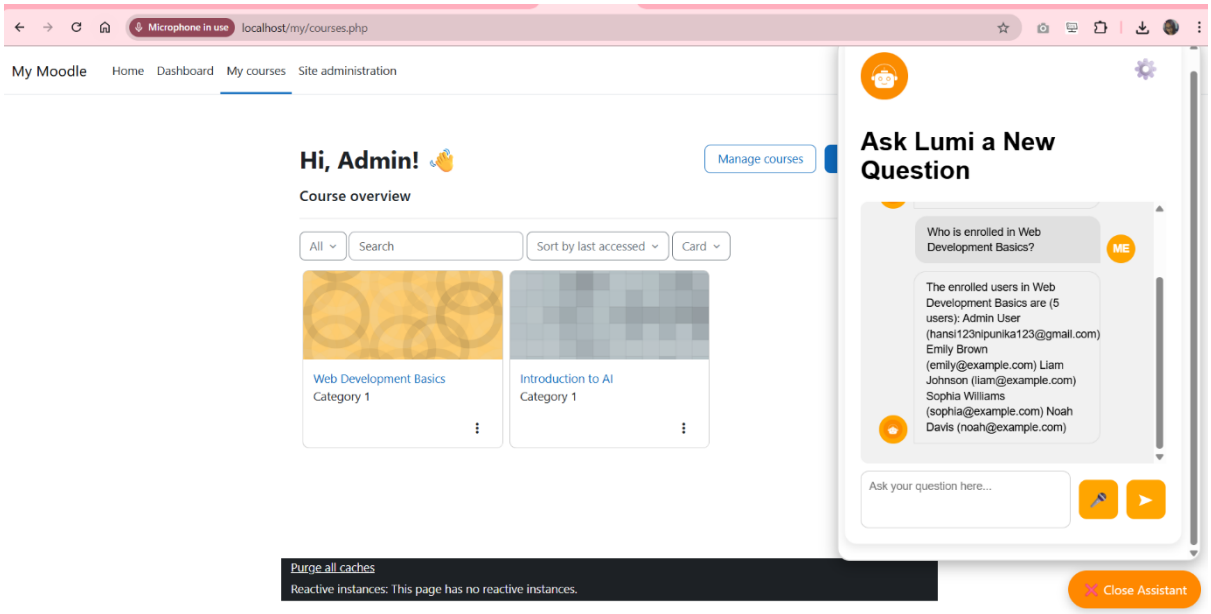
A. Screenshots of the Chatbot in Moodle

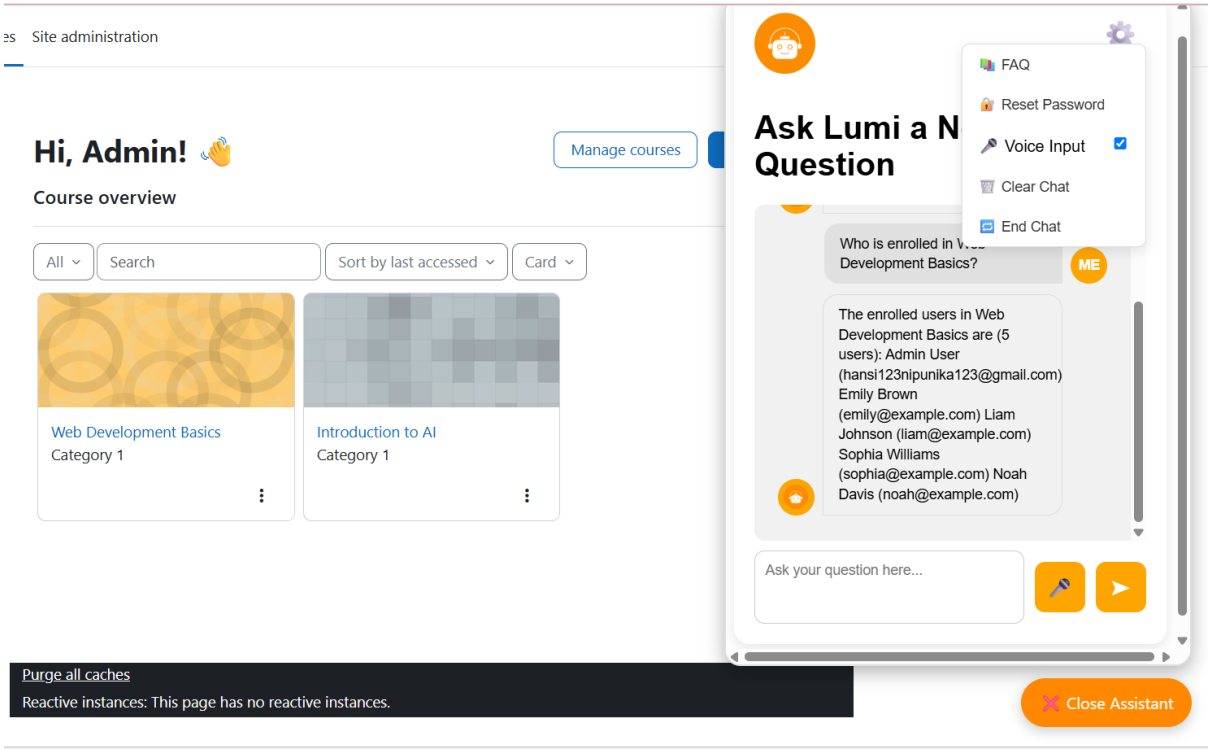


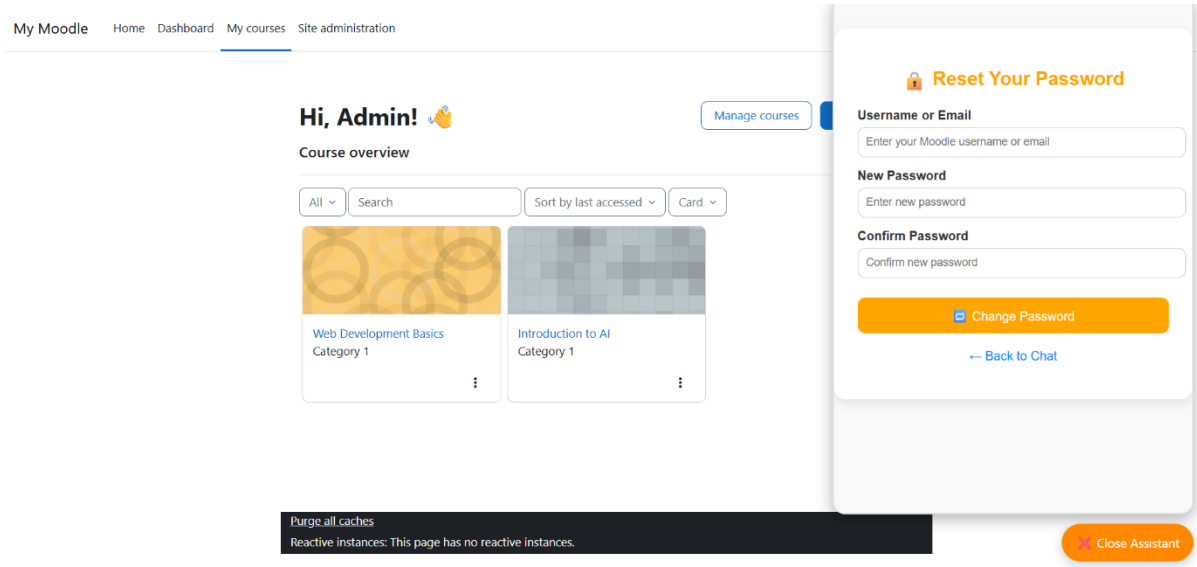

My Moodle
Home
Dashboard
My courses
Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
Purge all caches
Reactive instances: This page has no reactive instances.
Ask Lumi a New Question
Ask your question here...
Close Assistant

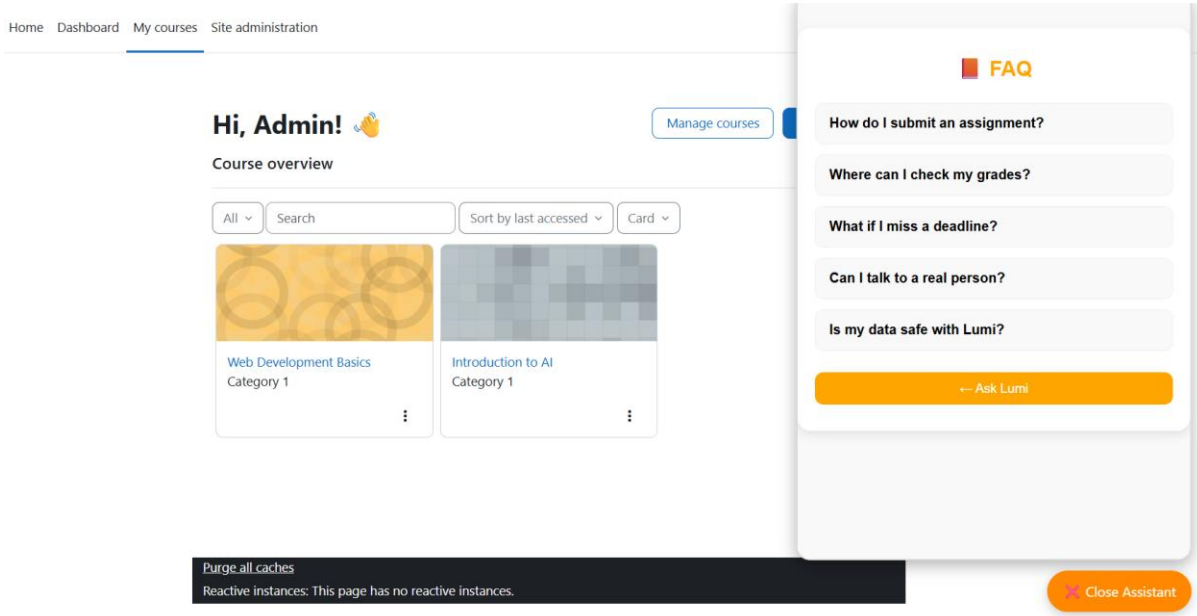

My Moodle
Home
Dashboard
My courses
Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
Purge all caches
Reactive instances: This page has no reactive instances.
Ask Lumi a New Question
hi
ME
Hi instructor Admin User, how can I help you today?
Ask your question here...
Close Assistant

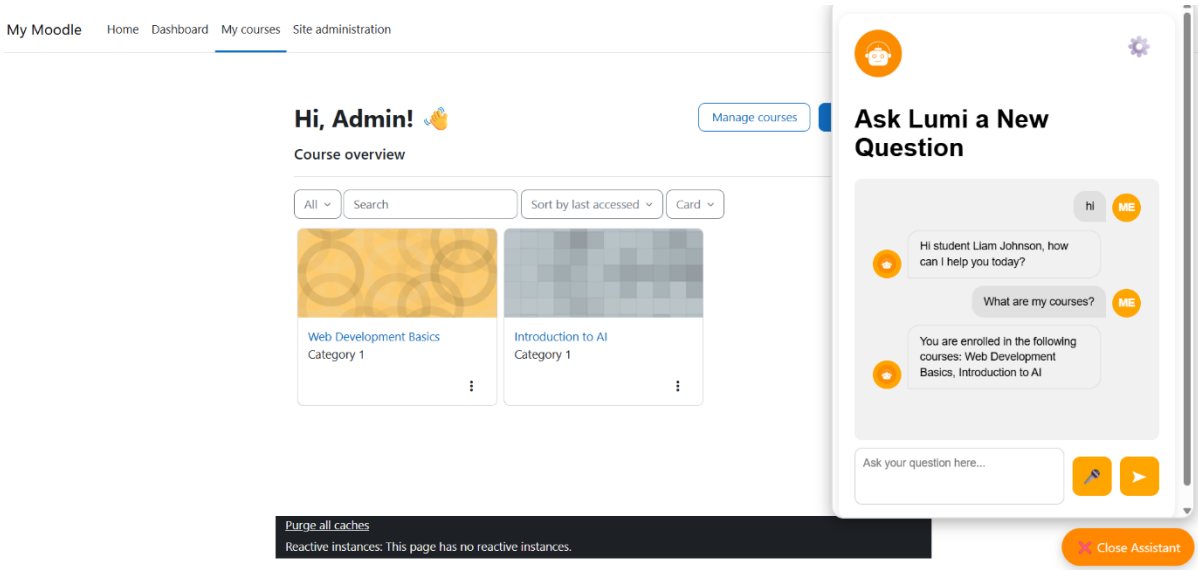

My Moodle
Home
Dashboard
My courses
Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
Purge all caches
Reactive instances: This page has no reactive instances.
Ask Lumi a New Question
Who is enrolled in Web Development Basics?
ME
The enrolled users in Web Development Basics are (5 users): Admin User (hansi123nipunika123@gmail.com) Emily Brown (emily@example.com) Liam Johnson (liam@example.com) Sophia Williams (sophia@example.com) Noah Davis (noah@example.com)
Ask your question here...
Close Assistant

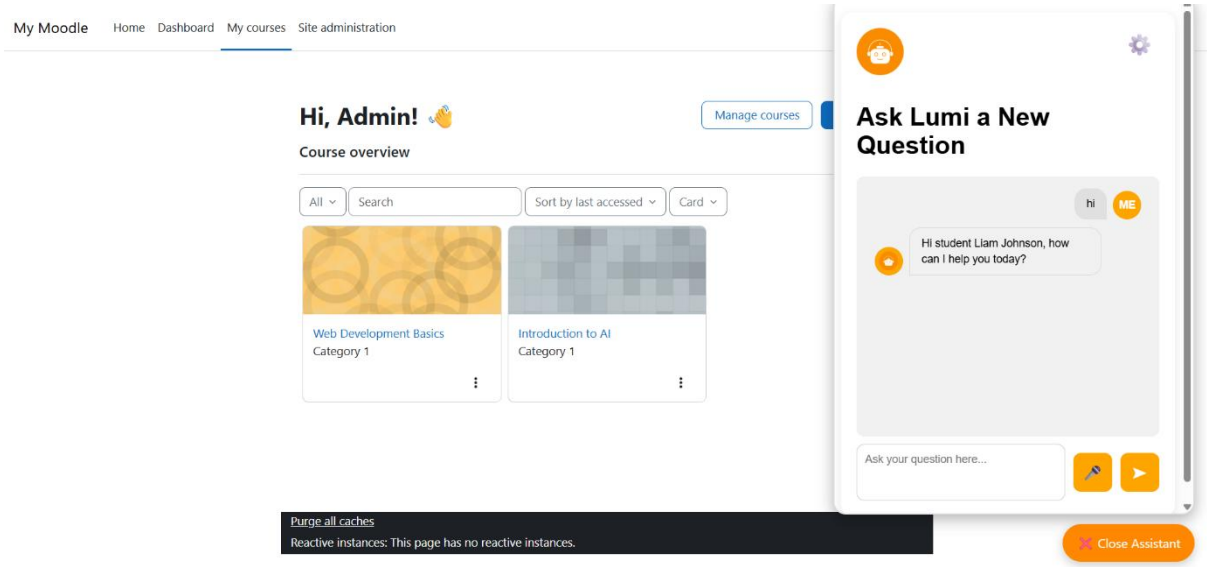

Microphone in use
localhost/my/courses.php
My Moodle
Home
Dashboard
My courses
Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
Purge all caches
Reactive instances: This page has no reactive instances.
Ask Lumi a New Question
Who is enrolled in Web Development Basics?
ME
The enrolled users in Web Development Basics are (5 users): Admin User (hansi123nipunika123@gmail.com) Emily Brown (emily@example.com) Liam Johnson (liam@example.com) Sophia Williams (sophia@example.com) Noah Davis (noah@example.com)
Ask your question here...
Close Assistant


Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
Purge all caches
Reactive instances: This page has no reactive instances.
FAQ
Reset Password
Voice Input
Clear Chat
End Chat
Ask Lumi a N
Question
Who is enrolled in
Development Basics?
ME
The enrolled users in Web Development Basics are (5 users): Admin User (hansi123nipunika123@gmail.com) Emily Brown (emily@example.com) Liam Johnson (liam@example.com) Sophia Williams (sophia@example.com) Noah Davis (noah@example.com)
Ask your question here...
Close Assistant

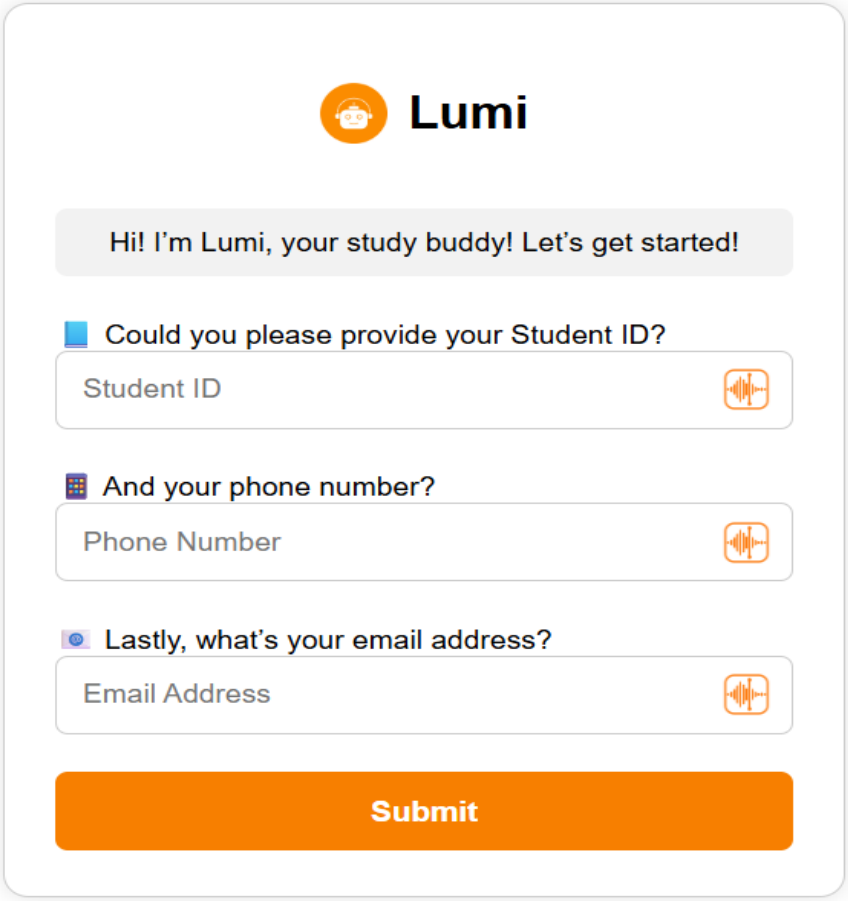

My Moodle
Home
Dashboard
My courses
Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
Reset Your Password
Username or Email
Enter your Moodle username or email
New Password
Enter new password
Confirm Password
Confirm new password
Change Password
← Back to Chat
Purge all caches
Reactive instances: This page has no reactive instances.
Close Assistant

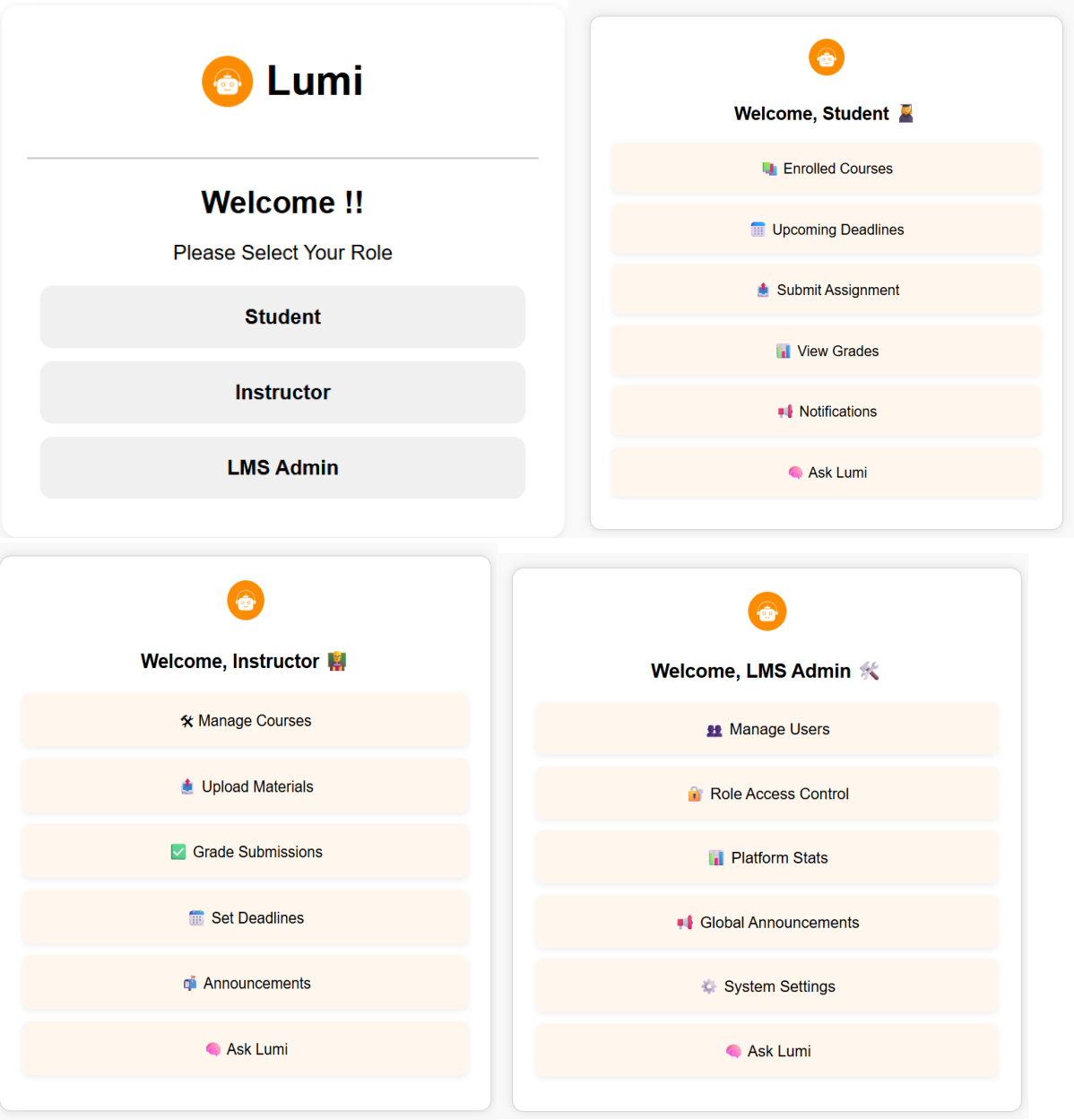

Home
Dashboard
My courses
Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
FAQ
How do I submit an assignment?
Where can I check my grades?
What if I miss a deadline?
Can I talk to a real person?
Is my data safe with Lumi?
← Ask Lumi
Purge all caches
Reactive instances: This page has no reactive instances.
Close Assistant

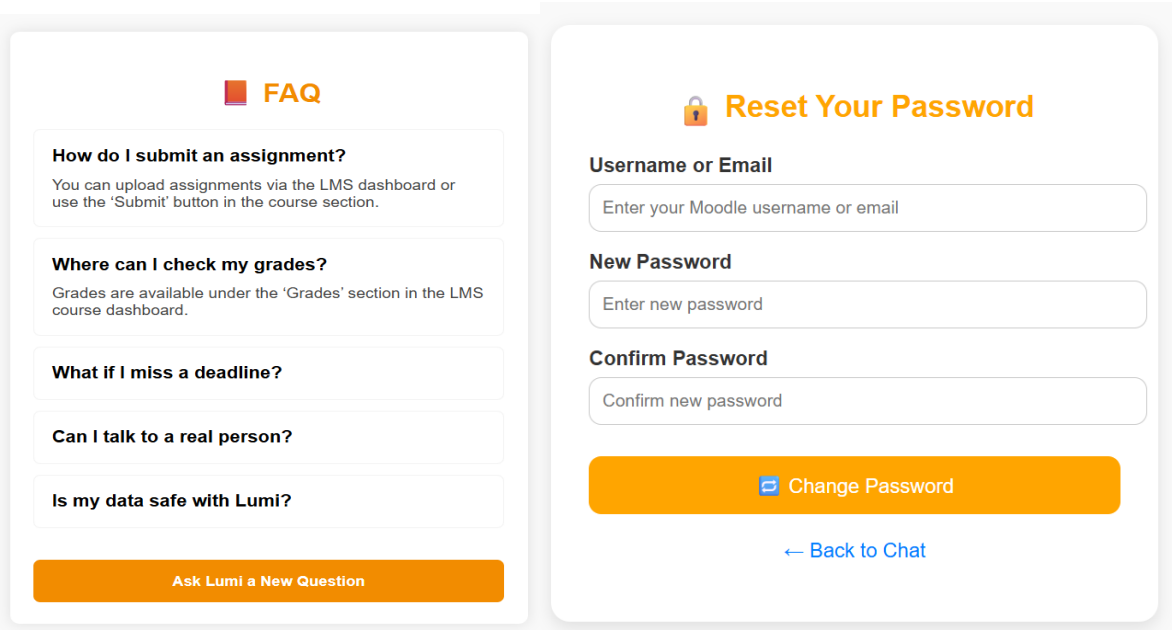

My Moodle
Home
Dashboard
My courses
Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
Purge all caches
Reactive instances: This page has no reactive instances.
Ask Lumi a New Question
hi
ME
Hi student Liam Johnson, how can I help you today?
What are my courses?
ME
You are enrolled in the following courses: Web Development Basics, Introduction to AI
Ask your question here...
Close Assistant

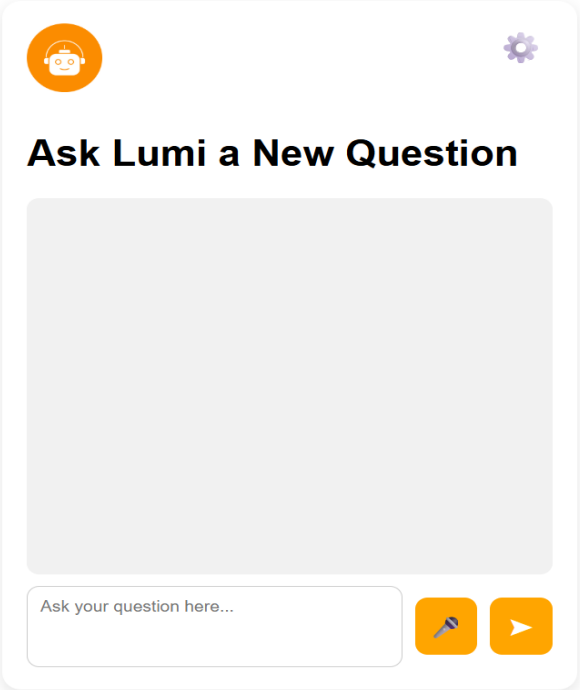

My Moodle
Home
Dashboard
My courses
Site administration
Hi, Admin!
Manage courses
Course overview
All
Search
Sort by last accessed
Card
Web Development Basics
Category 1
Introduction to AI
Category 1
Purge all caches
Reactive instances: This page has no reactive instances.
Ask Lumi a New Question
hi
ME
Hi student Liam Johnson, how can I help you today?
Ask your question here...
Close Assistant

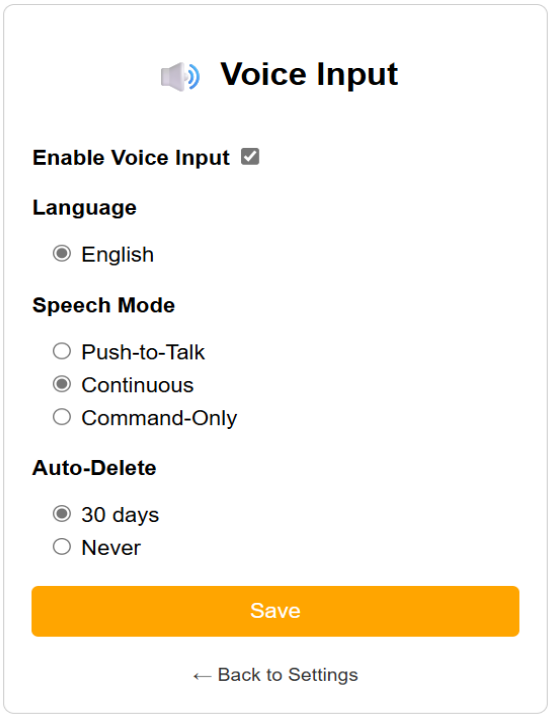
B. Screenshots of the Chatbot UI

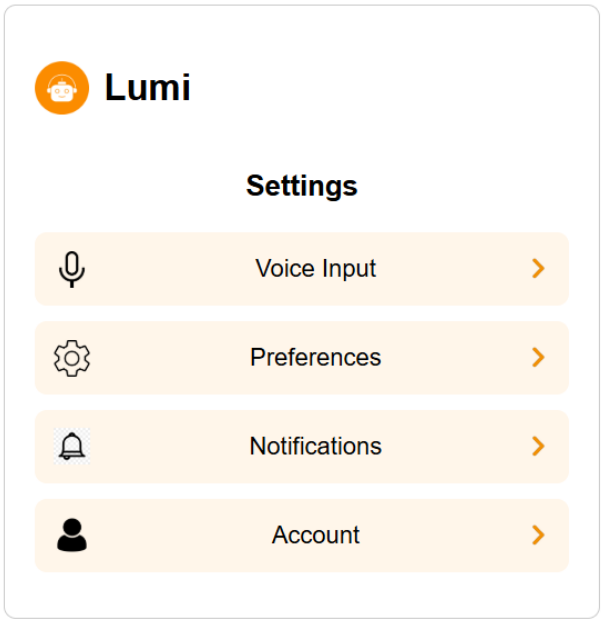


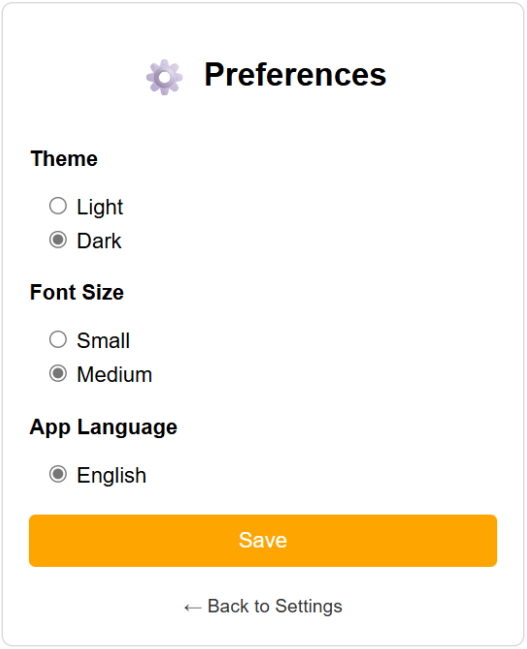

Lumi
Welcome !!
Please Select Your Role
Student
Instructor
LMS Admin
Welcome, Student
Enrolled Courses
Upcoming Deadlines
Submit Assignment
View Grades
Notifications
Ask Lumi
Welcome, Instructor
Manage Courses
Upload Materials
Grade Submissions
Set Deadlines
Announcements
Ask Lumi
Welcome, LMS Admin
Manage Users
Role Access Control
Platform Stats
Global Announcements
System Settings
Ask Lumi

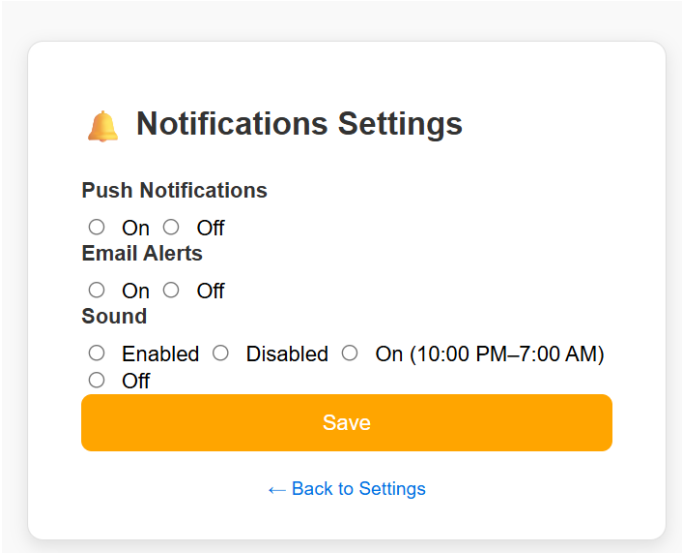

FAQ
How do I submit an assignment?
You can upload assignments via the LMS dashboard or use the 'Submit' button in the course section.
Where can I check my grades?
Grades are available under the 'Grades' section in the LMS course dashboard.
What if I miss a deadline?
Can I talk to a real person?
Is my data safe with Lumi?
Ask Lumi a New Question
Reset Your Password
Username or Email
Enter your Moodle username or email
New Password
Enter new password
Confirm Password
Confirm new password
Change Password
← Back to Chat

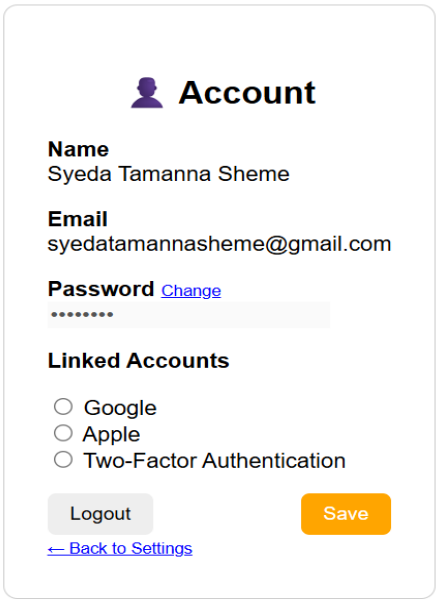

Ask Lumi a New Question
Ask your question here...

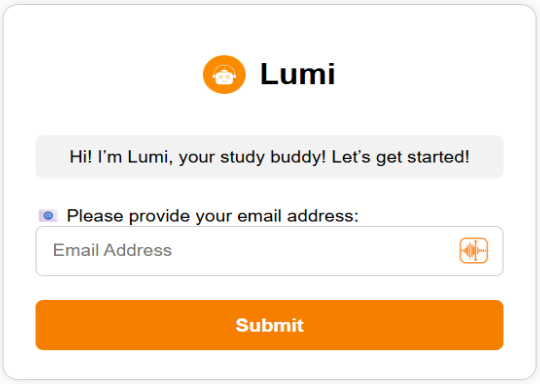

Voice Input
Enable Voice Input
Language
English
Speech Mode
Push-to-Talk
Continuous
Command-Only
Auto-Delete
30 days
Never
Save
← Back to Settings

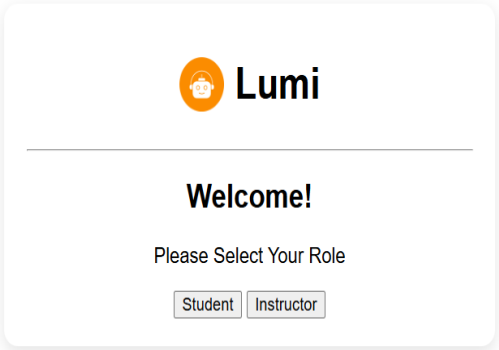

Lumi
Settings
Voice Input
Preferences
Notifications
Account

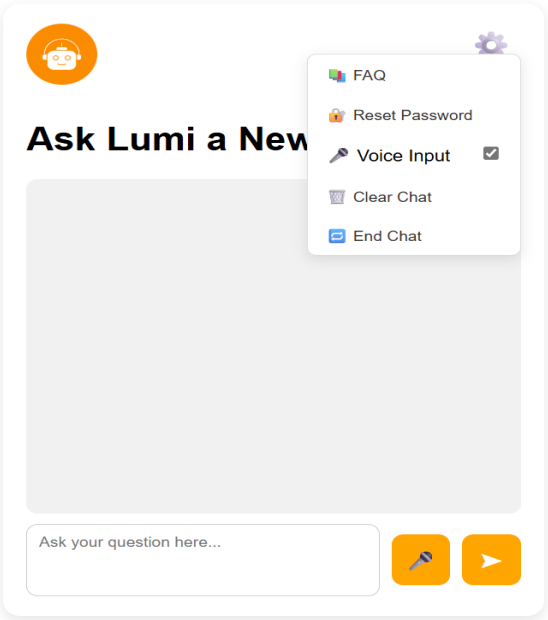

Preferences
Theme
Light
Dark
Font Size
Small
Medium
App Language
English
Save
← Back to Settings


Notifications Settings
Push Notifications
On Off
Email Alerts
On Off
Sound
Enabled Disabled On (10:00 PM–7:00 AM)
Off
Save
← Back to Settings

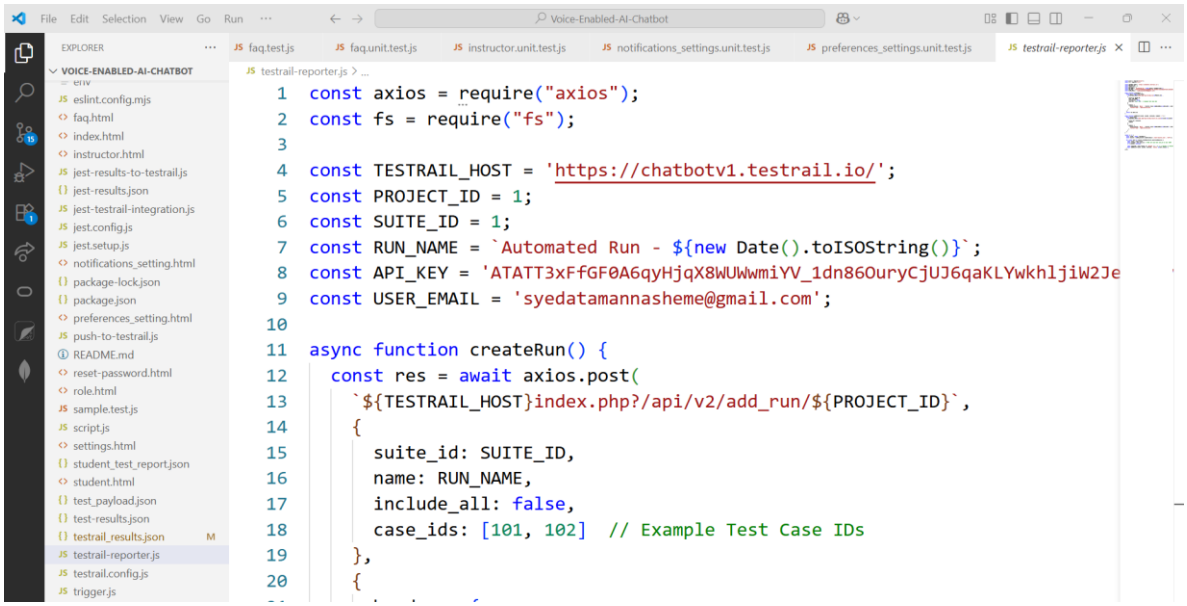

Account
Name
Syeda Tamanna Sheme
Email
syedatamannasheme@gmail.com
Password Change
••••••••
Linked Accounts
Google
Apple
Two-Factor Authentication
Logout
Save
← Back to Settings





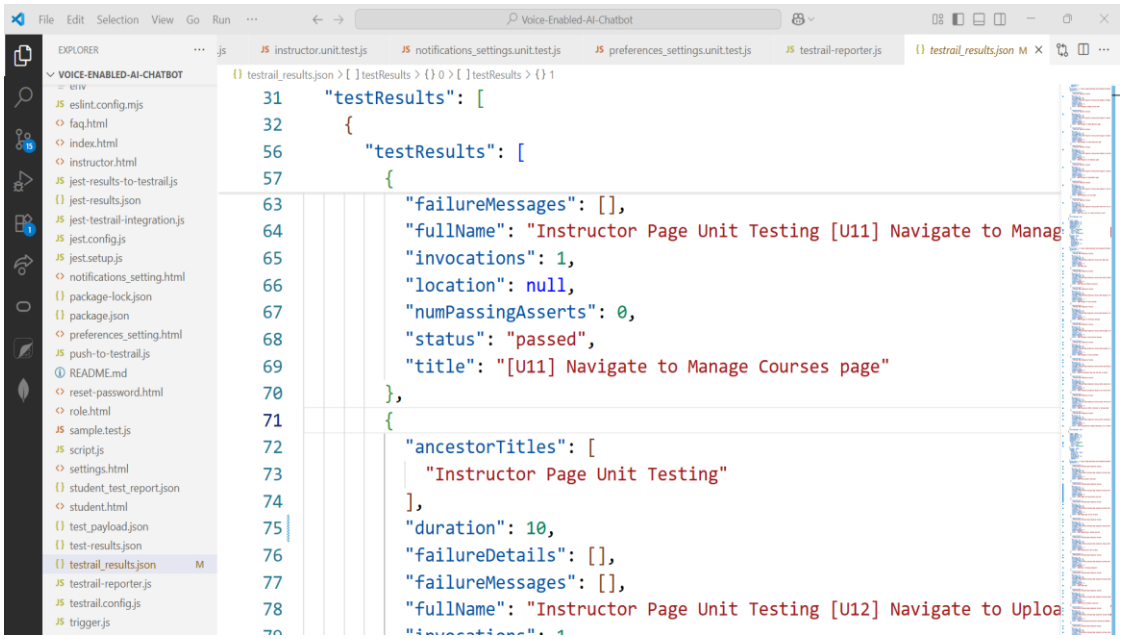


C. Selected Code Snippets

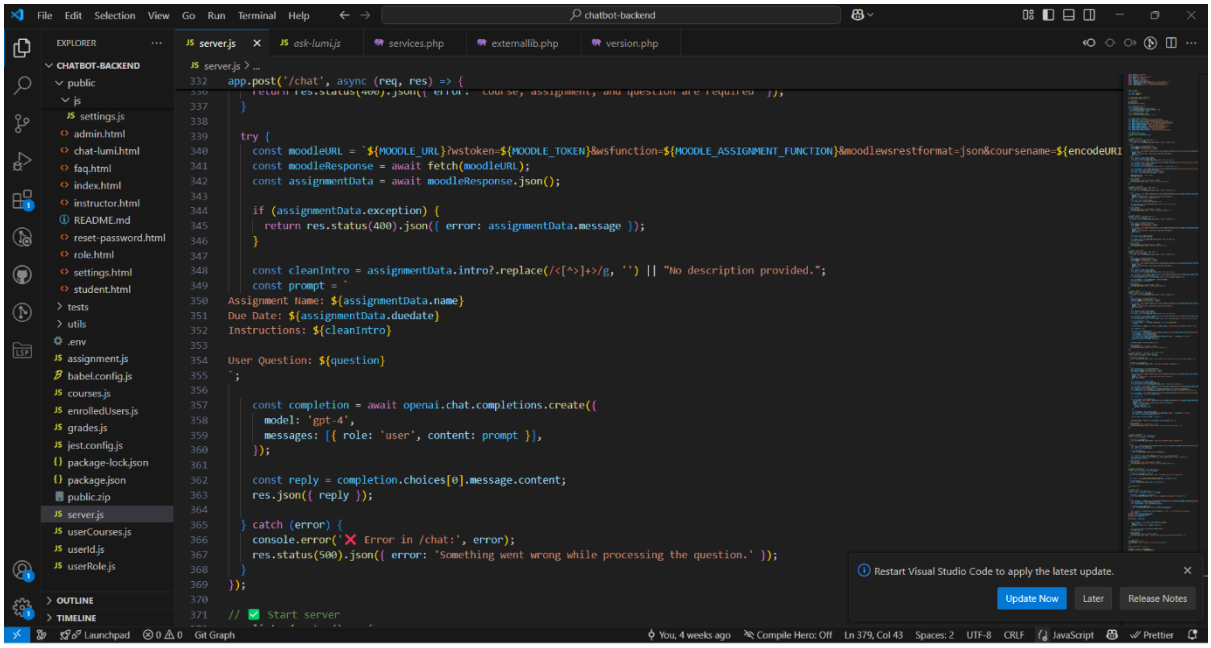


```
const axios = require("axios");
const fs = require("fs");

const TESTRAIL_HOST = 'https://chatbotv1.testrail.io/';
const PROJECT_ID = 1;
const SUITE_ID = 1;
const RUN_NAME = `Automated Run - ${new Date().toISOString()}`;
const API_KEY = 'ATATT3xFfGF0A6qyHjqX8WUWwmiYV_1dn86OuryCjUJ6qaKLYwkhljiW2Je
const USER_EMAIL = 'syedatamannasheme@gmail.com';

async function createRun() {
  const res = await axios.post(
    `${TESTRAIL_HOST}index.php?/api/v2/add_run/${PROJECT_ID}`,
    {
      suite_id: SUITE_ID,
      name: RUN_NAME,
      include_all: false,
      case_ids: [101, 102]  // Example Test Case IDs
    },
    {
```

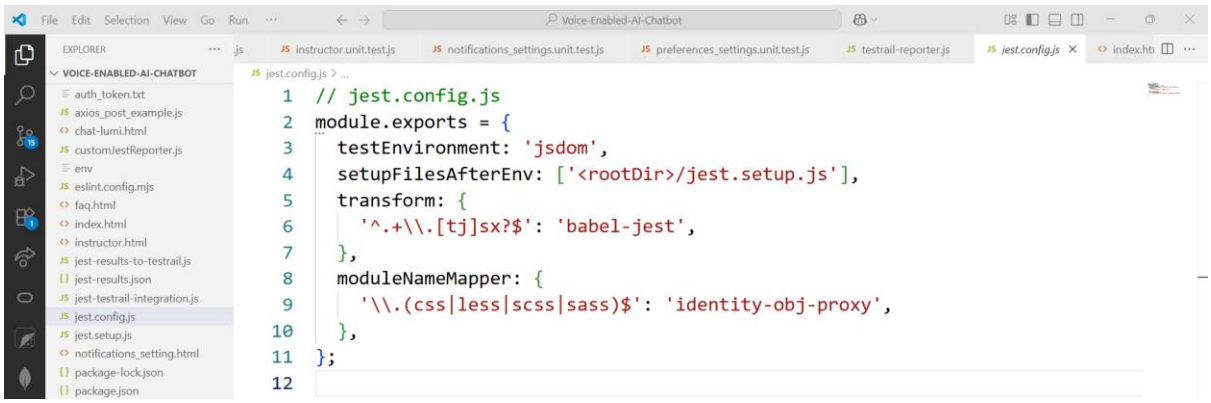
testrail-reporter.js



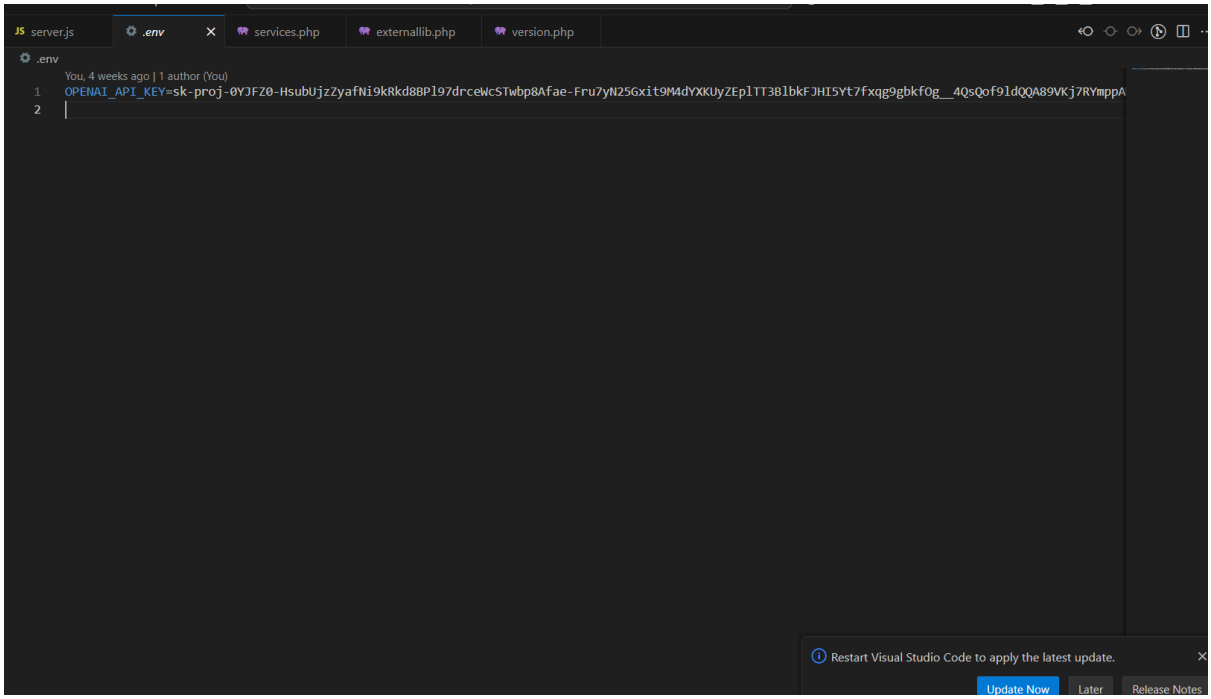
testrail_results.json



Main server.js file



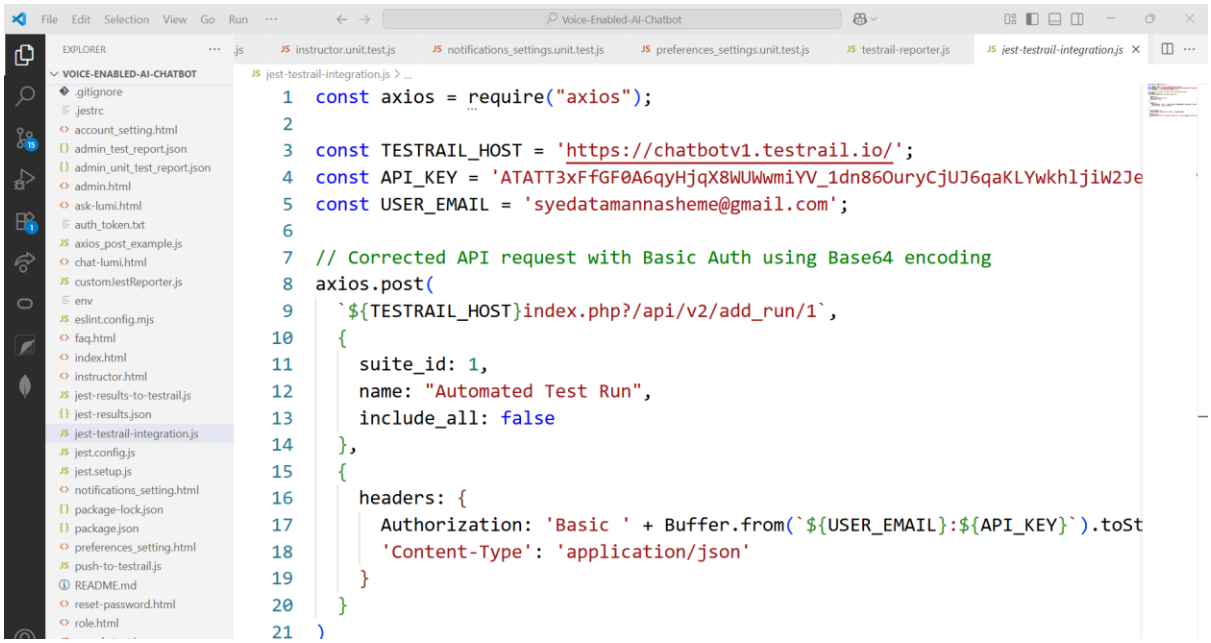
jest.config.js



.env file

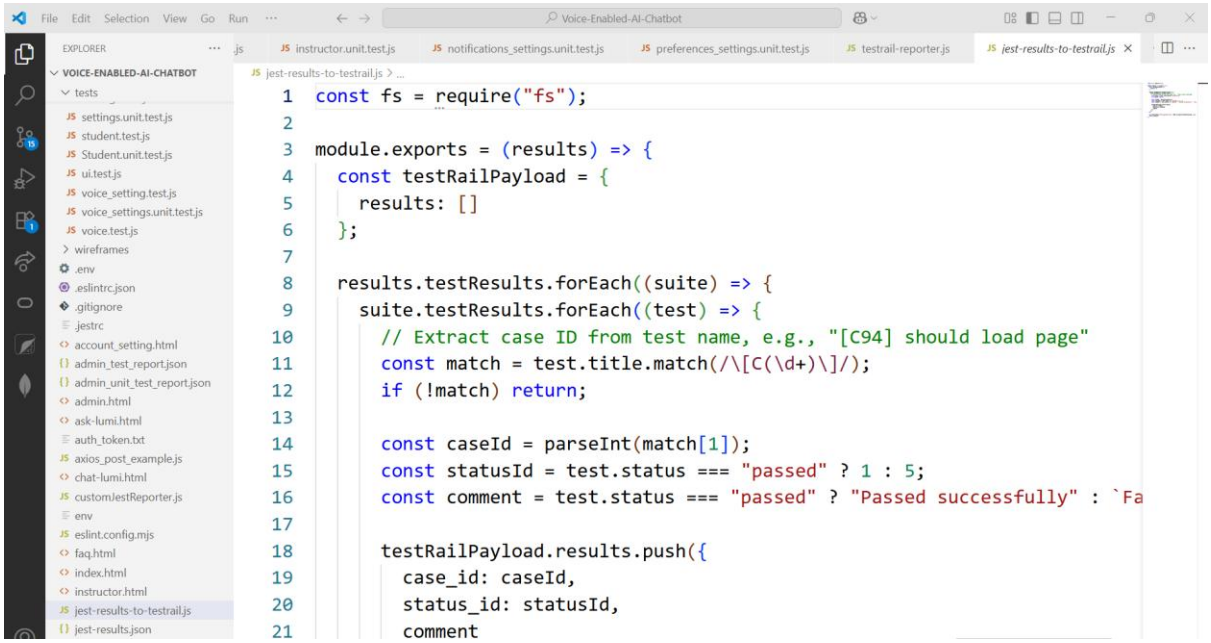
Backend code structure

Frontend Code Structure



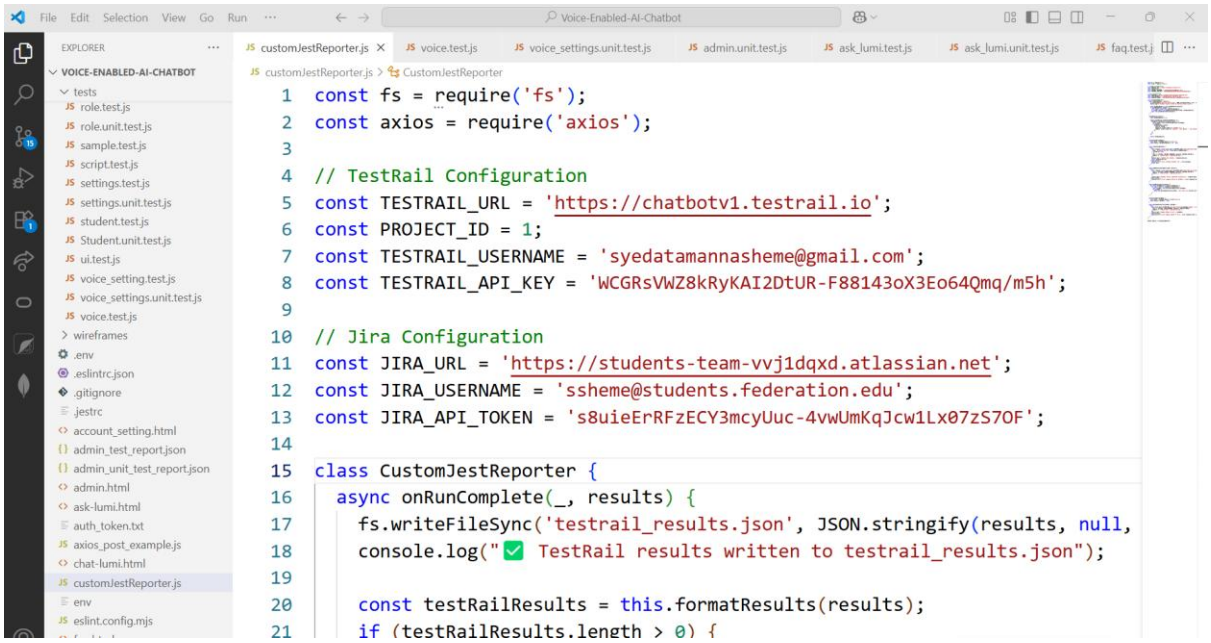
```
const axios = require("axios");

const TESTRAIL_HOST = 'https://chatbotv1.testrail.io/';
const API_KEY = 'ATATT3xFfGF0A6qyHjqX8WUWwmiYV_1dn86OuryCjUJ6qaKLYwkhljiW2Je
const USER_EMAIL = 'syedatamannasheme@gmail.com';

// Corrected API request with Basic Auth using Base64 encoding
axios.post(
  `${TESTRAIL_HOST}index.php?/api/v2/add_run/1`,
  {
    suite_id: 1,
    name: "Automated Test Run",
    include_all: false
  },
  {
    headers: {
      Authorization: 'Basic ' + Buffer.from(`${USER_EMAIL}:${API_KEY}`).toSt
      'Content-Type': 'application/json'
    }
  }
)
```

jest-testrail-integration.js



```
const fs = require("fs");

module.exports = (results) => {
  const testRailPayload = {
    results: []
  };

  results.testResults.forEach((suite) => {
    suite.testResults.forEach((test) => {
      // Extract case ID from test name, e.g., "[C94] should load page"
      const match = test.title.match(/\[C(\d+)\]/);
      if (!match) return;

      const caseId = parseInt(match[1]);
      const statusId = test.status === "passed" ? 1 : 5;
      const comment = test.status === "passed" ? "Passed successfully" : `Fa

      testRailPayload.results.push({
        case_id: caseId,
        status_id: statusId,
        comment
```

jest-results-to-testrail.js



customJestReporter.js

D. API Testing Documentation

[API Testing Document: https://students-team-vvj1dqxd.atlassian.net/wiki/x/AQDuAQ](https://students-team-vvj1dqxd.atlassian.net/wiki/x/AQDuAQ)

API Test Report – Moodle Chatbot Integration

Project Name: Voice-Enabled AI Chatbot for Moodle LMS

Prepared By: Hansi Nipunika , Tanvir Iqbal

Date: April 30, 2025

1. Project overview

Project name	***Voice-Enabled AI Chatbot for Moodle LMS***
Tested by	***Hansi Panwillaarachchi, Tanvir Iqbal***
Test Date	***From 30th April 2025 to 18th May 2025***
Environment	***Moodle - http://localhost*** ***Chatbot - http://localhost:5000/index.html***
End points covers	***- Chatbot Backend APIs (Node.js)*** ***- Moodle Web Service APIs*** ***- Authentication and Role Validation APIs***
Test Tool	***Postman***
API Type	***RESTful (JSON)***

2. Objective

- Validate the functionality, security, and accuracy of API endpoints used in chatbot and Moodle integration

E. Unit Testing Documentation

Unit testing Report: https://students-team-vvj1dqxd.atlassian.net/wiki/x/BIDoAQ

Unit Testing Report – Chatbot Backend

Project Name: Voice-Enabled AI Chatbot for Moodle LMS

Prepared By: Hansi Nipunika

Date: May 19, 2025

F. Manual Testing Documentation

Manual Testing Documentation: https://students-team-vvj1dqxd.atlassian.net/wiki/x/AYCaAg

Manual Testing Report – Voice Enabled AI Chatbot

Project Name: Voice-Enabled AI Chatbot for Moodle LMS

Prepared By: Syeda Tamanna Sheme

Date: April 25, 2025

1. Overview

This document summarizes the manual testing efforts carried out for the Voice-Enabled AI Chatbot. It covers the test planning, execution, bug reporting, and requirements traceability aspects.

2. Bug Report Summary

The following table highlights key bugs identified during manual testing:

Bug ID	Summary	Expected Result	Actual Result	Priority
C2	Voice Input - Student ID	Field populates with spoken ID	Field empty	High
C3	Voice Input - Phone Number	Field populates with spoken number	Partial capture	Medium
C4	Voice Input - Email	Field populates with spoken email	Invalid chars	Medium
C6	Invalid Student ID Input	Should reject input	Input accepted	Medium
C7	Invalid Email Input	Should reject input	Input accepted	Medium
C8	Successful Form Submission	Confirmation message shown	Submission failed	High
C9	Form Responsiveness	No horizontal scrolling	Mobile overflow	Medium
C10	Button Hover Effect	Visual feedback on hover	No effect	Low
C11	Voice Recognition Error Handling	Error notification shown	No error	Medium
C9	Form Responsiveness	No horizontal scrolling	Mobile overflow	Medium

G. Automation Testing Documentation

Automation Testing Documentation: https://students-team-vvj1dqxd.atlassian.net/wiki/x/EwCbAg

Automation Testing Report – Voice Enabled AI Chatbot

Project Name: Voice-Enabled AI Chatbot for Moodle LMS

Prepared By: Syeda Tamanna Sheme

Date: April 25, 2025

1. Overview

This report summarizes the results of automation testing conducted for the Voice-Enabled AI Chatbot project.
Jest was used as the test framework, integrated with TestRail for test case management and reporting.
The automation tests focused on validating component behavior, UI interactions, and functional correctness across core modules.

2. Automation Bug Report Summary

Test Case ID	Test Type	Test Scenario	Test Steps	Expected Result	Actual Result	Status	Comments
TC_I001	Integration Test	Verify Role Selection Page Loads	Open Role Selection Page	Page loads with title 'Choose Your Role'	Pass	Passed	Test passed successfully, behavior matches expectations.
TC_I002	Integration Test	Verify Student Role Button	Locate Student Role Button	Button labeled 'Student' is visible	Pass	Passed	Test passed successfully, behavior matches expectations.
TC_I003	Integration Test	Verify Instructor Role Button	Locate Instructor Role Button	Button labeled 'Instructor' is visible	Pass	Passed	Test passed successfully, behavior matches expectations.
TC_I004	Integration Test	Verify Admin Role Button	Locate Admin Role Button	Button labeled 'LMS Admin' is visible	Pass	Passed	Test passed successfully, behavior matches expectations.
TC_I005	Integration Test	Verify Navigation for Student Role	Click Student Role Button	Navigates to student.html	Fail	Failed	Issue detected - Navigates to student.html was

H. TestRail Report Documentation

TestRail Report Documentation: https://students-team-vvj1dqxd.atlassian.net/wiki/x/KgCbAg

TestRail Report Documentation

Project Name: Voice-Enabled AI Chatbot for Moodle LMS

Prepared By: Syeda Tamanna Sheme

Date: April 25, 2025

G.1 Overview

To manage and monitor the execution of manual and automated test cases, we integrated our QA workflow with TestRail, a comprehensive test case management platform. All test scenarios — including functional, non-functional, regression, and UI tests — were tracked and reported through TestRail during the development of the Voice-Enabled AI Chatbot for LMS project.

G.2 Test Suite Coverage

The following modules were covered in TestRail:

- Welcome Page
- Settings Page
- Voice Settings Page
- Ask Lumi Page
- Reset Password
- FAQ Page
- Chat Interface (Voice + Text Input)

Each module included Functional Test Cases and Non-Functional Test Cases (Performance, Accessibility, Security, Usability).

G.3 Test Execution Summary

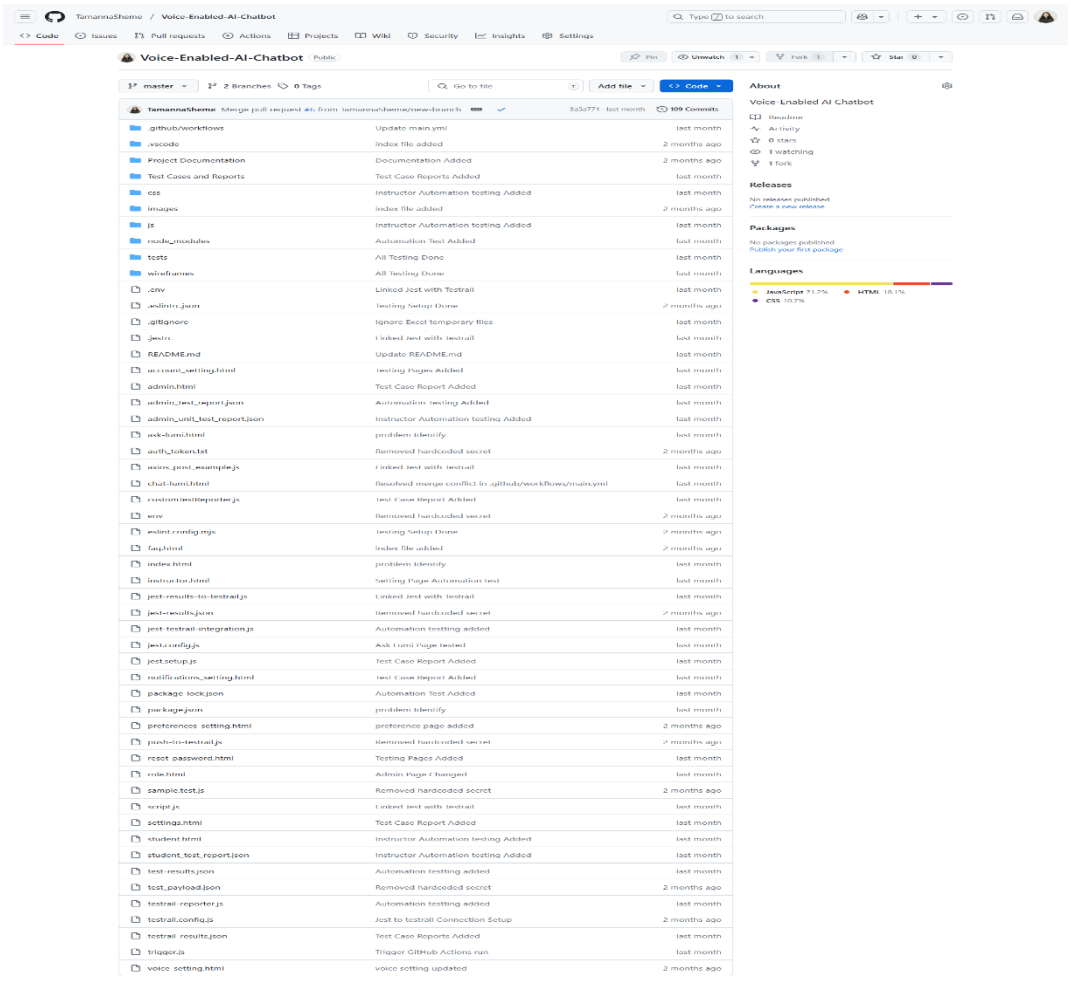
Module	Total Cases	Passed	Failed	Blocked	Untested
Welcome Page	9	✔ All Passed	-	-	-
Settings Page	6 + 4 Non-Functional	✔ All Passed	-	-	-
Voice Settings Page	6 + 3 Non-Functional	✔ All Passed	-	-	-
Ask Lumi Page	16 + 3 Non-Functional	✔ All Passed	-	-	-

G.4 Sample Test Case IDs and Titles

ID	Test Title	Result

I. GitHub Link:

GitHub Link: https://github.com/TamannaSheme/Voice-Enabled-AI-Chatbot



README

Voice-Enabled AI Chatbot

A voice-integrated AI chatbot designed to assist users within a Moodle-based Learning Management System (LMS). Built with a focus on accessibility, real-time interaction, user role customization, and robust testing.

Features

- **Voice Interaction** – Enables users to interact using speech (text-to-speech and speech-to-text).
- **Role-Based Access** – Custom interface and access for students, instructors, and admins.
- **Chat UI** – Clean, responsive chat interface with voice toggle.
- **Moodle LMS Integration** – Designed for seamless integration with Moodle environments.
- **UI Prototypes** – Created using Figma and translated to responsive web UI.
- **Comprehensive Testing** – Automated and manual testing conducted to ensure robust functionality and user experience.

Project Structure

```
├── css/              # All style sheets (Bootstrap + custom styles)
├── js/               # JavaScript for toggle, interaction logic
├── images/           # Static UI assets
├── wireframes/       # Figma-based design exports
├── index.html        # Main chatbot UI
├── role.html         # Role-based view
├── ask-lumi.html     # Lumi integration page
├── faq.html          # Help & FAQ page
├── settings.html     # User preferences (Tested UI)
├── tests/            # Automated test scripts (Unit and Integration)
├── Test Cases and Reports/ # Manual and Automated Testing Reports
```

Getting Started

To run locally:

1. Clone the repository:

```
git clone https://github.com/TamannaSheme/Voice-Enabled-AI-Chatbot.git
```

2. Open `index.html` in your browser.
3. Use the voice toggle to test speech interaction.

Technologies Used

- HTML5, CSS3, JavaScript (Vanilla)
- Bootstrap 5
- Figma (UI/UX Design)
- Git & GitHub
- Jest (Automated Testing)
- TestRail (Test Case Management)
- (Planned) Google Speech API / OpenAI API
- Moodle LMS (Integration Phase 2)

Project Status

✅ Sprint 1: UI design, Figma prototyping, initial frontend build, voice toggle ✅ Sprint 2: API integration, user authentication, backend connection ✅ Sprint 3: Instructor & student features, analytics, and feedback logic ✅ Automated testing: Unit and Integration Testing (Jest) ✅ Manual testing: Comprehensive Test Cases and Reports

License

This project is for academic use under Federation University – ITECH7415 Masters Project. All rights reserved by the project team.

Developed and Tested By

- Syeda Tamanna Sheme – Front-End Developer & Software Tester
- Tanvir Iqbal – Backend Developer
- Hansi Nipunika Panwillaarachchi Kotudura Bandanage – Backend Developer
- Afreen Begum – Business Analyst

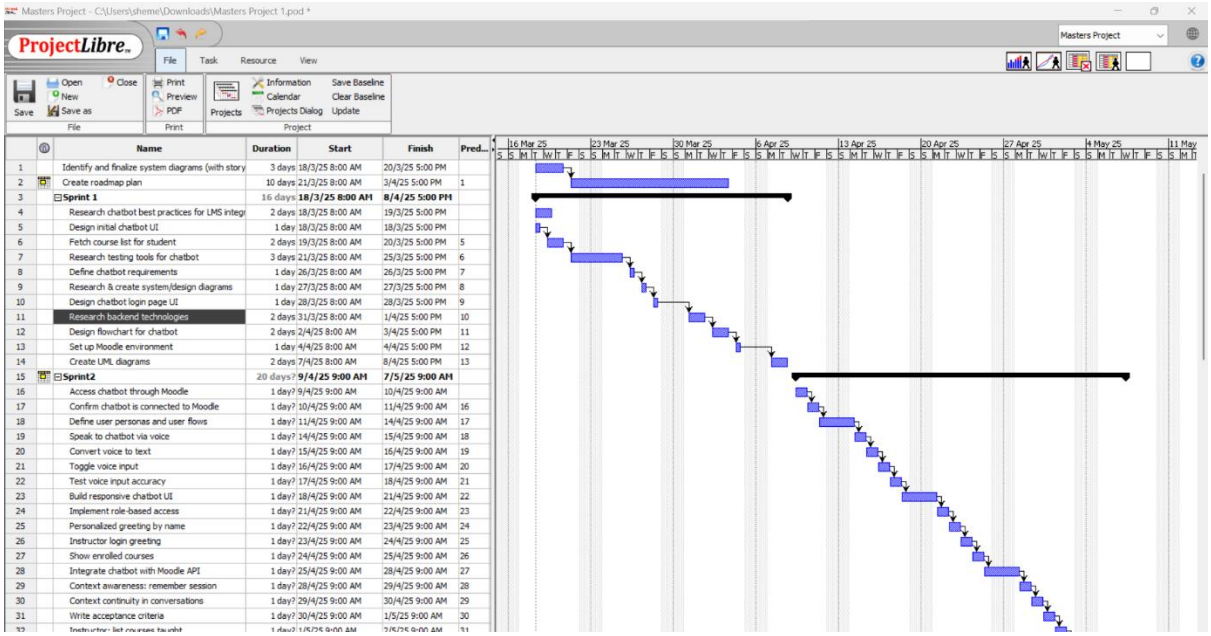
J. Project Timeline and Gantt Chart

Timeline Highlights:

Phase	Start Date	End Date	Duration	Key Activities
Project Kickoff	May 1, 2025	May 2, 2025	2 days	Team formation, tool setup, GitHub repo creation
Requirement Analysis	May 3, 2025	May 5, 2025	3 days	Functional analysis, Confluence structuring
UI Design (Figma)	May 6, 2025	May 9, 2025	4 days	Designing all chatbot-related interfaces
Frontend & Backend Dev	May 10, 2025	May 20, 2025	11 days	Feature implementation, RESTful APIs
Manual Testing & Bug Logging	May 15, 2025	May 23, 2025	9 days	Test case execution, Jira ticketing
Automation Testing (Jest)	May 18, 2025	May 28, 2025	11 days	Writing Jest cases, CI via GitHub
Test Report & Docs Finalization	May 25, 2025	May 30, 2025	6 days	Report collation, result analysis
Presentation & Client Feedback	May 31, 2025	June 2, 2025	3 days	Team presentation, assessment session

Gantt Chart View:

A Gantt chart exported from Project Libre visually outlines task sequences, dependencies, milestone deliveries, and resource assignments. It offers a clear timeline view of project progress and helped ensure on-time, coordinated execution across all development phases.



K. Team Contributions Summary

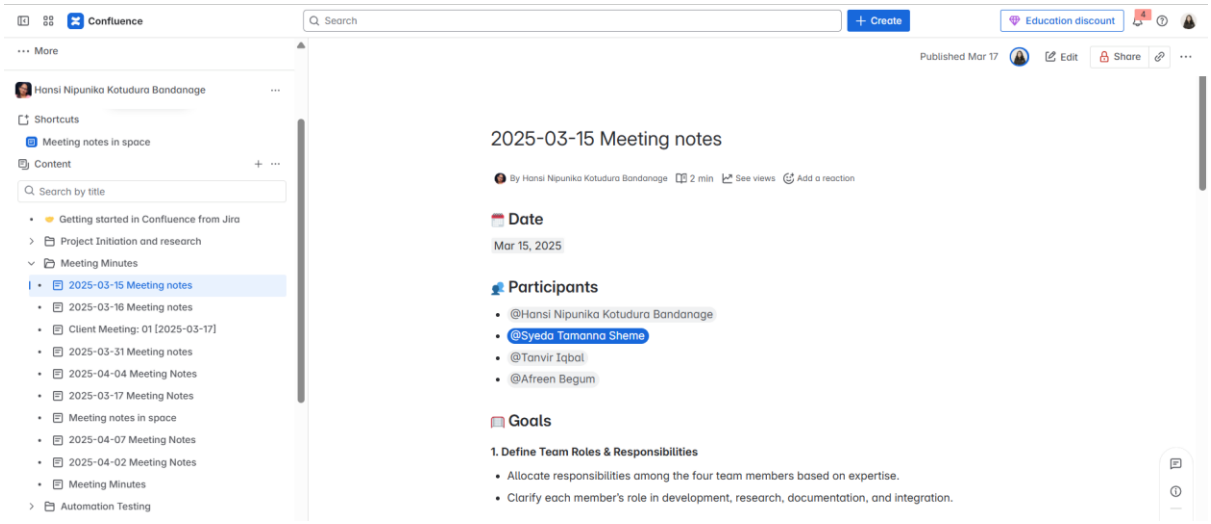
The project was driven by a collaborative team effort, with each member contributing their domain-specific expertise. Weekly meetings ensured coordination between development and testing. GitHub supported version control, Jira facilitated sprint tracking and bug reporting, and Confluence enabled seamless documentation and report writing.

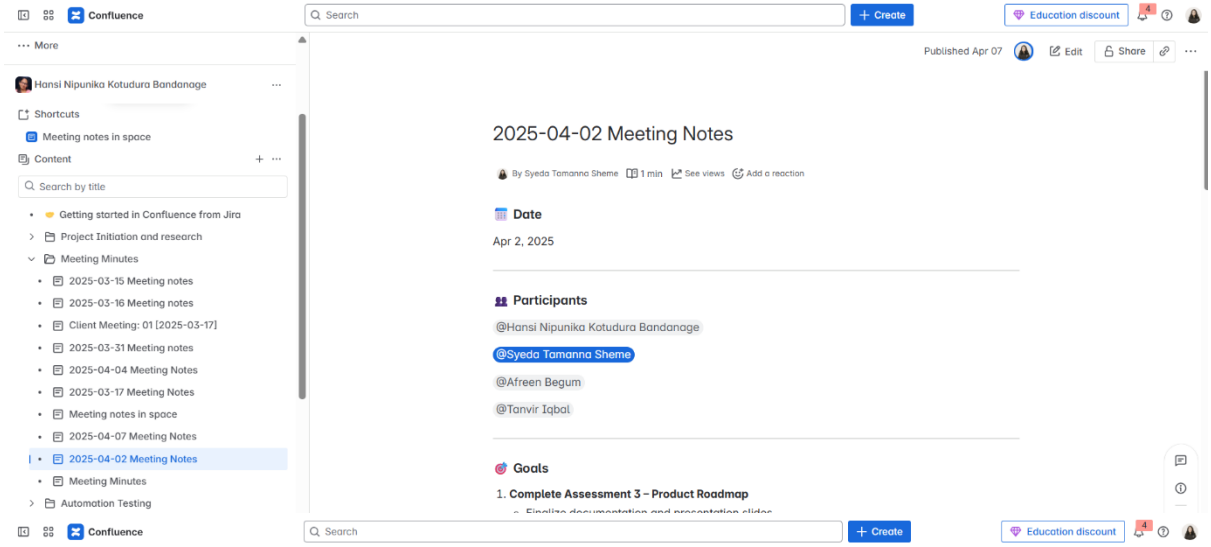
Name	Student ID	Role	Contributions
Syeda Tamanna Sheme	30432670	Front End Developer & Tester	- Designed UI wireframes in Figma for all chatbot pages - Developed frontend interfaces using HTML, CSS, Bootstrap, and JavaScript - Implemented voice input using Web Speech API - Led manual testing: created and executed test cases, logged bugs in Jira - Developed Jest automation tests and integrated results with TestRail
Tanvir Iqbal	30437681	Backend Developer	- Designed and implemented backend logic - Built RESTful APIs for chatbot interaction - Integrated user role-based routing and data handling - Assisted in TestRail API integration with Jest
Hansi Nipunika Panwillaarachchi Kotudura Bandanage	30423990	Backend Developer	- Handled database schema and role-based access control - Managed secure storage of user inputs and preferences - Collaborated on backend testing and bug fixes
Afreen Begum	30410327	Business Analyst	- Defined functional requirements and use case flows - Drafted SRS and RTM documentation - Conducted stakeholder analysis and identified chatbot use cases - Reviewed test plans and ensured user stories were fully covered

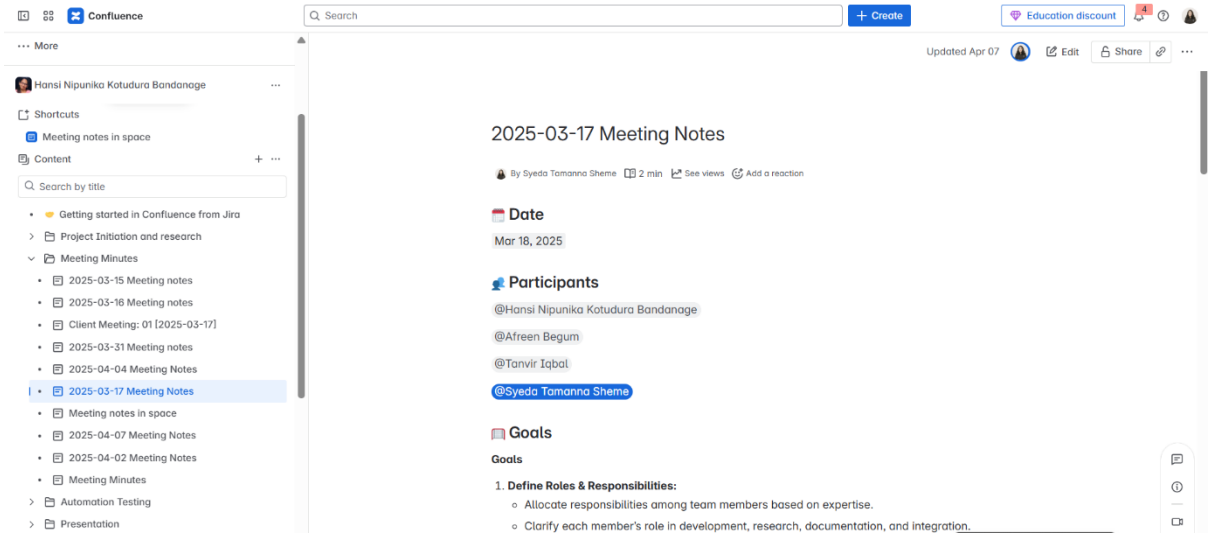
L. Meeting Records and Feedback

Meeting Cadence and Participation

Throughout the project, the team conducted biweekly meetings to ensure everyone was in the loop and things were getting done. All members of the team participated in every session. Meetings were held over the internet, with Zoom and Microsoft Teams. These meetings were helped along with follow-up emails, shared Trello boards and synced calendars. These weekly sessions were enablers for rapid updates, collaborative editing, rapid bug fixing and continued improvement.


Confluence
Search
Create
Education discount
More
Hansi Nipunika Kotudura Bandanage
Shortcuts
Meeting notes in space
Content
Search by title
Getting started in Confluence from Jira
Project Initiation and research
Meeting Minutes
2025-03-15 Meeting notes
2025-03-16 Meeting notes
Client Meeting: 01 [2025-03-17]
2025-03-31 Meeting notes
2025-04-04 Meeting Notes
2025-03-17 Meeting Notes
Meeting notes in space
2025-04-07 Meeting Notes
2025-04-02 Meeting Notes
Meeting Minutes
Automation Testing
Published Mar 17
Edit
Share
2025-03-15 Meeting notes
By Hansi Nipunika Kotudura Bandanage
2 min
See views
Add a reaction
Date
Mar 15, 2025
Participants
@Hansi Nipunika Kotudura Bandanage
@Syeda Tamanna Sheme
@Tanvir Iqbal
@Afreen Begum
Goals
1. Define Team Roles & Responsibilities
Allocate responsibilities among the four team members based on expertise.
Clarify each member's role in development, research, documentation, and integration.


Confluence
Search
Create
Education discount
More
Hansi Nipunika Kotudura Bandanage
Shortcuts
Meeting notes in space
Content
Search by title
Getting started in Confluence from Jira
Project Initiation and research
Meeting Minutes
2025-03-15 Meeting notes
2025-03-16 Meeting notes
Client Meeting: 01 [2025-03-17]
2025-03-31 Meeting notes
2025-04-04 Meeting Notes
2025-03-17 Meeting Notes
Meeting notes in space
2025-04-07 Meeting Notes
2025-04-02 Meeting Notes
Meeting Minutes
Automation Testing
Published Apr 07
Edit
Share
2025-04-02 Meeting Notes
By Syeda Tamanna Sheme
1 min
See views
Add a reaction
Date
Apr 2, 2025
Participants
@Hansi Nipunika Kotudura Bandanage
@Syeda Tamanna Sheme
@Afreen Begum
@Tanvir Iqbal
Goals
1. Complete Assessment 3 – Product Roadmap


Confluence
Search
Create
Education discount
More
Hansi Nipunika Kotudura Bandanage
Shortcuts
Meeting notes in space
Content
Search by title
Getting started in Confluence from Jira
Project Initiation and research
Meeting Minutes
2025-03-15 Meeting notes
2025-03-16 Meeting notes
Client Meeting: 01 [2025-03-17]
2025-03-31 Meeting notes
2025-04-04 Meeting Notes
2025-03-17 Meeting Notes
Meeting notes in space
2025-04-07 Meeting Notes
2025-04-02 Meeting Notes
Meeting Minutes
Automation Testing
Presentation
Updated Apr 07
Edit
Share
2025-03-17 Meeting Notes
By Syeda Tamanna Sheme
2 min
See views
Add a reaction
Date
Mar 18, 2025
Participants
@Hansi Nipunika Kotudura Bandanage
@Afreen Begum
@Tanvir Iqbal
@Syeda Tamanna Sheme
Goals
Goals
1. Define Roles & Responsibilities:
Allocate responsibilities among team members based on expertise.
Clarify each member's role in development, research, documentation, and integration.

Client Feedback Overview

Judge from what you posted the final stakeholder survey lauded the project in the following ways:

Feedback Area	Remarks
Documentation	Commended clarity in documentation, test reports, and wireframes.
Testing Quality	Recognized completeness and professionalism in both manual and automated tests.
Team Collaboration	Praised for confident delivery, good teamwork, and task ownership.
Rating	Received a strong rating of **8/10**, highlighting high technical capability.
Project Completion	Noted **full completion of objectives**, exceeding client expectations.
Standout Elements	Highlighted as one of the **most thorough teams** in the cohort.