# Software Requirements Specification (SRS) for AI Customized Learning Recommendation Chatbot

# 1. INTRODUCTION

## 1.1 Purpose

The purpose of this project is to develop an AI-powered Customized Learning Recommendation Chatbot that assists learners in understanding and mastering different topics through personalized quizzes, interactive feedback, and targeted study recommendations. The system aims to help students, self-learners, and educators identify knowledge gaps, improve retention, and enhance the learning process by leveraging Large Language Models (LLMs) integrated with a conversational user interface.

The chatbot will enable users to indicate their current understanding of a topic and select from predefined topics. Based on the context provided, the system will generate quiz questions, evaluate answers, and recommend tailored learning resources.

## 1.2 Document Conventions

| **Acronym** | **Description** |
| --- | --- |
| AI | Artificial Intelligence |
| LLM | Large Language Model |
| UI | User Interface |
| API | Application Programming Interface |
| DB | Database |
| MCQ | Multiple Choice Questions |

## 1.3 Intended Audience and Reading Suggestions

* **Educators & Learning Specialists**: To validate the pedagogical effectiveness of the system.
* **End Users**: Learners who will interact with the chatbot for study and quiz practice.

## 1.4 Project Scope

The AI Customized Learning Recommendation Chatbot will:

* Provide an interactive chat-based interface for learning and assessment.
* Allow users to choose from predefined topics.
* Provide instant feedback and explanations for each question.
* Recommend targeted study resources and follow-up exercises.

## References

* Streamlit Documentation – <https://docs.streamlit.io>
* OpenAI API Documentation – <https://platform.openai.com/docs>

# 2. OVERALL DESCRIPTION

## Product Perspective

The AI Customized Learning Recommendation Chatbot will function as a web-based application. It will consist of:

* **Frontend (Client):** An HTML, CSS, and JavaScript web interface for user interaction.
* **Backend (Server):** A Flask web server with Python-based logic for quiz generation, scoring, and recommendations.
* **LLM API Integration:** To generate questions, explanations, and study recommendations, likely using a framework like **LangChain**

## Product Features

* Interactive conversational UI for learning.
* Topic selection from a predefined list.
* Dynamic quiz generation with immediate feedback.
* Adaptive learning recommendations.

## 2.3 User Classes and Characteristics

* **Learners (Primary Users)**: Students, self-learners, or professionals seeking topic mastery. Requires basic computer literacy.
* **Educators (Secondary Users)**: Teachers and trainers who can review quizzes and monitor learner progress.
* **Administrators**: Manage topics, user data, and system configuration.

## 2.4 Operating Environment

* **Frontend:** Web browser with standard HTML, CSS, and JavaScript support.
* **Backend:** Python 3.9+ environment with the Flask and LangChain libraries installed.
* **LLM API**: Google Gemini or Groq.
* **OS Compatibility**: Windows, macOS, Linux.

## 2.5 Design and Implementation Constraints

* Requires an active internet connection for API access.
* LLM API usage is dependent on available credits and latency.
* The **Flask** backend must be configured to handle HTTP requests from the frontend and manage the application state.
* The HTML, CSS, and JavaScript UI needs to be responsive on both desktop and mobile devices.

## 2.6 Assumptions and Dependencies

* Users have stable internet access.
* API key for LLM service is available and valid.
* No severe API downtime from the LLM provider.

# 3. SYSTEM FEATURES

This project has a high priority for educational technology deployment as it addresses personalized learning needs.

## Stimulus/Response Sequences

User selects a predefined topic → System generates a quiz → User answers questions → System provides feedback.

User completes quiz → System recommends study materials and practice tasks.

## Functional Requirements

**FR-1: User Greeting and Level Input**

* The system shall greet the user upon opening the application.
* The user shall provide their self-assessed understanding level (Beginner, Intermediate, Advanced).
* The system shall store this information for customizing quizzes and recommendations.

**FR-2: Topic Selection**

* The system shall allow users to select from a predefined list of topics.

**FR-3: Quiz Generation**

* The system shall generate multiple-choice questions per topic.
* The questions shall be dynamically generated by the selected LLM (Google Gemini or Groq).
* Each question shall include four answer options and one correct answer.

**FR-4: Quiz Interaction**

* The system shall display questions with selectable radio buttons for answers.

**FR-5: Feedback and Scoring**

* The system shall evaluate user responses immediately after submission.
* The system shall display correct/incorrect answers with explanations.
* The system shall calculate a total score and highlight weak areas.

**FR-6: Recommendation Generation**

* Based on quiz performance, the system shall generate a personalized study plan.
* The system shall suggest additional practice tasks or resources for improvement.
* Recommendations shall be displayed in a readable, structured format within the chat interface.

# 4. EXTERNAL INTERFACE REQUIREMENTS

## 4.1 User Interfaces

* A chat-style interface built with HTML, CSS, and JavaScript for user interaction.
* A quiz UI with MCQs, rendered via HTML, CSS, and JavaScript.
* A feedback section with explanations and recommendations, also displayed in the web interface.

## 4.2 Hardware Interfaces

* Client devices: Laptop, desktop, tablet, or smartphone with a modern browser.
* Minimum screen resolution: 1280×720.

## 4.3 Software Interfaces

* **Operating System**: Cross-platform.
* **Frontend**: HTML, CSS, and JavaScript.
* **Backend:** Flask and **LangChain**.

# 5. NONFUNCTIONAL REQUIREMENTS

## 5.1 Performance Requirements

* Quiz generation within 3 seconds (API latency dependent).
* Support for at least 100 concurrent users.

## 5.2 Security Requirements

* API key encryption.

## 5.3 Software Quality Attributes

* **Availability**: 99% uptime.
* **Usability**: Simple, intuitive UI.
* **Maintainability**: Modular Python codebase for easy updates.
* **Scalability**: Support expansion to more topics and users.