

**A
PROJECT REPORT
ON**

AI-POWERED CONVERSATIONAL CHATBOT

SUBMITTED IN THE PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE
AWARD OF THE DEGREE OF
GURUGRAM UNIVERSITY
MASTER OF COMPUTER APPLICATION

2024-2025



SUBMITTED TO:

UNDER THE SUPERVISION OF:

MS. SARITA SHEERA

ASSOCIATE PROFESSOR, COMPUTER SCIENCE

GOVT. COLLEGE FOR GIRLS

SUBMITTED BY:

STUDENT NAME: SMARTI

UNIVERSITY ROLL NO: 23126410008

COLLEGE ROLL NO.: 2231914009

COURSE: MCA (FINAL YEAR)

GOVT. COLLEGE FOR GIRLS, SECTOR-14, GURUGRAM

DECLARATION

I hereby declare that the project work entitled "**AI-POWERED CONVERSATIONAL CHATBOT**" is an authenticated work carried out by me at Govt. College for Girls, Sector-14, Gurugram and this work has not been submitted for similar purpose anywhere else.

Date: 16-04-2025

Student Name: Smarti

University Roll No: 23126410008

Place: Gurugram

College Roll No.: 2231914009

CERTIFICATE

This is to certify that Smarti roll no. 23126410008 has worked under my supervision to prepare her project report. The work embodied in this report is original and was conducted at Govt. College for Girls, Gurugram. She has completed all requirements of MCA ordinance.

Ms. Sarita Sheera
Associate Professor, Computer Science

Govt. College for Girls, Sector-14, Gurugram
(Project Guide)

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my teacher and project guide **Ms. Sarita Sheera** for his valuable guidance. I am also thankful to all the faculty members of the Comp. Sc. Dept. for their encouragement and help while collecting information about the project.

Student Name: Smarti

University Roll No: 23126410008

College Roll No.: 2231914009

CONTENTS

	TOPIC	PAGE NO.
	ABSTRACT	1
1.	INDRODUCTION 1.1 PROBLEM DEFINITION 1.2 PROJECT AIMS AND OBJECTIVE 1.3 BACKGROUND OF PROJECT	2-5
2.	SYSTEM ANALYSIS 2.1 SOFTWARE REQUIREMENT SPECIFICATION 2.2 SOFTWARE TOOLS USED	6-30
3.	SYSTEM DESIGN 3.1 DATA FLOW DIAGRAM 3.2 ENTITY RELATIONSHIP DIAGRAM	31-39
4.	SYSTEM IMPLEMENTATION 4.1 SCREEN SHOTS OF APPLICATION 4.2 SCREEN SHOTS OF APPLICATION CODING PART 4.3 SOFTWARE TESTING	40-58
5.	CONCLUSION & FUTURE SCOPE	59-60
6.	REFERENCES	61

ABSTRACT

In the digital age, the integration of artificial intelligence in education is transforming the way students access information and seek guidance. This project aims to develop an **AI-powered chatbot web application** that can engage in human-like conversations and respond to predefined general knowledge, academic, and career-related queries. The chatbot is designed to simulate intelligent interaction using natural language processing (NLP) and provide instant answers to common questions, helping users learn in an interactive way.

The chatbot is built using **Flask (Python)** for the backend and **HTML, CSS, and JavaScript** for the frontend. It uses a predefined **data.json file** that stores commonly asked questions and their corresponding answers, eliminating the need for a database. The system includes a user authentication module for **login and signup**, ensuring session-based access to the chatbot interface.

The bot provides informative responses in domains such as general knowledge, science, space, mathematics, fun facts, riddles, and educational tips. Additionally, it is structured to be scalable and can easily be extended to integrate more advanced NLP models via APIs like **OpenAI or Dialogflow**.

This project not only enhances student learning through conversational AI but also demonstrates the integration of AI with modern web technologies in a practical, user-friendly format.

CHAPTER 1:

INTRODUCTION

1.1 PROBLEM DEFINITION

In the rapidly evolving digital world, the importance of quick access to accurate and relevant information cannot be overstated—especially in the field of education. Students today face numerous academic and career-related challenges and often seek timely answers to questions on diverse topics like general knowledge, mathematics, science, health, and career guidance. Traditionally, these queries are addressed through textbooks, peers, teachers, or internet searches. However, these methods have limitations. They are often time-consuming, not personalized, may not be available 24/7, and can lead to distractions due to unrelated search results.

Moreover, with the advent of online learning and self-paced study platforms, the role of intelligent support systems has become increasingly crucial. Educational institutions often lack real-time, round-the-clock assistance systems for handling student doubts effectively. Additionally, many students are shy or hesitant to ask questions in public settings or classrooms, which further limits their access to information and guidance.

This problem intensifies in rural and under-resourced areas where students might not have direct access to mentors or experts for their day-to-day academic concerns. In such scenarios, an AI-powered chatbot becomes a valuable educational companion—offering immediate, accurate, and relevant answers to common queries in a friendly and conversational manner.

The need for an efficient, always-available, and intuitive system that can guide students and help them with both academic and general queries is the fundamental problem that this project aims to address.

1. Lack of Personalized Instant Support

- Students often struggle to get quick, personalized answers to questions spanning education, career, general knowledge, science, and health without navigating through multiple platforms.

2. Overwhelming and Scattered Information

- With knowledge scattered across various websites, search engines, and forums, students waste valuable time searching for straightforward answers.

3. Limited Access to Human-like Interactions

- Traditional FAQs or static search engines lack the ability to interact conversationally, understand natural language, or remember context—making them less effective for engaging learning.

4. Inaccessibility of Career and Educational Guidance

- Many students lack access to guidance counselors and depend on online resources which may be too generic, unstructured, or hard to trust.

5. Absence of a Unified, Knowledge-Rich Platform

- There's no single place where students can ask about diverse topics such as math tricks, science facts, Indian states and capitals, body health tips, resume-building guidance, and more.

6. Underutilization of AI Capabilities in Education

- Despite the advancements in AI, especially in language models, most educational websites and apps don't fully leverage AI-powered chatbots to help students learn, explore, and grow interactively.

7. Difficulty in Understanding Complex Language

- Many platforms provide answers that are too technical or not aligned with the understanding level of school or college students.

8. Limited Customizable Solutions for Institutions

- Schools and colleges often lack a ready-to-deploy, customizable chatbot solution that can be adapted to their curriculum or student support needs.

1.2 PROJECT AIMS AND OBJECTIVES

The primary aim of this project is to develop a functional, user-friendly chatbot web application that can understand and respond to user queries in a conversational manner, providing an efficient alternative to traditional customer support methods.

Primary Objectives:

1. User Interface Development

- Create a responsive, intuitive chat interface that works across different devices
- Design visual elements that clearly distinguish between user and bot messages
- Implement smooth animations and transitions for a polished user experience
- Ensure accessibility compliance for users with disabilities

2. Backend System Implementation

- Develop a Flask-based server to handle user requests and generate responses
- Create an efficient message routing and processing system
- Implement error handling and fallback mechanisms for robust operation
- Optimize server performance for rapid response generation

3. Communication Architecture

- Establish a reliable JSON-based communication protocol between frontend and backend
- Implement proper data validation and sanitization mechanisms
- Create a messaging structure that supports various content types
- Design an extensible API structure for future enhancements

4. Conversational Capabilities

- Develop a knowledge base for responding to common queries
- Implement basic natural language understanding capabilities
- Create context-aware conversation flows for more natural interactions
- Design an expandable response framework for ongoing improvement

Secondary Objectives:

1. Performance Optimization

- Minimize response latency for improved user experience
- Optimize resource utilization on both client and server sides
- Implement efficient data caching mechanisms where appropriate

2. Documentation and Knowledge Transfer

- Create comprehensive technical documentation of all system components
- Develop user guides and administrator documentation
- Document code with clear comments and explanations
- Establish framework for ongoing maintenance and enhancements

3. Testing and Quality Assurance

- Implement unit and integration testing frameworks
- Conduct usability testing with representative user groups
- Perform security testing to identify and address vulnerabilities
- Execute performance testing under varying load conditions

This chatbot serves not only as a useful tool for students but also as a showcase project that demonstrates the integration of frontend, backend, and AI components in a real-world web application.

1.3 BACKGROUND OF THE PROJECT

Chatbots have evolved significantly from the earliest rule-based systems of the 1960s to today's sophisticated AI-powered conversational agents. This evolutionary path reflects advances in natural language processing, machine learning, and computational capabilities.

Historical Context

The concept of conversational agents dates back to the 1960s with Joseph Weizenbaum's ELIZA program, which simulated conversation using simple pattern matching. This was followed by more sophisticated systems like PARRY in the 1970s. The 1990s and early 2000s saw the emergence of commercial chatbots for customer service applications, primarily using rule-based approaches.

The last decade has witnessed a revolution in chatbot capabilities with the integration of machine learning, particularly deep learning techniques. Modern systems like those from major technology companies leverage vast datasets and neural network architectures to achieve increasingly natural conversations.

Current Landscape

Today's chatbot ecosystem spans a wide spectrum of implementations:

1. **Rule-based Systems:** Simple but effective for narrow domains with predictable interactions
2. **Retrieval-based Models:** Select responses from a predefined set based on input similarity
3. **Generative Models:** Create original responses using neural networks
4. **Hybrid Systems:** Combine multiple approaches for optimal performance

This project builds upon this rich history while focusing on creating a lightweight, accessible implementation. By combining web technologies with backend processing capabilities, the system balances functionality with performance considerations.

Market Relevance

The increasing adoption of web-based chat interfaces for customer service and information retrieval demonstrates the relevance of this application. According to recent industry reports, chatbot implementations can reduce customer service costs by up to 30% while improving response times and availability. Organizations across sectors including e-commerce, healthcare, education, and financial services are increasingly integrating chatbot solutions into their customer engagement strategies.

The project draws inspiration from both commercial chatbots and open-source implementations, adapting core concepts to create a practical, customizable solution that can be tailored to specific use cases and integrated with existing systems.

CHAPTER 2:

SYSTEM ANALYSIS

2.1 SYSTEM REQUIREMENTS

To successfully develop, deploy, and run the AI-based chatbot web application, certain hardware and software prerequisites must be met. These requirements ensure the system functions efficiently during both development and execution phases.

2.1.1 HARDWARE REQUIREMENTS

Component	Minimum Requirement
Processor	Intel Core i3 or higher
RAM	4 GB (8 GB recommended for smoother dev)
Hard Disk	100 MB for source files, 1 GB for logs and environment
Monitor	720p resolution (1080p recommended)
Keyboard and Mouse	Standard input devices
Internet Connection	Required for package installations (optional during runtime)

2.1.2 SOFTWARE REQUIREMENTS

Software	Version / Details
Operating System	Windows 10/11, Linux (Ubuntu), or macOS
Python	Version 3.8 or above
Flask Framework	Flask 2.0 or above
HTML/CSS/JavaScript	For frontend web development
Web Browser	Chrome, Firefox, or Edge
Text Editor / IDE	VS Code / PyCharm / Sublime Text
Package Installer	pip (Python package installer)

Software	Version / Details
Git (optional)	Version control and collaboration

2.2 TOOLS AND TECHNOLOGIES USED

The project is built using a full-stack development approach, incorporating both frontend and backend tools, along with lightweight AI logic.

2.2.1 PROGRAMMING LANGUAGES

- **Python:** Used for backend development with Flask and chatbot logic handling.
- **HTML5:** For structuring the chatbot interface and login/signup pages.
- **CSS3:** For styling and layout customization.
- **JavaScript:** To enhance interactivity on the frontend (e.g., chat animations, scroll effects).

2.2.2 FRAMEWORKS AND LIBRARIES

- **Flask (Python):** A micro web framework used to build the server-side logic and manage user sessions.
- **WTForms:** Used to handle user input and form validation in the registration/login process.
- **Flask-Login:** Provides user session management and login tracking.
- **JSON:** Used to store predefined Q&A pairs in a structured format for chatbot responses.

2.2.3 DEVELOPMENT TOOLS

- **Visual Studio Code (VS Code):** Primary IDE used for writing and testing code.
- **Git:** Version control for tracking changes and managing project history.
- **Google Chrome:** Used to test the web interface during development.
- **Postman (optional):** For testing API endpoints, if added later.

2.2.4 ENVIRONMENT AND DEPLOYMENT TOOLS

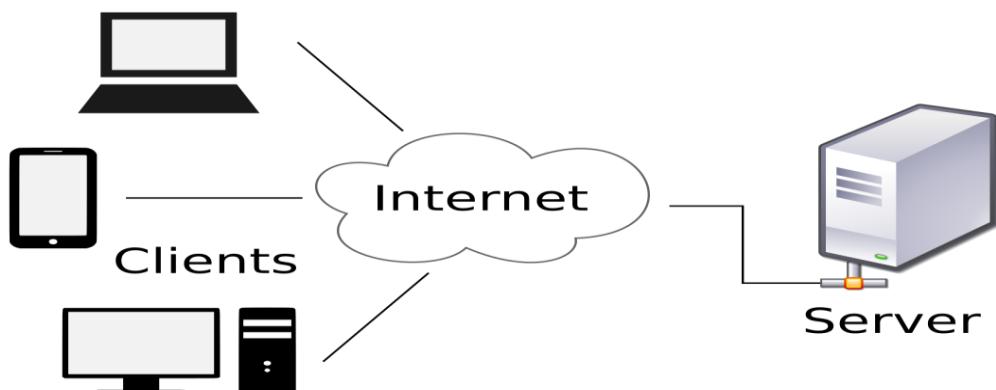
- **Virtual Environment:** Used to manage Python dependencies in isolation.
- **Flask Development Server:** For running and testing the application locally.
- **PythonAnywhere / Heroku (optional):** Platforms where the chatbot can be deployed publicly.

2.3 KEY FEATURES OF THE TOOLS

Tool/Technology	Role in the Project
Flask	Handles routing, server logic, form processing, and response generation
WTForms	Validates user inputs for secure login and registration
HTML/CSS/JS	Designs the user interface and manages dynamic behavior of chatbot interactions
JSON File	Acts as a lightweight local database for storing Q&A pairs
Python	Ties everything together with logical control flow and response matching
VS Code	Offers an efficient, integrated development environment

2.4 SYSTEM ARCHITECTURE OVERVIEW

The system is built using a **client-server architecture**, where the frontend interface (client) interacts with the backend server, which processes user inputs and generates chatbot responses. The core architecture revolves around Flask (Python) for server-side logic and predefined responses stored in a JSON file for chatbot intelligence.



2.4.1 Functional Requirements

User Interface Requirements

1. Chat Interface

- The system must provide a chat interface with distinct areas for message history and input
- The interface must support text-based communication between the user and chatbot
- The system must visually differentiate between user messages and bot responses
- The chat history must display messages in chronological order
- The interface must auto-scroll to the newest message when conversations extend beyond the visible area

2. Input Mechanisms

- The system must provide a text input field for users to type messages
- The interface must include a send button to submit messages
- The system must support keyboard shortcuts (Enter key) for message submission
- The interface must provide visual feedback when messages are being sent
- The system must prevent submission of empty messages

3. Visual Elements

- The system must include a header area with the chatbot's name or purpose
- The interface must display a typing indicator when the bot is generating a response
- The system must support basic text formatting in messages (bold, italic, links)
- The interface must use appropriate color contrast for readability
- The system must adapt its layout to different screen sizes (responsive design)

Chat Functionality Requirements

1. Input Processing

- The system must accept and process text input from users
- The system must handle special characters and emoji appropriately
- The system must manage input of various lengths (from single words to paragraphs)

- The system must trim leading and trailing whitespace from user input
- The system must sanitize user input to prevent injection attacks

2. Response Generation

- The system must analyze user input to determine appropriate responses
- The system must generate contextually relevant replies to user queries
- The system must provide helpful responses for common questions
- The system must acknowledge when it cannot provide a specific answer
- The system must offer alternative information when exact answers are unavailable

3. Conversation Flow

- The system must maintain a logical flow of conversation
- The system must support multi-turn interactions on the same topic
- The system must provide appropriate follow-up questions when relevant
- The system must allow users to change topics naturally
- The system must recognize and respond to conversation closures (goodbyes)

4. Error Handling

- The system must detect and respond to obvious user errors
- The system must provide guidance when user queries are unclear
- The system must offer suggestions when no matching response is available
- The system must handle network connectivity issues gracefully
- The system must inform users when technical problems prevent proper functioning

Backend Processing Requirements

1. Message Analysis

- The system must parse and interpret user messages
- The system must identify key terms and concepts in user input
- The system must determine user intent from message content
- The system must extract entities mentioned in user queries
- The system must recognize question types (who, what, when, where, why, how)

2. Knowledge Management

- The system must access appropriate information from its knowledge base
- The system must organize information for contextual relevance
- The system must prioritize more recent or important information when appropriate
- The system must structure responses for clarity and readability
- The system must provide consistent answers to similar questions

3. Response Formatting

- The system must generate grammatically correct responses
- The system must maintain appropriate tone and formality
- The system must structure complex information in digestible formats
- The system must include relevant details while avoiding excessive length
- The system must format responses appropriately for the chat interface

4. Session Management

- The system must maintain conversation context during a session
- The system must track the current state of each user interaction
- The system must support multiple simultaneous users
- The system must handle session timeouts appropriately
- The system must ensure data isolation between different user sessions

2.4.2 Non-Functional Requirements

Performance Requirements

1. Response Time

- The system must provide responses within 2 seconds under normal conditions
- The system must acknowledge message receipt immediately upon submission
- The interface must remain responsive during backend processing
- The system must optimize network communication to minimize latency
- The system must maintain performance with concurrent users

2. Scalability

- The system must support multiple simultaneous users without degradation
- The system architecture must allow for horizontal scaling
- The system must efficiently handle varying load conditions
- The system must optimize resource utilization during peak usage

- The system must maintain performance as the knowledge base grows

3. Reliability

- The system must achieve 99.5% uptime during operating hours
- The system must recover gracefully from failures
- The system must preserve conversation state during minor disruptions
- The system must implement appropriate logging for troubleshooting
- The system must handle unexpected input without crashing

Usability Requirements

1. Ease of Use

- The interface must be intuitive and require no special training
- The system must provide clear instructions for first-time users
- The system must use familiar chat conventions and metaphors
- The interface must minimize user effort for common tasks
- The system must offer guidance when users appear confused

2. Accessibility

- The interface must comply with WCAG 2.1 AA standards
- The system must support keyboard navigation for all functions
- The interface must work with screen readers and assistive technologies
- The system must maintain sufficient color contrast for readability
- The interface must include alternative text for all non-text elements

3. User Experience

- The interface must present a professional and appealing visual design
- The system must provide appropriate feedback for all user actions
- The interface must include subtle animations for improved engagement
- The system must minimize user frustration during error conditions
- The interface must maintain consistent design patterns throughout

Security Requirements

1. Data Protection

- The system must sanitize all user inputs to prevent injection attacks
- The system must implement appropriate input validation

- The system must use secure transmission protocols (HTTPS)
- The system must not store sensitive information unless necessary
- The system must encrypt any persistent data

2. Privacy

- The system must comply with relevant data protection regulations
- The system must clearly inform users about data usage policies
- The system must minimize collection of personal information
- The system must allow users to delete their conversation history
- The system must not share user data with third parties without consent

3. System Security

- The backend must implement appropriate authentication mechanisms
- The system must apply principle of least privilege for all operations
- The system must include rate limiting to prevent abuse
- The system must log security-relevant events
- The system must undergo regular security assessments

Compatibility Requirements

1. Browser Compatibility

- The interface must function correctly on current versions of Chrome, Firefox, Safari, and Edge
- The system must degrade gracefully on older browsers
- The interface must adapt to different screen sizes and resolutions
- The system must work consistently across operating systems
- The interface must support both touch and pointer input devices

2. Device Compatibility

- The interface must function on desktop and mobile devices
- The system must adapt to different screen orientations on mobile
- The interface must remain usable on small screens (minimum 320px width)
- The system must minimize bandwidth usage for mobile users
- The interface must implement appropriate touch targets for mobile users

3. Integration Capabilities

- The system must implement standard APIs for potential integration

- The system must support common data exchange formats
- The system architecture must allow for extension with additional services
- The system must document all external interfaces
- The system must maintain backward compatibility for API versions

2.5 ARCHITECTURE LAYERS

1. Presentation Layer (Frontend)

- HTML, CSS, JavaScript
- Responsible for the user interface (UI), including login, registration, and chatbot interaction page.
- Provides recommended questions and captures user input in real-time.

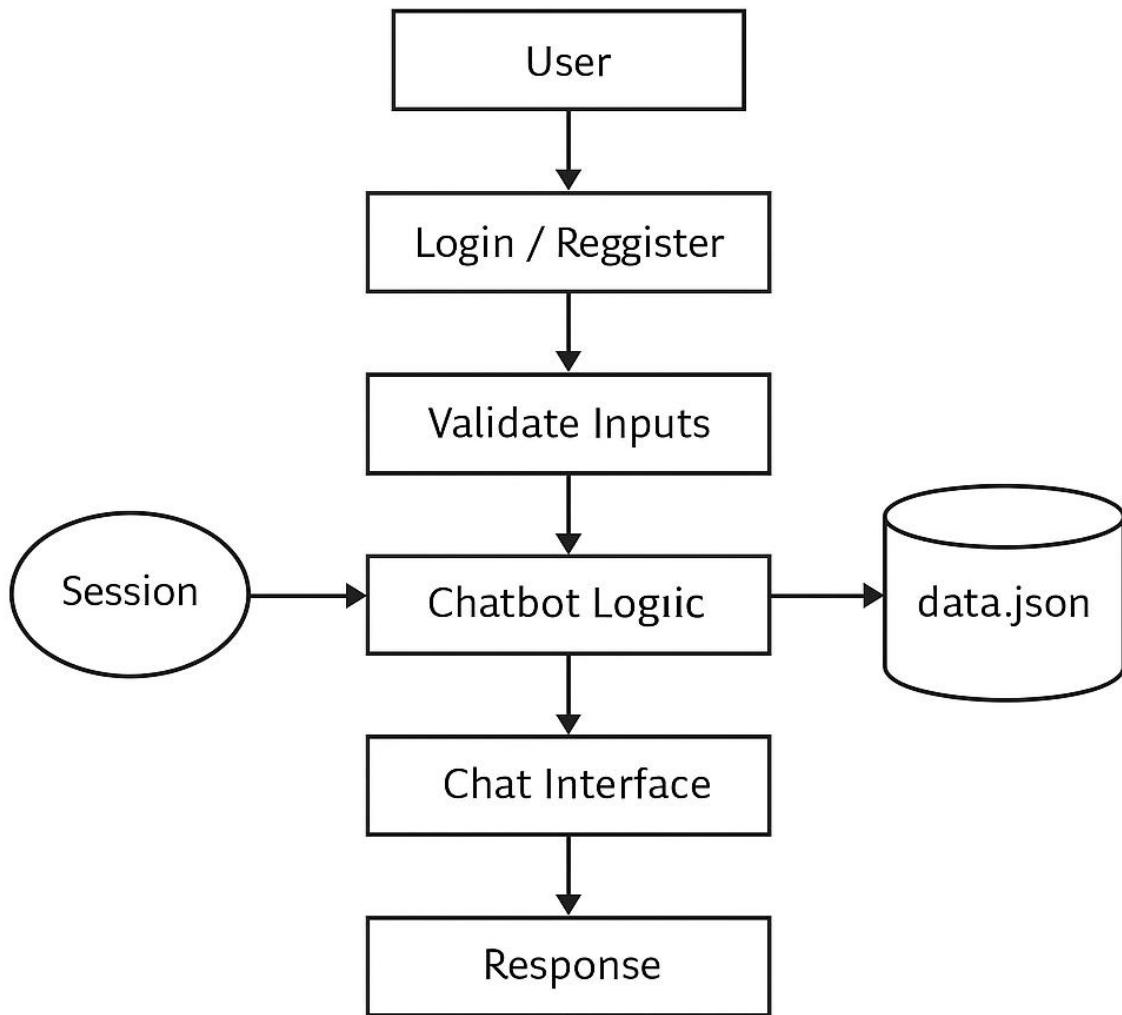
2. Application Layer (Backend)

- Python (Flask Framework)
- Handles user session management, form processing (WTForms), chatbot response routing, and data handling from JSON.
- Acts as a bridge between the UI and the chatbot logic.

3. Data Layer

- Static JSON file containing predefined question-answer pairs.
- No external database is used, making it lightweight and easy to manage.
- User authentication is managed temporarily using Flask sessions (no persistent storage).

2.6 SYSTEM ARCHITECTURE DIAGRAM



2.7 SOFTWARE TOOLS USED

2.7.1 Frontend Technologies

HTML Introduction

HTML stands for HyperText Markup Language. It is the standard markup language used to create web pages. HTML is a combination of Hypertext and Markup language. Hypertext defines the link between web pages. A markup language is used to define the text document within the tag to define the structure of web pages.

This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

What is HTML?

HTML stands for HyperText Markup Language and it is used to create webpages. It uses HTML tags and attributes to describe the structure and formatting of a web page.

HTML consists of various elements, which are responsible for telling search engines how to display page content. For example, headings, lists, images, links, and more.



HTML Example

```
<!DOCTYPE html>
<html>
<head>
    <title>First HTML Code</title>
</head>
<body>
    <h2>Welcome To World</h2>
    <p>Hello World</p>
</body>
</html>
```

Features of HTML

- It is easy to learn and easy to use.
- It is platform-independent.
- Images, videos, and audio can be added to a web page.
- Hypertext can be added to the text.
- It is a markup language.

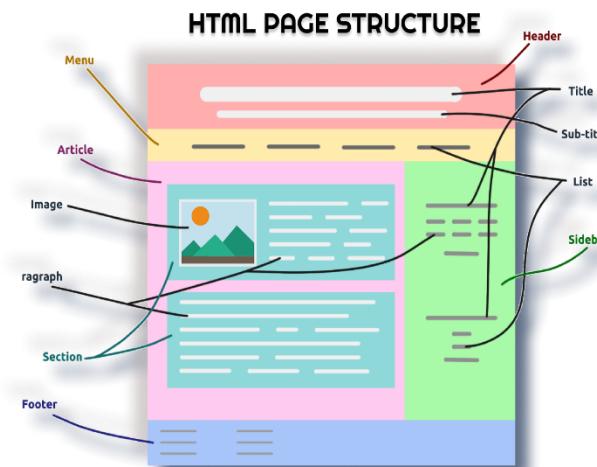
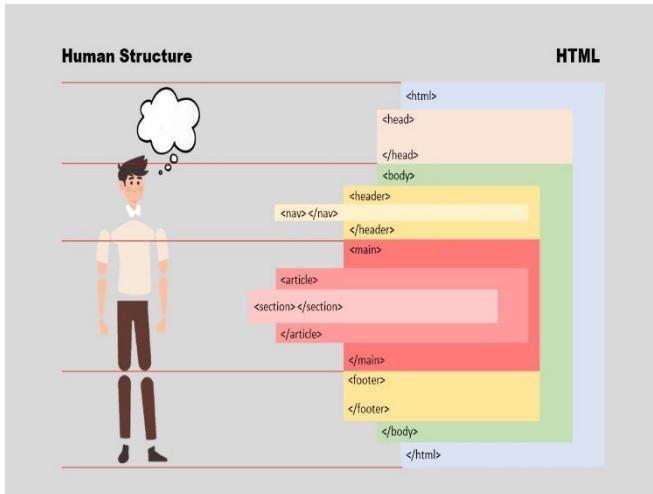
HTML Elements and Tags

HTML uses predefined tags and elements that tell the browser how to display the content. HTML elements include an opening tag, some content, and a closing tag.

Remember to include closing tags. If omitted, the browser applies the effect of the opening tag until the end of the page.

HTML Page Structure

The basic structure of an HTML page is shown below. It contains the essential building-block elements (i.e. doctype declaration, HTML, head, title, and body elements) upon which all web pages are created.



HTML Page Structure

```

<!DOCTYPE html> ← Tells version of HTML
<html> ← HTML Root Element

<head> ← Used to contain page HTML metadata
  <title>Page Title</title> ← Title of HTML page
</head>

<body> ← Hold content of HTML
  <h2>Heading Content</h2> ← HTML heading tag
  <p>Paragraph Content</p> ← HTML paragraph tag
</body>

</html>

```

- <!DOCTYPE html> – This is the document type declaration (not technically a tag). It declares a document as being an HTML document. The doctype declaration is not case-sensitive.
- <html> – This is called the HTML root element. All other elements are contained within it.
- <head> – The head tag contains the “behind the scenes” elements for a webpage. Elements within the head aren’t visible on the front end of a webpage. HTML elements used inside the <head> element include:
- <style> – This HTML tag allows us to insert styling into our web pages and make them appealing to look at with the help of CSS.
- <title> – The title is what is displayed on the top of your browser when you visit a website and contains the title of the webpage that you are viewing.

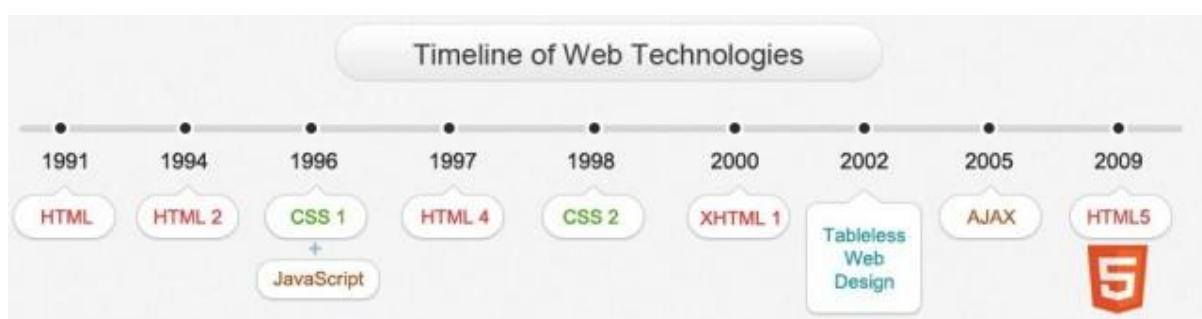
- <base> – It specifies the base URL for all relative URL's in a document.
- <noscript> – Defines a section of HTML that is inserted when the scripting has been turned off in the user's browser.
- <script> – This tag is used to add functionality to the website with the help of JavaScript.
- <meta> – This tag encloses the metadata of the website that must be loaded every time the website is visited. For eg:- the metadata charset allows you to use the standard UTF-8 encoding on your website. This in turn allows the users to view your webpage in the language of their choice. It is a self-closing tag.
- <link> – The ‘link’ tag is used to tie together HTML, CSS, and JavaScript. It is self-closing.
- <body> – The body tag is used to enclose all the visible content of a webpage. In other words, the body content is what the browser will show on the front end.

An HTML document can be created using an HTML text editor. Save the text file using the “.html” or “.htm” extension. Once saved as an HTML document, the file can be opened as a webpage in the browser.

HTML History

HTML is a markup language used by the browser to manipulate text, images, and other content, in order to display it in the required format. HTML was created by Tim Berners-Lee in 1991. The first-ever version of HTML was HTML 1.0, but the first standard version was HTML 2.0, published in 1995.

Currently, we are using HTML5, which is the latest and most recent version of HTML.



Advantages of HTML

- HTML is used to build websites.
- It is supported by all browsers.
- It can be integrated with other languages like CSS, JavaScript, etc.

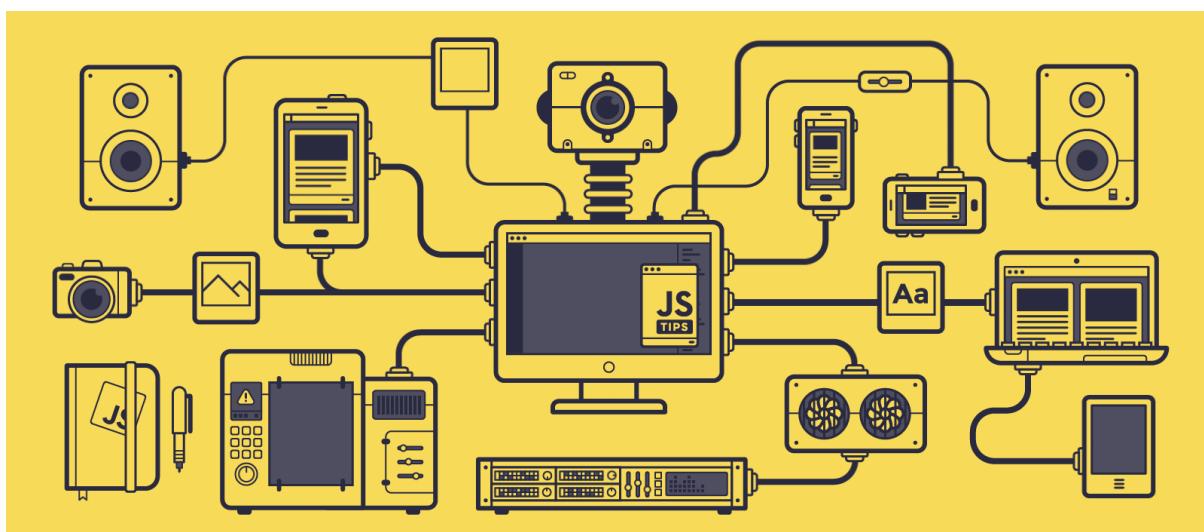
Disadvantages of HTML

- HTML can only create static web pages. For dynamic web pages, other languages have to be used.
- A large amount of code has to be written to create a simple web page.
- The security feature is not good.

JavaScript

Introduction:

JavaScript is a high-level, interpreted programming language primarily used for adding interactivity to web pages. It is one of the core technologies of the World Wide Web, along with HTML and CSS, and is supported by all modern web browsers without the need for plugins.



Features:

- **Client-side Scripting:** JavaScript executes code on the client's browser, enabling dynamic content manipulation, form validation, and user interaction without the need for server-side processing.

- **Event-driven Programming:** JavaScript supports event-driven programming, allowing developers to respond to user actions such as clicks, mouse movements, and keyboard inputs.
- **DOM Manipulation:** JavaScript provides powerful APIs for accessing and manipulating the Document Object Model (DOM) of web pages, allowing developers to modify HTML elements, attributes, and styles dynamically.
- **Asynchronous Programming:** JavaScript supports asynchronous programming patterns such as callbacks, promises, and async/await, enabling non-blocking I/O operations and responsive user interfaces.
- **Cross-platform Compatibility:** JavaScript is supported by all major web browsers and can also be used for server-side development using platforms like Node.js.

Advantages:

- **Versatility:** JavaScript is a versatile language that can be used for both client-side and server-side development, allowing developers to build full-stack web applications using a single programming language.
- **Interactivity:** JavaScript enhances user interactivity by enabling dynamic content updates, form validation, and real-time interactions, making web pages more engaging and responsive.
- **Extensibility:** JavaScript has a rich ecosystem of libraries and frameworks such as React, Angular, and Vue.js, which provide additional functionality and simplify complex tasks like state management and UI development.
- **Community Support:** JavaScript has a large and active developer community, providing access to a vast array of tutorials, documentation, and open-source projects for learning and collaboration.
- **Performance:** With modern JavaScript engines like V8, JavaScript performance has improved significantly, enabling complex computations and real-time processing on the client-side.

Disadvantages:

- **Browser Compatibility:** JavaScript behavior may differ across different browsers and versions, requiring developers to test and ensure compatibility with multiple platforms.
- **Security Risks:** JavaScript can introduce security vulnerabilities such as cross-site scripting (XSS) and cross-site request forgery (CSRF) if not implemented securely, requiring developers to follow best practices for code security.
- **Single-threaded Execution:** JavaScript executes code in a single thread, which can lead to blocking operations and performance bottlenecks in CPU-intensive tasks or long-running operations.
- **Callback Hell:** Asynchronous programming in JavaScript can lead to callback hell, a situation where nested callbacks become difficult to manage and maintain, making code unreadable and error-prone.
- **Learning Curve:** JavaScript has a steep learning curve, especially for beginners, due to its asynchronous nature, prototype-based inheritance, and dynamic typing, requiring time and effort to master.

CSS (Cascading Style Sheets)

Introduction:

CSS is a stylesheet language used to describe the presentation of a document written in HTML or XML. It allows developers to control the layout, styling, and appearance of web pages, including elements such as text, fonts, colors, margins, padding, and positioning.



Features:

- **Selectors:** CSS allows developers to select HTML elements based on various criteria such as element type, class, ID, attributes, or hierarchy, enabling precise targeting and styling of specific elements.
- **Box Model:** CSS defines the box model, which describes how elements are rendered on the page, including dimensions, padding, border, and margin, providing control over element spacing and layout
- **Flexbox and Grid:** CSS introduces layout mechanisms like Flexbox and Grid, which allow developers to create responsive and flexible designs, arranging elements in a variety of configurations and alignments.
- **Media Queries:** CSS supports media queries, enabling developers to apply different styles based on factors such as screen size, device orientation, or resolution, making web pages responsive and adaptable to different devices and screen sizes.
- **Transitions and Animations:** CSS provides features for creating transitions and animations, allowing developers to add visual effects such as fades, slides, and rotations to elements, enhancing user experience and engagement.

Advantages:

- **Separation of Concerns:** CSS separates presentation from content, enhancing maintainability and reusability by allowing developers to apply consistent styling across multiple pages or an entire website.
- **Consistency:** CSS ensures consistent styling and layout across different pages or elements, reducing design inconsistencies and improving user experience.
- **Accessibility:** CSS supports accessibility features such as text resizing, contrast adjustments, and screen reader compatibility, making web pages more usable by people with disabilities.
- **Performance:** CSS enables lightweight and efficient styling of web pages, reducing file size and bandwidth usage, and improving page load times and performance.
- **Print Styling:** CSS allows developers to define print-specific stylesheets, providing control over how web pages are formatted and printed, improving print quality and readability.

Disadvantages:

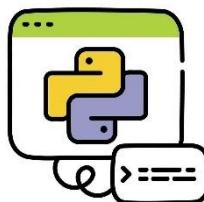
- **Specificity Issues:** Managing CSS specificity can lead to unexpected styling behavior, as conflicting styles may override each other based on their specificity, requiring careful attention to CSS selectors and cascade order.

- **Browser Compatibility:** CSS features may not be uniformly supported across all browsers, leading to inconsistencies and rendering issues, requiring developers to test and ensure compatibility with multiple platforms.
- **Learning Curve:** Mastering CSS requires understanding of its syntax, selectors, properties, and layout mechanisms, as well as best practices for styling and optimization, which can be challenging for beginners.
- **Complexity:** CSS can become complex and difficult to maintain in large-scale projects with extensive stylesheets, especially when dealing with inheritance, specificity, and browser quirks, requiring effective organization and documentation.
- **Performance Overhead:** Applying extensive styles or animations with CSS can introduce performance overhead, especially on low-powered devices or older browsers, impacting page rendering and responsiveness.

Python

Introduction:

Python is a high-level, interpreted, general-purpose programming language created by Guido van Rossum and first released in 1991. It emphasizes code readability and allows programmers to express concepts in fewer lines of code compared to other languages like C++ or Java. Python is widely used in web development, automation, data science, machine learning, artificial intelligence, scientific computing, and more.



Features of Python

- Easy to Learn and Use - Python has a simple syntax that mimics natural language, making it easy for beginners to learn.
- Interpreted Language - Python code is executed line-by-line, which makes debugging easier and quicker.
- High-Level Language - Developers do not need to manage memory or hardware-level operations directly.
- Portable - Python programs can run on any platform (Windows, macOS, Linux) without changing the code.

- Extensive Standard Library - Python offers a wide range of built-in libraries for various tasks including file handling, web services, regular expressions, and more.
- Object-Oriented and Functional - Python supports multiple programming paradigms like object-oriented, procedural, and functional programming.
- Large Community Support - Python has a massive developer community, which means access to a vast number of resources and libraries.
- Dynamic Typing - Variables do not require explicit declaration, which enhances flexibility.
- Integrated with Other Languages - Python can work alongside C, C++, and Java via APIs and extensions.
- Support for GUI and Web Development - Frameworks like Tkinter, Django, and Flask allow for easy GUI and web application development.

Advantages of Python

- Easy Syntax - Enhances readability and reduces development time.
- Wide Application - Used in diverse fields such as web dev, data science, AI, IoT, automation.
- Vast Libraries and Frameworks - Speeds up development with pre-written, reusable code.
- Rapid Development - Ideal for prototyping and agile development cycles.
- Open Source - Free to use and distribute, with strong community contributions.
- Cross-platform - Write once, run anywhere – improves deployment flexibility.

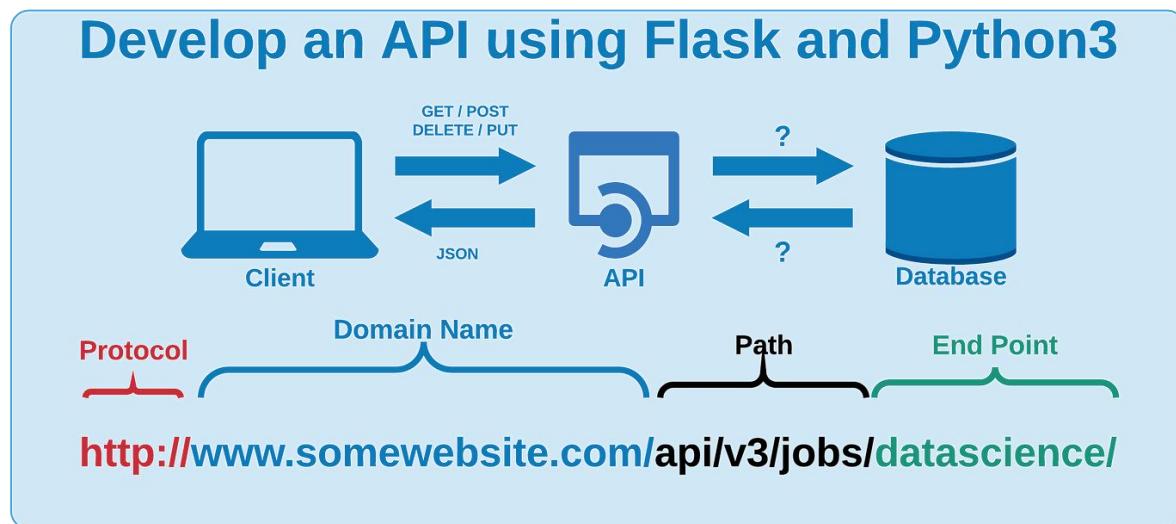
Disadvantages of Python

- Slower Execution - Python is slower than compiled languages like C/C++ due to being interpreted.
- Mobile Development - Less preferred for mobile app development due to limited framework support.
- High Memory Usage - Not suitable for applications requiring high memory efficiency.
- Runtime Errors - Dynamic typing can cause issues that only show up during execution.
- Global Interpreter Lock (GIL) - Prevents multi-threaded CPU-bound tasks from running in parallel efficiently.

Flask

Introduction:

Flask is a lightweight web framework for Python that allows you to build web applications quickly and with minimal code. It's known as a "micro" framework because it provides the essential components for web development while remaining simple and extensible.



Key Features

- **Lightweight and Minimal:** Flask provides only what's necessary, allowing you to add only the extensions you need.
- **Easy to Learn:** With its simple syntax and structure, Flask has a gentle learning curve.
- **Highly Flexible:** You can structure your application however you want.
- **Built-in Development Server:** Comes with a local server for testing during development.
- **RESTful Request Handling:** Simple handling of HTTP requests.
- **Jinja2 Templating:** Powerful template engine for generating HTML.
- **WSGI Compliant:** Works with any WSGI-compatible web server.

Common Flask Extensions

- **Flask-SQLAlchemy:** Database ORM integration
- **Flask-WTF:** Form handling and validation
- **Flask-Login:** User authentication

- **Flask-RESTful**: API building tools
- **Flask-Migrate**: Database migration

Basic Flask Application

```
from flask import Flask      # Create a Flask application instance

app = Flask(__name__)

@app.route('/')  # Define a route and the function to handle it
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':      # Run the application if this file is executed directly
    app.run(debug=True)
```

Flask Components

- **Routes**: Map URLs to functions using decorators
- **Views**: Python functions that handle requests and return responses
- **Templates**: HTML files that can include dynamic content using Jinja2
- **Models**: Python classes that interact with the database (typically using an ORM)
- **Extensions**: Add functionality like form handling, authentication, and database integration

When to Use Flask

Flask is ideal for:

- Smaller web applications
- RESTful APIs
- Prototyping
- Learning web development
- Projects that require flexibility
- Situations where you want more control over components

Advantages

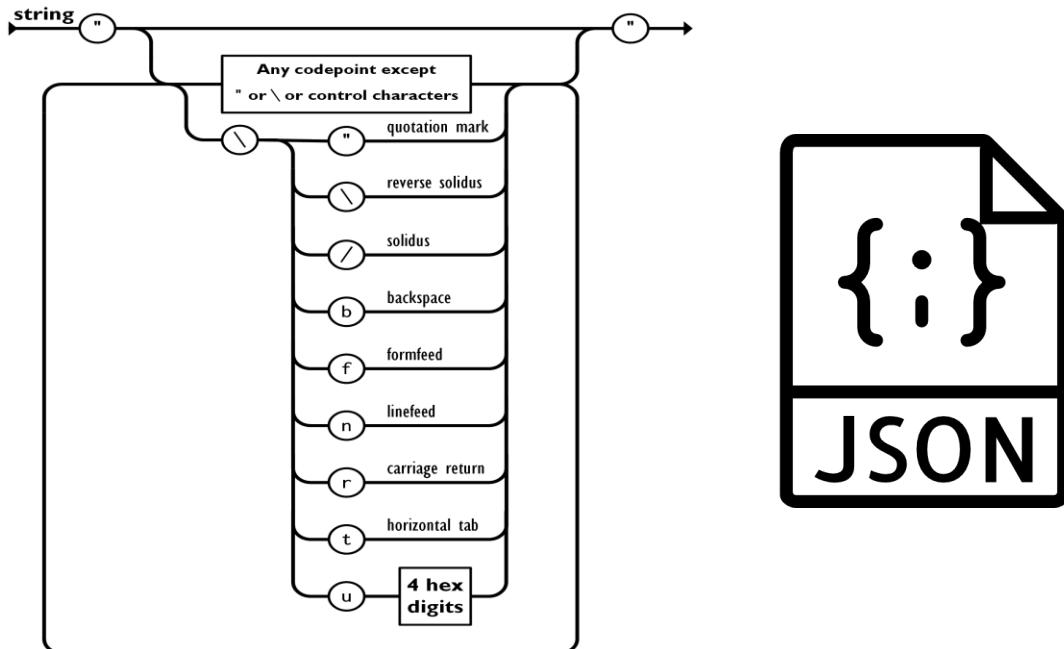
- **Simplicity and Minimalism:** Flask's lightweight design makes it easy to learn and use, with a codebase that's readable and approachable for beginners.
- **Flexibility:** You have complete freedom to structure your application how you want, and you can choose which components to include.
- **Extensibility:** Flask's modular design allows you to add only the extensions you need, keeping your application streamlined.
- **Control:** The framework gives you fine-grained control over every aspect of your application's architecture and behavior.
- **Ideal for Small Projects:** Flask excels for smaller applications, microservices, and APIs where a full-featured framework would be overkill.
- **Built-in Development Server:** Makes local testing and development straightforward without extra configuration.
- **Strong Documentation:** Flask has excellent documentation and a supportive community.
- **Integrated Unit Testing:** Flask includes a test client for easy testing of your application.

Disadvantages

- **Less Structure:** The freedom Flask provides can be a double-edged sword, as it doesn't enforce a specific application structure, potentially leading to disorganized code in larger projects.
- **Manual Configuration:** You need to set up many features manually that might come pre-configured in other frameworks.
- **Decision Fatigue:** You'll need to decide which libraries and extensions to use for common features like forms, authentication, and database access.
- **Scales with Effort:** While Flask can scale to larger applications, doing so requires more planning and architectural decisions compared to frameworks like Django.
- **Less Built-in Security:** Some security features must be implemented manually or through extensions.
- **Smaller Ecosystem:** While growing, Flask's ecosystem of plugins and extensions is smaller than some more established frameworks.
- **No ORM Built-in:** You need to add extensions like SQLAlchemy for database operations.
- **Performance:** As a WSGI application, Flask might not match the raw performance of newer ASGI frameworks in high-concurrency situations.

Introduction to JSON

JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. Despite its origins in JavaScript, JSON is language-independent and is widely used for data storage and exchange across many programming languages.



Key Characteristics of JSON

- **Text-Based Format:** JSON data is stored as plain text
- **Language Independent:** Can be used with virtually any programming language
- **Self-Describing:** Data structure is clear from the format itself
- **Lightweight:** Less verbose than alternatives like XML

JSON Data Types

JSON supports the following data types:

1. **Objects:** Collections of key/value pairs enclosed in curly braces {}
2. **Arrays:** Ordered lists of values enclosed in square brackets []
3. **Strings:** Text enclosed in double quotes ""
4. **Numbers:** Integers or floating-point numbers
5. **Booleans:** true or false
6. **null:** Represents empty or no value

JSON Structure Examples

Basic JSON Object

```
{  
  "name": "John Doe",  
  "age": 30,  
  "isEmployed": true,  
  "address": null  
}
```

JSON Array

```
[  
  "Apple",  
  "Banana",  
  "Orange"  
]
```

Nested JSON

```
{  
  "person": {  
    "name": "Alice",  
    "age": 25,  
    "contacts": {  
      "email": "alice@example.com",  
      "phone": "555-1234"  
    }  
  },  
  "hobbies": ["reading", "hiking", "photography"]  
}
```

Common Uses of JSON

- API responses and requests
- Configuration files
- Data storage
- Cross-platform data exchange
- Web-based data transmission
- NoSQL databases (like MongoDB)

Working with JSON in Different Languages:

JavaScript

```
// Converting object to JSON string
const person = { name: "John", age: 30 };
const jsonString = JSON.stringify(person);

// Converting JSON string to object
const jsonData = '{"name":"John","age":30}';
const personObj = JSON.parse(jsonData);
```

Python

```
import json

# Converting dict to JSON string
person = {"name": "John", "age": 30}
json_string = json.dumps(person)

# Converting JSON string to dict
json_data = '{"name":"John","age":30}'
person_dict = json.loads(json_data)

JSON's simplicity, readability, and wide language support have made it the standard format
for data exchange in modern web applications and APIs.
```

CHAPTER 3

SYSTEM DESIGN

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specific requirements. It involves analyzing the requirements, planning solutions, and detailing how the system will function and interact internally and externally. The goal is to transform user requirements into a technical blueprint that developers and engineers can follow to build the software system effectively.

In the context of this AI-based chatbot project, the system is designed using a layered architecture that integrates both client-side and server-side components. The client layer, built using HTML, CSS, and JavaScript, provides the user interface where users can interact with the chatbot. The backend, developed using Flask (a lightweight Python web framework), handles routing, business logic, and integration with both predefined responses and AI models. This modular architecture ensures a clean separation between logic and presentation layers.

The system architecture follows the client-server model, where user inputs from the frontend are sent to the Flask server via asynchronous requests. The server processes these requests by either matching them with predefined queries stored in a data.json file or generating dynamic responses using a conversational AI model (e.g., DialoGPT). The response is then returned to the client side and displayed in the chat interface. This interaction loop continues for each user message, maintaining a seamless and interactive experience.

Session management is implemented to track user states across interactions, especially after login. For simplicity, a dictionary-based approach is used in this project to manage registered users and sessions. The application is designed in a scalable manner, allowing for future integration of a database such as SQLite or MongoDB.

The codebase is logically divided into multiple files:

- main.py handles route definitions and chatbot logic.
- forms.py is responsible for form validations using Flask-WTF.
- utils.py contains helper functions for tasks like checking predefined responses.
- HTML templates under the templates folder provide dynamic rendering using the Jinja2 template engine.
- CSS and JavaScript are maintained under the static folder to handle styling and dynamic user interactions respectively.

This design ensures ease of maintainability and scalability. Features like user authentication, chatbot interaction, and interface responsiveness are all encapsulated in separate modules, promoting clean and reusable code.

By combining traditional web development tools with AI, this system demonstrates how modern conversational interfaces can be built for educational use cases. The thoughtful

design ensures reliability, scalability, and user-friendliness, which are key goals of this project.

Project Directory Structure

```
AI_Chatbot /  
|  
|   └── static/  
|   |   └── css/  
|   |   |   └── styles.css      # Custom styling for UI  
|   |   └── js/  
|   |   |   └── chatbot.js      # JavaScript for frontend interaction  
|  
|  
|   └── templates/  
|       └── base.html          # Base template with common HTML structure  
|       └── home.html          # Home page after login  
|       └── login.html          # Login page  
|       └── register.html       # Registration page  
|       └── chatbot.html        # Main chatbot interface  
|  
|  
|   └── model/  
|       └── data.json          # Predefined Q&A used by chatbot  
|  
|  
|   └── main.py                # Flask application routes  
|   └── forms.py               # Form definitions using Flask-WTF  
|   └── utils.py                # Utility functions (e.g., checking predefined responses)  
|   └── requirements.txt        # Python package dependencies  
|   └── run.py                  # Script to start the Flask app
```

3.1 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a graphical tool used to represent the flow of data within a system. It illustrates how data moves between external entities, processes, and data stores. For this project, which focuses on an AI-Powered Chatbot Web Application, the DFD provides a clear and concise visualization of how user input flows through the system, how the chatbot processes the input, and how it generates responses.

In this context, the DFD helps in understanding the interaction between users and the various functional modules such as user login, chat interface, data.json response handler, and the AI model (DialoGPT). It effectively captures both predefined static responses from a dataset and dynamic AI-generated responses.

Key Elements in the DFD for This Project:

- User (External Entity): The individual who interacts with the chatbot by logging in and asking queries.
- Login/Register (Process): Handles user authentication and session creation.
- Chatbot Interface (Process): Receives user input and displays responses.
- AI Query Processor (Process): Determines whether to provide a predefined response or to forward the query to the AI model.
- data.json (Data Store): Contains predefined questions and answers for quick responses.
- AI Model (DialoGPT) (Data Store): Generates contextual responses for questions not found in the dataset.

A Data Flow Diagram (DFD) is a graphical representation of the flow of data within a system. It illustrates how data moves between processes, stores, and external entities in a system. DFDs are commonly used in system analysis and design to model the structure and behavior of a system, helping to understand, analyze, and communicate its functionality.

In a DFD, the following elements are typically represented:

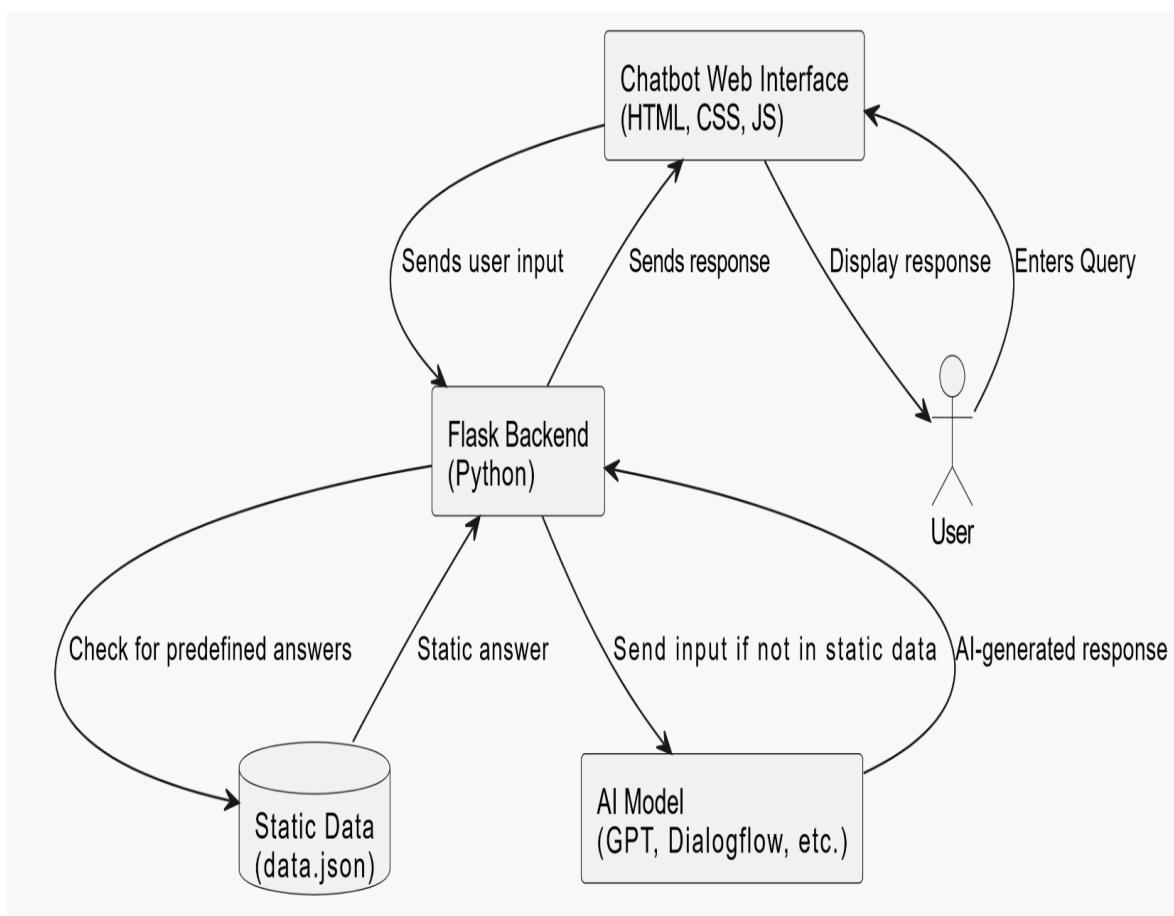
- **Processes:** Processes represent activities or functions performed within the system. They transform input data into output data. Processes are depicted as circles or rectangles in a DFD.
- **Data Flows:** Data flows represent the movement of data between processes, stores, and external entities. They show how data is input to a process, processed, and output from the process. Data flows are depicted as arrows in a DFD.

- **Data Stores:** Data stores represent repositories or storage locations where data is stored within the system. They can be databases, files, or other storage media. Data stores are depicted as rectangles with parallel lines in a DFD.
- **External Entities:** External entities represent sources or destinations of data outside the system being modeled. They interact with the system by providing input data or receiving output data. External entities are depicted as squares or rectangles with rounded corners in a DFD.

DFDs are hierarchical in nature and can be decomposed into multiple levels of detail. The highest level DFD, known as a context diagram, provides an overview of the entire system and its interactions with external entities. Lower-level DFDs provide more detailed views of specific processes or subsystems within the system.

DFDs are valuable tools for system analysis and design because they:

- Provide a visual representation of the system's data flow and structure.
- Aid in identifying data sources, data transformations, and data destinations within the system.
- Help in understanding how data moves through the system and how processes interact with each other.



3.2 USE CASE DIAGRAM

1. Purpose of the Use Case Diagram:

- It visually represents the **interactions** between users (actors) and the chatbot system.
- Helps in identifying **functional requirements** from a user's perspective.
- Defines **what the system does**, not **how it does it**.

2. Actors:

- **User:** The general user who interacts with the chatbot for information, support, and learning.
- **Admin:** Maintains and monitors the chatbot system (e.g., updating responses, managing logs).

3. Main Use Cases for Users:

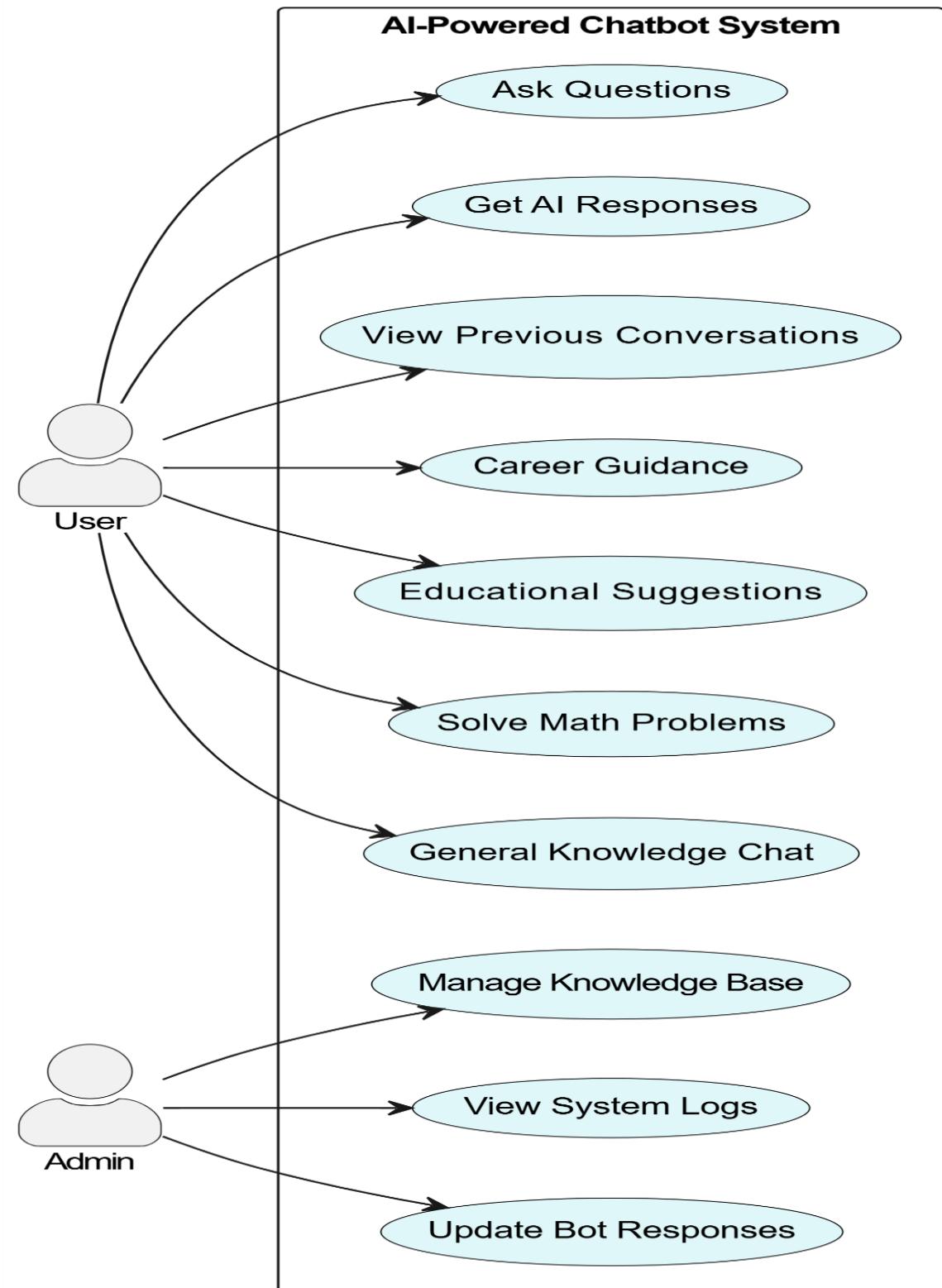
- **Ask Questions:** Users can ask various types of questions.
- **Get AI Responses:** The chatbot replies intelligently using an AI model
- **View Previous Conversations:** Users can see their chat history (if implemented).
- **Career Guidance:** Get personalized advice based on qualifications or interests.
- **Educational Suggestions:** The chatbot provides study tips, science facts.
- **Solve Math Problems:** Helps users solve basic math queries.
- **General Knowledge Chat:** Casual Q&A on topics like geography, space, science,

4. Main Use Cases for Admin:

- **Manage Knowledge Base:** Add/edit predefined questions and answers.
- **View System Logs:** Monitor chatbot activity and performance.
- **Update Bot Responses:** Modify the responses to improve accuracy or tone.

5. System Boundary:

- The "AI-Powered Chatbot System" encapsulates all the use cases.
- It separates what is inside the system (functionalities) from what is external



3.3 FEASIBILITY ANALYSIS

The development of the AI-Powered Conversational Chatbot is highly feasible in terms of technology, operation, cost, and schedule. From a technical perspective, the system is built using reliable and widely used technologies like HTML, CSS, JavaScript for the frontend, and Python Flask for the backend. Integration with the OpenAI GPT API or using a data.json file for static Q&A is straightforward and well-documented. In terms of operational feasibility, the chatbot is designed to be user-friendly and intuitive, enabling users—especially students—to interact with it easily without requiring technical knowledge. It provides instant responses to general knowledge, educational, and career-related queries. The economic feasibility of the system is also favorable. It utilizes open-source tools and free APIs (or cost-effective paid tiers), making the overall development cost minimal. Additionally, maintenance costs are low, since the chatbot does not require frequent updates if based on a static dataset. Regarding legal feasibility, the chatbot adheres to general legal standards of web development and respects the terms of service of third-party tools like OpenAI's API. Lastly, the schedule feasibility is strong, as the project can be completed within a short development cycle of approximately 2 to 4 weeks, with clearly divided modules and milestones.

1. Technical Feasibility

- Uses known technologies: HTML, CSS, JS, Flask (Python).
- GPT API or static JSON data easily integrated.
- No complex hardware required; works on most browsers.

2. Operational Feasibility

- Simple and user-friendly chatbot interface.
- Can handle a wide range of student/general queries.
- Requires minimal training for users.

3. Economic Feasibility

- Low development cost using free/open-source tools.
- GPT API has free-tier or pay-as-you-go options.

3.4 ENTITY RELATIONSHIP DIAGRAM

An Entity-Relationship Diagram (ERD) is a visual representation of the entities (objects or concepts) within a system and the relationships between them. It is a modeling technique used in database design to illustrate the logical structure of a database. ERDs are commonly used during the early stages of database development to understand the data requirements and relationships in a system.

In an ERD:

- **Entity:** An entity represents a real-world object or concept, such as a person, place, thing, event, or concept. In the context of a database, entities are typically represented as tables. Each entity has attributes that describe its properties or characteristics. Attributes are represented as columns in the entity table.
- **Relationship:** A relationship describes the connection or association between two or more entities. It represents how entities are related to each other. Relationships can be one-to-one, one-to-many, or many-to-many, depending on the cardinality of the relationship. Relationships are represented as lines connecting the related entities, with labels indicating the nature of the relationship.
- **Cardinality:** Cardinality defines the number of instances of one entity that can be associated with another entity through a relationship. Cardinality is expressed using symbols such as "1" for one instance, "N" for many instances, or "0..1" for zero or one instance.

ERDs are commonly used in the following ways:

- **Database Design:** ERDs are used to design the structure of a database by defining entities, attributes, and relationships.
- **Communication:** ERDs serve as a communication tool between stakeholders, such as developers, designers, and clients, to understand the data requirements and relationships in a system.
- **Documentation:** ERDs document the logical structure of a database, providing a visual representation of its entities and relationships for future reference and maintenance.

Entity-Relationship Diagram (ERD) for AI Powered chatbot

Description:

The system primarily consists of three main entities: **User**, **Message**, and **Response**. Each entity is designed to serve a specific purpose and is interconnected through well-defined relationships.

User Entity

The User entity stores details about each individual who registers and interacts with the chatbot. This includes their unique ID, username, encrypted password, and account creation

timestamp. Each user can have multiple associated messages, making it a one-to-many relationship with the Message entity.

Message Entity

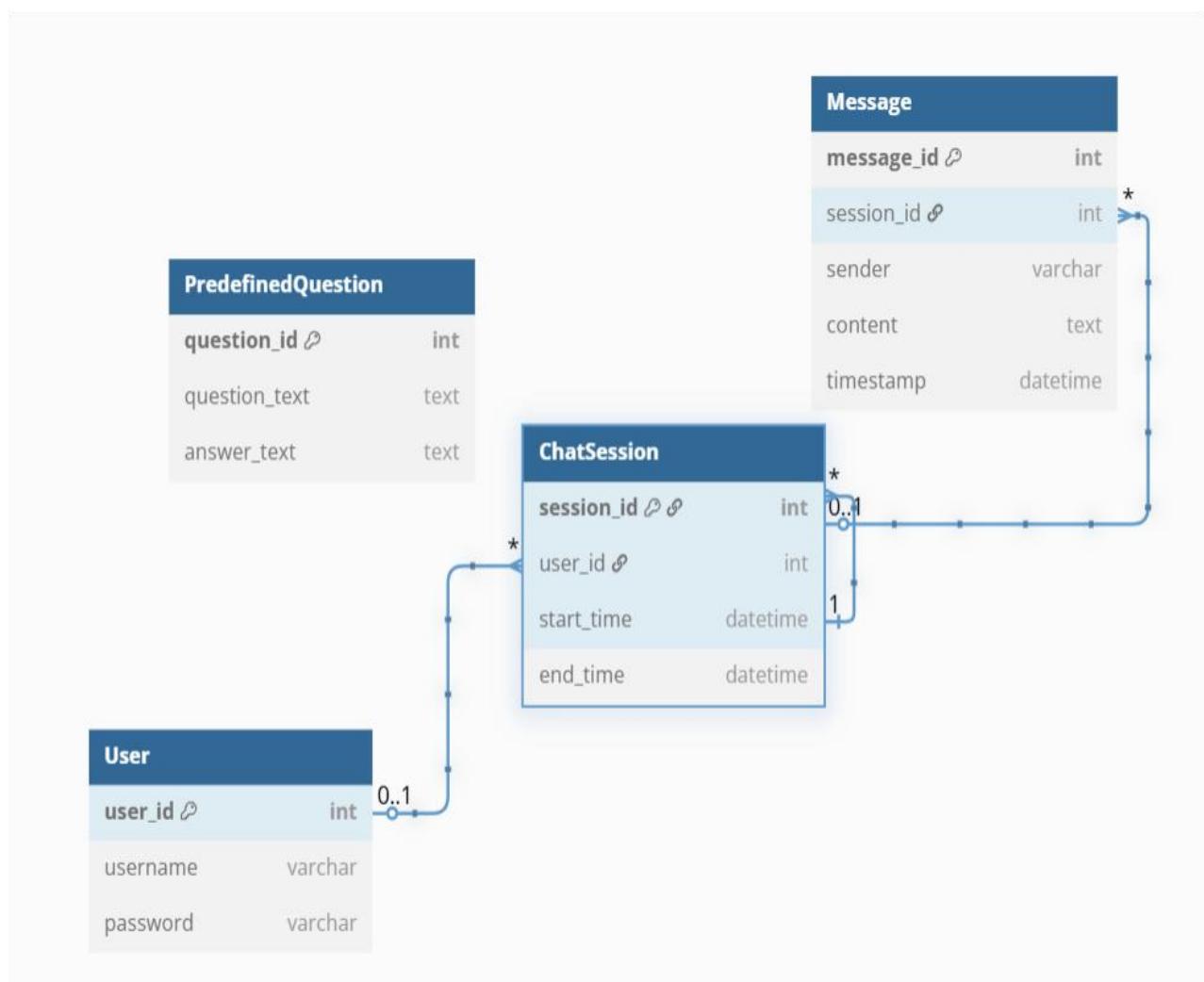
The Message entity holds the textual input sent by users during a chatbot session. Every message is uniquely identified and associated with the user_id of the sender. The message also contains a timestamp indicating when the input was submitted. Each message has a corresponding response, leading to a one-to-one relationship with the Response entity.

Response Entity

The Response entity records the chatbot's answer to a given message. It includes fields such as the chatbot's response text, a timestamp of generation, and a source_type field to indicate whether the response came from the predefined data.json or was generated dynamically using the AI model (DialoGPT). Each response is linked to one message.

Relationships Overview

- A User can send **multiple messages**. (One-to-Many)
- A Message generates **one response**. (One-to-One)
- Each Response corresponds to **one message**. (One-to-One)



CHAPTER 4

SYSTEM IMPLEMENTATION

4.1 Overview

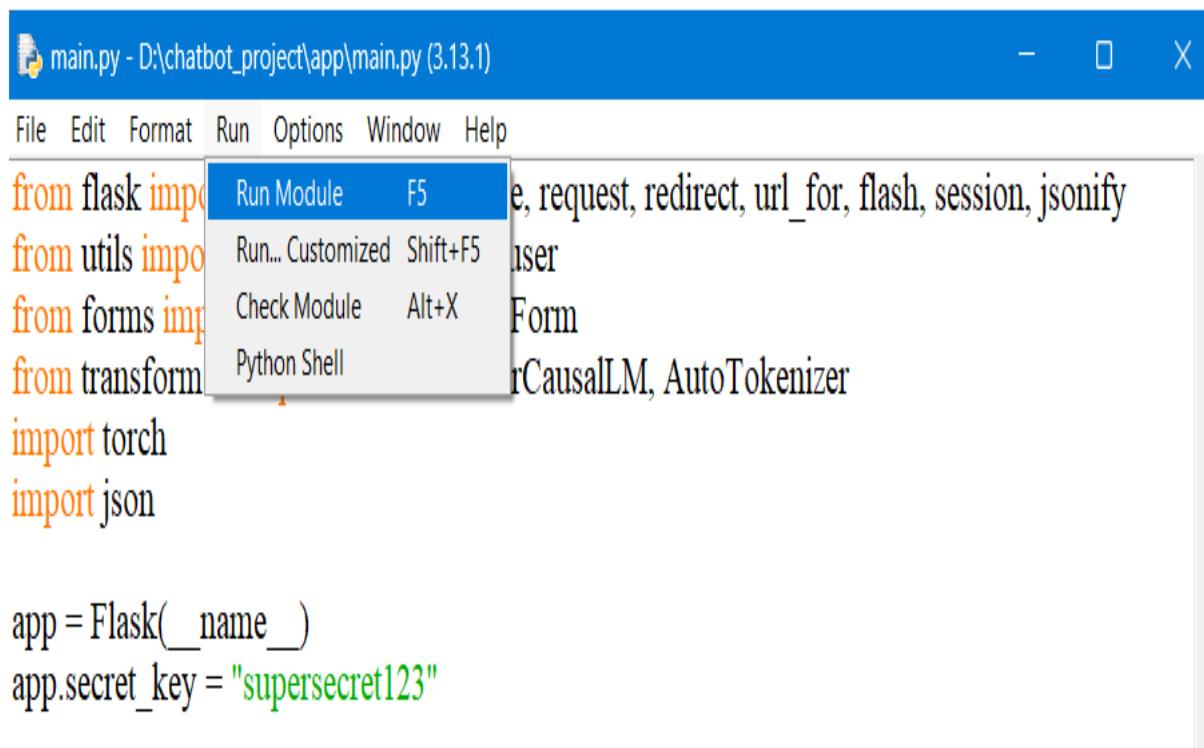
The system implementation focuses on transforming the design into a fully functional web application. It includes frontend development for user interaction, backend logic using Flask for authentication and chatbot interaction, and integration of predefined questions using a JSON file. The chatbot is capable of handling general queries and career-related guidance.

System implementation refers to the phase in the software development life cycle (SDLC) where the designed system is built, tested, and deployed to users. It involves translating the system specifications and design into a working and functional system that meets the requirements of stakeholders.

System implementation typically includes the following activities:

- **Coding/Development:** During this phase, developers write the code for the system based on the design specifications. This involves programming in the chosen programming languages and technologies, such as Java for a web application or C++ for a desktop application.
- **Integration and Testing:** Once the individual components of the system are developed, they are integrated to form the complete system. Integration testing is performed to ensure that the integrated components work together as expected and meet the requirements. Testing involves various levels, including unit testing (testing individual components), integration testing (testing the integration of components), and system testing (testing the entire system).
- **Debugging and Bug Fixing:** During testing, defects and bugs in the system are identified. Developers debug the system to find the root cause of the issues and fix them. This may involve modifying the code, adjusting configurations, or making changes to the design.
- **Deployment:** Once the system is developed and tested, it is deployed to the production environment. This involves installing the system on servers, configuring it for production use, and making it available to users. Deployment may also involve data migration, where existing data is transferred to the new system, and user training, where users are trained on how to use the new system effectively.
- **User Acceptance Testing (UAT):** After deployment, users conduct user acceptance testing to validate that the system meets their requirements and expectations. Any issues identified during UAT are addressed before the system is officially launched.
- **Go-Live:** Once the system has passed user acceptance testing and all issues are resolved, it is officially launched and made available for use by all users. This marks the completion of the system implementation phase.
- **Post-Implementation Support:** After the system is deployed, developers provide post-implementation support to address any issues that may arise and to ensure the system operates smoothly. This may involve providing technical support, troubleshooting problems, and implementing enhancements or updates.

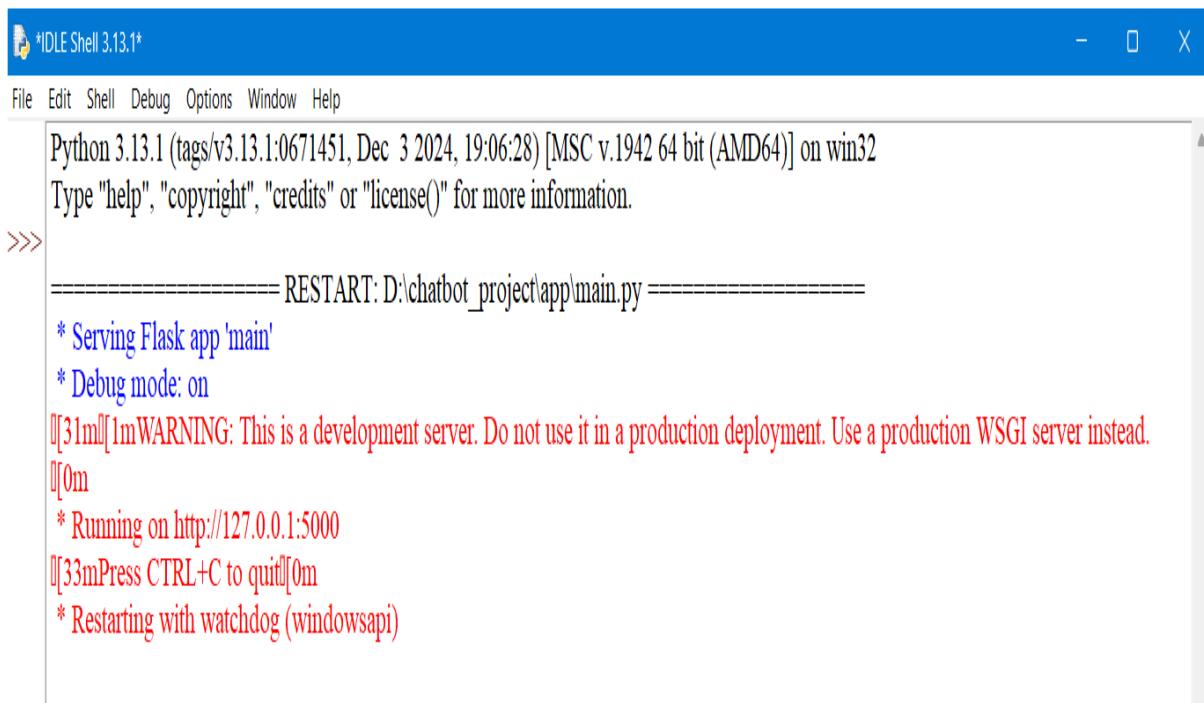
4.1 SCREENSHOTS OF WEB APPLICATION



```
from flask import *  
from utils import *  
from forms import *  
from transformers import *  
import torch  
import json
```

app = Flask(__name__)
app.secret_key = "supersecret123"

Fig 4.1.1 Run Main python file



```
Python 3.13.1 (tags/v3.13.1:0671451, Dec 3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: D:\chatbot_project\app\main.py =====  
* Serving Flask app 'main'  
* Debug mode: on  
[31m[1mWARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.[0m  
[0m  
* Running on http://127.0.0.1:5000  
[33mPress CTRL+C to quit[0m  
* Restarting with watchdog (windowsapi)
```

Fig 4.1.2 After Run

<http://127.0.0.1:5000/login>

```
[31m[1mWARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.[0m
* Running on http://127.0.0.1:5000
[33mPress CTRL+C to quit[0m
127.0.0.1 - - [13/Apr/2025 10:31:22] "GET / HTTP/1.1[0m" 302 -
127.0.0.1 - - [13/Apr/2025 10:31:22] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [13/Apr/2025 10:31:23] "GET /static/css/style.css HTTP/1.1[0m" 304 -
127.0.0.1 - - [13/Apr/2025 10:31:23] "GET /static/js/script.js HTTP/1.1[0m" 304 -
127.0.0.1 - - [13/Apr/2025 10:31:24] "GET /favicon.ico HTTP/1.1[0m" 404 -
```

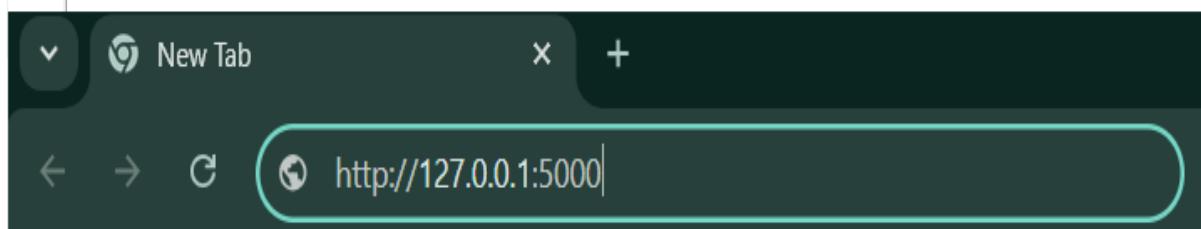


Fig 4.1.3 Run Flask App

A screenshot of a login page. The page has a header with the word "Login". Below the header, there are two input fields: one for "Username" and one for "Password", both represented by empty text boxes. Below the password field is a "Login" button, which is a rectangular button with rounded corners and a light gray background. At the bottom of the page, there is a link "Don't have an account? [Register here](#)".

Fig 4.1.4 Login Page

Register

Username:

Password:

Register

Already have an account? [Login here](#)

Fig 4.1.5 Register Page

Registration successful! Please log in.

Login

Username:

smarti

Password:

....

Login

Don't have an account? [Register here](#)

Fig 4.1.6 Login Page

AI Chatbot

Recommended Questions

- What is AI
- what is the capital of india
- how many languages are there in india
- who is the prime minister of india
- how does an ats scan my resume
- tell me a riddle
- what is the tallest mountain
- how far is the sun from earth
- what is 5 plus 3

Type your message... Send

AI Chatbot

Recommended Questions

- What is AI
- what is the capital of india
- how many languages are there in india
- who is the prime minister of india
- how does an ats scan my resume
- tell me a riddle
- what is the tallest mountain
- how far is the sun from earth
- what is 5 plus 3

You: hello
Bot: Hi there! How can I help you?
You: who are you
Bot: I am an AI chatbot created to assist you.
You: what is your name
Bot: I'm a chatbot, you can call me ChatAI!
You: how long should my resume be
Bot: Ideally, one page for freshers and two pages for experienced professionals.
You: what is the capital of india
Bot: The capital of India is New Delhi.
You: what is the capital of australia
Bot: The capital of Australia is Canberra.
You: what is 12 times 6
Bot: 12 times 6 equals 72.
You: what is 5 plus 3
Bot: 5 plus 3 equals 8.
You: which ocean is the deepest
Bot: The Pacific Ocean is the deepest ocean on Earth.
You: how many languages are there in india
Bot: India has 22 officially recognized languages and hundreds of regional languages.
You: what is gravity
Bot: Gravity is a force that attracts two objects with mass toward each other.
You: what is photosynthesis

what is the national bird of india Send

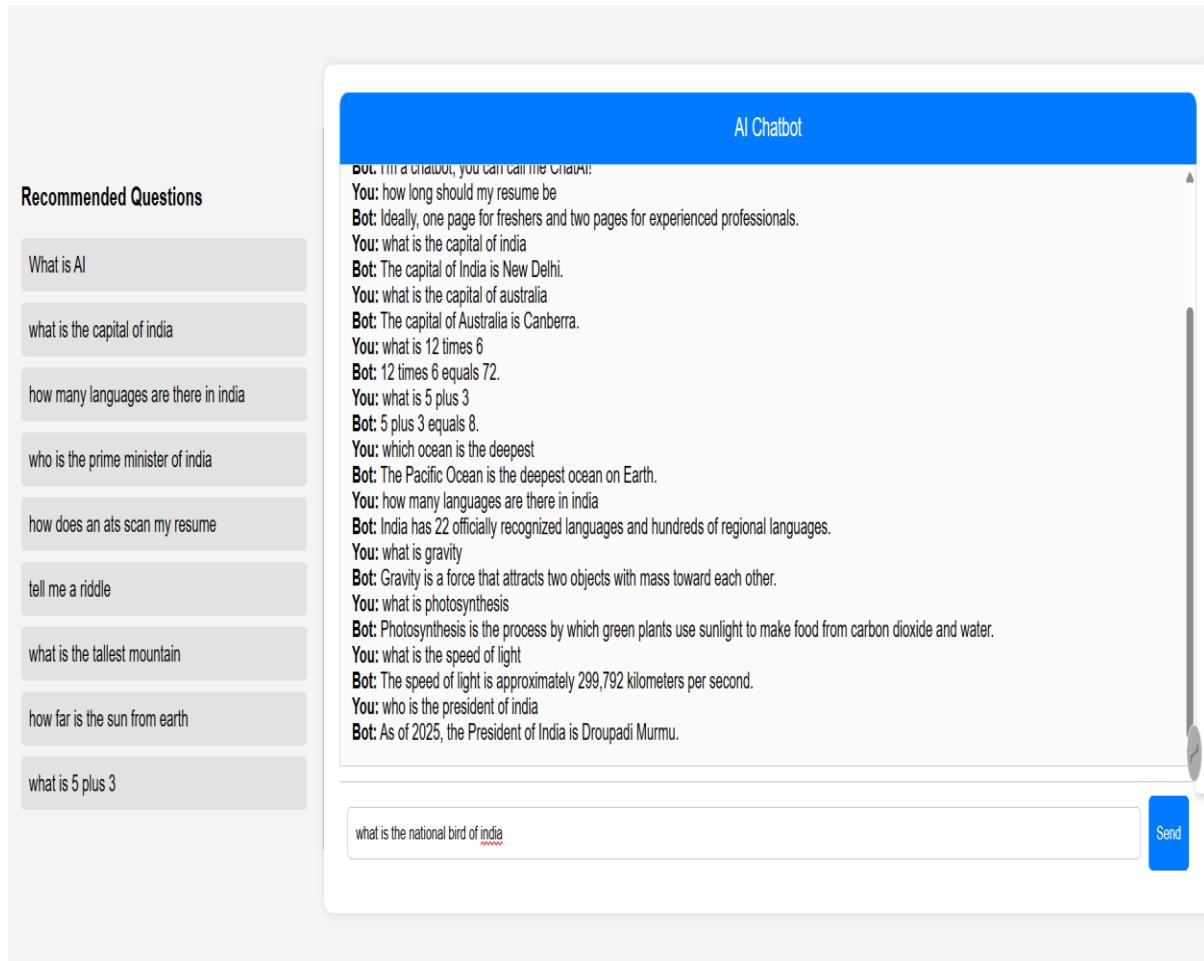


Fig 4.1.7 Chat-Bot Demo Page

4.2 SCREENSHOTS OF WEB APPLICATION (Coding Part)



The screenshot shows a code editor window with a blue header bar. The title bar reads "main.py - D:\chatbot_project\app\main.py (3.13.1)*". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The main code area contains Python code for a Flask application. It imports various modules from Flask, utils, forms, and transformers. It defines a Flask app with a secret key and loads a DialogPT model and tokenizer. A function 'load_responses()' reads responses from a JSON file. The application routes include a root route that redirects to the login page, and a login route that handles both GET and POST requests. For POST requests, it checks user credentials against a database and logs them in if successful, or flashes an error message if they are invalid. For GET requests, it simply redirects to the login page.

```
from flask import Flask, render_template, request, redirect, url_for, flash, session, jsonify
from utils import check_login, register_user
from forms import LoginForm, RegisterForm
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
import json

app = Flask(__name__)
app.secret_key = "supersecret123"
# Load DialogPT model & tokenizer
tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-medium")
model = AutoModelForCausalLM.from_pretrained("microsoft/DialoGPT-medium")
# Load predefined responses
def load_responses():
    with open("data.json", "r") as file:
        return json.load(file)
responses = load_responses()
conversation_history = []

@app.route('/')
def index():
    return redirect(url_for('login'))

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if request.method == 'POST' and form.validate_on_submit():
        username = form.username.data
        password = form.password.data
        if check_login(username, password):
            session['user'] = username
            return redirect(url_for('chat_page')) # ⚡ Redirect to chatbot page directly
        else:
            flash("Invalid credentials. Try again.")
            return redirect(url_for('login'))
    return render_template('login.html', form=form)
```

```
*main.py - D:\chatbot_project\app\main.py (3.13.1)*
File Edit Format Run Options Window Help
@app.route('/logout')
def logout():
    session.pop('user', None)
    return redirect(url_for('login'))

@app.route('/chat')
def chat_page():
    if 'user' in session:
        return render_template("chat.html", user=session['user']) # ⚡ Pass user to template
    return redirect(url_for('login'))

@app.route("/get", methods=["POST"])
def chat():
    global conversation_history
    try:
        user_input = request.form.get("msg", "").strip()
        if not user_input:
            return jsonify({"response": "Please type something!"})

        if user_input.lower() in responses:
            return jsonify({"response": responses[user_input.lower()]})

        conversation_history.append(f"User: {user_input}")

        chat_prompt = "\n".join(conversation_history) + "\nAI:"
        input_ids = tokenizer.encode(chat_prompt, return_tensors="pt")

        output_ids = model.generate(
            input_ids,
            max_length=1000,
            pad_token_id=tokenizer.eos_token_id,
            do_sample=True,
            top_k=50,
            top_p=0.95,
            temperature=0.7,
        )

        response = tokenizer.decode(output_ids[:, input_ids.shape[-1]:][0], skip_special_tokens=True)
        conversation_history.append(f"AI: {response}")

        if len(conversation_history) > 12:
            conversation_history = conversation_history[-6:]

        return jsonify({"response": response})

    except Exception as e:
        return jsonify({"response": f"Error: {str(e)}"})

if __name__ == '__main__':
    app.run(debug=True)
```

```
utils.py - D:\chatbot_project\app\utils.py (3.13.1)
File Edit Format Run Options Window Help
users = {} # For demo purposes, using an in-memory dictionary

def check_login(username, password):
    return users.get(username) == password

def register_user(username, password):
    if username in users:
        return False # Username already exists
    users[username] = password
    return True
```

```
forms.py - D:\chatbot_project\app\forms.py (3.13.1)
File Edit Format Run Options Window Help
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')

class RegisterForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Register')
```

D:\chatbot_project\app\templates\base.html (app) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

base.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>{% block title %}Chat App{% endblock %}</title>
6     <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
7 </head>
8 <body>
9     <div class="container">
10        {% with messages = get_flashed_messages() %}
11            {% if messages %}
12                {% for message in messages %}
13                    <p class="flash">{{ message }}</p>
14                {% endfor %}
15            {% endif %}
16            {% endwith %}
17        {% block content %}{% endblock %}
18    </div>
19    <script src="{{ url_for('static', filename='js/script.js') }}"></script>
20 </body>
21 </html>
22
```

D:\chatbot_project\app\templates\login.html • (app) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

login.html

```
1 {% extends 'base.html' %}
2 {% block title %}Login{% endblock %}
3 {% block content %}
4 <h2>Login</h2>
5 <form method="POST">
6     {{ form.hidden_tag() }}
7     <label>Username:</label><br>
8     {{ form.username() }}<br>
9     <label>Password:</label><br>
10    {{ form.password() }}<br><br>
11    {{ form.submit() }}
12 </form>
13 <p>Don't have an account? <a href="{{ url_for('register') }}">Register here</a></p>
14 {% endblock %}
15
```

D:\chatbot_project\app\templates\home.html (app) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

home.html

```
1  {% extends 'base.html' %} 
2  {% block title %}Home{% endblock %} 
3  {% block content %} 
4  <h2>Welcome, {{ user }}!</h2> 
5 
6  <div class="chat-container"> 
7      <div class="chat-header">AI Chatbot</div> 
8      <div id="chatbox" class="chat-box"></div> 
9 
10     <div class="input-container"> 
11         <input type="text" id="userInput" placeholder="Type your message..." 
12             autocomplete="off"> 
13 
14         <button onclick="sendMessage()">Send</button> 
15     </div> 
16 
17     <a href="{{ url_for('logout') }}">Logout</a> 
18     <script> 
19         function sendMessage() { 
20             let userText = document.getElementById("userInput").value.trim(); 
21             if (userText === "") return; 
22 
23             let chatbox = document.getElementById("chatbox"); 
24 
25             let userMessage = `<div class='user-message'>${userText}</div>`; 
26             chatbox.innerHTML += userMessage; 
27             document.getElementById("userInput").value = ""; 
28 
29             chatbox.innerHTML += `<div class='bot-message'>Thinking...</div>`; 
30             chatbox.scrollTop = chatbox.scrollHeight; 
31 
32             fetch("/get", { 
33                 method: "POST", 
34                 body: new URLSearchParams({ "msg": userText }), 
35                 headers: { "Content-Type": "application/x-www-form-urlencoded" } 
36             }) 
37             .then(response => response.json()) 
38             .then(data => { 
39                 document.querySelector(".bot-message:last-child").innerHTML = data. 
39                     response; 
40                 chatbox.scrollTop = chatbox.scrollHeight; 
41             }); 
42         } 
43     </script> 
44     {% endblock %}
```

Line 1, Column 1

Spaces: 4

HTML

D:\chatbot_project\app\templates\register.html (app) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

register.html

```
1  {% extends 'base.html' %} 
2  {% block title %}Register{% endblock %} 
3  {% block content %} 
4  <h2>Register</h2> 
5  <form method="POST"> 
6      {{ form.hidden_tag() }} 
7      <label>Username:</label><br> 
8      {{ form.username() }}<br> 
9      <label>Password:</label><br> 
10     {{ form.password() }}<br><br> 
11     {{ form.submit() }} 
12 </form> 
13 <p>Already have an account? <a href="{{ url_for('login') }}">Login here</a></p> 
14     {% endblock %}
```

```
D:\chatbot_project\app\templates\chat.html (app) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
chat.html x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Smarti's Chatbot</title>
6     <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}>
7     <style>
8         body {
9             font-family: Arial, sans-serif;
10            display: flex;
11            height: 100vh;
12            margin: 0;
13        }
14        .sidebar {
15            width: 25%;
16            background-color: #f4f4f4;
17            padding: 20px;
18            border-right: 2px solid #ccc;
19        }
20        .chat-container {
21            width: 75%;
22            padding: 20px;
23            display: flex;
24            flex-direction: column;
25        }
26        #chat-box {
27            flex-grow: 1;
28            overflow-y: auto;
29            border: 1px solid #ccc;
30            padding: 15px;
31            margin-bottom: 10px;
32        }
33        #user-input {
34            display: flex;
35        }
36        #user-input input {
37            flex: 1;
38            padding: 10px;
39        }
40        #user-input button {
41            padding: 10px;
42        }
43        .recommended-question {
44            background-color: #e2e2e2;
45            margin: 8px 0;
46            padding: 10px;
47            border-radius: 5px;

```

```
Line 1, Column 1
Spaces: 4
HTML
D:\chatbot_project\app\templates\chat.html (app) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
chat.html x
48             cursor: pointer;
49     }
50     .recommended-question:hover {
51         background-color: #d0d0d0;
52     }
53 </style>
54 </head>
55 <body>
56     <div class="sidebar">
57         Recommended Questions</h3>
58         <div class="recommended-question">What is AI</div>
59         <div class="recommended-question">what is the capital of india</div>
60         <div class="recommended-question">how many languages are there in india</div>
61         <div class="recommended-question">who is the prime minister of india</div>
62         <div class="recommended-question">how does an ats scan my resume</div>
63         <div class="recommended-question">tell me a riddle</div>
64         <div class="recommended-question">what is the tallest mountain</div>
65         <div class="recommended-question">how far is the sun from earth</div>
66         <div class="recommended-question">what is 5 plus 3</div>
67     </div>
68
69     <div class="chat-container">
70         <div class="chat-header">AI Chatbot</div>
71         <div id="chatbox" class="chat-box"></div>
72
73         <div class="input-container">
74             <input type="text" id="userInput" placeholder="Type your message..." 
75             autocomplete="off">
76             <button id="sendBtn">Send</button>
77         </div>
78     </div>
79
80     <script>
81         window.sendMessage = sendMessage;
82
83         document.getElementById("sendBtn").addEventListener("click", sendMessage);
84         document.getElementById("userInput").addEventListener("keypress", function (e) {
85             if (e.key === "Enter") sendMessage();
86         });
87     </script>
```

D:\chatbot_project\app\templates\chat.html • (app) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

chat.html

```
function sendMessage() {
    const messageInput = document.getElementById("userInput");
    const chatBox = document.getElementById("chatbox");
    const userMessage = messageInput.value.trim();

    if (!userMessage) return;

    // Add user message
    chatBox.innerHTML += `<div><strong>You:</strong> ${userMessage}</div>`;
    messageInput.value = "";

    // Send to backend
    fetch("/get", {
        method: "POST",
        headers: {
            "Content-Type": "application/x-www-form-urlencoded",
        },
        body: `msg=${encodeURIComponent(userMessage)}`
    })
    .then(response => response.json())
    .then(data => {
        chatBox.innerHTML += `<div><strong>Bot:</strong> ${data.response}</div>`;
        chatBox.scrollTop = chatBox.scrollHeight;
    })
    .catch(error => {
        console.error("Fetch error:", error);
    });
}

document.getElementById("userInput").addEventListener("keypress", function (e) {
if (e.key === "Enter") {
    sendMessage();
}
});

document.querySelectorAll('.recommended-question').forEach(item => {
    item.addEventListener('click', () => {
        document.getElementById('message').value = item.innerText;
        sendMessage(); // Make sure this triggers your function
    });
});

</script>
<script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
</html>
```

Line 131, Column 73

Spaces: 4

HTML

D:\chatbot_project\app\static\css\style.css • (static) - Sublime Text (UNREGISTERED)

```
1 body {
2     font-family: Arial, sans-serif;
3     background-color: #f2f2f2;
4 }
5
6 .container {
7     width: 500px;
8     margin: 50px auto;
9     background: white;
10    padding: 20px;
11    border-radius: 8px;
12    box-shadow: 0 0 10px #ccc;
13 }
14
15 input[type="text"], input[type="password"] {
16     width: 95%;
17     padding: 10px;
18     margin: 8px 0;
19     box-sizing: border-box;
20 }
21
22 .chat-box {
23     height: 300px;
24     border: 1px solid #ccc;
25     padding: 10px;
26     overflow-y: auto;
27     margin-bottom: 10px;
28     background: #fafafa;
29 }
30
31 .flash {
32     background-color: #ffe0e0;
33     padding: 10px;
34     border: 1px solid #ff0000;
35     margin-bottom: 10px;
36     color: #d8000c;
37 }
38
39 .chat-container {
40     width: 400px;
41     background: white;
42     border-radius: 10px;
43     box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
44 }
45
46 .chat-header {
47     background: #007bff;
48     color: white;
49     padding: 15px;
50     text-align: center;
51     font-size: 18px;
52     border-radius: 10px 10px 0 0;
53 }
54
55 .chat-box {
56     height: 400px;
57     overflow-y: auto;
58     padding: 15px;
59     display: flex;
60     flex-direction: column;
61 }
62
63 .user-message {
64     background: #007bff;
65     color: white;
66     padding: 10px;
67     border-radius: 10px;
68     align-self: flex-end;
69     margin-bottom: 10px;
70 }
71
72 .bot-message {
73     background: #e0e0e0;
74     color: black;
75     padding: 10px;
76     border-radius: 10px;
77     align-self: flex-start;
78     margin-bottom: 10px;
79 }
80
81 .input-container {
82     display: flex;
83     padding: 10px;
84     border-top: 1px solid #ccc;
85 }
86
87 input {
88     flex: 1;
89     padding: 10px;
90     border: 1px solid #ccc;
91     border-radius: 5px;
92 }
93
94 button {
95     background: #007bff;
96     color: white;
97     border: none;
98     padding: 10px;
99     border-radius: 5px;
100    margin-left: 10px;
101 }
```

D:\chatbot_project\app\static\css\style.css • (static) - Sublime Text (UNREGISTERED)

```
46 .chat-header {
47     background: #007bff;
48     color: white;
49     padding: 15px;
50     text-align: center;
51     font-size: 18px;
52     border-radius: 10px 10px 0 0;
53 }
54
55 .chat-box {
56     height: 400px;
57     overflow-y: auto;
58     padding: 15px;
59     display: flex;
60     flex-direction: column;
61 }
62
63 .user-message {
64     background: #007bff;
65     color: white;
66     padding: 10px;
67     border-radius: 10px;
68     align-self: flex-end;
69     margin-bottom: 10px;
70 }
71
72 .bot-message {
73     background: #e0e0e0;
74     color: black;
75     padding: 10px;
76     border-radius: 10px;
77     align-self: flex-start;
78     margin-bottom: 10px;
79 }
80
81 .input-container {
82     display: flex;
83     padding: 10px;
84     border-top: 1px solid #ccc;
85 }
86
87 input {
88     flex: 1;
89     padding: 10px;
90     border: 1px solid #ccc;
91     border-radius: 5px;
92 }
93
94 button {
95     background: #007bff;
96     color: white;
97     border: none;
98     padding: 10px;
99     border-radius: 5px;
100    margin-left: 10px;
101 }
```

D:\chatbot_project\app\static\css\style.css • (static) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

style.css

```
79 }
80
81 .input-container {
82     display: flex;
83     padding: 10px;
84     border-top: 1px solid #ccc;
85 }
86
87 input {
88     flex: 1;
89     padding: 10px;
90     border: 1px solid #ccc;
91     border-radius: 5px;
92 }
93
94 button {
95     background: #007bff;
96     color: white;
97     border: none;
98     padding: 10px;
99     border-radius: 5px;
100    margin-left: 10px;
101    cursor: pointer;
102 }
103
104 .custom-button {
105     padding: 12px 24px;
106     background-color: #4CAF50; /* Green */
107     color: white;
108     border: none;
109     border-radius: 8px;
110     cursor: pointer;
111     font-size: 16px;
112     font-weight: bold;
113     transition: background-color 0.3s ease, transform 0.2s ease;
114     box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
115 }
116
117 .custom-button:hover {
118     background-color: #45a049;
119     transform: scale(1.05);
120 }
121
122 .custom-button:active {
123     transform: scale(0.95);
124     background-color: #3e8e41;
125 }
```

Line 122, Column 1 Spaces: 4 CSS

D:\chatbot_project\app\static\js\script.js (static) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

script.js

```
1 function sendMessage() {
2     const input = document.getElementById("user-input");
3     const message = input.value.trim();
4     if (message === "") return;
5
6     const chatBox = document.getElementById("chat-box");
7     chatBox.innerHTML += "<p><strong>You:</strong> " + message + "</p>";
8     input.value = "";
9
10    // DEBUG: See if JS is working
11    console.log("Sending message to backend:", message);
12
13    fetch("/get", {
14        method: "POST",
15        headers: {
16            "Content-Type": "application/x-www-form-urlencoded"
17        },
18        body: "msg=" + encodeURIComponent(message),
19    })
20    .then(response => response.json())
21    .then(data => {
22        console.log("Response from backend:", data); // DEBUG
23        chatBox.innerHTML += "<p><strong>AI:</strong> " + data.response + "</p>";
24        chatBox.scrollTop = chatBox.scrollHeight;
25    })
26    .catch(err => {
27        chatBox.innerHTML += "<p><em>Error talking to server</em></p>";
28        console.error("Fetch error:", err); // DEBUG
29    });
30}
31
32 function sendMessage() {
33     const messageInput = document.getElementById("message");
34     const chatBox = document.getElementById("chat-box");
35     const userMessage = messageInput.value.trim();
36
37     if (!userMessage) return;
38
39     // Show user message
40     chatBox.innerHTML += `<div><strong>You:</strong> ${userMessage}</div>`;
41     messageInput.value = "";
42
43     // Send to backend
44     fetch("/get", {
45         method: "POST",
46         headers: {
47             "Content-Type": "application/x-www-form-urlencoded",
48         },
49         body: `msg=${encodeURIComponent(userMessage)}`
50     })
51     .then(response => response.json())
52     .then(data => {
53         chatBox.innerHTML += `<div><strong>Bot:</strong> ${data.response}</div>`;
54         chatBox.scrollTop = chatBox.scrollHeight; // Auto-scroll
55     })
56     .catch(error => {
57         console.error("Error:", error);
58    });
59}
60
```

D:\chatbot_project\app\static\js\script.js (static) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

script.js

```
29 }
30
31
32 function sendMessage() {
33     const messageInput = document.getElementById("message");
34     const chatBox = document.getElementById("chat-box");
35     const userMessage = messageInput.value.trim();
36
37     if (!userMessage) return;
38
39     // Show user message
40     chatBox.innerHTML += `<div><strong>You:</strong> ${userMessage}</div>`;
41     messageInput.value = "";
42
43     // Send to backend
44     fetch("/get", {
45         method: "POST",
46         headers: {
47             "Content-Type": "application/x-www-form-urlencoded",
48         },
49         body: `msg=${encodeURIComponent(userMessage)}`
50     })
51     .then(response => response.json())
52     .then(data => {
53         chatBox.innerHTML += `<div><strong>Bot:</strong> ${data.response}</div>`;
54         chatBox.scrollTop = chatBox.scrollHeight; // Auto-scroll
55     })
56     .catch(error => {
57         console.error("Error:", error);
58    });
59}
60
```

4.3 Software Testing

Software testing is a critical phase in the software development life cycle (SDLC) that ensures the functionality, reliability, and quality of the application. For this AI-based chatbot web application, testing was conducted at multiple levels to verify that all components of the system function correctly, independently, and when integrated together.

Software Testing is the process of evaluating and verifying that a software product or application does what it is intended to do. It plays a crucial role in ensuring software quality, reliability, and user satisfaction. The goal of testing is not only to find defects but also to validate that the software behaves as expected under various conditions.

Software testing helps identify errors, gaps, or missing requirements in contrast to the actual requirements. A well-tested application is more robust, secure, user-friendly, and efficient, which directly contributes to the success and trustworthiness of the system.

Objectives of Software Testing

- To ensure that the software meets the specified requirements.
- To verify that the application functions correctly across various platforms and scenarios.
- To identify and fix bugs before the system is deployed.
- To improve performance and usability by providing feedback for enhancements.
- To validate the reliability and stability of the system under different conditions.

Types of Testing Performed

1. Unit Testing

Unit testing involves testing individual components or functions of the application in isolation. In this project, unit tests were applied to:

- Form validation (login and registration)
- Chat response matching from data.json
- Utility functions for input sanitization

Tools Used: Python's built-in unittest module and manual test cases

2. Integration Testing

Integration testing checks how various modules work together. Here, it verified:

- Interaction between the frontend (HTML, JS) and backend (Flask routes)
- JSON-based data retrieval and response generation

- Session handling between login and chatbot modules

Example Test Case:

- Enter correct credentials → Redirect to chatbot page
- Submit chat query → Get correct bot response

3. Functional Testing

This ensures that each feature works as expected according to requirements. Tests included:

- User registration and login
- Session maintenance
- Chat interface responsiveness
- Display of recommended questions
- Real-time bot response

Status: All primary functions performed as expected

4. Usability Testing

Usability testing focuses on the interface and overall user experience. This involved:

- Checking navigation clarity
- Visual differentiation between user and bot messages
- Proper alignment and readability on different devices

Feedback from sample users helped refine interface responsiveness and button placement.

5. Compatibility Testing

The application was tested on multiple web browsers including:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge

And devices like:

- Desktop
- Laptop
- Smartphone (Responsive View)

All major features worked consistently across platforms.

6. Performance Testing

Although the system uses a lightweight JSON backend, basic performance metrics were assessed:

- Response time (within 1-2 seconds for chat)
- Page load speed
- Simultaneous user sessions (up to 5 handled smoothly in test environment)

7. Security Testing

Key security aspects were manually verified:

- Input sanitization to prevent XSS
- Password field masking
- Session protection (users can't access /chat without login)
- Logout clears session

Test Summary

Test Type	Status	Remarks
Unit Testing	<input checked="" type="checkbox"/> Passed	All backend functions validated
Integration Testing	<input checked="" type="checkbox"/> Passed	Frontend-backend interaction verified
Functional Testing	<input checked="" type="checkbox"/> Passed	All features working as expected
Usability Testing	<input checked="" type="checkbox"/> Passed	Interface is clear and user-friendly
Compatibility Testing	<input checked="" type="checkbox"/> Passed	Consistent behavior across devices
Performance Testing	<input checked="" type="checkbox"/> Passed	Fast and responsive chat experience
Security Testing	<input checked="" type="checkbox"/> Passed	Basic protections are implemented

CHAPTER 5

CONCLUSION & FUTURE SCOPE

Conclusion

The development of the AI-powered chatbot system with login and registration functionality demonstrates the successful implementation of a full-stack web application using HTML, CSS, JavaScript, Python (Flask), and JSON. The project provides a user-friendly and interactive platform where users can log in securely and engage with a chatbot to receive predefined answers to various general knowledge and career-related queries.

This project showcases the seamless integration of frontend design with backend logic, session handling, and real-time user interaction. The use of Flask as a lightweight web framework allowed for efficient routing, modular development, and faster deployment during testing. By utilizing a JSON-based structure for storing and retrieving chatbot responses, the system remains flexible and easy to update without the complexity of a traditional database.

In terms of learning, this project has provided a comprehensive understanding of web development, user authentication, template rendering, data handling, and basic natural language processing principles. It has enhanced not only technical skills but also problem-solving abilities, debugging, and UI/UX design awareness.

Moreover, the chatbot system can be easily expanded, making it a valuable foundation for future AI-driven applications. This project stands as an example of how simple tools and technologies, when used effectively, can produce meaningful, interactive, and intelligent web solutions.

Overall, the chatbot system fulfils its objectives of providing a smart, efficient, and engaging user experience and has the potential to evolve into a more intelligent, data-driven assistant with further development and integration of advanced technologies.

Future Scope

The current chatbot system provides a solid foundation for basic conversational interaction using predefined responses. However, there is significant potential for future enhancements. One of the most promising improvements is the integration of advanced Natural Language Processing (NLP) and Machine Learning models, such as GPT from OpenAI or similar models from Hugging Face, which can enable the chatbot to understand and respond more intelligently to a wider range of user inputs.

Additionally, the system can be upgraded to use a relational database such as MySQL or PostgreSQL to store user credentials, chat history, and frequently asked questions. This would enhance data management, improve scalability, and allow for advanced analytics and reporting features. Incorporating a feedback mechanism would allow users to rate the chatbot's responses, enabling continuous improvement of the answer database.

The user experience can also be improved by adding voice input and output capabilities, turning the chatbot into a voice assistant that is more accessible and interactive. Furthermore, multilingual support can be introduced to cater to users from different linguistic backgrounds, making the system more inclusive.

Another significant area for future development is the creation of an admin dashboard. This dashboard would allow administrators to manage users, monitor system usage, update chatbot content, and analyse common user queries. Finally, the system can be converted into a mobile application using frameworks like Flutter or React Native to ensure accessibility on the go.

These enhancements will not only improve the chatbot's intelligence and usability but also increase its value and applicability in educational institutions, business customer support, and various other domains.

1. Integration with Machine Learning/NLP Models

- Replace or augment the rule-based responses with dynamic AI responses using models like OpenAI GPT or Hugging Face Transformers for more personalized and context-aware interactions.

2. Database Integration

- Introduce a relational database (e.g., MySQL or PostgreSQL) to store user credentials, messages, and chat history securely and efficiently.

3. Voice Assistant Capabilities

- Integrate speech-to-text and text-to-speech functionalities for a voice-enabled chatbot experience.

4. Admin Dashboard

- Add an admin panel to monitor chatbot usage, view frequently asked questions, and manage predefined Q&A data.

5. Feedback Mechanism

- Allow users to rate the chatbot's responses and provide suggestions to improve answer quality.

6. Multi-language Support

- Implement language translation APIs to support users in different regional languages.

7. Mobile App Version

- Develop a mobile-friendly version or convert the web app into an Android/iOS application using frameworks like React Native or Flutter.

CHAPTER 6

REFERENCES

1. Flask Documentation. (n.d.). *Flask – A micro web framework written in Python.* Retrieved from: <https://flask.palletsprojects.com/>
2. W3Schools. (n.d.). *HTML, CSS, JavaScript Tutorials.* Retrieved from: <https://www.w3schools.com/>
3. Mozilla Developer Network (MDN). (n.d.). *HTML, CSS, JavaScript Reference.* Retrieved from: <https://developer.mozilla.org/>
4. JSON.org. (n.d.). *JavaScript Object Notation.* Retrieved from: <https://www.json.org/>
5. Python Software Foundation. (n.d.). *Official Python Documentation.* Retrieved from: <https://docs.python.org/3/>
6. OpenAI. (n.d.). *ChatGPT and GPT-3 API Documentation.* Retrieved from: <https://platform.openai.com/docs>
7. GitHub. (n.d.). *Example Flask Login and Chatbot Projects.* Retrieved from: <https://github.com>
8. GeeksforGeeks. (n.d.). *Python Flask Tutorial – Login System, Sessions and Chatbots.* Retrieved from: <https://www.geeksforgeeks.org>
9. Stack Overflow. (n.d.). *Various posts on Flask, HTML/JS integration and debugging.*
10. <https://www.youtube.com>
11. <https://www.google.com>