## UNIVERSITY OF MUMBAI



## Master of Computer Application

**In-Semester Capstone Project** **Report on**

**Zentra Chatbot**

## Submitted by – Priyanshu Gupta

## Roll no –47

(Semester-I)

**Year 2024-25**

Under the Guidance of

Prof. Santosh Jain

**Table of Contents**

|  |  |
| --- | --- |
| **1. Introduction** | **3** |
| Introduction of Project | **3** |
| Project Purpose | **3** |
| Project Objective | **4** |
| Scope of Project | **4** |
| **2. System Study** | **5** |
| [Requirement Analysis](#_2ptdtwfl9s51) | **5** |
| [Planning and Scheduling](#_fr7xgn37ajgb) | **6** |
| [Preliminary Product Description](#_slrrsjdku3db) | **7** |
| [Justification of Platform](#_py6brstcpsgo) | **8** |
| [Conceptual Model](#_87uzqfvn74ed) | **9** |
| **3. Analysis and Design** | **10** |
| [Hardware Requirements](#_undo72sgse0x) | **10** |
| [Software Requirements](#_4s2vbmw06a27) | **10** |
| [Actual Gantt Chart](#_tetgs5v3ol60) | **12** |
| [System Design](#_jztlkqwczoja) | **12** |
| [UML Diagrams](#_bg7gtcp7phv6) | **13** |
| [Flow Chart](#_xa7w053gd3ke) | **13** |
| [Data Flow Diagram](#_wms6mzgfw84q) | **15** |
| [Entity Relationship Diagram](#_yc9v7aawjwmd) | **17** |
| [Description of Entities](#_sypmrqgs7tsq) | **18** |
| **4. Testing and Validation** | **20** |
| Code Efficiency | **20** |
| [Testing Approach](#_ei0s52qp3rxl) | **21** |
| [Unit Testing](#_80znhqljl9r0) | **21** |
| [Integration Testing](#_nsfn8ne501j9) | **22** |
| **5.User Manual** | **23** |
| User Documentation | **23** |
| Outputs | **24** |
| Code Snippets | **29** |
| **6.Conclusion** | **31** |
| Conclusion | **31** |
| Limitations | **31** |
| Future Scope | **32** |

1. **Introduction**
2. **Introduction of Project**

Zentra Chatbot is a full-stack web application that enables users to have intelligent conversations with website content. The application integrates modern web technologies with advanced AI processing to create a seamless conversational experience around any website's information.

1. **Project Purpose**

The purpose of Zentra Chatbot is to make website information more accessible through natural language conversation. It serves as a bridge between users and complex website content, allowing them to quickly extract relevant information without manually navigating through multiple pages. This enhances user experience and information accessibility.

1. **Project Objective**
   1. Develop a conversational interface that can process and understand website content
   2. Enable natural language questioning about any website's information
   3. Provide accurate, relevant responses based on scraped website data
   4. Store and manage conversation history for future reference
   5. Support document upload and processing for multiple file formats
   6. Create an intuitive, responsive user interface for easy interaction
2. **Scope of Project**

The scope of Zentra Chatbot encompasses:

* User authentication system with secure login/registration
* Website URL processing and content extraction
* Document upload and processing capabilities
* AI-powered conversational interface
* Chat history management
* Responsive, intuitive UI/UX design
* Three-tier architecture (React frontend, Node.js backend, Python AI service)
* Vector database integration for efficient information retrieval

1. **System Study**

* **Requirement Analysis**
  + **Functional Requirements**

1. **User Management**
   * User registration and authentication
   * Secure login with JWT token authentication
   * User profile management
2. **Content Processing**
   * Website URL validation and processing
   * Intelligent web scraping with depth control
   * Support for JavaScript-heavy websites (using Selenium)
   * Document upload and processing (PDF, DOCX, TXT, etc.)
   * Content vectorization and storage
3. **Conversation Management**
   * Natural language query processing
   * Context-aware responses
   * Chat history storage
   * Real-time conversation updates
   * Message formatting with Markdown support
4. **Interface Requirements**
   * Responsive web design
   * Chat message history view
   * Website/document upload interface
   * Previous websites management
   * Progress indicators for lengthy operations

* + **Non-Functional Requirements**

1. **Performance**
   * Fast response time for user queries
   * Efficient website processing
   * Optimized vector search for relevant information
2. **Security**
   * Secure authentication
   * Data protection
   * Input validation
3. **Scalability**
   * Support for multiple concurrent users
   * Efficient database operations
4. **Usability**
   * Intuitive user interface
   * Clear error messages
   * Progress indicators
   * Responsive design for different devices

* **Planning and Scheduling** 
  + **Project Phases**

1. **Requirements Analysis** - Understanding user needs and system requirements
2. **Architecture Design** - Designing the three-tier architecture
3. **Frontend Development** - Building the React-based user interface
4. **Backend Development** - Creating Node.js API endpoints and database integration
5. **AI Service Development** - Implementing the Python-based NLP capabilities
6. **Integration** - Connecting all components into a cohesive system
7. **Testing** - Validating functionality and performance
8. **Deployment** - Making the application accessible to users

* **Key Milestones**

1. Requirements finalization
2. Architecture design approval
3. Frontend prototype completion
4. Backend API implementation
5. AI module integration
6. Full system integration
7. User testing completion
8. Final deployment

* **Preliminary Product Description**

Zentra Chatbot is a web-based application that allows users to interact with website content through a conversational interface. Users can input any website URL, and the system will process the site, extracting and analysing its content. After processing, users can ask questions about the website in natural language, receiving relevant, accurate responses derived from the site's content.

The system also supports document upload functionality for various file formats, converting document content into a format suitable for natural language questioning. The conversational interface provides a chat-like experience with support for rich text formatting, code syntax highlighting, and context-aware responses.

All interactions are securely stored in the user's history, allowing them to revisit previous conversations and websites. The application features a clean, modern interface with a three-panel layout containing chat history, conversation area, and website/document management.

* **Justification of Platform**
  + **Technology Stack Selection**

1. **Frontend: React.js with Material UI**
   * **Justification**: React provides component-based architecture for building interactive UIs efficiently. Material UI offers pre-designed components that implement Google's Material Design, ensuring a professional, consistent user experience.
2. **Backend: Node.js with Express**
   * **Justification**: Node.js offers high performance, non-blocking I/O for handling multiple concurrent connections. Express provides a robust framework for building RESTful APIs efficiently.
3. **Database: MongoDB**
   * **Justification**: NoSQL database suitable for storing semi-structured data like chat histories and user profiles. Offers flexibility in schema design and scales horizontally.
4. **AI Processing: Python with LangChain**
   * **Justification**: Python is the industry standard for AI/ML development. LangChain provides a framework for working with language models, making it easier to build conversational applications.
5. **Vector Storage: Chroma**
   * **Justification**: Purpose-built vector database for storing embeddings, enabling efficient similarity search needed for retrieving relevant information from processed content.
6. **Web Scraping: Beautiful Soup and Selenium**
   * **Justification**: Beautiful Soup provides efficient HTML parsing, while Selenium enables processing of JavaScript-heavy websites that require browser rendering.

* **Conceptual Model**

**The Zentra Chatbot system is conceptualized as a three-tier architecture:**

1. **Presentation Layer (Frontend)**
   * User interface components (login, chat, website processing)
   * State management
   * API communication with backend
2. **Application Layer (Backend)**
   * Authentication services
   * Chat management
   * Website tracking
   * Communication with AI service
3. **AI Processing Layer**
   * Website content extraction
   * Document processing
   * Vector storage management
   * Query processing
   * Response generation
4. **Analysis and Design**
   * **Hardware Requirement**

* **Development Environment:**
  + Processor: Intel Core i5/i7 or equivalent
  + RAM: 16GB minimum
  + Storage: 256GB SSD minimum
  + Network: Broadband internet connection
* **Deployment Environment:**
  + Server with minimum 4 CPU cores
  + 16GB RAM minimum
  + 50GB storage minimum
  + Stable internet connection with sufficient bandwidth
  + **Software Requirement**
* **Development Tools:**
  + Visual Studio Code or similar IDE
  + Git for version control
  + Node.js (v14 or higher)
  + Python (v3.8 or higher)
  + MongoDB (local or cloud instance)
  + Chrome/Firefox browser for testing
* **Frontend Dependencies:**
  + React.js
  + Material UI
  + Axios for API requests
  + React Markdown
  + React Syntax Highlighter
* **Backend Dependencies:**
  + Express.js
  + Mongoose for MongoDB interaction
  + JWT for authentication
  + Bcrypt for password hashing
  + CORS for cross-origin resource sharing
* **AI Service Dependencies:**
  + LangChain framework
  + Vector database (Chroma)
  + Beautiful Soup for web scraping
  + Selenium for JavaScript-heavy websites
  + PDF processing libraries
  + **Actual Gantt Chart**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Name** | **Project Started : 01-02-2025** | | | | | |
| **Week 1 (15/2 - 28/2)** | **Week 2 (1/3 - 14/3)** | **Week 3 (15/3 - 28/3)** | **Week 4 (29/3 - 11/4)** | **Week 5 (12/4 - 25/4)** | **Week 6 (26/4 - 30/4)** |
| **Project Setup & Planning** |  |  |  |  |  |  |
|  |
| **Backend Development** |  |  |  |  |  |  |  |
|  |
| **Frontend Development** |  |  |  |  |  |  |  |
|  |
| **AI Service Development** |  |  |  |  |  |  |  |
|  |
| **Integration Phase** |  |  |  |  |  |  |  |
|  |
| **Testing & Refinement / Documentation** |  |  |  |  |  |  |  |
|  |

* + **System Design**
* **Architecture Overview**

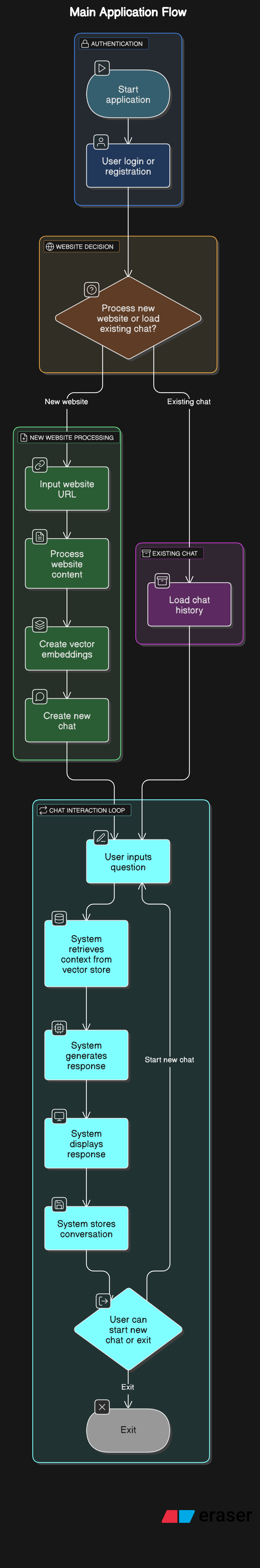
**Zentra Chatbot uses a microservices architecture with three main components:**

1. **React Frontend (Port 3000)**
   * Provides the user interface
   * Manages application state
   * Communicates with both backends
2. **Node.js Backend (Port 4000)**
   * Handles user authentication
   * Manages chat history and website data
   * Provides REST APIs for the frontend
3. **Python Flask Backend (Port 5000)**
   * Processes websites and documents
   * Manages vector storage
   * Handles natural language queries
   * Generates responses using AI

* **UML Diagrams**
  + **Flow Chart**

**The main application flow follows these steps:**

1. Start application
2. User login/registration
3. Decision: Process new website or load existing chat?
4. If new website:
   * Input website URL
   * Process website content
   * Create vector embeddings
   * Create new chat
5. If existing chat:
   * Load chat history
6. Chat interaction loop:
   * User inputs question
   * System retrieves context from vector store
   * System generates response
   * System displays response
   * System stores conversation
7. User can start new chat or exit

****

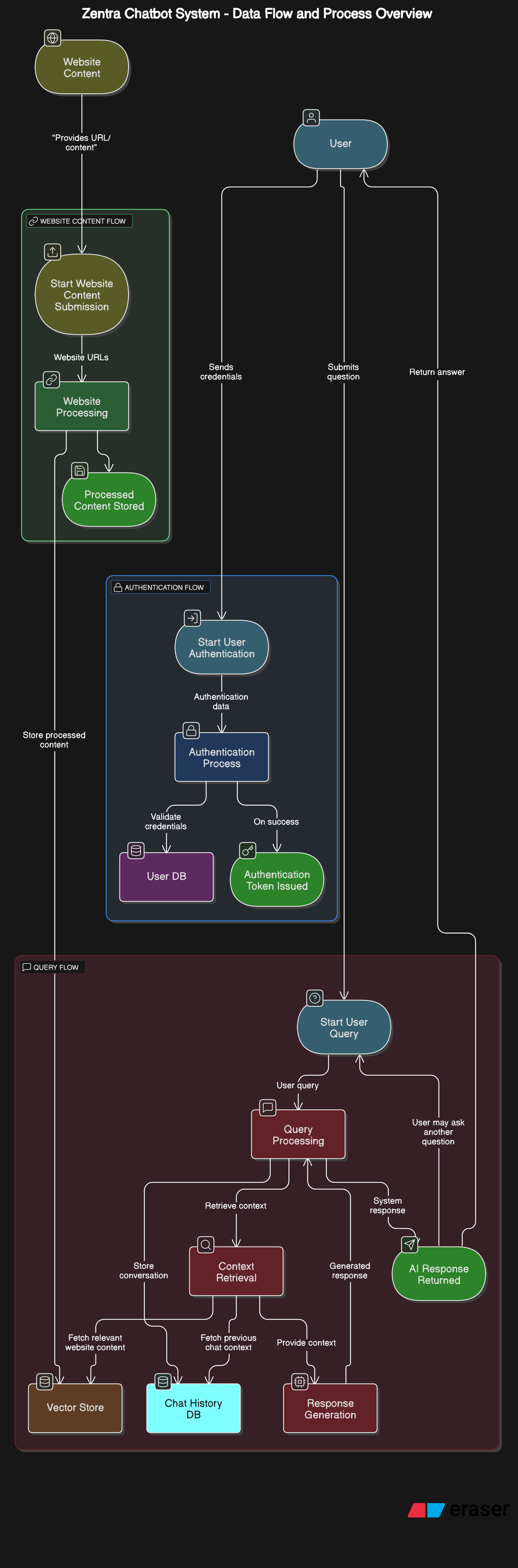
* + **Data Flow Diagram**

**Level 0 DFD**

* **External entities:** User, Website Content
* **Process:** Zentra Chatbot System
* **Data stores:** User DB, Chat History DB, Vector Store
* **Data flows**: Authentication data, Website URLs, User queries, System responses

Level 1 DFD

1. **Authentication Process**
   * Input: User credentials
   * Output: Authentication token
   * Storage: User database
2. **Website Processing**
   * Input: Website URL
   * Output: Processed content
   * Storage: Vector database
3. **Query Processing**
   * Input: User question
   * Processing: Context retrieval, response generation
   * Output: AI response
   * Storage: Chat history database

****

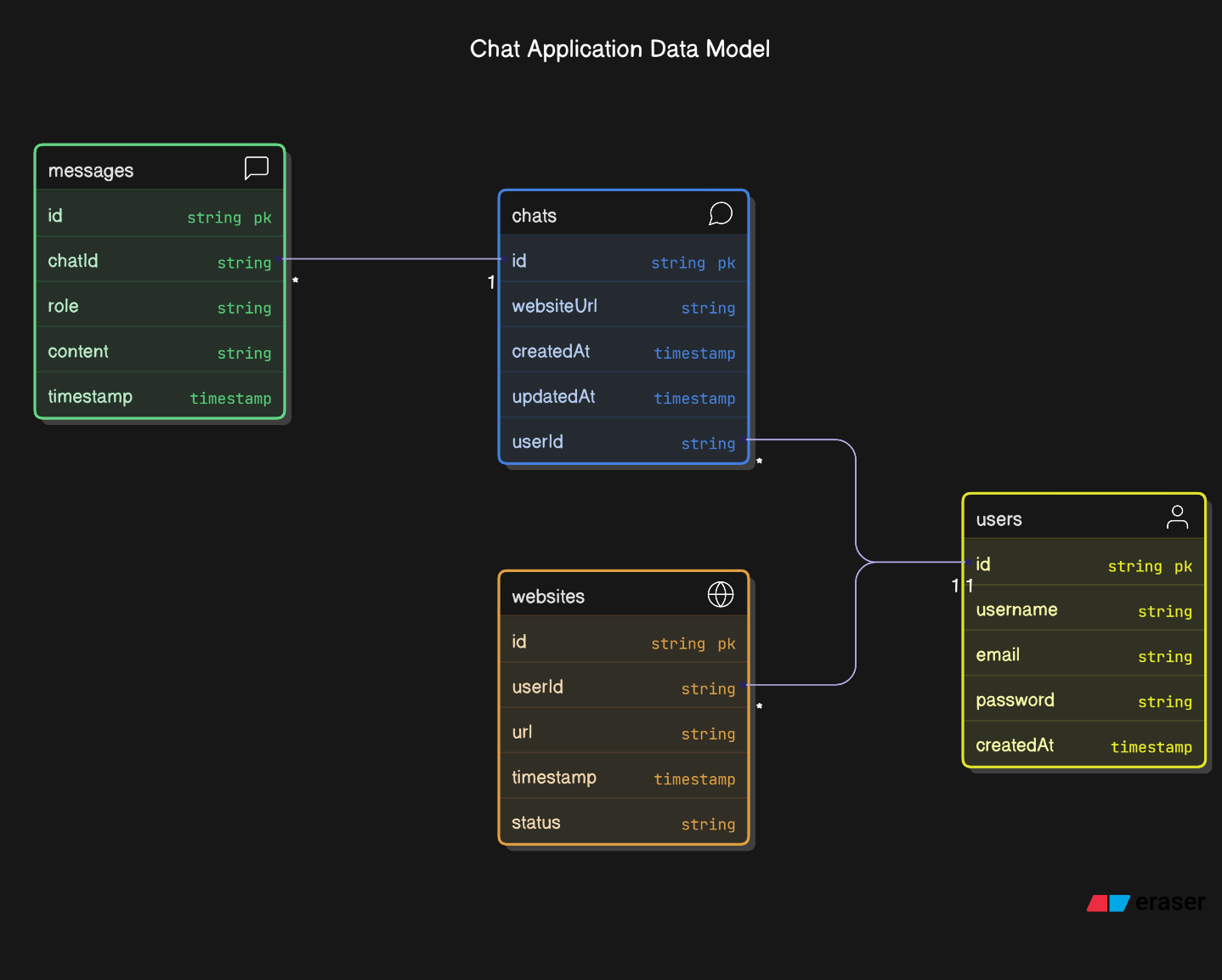
* + **Entity Relationship Diagram**

**Key Entities**

1. **User**
   * Primary Key: id
   * Attributes: username, email, password, createdAt
2. **Chat**
   * Primary Key: id
   * Foreign Key: userId (references User.id)
   * Attributes: websiteUrl, createdAt, updatedAt
3. **Message**
   * Primary Key: id
   * Foreign Key: chatId (references Chat.id)
   * Attributes: role, content, timestamp
4. **Website**
   * Primary Key: id
   * Foreign Key: userId (references User.id)
   * Attributes: url, timestamp, status

Relationships

* **User (1) → (N) Chat: One user can have many chats**
* **Chat (1) → (N) Message: One chat can have many messages**
* **User (1) → (N) Website: One user can process many websites**

****

* + **Description of Entities**

**User Entity**

Represents a registered user of the system. Stores authentication information and profile details.

* id: Unique identifier (MongoDB ObjectId)
* username: User's display name
* email: User's email address (unique)
* password: Hashed password
* createdAt: Account creation timestamp

**Chat Entity**

Represents a conversation session related to a specific website or document.

* id: Unique identifier
* userId: Reference to the user who owns this chat
* websiteUrl: URL of the website or document name
* messages: Array of message objects
* createdAt: Chat creation timestamp
* updatedAt: Last activity timestamp

**Message Entity**

Represents a single message within a chat, from either the user or the assistant.

* role: Either 'user' or 'assistant'
* content: The message text content
* timestamp: When the message was sent

**Website Entity**

Represents a processed website stored for future use.

* url: The website URL
* userId: Reference to the user who processed this website
* timestamp: When the website was processed
* status: Processing status (complete, failed)

****

1. **Testing and Validation**

* **Code Efficiency**

**The codebase was optimized for performance in several key areas:**

1. **Website Processing**
   * Implemented breadth-first search for efficient web crawling
   * Added depth limits to prevent excessive crawling
   * Used smart detection for JavaScript-heavy websites
   * Implemented priority-based link processing
2. **Vector Storage**
   * Optimized chunk size for text splitting
   * Used efficient embedding models
   * Implemented vector store mapping for quick lookups
3. **Frontend Performance**
   * Implemented virtualized lists for chat history
   * Used React.memo for component optimization
   * Implemented efficient state management
4. **Backend Efficiency**
   * Added database indexing for common queries
   * Implemented pagination for large result sets
   * Used connection pooling for database operations

* **Testing Approach**

**The testing strategy followed a comprehensive approach:**

1. **Manual Testing**
   * User interface functionality
   * End-to-end workflows
   * Edge case handling
2. **Automated Testing**
   * Unit tests for core functions
   * Integration tests for component interaction
   * API endpoint testing
3. **Performance Testing**
   * Load testing for concurrent users
   * Response time measurement
   * Resource utilization monitoring

* **Unit Testing**

**Unit tests were created for core functionality:**

1. **Authentication Tests**
   * User registration validation
   * Login process
   * JWT token verification
2. **Chat Processing Tests**
   * Message formatting
   * Chat history persistence
   * Chat renaming and deletion
3. **Website Processing Tests**
   * URL validation
   * Content extraction
   * Error handling

* **Integration Testing**

**Integration tests verified the interaction between system components:**

1. **Frontend-Backend Integration**
   * API request/response validation
   * Authentication flow
   * Error handling and display
2. **Backend-AI Service Integration**
   * Website processing workflows
   * Query-response cycles
   * Data persistence
3. **Database Integration**
   * Data storage and retrieval
   * Relationship integrity
   * Transaction handling
4. **User Manual**

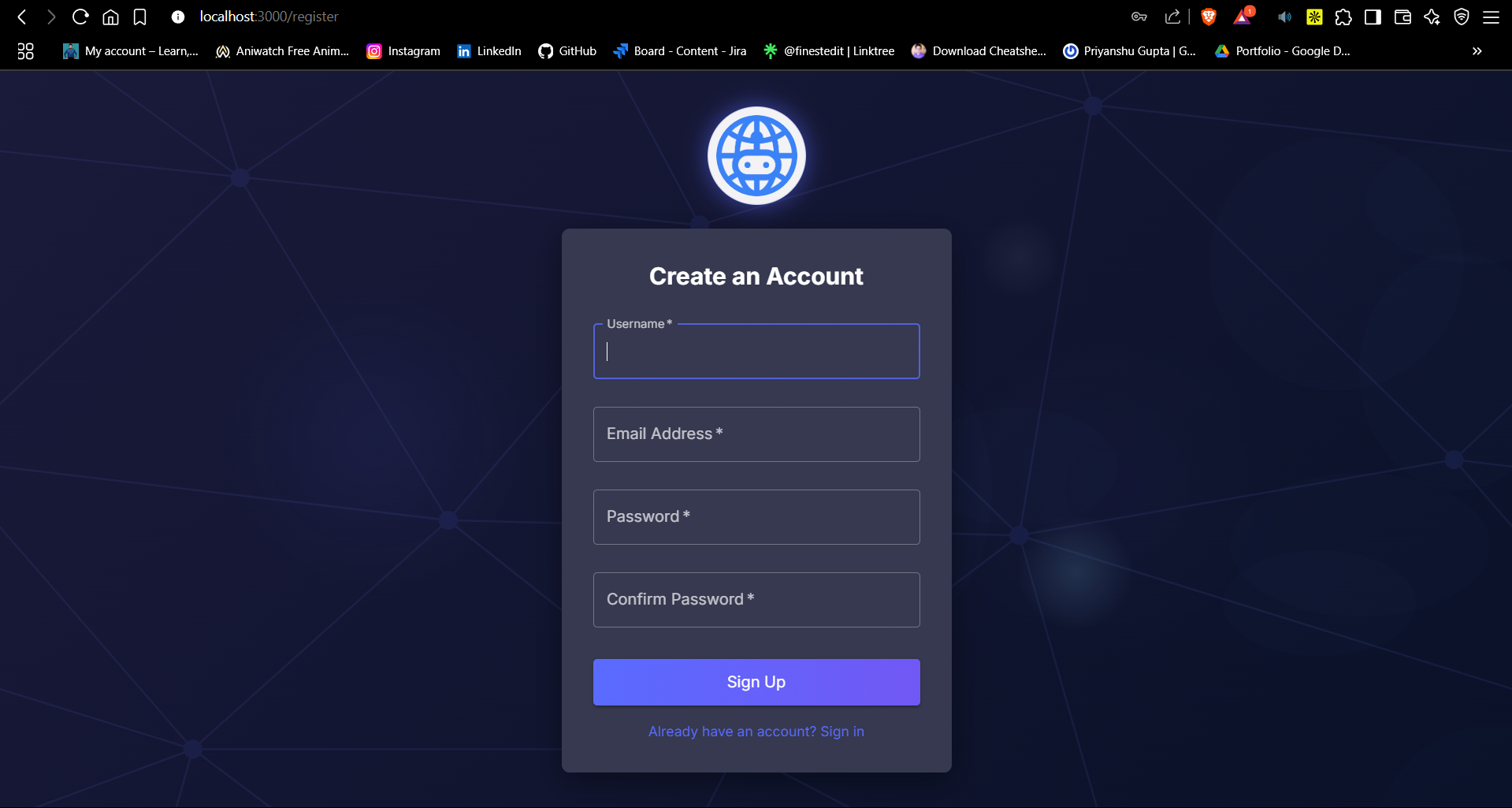
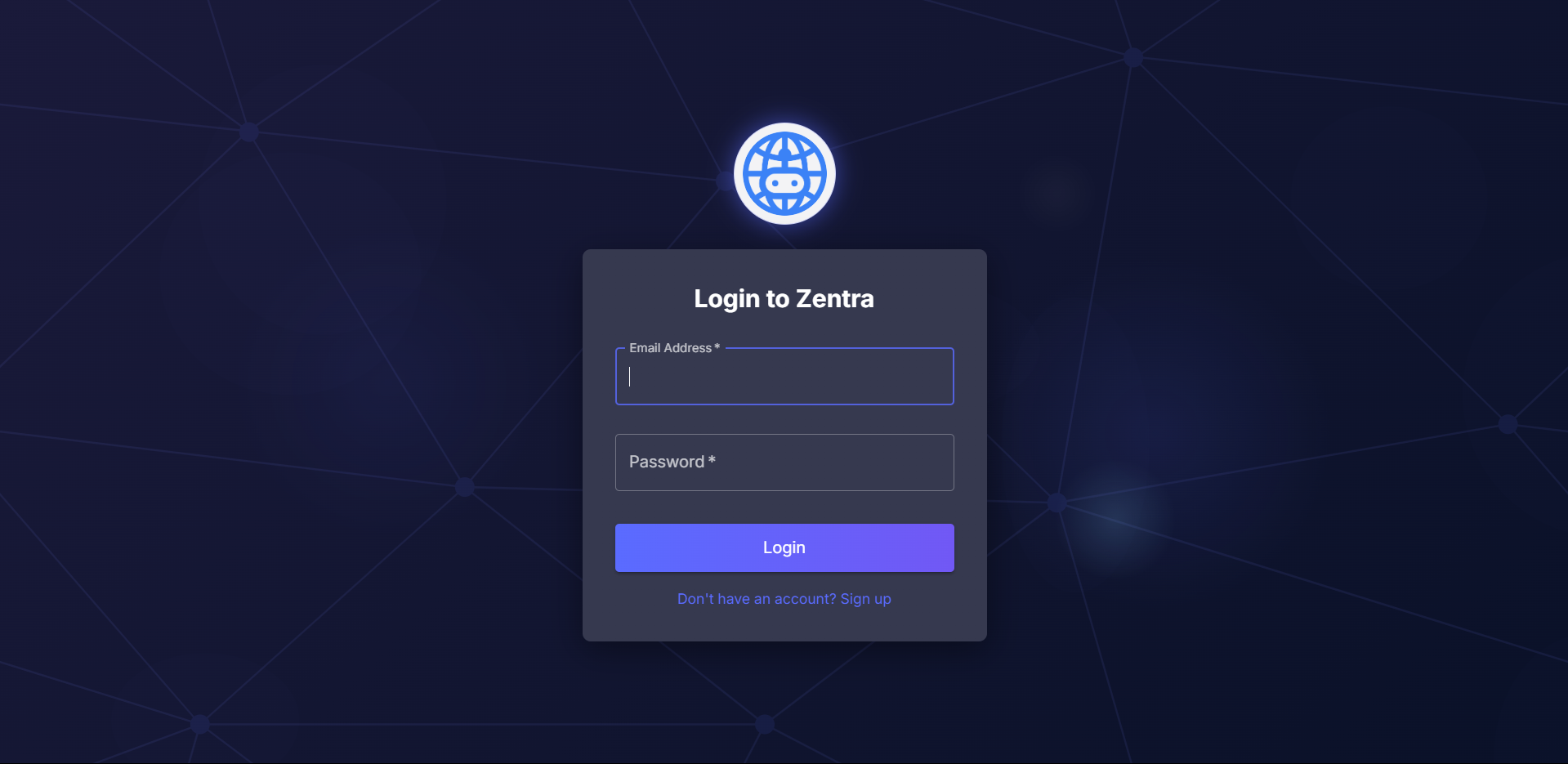
* **User Documentation**

**Getting Started**

1. **Account Creation**
   * Navigate to the login page
   * Click "Sign Up" to create a new account
   * Enter your email, username, and password
   * Click "Register" to create your account
2. **Logging In**
   * Enter your email and password
   * Click "Login" to access your dashboard
3. **Processing a New Website**
   * Click "New Chat" in the sidebar
   * Enter a website URL in the provided field
   * Click "Start Chat" to begin processing
   * Wait for the processing to complete (progress will be displayed)
4. **Asking Questions**
   * Once website processing is complete, type your question in the message box
   * Press Enter or click the send button to submit your question
   * The system will display a typing indicator while generating a response
   * Read the response and continue the conversation
5. **Uploading Documents**
   * Click on "Upload Document" in the right sidebar
   * Select a document from your computer (supported formats: PDF, DOCX, TXT, MD, CSV, JSON)
   * Wait for processing to complete
   * Start asking questions about the document content
6. **Managing Chat History**
   * View previous chats in the left sidebar
   * Click on any chat to continue the conversation
   * Use the menu (three dots) to rename or delete a chat
7. **Loading Previous Websites**
   * Click "Refresh List" in the right sidebar to see previously processed websites
   * Click "Load" on any website to start a new chat with that website
   * Click "Remove" to delete a website from your history

* **Outputs**

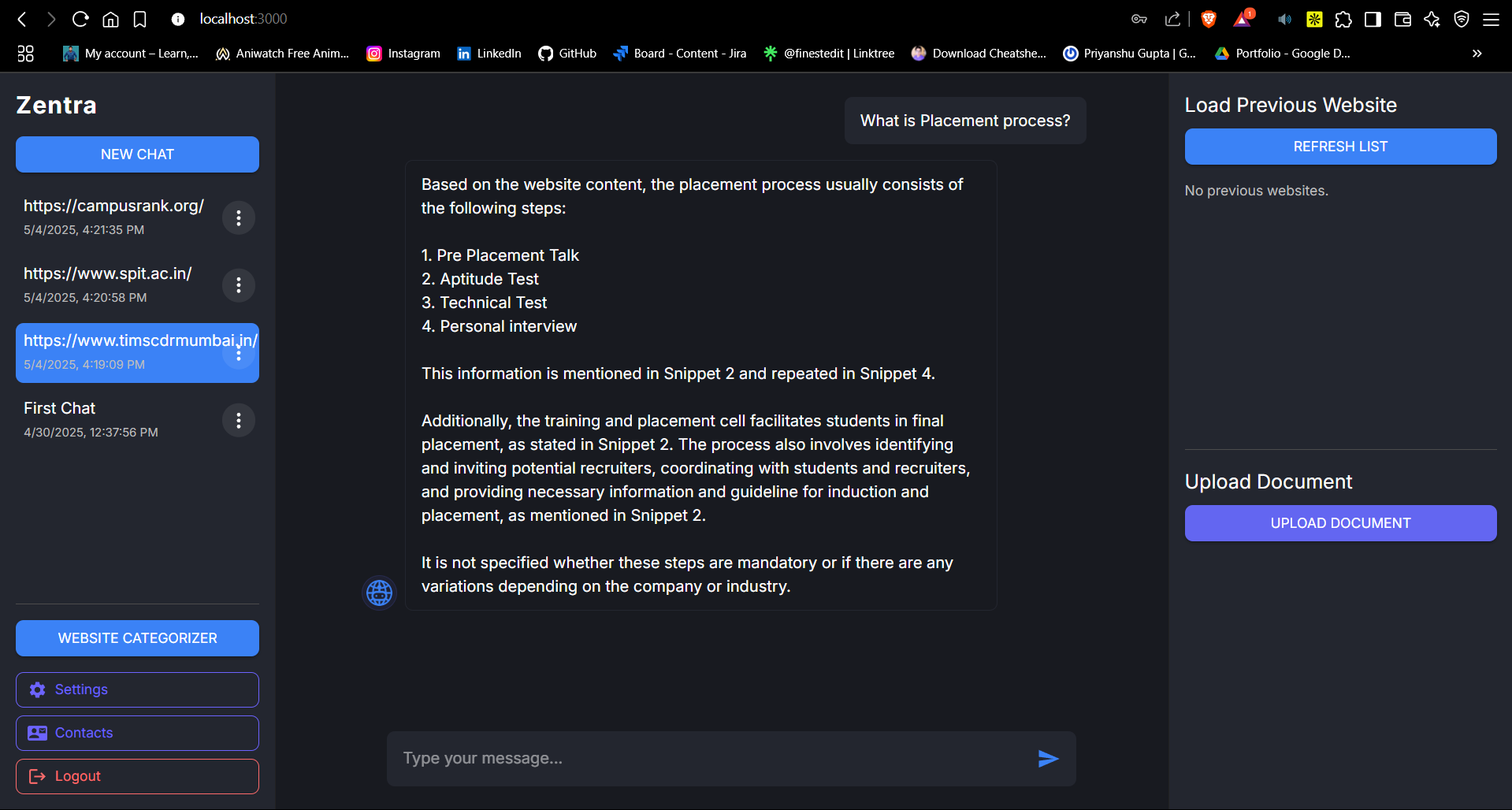
**Login & Registration Page**



**Chat Interface**

The main chat interface features:

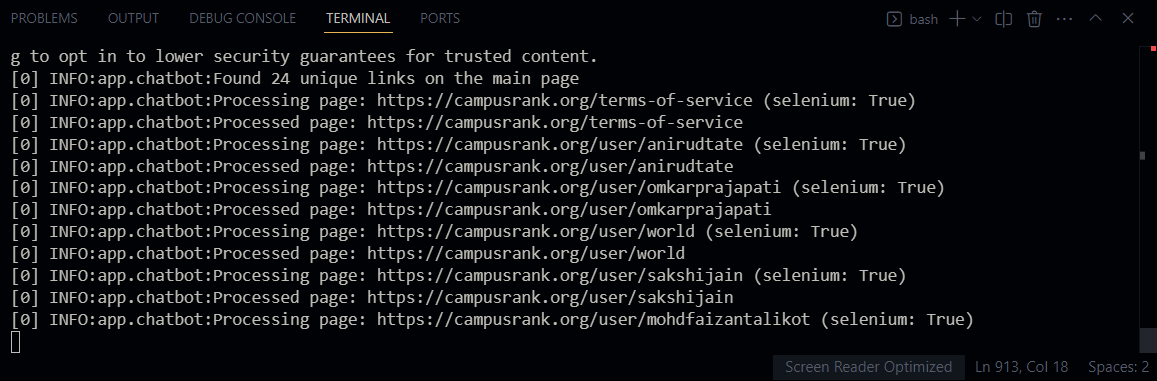
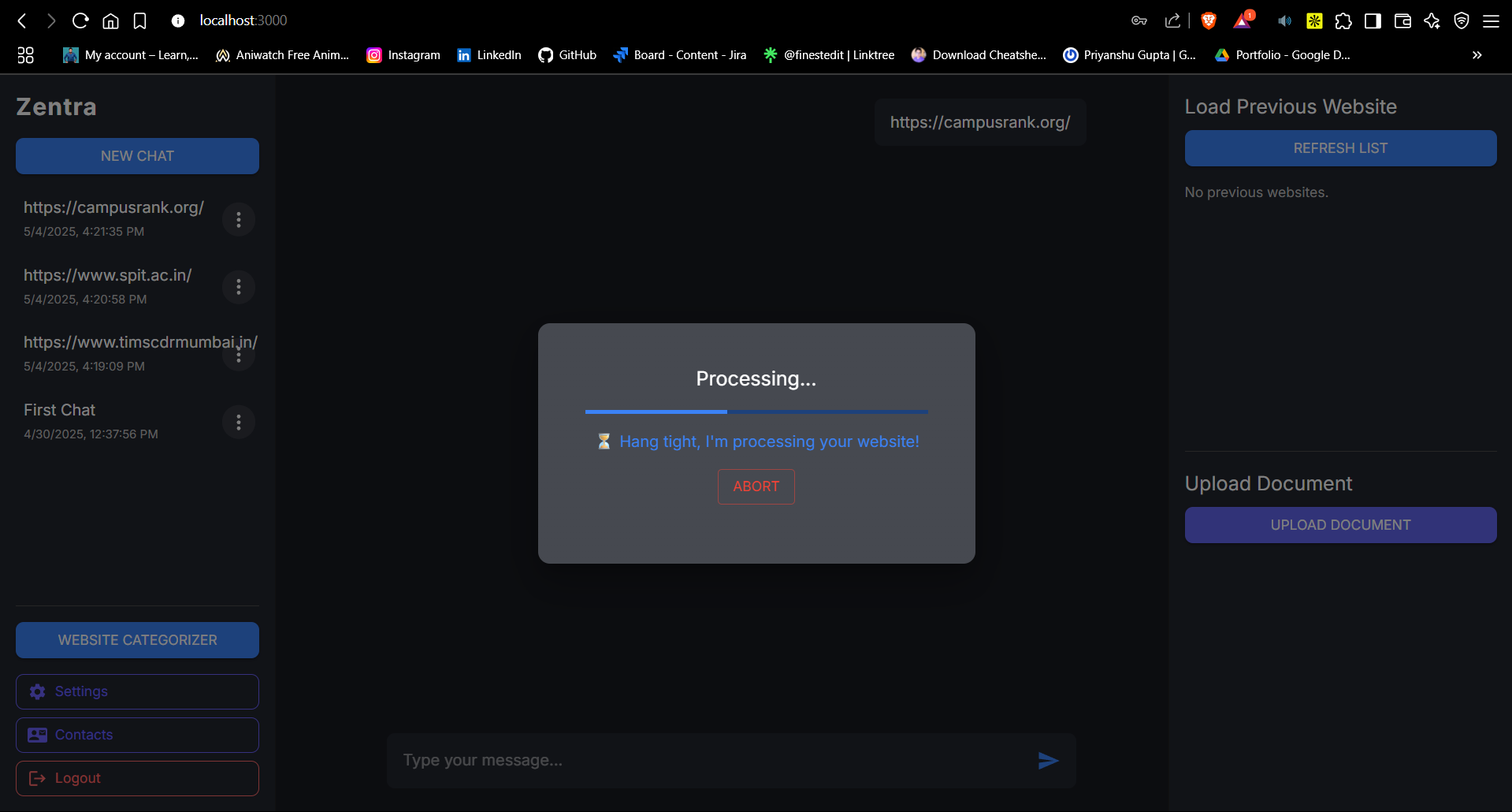
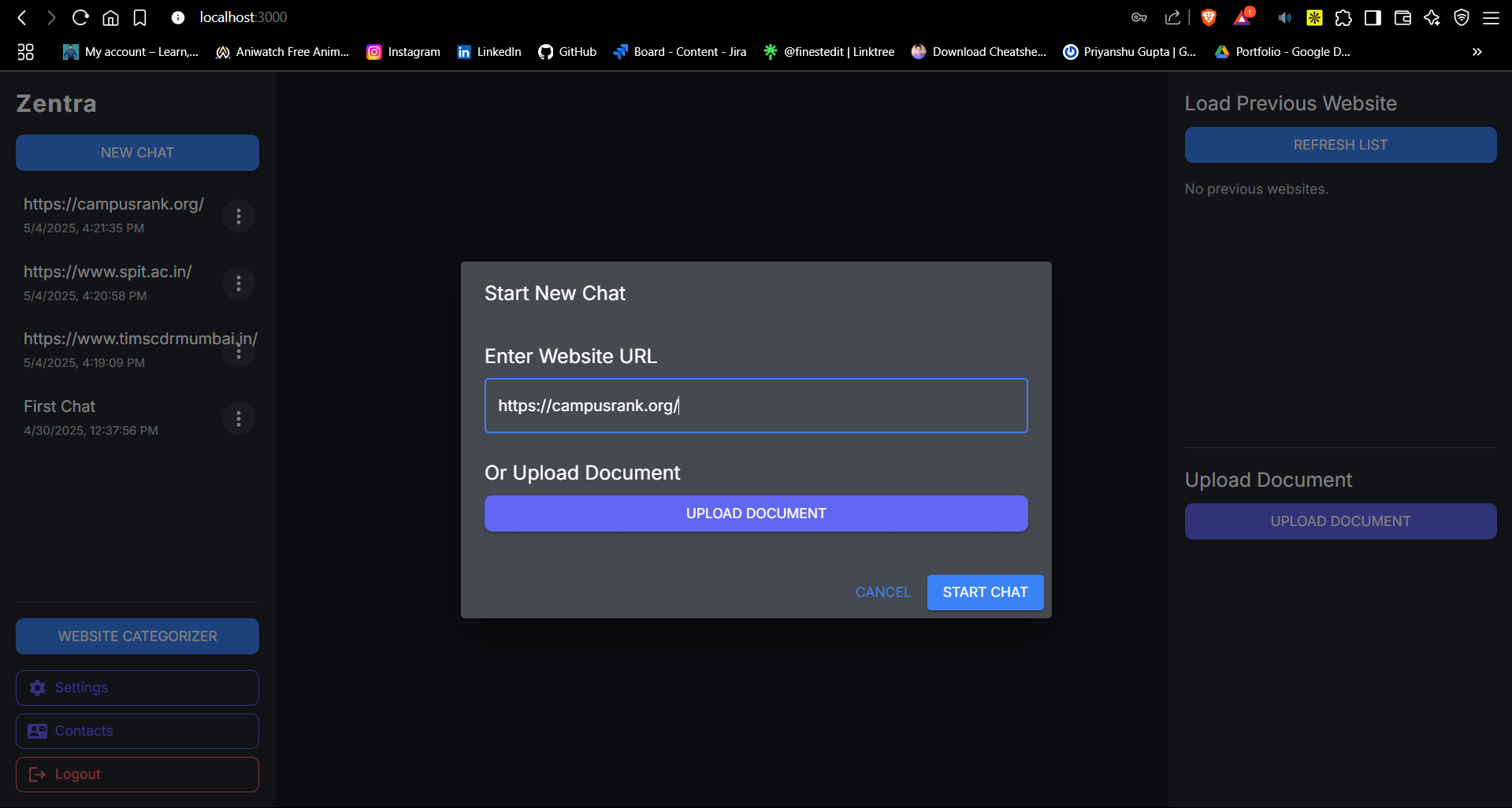
* Three-panel layout with chat history, conversation area, and website/document management
* Dark theme for comfortable viewing
* Message bubbles with clear visual distinction between user and assistant messages
* Markdown formatting in responses for improved readability
* Code syntax highlighting for technical content
* Typing indicators to show when the assistant is generating a response



**Website Processing**

When processing a website, users will see:

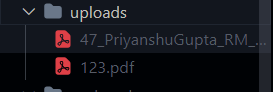
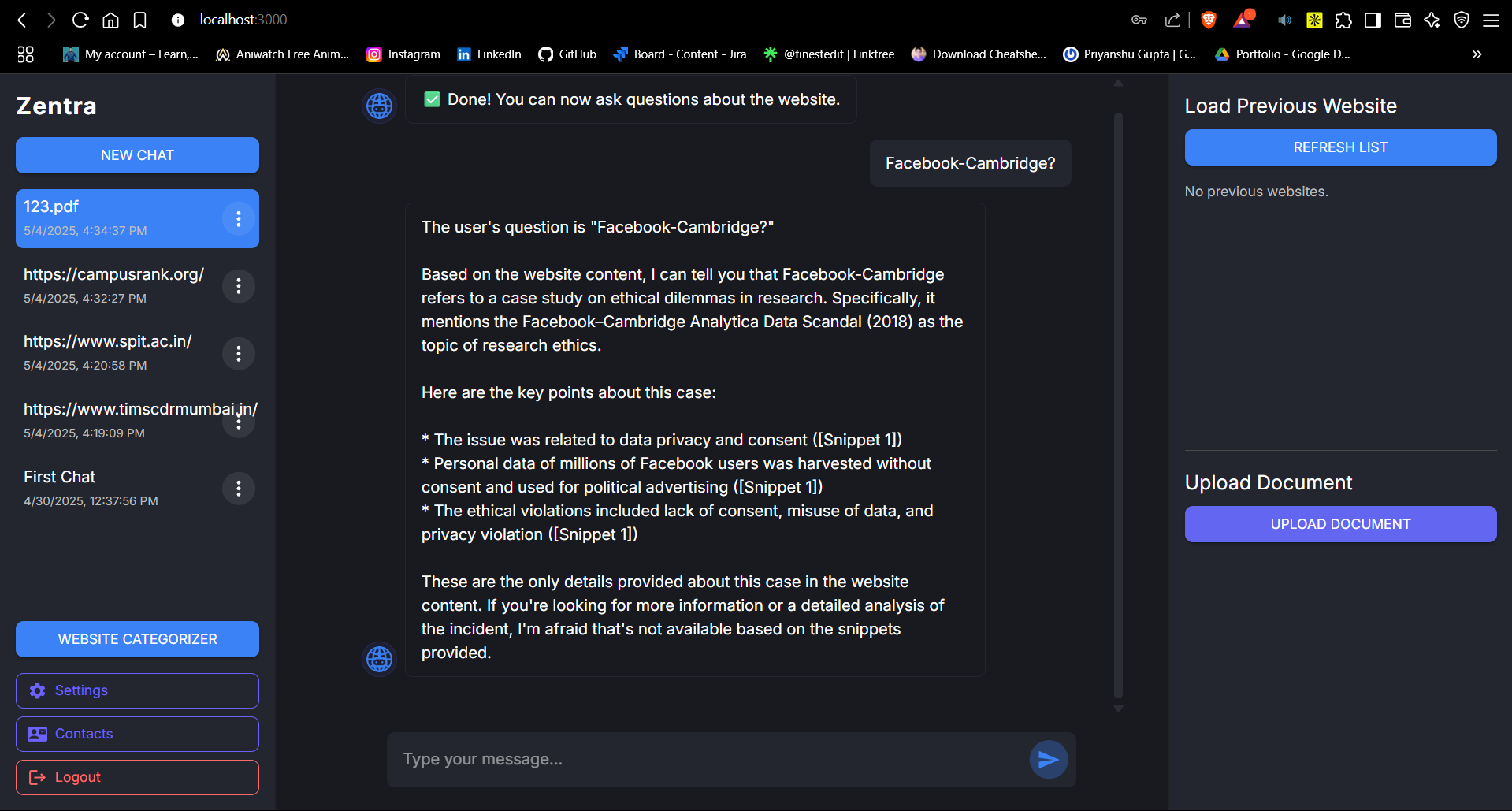
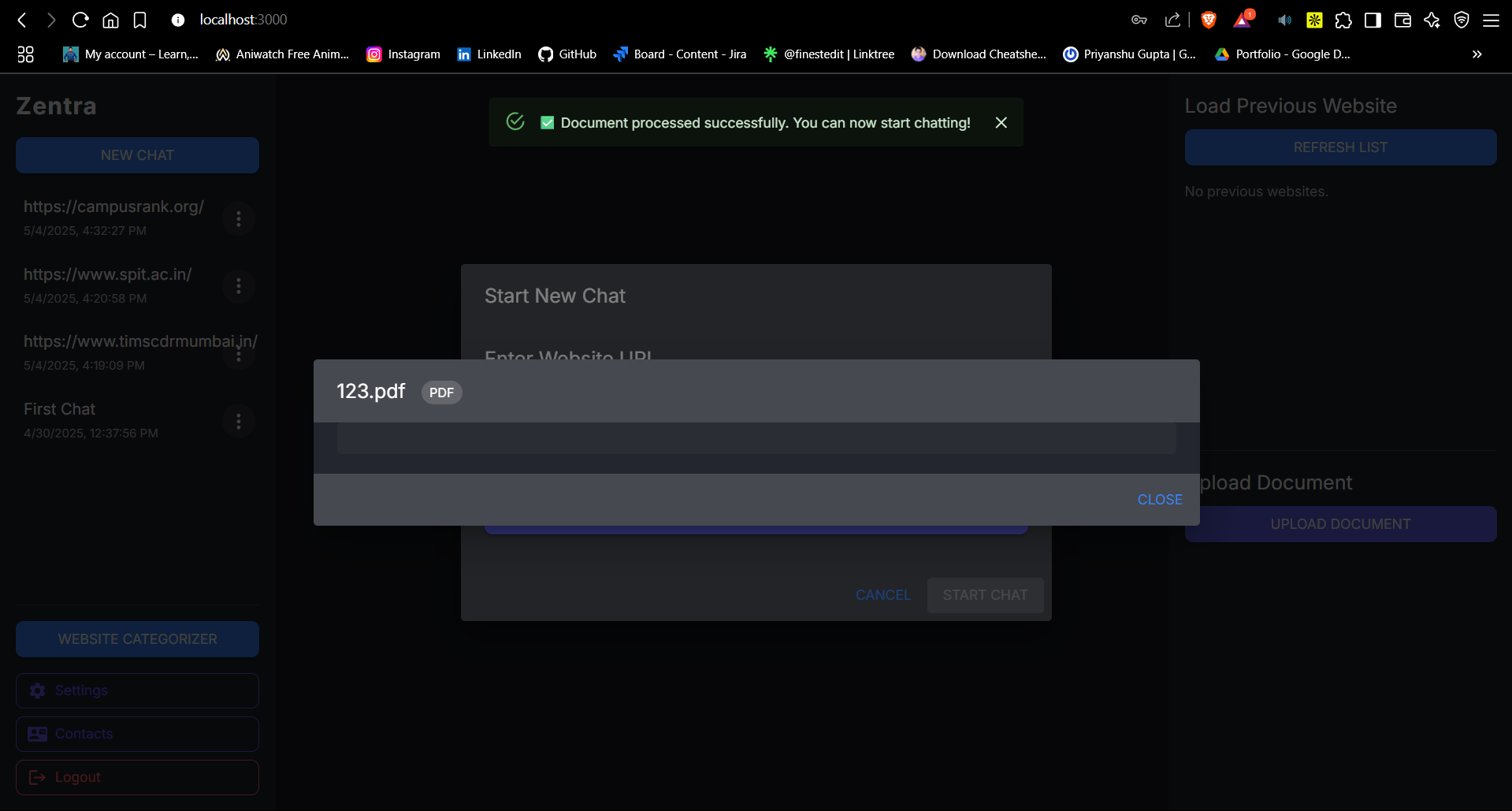
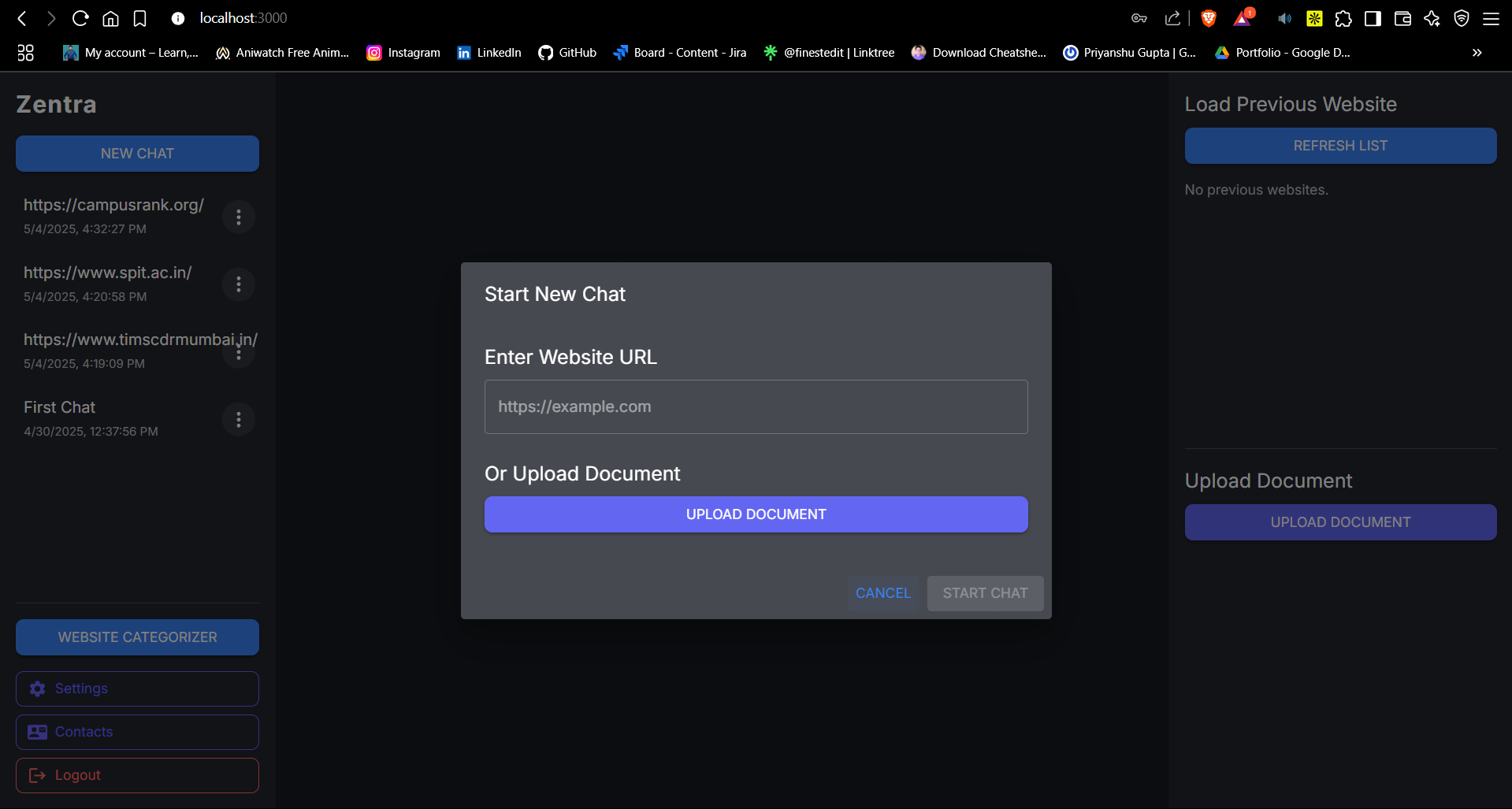
* Progress dialog showing current activity
* Status updates for each page being processed
* Processing summary upon completion



**Document Upload**

When uploading documents, the system provides:

* File selection interface
* Processing status indicators
* Preview of document content
* Confirmation of successful processing



* **Code Snippets**

#### React Component for Chat Messages

{messages.map((msg, idx) => (

  <Box

*key*={idx}

*sx*={{

      mb: 2,

      display: 'flex',

      justifyContent: msg.role === 'user' ? 'flex-end' : 'flex-start',

      alignItems: 'flex-end',

      width: '100%',

    }}

  >

    {*/\* Bot avatar for bot messages \*/*}

    {msg.role !== 'user' && (

      <Box *sx*={{ mr: 1, display: 'flex', alignItems: 'flex-end' }}>

        <img

*src*={BotLogo}

*alt*="Bot"

*style*={{

            width: 36,

            height: 36,

            borderRadius: '50%',

            background: 'rgba(100, 108, 255, 0.08)',

            border: '1px solid #5A6BFF22',

            padding: 4,

            boxSizing: 'border-box',

            objectFit: 'cover',

          }}

        />

      </Box>

    )}

    <Box

*sx*={{

        bgcolor: msg.role === 'user' ? '#23262F' : 'background.default',

        color: '#fff',

        px: 2,

        py: 1.5,

        borderRadius: 2,

        maxWidth: '80%',

        boxShadow: 0,

        fontSize: '1rem',

        wordBreak: 'break-word',

        textAlign: 'left',

        mx: msg.role === 'user' ? 2 : 0,

        display: 'flex',

        alignItems: 'center',

        border: msg.role !== 'user' ? '1px solid #23262F' : 'none',

      }}

    >

      <Box *sx*={{ whiteSpace: 'pre-line' }}>

        {msg.role !== 'user'

          ? msg.content.replace(/^[]?website[\]]?[:\s-]\*/i, '')

          : msg.content}

      </Box>

    </Box>

  </Box>

))}

1. **Conclusion**

* **Conclusion**

Zentra Chatbot represents a successful implementation of conversational AI technology applied to website content interaction. By combining web scraping, natural language processing, and modern web development practices, the application provides an intuitive way for users to extract information from websites through natural conversation.

The three-tier architecture offers a scalable, maintainable solution that separates concerns while allowing for efficient communication between components. The React frontend delivers a responsive, user-friendly interface, while the Node.js backend handles user authentication and data persistence. The Python AI service powers the core functionality with advanced natural language processing capabilities.

* **Limitations**

1. **JavaScript-Heavy Websites**
   * Some highly dynamic websites may not be fully scraped, especially those with complex JavaScript navigation or anti-scraping protections.
2. **Processing Time**
   * Large websites with many pages can take significant time to process, affecting user experience.
3. **Response Accuracy**
   * While the AI model provides high-quality responses, it may occasionally generate inaccurate information or misinterpret complex queries.
4. **Resource Intensity**
   * The application requires significant computing resources, especially for vector storage and retrieval operations.
5. **Limited Media Processing**
   * The system primarily processes text content and has limited capability to understand or describe images, videos, or interactive elements.

* **Future Scope**

1. **Enhanced Media Processing**
   * Integration with vision models to process and describe images on websites
   * Video content summarization and question-answering capabilities
2. **Multi-Website Integration**
   * Ability to process and compare information across multiple websites simultaneously
   * Integration with specific industry knowledge bases
3. **Real-Time Updates**
   * Implementation of periodic re-crawling to keep information up-to-date
   * Change detection to highlight new or modified content
4. **Advanced Personalization**
   * User preference learning to customize responses based on interaction history
   * Domain-specific tuning for specialized knowledge areas
5. **Mobile Application**
   * Development of mobile apps for iOS and Android platforms
   * Offline capabilities for previously processed websites
6. **Enterprise Integration**
   * API expansion for integration with other business tools
   * Role-based access control for team environments
   * Custom deployment options for enterprise intranet use