**Generative AI with IBM Cloud**

**1  Introduction**

**1.1  Project Title**

**CitizenAI – AI‑Powered Government Scheme Advisor**

**1.2  Background & Motivation**

Across India, more than 1,000 active welfare schemes exist at the central, state, and local levels. Yet government surveys show that a significant segment of eligible citizens never apply—primarily because they cannot discover the programs, do not understand eligibility, or face language barriers. CitizenAI was conceived as a single conversational gateway that demystifies these schemes, lowers the learning curve, and personalizes recommendations via AI.

**1.3  Team Members**

**Team ID :** LTVIP2025TMID35138

**Team Leader :** Nimmakayala Venkata Sai Rahul

**Team member :** Tammana Chaitanya Naga Sri Sai

**Team member :** Yeruva Sri Deekshitha

**Team member :** Koyyalagadda Karthik

**2. Project Overview**

**Purpose:**  
CitizenAI is designed to simplify citizens’ access to government benefits by offering personalized, conversational assistance via web chat. It aims to bridge informational divides, empower underserved communities, and enhance civic transparency.

**Goals:**

* Deliver scheme recommendations based on age, location, income, and language preferences.
* Provide a multilingual chatbot interface with voice and text support.
* Ensure admins can manage scheme data and monitor user interaction metrics.

**Key Features:**

* **AI Chatbot** that understands intent and context
* **Natural Language Understanding** via IBM Watson Assistant
* **Multilingual Support** (English, Hindi, Telugu)
* **User Authentication** with JWT and social logins
* **Admin Dashboard** for analytics and scheme management

**3. Architecture**

**3.1 Frontend Architecture**

Built with React.js, the frontend includes a component-driven UI, using React Router for navigation. Core modules:

* **Chat Interface** – Real-time user interaction
* **Dashboard** – Personalized recommendations
* **Admin Panel** – Scheme add/edit and analytics

State management via Context API, styled with Bootstrap & modular CSS. Axios is used to handle HTTP requests securely.

**3.2 Backend Architecture**

Node.js with Express.js forms the core server. Key modules include:

* **Auth** – JWT issuance, protected routes
* **Chat** – Handles text/voice queries
* **Scheme** – CRUD operations for government schemes
* **Admin** – Logs retrieval, analytics data

Structure is modular: routes → controllers → services → models. Middleware layers handle error tracking, inputs sanitization, and token verification.

**3.3 Database Architecture**

Using MongoDB with the following schema:

* **User** – { name, email, passwordHash, role, langPref }
* **Scheme** – { title, desc, eligibilityCriteria, region, link }
* **ChatLog** – { userId, query, response, timestamp }

All schema definitions use Mongoose, ensuring validation. The database supports fast indexing on eligibility parameters to speed queries.

**4. Setup Instructions**

**Prerequisites**

* Node.js v18+
* NPM or Yarn
* MongoDB (local or Atlas)
* IBM Watson API credentials
* Git

**Installation Steps**

bash

git clone : https://github.com/Yeruva-Sri-Deekshitha/CITIZENAI.git

cd CITIZENAI

**Create .env in /server/:**

# Copy this file to .env and fill in your actual values

# MongoDB Configuration

MONGO\_URI=mongodb://localhost:27017/citizenai

# Gemini AI Configuration

GEMINI\_API\_KEY=AIzaSyDous5q9TRJdnd6iLNJMsJtSlYcp5IluHU

**Then install:**

bash

cd server && npm install

cd ../client && npm install

**5. Folder Structure**

📁 Citizenai  
├── 📁 database  
│ ├── db.py  
│ ├── user\_model.py  
│ └── 📁 **pycache**  
│ ├── db.cpython-311.pyc  
│ ├── db.cpython-312.pyc  
│ ├── user\_model.cpython-311.pyc  
│ └── user\_model.cpython-312.pyc  
├── 📁 models  
│ ├── chat\_model.py  
│ ├── sentiment\_model.py  
│ └── 📁 **pycache**  
├── 📁 routes  
│ ├── auth\_routes.py  
│ ├── chat\_routes.py  
│ ├── dashboard\_routes.py  
│ ├── sentiment\_routes.py  
│ └── 📁 **pycache**  
├── 📁 static  
│ └── 📁 css  
├── 📁 templates  
│ ├── about.html  
│ ├── chat.html  
│ ├── dashboard.html  
│ ├── index.html  
│ ├── landing.html  
│ ├── login.html  
│ └── signup.html  
├── 📁 utils  
│ ├── auth\_decorators.py  
│ ├── text\_cleaning.py  
│ └── 📁 **pycache**  
├── 📁 venv  
│ ├── Include/  
│ ├── Lib/  
│ ├── Scripts/  
│ └── pyvenv.cfg  
├── .env  
├── .gitignore  
├── app.py  
├── README.md  
└── requirements.txt

**6. Running the Application**

cd CITIZENAI

venv\Scripts\activate

set FLASK\_APP=app.py

set FLASK\_ENV=development

python app.py

**7. API Documentation**

| **Endpoint** | **Method** | **Body** | **Response** |
| --- | --- | --- | --- |
| /api/auth/register | POST | { name, email, password } | { token, user } |
| /api/auth/login | POST | { email, password } | { token, user } |
| /api/chat | POST | { query } | { response, schemeList } |
| /api/schemes | GET | — | [ { \_id, title, desc, eligibility }, ... ] |
| /api/admin/logs | GET | Requires JWT admin token | [ { userId, query, timestamp, response } ] |

**8. Authentication & Authorization**

* **JWT tokens** are assigned on login with 1-hour expiry
* Passwords hashed via bcrypt
* **Role-based Middleware** ensures admin-only routes
* **Session Security**: API access blocked if token expires or is tampered
* HTTPS ensures encryption of tokens in transit

**9. User Interface (UI)**

The CitizenAI platform offers a user-centric interface built with Flask and HTML templates, tailored to ensure simplicity, accessibility, and clarity across diverse user groups. Below are key UI components included in the system:

1. **Login / Register Screens**

The login and registration screens are designed to offer a seamless onboarding experience. Key features include:

* Clean and minimalist form layout with email and password fields
* Error handling for invalid credentials or mismatches
* Links to switch between login and registration forms
* Responsive design for mobile and desktop users
* Secure password fields with toggle visibility
* Optional registration via social platforms

**2. Chat Interface with AI Conversation**

The chat interface is the core component where users interact with the AI assistant. The layout is designed to be intuitive and responsive:

* Chat window on the right with a scrolling message pane
* Message input bar fixed at the bottom
* Displays both user messages and AI responses
* Option to upload documents (PDFs) or type free-text queries
* AI responds in real-time using IBM Watson or NLP model
* Context-aware, guided question prompts may be included

1. **Scheme Recommendation Panel**

This panel dynamically displays government schemes based on user queries or uploaded documents. Key UI elements include:

This internal-facing dashboard allows admin users to monitor platform usage and performance:

* Statistics on number of users, queries handled, and schemes recommended
* Logs of document uploads, login sessions, and AI accuracy reports
* Charts and graphs using chart.js or similar (if implemented)
* Access to sentiment analysis results (from sentiment\_model.py)

1. **Admin Dashboard (For Usage Insights)**

This internal-facing dashboard allows admin users to monitor platform usage and performance:

* Statistics on number of users, queries handled, and schemes recommended
* Logs of document uploads, login sessions, and AI accuracy reports
* Charts and graphs using chart.js or similar (if implemented)
* Access to sentiment analysis results (from sentiment\_model.py)

UI Technologies Used

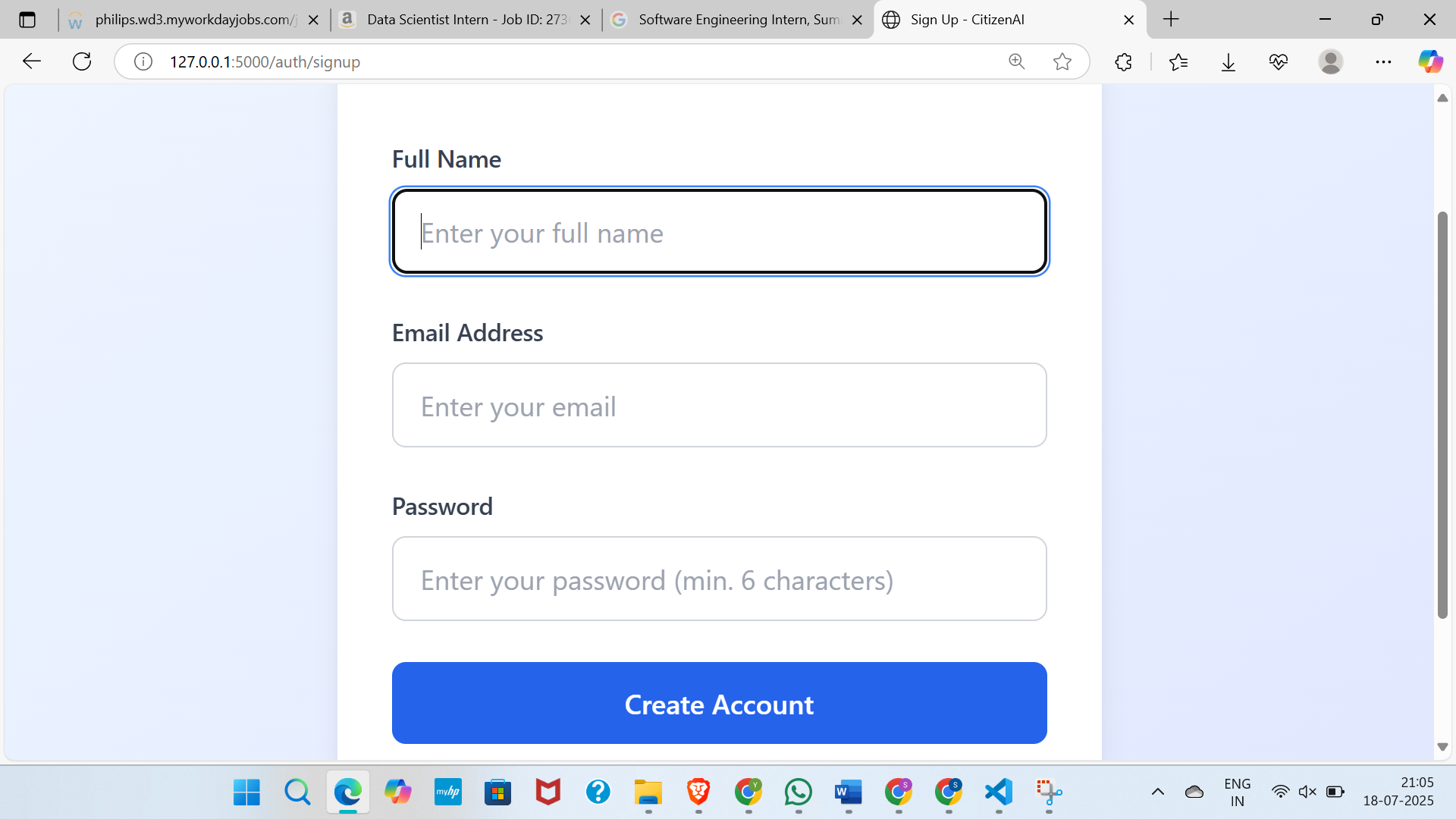
* HTML5, CSS3
* Bootstrap (for responsive layout)
* Flask templates (Jinja2)
* JavaScript (for dynamic interactions)
* Optional: Chart.js for graphs and analytics

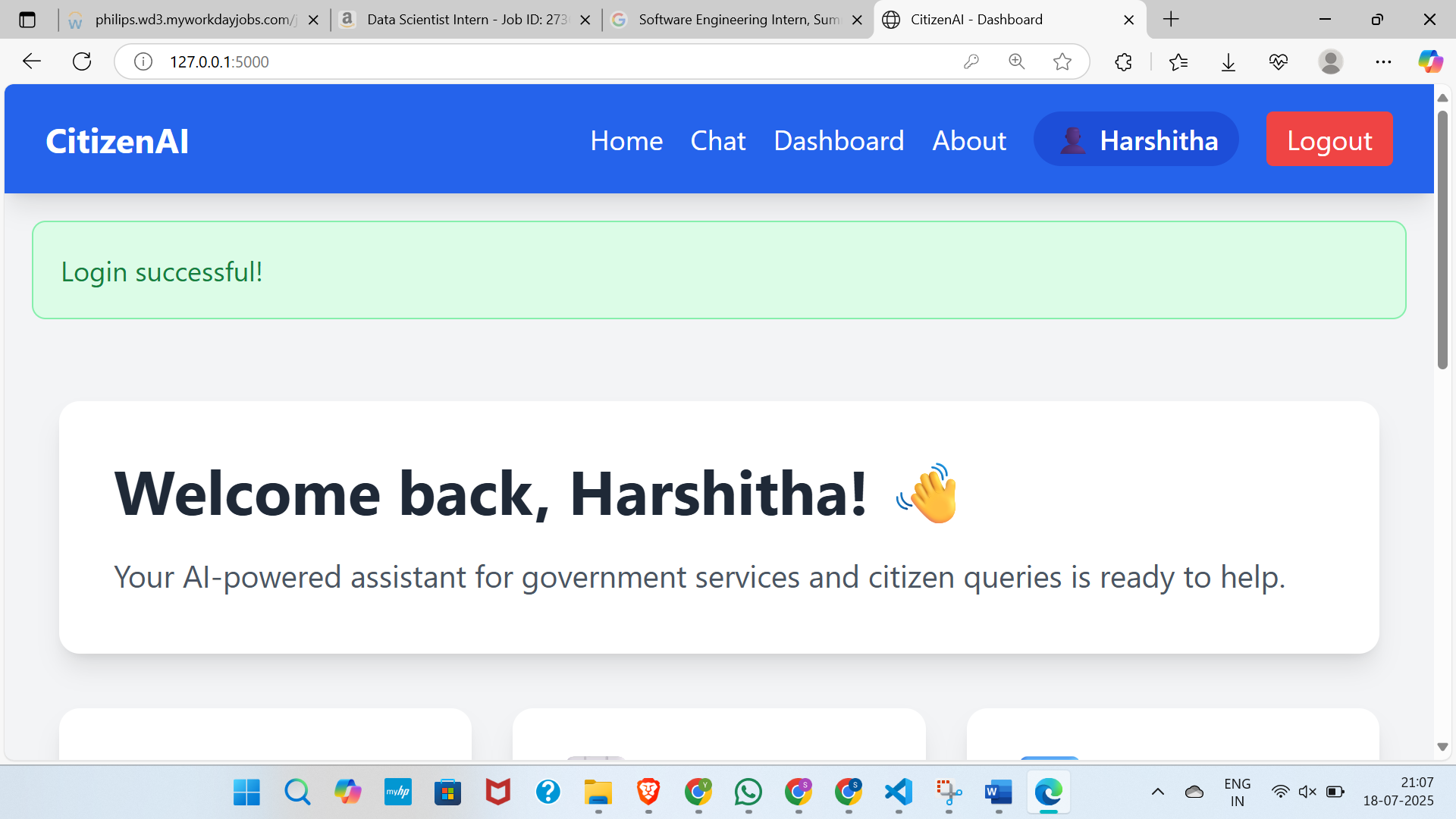
**10. Testing Strategy**

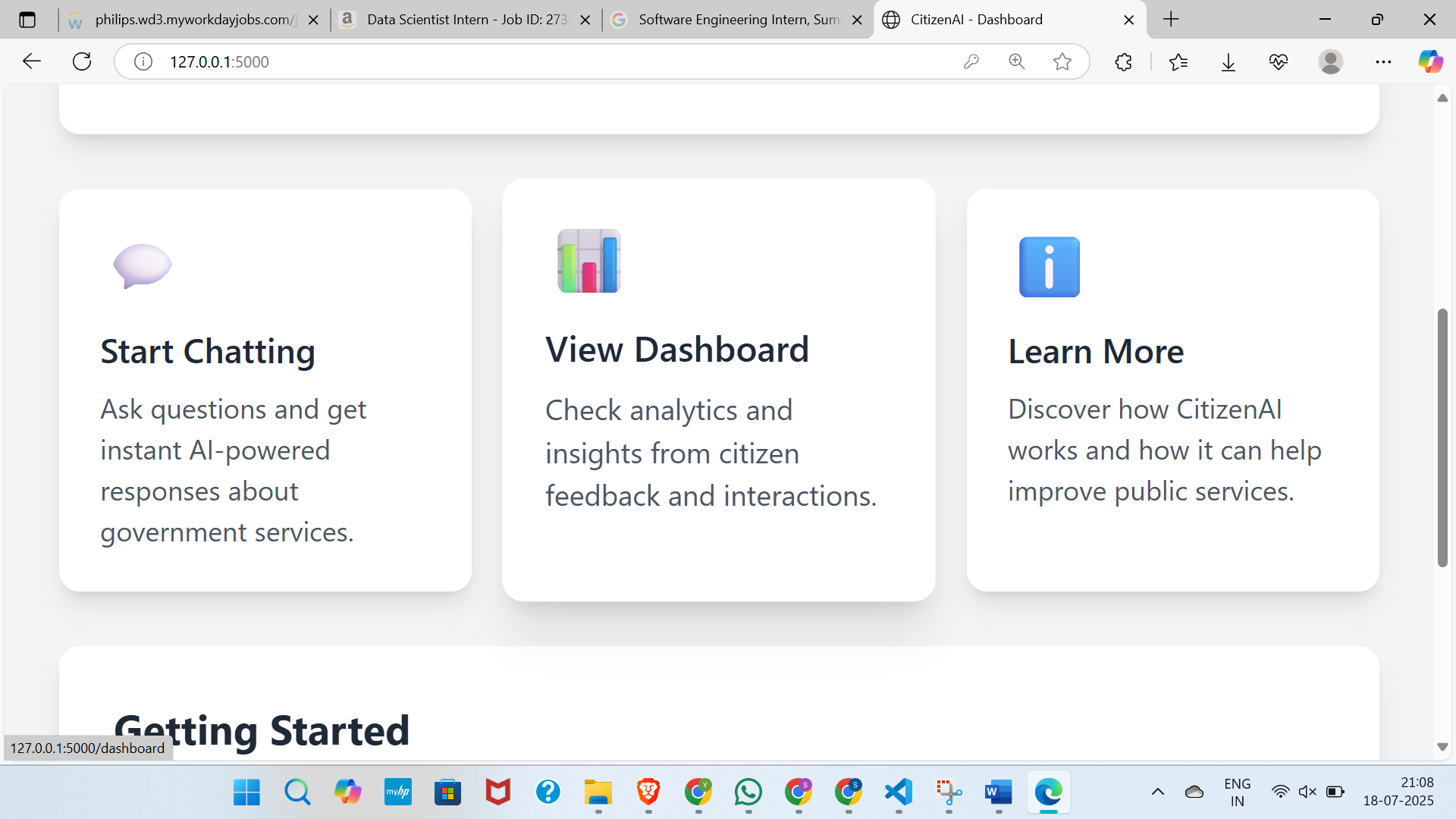
* **Backend**: Mocha + Chai for route tests, Supertest for request simulation
* **Frontend**: React Testing Library & Jest for components
* **End-to-end**: Postman test suite covering auth, chat, scheme retrieval
* **Performance**: Response time measured using Postman, should be under 2 seconds

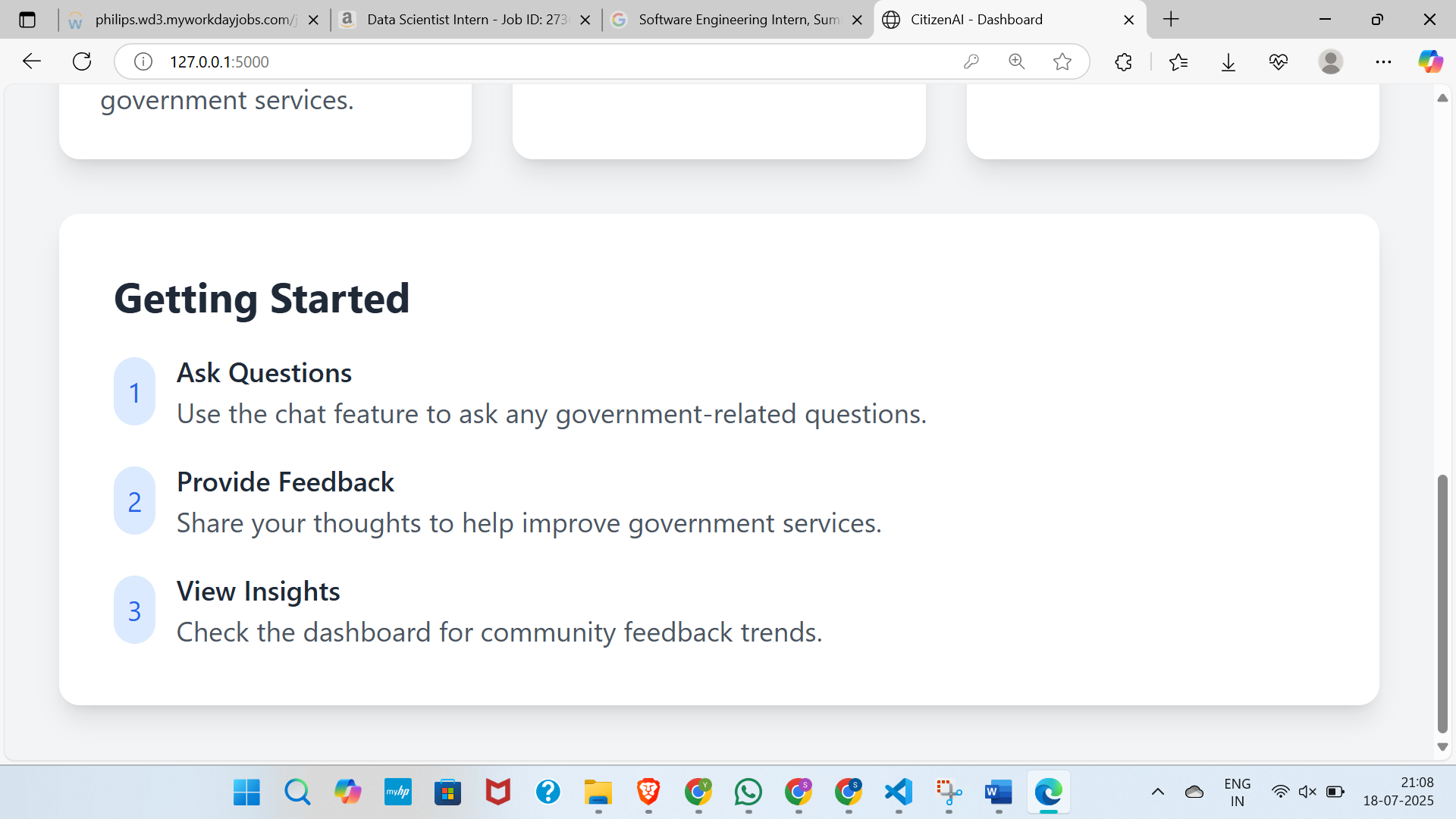
**11. Screenshots or Demo**

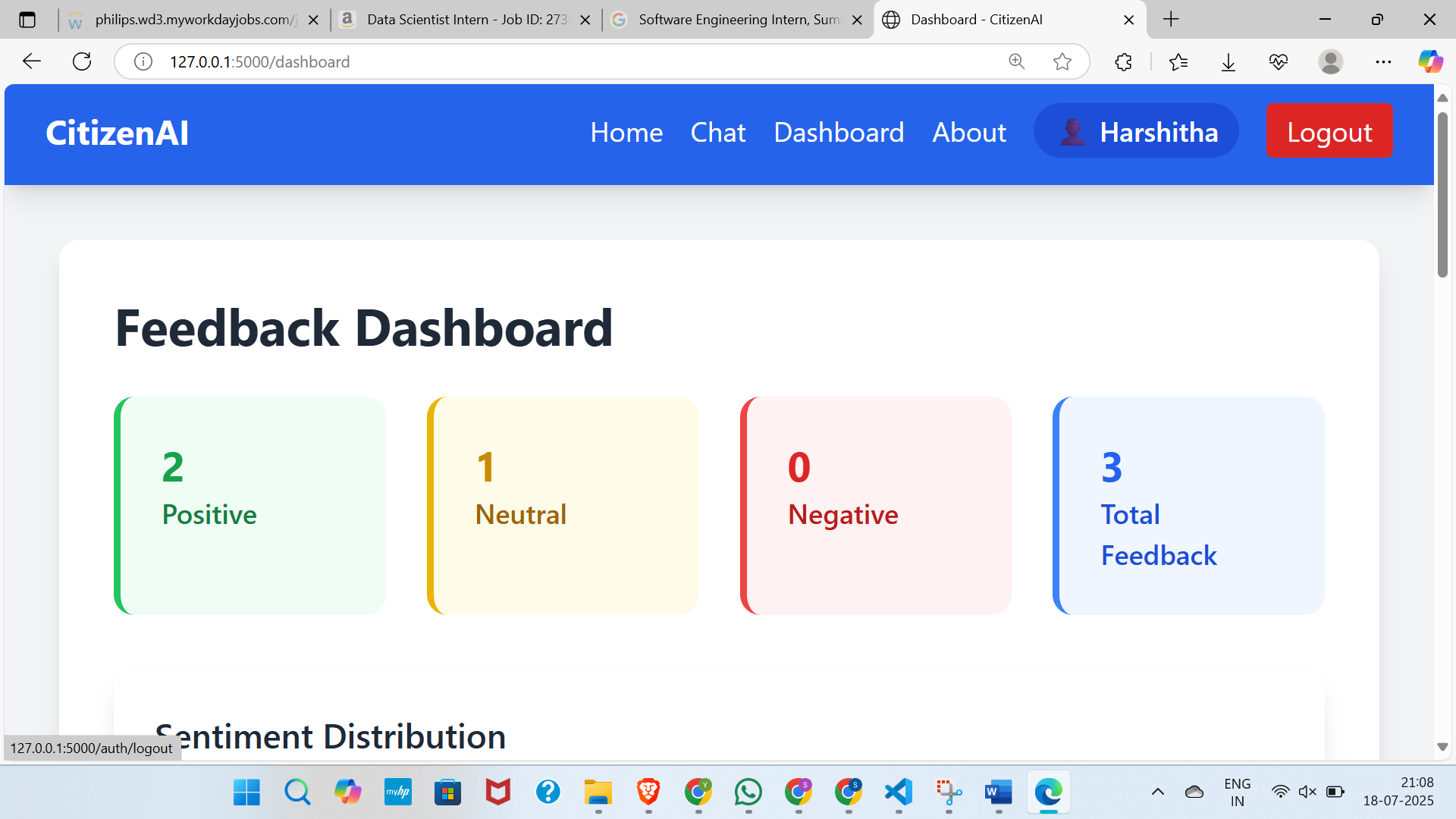


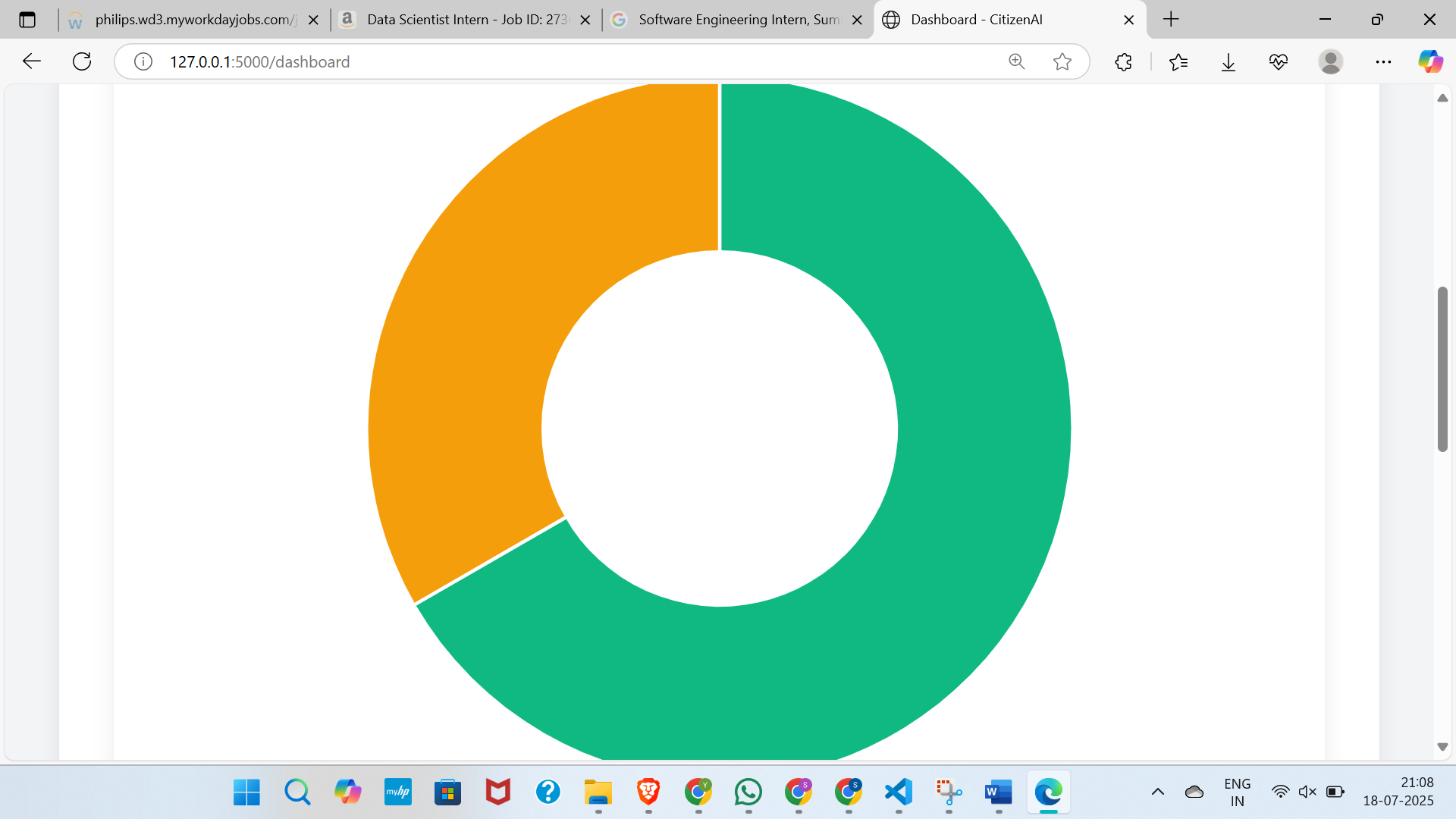


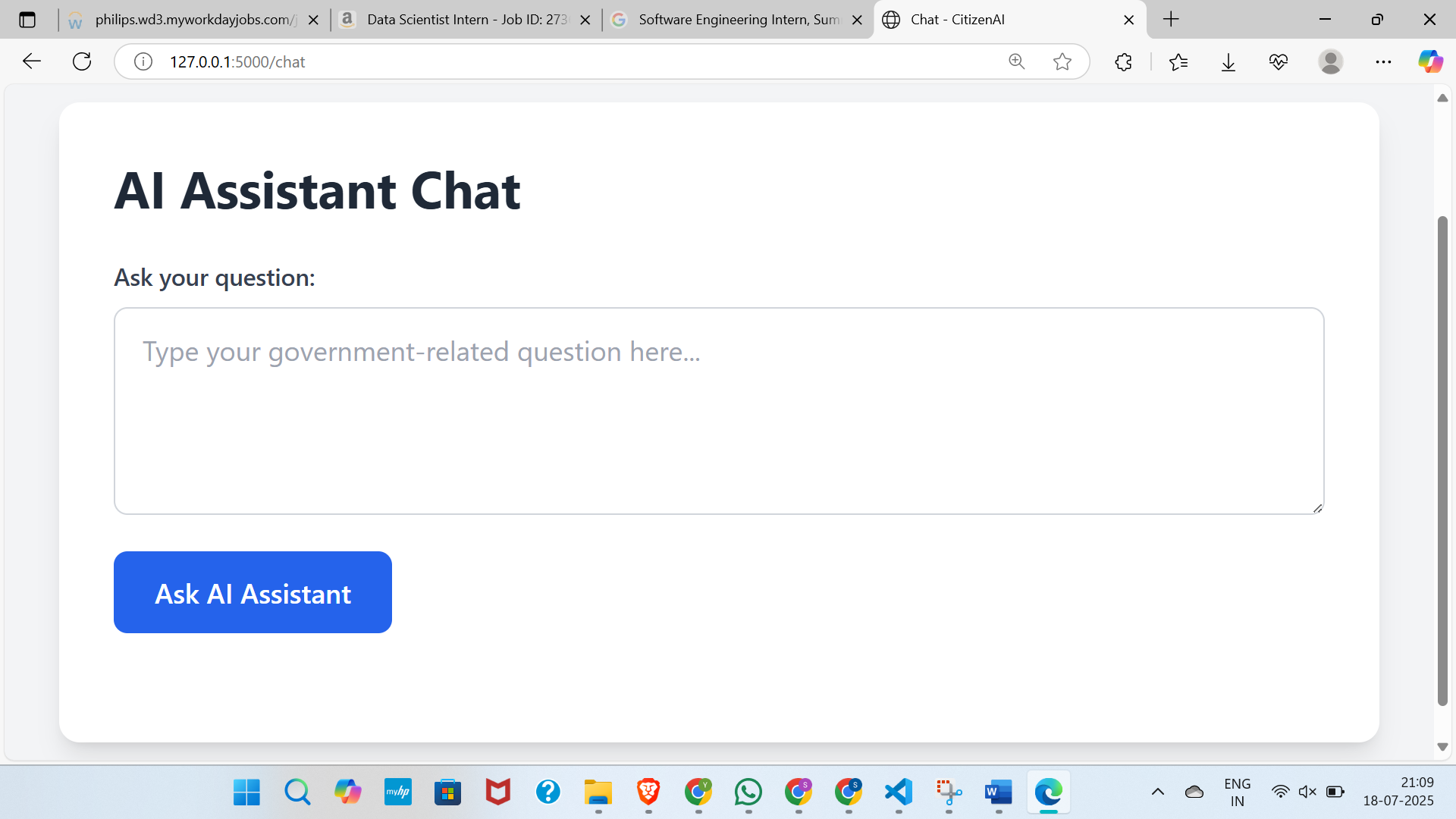


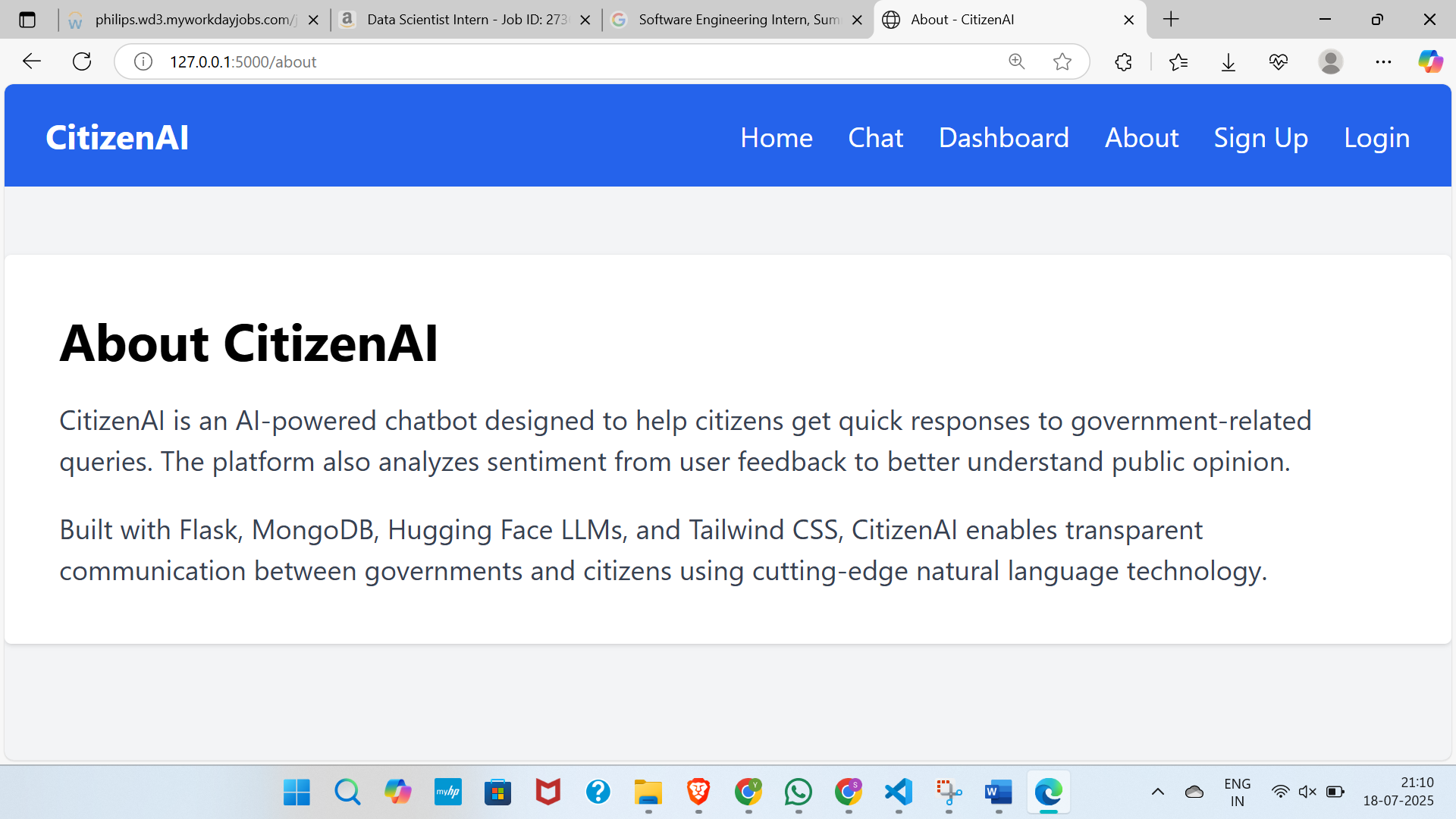


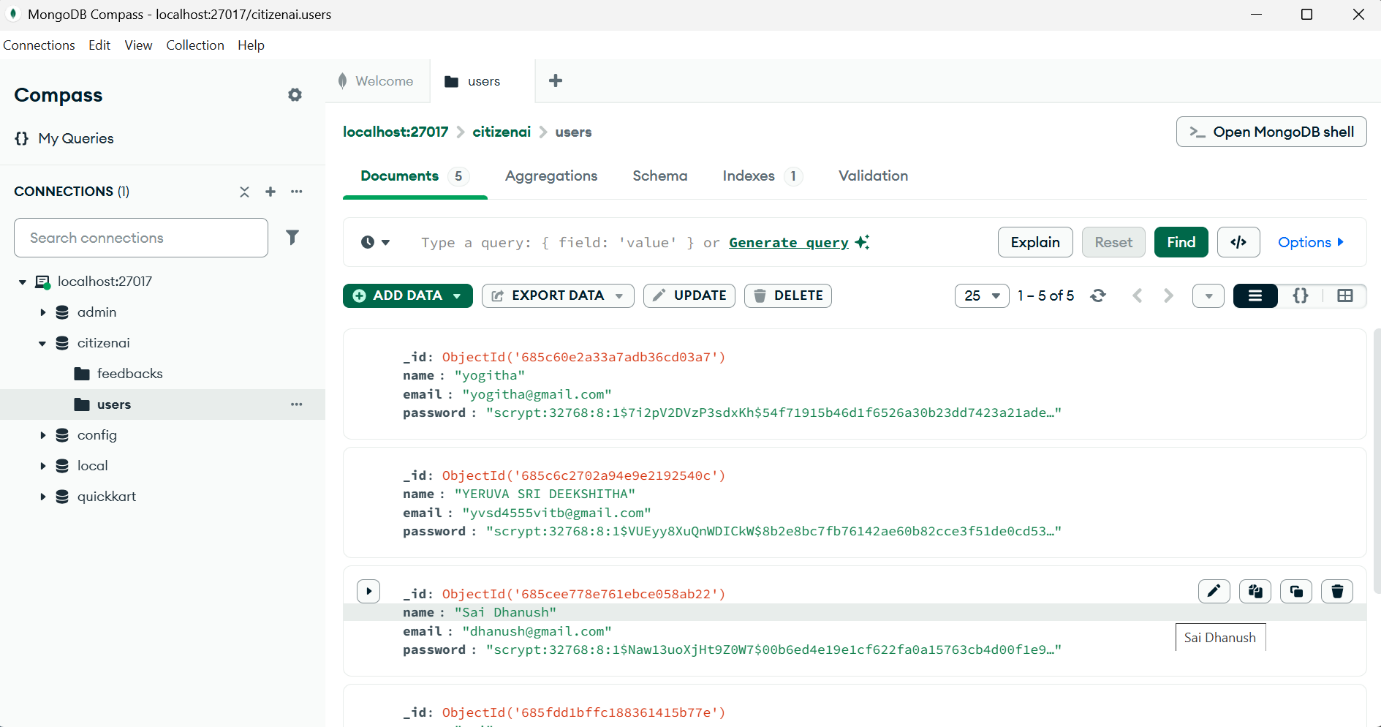










****

**12. Known Issues**

As with any evolving software system, CitizenAI currently has a few known limitations that the development team is working to address in upcoming versions. These issues, while not critical, may affect user experience in specific scenarios.

1. **Limited Response Accuracy for Hindi Inputs**  
   Although the platform supports basic multilingual interactions, certain queries submitted in Hindi—particularly those involving informal language or mixed Hindi-English—tend to produce incomplete or inaccurate responses. This limitation arises due to the constraints of the natural language processing (NLP) model and the lack of deep regional language training.

* Impact: Users submitting questions in Hindi may receive partial answers or irrelevant information.
* Temporary Recommendation: Users are advised to phrase their questions in English for more accurate results.
* Planned Fix: Integration of advanced multilingual models trained on Indic languages is under consideration.

1. **PDF Upload Size Restrictions**  
   The file upload feature enables users to extract schemes from government documents in PDF format. However, large files exceeding 5MB in size may lead to application slowdowns or timeouts, especially when hosted on resource-limited environments.

* Impact: Uploading heavy PDFs may result in failure or long wait times.
* Temporary Recommendation: Users should limit uploads to smaller or segmented documents.
* Planned Fix: The backend architecture is being redesigned to support background job queues and asynchronous file processing.

1. **Absence of Pagination in Admin Panel**  
   The admin interface currently displays all user logs and scheme records in a single list. As data grows, this design creates challenges in performance, page load speed, and user navigation.

* Impact: Admins may experience delays when viewing long lists of records.
* Temporary Recommendation: Manual filtering and search can help manage navigation.
* Planned Fix: Implementation of pagination and real-time search filters using JavaScript libraries is planned.

1. **No Two-Factor Authentication (2FA) or Brute Force Protection**  
   User authentication currently follows standard login procedures with secure password hashing. However, the system does not include additional security layers such as OTP verification, 2FA, or account lockout mechanisms after repeated login attempts.

* Impact: May reduce resilience against unauthorized access attempts.
* Planned Fix: Future versions will include OTP-based verification and basic rate-limiting measures.

1. **Mobile Interface Optimization Needed**While the user interface is responsive, the experience on smaller mobile screens may not be fully optimized. Buttons, input fields, and layout may require adjustments for a better mobile experience.

* Impact: Users on mobile devices may face minor display or navigation issues.
* Planned Fix: A mobile-first UI redesign is in the roadmap using modern responsive frameworks.

**13. Future Enhancements**

As part of the long-term roadmap for CitizenAI, several valuable enhancements have been identified to further expand the platform’s reach, usability, and intelligence. These improvements aim to enhance accessibility, personalize user interactions, and deepen integration with government systems and services.

1. **Voice Chat Integration (Watson Speech-to-Text)**  
   To improve accessibility and inclusivity, a voice-based chat interface is planned. By integrating IBM Watson Speech-to-Text, users will be able to converse with the platform using spoken Hindi or English. This enhancement will especially benefit visually impaired individuals and users who are less comfortable with typing.

* Benefits: Hands-free interaction, improved accessibility, and enhanced user engagement.
* Planned Workflow: Voice input → Transcription via Watson STT → Text processing by AI engine → Voice output (TTS optional).

1. **Mobile Application (React Native Implementation)**A cross-platform mobile app is planned using React Native. This will allow users to access CitizenAI on both Android and iOS devices with a native experience.

* Benefits: Broader accessibility, offline capabilities, and better reach in rural areas.
* Key Features: Voice input, push notifications, offline document upload, mobile-optimized chat interface.

1. **Multi-Region Support (Indian States & Languages)**  
   Currently, the platform supports a limited set of schemes. The next phase will expand coverage across multiple Indian states, enabling regional language support and region-specific scheme recommendations.

* Benefits: Tailored recommendations based on user location and demographics.
* Implementation: Dynamic scheme database filtered by state selection, regional filters, and integration of multilingual NLP models.

1. **Government API Integration (Aadhaar, DigiLocker)**To enhance personalization and automate eligibility checks, future releases aim to incorporate secure government APIs such as Aadhaar authentication and DigiLocker document access.

* Benefits: Streamlined user verification, real-time scheme eligibility checks, reduced manual data entry.
* Considerations: Compliance with data privacy laws and secure encryption protocols (e.g., OAuth 2.0, PKI).

1. **Intelligent User Tracking & Predictive Analytics**  
   Leveraging machine learning, the platform will begin tracking user queries and interaction patterns to improve scheme recommendations. This includes personalizing scheme results based on behavior, history, and preferences.

* Benefits: Smarter, data-driven suggestions and proactive alerts for newly applicable schemes.
* Planned Tools: Integration with a recommendation engine using user segmentation and real-time feedback loops.