Project Title :Inteligent citizen engagement platform.

1. Introduction:

Team ID : NM2025TMID06775

Team Size : 4

Team Leader : VENNILA G

Team member : SINDHU M

Team member : SHALINI B

Team member : SATHYA S

Summary

- **Purpose**:Improvecommunicationandcollaborationbetweencitizensand government.
- **Technology**:UsesAI,chatbots,analytics,anddigitaltoolstoenhanceengagement.
- **Functions**:
  - Two-waycommunication(chatbots,apps,portals).
  - Real-timeissuereportingandtracking.
  - Surveys,polls,andparticipatorybudgeting.
  - Dataanalysistoguidepublicdecisions.
- **Outcomes**:
  - Fasterservicedelivery.
  - Morecitizeninvolvement.
  - Bettertrustinpublicinstitutions.
  - Smarter,data-drivengovernance.
-

Milestone1:ProjectSetupandArchitecture

Inthismilestone, wefocusonconfirmingthecoreAImodelandlibraries, definingtheoverallsystem structure, and setting up the development environment.

Activity1.1:SelectandConfirmAIModel

1. ReviewthekeyfunctionalitiesofCitizenAI,includingchatresponses,sentimentanalysis, concern reporting, and dashboard insights.
2. IdentifythetypeofAIcapabilitiesneeded:naturallanguageunderstanding(NLU),text generation, and basic text analysis.

ConfirmAIModel&Libraries:

1. ConfirmtheselectionoftheIBMGranitemodelforcoreAI capabilities.
2. SpecifythenecessaryPythonlibraries:Flaskforthewebframework,PyTorchfortheAI modelbackend,andHuggingFacelibraries(transformers,accelerate,bitsandbytes)for model handling.

ExploreLibraryDocumentation:

1. Reviewdocumentationforselectedlibrariestounderstandmodelloading,inference, quantization, and device handling.
2. ExamineFlaskdocumentationforrouting,templating,andsessionmanagement.

Activity 1.2: Define the Architecture of the Application

- DraftanArchitectural Overview:

1. Design a structured architecture including the Flask backend, HTML/CSS frontend, integrationpointsfortheIBMGraniteAImodel,andtheapproachfordatahandling(initially in-memory history, planning for future persistent storage).
2. Definehowuserrequestsanddatawillflowthroughthesystem components.

DefineDataFlow:

1. Mapthefloworuserinput(chatmessages,feedback,concerns)tothebackend,AI processing, data storage, and back to the frontend for display.

Activity1.3:SetUptheDevelopment

- **EnvironmentInstall Necessary Tools:**

2. EnsurePython(3.7+)andpipareinstalledformanagingprojectdependencies.

InstallFlaskand Dependencies:

- UsepiptoinstallFlaskandanyotherrequiredbackendlibraries: pip

install Flask

Install AI/ML Libraries:

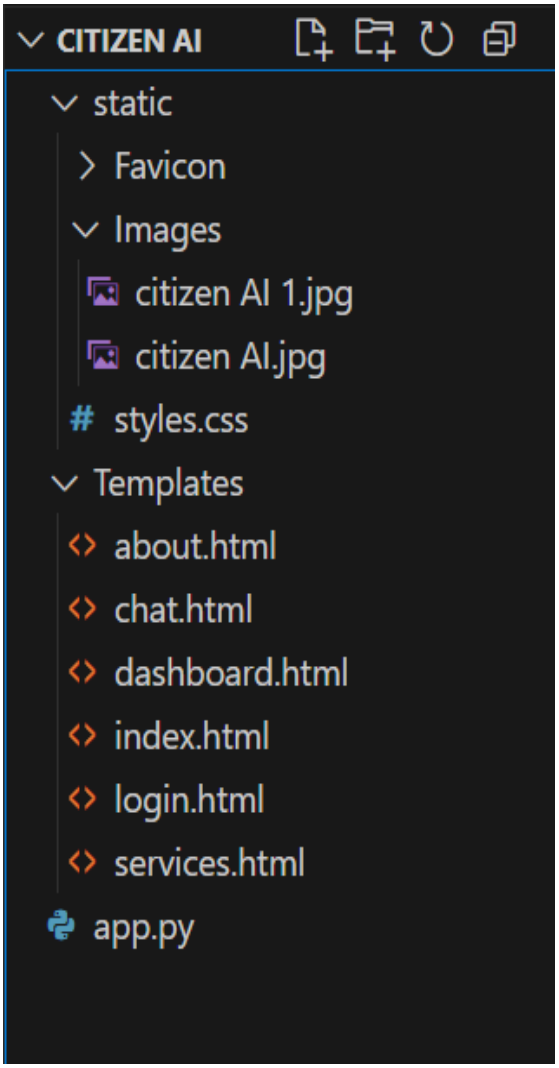
- InstallthenecessarylibrariesforAImodelintegration: pip

install torch transformers accelerate bitsandbytes

(EnsureyouinstallthecorrectPyTorchversionforyourCUDAsetupifusingaGPU).

Set Up Application Structure:

- Createthebasicprojectdirectorystructure:app.py,templates/(forHTMLfiles), andstatic/(withcss/,Images/,Favicon/subfolders).



Milestone2:CoreFunctionalities

Core Functionalities Development This milestone focuses on building the essential backend capabilities of the Citizen AI platform. It includes implementing Flask routes and establishing user authentication and session management. A core activity is integrating the IBM Granite AI modeltohandlecitizenqueries,alongsidedevelopingthelogicforfeatureslikechatresponses, sentiment analysis, and concern reporting.

Activity2.1:DeveloptheCoreFunctionalities

Activity2.1:DeveloptheCore Functionalities

```
<!-- Chat Section -->
<section class="chat-section">
    <h2>Ask the Assistant</h2>
    <form action="/ask" method="POST">
        <label for="question">Ask a Question:</label>
        <input type="text" id="question" name="question" required />
        <button type="submit">Submit</button>
    </form>

    {% if question_response %}
    <div class="response">
        <h3>Response:</h3>
        <div class="response-text">
            {{ question_response | safe }}
        </div>
    </div>
    {% endif %}
</section>
```

This HTML code defines the Chat Section of the page. It includes a form titled "Ask the Assistant" where users can input a question using a text field and submit it with a button. The form sends the questiontothe/askrouteonthebackend. Belowtheform, there'sasectiondesignedtodisplaythe

response.Ifaquestion_responsevariableisreceivedfromtheserver,itscontentisrenderedhere, showing the user the assistant's answer to their question.

SentimentAnalysis:

```
<!-- Sentiment Analysis Section -->
<section class="sentiment-section">
    <h2>Feedback Sentiment</h2>
    <form action="/feedback" method="POST">
        <label for="feedback">Enter Feedback for Sentiment:</label>
        <textarea id="feedback" name="feedback" rows="4"></textarea>
        <button type="submit">Submit</button>
    </form>

    {% if sentiment %}
    <div class="response">
        <h3>Sentiment:</h3>
        <p>{{ sentiment }}</p>
    </div>
    {% endif %}
</section>
```

ThisHTMLcodedefinestheSentimentAnalysisSectionofyourapplication.Itprovidesaformtitled "Feedback Sentiment" where users can enter their feedback into a text area and submit it. The submitted feedback is sent to the /feedback route for backend processing. The code checks if a sentiment result is available from the server. If a sentiment value (such as Positive, Negative, or Neutral) is present, it is displayedinaresponsearea, allowingusers toseetheanalyzedsentiment of their feedback.

ConcernSubmission:

```
        <!-- Concern Submission Section -->
        <section class="concern-section">
            <h2>Report a Concern</h2>
            <form action="/concern" method="POST">
                <label for="concern">Describe Your Concern (for Issue Identification):</label>
                <textarea id="concern" name="concern" rows="4"></textarea>
                <button type="submit">Submit</button>
            </form>

            {% if concern_submitted %}
            <div class="response">
                <h3>Concern Submitted:</h3>
                <p>Your concern has been recorded.</p>
            </div>
            {% endif %}
        </section>
    </main>
</body>
</html>
```

ThisHTMLcodesnippetcreatesaConcernSubmissionSectionin awebpage,allowingusersto report issues:

1. The<section> blockwithclassconcern-sectionholdstheentireform.
2. Itincludesaheading(<h2>Report aConcern</h2>)tolabelthesection.
3. The<form> sendsaPOST request tothe/concernroutewhensubmitted.
4. Insidetheform, there'sa<label> anda<textarea> whereuserscantypetheirconcern.
5. The<buttontype="submit"> allowsuserstosubmittheir input.
6. Theformfieldisnamed"concern",whichisusedbytheservertoretrievethesubmittedtext.
7. Aftersubmission,ifconcern_submittedisTrue,aconfirmationmessageisshown.

8. ThisisdoneusingJinjatemplating: {%if concern_submitted%}and{%endif

%}.

9. Iftriggered,a<divclass="response">appearswithasuccessmessage.
10. ThissnippetisdesignedforintegrationwithaFlask backendthathandlesformsubmissions and sets concern_submitted accordingly.

Activity2.2:ImplementtheFlaskBackendfor

Managing Routing and User Input Processing

This activity involves writing the Python code in app.py to define the web application's structure, handledifferentpagerequests,processdatasubmittedbyusers,andintegratewiththeAImodel logic.

• DefineRoutesinFlask:

1. Setupdistinctroutesinapp.pyusingthe@app.route() decoratorforeachpageandkey interactionpoint:/(Home),/about,/services,/chat,/dashboard,/login(GETandPOST), /logout.
2. Defineroutesspecificallyforhandlingformsubmissions:/ask(forchatquestions),/feedback (for sentiment analysis input), and /concern (for reporting issues), ensuring they accept POST requests.
3. LinkeachdefinedroutetoacorrespondingPythonfunctionthat willexecutewhenthat URL is accessed or form is submitted.
4. EnsurethatwithineachroutefunctionitheappropriateHTMLtemplateisrenderedusing render_template(), passing any necessary data to the template.

a. ProcessUserInput:

1. EnsurethatHTMLforms(e.g.,inchat.html,login.html) havethecorrectmethod="POST"and action="{{ url_for('route_name') }}" attributes to send data to the defined backend routes.
2. WithintheFlaskfunctionshandlingPOSTrequests(e.g.,ask_question(), submit_feedback(), login()), use Flask’s request.form to safely retrieve data submitted by the user from theHTML forms (e.g., request.form.get('question'), request.form.get('username')).
3. Performanynecessarybasicvalidationontheretrieveduserinput(e.g.,checkingiffields are empty).

- **IntegrateAIModelCallsandLogic:**

1. WithintheroutefunctionsthatrequireAIprocessing(e.g.,the/askroute),callthepreviously implemented AI helper functions (like granite_generate_response()).
2. Passtheprocesseduserinput(e.g.,theuser'squestion)asargumentstotheAIhelper functions.
3. Inrouteshandlingfeedbackorconcerns(e.g.,/feedback,/concern),calltherelevant processing functions (like analyze_sentiment()) with the user- provided text.
4. CaptureandprocesstheresultsreturnedbytheAIhelperfunctions ordataprocessinglogic (e.g., the generated text response, the sentiment label).
5. Prepareretheresultstobesent backtothefrontendby passingthemasargumentstothe render_template() function (e.g., render_template("chat.html", question_response=response)).

Milestone3:ApplicationLogicandDataHandling

This milestone is dedicated to implementing the specific functionalities of the CitizenAI platform. It involves developing the core logic for processing chat interactions and performing sentiment analysis on feedback. Setting up the application's data storage (initially in-memory history) and creatingthelogictopreparedataforthedashboardviewarealsokeycomponentsofthismilestone.

Activity3.1:Writing the Main Application Logicin app.py

This activity involves implementing the Python functions within app.py that define how the applicationrespondstospecificuseractionsand integratesthecoreAIanddataprocessing functionalities.

1. **DefinetheCoreRoutesinapp.py**

SetupseparateFlaskroutesusingthe@app.route()decoratorinapp.pyforthespecificactionsthat involve processing user input and triggering application logic:

1. /ask?Acceptsauser'squestionfromthechatinterfaceandgeneratesa

   response using the IBM Granite model.

2. /feedback?Acceptsuserfeedbacktextandperformssentimentanalysison it.

3. /concern?Acceptsauser'sreport of aconcernorissueforlogging.
4. /login(withmethods=['POST'])?Handlesthesubmissionofusernameand

password for user authentication.

Eachoftheserouteswillserveastheentrypointinyourbackendtoprocessspecificuserrequests and initiate the corresponding application logic.

2. SetUpRouteHandlersfor Each Feature

Foreachoftheroutesdefinedabove,implementthecorrespondingPythonfunctioninapp.py. These functions act as the handlers for incoming requests to these routes:

1. Capture user inputs from the HTML forms associated with these routes using request.form.get('input_name'),ensuringsaferetrievalofdatalikethequestiontext, feedback content, concern details, username, and password.
2. (Optionalbutrecommended)Includebasicvalidationstepstocheckifsubmitteddatais present and in the expected format before proceeding with processing.
3. Preparethe datatobepassedtothenextstepsinthe workflow(e.g.,toAIfunctionsordata storage).
4. Determinetheappropriateresponsetosendbacktotheuser,whichtypically involves rendering an HTML template (render_template()) or redirecting to another page (redirect()).

For example:

1. The/askhandlerwillretrievethequestiontextsubmittedviathechatform.
2. The/feedbackhandlerwillretrievethefeedbacktext enteredbytheuser.
3. The/login(POST)handlerwillretrievetheenteredusernameandpassword.

3. IntegrateAIModelCallsand Logic in Each Function

Withintherelevantroutehandlerfunctionsdefinedinstep2,integratethecallstoyourAImodel and other application logic:

1. In the /ask route handler function (e.g., ask_question()), implement the call to your IBM Graniteinferencefunction(e.g.,granite_generate_response()),passingtheuser'squestion as input.
2. Inthe/feedbackroutehandlerfunction(e.g.,submit_feedback()),implementthecalltoyour sentiment analysis function (e.g., analyze_sentiment()), passing the user's feedback text.
3. Processtheresultsreturnedbythesefunctions(e.g., storethesentimentresult,getthe generated text).
4. FormattheAI-generatedoutput orprocessingresultsasneededfordisplayonthefrontend.
5. Pass the final results to the render_template() function to display them clearly on the appropriateHTMLpage(e.g.,passingthegeneratedresponsetochat.html,passingthe sentiment result to chat.html).

For example:

1. Theask_question()functionwillcallgranite_generate_response()withtheuser's questionandthenrenderchat.html,includingtheoriginalquestionandtheAI's response.

2. Thesubmit_feedback()functionwillcallanalyze_sentiment()withthefeedbackandthen render chat.html, indicating the sentiment result.
3. **HTMLpages Routes:**

```python
# ------------------ Routes ------------------

@app.route('/')
def index():
    # Index page - does NOT require login
    return render_template("index.html")

@app.route('/about')
@login_required # Requires login to access
def about():
    return render_template("about.html")

@app.route('/services')
@login_required # Requires login to access
def services():
    return render_template("services.html")
```

Chat.htmlpageRoute:

```python
@app.route('/chat')
@login_required # Requires login to access
def chat():
    return render_template("chat.html")

@app.route('/ask', methods=['POST'])
@login_required # Requires login to access the chat functionality
def ask_question():
    question = request.form['question']
    response = granite_generate_response(question)
    # When rendering chat.html after a POST, pass the necessary data
    return render_template("chat.html", question_response=response)
```

SentimentAnalysisRoute:

```python
@app.route('/feedback', methods=['POST'])
@login_required # Requires login to submit feedback
def submit_feedback():
    feedback = request.form['feedback']
    sentiment = analyze_sentiment(feedback)
    history.append({
        'date': datetime.now(),
        'sentiment': sentiment,
        'issue': '' # Issue will be handled in the concern route
    })
    # When rendering chat.html after a POST, pass the necessary data
    return render_template("chat.html", sentiment=sentiment)
```

ConcernSubmissionRoute:

```python
@app.route('/concern', methods=['POST'])
@login_required # Requires login to submit a concern
def submit_concern():
    concern = request.form['concern']
    history.append({
        'date': datetime.now(),
        'sentiment': 'Neutral', # Sentiment might be added later or remain neutral if only concern is submitted
        'issue': concern
    })
    # When rendering chat.html after a POST, pass the necessary data
    return render_template("chat.html", concern_submitted=True)
```

DashboardpageRoute:

```python
@app.route('/dashboard')
@login_required # Requires login to access
def dashboard():
    # Dashboard page - requires login
    last_7_days = datetime.now() - timedelta(days=7)
    filtered = [h for h in history if h['date'] > last_7_days]

    sentiment_counts = {'Positive': 0, 'Neutral': 0, 'Negative': 0}
    issues = []

    for h in filtered:
        sentiment_counts[h['sentiment']] += 1
        if h['issue']:
            issues.append(h['issue'])

    return render_template("dashboard.html", data={
        'positive': sentiment_counts['Positive'],
        'neutral': sentiment_counts['Neutral'],
        'negative': sentiment_counts['Negative'],
        'issues': issues
    })
```

LoginpageRoute:

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    # Login page - does NOT require login
    if request.method == 'POST':
        username = request.form.get('username') # Use .get() for safer access
        password = request.form.get('password') # Use .get() for safer access

        # --- Basic Authentication (Replace with secure method) ---
        # In a real application, you would check username and password against a database
        # Updated credentials
        if username == "lasya@gmail.com" and password == "lasya":
            session['user'] = username # Store username in session
            # Redirect to the page the user was trying to access, or to the about page if no next URL
            next_url = request.args.get('next')
            # Redirect to the about page if no specific 'next' URL was provided
            return redirect(next_url or url_for('about'))
        else:
            # Handle invalid login (e.g., show an error message on the login page)
            return render_template("login.html", error="Invalid email or password")
    # For GET request, render the login form
    return render_template("login.html")
```

LogoutpageRoute:

```python
@app.route('/logout')
def logout():
    # Logout route - logs the user out and redirects to index
    session.pop('user', None) # Remove user from session
    return redirect(url_for('index')) # Redirect to home page after logout
```

Milestone4:FrontendDevelopment

Thismilestonefocusesoncreatingtheuserinterfacefor CitizenAI.Itinvolvesdesigningthelayoutof each page using HTML and applying styling with CSS to ensure a user-friendly and visually appealing experience. Building interactive components like forms for chat, login, feedback, and concerns, and ensuring they correctly display data from the backend, are key tasks.

Activity4.1:DesigningandDevelopingtheUser

Interface

Thisactivityinvolvescreatingthestructureandvisualappearanceofallthewebpagesinyour CitizenAI application using HTML and CSS.

1. **SetUpthe BaseHTMLStructure**

DevelopthenecessaryHTMLfilesthatrepresenteach pageofyourapplication.

1. CreatethemainHTMLtemplatefiles:index.html,about.html,services.html,chat.html, dashboard.html, and login.html.
2. IncludethestandardHTML5boilerplate(<!DOCTYPEhtml>, <html>,

<head>,<body>)ineachfile.

3. Incorporateaconsistentheaderandnavigationmenuinpageswhereappropriate(e.g.,on protected pages after login) to allow users to easily access different sections like:
    a. About
    b. Services
    c. Chat
    d. Dashboard
    e. Login/Logout(conditionaldisplaybasedonsession)

UsesemanticHTMLelementssuchas<header>, <nav>,<main>, <section>, <footer>,<h1>,<p>,<ul>,<li>,<form>,<input>,<button>,<textarea>tocreateaclean, organized, and accessible page structure.

2. DesigntheLayoutandStyling Using CSS

CreateandapplyCSSrulestocontrolthevisualpresentationandlayout of yourwebpages.

1. CreateorupdatethemainCSSstylesheet (static/css/styles.css)todefinetheoveralllook and feel, including font styles, colors, and spacing for common elements.
2. ImplementlayouttechniqueswithinyourCSS(e.g.,usingFlexboxforcenteringelementslike the login box, or adjusting margins and padding for content areas) to arrange elements on the page effectively.
3. (Optionalbutrecommended)ConsiderimplementingmediaqueriesinyourCSStoensure the layout and styling are responsive and look good on different screen sizes (desktops, tablets, mobile phones).
4. Applyspecificvisualstyles(backgrounds,borders,textcolors)toindividual elementsorsections,potentiallyusinginline<style>blocksinspecificHTML

filesforpage-uniquestyles(likethebackgroundimage onindex.htmlor about.html).

3. CreateSeparatePagesfor Each Core Functionality

Developthespecificcontentandinteractiveelementsforeachdistinctpageofthe application.

1. index.html: DesignthelandingpagewithanintroductiontoCitizenAIandaclearcallto action (e.g., the "Get Started" button linking to login).
2. login.html:Createthepagewiththeloginform,includinginputfieldsforusername/emailand password, and a submit button. Include a placeholder for displaying login error messages.
3. about.html:Developthepagecontaininginformationabouttheproject'smission,features, and impact.
4. services.html:CreateapagedetailingtheservicesofferedbyCitizenAI.
5. chat.html: Design the interface for the AI chat assistant, including a form for user input (question)andadedicatedareatodisplaytheAI'sgeneratedresponse.Thispagemight also include forms for submitting feedback and concerns.
6. dashboard.html:Buildthepagetodisplay aggregateddata,suchassentimentcountsanda list of reported issues.

Eachofthesepageswillcontainthenecessaryuserinputformsanddesignatedareaswhere dynamic content from the backend (like AI responses or dashboard data) will be displayed.

Activity4.2:CreatingDynamicTemplateswith

Flask'srender_template

ThisactivityfocusesonusingFlask'sbuilt-intemplatingenginetopopulatetheHTML structures created inActivity4.1withdynamic datagenerated by thebackend.

1. Integrate Flask’s render_templateforDynamic Content Rendering:

WithineachFlask routefunctioninapp.pythat servesanHTMLpage:

1. Utilizetherender_template('filename.html',...)function.Thisfunctiontakesthenameofthe HTML file located in the templates folder as its primary argument.
2. Pass Python variables containing dynamic data as keyword arguments to the render_template() function (e.g., render_template('chat.html', question_response=ai_response,sentiment=feedback_sentiment)).Thismakesthese variables accessible within the specified HTML template.
3. IntheHTMLtemplates,useJinja2templatesyntax({{variable_name}}todisplayvariable values, {%if condition%} for conditionalrendering, {%for item inlist %} for loops, etc.) to access and display the data passed from the Flask backend. This allows the frontend to dynamically render AI-generatedresponses, sentiment results, dashboardstatistics, error messages, and other variable content based on the backend's processing.

Activity4.2.1:BindBackendDatato HTML Templates

BindingbackenddatatoHTMLtemplatesiscrucialfor dynamicwebpagesinFlask.Yourapp.py processesuserinput andgeneratesresults, suchasAI responsesorsentiment analysis. Flask's render_template() function then sends these results as variables to your HTML files. Within the HTML, Jinja2 templating syntax, like {{ variable_name }}, is used to display the value of these variables.Thismakesthecontentshowntotheuserupdatedynamicallybasedonthebackend's processing.

Milestone5:Deployment

This milestone focuses on preparing and launching the CitizenAI application. It involves setting up thedeploymentenvironment,ensuringallnecessaryPythonlibraries(Flask,PyTorch,HuggingFace libraries) and dependencies are correctly installed. The primary goal is to launch the Flask applicationlocally toverify that theappruns smoothly inatest environment, allowingfor finaltesting and refinement before potential cloud deployment.

Activity5.1:SetUpaVirtualEnvironment

Toensuredependencyisolation,createandactivateavirtualenvironmentbeforeinstallingrequired packages.

“python -m venv env

sourceenv/bin/activate(Linux/Mac)
env\Scripts\activate(Windows)pipinstall-r
requirements.txt”

ThisensuresthatFlask,GeminiAPIlibraries,andotherdependenciesareinstalledandavailable.

5.2:ConfigureEnvironment Variables

```
# ------------------ Model Setup ------------------
# Note: Running large models locally requires significant resources (GPU, RAM)
model_path = "ibm-granite/granite-3.3-8b-instruct"

# Determine the device to use (GPU if available, otherwise CPU)
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")

# Load the tokenizer first
tokenizer = AutoTokenizer.from_pretrained(model_path)
```

ThisPythonsnippetsetsupanIBMGraniteAImodelforuseinaproject:

1. model_pathisdefinedas"ibm-granite/granite-3.3-8b-instruct",specifyingthe path to the pretrained IBM Granite model.

2. Acommentnotesthatlargemodelsrequireconsiderablehardwareresources(RAM,GPU).
3. The codedetermines whether a GPU (cuda)is available usingtorch.cuda.is_available().
4. IfaGPUisavailable, itsetsdevice="cuda";otherwise,it defaultsto"cpu".
5. Itprintsthechosendevicefortransparency: Usingdevice:cudaorcpu.
6. AutoTokenizer.from_pretrained(model_path)loadsthetokenizerfromthespecifiedmodel path.
7. Thetokenizerisessentialforconvertinguserinput intotokenIDsthemodelcanunderstand.
8. Thissetupistypicallypart ofalargerpipelineforgeneratingAIresponsesor predictions.
9. ItleveragesHuggingFace'stransformerslibraryfunctionality.

10. Themodelislikelyusedfortextgeneration,classification,orinteraction(e.g.,chatassistant).

Activity5.2:TestingandVerifyingLocal Deployment

1. **StarttheFlaskApplication**

Runthefollowingcommandtolaunchtheapplicationlocally:

pythonapp.py

Milestone6:functionaltestingandverify

Indexpage:

ThisisthelandingpageoftheCitizenAIwebapplication,designedforcivicengagementthroughAI. Here's a breakdown:

1. HeaderSection:

a. Showsthemaintitle: "WelcometoCitizenAI".

b. Containsnavigationlinks:About,Services,Chat,Dashboard,andLogin.

2. Left Panel (Intro Section):

a. Headline:"EmpoweringCitizensThroughAI".
b. Describes CitizenAI as an intelligent assistant helping citizens engage with governmentservices,providefeedback,andcommunicatemoreeffectively.
c. Includesaprominent"GetStarted"button,likelyredirectingtouserinteractionor signup.

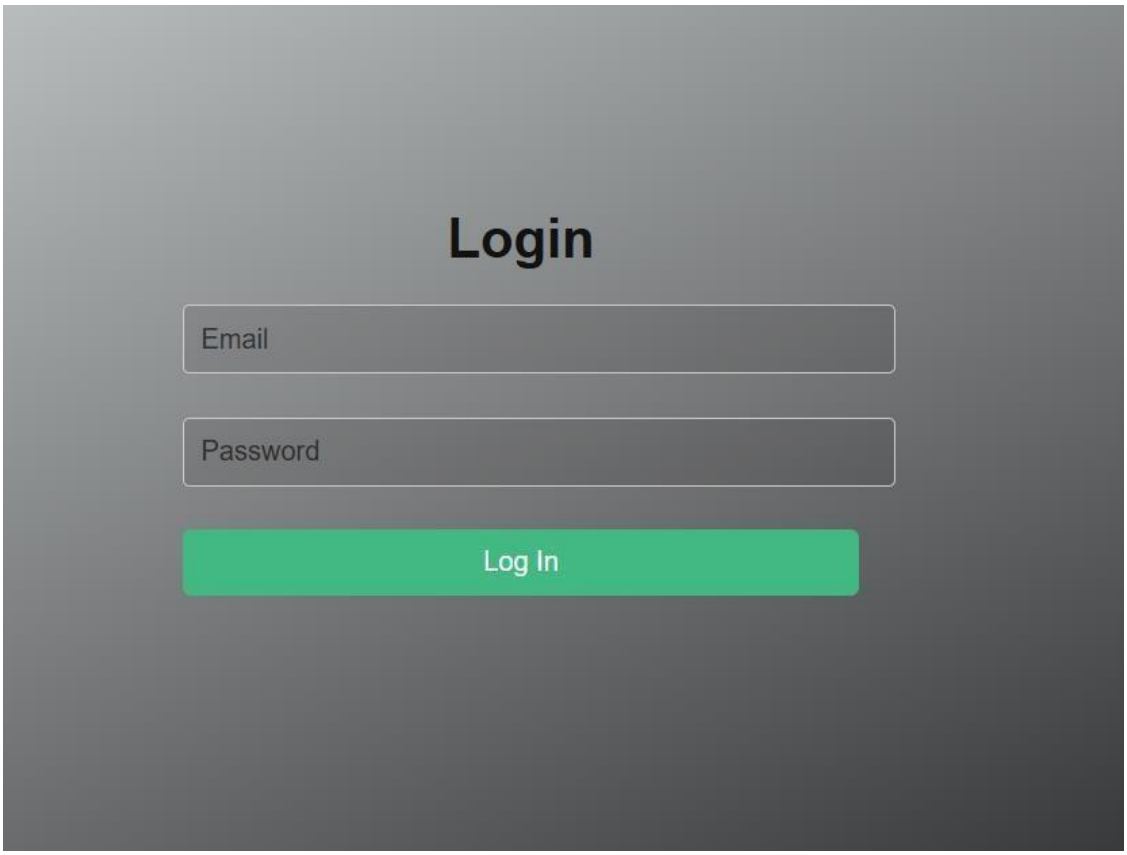
3. Right Panel (Visual):

a. Featuresadigital,futuristicbackgroundwithahumanheadsilhouetteandglowing circuit lines.
b. Overlaidwithcode andAI-relatedtext toemphasizethetech-drivennatureof the platform.

4. Purpose:

a. ThepagepromotesanAI-poweredcivicplatformaimedatbuildingasmarter, responsive government-citizen relationship.
b. Itencouragesuserstobegininteractingwiththesystembyclicking **Get Started**.

Login page:

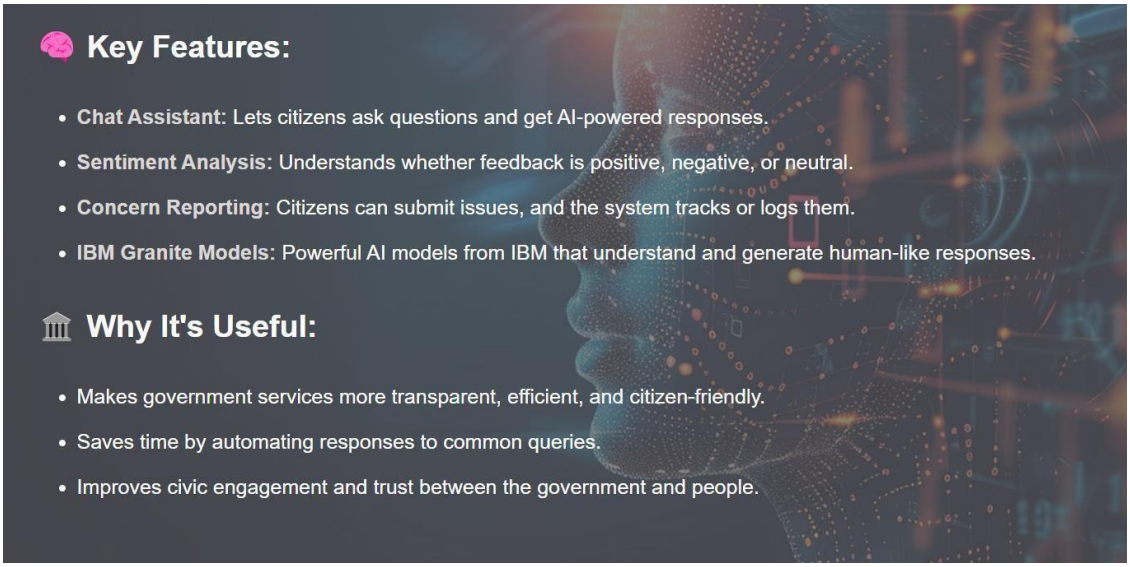


This page is the Login page for the application. Its primary function is to allow users to securely accesstheplatform.YoucanseeinputfieldsforenteringyourEmail(orusername)andPassword, along with a Log In button to submit your credentials. This page is where users authenticate themselves before gaining access to the protected features and content of the application.

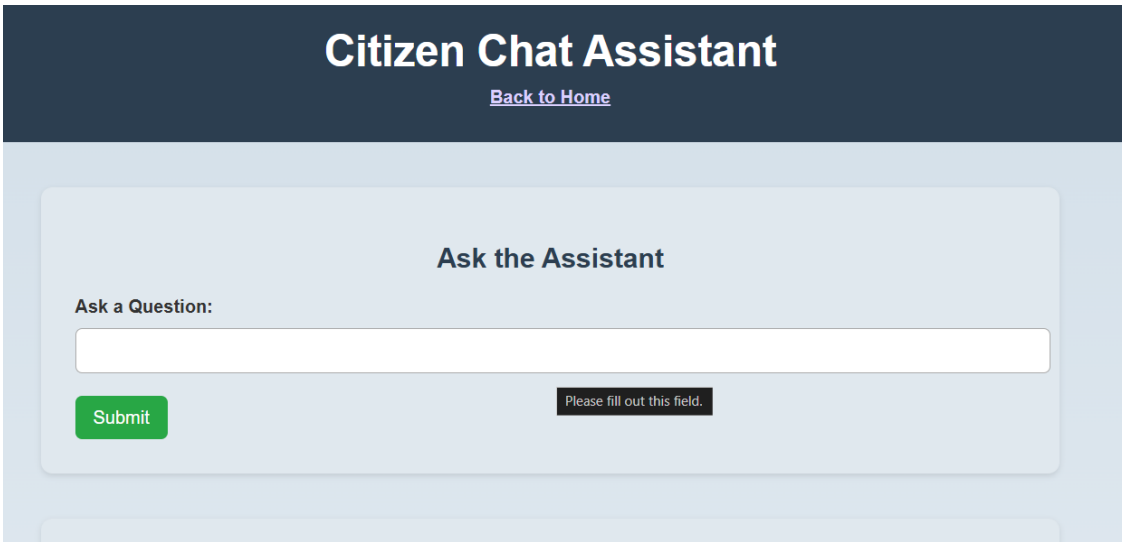
AboutPage:

This page is the "About Citizen AI" page. It serves to introduce users to the project, explaining what Citizen AI is and outlining its mission to improve interactions between governments and citizens using AI. The page provides a brief description and then breaks down the concept "In Simple Terms,"likelyhighlightingkeyfunctionslikeaskingquestionsandreportingissues.Italsoincludesa "Back to Home" link for easy navigation.

This sectionofthepagehighlights thecoreaspects oftheCitizenAIplatform.Itdetails theKeyFeaturesoffered,suchastheChatAssistantforAI-poweredresponses, SentimentAnalysisforfeedback,andConcernReportingforissues,mentioningtheuse ofIBMGraniteModels.ItalsoexplainsWhyIt'sUseful,outliningbenefitslikemaking government services more transparent and efficient, and improving civic engagement andtrust.

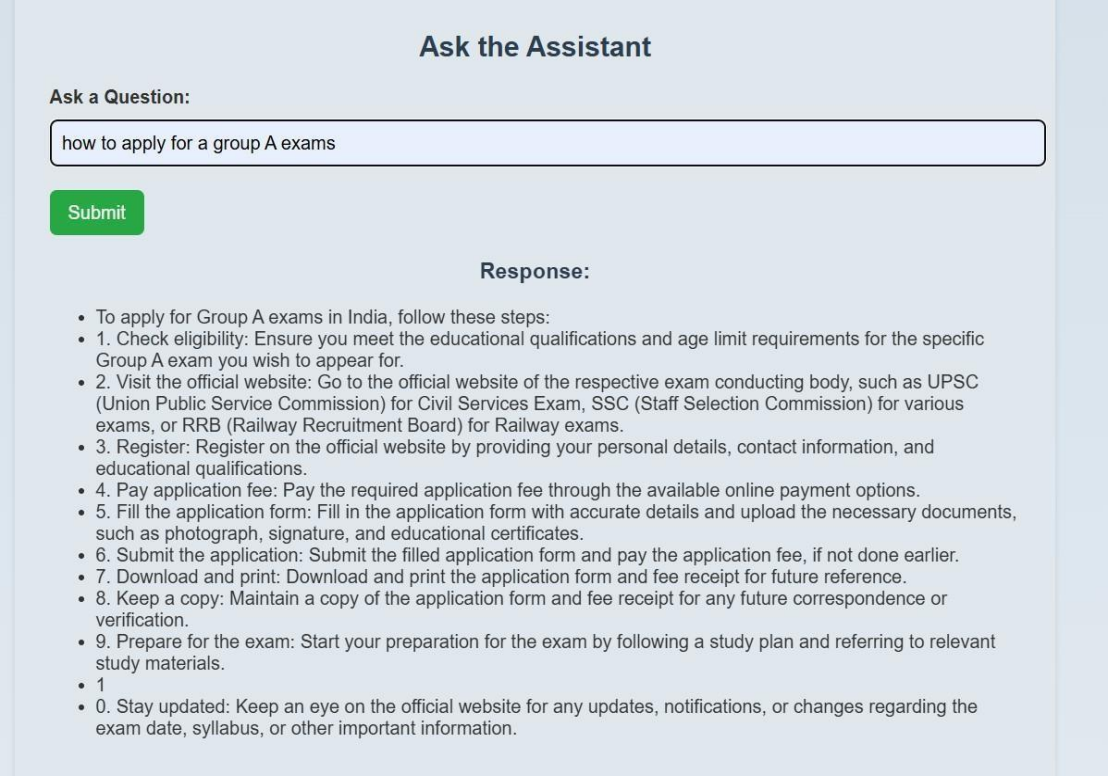


Chat Page:

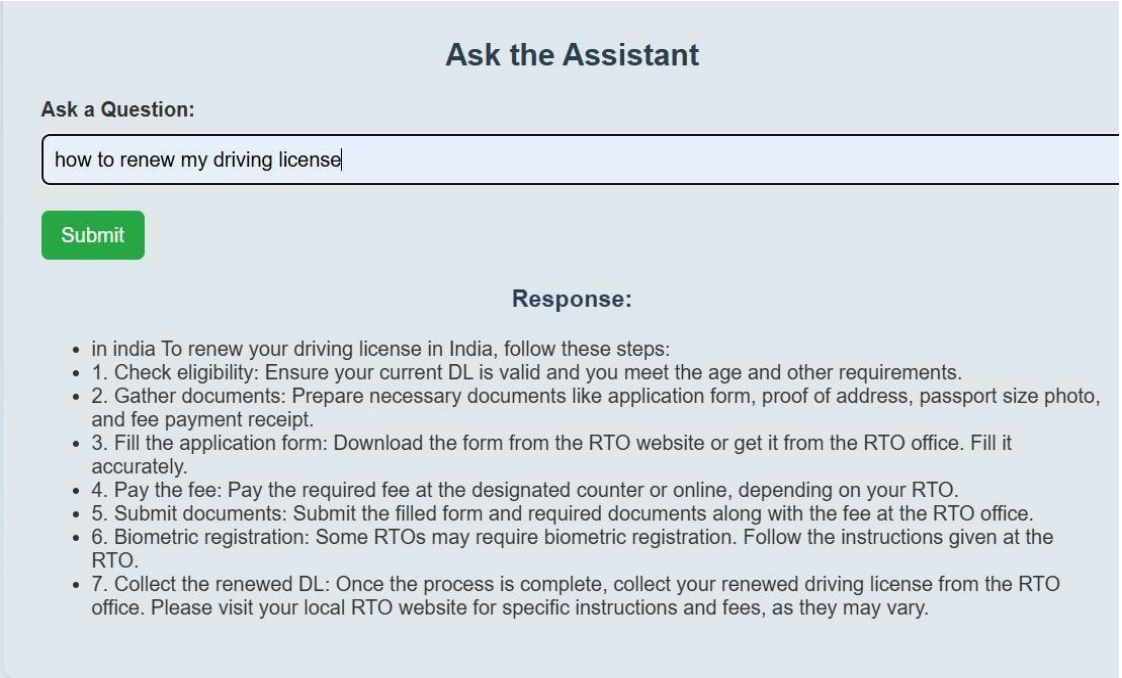


Thispageisthe**"CitizenChatAssistant"**.It'swhereyoucandirectlyinteractwiththeAI assistant.You'llfindasectiontitled"AsktheAssistant"withatextfieldto**typeyour question**anda**Submitbutton**tosendit.The"BacktoHome"linkallowsyoutoreturn tothemainpage.Themessage"Pleasefilloutthisfield"indicatesthatyouneedtoenter text in the question box before submitting.

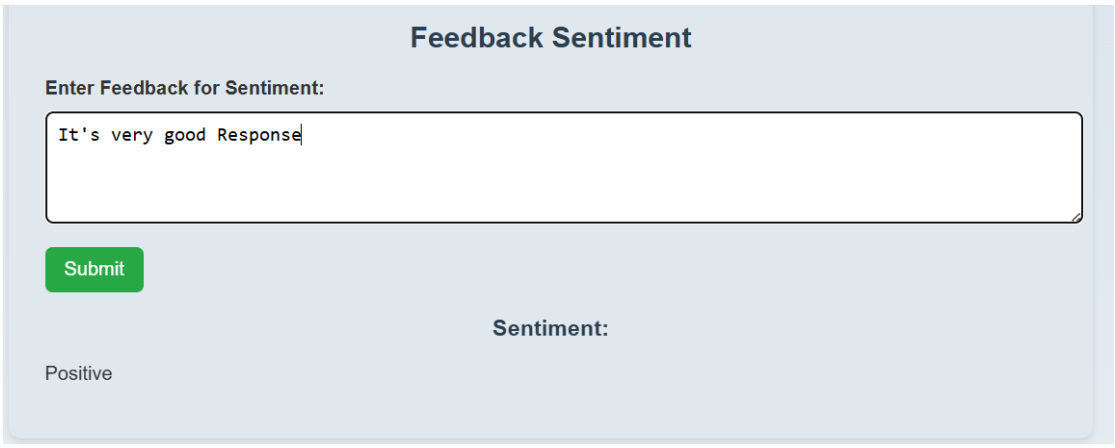
Response1:



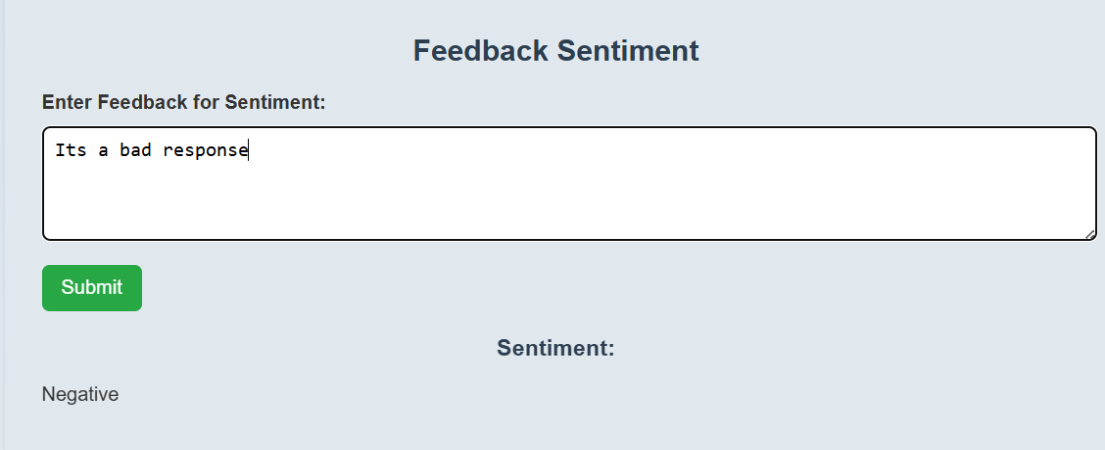
Response2:



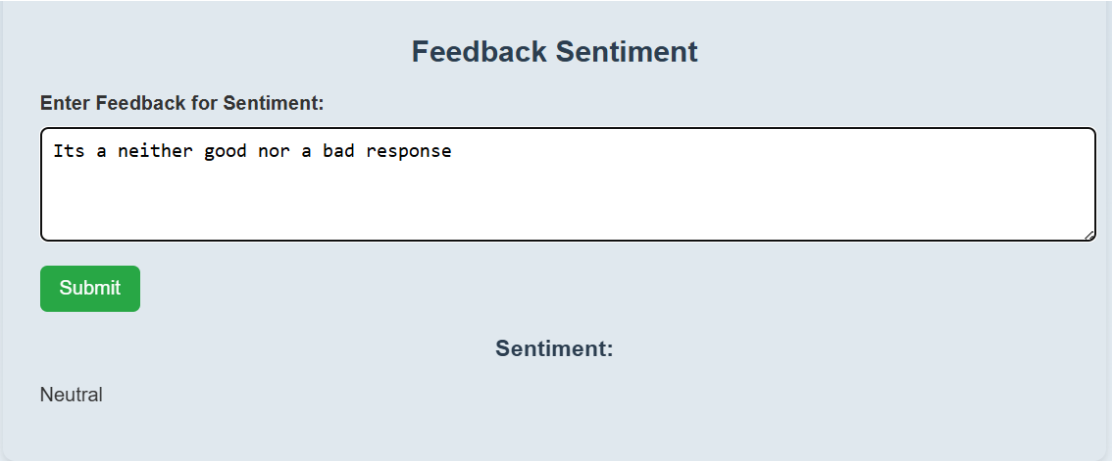
SentimentAnalysis:Sentiment1



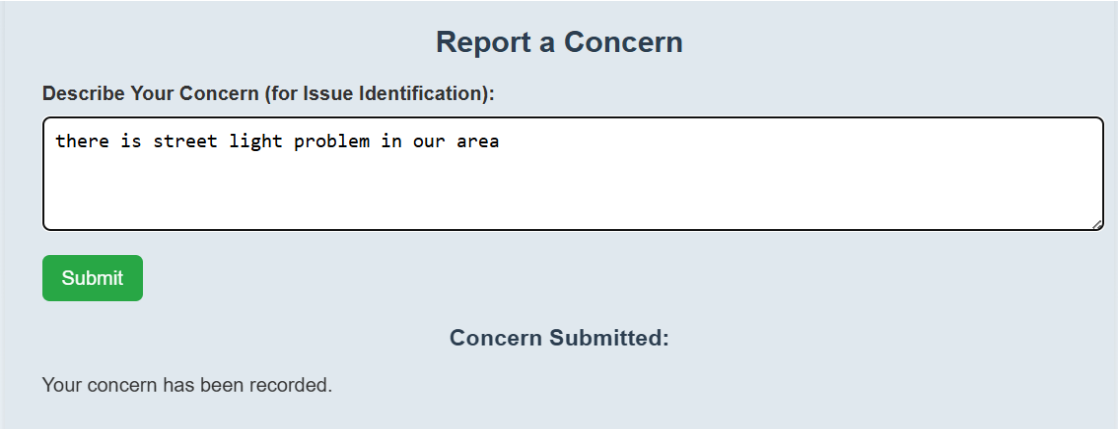
Sentiment2:



Sentiment3:



ConcernReporting:



Dashboardpage:

This page is the "Citizen Insights Dashboard". It provides an overview of the feedback and issues reportedbycitizens.YoucanseeasummaryoftheWeeklySentimentAnalysis,showingcountsfor Positive, Neutral, and Negative feedback. Below that, there's a section for Recent Citizen Issues, listing the concerns that have been reported. This dashboard helps to quickly visualize citizen sentiment and identify common issues.

Conclusion

Your AI-powered CitizenAI platform is designed to enhance interaction, accessibility, and transparency between citizens and government services. By integrating an AI chat assistant, sentiment analysis, concern reporting, and dashboard insights, the platform empowers users to easily access information, provide feedback, and report issues. With a Flask backend and an interactiveHTML/CSSfrontend,poweredbytheIBMGraniteAImodel,yourprojectensuresauser- friendly experience while providing smart and responsive civic engagement tools. This innovative solution simplifies communication and fosters trust, making civic participation more convenient and efficient for all.

Real-WorldExamples

- **311Systems(USA)**-PlatformslikeNYC311allowresidentstoreportnon-emergencyissues.
- **Decidim(Spain)**-Open-sourceplatformforparticipatorydemocracy.

- **MyGov(India)**–Nationalcitizenengagementplatformforfeedbackand participation.
-

🤝ConsultationRoleinthePlatform

Consultationisoneofthe**corefunctions**ofanintelligentengagementplatform.Itallows governmentsto:

- **Askforcitizeninput**onnewpolicies,projects,orchanges.
- Use**digitalsurveys,polls,andforums**togatherfeedback.
- **Analyzeresponses**usingAItounderstandpublicopinionorconcerns.
- **Makedecisions**basedonreal-time,inclusivecitizendata.

Example:Beforebuildinganewhighway,acitycanconsultresidentsusingtheplatformto voteondesignoptions,voiceconcerns,orsuggestimprovements.

✅BenefitsofSmartConsultation

Benefit	Description
Inclusivity	Reachesmorepeoplethanin-person meetings.
Transparency	Citizenscanseehowtheirinputisused.
Speed	Fasterfeedbackcollection andanalysis.
Trust	Citizensfeelheardandvaluedindecisions.

📈Benefits

ForGovernments	ForCitizens
Betterdecision-makingvia data	Easyaccesstoservices
Fasterresponsetopublic issues	Transparentandaccountable systems

Improvedtrustand transparency	Directinfluenceonlocalpolicies
Reducedadministrative burden	Real-timefeedbackand communication