CITIZEN AI

Project Documentation

1. Introduction :

- Project title : CITIZEN AI
- Team member : PRIYADHARSHINI B
- Team member : PAVITHRA.R
- Team member : NANDHINI.A
- Team member : MARTHA.G

2. About:

This document explains the Python code for the Citizen AI Platform, an application built using Gradio and the IBM Granite language model. The platform provides various services, including city analysis, citizen services, an AI assistant chat, and a feedback form, all accessible through a secure login system.

3. Dependencies and Setup:

The code begins by installing the necessary libraries using pip:

- transformers: For using the IBM Granite language model.
- torch: The PyTorch library for deep learning operations.
- gradio: The library used to build the web-based user interface.

```
!pip install transformers torch gradio -q
```

The script then imports the required modules from these libraries: gradio as gr, torch, AutoTokenizer, and AutoModelForCausalLM.

4. Model Loading:

The core of the application is the ibm-granite/granite-3.2-2b-instruct model. The code loads this pre-trained model and its corresponding tokenizer from the Hugging Face Transformers library. It checks for a GPU (torch.cuda.is_available()) to optimize performance by moving the model to the GPU if available.

```
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)
```

5. AI Response Generation:

The generate_response function is a reusable helper function for generating text using the loaded model. It takes a prompt and a max_length as input. It tokenizes the prompt, generates a response from the model, and decodes the output back into a human-readable string.

```
def generate_response(prompt, max_length=1024):
    # Tokenize the input prompt
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True,
max_length=512)

    # Generate a response from the model
    with torch.no_grad():
        outputs = model.generate(...)

    # Decode the response and return it
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return response.replace(prompt, "").strip()
```

6. Application Functions:

The code defines several functions that represent the core features of the application, each of which uses the generate_response function with a specific prompt:

- **city_analysis(city_name):** Formulates a prompt to request a detailed analysis of a city's safety, including crime rates and traffic safety, and returns the AI's response.
- **citizen_interaction(query):** Creates a prompt designed for a "government assistant" persona to provide information on public services and civic issues.
- **assistant_chat(message, history):** Manages a conversational chat flow, taking user messages and chat history to generate a new AI response.
- **collect_feedback(name, rating, comments):** This is a simple

function that simulates collecting user feedback and returns a confirmation message.

7. Login System:

A basic, in-memory login system is implemented using a dictionary VALID_USERS. The login function checks the provided username and password against this dictionary. If the credentials are valid, it updates the Gradio UI to hide the login form and show the main application content.

```
VALID_USERS = {
    "citizen": "ai123",
    "admin": "admin123",
    "guest": "guest123"
}

def login(username, password):
    if username in VALID_USERS and VALID_USERS[username] == password:
        return gr.update(visible=False), gr.update(visible=True), f"✅ Welcome
{username}!"
    else:
        return gr.update(visible=True), gr.update(visible=False), "❌ Invalid
Username or Password"
```

8. Gradio UI Layout:

The user interface is built using gr.Blocks(), which allows for a more complex and dynamic layout.

- **Login Group:** A gr.Group is used to contain the username and password textboxes, and the login button. This group is initially visible.
- **Main Group:** Another gr.Group contains all the application tabs. This group is initially hidden (visible=False) and is only made visible after a successful login.
- **Tabs:** The gr.Tabs() component organizes the different features into separate, easy-to-navigate tabs:
    - **City Analysis:** Contains a textbox for a city name and a button to get an analysis.
    - **Citizen Services:** Has a textbox for a citizen query and a button to get a government-style response.
    - **AI Assistant:** A standard chatbot interface with a text input and a clear button.
    - **Feedback:** A form with fields for name, rating, and

comments.

9. Linking UI and Functions:

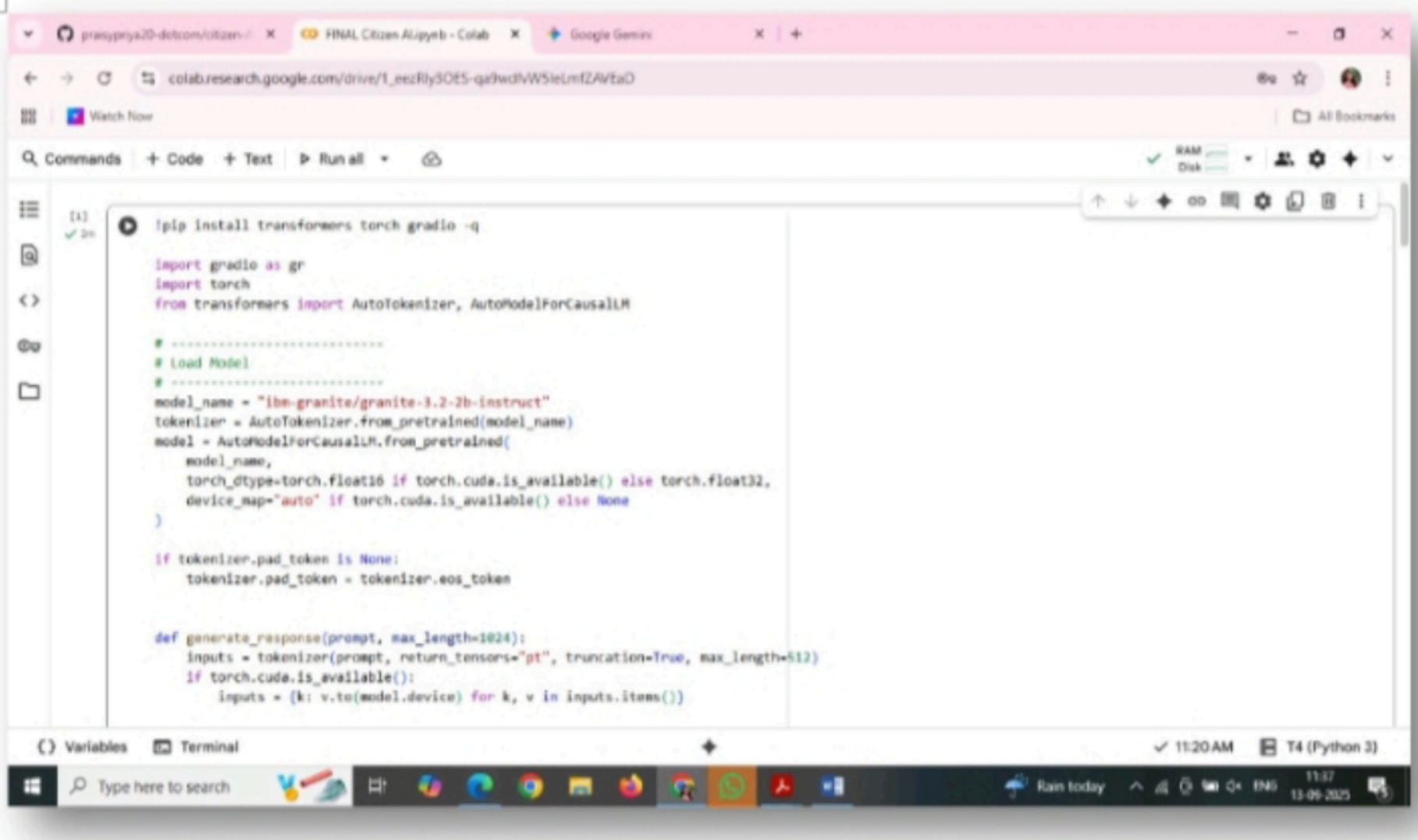
The click method on the Gradio components is used to link UI events (like a button click) to the Python functions.

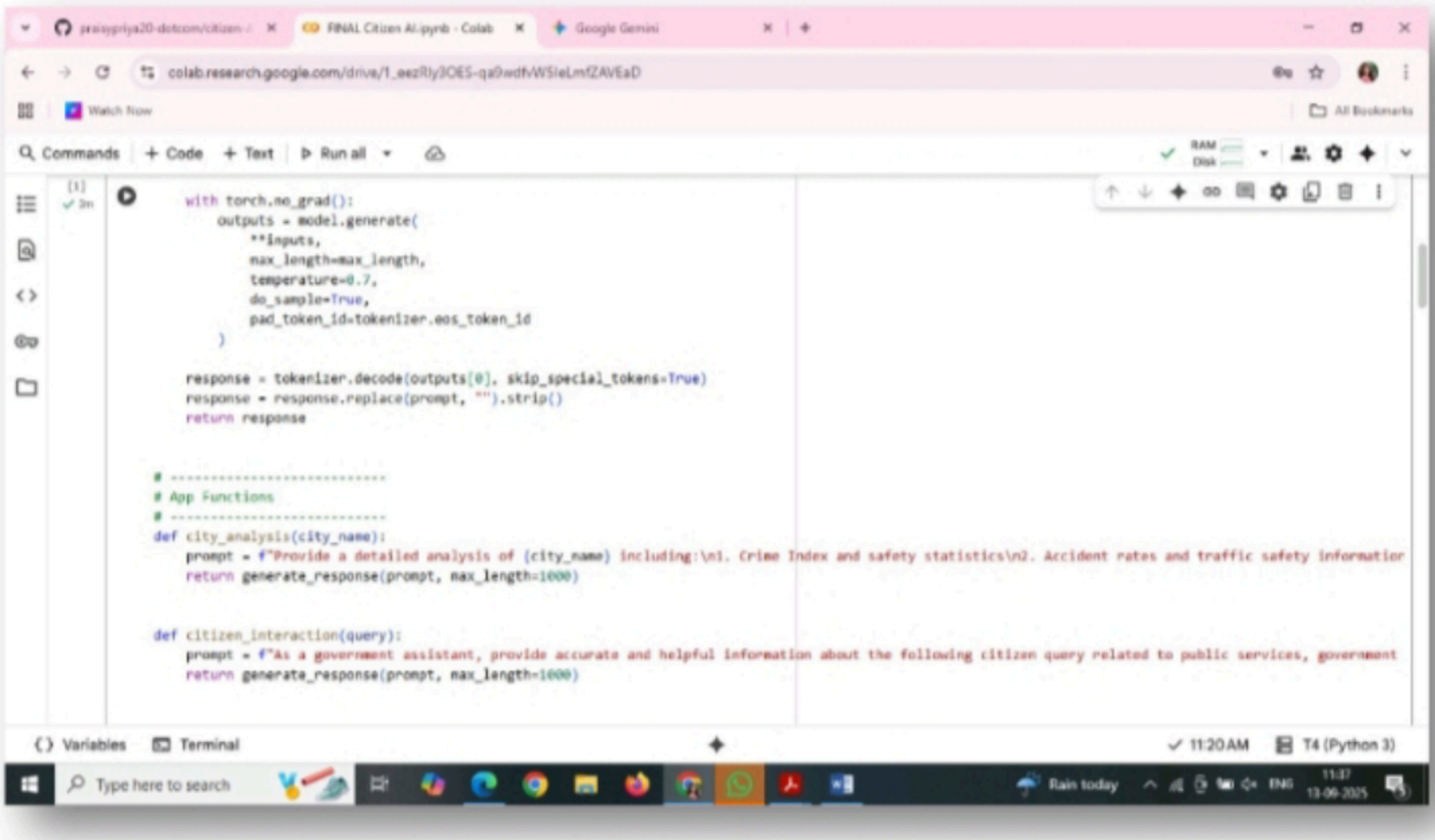
- login_btn.click(...): When the login button is clicked, the login function is called. The outputs of this function are used to update the visibility of the login and main groups, and display a message.
- analyze_btn.click(...): Calls the city_analysis function and displays the output in the city_output textbox.
- query_btn.click(...): Calls the citizen_interaction function.
- msg.submit(...): Handles the chat functionality, calling assistant_chat to manage the conversation flow.
- submit_btn.click(...): Calls collect_feedback to process the feedback form data.

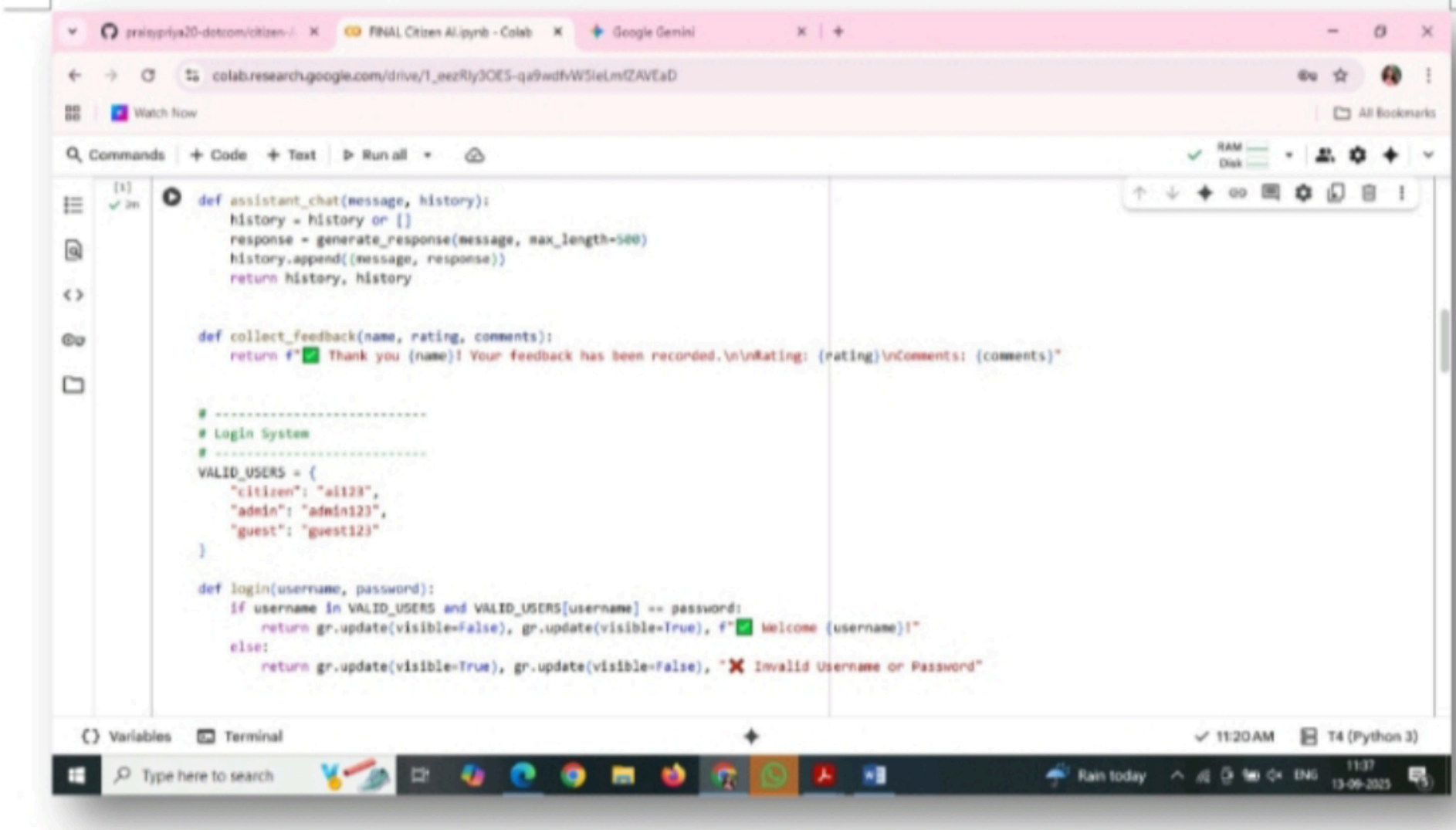
10. Launching the Application:

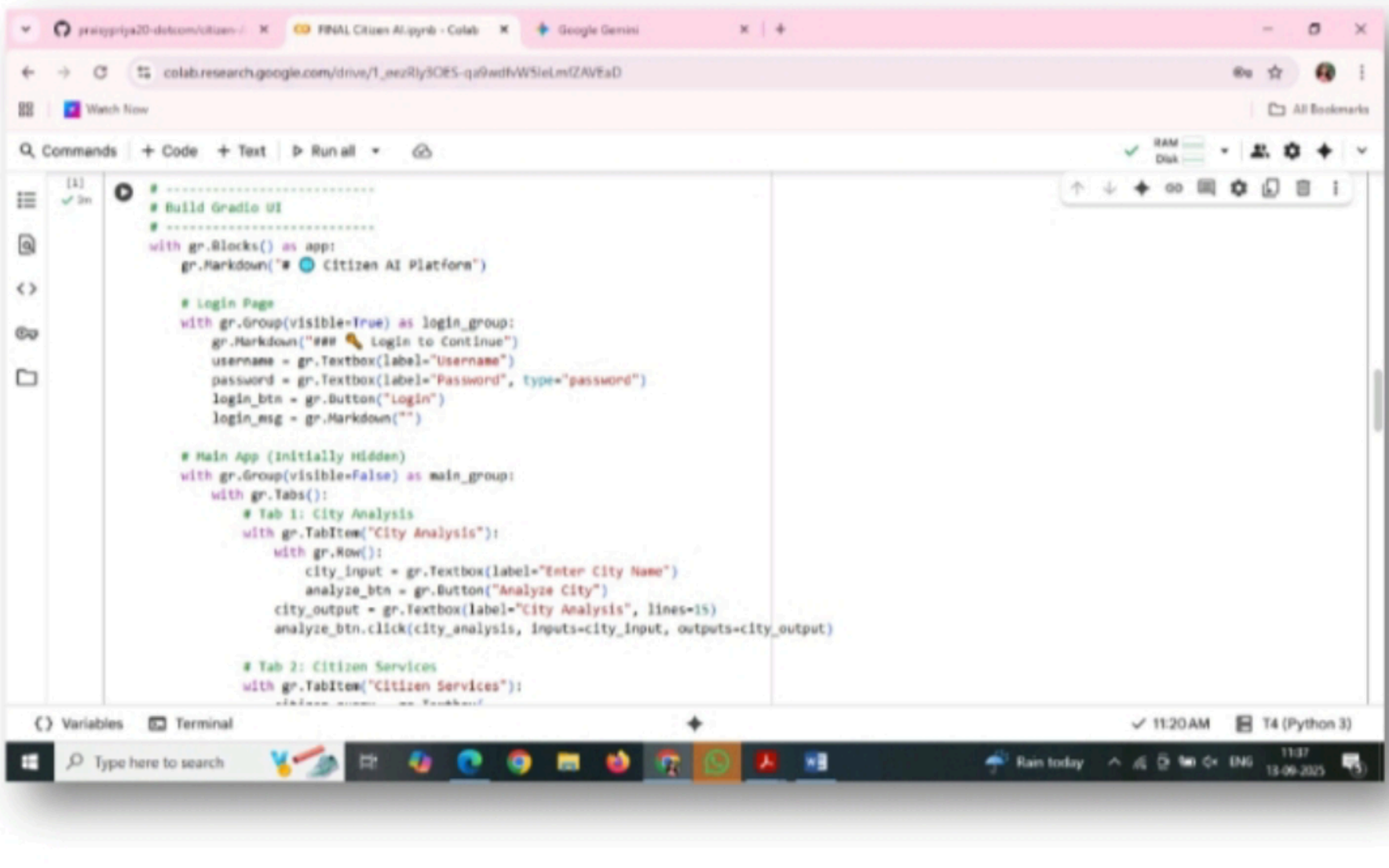
Finally, app.launch(share=True) starts the Gradio web server, making the application accessible in a web browser. The share=True argument creates a public, temporary URL that can be shared with others.

11. Screenshots:


FINAL Citizen AI.ipynb - Colab
Google Gemini
colab.research.google.com/drive/1_eezRly3OES-qa9wdfvW5leLmfZAVEaD
Watch Now
All Bookmarks
Commands
+ Code
+ Text
Run all
RAM
Disk
[1]
2m
!pip install transformers torch gradio -q
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
# ----------------------------
# Load Model
# ----------------------------
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
model_name,
torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
device_map="auto" if torch.cuda.is_available() else None
)
if tokenizer.pad_token is None:
tokenizer.pad_token = tokenizer.eos_token
def generate_response(prompt, max_length=1024):
inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
if torch.cuda.is_available():
inputs = {k: v.to(model.device) for k, v in inputs.items()}
Variables
Terminal
11:20 AM
T4 (Python 3)
Type here to search
Rain today
ENG
11:37
13-09-2025


FINAL Citizen AI.ipynb - Colab
Google Gemini
colab.research.google.com/drive/1_eezRly3OES-qa9wdfvW5leLmfZAVEaD
Watch Now
All Bookmarks
Commands
+ Code
+ Text
Run all
RAM
Disk
[1]
2m
with torch.no_grad():
outputs = model.generate(
**inputs,
max_length=max_length,
temperature=0.7,
do_sample=True,
pad_token_id=tokenizer.eos_token_id
)
response = tokenizer.decode(outputs[0], skip_special_tokens=True)
response = response.replace(prompt, "").strip()
return response
# ----------------------------
# App Functions
# ----------------------------
def city_analysis(city_name):
prompt = f"Provide a detailed analysis of {city_name} including:\n1. Crime Index and safety statistics\n2. Accident rates and traffic safety information
return generate_response(prompt, max_length=1000)
def citizen_interaction(query):
prompt = f"As a government assistant, provide accurate and helpful information about the following citizen query related to public services, government
return generate_response(prompt, max_length=1000)
Variables
Terminal
11:20 AM
T4 (Python 3)
Type here to search
Rain today
ENG
11:37
13-09-2025


FINAL Citizen AI.ipynb - Colab
Google Gemini
colab.research.google.com/drive/1_eezRly3OES-qa9wdfvW5leLmfZAVEaD
Watch Now
All Bookmarks
Commands
+ Code
+ Text
Run all
RAM
Disk
def assistant_chat(message, history):
history = history or []
response = generate_response(message, max_length=500)
history.append((message, response))
return history, history
def collect_feedback(name, rating, comments):
return f"✅ Thank you {name}! Your feedback has been recorded.\n\nRating: {rating}\nComments: {comments}"
# ----------------------------
# Login System
# ----------------------------
VALID_USERS = {
"citizen": "ai123",
"admin": "admin123",
"guest": "guest123"
}
def login(username, password):
if username in VALID_USERS and VALID_USERS[username] == password:
return gr.update(visible=False), gr.update(visible=True), f"✅ Welcome {username}!"
else:
return gr.update(visible=True), gr.update(visible=False), "❌ Invalid Username or Password"
Variables
Terminal
11:20 AM
T4 (Python 3)
Type here to search
Rain today
ENG
11:37
13-09-2025

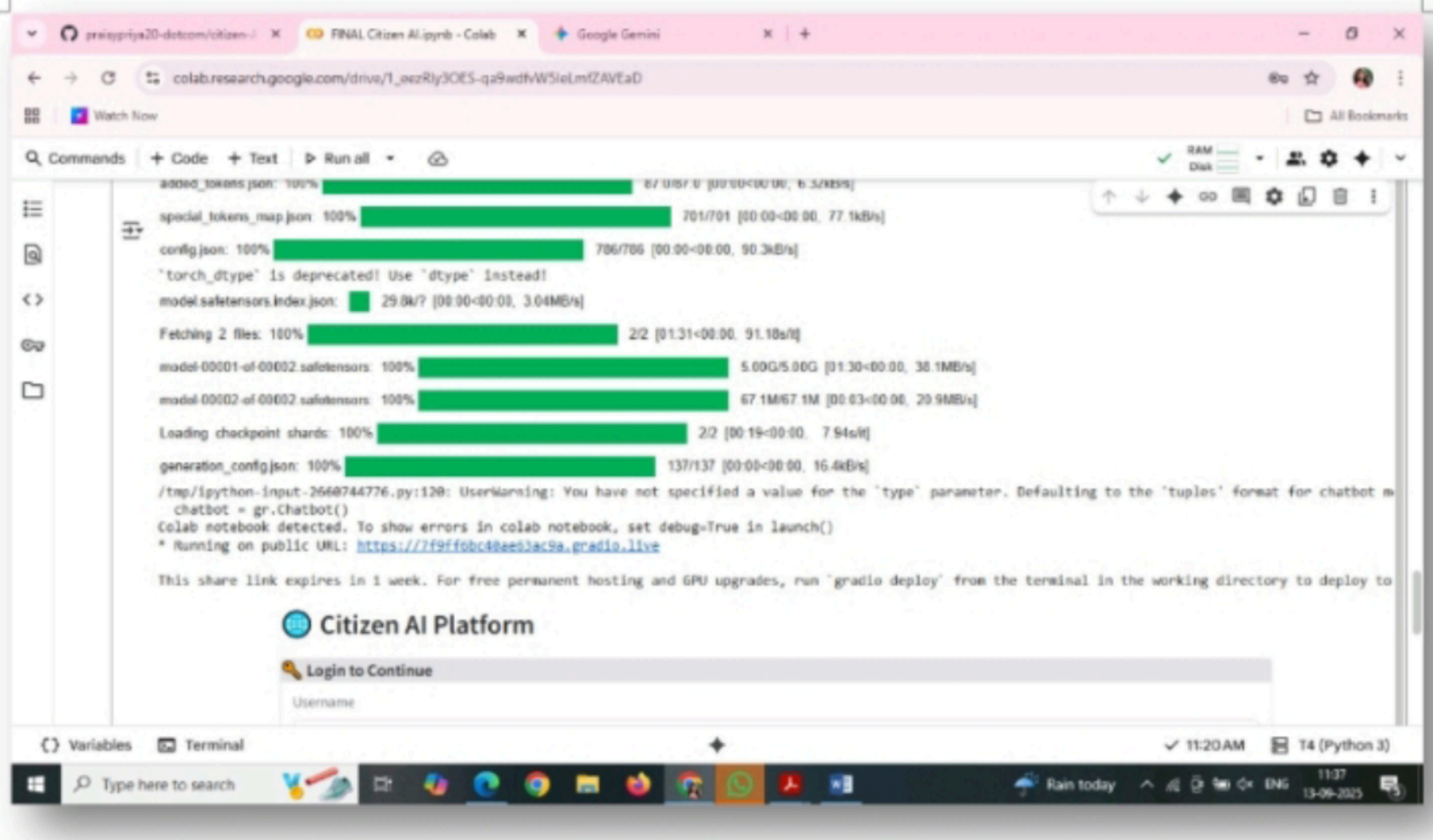

FINAL Citizen AI.ipynb - Colab
Google Gemini
colab.research.google.com/drive/1_eezRly3OES-qa9wdfvW5leLmfZAVEaD
Watch Now
All Bookmarks
Commands
+ Code
+ Text
Run all
RAM
Disk

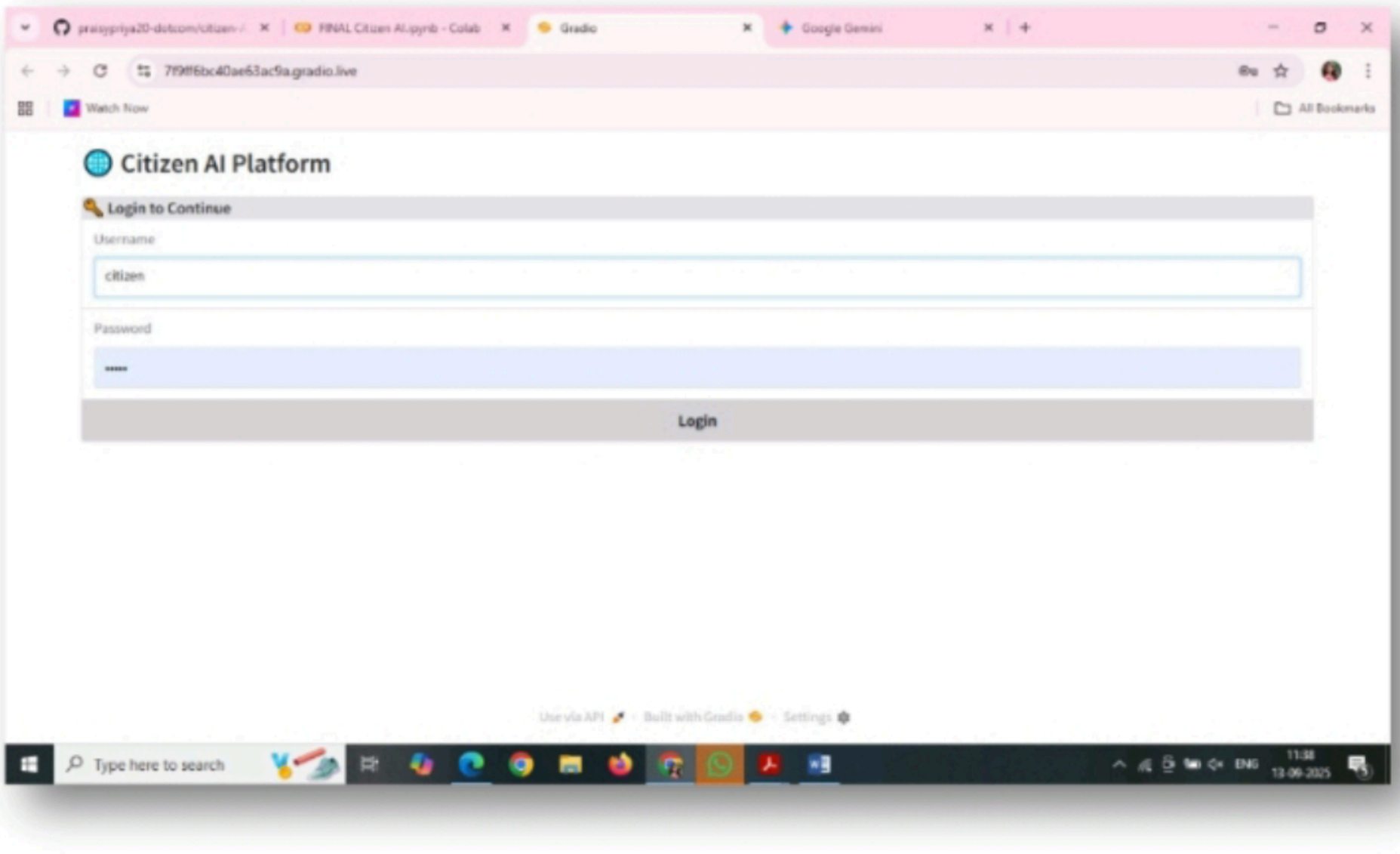
Build Gradio UI
----
with gr.Blocks() as app:
gr.Markdown("# 🌐 Citizen AI Platform")
# Login Page
with gr.Group(visible=True) as login_group:
gr.Markdown("### 🔧 Login to Continue")
username = gr.Textbox(label="Username")
password = gr.Textbox(label="Password", type="password")
login_btn = gr.Button("Login")
login_msg = gr.Markdown("")
# Main App (Initially Hidden)
with gr.Group(visible=False) as main_group:
with gr.Tabs():
Tab 1: City Analysis
with gr.TabItem("City Analysis"):
with gr.Row():
city_input = gr.Textbox(label="Enter City Name")
analyze_btn = gr.Button("Analyze City")
city_output = gr.Textbox(label="City Analysis", lines=15)
analyze_btn.click(city_analysis, inputs=city_input, outputs=city_output)
# Tab 2: Citizen Services
with gr.TabItem("Citizen Services"):
Variables
Terminal
11:20 AM
T4 (Python 3)
Type here to search
Rain today
ENG
11:37
13-09-2025

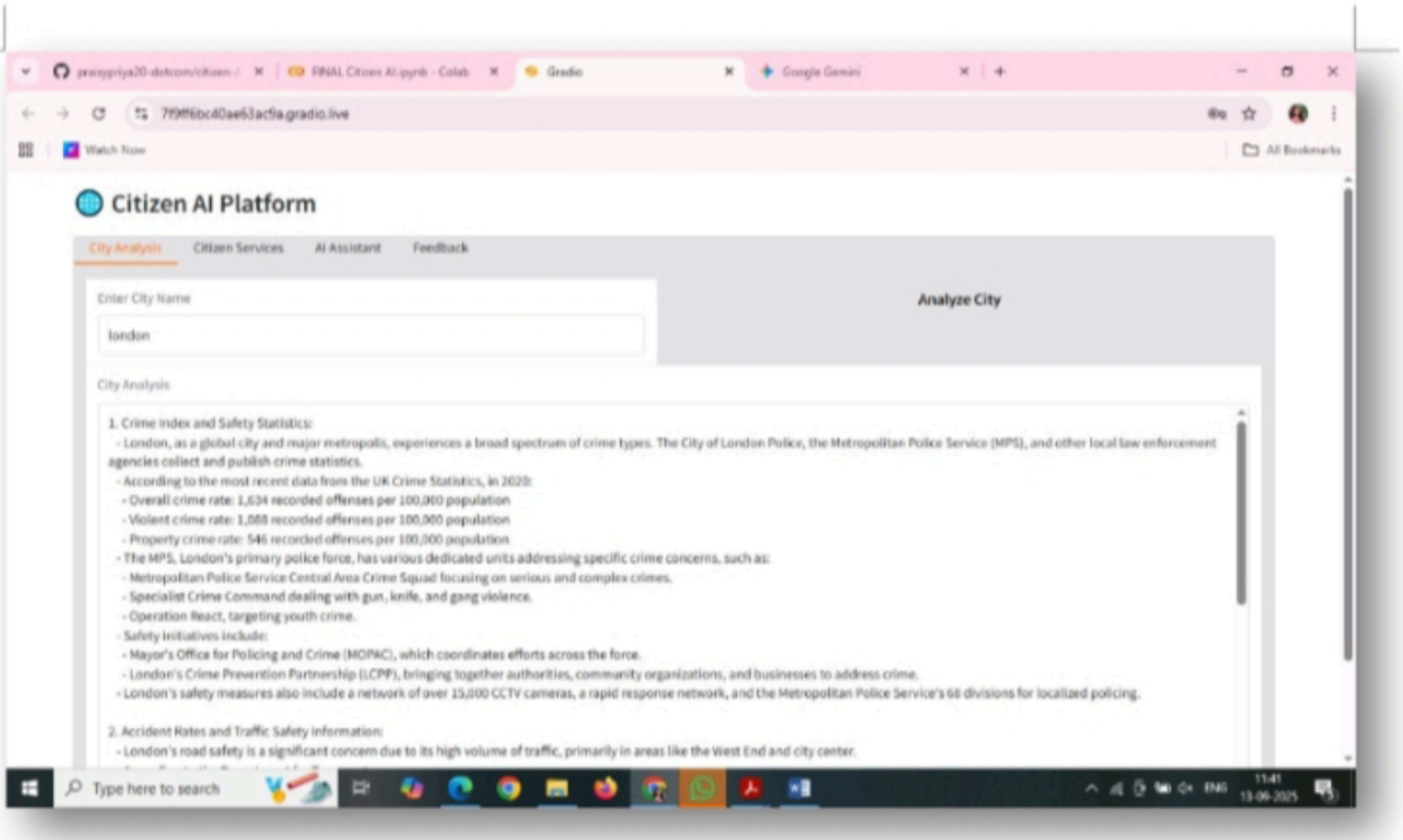
```
            citizen_query = gr.Textbox(
                label="Your Query",
                placeholder="Ask about public services, government policies, civic issues...",
                lines=4
            )
            query_btn = gr.Button("Get Information")
            citizen_output = gr.Textbox(label="Government Response", lines=15)
            query_btn.click(citizen_interaction, inputs=citizen_query, outputs=citizen_output)

        # Tab 3: Assistant
        with gr.TabItem("AI Assistant"):
            gr.Markdown("### 🤖 Chat with Assistant")
            chatbot = gr.Chatbot()
            msg = gr.Textbox(placeholder="Type your message here...")
            clear = gr.Button("Clear Chat")
            msg.submit(assistant_chat, [msg, chatbot], [chatbot, chatbot])
            clear.click(lambda: None, None, chatbot, queue=False)

        # Tab 4: Feedback
        with gr.TabItem("Feedback"):
            gr.Markdown("### 📝 Share Your Feedback")
            name = gr.Textbox(label="Your Name")
            rating = gr.Radio(["Excellent", "Good", "Average", "Poor"], label="Rate Us")
            comments = gr.Textbox(label="Comments", lines=4, placeholder="Enter your suggestions or issues here...")
            submit_btn = gr.Button("Submit Feedback")
            feedback_output = gr.Textbox(label="Response", interactive=False)
            submit_btn.click(collect_feedback, inputs=[name, rating, comments], outputs=feedback_output)
```

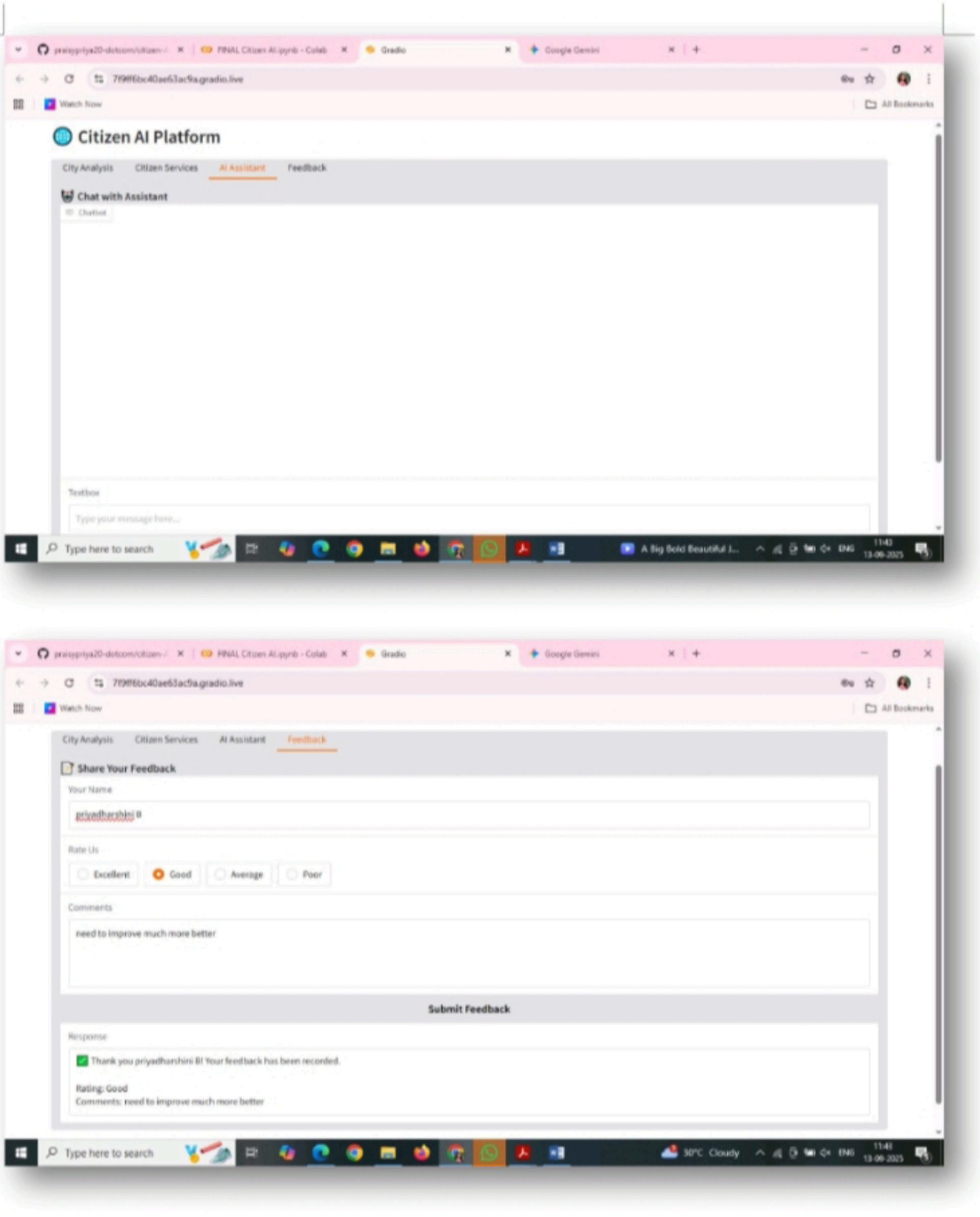
```
    # Connect login
    login_btn.click(login, inputs=[username, password],
                    outputs=[login_group, main_group, login_msg])


# Launch app
app.launch(share=True)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Co
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 8.68k/? [00:00<00:00, 736kB/s]
vocab.json: 777k/? [00:00<00:00, 19.9MB/s]
merges.txt: 442k/? [00:00<00:00, 22.2MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 91.9MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 6.32kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 77.1kB/s]
config.json: 100% 786/786 [00:00<00:00, 90.3kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
```


FINAL Citizen AI.ipynb - Colab
Google Gemini
colab.research.google.com/drive/1_oezRly3OES-qa9wdfvW5leLmfZAVEaD
Commands
+ Code
+ Text
Run all
special_tokens_map.json: 100% 701/701 [00:00<00:00, 77.1kB/s]
config.json: 100% 786/786 [00:00<00:00, 90.3kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 3.04MB/s]
Fetching 2 files: 100% 2/2 [01:31<00:00, 91.18s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:30<00:00, 38.1MB/s]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:03<00:00, 20.9MB/s]
Loading checkpoint shards: 100% 2/2 [00:19<00:00, 7.94s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 16.4kB/s]
/tmp/ipython-input-2660744776.py:120: UserWarning: You have not specified a value for the `type` parameter. Defaulting to the 'tuples' format for chatbot m
chatbot = gr.Chatbot()
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://7f9ff6bc40ae63ac9a.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to
Citizen AI Platform
Login to Continue
Username
Variables
Terminal
11:20 AM
T4 (Python 3)
Type here to search
Rain today
ENG
11:37
13-09-2025


FINAL Citizen AI.ipynb - Colab
Gradio
Google Gemini
7f9ff6bc40ae63ac9a.gradio.live
Citizen AI Platform
Login to Continue
Username
citizen
Password
Login
Use via API
Built with Gradio
Settings
Type here to search
ENG
11:38
13-09-2025


FINAL Citizen AI.ipynb - Colab
Gradio
Google Gemini
7f9ff6bc40ae63ac9a.gradio.live
Watch Now
All Bookmarks
Citizen AI Platform
City Analysis
Citizen Services
AI Assistant
Feedback
Enter City Name
london
Analyze City
City Analysis
1. Crime Index and Safety Statistics:
- London, as a global city and major metropolis, experiences a broad spectrum of crime types. The City of London Police, the Metropolitan Police Service (MPS), and other local law enforcement agencies collect and publish crime statistics.
- According to the most recent data from the UK Crime Statistics, in 2020:
- Overall crime rate: 1,634 recorded offenses per 100,000 population
- Violent crime rate: 1,088 recorded offenses per 100,000 population
- Property crime rate: 546 recorded offenses per 100,000 population
- The MPS, London's primary police force, has various dedicated units addressing specific crime concerns, such as:
- Metropolitan Police Service Central Area Crime Squad focusing on serious and complex crimes.
- Specialist Crime Command dealing with gun, knife, and gang violence.
- Operation React, targeting youth crime.
- Safety initiatives include:
- Mayor's Office for Policing and Crime (MOPAC), which coordinates efforts across the force.
- London's Crime Prevention Partnership (LCPP), bringing together authorities, community organizations, and businesses to address crime.
- London's safety measures also include a network of over 15,000 CCTV cameras, a rapid response network, and the Metropolitan Police Service's 68 divisions for localized policing.
2. Accident Rates and Traffic Safety Information:
- London's road safety is a significant concern due to its high volume of traffic, primarily in areas like the West End and city center.
Type here to search
ENG
11:41
13-09-2025
7f9ff6bc40ae63ac9a.gradio.live
Watch Now
All Bookmarks
Citizen AI Platform
City Analysis
Citizen Services
AI Assistant
Feedback
Your Query
public services
Get Information
Government Response
Dear Citizen,
Thank you for your interest in public services. Public services are essential resources provided by the government to ensure the well-being, safety, and prosperity of its citizens. Here are some key areas where public services are crucial:
1. **Education**: Public schools provide foundational learning for children, ensuring access to quality education regardless of their socio-economic background. Many government programs offer additional support for students, including scholarships, grants, and special needs services.
2. **Healthcare**: The government plays a significant role in ensuring access to affordable healthcare services. This includes public hospitals, clinics, and community health centers, as well as initiatives to promote preventive care and public health.
3. **Infrastructure**: Public services involve maintaining and improving the nation's infrastructure, including roads, bridges, public transportation, water systems, and sanitation facilities. These services support economic growth and enhance the quality of life for citizens.
Type here to search
A Big Bold Beautiful J...
ENG
11:42
13-09-2025



12. Future Enhancements:

- **User Sessions and History Tracking:** The project plans to add the ability to track user sessions and interaction history. This will allow for a more personalized experience.
- **Security:** For secure deployments, the project can integrate

token-based authentication (JWT or API keys), OAuth2 with IBM Cloud credentials, and role-based access for different users (e.g., admin, citizen, researcher).