**Citizen AI**

# Project Documentation

## 1.Introduction

* Project title : Citizen AI: Intelligent Citizen Engagement Platform
* Team member : K. Anusiya
* Team member : K. Dhanasri
* Team member : S. Mounika
* Team member : S. Agalya

## 2.Project overview

### **Purpose**:

The Citizen AI Intelligence & Engagement Platform is designed to strengthen the connection between citizens and governments by leveraging artificial intelligence, data-driven insights, and digital communication tools. Its purpose is to enable more inclusive, transparent, and responsive governance by providing a platform where citizens can easily share feedback, participate in decision-making, and access reliable information.

The Citizen AI platform is an intelligent citizen engagement and governance solution that leverages artificial intelligence, analytics, and digital communication tools to transform how governments, civic bodies, and communities interact. To enable inclusive, transparent, and data-driven engagement between citizens and authorities, ensuring that people’s voices are heard, understood, and acted upon in real time.

### **Objectives**

Enhance citizen participation by offering accessible digital channels for feedback, queries, and collaboration.

Leverage AI-driven insights to analyze trends, detect emerging community needs, and recommend policy/service improvements.

Improve governance transparency by making public information more accessible, reliable, and easy to understand.

Promote inclusivity through multilingual, user-friendly, and accessible design for diverse populations.

Strengthen trust between governments and citizens by fostering continuous, two-way communication.

##### 

### **Key Features**

AI-powered chatbots and virtual assistants for citizen queries.

Sentiment and trend analysis of citizen feedback.

Issue tracking and resolution dashboards for governments.

Multilingual support to ensure inclusivity.

Open data and transparency tools for public accountability.

### 

### **Expected Impact**

More responsive and citizen-centric governance.

Increased trust and participation in civic processes.

Data-informed policy making and service delivery.

A foundation for smart, connected, and inclusive communities.

## **3.Architecture**

### **1. Frontend (Citizen & Government Interface)**

The frontend is the interaction layer designed for usability, inclusivity, and accessibility.

Citizen Portal / Mobile App

Multilingual support (text & voice)

AI-powered chatbot / virtual assistant

Feedback submission (text, voice, video, images)

Service request forms & tracking

Notifications & alerts (policy updates, service status)

Community discussion forums

Government / Admin Dashboard

Issue tracking & resolution workflow

Real-time sentiment and trend analytics

Geospatial mapping of citizen issues

KPI monitoring dashboards (engagement, service response times)

Policy suggestion & data visualization tools

#### **Tech stack options:**

Web: React / Angular / Vue

Mobile: Flutter / React Native

UI frameworks: Material UI / TailwindCSS

Accessibility: WCAG 2.1 compliantool

### **2. Backend (AI + Data + Service Layer)**

The backend powers data processing, AI intelligence, and system integration.

Application Layer

API Gateway (manages citizen ↔ government interactions)

Authentication & Identity Management (OAuth 2.0, Single Sign-On)

Role-based access control (citizen, admin, policymaker)

AI/Analytics Layer

Natural Language Processing (multilingual support, sentiment analysis)

Speech-to-Text & Text-to-Speech services

Predictive analytics (trend detection, emerging issues)

Recommendation engine (policy/service improvements)

Data Layer

Citizen feedback database (structured + unstructured data)

Analytics warehouse (BigQuery / Snowflake / PostgreSQL)

Open data repository for transparency

Data lake for unstructured inputs (images, voice, video)

Integration Layer

Connects with government service portals (healthcare, transport, utilities)

Open APIs for third-party civic apps

GIS systems for location-based issue mapping

#### **Tech stack options:**

Backend frameworks: Node.js / Django / Spring Boot

Databases: PostgreSQL, MongoDB, Elasticsearch

AI/ML: TensorFlow / PyTorch / Hugging Face NLP models

Messaging/Queue: Kafka / RabbitMQ

Cloud: AWS / Azure / GCP (for scalability, AI APIs, serverless functions)

### **3.Security & Compliance**

End-to-end encryption for citizen data

Data anonymization for privacy protection

Compliance with GDPR, local data laws

Role-based access & audit trails

### **4. Deployment Architecture**

Microservices architecture for scalability

Kubernetes / Docker for container orchestration

CI/CD pipelines for continuous updates

Load balancers & caching (Redis/CDN) for performance

## **4.Setup Instructions**

### **Prerequisites:**

#### 1. **Infrastructure** – Cloud account (AWS/Azure/GCP) or local server with Docker & Kubernetes installed.

#### **2. Databases** – Install PostgreSQL (for structured data) and MongoDB (for feedback & media).

### **3. Backend & AI** – Python 3.10+ (AI/ML services) and Node.js 18+ (APIs & integrations).

### **4. Frontend** – React/Angular/Flutter environment set up for web/mobile apps.

### **5. Security & Config** – SSL certificates, API keys, and environment variables (.env file) configured.

### **Installation process:**

#### **1. Clone the Project**

git clone https://github.com/your-org/citizen-ai.git

cd citizen-ai

#### **2. Install Dependencies**

Frontend: npm install

Backend/AI: pip install -r requirements.txt

#### **3. Set Up Databases**

Start PostgreSQL and MongoDB

Update .env file with DB connection details

#### **4. Start Services**

Using Docker: docker-compose up -d

Or run manually:

npm start (frontend)

python app.py (backend)

#### **5. Access the Platform**

Citizen Portal → http://localhost:3000

## **5.Folder Structure**

1. /frontend – Web & mobile citizen portal

/src/components – Reusable UI components

/src/pages – Screens (login, dashboard, feedback, etc.)

/src/services – API calls & integrations

2. /backend – Core APIs & services

/api – REST/GraphQL endpoints

/models – Database schemas (PostgreSQL, MongoDB)

/controllers – Business logic

/routes – API routes

/auth – Authentication & authorization

3. /ai-services – AI/ML intelligence layer

/nlp – NLP models (sentiment, multilingual processing)

/speech – Speech-to-text, text-to-speech modules

/recommendations – Policy/service recommendation engine

4. /database – Database setup & migrations

/migrations – SQL migration scripts

/seeds – Sample/test data

5. /config – Environment & configuration files

.env – Environment variables

db.config.js – Database config

app.config.js – App-wide settings

6. /docs – Documentation

Setup guides, API docs, architecture diagrams

7. /scripts – Deployment & automation scripts

CI/CD pipelines

Kubernetes manifests / Helm charts

8. /tests – Unit & integration tests

Backend API tests

AI model validation

Frontend UI tests

9. /logs – Application logs (ignored in git)

10. /docker – Dockerfiles & docker-compose setups

## **6.Running the Application**

### **1. Navigate to Project Directory**

Open terminal/command prompt and move into the project’s root folder.

### **2. Set Environment Variables**

Configure database URL, API keys, and authentication tokens in .env or environment settings.

### **3. Start Database & Services**

Run database (e.g., PostgreSQL/MySQL/MongoDB) and any required message brokers (e.g., Redis/Kafka).

### 

### **4. Run Application Server**

Use command (example):

npm start # For Node.js

python app.py # For Python Flask/Django

docker-compose up # If using Docker

### **5. Access the Platform**

Open browser and go to http://localhost:3000 (or configured port) to interact with the application.

### 

### **1. Open Backend Folder** → Start the backend service (FastAPI).

### 

### **2. Open Frontend Folder** → Start the frontend interface (Streamlit).

### **3. Connect Both** → Ensure the frontend is linked to the backend API URL.

### **4. Access in Browser** → Backend runs on one port, frontend on another.

### 

### **5. Test & Use** → Open the frontend page, confirm it communicates with the backend, and begin using the platform.

## **7.API Documentation**

### **1. Authentication & User Management**

Login API – for user login/authentication

Register API – for creating new user accounts

Profile API – fetch or update user profile details

### **2. Citizen Engagement Features**

Post Issue API – citizens can raise issues/concerns

View Issues API – fetch list of submitted issues

Comment/Feedback API – add citizen comments or feedback

Vote/Like API – support or prioritize an issue

### 

### **3. Analytics & Insights**

Dashboard Data API – returns statistics and engagement analytics

Trends API – shows trending issues or topics

### **4. Administration & Moderation**

Manage Issues API – approve, resolve, or close issues

User Roles API – assign admin/citizen roles

Reports API – generate reports on citizen engagement

### 

### **5. Utility APIs**

Health Check API – confirm backend is running

Notifications API – send alerts/updates to citizens

## **8.Authentication**

### **1. User Registration**

Citizens can create an account with basic details (name, email/phone, password).

System validates uniqueness and stores credentials securely (hashed passwords).

### **2. User Login**

Citizens log in using email/phone and password.

Backend issues a secure token (JWT or session token) upon successful login.

### 

### **3. Token-Based Access**

Each request from the frontend to the backend must include the token (usually in the header).

Token ensures only authenticated users can access protected APIs.

### **4. User Roles**

Roles may include Citizen, Admin, Moderator.

Role decides what actions the user can perform (e.g., posting issues, moderating content).

### **5. Logout & Session Management**

Users can log out, invalidating their token/session.

Token expiration ensures security (auto-logout after a set time).

## **9.User Interface**

### **1. Login & Registration Screen**

Secure access for citizens and administrators

Options for sign-up, login, and password reset

### 

### **2. Citizen Dashboard**

Overview of active issues, recent updates, and notifications

Quick actions (raise issue, give feedback, vote)

### **3. Issue Submission Form**

Simple form for citizens to submit concerns or suggestions

Fields: title, description, category, attachments (optional)

### **4. Engagement Feed / Community Wall**

List of submitted issues and feedback

Options to comment, vote, or share opinions

### **5. Analytics & Insights Panel**

Charts showing trending issues, participation levels, and resolutions

Helps citizens and admins track engagement impact

### **6. Admin Panel (for moderators)**

Manage user roles and permissions

Approve, resolve, or close citizen issues

Generate engagement reports

## **10.Testing**

### 

### **1. Unit Testing**

Test individual backend APIs (e.g., login, register, issue submission).

Ensure each function works correctly in isolation.

### **2. Integration Testing**

Test how frontend (Streamlit) communicates with backend (FastAPI).

Verify data flow between UI forms and APIs.

### 

### **3. Functional Testing**

Check core features: authentication, issue submission, feedback, voting, analytics.

Ensure workflows behave as expected.

### **4. User Interface (UI) Testing**

Test layout, responsiveness, and ease of navigation in Streamlit.

Validate accessibility and mobile compatibility.

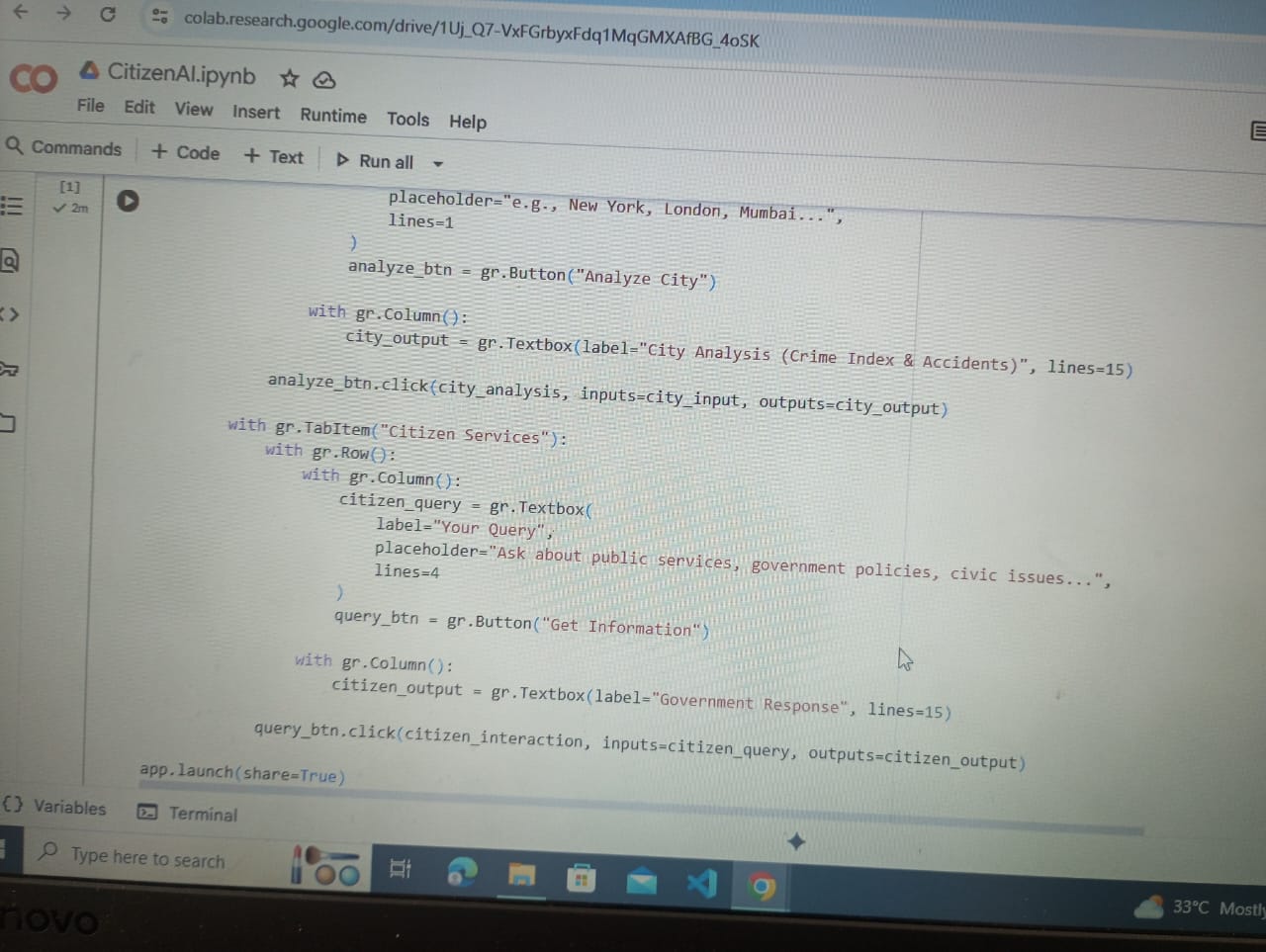
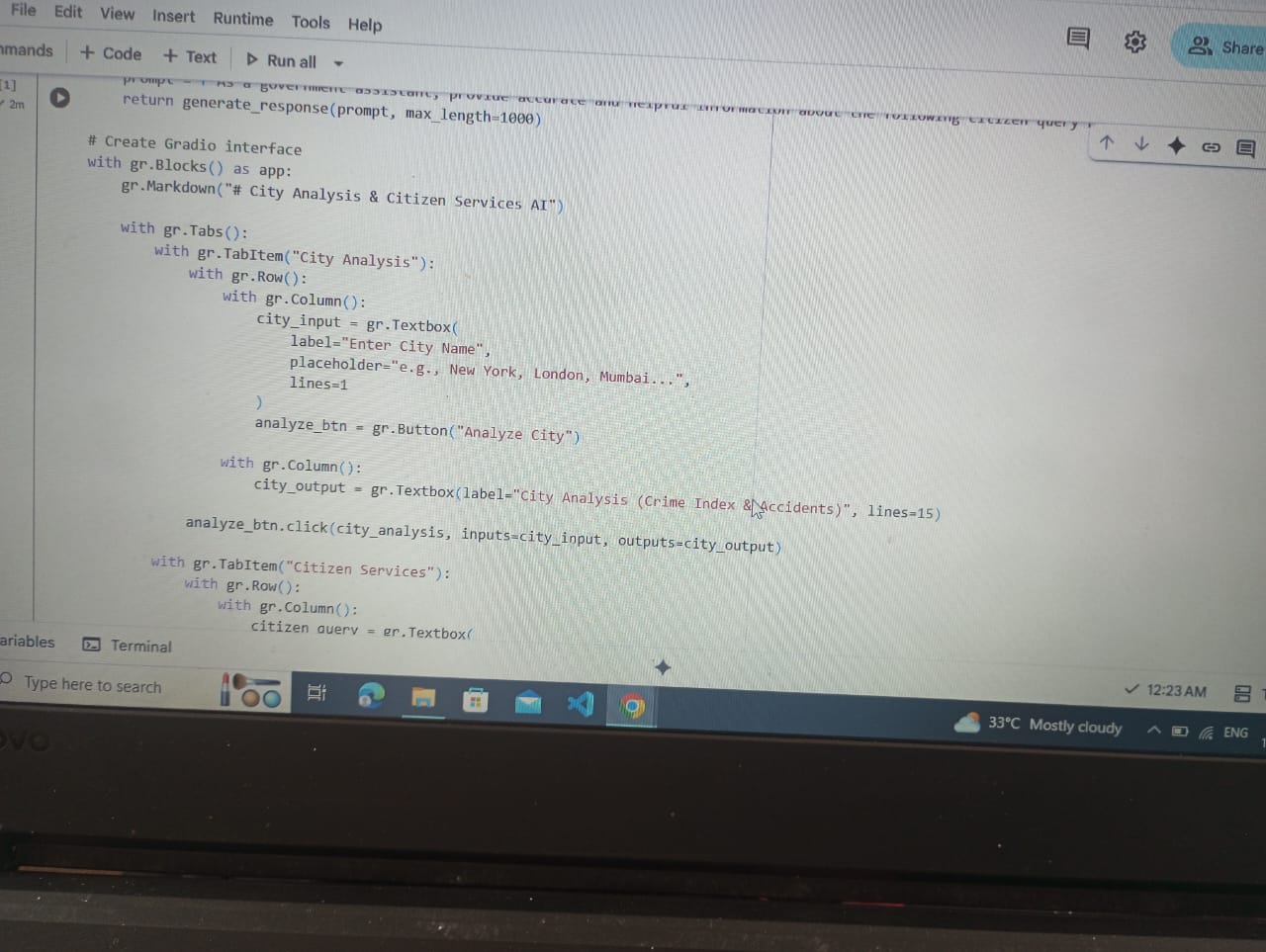
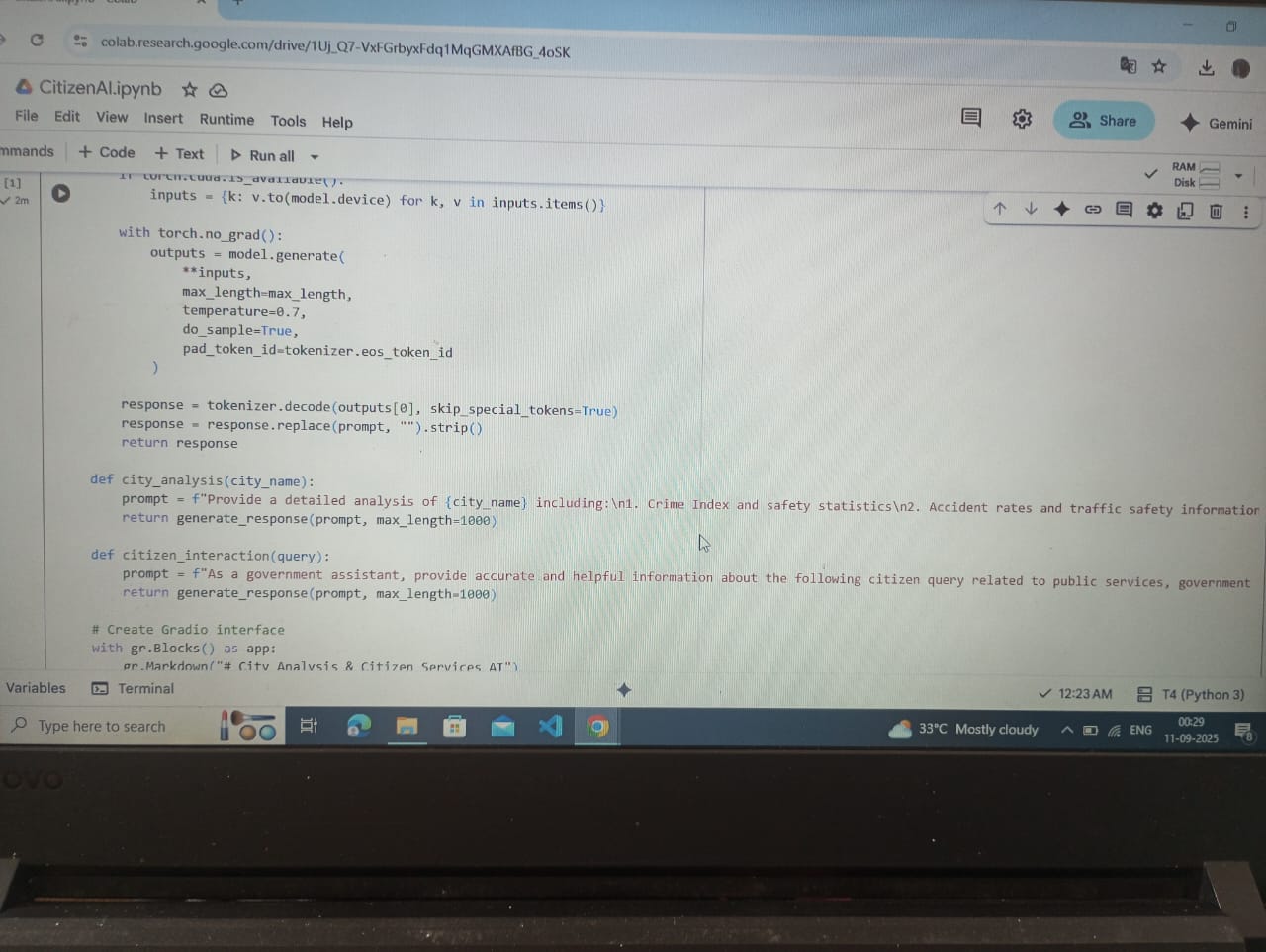
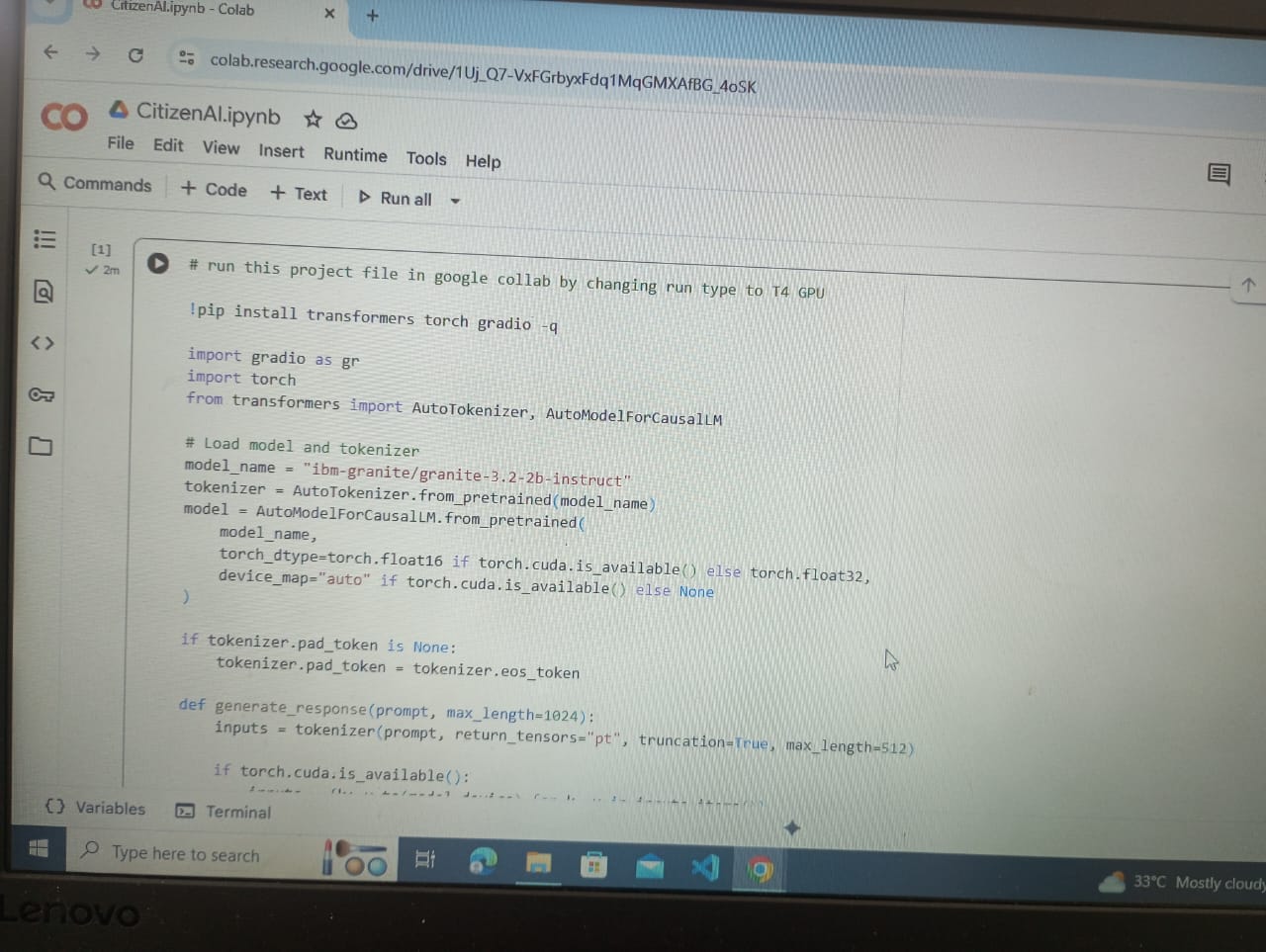
### **5. Performance & Security Testing**

Measure response times, load handling, and scalability.

Test authentication security, token expiration, and data protection.

## 11.Screenshot:

### **coding**



## 