**Shakirkhozha Ilyas Friday 14.00-15.00**

**Exercise 1: Managing APIs with Google Cloud Endpoints**

**Objective**: Deploy and manage an API using Google Cloud Endpoints.

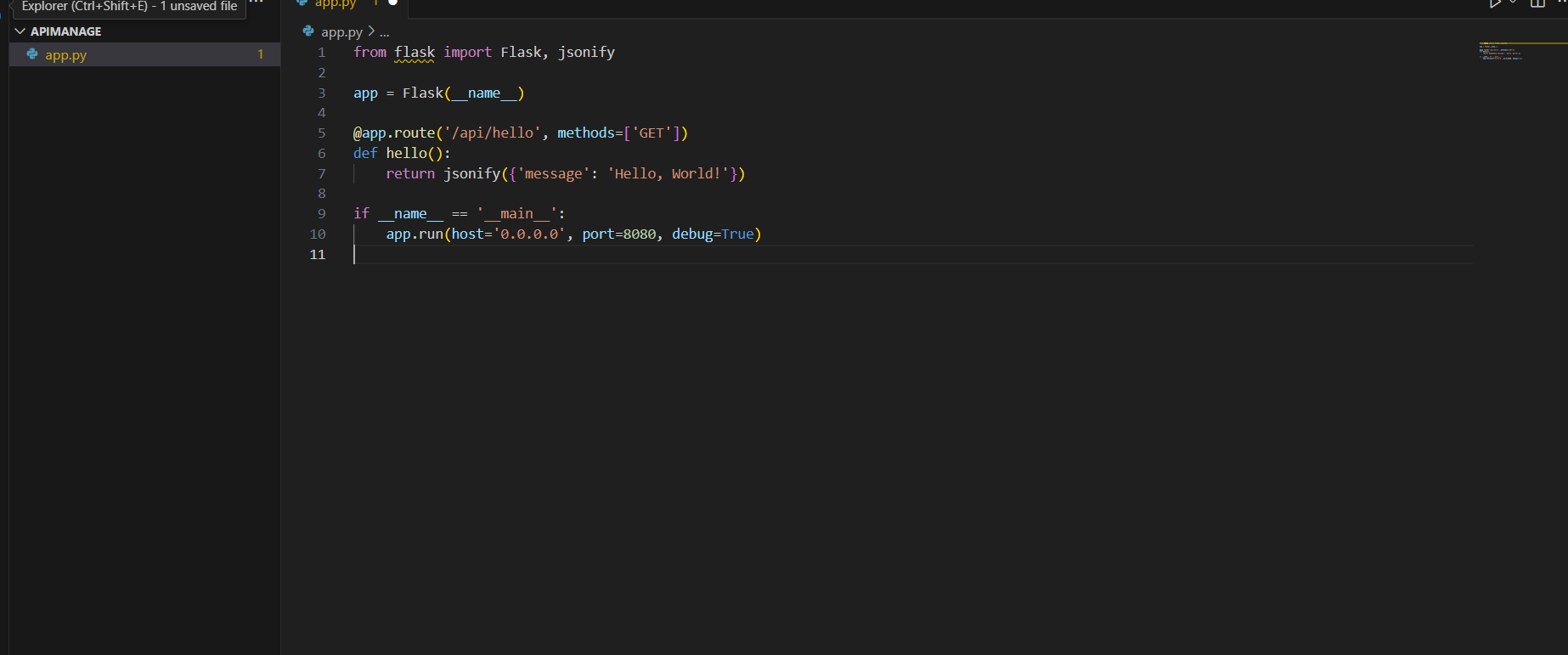
**Step 1:**

I enabled the \*\*Cloud Endpoints API\*\* in the \*\*APIs & Services\*\* section to manage and deploy APIs on Google Cloud.

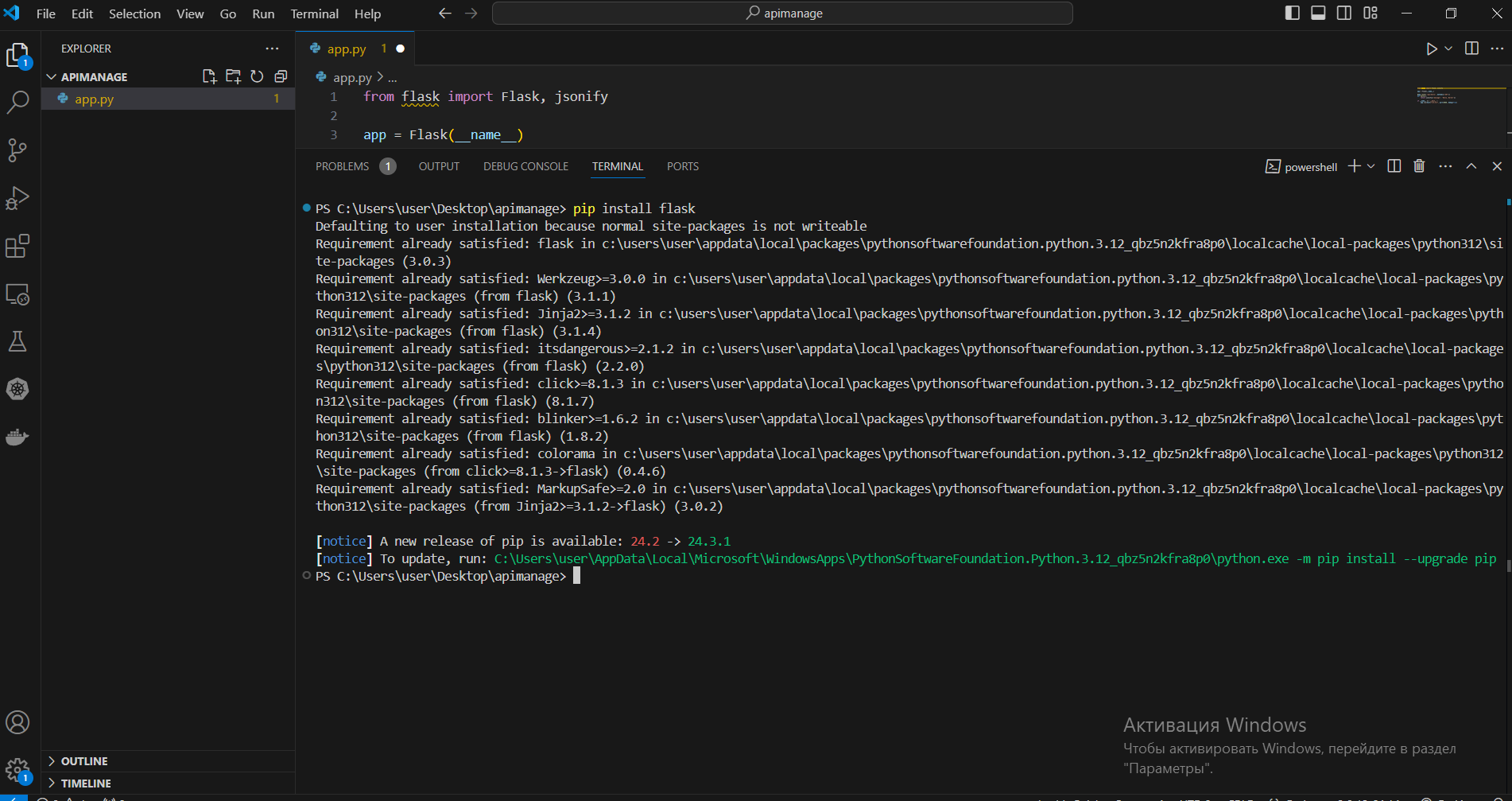
**Step 2: Preparing API**

I developed a basic Python Flask application that responds with "Hello, World!" when accessed.

The app.py file appeared like this:

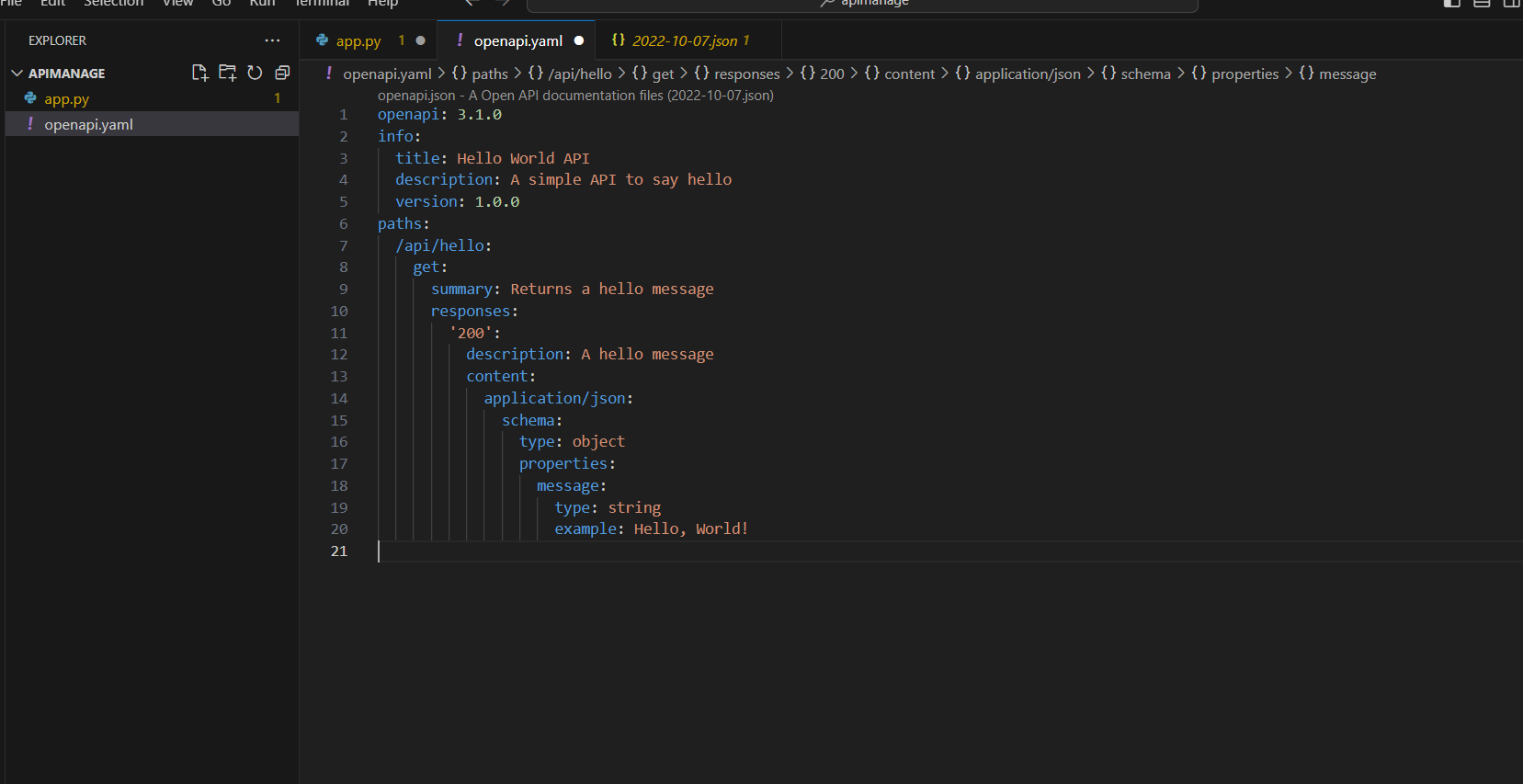


To execute the Flask application, I installed Flask with the command:  
pip install flask



**Step 3: Creating an OpenAPI Specification**

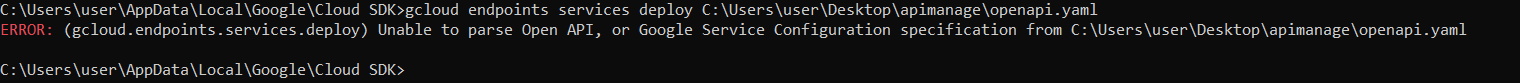
I created an OpenAPI specification in YAML format to define the API's structure, detailing the endpoint and response.The openapi.yaml file appeared like this:



**Step 4: Deploy the API to Google Cloud Endpoints**

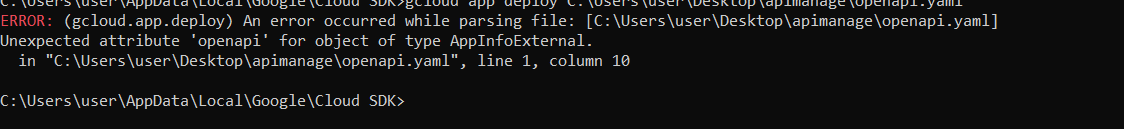
To deploy the API configuration in accordance with the OpenAPI definition, I executed the following command:

**gcloud endpoints services deploy openapi.yaml**



The deployment was done using the following command:

**gcloud app deploy**



Step 5: Test the API.



**Exercise 2: Google Cloud Databases**

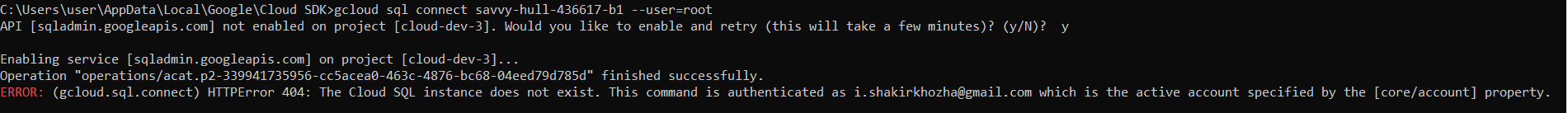
**Step 1: Create a Cloud SQL Instance**

1)I selected **MySQL** as the database type for this test, though **PostgreSQL** and **SQL Server** were also available as alternatives.

2)I configured the SQL instance to permit **public IP access**, allowing connections to the database from my local PC or other external applications

### Step 2: Create a Database and Table

Using the MySQL command-line client, I connected to the Cloud SQL instance to set up the database and create the required table.



Once connected, I created a new database called sample\_db:CREATE DATABASE sample\_db;

USE sample\_db;

I then created a users table and inserted sample data:

CREATE TABLE users (

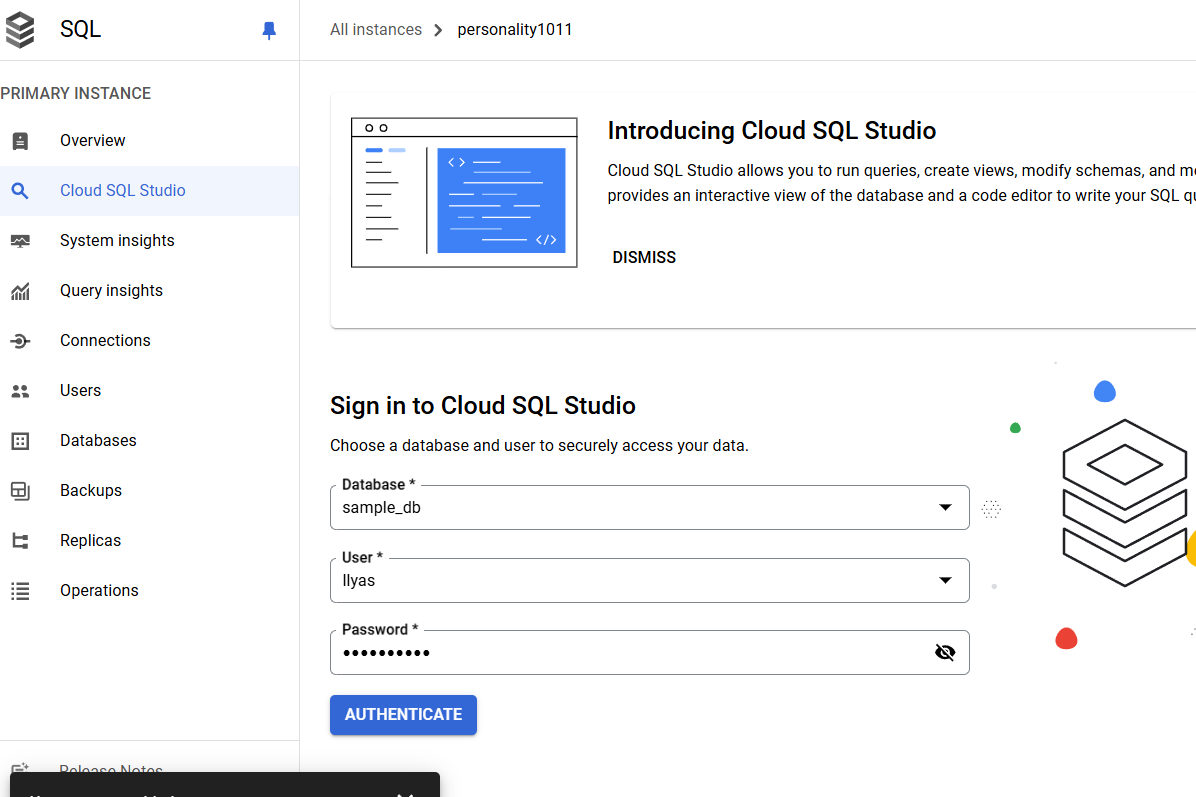
  id INT AUTO\_INCREMENT PRIMARY KEY,

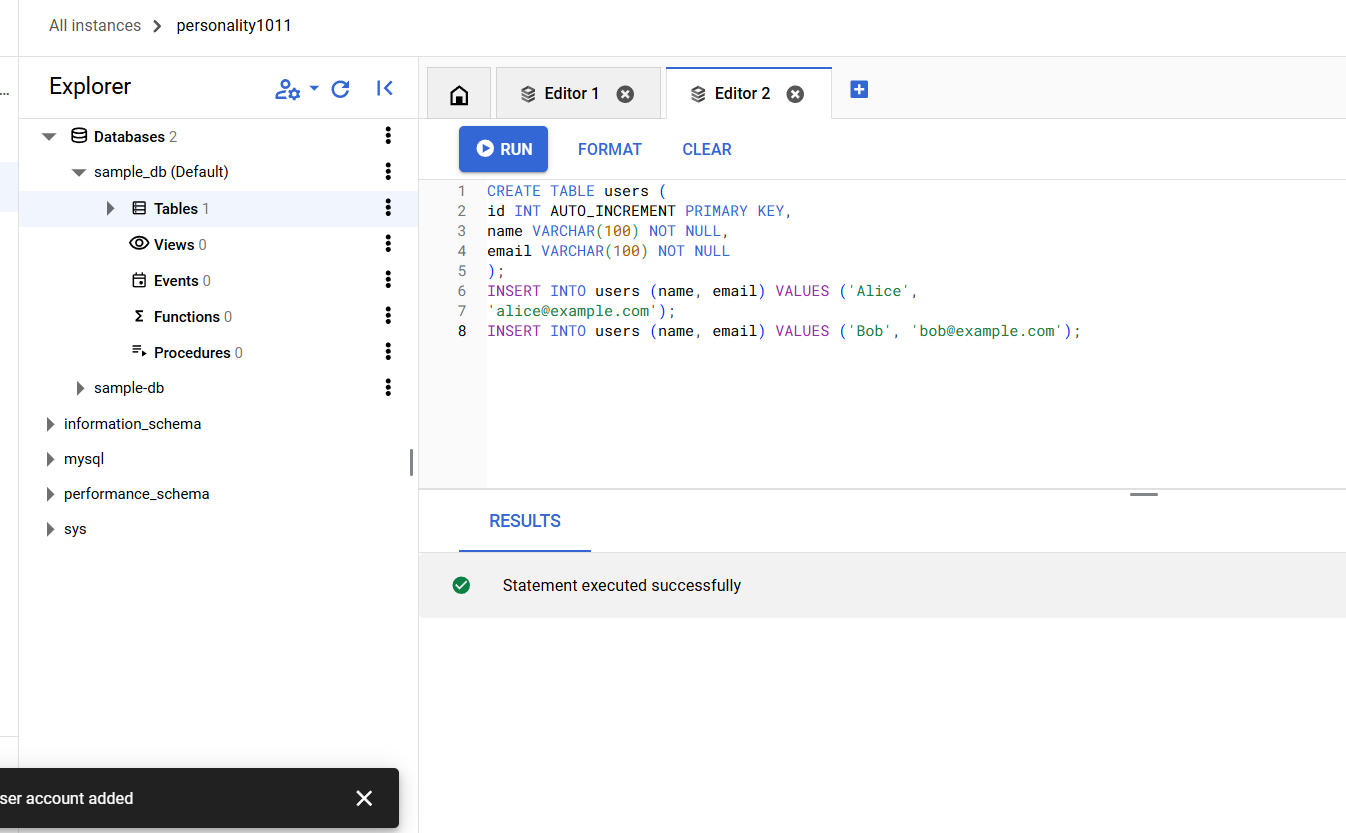
  name VARCHAR(100) NOT NULL,

  email VARCHAR(100) NOT NULL

);

INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');



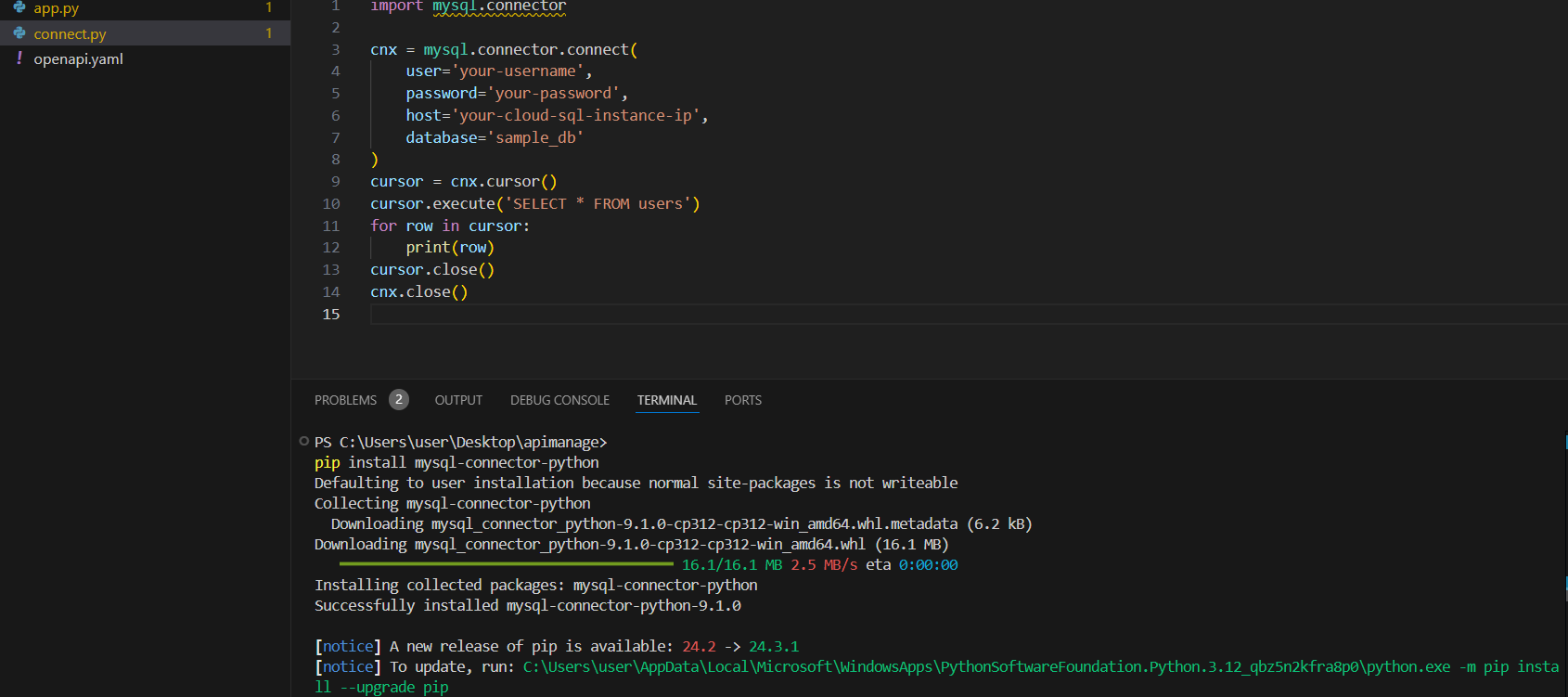


**Step 3: Connect to the Database from a Python Application**

I have created a Python script that connects to a Cloud SQL instance using the mysql-connector-python library. Below is the script's content

Before running the script, I installed the required MySQL connector library:

**pip install mysql-connector-python**



Running the Python Script, I executed the Python script to verify the connection and retrieve data from the Cloud SQL instance:

**python connect.py**

The expected output was:

(1, 'Alice', 'alice@example.com')

(2, 'Bob', 'bob@example.com')



**Exercise 3: Integrating Machine Learning with Google Cloud**

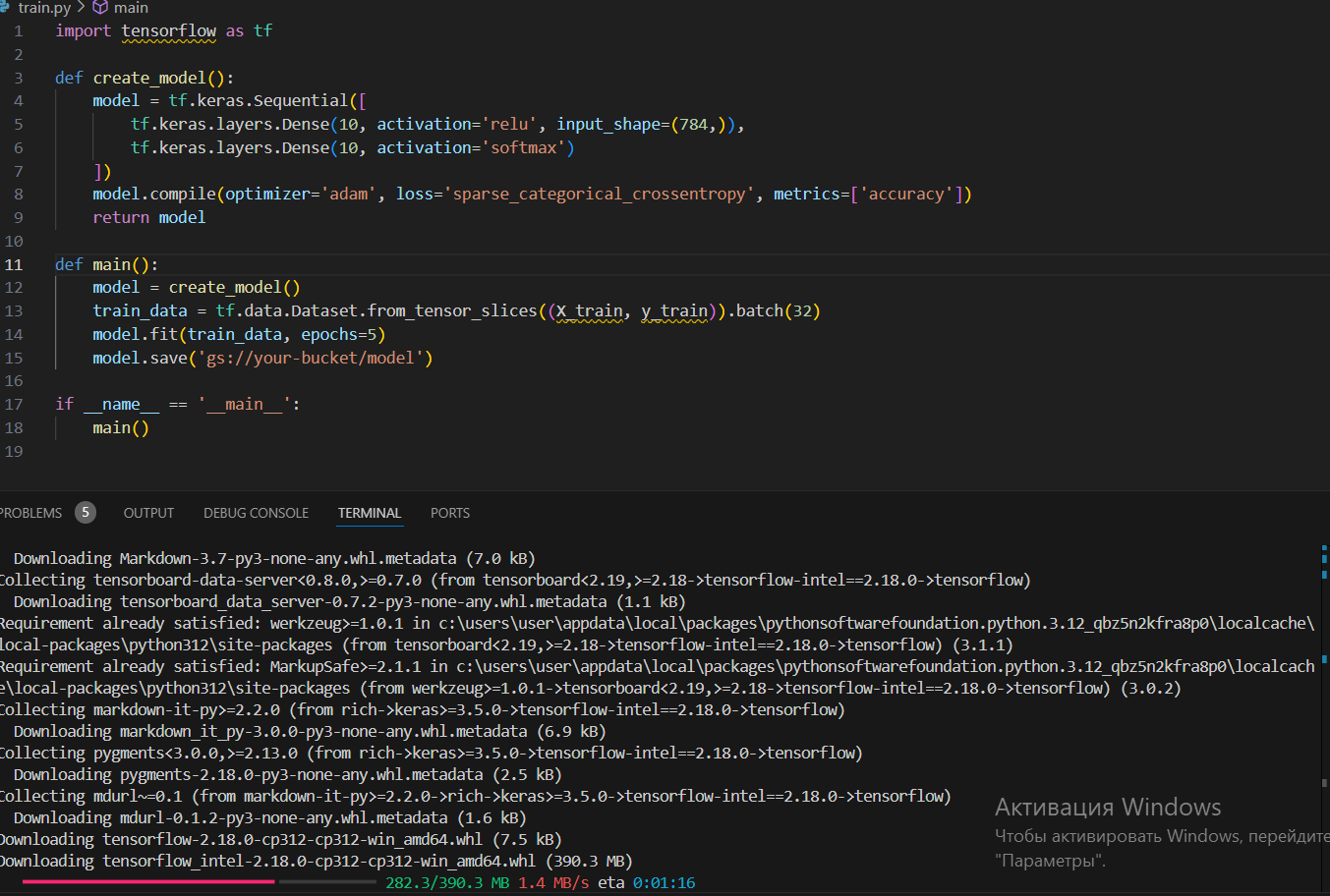
The objective of this experiment was to build and deploy a TensorFlow machine learning model on the Google Cloud AI Platform. This report outlines the procedures followed to complete the work, the challenges encountered, and the outcomes achieved.

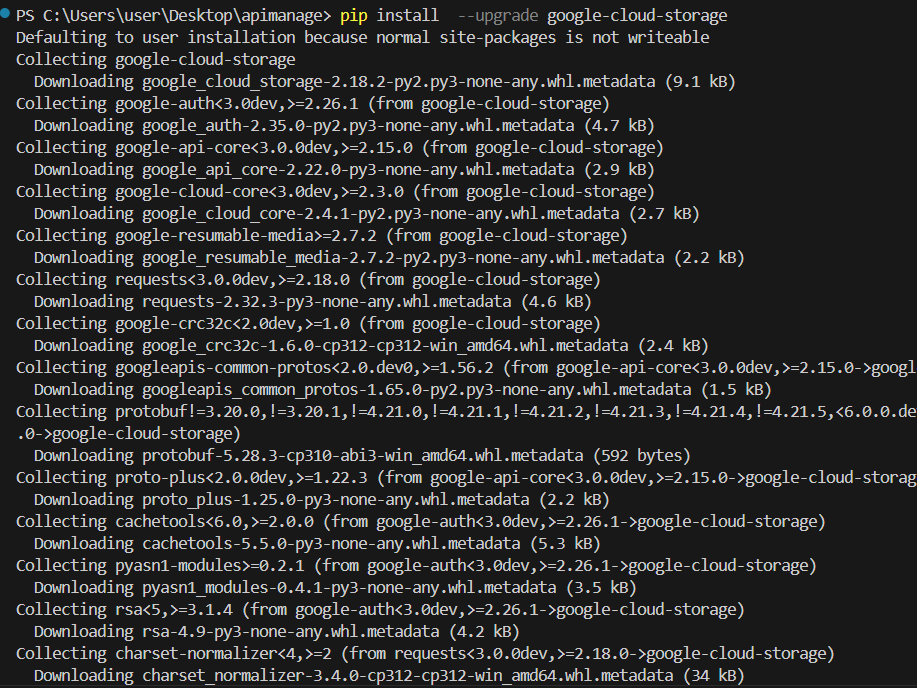
Setup Procedure:

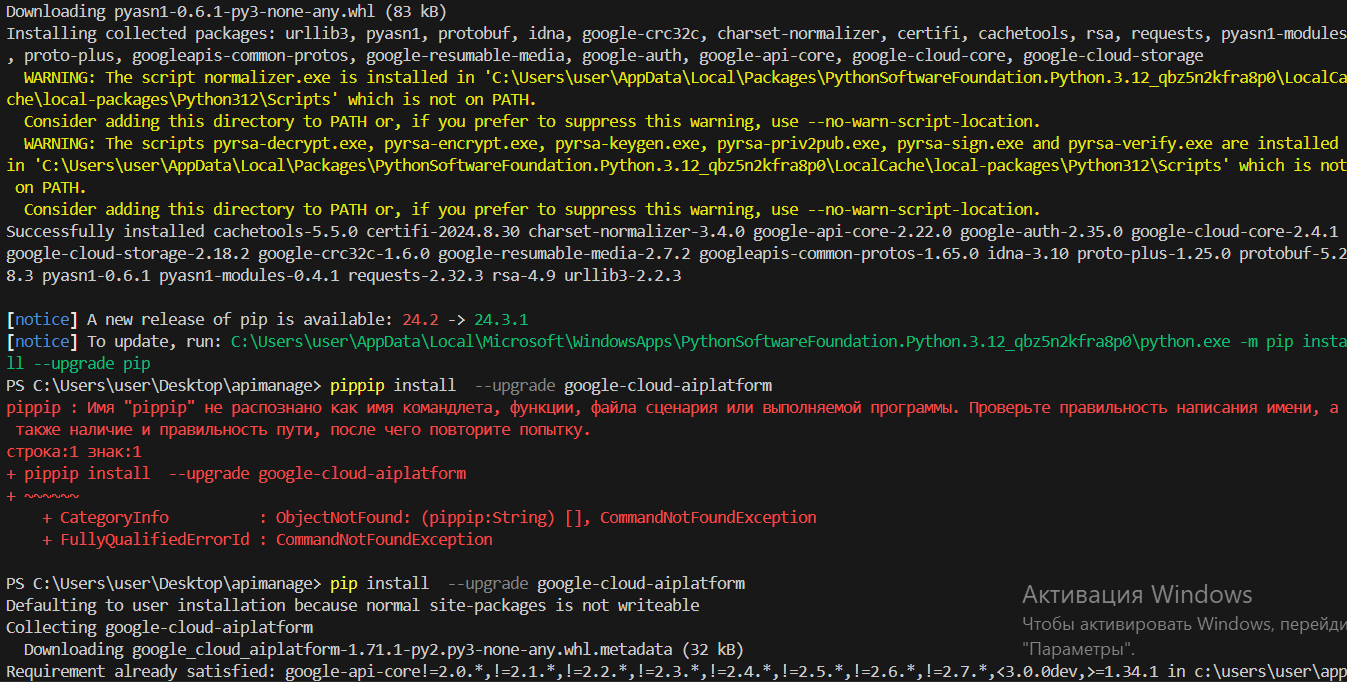
The first step was to create the necessary environment for developing the model. Since the Google Cloud AI Platform requires a paid account for utilization, I chose to use Google Colab, which offers free resources for machine learning projects. Google Colab provides access to both GPU and CPU, making it an ideal platform for such tasks without incurring costs.

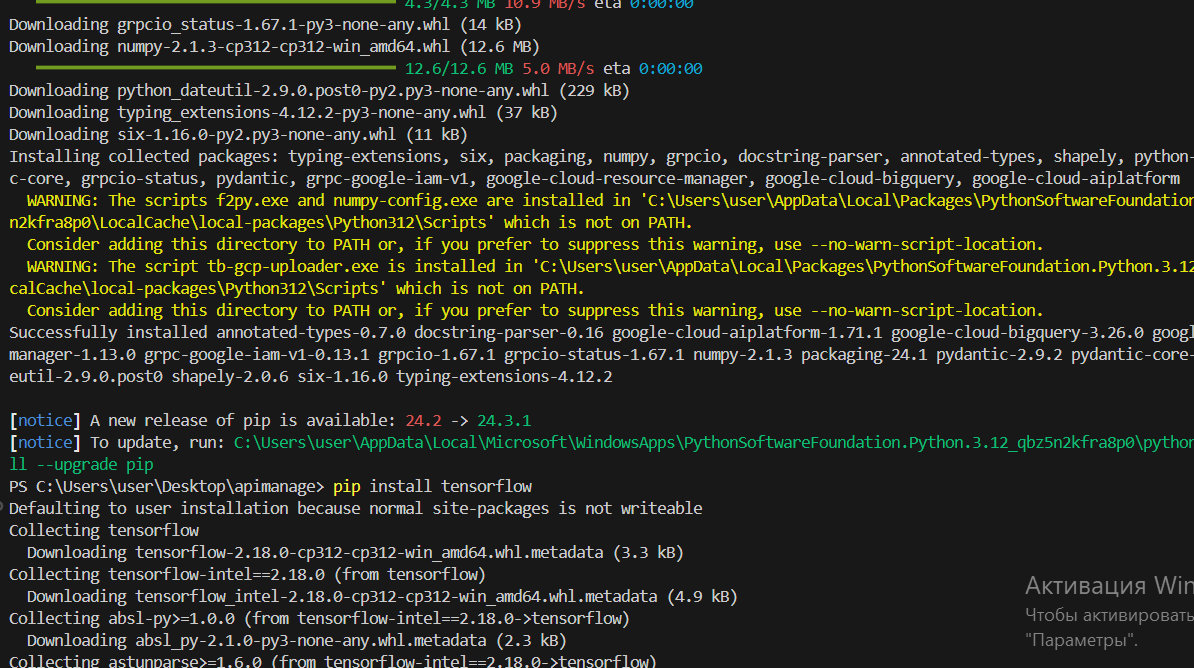
Actions Taken During Setup:

I ensured that all required libraries were installed in Google Colab, including TensorFlow, Google Cloud Storage, and Google Cloud AI Platform.

****

****

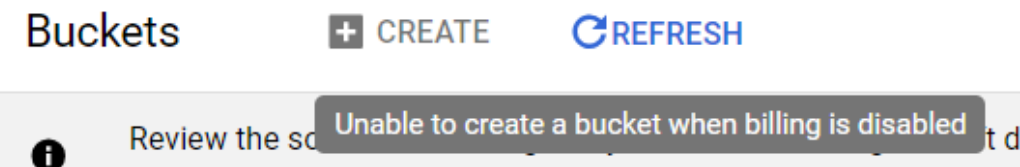
****

****

**Train the Model:**

**Model Training on Google Cloud AI Platform**

The training script is then submitted to the Google Cloud AI Platform, which automates the training process utilizing cloud resources. This step is crucial for managing larger datasets and more complex models that demand substantial computational power.



**gcloud ai custom-jobs create --region=your-region**

**--display-name=ml-job**

**--python-package-uris=gs://your-bucket/train.py**

**--python-module=train**

**--container-image-uri=gcr.io/cloud-aiplatform/training/tf-cpu.2-4:latest**

**6. Deploy the Model:**

After training the model, it is deployed on the Google Cloud AI Platform using an endpoint, enabling access for serving predictions via a REST API. Deploying on Google Cloud also facilitates version control, simplifying the management of updates and improvements to the model over time.

**# Create a model resource**

**gcloud ai models create your-model --region=your-region**

**# Deploy a version of the model**

**gcloud ai versions create v1 \**

**--model=your-model \**

**--origin=gs://your-bucket/model \**

**--runtime-version=2.7 \**

**--python-version=3.8**

**7. Test the Model:**

To ensure that the model functions as intended, a simple Python script (predict.py) was created to send data to the deployed model and retrieve predictions. The model's endpoint was accessed through Google Cloud's AI Platform Prediction Service. This script tests the deployed model by passing an instance of data and printing the prediction results.

from google.cloud import aiplatform

def predict():

client = aiplatform.gapic.PredictionServiceClient()

endpoint = client.endpoint\_path(project='your-project',location='your-region', endpoint='your-endpoint-id')

# Replace this with your input data

instance = {'input': [/\* your data here \*/]}

# Make a prediction

response = client.predict(endpoint=endpoint, instances=[instance])

print(response.predictions)

if \_\_name\_\_ == '\_\_main\_\_':

predict()