Project Report: User Authentication System using Firebase, Google App Engine, and Firestore

Introduction

In the ever-evolving digital world, the need for secure and scalable authentication systems has grown significantly. User authentication is the gateway to any modern application and must ensure confidentiality, integrity, and scalability. Traditional user management often leads to complex backend systems, especially when considering security measures and scalability. Firebase Authentication, combined with Google Cloud's serverless infrastructure like App Engine and Firestore, simplifies this process.

This project showcases the implementation of a secure, cloud-hosted authentication system that verifies user credentials using Firebase Authentication, serves the backend through Google App Engine using Python and Flask, and optionally logs user information in Firestore. This integration provides a seamless, efficient, and scalable solution for managing user identity and access.

Objectives

- Build a secure user authentication system using Firebase Authentication.
- Deploy a lightweight backend using Flask on Google App Engine.
- Use Firestore to optionally store and manage user credentials or logs.
- Create a modular and scalable cloud-native authentication flow.

Tools and Technologies Used

- **Firebase Authentication** – For secure user sign-in and credential management.
- **Google App Engine** – Hosts the Flask-based backend application.
- **Cloud Firestore (Native Mode)** – Stores authenticated user data.
- **Firebase Admin SDK** – Provides server-side authentication and token verification.
- **Flask** – Lightweight Python web framework for building the backend.
- **Gunicorn** – WSGI server to run Flask applications in production.
- **Google Cloud Console** – For managing cloud services, projects, and deployments.

System Architecture

The system is composed of several modular components that interact over secure channels:

1. **User Interface (Frontend)**: A mobile or web client authenticates users through Firebase Authentication.
2. **Authentication Token**: After login, Firebase issues a secure ID token.
3. **API Request**: The frontend sends this token to the backend via a POST request.
4. **Backend Processing**: The Flask backend deployed on App Engine verifies this token using Firebase Admin SDK.
5. **Data Storage (Optional)**: On successful verification, the user's details can be stored in Firestore.
6. **Response**: The backend responds with a success or error message based on the verification.

This architecture supports scalability, is serverless, and minimizes infrastructure overhead while ensuring security and responsiveness.

Implementation Steps

1. **Set Up Firebase and Firestore**
    - Create a Firebase project.
    - Enable Firebase Authentication.
    - Enable Firestore in Native Mode.
2. **Create Google Cloud Project**
    - Set up a new Google Cloud project.
    - Enable the App Engine and Firestore APIs.
    - Create an App Engine application (select region and runtime as Python).
    - Link a billing account (mandatory for deployment).
3. **Configure Firebase Admin SDK**
    - Generate the private service account key.
    - Save it as firebase_config.json in the project directory.
4. **Develop the Backend (main.py)**
    - Create Flask routes for basic health check and token verification.

- Use Firebase Admin SDK to validate tokens.

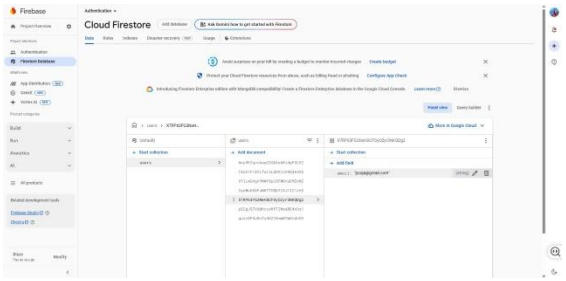
5. **Prepare Deployment Files**
   - **requirements.txt**: List all Python packages (Flask, firebase-admin, gunicorn).
   - **app.yaml**: Specify runtime, entry point, and scaling options.
6. **Deploy the Application**
   - Initialize and authenticate gcloud CLI.
   - Use gcloud app deploy to deploy the app.
   - Access the app using the URL provided by App Engine.

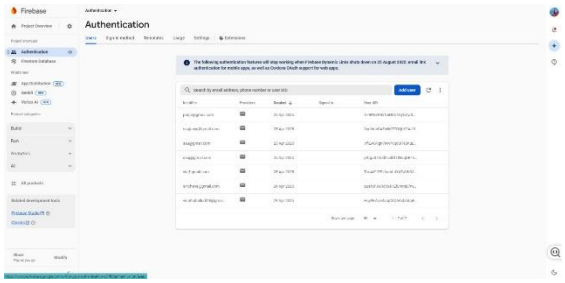
Code Summary

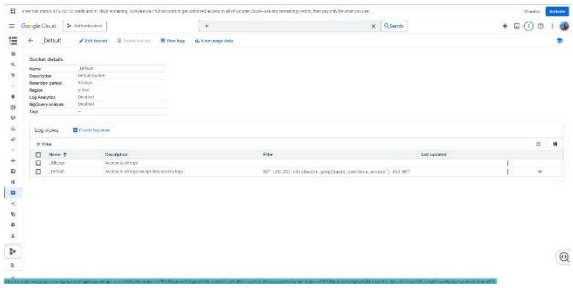
```
from flask import Flask, request, jsonify
import firebase_admin
from firebase_admin import credentials, auth

app = Flask(__name__)

cred = credentials.Certificate("firebase_config.json")
firebase_admin.initialize_app(cred)

@app.route('/')
def home():
    return 'Firebase Auth App Running!'

@app.route('/verify', methods=['POST'])
def verify():
    data = request.get_json()
    token = data.get("idToken")
    try:
```

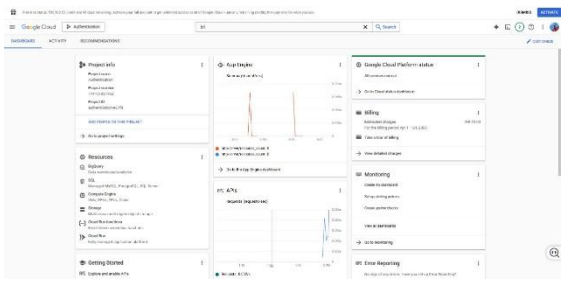
```
        decoded_token = auth.verify_id_token(token)
        return jsonify({"uid": decoded_token["uid"], "message": "Token verified"})
    except Exception as e:
        return jsonify({"error": str(e)}), 401
```

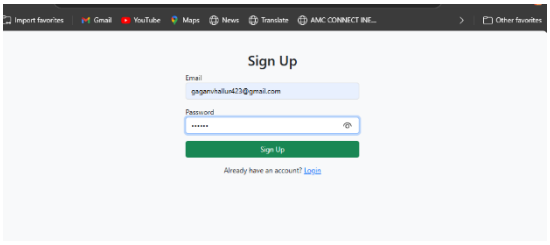
Project Screenshot

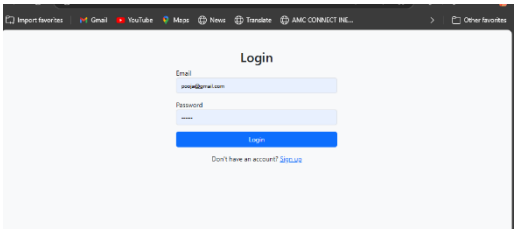


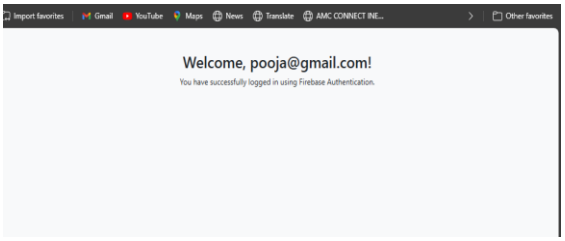


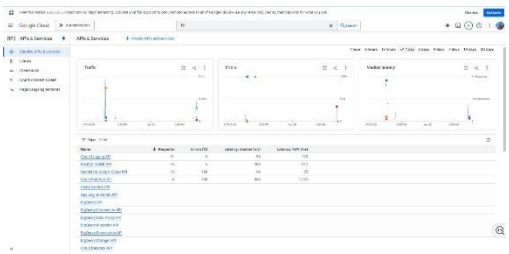












Challenges Faced

- Deployment failed initially due to missing billing account setup.
- Permissions and access configurations for Cloud Build and storage buckets.
- Compatibility issues between Flask and Werkzeug versions.

Results

- Deployed Flask API on Google App Engine.
- Verified Firebase ID tokens securely.
- Successfully hosted a serverless backend with optional Firestore logging.

Future Scope

- Add frontend UI for sign-up and login using HTML or React.
- Expand backend to support direct user registration.
- Log user login attempts, roles, and analytics in Firestore.
- Integrate with other GCP services like Cloud Functions, Pub/Sub, etc.

Conclusion

This project illustrates the development of a modern authentication microservice leveraging cloud technologies. Firebase simplifies identity management while App Engine and Firestore allow scalable hosting and data persistence without manual infrastructure setup. The modular and secure system provides a solid foundation for real-world applications such as learning platforms, administrative dashboards, and mobile apps. With minimal maintenance and high extensibility, this architecture can be seamlessly integrated into more complex cloud-based ecosystems.