**Backend Challenge**

Name: Abhiroop Basak

Email: [abhiroopbasak.tech@gmail.com](mailto:abhiroopbasak.tech@gmail.com)

1. **Overview**

The platform Collect gathers data and needs to store it efficiently to enable plug-in use system for various features once the data is collected in the primary database. A robust system design is to be built to ensure high availability and data security along with keeping in mind, system health checks and data store both on cloud and bare metal.

1. **System Architecture**

The entire system is built on two different areas: the cloud infrastructure and the bare metal device through which the user uploads data and requests for analysis. The two infrastructure designs are discussed as below:

The data on cloud is stored on MongoDB. MongoDB is a high availability and secured database to store all information related to forms. Another database can be set up to monitor the system details. This can be obtained by either setting up an SQL database or using Monitor System Logs as provided by Google Cloud.

The data can be read directly from MongoDB using a python script. The data can then be manipulated as and how required. A queue has been set up between the requests from the main website and the master controller to ensure smooth synchronization as third-party applications may take up time in authentication and processing.

Detailed explanation of the system can be found in the diagram attached: <https://github.com/abhiroopbasak/atlan_task1/blob/main/Atlan.dia>

III. **Google Sheets use case**

The problem talks about integrating Goggle Sheets with the data storage part so that organizations can collect their CRM to the sheets and generate graphs.

1. **Approach taken**

Google Cloud offers APIs to be connected with its products to enable high security transactions between databases and data stores. Google sheets can be connected to Google cloud account and can be used to carry out the procedure. The steps required to do so include the following:

**Uploading data into Google Sheets**

STEP 1: Log into the Google Console and create a project

STEP 2: Enable Google Drive API into the account

STEP 3: Create a Google Service account to use application data for transactions

STEP 4: Provide Editor or Admin access to the account.

STEP 5: Create a Key and download the Key file. I have used JSON format

STEP 6: Enable Google Sheets API

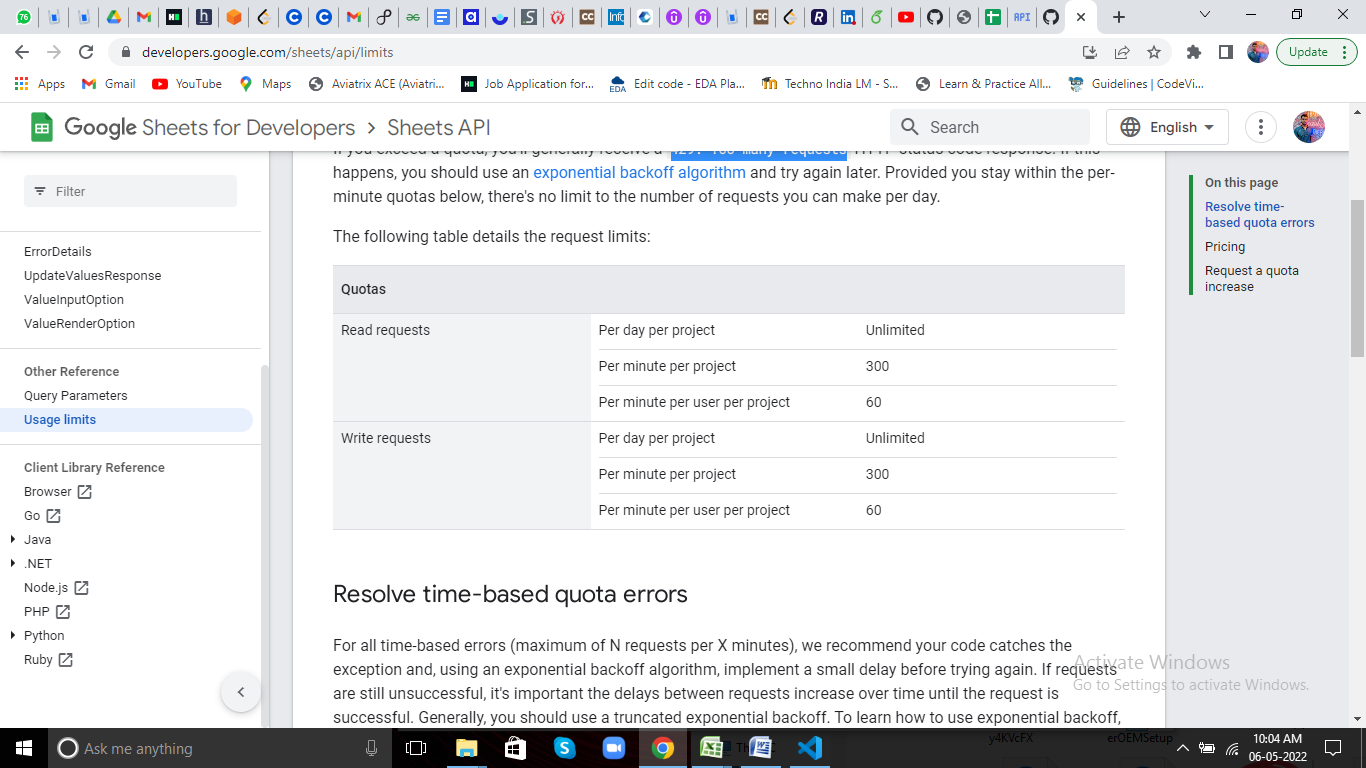
STEP 7: Obtain the google service account email id from the Key file and enable access to the email account from the sheet where the data needs to be dumped.

STEP 8: Write a script that uses ‘oauth2client’ library and the credentials from the key file to access the Spreadsheet and update data.

The data can be read and written into the Google sheets using the code. This will be fetched directly from the form storage and ensure both as a security storage in case of damages and an easier interface for users to obtain their data.

**Limitations:**

1. Google has a limitation on the number of read and write requests made into Google Sheets. While entering huge number of requests, an error message is received which reads as “***429: Too many requests***”. The details of the limitations are displayed below.



This can cause an issue of scalability with huge number of data transaction. The issue can however be solved by storing cache after a certain number of requests inducing a small delay to avoid an issue with exceeding the limit.

**Graph generation**

The Google Sheets API can be used to generate graphs as per specification and requirement through scripts. Google Sheets API provides a series of graphs such as column chart and pie chart that can be generated using JSON calls. The code for extracting the same can be added as plug-ins through functions without affecting the central Spreadsheet.

**Opportunities**

1. Only specific graphs allowed by Google Sheets can be added to the system without further authentication from Google at the site of the Spreadsheet. Provisional web pages to display graphs are also available as per unique requirement. This is because a copy of the dataset is obtained while implementing the code at the client side.

**Limitations**

1. Connecting to Spreadsheet for the first time may take some time. This depends further on the readability of the sheet and the number of sheets associated with the account.
2. **Implementation**

The demo code is designed to implement uploading to Google Sheets. Further storage security can be enhanced by using cloud databases and storage as mentioned previously. However in the demo, the plug-in format is displayed.

**Opportunities**

1. ***Middleware*** may be added to enhance the performance of the data.
2. ***Client side cache database*** can help in ensuring data is not lost and robustness in case of network failure.
3. ***Containers*** can be used to store the image of the model and preserve in case of system damage.

**Guide to GitHub:**

1. Atlan.dia: **System design**
2. Columnchart.json,piechart.json: **json plug-ins**
3. Manage.py: **Django run file**

View Now: <https://atlanbackend.abhiroopbasak1.repl.co/>

Code: <https://github.com/abhiroopbasak/atlan_task1>

Development: <https://replit.com/@AbhiroopBasak1/atlanbackend>