BACKEND CHALLENGE DOCUMENTATION (NANDA M.P)

https://github.com/whitedevil31/atlan-backend-challenge

INTRODUCTION:

First and foremost thank you for giving me an opportunity to showcase my skills and inviting me to the challenge round . I hope the solution I have developed fits in your use case and could solve the issue in the current scenario

I went through the problem statement ,tinkered with the Atlan Connect platform and was finally able to find an idea to implement for the presented tasks.
To list out I identified these as the primary list of tasks to deal with :

TASK #1 Design an efficient way to model the questions and responses of the forms and how I would store them in the data store

TASK #2 Efficiently execute the post-submission triggers (the events that the customer expects to run after a form is submitted). The logic should be fail-safe and it is expected to not to miss any response and should scale to million requests .

TASK #3 Design a logic such that new upcoming triggers should be easily be plugged into the existing code and consistency should be maintained

TASK #4 Monitor health checks ,implement logging for the application . Also to handle limitations when interacting with external services which in my case would be Google Sheets API and the mailer service(Mailgun)

In this documentation ,I would brief through each TASK and explain how I approached it ,what were my various approaches and why I finally decided to implement a particular solution .I have also attached a postman collection.Please refer to it for all the endpoints discussed in the document

POSTMAN COLLECTION (https://www.postman.com/collections/c22d05a8c75dba16d75f)

Golang - As it was mentioned in the Job description

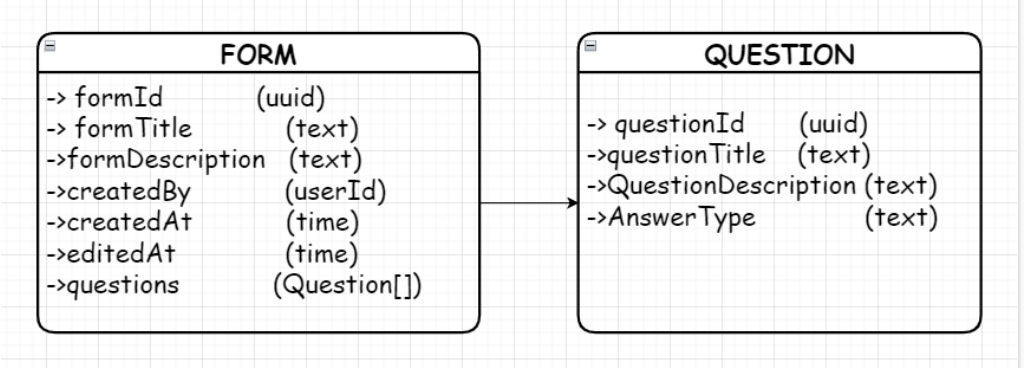
MongoDB - MongoDB is my go-to DB for any database interaction

Kafka -To handle event triggers

Docker to containerize the application

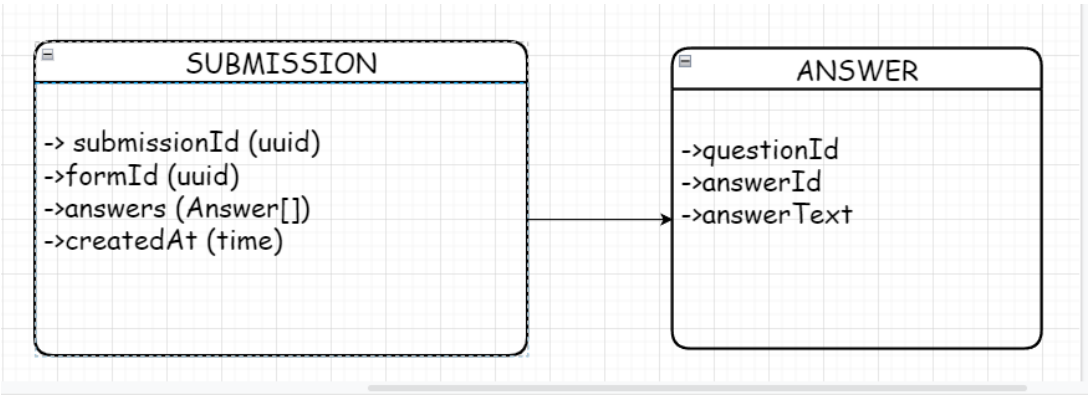
TASK #1(To design an efficient way to model the questions and responses of the forms and how I would store them in the data store)

So initially when creating a form we will add all the questions which are to be required in the form.The schema of the form is described below



Once a user adds all the required questions and fills necessary information like QuestionTitle ,AnswerType which could be optional/required depending upon the use-case and clicks on the create form button in the frontend the form will be created.
The backend logic will be simple. We will have a REST API Endpoint of POST method (http://localhost:8080/api/add_form) to handle the request and save the formData .We will store the form data in the **"forms" collection.**
When a form is opened its data can be fetched from the forms collections using the formId .Once it is filled and submit button is clicked ,we will have a REST API Endpoint of POST method(http://localhost:8080/api/submit_form) to handle the submission .

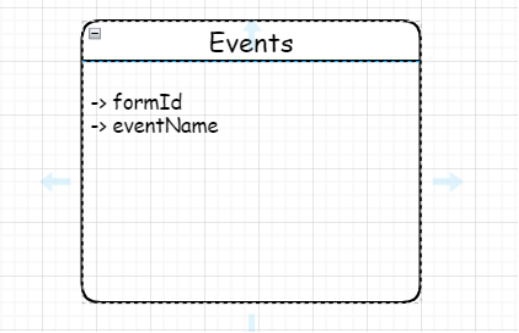
The schema for the submission is described below



The intuition behind saving the Id of all the entities instead of actual data of the entity itself is once we have the ID we can use the query/aggregation feature of MongoDB to bring all the required additional data .We can even **compute analytics** for a particular answer/questions through this _id. The _id of MongoDB **enables faster query** over other keys and also by this we **don't consume redundant data** onto the disk .We will save this submissionData in the

“submissions” collection.

For handling the list of events to trigger for a specific form we will maintain an “events” collection where we save events along with the formId .
We can add events by sending a POST request to (http://localhost:8080/api/add_event) and get all events for a form by sending a GET request to (http://localhost:8080/api/get_events/{formId})



#TASK 2 (Efficiently execute the post-submission triggers)
Here I would like to demonstrate the scenario by executing these two events after our form is submitted .
EVENT 1 : Add the submission result in a **Google sheet**
EVENT 2: Send a Email notification to the user to confirm his form submission (I picked Email service instead of SMS as there aren’t enough free SMS service providers and both Email and SMS examples are almost similar)

Initially I thought of simply executing these events in the form of conventional functions or HTTP CALLS (Assuming these event logics are written in an external API) then given the condition that the solution needs to be **failsafe,scalable,fault tolerance** lead me to a conclusion that this can’t be a ideal solution . It took me a day to find out about **Apache Kafka** and I believe it could provide a really good solution here .

Kafka acts as our event streamer here . Once our form response is stored in the database,our primary API will act as a producer-api and it will send messages to the consumer-api and return the response .The consumer will execute the functions(trigger events) then in its own server . By this we don’t need to wait for the events to finish to return the HTTP response from our producer.

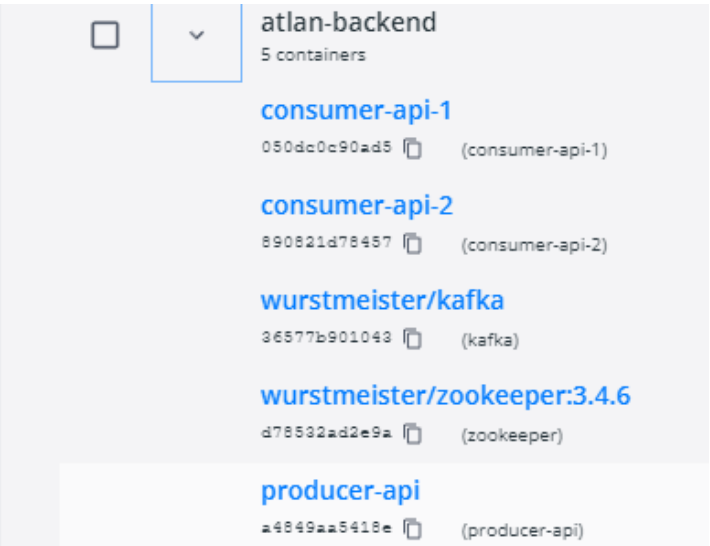
I found multiple ways to achieve the above task
#WAY1 : Trigger events as TOPICS | single consumer in single consumer group for all topics
#WAY2 : Trigger events as TOPICS | single consumer in multiple consumer groups with different topics
#WAY 3 Trigger events as TOPICS | one consumer group with multiple consumers

#WAY 4 Have a single topic | multiple consumer groups with multiple consumers with specific topics

I chose WAY 4 after researching a lot on various forums . The reason for using this approach is that even though will be streaming on a single topic since we have multiple consumers (number of consumers will be almost equal to **number of partitions of our primary topic**) in the consumer group the **traffic will be distributed** and if incase we are to increase the number of topics we can go ahead with multiple consumers in multiple consumer groups with different topics .
One more reason why **I didn't go ahead** with the other approaches was the **lack of dynamic topic creation** support in the consumers and creating a consumer for each topic could lead to a lot of **unwanted deployments in the cluster in future**

By this **#WAY4** approach we can expect a really **good throughput with low latency .**

In my codebase I have a **single Kafka broker** with a **single producer with 2 partitions** for the topic . A **Zookeeper** to track our cluster information like Kafka brokers ,leader election and configuration data of topics and partitions . **Two consumers**(for demonstration purpose) in a single consumer group listening to specific topics .We can easily increase the consumers in our consumer group by simple duplicating the main deployment of our consumer
I have also implemented the **concurrency** approach of Go by creating a separate **Goroutine** for a util function call and used channels to transfer data



TASK #3 (Design a logic such that new upcoming triggers should be easily be plugged into the existing code and consistency should be maintained)

In our current flow the producer API sends messages to a single topic and these message consists of a key,value pair where the **key is the type of event to trigger and value is the data**

So based on the key I determine what type of event to trigger and call it .

```
data := &SubmitResponse{}
json.Unmarshal([]byte(m.Value), data)
events := new(events.FunctionStruct)

functionCall := reflect.ValueOf(events).MethodByName(string(m.Key))
functionCall.Call([]reflect.Value{reflect.ValueOf(data)})
```

I used the built-in **reflect package** of Go to call functions based on an input string in a dynamic fashion . Using this approach we maintain consistency of the codebase and **avoid the need of multiple if else/switch statements to alter between different functions** which could be lengthy as well as **involve lot of redundant code**

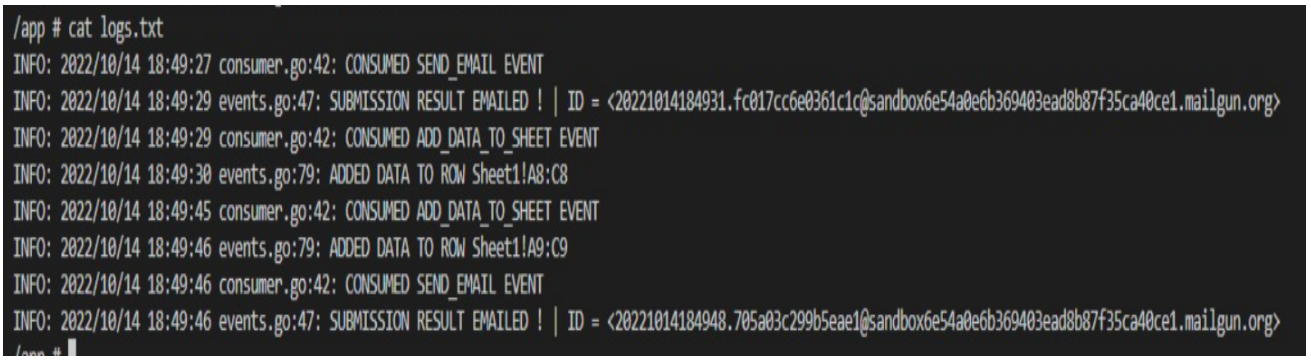
All the functions we need to add can be included in the events.go file with the event name as the function name .
So the entire flow for adding a event to an form is :
STEP 1: Send a POST request to (http://localhost:8080/api/add_event) to attach an event to a form
STEP 2 : Define the business logic in the events.go file as a function with function name as the **event name** which we used in the the POST method above(refer to postman collection)
That's it ,just two steps and that's all we have to do to add a new trigger to our form.
By this approach, we can easily integrate all our new logic which makes the process more generic and can be **considered as just plug and play fashion**

TASK #4 (Monitor health checks ,implement logging for the application ,external API limitation)

IMPLEMENT LOGGING

There are various approaches to implement logging for our API .I went ahead to implement a simple custom log handler which could add logs to a common file **logs.txt** . We can pass the required data to log into and can even classify logs based on the context . But in production grade applications it is advised to use a popular package since it could provide a lot of built

in features and I found **Zap** to be a really good fit



MONITORING APPLICATION

I was not really sure on how to implement a monitoring service from scratch . But on reading up I found that it is advisable to use open source applications for such purposes and I found **Prometheus** to be a suitable one . It provides all the necessary monitoring features like Service metrics,host metrics,service discovery etc and in particular to Kuberenetes based applications it is the best solution . Along with this we can use **Grafana** to provide visual support like graphs for the prometheus monitoring and it even has its own dashboard support . I haven't implemented the monitoring support in this current API due to lack of time constraints but I believe it is pretty straight forward and can be integrated easily

THIRD PARTY API INTEGRATION AND LIMITATIONS

I am using two third party applications in our server namely **Google Sheets** and **Mailgun API .**

Google sheets API

Initially I created a developer account but it had a lot of complications regarding token management so I went ahead to switch to a **service account** . Now I don't need to manage refresh/access tokens and I have a json which has all secrets for authentication
The sheets API does have a rate limit and it's **300 read/write per minute for an accoun**t and it's **60 read/write per minute per project** and per day per project limit is unlimited .This rate limit refreshes every minute .
Once the limit is reached the server returns a **429** status error . The user might need to wait for the next minute for the limit to refresh
We can **request for additional higher quota** for which we might need to pay upfront and the process could take around 2-3 days

Mailgun API

I already had a mailgun account and a subscription which I got from my Github student developer pack . The limitation of that account was 20,000 emails overall and 100 email verifications . But it was restricted to 100 emails per hour .
Upon my request for educational purposes they removed the limit .But now it's been a

year and my account is reverted back to a conventional account.

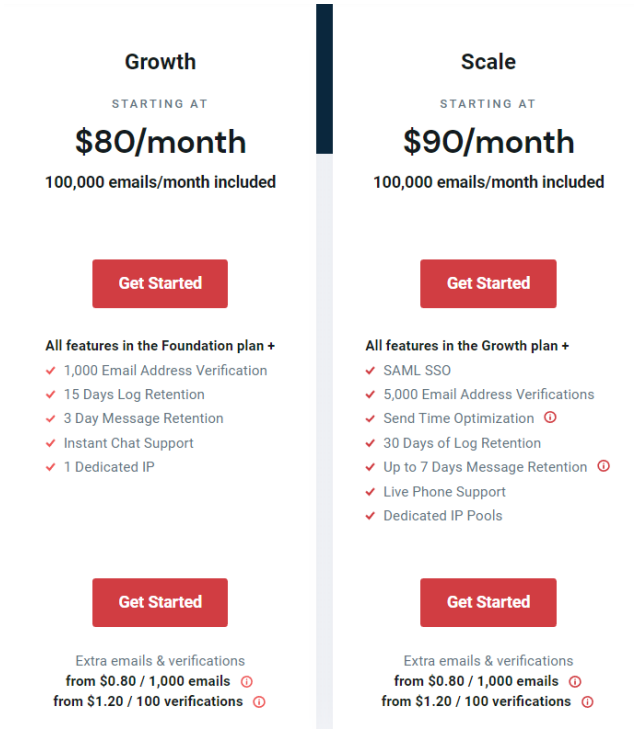
Hello Nanda,

Thank you for contacting Mailgun support.

After reviewing the account in detail, we have removed the probation, and now the account is fully enabled. This means that your emails will no longer be restricted to 100 per hour. The probationary status originated from our reputation monitoring system flagging various factors on the account. We reviewed these factors and cleared them from the system for your account.

For a conventional account it is 5000 emails /month and 300 emails/day and can be sent to only authorised recipients (Max is 5 recipients) . In our server we will send **Email to only np6771@srmist.edu.in account as I verified only that email** . When using a paid account we can send email to any account we require

We can pay for Growth/Scale type accounts which provides a really good number of domains and emails/month and maintenance of log feature is also provided

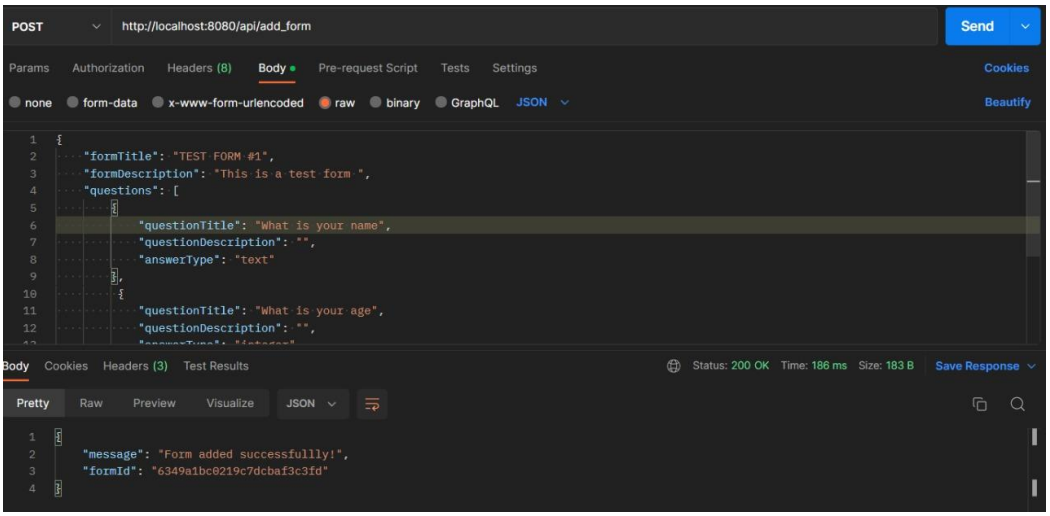


TO RUN THE APPLICATION

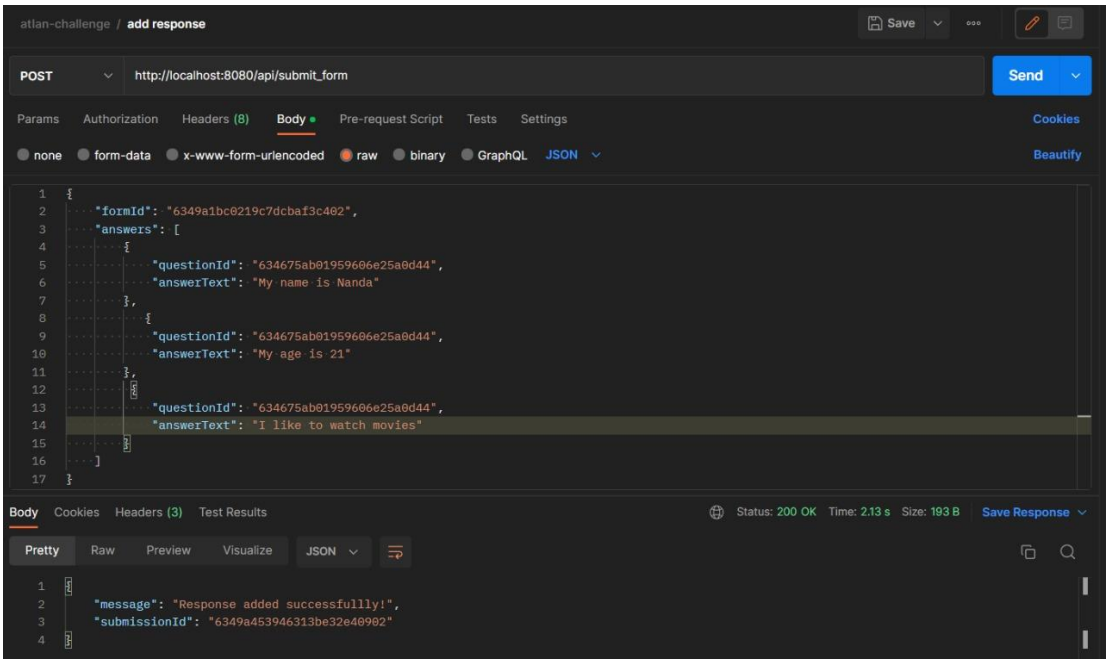
I have dockerized the whole application and it can be run from running the docker-compose file in the root directory by executing **docker-compose up** command

But the application uses many environment variables and secrets like the domain , API keys of the third party service I use and database passwords . I have not included them in the repository considering safety .You can run the application using your own suitable keys or I can demonstrate it with my keys on a online-meet.As a proof of my work I am including screenshots of all the key factors of in the application

TO CREATE A FORM



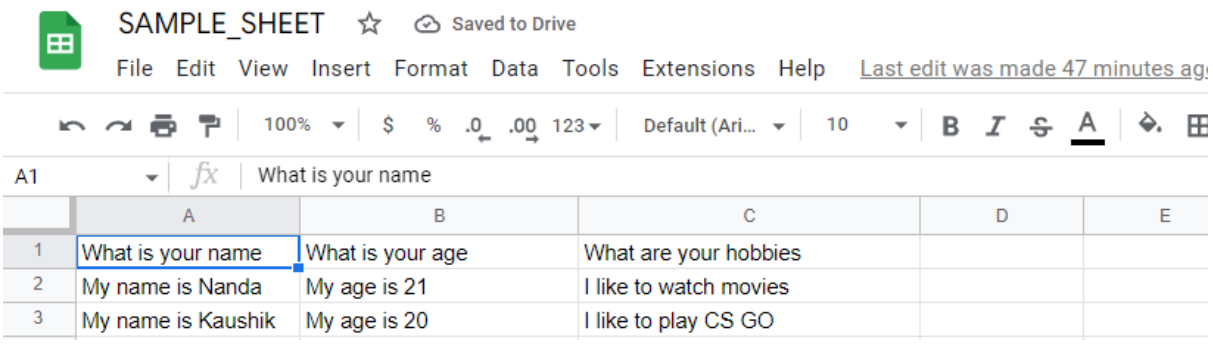
TO SUBMIT A RESPONSE



DOCKERIZED APPLICATION

```
PS D:\projects\go\atlan-backend\consumer-api-1> docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS          PORTS                                                NAMES
ab03e910dc4f   consumer-api-2                 "./consumer"             About a minute ago   Up 11 seconds   0.0.0.0:3000->3000/tcp                               consumer-api-2
9528af53eb59   producer-api                   "./producer"             About a minute ago   Up 11 seconds   0.0.0.0:8080->8080/tcp                               producer-api
9640eabc0c6b   consumer-api-1                 "./consumer"             About a minute ago   Up 11 seconds   0.0.0.0:5000->5000/tcp                               consumer-api-1
36577b901043   wurstmeister/kafka             "start-kafka.sh"         3 days ago       Up 16 seconds   0.0.0.0:9092->9092/tcp                               kafka
d78532ad2e9a   wurstmeister/zookeeper:3.4.6   "/bin/sh -c '/usr/sb…"   3 days ago       Up 17 seconds   22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp   zookeeper
```

ADD DATA TO SPREADSHEET



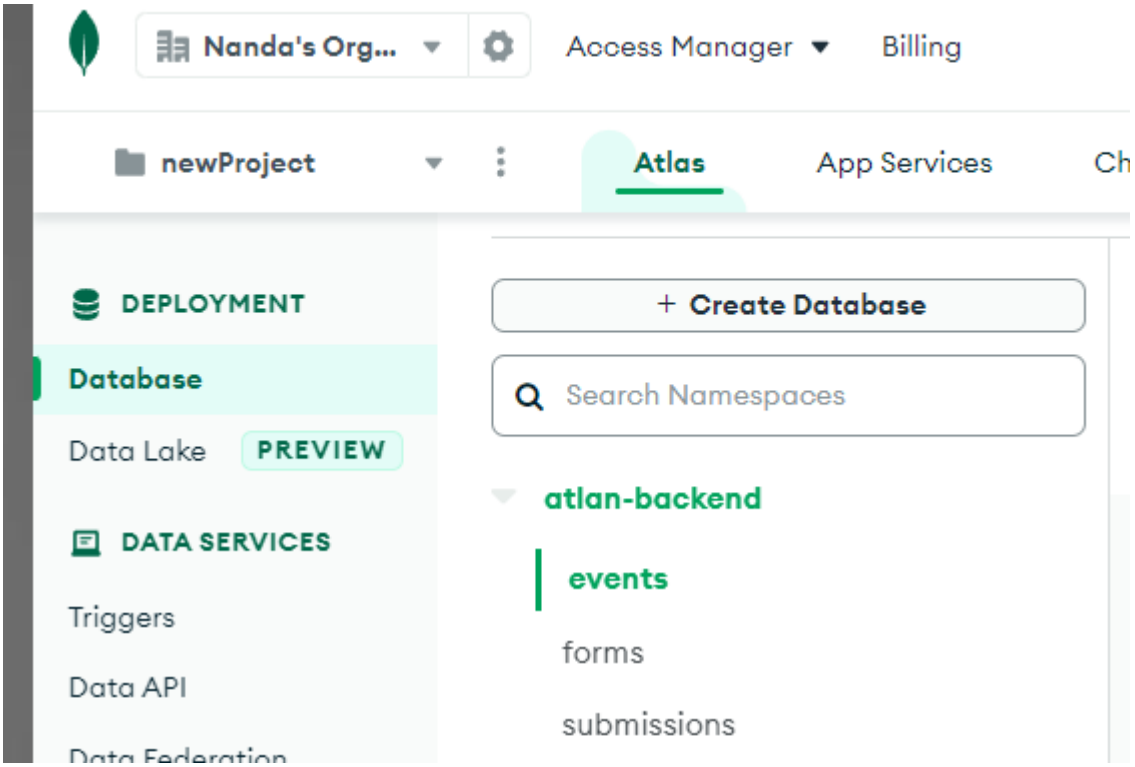
E-MAIL THE USER(I didn't add the data as I didn't have time to create a table format)

LOGS

```
/app # cat logs.txt
INFO: 2022/10/14 18:49:27 consumer.go:42: CONSUMED SEND_EMAIL EVENT
INFO: 2022/10/14 18:49:29 events.go:47: SUBMISSION RESULT EMAILED ! | ID = <20221014184931.fc017cc6e0361c1c@sandbox6e54a0e6b369403ead8b87f35ca40ce1.mailgun.org>
INFO: 2022/10/14 18:49:29 consumer.go:42: CONSUMED ADD_DATA_TO_SHEET EVENT
INFO: 2022/10/14 18:49:30 events.go:79: ADDED DATA TO ROW Sheet1!A8:C8
INFO: 2022/10/14 18:49:45 consumer.go:42: CONSUMED ADD_DATA_TO_SHEET EVENT
INFO: 2022/10/14 18:49:46 events.go:79: ADDED DATA TO ROW Sheet1!A9:C9
INFO: 2022/10/14 18:49:46 consumer.go:42: CONSUMED SEND_EMAIL EVENT
INFO: 2022/10/14 18:49:46 events.go:47: SUBMISSION RESULT EMAILED ! | ID = <20221014184948.705a03c299b5eae1@sandbox6e54a0e6b369403ead8b87f35ca40ce1.mailgun.org>
```

```
PS D:\projects\go\atlan-backend> cd consumer-api-1
PS D:\projects\go\atlan-backend\consumer-api-1> cd ../
PS D:\projects\go\atlan-backend> docker exec -it 9640eabc0c6b /bin/sh
/app # ls
Dockerfile  consumer    consumer.go  events       final.json  go.mod      go.sum      logger      logs.txt
/app # cat logs.txt
ERROR: 2022/10/14 19:25:38 consumer.go:40: FAILED TO INITIATE KAFKA CONSUMER
INFO: 2022/10/14 19:25:38 consumer.go:42: CONSUMED  EVENT
INFO: 2022/10/14 19:29:13 consumer.go:42: CONSUMED ADD_DATA_TO_SHEET EVENT
INFO: 2022/10/14 19:29:15 events.go:79: ADDED DATA TO ROW Sheet1!A10:C10
/app # exit
PS D:\projects\go\atlan-backend> docker exec -it ab03e910dc4f /bin/sh
/app # cat logs.txt
INFO: 2022/10/14 19:29:13 consumer.go:42: CONSUMED SEND_EMAIL EVENT
INFO: 2022/10/14 19:29:15 events.go:47: SUBMISSION RESULT EMAILED ! | ID = <20221014192918.77650a44f9ff098d@sandbox6e54a0e6b369403ead8b87f35ca40ce1.mailgun.org>
/app #
```

MONGODB ATLAS



I think I have covered all the use cases and tasks that were required for the assignment .
I would really grateful to have an opportunity to meet and interact with the team and discuss regarding the upcoming need and requirements for the internship

Thank you