Atlan Backend Challenge Solution

Techstack used:

- Springboot
- MongoDB
- Junit , Mockito (for unit testing)

Services used:

- Whatsmate API (for slang words)
- Twilio SMS service (for SMS generation)
- Google Sheets API (for Google Sheet Generation)

Prerequisites

- Java 17 SDK
- MongoDB installed locally
- Account on Whatsmate API - https://whatsmate.github.io/2016-08-18-translate-text-java/
- Account on Twilio - https://www.twilio.com/en-us
- Google Sheets API enabled (Do it from Google Cloud)

About the Techstack

Why Springboot?

- Springboot allows free and version conflict dependency management through the starter POM.
- It is easy to set up in very less time and run standalone web applications and microservices smoothly. So, it reduces the development time by which we can focus and invest more time on our business logics.
- We can use annotations instead of using XML configurations.
- It has an in-memory database and Embedded Server (Tomcat) which reduces a lot of boilerplate code which saves time.
- It has a large community of helpful users so we can find many discussions and tutorials online.

Why Mongo DB?

- Mongo DB supports flexible document schemas, which allows us to have objects in one collection with different sets of fields.
- It stores and represents data in the form of a document so we can access it from any language in their corresponding data structures like dictionaries in Python, Maps in Java and Objects in Javascript, etc.
- It has a special feature called Mongo DB Query API which allows us to query deep into the documents. In very few lines we can perform complex analytics pipelines
- It is easy to install and we get a good UI along with it called Mongo DB compass.
- Mongo DB provides another approach of storing data in cloud by it Mongo DB Atlas product. We will only pay as much as we use. We can also adjust our cluster ti automatically scale when needed.

Project Architecture

Approach - We will use Microservices Architecture in this challenge. Since we know that more actions may be added in future in this form project, we should go for a **plug and play** approach. We will make 4 individual microservices according to the tasks and make them loosely coupled with our form project. We can also use these microservices individually by calling their corresponding APIs.

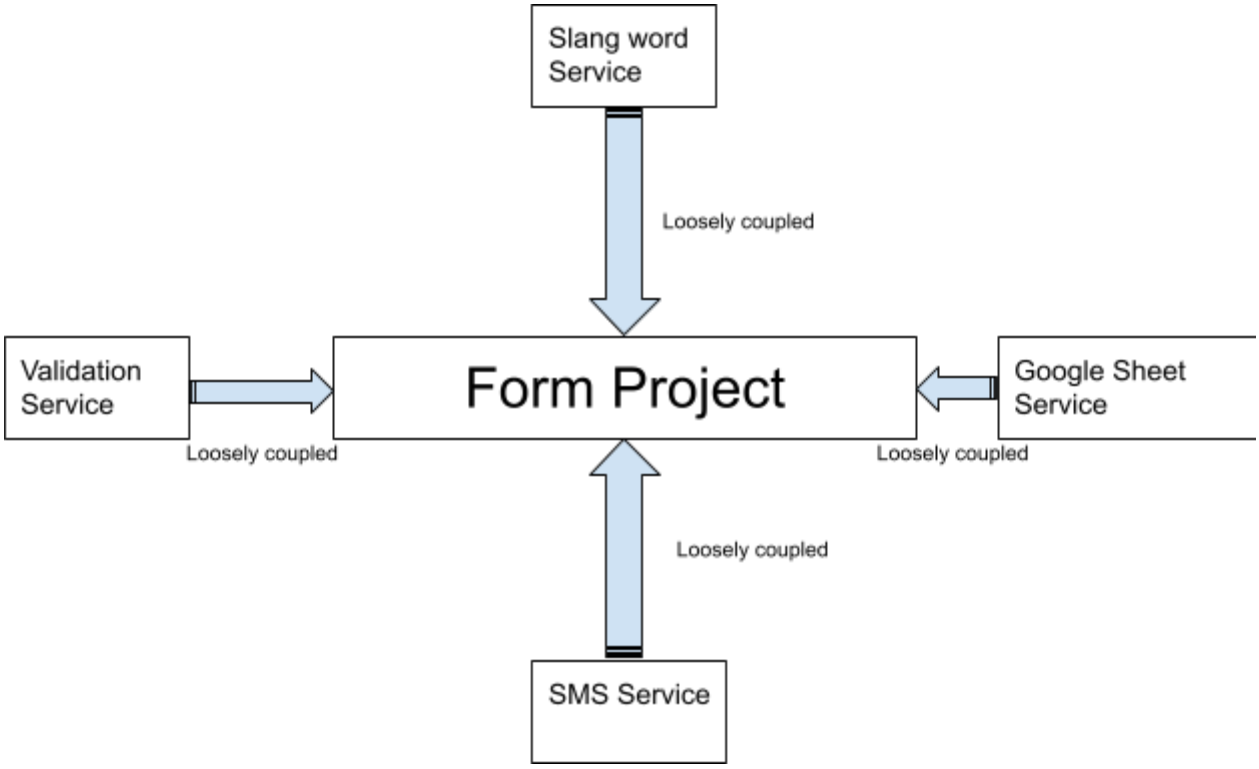
Pros:

- We can easily scale up the form project.
- Improved fault tolerance
- We can easily understand the codebase
- We can independently deploy each microservice.

Cons:

- Requires more resources
- Global Testing of the project is more complex.

Let’s understand this approach with the help of a block diagram:



Task 1 Solution - Slang Word Problem

Thought:

We know that slang words are the words which we use in a very casual way in our daily lives. We don't use them for writing purposes because they seem unprofessional. So we can't just translate the input word into the desired language. So, there can be two approaches for solvig this problem.

Approach 1:

Assumption : The input word must be in english
Input request format : **{ "toLang": "lang", "text": "text"}**

We can use a database where the entries are in this manner:

toLang	**mappings**

“hi”	*List of mappings*
“gu”	*List of mappings*

Data Structure - **List<String, List<Map<String, String>>>** will be stored in our database and the searching will happen according to the language code whenever the request is arrived at in the form of JSON format (request format is described above).

Examples of language code : “hi” for hindi, “ta” for tamil, “gu for gujarat”, etc

Let’s see the structure of the mapping of words inside the DB.

Example : Hindi words

Input Text	**Output Text**
“Nice”	“झकास”
“No”	“ना”
“Yes”	“हम्म”

Pros:

- We can map the words by our choice, we can change the mapping any time.
- No dependency on third party APIs.
- Once a mess and costly but for a long time and for business purposes it is efficient.
- We can provide the service to the customers also which will be a good source of earning also.

Cons:

- Searching complexity will be a big issue because there will be a lot of languages and words which increase the latency.
- We can use good and efficient algorithms and sorting techniques but still the searching complexity will be a challenge.

Approach 2:

Instead of creating a local Database we can use any third party API for getting slang words.
In this project, we used whatsmate API service which is a translator API. Since slang word APIs were not free so we use translator API just for Demo purpose.

We provide HTTP request to the end point in such JSON format:

{“fromLang”: “en”, “toLang”: “hi”, “text”: “what”}

The output will be **“क्या”**

Language codes for different languages -
http://api.whatsmate.net/v1/translation/supported-codes

Pros:

- We can use this for small level projects without the headache of creating such a large local database.
- Easy to plugin and use.

Cons:

- Can’t be used for large projects because latency may increase when lots of requests hit the end point.
- We have to pay a huge amount to the third party.

Task 2 - Validation Problem

Thought :

We will create an API on which customer details are sent in the form of JSON. We will send customer id, Email, Name, Monthly Income, Monthly Savings and Mobile Number in the request and in response we will get a JSON which will describe the validity of the input fields.

Approach:

We will send request in JSON in this format:

{“id”: “ID”, “email”: “Email@email.com”, “name”: “Name”, “monthlyIncome”: xxxx, “monthlySavings”: xxxx, “mobileNumber”: “xxxxxxxxxx”}

Logic:

- Email should match the email format, like it should have “@ and .com/.in etc at the end”.
- Name should not contain any digit in it.
- Monthly Income should always be greater than or equals to monthly savings.
- Mobile numbers should be of 10 digits and must start with 6-9.

Response:

In response we will get a JSON like this:

{“nameValidity”: true, “emailValidity”: true, “incomeValidity”: true, “mobileNumberValidity”: true}

According to these boolean values we can display the desired message in the frontend.

Task 3 - Google Sheets Problem

Thought:

Every organization needs a feature by which all the latest data of users can be accessed anytime by a single click in the google sheets which can be further used in generating insights. We can use google sheets API to create a sheet and store the data in it on a single click.

Approach 1:

We can use google app script to store the customer data directly in the google sheets. It means google sheets will act as a database in this case.

The data will be stored in the form of rows and columns and when a new customer is added a new row will be created in the sheet.

We can share the sheet for generating further insights.

Pros:

- We can link the sheet with various insights generating tools like google data studio and also perform various ML operations easily for generating insights.
- Live insights will be generated as the sheet data will be dynamic.
- Can be used for small levels.

Cons:

- Customer data may be lost when the corresponding google account will be down or facing issues since we don’t have any local backup for storing the data.
- Cannot be used on a large scale when we cannot afford the data loss.

Approach 2:

We will use this approach in our project. We will create an API, on hitting it all the data of all users will be fetched from the Database and will be stored in the google sheet. A google sheet

will be created instantly and its name will be the Date and Time when the request is hitted. In the response will get the **sheet id and sheet url**. This sheet can be further shared for generating the insights.

We have to make a project in Google Cloud Platform and enable the Google Sheets API and change some configuration settings.

Refer to this video for setting up the configurations - https://www.youtube.com/watch?v=Z808S_eSKmI&list=PL2IQ9VnvNu0XF6DrZzsTfu52dHzQNIzRG

Pros:

- We are fetching data from a local database so data cannot be lost even though the Google Sheets API is not working.

Cons:

- We cannot generate live insights from the google sheet. We have to generate another pipeline from the local DB for generating the live insights.

Task 4 - SMS problem

Thought:

We can use various third party services like Fast-SMS or Twilio SMS service for sending the message.

Approach:

We will use Twilio SMS service in this project. We have to create an account on twilio and generate our unique number. We have to verify the recipient numbers and add in the list.

We will create an API in which we will send the recipient number and the message body in the form of JSON like this:
{“toMobileNumber”: “xxxxxxxxxx”, “messageBody”: “text}

In the logic layer the twilio sms api is called and the message will be sent to the recipient number.

Project flow

In this project we have provided APIs for individual services and also applied these services in a form project.

APIs:

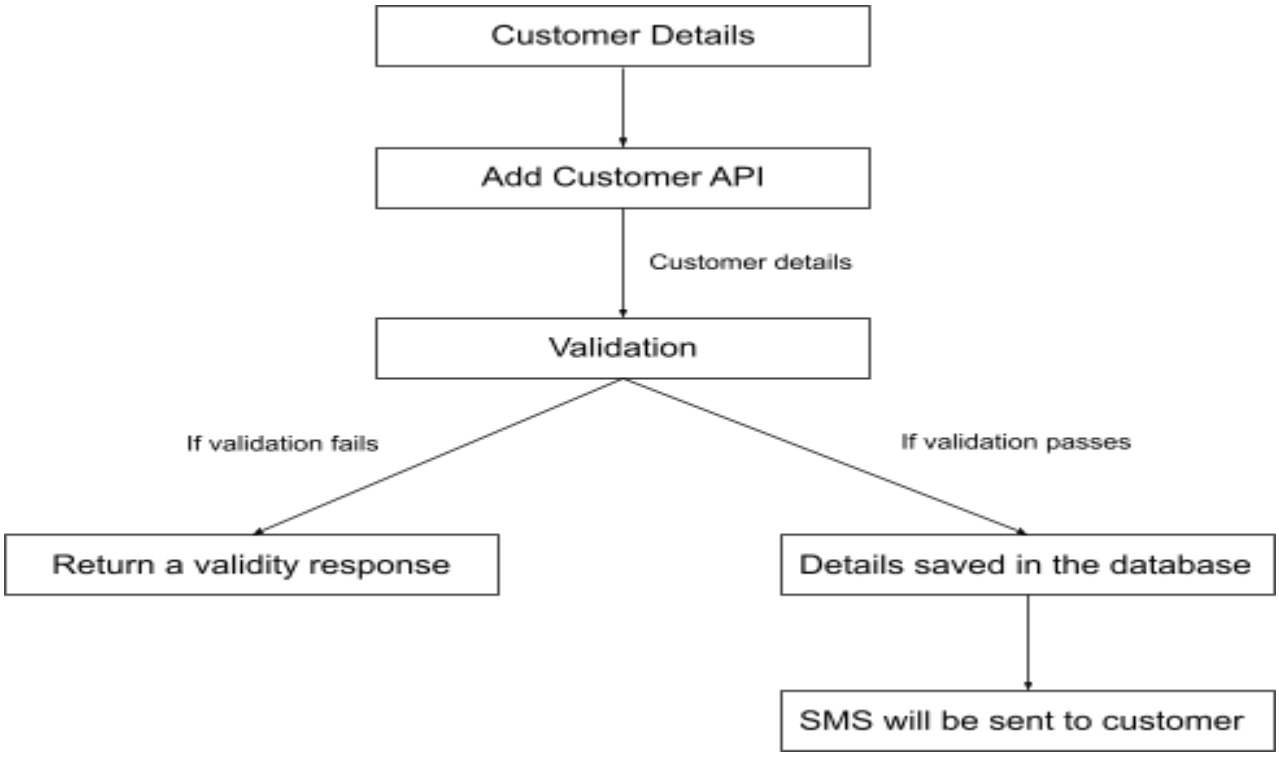
- **For getting all customers - http://localhost:8080/getAllCustomers**
  Type: GET
- **For adding new customer - http://localhost:8080/addCustomer**
  Type: POST, Input JSON: {
  "email": "tushartg600@gmail.com ",
  "name": "Tushar Gupta",
  "monthlyIncome": 1000,
  "monthlySavings": 700,
  "mobileNumber": "8005791568"
  }
- **For creating Google Sheet - http://localhost:8080/createSheet**
  Type: GET
- **For getting the slang word - http://localhost:8080/giveSlang**
  Type: POST, Input JSON: {
  "fromLang": "en",
  "toLang": "hi",
  "text": "What"
  }
- **For sending SMS - http://localhost:8080/sendSMS**
  Type: POST, Input JSON: {
  "mobileNumber": "8005791568",
  "messgeBody": "Message Body"
  }
- **For validating the customer details - http://localhost:8080/validateCustomer**
  Type: POST, Input JSON: {
  "id": “21122121”,
  "email": "tushartg600@email.com",
  "name": "Tushar Gupta",
  "monthlyIncome": 2000,
  "monthlySavings": 1500,
  "mobileNumber": "8005791568"
  }

In this project, We will send customer details from a form/frontend to our backend. The input will go under a validation process in which the fields are validated, if the validation is successful, the details will be saved in the database and a SMS will be sent to the customer's mobile number in which the details will be present. If the validation is unsuccessful, a validation response will be sent to the user in which the details about the validity of the entered fields will be mentioned.

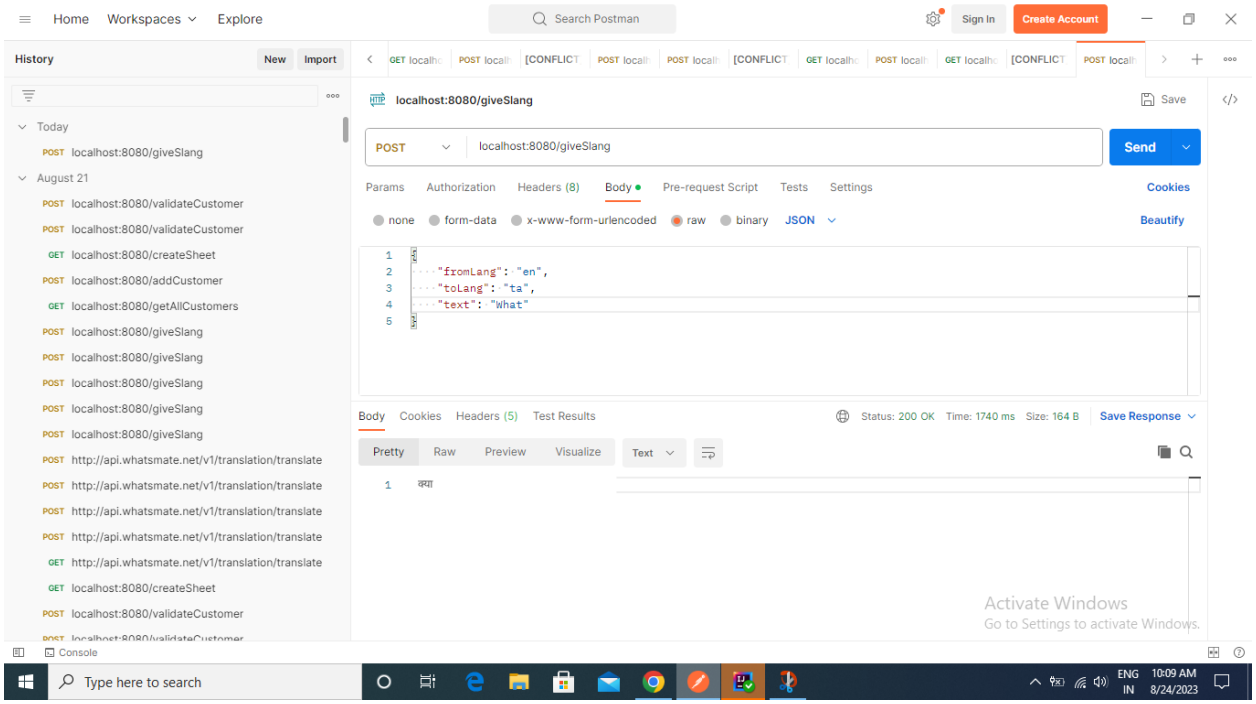
The validation response will be like -> **{"nameValidity": true, "emailValidity": true, "incomeValidity": true, "mobileNumberValidity": true}**

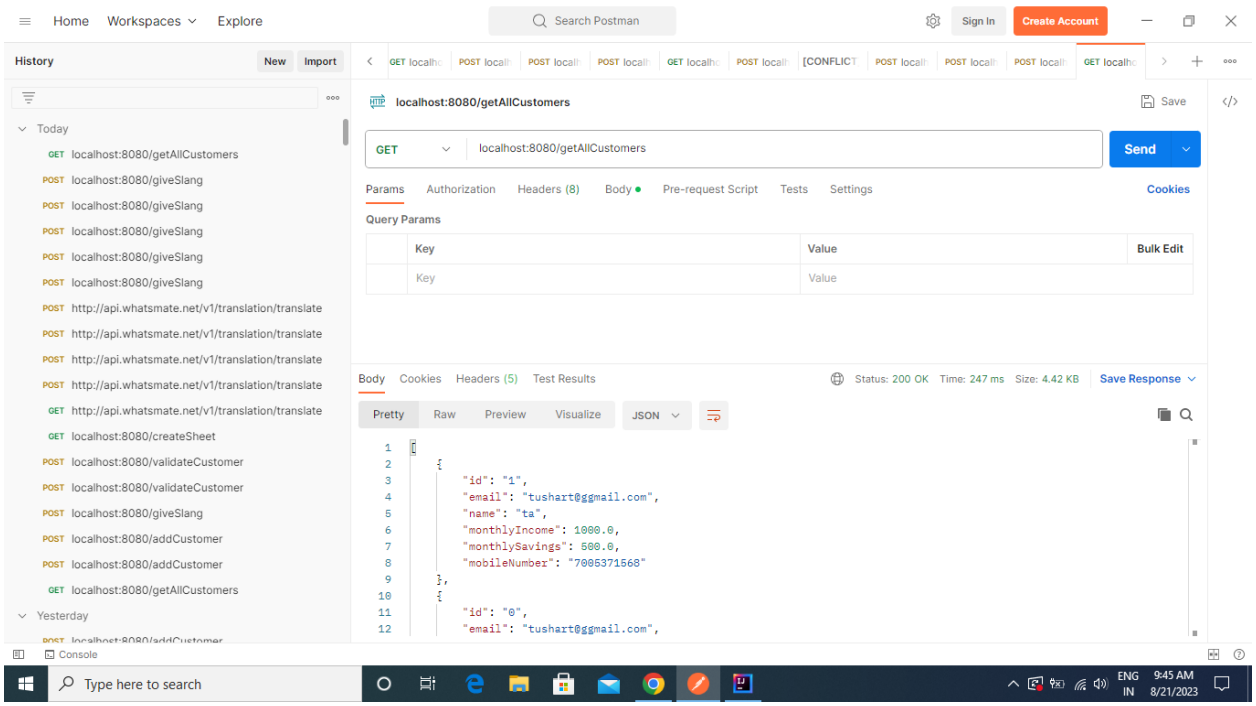
A separate API will be there for getting the slang words and create the google sheet in which the details of all the customers will be present.

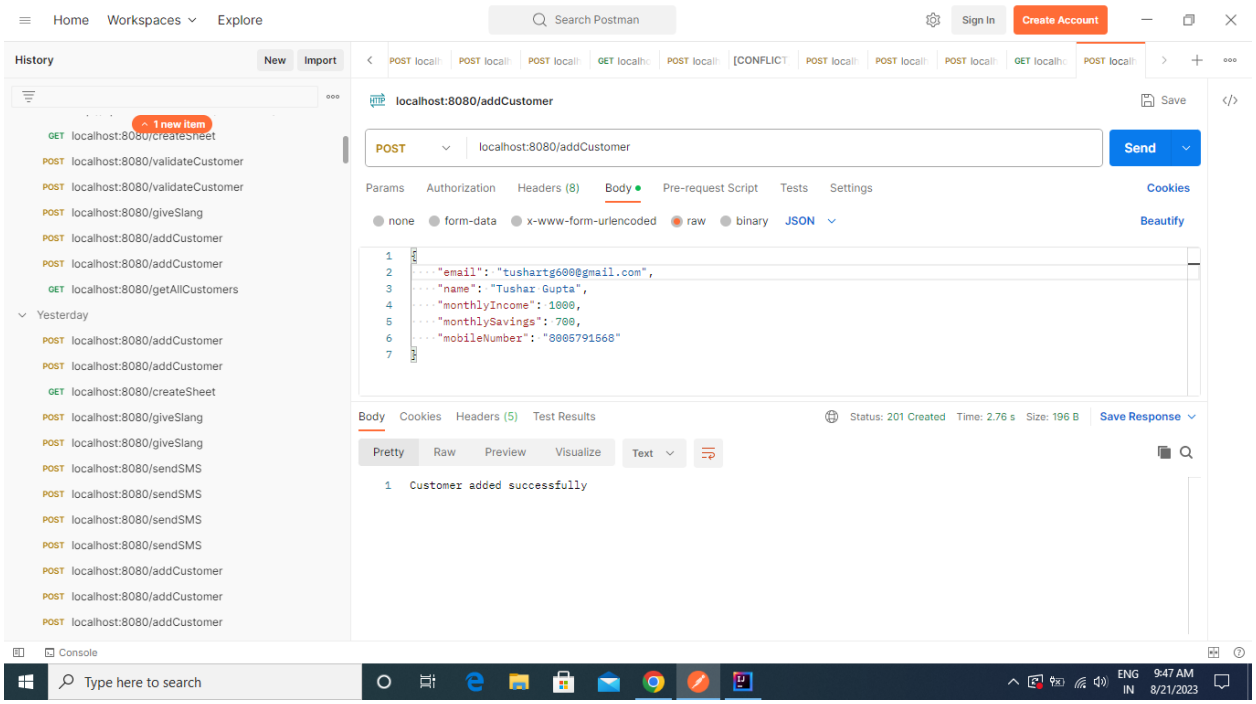
Flow Chart:

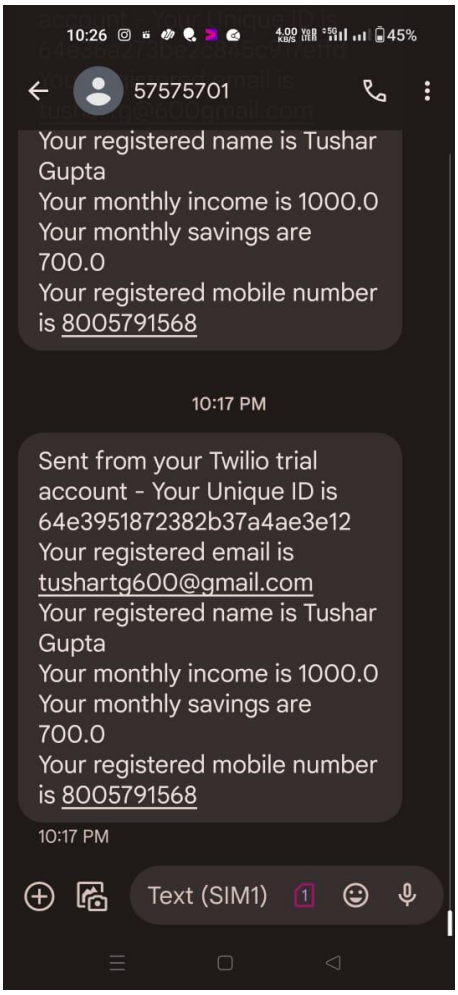


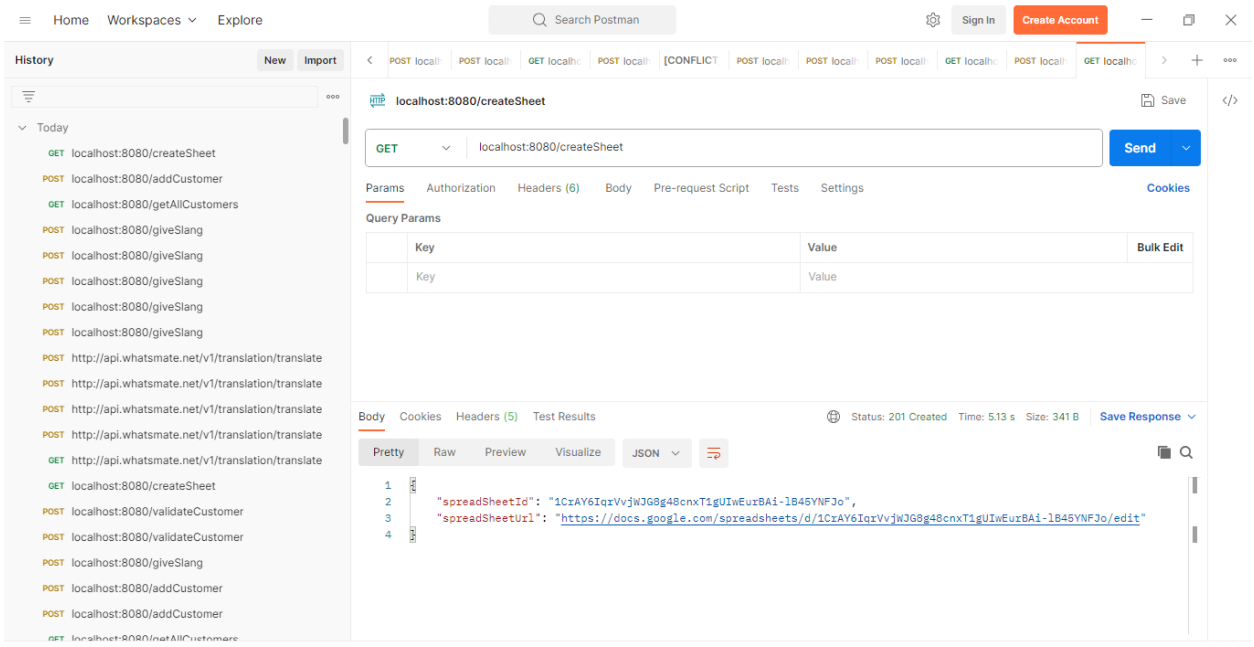
Screenshots

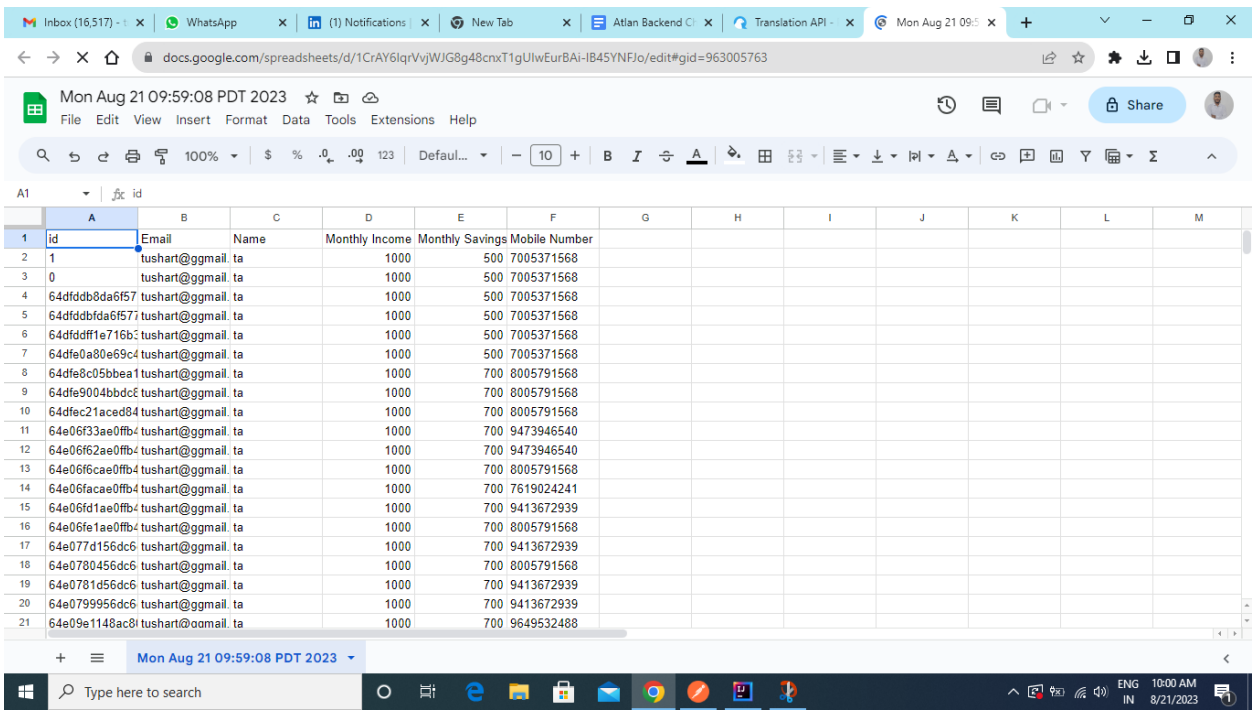




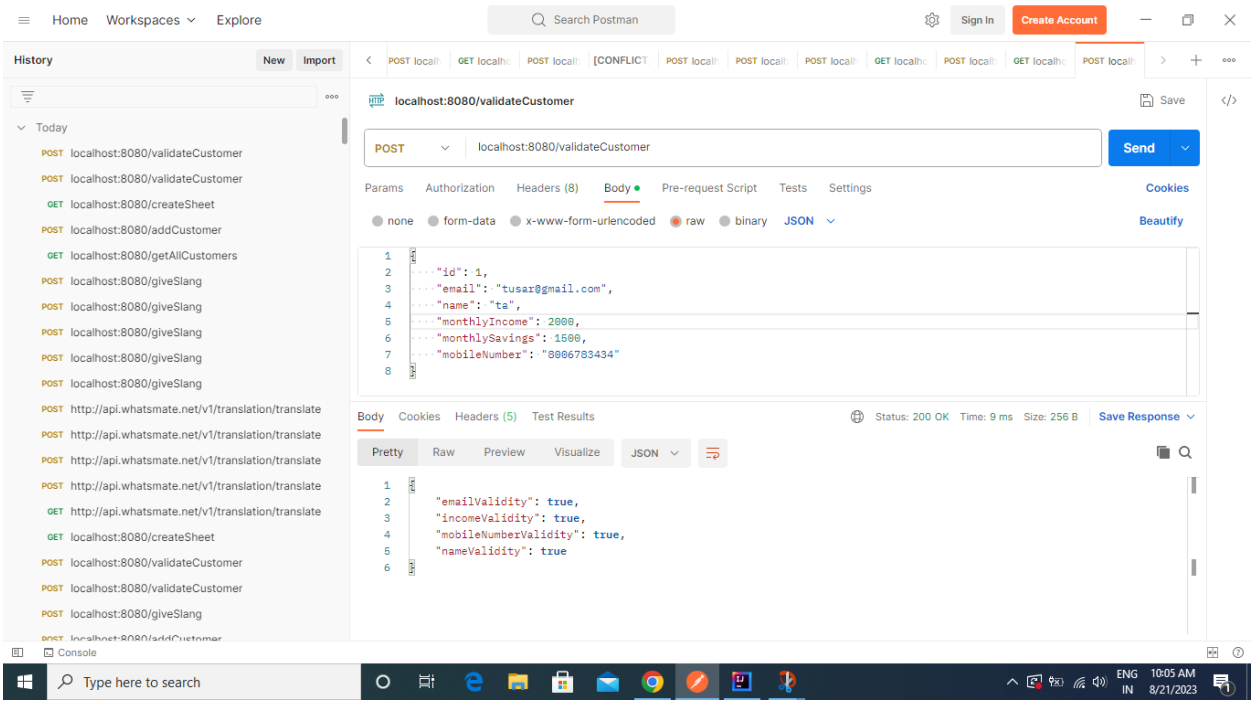

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
History
New
Import
1 new item
GET localhost:8080/createSheet
POST localhost:8080/validateCustomer
POST localhost:8080/validateCustomer
POST localhost:8080/giveSlang
POST localhost:8080/addCustomer
POST localhost:8080/addCustomer
GET localhost:8080/getAllCustomers
Yesterday
POST localhost:8080/addCustomer
POST localhost:8080/addCustomer
GET localhost:8080/createSheet
POST localhost:8080/giveSlang
POST localhost:8080/giveSlang
POST localhost:8080/sendSMS
POST localhost:8080/sendSMS
POST localhost:8080/sendSMS
POST localhost:8080/sendSMS
POST localhost:8080/addCustomer
POST localhost:8080/addCustomer
POST localhost:8080/addCustomer
localhost:8080/addCustomer
Save
POST
localhost:8080/addCustomer
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
JSON
Beautify
{
"email": "tushartg600@gmail.com",
"name": "Tushar Gupta",
"monthlyIncome": 1000,
"monthlySavings": 700,
"mobileNumber": "8005791568"
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 2.76 s
Size: 196 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Customer added successfully
Console
Type here to search
ENG
IN
9:47 AM
8/21/2023


10:26
45%
57575701
Your registered name is Tushar Gupta
Your monthly income is 1000.0
Your monthly savings are 700.0
Your registered mobile number is 8005791568
10:17 PM
Sent from your Twilio trial account - Your Unique ID is 64e3951872382b37a4ae3e12
Your registered email is tushartg600@gmail.com
Your registered name is Tushar Gupta
Your monthly income is 1000.0
Your monthly savings are 700.0
Your registered mobile number is 8005791568
10:17 PM
Text (SIM1)





	A	B	C	D	E	F
1	id	Email	Name	Monthly Income	Monthly Savings	Mobile Number
2	1	tushart@ggmail.	ta	1000	500	7005371568
3	0	tushart@ggmail.	ta	1000	500	7005371568
4	64dfddb8da6f57	tushart@ggmail.	ta	1000	500	7005371568
5	64dfddbfda6f57	tushart@ggmail.	ta	1000	500	7005371568
6	64dfddff1e716b	tushart@ggmail.	ta	1000	500	7005371568
7	64dfe0a80e69c4	tushart@ggmail.	ta	1000	500	7005371568
8	64dfe8c05bbea1	tushart@ggmail.	ta	1000	700	8005791568
9	64dfe9004bbdc8	tushart@ggmail.	ta	1000	700	8005791568
10	64dfec21aced84	tushart@ggmail.	ta	1000	700	8005791568
11	64e06f33ae0ffb4	tushart@ggmail.	ta	1000	700	9473946540
12	64e06f62ae0ffb4	tushart@ggmail.	ta	1000	700	9473946540
13	64e06f6cae0ffb4	tushart@ggmail.	ta	1000	700	8005791568
14	64e06facae0ffb4	tushart@ggmail.	ta	1000	700	7619024241
15	64e06fd1ae0ffb4	tushart@ggmail.	ta	1000	700	9413672939
16	64e06fe1ae0ffb4	tushart@ggmail.	ta	1000	700	8005791568
17	64e077d156dc6	tushart@ggmail.	ta	1000	700	9413672939
18	64e0780456dc6	tushart@ggmail.	ta	1000	700	8005791568
19	64e0781d56dc6	tushart@ggmail.	ta	1000	700	9413672939
20	64e0799956dc6	tushart@ggmail.	ta	1000	700	9413672939
21	64e09e1148ac8	tushart@ggmail.	ta	1000	700	9649532488



Unit Testing Part

Unit testing of a project is as important as the working of the project. We can't say that the project is working perfectly without doing the unit testing.

The unit testing of this project is done with JUnit framework. We used mocktio for mocking and tested all the functions present in the project.

Important Points:

- The code coverage and the conditional coverage is greater than **80%** which is a standard mark considered in backend development.
- We didn't test the functions of Slang Word, SMS and Google Sheets service because they are trusted third party services and we assume that they are already tested.

Challenges

How to make the system failsafe?

For making the system failsafe we have to focus majorly on three areas:

Scalability, Availability and Security

- For Scalability and Availability we can do multiple deployments of the same server by keeping in mind that the state of the web app is maintained. It will improve the server performance, reduce dependability, improve database functions and also will be cost effective.
- For Security, we should follow some important measures, for example:
    - We should backup the data regularly and keep it secure.
    - Firewall should be used.
    - Unused ports and services should be closed immediately.
    - Run a virus checker for monitoring inbound and outbound traffic.

We can make replicas of the primary database and set up them at different sites. There are many advantages of data replication:

- It makes the application more reliable because if there is hardware or machinery failure, the data will still be accessible from another location.
- It will help to access the data faster because the data will be accessed from the nearest replica and reduce the latency. It also reduces the load on a single replica.
- We can get advanced analytics by synchronization and consolidation of various sources into data warehouses. By immediately copying data into other locations, we can get better insights and decision making.

Project Link

https://github.com/tushartg12/Atlan_Backend_Challenge

Important Notes

- **Since we are using some third party services, please create accounts on the respective platforms and use your credentials.**
- **In Twilio SMS service please add the caller ID and verify the number on which you want to send the SMS.**
- **I have created a package named as “Credentials”, inside it paste Whatsmate API credentials (CLIENT_ID and CLIENT_SECRET) in “SlangServiceCredentials” file and Twilio credentials (ACCOUNT_SID and AUTH_TOKEN) in “SmsServiceCredentials” file.**
- **Paste the credentials of the Google Sheets API (generated in JSON format) in the “credentials.json” file.**

- **It is strongly suggested to replace your credentials with the given ones but for your convenience i have provided my credentials, so you can test this project. Use the above JSON inputs for happy path testing(given in APIs section).**