ACTION PLAN BY TEAM 33

HASURA PRODUCT DEVELOPMENT FELLOWSHIP

Members:

T33PF1	Shivam shivamd20@gmail.com	ReactJS
T33PF1	Swetha Gunta svgunta@gmail.com	React-Native
T33PF1	Joel Kingsley joelkingsley.r@gmail.com	Python-Flask
T33PF1	Uddhav Wani uddhavwani9@gmail.com	Python-Flask
T33NE1	Abhinav Anand abhinavanand.rockstar@gmail.com	NodeJS-Express
T33NE1	Swathi Ainala swathi.ainala@gmail.com	ReactJS

Objective:

To create an application that allows users to add data to a table, triggers a zap (from Webhook to Google Sheets) that adds the name of the table to Google Sheets as a new row. The user will be notified that the data is saved, and the zap is triggered.

Architecture:

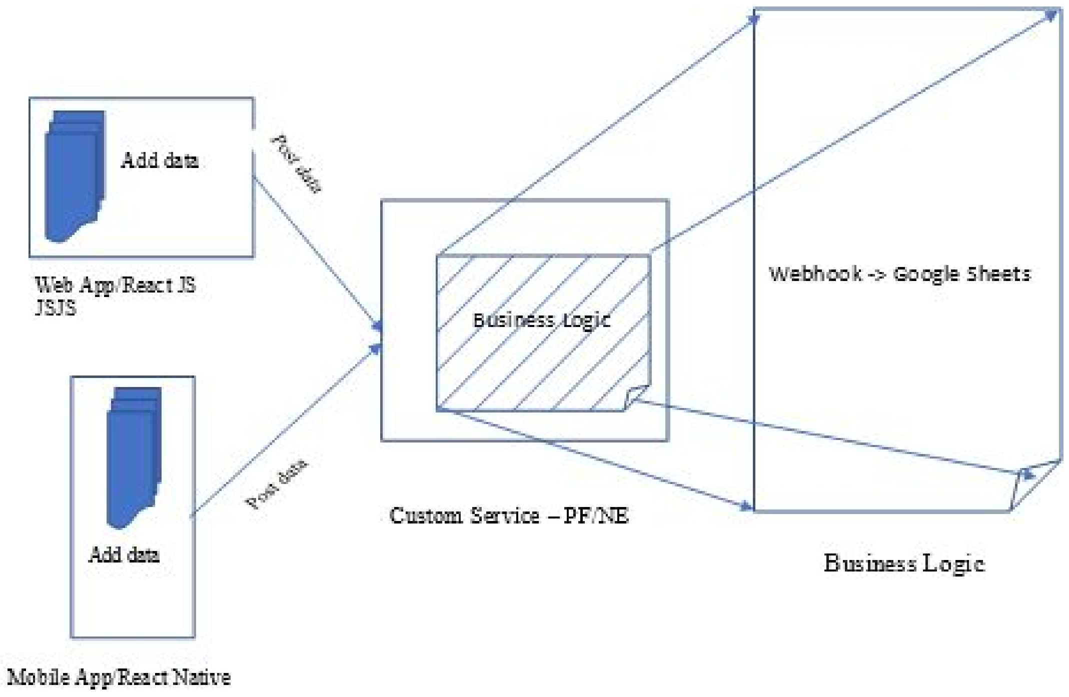


Figure 1: Architecture of the App

The project consists of a Front End and a Back End. The user creates a new table or selects an existing table and populates the table with data. On the click of the button named "Save Data" button the front end posts a request, that carries the data, via an URL, a custom microservice. This custom microservice then triggers a zap, which is activated from WebHook app to Google Sheets app. The Table name, a JSON String is now carried via a POST request to the WebHook app. This WebHook app copies the table name to Google Sheets as a new row. Once a trigger is made by the backend the user is notified that the data is saved, and the zap triggered.

Basic Flow Diagram:

Python Flask/NodeJS-Express:

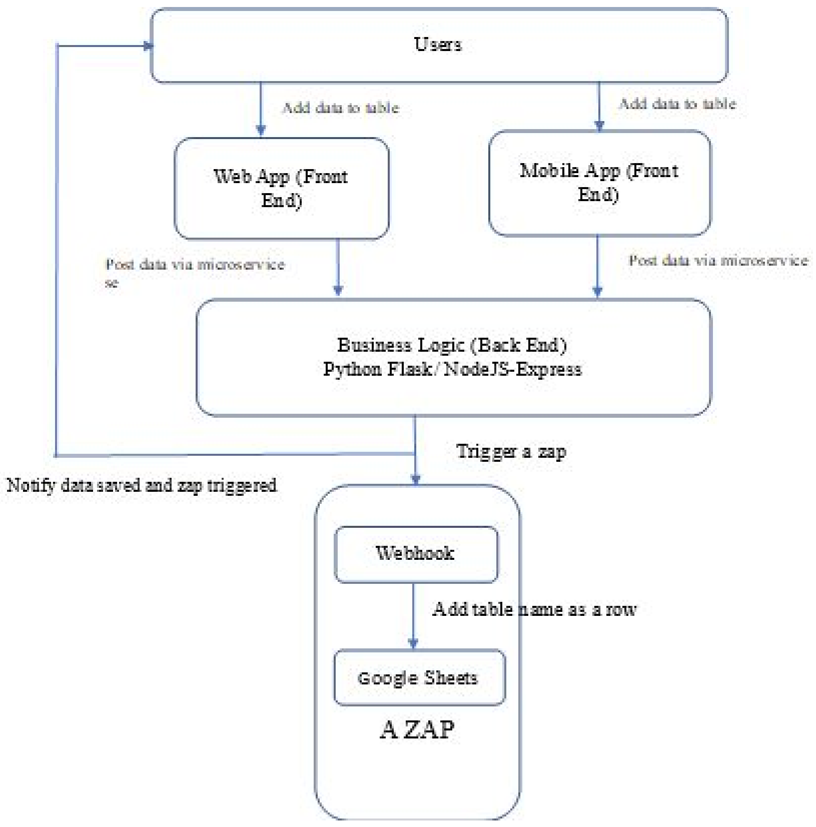

Users
Add data to table
Add data to table
Web App (Front End)
Mobile App (Front End)
Post data via microservice se
Post data via microservice
Business Logic (Back End) Python Flask/ NodeJS-Express
Trigger a zap
Notify data saved and zap triggered
Webhook
Add table name as a row
Google Sheets
A ZAP

Figure 2: Basic flow diagram with Python Flask/NodeJS-Express as Back End

Each team consists of one Back-End developer and two Front-End developers, one for Web Application and the other for Mobile Application.

Back-End:

1. Python-Flask:

The high-level idea in the backend is to host end points for both the web as well as the mobile app, through which registered users can create and manage tables in their accounts. At the same time each user will get a shared link to a Google Sheet in their username, which automatically gets updated with the latest list of tables in their account.

Initially, we are going to start off by implementing the basic CRUD operations for the tables. When the user creates a new table, and clicks on the create button, a JSON POST request containing the table's details is made to the server, which would store the details in a database.

At the same time, a trigger in the form of a JSON POST request is made by the server to the Zapier Webhook, which takes the request and makes changes in the Google sheet alloted for the user from whom the request is made.

Simple Trigger in Python:

```
#Python 3 Script to send a POST request containing JSON

import json
import requests

# Set the webhook_url to the one provided by Zapier when you create it
webhook_url = 'https://hooks.zapier.com/hooks/catch/XXXXX/YYYYY/'

zapier_data = {'key': "value"}

response = requests.post(
webhook_url, data=json.dumps(zapier_data),
headers={'Content-Type': 'application/json'}
)
print(response)
```

Subtasks:

1. Create endpoints for CRUD operations.
2. Create database.
3. Trigger Zap when changes are made in a table.

Dependencies:

- PostgreSQL - PostgreSQL is a powerful, open source object-relational database system. Here, it will be storing the tables of all the users.
- Zapier - Zapier is an online tool that is used to connect two or more apps to automate workflows. We'll be connecting the Zapier webhook and Google Sheets to get a spreadsheet of table names for each user.
- Hasura Authentication (in later stages) - Hasura Auth APIs let you create, authenticate and manage user accounts on your Hasura project. It also lets you manage sessions and roles for users. It will be used to manage users for the app.

Resources:

- https://docs.hasura.io/0.15/manual/data/create-tables.html
- https://zapier.com/help/zap-creation/

2. NodeJS-Express:

Simple Trigger in Node:

```
POST data.<cluster-name>.hasura-app.io/v1/query HTTP/1.1

Authorization: Bearer <admin-token>

Content-Type: application/json

{

"type" : "run_sql",

"args" :

   {

  "sql" : "CREATE TABLE Student (

  id SERIAL NOT NULL PRIMARY KEY,

  title TEXT NOT NULL,

  description TEXT );"

   }

}
```

Generate API code and make it available it in server.js file as a post request

Subtasks:

1. Create a custom service that adds data to a table and triggers a Zapier Zap's webhook that adds the name of the table to a Google Spreadsheet as a new row.
2. Create the table using Hasura APIs.

Documentation:

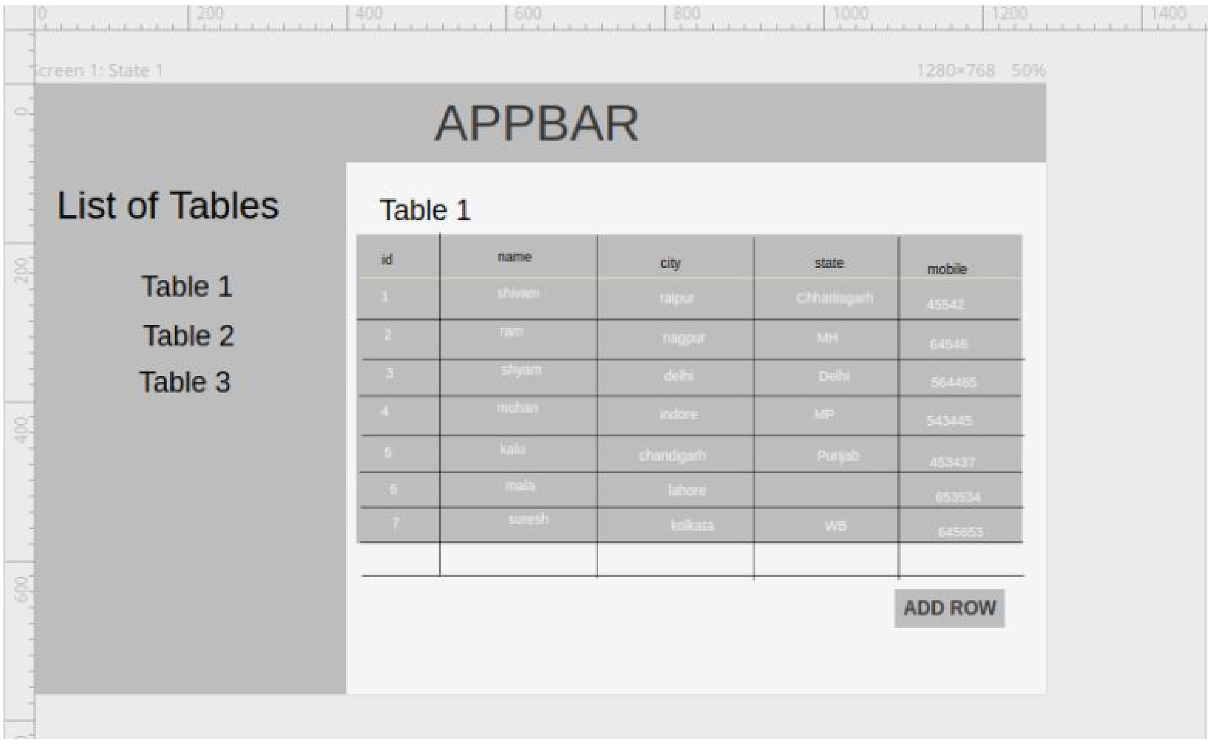
- Hasura API console

Resources:

- https://docs.hasura.io/0.15/manual/data/create-tables.html
- https://zapier.com/help/zap-creation/

Front End:

1. React JS



id	name	city	state	mobile
1	shivam	raipur	Chhattisgarh	45542
2	ram	nagpur	MH	64546
3	shyam	delhi	Delhi	564465
4	mohan	indore	MP	543445
5	kalu	chandigarh	Punjab	453437
6	mala	lahore		653534
7	suresh	kolkata	WB	645653

Figure 3: Wire-frame for ReactJS

2. React Native

React Native is a JavaScript framework that uses the same design as React to build the user interface for native mobile devices. In other words, it uses the same building blocks, declarative components, as that of iOS and Android Apps. It is used not to build a mobile web app but a mobile app that is indistinguishable from the app built using Objective-C or Java.

Setting up the environment:
1. Install Node js and npm
2. Install React Native
3. Install Visual Studio for Android
4. Create app
5. Install NativeBase, React Navigation and link peer dependencies
6. Running the app

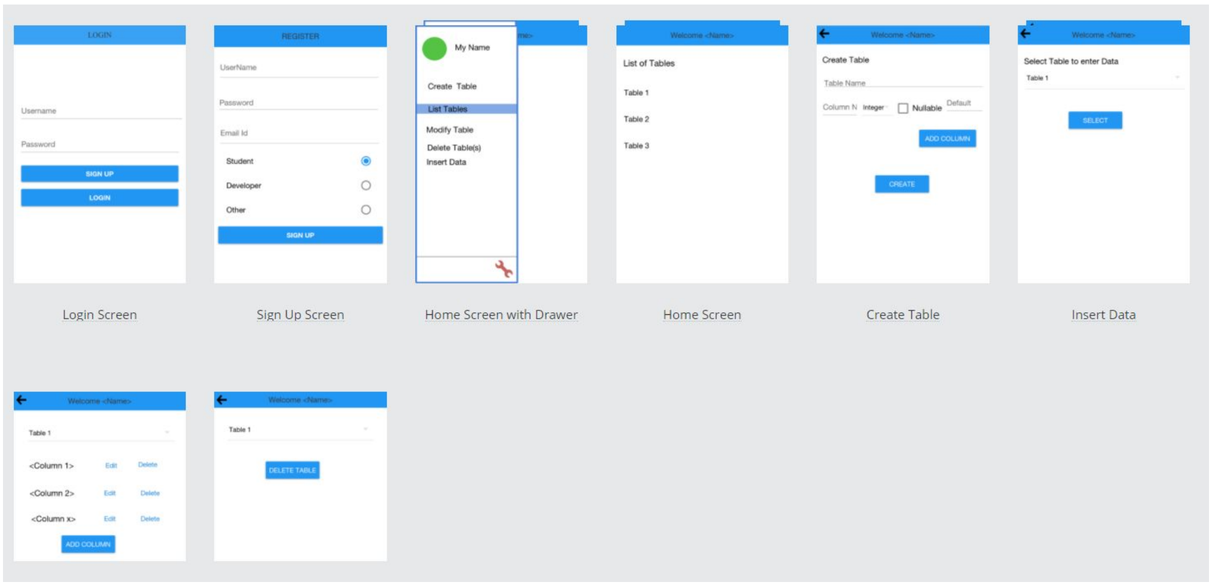


Figure 4: Wireframe for the Mobile App

The Screen Transitions are as follows:

- *Login Screen*: The user logs in using the credentials. If the user isn't registered before he/she registers himself after clicking the "SignUp" button
- *Sign Up Screen:* The user if isn't registered registers himself in this screen. As soon as the user registers himself he is taken back to the Login Screen
- *Home Screen with Drawer:* As soon as the user logs in with the right credentials he/she is navigated to the Home screen which has different permissions. It displays a List of Tables that can be edited and modified. It also contains a drawer through which one can navigate to different screens.
- *Create Table:* In this page the user enters the name a Table. The table name is accepted if the table doesn't exist already. The user is also asked to enter the column names and their types.
- *Insert Data:* This page allows the user to select a table in which the user wants to insert data and then populate the data column-wise.
- *Modify Data:* This page allows the user to modify the column names and add extra columns if necessary.