IIT Madras

Centre For Innovation

Butterfly®

EE01: Automatic Cooking Machine

ELECTRONICS CLUB

Team Members

EE19B140	**Sarthak Vora**
Work did: Built an **IOT** based codebase to integrate with a cloud server and a mobile based app. Assembled flutter app on a touchscreen. **Email** : ee19b140@smail.iitm.ac.in	
CH19B029	Sundar Raam S
Work did: Product design Email : ch19b029@smail.iitm.ac.in	
EE19B027	Snehan J
Work did: Part of electronics module Email : ee19b027@smail.iitm.ac.in	
EE19B055	Santosh G
Work did: Worked on the front-end and back-end of the app development, set up the server, established an end-to-end connection from mobile to server using APIs. Collaborated with Sarthak to establish communication of the machine to WiFi Hotspot and to the server. Email : ee19b055@smail.iitm.ac.in	
EE18B006	Gayatri Ramanathan Ratnam
Work did: Worked on the front end module of the machine: LCD + Keypad model, resistive screen model and finally helped Sarthak integrate the RPi touchscreen. Moreover, ideated on the 5 layer structure of the new machine. Email: ee18b006@smail.iitm.ac.in	

EE19B023	Gagan Deep G
<u>Work did:</u> App development on flutter Email: ee19b023@smail.iitm.ac.in	
EE18B044	Cibi
<u>Work did:</u> Built fusion models of the automatic cutters and mixers for the machine in the first phase of the project which was later taken over by Sundar Raam. Email: ee18b044@smail.iitm.ac.in	
<u>EE17B069</u>	Sundar Raman P
<u>Work did:</u> CFI point of contact with Butterfly team and managed the project Email: ee17b069@smail.iitm.ac.in, sundarramanp2000@gmail.com	

Table of Contents

Mechanical Module

Deliverables

From design point of view, the objective of the team was to make a bigger, more complex system that would be able to do most of the practices in Indian cooking. Automation of some of the things that are manually done in the current model also was a deliverable.

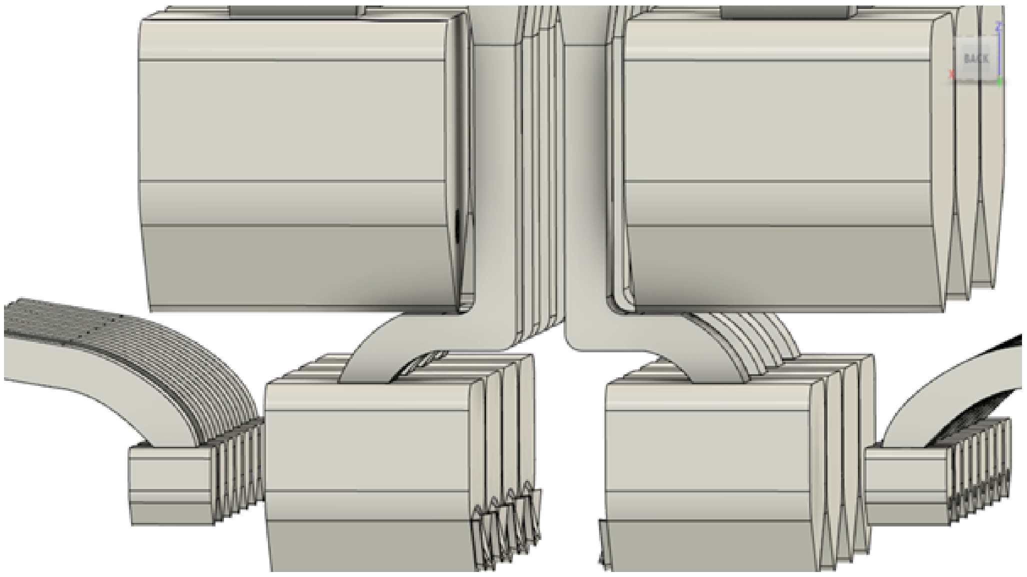
Workflow of different modules

First the team worked on deciding how much levels are to be considered in the whole machine and what each level was supposed to do. Taking some of the standard Indian dishes and analysing its cooking process, the team proposed a 5-level configuration. The detailed explanation of all levels is given below. The team designed a prototype for levels week by week. Camera aperture-like lid for level 5 was the first to be designed. Then level 4 and level 3 were designed. Then designing of level 2 took a lot of iterations. Finally the complete model was finished with level 1 and level 5.

Level 1

6 large boxes are laid out on the top row and they are top loaded. 8 medium boxes + 16 small boxes are laid on the second row. Medium boxes are top loaded while small boxes are side loaded. The exit of the ingredient contents has been put on the sides for the middle box. Typically, it is used to store vegetables. So, the outlet is large and spring operated. It is designed to release one vegetable at a time. The other boxes have small outlets and will be placed at the bottom and the actuation to release ingredients will be similar to the previous version. And have to think about how to measure weight. Loading the ingredient is made simple. It is channelised into the box. After finalising the position of outlet, we need to realign boxes so that inlet channels and other boxes don't interfere with the outlet as shown in the figure below.

Number of actuators required in this level equals the number of boxes used. So currently there are 26 actuators needed. The boxes in the level and the channels can be made in plastic, so it is a completely 3D printable level.



Level 2

Level 2 is meant for slicing and dicing. Dicing means cutting vegetables into cubes, while slicing means transverse cutting. This is supposed to be the most complex level of the machine. It has the highest number of actuators with unique functions. Also, this is the only movable level of the machine. This is the level that every ingredient has to pass through to reach level 3 or level 4. Only the ingredients that directly fall into the level 5 can skip this level, which again is not preferable, as it has to fall through more than 1 m. Motion in 2 dimensions is possible in this level. One direction is achieved by Belt drive and other is through rack and pinion. The chopper-dicer and slicer are mounted on a plastic container. It is better to 3D print this part to reduce its weight. The motors to do the slicing and dicing are to be operated with a tiny but powerful motor through gears that can be mounted on the plastic container itself. Ingredients come out of this level through spring operated doors. The motors that are used to close and open the bottom of the slicer and chopper need to be lightweight and needn't be very powerful. The motors that are used to drive the 2 containers on the rack need to be really strong but need to be light-weight. And this motor has to support the weight of the other 2 sets of motors. The total number of actuators calculated to be on this level are

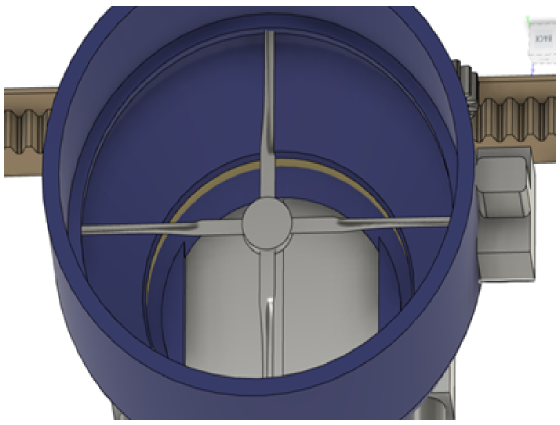
4 per container for motion along pinion – 8 (at most)

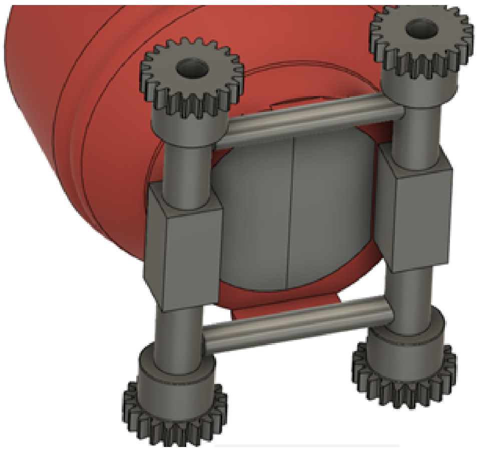
2 per container for opening the bottom – 4

1 per container for operating the dicer/slicer – 2 (at least)

Motors on the belt drive – 4

The belt drive and rack and pinion drive parts are better to be procured from the industrial robotic manufacturers. Disadvantages in this level is mainly about the weight issues. The motor driver boards or any electronic components other than the motors should be avoided on the pinion, it can be hosted on the belt rail and connected to this with slack wires. Belt drive is rigidly fixed to opposite walls of the system. The two containers just balance on the pinion with a small ridge, we can think of some alternatives to completely embrace the slicer and chopper to add stability.





Level 3

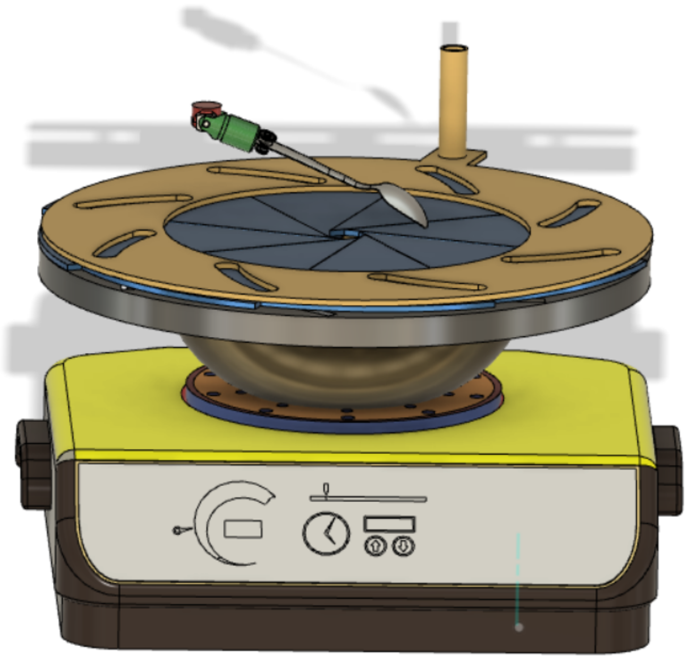
Level 3 is meant for boiling/frying before cooking. It has a base protruding from the front wall. And a pan is mounted over it. Pan has to be custom made to make all contents flow out of the vessel at the maximum angle of tilt. There also is a small pin that acts as the axle of tilt for the pan. Important considerations in this level are about heating elements. It has to be efficient but at the same time occupy minimum space possible. Motors used in this level are 2 per heater. One can be large but powerful enough to lift the metallic pan. The smaller motor that is used to sweep the water sieve has to be placed on the pan. Water sieve is placed for the purpose of draining water in case boiling is happening. So, the motor for this should be heat resistant, and wiring for this motor also is an issue to be tackled. Water coming out of the boiler and excess oil out of the fryer is removed through a small plastic pipe attached to the pan. The channel that is used to transport contents from level 3 to level 4 can be made in plastics and is very easy to 3D print. Robotic arms used can also be 3D printed. Water and oil supply for this level is very crucial. Sometimes ingredients would be waiting in this level without getting cooked because this is the only way into level 4. Sometimes the ingredients will be cooked on ghee or butter or nothing. From this level all levels further down need water and oil supply. Therefore, it was decided to mount water and oil storage above this level. So additional 2 motors have to be used. Flexible PPE pipes are planned to be used for supply of water/oil. It can be constricted to restrict flow**.**

Level 4

This level is fairly simple with the mixer planted along the side of the machine. A very important consideration to be made is how far we can reduce the weight of mixie. Clearly commercially sold mixies are way too heavy for this. In this level, there is a pipe that starts from level 3 and drops directly above level 4. And a lid which at half rotation will deflect the contents out and pushes it directly to level 5. And if it is completely open it lets the contents inside the mixie. And the lid closes while mixie is functional. The bottom lid remains open always except only if the mixie has to function. In the current design, the motor of the mixie is attached to the lid, but that is not very efficient, because lifting the motor every time is expensive. So, we're still thinking of ways to attach the motor to the fixed potion. Also the motor for the top lid needs some fixed support to operate, for which we have kept some supports extending to the other side of the mixer vessel.

Level 5

Mixing the contents while cooking and closing it are the two major concerns. While closing the vessel is executed by using camera lid type of motion, this actuation technique is very delicate. And since it is one of the few parts that the user will directly handle, this might be a serious problem. If we use the spoon stirring kind of actuation for mixing, we have to employ really small motors, because three motors with three different orientations have to be attached to the small gaps available. Also, one more issue I found with using a spoon is there is no room for placing the spoon right above the centre of the cooktop, so there is some interference with the lid and all parts of the vessels aren't accessible. The disadvantage of using blades like in the existing model is difficulty in handling, cleaning especially and it doesn't fit all types of vessels. Most of the parts in this level can be manufactured in the existing butterfly facilities, like stove, vessel, spoon, etc. Other parts like camera aperture lid are to be manufactured. Also, since the container walls move in towards the starting of level 1, there is less place for the electronic parts to be mounted and wired.



Challenges and Future Works

Future work in this module is about deciding what actuators to buy using the power/space specification. Alter the model to fit the actuator in the given power/space specification. Detailing the model by mentioning the electrical, oil and water routes. Adding stability to the structure and choosing what material is best suited for each of the modules. To know what is the best manufacturing technique, for the given geometry and material.

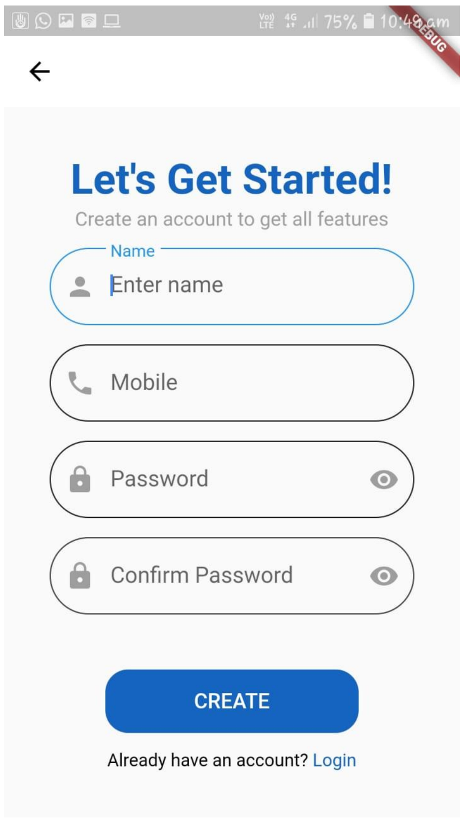
Software Module - App Development

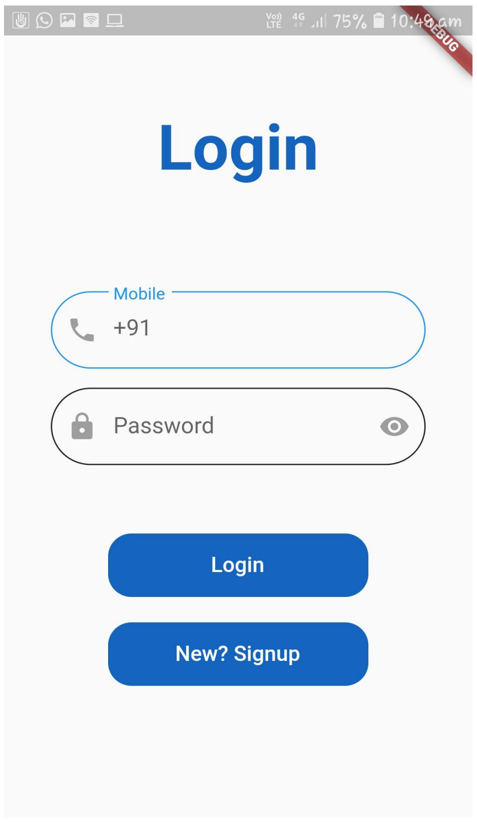
Front End

This is what the user sees on their mobile phone etc, we tried to provide the best experience to the user by implementing the logics in Flutter.
Why Flutter? Flutter is a not so tough language to learn and is easy to catch up and the user base around the world is huge to extend support when needed. I could easily find the answers to my needs on the internet, from seniors without much hassle.
Flutter provides various useful tools required in development of such an application, with various features, not just that Flutter can also be used to create the application compatible with both Android and iOS, without having to spend much time.

The Implemented features are being described below in detail:

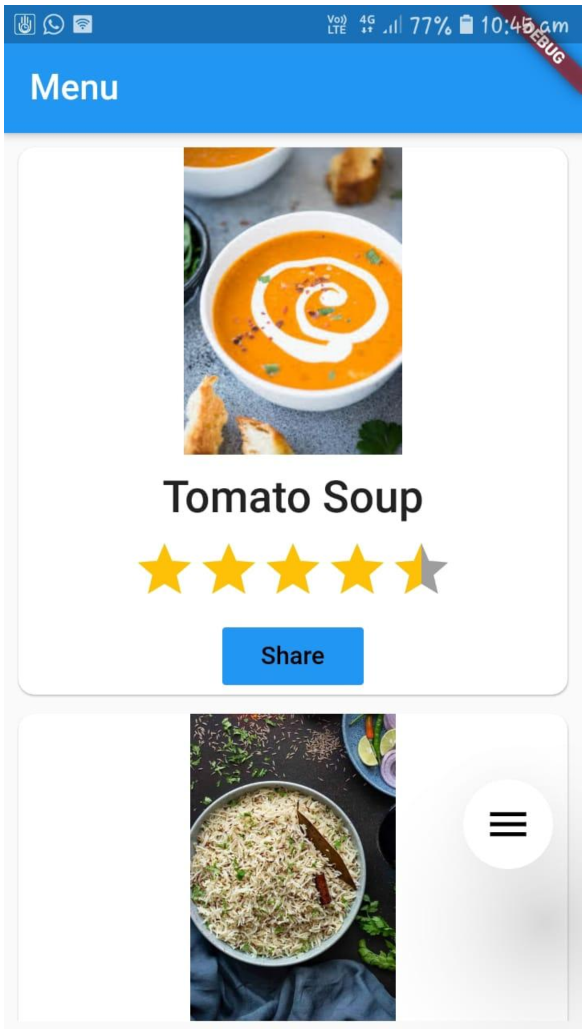
Login/SignUp : When the application is loaded, the login screen is displayed first. Any new user can create an account. The user is asked for Name, Mobile Number and a Password. An OTP is sent to the mobile number for verification. Once this part is done, the user can then login. The user needs to enter their username and password. The credentials are verified from the database.
After successful login, the user is taken to the menu screen where the dishes are displayed which can be cooked with customisations.

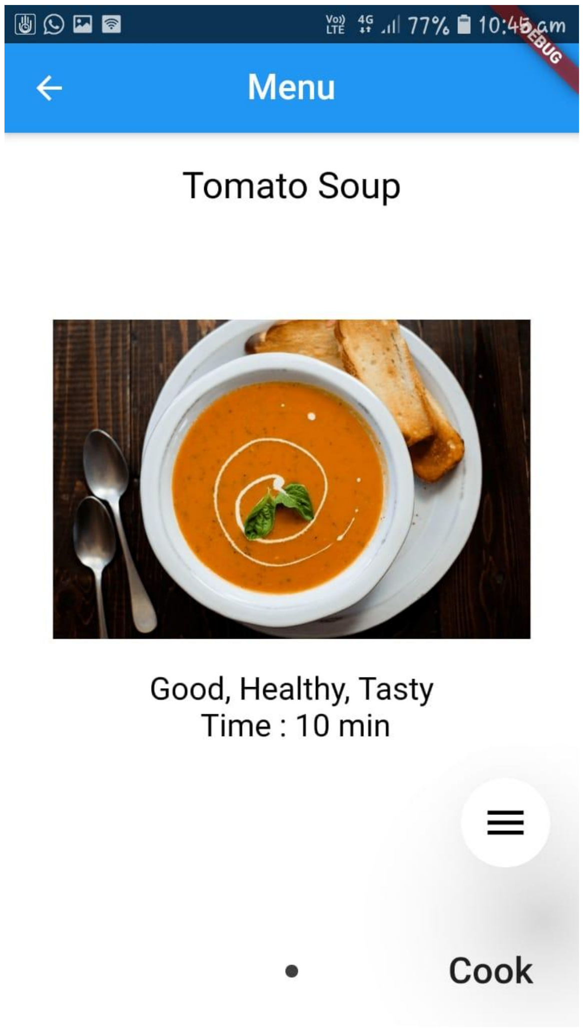




Menu Page:

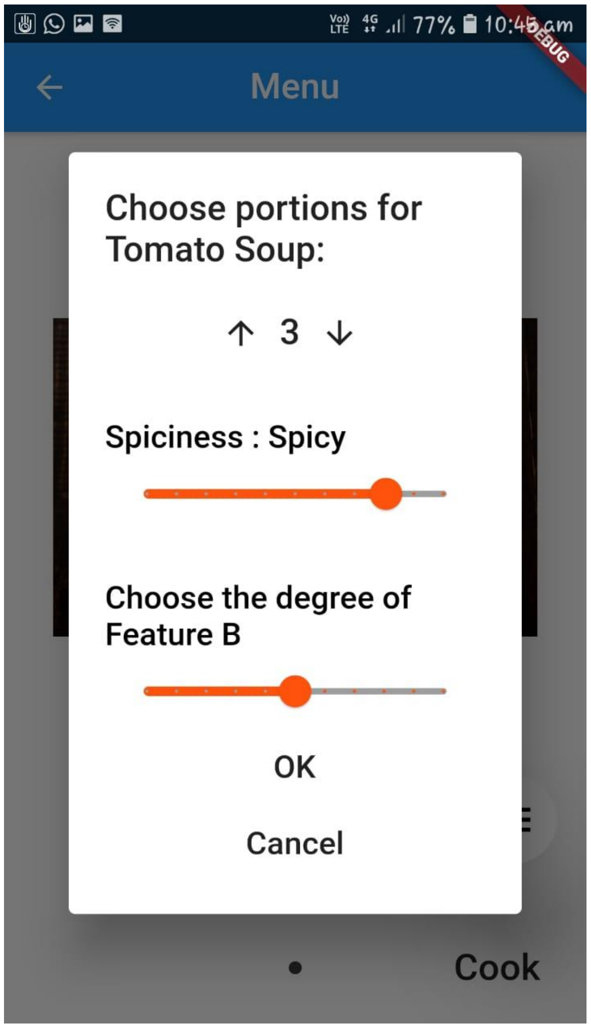
In the menu screen, the user can see all the available recipes and can select one of them to cook. The initial page will have brief information about the dish, such as the name, picture and rating.The display is shown in the below image. The recipe can be selected by tapping on the corresponding tile, which takes the user to another screen that gives an elaborate description of the item being cooked such as the time required, ingredients necessary
To start cooking, the user needs to tap on the ‘Cook’ button on the bottom right of the screen. It displays an alert box that asks the user for the custom inputs regarding the number of portions required, and the level of spiciness/tanginess etc.





Ordering:

The user can give their custom requirements and tapping on ‘OK’ sends the information to the database which confirms the order. The information is being sent using APIs to connect to the server.
After this process, the user is sent back to the menu screen.
Now, the user can check the details regarding that order from the ‘being cooked’ screen which can be accessed using the floating action button on the bottom right. There shall be updates from the machine to server which inturn will notify the mobile app and hence the user can see the live updates such as the stage of cooking, time left

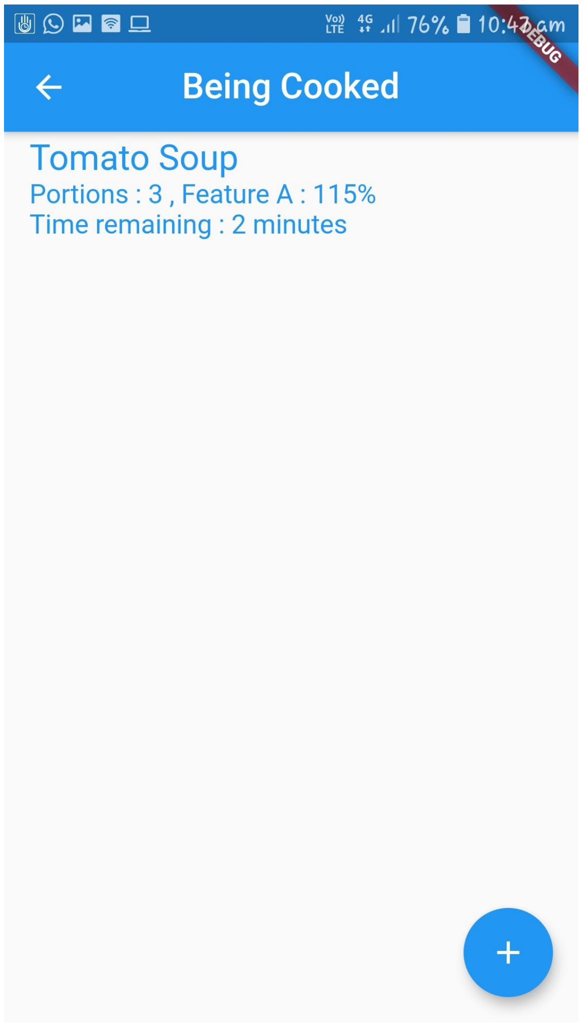


Choose portions for Tomato Soup:

↑ 3 ↓

Spiciness : Spicy

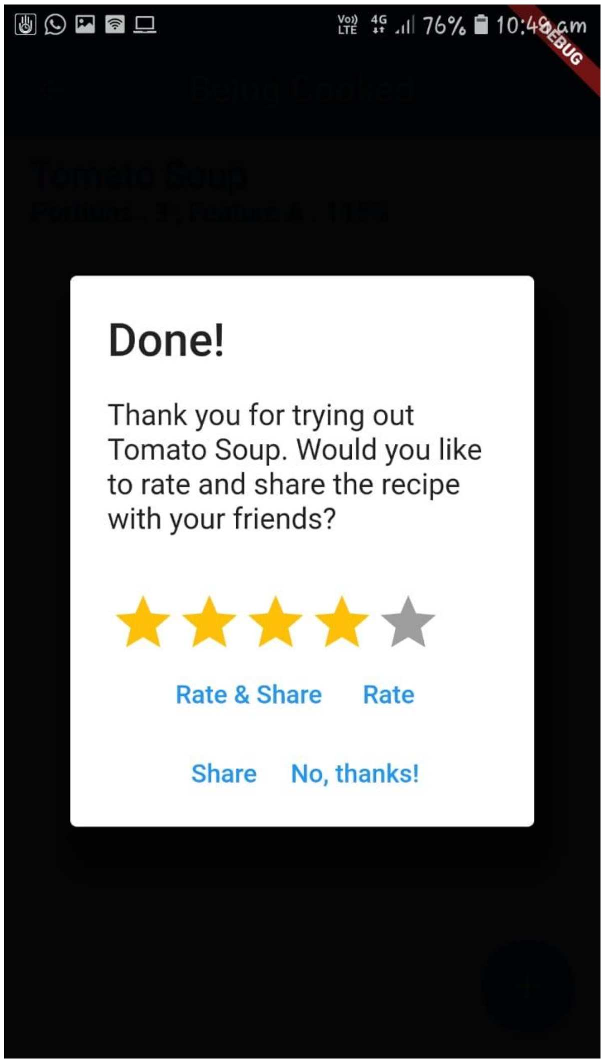
Choose the degree of Feature B



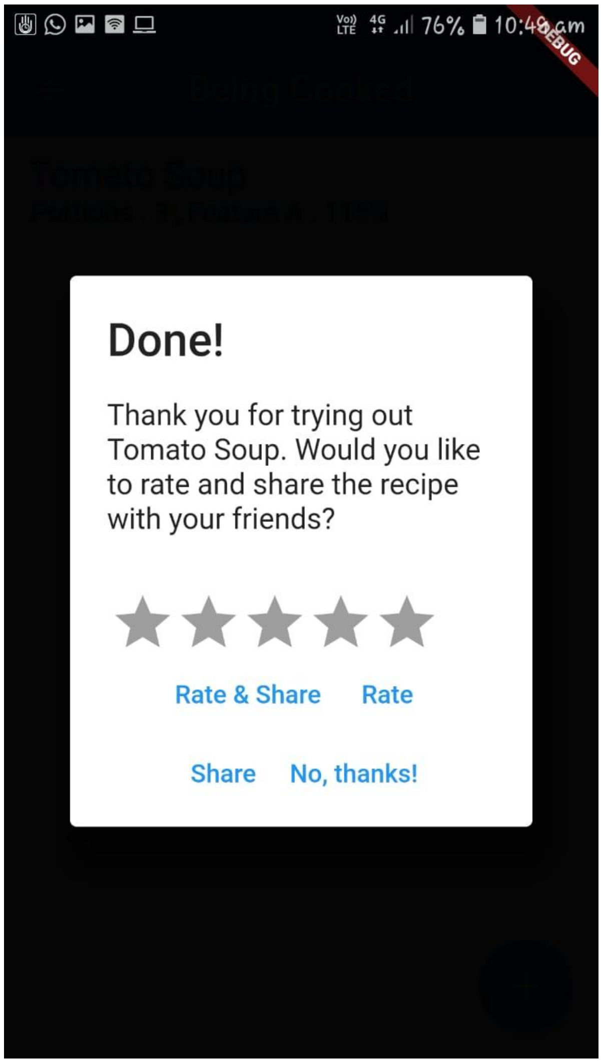
Tomato Soup
Portions : 3 , Feature A : 115%
Time remaining : 2 minutes

Notifications:

The order is displayed as shown in the above image. The estimated time remaining is shown in minutes. Once the cooking is finished the app asks the user to rate the recipe and/or share the recipe with friends. The image is shown below


Done!

Thank you for trying out Tomato Soup. Would you like to rate and share the recipe with your friends?

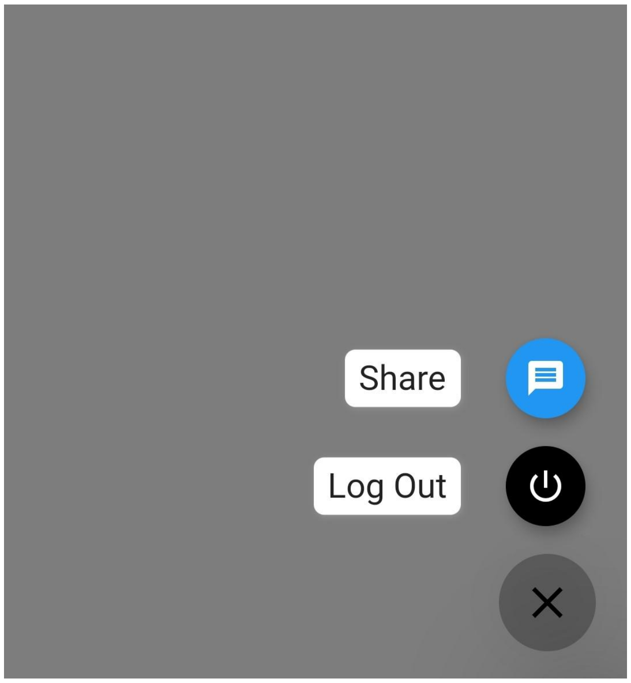

Done!

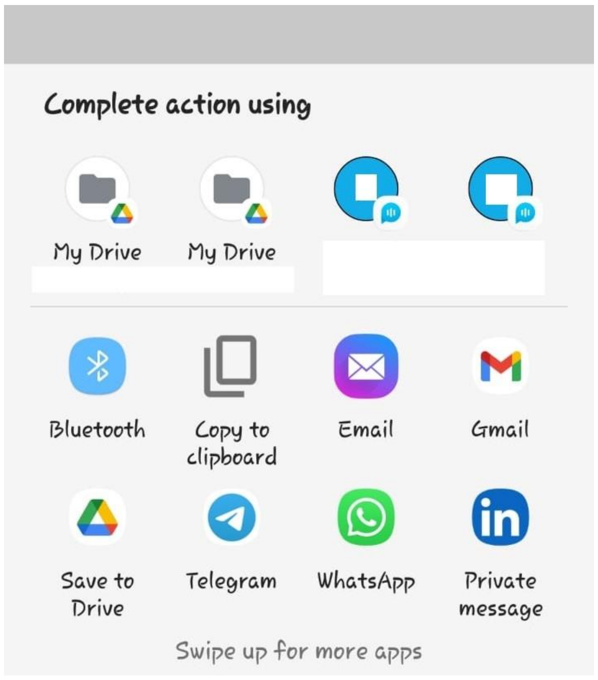
Thank you for trying out Tomato Soup. Would you like to rate and share the recipe with your friends?

Share Option:

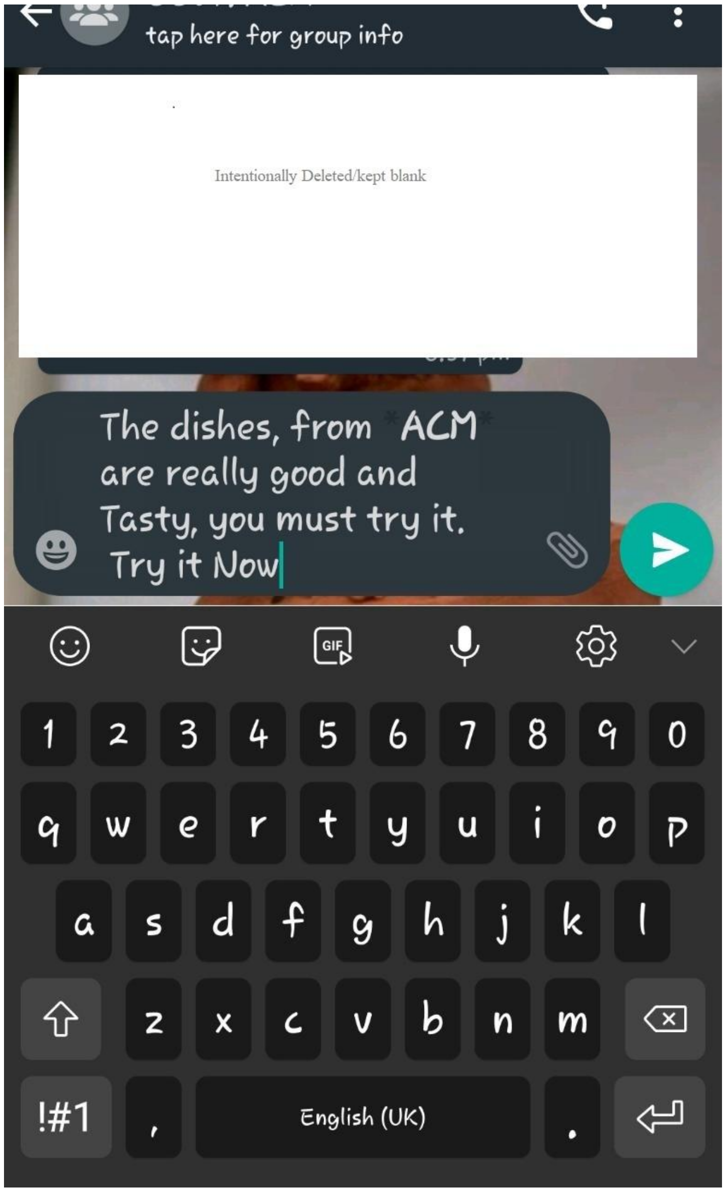
The user will have an option to share their experiences and the dishes they have tried in the machine to their friends, announce the same in social media over various platforms such as WhatsApp, Instagram, Email, SMS etc.

The share message can be custom made to advertise the app, machine and the services offered. This will be a great step to spread things across people and attract them as customers.





CΦ


tap here for group info

Intentionally Deleted/kept blank

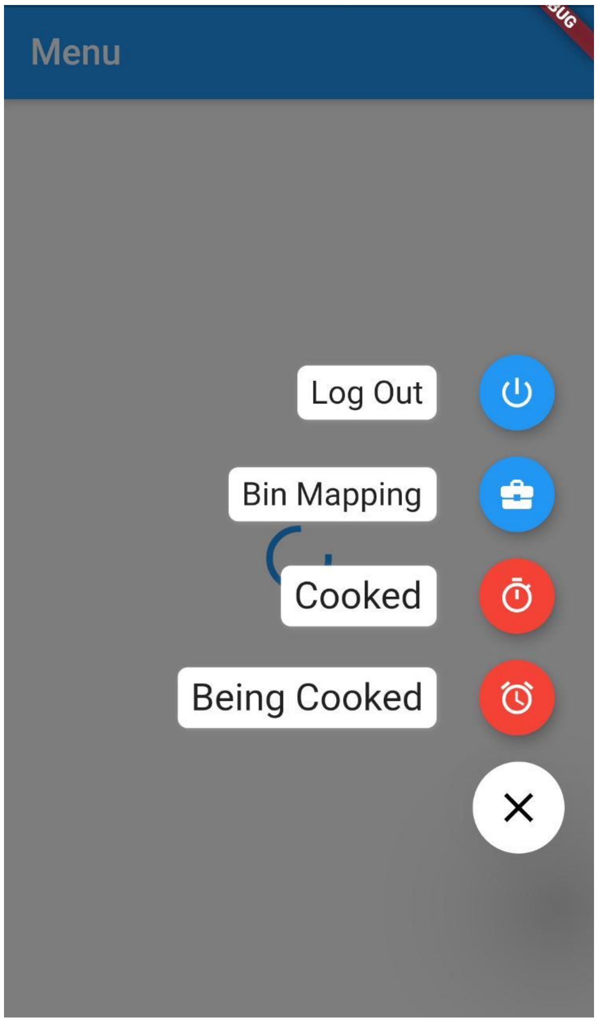
The dishes, from *ACM* are really good and Tasty, you must try it. Try it Now

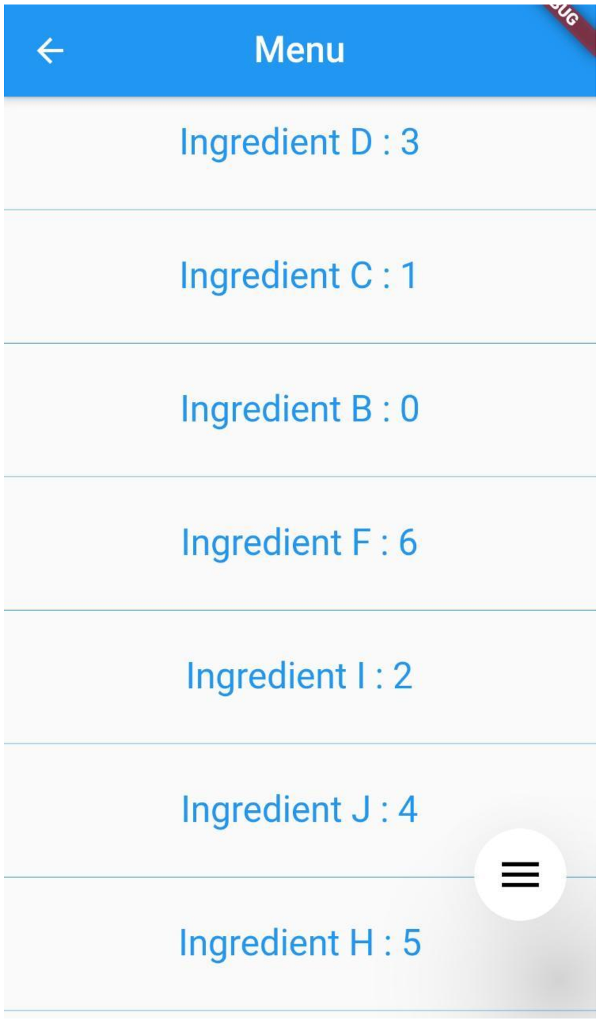
The above is an example of the share option using WhatsApp, the text hasn't been typed but rather came directly from the App.

This shall be very similar in case of other platforms, one common message can be used to spread the information over multiple platforms.

Other Features:

The user can also see the history of cooked items from the ‘cooked’ list which is also available in the floating action button. This list contains the items already cooked by the machine.





Ingredient D : 3

Ingredient C : 1

Ingredient B : 0

Ingredient F : 6

Ingredient I : 2

Ingredient J : 4

Ingredient H : 5

There is another screen called ‘bin mapping’ which is used to track the ingredients present in the bins. This screen can also be accessed from the floating action button. The user needs to update this list depending on the ingredients present in the bins.

Backend-Server

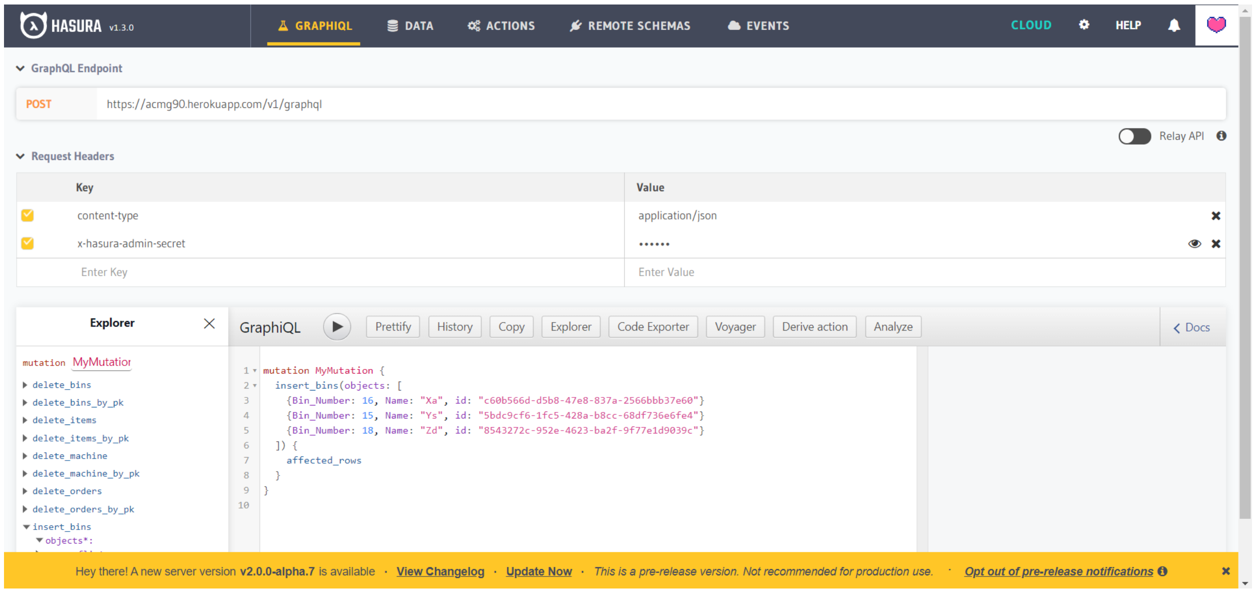
Server is basically a database, like a pendrive, where information is stored and can be accessed without physical connection.

There are multiple servers that provide service to the users like AWS, Firebase, Heroku-Hasura etc..
We chose to work with heroku, as it is a free service and user-friendly. There isn't explicit need to know much and can be learnt and be used quickly.

The GRAPHQL Endpoint of the server: Information in the server can be requested, modified etc.
To do that we let the server know what we want by putting a request to the GRAPHQL Endpoint.

APIs are designed to do the same. The data can also be modified from the terminal within, the following example shows the syntax, on how to do that.

"Query" command can be used to find out what all information is there in the data table
"Mutation" command is used to alter the data as per the requirement.



POST https://acmg90.herokuapp.com/v1/graphql

Key	Value
content-type	application/json
x-hasura-admin-secret	••••••
Enter Key	Enter Value

```
mutation MyMutation
delete_bins
delete_bins_by_pk
delete_items
delete_items_by_pk
delete_machine
delete_machine_by_pk
delete_orders
delete_orders_by_pk
insert_bins
  objects*:
```

```
mutation MyMutation {
  insert_bins(objects: [
    {Bin_Number: 16, Name: "Xa", id: "c60b566d-d5b8-47e8-837a-2566bbb37e60"}
    {Bin_Number: 15, Name: "Ys", id: "5bdc9cf6-1fc5-428a-b8cc-68df736e6fe4"}
    {Bin_Number: 18, Name: "Zd", id: "8543272c-952e-4623-ba2f-9f77e1d9039c"}
  ]) {
    affected_rows
  }
}
```

Hey there! A new server version **v2.0.0-alpha.7** is available · View Changelog · Update Now · *This is a pre-release version. Not recommended for production use.* · *Opt out of pre-release notifications*

Authentication:

To access the database to put any requests we have to login, which is being done both through the website and the app(Flutter command):

Through website:

URL: https://acmg90.herokuapp.com/console/login
After typing the password, you shall be given access to find the data.

Through app(Flutter APIs):

All the necessary details such as GRAPHQL endpoint, format, password are fed in and every time a request is to be made, the request is authenticated and access to data is given
The data received from the server is then processed and displayed to the user, as discussed in the frontend part.

```
static const API = 'https://acmg90.herokuapp.com/v1/graphql';
var headers = {
  'content-type': 'application/json',
  'x-hasura-admin-secret': 'acmcfi'
};
```

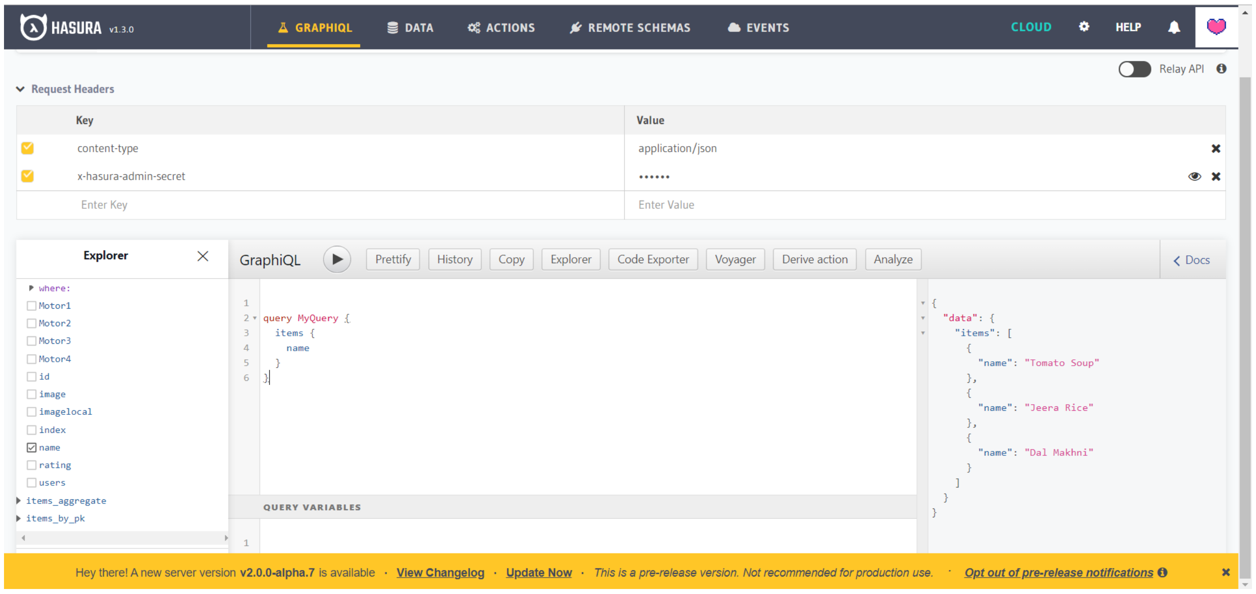
Example of a **Query** command:

This command can be used to explicitly find all the information/data that we want to know

1) In GRAPHQL Endpoint:

```
query MyQuery{
    items {
        name
        }
    }
```

The above example will get all the data stored in the Column “**name**” in the table “**items**”
On running the above, the result is seen towards the right.



Key	Value
✓ content-type	application/json
✓ x-hasura-admin-secret	••••••
Enter Key	Enter Value

```
▸ where:
☐ Motor1
☐ Motor2
☐ Motor3
☐ Motor4
☐ id
☐ image
☐ imagelocal
☐ index
☑ name
☐ rating
☐ users
▸ items_aggregate
▸ items_by_pk
```

```

query MyQuery {
  items {
    name
  }
}
```

```
{
  "data": {
    "items": [
      {
        "name": "Tomato Soup"
      },
      {
        "name": "Jeera Rice"
      },
      {
        "name": "Dal Makhni"
      }
    ]
  }
}
```

Hey there! A new server version **v2.0.0-alpha.7** is available · **View Changelog** · **Update Now** · *This is a pre-release version. Not recommended for production use.* · ***Opt out of pre-release notifications***

2) In Flutter:

```
datas =
            '{"query":"query MyQuery {\n   orders {\n     item_name\n
id\n              is_cooked\n           portions\n           EndTime\n
}\n}\n","variables":null,"operationName":"MyQuery"}';
```

This is to write an API in flutter to accomplish the task of getting data from the server.

Example of a **Mutation** command:

1) In GRAPHQL Terminal:

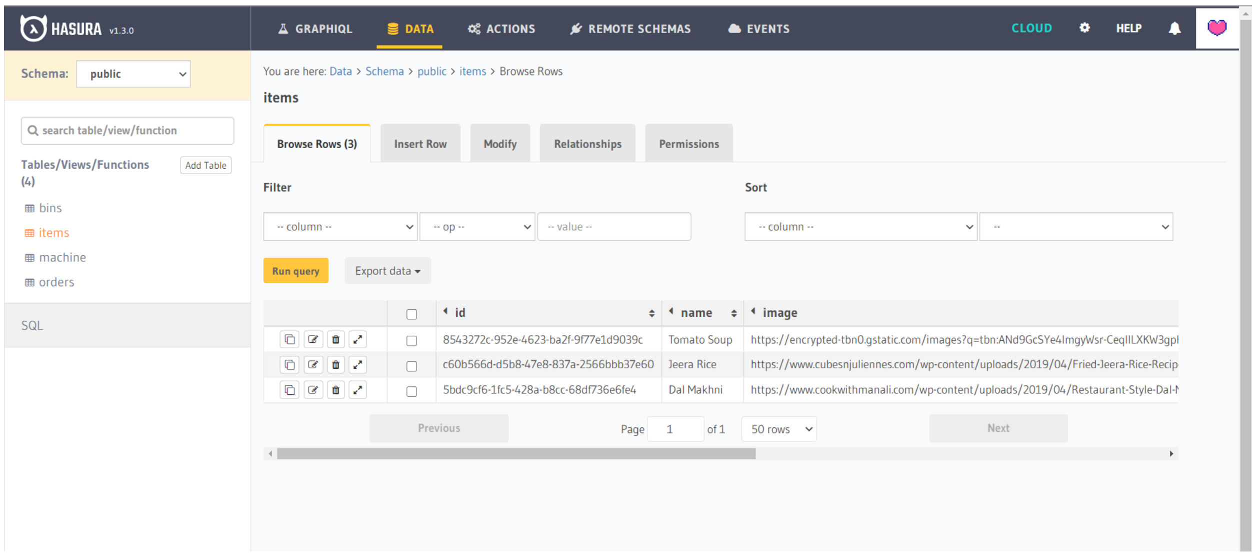
```
mutation MyMutation {
  insert_bins(objects: [
    {Bin_Number: 16, Name: "Xa", id: "c60b566d-d5b8-47e8-837a-2566bbb37e60"}
    {Bin_Number: 15, Name: "Ys", id: "5bdc9cf6-1fc5-428a-b8cc-68df736e6fe4"}
    {Bin_Number: 18, Name: "Zd", id: "8543272c-952e-4623-ba2f-9f77e1d9039c"}
  ]) {
    affected_rows
  }
}
```

2) In Flutter:

```
req =
          '{"query":"mutation MyMutation { update_bins(where: {id: {_eq:
\\"${item.binid}\\"}}, _set: {Bin_Number: \\"${item.binnum}\\"}) {
affected_rows returning { Name id Bin_Number }
}}","variables":null,"operationName":"MyMutation"}';
```

Data Tables:

The following is an example of a Table where data of various parameters is stored.



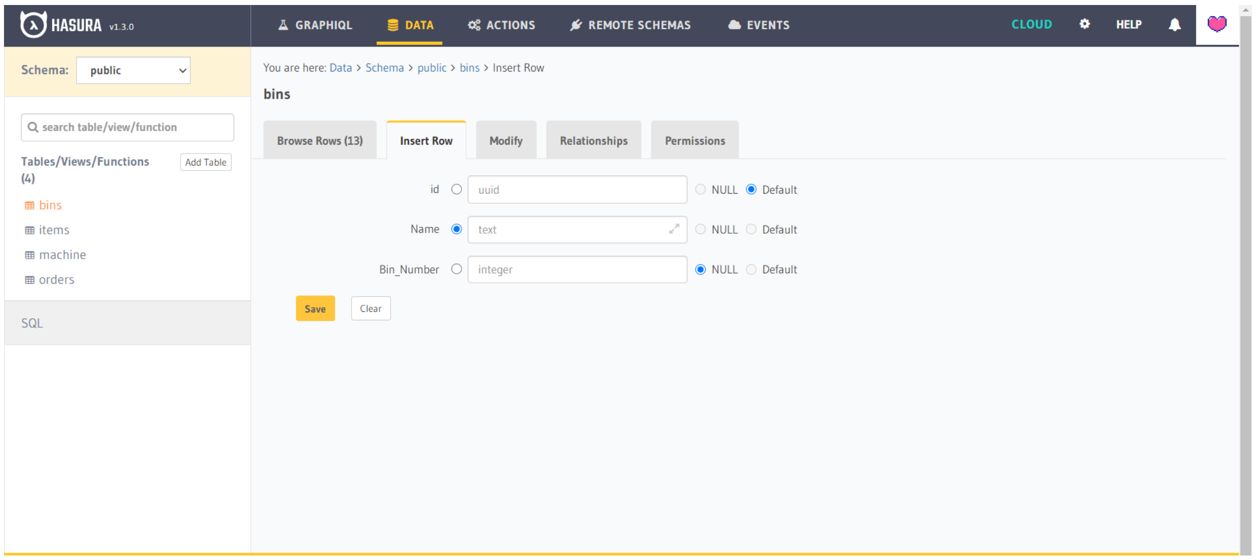
You are here: Data > Schema > public > items > Browse Rows

items

id	name	image
8543272c-952e-4623-ba2f-9f77e1d9039c	Tomato Soup	https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSYe4ImgyWsr-CeqIILXKW3gpl
c60b566d-d5b8-47e8-837a-2566bbb37e60	Jeera Rice	https://www.cubesnjuliennes.com/wp-content/uploads/2019/04/Fried-Jeera-Rice-Recip
5bdc9cf6-1fc5-428a-b8cc-68df736e6fe4	Dal Makhni	https://www.cookwithmanali.com/wp-content/uploads/2019/04/Restaurant-Style-Dal-N

Page 1 of 1 50 rows

Data can be added/altered through the above Mutation method or directly:



You are here: Data > Schema > public > bins > Insert Row

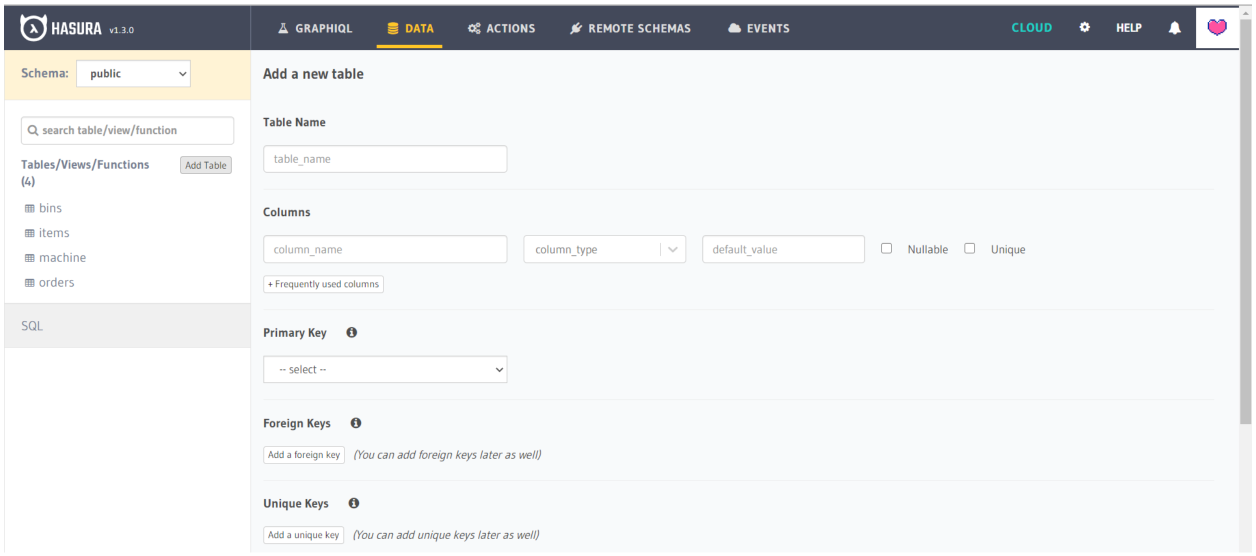
bins

id	○ uuid	○ NULL	● Default
Name	● text	○ NULL	○ Default
Bin_Number	○ integer	● NULL	○ Default

The Required data can be added to the following which shall be saved and be reflected in the server immediately.

To Create a new table:

Add table option is chosen and the required parameters are fed in, after which the table is ready and the necessary data can also be filled in.



Add a new table

Table Name

table_name

Columns

column_name | column_type | default_value | ☐ Nullable | ☐ Unique

+ Frequently used columns

Primary Key

-- select --

Foreign Keys

Add a foreign key (You can add foreign keys later as well)

Unique Keys

Add a unique key (You can add unique keys later as well)

The entire app can be found in https://github.com/Santosh-Gudlavalleti/FlutterApp1.git

Electronic Module : IOT System

PHASE 1 (Getting started with ESP32):

1. **Why ESP32 microcontroller ?**

   Initially a PIC microcontroller was used by the Butterfly team, but we decided to use ESP32 devkit-C module mainly for its WiFi and bluetooth capabilities. Also, because it can be on voltages 3.3V and 5V which can be useful for running motors on it.
   We also decided upon using the Arduino IDE for coding the ESP32 over the espressif IDF because it was easier to program and there were inbuilt libraries to access WiFi/ bluetooth capabilities.

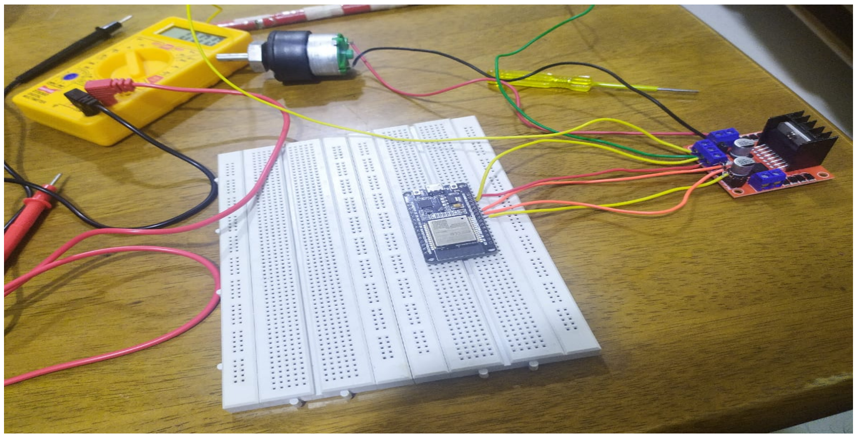
2. **Usage of ESP32 in the ACM machine and choosing the right motor driver**

   We initially carried out some basic codes to explore the different functionalities of ESP32 mainly using its bluetooth and WiFi libraries. The main usage of the hardware aspect of ESP32 was to integrate the motors and control them.
   We decided to use the L298N dual H-bridge motor driver as it has high motor supply voltage and current of around 40 volts and around 2 amps of current respectively. So, with the help of a multimeter we measured the voltage and currents used up by a 30V DC motor.

3. **Tasks accomplished by using ESP32**

   Initial phase of this project employed the use of ESP32 microcontroller (WiFi, bluetooth capabilities) to connect to the cloud. As a part of testing, we were successfully able to connect ESP32 to **AWS IOT cloud**. Further functionalities (motors etc.) were also tested which may be used in the original machine for overall functioning.
   Further experimentation and research was done on **AWS IOT** which was planned to be the cloud server for the project. ESP32 was connected to AWS IOT Core and was able to send and receive data from the cloud in **JSON** format.

The configuration process of the ACM machine (i.e ESP32 circuitry) with WiFi network along with mobile application was also tested. The basic configuration of any IOT device works like this. Refer this video to gain a better understanding on how configuration was implemented in the case of our machine.
The relevant codes of all the information shared above is available here.



PHASE 2 (Switching to RaspberryPi and capacitive touchscreen):

1. Why capacitive touchscreen ?

Touchscreens are of two types majorly: Resistive and capacitive. While the resistive screen is cheaper, it requires hard presses to activate and is not as sensitive to touch as a capacitive touchscreen.

The capacitive touchscreen on the other hand recognises actions like swiping on the screen, and provides a smoother and a more colourful interface for the user. It is the type of touchscreen that we use in our mobile phones.

2. Why RaspberryPi over ESP32 ?

The second phase of the project envisioned the use of RaspberryPi because of two major reasons -

- Ability to process and run an app on a touchscreen interface.
- Android OS/SDK support.

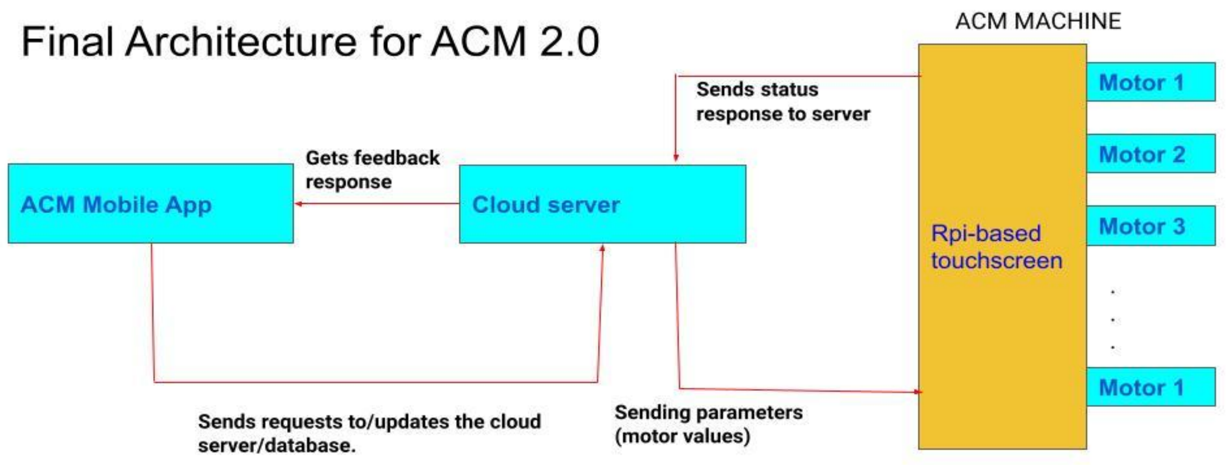
The idea was to have a **touchscreen interface** on the **machine** itself for a user to operate the machine through a mobile app as well as the on-machine touchscreen. ESP32 is a simple microcontroller which doesn't have the required computation power of running an app and controlling motors at the same time. RaspberryPi is a much advanced and efficient microprocessor and seems to be the best fit for our application according to this reference from the official documentation of Android Things. It can run the app on touchscreen and simultaneously control the internal circuitry of the machine by sending relevant control signals to all the motors.

RaspberryPi 3B+, **USB cable**, **SD card**, **capacitive touchscreen** was shipped to one of the team member's (Sarthak) house. The RaspberryPi was subsequently tested along with the touchscreen for around a month.

3. **How does RaspberryPi work in the ACM machine ?**

The main motive of Raspberry Pi is to integrate the above explained mobile application and cloud server with the cooking machine. **Raspberry Pi 3B+** with a **capacitive touchscreen** can operate the same **Flutter mobile application** as explained before.
The basic workflow of the whole system is shown here -

Final Architecture for ACM 2.0


ACM MACHINE
Sends status response to server
Gets feedback response
ACM Mobile App
Cloud server
Rpi-based touchscreen
Motor 1
Motor 2
Motor 3
Motor 1
Sends requests to/updates the cloud server/database.
Sending parameters (motor values)

- Flutter-pi library was used for running flutter engines on Rpi.
- Possible to use Android OS too.

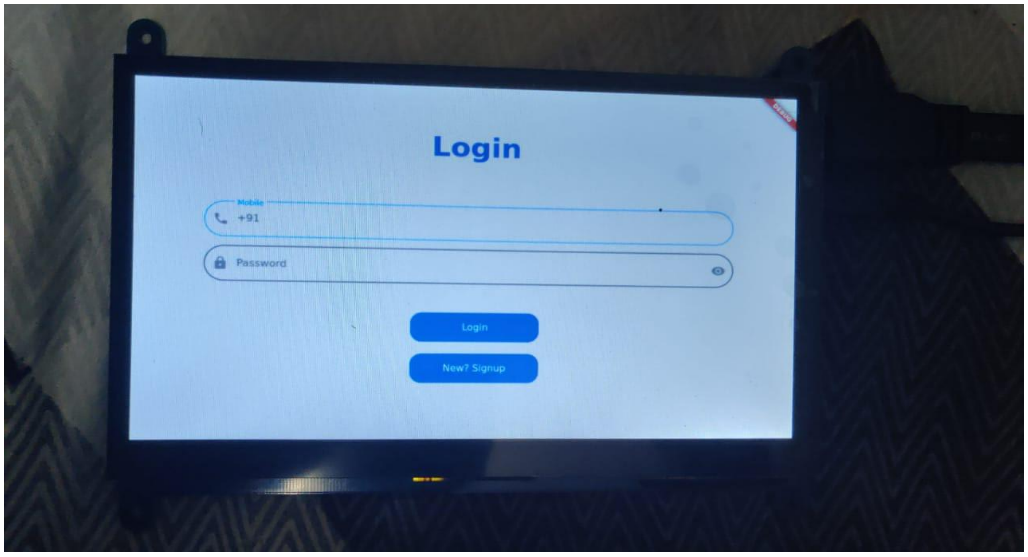
➔ Boot RaspberryPi OS onto RaspberryPi through a 16GB SD Card. Follow the official documentation to install the same.
➔ Flutter-pi was used to install Flutter libraries in Raspberry-Pi. This library creates a flutter engine which can easily run Flutter apps on RaspberryPi. Flutter-pi can be installed after booting RaspberryPi OS.
➔ Whenever a user clicks on a particular recipe on the **app** or on the **touchscreen,** relevant data/parameters are first sent to the **cloud server** and the database is updated.
➔ The flutter app running on RaspberryPi will constantly ping the database for any possible updates or changes.
➔ The cloud server sends relevant **parameters** to RaspberryPi to operate the motors connected to the machine.
➔ The flutter code running on RaspberryPi also supports several **Python codes** which are responsible for running different motors attached to the machine. This is a better option than running flutter codes for running motors because of the wider scope and ease of integrating hardware for the engineer.
➔ The parameters obtained from the cloud are used in these Python functions which in turn operate the motors accordingly.
➔ From the touchscreen point of view, the process is almost exactly the same as that for the mobile application.
➔ The above cycle keeps on repeating as and when the user selects more food recipes in the future.

The reference for the Flutter app code used for RaspberryPi is https://github.com/Santosh-Gudlavalleti/FlutterApp1.git

The above app is run using Android SDK on RaspberryPi. Unfortunately, Android Things stopped their support for RaspberryPi because of which we cannot run official Android support on RaspberryPi. There are several third-party services that provide **Android OS** support for RaspberryPi, however their long-term functioning can't be guaranteed. Ideally, Android OS should have been booted and run in RaspberryPi as a result of which ACM app can be displayed on an android interface on the touchscreen. However, due to the above constraints, we booted RaspberryPi OS onto the RaspberryPi and ran the app over RaspberryPi OS using Android SDK itself.

4. Tasks accomplished with RaspberryPi and capacitive touchscreen

Raspberry Pi was successfully booted with RaspberryPi OS using an SD card.
Several online videos and references were referred to install the RaspberryPi OS
The capacitive touch screen was tested using the **flutter-pi** plugin mentioned above and the flutter app was successfully working on the Touchscreen. The video for the same can be found here.
The Flutter app used for RaspberryPi was modified such that only a single flutter app, running on RaspberryPi, can listen to the server/database along with touchscreen inputs, and subsequently operate all the motors for cooking a dish. However, the app has very basic features as of now and some advanced features like changing WiFi connection inside the app itself, pop-up notifications, Google-based keyboard for typing are yet to be implemented.
As of now, during the testing phase, the user can change WiFi connections in RaspberryPiOS itself in the '**network connection'** section.



5. **Future tasks with RaspberryPi**

   Android OS can be directly booted onto RaspberryPi. The flutter-app can then be installed through Google Playstore and the app can be operated. Since the app is running in an Android OS environment, all the minute features/functionalities like **Connecting to WiFi**, **Changing WiFi connection**, **pop-up notifications**, **update through Google Playstore** will be readily available without any additional hard work.
   However, currently the app is running on Android SDK, i.e over RaspberryPi OS, it doesn't support Android based features as mentioned above. Hence, there might arise a need to manually code for the above features in a flutter app running on **Android SDK**.
   The motor connections to RaspberryPi and the motor codes (in **Python**) need to be modified depending on the type of motors to be used by the company for the new ACM 2.0 model. The Python files are already available in the Flutter app code referenced above. Hence, the developer just needs to modify/change the python codes in the '**starflut files**' folder in the above app

   The RaspberryPi 3B+ along with the capacitive touchscreen has already been tested once successfully at CFI (Centre for Innovation), IIT Madras by our mentor Sundar and Karthik sir from the Butterfly team.

Conclusion and Future Plans

- The project is almost done with its prototyping phase. **3D-designs** have been modelled and hardware components/codes have been tested physically at **CFI IITM**.

- This project is a collaboration with **Butterfly company.** The company has agreed to continue this project over the next academic year as well. The main aim is to get started with the production phase of the 3D-models.

Learning Outcomes:

- Fusion 360 (3D designing and modelling)
- Microcontroller programming (Arduino IDE in **ESP32**)
- App Development (Flutter based)
- Motor drivers and motor functioning
- Cloud server database handling
- RaspberryPi

Bibliography

https://github.com/Santosh-Gudlavalleti/FlutterApp1.git
https://github.com/Sarthak-22/EE01-ACM
https://www.raspberrypi.org/documentation/
https://github.com/Developers-Goldenmace/AWS-IoT-with-LAMBDA-FUNCTION/blob/master/Lambda.py
https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html
https://github.com/ardera/flutter-pi
https://www.youtube.com/watch?v=SHc3NB1LdlI