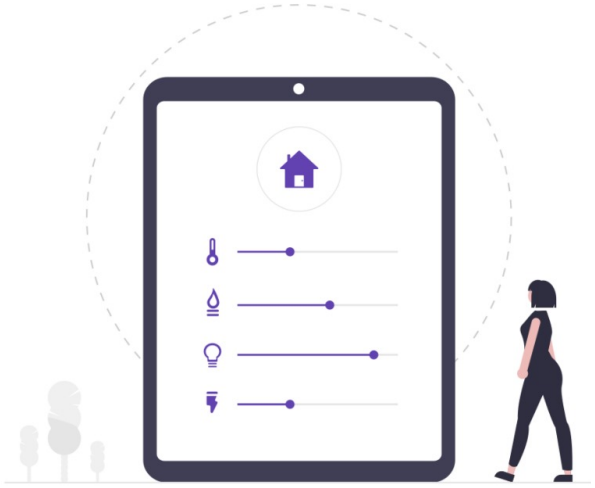
NYU TANDON

Principles of Database Systems (CS-GY 6083)

Smart Home Energy Management Systems (SHEMS)



Project Members

Jayanth Anala - ja4874

Venkatesh Damaraju - vd2348

CS-GY 6083 (Fall 2023)

Under the guidance of

Prof. Torsten Suel

Computer Science & Engineering

Introduction:

Smart Home appliances are the latest solution developed to provide higher control over electronic devices while making them easily accessible. Energy efficiency is the major question in the context of a Smart Home ecosystem. An advanced Smart Home Energy Management System is required to provide control over a robust system like this. The design developed here tries to address the major requirements provided in the problem statement using MYSQL database design procedure. We created a complex schema with multiple tables considering all possible scenarios of the problem statement. The design is developed keeping the basics of the database system in mind. It is completely normalized in all forms to eliminate redundancy at the same time retaining all the functional dependencies for the given problem statement. The UI is integrated with intuitive elements which are interactive and easy to use. All the graphs are visually appealing while presenting the required information.

Change in design from Part - 1:

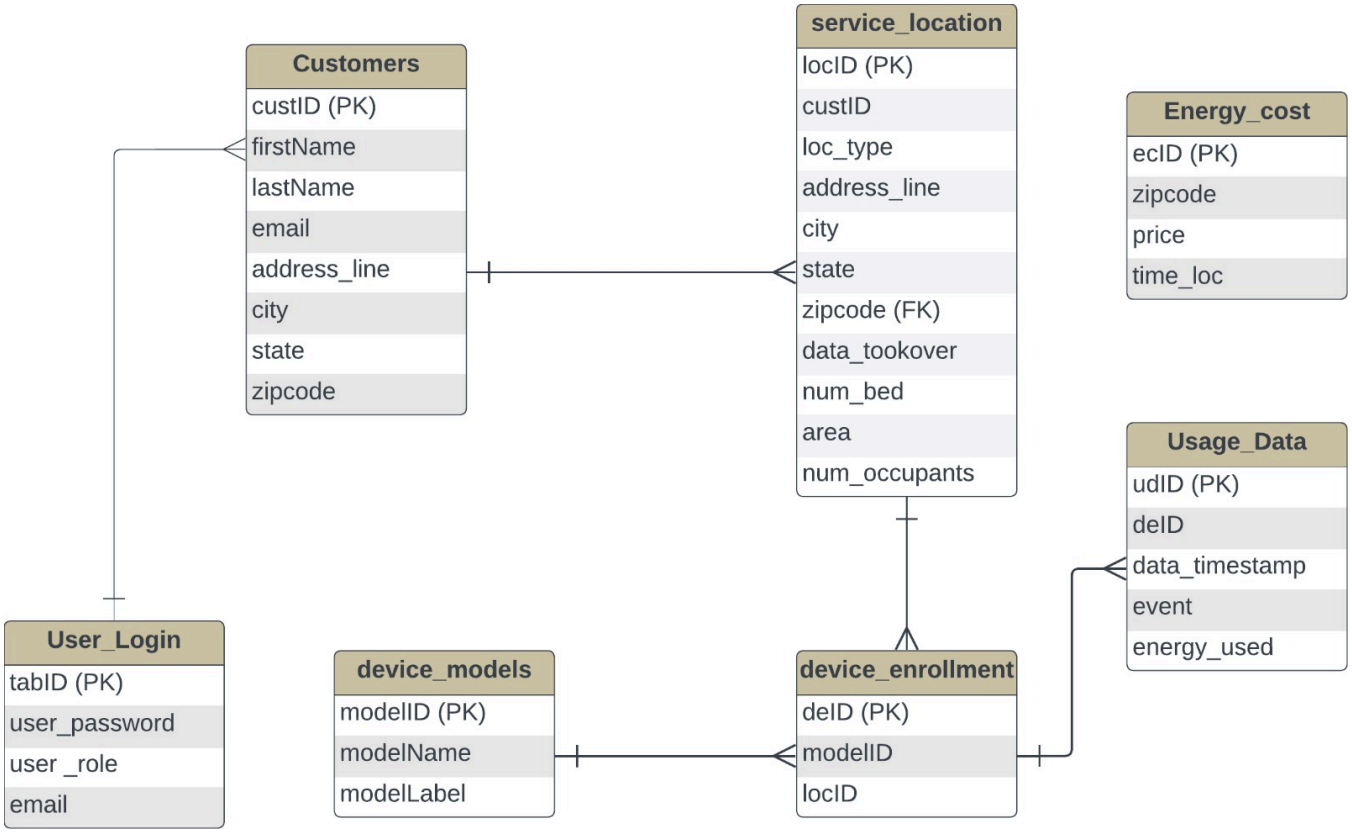
- The design in the part-1 did not include fields to store the User's login information for the UI.
- To include this we introduced a new table **User_Login**.

  This table contains the following fields:
  - tabID - INT - Unique identifier of the table
  - custID - INT - Customer ID from Customer table
  - User_password - VARCHAR - To store the hash of the password
  - User_role - VARCHAR - To store role of the user
  - Email - VARCHAR - To store email of the customer used for login

- The primary key of the table is tabID which is an auto generated key variable.
- CustID is the customer ID generated in the Customers table. It is referenced in this table.
- User_password stores the password of the User. Note that we are storing the hash encrypted value of the password generated by user.
- User_role is generally kept as 'User' for all the users. It is included in the design for an Admin Functionality. Only for admin it is set to 'Admin'. There are also few other roles stored in this value

- Email field stores the email address of the user which is used as login username.
- There are still no Weak Entities in the design.

Revised ER Digram:

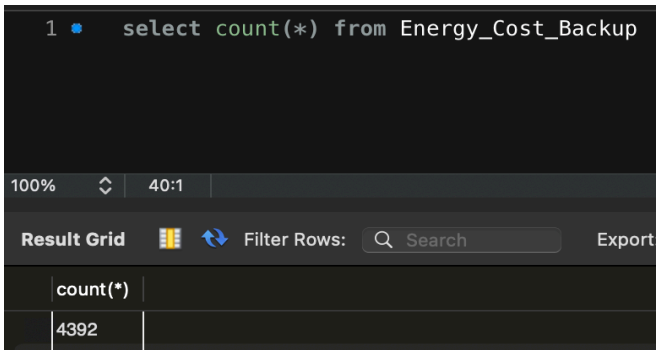


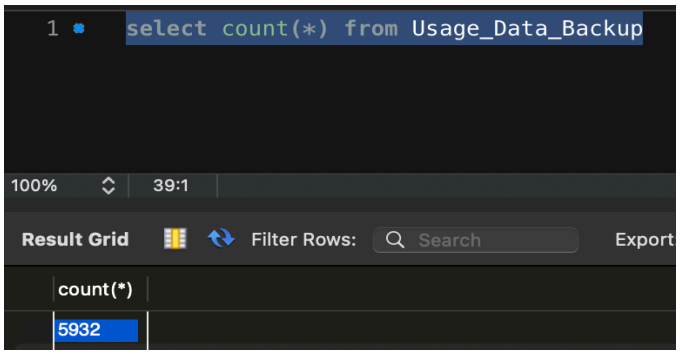
Quick Database Description:

For the application, we have given 8 users data in the Customers table. Then for two customers we gave two service locations and one for the rest of the customers in the Service Location. For every service location we enrolled almost 10 different devices. Also there are 20 models of different devices in the Device Models table.

As explained earlier, we have chosen to give usage data for an interval of 15 minutes. All appliances enrolled in the service location send data and their event status for every 15 minutes. Then for every different timestamp mentioned corresponding price data is also noted

for that zip code in the Energy Prices table. Finally for testing there are nearly six thousand





records in Usage Data and nearly 6000 records in the Energy Prices table

Normalization and Efficiency:

Usually, the normal forms are good measure to indicate the efficiency of a design as they are primarily focused on eliminating the redundancy present in the design.

• This design does not have any redundant values in it.

• Only case where there can be a possibility of data repeating is in the table device_enrollment where a customer can have multiple devices of the same model (Eg Lights of same model).

• This cannot be considered because it is mentioned that each device operates individually and we need to store each of its data in the usage_data separately.

• As a result the deID which is maintained as the primary key actually helps us in identifying each device uniquely and the usage is tracked based on it. Hence, cannot be considered as a redundancy.

Usage of Index:

Indexes are powerful and most useful in cases of searches and joins. Since we need to use all the tables when presenting information for various locations and models, proper indexing is required to increase the performance of the model. Along with the index on the primary key, we have introduced two clustered index in the database.

- **Index on delD in Usage_Data :** Usage_data is the prime table that is considered for any analysis and it is joined in almost all the cases. To achieve better performance we used index on delD so that the data pertaining to specific device is easily identified in all cases
- **Index on locID in Device_Enrollment:** While the records in this test are limited in Device Enrollment, in a real life scenario when we have thousands of customers and then tens of devices for each location, we should be able to filter them, and the connection flows through locID as Service Location acts a bridge between tables. So a clustered index on this would increase the efficiency of the system.

Tech Stack Used for Application:

Frontend - Flutter for UI

Backend - Node JS for Backend Routing

Database - MySQL database hosted on Digitalocean

Packages - Syncfusion Flutter Charts, JsonWebTokens, Dio

UI Design Explanation:

The entire UI design consists of 7 pages in total.

1. Login
2. Signup
3. Home Page
4. New Device
5. New Service Location
6. My Usage
7. Metrics

1. Login

The opening page where the user enters his email address (User Name) and password details. Signup option is provided here for new users. Once the user enters valid information, a new route is maintained at the backend, that would validate the data by sending a request to the database, it generates a JWT token which is then stored and used in rest of the UI.

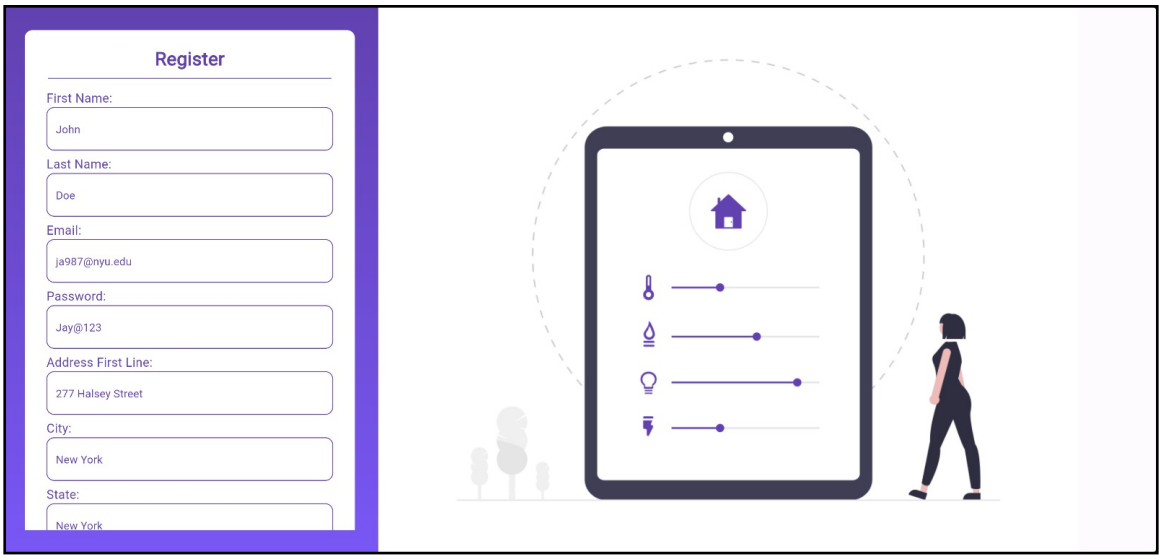
Basic validations of email and password (such as one capital letter, one number, one special

character and a minimum length of 8 characters) are all done using Flutter.

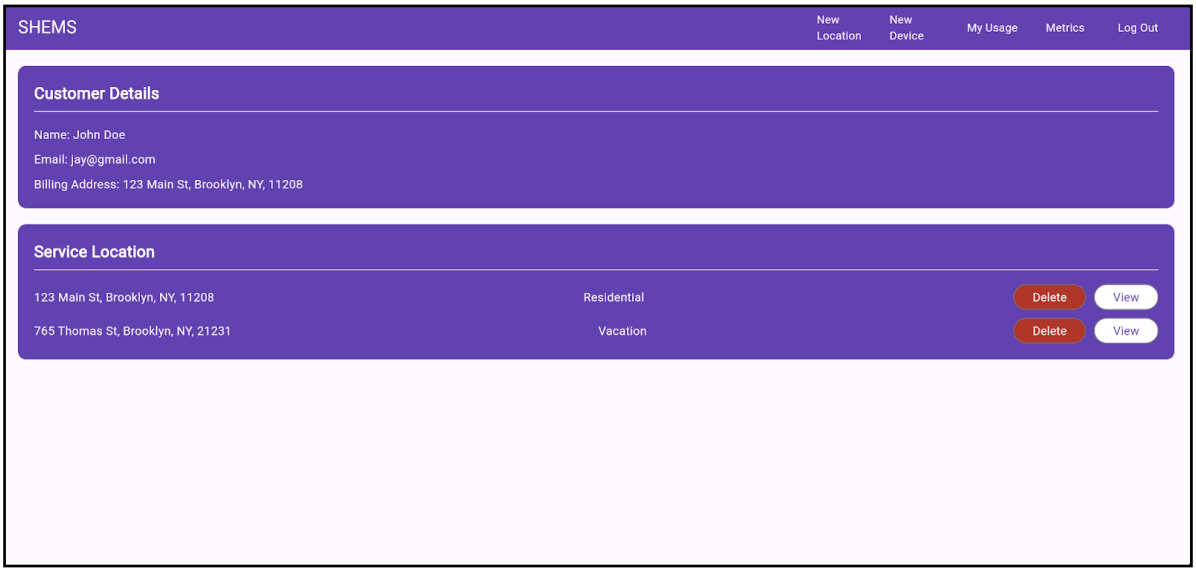
2. Signup

Basic information of the user such as first and last name, billing address of the user including city, state and zipcode with primary details and the email address.

Basic validations to the fields such as email validation using regular expressions, zipcode validation, fields data type validation are handled directly in the front end using Flutter. Once the user is successfully registered then a token is generated and then the user is redirected to Home page.



3. Home Page



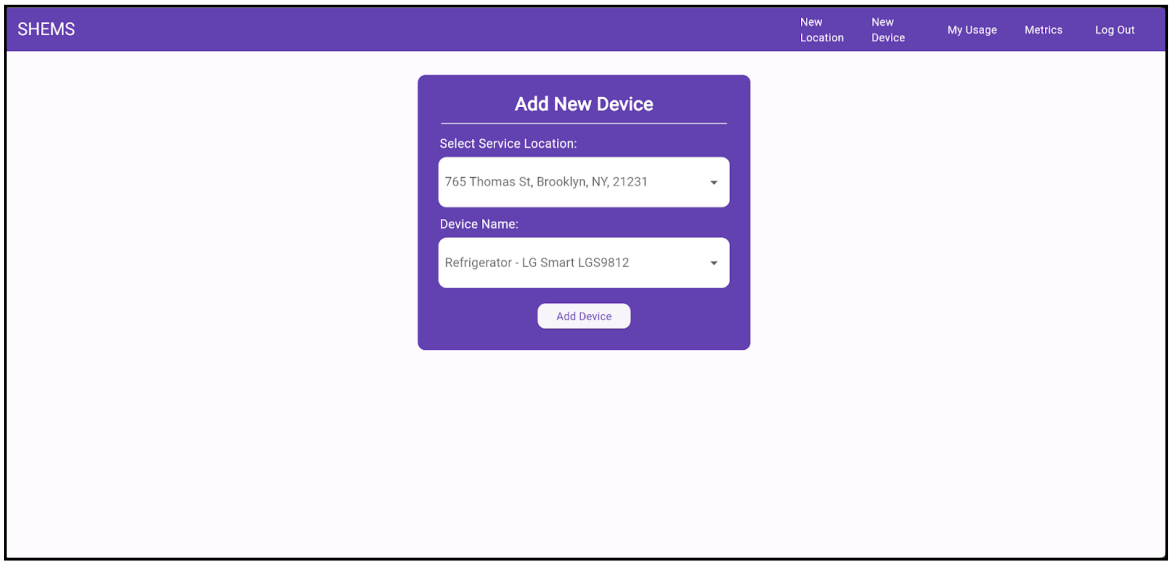
The Home page has 3 sections in the design.

- Navigation bar at the top
- Basic user information: We include details of the customer such as name and email address and his billing address.
- Basic location information: For a given customer all the service locations enabled for him are presented. All the service locations are listed as blocks with 2 buttons on the right end. When we click on view, then the division expands showing all the devices registered for the service location. The device model and make are listed here. A Delete button is

provided for both service location and devices related to it. Cascading delete action is performed when the button is clicked.

The navigation bar is located at the top which allows the user to view different interfaces enabled for him. It includes 5 different segments.

- **New Devices**

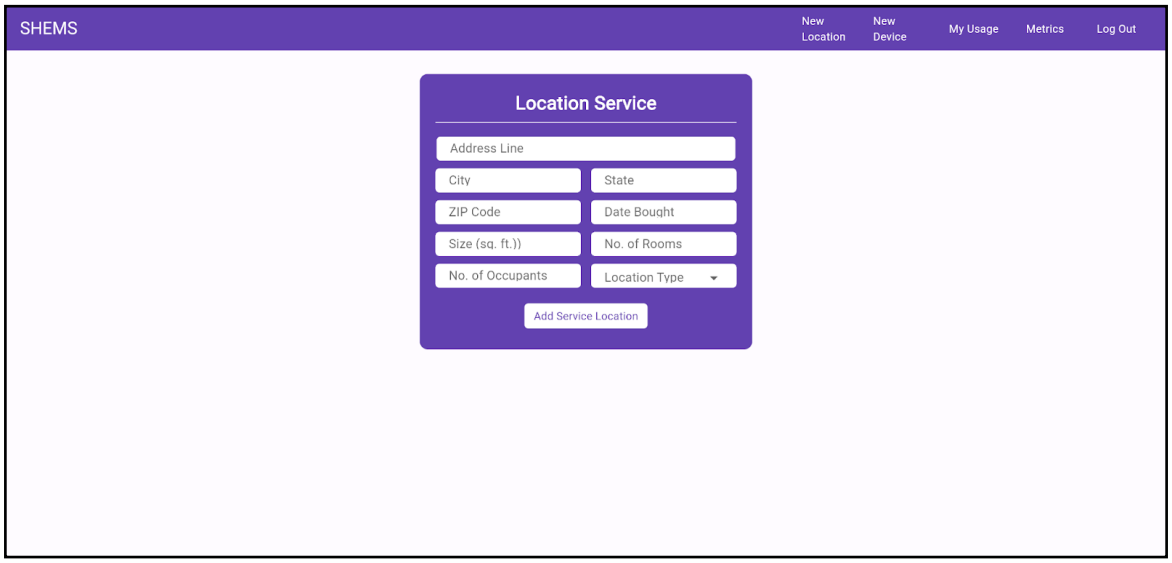


For a given user and a new device can be added for. Here two selections need to be made by the user. First the user must select the location for which he/she wants to add the device.User can select a model and device from the selection menu given. All the models are stored in the User_Models table of the database. These are presented to the user.These models need to be already present in the db. No new type of model can be added here. It is left for the admin of the database. The user can only use the present models and register it for his service location.

- **New Locations**

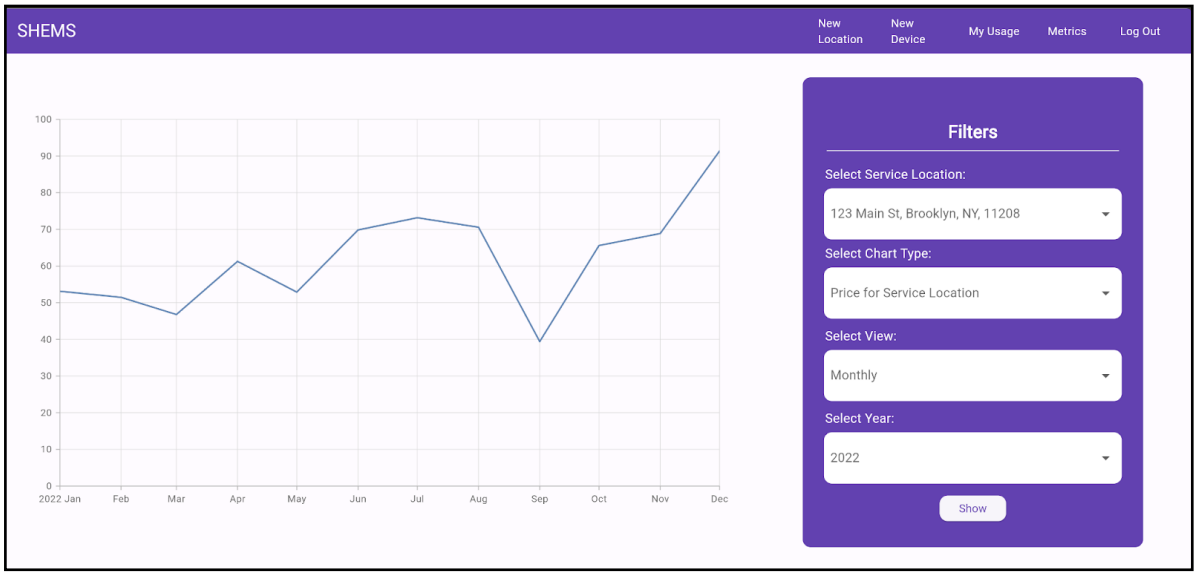
- For a given user, a new service location can be added using this module. Complete information regarding the house such as number of bedrooms, area, number of occupants, size and the data at which the location is brought is required here.
- Along with these details we also allow the user to select what type of location they want to register this as. This selection is also predefined by the database admin. User can only

select from the given values. Note that users can have a few number of service locations of the same type. On Successful registration of service location, the user is redirected to the home page.



- **My Usage**

All the statistics related to the user are presented here. This page consists of two sections, the left half of the page has the graphs related to the usage while right half of the page consists of the selection that the user can make to see a variety of graphs.

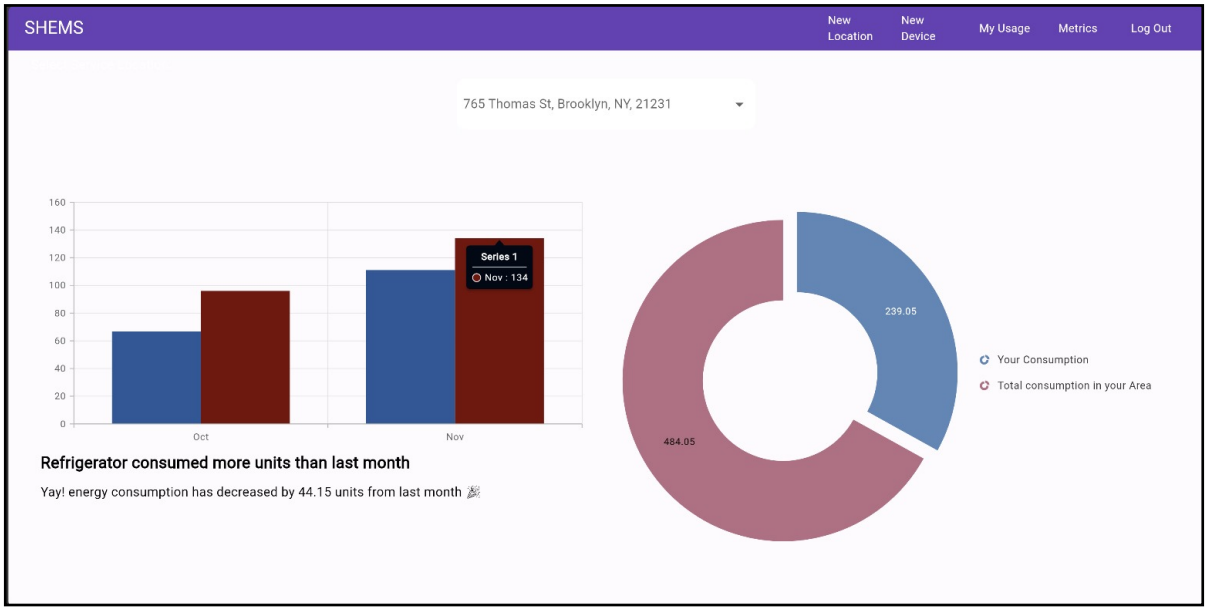


- **Metrics (Wow Page - Extra Credit)**

In this page we have given general insights related to the user. We believed that general statistical information would be cumbersome for some users and difficult to draw important insights. So we decided to present some quick insights that are readily available for the user as soon as the page loads.

On the left side we compare the user's latest billing information with previous billing information that is the energy consumption and total price.
Along with it we compare the user's appliances data, and see what contributed to increase or decrease in the billing and energy consumption.

On the other side, we have the information of total energy consumed by the user in the latest billing month and compare it with overall energy consumption of all users in the same zip code and compare them and indicate it as a percentage.
Here we choose the same zip code because the price depends on the zip code and it varies based on it.



Explanation of the Usage Dashboard (Graphs):

There are 5 types of analysis that can be done on the dashboard:

1. Price for Service Location
2. Energy Consumption for Service Location
3. Price for Appliances (Service Location)
4. Energy Consumption for Appliances (Service Location)
5. Price and Energy for all Service Locations

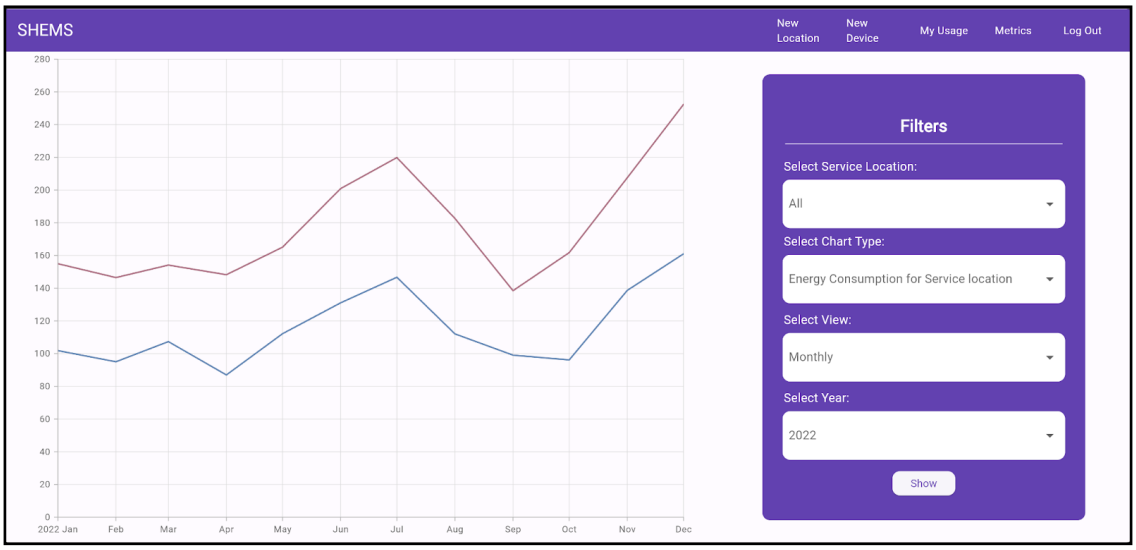
—— Various mixture of filters shown above can lead to different graphs with various time frames like Daily, Yearly, Monthly and also year/month selection accordingly.

- The menu consists of five sections which changes dynamically based on the selection of the first few options.
- Using the first option, the user can select any one hi/her service location. There is also a generic option called ‘All’ which would show the statistics of all the locations combined that belong to the user.
- In the second option the user can select the type of metric he wants to see as a graph. There are four different types of values present in this option.
- Primary classification of graphs is done in two ways, based on the energy and the price. Then a further classification based on service location (which combines all devices data) and then an overview of all appliances data available for the user for the other selection.
- And then the timeframe selection is available for all the graphs as three different options.
- Based on Year, Month and a selection of which time frame like whether the graph needs to be Daily, Yearly or Monthly.

All Graph Selection:

- In this graph, at first the user selects the chart type that needs to be displayed. By default for all locations a monthly comparison is displayed (Latest year is selected as default and the data we have 2022 as the latest year).

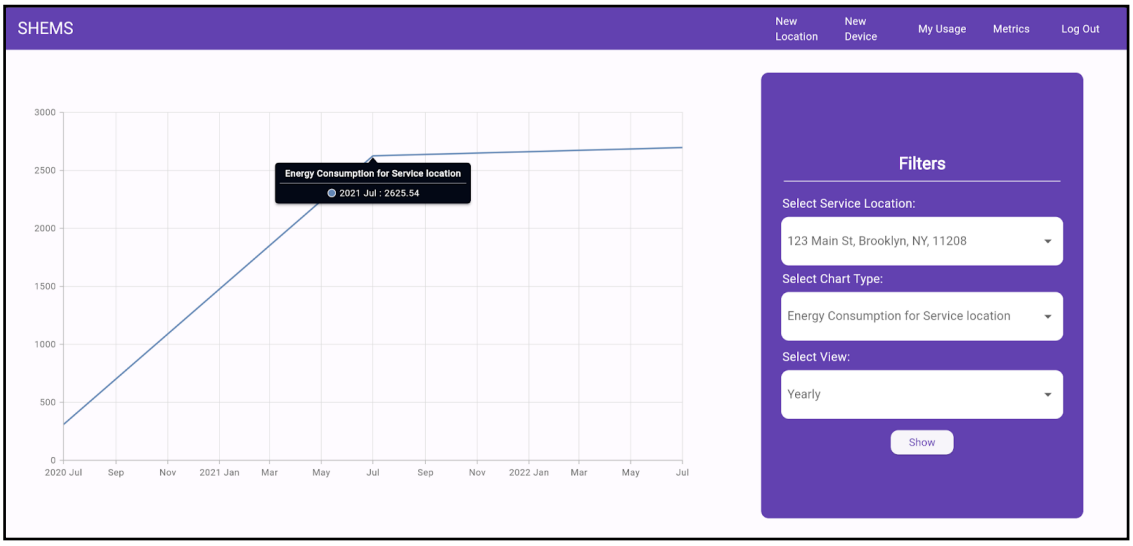
- Since there are 2 locations for this user in our database, a comparison of 2021 and 2022 data is displayed as two lines shown. If there are more locations then each of them are shown as multiple lines in the graph.
- When a yearly graph is selected then a comparison for all locations for both the years are presented.



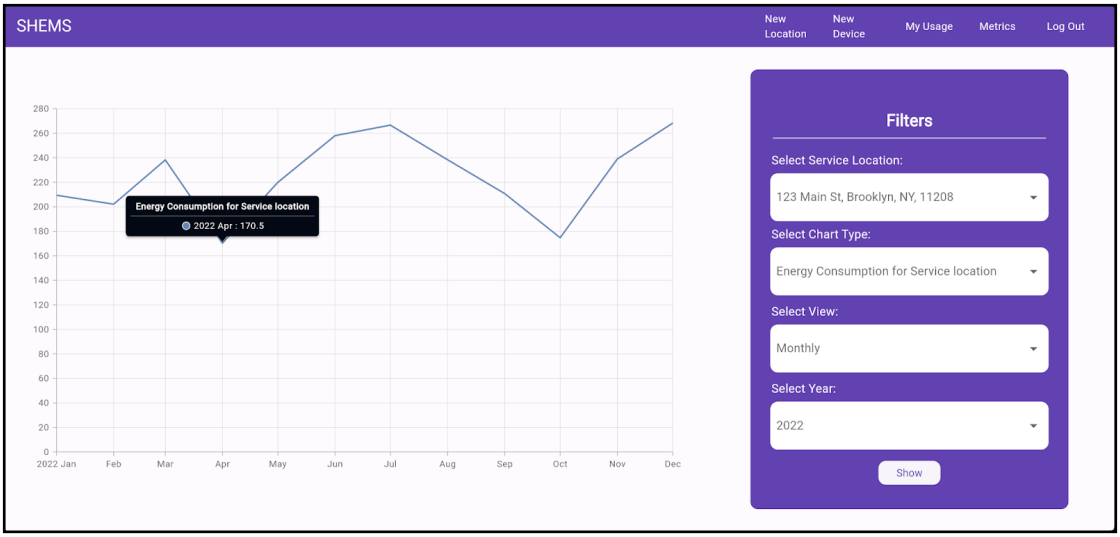
Energy Consumption Graphs

As explained earlier, graphs are shown based on the energy used in a service location. For such graphs user first need to select a location.

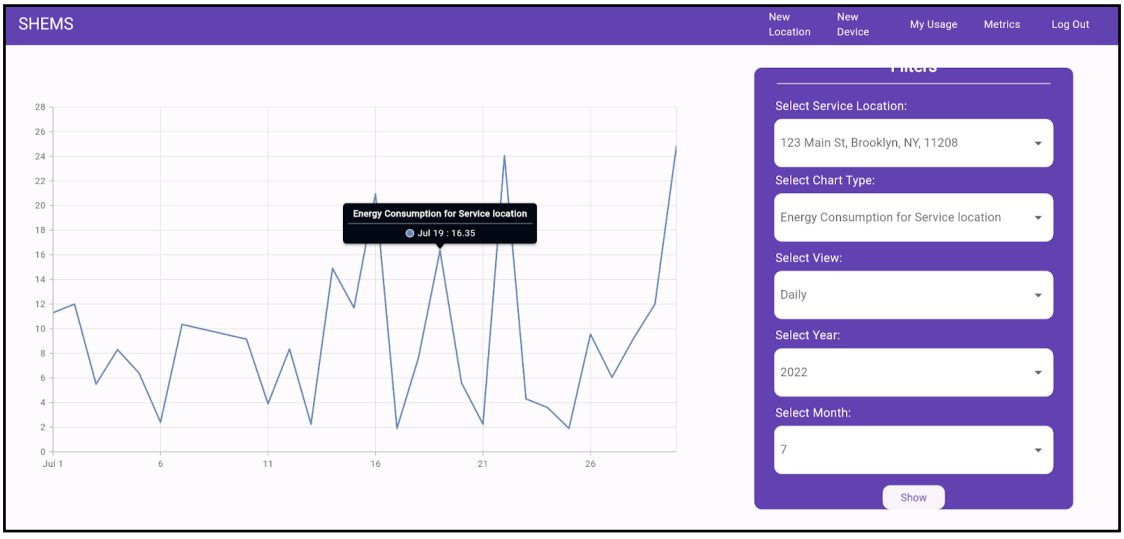
When an yearly is selected as time frame, the year selection as dropdown is then disabled and a graph based on all years is displayed.



When a Monthly view is selected, then year selection is enabled and the user now needs to select a year. Based on the selection, complete monthly analysis is presented as a line plot. The trend of energy consumption is analyzed here.

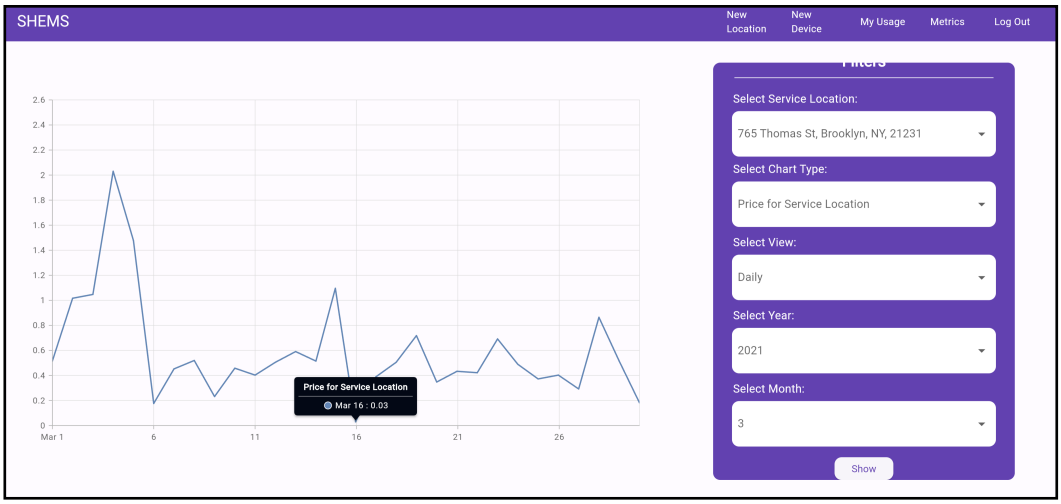


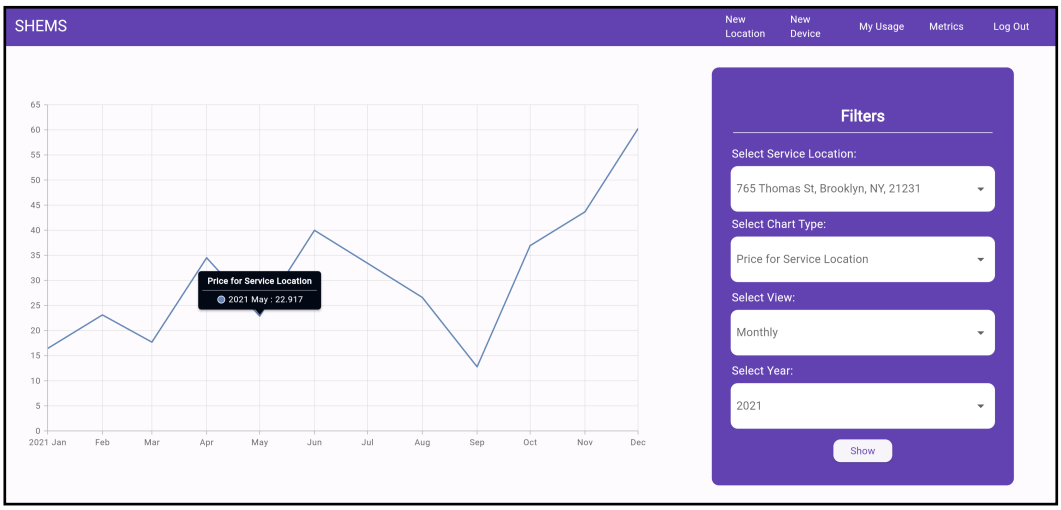
When a daily view is selected, then year and month selection is enabled. Once the user selects a month and year, then we show a graph that represents the daily usage pattern for the month.

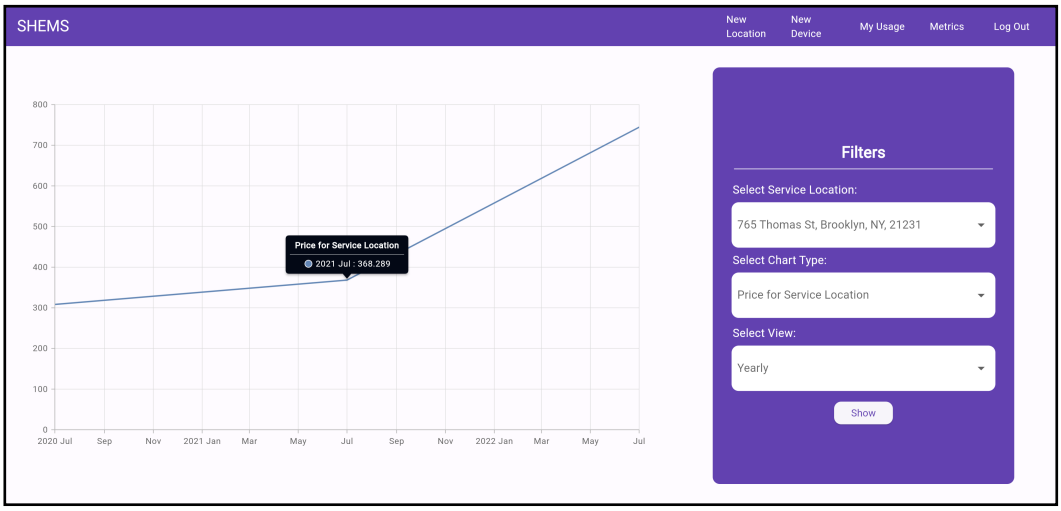


Price Graphs

Similar to the energy consumption graphs, the price associated with each selection is also presented. Note that the selections are similar to the energy consumption. This is a sample of how the graphs are shown in the UI.

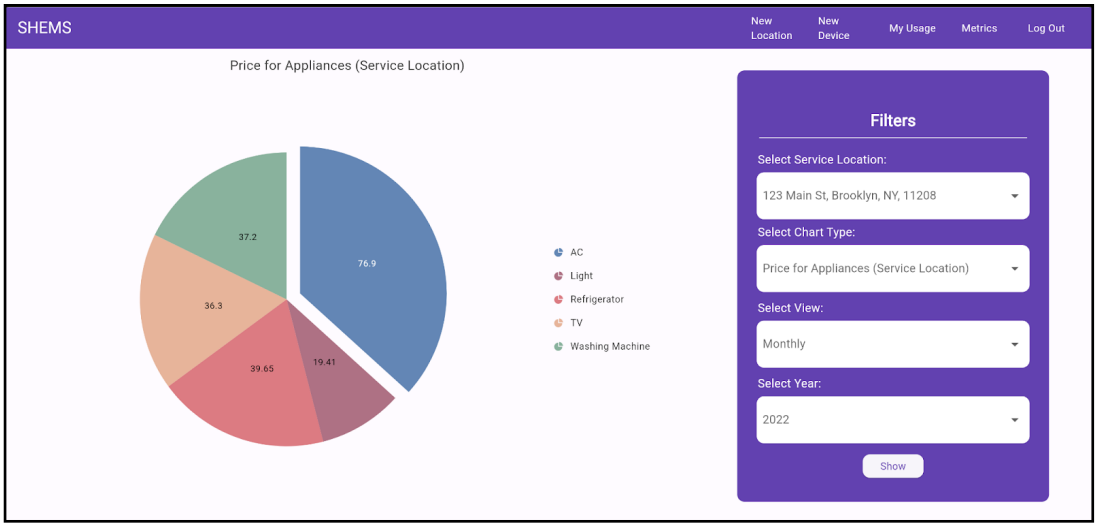




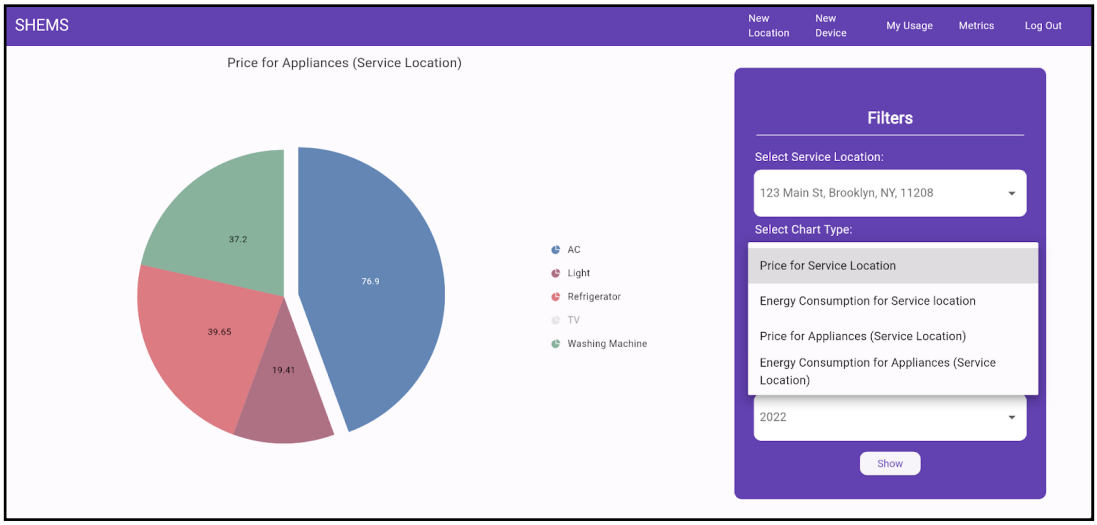


Appliance Based Graphs

For appliance based graphs we are showing Pie charts. Here we make comparisons between appliances. The user is asked to select a year as default, once an year is selected, the consumption of all appliances are marked in the graph and shown. The graphs are made



interactive in such a way that user can click on the appliances in the legend and the graphs excludes these selections and shows for the rest of the appliances.



Backend Explanation:

We used Node.Js as our backend. We've built a REST-API to connect with the client, as we flutter on the frontend which allows us to build the mobile optimized version of the application. We can use the same API for multiple platforms like postman during testing. We used **ngrock** to port the localhost to public so it's easy for us to collaborate as a team. Here are some of the sessions recorded during testing of the application.

```
ngrok

Build better APIs with ngrok. Early access: ngrok.com/early-access

Session Status                online
Account                       Jayanth Anala (Plan: Free)
Version                       3.5.0
Region                        United States (us)
Latency                       24ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://c8e1-173-77-39-80.ngrok-free.app -> http://localhost:8000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              160     0       0.00    0.01    5.22    12.46

HTTP Requests
-------------

POST /u/zipcodemetrics        200 OK
POST /u/wowpage               200 OK
POST /u/zipcodemetrics        200 OK
POST /u/wowpage               200 OK
GET  /u/serviceloc            304 Not Modified
POST /u/zipcodemetrics        200 OK
POST /u/wowpage               200 OK
GET  /u/serviceloc            304 Not Modified
POST /u/zipcodemetrics        200 OK
POST /u/wowpage               200 OK
```

Security Measures:

Measures to guard against SQL injection:

☑Stored procedures and prepared statements (Placeholders)

We never accept raw input from a user and input it directly into your query string. Instead, we used placeholders (?) (or parametrized queries) which would look like this

```
// -- Energy Usage - Daily based on Location --
const DailyUsage = async (req, res) => {
  try{
    var query;
    var {chart, year, month, locID} = htmlspecialchars(xss(req.params));
    if(chart==0){ //price
      query = "select DAY(ud.data_timestamp) as day,sl.locID,\
      SUM(ud.energy_used * ec.price) as y_axis\
      from Usage_Data_Backup ud inner join Device_Enrollment de on ud.deID = de.deID\
      inner join Service_Location sl on sl.locID = de.locID\
      inner join Energy_Cost_Backup ec on ec.zipcode = sl.zipcode and \
      DATE_FORMAT(ec.time_loc, '%Y-%m-%d %H:00:00') = DATE_FORMAT(ud.data_timestamp, '%Y-%m-%d %H:00:00')\
      where Month(ud.data_timestamp) = ? and YEAR(ud.data_timestamp)=?\
      group by day,sl.locID order by day;"
    }else if(chart==1){ //energy
      query = "select DAY(ud.data_timestamp) as day,sl.locID,\
      SUM(ud.energy_used) as y_axis\
      from Usage_Data_Backup ud inner join Device_Enrollment de on ud.deID = de.deID\
      inner join Service_Location sl on sl.locID = de.locID\
      where Month(ud.data_timestamp) = ? and YEAR(ud.data_timestamp)=?\
      group by day,sl.locID order by day;"
    }

    con.query(query,[month,year],async (error, results) => {
```

```
}else if(chart==2){ // Price - Monthly Based on model and location
  query=`select\
  sl.locID as Location_ID,YEAR(ud.data_timestamp) as year_occur,\
  dm.deviceType x_axis,\
  SUM(ud.energy_used * ec.price) as y_axis\
  from Usage_Data_Backup ud inner join Device_Enrollment de on ud.deID = de.deID\
  inner join Service_Location sl on sl.locID = de.locID\
  inner join Device_Models dm on dm.modelID = de.modelID\
  inner join Energy_Cost_Backup ec on ec.zipcode = sl.zipcode and \
  DATE_FORMAT(ec.time_loc, '%Y-%m-%d %H:00:00') = DATE_FORMAT(ud.data_timestamp, '%Y-%m-%d %H:00:00')\
  where  sl.locID=${locID} and YEAR(ud.data_timestamp) = 2022\
  group by Location_ID,year_occur,x_axis order by year_occur,x_axis;`
```

(notice the ? as the placeholder): By using placeholders, the malicious SQL will be escaped and treated as a raw string, not as actual SQL code.

☑Check and sanitize inputs from users before concatenating them into query strings.

Backend:

We are using the htmlspecialchars and xss on the backend to sanitize the inputs we receive from client before passing it to see queries.

```
controllers > userController.js > login > con.query("SELECT * FROM User_Login WHERE email = ?") callback
1   const bcrypt = require("bcrypt");
2   const con = require('../config/dbconfig.js');
3   const jwt = require('jsonwebtoken');
4   const xss = require('xss');
5
6   // ---- Login Routes ----
7   const login = async (req, res) => {
8     try{
9       const { email, password } = xss(req.body);
10      con.query("SELECT * FROM User_Login WHERE email = ?", [email], async (error, results) => {
```

```
await bcrypt.compare(password, results[0].user_password, (err, result) => {
  if (err) { throw (err); }
  else if(result){
    var custID = results[0].custid;
    var token = jwt.sign({custID,email}, process.env.JWT_ACCESS_TOKEN, { expiresIn: '24h' });
    res.json({
      status: true,
      message: "Logged In Successfully",
      data: {"token": htmlspecialchars(token)},
    });
```

Frontend:

On frontend/UI side we are sanitising the input and validating it using various conditions specific to the input field which promises clean data to backend. But we're still sanitizing it on the backend for safety. In the below screenshots you can see the validations like regex and input length in action.

```
child: TextFormField(
  autovalidateMode: AutovalidateMode
      .onUserInteraction,
  validator: (value) {
    if (value!.isEmpty) {
      return 'Please enter some text';
    }
    if (value.length < 6) {
      return 'Password greater than 6 characters';
    }
    return null;
  },
```

```
336                 child: TextFormField(
337                   autovalidateMode: AutovalidateMode
338                       .onUserInteraction,
339                   validator: (value) {
340                     if (value!.isEmpty) {
341                       return 'Please enter some text';
342                     }
343                     if (!RegExp(r'^[a-z]+$')
344                         .hasMatch(value)) {
345                       return 'No Numbers or Special Characters';
346                     }
347                     return null;
348                   },
```

```
child: TextFormField(
  autovalidateMode: AutovalidateMode
      .onUserInteraction,
  validator: (value) {
    if (value!.isEmpty) {
      return 'Please enter some text';
    }
    if (!RegExp(
          r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+")
        .hasMatch(value)) {
      return "Please enter a valid email";
    }
    return null;
  },
```

☑Guard against cross-site scripting, outputs to be returned to user's browsers should be checked or sanitized to avoid scripts

```
await bcrypt.compare(password, results[0].user_password, (err, result) => {
  if (err) { throw (err); }
  else if(result){
    var custID = results[0].custid;
    var token = jwt.sign({custID,email}, process.env.JWT_ACCESS_TOKEN, { expiresIn: '24h' });
    res.json({
      status: true,
      message: "Logged In Successfully",
      data: {"token": htmlspecialchars(token)},
    });
```

```
// --- Add Service Location ----
const addLoc = async (req, res) => {
  try{
    const {loc_type, address_line, city, state, zipcode, date_tookover, area, num_bed, num_occupants} = xss(req.body);
    const custID = req.user.custID;
    con.query("INSERT INTO Service_Location (custID, loc_type, address_line, city, state, zipcode, date_tookover, area,
      if(error){
```

☑**To support concurrent access:**

1. Used Async/Await:

Node.js supports asynchronous programming using the async/await, which can be beneficial when dealing with concurrent operations. We can ensure that our code is written in a non-blocking manner to handle multiple requests simultaneously since we implemented all the routes and transactions using async/await.

```
// --- Get Service Location ----
const getLoc = async (req, res) => {
  try{
    con.query("SELECT * FROM Service_Location WHERE custID=?",[req.user.custID],async (error, results) => {
      if(error){
        res.json({
          status: false,
```

2. Connection Pooling:

We used connection pooling to efficiently manage and reuse database connections. Connection pooling helps handle concurrent requests by reusing existing connections rather than creating a new connection for each request.

```
// --- Add Service Location ----
const addLoc = async (req, res) => {
  try{
    const connection = await pool.getConnection();
    const {loc_type, address_line, city, state, zipcode, date_tookover, area, num_bed, num_occupants} = xss(req.body);
    const custID = req.user.custID;
    con.query("INSERT INTO Service_Location (custID, loc_type, address_line, city, state, zipcode, date_tookover, area,
    });
  }catch(error){…
  }finally {
    connection.release();
  }
}
```

☑Additional Measures:

1. The app.use(cors()) in server.js line adds the CORS middleware to our express application, allowing it to handle cross-origin requests. With allowed listed and blocked lists.

Some queries used for Wow Page and explanation:

Query 1:

```
WITH datas AS (
  SELECT sl.locID AS Location_ID,
    MONTH(ud.data_timestamp) AS Month_occur,dm.deviceType AS device_type,
   SUM(ud.energy_used) AS energy_used,SUM(ud.energy_used * ec.price) AS price_monthly
  FROM
    Usage_Data_Backup ud
    INNER JOIN Device_Enrollment de ON ud.deID = de.deID
    INNER JOIN Service_Location sl ON sl.locID = de.locID
    INNER JOIN Device_Models dm ON dm.modelID = de.modelID
    INNER JOIN Energy_Cost_Backup ec ON ec.zipcode = sl.zipcode AND
DATE_FORMAT(ec.time_loc, '%Y-%m-%d %H:00:00') = DATE_FORMAT(ud.data_timestamp, '%Y-%m-%d
%H:00:00')
  WHERE
      MONTH(ud.data_timestamp) IN (MONTH(NOW() - INTERVAL 1 MONTH), MONTH(NOW() - INTERVAL 2
MONTH))
       AND MONTH(ec.time_loc) IN (MONTH(NOW() - INTERVAL 1 MONTH), MONTH(NOW() - INTERVAL 2
MONTH))
  GROUP BY Location_ID, Month_occur, device_type
  ORDER BY Month_occur, device_type),
combined AS (
  SELECT d1.Location_ID,
    d1.device_type,d1.month_occur AS m1,d2.month_occur AS m2,d1.energy_used AS e1,
    d2.energy_used AS e2 FROM datas d1
 JOIN datas d2 ON d1.Location_ID = d2.Location_ID AND d1.device_type = d2.device_type AND d1.month_occur !
= d2.month_occur )
select c.*,(c.e1 - c.e2) as diff  from combined c where c.Location_ID = 1 order by diff  DESC LIMIT 1;
```

This query is used to extract data of the appliance that caused a major difference in energy consumption, when the last two billings are compared.
We assume that the billing happens at the end of every month, so as a user views the data the previous two billings would be of the last months.

At first the calculate the price consumed by all the appliances in last two months in a CTE, later self join it to get the comparison between months, care is taken to join appliances of same type and different months data are joined. Finally using the above result as another CTE we obtain the record which caused the maximum difference.

Query 2:

```
WITH Ids as (Select locID from Service_Location where zipcode = (select zipcode from Service_Location where
locID = 1)), datas as( select sum(ud.energy_used) as total_energy,
```

```
 month(ud.data_timestamp) as month_occur, sl.locID as locID, sl.zipcode as zipcode
 from Usage_Data_Backup ud inner join Device_Enrollment de on ud.deID = de.deID
inner join Ids ids on de.locID = ids.locID inner join Service_Location sl on sl.locID = ids.locID
where  YEAR(ud.data_timestamp) = 2022 and month(ud.data_timestamp) = MONTH(NOW() - INTERVAL 1 MONTH)
group by sl.locID,month_occur,zipcode order by locID,month_occur)
select d1.locID as loc1,d1.total_energy as loc_energy,sum(d2.total_energy) as total_energy,
d1.total_energy * 100 /sum(d2.total_energy)  as percent_used
from datas d1 join datas d2
where d1.locID = 1 group by d1.locID,loc_energy
```

This query is used to calculate the average percentage of energy used by a user in the last billing month when compared to others in the same zipcode. We first extract all users present in the zip code and store it in a CTE .Since price varies based on the time of consideration, we first calculate the total price for all the users in that zip code and store it in another CTE. Now we perform self join to get the comparison, care is taken to avoid same record join on one side and also ignore all other records except the main user on the other hand.
Finally we calculate the percentage of usage.

* Work for Extra Credit:
    1. The wow page mentioned above and some more interesting graphs.
    2. The app is mobile optimized as well, It is completely usable from a mobile phone.