B. Tech Project: Android App for Power-plants

Prahlad Chaudhary (17085054), Sarthak Choudhary (17085066), Shashank Kumar Singh (17085068), Vishal Indora (17085076)
Supervised by: Dr. Soumya Ranjan Mohanty

1. Abstract

Various electrical parameters in a power plant are calculated through smart energy meters. These meters upload the data to a central server (computer), which performs calculation and displays the required parameters to the user through a web-app.

The developed app aims to replicate, or mimic this system, essentially adding portability to the present system, allowing a user to access the data from possibly anywhere. The various parameters calculated within the app include charges for each time block, the total charges for a day, and the additional penalties if there is sustained deviation for more than 12 consecutive time blocks.

Finally, the app makes use of polynomial regression to predict future scheduled generation on the basis of past actual generation as a means to minimize the difference between the two, and consequently, to minimize costs.

2. Background

Electrical power is supplied from sellers to buyers. This distribution of power is determined by a number of constraints based on price and other parameters. Two important parameters are the actual generation (AG) and the scheduled generation (SG).

Sellers generate electricity as per a schedule dividing a day into 96 blocks of 15 minutes each. However, the buyer can demand more than the scheduled amount of power. This leads the seller to incur a loss, consequently slowing the generator blades as mechanical power is converted to electrical power, this loss must be compensated. Alternatively, since the consumer used more electricity than they were allotted, they should pay extra.

The charge for deviation for all time blocks are payable for overdrawal by the buyer and underinjection by the seller; Similarly, charges are receivable for underdrawal by the buyer and over-injection by the seller. However, the seller can choose to overinject as much as he wants; or the buyer can underdraw as much as he wants to get paid more. As a result, charges for deviation in excess of min(12% of SG, 150MW) are 0.

3. Terms Used

Terms	Meaning
Time block	Each day is divided into 96 time blocks of 15 minutes each
Scheduled Generation (SG)	It refers to the power which is scheduled to be provided by generators
Actual Generation (AG)	It refers to the power which is actually consumed by the buyers or provided by the sellers
Declared Capability	It is a measure of the contribution that a power station makes to the overall capacity of a distribution grid.
Android WebView	Built in browser in android SDK.
DOM	Document Object Model

Table 1: Terms used

4. Working of the App

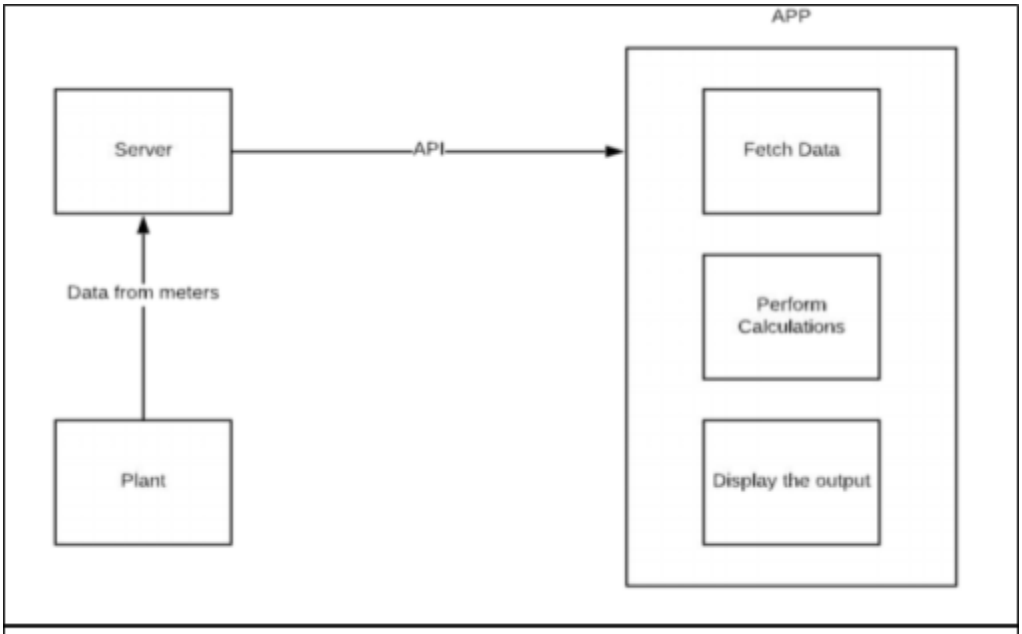


Figure 1: Basic working of the app

The installed smart meters read the relevant data and upload it on a central server in the powerplant through a common network connection, say WiFi or Ethernet. The server (computer) stores this data in its database.

The server features a bunch of APIs (Application Programming Interfaces) which are used to exchange data between the web-app and master server. The data exchange takes place through these APIs, and the web-app displays the data when requested.

5. Calculations

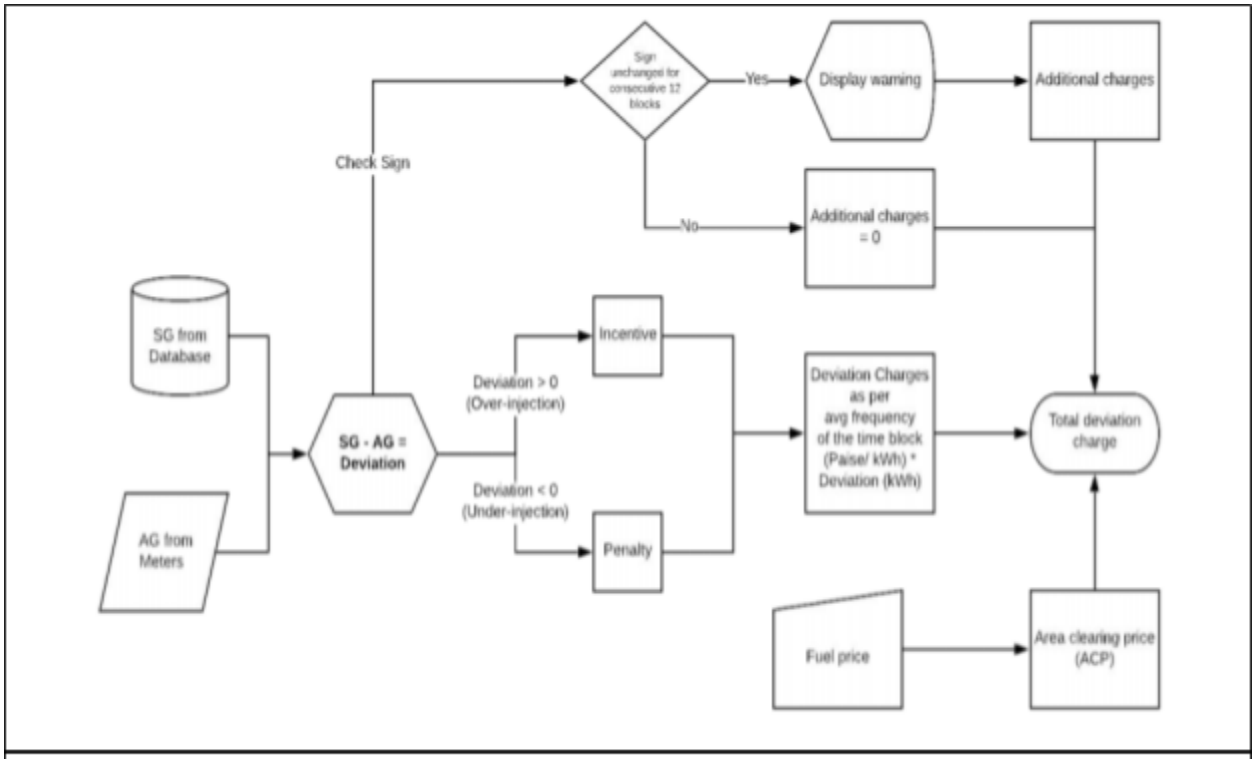


Figure 2: Calculations for deviation charge

Frequency(Hz)	UI Rate	DC(MW)	SG(MW)	AG(MW)	DC-SG(MW)	AG/SG(%)	Deviation(MW)	Deviation(%)	Deviation Charge	...	Total Deviation & Additional Charge	Fuel Cost	Net Gain/Net Loss	Opportunity Dev
49.92	303.040	548.5	311.31	297.54	237.19	95.576756	-13.77	-4.42	-10432.15	...	-10432.0	10844.0	412.0	-39.313631
50.02	238.904	548.5	311.31	293.54	237.19	94.291863	-17.77	-5.71	-10613.31	...	-10613.0	13994.0	3381.0	-39.313631
50.01	238.904	548.5	311.31	294.81	237.19	94.699817	-16.50	-5.30	-9854.79	...	-9855.0	12994.0	3139.0	-39.313631
49.98	303.040	548.5	311.31	296.20	237.19	95.146317	-15.11	-4.85	-11447.34	...	-11447.0	11899.0	452.0	-39.313631
49.93	303.040	548.5	311.31	296.12	237.19	95.120619	-15.19	-4.88	-11507.94	...	-11508.0	11962.0	454.0	-39.313631
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
49.98	303.040	548.5	505.22	486.39	43.28	96.272911	-18.83	-3.73	-14265.61	...	-14266.0	14829.0	563.0	-50.521615
49.97	303.040	548.5	505.22	489.26	43.28	96.840980	-15.96	-3.16	-12091.30	...	-12091.0	12569.0	477.0	-50.521615
49.97	303.040	548.5	505.22	487.35	43.28	96.462927	-17.87	-3.54	-13538.31	...	-13538.0	14073.0	534.0	-50.521615
49.95	303.040	548.5	505.22	485.81	43.28	96.158109	-19.41	-3.84	-14705.02	...	-14705.0	15285.0	580.0	-50.521615
50.01	238.904	548.5	505.22	483.87	43.28	95.774118	-21.35	-4.23	-12751.50	...	-12752.0	16813.0	4062.0	-50.521615

Figure 3: Various parameters that are calculated

6. Hosting the Data on Server

6.1 Server Setup

Since the application is completely dependent on server side REST APIs for data so we have managed to replicate the present power-plant system. A RESTful API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating and deleting of operations concerning resources respectively.

We've hosted the data on an EC2 instance on AWS (Amazon Web Services), An EC2 instance is nothing but a virtual server in Amazon Web services terminology. It stands for Elastic Compute Cloud. The following image shows the AWS Dashboard and the instance on which the data is being stored and fetched.

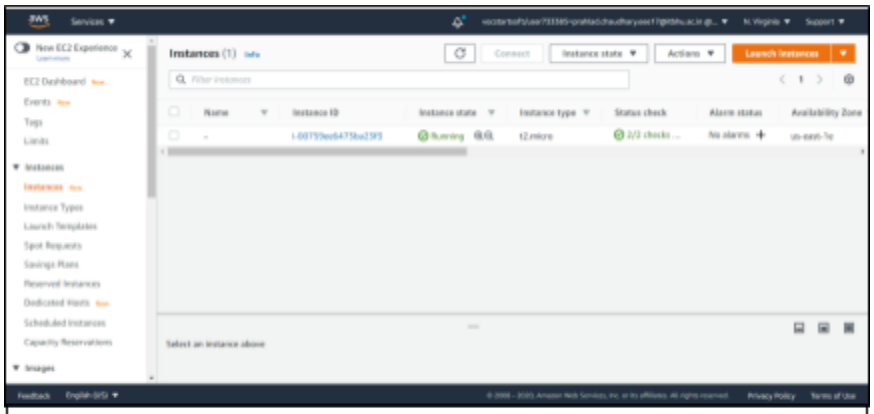


Figure 4: AWS Dashboard

Currently, the AWS server is accessed by the android app by contacting the REST APIs. While running the app at the powerplant, it can be easily modified to access data from the powerplant's server APIs, while maintaining the same functionality.

The data hosted on EC2 is randomized within the respective ranges and dynamically updated for every time block (15-minute interval) and any required calculations are performed on the server itself. This frees up the resources on the user's device and makes the app faster.

We used Python and Django REST framework to build the server side application. The Django REST framework is a powerful and flexible toolkit for building Web APIs, and has a bunch of REST APIs responsible for healthy interaction of data between server and Android Application.

6.2 Generating the Data

Ranges for various parameters were known and pseudorandom number generators (present in every programming language) were used to generate the data.

The objective is to update variables in real time for each time block. This is trivial if this code is run constantly as we can just use a while loop to do so. However, due to the limits of the server, the entire code is run every time the data is accessed, which makes it complex.

6.2.1 Idea

The random function present in every programming language is actually a pseudorandom number generator. What this means is that if we give it a certain input, it can only return a certain output. As an example, it can be thought of as the function f(x) = x+1: if we give 1 as an input it can only give 2 as an output.

Along similar lines, if we give an input to the random function that stays the same for a given time block and changes for another time block, we will be able to generate the required variables. This idea is utilized in the following approach.

6.2.2 Approach

Every time someone accesses the data on the server, the following process takes place:

- The only input taken by the server is the current date and time which is taken from an internal clock present on every server (and on every device). On the basis of the current time, we find out the current block number and the current block time. Let's call them the parent variables.
- On the basis of the parent variables a function calculates the previous or next block time as needed. These functions can calculate the previous or next block time given a block time as an input. In this way, we can calculate the block time for 2 blocks ahead by simply using the next block function two times. By using this function multiple times we can calculate the block time for a block that is an arbitrary number of time blocks ahead or behind.

The seed value or the input value for the random function is calculated using functions which have the inputs block no., day and month. The day and month are directly accessed using the server clock and the block no. has been calculated using the method discussed above.

The function which returns the seed value for the variables that are updated in every time block takes input as block no., day and month. Now, all three of these values cannot be the same for a block. We require a function which can map these three values to a single value which is unique. One might notice that it is impossible that the same value for a certain parameter does not repeat over a month or a year. This is allowed to happen in our approach using the random function itself. Although rare, the function allows multiple seed values to give a certain output remarkably reflecting real world conditions.

The seed function is inspired from the Rabin-Karp string matching algorithm.
Since the maximum values of the three variables block no., day and month are 96, 31 and 12, the highest of these is chosen. Now, we're using this value as a 'base' and exponentiating, which is exactly how we calculate values in the common number system. The function is as follows:

```
def function_seed(blk_no, day, mon):
    return (96 * 96) * blk_no + 96 * day + mon
```

We can imagine that we are using a number system of base 96, and if we just input any number in the first, second and third place we'll obtain a unique number.

To illustrate this even further, consider the normal base 10 representation: to obtain any three digit number, we can arbitrarily choose three numbers from 0 to 9. Suppose 123 is chosen.

$$123 = (10 * 10) * 1 + 10 * 2 + 3$$

This is exactly similar to the function above with the only exception being the change of base. Similarly, the function which updates the parameters which need to be changed every 24 hours takes in only day and month as an input. The maximum of these is 31 (for days) and consequently a base of 31 is chosen. The rest of the procedure is the same as above.

6.3 APIs

The various REST APIs available on the AWS server are:

6.3.1 Powerplant Data: [Link]

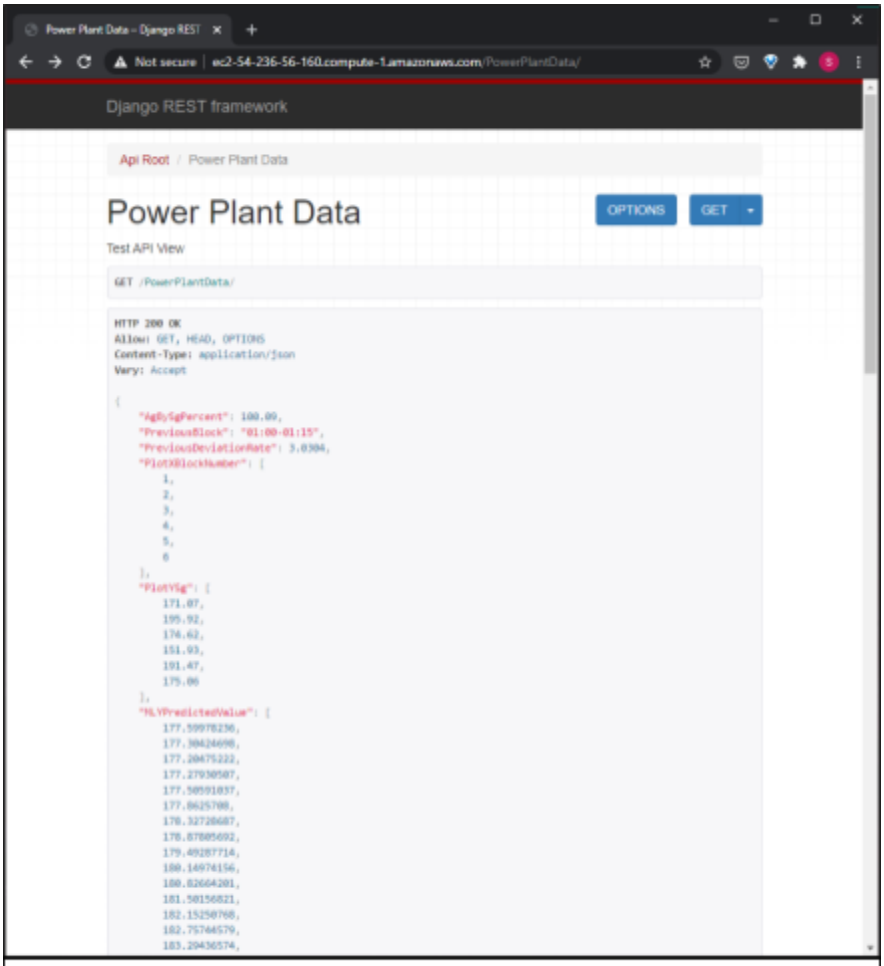


Figure 5: Powerplant data API

6.3.2 Time Data: [Link]

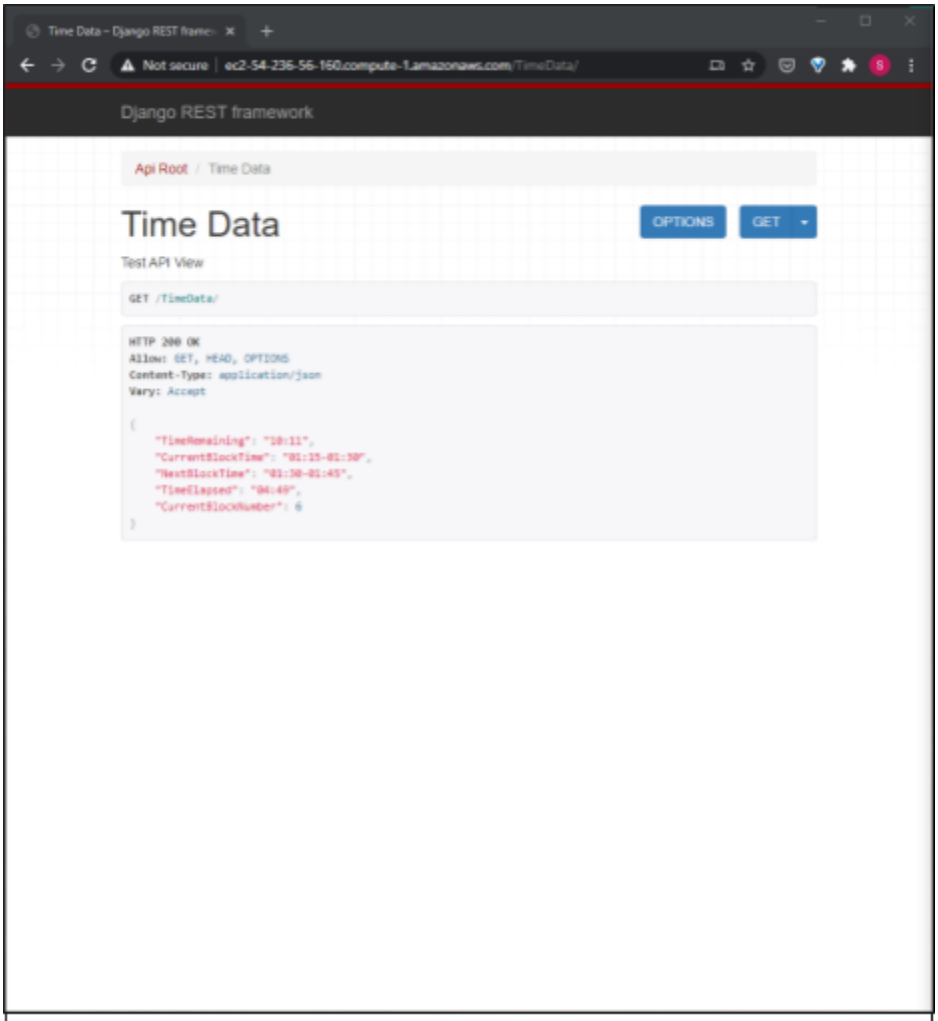
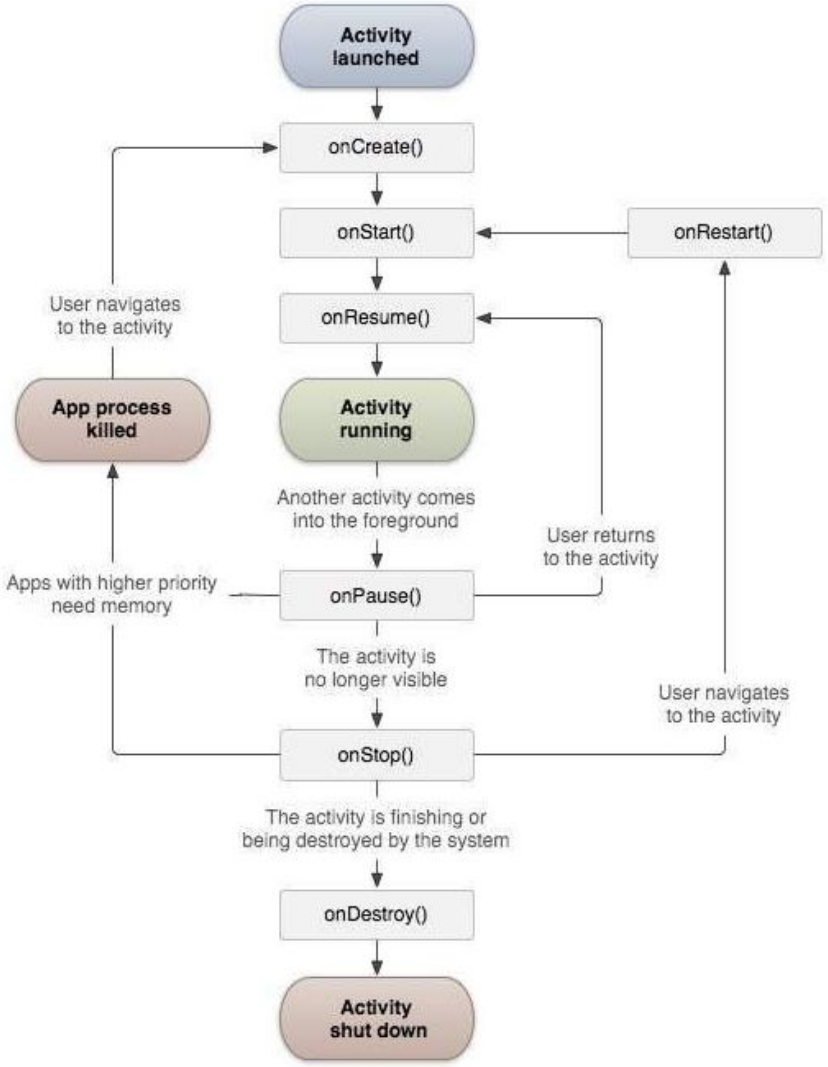


Figure 6: Time Data API

7. Developing the App

7.1 Android activities

An activity provides a window in which the app draws its UI. The Android system initiates its program within an Activity starting with a call on onCreate() callback method (or function). There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the following activity life cycle diagram:



7.2 Fetching the data

For fetching the data from the apis, we have used the Retrofit Client Library. This is a built-in library for making network calls over the web and getting data in the form of JSON format. We only needed to add the dependency for the Retrofit Library (similar to import statement in programming).

7.3 Plotting the graph

For plotting we have used the MPAndroidChart library. It provides functions (or methods) to easily create various kinds of graphs. Additionally, it also offers various customization options: layouts, colours, etc. We stored the X-values and the Y-values in the form of arrays, and used various in-built methods to plot the graph.

7.4 Additional Features

The following additional features were also implemented in the app:

- Alarm: An alert will be shown if there is sustained deviation in one direction for more than 11 time blocks.
- Prediction model: A machine learning model to predict the actual generation is also implemented

8. Model to predict actual generation (AG)

8.1 Data

Hourly electricity consumption data of a regional transmission organization (RTO in the US was used (link). The data was downloaded as a CSV file with each row having two values: datetime and the consumption in MW. The data was cleaned (by removing the days that didn't have 24 data points), resulting in a total data of 209 days in 5022 data points. Since the value of energy consumption (as per our needs) should lie between 150MW to 200MW. The data was scaled to lie within this range.

Datetime	AEP_MW
2004-12-31 1:00:00	13478
2004-12-31 2:00:00	12865
2004-12-31 3:00:00	12577
2004-12-31 4:00:00	12517
2004-12-31 5:00:00	12670
2004-12-31 6:00:00	13038
2004-12-31 7:00:00	13692
2004-12-31 8:00:00	14297
2004-12-31 9:00:00	14719
2004-12-31 10:00:00	14941
2004-12-31 11:00:00	15184
2004-12-31 12:00:00	15009
2004-12-31 13:00:00	14808
2004-12-31 14:00:00	14522
2004-12-31 15:00:00	14349

Figure 7: Data for prediction model

8.2 Observations (from data)

First, we plotted the energy consumption for a couple of days to check if there's a pattern. We noticed that the data resembles a polynomial function with peaks around the 40th and the 80th block. Consequently, we decided to implement a polynomial regression model for prediction.

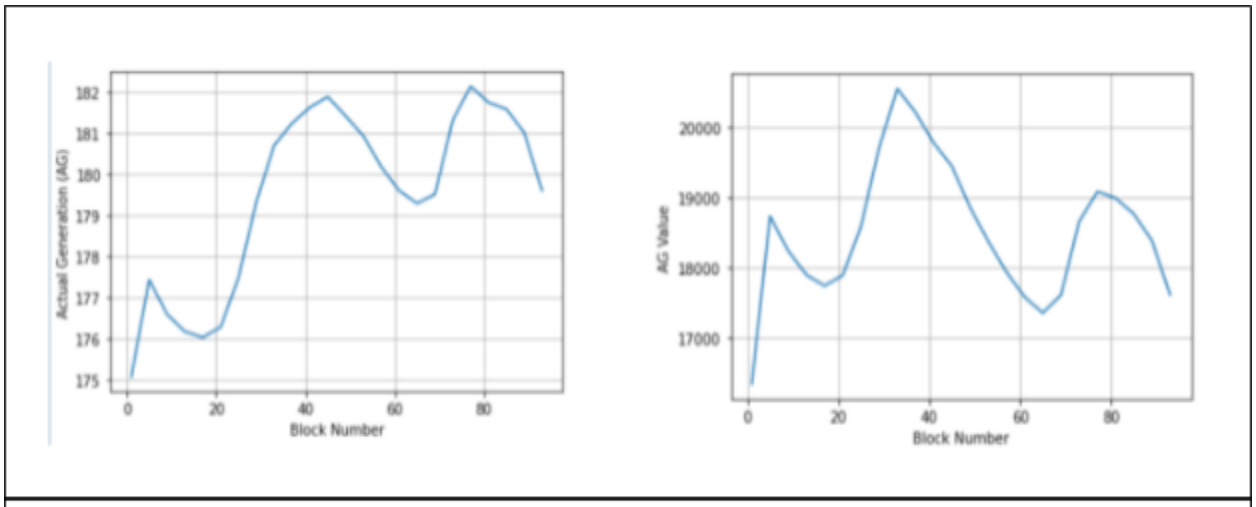


Figure 8: Observations from data

8.3 Applying Polynomial Regression

We have data in the form of x_i (input data) and y_i (output data). We are required to predict a relationship between them. Suppose we have n such points (x_i, y_i).

To predict a first-degree polynomial using these points, we'll consider the predicted polynomial as follows:

$$\bar{y} = (\beta_0 + \beta_1 x_i)$$

Here, β_0, β_1 are the parameters to be determined. To determine these parameters, we first get the cost function J as follows:

$$J = (\bar{y} - y_i)^2 = \sum_{i=1}^{n} (\beta_0 + \beta_1 x_i)^2$$

The cost function simply takes the squared error of our predicted value from the actual values. In order to have good predictions, it is required that the value of the cost function is minimum. For this, we partially differentiate the cost function with respect to each of the unknowns as follows:

$$\frac{\delta J}{\delta \beta_0} = 0 \qquad \frac{\delta J}{\delta \beta_1} = 0 \qquad \frac{\delta J}{\delta \beta_2} = 0$$

$$\beta_0 n + \beta_1 \sum x_i = \sum y_i \quad ...(1)$$

$$\beta_0 \sum x_i + \beta_1 \sum x_i^2 = \sum y_i x_i \quad ...(2)$$

Combining equations (1) and (2) in matrix form, we get

$$\begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum {x_i}^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \end{bmatrix}$$

$$\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum {x_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum y_i \\ \sum y_i x_i \end{bmatrix}$$

Along similar lines, if we want to predict using a quadratic polynomial of the following form:

$$\bar{y} = (\beta_0 + \beta_1 x + \beta_2 x^2)$$

We will partially differentiate with respect to β_0, β_1, and β_2. We'll obtain three equations which will then be combined into the following matrix form:

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix}$$

or

$$\boxed{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix}^{-1} \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix}}$$

Based on the previous discussion, we can form the following generalization for a k-degree polynomial prediction function:

1. The cost function is obtained by taking the square of the difference of the predicted value from the actual value.
2. The cost function is differentiated with respect to $\beta_0, \beta_1 \ldots \beta_k$ to obtain (k+1) equations.
3. These equations are then combined into a single matrix equation and the value of the β matrix is found by solving the equation by taking inverse.

In general, we have the following:

$$\begin{bmatrix} \beta_0 \\ \dots \\ \beta_k \end{bmatrix} = \begin{bmatrix} n & \dots & \sum x_i^k \\ \dots & \dots & \dots \\ \sum x_i^k & \dots & \sum x_i^{2k} \end{bmatrix}^{-1} \begin{bmatrix} \sum y_i \\ \dots \\ \sum x_i^k y_i \end{bmatrix}$$

8.4 Predictions

We splitted the data into training and testing sets. The function for polynomial regression was trained using the training set. A split of 40% training data to 60% testing data gave the most desirable results. On further increasing the size of training data, the function lost the ability to generalize (due to overfitting). Finally, the prediction was done on the testing set.

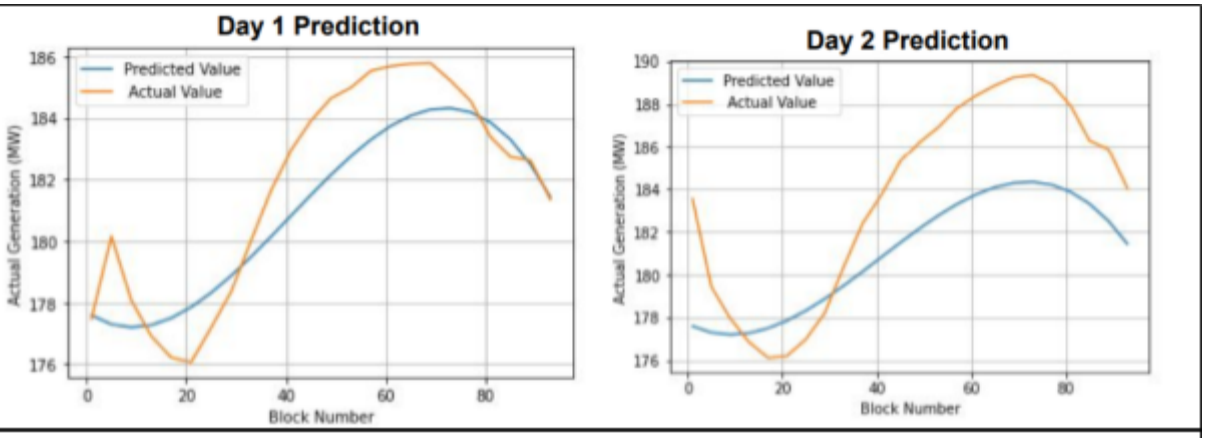


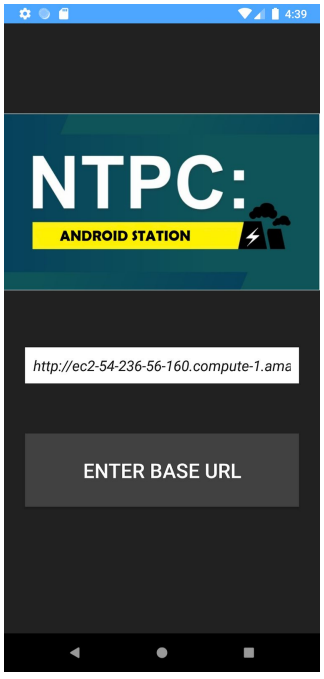
Figure 9: Predictions

9. Demonstration of the App

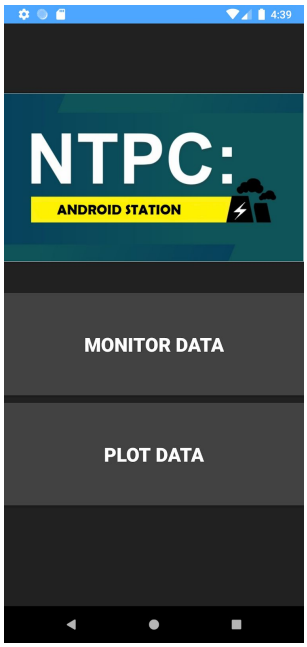
This section shows the app in action:

9.1 Loading screen: On launching the app, the following loading screen appears. The necessary resources are loaded in the background and the user is redirected to the base URL screen(next).

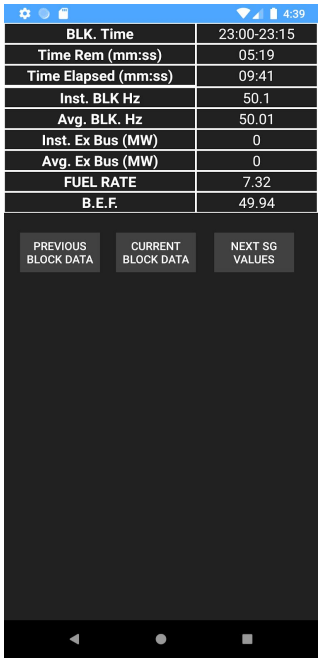
9.2 Enter Base URL: this provides the user the option to enter the address of the server (where the data is stored).



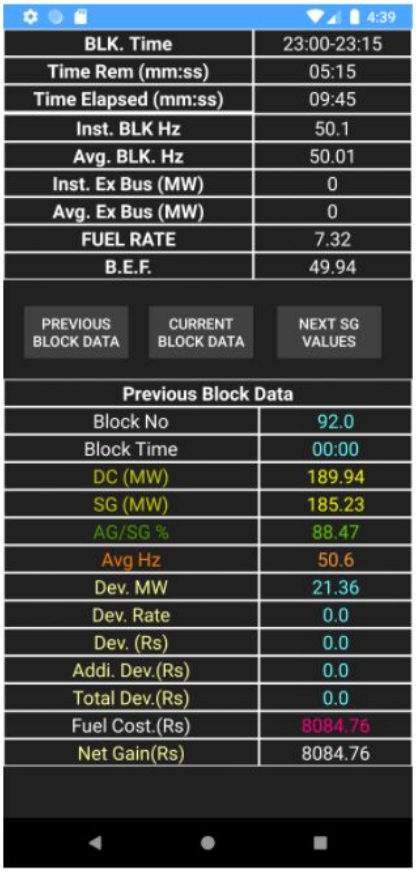
9.3 Main Menu: On adding a valid URL, the user is taken to the following screen where he can either choose to see the graph of the prediction model, or view the data.



9.4 Monitor Data: This screen shows the data for the current time block. There are also three tabs below which, upon clicking, show the data for the previous, (additional data) for the current and SG values for the next four blocks. We've used mock data, i.e. we've used randomized values based on appropriate ranges for each data.



9.5 Monitor Data (tabs): the following shows the various data parameters that are available on clicking the three tabs.

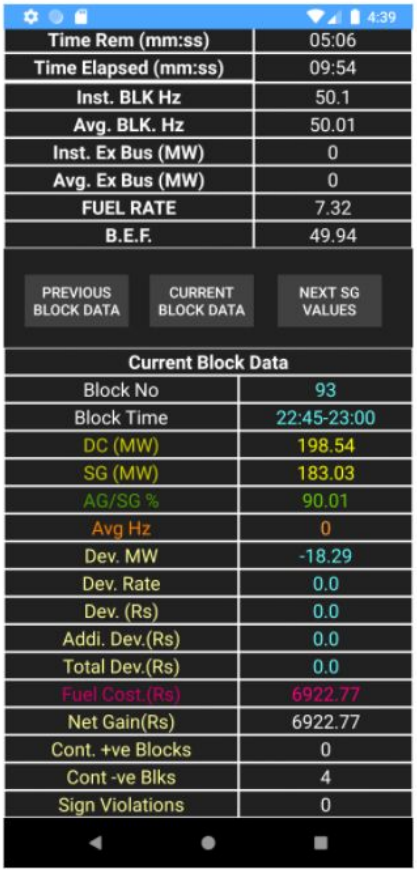


BLK. Time	23:00-23:15
Time Rem (mm:ss)	05:15
Time Elapsed (mm:ss)	09:45
Inst. BLK Hz	50.1
Avg. BLK. Hz	50.01
Inst. Ex Bus (MW)	0
Avg. Ex Bus (MW)	0
FUEL RATE	7.32
B.E.F.	49.94

PREVIOUS BLOCK DATA | CURRENT BLOCK DATA | NEXT SG VALUES

Previous Block Data

Block No	92.0
Block Time	00:00
DC (MW)	189.94
SG (MW)	185.23
AG/SG %	88.47
Avg Hz	50.6
Dev. MW	21.36
Dev. Rate	0.0
Dev. (Rs)	0.0
Addi. Dev.(Rs)	0.0
Total Dev.(Rs)	0.0
Fuel Cost.(Rs)	8084.76
Net Gain(Rs)	8084.76

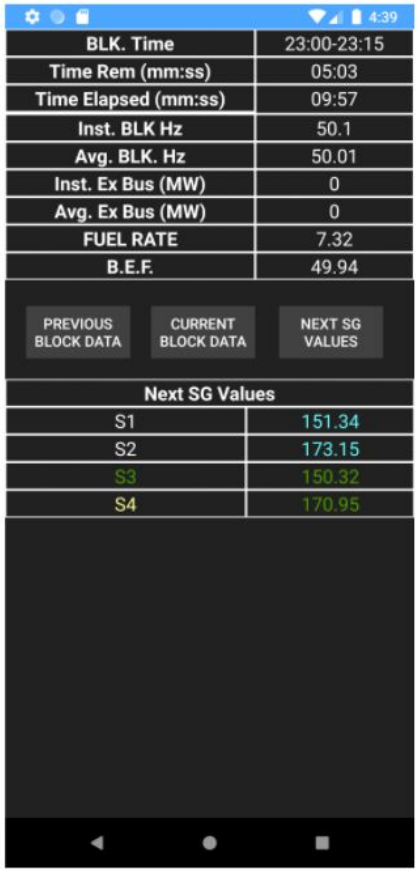


Time Rem (mm:ss)	05:06
Time Elapsed (mm:ss)	09:54
Inst. BLK Hz	50.1
Avg. BLK. Hz	50.01
Inst. Ex Bus (MW)	0
Avg. Ex Bus (MW)	0
FUEL RATE	7.32
B.E.F.	49.94

PREVIOUS BLOCK DATA | CURRENT BLOCK DATA | NEXT SG VALUES

Current Block Data

Block No	93
Block Time	22:45-23:00
DC (MW)	198.54
SG (MW)	183.03
AG/SG %	90.01
Avg Hz	0
Dev. MW	-18.29
Dev. Rate	0.0
Dev. (Rs)	0.0
Addi. Dev.(Rs)	0.0
Total Dev.(Rs)	0.0
Fuel Cost.(Rs)	6922.77
Net Gain(Rs)	6922.77
Cont. +ve Blocks	0
Cont -ve Blks	4
Sign Violations	0



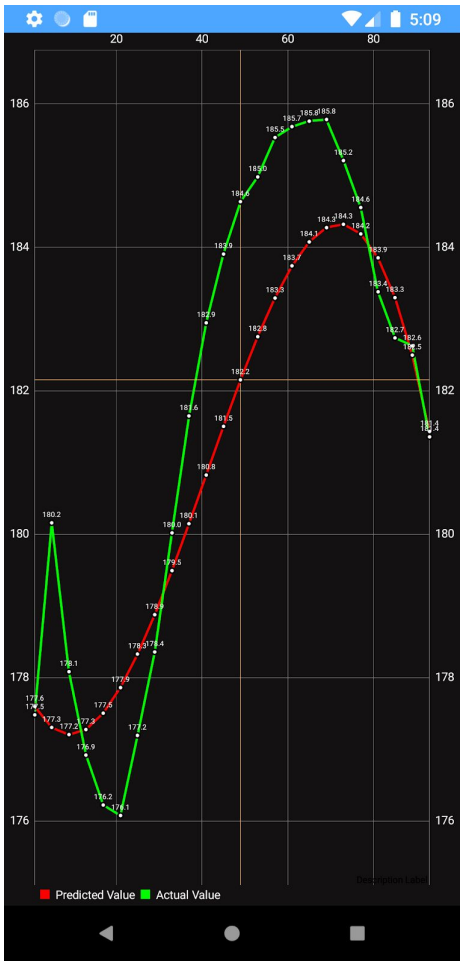
BLK. Time	23:00-23:15
Time Rem (mm:ss)	05:03
Time Elapsed (mm:ss)	09:57
Inst. BLK Hz	50.1
Avg. BLK. Hz	50.01
Inst. Ex Bus (MW)	0
Avg. Ex Bus (MW)	0
FUEL RATE	7.32
B.E.F.	49.94

PREVIOUS BLOCK DATA | CURRENT BLOCK DATA | NEXT SG VALUES

Next SG Values

S1	151.34
S2	173.15
S3	150.32
S4	170.95

9.6 Graph: on clicking plot data (in the menu activity), we get the graph for the prediction model as follows:



10. References

10.1 Constraints and protocols for calculations of various parameters

- Letter to Generators SCED (Dated 18 April 2019) - POSOCO
- DSM 5th Amendment (Dated 28 May 2019) - Central Electricity Regulatory Commission
- Notification 132 (Dated 6 January 2014) - Central Electricity Regulatory Commission

10.2 Polynomial Regression Model

- Eva Ostertagová, Modelling using Polynomial Regression, Procedia Engineering, Volume 48, 12012, Pages 500-506, ISSN 1877-7058, https://doi.org/10.1016/j.proeng.2012.09.545.