



Faculty of Engineering and Applied Science

SOFE 4610U Design & Analysis of IoT Software

Assignment 2

Group #3

Group Member 1

Name: Sarthak Sharma

Student ID: 100604428

Group Member 2

Name: Adam Wong Chew Onn

Student ID: 100598499

Group Member 3

Name: Mitul Patel

Student ID: 100700131

Group Member 4

Name: Matthew English

Student ID: 100704553

10 Step Approach

1.Purpose & Requirement Specification:

Purpose: The purpose of our project is to create a smart cooling system.

Behaviour: The system will be able to sense the temperature and decide whether the fan is turned on or off.

System Management Requirement: The system should provide local monitoring and control functions.

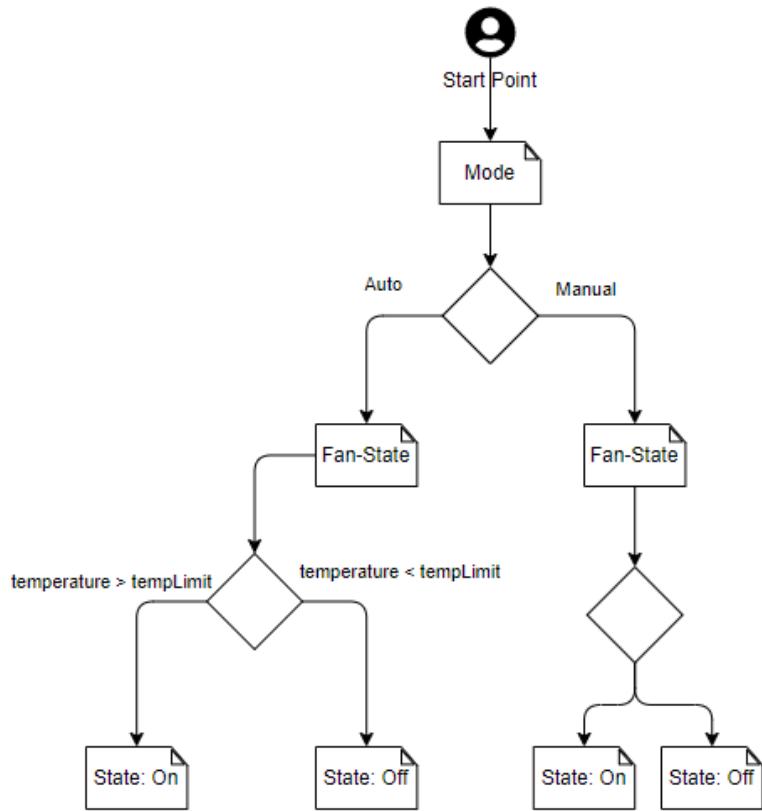
Data Analysis Requirement: The system must read and display the active temperature to the user

Application Deployment Requirement: The application should be deployed locally with a

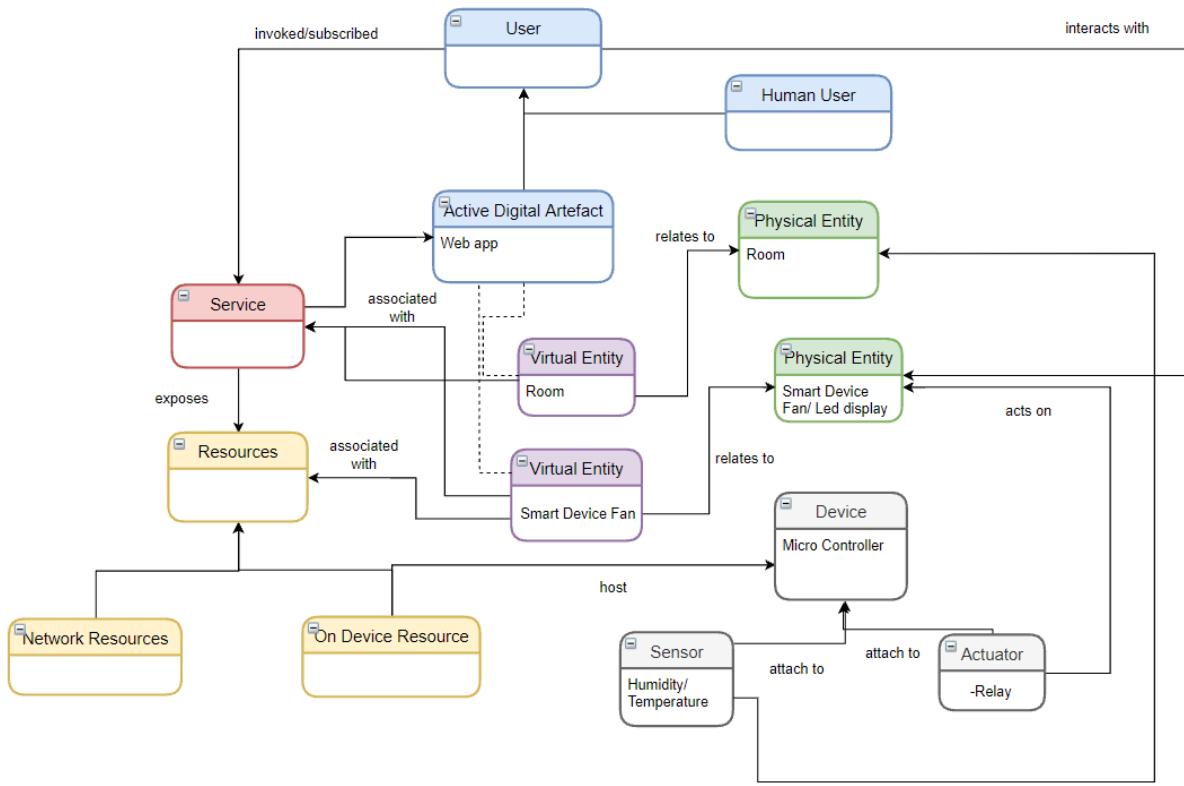
User Interface Requirement: The user should be able to see the current temperature and manually turn the fan on and off

Backend Requirement: The data should be set from the microcontroller to the django backend using a post request. Furthermore, the application should display temperature data on the app.

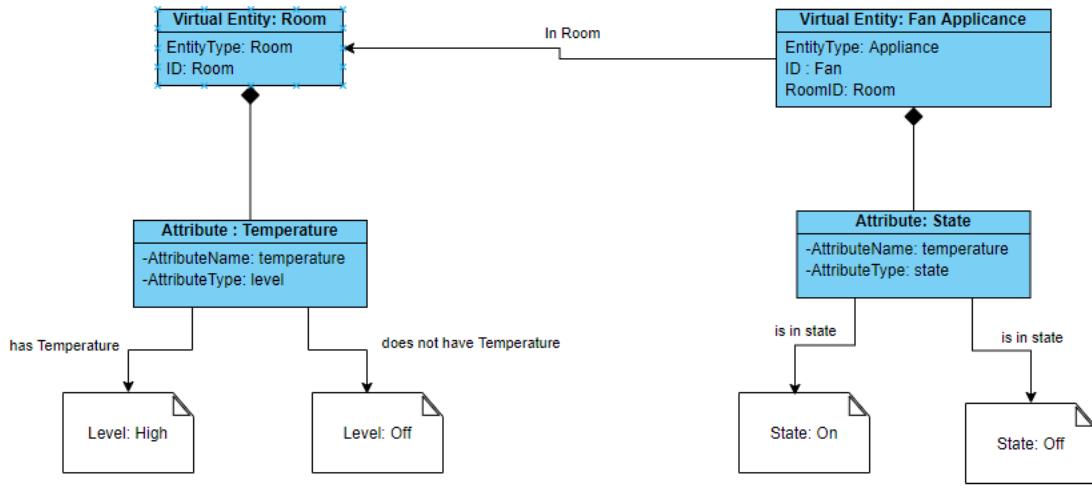
2.Process Specification:



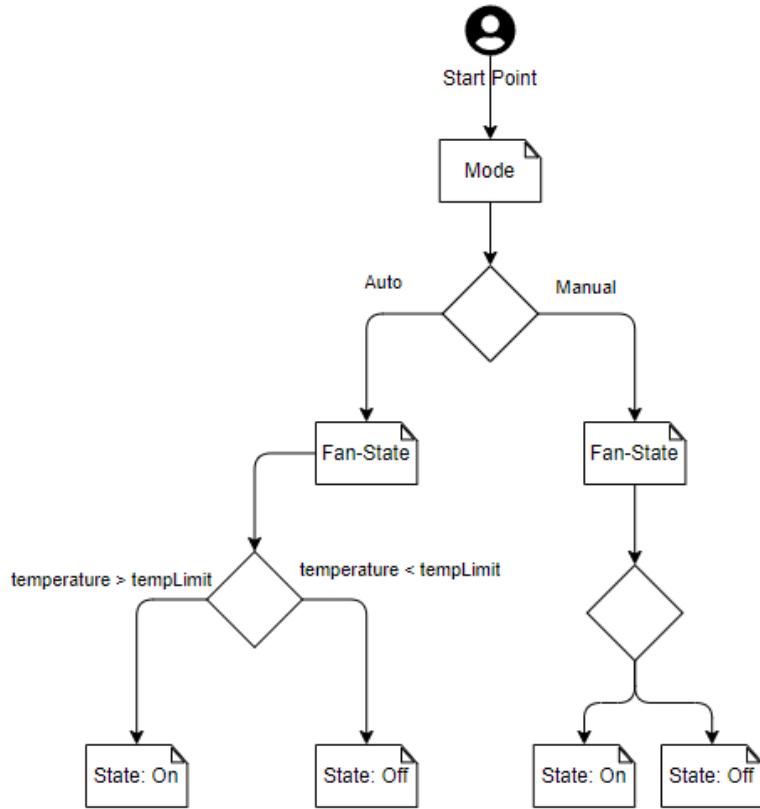
3.Domain Model Specification:



4. Information Model Specification

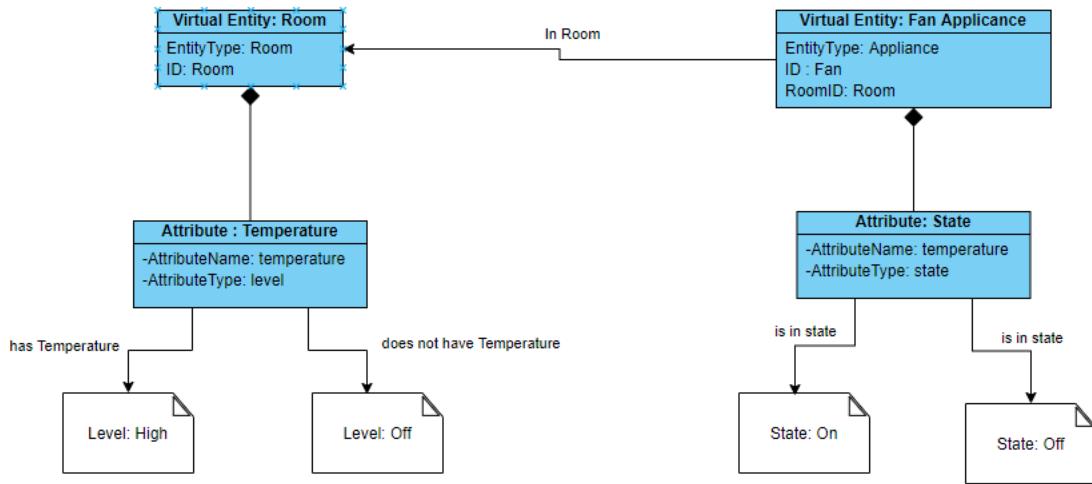


5.Service Specifications:



Mode Service allows the user to manually turn the fan on and off instead of the auto mode.

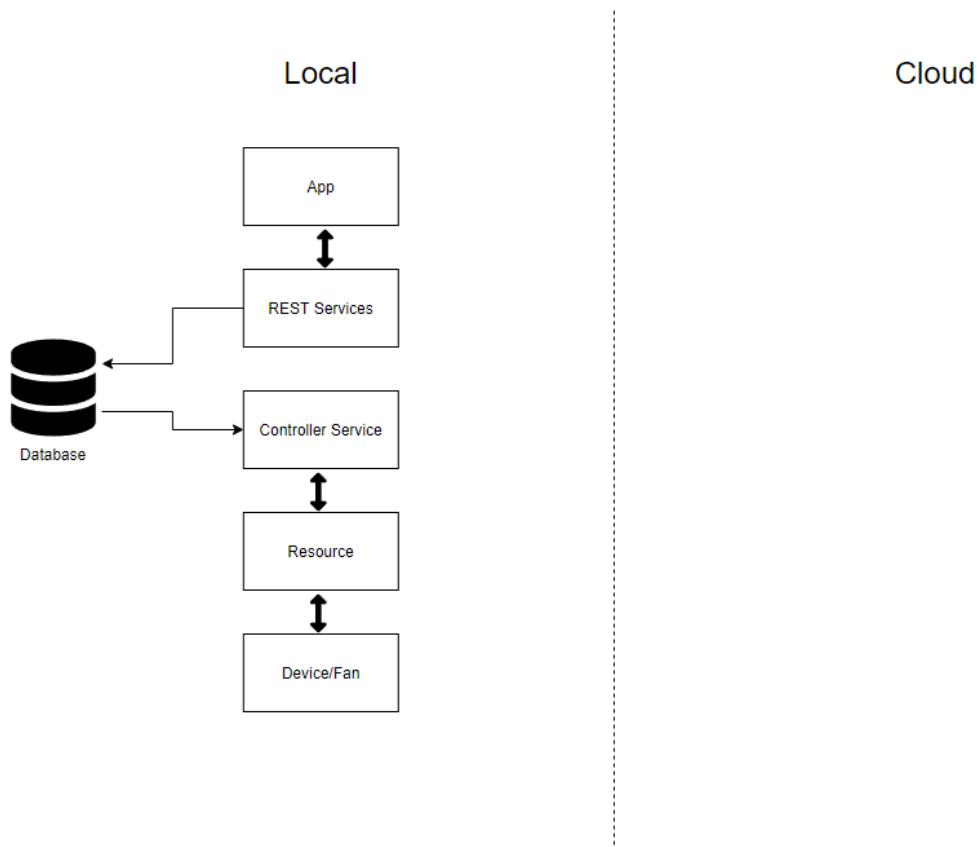
State Service: Sets the fan appliance on and off depending on current temperature.



State Service: Sets the fan appliance on and off depending on current temperature.

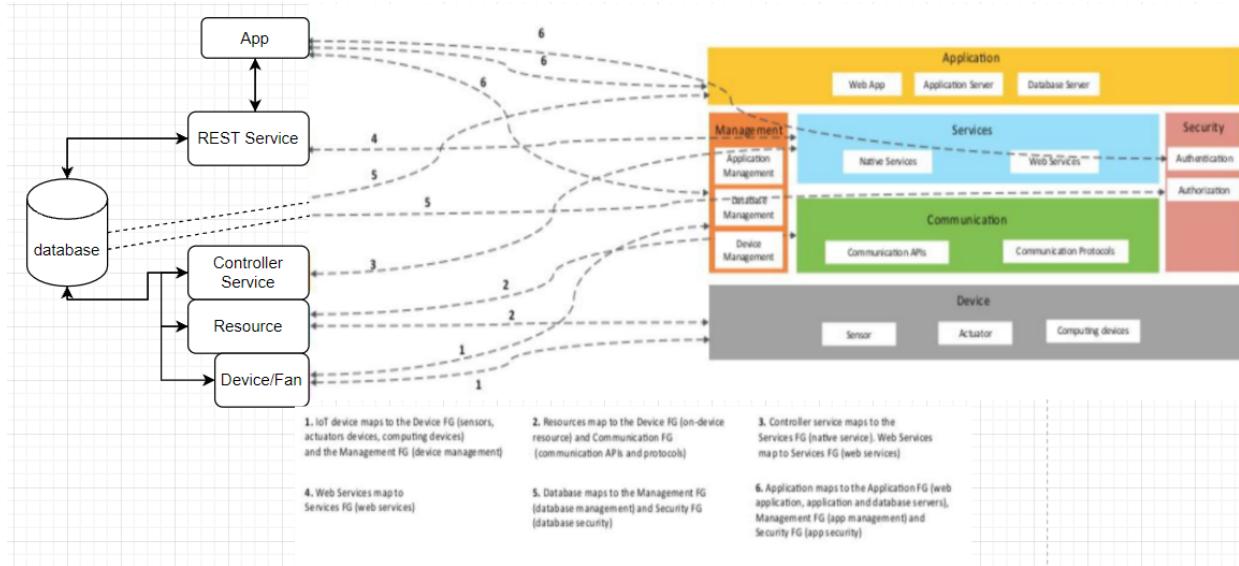
Controller Service: Auto mode monitors the temperature and switches the fan on and off. In manual mode, the user is able to change the state of the fan by switching it on/off.

6.Iot Level Specification:



7. Functional View Specification:

Diagram



User case:

UC 1- User should be able to view their device information from the application

UC 2- Devices must communicate with Django applications with high level api rest calls.

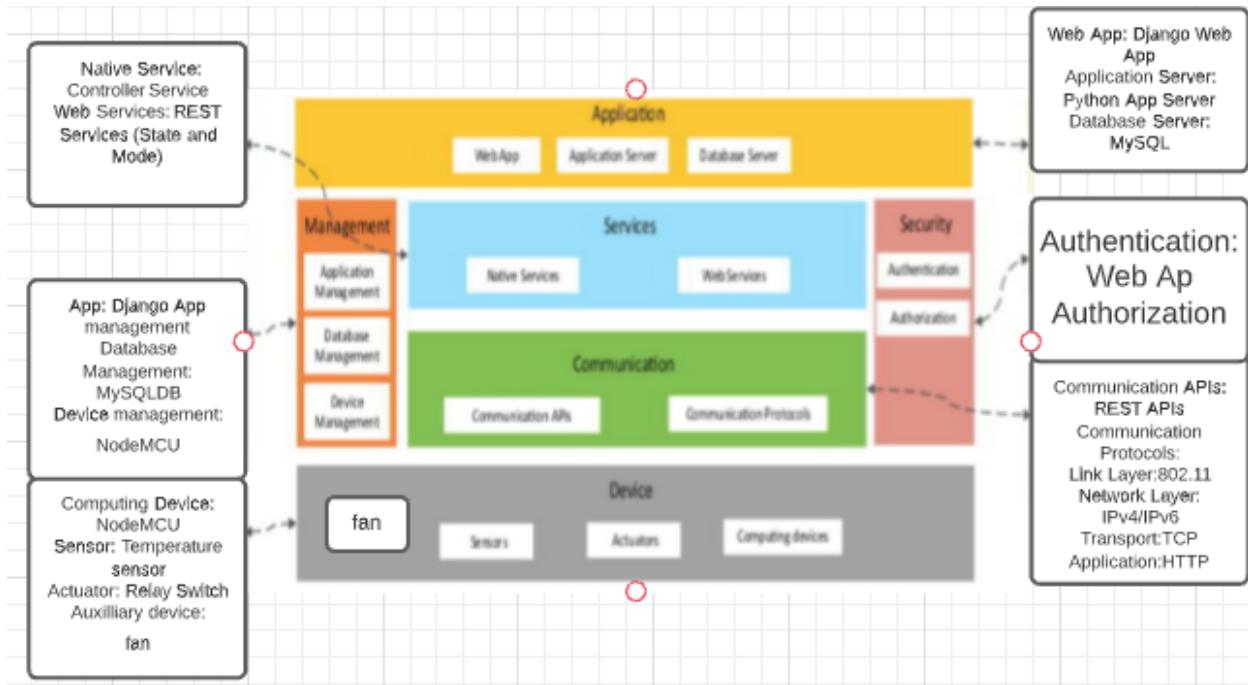
UC 3- Users should be authenticated using tokens and app should manage users

UC 4- The device should turn on the fan when the temperature passes a certain range.

These use cases are mapped to the application developed below. The mappings are available on the github along with the project: https://github.com/Mitul2000/Smart_Cooling_System_IoT

8. Operational View Specification:

Diagram



2.Implement your design using the Django REST framework. Submit a GitHub link of the code and tests for the implementation. In the README file create a table of contents to help navigate your code and show that your implementation performs the functionality if the use case is picked for design.

Github link : https://github.com/Mitul2000/Smart_Cooling_System_IoT

Admin page:

The screenshot shows the Django Admin interface for the 'Users' model. The left sidebar has sections for AUTH TOKEN, AUTHENTICATION AND AUTHORIZATION, and TEMPERATURE. The AUTHENTICATION AND AUTHORIZATION section contains 'Groups' and 'Users'. The TEMPERATURE section contains 'Devices'. The main area displays a list titled 'Select user to change' with one entry: 'Mitul' (username: patelmitul5053@gmail.com). A 'FILTER' sidebar on the right allows filtering by staff status (All, Yes, No), superuser status (All, Yes, No), and active status (All, Yes, No).

Device Model

The screenshot shows a code editor with multiple tabs open. The current tab is 'models.py - lot_'. The code defines a 'Device' model with fields: name, temp, status, description, and date_log. It also includes a __str__ method. The code editor interface includes a top bar with 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help' menus. The left sidebar shows project structure with 'OPEN EDITORS' and 'IOT_ASSIGNMENT2' sections.

```
src > temperature > models.py > Device
1   from django.db import models
2
3   # Create your models here.
4   class Device(models.Model):
5       name = models.CharField(max_length=100)
6       temp = models.CharField(max_length=100)
7       status = models.CharField(max_length=100)
8       description = models.TextField()
9       date_log = models.DateTimeField(auto_now=True)
10
11      def __str__(self):
12          return self.name
```

User Token:

The screenshot shows a user interface for managing authentication tokens. On the left, there's a sidebar with 'AUTH TOKEN' and 'TOKENS' sections, along with 'AUTHENTICATION AND AUTHORIZATION', 'Groups', and 'Users' sections. The main area is titled 'Select token to change' and displays a table with one row. The table has columns for 'KEY', 'USER', and 'CREATED'. The key '76a37de20178f7b898b748efc4c66908eca64e78' is listed under 'USER' as 'Mitul' and 'CREATED' as 'Oct. 31, 2021, 9:39 p.m.'. Below the table, it says '1 token'.

Test 1:

Making POST Request with authentication token:

The screenshot shows a Postman request configuration and its response. The request is a POST to 'http://127.0.0.1:8000/'. The 'Body' tab is selected, showing form-data with fields 'name' (NODEMCU) and 'temp' (28.4 C). The response tab shows a status of 200 OK with a response body of `{"name": "NODEMCU", "temp": "28.4 C"}`.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
name	NODEMCU			
temp	28.4 C			
Key	Value	Description		

Device Object in admin view

The screenshot shows a left sidebar with categories: AUTH TOKEN, AUTHENTICATION AND AUTHORIZATION, and TEMPERATURE. Under TEMPERATURE, there is a section for Devices with a '+ Add' button. The main area is titled 'Select device to change' and shows a list of devices. The first device is 'DEVICE' (checkbox selected), followed by two 'NODEMCU' entries. Below the list, it says '2 devices'. At the top right, there is an 'Action:' dropdown set to '—', a 'Go' button, and a status '0 of 2 selected'.

Test2:

Making POST Request with authentication token for registering new device:

The screenshot shows a POST request to 'http://127.0.0.1:8000/'. The Body tab is selected, showing form-data with 'name' set to 'New Device Test' and 'temp' set to '30.4 C'. To the right, a separate window shows the device list after the POST request, now including the new device 'New Device Test' along with the existing 'NODEMCU' entries, totaling 3 devices.

New Device attributes:

The screenshot shows the Django administration interface. On the left, there's a sidebar with categories: AUTH TOKEN (Tokens), AUTHENTICATION AND AUTHORIZATION (Groups, Users), and TEMPERATURE (Devices). The main area is titled "Change device" and shows a "New Device Test" entry. The fields are: Name (New Device Test), Temp (30.4 C), and Status (empty). Below these is a large Description field. At the bottom right is a red "Delete" button.

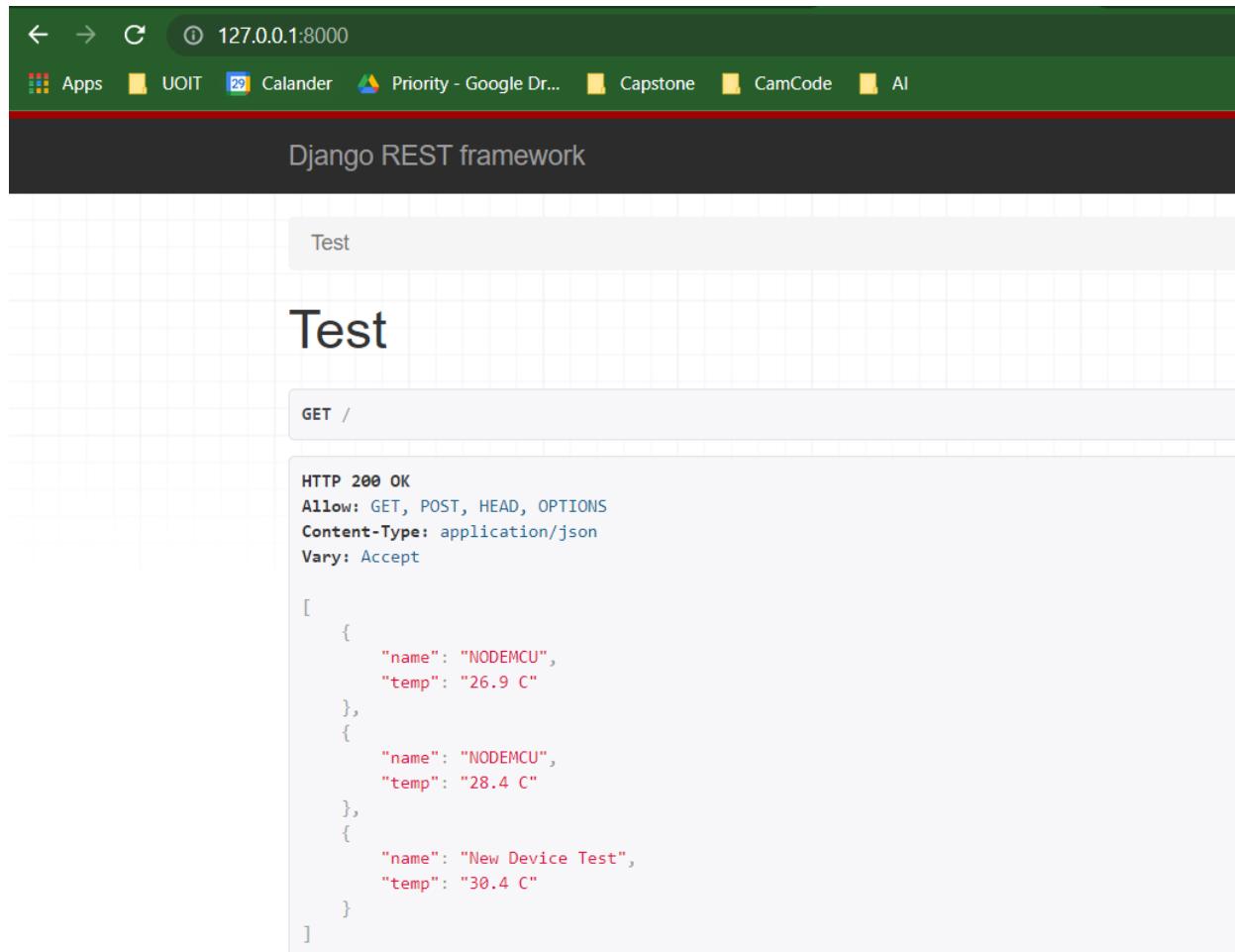
Test 3: sending post request with authentication token

The screenshot shows a Postman request configuration. The method is set to POST, and the URL is http://127.0.0.1:8000/. The "Body" tab is selected, showing form-data with two fields: "name" (Value: Auth Test) and "temp" (Value: 30.4 C). Below the body, under "Headers", there are 10 items listed. In the "Body" section, there is also a table with columns "KEY" and "VALUE". The first row has "name" in "KEY" and "Auth Test" in "VALUE". The second row has "temp" in "KEY" and "30.4 C" in "VALUE". The third row has "Key" in "KEY" and "Value" in "VALUE". At the bottom, the "Pretty" option is highlighted with a red box, and the response body is shown as: {"detail": "Invalid token."}

Test 4:

Get request to view all devices and temperature on dashboard page:

This list can also be filtered based on the user.



The screenshot shows a browser window with the address bar displaying "127.0.0.1:8000". Below the address bar is a navigation bar with several items: Apps, UOIT, Calander, Priority - Google Dr..., Capstone, CamCode, and AI. The main content area has a dark header bar with the text "Django REST framework". Below this is a light gray section containing the word "Test". Underneath is a larger section with the word "Test" in large bold letters. At the top of this section is a button labeled "GET /". Below the button, the response is shown in a code block:

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[{"name": "NODEMCU", "temp": "26.9 C"}, {"name": "NODEMCU", "temp": "28.4 C"}, {"name": "New Device Test", "temp": "30.4 C"}]
```