1. Open “**Putty**” software and type the IP address of Raspberry Pi and the port number is “**22**”. If there’s any confusion with IP address, connect the HDMI cable to Pi and look out for the IP address in the WiFi menu (present on taskbar). Enter the credentials as “**login:pi**” and “**password:raspberry**”.

2. Run “**VNC Server**” and type IP address of Pi followed by a “colon” and “1”. eg: “**192.168.100.5:1**” and in enter the credentials as “login:pi” and “password:raspberry”.

3. To host raspberry Pi server over raspbian, run the IP address of raspberry PI followed by its port number. E.g: “**192.168.100.5:5000**”

4. It may happen, error shows that **“Error 98: Port already in use**”, so stop the port from running by the command → **“fuser -k 5000/tcp**” (where 5000 is the port number).

5. To run a script, open the file and press “**F5**”

6. Install “Flask” : **pip3 install Flask**

7. Install “Paho-MQTT”: **pip3 install paho-mqtt**

8. **@app.route('url/to/route’)** is used to tell Flask what URL should trigger our function.

9. **app.run(debug=True, host='0.0.0.0', port=5000)** to use the flask executable to start your server and run on your machines IP address.

10. **login_webservice() function:** 192.168.100.5:5000/api/user/Login → Validates the user login using POST method and in response returns a static JSON which contains user’s information.

11. **request.get_json()** is used to parse and return the data as JSON.

12. **getUserDeviceInfo() function:** 192.168.100.5:5000/api/mobile/GetUserDeviceInfo → a URL which uses POST method and returns a JSON which creates a UI on Android device.

13. **appChangeStatus() function:** 192.168.100.5:5000/api/mobile/ChangeSwitchStatus → To change the status of the appliances, we need to send JSON data to the server using POST method. Later on, JSON is modified:
    - “Devices” is renamed to “Device” and “Devices” array is replaced by “Device” object.
    - Sequence is added to JSON and name is “SEQ” and value changes according to number of switches.

14. In order the JSON to be published, it should be of “string” format, so we convert it by using “**json.dumps(json_content)**” function.

15. **mqtt_connect():** The function create a client and connect it using host ID (192.68.100.5) and port number (1883). And finally we set the loop running and returns the “client”.

16. To **publish** a payload (message), we use **client.publish(topic, payload).** Here the **topic** is: nGShelter/WriteGpio/1011 and **payload** is the modified JSON.

17. **deviceChangeStatus(): It serves 2 function:**
    - Whenever the microcontroller gets booted, in order to get back the last appliances’ status, it asks the MQTT broker on the topic “nGShelter/GetDeviceStatus/1011” to publish a JSON payload on the topic “nGShelter/WriteGpio/1011”.
    - Inorder the UI to get reflected on Android APP whenever the user changes appliance status from switch, deviceChangeStatus() comes to action. The switch publishes a JSON payload on the topic “nGShelter/PublishSwitchStatus/1011” and the same JSON payload is published to Android APP on the topic ”nGShelter/ReadGpio/1011”.

18. The message payload received on any topics and if they are needed to be published further, first they needed to be decoded to “utf-8” format which is done by **msg.payload.decode("utf-8")** function.

19. To install MySQL → **sudo apt-get install mysql-server**

20. Later on, install MySQL connector → **sudo apt-get -y install python3-mysql.connector**

21. Create user and grant privileges:
    - **CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';**
    - **GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';**
    - **FLUSH PRIVILEGES;**

22. **Flask** → A web framework
    Libraries imported from Flask:
    - **render_template** (Line 1): Instead of returning hard code HTML from the function, a HTML file can be rendered by the **render_template()** function.
    - **request** (Line 1): The data from a client’s web page is sent to the server as a global request object. In order to process the request data, it should be imported from the Flask module.

- **redirect** (Line 1)**:** Flask class has a **redirect()** function. When called, it returns a response object and redirects the user to another target location with specific status code.

23. **Mysql.connector** (Line 2)→ MySQL Connector Python is the official Oracle-supported driver to connect MySQL through python.
    - **Error, errorcode** (Line 3): client error codes defined as module attributes with the error number as value. Using error codes instead of error numbers could make reading the source code a bit easier

24. **Sys** (Line 4): The sys module provides information about constants, functions and methods of the Python interpreter. dir(system) gives a summary of the available constants, functions and methods. Another possibility is the help() function. Using help(sys) provides valuable detail information.

25. **app = Flask(__name__)** (Line 6) : once we import *Flask*, we need to create an instance of the *Flask* class for our web app. That's what **app = Flask(__name__)** does. **__name__** is a special variable that gets as value the string "__main__".

26. **def __init__(self)** (Line 9)**:** the **self** variable represents the instance of the object itself. The __init__ method is roughly what represents a constructor in Python. When you call **SQL_Ops()** Python creates an object for you, and passes it as the first parameter to the __init__ method. Any additional parameters (e.g., A(24, 'Hello')) will also get passed as arguments--in this case causing an exception to be raised, since the constructor isn't expecting them.

27. **self.connection = mysql.connector.connect(**
    host = "localhost",
    user = "admin",
    passwd = "raspberry",
    database = "nGShelter"
) (Line 12)

    → To connect to the MySQL database, we require hostname (**host**) on which MySQL database is running, the login credentials (**user** and **password**) of database, and **name of the database**.

28. **self.mycursor = self.connection.cursor()** (Line 18): To create an instance of connection object, so that SQL operation can be done in other methods (function).

29. **self.connection.rollback()** (Line 18): When one of the transaction fail to execute, and you want to revert or undo all your changes, then you need to call a rollback method of MySQL connection object.

30. **self.mycursor.execute(“** Query **”)** (Line 25)**:** To execute a MySQL query using Python we use **mycursor.execute** and **self** variable represents the instance of the object itself.

31. **self.mycursor.fetchall()** (Line 26)**:** The method fetches all (or all remaining) rows of a query result set and returns a list of tuples. If no more rows are available, it returns an empty list.

32. **self.connection.commit()** (Line 28)**:** This method sends a COMMIT statement to the MySQL server, committing the current transaction. Since by default Connector/Python does not autocommit, it is important to call this method after every transaction that modifies data for tables that use transactional storage engines.

33. Description of Queries:
    - **self.mycursor.execute("Select RoomName from ROOM where ID like '"+room_id+"'")** (Line 25) → To select all the **RoomName** column from MySQL database, if the **ID** column matches **room_id**.

    - **self.mycursor.execute("Insert into `Device`(`Room_ID`, `Room_Name`, `SKU_ID`, `Switch_type`, `Module_Name`, `No_of_switch`) Values ('"+room_id+"', '"+room_name[0][0]+"', '"+sku_id+"', '"+switch_type+"', '"+module_name+"', '"+no_of_switches+"')")** (Line 27) → To Insert data into **`Room_ID`, `Room_Name`, `SKU_ID`, `Switch_type`, `Module_Name`, `No_of_switch`** column of **Device** table, where the values equals **room_id, room_name[0][0], sku_id, switch_type, module_name, no_of_switches.**

    - **self.mycursor.execute("Create TABLE IF NOT EXISTS Switch(ID INT AUTO_INCREMENT, Dev_ID VARCHAR(255), SEQ VARCHAR(255), Switch_Name VARCHAR(255), Is_Dimmable INT, PRIMARY KEY(ID))")** (Line 31) → To create a **Table** if it doesn't exists in database. The column name are **ID, Dev_ID, SEQ , Switch_Name, Is_Dimmable** and type of value it holds (basically its datatype) are **VARCHAR, INT.** The **ID** column is auto increment, i.e., the user doesn’t need to insert values in it.

    - **self.mycursor.execute("UPDATE `Device` SET `Module_Name` = '" + arr[5] + "' WHERE `ID` = '" + dev_id + "'")** (Line 39) → To update a Table (**Device**) and it’s columns (**Module_Name**) with new value (**arr[5]**). **WHERE** clause represents the **if** condition matches that query should be executed only if **ID** equals **dev_id.**

34. HTML and JINJA2 codes and description:
    - **if request.method == 'POST'** (Line 92)**:** The result of request.method == "POST" is a boolean value - True if the current request from a user was

performed using the HTTP "POST" method, of False otherwise (usually that means HTTP "GET", but there are also other methods).

- **new_module_name = request.form["new_module_name"]** (Line 95)→ sending request to the specified **<form>** tag of HTTP page and saving the response as response object.

- **return(redirect('/Device'))** (Line 113)→ After execution of a block of code, to **redirect** to outside **URL** (**'/Device'**)

- **return(render_template("edit_device.html", data = data, url_data = dev_id))** (Line 124)→ After execution of a block of code, **return** an HTML page (**edit_device.html**) which contain some dynamic data to be shown. It is passed on to the page using JINJA2 template using variables (**data, url_data**) and the value to be passed is stored in **data, dev_id.**

35. Function declared and their details:
    - Function declared in **SQL_Ops()** class:
        - **add_device**(**self, room_id, switch_type, sku_id, module_name, no_of_switches)** (Line 24)**:** To add a new device details to your Room (in your database). The parameters **room_id** defines the **ID** of the room (Check **ROOM Table** in database) in which the device has been added. **switch_type** defines the type of switch (Relay 10A, Relay 20A or Triac) of the device installed. **Sku_id** is the unique ID of the device that is given at the time of manufacture. **module_name** is the name of the device, it can also be renamed according to user's choice. **no_of_switches** defines the switch count of the device (no of relays/triac present in the device)

        - **Create_switches(self, arr)** (Line 30)**:** After the user has created/added a device, he needs to add the details of the switch i.e., **Dev_ID, SEQ, Switch_Name, Is_Dimmable. Dev_ID** describes about the **ID** of the device (See **Device** Table in database for further details) in which switches are going to be created. **SEQ** defines the sequence of the switch in which they are going to be numbered. **Switch_Name** represents the name of the switch (like "switch 1" and later on it can be updated). **Is_Dimmable** tells whether the appliance connected to the switch should be made dimmable or not.

        - **update_device(self, arr)** (Line 36): After the user has created his switch and filled all its value, and in case he needs to update his switch details like his switch name, etc. this function will be called to update the value in the database. Here, The **Module_Name, SEQ, Switch_Name, Is_Dimmable** value is updated the to know about which device details values are to be updated, **ID** is passed to it whose value is obtained from

the URL which is dynamically generated after clicking on "**Edit Switches"** button on the webpage.

- **create_room(self, userName, roomName)** (Line 48)**:** To create a room under a specified user, this function is called to insert the details of the username and his roomname in the database under **ROOM** table (check **ROOM** table in the database). Also, this function checks that if any of the Room name is already present with that user, if so then it will fail to create the room and will redirect to Room webpage.

- **def update_room(self, userName, roomName, newName)** (Line 68)**:** After the user has created his Room and filled all its value, and in case he needs to update his Room details like his room name, then this function will be called to update the value in the database. It will be updated in both the **Room** and **Device** table.

- Function declared globally:
  - **main()** (Line 254)**:** This method is called from **__main__** function. It contains the set of functions which gets executed after the script runs.
    Note: Every module in python has a special attribute called **__name__** . The value of __name__ attribute is set to **main** when module run as main program

  - **Room_page()** (Line 246)**:** This function routes to **'/Room'** webpage. The function is used to show the list of users and their rooms. It displays data (**userID, RoomName**) from **ROOM** table. A "**Create Room**" button is also present which redirects to **'/Room/Create'** webpage.

  - **Room_Creation()** (Line 229): This function routes to **'/Room/Create'** webpage. This function is used to create a Room under a predefined user. It displays list of username from **User_details** table. Later on, after selecting the User name, we can input the Room name and submit the details.

  - **Room_edit()** (Line 220): This function routes to **'/Room/edit/<no_of_room>'** webpage. This function is used to edit the Room name. The url is dynamically created when the user click "**Edit Room**" button from the **'/Room'** webpage. The **ID** of the **ROOM** table is passed inside **no_of_room** variable. Also, **userID, RoomName** is passed to the webpage.

  - **Submit_Room()** (Line 207)**:** This function routes to **'/Submit/Edit/Room'** webpage. This function is used to get values from the form after the user modifies his Room name from **Room_edit()** function. The values returned

from the webpage are **userName, roomName, newName. userName** defines the email ID of the user**, roomName** defines the old Room name**, newName** defines the modified Room name.

- **Device_page()** (Line 198)**:** This function routes to **'/Device'** webpage. This function is used to Create a **Device** table if it doesn't exist in the database. It also returns **ID, SKU_ID, Module_Name, Room_Name, Switch_type, No_of_switch, Room_ID** from the **Device** table in order to list the details of the device in the webpage.

- **Select_Room()** (Line 189)**:** This function routes to **'/Device/Create'** webpage. This function is used to fetch the data (**User name** and **Room name**) from **ROOM** table and return to **'Create_device.html'** webpage. The webpage shows the list of User name and Room name with "**Add Device**" button. which routes to **'/Device/Create/<room_id>'** page.

- **Add_Devices()** (Line 179): This function routes to **'/Device/Create/<room_id>'** webpage. This function is used to add the device details. But before adding the details, we should have an idea of what are the inputs available for **switch_type_select** ('Triac', 'Relay 10A', 'Relay 20A'), **switch_no** ('1', '2', '4', '6', '10') and **room_id** which defines in which Room, the device details are to be added. **room_id** is dynamically generated after clicking the "**Add Device**" button. These all details are sent to '**add_device.html**' webpage.

- **Submit_Device()** (Line 165): This function routes to '**/Submit/Device/<room_id>**' webpage. This function is used to submit the Device details like **room_id, switch_type, sku_id, module_name, no_of_switches. room_id** is fetched from the **ID** column. **switch_type** is the type of switch which is selected from the dropdown list of **'Triac', 'Relay 10A', 'Relay 20A'**. **sku_id** is the unique ID of the device that is given at the time of manufacture. **module_name** is the name of the device, it can also be renamed according to user's choice. **no_of_switches** defines the switch count of the device (no of relays/triac present in the device). After submitting the details into the database and then redirects to **'/Device'** URL.

- **Add_switches()** (Line 152)**:** This function routes to '**/Switches/<no_of_dev>**' webpage. This function is used to send Device details (**Room_ID, Room_Name, Switch_type, Module_Name, No_of_switch, ID**) from **Device** table and **userID** from **ROOM** Table using **ID** to the "**add_switches.html**" webpage.

- **Submit_switch()** (Line 126)**:** This function routes to '**/Submit/Switch/<dev_id>**' webpage. This function is used to get data (**No_of_switches, seq_arr, switch_name_arr, isEnabled_arr, dev_id**) from "**add_switches.html**" webpage. **No_of_switches** defines the no of switches present in the device. **Seq_arr**, **switch_name_arr, isEnabled_arr,** is the list of sequence, switch name(s), and if dimming is enabled of the switches obtained after filling the form. After inserting these values, the web page is redirected to '**/Device**' URL.

- **Edit_device()** (Line 115)**:** This function routes to '**/Device/Edit/<dev_id>**' webpage. This function is used to edit the Device details. Firstly, this function sends data (**Room_Name, Switch_type, Module_name, No_of_switch, userID, dev_id**) to the "**edit_device.html**" webpage. **Dev_id** is the Device ID **(ID** column from **Device** table).

- **Submit_edit_device()** (Line 83)**:** This function routes to '**/Submit/Device/Edit/<dev_id**' webpage. This function is used to submit the modified Device details (**no_of_switches, seq_arr, switch_name_arr, isEnabled_arr, new_module_name**). After getting these values, we update in the database and redirect **'/Device'** URL.