INTERNALS OF APPLICATION SERVER

TEAM-3 REPORT

APPLICATION MANAGER, PLATFORM MANAGER, MONITORING MODULE, BOOTSTRAP

AYUSH KHASGIWALA (2020201088)

NAMAN JUNEJA (2020201072)

SOMYA LALWANI (2020201092)

APPLICATION MANAGER

The module is used for authenticating the users and blocking invalid users. When the user is successfully authenticated, he must further enter the algorithm/ application which he/she wants to run. Different files provided by the user are checked for their correct format.

A) Functional Overview

Application manger can be broken down into the following submodules:

1. To develop a web page for user registration and login.
2. The web page will also get the application file for the platform to be deployed and respective application requirements from the user.
3. Register**:** New users can sign up using a username and password. Existing users can login using their respective username and password and the module will authenticate it. Also handle the login and registration for Application developer and platform admin.
4. 2 types of users can login into the system:
    a. Application User: The user who can run the application that is deployed on the platform and can provide input to the application.
    b. Application Admin(Developer): The user who can provide the application and the config file to the platform which will be sent to the Platform Manager Module for further application deployment.
5. Format Checking: Different files provided by the user are checked for their correct format.
6. To parse the config file received from the user, extract the requirements, and pass it to the respective module. For example, the sensor's details, field, controller type and notification type.

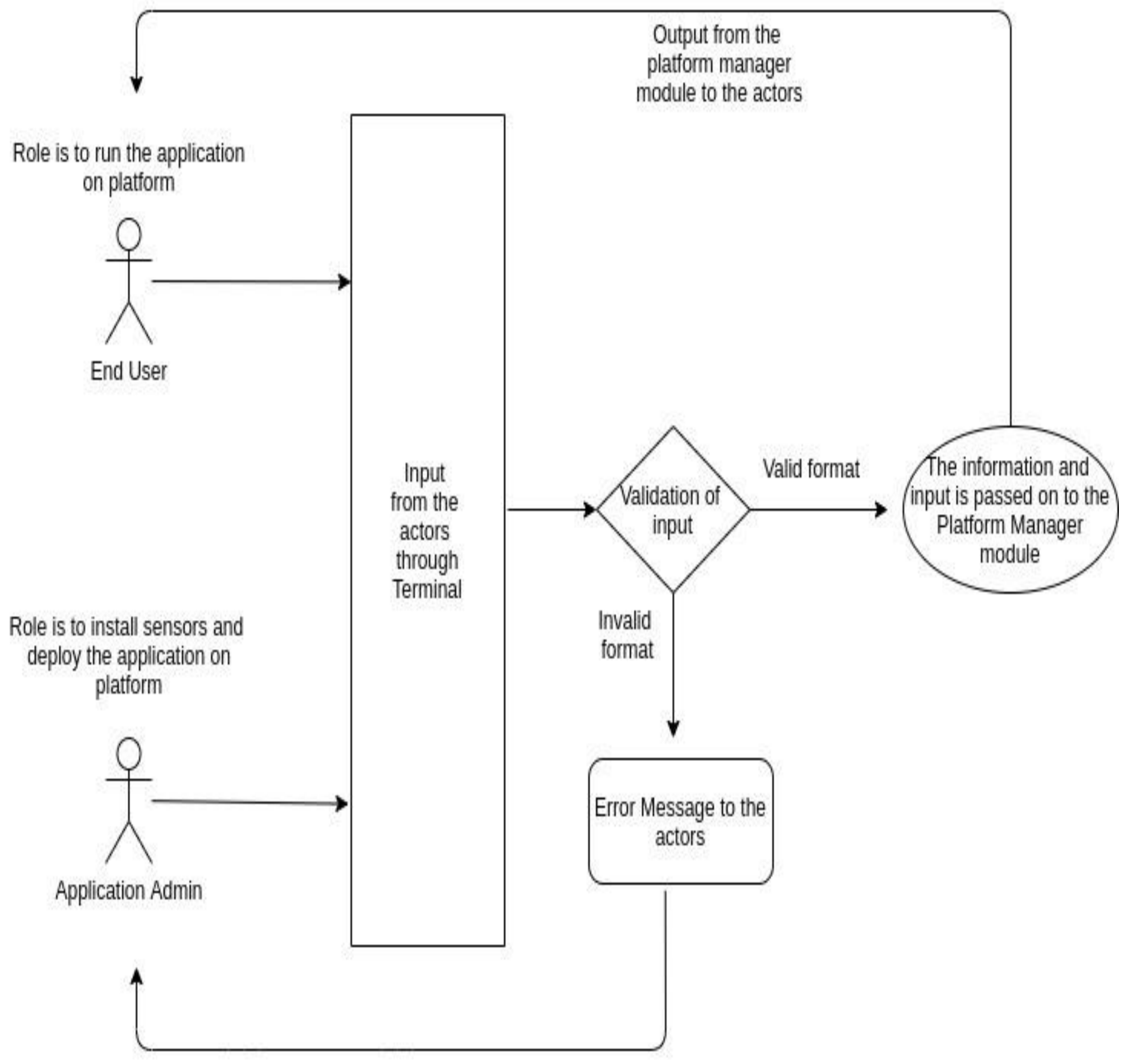
B) Technologies Used

- Flask
- MongoDB

- HTML
- CSS
- Python
- Socket Programming

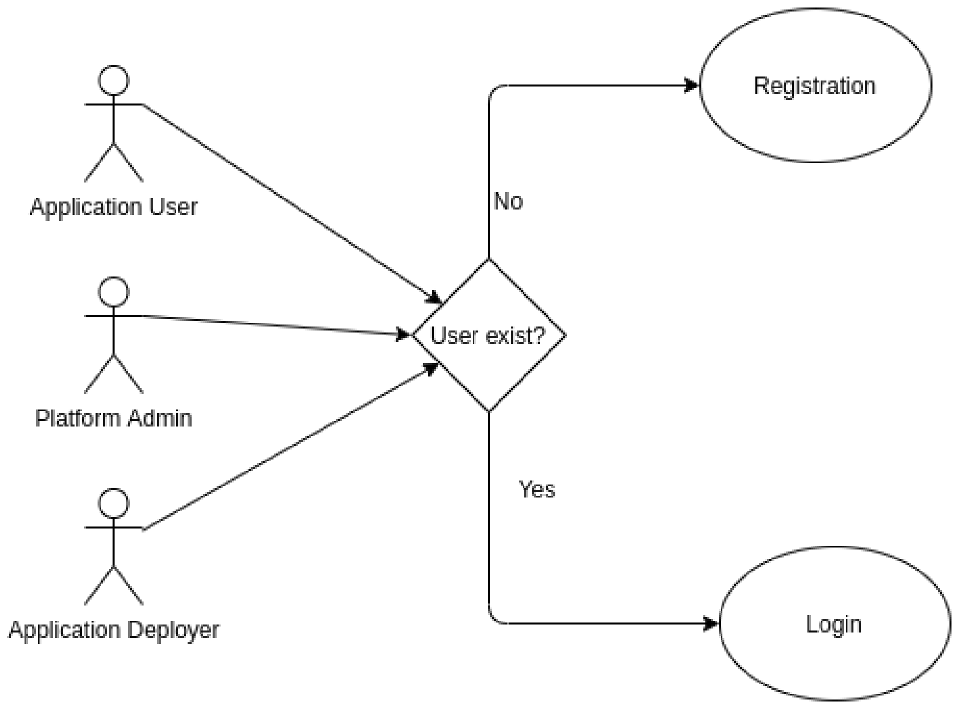
C) Functions (Use cases)

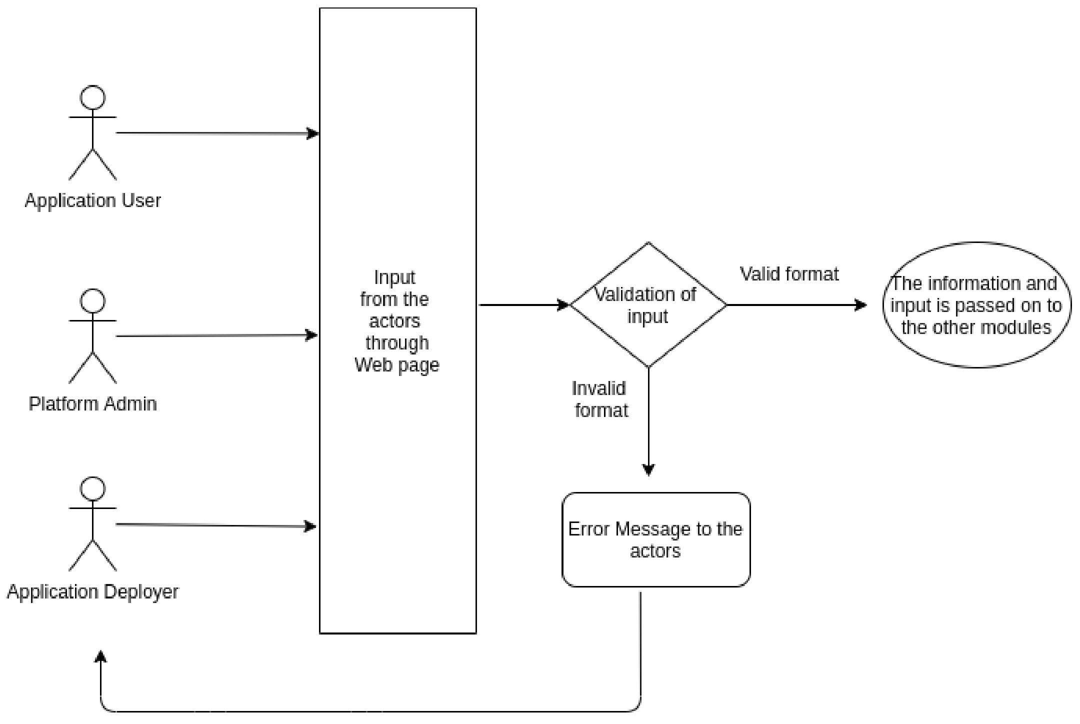
1. When the user is not registered, then register the user first.
2. If the user is already registered, then check whether the user enters the correct user id and password.
3. If it is correct, then allow the user to enter the application file.
4. Check for the type of user based on username and redirect to the respective user’s page.
5. Check whether the application user has entered valid commands and executes those commands internally.
6. Provide different functions like add sensors, notifications, and controller to the admin user, and send these instructions to Sensor Manager.
7. To check the valid application format of the file received from the Application Development User. If the format is valid then pass the application on to the scheduler.
8. Check for the format of the config file & extract the requirements.
9. Check the format of different files provided by the user.
10. Take the application for deployment and send to further modules for deployment in the scheduler modeule.
11. Give the end user option to run a given application, start and stop it.



Output from the platform manager module to the actors
Role is to run the application on platform
End User
Input from the actors through Terminal
Validation of input
Valid format
The information and input is passed on to the Platform Manager module
Invalid format
Role is to install sensors and deploy the application on platform
Application Admin
Error Message to the actors

FIGURE : Application Manager - Processing Input from different actors

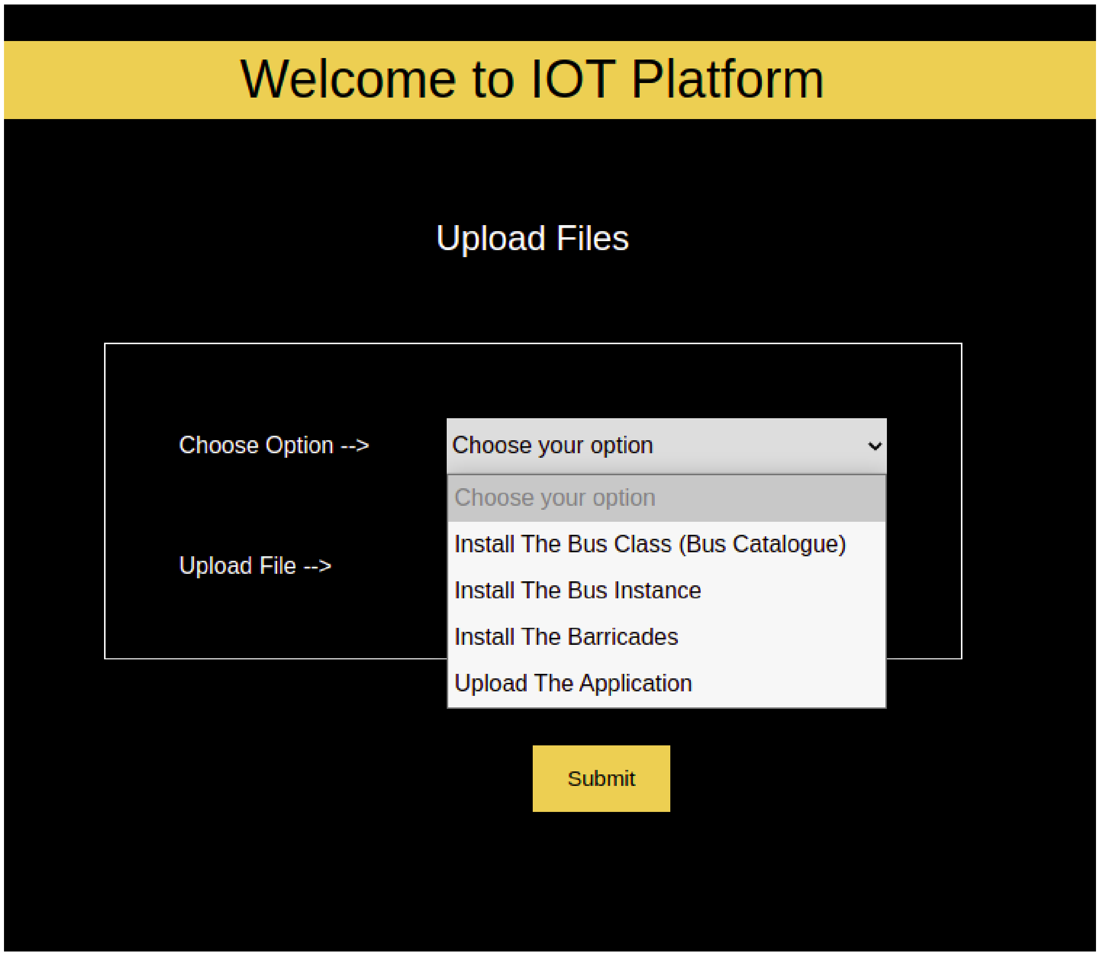

Application User
Platform Admin
Application Deployer
User exist?
No
Yes
Registration
Login


Application User
Platform Admin
Application Deployer
Input from the actors through Web page
Validation of input
Valid format
The information and input is passed on to the other modules
Invalid format
Error Message to the actors

1. APPLICATION ADMIN USER:

Designed according to the sample application:

1. Install the Bus Class
2. Install the Bus Instance
3. Install the Barricades
4. Upload the Application



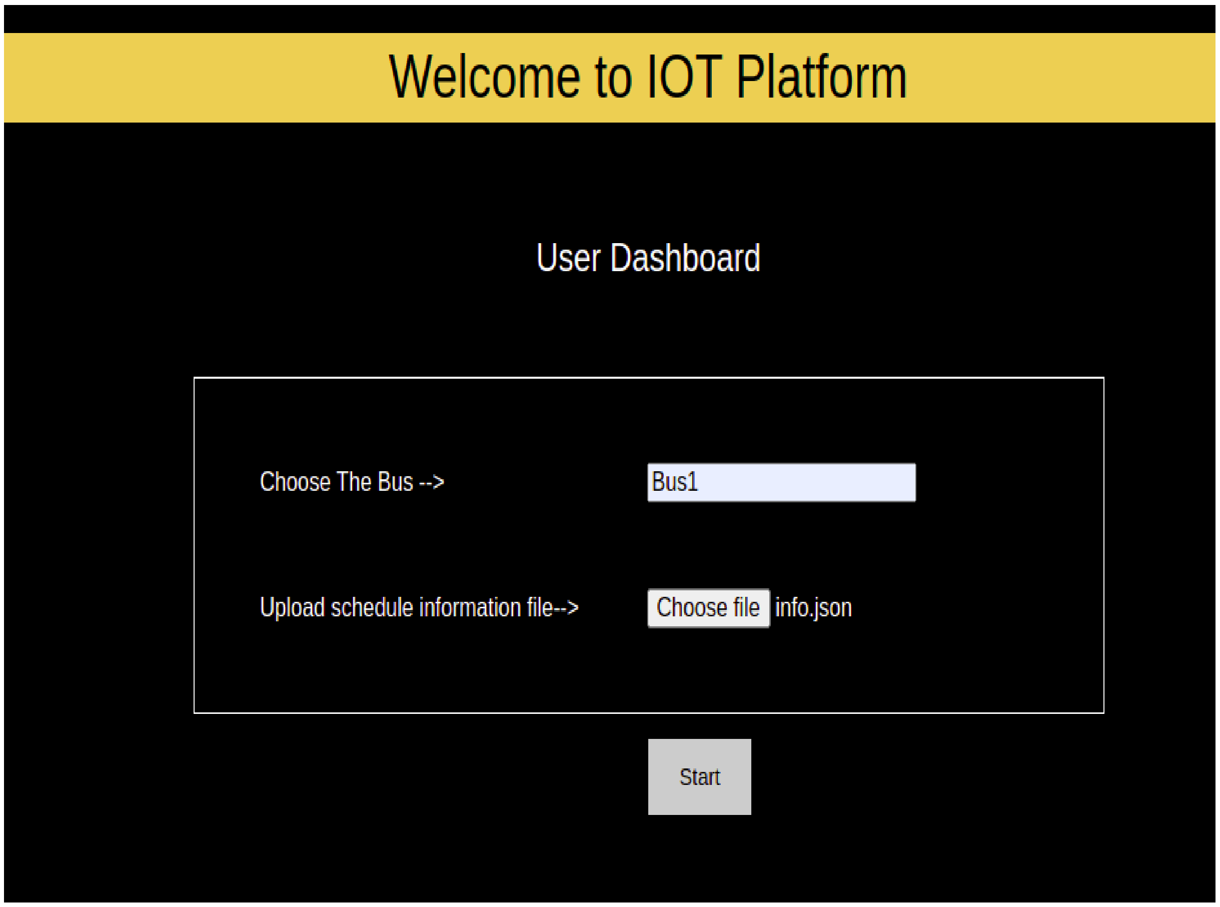
- The **Application Admin** is the one who will deploy the application and will ask the platform to install the sensor class(type) if it is not available on the platform. This will be done by providing a config file(sensorClass.json) to the platform
- The Application Admin will also install the instances of the sensor class(type) and this will be done by providing the the config file(sensorInstance.json) to the platform
- And then the application admin will give the platform the application file(code file) and the config file(named -req.json)(to run the application file)

- On sending the file for deploying to the platform manager, then the platform manager will communicate to the sensor manager to validate the config file given to deploy the application. If any of the instances are not available then it will tell the platform manager and then the platform manager will then send the msg to the application manager .
- Then the application manager will show the message to the application admin and by seeing that message the application admin can then install the sensor class(type) and the instances and then again deploy the application. In this way the application will be deployed by the application admin

2. END USER:

The following options are given to the end user:

1. Choose the Bus
2. Upload the schedule information file (consists of start time, end time, recurring bit, etc. Details – to be sent to the scheduler)



- The End User is the one which will run the application deployed on the platform.
- The user will be given a list of the applications (received from Platform Manager module) which have been already deployed on the platform.
- Based on this list, will be asked to enter the following details: application name,user config file, start time & end time. These details are then sent to the Platform Manager module.
- In the Platform_Manager module, data will be read from the user config.json file and based on the location, we use the getSensorIdByLocation to get all the sensor IDs located at that location.
- In continuation with this connection (between Platform Manager and User), the Platform Manager will create new threads and send the following details to the deployer module : list of all the sensor IDs present at that location, start time and the end time.
- On the User terminal, the progress of the running application is shown and the final output of the query asked by the user will be presented in another terminal which is communicated via Scheduler & Deployment Module.

PLATFORM MANAGER

A) Functional Overview & Capabilities of each part

1. It communicated between the front end (flask HTML pages I.e. Application Manager Module) with the back end of the platform (various modules like Sensor Manager, Scheduler, etc.)
2. In this module, we will be communicating with the Application Manager, Sensor Manager and the Scheduler module. As soon as the module is run and the server is established, we create new threads to communicate with different modules mentioned above.
3. Make the platform portable by packaging the platform. This package consists of a local folder named “Files_upload” in the system it is present.
4. The files received from the application are validated and then configured. For validation, the files need to be parsed and checked for various aspects.

For configuration, the module needs to communicate with the Sensor Manager by calling its functions.

5. These received files are found from the computer and stored to the local storage folder “Files_upload” for further use in the application making the location unknown to the other users.

B) Technologies used

- MongoDB
- Python
- Socket Programming

C) Functions (Use cases)

- In this module, we will be communicating with the Application Manager, Sensor Manager and the Scheduler module.
- As soon as the module is run and the server is established, we create new threads to communicate with different modules mentioned above.
- The input from the user/admin will be received at the platform manager and then the module will first identify the user i.e. the user will be either Application admin or the End user.
- For the **Application Admin**, it will first receive the different config files i.e the sensorDetails.json, sensorInstance.json or programSensorConfig.json from the application manager.
- 3 types of tasks can be done : to install a new sensor class, to install a new sensor instance, to deploy the application.
- The module will send the data from that config files to the sensor manager module for verification or for the installation of the sensor type and sensor instances.
- Then the message received from the sensor manager module will be passed on to the application manager module.
- Once, the application admin thinks that all the sensors that their application needs are installed, then he can send the application to get it deployed on the platform.

- Then the platform manager will send it for validation to the sensor management module and the validation response is received at the platform manager side
- If all the conditions are met then the platform manager will deploy it to the platform and it will be then available to the user.
- For the **End User**, the module returns a list of all the applications already deployed on the platform to the end user.
- If no module has been deployed yet, it will navigate back to the end user page with the status of "No application deployed yet!".
- In the case, where the end user receives a list of deployed application's name, the end user will send the following information back to the module : appname, start_time, end_time and location.
- Data will be read from the user config file(.json) file and based on the location, we use the getSensorIdByLocation to get all the sensor IDs located at that location.
- In continuation with this connection (between Platform Manager and User), the module creates new threads to communicate with the Scheduler Module.
- In these threads, it sends the following details to the deployer module: list of all the sensor IDs present at that location, location, start time and the end time. The output of this call is the status of the application.
- On the User terminal, the progress of the running application is returned from Platform

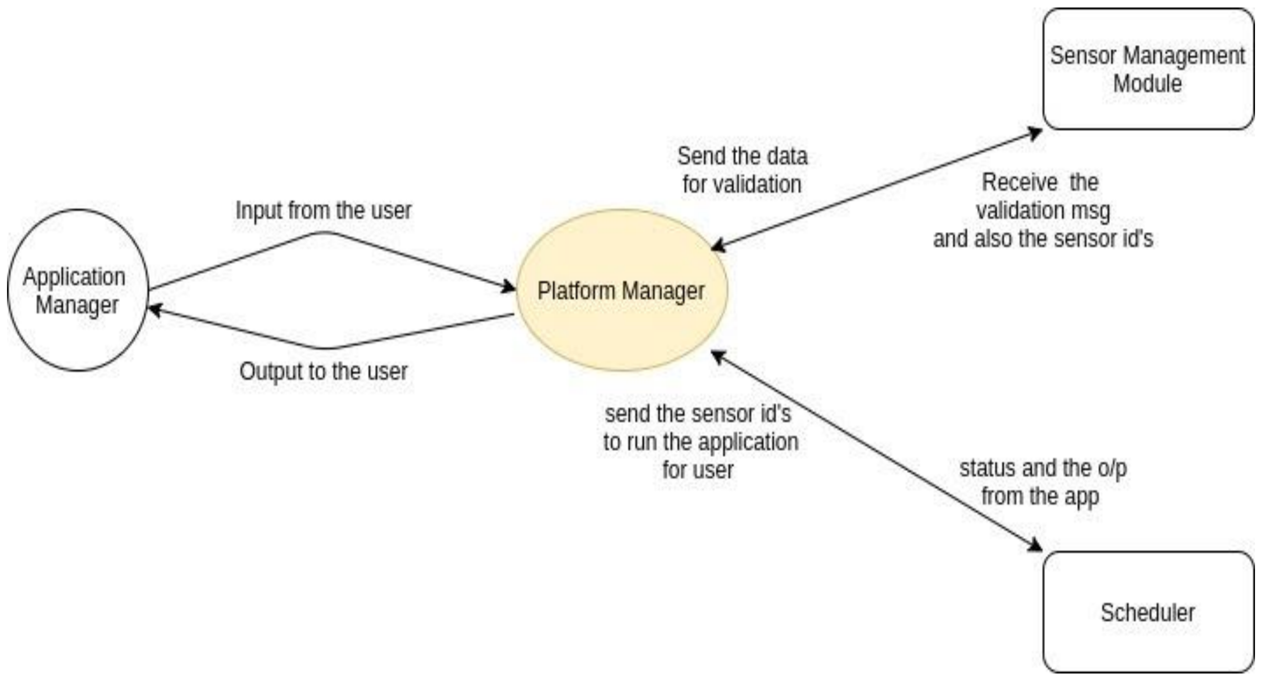

Sensor Management Module
Send the data for validation
Receive the validation msg and also the sensor id's
Input from the user
Application Manager
Platform Manager
Output to the user
send the sensor id's to run the application for user
status and the o/p from the app
Scheduler

Diagram 3 - Platform Manager

MONITORING MODULE

A) Functional Overview:

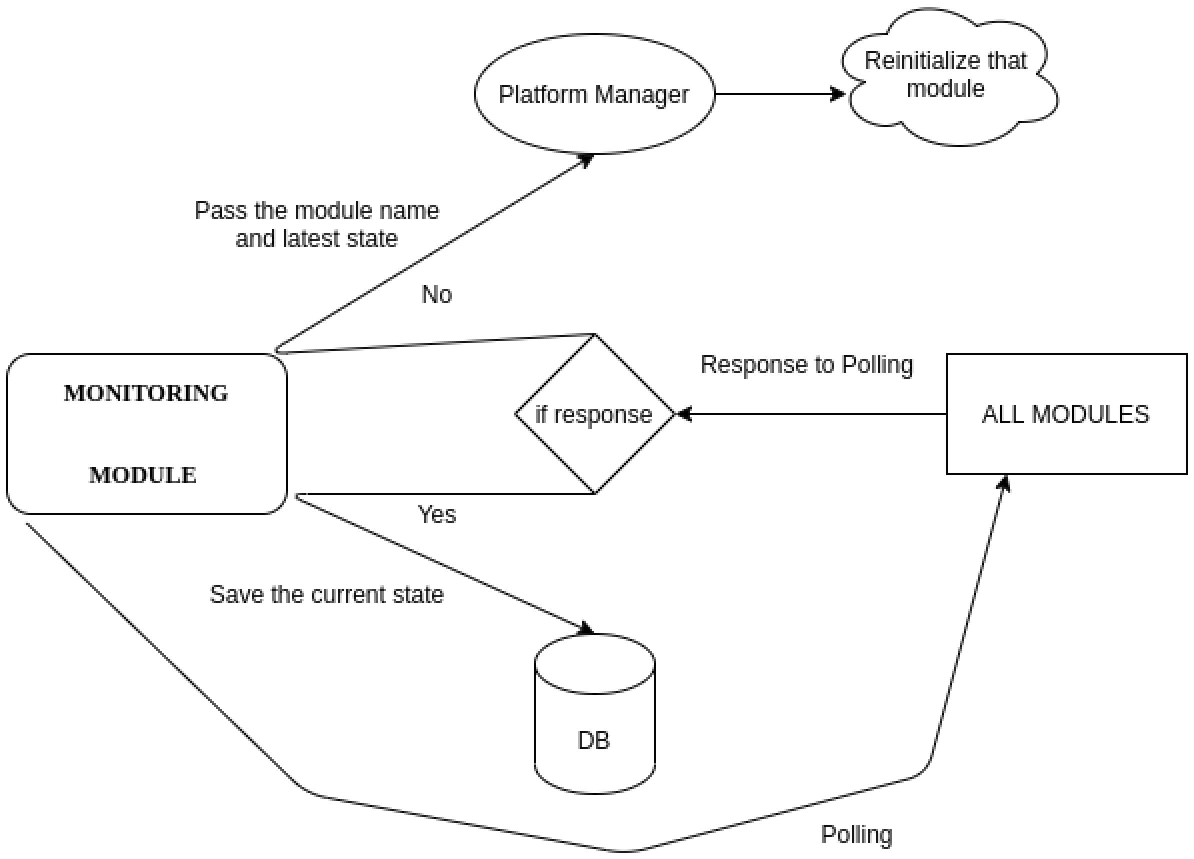
- The module will keep polling to check the current status of all the modules.
- In case no response is received during polling from a specific module, the last consistent state of the respective module will be loaded again from the log files created periodically.

B) Technologies used:

- Socket Programming
- Log files

C) Use Case:

- Send continuous polling messages to all the modules.
- The modules save their last consistent state
- In case of no response, the respective module is reinitialized with its last consistent state.


Platform Manager
Reinitialize that module
Pass the module name and latest state
No
Response to Polling
MONITORING MODULE
if response
ALL MODULES
Yes
Save the current state
DB
Polling

BOOTSTRAP

A) Functional Overview:

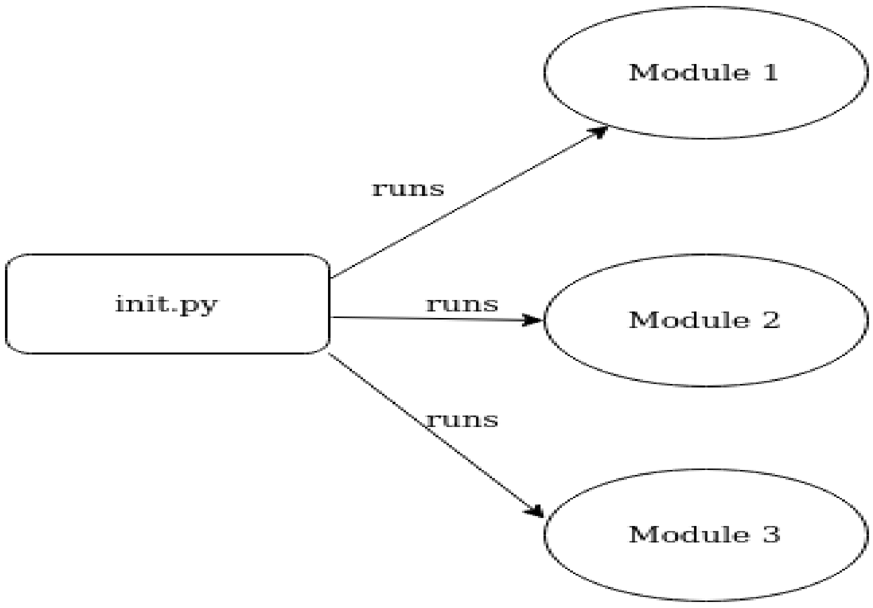
- The module will initialize and run all the modules present on different Virtual Machines.

B) Technologies used:

- Python (multithreading - socket programming)

C) Use Case:

- Initializes all the modules.
- Opens a new terminal screen wherever necessary.


Module 1
runs
init.py
runs
Module 2
runs
Module 3