INTERNALS
OF
APPLICATION SERVER

GROUP – 3
Design Document

AMANPREET KAUR (2020202005)

AMIT JINDAL (2019201037)

ANCHAL SONI (2020201099)

ANUPAM MISHRA (2019201082)

ARCHIT GUPTA (2020201075)

AYUSH KHASGIWALA (2020201088)

AYUSHI PANDEY (2020202010)

KUNAL KANDHARI (20202016)

NAMAN JAIN (2020201080)

NAMAN JUNEJA (2020201072)

SOMYA LALWANI (2020201092)

UPINDER SINGH (2019201083)

TABLE OF CONTENTS

1. Introduction
2. Use Cases/Requirements

3. Test Cases

4. Solution design consideration

4.1 Big picture

4.2 Environment to be used

4.3 Technologies to be used

4.4 Overall system flow and interaction

4.5 IoT system design

4.6 Server system design

4.7 Approach for communication and connectivity

4.8 Registry and repository

4.9 Server and service lifecycle

4.10 Scheduler

4.11 Load balancing

4.12 Interaction between modules

4.13 Wire and file formats

5. Application Model and User's view of system

5.1 Overview of the Application Development model

5.2 Application artifacts

5.3 User admin interactions

6. Key Data Structure

7. Interactions & Interfaces

8. Persistence

9. Modules

1. INTRODUCTION

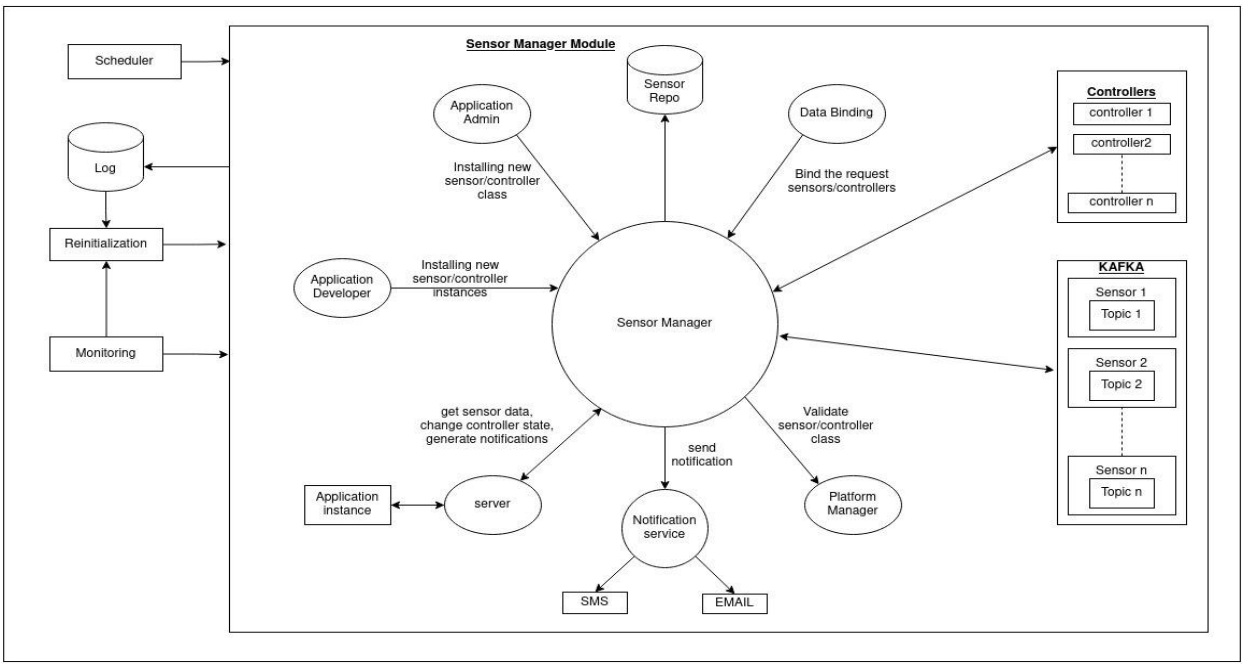
It is an **end-to-end IoT platform applicable for enterprise IoT projects of any scale**. It provides a range of features that enable developers to build advanced applications for smart products and flexibly manage their connected devices. With the IoT features provided by this platform, you can create and deploy your IOT applications.

1.1 Sensor Management & Communication Module:

The sensor management module is responsible for dynamically handling sensors/controllers that is installing them in the platform, managing them, binding them with the application, and providing sensor data requested by application and changing the state of the controller.

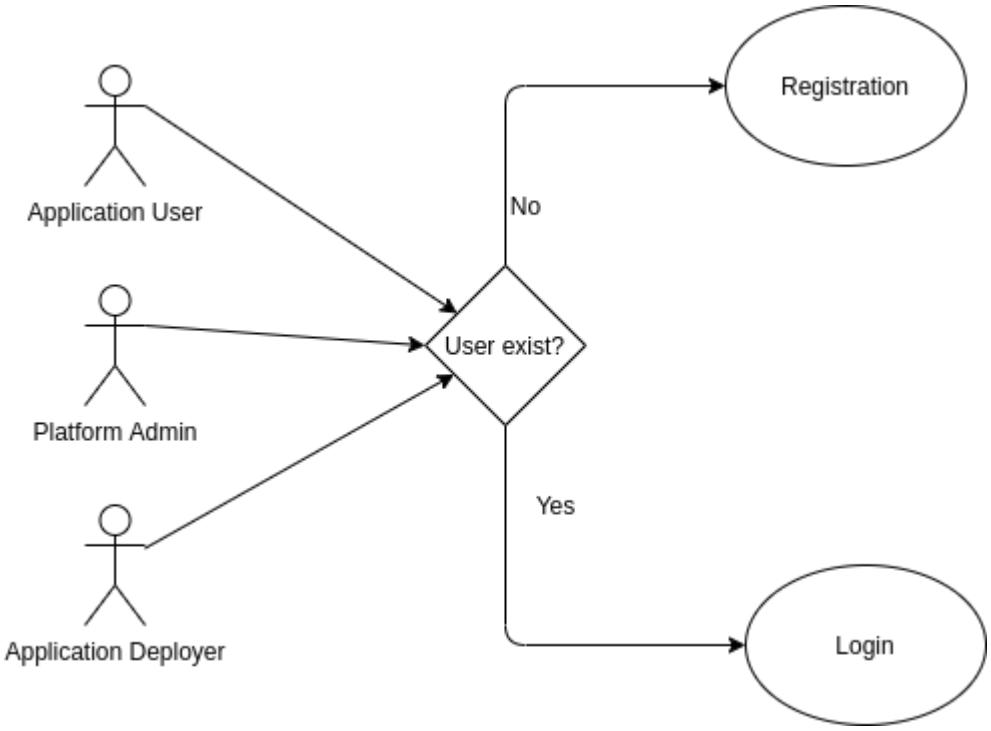
Also notifying the user, application developer upon certain events through the requested notification services.

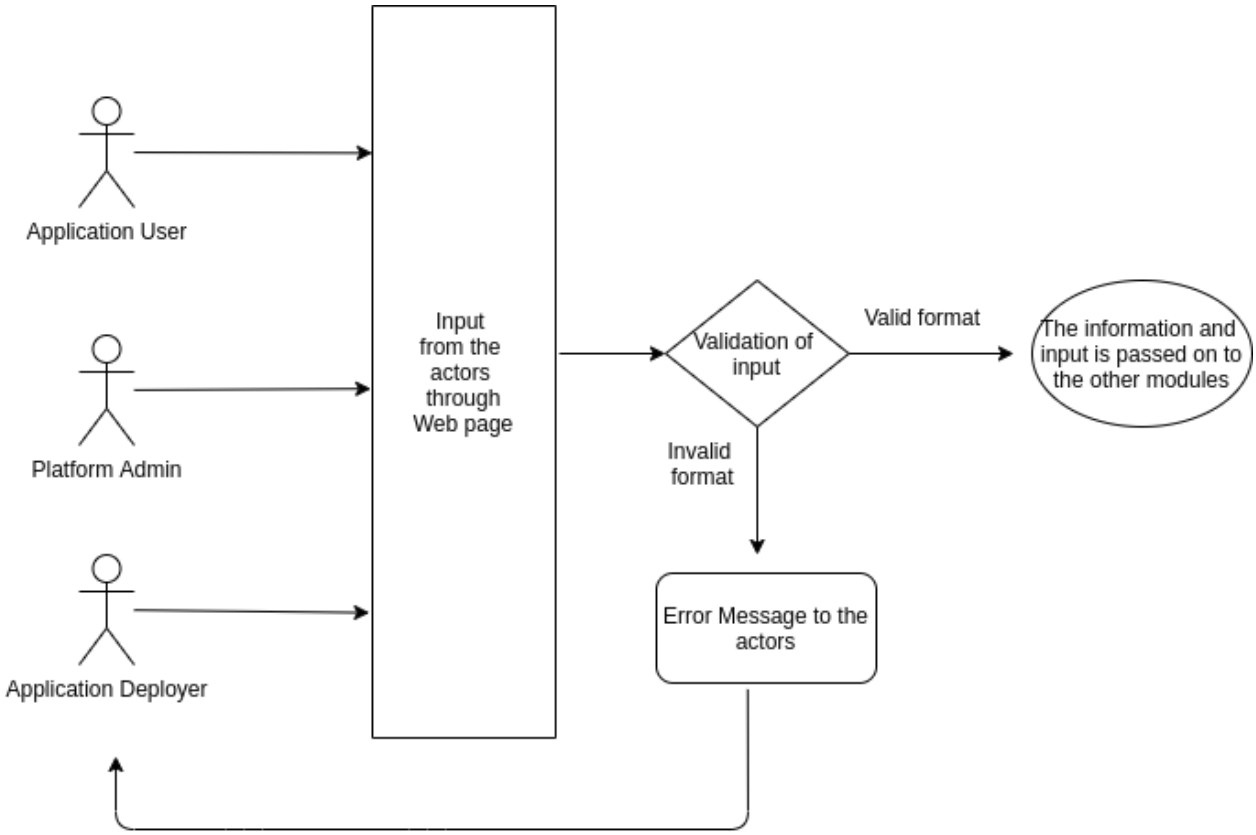
Block diagram of module:



1.2 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.

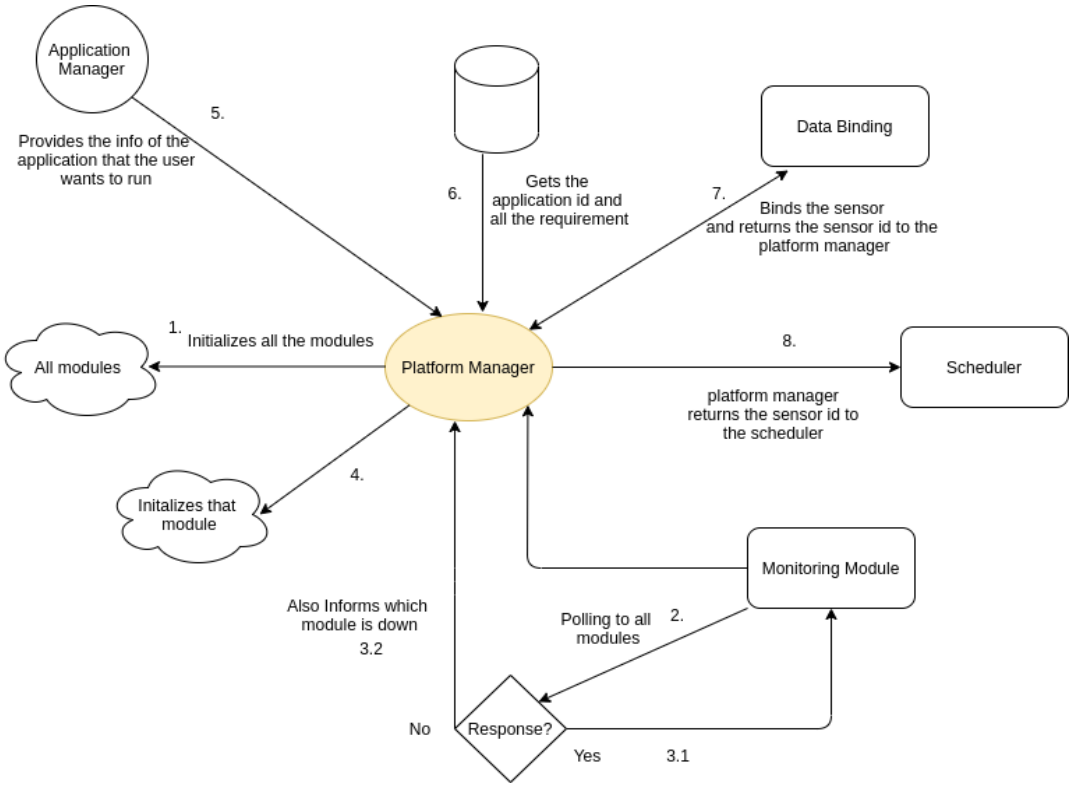




1.3 Platform Manager

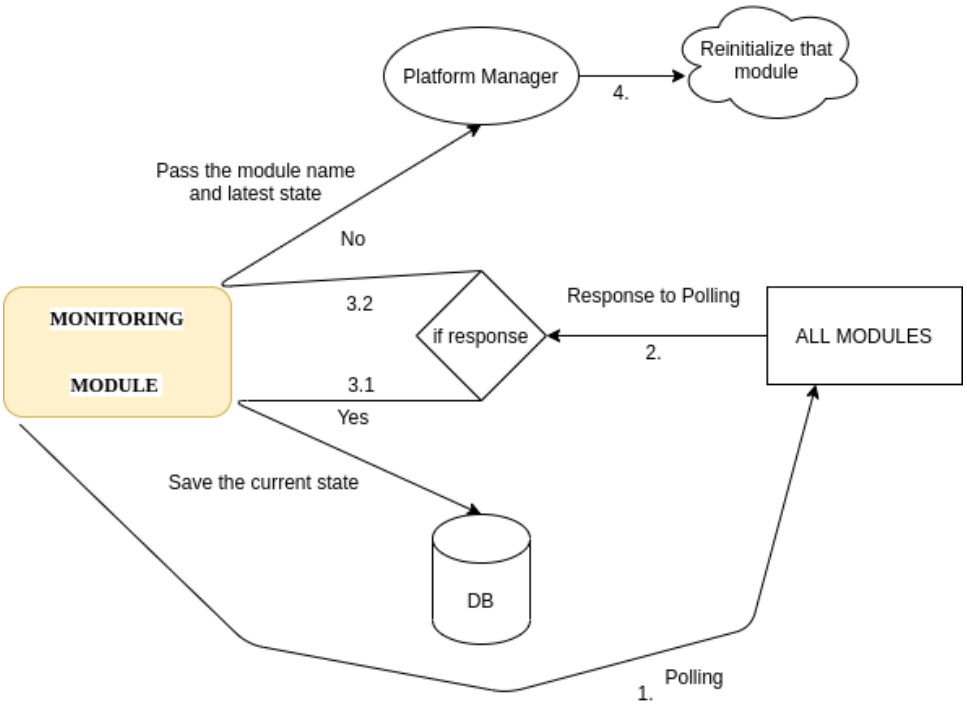
It reads the configuration file received by the Application Manager, the platform is started and accordingly every module is initialized. It initializes every module, repository and registries of the platform. If the system goes down, then the platform can be initialized at that very instant, so it provides fault tolerance(reinitialization).

Make the platform portable by packaging the platform. This package consists of a script file to resolve all the dependencies required by the platform. 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.



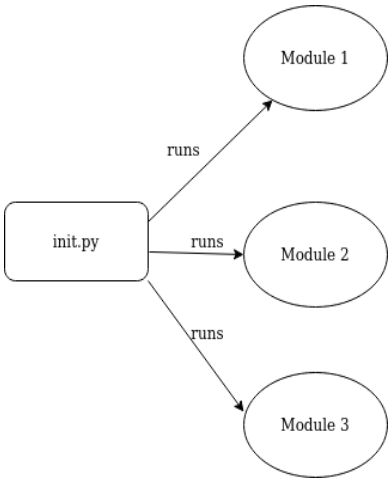
1.4 Monitoring Module

The module will keep polling to check the current status of all the modules. While polling, details about the last consistent state of every module will be sent to the monitoring module and the data received will be maintained in the database. In case no response is received during polling from a specific module, the last consistent state of the respective module will be loaded from the database. The request to reinitialize the respective module will be sent along with the last consistent state of the module.



1.5 Bootstrap Module

- The module will initialize and run all the modules present on different Virtual Machines.
- Opens a new terminal screen wherever necessary.



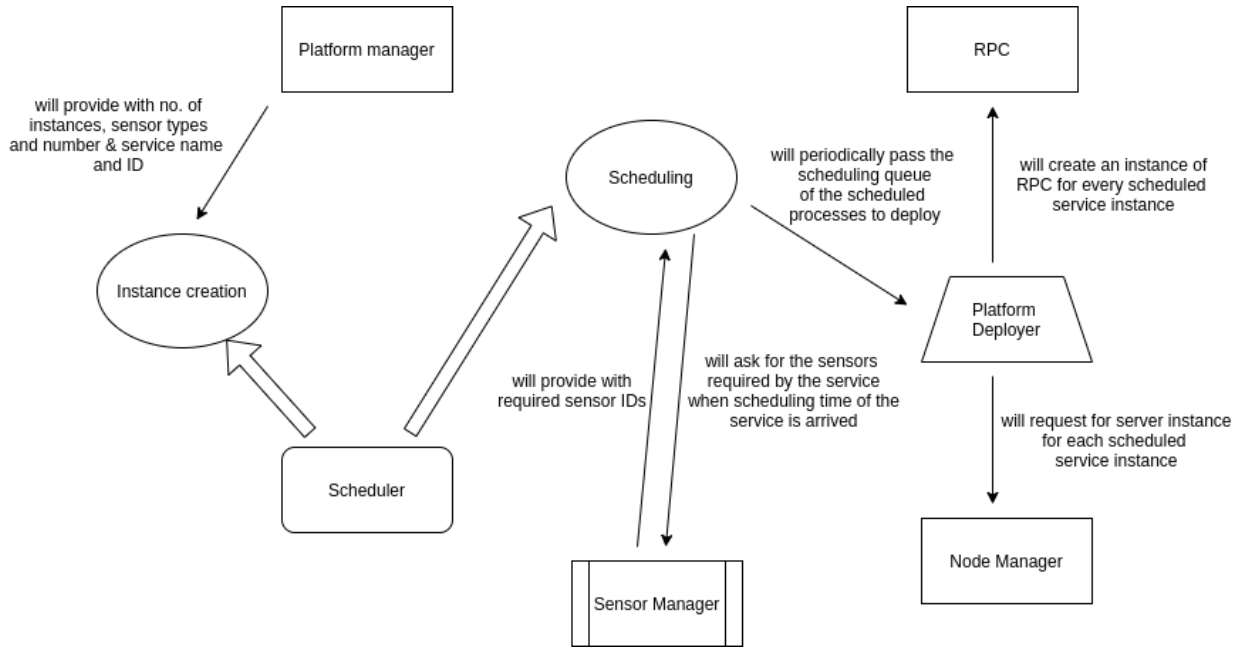
1.6 Scheduling Module

The platform manager will provide scheduler with the information about following:

- Name and id of the service
- number of instances to be created
- Scheduling information (start time, end time)
- types of sensors required for the service to be created.

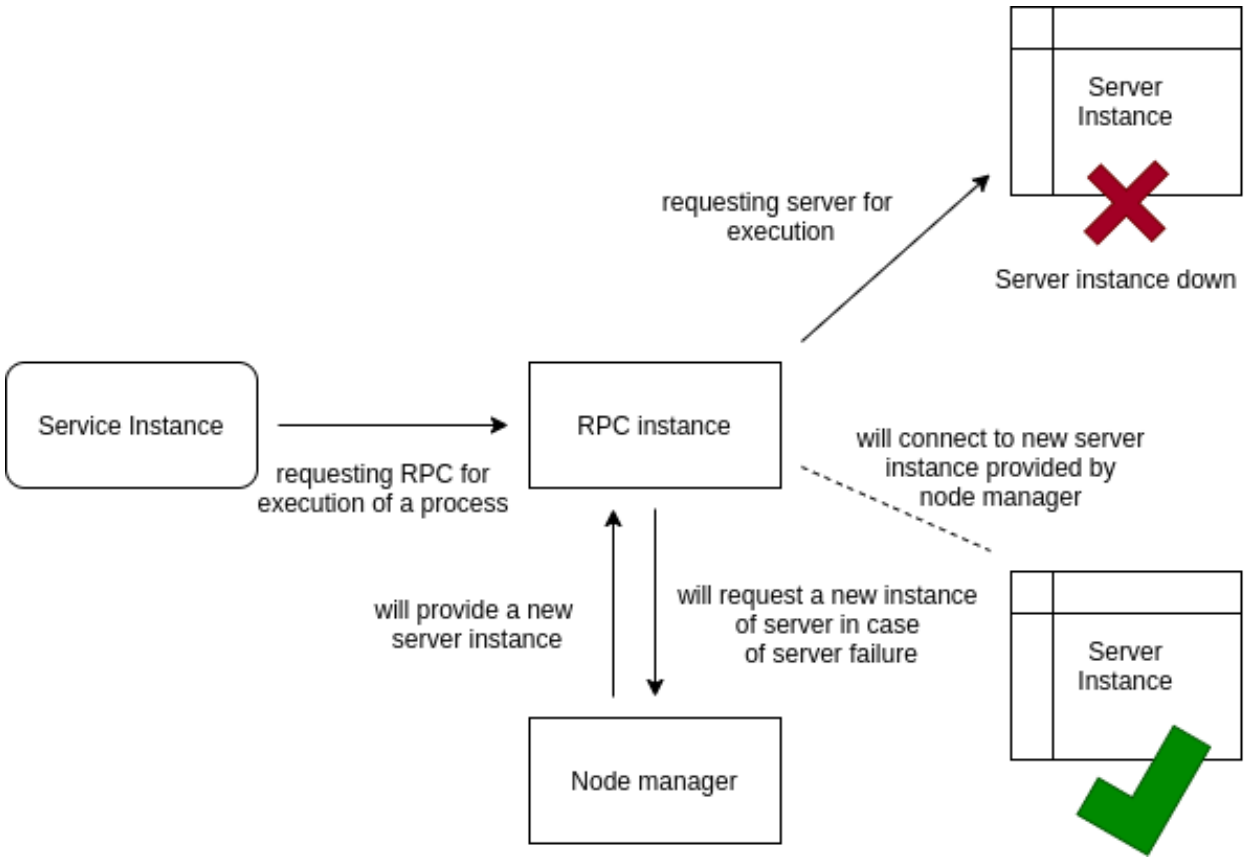
For every instance, a separate thread will be created and required sensor types and number of sensor instances will be binded with each service instance at the time of scheduling. Each thread created will be scheduled according to the scheduling information (start time and end time) in a scheduling queue.

After service instance creation, when the start time of a process comes then the scheduler will first call the sensor manager for required sensor types and number of sensors required which will return the required sensor ids and then sensors to service binding will be performed.



1.7 Fault Tolerance Module

For fault tolerance we will be implementing the above communication using RPC so that if a server goes down during the execution of a process, then the process will wait at that particular piece of code until it gets response from the RPC and the RPC will wait for the node manager to assign a new instance of the server for execution. Now, when a new instance of the server is assigned to the RPC by the node manager the RPC module will resend the previous request again which failed and the service will start receiving the response from the RPC module from that point of code again.



2. Use Cases/Requirements

- **Sensor Management, Communication Module & Notification Services:**

Use cases:

- **Registration:** The application admin will be given the rights to register new sensor/controllers/notification types.
- **Updation:** The application admin will be able to modify the attributes of the existing sensor/controller/notification types.
- **Validation:** Whenever any creation or updation of config files of sensor/controller/notification types happens, those will be thoroughly validated against the required format.
- **Provide sensor data:** An application instance can ask for the data of any sensor/controller and this module is responsible for providing that data to that application.
- **Changing states:** To change the state of the controller based on the output of the application instance.
- **Generate notifications:** To generate notifications based on processing done by application instance. The notifications can be either EMAIL based or SMS based.
- **Fault Tolerance:** If any of sensors or sensor manager goes down, it will automatically resume its services.
- **Logging:** Upon stoppage, sub modules can resume its services from the previous functional state.

- **Application Manager:**

Use cases:

- When the user is not registered, then register the user first.
- If the user is already registered, then check whether the user enters the correct user id and password.
- If, it is correct then Check for the user type and redirect them to their redirect the user to the respective user page
- Then allow the application deployer(actor) to enter the application file and config file.
- If the user is an application user then check whether the application user has entered valid commands and executes those commands internally.
- If the actor is a platform developer then provide different functions like add/modify/delete sensors, notifications and controllers to the admin user, and send these instructions to Sensor Manager.
- 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.

- Check for the format of the config file & extract the requirements.
- Check the format of different files provided by the user.

- **Platform Manager:**

  Use cases:

  - It reads the configuration file received by the Application Manager, the platform is started and accordingly every module is initialized.
  - It initializes every module, repository and registries of the platform and create the running instances of all parts of the platform.
  - If the system goes down, then the platform can be initialized at that very instant, so it provides fault tolerance.
  - It also interact with the monitoring module and the monitoring module keeps an eye on all the different modules and if any of the module goes down due to some reason then the monitoring module sends a msg to platform manager and ask to reinitialize that module
  - When the application user will try to run the application then the application manager will inform the platform manager with the application id
  - Then platform Manager will interact with the algorithm repo and get the information
  - Then Finally platform manager will return the sensor id to the scheduler which will do further job

- **Monitoring Module:**

  Use cases:

  - Send continuous polling messages to all the modules.
  - The modules send their last consistent state as a response to the polling message.
  - Upon successful receival, these states are saved into the database.
  - In case of no response, the respective module is reinitialized with its last consistent state.

- **Scheduler:**

  Use cases:

  - To check the valid start time and end time.
  - To check for the valid no. of instances.
  - To create service instance.
  - Periodically pass the scheduling queue to the deployment manager for deployment of the service.
  - Call the sensor manager for required sensor types and number of sensors to perform binding.

- **Node Manager And Load Balancing Module:**

  Use cases:

  - All the Nodes are busy
    When all the nodes are busy, then which node to be selected to run the service.
  - Requested node available or not
    Requested service node is not available, then the appropriate error message should be sent.
  - No node is active
    Suppose all the servers are not active, then the first task is to up the servers and then run the service.
  - Different Load balancing algorithms
    Try for different load balancing algorithms like:
    • Round robin
    • Less work
    • Hash
    • Power of 2 choices

3 Test cases

3.1 Test cases- used to test the team's module

- **Sensor Management, Communication Module & Notification Services-**

  A. Installation and Modification of sensors,controllers and notifications: Platform Admin can call our module services to perform the following tasks:

  - Installation of new sensors/controller/notifications.
  - Change the configurations of the installed sensors/notifications.

  Input: JSON file

  Output: Boolean (returns true if the configuration file for sensor/controller/notification services provided was valid otherwise false)

  B. Validation of sensor classes: Platform Manager module can call our APIs to validate the sensors/notification classes used in deployed applications.

  Input: JSON file

Output: Boolean (returns true if the application has requested for the correct sensors and in a predefined way)

C. Getting sensor data: Deployed application module can call our API through RPC to get the data from registered sensors for processing the requirement of the deployed algorithms.

API call: getSensorData(sensorId)

Output: List of the data produced by the sensor if the right sensor ID was used to make the request otherwise an appropriate error message is propagated to the user.

D. Changing Controller state: Deployed application module can call our API through RPC to change the state of the controller.

API call: changeControllerState(controllerId, stateField, newValue)

Output: Boolean (returns true if the state of the controller is changed successfully, otherwise if parameters passed to the API are not correct, an error message will be displayed.)

E. Send Notification: Deployed application modules can call our API through RPC to generate notifications based on the processing done by various algorithms.

API call: sendNotification(notificationClass)

Output: Boolean (returns true if the notification is rendered successfully otherwise an error message will be given.)

F. Install New Barricades: Deployed application modules can call our API through RPC to install new barricades.

API call: installNewBarricades(barricadesList), where barricadeList is list of GPS coordinates of all the barricades

Output: Boolean (returns true if barricades are installed successfully otherwise an error message will be given.)

- **Application Manager –**

Test Cases: - Other modules can use the data extracted from the config file by the Application Manager such as the sensor's details, field, controller type and notification type.

Also validated applications can be used by the other modules (deployment module)

Also, the parameters that the user provides to the Application manager will passed to the deployment Module

A] Test Case 1 – User Not Registered

If the user is not registered, then first register it.

Input – User details, as well as user role

Output- Login the user

B] Test Case 2 – User Authentication

If the user is registered, then he can define its role first and then login into the platform using the id and password. The platform will authenticate the user from its database. If the id and password matches, then the user is logged into the platform otherwise an error message is printed.

Input - Login Details

Output- If the details are correct then – User is logged in and directed to the dedicated. user page, otherwise, he is taken back to the login page.

C] Test Case 3 – if the user is Platform Admin.

Then the admin is given a choice to add/modify/delete sensors, notifications and controllers and the admin gives input and then the input gets validated in the backend and passes to the sensor manager and the sensor manager deals with it.

Input- Input from the user through html form

Output – Validation results sent to user and sensor manager(if feasible) both

D] Test Case 4 – Application user input handling

The user who can run the application that is deployed on the platform and can provide input to the application.

Input – parameters

Output- Check for the application to exist and load results on the web page

E] Test Case 5 – Application user input handling

The user who can run the application that is deployed on the platform and can provide input to the application.

Input – no of parameters

Output- Check for the valid number of parameters to run the application load results on the web page

F] Test Case 6 – Application Deployer input handling

The application deployer gives config file and the application file (that needs to be deployed on the platform) and the application manager validates the files and if the format is not correct then error msg is displayed to the deployer otherwise the input files are passed to the other modules for deployment.

Input- Input of the config file and the application file.

Output – validation Message will be displayed on the web page.

G] Test Case 7 – Parsing the config file

This module will parse the config file and get the details from the file and if there is some error in the file, then display the error on the web page otherwise the details will be stored and it will be used by the other modules for their working.

Input- Config file

Output- validation message

H] Test Case 8 - Format checking of files provided by application deployer

Different files (config files) provided by the user are checked for their correct format.

Input - Different files

Output - validation message

I] Test Case 9 – Web page is closed

If all the web pages to this login/signup pages are closed.

Input – Deploy an application and then close all possible web pages to connect the user and start again to login

Output- Login the user and the application deployed must be in running state with all the components being restored properly

J] Test Case 10 – Interaction with other modules

After checking valid formats, the instructions or files are passed to various modules.

Input – Data to be sent and checking for connection to be present.

Output- For a valid and secure connection, send the data; else report connection error.

- **Platform Manager-**

Firstly, it initializes all the modules and this module interacts with the monitoring module and the module will inform this module when and which modules are down and then the platform manager will reinitialize the module.

Test Case 1 :- **Initializing the connection between different modules before calling them**

In the initial state, the platform manager starts all the modules that will be required at that time. So before initialising these modules, we need to check the connection between PLatform Manager & the respective module.

Input – No input, different modules are connected in code

Output- All the modules are connected, and if not error is thrown.

Test Case 2 :- Initializing the different modules

In the initial state, the platform manager starts all the modules that will be required at that time

Input – No input, different modules are called in code

Output- All the modules are initialized and instances of all the modules are created, and if not error is thrown.

Test Case 3 :– If the platform or the modules goes down

If a situation occurs where the platform goes down due to some reason, then this module will initialize all the modules and load the modules with their last state.

Input – get the module name and the last state of the module form the monitoring module

Output- reinitialization of module

Test Case 4 :– Interaction with Sensor Manager

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

Input – checking for connection to be present.

Output- For a valid and secure connection, send the data; else report connection error.

Test Case 5 :- Receiving the application name from application manager

The application manager will provide the application name to the platform manager and so that platform manager can get the sensor info.

Input - application name from the end user

Output - interact with algo. repo

- **Monitoring Module-**

This module polls every other module and keeps saving their current data in the database so that this data could be used in case of failure of the module and reinitialization process.

Test Case 1- No reply to polling messages

If any module does not reply to the polling messages, then the monitoring module must retrieve its last saved state for reinitialization.

Test Case 2- Passing module and latest state to Platform Manager for reinitialization

If any module does not reply to the polling messages, then it gives its previous saved state to the platform manager so that it could reinitialize

Test Case 3- Saving state in db repo

The connection with repo(db) must be made and securely data must be sent & saved, if any error it must be reported

- **Scheduler:**

Test Cases:

1. **Validating the start and end time:**

API call can be made to start_end_time_validator providing the input JSON consisting of the various details including start time and end time of a process which will return Boolean after checking if the start and end time are valid and can be scheduled.

Input: JSON (config file consisting start time and end time).

Output: Boolean

2. Validating before instance creation:

API call will be made to the instance_validator which will return true if an instance of a particular service can be created or not. This will include various checks including constraints like server availability, sensor availability etc.

Input: JSON(file consisting of sensor type, no. Of sensors, controller information)

Output: Boolean

3.2 Project Test Cases

- **Integration testing**

- Providing different type of application deployer files and deploying simultaneously
- Checking whether each module is calling other modules properly.
- Checking for load balancing scenarios
- Checking for fault tolerance scenarios
- Checking for connectivity between different modules and their interactions.
- Validating data flow between various modules.
- Only appropriate and valid data exchanges take place.

- **How & what to test, after integrating with other modules**

- Connectivity between different modules via sending and receiving data between modules.
- Trying to check if application can handle different types of users simultaneously.
- Trying to check if the application can handle different applications being deployed simultaneously.
- Checking if any module fails, the system doesn't go down or produces wrong or faulty outputs.
- Calling for different services from other services and verifying outputs.
- If any module is not working, Platform manager should poll and get information and finally reinitialize it.

4. Solution design considerations

4.1 Big picture

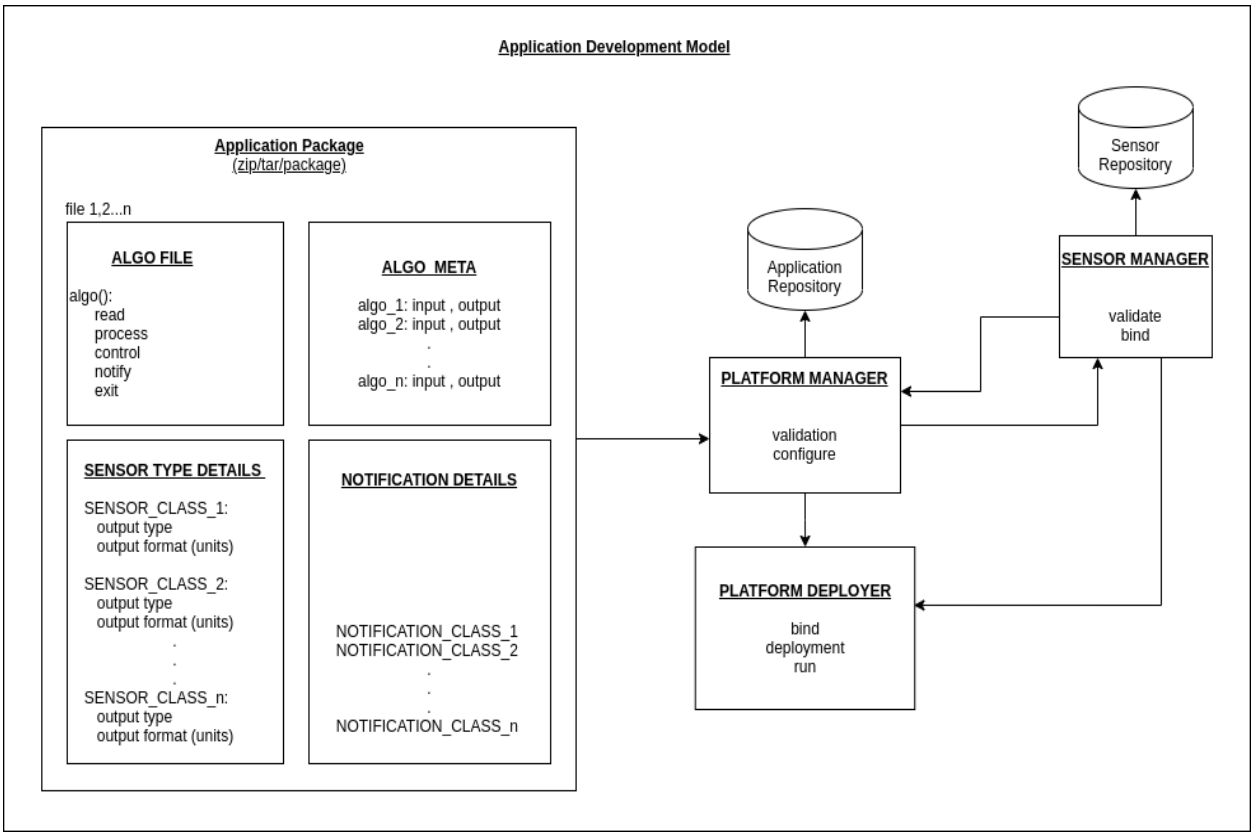
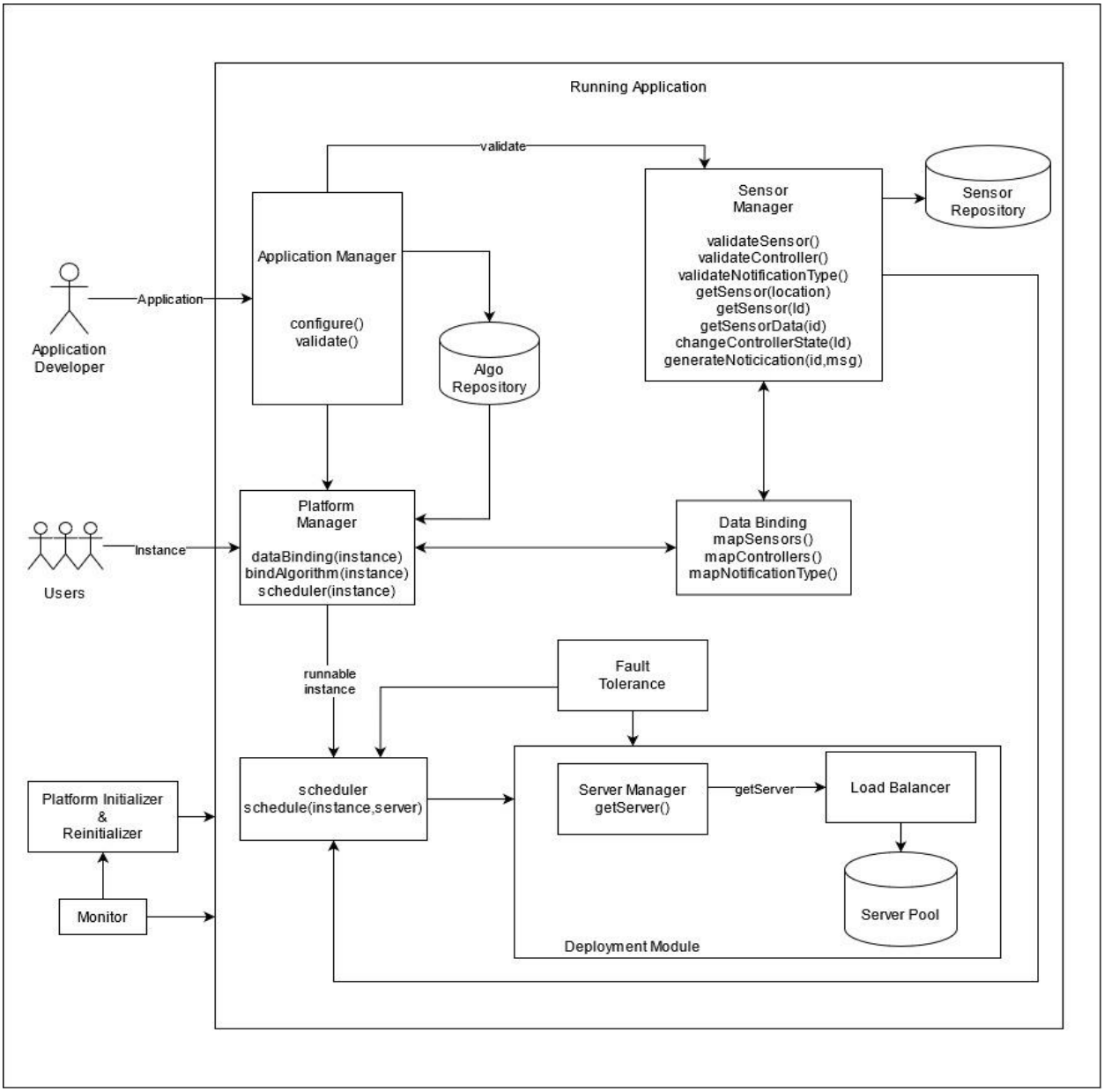


Diagram when an application runs:



4.2 Environment to be used

- 64-bit OS(Linux)
- Minimum RAM requirements: 4GB
- Processor: Intel i3 5th Generation

4.3 Technologies to be used:

- **Flask:** It provides simplicity, flexibility and fine-grained control. The APIs exposed from the sensor manager will be responsible for providing communication between various other modules.

- **Kafka:** Sensor data generated is stored in Kafka topics which is then used by different sub modules and methods.

- **Mongodb:** It is the sole database repository for our platform containing collections corresponding to each asset such as sensor classes, sensor instances, notification types etc.

- **Azure VMs:** The Virtual machines on the cloud platform Azure are used to host the various modules of our platform and communications between them is handled in a distributed manner.

4.4 Overall system flow & interactions

- For deployment of the application, the application developer will connect to the platform on the given port. On connecting to that port the application developer will get a package describing the format of all the artefacts (configuration files, interface, APIs) required to deploy application on his platform.
- Then the application developer will provide the package to the platform containing all the artefacts.
- The platform manager will validate the provided package
- On successful deployment, user can connect to the platform on the same port. And platform manager will display the algorithms meta file to the user. And list of all available location where this algorithm can run i.e., there instance can be created.
- User will in return provide the algorithm name and location where he wants to run that algorithm.
- Platform manager will call sensor manager providing it the name of location to get the configurations details of all the sensors of that location for binding them.
- After successful binding of sensors to application instance, application will run and various APIs calls will be made for the sensors and notification using their respective APIs.

4.5 IOT system design

As you can see, the Internet of Things brings about a range of entirely new experiences. All of these experiences will require new modes of interaction. And those will have to be designed carefully.

Designing an IoT solution comes with a set of totally new design challenges. Consider that IoT systems usually consist of multiple elements such as physical device sensors, actuators, and

interactive devices, the network that connects these devices, as well as the data gathered and then analyzed to create a meaningful experience.

That's why IoT implementations require various types of design – from industrial product design to service and business design. All of these factors impact the total user experience on the IoT system.

Our application will provide a generic method for deploying any type of application – from industrial product to business design, which will also allow any type of user to use our application.

4.6 Server system design

a. Applications are usually designed so that one computer acts as a server, providing a service to other computers on a network. To access a server, a program is run on a user's computer, this is called a client program. The program establishes a connection through the network allowing communication with the server.

We also have a server-client similar application, the server is once started connects to the repository & other modules. The user (we have defined 3 types of user- application deployer, application user, platform admin) can enter from the client side of our application and perform specific tasks.

b. Systems design is the process of defining elements of a system like modules, architecture, components and their interfaces and data for a system based on the specified requirements.

Before implementing our project, we have also defined all the modules which will be used, designed those modules accordingly. Also, we have defined the interaction between various modules beforehand for an easy workflow during the implementation of the project.

4.7 Approach for communication & connectivity

Although modules may appear as self-contained elements, because they are all parts of a larger whole there must always be relationships between them. we say that the interface to a module defines how other modules can use that module. Sometimes, we can define more than one interface for a module to allow yourself to be more precise about which services will be offered to different kinds of client.

4.8 Registry & repository

Sensor Repository: A Database used to store the configuration details of the installed sensors/controllers/notification types and their fields.

Data Repository: It will store the state information for every module which helps in monitoring and fault tolerance and proper communication among various modules.

Algo Repository: It will store the algorithmic logics of different services deployed by the developers.

Logging Repository: A directory of log files used to store the logs of sensor manager, sensor data and controller state.

4.9 Server & service lifecycle

Server life cycle is the series of states through which a Server instance can transition. These states cause specific changes to the operational state of a server instance and help to identify the accurate status of the running server.

4.10 Scheduler

This module will handle the following tasks:

1. creation of instances of the service
2. scheduling of the services by requesting the server module and sensor module for instances of server and sensors.

After service instance creation, when the start time of a process comes, the scheduler will first call the sensor manager for required sensor types and number of sensors required which will return the required sensor ids and then sensors to service binding will be performed.

4.11 Load Balancing

Load balancer module will ensure the load on any instance will be within certain threshold value.

For load balancing we can use following algorithms:

- *Round robin*: Requests are distributed across the group of servers sequentially.
- *Less work*: Assign more tasks to the servers by performing less (the method can also be weighted.
- *Hash*: allocates queries according to a hash table.
- *Power of Two Choices*: pick two servers at random and choose the better of the two options.

4.12 Interactions between modules

This module will handle communication between all the modules of the system such as:

a. Platform Manager and Sensor Manager:

Platform manager will provide the configuration file of the sensors to the sensor manager which in return will validate the configuration with its sensor repository.

b. Scheduler and Deployer:

Scheduler will provide the application to be run to the deployer which has to be deployed.

c. Deployer and Server Life Cycle Manager:

Deployer will ask Server Life Cycle Manager for instance of server on which application can be deployed.

d. Fault Tolerance and Monitoring and other modules:

This will run in parallel with all the modules in the system and perform action according to any fault that occurs during execution of those modules.

e. Sensor Manager and Node:

Node of the deployed application will call Sensor Manger to perform following operations:

Application can call our APIs through RPC to get the data from registered sensors for processing the requirement of the deployed algorithms.

Application can call our APIs through RPC generate notifications based on the processing done by various algorithms.

f. Application Module and Platform Initializer:

The platform initializer reads the configuration file received by the Application Manager, the platform is started and accordingly every module is initialized.

g. Applications Module and Sensor Manager:

The Application Module **p**rovides provision for admin login and gives him/her admin privileges such as add/modify/delete sensors, notifications and controller. Send these instructions to the Sensor Manager Module.

h. Application Manager and all other modules:

The parameters given by the user will be validated and then passed to the other modules as per their needs (only if required).

i. Platform Manager and all other modules:

Platform Manager reads the configuration file & initialize modules, repository and registries. If the system goes down due to some reason, then we can use the platform initializer to reinitialize the platform.

j. Platform Manager and Scheduler:

The requests received by the Platform Manager are sent to Scheduler, which schedules the calls/requests.

k. Monitoring Module and all other modules:

The monitoring modules will keep polling to check the current status of all the modules. The modules send their last consistent state as a response to the polling message. Upon successful receival, these states are saved into the database. In case of no response, the respective module is reinitialized with its last consistent state.

4.13 Wire and file formats

Four types of config files will be generated in the platform-

1. Sensor binding
2. Deploying
3. Developing
4. Packaging

Out of these files, Sensor binding config file will be taken as input from application manager and rest will be generated while application development. All of them will be in JSON format.

After validation of sensor binding config file, a deploy config file will be generated containing scheduling information.

5 Application Model and User's view of system

5.1 Overview of the Application Development model

- **What is an application- for each type.**

  An application, also referred to as an application program or application software, is a computer software package that performs a specific function directly for an end user or, in some cases, for another application. An application can be self-contained or a group of programs. The program is a set of operations that runs the application for the user.

- **What are the files (config, et al)**

  Different config files will be provided by application developer for deploying an application like sensorConfig.json, notificationType.json applicationAlgorithm.py, applicationMetaData.py

- **How are analytics rules defined**

  Whenever application developer wants to deploy any application to the platform, first it needs to connect to platform manager and get the submission format consisting of various

configuration files and accordingly application developer will create the submission module.

- **Structure of the files**

  All the input files are in json format. The config files from the application developer will contain the sensor type and the sensor controller information. Also it will provide a Algorithm file containing the logic of the application to be deployed.

  The end user will provide a file containing the Application name(ID), Parameters and the Location(class) of the sensors in json format.

- **How & Where is it deployed**

  As soon as the platform is initialized, the Platform Manager module will be called by application deployer which will read the configuration file and then it will do all validations on sensors, notifications like type checking and config checking. Accordingly initializes every module, repository and registry of the platform. If the system goes down then platform can be initialized at that very instant so it provides fault tolerance. This will further allow for more interoperability between components from various sources.

  Also, make the platform portable by packaging the platform. This package consists of a script file to resolve all the dependencies required by the platform.

  The scheduler will periodically pass the scheduling queue to the deployment manager for deployment of the service.

  Scheduler will provide the service instance (at scheduled time) to the platform deployer and now the platform deployer will call the node manager requesting the server and on receiving the ip address of the server the binding of the instance with the server will be done.

- **How to start/access/invoke/stop**

  To start application developer will connect to platform manager, which will invoke/initialize all the modules. The monitoring module will keep on checking if the module is alive or dead, if finds the module dead – it reinitializes the respective module.

5.2 Application artifacts

- SensorConfig.json
- notificationType.json
- ApplicationAlgorithm.py
- applicationMetaData.py

5.3 User admin interactions

- User Interaction

  We have defined 3 types of users – application user, application deployer, platform administrator. Every user has its own functionality. The Platform Admin will provide provision for admin login and give him/her admin privileges such as add/modify/delete sensors, notifications and controller. The Application User can run the application that is deployed on the platform and can provide input to the application. The Application Developer will provide the application and the config file to the platform for application deployment.

- Configurations

  All the files will be sent to the application for deployment which is explained in the individual modules.

- Deployment

  Applications could be deployed by the application deployer by giving the sensor, controller and notification requirements.

6. Key Data structures

- The application packages need to be delivered in the zipped format to that platform containing all the required files.
- The zip file must contain config files of the sensors/controllers/notification types in the required format.
- Also along with that it must have the python files having the 6.1application logic.
- Sensor/controllers details will be stored in Sensor Repository.
  - Solution level data structures
    → Sensor Repo
    → Algorithm Repo
    → Server Pool Repo
  - Wire formats

    Interaction between different modules is done using either json files or parameter passing methods.

  - APIs - interaction data objects

    All the modules will interact with each other by calling respective APIs.

  - App Logic definitions

    Application repository will store the algorithmic logics of different services deployed by the developers.

  - User's UI

    The front-end of the application will allow users to deploy, manage and use the application by providing interface to give input to our application.

7. Interactions & Interfaces

Various modules will interact with each other using API calls. Using API calls various methods will be called to do appropriate functions. Interactions between various modules is specified in 4.13

8. Persistence

1. Config Info

The repository will store following data -

- The configuration details of the installed sensors/controllers/notification types and their fields
- The state information for every module (helps in monitoring and fault tolerance and proper communication among various modules)
- The algorithmic logics of different services deployed by the developers.

2. Transient State

The state information for every module (helps in monitoring and fault tolerance and proper communication among various modules)

9. The modules

1. Sensor Management, Communication Module & Notification Services:

i. Lifecycle of the module

– How will the module/its components be started, monitored and stopped

This module will be started at the start of the platform and will remain up till the end of platform lifecycle. The modules will be managed by Monitoring module via continuous polling.

– How will any user's application components need be made available/deployed/setup on this module.

Platform Admin/Application Developer/ User applications can access our module via API calls. First, when the platform manager gets the zip file of an application from the application developer to be deployed on the platform, it calls the sensor configuration validator (APIs) to check if the application developer has correctly specified the sensors' requirement. After validation, when the user runs desired algorithms of the application,

those algorithms will access the data of preferred sensors via the API calls (such as: getSensorData(sensorID)) that will be exposed by the sensor management module and for changing the state of the controller a generating notifications respective APIs will be called.

– Considerations

The class of sensor and the notification is known to the application developer who is writing the algorithms that are accessing these sensor classes and/ or notification classes. This information can be made available to the app developer during the initial phase of the development of his application.

ii. List the sub modules

(A) Communication module

The key major components of this module are:

- Interaction with modules
- Interaction with sensors
- Interaction of different actors with the platform.

(B) Sensor management

The various operations handled by sensor management module are:

- Sensor Manager
- Sensor config validator
- Sensor Data management.
- Controller state changer.

(C) Notification service.

For generating notification based on actions triggered.

iii. Block diagrams

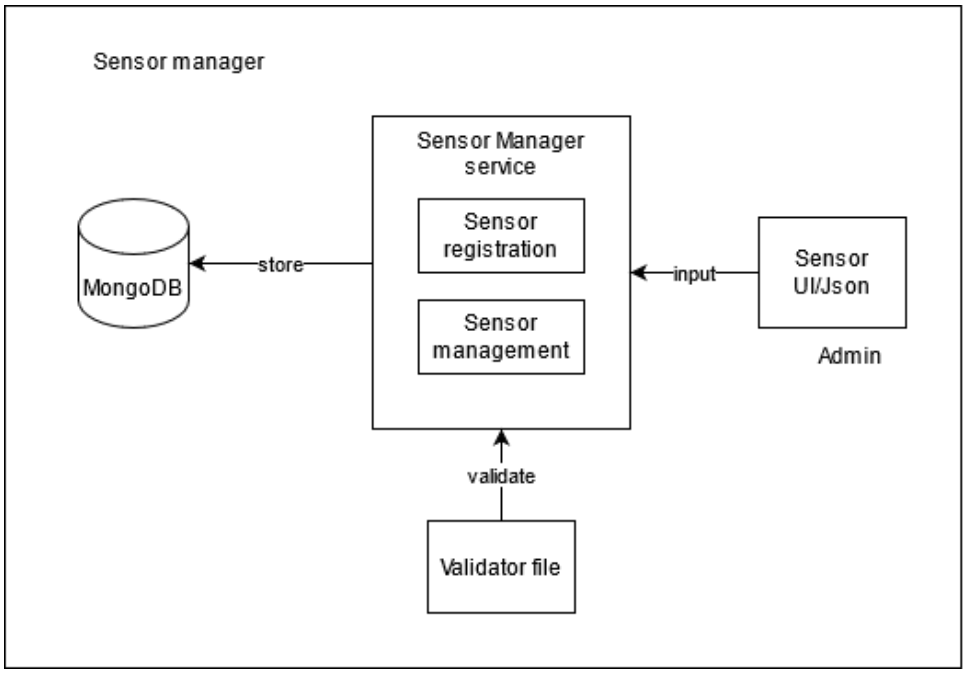


Fig: Sensor Management Service

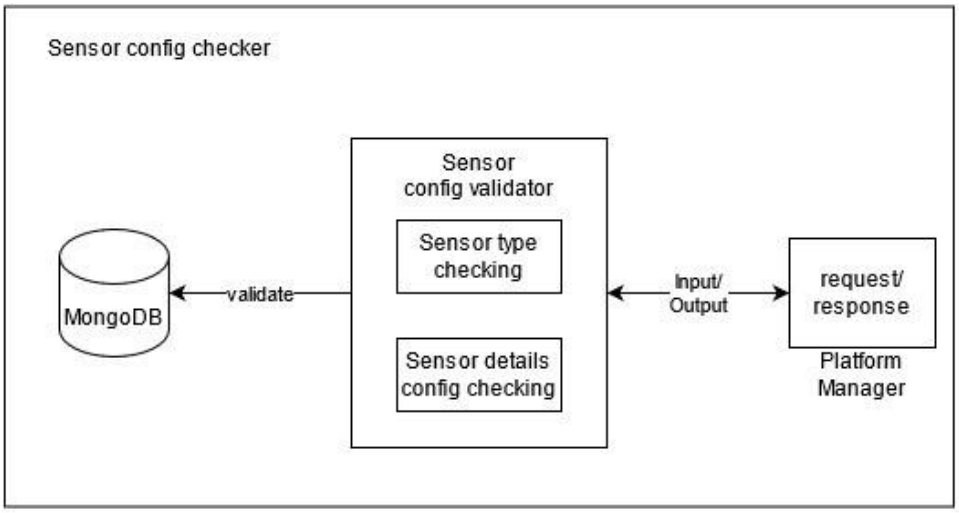


Fig: Sensor config validator

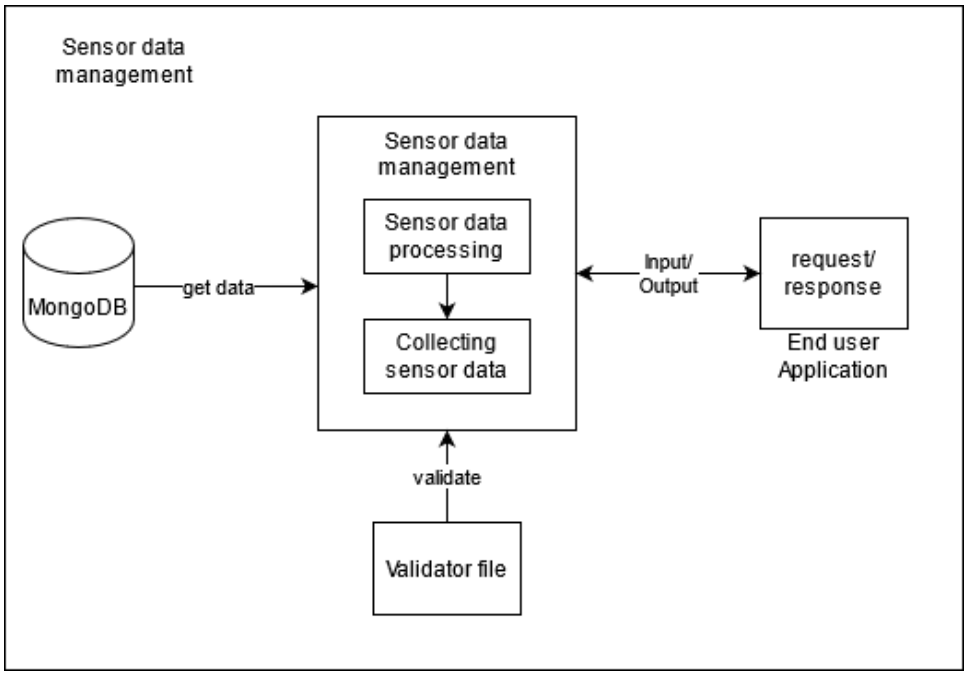


Fig: Sensor Data Management Service

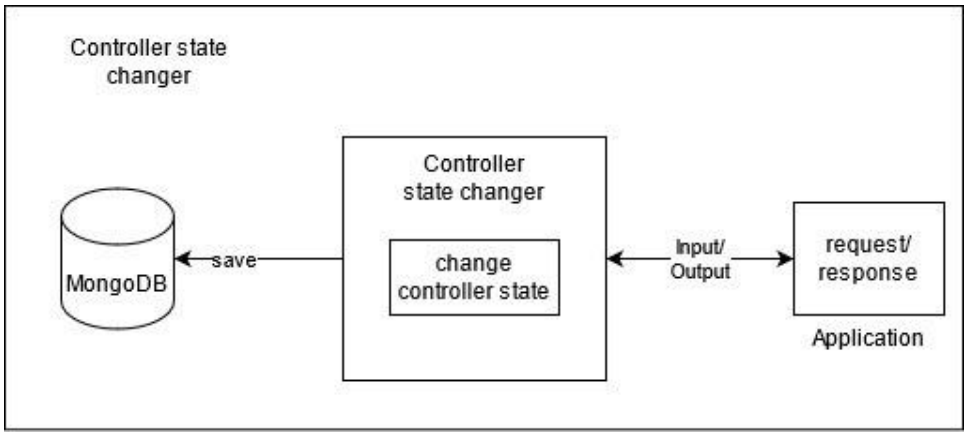


Fig: Controller Management Service

iv. Describe the interactions among sub-modules

(1) Sensor Management Service and Sensor Config Validator:

Upon receiving the application package from Application Developer, sensor management service will call the sensor config validator to validate the various config files int the application package for type and class checking of sensor and notification config.

(2) Sensor Data Management Service and Controller Management Service:

If there is a requirement to call the controller in order to change the state of the environment then sensor data management service will call the controller management service using API.

(3) Sensor Data Management Service and Notification Service:

Upon certain event such crossing certain threshold then sensor data management service will invoke the notification service mentioned by the Application developer.

2. Instance Creation, Scheduling and Deployment manager

i. Lifecycle of the module

How will the module/its components be started, monitored and stopped (if applicable)?

This module will be started at the start of the platform and will remain up till the end of platform lifecycle. The modules will be managed by Monitoring module via continuous polling.

How will any user's application components need be made available/deployed/setup on this module.

The platform manager will provide scheduler with the information about following:

- Name and id of the service
- number of instances to be created
- Scheduling information (start time, end time)
- types of sensors required for the service to be created.

For every instance, a separate thread will be created and required sensor types and number of sensor instances will be binded with each service instance at the time of scheduling. Each thread created will be scheduled according to the scheduling information (start time and end time) in a scheduling queue.

After service instance creation, when the start time of a process comes then the scheduler will first call the sensor manager for required sensor types and number of sensors required which will return the required sensor ids and then sensors to service binding will be performed.

The scheduler will periodically pass the scheduling queue to the deployment manager for deployment of the service.

Scheduler will provide the service instance (at scheduled time) to the platform deployer and now the platform deployer will call the node manager requesting the server and on receiving the ip address of the server the binding of the instance with the server will be done.

Any other considerations (based on the module's functionality)

For fault tolerance we will be implementing the above communication using RPC so that if a server goes down during the execution of a process, then the process will wait at that particular piece of code until it gets response from the RPC and the RPC will wait for the node manager to assign a new instance of the server for execution. Now, when a new instance of the server is assigned to the RPC by the node manager the RPC module will resend the previous request again which failed and the service will start receiving the response from the RPC module from that point of code again.

ii. <u>List the sub modules</u>

(A) Instance creation module

Key major components of this module are:

- Take request from platform manager module
- Interaction with sensor manager
- Interaction with server lifecycle manager
- Creation of new instance of service

(B) Scheduling

Key major components of this module are:

- Requesting server instance from server module
- Requesting sensor instances from sensor module
- Passing the scheduler Queue consisting of the service requests to be scheduled.

(C) Fault Tolerance

Key major components of module are:

- Implementation of RPC based communication between scheduler and node manager in order to continue from the same state in case of server failure.

iii. <u>Brief overview of each sub module</u>

Sensor Manager:

This module is responsible for managing the sensors on the platform. It will allow various functionalities like registration of new sensors, managing them, etc.

Describe the functionality of the module

- Sensor registration
- Sensor Management
- Controlling sensor data flow rate

List the interactions

- Platform admin will interact with the Sensor Manager module.

3. Sensor config validator

Describe the functionality of the module

- Sensor configuration checker.

List the interactions

- Platform Manager will interact with Sensor config validator module.

Sensor Data management

- **Describe the functionality of the module**

  Sending sensor data
  Sensor data processing

- **List the interactions**

  Applications will interact with Sensor Data Management module.

  APIs:

  - getSensors(locationName)
  - getSensorData(sensorId)
  - generateNotification(notificationclass)

iv. <u>Interactions between other modules</u> (describe as relevant)

The platform Admin will have the option to register new sensor(s) onto the platform, the config file will be provided by him and after validation of that file, the sensor will be stored in the repository and installed on to the platform.

After the application is deployed on the platform, a request can be made to for sensors' data. On top of that, various notifications can be generated based on the current state of the sensors and their can be callbacks that can be triggered based on the events performed.

4. APPLICATION MANAGER, PLATFORM MANAGER, MONITORING MODULE, BOOTSTRAP MODULE

i) Lifecycle of the module

- How will the module/its components be started, monitored and stopped (if applicable).

This module will be started at the start of the platform, initialized by platform initializer and will remain up till the end of platform lifecycle. The modules will be managed by monitoring modules via continuous polling.

- How will any user's application components need be made available/deployed/setup on this Module. The web page will also get the application file for the platform to be deployed and respective application requirements from the user.

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.

The Application Development user who can run the application that is deployed on the platform and can provide input to the application. The Platform Manager reads the configuration file received by the Application Manager, the platform is started and accordingly every module is initialized.

- **Any other considerations (based on the module's functionality)**

The zip file name and the name of the main file with details must be the same and should be the respective Application's name.

ii) Block Diagrams of different sub module

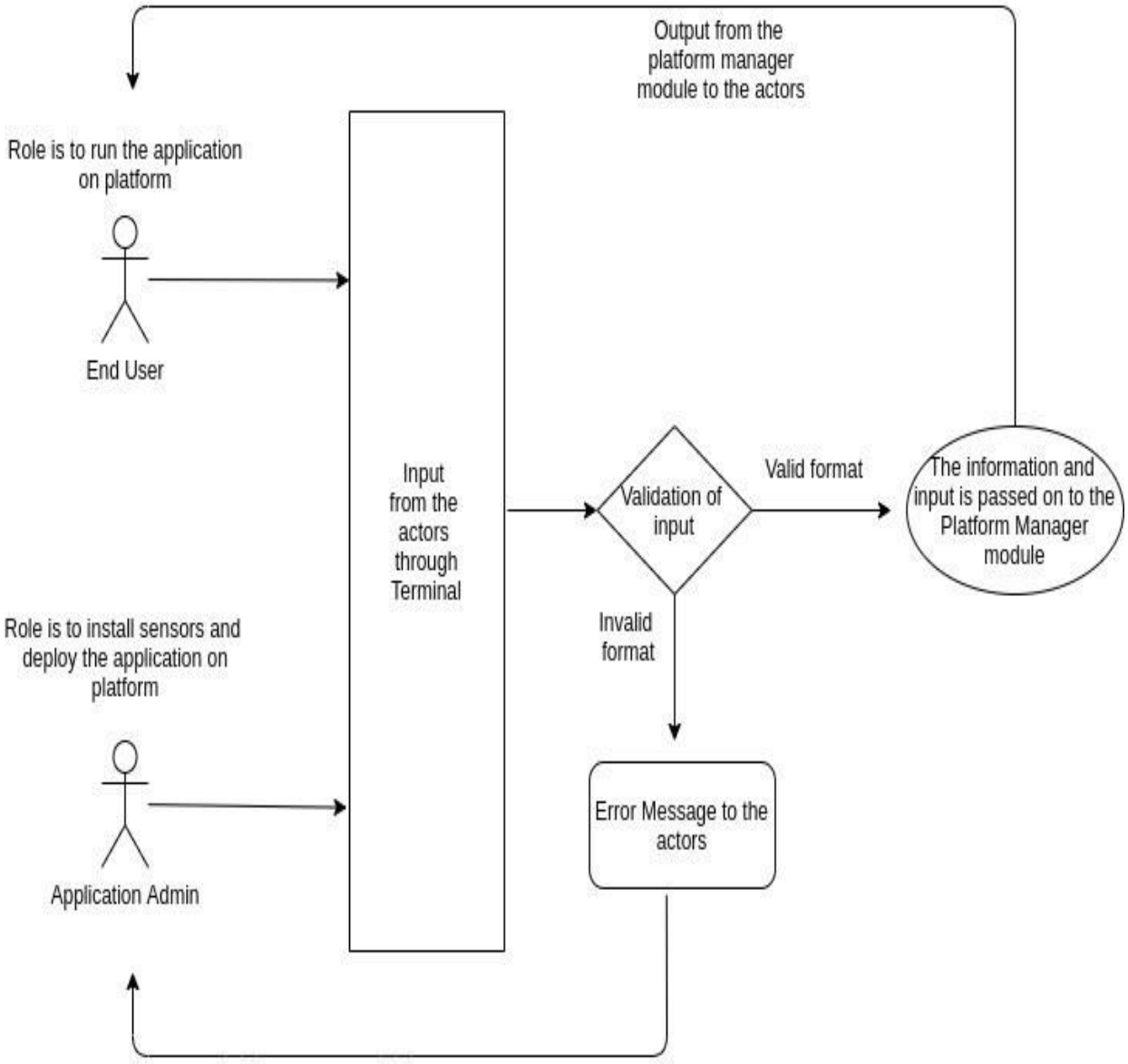


Figure: application manager module

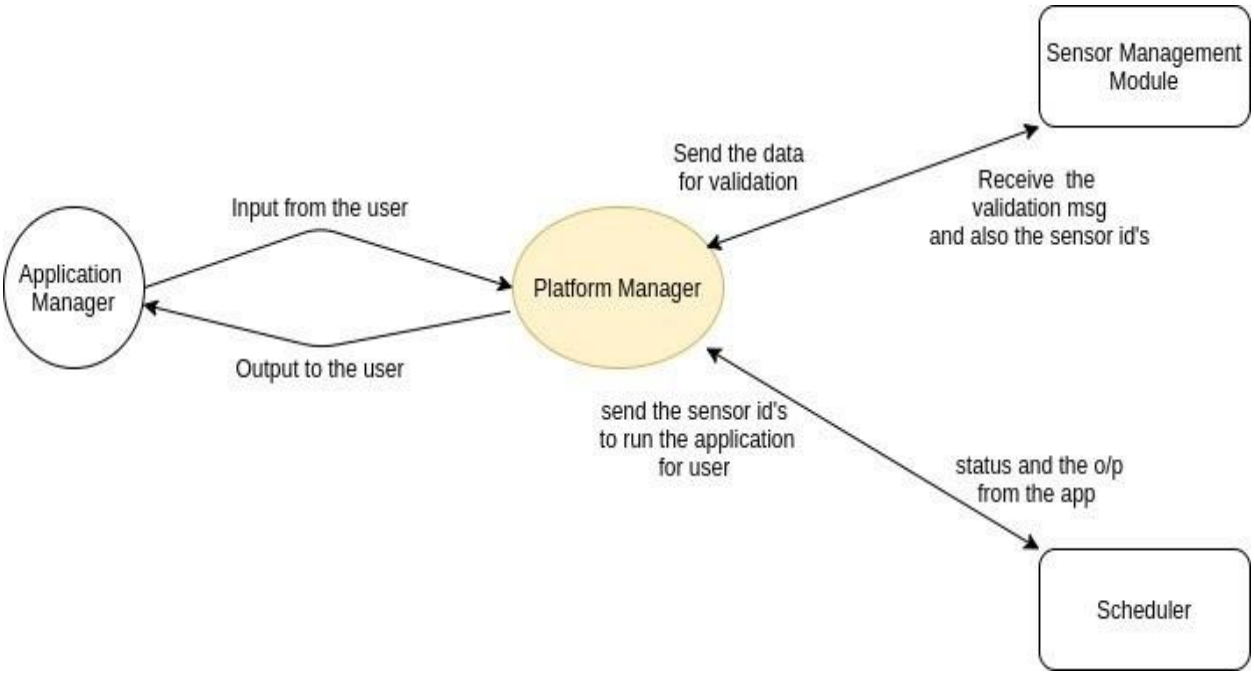


Figure: platform manager

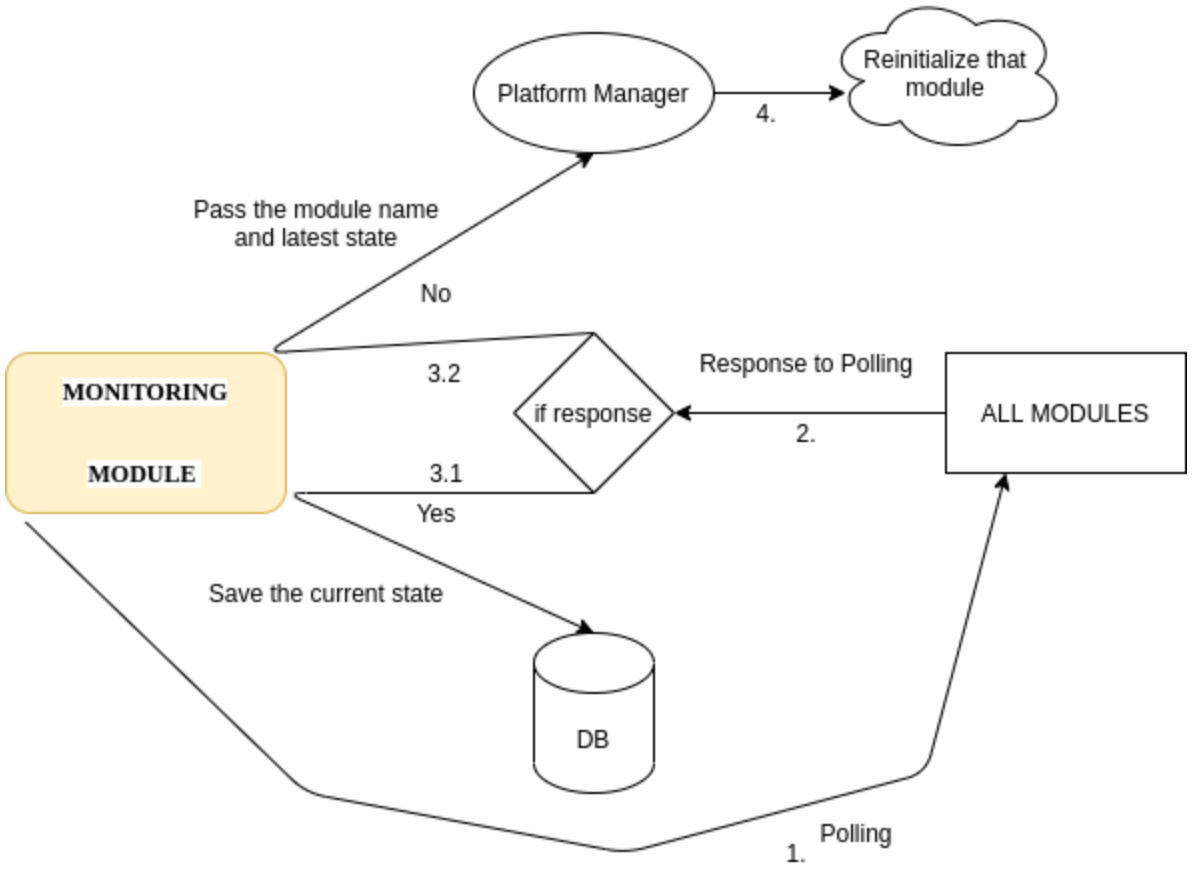


Figure: monitoring module

iii) List the sub modules

The sub-modules of Application Manager are:

- login module
- application user module
- application deployer modules
- admin module
- file validation module.

The sub-modules of Platform Manager are:

- initializer module
- re-initializer module
- Communication module

The sub-modules of Monitoring Module are:

- Connection with various modules
- Re-initialization of module

- Connection with different system on which module is present

iv) Brief overview of each sub module

Application Manager:

- The login module of Application Manger will let Existing users can login using their respective username and password and the module will authenticate it. New users can sign up using a username and password.
- Platform Admin of Application Manager provides provision for admin login and gives him/her admin privileges such as add/modify/delete sensors, notifications and controllers. Send these instructions to the Sensor Manager Module.
- The Application User module of Application Manager lets the user who wants to run the application that is deployed on the platform and provides input to the application.
- The Application Developer module of Application Manager lets the application user provide the application and the config file to the platform which will be sent to the Platform Manager Module for further application deployment.
- The File validation module of Application Manger parses the config file received from the user, extracts the requirements and passes it to the respective module.

Platform Manager:

- The initializer module of Platform Manager initializes modules, repositories and registries & creates running instances of all parts of the platform.
- The re-initializer module of Platform Manager will reinitialize the platforms if the system goes down due to some reason.

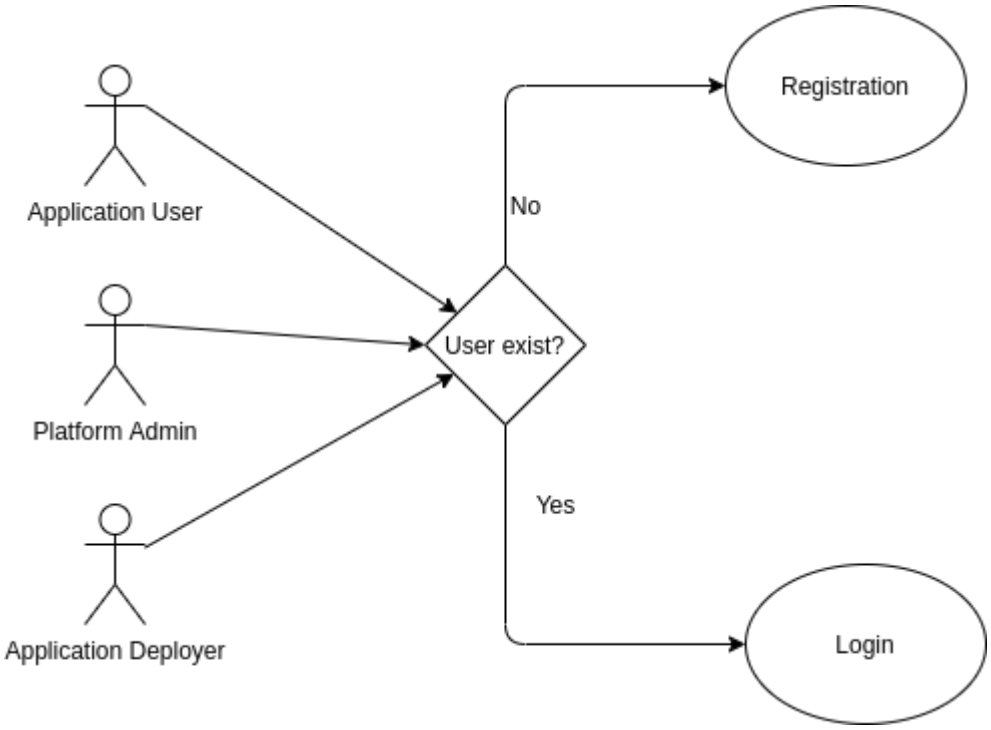
Monitoring Module

- 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.

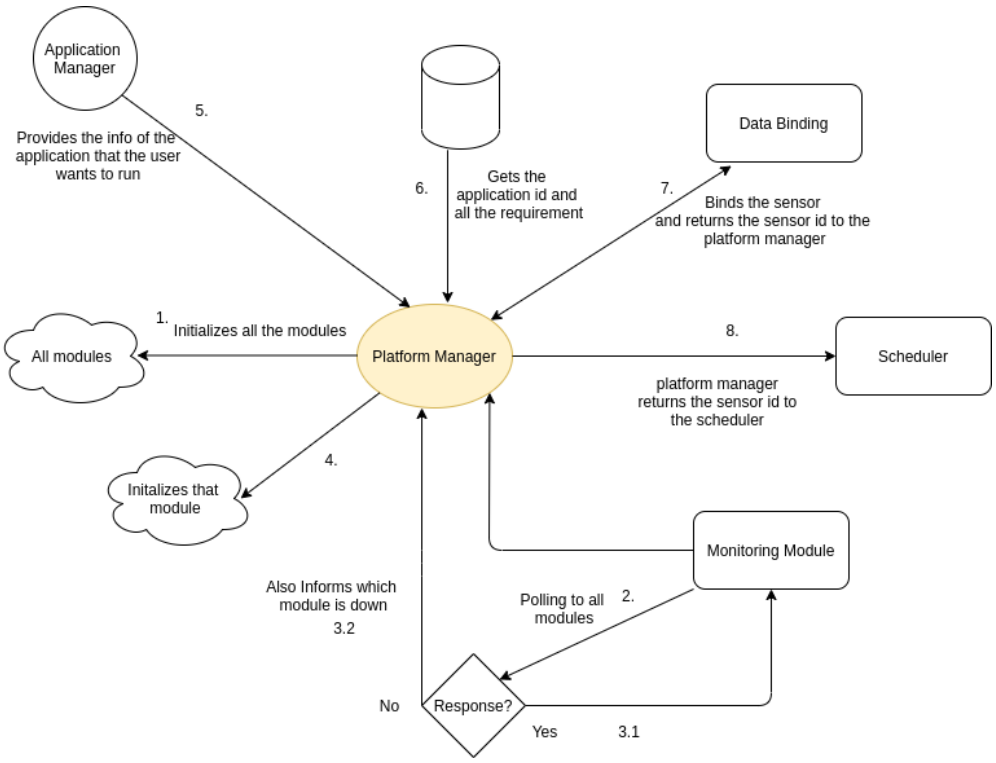
Bootstrap Module

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

Interactions between sub modules



The above diagram shows the different types of users involved in the Application Manager module. The login module of Application Manger will let Existing users can login using their respective username and password and the module will authenticate it. New users can sign up using a username and password. Platform Admin of Application Manger provides provision for admin login and gives him/her admin privileges such as add/modify/delete sensors, notifications and controller. Send these instructions to the Sensor Manager Module. The Application User module of Application Manger lets the user who wants to run the application that is deployed on the platform and provides input to the application.



Interactions between other modules

Application Module and Platform Initializer: The platform initializer reads the configuration file received by the Application Manager, the platform is started and accordingly every module is initialized.

Application Manager and all other modules: The parameters given by the user will be validated and then passed to the other modules as per their needs (only if required).

Platform Manager and all other modules: Platform Manager reads the configuration file & initialize modules, repository and registries. If the system goes down due to some reason, then we can use the platform initializer to reinitialize the platform.

Platform Manager and Scheduler: The requests received by the Platform Manager are sent to Scheduler, which schedules the calls/requests.

Monitoring Module and all other modules: The monitoring modules will keep polling to check the current status of all the modules. The modules send their last consistent state as response to the polling message. Upon successful receival, these states are saved into the database. In case of no response, the respective module is reinitialized with its last consistent state.

6.Node Manager, Load Balancer and Fault Tolerance Module

i.) Lifecycle of the module

How will the module/its components be started, monitored and stopped (if applicable)?

This module will be started at the start of the platform, initialized by platform initializer and will remain up till the end of platform lifecycle. The modules will be managed by Monitoring module via continuous polling.

How will any user's application components need be made available/deployed/setup on this Module?

User applications will run on a server instance provided by Node manager and will be continuously monitored by Fault Tolerance module for smooth running of the user's application.

ii.) List the sub modules

- Heartbeat Manager
- Node Manager
- Load Balancer
- Fault Tolerance

iii.) Brief overview of each sub module

- Node Management will handle the creation of all server instances.
- For every new request for a node(server), an appropriate instance of a node(server) will be returned by the node manager
- Load Balancer component will assign to the appropriate server node using some load balancing algorithms so that load is evenly distributed
- Load balancer will ensure load on server within threshold limit
- Fault tolerance will monitor the health of each server instances.
- Fault tolerance will also handle the case of node failures.
- The main server may be subdivided into several smaller instances, with different applications getting connected to different instances.

iv.) Interactions between other modules

- Deployer communicates with load balancer using Kafka for assigning appropriate server node.
- During execution server nodes interacts with communication manager as per the algorithm.

- Deployer then communicates with notification and action manager and forwards appropriate actions.