INTERNALS
OF
APPLICATION SERVER

GROUP – 3
Requirements 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)

1) Overview

1.1) 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.

All the features on this platform are implemented with portable microservices, and this platform is based on flexible microservice architecture. This means that **you can separately customize each feature, add new ones, or replace any existing ones with some third-party tools**.

Out of the box, this platform provides all the IoT features you may need for a typical IoT application - from data collection and device management to IoT dashboards and analytics. You can also take advantage of open APIs to integrate this platform's features to your own modules and applications.

1.2) Scope:

Generic platform: Supports all kinds of IOT applications.

2) Intended Use:

2.1) Intended Use:

To enable deploying, running, scheduling fault tolerant IOT applications.

2.2) Assumptions and dependencies:

2.2.1) Assumptions:

App developer (user of the platform) needs to provide a config file as a blueprint to its application as input to the platform.

2.2.2) Dependencies:

1. External services like docker, REST APIs.
2. Kafka, for communication between different modules
3. MongoDB.

3) System Features and Requirements

3.1) Platform Requirements

a) Deployment of the platform

- 64-bit OS (Linux/Windows/MacOS)
- Minimum RAM requirement: 4GB
- Processor: intel Pentium i5 8th generation(minimum)
- Python framework is used to develop a platform
- NFS: Network file sharing.
- Bash shell scripts for automation.

b) Different actors on the platform

- **Platform developer:** The one developing the platform
- **Application developer:** Who will be building end user application.
- **Platform admin:** Who will have having managing/ controlling access to the platform.
- **Platform user:** The end user who will be using the deployed application.

c) Applications overview of the platform

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.

3.2) Functional Requirements

a) Registering sensors

Initial setup of sensors, this module is responsible for registration of new sensor/ in the platform and binding them with the platform gateway.

b) Interaction with IoT sensors

- The data produced by sensors will be stored in kafka topics and whenever any application instance requests for data from a sensor, the sensor management will act as an intermediate and will collect data from that sensor's topic and provide it to the application who requested for that data.

c) Development of application on the platform

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.

d) Identification of sensors for data binding

This module will be called by platform manager to validate the requested sensor details and will do the binding of sensor to the appropriate application instance.

e) Scheduling on the platform

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.

f) Starting and Stopping services

To implement this module, the Platform Initializer will be responsible for starting (activating / initializing) all the other required modules, repositories and files of the system.

Also, services which are not required (in near future) will be stopped.

g) Communication module

This module will be responsible for handling the communication between all the modules of the system.

The key major components of this module are:

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

h) Server and service life cycle

1. Service Lifecycle Manager

- Service lifecycle manager will receive details of service in json format from the scheduler. It will then communicate with the server LC manager to get the IP/Port of server on which it will run it. The topology manager will give instructions on start/stop for any service-to-service LC manager.
- After that it will give details to the deployment module to deploy the service on the runtime server.
- Functionalities of this module are:
  → Run a Service instance
  → Kill all Service instances running on single machine input (IP, username)
  → Kill a single Service instance input (IP address, hostname, serviced)

2. Server Lifecycle Manager

- It will check which node is having less load which is updated in the registry by monitoring service and After applying the load balancing algorithm, it will give the response back to the service lifecycle manager. The response will contain the assigned Server Ip and Server port.
- State of each node is saved, so if any node fails then according to the last saved state it is resumed.

j) Deployment of application on 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.

k) Registry & repository

Registry: It will be used to store run-time information for applications and platform modules. MongoDB will be used for this.

Repository: There will be 3 types of repositories and they are as follows:

i) **Application repository:** It will store all the static files of applications deployed by users with the help of NFS and this will be used to store all application code.
ii) **Platform repository:** It will store all the static files of all the modules of our application server platform with the help of NFS.

iii) **Sensor Repository:** It will store the sensor meta details like sensor type, sensor output format, sensor location, sensor data generation rate etc.

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

l) Load Balancing

Load balancer will request nodes from the Node manager. Node Manager allocates new nodes to Load balancer for execution of an algorithm and sends a list of active nodes to load balancer. The following load balancing algorithms can be used by Load balancer:

- Round robin
- Less work
- Hash
- Power of 2 choices

m) Interactions between modules

Various modules will interact with each other using API calls. Using API calls various methods will be called to do appropriate functions.

n) Packaging details

To make the platform portable we will do packaging of platform which will consist of a script file which needs to be run on the new machine before deployment of platform to resolve all the dependencies required by platform.

o) Configuration file details

Configuration files (or config files) are files used to configure the parameters and initial settings. This will provide the config data which will be used by platform initializer to initialize all the modules. The platform initializer reads the config file and discovers the dependencies and libraries required for the application.

p) Interaction of different actors with the platform

This module will provide details of how different actors will interact with the system. The user will have to provide certain information to connect to the platform:

1. The **user** will use the application and all the platform services.
2. The **application developer** will
    - provide the application and the configuration file which will contain the information of the sensors etc.
    - Be able to check the health and the status of the application at any time.
    - A single application developer can deploy multiple applications on the platform.
3. The **platform admin** will

- Authenticate the user.
- Keep a status check on all the current running applications on the platform.

4. The **platform developer** will
    - Monitor the communication of all the modules.
    - Start and stop the modules, registries and repositories.

q) Sensor Manager

The sensor management module is responsible of dynamically handling sensors that is binding them with the application, allowing new sensor to be installed in the platform.

3.3) Non-Functional Requirements

Nonfunctional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across different backlogs. Also known as system qualities, nonfunctional requirements are just as critical as functional Epics, Capabilities, Features, and Stories. They ensure the usability and effectiveness of the entire system.

a) Fault tolerance

The Monitoring and Fault Tolerance Module is responsible for continuous monitoring status of all the modules via the scheduler, deployment manager, sensor manager, application manager, action manager and their corresponding nodes on which they are running.

If the monitoring system detects abnormal behavior of any module, it finds out whether the nodes on which module is running is not working or the instance of module itself is not working.

For the first case it will request fault tolerant manager to run same module on different

node and reflect changes if needed everywhere and for the second case it will request the

fault tolerant manager to run the instance of the module on the same node.

a. **Platform:** Identify the faulty module and report to the platform admin for further action.
b. **Application:** we are saving the state of each service so that in case of a crash we can restart the process and can resume the service from the last updated state.

b) Scalability

Scalability is an attribute of a tool or a system to increase its capacity and functionalities based on its users' demand.

a. **Platform:** Platform components like MongoDB, server instances are scalable as per requirements.
b. **Application:** Scalability of application like adding more servers during run time cannot be handled.

c) Accessibility of data

The application can call this method by using the API call and providing the input in the form of sensor name and number of sensor data that is required.

- **Application:** The application can call methods by providing the input in the form of sensor name and number of output data that is required.
- **Sensors:** Kafka will produce sensor queues, where sensor data will be stored until some application demands it.

d) UI and CLI for interaction:

User interface and/or Command Line interface provides app developer to interact with the platform and use its services.

CLI (Command Line Interface) or UI (User Interface) which will be used in the application will provide a platform/medium where user will prompt by writing a command and get the response from system, for this, users have to be compelled to kind command or train of command for performing the task.

This interface will provide the following services:

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. Also check whether the user has entered the valid application format.
5. If the format is valid then pass the application on to the scheduler.

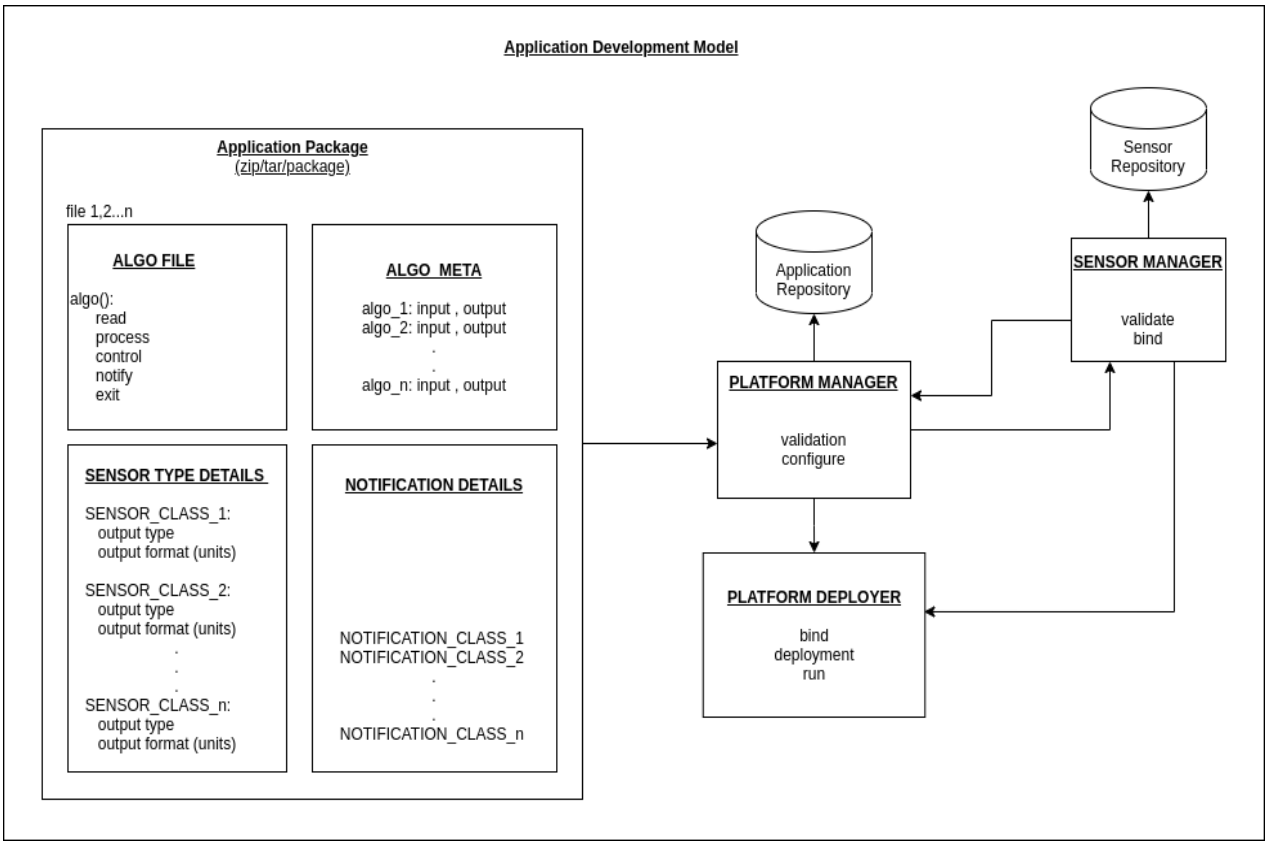
e) Security -Authentication and Authorization:

User authentication and authorization will be managed by the application manager module of the platform. When the user is successfully authenticated, she/he must further enter the algorithm/application which the user wants to run.

The communication module will take care of security and end to end encrypted message passing between modules.

4) List the key functions

a) Application Development model and its key features



Application Development key features:

- **Application Package:** A Meta package consists of various configuration details of the application i.e., consists of various details such as sensor class (created on the basis of sensor types, number of fields and their type, output type), notification types, algo and algo metadata JSON files and various scripts files required for the development of the application.
- **Platform Manager:** The Platform Manager is responsible for the **validation** of the given package/xml/JSON files of an application. It will **configure** them with the repository to check the availability of the sensor/controller/notification instance for the application. It will do **binding** of the sensors to the application instances then it will forward the module to the platform deplorer for scheduling it.
- **Sensor Manager:** This module is responsible for handling the incoming request and validating it with sensor repository and binding them to the application instances.
- **Platform Deployer:** It will take the module from Platform manager and will be responsible for doing scheduling on it, creating its instances, deploying it, running it, and monitoring them.

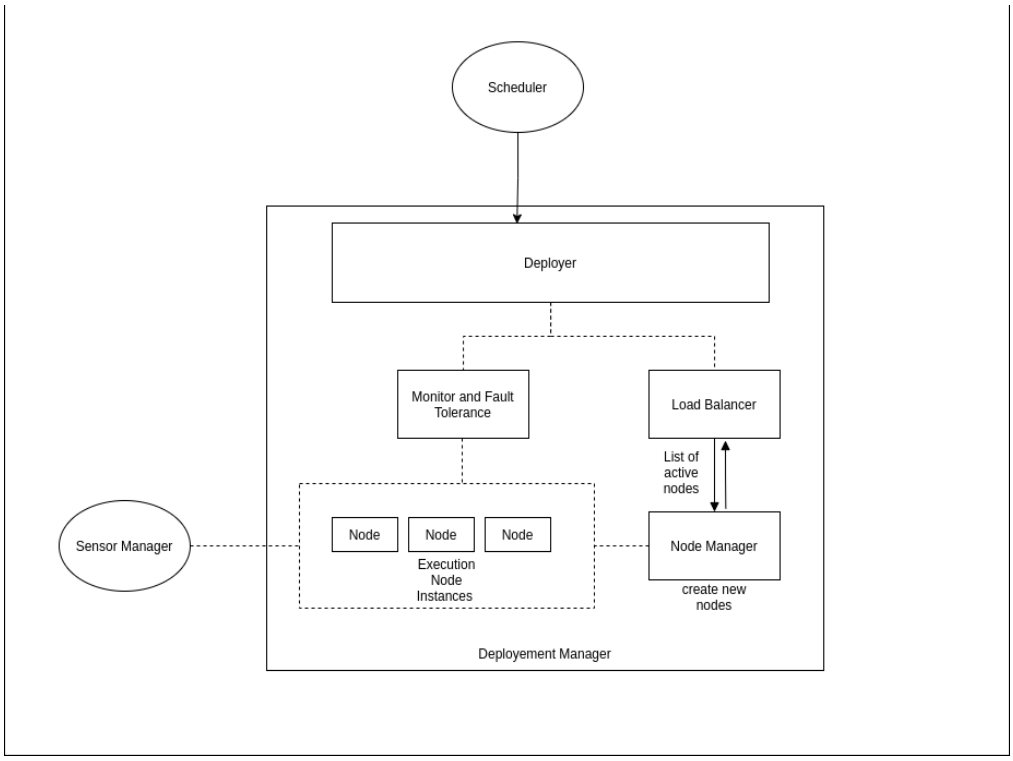
The key steps in the whole application development process:

The key steps in whole development process are:

- The application will give the package consisting of various configuration files/script files to the platform manager which will be the point of contact to the application.

- Platform manager will do validation of the config files like proper formatting should be there, all required fields should be present, and it will communicate with the sensor manager module for sensor validation part.
- On successful verification, platform manager will forward the application to the platform deployer who will do the binding part, create required number of instances, schedule the instances of the application, run the instances and will do monitoring on them.

b) A block diagram listing all major components (components)



c) Brief description of each component

- **SENSOR MANAGEMENT MODULE**

  The sensor management module is responsible for dynamically handling sensors that are binding them with the application, allowing new sensors to be installed on the platform.

  The various operations handled by communication modules are:

  1. Sensor Manager
  2. Sensor config validator
  3. Sensor Data management.
  4. Controller state changer.

- **DEPLOYMENT MANAGER**

  The Deployment Manager is responsible for the deployment of an algorithm on a machine. It will bind the sensors to the algorithm instance and be responsible for deploying algorithm instances stored in repository to run. It also takes care of a system where the algorithm needs to be run.

- **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.

- **PLATFORM INITIALISER**

  It reads the configuration file when the platform is started and accordingly initializes every module, repository and registry of the platform. If the system goes down, then the platform can be initialized at that very instant, so it provides fault tolerance.

- **DATA BINDING**

  This module will communicate with the sensor by sending the sensor id to the sensor manager and will receive the data that will be generated by the particular sensor or multiple sensors. Now, once the data is received, the module will be converting the data into the specific format which is required by the application. This processed data will be further used by the application.

d) List the 4 major parts (each team will take one part)

Team 1 - Communication module, sensor management and notification service.

Team 2 - Instance Creation, Scheduling and Deployment manager

Team 3 - Application manager, Platform Manager, Data Binding, Monitoring Module.

Team 4 - Node Management, Load Balancing, Fault Tolerance

5) Use cases

a) List what the users can do with the solution

- Real-time monitoring functions, configurable alerts and notifications, pluggable cloud services, and integration with consumers' smartphones and other devices.

- Predictive maintenance of equipment, collecting sensor data for real-time production analytics and ensuring safety, and end-to-end cargo delivery tracking in industrial, agriculture, and transportation sectors.

b) At least 5 usage scenarios:

- **Smart Classroom:** Our platform can be used to build smart classroom where staff can monitor class remotely and can take remote attendance, control temperature/ lighting remotely.
- **Smart Parking:** Our platform can be used to build a smart parking system where parking staff can monitor parking area remotely and can take note of vacant parking spots and can guide car drivers to those spots and can control the lighting remotely.
- **Connected Ports:** Our platform is responsible for predicting the best time to move and depart cargo ships at ports to reduce waiting times and costs. Calm water and weather conditions allow for lower fuel consumption rates, facilitate cost-effective per-ship payloads, and help ensure the safe arrival of cargo.
- **Smart Toll Plaza:** Our platform can be used for building a smart Toll Plaza that will be responsible for collecting toll tax from vehicles using electromagnetic fields to automatically identify the taxes attached to vehicles, Smart toll plaza will also be monitoring vehicles and driver and if found breaking any rule then suitable fine will be imposed on vehicle owner.
- **Smart Health Application:** Our platform can be used for building a smart health application where Wearable sensors can be used to gather physiological and movement data thus enabling patient's status monitoring. Sensors to monitor vital signs (e.g., heart rate and respiratory rate) would be deployed, for instance, when monitoring patients with congestive heart failure or patients with chronic obstructive pulmonary disease undergoing clinical intervention.

6) Primary test case for the project (that you will use to test)

a) Name of use case

Transport facility using buses for covid vaccination drive

b) Domain/company/environment where this use case occurs

The covid vaccination drive is ongoing in the entire country and this transport facility will help people from various places to use this transportation in order to go to the vaccination centers easily and return to their homes after getting vaccinated in a safe and secure way.

c) Description of the use case (purpose, interactions and what will the users benefit)

It will be used to build a **smart platform** that will be responsible for providing transport facilities using buses which can do the following things:

- Whenever someone boards a bus, he/she will do biometric check-in.

- Fare can be calculated based on distance multiplied by a fixed rate & will be displayed on dashboard for that bus application instance.
- When the bus is not empty, if temperature is more than a threshold switch on the air conditions or lighting is lower than a lux level switch on the lights.
- if more than two (>=3) buses come in a circle of given radius, except one send buzzer command to the rest.
- whenever a bus comes closer to a barricade by a threshold distance start/trigger an algorithm which sends an email to administration mentioning unique identifier of the bus& an email body notifying them.

d) What is the "internet: angel around these things

Various sensors in this application help track the buses used for transportation and redirect them to the places where they are needed the most.

The fare of the transportation is calculated based on the distance covered by the bus from start point to the end point for that person and is displayed via a message to the passenger.

When the bus is not empty, if temperature is more than a threshold switch on the air conditions or lighting is lower than a lux level switch on the lights.

e) Information model

- **GPS:** Sends placeholder-id & co-ordinates, each bus is installed with one such sensor, one sensor is at IIIT-H campus and each police barricade has one such sensor.
- **Biometric:** Sends placeholder-id & person-id (it will be unique on place-id level).
- **Temperature:** Sends current temperature of the place.
- **Light Sensors:** Sends Lux Level of the place.

And all the information is maintained using databases on cloud.

f) How is location and sensory information used?

The location of the buses is used to calculate fares for the passengers and other sensors such as temperature and light sensors are used to turn AC on or off.

g) Processing logic on the backend

To calculate the fare for a person, the starting location and end location for that person is used and based on that distance, a fixed rate is multiplied by that, and that fare is sent as an SMS to the passenger.

When the bus is not empty, if temperature is more than a threshold switch on the air conditions or lighting is lower than a lux level switch on the lights. This is achieved by maintaining temperature sensors and light sensors.

h) User's UI view- what is their UI, where and all will they do with this system

The users can view their fare for the trip on their mobile phones as an SMS will be sent for that. Whenever a bus crosses the barricades, an email will be sent to the college authorities. Whenever more than three buses come inside a circle of given radius, buzzer command is sent off.

7) Subsystems

A) Key subsystems

I. Communication module, Sensor Management and notification services.
II. Scheduling of application, Fault-Tolerance.
III. Application manager, Platform Manager, Data Binding, Monitoring Module.
IV. Node Management, Load Balancing, Fault Tolerance

B) A block diagram of all subsystems

Team 1: Communication module, Sensor/Controller Management and notification services module:

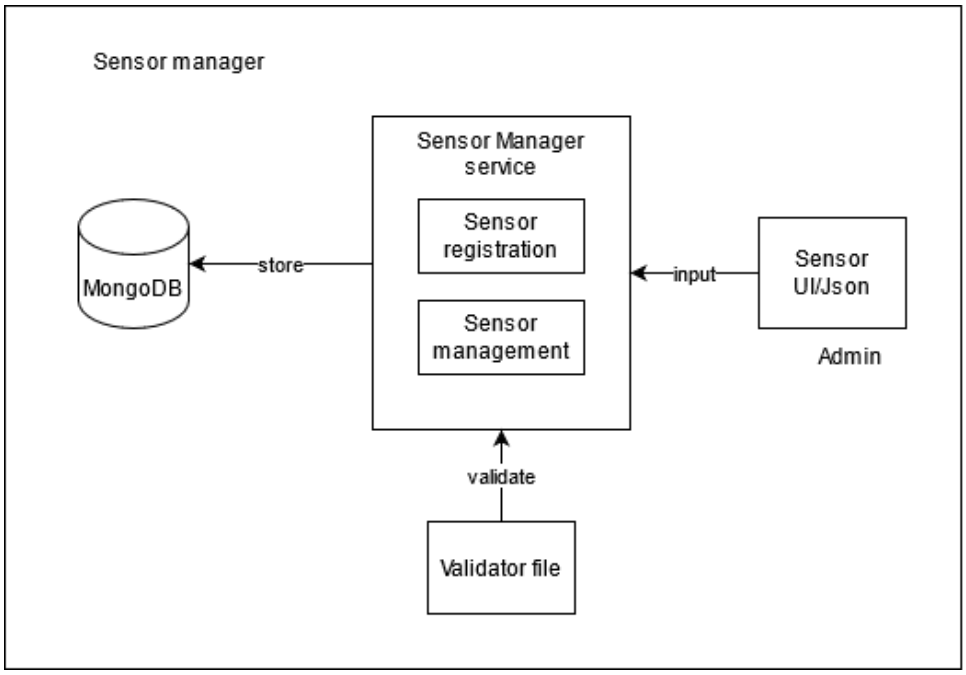


Fig: Sensor Management Service

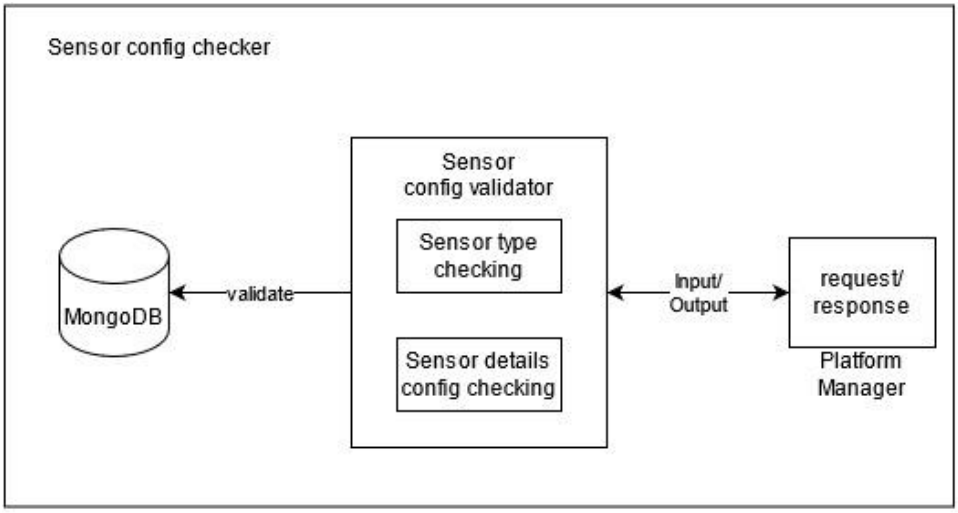


Fig: Sensor config checker

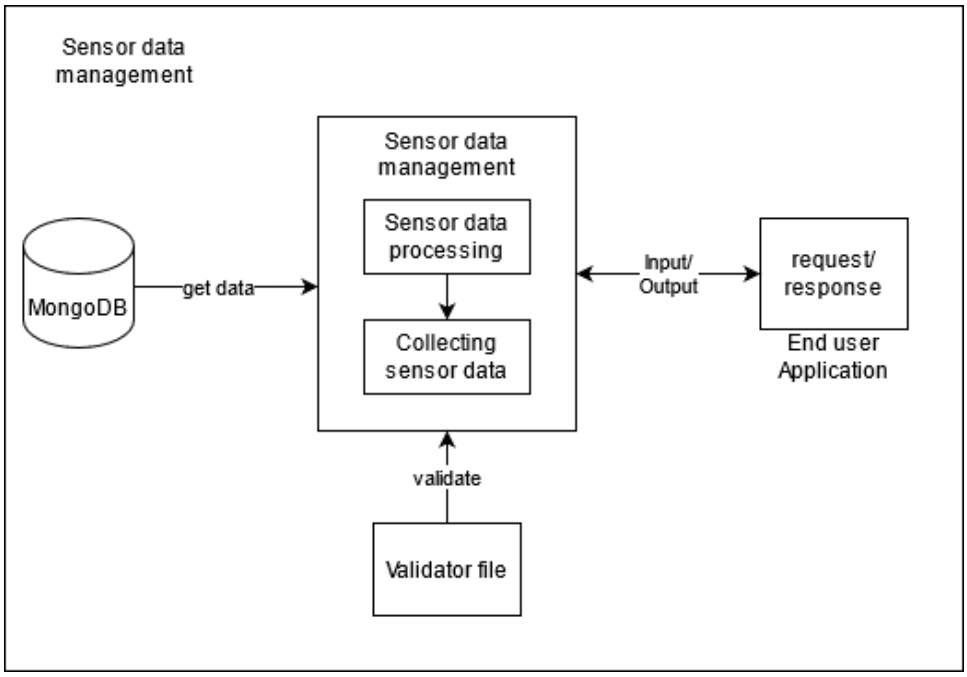


Fig: Sensor Data Management Service

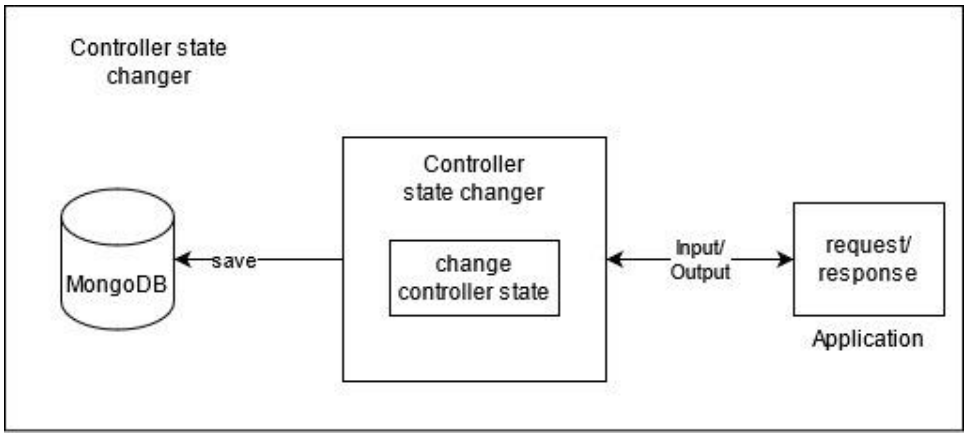


Fig: Controller Management Service

C) Interactions involved across these subsystems

- **Application Manager and Scheduler:**

  Application manager will provide the validated configuration file of the application to be scheduled to application with the start time, end time.

- **Application Manager and Sensor Manager:**

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

- **Scheduler and Deployer:**

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

- **Deployer and Node manager:**

  Deployer will ask Node manager for instance of server on which application can be deployed.

- **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.

- **Sensor Manager and Node:**

All the data requested by any application for any sensor will go to the sensor manager through node and sensor manager will return the requested data.

D) Registry & Repository

- **Registry**
  1. It holds dynamic content.
  2. It will save the state and progress of each module running in our platform with file locations.
  3. It collects data from each running module so when required to restore the state of the module.
  4. For restoring, machine stats are given to the topology manager.
- **Repository**
  1. Module needs to save dependent data, so it gets saved in its repository.
  2. Mainly split among users and services corresponding through that module.
  3. It collects data from the corresponding module and saves the config file of that module.
  4. When required, it sends data requested by module from its repo, it can be for authentication, type checking or formatting.
  5. Example: App repo, Algo repo, etc.

8) Brief overview of each of the four parts

a) What are the four parts (each team will work on one part)

- Communication module, Sensor Management and notification services.
- Scheduling of application, Fault-Tolerance
- Application manager, Platform Initialization, Data Binding, Monitoring
- Node Management, Load Balancing, Fault-Tolerance

I. Communication module, Sensor Management and notification services.

1.1 Communication module:

This module will be responsible for handling communication between all the modules of the system.

The key major components of this module are:

1. Interaction with modules
2. Interaction with sensors
3. Interaction of different actors with the platform.

1.2 Sensor Management:

The communication module is responsible for dynamically handling sensors that is binding them with the application, allowing new sensors to be installed on the platform.

The various operations handled by communication modules are:

- **Sensor Manager:**

  This module is responsible for handling the incoming request and binding them with the corresponding function call. It will allow various functionalities like registration of new sensors, managing them, etc.

  Various sub tasks of this module are:

  - **Sensor registration**
  - **Sensor Management**

- **Sensor config validator:**

  This module will be called by the platform manager to check the configuration details of the sensors requested by application to be deployed on our platform.

  This module will analyse the class of sensors required by application (class of sensors are dependent on its various fields) and in return respond to whether that class is present or not.

- **Sensor data management:**

  This module is responsible for providing the end user functionalities of the sensor to the end user application.

  Various sub tasks of this module are:

  - **Sensor data processing**
  - **Collecting sensor data**

- **Controller state changer:**

  This module is responsible for changing the state of the controller based on the output generated by the application by processing the sensors data.

1.3 Notification Service:

This module is responsible for-

1. Validating the notification service type requested by the deploying application. It will be called by the platform manager.
2. Notifying the user, application developer upon certain events through the requested notification services.

1. List of sub-systems in this part

a. Sensor Manager-

This module is responsible for handling the incoming request and binding them with the corresponding function call. It will allow various functionalities like registration of new sensors, managing them, etc.

The various services provided be this module are:

- **Sensor registration:** This module is responsible for the registration of new sensors on the platform and binding them with the platform gateway.
- **Sensor management:** This service can only be called after the registration of sensors. The functionality for analysing and modifying the data generation rate of the sensor is provided.

b. Sensor config validator:

This module will be called by the platform manager to check the configuration details of the sensors requested by application to be deployed on our platform.

This module will analyse the class of sensors required by application (class of sensors are dependent on its various fields) and in return respond to whether that class is present or not.

c. Sensor data management:

This module is responsible for providing the end user functionalities of the sensor to the end user application.

Various sub tasks of this module are:

- **Sensor data processing:** This module will be called by the end application for doing various processing on sensor data. Like conversion of sensor data from one format to another format (as per demand of the application).
- **Collecting sensor data:** The application can call this method by providing the input in the form of sensor name and number of output data that is required.

d. Controller state changer:

This module is responsible for changing the state of the controller based on the output generated by the application by processing the sensors data.

1. List of services/capabilities in the part

1. Sensor registration
2. Sensor management
3. Sensor data processing
4. Collecting sensor data

2. Interactions between this and other parts. Nature/purpose of interactions, likely interchange info/services.

In Admin UI part, he will be having 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. After the application is deployed on the platform, a request can be made to sensor data processing part for defining the way application requires the data and then a request can be made to sensors' data.

II. Instance Creation, Scheduling and Deployment manager

Introduction

Distributed scheduling is a resource management component of a system which schedules jobs around various working modules of the system to balance load and maximize overall performance of the system.

Functional Overview

1. Service instance creation:

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.

2. **Scheduling**

   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.

3. **Calling deployment manager**

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

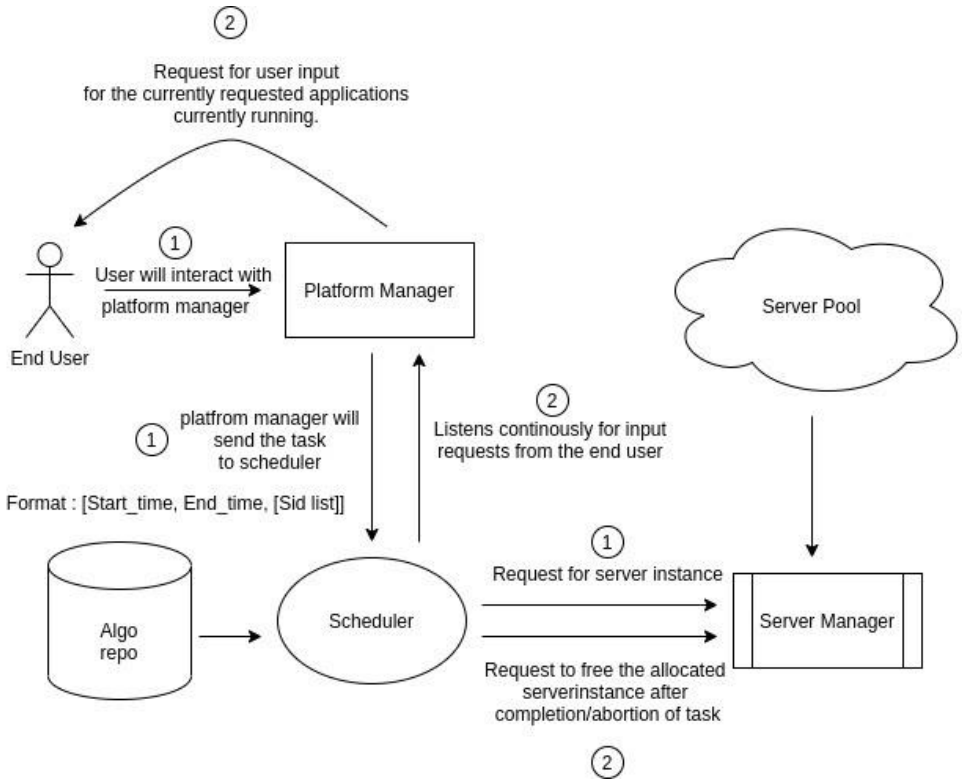
4. **Platform deployer**

   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.

5. **Fault tolerance**

   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.

Block Diagram



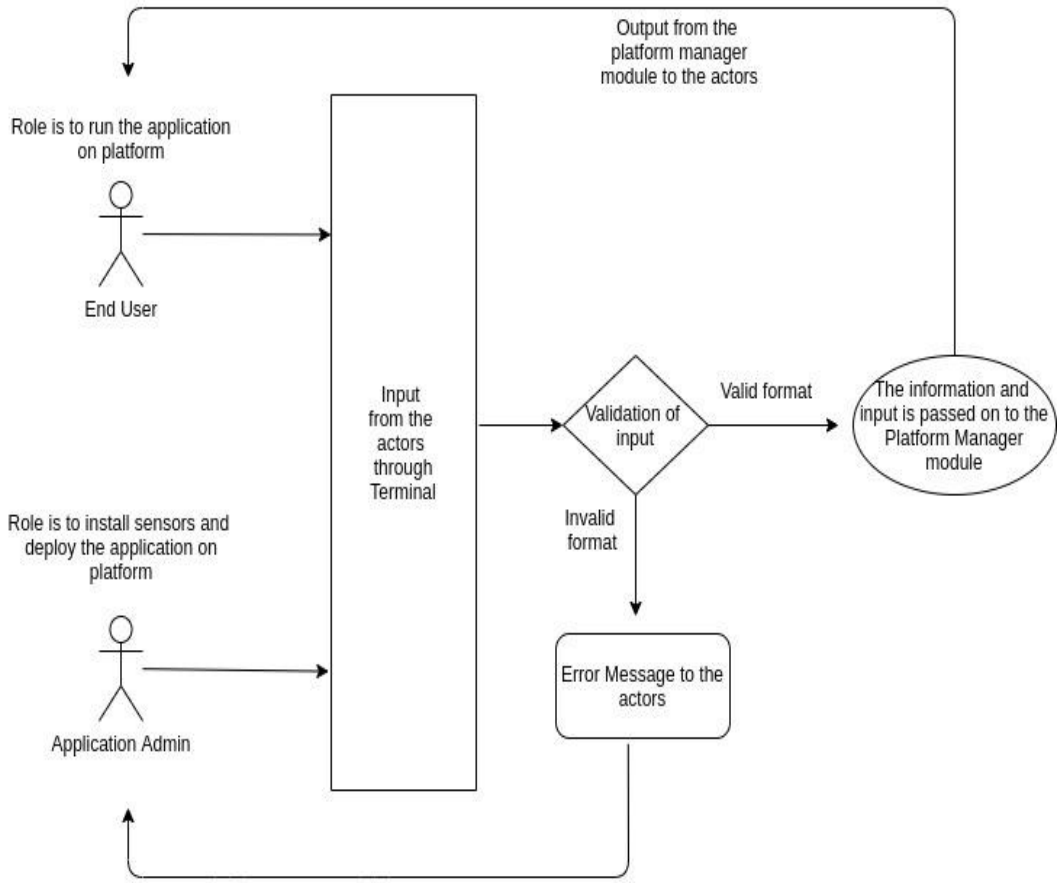
III. Manager, platform manager, data binding, monitoring module

3.1 Application Manager

A) Overview

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. The web page will also get the application file for the platform to be deployed and respective application requirements from the user. To parse the config file received from the user, extract the requirements and pass it to the respective module.

B) Block Diagram



C) List of sub-systems in this part

- User Authentication: 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 allow the user to enter the application file.
- Format Authentication: The data received must be checked for the specified format.
- Parsing and passing: 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.

D) List of services/capabilities in the part

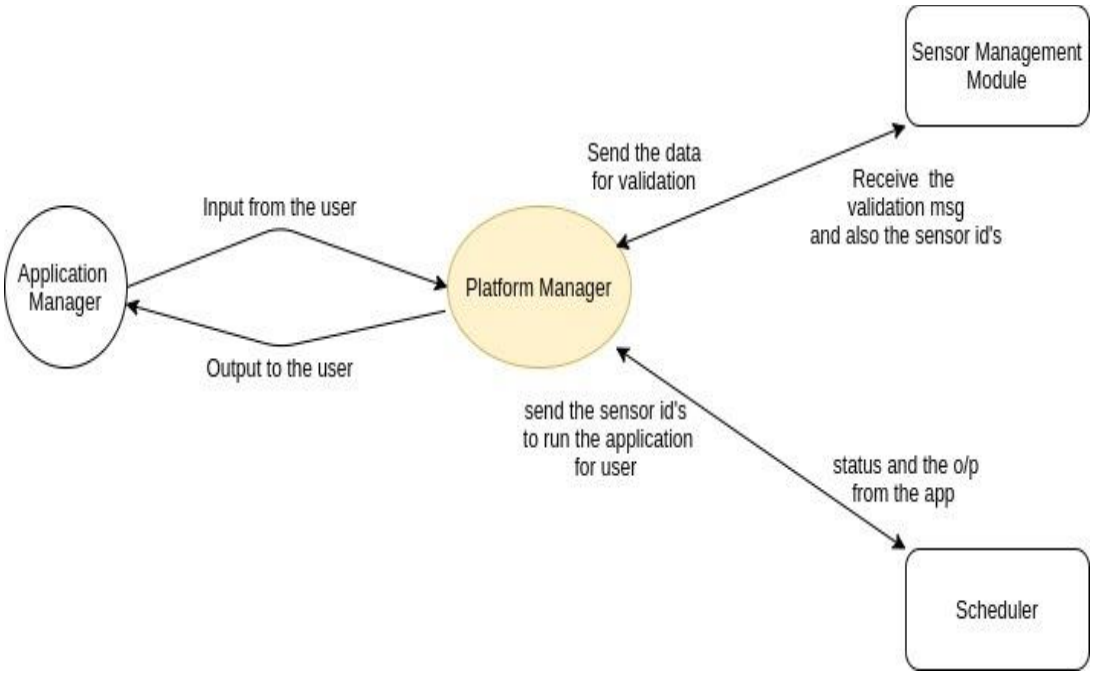
1. Register User
2. Log in User
3. Authenticate User
4. Check Details
5. Parsing and passing requirements.

3.2) Platform Manager

A) Overview

It reads the configuration file received by the Application Manager, the platform is started and accordingly 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. Make the platform portable by packaging the platform. The files received from the application are validated and then configured.

B) Block diagram



3.3 Bootstrap Module

A) Functional Overview

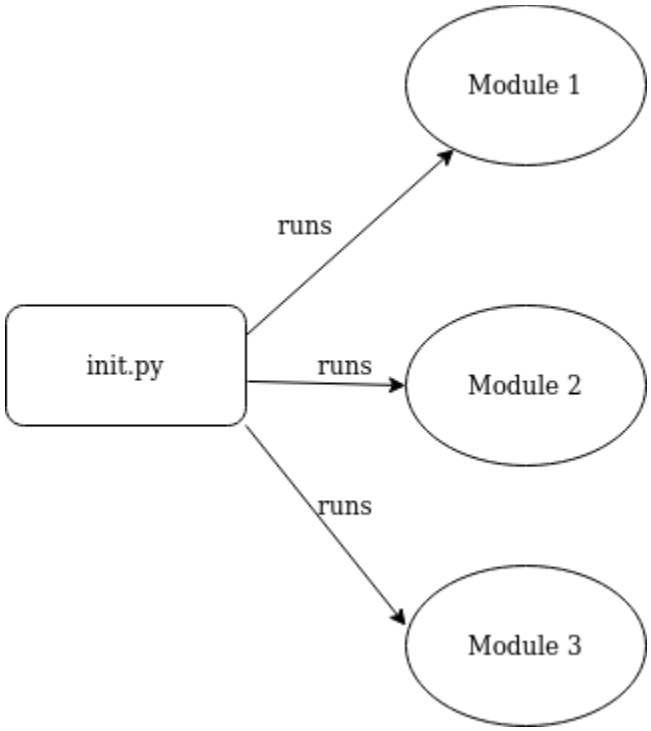
- The module will initialize and run all the modules present on different Virtual Machines.

B) Technologies used:

- Python (multithreading - socket programming)

C) Use Case:

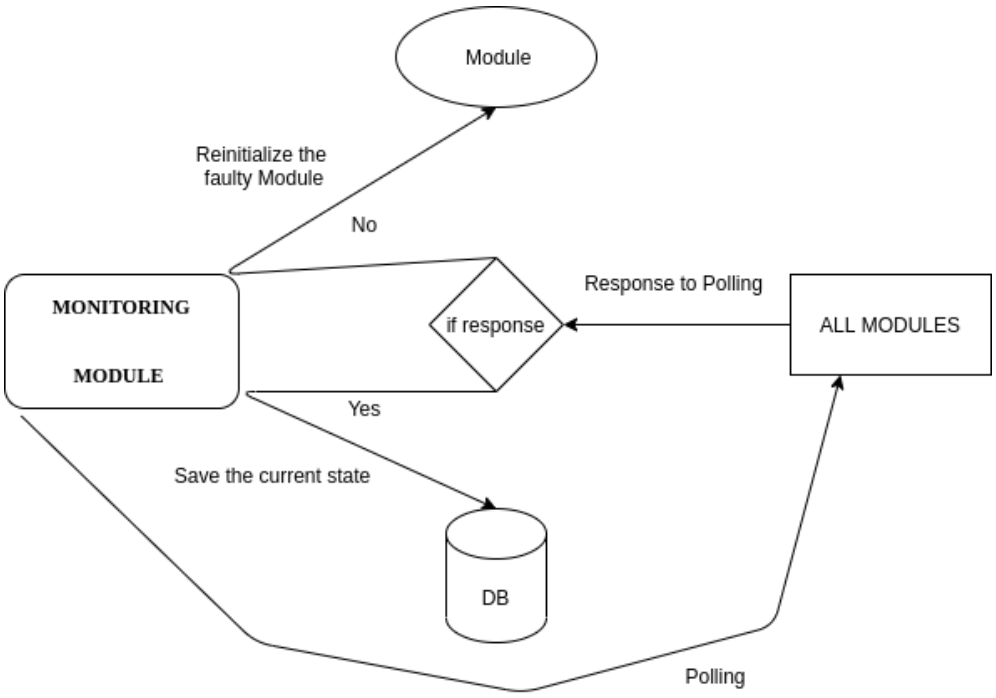
- Initializes all the modules.
- Opens a new terminal screen wherever necessary.



3.4 Monitoring Module

A) Functional Overview:

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.



C) List of services/capabilities in the part

- Polling
- Saving last consistent state
- Reinitialize in case of failure

IV. Node management, Load Balancing and Fault-Tolerance

a) Functional Overview of each part

- 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 the appropriate load balancing algorithm.
- 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.

b) List of sub-systems in this part

- Node Manager
- Load Balancer
- Fault tolerance

c) List of services/capabilities in the part

a. Load Balancing.
b. Health monitoring.
c. Node Management