IAS Project
Group 6

AI-Based Application Platform

Group Requirements Document
March 2022

Table of Contents

1.Overview

1.1. Introduction

- In this project, we are building an AI platform that will have all the necessary components required to host an AI application. It will incorporate all the required hardware, firmware, databases, frontend and backend services, and communication means for data transfer. It will also have monitoring and fault tolerance services for deployed applications.

1.2. Scope

- One can host any AI-based application on this platform.

2.Intended Use

2.1. Intended Use

- Data scientists will use this platform to deploy trained datasets as and when commissioned by us which can be used by app developers using the platform for their AI applications.
- App developers can use this platform to develop, build and host any AI-based applications. There will be sensors provided to applications that will enable the data transfer with the hosted application. App developers will just have to build an AI app that will make use of one or more of the already existing trained datasets and submit it to the platform.
- End users can request to deploy one of these developed applications and feed data via sensors and get predicted output directly without having to worry about any scripts involved.

2.2. Assumptions and dependencies

- Sensors and models required are available before application execution.
- The application Configuration file should follow the protocol as laid down by the platform.
- Scripts will be written in python language.

3.System Features and Requirements

3.1. Platform Requirements

3.1.1. Deployment of platform

- The Scheduler learns all the information related to the deployment of the application and then gives the deployer-related information.
- The Deployer is responsible to check the type of environment needed to run the application.
- If the environment needed is the standalone environment, the Application will be run on a new node.
- If the environment needed is shared, the Deployer will take the existing nodes list from the node manager and use Load Balancer to find the node with the least load.
- After an application is finished running, the node is returned to Node Manager.

3.1.2. Different actors on the platform

- **Platform Admin:** It is responsible for managing and handling the admin work of the platform, for e.g. adding a new sensor to the platform.
- **Application Developer:** He is responsible for giving the requirements for the application for the deployment of the application on the platform.
- **Data Scientist:** He is responsible for training different types of models and providing them to the platform.
- **End-User:** Responsible for the usage of applications and services related to an application.
- **Platform Developer:** Developing components of the platform and managing them.

3.1.3. Applications overview of the platform

- The platform supports applications that use AI models along with sensors available on the platform.
- Actors need to specify their part of the information in order to run an application.
- The sample application which could be designed for such a platform is described as follows:

- Surveillance in exam halls displaying names of students on screen giving the exam.
- Sensors Involved: Camera
- AI Model Involved: Face Detection
- **USE CASE**
  - During the exam, Examiner will be seeing the screen in which students giving exams face will be highlighted in a box and their name will be displayed above the box.

3.2. Functional Requirements

3.2.1. AI Model

- Development of AI Model
- Packaging of Model
- Upload and deployment of the model

3.2.2. Application

3.2.2.1. Registering sensors

- Whenever the platform is initialized and/or a new sensor is added to the platform, sensor registration has to be done to begin data transfer with the sensor.
- Details like sensor id, input and output type, location, data transfer frequency, etc. are taken as inputs while registering the sensor and stored into the platform database.

3.2.2.2. Interaction with sensors

- Each sensor will have a unique sensor instance id which can be used to identify and address that sensor uniquely.
- We will decide on some standard to establish communication between sensors and other IoT hardware/software.
- Input data will be read from the sensor and will be sent to the application instance to be processed.

3.2.2.3. Development of application on the platform

- Firstly, the application developer will create an application and attach necessary config files(App config, Sensor type

config, Model config), and send them to the application manager.
- The application manager will verify the sensors and model which will be used by communicating sensor and model manager.
- The sensor manager will check its database to see if a sensor of the same type as specified is already present or not. Similarly, for the model manager as well.
- On successful verification of the config files, a request for the deployment will be sent to the deployer and, an instance of the application will run on a node.
- Instances of sensor type defined in the sensor instance config file from the user and the models from the application zip will also be created and assigned to that application instance.
- The sensor will then read data when available and commute it to the application instance which will then make a prediction using the model instances provided if possible and return the result.

3.2.2.4. Identification of sensors for data binding

- When registering a sensor, each sensor is assigned a unique sensor instance id which can be used to uniquely address it whenever needed. Also, other details as provided in the sensor type/instance config file by the app developer will be stored too.
- When the sensor manager receives a data binding request, it will identify which sensor to bind based on the id passed and it will provide the application with the specified sensor.

3.2.2.5. Data Binding to the application

- Application instances will run on nodes provided by the node managers. Each node will communicate with the sensor manager by asking or sending data along with the sensor id of a particular sensor specified in the config file. Also, each node will be assigned instances of models specified in the config file of that application too.
- Based on the id passed, the sensor manager will assign a sensor instance of that sensor to the node which will relay the data being inputted.

3.2.3. Scheduling on the platform

- The scheduler will receive the applications' config file and read the respective attributes such as start time, end time, priority, etc and add the application to the priority queue which will be sent to the deployer.
- In the case of a normal job, the scheduler will simply add it to the priority queue for its later execution. Whereas in the case of the cron job, it will be added to the priority queue every time considering its end time.

3.2.4. Acceptance of scheduling configuration

- The scheduling configuration data is submitted along with other configuration files by the End-user to the application manager.
- The application manager passes the scheduler config file to the Scheduler which in turn validates the respective format and data.

3.2.5. Starting and Stopping Services

- After the validation of all the required configuration files, the scheduler reads the start time, end time, priority, etc., of the application and adds the request to deploy in the priority queue.
- The request from the priority queue goes to the deployer which then deploys the application being in contact with the node manager.
- Normal jobs are sent to the deployer only once. Cron jobs are added to the priority queue multiple times even after being sent to the deployer until certain conditions pass.
- When the end time, if specified, is reached or when the end-user exits the application, the scheduler triggers a signal to the deployer for stopping the required service.

3.2.6. Communication model

- Communications involved in our platform are:
    - Data is inputted into the sensor continuously which is then transmitted to the application instance as and when requested.
    - Continuous monitoring of nodes, services, and applications for fault tolerance.
    - Communication between the nodes and the node manager
    - Communication between the node manager and deployer

- Communication between deployer and application manager, sensor manager, and model manager.

3.2.7. Server and service life cycle

- The server life cycle is taken care of by the Node manager as it has the information of the free nodes and active nodes. It is the one that allocates new nodes which the deployer uses to deploy the application.

- Service lifecycle is being handled by the Deployer-Scheduler duo. When the application reaches its end time or when the user wants to quit the application, the scheduler signals the deployer to kill the instance of the application running at one of the active nodes.

3.2.8. Deployment of application on the platform

- The deployment of the application is done by the Deployer.
- Deployer checks the environment depending on which it deploys the app on a shared environment or standalone environment.
- In the case of a standalone environment, an instance of the new node is being used for the deployment, whereas in the case of a shared environment, the existing instance of the active node is used to deploy, whose selection is judged by the Load balancer.

3.2.9. Registry & repository

- This is the data store of our platform where the database of different components will exist. App data, sensor data, AI model, configuration files, etc will be stored.

3.2.10. Load Balancing

- In the case of a shared environment, the application will be deployed to the existing node.
- Load balancer fetches all the active nodes from the node manager and checks which node is having the least load among all.
- The node having the least node is selected and its information is passed to the deployer for the deployment.

3.2.11. Interactions between modules

- *Application Manager - Authentication Manager*: Users' details and their authentication and authorization are taken care of by the Authentication Manager and pass the corresponding data to the Application Manager for which the request is raised.

- *Application manager - Scheduler*: The application manager will send the scheduler the start time, end time, priority, etc., fetched from the end-user, based on which the application will be pushed into the priority queue for further deployment.

- *Scheduler - Deployer*: Scheduler will add the application deployment request into the priority queue, based on the output of the queue, the scheduler will pass application data to the deployer for deployment.

- *Application Manager - Deployer*: The deployer after deploying the application on one of the nodes, will send the deployment status along with the running application instance ID.

- *Deployer - Load balancer*: The deployer will ask for the active node to the load balancer, in return Load balancer will perform some calculations and return the node information that the deployer will use to deploy the application.

- *Load Balancer - Node Manager*: Load balancer fetches the list of all the active nodes from the node manager, and finds the node with the least load among all.

- *Platform Manager - Sensor Manager - Application Manager - AI Model Manager*: The existing model details and sensor details residing on the platform are passed to the Application manager for integration with the application. Platform Manager can add new sensors to our platform via Sensor Manager.

- *Analytic Manager, Fault-Tolerant Manager*: These subsystem monitors the different subsystems at regular intervals and check the working status of these. In case any system doesn't respond properly, that subsystem is reinitialized.

3.2.12. Packaging details

- Files associated with the platform are listed as follows:

- Application.zip -> Contains configuration files, models required, sensor types needed and a number of instances of each sensor required.

- Model.zip -> Contains a pickle file of the trained model and config file which will contain input JSON object format.

3.2.13. Configuration files details

- There will be the following config files:
  - Trained model config file: It will contain the input format and output format for that model in the form of a JSON object.

    model.config

```
{
   "model_name": "abc model",
   "model_author": "Bruce",
   "input": [
       {
           "field_name": "field1",
           "field_type": "int"
       },
       {
           "field_name": "field2",
           "field_type": "str"
       }
   ],
   "output": [
       {
           "field_name": "field1",
           "field_type": "int"
       },
       {
           "field_name": "field2",
           "field_type": "str"
       }
   ]
}
```

- Application/config file: It will contain the application name.

  app.config

```
{
   "app_name": "hello_world"
}
```

- Application/sensor type file: It will contain the type of sensor needed for that application with all necessary details for the sensor.

  Sensor type config

```
[
   {
       "sensor_name": "sensor_A",
       "sensor_type_id": "231",
       "sensor_manufacturer_name": "No_idea",
       "sensor_model_number": "007xyz"
   },
   {
       "sensor_name": "sensor_A",
       "sensor_type_id": "231",
       "sensor_manufacturer_name": "No_idea",
       "sensor_model_number": "007xyz"
   }

]
```

- Application/sensor instance file: It will contain the number of instances of each sensor needed for the application.

  Sensor instance config

```
[
   {
       "sensor_instance_id": "231",
       "sensor_ip": "localhost",
       "sensor_port": "007xyz",
       "sensor_location": "some string"
   },
   {
       "sensor_instance_id": "231",
       "sensor_ip": "localhost",
```

```
        "sensor_port": "007xyz",
        "sensor_location": "some string"
    }
]
```

- Application/model file: It will contain info about all the models needed for this application

  App-model config

```
[
    "model_name": "Model1",
    "model_id": 1234
]
```

- Application/scheduler file: It will contain info about when the application should be deployed automatically if not deployed manually already.

  Scheduler config

```
{
    "app_name": "Appname",
    "start_time": "03/12/2022 12:00",
    "end_time": null,
    "interval": "15 min"
}
```

- New Sensor Config: This file will contain the data for the new sensors introduced to the platform by the platform admin. It is used for sensor registration.

  New Sensor config

```
[
    {
        "sensor_name": "sensor_A",
        "sensor_type": "typeA",
        "is_existing": true,
        "sensor_type_id": "231",
        "sensor_manufacturer_name": "No_idea",
        "sensor_model_number": "007xyz",
        "output_format": [
```

```
            {
                "field_name": "field1",
                "field_type": "int"
            },
            {
                "field_name": "field2",
                "field_type": "str"
            }
        ],
        "sensor_instance": [
            {
                "sensor_ip": "localhost",
                "sensor_port": "007xyz",
                "sensor_location": "some string"
            }
        ]
    },
    {
        "sensor_name": "sensor_A",
        "sensor_type": "typeA",
        "is_existing": false,
        "sensor_type_id": null,
        "sensor_manufacturer_name": "No_idea",
        "sensor_model_number": "007xyz",
        "output_format": [
            {
                "field_name": "field1",
                "field_type": "int"
            },
            {
                "field_name": "field2",
                "field_type": "str"
            }
        ],
        "sensor_instance": [
            {
                "sensor_ip": "localhost",
                "sensor_port": "1234",
                "sensor_location": "some string"
            }
        ]
    }
```

```
]
```

3.2.14. Interaction of different actors with the platform

- *End-User*
  - Its responsibility is to use applications and all its related services supported by the platform.
  - User Authentication is needed first to access the platform.

- *Application Developer*
  - This actor is responsible for giving the source code and all the information related to the application in the configuration file.
  - It can also request the report to have knowledge of the status of the application deployed by him.

- *Platform Admin*
  - It is responsible for the registration of sensors and sending the report if a developer requests the status check for its deployed application.

- *Data Scientist*
  - It is responsible to build and train AI models and provide them to the model manager so that these can be stored in the AI database.

- *Platform Developer*
  - It is responsible for starting and stopping all major components of the platform.

3.3. Non-Functional Requirements

3.3.1. Fault tolerance

3.3.1.1. Platform

- To make sure that all of the platform's components are working properly, and to reinitialize those that aren't.

3.3.1.2. Application

- To determine whether the application instances are functioning appropriately. Re-initialize another app instance if they aren't working. If an active node goes down, all the

applications/modules executing on that node are re-initialized on another node.

3.3.2. Scalability

3.3.2.1. Platform:

- Any number of AI models and sensors can be added to the platform.

3.3.2.2. Application

- If an application instance consumes resources beyond a certain threshold, it will be transferred to a new node at runtime that is dedicated to this application instance.

3.3.3. Accessibility of data

3.3.3.1. Application

- In its configuration file, the application will list the sensors and the models, it will use. The application should be able to access the data coming in from specified sensors.

3.3.3.2. Sensors

- Sensors can receive data from the app based on certain events.

3.3.3.3. Models

- Models can receive data from the app, and output the resulted prediction to it

3.3.4. Specification about application

- **Performance:** Overall the hosted applications will run pretty fast as they are just instances running on separate nodes with pre-trained datasets.
- **Accuracy:** The accuracy of the AI applications will depend on the accuracy of the models trained by the data scientists.
- **Reliability:** As long as there is enough space on the platform, it will work reliably. Hosted applications may crash because of compilation errors or some other errors but it wouldn't concern the overall working of the platform and it will continue to function.
- **Security:** The platform is secure overall with no way to access the user details database by any of the users except for admins.

- **Usability:** The platform is really easy to use and doesn't need much prior knowledge because of the user-friendly and interactive UI.

3.3.5. UI and CLI for interaction

- **Data Scientist:** He should be able to use the UI and CLI to upload his model zip files as well as the configuration files to the platform.

- **Application Developer:** The application Developer can use the platform's UI to upload an application zip file containing application code, sensor information, model information, and configuration files.

- **Sensor Registration**: There should be UI/CLI to register sensor information with the platform.

3.3.6. Security - Authentication and Authorization

- **Authentication:** Users should only be able to connect to the platform after authenticating themselves with the platform. There should be a proper authentication mechanism incorporated on the platform.

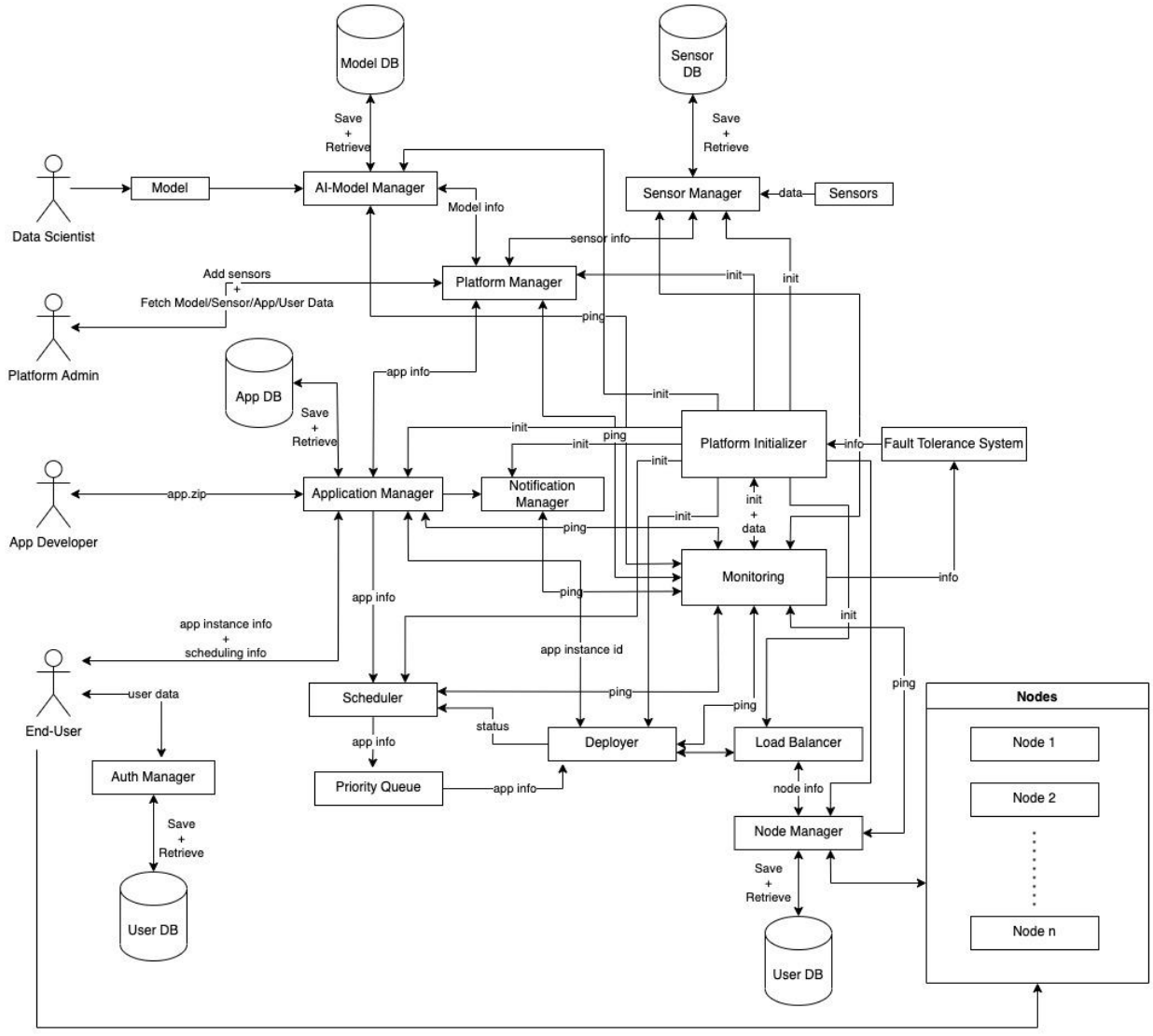
- **Authorization:** Depending on their roles, different stakeholders should have different access levels. The platform should be able to authorize users to use it according to their access level.

3.3.7. Persistence

- Data generated during application execution should remain in the database. Snapshots should be taken at regular intervals to ensure that data is preserved in the event of a failure. If a discrepancy or problem is found in the functioning of any of the modules, the state of that module can be recovered from the database, and the fault tolerance module will re-initialize it.

4. List the key functions

4.1. A block diagram listing all major components.



4.2. Brief description of each component

- **Platform initializer**
  - Platform initializer initializes other main components of the platform on the server which are necessary to run the platform.
  - It also builds config files to be used by every component.

- **Scheduler**
    - The scheduler is an important component that is responsible for scheduling instances of the application or application itself and stopping them in case of time-bounded instances.
    - The scheduler takes config files from the repository, as well as from the user.
    - Schedular validates the config files and data provided then sends the information to the deployer component at the specified time to deploy the desired application on a node.

- **Deployer**
    - The deployer is responsible for actually running and deploying the application on the node.
    - Deployer is connected to a load balancer and checks the job to be run requires a shared or standalone environment. Accordingly, it retrieves the nodes from the node manager, or the load balancer, and deploys the instance.
    - It then conveys to the application manager about the node and the application/instance running on it.

- **Load Balancer**
    - Load Balancer primarily communicates with Deployer and Node Manager.
    - At the request of the deployer, it retrieves a list of active nodes and provides the efficient node with minimal load.

- **Application Manager**
    - The application manager is responsible for providing the facility for deploying an application or running an instance of an application.
    - This module is also responsible for requesting event-based actions.

- **Authentication Manager**
    - This Module will authenticate every incoming user.

- **Nodes and Node Manager**
    - Nodes are the computing object which is responsible for the execution of an application instance or various platform components that take place.

    - Node Manager needs to manage multiple nodes i.e. their load, functionality, usage, etc and provides the deployer a new node when needed.

- **Monitoring and Fault Tolerance**
    - This module continuously monitors the components for failures and allows the system to operate properly, even in case of failure of one or more of its components.

- **Platform Initialization**
    - This module is used to initialize all the components and other modules of the platform, using their configuration.

- **Sensor Manager**
    - Sensor managers take care of sending data to applications as per request and interact with the sensor database for sensor-related data.

- **Model Manager**
    - The model manager is responsible for managing the model uploaded by the data scientists on the platform. Also provides an output of the model to the application instances when needed.

- **Notification Manager**
    - This is responsible to notify the user of the platform/application via email or text message, SMS.

4.3. List the 4 major parts each team will take one part

- Application Management, Notification Management.
- Node Management, Scheduler, Deployer, and Load Balancer
- Platform Initializer, Monitoring & Fault Tolerance

- Model Manager, Sensor manager & Platform Management.

5. Use cases

5.1. List what the users can do with the solution?

- Users can do the following tasks on the platform:
    - Can import their model along with config file in the specified format as artifacts that can be deployed on the platform.
    - Can request predictions from their models and get predicted target values.
    - Can build, deploy IoT-based applications as well.
    - Can request to run existing instances of any application, the platform will take care of it.
    - Need not worry about resource management such as load balancing, the health of the components, communication mechanism, etc.

5.2. Who are the type of user's name, domain/vertical, role, what they are trying to do when they need the system?

- Following are the types of users who come into the picture when the usage of the platform is concerned:
    - **Application developer**: build and upload application to the platform along with the specified configuration files.
    - **Data Scientist**: Uploads trained rule-based AI models in some specified format to the platform.
    - **Application user**: normal user who uses an existing instance application on the platform for a certain period.
    - **Platform admin**: Provide the info such as Input formats, API beforehand.

5.3. At least 5 usage scenarios. Give name and a brief 3-5 lines description.

- **Wellness and fitness analysis**: The data from various smart devices such as wearables, fitbits, personal medical devices, mobile phones, etc can be used in Anomaly detection in recorded medical data or finding any historic correlation resulting in real-time disease management at low cost and improved quality of life for patients.

- **Smart attendance using face recognition**: The data collected from cameras can be further fed into the AI model to recognize the people present in class. This could act as an attendance system without any manual intervention.

- **Utilities usage prediction:** The sensory data from Energy, water, or gas meters can be used in historical analysis, usage prediction, and demand-supply prediction. This can significantly reduce the cost and resource-saving.

- **Real-time vehicle tracking and optimization** for logistics: Improved service levels, lower cost, the carbon footprint can be achieved using the interference of data from RFID tags, onboard vehicle gateway devices to deduce visualization, predictions, optimization, and decision support systems for associated transport systems.

- **Agriculture**: Rule-based AI model can help to interfere with the type of crops to plant, the water requirements, etc, based on the sensory data like temperature, humidity, soil moisture, etc, along with the weather pattern of the area.

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

- **Smart attendance using face recognition**(Test use Case): Online classes have taken over the education sector during this covid time by storm and now have made children forget the importance of a classroom and interaction with their peers and professors. But, one of the most tedious tasks for a professor was to take the attendance of the class. To simplify that there have been many new ideas in this field such as biometric, card swipe, etc. We use Facial Recognition to identify the person in the class and automatically mark their attendance. Here, we use one sensor which is a camera that will be installed in every class.

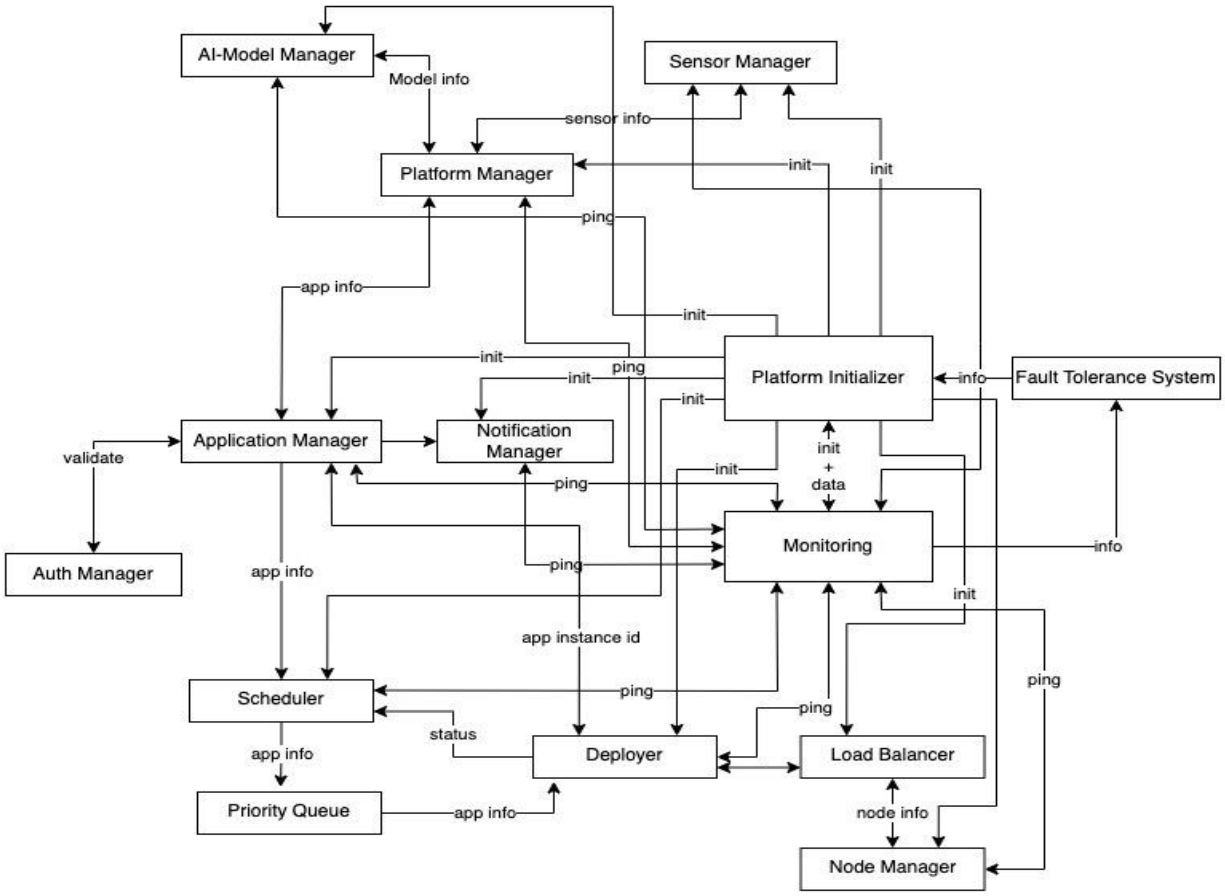
- **Use Cases:-**
    - When a student enters the class, the camera will detect the photo and send it to the model which will create a box around his face and write the name above it. The professor can then in one click add their attendance to the system.
    - As there are also offline exams held in the same classes, this recognition technique can be used to proctor the exam also, as the name and details of the person will be visible on the screen.
    - Hostel security can be amped up by using the technology. Figuring out if a person has entered the hostel which he does not have permission to, or creating ruckus in the campus can easily be detected using our application. If someone is found guilty, all their details will be forwarded via email to the respected authority to take action.

7. Subsystems

7.1. Key subsystems in project

- Platform Manager
- Application Manager
- Notification Manager
- Model Manager
- Sensor Manager
- Authentication Manager
- Node Manager
- Load Balancer
- Fault Tolerance and Monitoring
- Initializer
- Deployer

7.2. A block diagram of all subsystems



7.3. Interactions involved across these subsystems

- Application Database
- Node Database
- Sensor Database
- AI Database
- User Database

7.4. Protocols Mechanisms.

- *Communication using socket*: The client should know the server IP and the port on which it is running.

- *Communication using Kafka*: All the communicating parties should know the Kafka server details.

7.5. Registry & Repository

- *Application Manager - Authentication Manager*: Users' details and their authentication and authorization are taken care of by the Authentication Manager and pass the corresponding data to the Application Manager for which the request is raised.

- *Application manager - Scheduler*: The application manager will send the scheduler the start time, end time, priority, etc. based on which the application will be pushed into the priority queue for further deployment.

- *Scheduler - Deployer:* Scheduler will add the application deployment request into the priority queue, based on the output of the queue, the scheduler will pass application data to the deployer for deployment.

- *Application Manager - Deployer:* The deployer after deploying the application on one of the nodes, will send the application instance ID.

- *Deployer - Load balancer:* The deployer will ask for the active node to the load balancer, in return Load balancer will return the node information that the deployer will use to deploy the application.

- *Load Balancer - Node Manager:* Load balancer fetches the list of all the active nodes from the node manager, and finds the node with the least load among all.

- *Platform Manager - Sensor Manager - Application Manager - AI Model Manager*: The existing model details and sensor details residing on the platform are passed to the Application manager for integration with the application. Platform Manager can add new sensors to our platform via Sensor Manager.