Requirement Document

## **Group-4**

**Team Members:**

ADITYA SHARMA 2021201016

PULKIT GUPTA 2021201037

BHUPENDRA SHARMA 2021201020

SHREYASH AGRAWAL 2021201074

SANKALP THAKUR 2021201042

KAMAL KUMAR PHOOLWANI 2021201054

ANCHAL JAKHMOLA 2021201051

TEJ MITTAL 2021202021

VINEET AGRAWAL 2021201049

SHRAVAN SHARMA 2021201058

RAHUL KATYAL 2021201083

AARADHYA GUPTA 2019114010

ASHWIN VAISH 2019114014

**INDEX**

**1) Introduction**

**1.1) Scope**

**2) Intended Use**

**2.1) Assumptions and dependencies**

**3) System Features and Requirements**

**3.1) Platform Requirements**

**3.1.1) Deployment of the platform**

**3.1.2) Different actors on the platform**

**3.1.3) Applications overview of the platform**

**3.2) Functional Requirements**

**3.2.1) AI Model**

**3.2.3) Communication model**

**3.2.4) Server and service life cycle**

**3.2.5) Deployment of application on the platform**

**3.2.6) Registry & Repository**

**3.2.7) Interactions between modules**

**3.2.8) Packaging details**

**3.2.9) Configuration files details**

**3.2.10) Interaction of different actors with the platform**

**3.3) Non-Functional Requirements**

**3.3.1) Scalability**

**3.3.2) Accessibility of data**

**3.3.3) Specification about the application**

**3.3.4) UI and CLI for interaction**

**4) Key Functions**

**4.1) Block Diagram of all components**

**4.2) Brief description of each component**

**4.3)Major Parts on which the team will work**

**5) Use cases**

**5.1) List what the users can do with the solution**

**5.2) Different users involved are**

**5.3) Five usage scenarios**

**6) Primary test case**

**6.1) Name of use case**

**6.2) Company executing the use case**

**6.3) Description of the use case purpose, interactions and what will the users**

**benefit from**

**7) Subsystems**

**7.1) Key subsystems in the project**

**7.2) Block diagram of all subsystems**

**7.3) Interactions involved across these subsystems**

**7.4) External interfaces with the system**

**7.5) Registry and repository**

**8) Brief overview of each of the four parts**

**8.1) The four major parts on which the teams will work**

**8.2) Functional overview of each part**

**1) Introduction**

According to McKinsey Global Survey on AI, AI adoption continues to grow and the benefits remain significant— though, in the COVID-19 pandemic’s first year, they were felt more strongly on the cost-savings front than the top line. As AI adoption grows, AI model building is also growing, and therefore applications that are using AI models are also increasing.

AI model development and application development are two separate entities. So we are building a platform that connects these data scientists (who build the AI model) and the application developer.

The data scientist can upload and deploy his/her AI model to our platform and our platform makes available this model to the app developer as a service, so, the app developer can use these AI models by API call to his/her application.

The app developer will upload/deploy his/her application to our platform and then our platform makes available this application to the end-user. Then, the end-user can use this application by providing details about the sensors that are going to be used by the application as input.

**1.1) Scope:**

Our platform provides a set of services. The data scientist will upload the AI model to our platform according to the contract. This AI model will be used by the app developer to create AI-based applications so our platform provides these AI models as a service.

The app developer will upload the application to the platform according to provided contract and our platform make available this application to the end-user. End-user will use this application by providing information related to which sensors will be used in this application.

Other than this our platform provides other services also, like authentication service which authenticate all incoming requests to the platform, validation service that validates all files that are provided by actors according to the specified format, load balancing to equally distribute the load to all nodes.

Other services like scheduler, service life cycle manager, bootstrap mechanism, sensor management, node management are also provided by our platform.

**2) Intended Use:**

This platform will be used by a data scientist for deploying AI models that he/she builds according to a contract provided by us. The data scientist will require to provide a contract in which he/she need to mention how this model is going to be used by some application developer what type of data it will take as input and what type of data it will give as an output.

A contract that is provided by the data scientist will be used by the application developer to use a particular AI model. So, application developers need to give input to the AI model as mentioned and will get output as mentioned. Application developers will use this model and build their applications. This application will be deployed to the platform by the application developers.

End users will use applications that are deployed by application developers. so, our platform makes available all deployed applications to end-users.

**2.1) Assumptions and dependencies:**

We will be assuming that the data scientist will deploy the model by following all the constraints that are provided in the required file otherwise our platform will not work as expected. There is some fault tolerance provided by our platform which will handle these errors up to some extent but we expect the user to follow all these constraints.

We are assuming that all the AI models that are deployed to the platform are runnable according to some normal configurations of processor requirements and space requirements.

**3) System Features and Requirements**

**3.1) Platform Requirements**

**3.1.1) Deployment of the platform**

A computation resource would be required to build and deploy our platform. It can be an on-premise or cloud system and for microservice, a virtual instance or container will also be required.

**3.1.2) Different actors on the platform**

**Data Scientist:**

Data Scientist will build ML/AI Model which will be deployed on the platform and used by Application Developers.

**Platform Developer:**

Platform Developer will develop and integrate all the major components of the platform and simultaneously do the testing.

**Platform Administrator:**

Platform Administrator will be responsible for core administration, support platform technologies, and will ensure overall performance, availability. Along with this administrator also handle sensor registration that will be supported by our platform.

**Application Developer:**

Application Developer will build and deploy the application using the ML/AI model present in the platform and sensor available on the platform.

**User:**

All the applications deployed on the platform will be used by this actor.

**3.1.3) Applications overview of the platform**

Applications related to AI, which involves trained models and collecting sensor data can be deployed on the platform.

* Different actors involved need to specify their part of the information, in order to run the application.
* The sample application for understanding is as follows:

An application to know whether a person would survive the titanic voyage. So the user will give the input on the application and the output will predict if the person will survive or not

->Use Cases involved:

The trained model will take input from sensor data and will give appropriate output.

Hence, such applications with trained models, sensor details, and algorithms can be implemented on the platform.

**3.2) Functional Requirements**

**3.2.1) AI Model**

**1. Development of AI Model**

* Data scientists will build the required machine required model in python. It will involve the preprocessing of the input as required before inserting input into the machine learning model.
* Data Scientist will save the final model as a .pkl file.

**2. Packaging of Model**

Data scientists will have to bundle up the required files and upload the zip for the model to deploy on the platform. The zip must contain the following files:

* **Pickle file**: This file will have the final saved model.
* **Contract.json:** This file will contain all the procedures which need to be generated on the platform to run the .pkl file, which will eventually run the AI model built by Data Scientists.
  + **Other required files:** This will contain details about the initial configuration that is required before running the model on the platform, like the .py file which contains class logic in which pre/post-processing, etc functions will be defined.

The above files will be zipped and uploaded on the platform.

**3. Upload and deployment of the model**

Model upload:

* The data scientist will provide a zip file containing the files mentioned above.
* Before uploading the model, we validate files present in a zip file.
* After validation AI model uploader service creates two files, an onboarding script for running this model and an API for this model.
* AI model uploader will store model name, model ID, and path where these files are present.

Model deployment:

* Model deployer will have model ID, by using this ID it will fetch address where model present.
* Then model deployer will run an onboarding script that presents on fetched address from the database.
* This onboarding script will install all prerequisite modules and make a model in a runnable state.
* After deployment of the AI model, it will be present to predict value at a given endpoint.

**3.2.2) Application**

**1. Registering sensors**

* When a new sensor is added, it gets registered on the platform. While registration details of sensors like sensor id, Sensor Type, location, Streaming rate, etc., are stored on the sensor repository.
* Registration of the sensor is done to retrieve information and data when the application developer uses them.

**2. Interaction with sensors**

All the sensors will connect through the sensor manager that gets the raw data of streams. Sensor Manager will process the raw data into a proper model input form and stream it which is consumed by the application on the server.

**3. Development of application on the platform**

* The application developer will create an application and attach then zip all the required files(format specified on the platform) and upload zip on the platform.
* App Validator will verify all the uploaded files, extract the necessary details from uploaded files and store them into the database.
* It will then send AI model details to Life Cycle Server and sensors details to the Sensor manager.
* When the end-user request to use the application, an instance of the application will run on the node and be assigned to the user.
* AI model and sensors will be used as required by the application.

**4. Identification of sensors for data binding**

* During sensor registration, properties of the sensor like sensor id, type, location, etc., are stored on the sensor repository. The sensor manager identifies each sensor uniquely based on these parameters.
* When the sensor manager receives a request, the manager identifies the sensor based on the parameters received.

**5. Data Binding to the application**

* When an application needs sensor data or any processing that needs sensor data, it communicates with the sensor manager and passes the sensor's parameters(sensor id, location, type, etc.) from which it needs data.
* The sensor manager on receiving the request will identify the sensor based on the parameters received. It will then collect the data from the sensor and return the data in the specified format.

**3.2.3) Communication model**

Following communication types are required in our platform:

* The continuous stream of data is produced by sensors and consumed by applications whenever required.

**3.2.4) Server and service life cycle**

Server Life Cycle Manager manages the life cycle of the running server. It can be considered as a node manager that keeps the information about available and free nodes at any instance if time. Whenever the service lifecycle manager requests for a new node or a list of active nodes, the server lifecycle manager provides the same.

The service Lifecycle Manager acts as a deployment manager. It receives details of service from the scheduler. It communicates with the server lifecycle manager for getting the address of the node/server on which the service is to be run.

**3.2.5) Deployment of application on the platform**

After all validations, the job is handed over to the deployment service by the scheduler. The deployer checks if the job requires a free node, or tries to run the job on one of the existing active nodes. In the case of shared environment jobs, the deployer makes use of Load Balancer, to select the node with the least load. Once the job or algorithm is done running, and the node is no longer being used, it is free and can be listed for use again.

**3.2.6) Registry & Repository**

The repository will consist of all static data that need to be persisted in the system like Node information, application data, and User data) The registry will be responsible to keep the metadata when the platform is up.

**3.2.7) Interactions between modules**

The following modules will be present in our Platform:

* Deployment Service
* Sensor manager
* Node Manager
* Scheduler
* Request Manager
* Bootstrap Mechanism
* Service Lifecycle Manager

**Request Manager and Scheduler**

Request Manager will receive a request from the user and provide the details of the request to the scheduler. This request is the end user’s request to use an application.

**Scheduler and Deployment Service**

The scheduler will identify the application that is to be used by the end-user and it will request a Deployment service to Deploy it.

**Deployment Service and Node manager**

Deployment service will provide application details and provide it to the node manager and it is the responsibility of the node manager to provide node to the application by identifying the load distribution.

**Service Lifecycle Manager and Deployment Service**

The service lifecycle manager will pass the signal(that is receiver from bootstrap mechanism) to initiate startup service to deployment service.

**The bootstrap mechanism and Service Lifecycle Manager**  
 On platform startup, the Bootstrap mechanism will send a signal to the Lifecycle manager to initiate the service on the platform.

**Bootstrap mechanism and Node Manager**

On platform startup, the Bootstrap mechanism will send a signal to the Node manager to initiate the node needed at the system start.

**Sensor Manager and Node Manager**

During application execution (that is the request by the user), the Application needs data from sensor. So in this scenario, the Senso manager will interact with the node manager via message broker and node manager will pass sensor data to an appropriate node that is execution the application.

**3.2.8) Packaging details**

We will mainly have 2 packages:

**Model Package-**This Will contain all the files required for a model(like a contract file, pickle file, and other required files).

**Application Package-**This package will consist of the code for the main application along with other dependent files related to the application.

**3.2.9) Configuration files details**

**Model Configuration File**

It will contain configuration and executable files related to a model following a predefined contract between Data scientist and Platform.

**Application Configuration File**

It will contain configuration and executable files related to a model following a predefined contract between Data scientist and Platform.

**Sensor Configuration File**

It will contain configuration files with details related to the sensor.

**3.2.10) Interaction of different actors with the platform**

**Data Scientist:**

Data Scientists will interact with the platform to upload the ML models built by them.

**Platform Administrator:**

The Platform Administrator will interact with the Analytic module to check the status of the system. It will interact with the request manager for sensor registration etc.

**Application Developer:**

The Application Developer will interact with the Application Database to pick a suitable model for its application and upload the built application to our platform. It can interact with the sensor manager to turn on/off the sensor as per the application necessity.

**End-User:**

The user will interact with the system using a user interface and will use the application developed by the Application developer. It will have only access to the presentation part of the platform and need to authenticate in order to use the platform services.

**3.3) Non-Functional Requirements:**

**3.3.1) Scalability**

**1. Platform**

Any number of AI models can be deployed and sensors can be registered to the platform.

**2. Application**

If an application instance is consuming resources above a specified threshold, then that instance will be shifted to a new node.

**3.3.2) Accessibility of data**

**1. Application**

The application will specify in the configuration which AI models, sensors it will use at the time of deployment. Based on the entry of the model and sensor in the configuration file it will be able to use the services of the platform.

**2. Sensors**

Sensors will be able to access data based on the requirements of the application developed.

**3.3.3) Specification about the application**

* Scalability
* Reliability
* Usability
* Maintainability

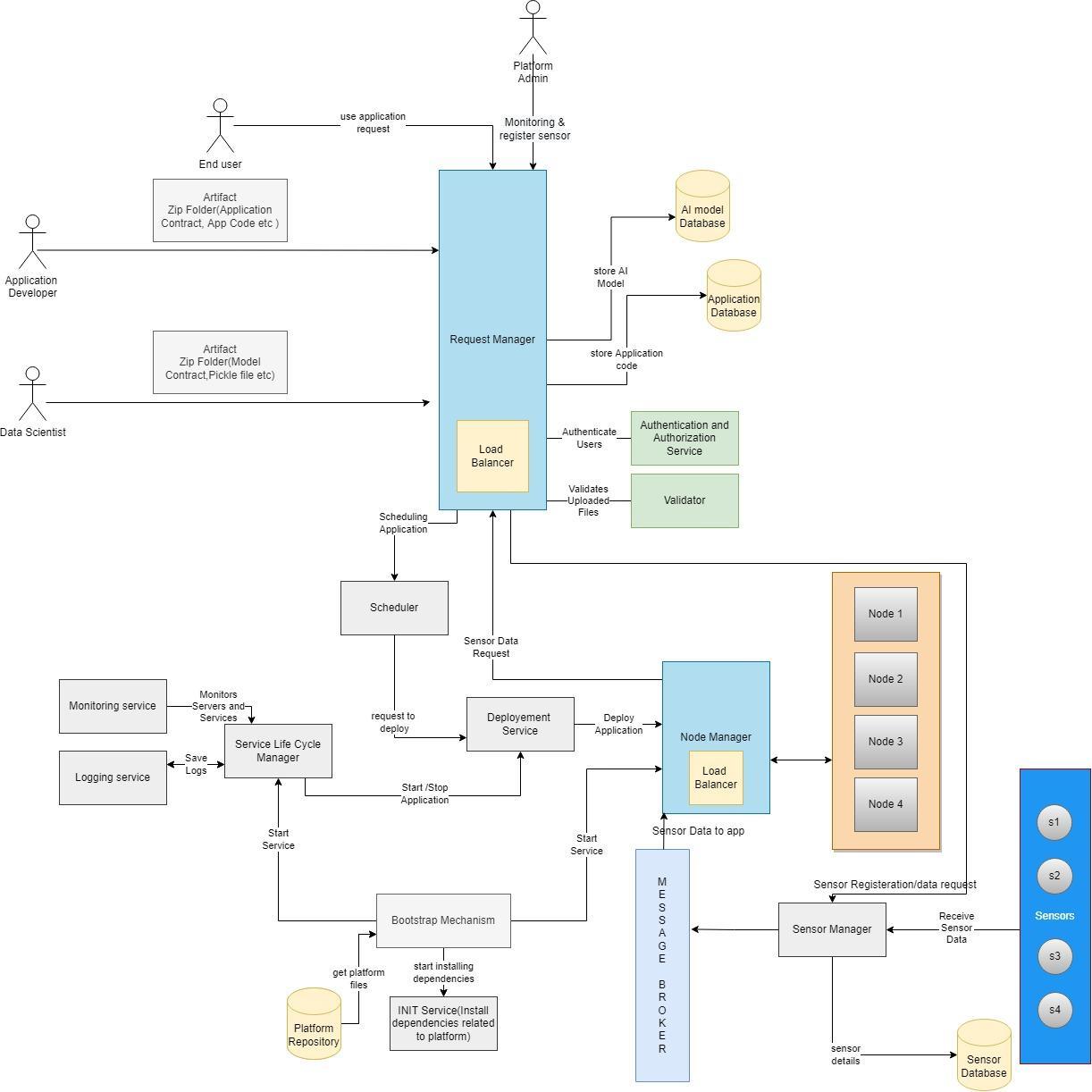
**3.3.4) UI and CLI for interaction**

**UI for ML Developer:** ML Developer will interact with the platform and can upload the AI Model package and can check for existing packages.

**UI for Application Developer:** The application Developer can interact with the platform and can build its application using existing models.It can also monitor the running applications.

**4) Key Functions**

**4.1) Block Diagram of all components**



**4.2) Brief description of each component:**

* **Authenticator:** The actors(Data Scientist, App Developer) would Register and log in to the Platform. The Authenticator will authenticate and Authorize the user so that they can perform their respective tasks.
* **Request Manager:** Users' requests will now be routed to the Request Manager, who will handle all of the routine tasks. The Request Manager would also be in charge of Load Balancing, ensuring that traffic is distributed evenly across all servers and that no server is overburdened.
* **Service Lifecycle Manager:** The Service Lifecycle Manager will be in charge of all service-related tasks such as starting a new service instance, recovering a service instance, terminating the entire service or terminating a service instance. It will also be in charge of monitoring all services.It is also in charge of providing an easy-to-use general-purpose logging system that records all events associated with a specific service.
* **Scheduler**: It will perform the scheduling of services on the platform. It will schedule model deployment or app deployment. It will schedule the model for a particular time interval (between start time and end time).
* **Deployer:** It will perform deployment of Machine Learning Models created by the data scientist and Applications created by App Developers. Basically, it will package the model and deploy it to any server instance.
* **Platform Manager:** The platform manager will perform tasks related to sensors and controllers. Such as adding a new sensor and controller to the database. Sensor Management, Fault Tolerance, Controller, Communication will be done through this.
* **Repository Manager:** All the data will be present in the Repository (User data, as well as some model-related data like .pkl file and config file). So all this data related management will be done through the Repository Manager.
* **UI Manager:** All the data that the user should provide should be through an interface. So the task related to the management of the user interface is done by UI Manager.
* **Validator:** Checking if the input and output are in the correct form and valid is another important function. This will be done by Validator. The validator will also be responsible for the validation of the contracts (checking if the data given is as per the contract or not).
* **Bootstrap Mechanism:** It will do all the platform startup-related tasks. It will start all the essential services which will be present in the INIT service and access the corresponding files in the Platform Repository.
* **Sensor Manager:** It will manage all the sensor-related data like Sensor Type, Location (IP Address, Port number, Physical Location), and Assign a Sensor ID to it, which will be used when the end-user will be using the app. It will also send these details to the Message Broker.

**4.3)Major Parts on which the team will work:**

* Request manager, and Application upload
* Sensor manager, and service lifecycle manager
* App deployment, and ML model Deployment
* ML model upload, logging, and Bootstrap mechanism

**5) Use cases**

**5.1) List what the users can do with the solution**

Users should be able to use services provided by the applications (deployed on the platform) using Application ID and Application Name. Users would be provided with details of the location of sensors, Number of instances of Application, Scheduling details, etc. but implementation details would not be provided to them.

**5.2) Different users involved are.**

* **App developer**

Their role is to create applications using models already deployed on the platform. They would require Sensor Type and Model and would be using contracts given by data scientists.

* **Data scientist**

Their role is to create a model and deploy it on the platform which then can be used by Application Developers. They would be deploying a zip file that contains the model(pickle file) and other dependencies.

* **Enduser**

These users would be using services provided by the application deployed(by Application Developer) on the platform using Application ID, Application Name, etc.

* **Admin**

It will handle the registration of sensors that will be allowed by our platform. It will also handle the administration of the platform.

**5.3) Five usage scenarios.**

* **Face Mask Detection:**

We will be using our model to detect whether the person on the webcam is wearing a mask or not.

* **Smart home:**

Users would be able to turn ON the AC without being physically present inside the room. While locking the door, users can set controllers to automatically close the curtains, turn off devices and ensure their home is protected against any trespassers.

* **Smart factory:**

Users would be able to monitor an entire production process, from supply chain management to manufacturing tools and even the work of individual operators on the shop floor.

* **Face Recognition:**

This is used by law enforcement to search for a criminal or person of interest on the run.

* **Signature Verification:**

Users could verify a signature just by scanning it via an application that could be deployed on the platform and which could help prevent fraud and thus increase the accuracy.

**6) Primary test case**

**6.1) Name of use case:**

Create an AI application using an ML model.

**6.2) Company executing the use case:**

Consumer technology services company.

**6.3) Description of the use case purpose, interactions and what will the users benefit from:**

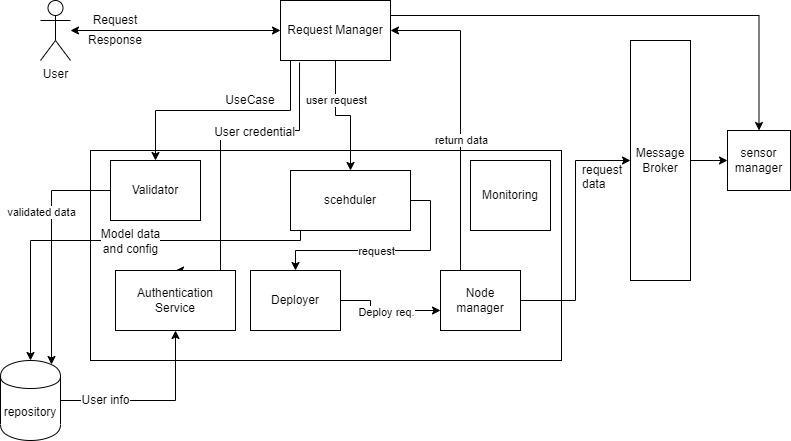
* Machine Learning models will be generated by the data scientists, who will aggregate the data, and utilize an ML algorithm.
* App developers who wish to utilize will create applications utilizing available models and adding them to their projects.
* Users benefit as the scientists creating the model and developers creating the applications have a common platform to speed up development.

**7) Subsystems**

**7.1) Key subsystems in the project**

* Authentication
* Deployment
* Scheduling
* Logging
* Monitoring
* Request Manager

**7.2) Block diagram of all subsystems:**



**7.3) Interactions involved across these subsystems:**

* Request managers can receive a request from any actor. The request is directed to its respective service or validator as needed. The request is scheduled according to the availability of resources.
* The request is authorized by authentication and authorization service.
* The AI runtime server fetches the model data from the Repository and returns predictions to the Request manager.
* The monitoring service interacts with the gateway, server instances, and registry to update the current state of the system.
* The logging system interacts with the central server and saves logs in the log files.
* Nodes communicate with the sensor through the node manager

**7.4) External interfaces with the system:**

* **User interfaces:**
  + An interface where ML developers can upload the models.
  + An interface that allows users to use ML services and add applications.
  + An interface that allows the user/admin to see all the resources and their status.
* **Sensor interfaces:**

An interface that takes data from the sensors.

**7.5) Registry and repository:**

* The registry will contain the stats of every service (what instances are running on what machines, binary files location of each service in the common file system, etc.)
* The repository will contain one directory per user which will, in turn, contain one directory per service and will contain all the version revisions of that service.

**8) Brief overview of each of the four parts**

**8.1) The four major parts on which the teams will work:**

* Request manager, and Application upload
* Sensor manager, and service lifecycle manager
* App deployment, and ML model Deployment
* ML model upload, logging, and Bootstrap mechanism

**8.2) Functional overview of each part:**

**Team 1: Sensor manager, and service lifecycle manager:**

**Functional Overview**

**Service Lifecycle Manager:**The Service Lifecycle Manager will be in charge of all service-related tasks such as starting a new service instance, recovering a service instance, terminating the entire service or terminating a service instance. It will also be in charge of monitoring all services.It is also in charge of providing an easy-to-use general-purpose logging system that records all events associated with a specific service. Service Lifecycle Manager acts as a deployment manager. It receives details of service from the scheduler. It communicates with the server lifecycle manager for getting the address of the node/server on which the service is to be run.

**Sensor Manager:**During application execution (which is initiated by the user), data from the sensor is required. Sensor manager will communicate with node management via message broker in this situation, and node manager will transfer sensor data to the relevant node that is running the application.It will keep track of all sensor-related information, such as Sensor Type, Location (IP Address, Port Number, Physical Location), and assign a Sensor ID to it, which will be utilized when the end user uses the app. These details will also be sent to the Message Broker.

**List of sub-systems in this part**

* Service Lifecycle Manager
* Sensor Manager

**List of services/capabilities in the part**

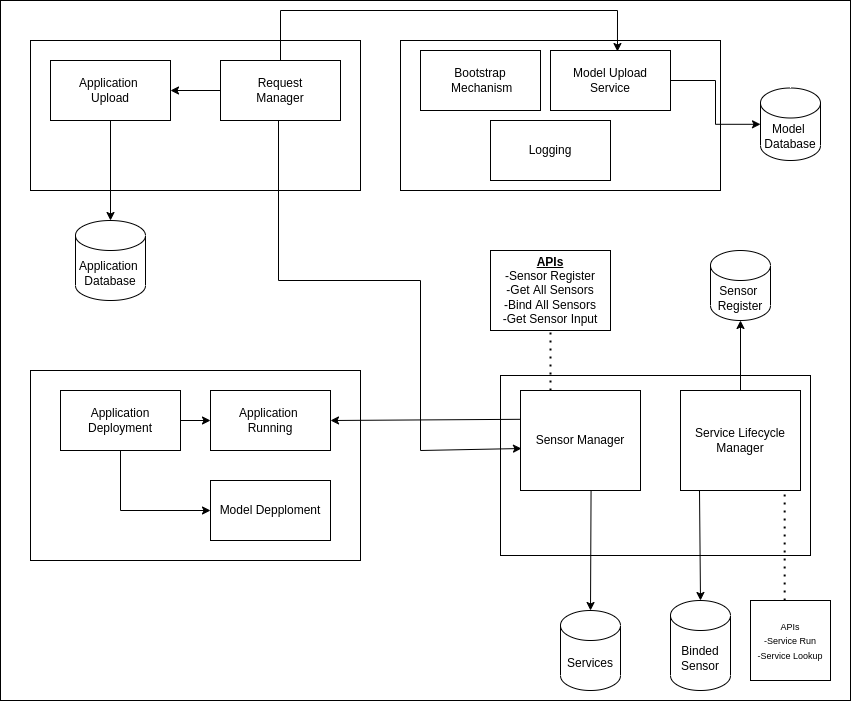
**Service Lifecycle Manager:**

* It is responsible for all the service related tasks.
* Starting a new service instance is done by Service LCM.
* It is also responsible for terminating a service instance.
* Monitoring of all the services is also done by Service LCM.
* It also records all events associated with a specific service through a logging system.
* It also receives the details of service from the scheduler so that it can start the service.

**Sensor Manager:**

* It will add all the types of sensors supported by the platform to the Sensor Register database.
* Controller registration is performed in the same way as Sensor registration
* If the sensor is registered successfully then a success response is sent to the platform admin else not.
* It will bind all the sensors by the port number, IP and location provided by the end user of the app.
* If the sensors are binded successfully then a success response is sent to the platform admin else it will send an error message.

**Interactions between this and other parts:**

****

**Team 2: App deployment, and ML model Deployment:**

**Functional Overview:**

* **Model deployer:** This service is responsible for deploying the model. The service is provided with 500 predetermined ports. The model deployer keeps track of these ports and deployers the model on an empty port.
* **Application deployer:** This service is responsible for deploying the application. It is also responsible for ensuring all the resources like sensors and models are available before the app is deployed. Again, 500 predetermined ports are given to the application deployer, and the application is started on one of these empty ports. The app deployer reads the contract and finds the models that are required. Then it checks if the model is deployed or not. If the model is not deployed then it sends the request to the model deployer for deployment. It also takes the ip port of the sensor from the user and sends a binding request to the sensor manager. The request returns a binding id which is passed to the application at startup.

**List of sub-systems in this part**

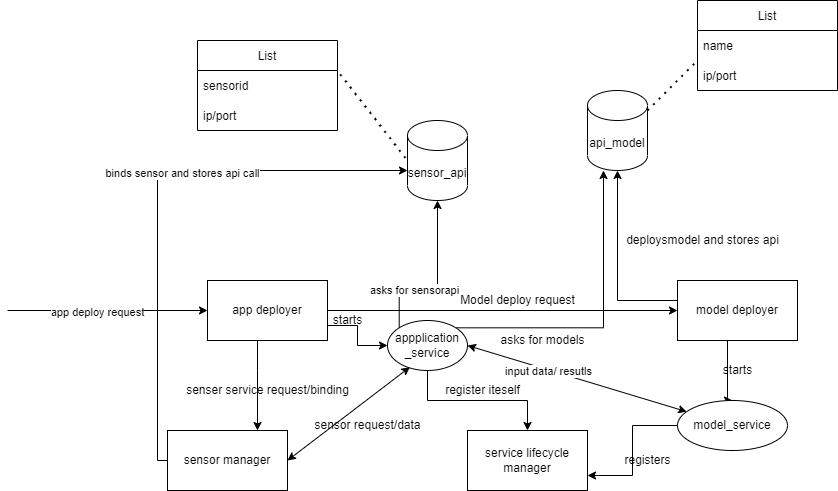
* Model Deployer
* Application Deployer

**List of services/capabilities in the part:**

This part is responsible for :

* Deployment of models as a service.
* Deployment of Application.
* Checking if the model is deployed. If not send a deployment request.
* Help bind the sensor to the platform by taking input from the user.
* Keeping track of Open and occupied ports for the deployment of model and application.

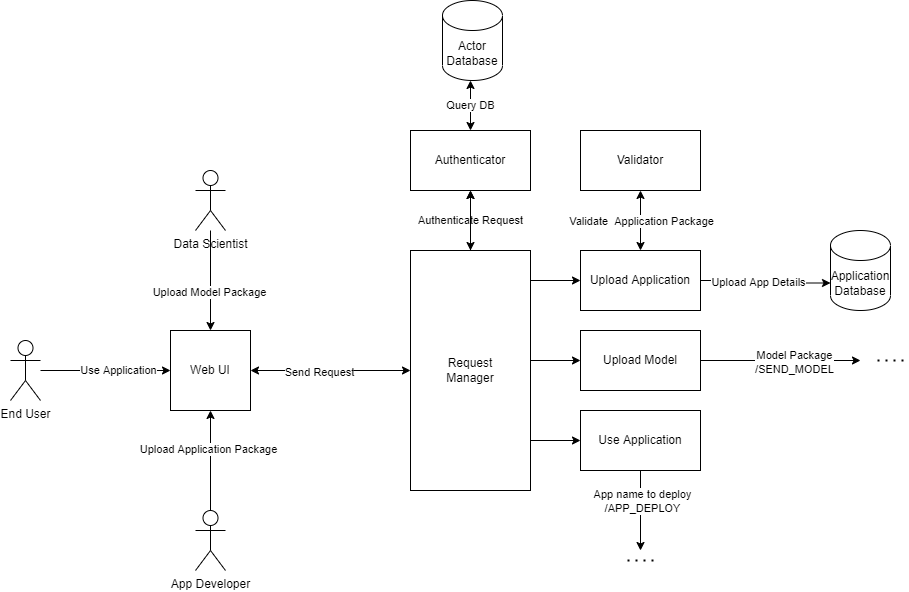
**Interactions between this and other parts:**



**Team 3: Request manager, and Application upload:**

Our team objective was to build a request manager for the platform, application upload functionality, and validator. Request Manager's objective was to route the request in the platform to proper endpoints. All the requests whether from actors or inside the component will be handled by the request manager. Application upload service provides application developers to upload their application, and the validator validates the zip uploaded by the application developer. It will validate if all the required files are present as well as contract.json inside the zip folder.

1. **Request Manager**: This component is responsible for taking requests from 3 users that our platforms have. Users will interact with WEB UI and for each user, we have a different view present. User will end request and Request Manager will resolve request and authenticate the user and will redirect it to the relevant module. Request managers will have different endpoints responsible for different functionalities.
2. **Validator**: Validator is responsible for validating the Application package by checking its Schema. It will check the contract and validate if all possible fields are there.
3. **Authenticator**: Authenticator will get a request from the request manager for different actors and it will check if the user is valid and will respond to the request manager accordingly.
4. **Upload Application:** This component will receive an Application package, send it to the validator for validation of the package, and will upload application details in Application DB if the package is valid.
5. **Upload Model:** This component will receive a model package and will send it to the service responsible for Model validation and upload.
6. **Use Application:** This component will receive the request from the end-user for using an application. It will call Deploy app API by providing the app name.



**Team 4: ML model upload, logging, and Bootstrap mechanism**

**Functional Overview:**

* **ML Model uploader:** This service is responsible for uploading the model to the platform. It takes a zip file on the */upload\_model* endpoint and validates it. This zip file contains the model pickle file, contract.json, and other required python files. This service generates model API and created an onboarding script for a given model that requires at the time of model deployment.
* **Logging:** This service is responsible for storing all logs created from all services. At the given endpoint, this service takes logs and stores them in the database.
* **Bootstrapping:** This service is responsible for starting the platform. It is basically a script file that first installs required modules and then starts all services.

**List of sub-systems in this part**

* **ML Model uploader**
* **Logging**
* **Bootstrapping**

**List of services/capabilities in the part:**

This part is responsible for :

* Validation of zip file and contract file.
* Creation of API for each model to be uploaded.
* Creation of onboarding script that is responsible for the deployment of the AI model.
* Creation of logging API.
* Creation of bootstrapping script file.

**Interactions between this and other parts:**

#### 

#### 