**INDEX**

# **1.** **Intro to the Project**

### **2.** **Use Cases/Requirements**

**3.** **Test cases**

**3.1 Overall Test cases**

# **4.** **Solution design considerations**

**4.1** **Design big picture**

**4.2** **Environment to be used**

# **4.3** **Technologies to be**

# **used**

# **4.4** **Overall system flow & interactions**

**4.5** **IOT system design**

# **4.6** **Interaction with Sensors**

**4.7** **Server system design**

# **4.8** **Approach for communication & connectivity**

**4.9** **Registry & repository**

**4.10** **Node & service lifecycle manager**

**4.11** **Scheduler**

# **4.12** **Load Balancing**

**4.13** **Interactions between modules**

**4.14** **Wire and file formats**

**5.** **Application Model and User’s view of the system**

# **6.** **Key Data structures**

# **7.** **Interactions & Interfaces**

# **8.** **Persistence**

**9.** **The modules; that the four team will work on**

**Project: Design Outline**

# **1.** **Intro to the Project**

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 info 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.

### **2.** **Use Cases/Requirements (Very detailed documentation of requirements. For the group documentation it will be about overall platform while in the Team documentation it will be more focused about specific components)**

* The contracts provided to actors (Data Scientist, App Developer, etc.) should be followed properly.
* All files which are uploaded are validated and verified according to the platform norms shared with the user.
* Any sensor to be binded is validated first and then configured for use in an application instance.

**3.** **Test cases (will clarify what your module will do. Build on your operations and use cases listed)**

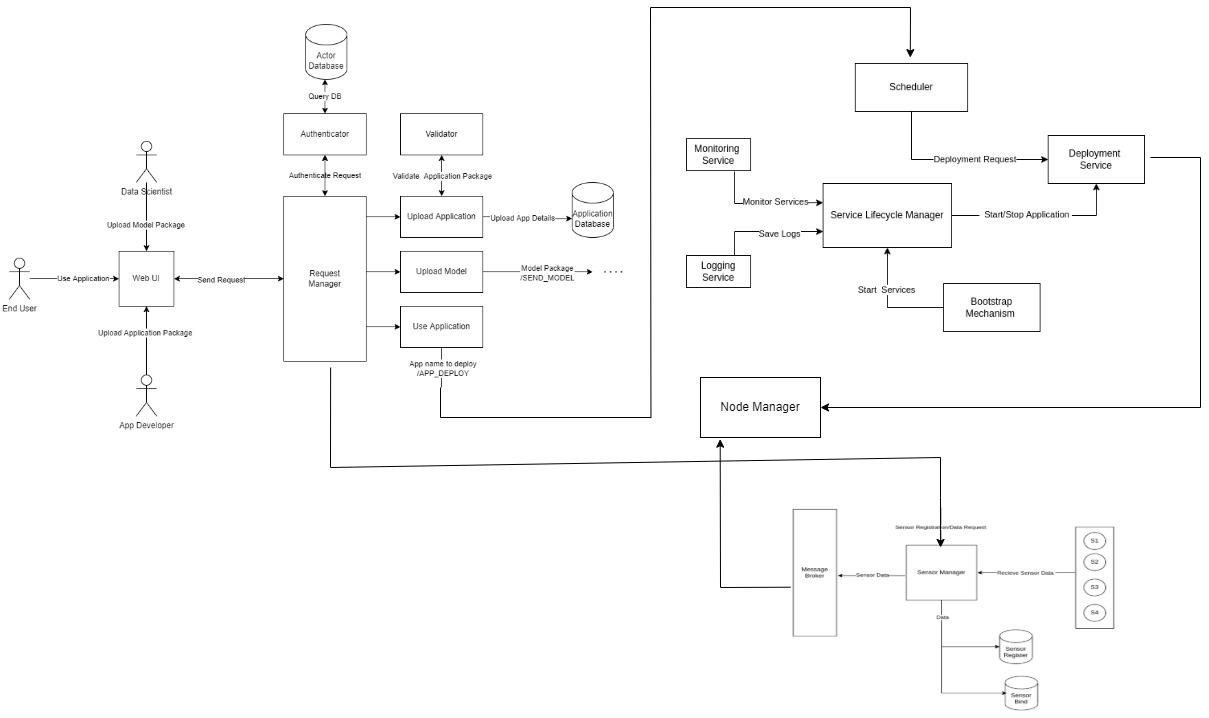
**3.1 Overall Test cases**

* Application Package/Zip validation testing
* Application Deployment Testing
* Sensor type configuration testing
* Sensor instance registration testing
* User authentication testing
* API calls testing
* Fault Tolerance testing

# **4.** **Solution design considerations**

(Group level- to be discussed by all four teams, and captured in all four team-design documents).

# **4.1** **Design big picture**



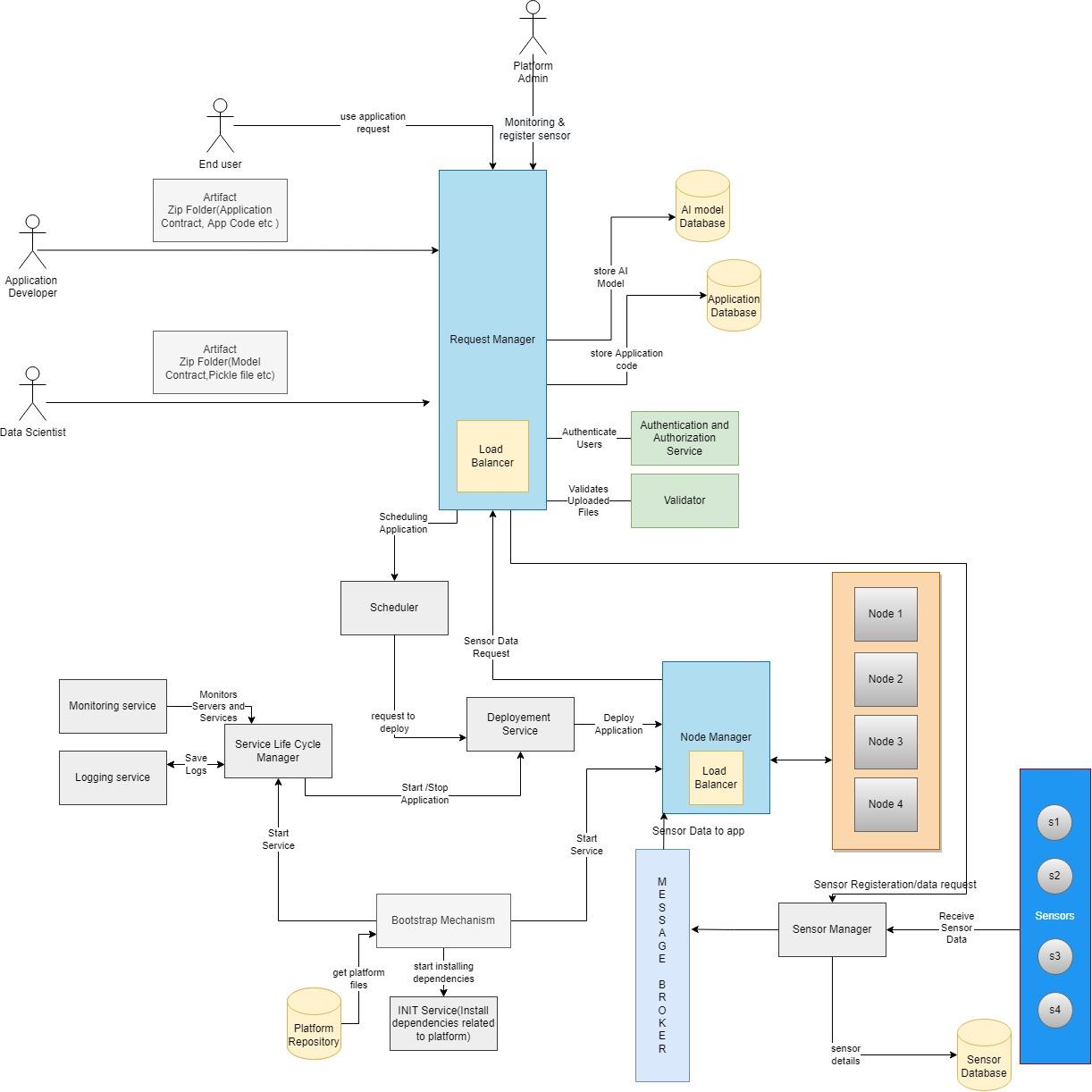
# **4.2** **Environment to be used**

* OS: Linux

# **4.3** **Technologies to be used**

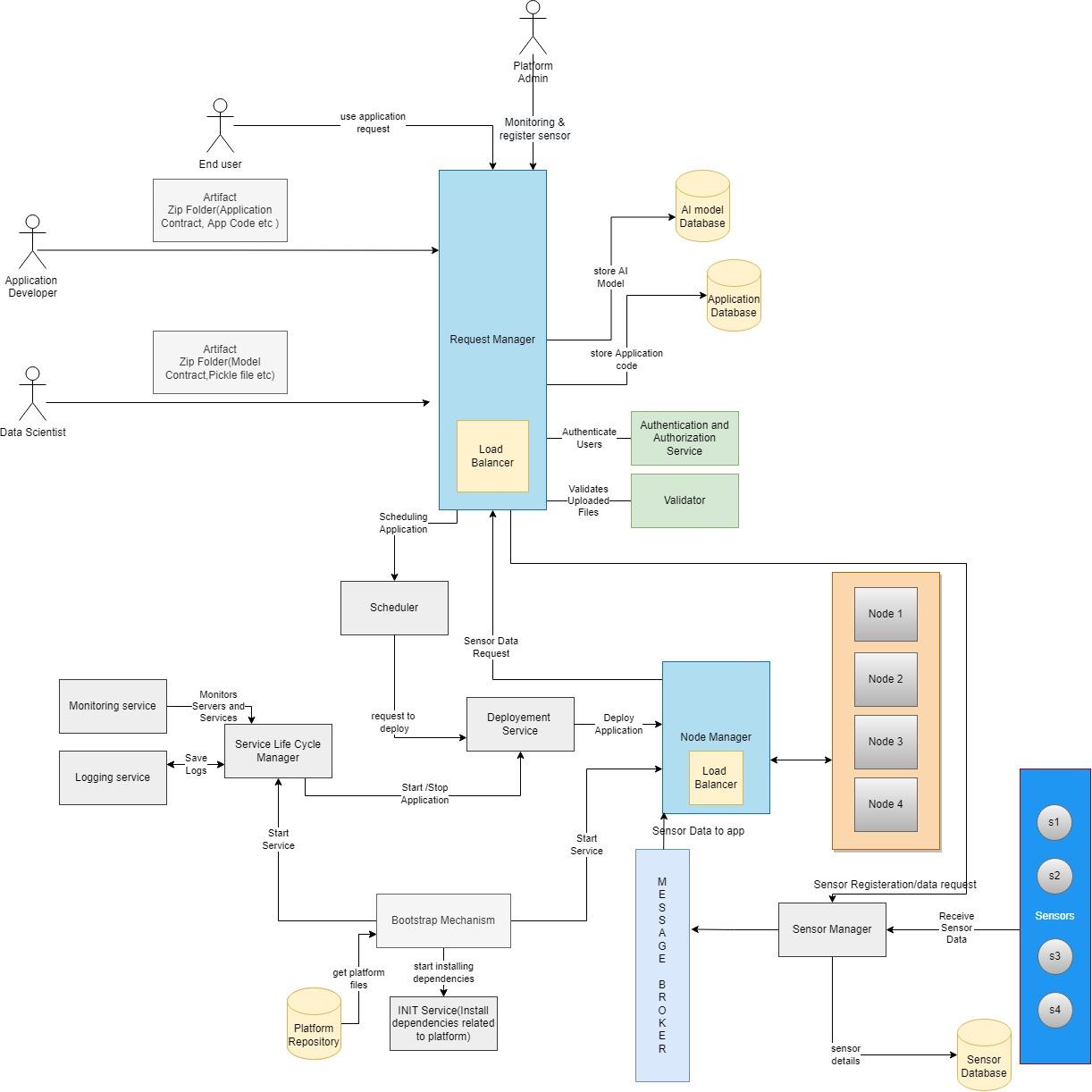
* Flask : Python Framework
* Developing REST APIs for communication between all services
* Third-party libraries for support
* Apache Kafka
* Docker
* Mysql

# **4.4** **Overall system flow & interactions**



**4.5** **IOT system design**

# **4.6** **Interaction with Sensors**



**4.7** **Server system design**

# **4.8** **Approach for communication & connectivity**

* REST APIs for intercommunication
* Kafka Message queue

**4.9** **Registry & repository**

* Actor Database
* Application, AI Model and sensor Database
* Application and AI model Repository
* Platform Database
* Node Database

# **4.10** **Node & service lifecycle manager**

# **Node manager**

It maintains the information about the nodes running in the system, free nodes and the busy ones. It provides this information to deployment manager to deploy any application.

# **Service Lifecycle manager**

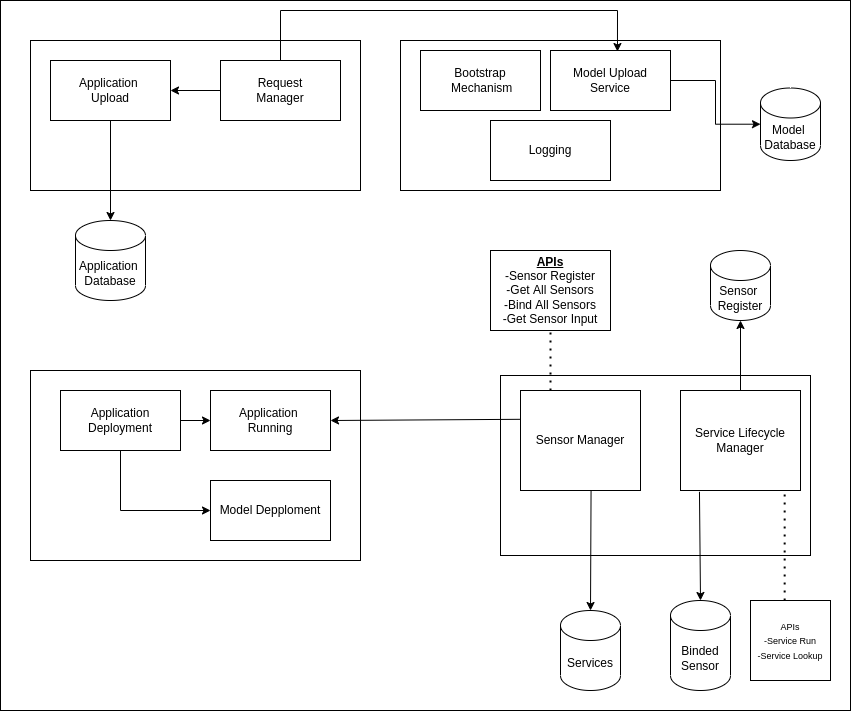
It keeps track of all the running services and is also responsible for either stopping or restarting services if need arises.

# 

**4.11** **Scheduler**

# **4.12** **Load Balancing**

**4.13** **Interactions between modules**

****

# **4.14** **Wire and file formats**

* JSON (for contracts)
* Pickle (for model)
* ZIP (uploading models and applications)

**5.** **Application Model and User’s view of the system**

Overview of the application dev model (dev steps/flow)

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

An application is a service or combination of services that will be made available to the end user through api. These services will be deployed on the platform after passing through various steps like validation, scheduling and availing the resources.

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

* Contract file: All dependencies and requirement are mentioned in it.
* App file: This is application developed by the app developer.
* Script file: This will be used to run the application on platform

**Structure of the files**

* Contract file:



**How is it deployed/setup. And where (on which module/process/location et al)**

Application developer provides the required file and send it to request manager, then request manager forward this request to application uploader. Application uploader will validate these files and after validation, application uploader will upload application to database.

When End user request to run the application then application deployer service will handle this request. Application deployer service takes info like, app name, sensor to be used, sensor location etc. After taking these info, application deployer service will check if required model in this application is in running state or not and also bind the sensor that are required in this application.

After making all dependencies up, application deployer will deploy the application by running the script file that are provided by the application developer.

**How to start & stop**

The end user will send request to run the particular application to request manager than that request will be forwarded to the application deployer service. Then application deployer service will start the application.

After using the application, end user will send a new request to stop the application then application deployer will stop the running application.

**How to access/invoke**

The end user will access the functionality of the application though the apis provided by the services of that application. After the application is started the the frontend/ ui is provided with the api call of that service by the application these api calls can then be used for interaction between end user and the application.

**More..?** NO

Application artifacts

- (what all does the user build, package, deploy or configure)

- Service files

The service files are are python files that would contain flask server code with end functionality provided as api. The ports and IP are provided at runtime by the platform. The service files are assumed to have api calls already present in them.

- Packaging

An application can have multiple services. The application package name should be same as the name of application name. The package contains a sub directory names services, a configuration file and a shell script to initiate an application. The services directory will contain a directory for each service where the directory name is same as service name. Each of these directories will contain a service files , a shell script to run a service and a contract.

- Configuration

The configuration files for application must contain the application name, for each service - service name, relative location of run.sh , models required by the services of the application, and the sensors that are used by the service.

User admin interactions

- **How will the user use this system**

→Application developer will send the app.py file,contract.json file and run.sh file which will be zipped and send to the request manager for deployment of the application. Hence an application developer will use this system to deploy an application using the models and sensors present with the system.

End User will select which application to use that are present in the platform using the UI provided to him and finally the end user will use this to get predictions, etc.

-  **what & how to configure**

Application developer will send the services files,contract.json file and run.sh file which will be zipped in format aforementioned and send to the request manager the configue files for application must contain the application name, for each service - service name, relative location of run.sh , models required by the services of the application, and the sensors that are used by the service.

-  **What are How to deploy**

There are two things to be deployed one is the model deployment and other one is the application deployment. Application deployment will be done using the contract.json file and during this process we will check whether a model is deployed or not if a given model is not deployed then we will deploy it and continue with the application deployment part.

# **6.** **Key Data structures**

# **7.** **Interactions & Interfaces**

APIs

**POST /LOGIN\_USER/{ROLE}:** This API is used to send authentication requests for a user/actor along with its role and will provide access to corresponding users.

**POST /VALIDATE\_APP:**This API will take the App Package and validate if it is correct and can be used.

**POST /UPLOAD\_APP:**This API will save the relevant details of the App into the APP DB.

**POST /AUTHENTICATE:**This API will check if the credentials provided by the actor is valid by checking it with ACTOR DB.

**POST /UPLOAD\_MODEL:**This API is responsible for taking the model package,unzipping it,validating it and uploading it into MODEL DB.

**POST /DEPLOY\_APP:**This API will be used when an end user needs to use an application.Then the request will be sent to this API along with the app name and will be deployed further.

**REGISTER SENSOR:**This API will be used when the platform admin want to add a new sensor type to the database.

**BIND SENSORS:**  This API will be used when End user will send the sensor data to be binded so that the end user can use the application.

**SERVICE START:** Used to Start a new service instance when the application deployer wants to start a service.

**SERVICE LOOKUP:** It will be used by other components to find the port number of running service.If service is not running then error message will be returned .

# **8.** **Persistence**

**9.** **The modules; that the four team will work on**

**MODEL UPLOAD:**

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.

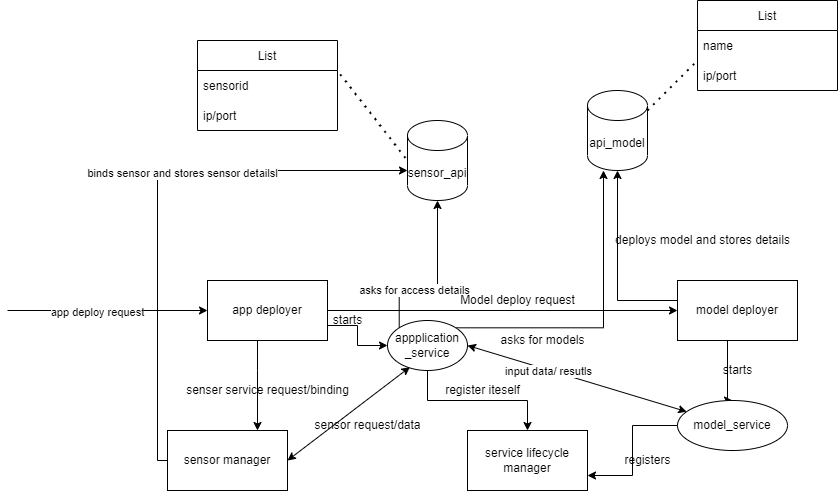
**Interaction with other module**

#### 

Internal design overview (additional details like above, for each model)

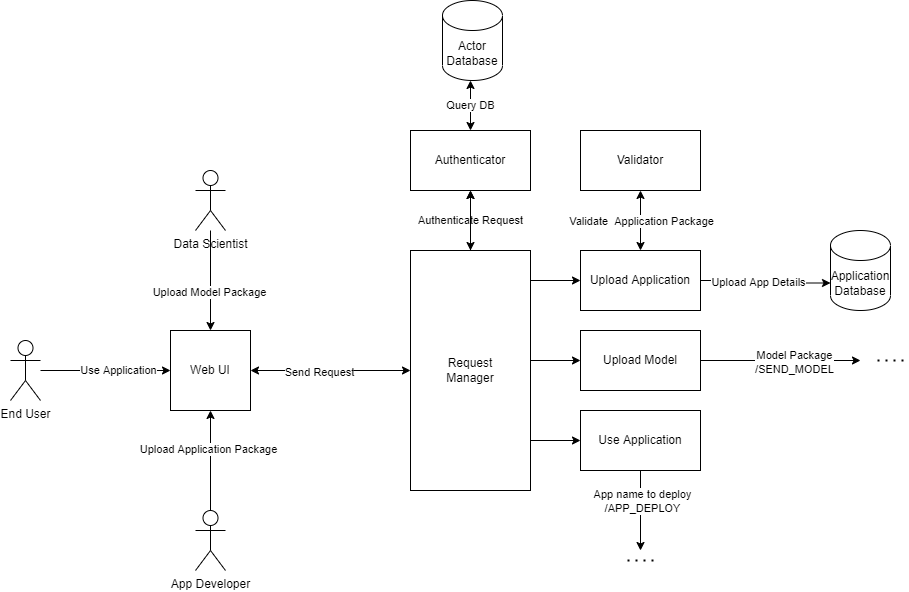
**APPLICATION AND MODEL DEPLOYMENT**

* Deployment of application after the request is received from the user.
* Deployment of models when the application that requires the model is being deployed.



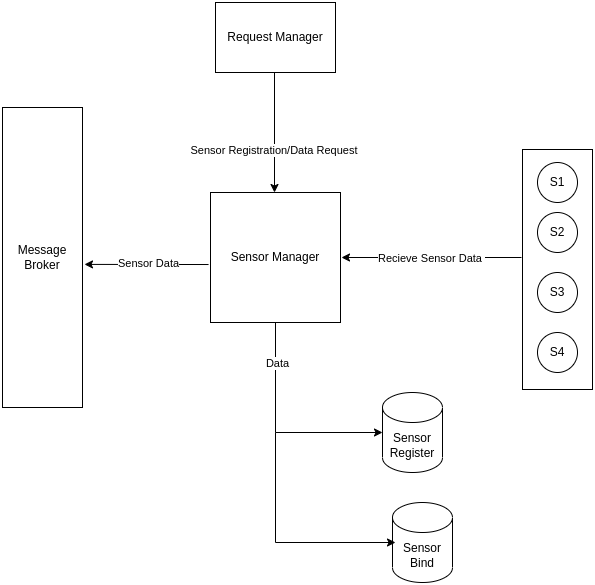
**REQUEST MANAGER,APP VALIDATE & UPLOAD,AUTHENTICATOR and WEB UI**

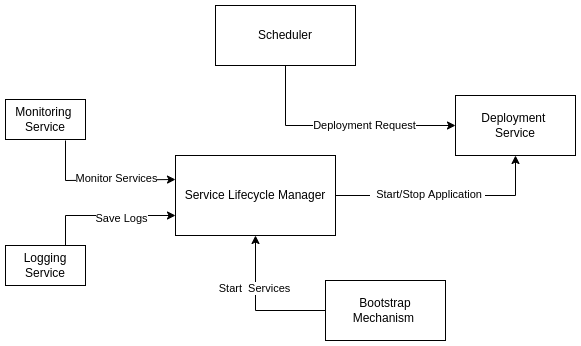
* Create a WEB UI for interaction of different actors with the platform.
* Validate and upload application details to Application DB.
* Authenticate request from actors and allow access to them



**Service Lifecycle Manager and Sensor Manager**

* The Service Lifecycle Manager will be in charge of all service-related tasks
* It will also be in charge of monitoring all services.
* Sensor manager will perform Sensor registration and Sensor Binding





# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **Low Level Design (for each of the modules)**

1. Lifecycle of the module

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

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

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

2. List the sub modules A block diagram

Describe the interactions among sub-modules

3. Brief overview of each sub module

- For each sub module:

#(parts of the whole solution

#give a meaningful name for each part)

- Describe the functionality of the module (7-10, bullet list)

- A brief block diagram of the internals (constituents)

- List the interactions (with which other module, external entities, file or network, comm. Framework et al)

- APIs & Classes .. and

4. Interactions between sub modules

- Interactions among the sub-modules

- APIs & .. such

- More..? (describe as relevant)

5. Interactions between other modules (describe as relevant)

# **6.** **Other design considerations**

(describe as relevant)