Tampere University

# ASSIGNMENT 02:
# MONITORING OF IOT DATA USING MVC PATTERN

## Design, Integration and Implementation

**Course:**

Industrial Informatics (AUT.840-2022-2023-1)

**Professor:**

Luis Gonzales Moctezuma

**Students:**

Nadun Ranasinghe (150906193)

Stepan Zalud (151062243)

**Date:**

14-12-2022

# Index

## Introduction

The following report encompasses all the details in regard to the implementation of the given tasks for Assignment 02 for the module Industrial Informatics. Two tasks were given to be completed and this report hopes to give a brief idea on how it had been completed.

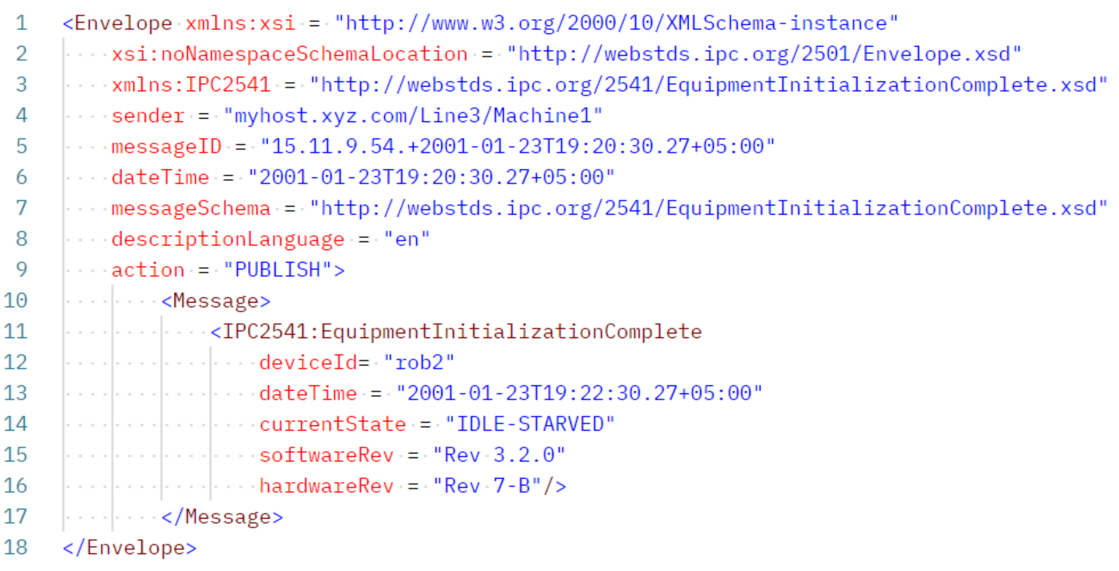
## Task 01

### Goal:

The goal of task 1 is to collect data that are provided by robot and publish them to broker (in this case HiveMQ). From here, data can be distributed as is needed. Final outcome should be data formatted that way, that they can be processed the same way that is done so with mocked data in task 2.

### Data published by robot:

Data that are provided by robot give information about robots change state. In this case 3 possible states are considered: READY-PROCESSING-EXECUTING, IDLE-STARVED and DOWN. Further valuable data contain timestamp telling when this event happened and robot id. The robot follows the CAMX machine state, which means that they are published in XML format. Example message from robot can be seen in Figure 1?. Events are received by the Flask server from the subscribed services of the server and the requests sent via the REST API by the client PC where are the provided data processed further.

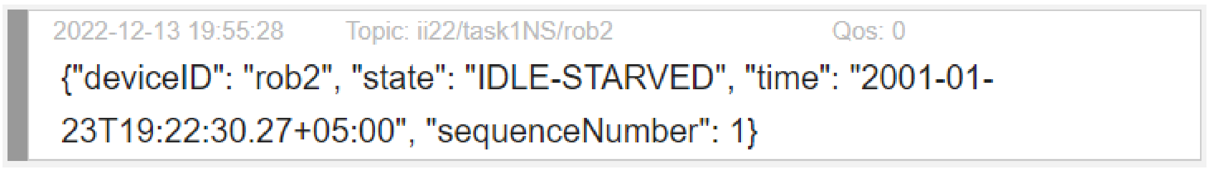


```xml
<Envelope xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation = "http://webstds.ipc.org/2501/Envelope.xsd"
    xmlns:IPC2541 = "http://webstds.ipc.org/2541/EquipmentInitializationComplete.xsd"
    sender = "myhost.xyz.com/Line3/Machine1"
    messageID = "15.11.9.54.+2001-01-23T19:20:30.27+05:00"
    dateTime = "2001-01-23T19:20:30.27+05:00"
    messageSchema = "http://webstds.ipc.org/2541/EquipmentInitializationComplete.xsd"
    descriptionLanguage = "en"
    action = "PUBLISH">
        <Message>
            <IPC2541:EquipmentInitializationComplete
                deviceId= "rob2"
                dateTime = "2001-01-23T19:22:30.27+05:00"
                currentState = "IDLE-STARVED"
                softwareRev = "Rev 3.2.0"
                hardwareRev = "Rev 7-B"/>
        </Message>
</Envelope>
```

Figure 1: Example of data provided by robot

## Processing the data:

Processing of data happens via python program. In order to create a functional program, flask, paho.mqtt.publish, json and xmltodict libraries were used. Each time an event happens, contained data are obtained from „/event" endpoint and desired data (deviceID, dateTime, currentState) are extracted from message. The program then prepares final message in JSON format. On top of already obtained data, every JSON message is attached sequence number based on deviceID. Program is designed in a way that allows processing of arbitrary number of devices without a need of changing the program. Final JSON message is then published to topic: ii22/task1NS/{deviceId} in HiveMQ server. Final message format example can be seen in figure 2.



```
2022-12-13 19:55:28        Topic: ii22/task1NS/rob2                    Qos: 0
{"deviceID": "rob2", "state": "IDLE-STARVED", "time": "2001-01-
23T19:22:30.27+05:00", "sequenceNumber": 1}
```

Figure 2: Example of message published to HiveMQ

## Conclusion:

In this task program that collects event as HTTP and publishes them to broker server as MQTT was designed. Eventhough task1 and task2 are not implemented together both parts could be connected easily if they were to be deployed on real hardware. The goal of this project is to solve a given orchestration task with the help of Object Oriented Programming. Also, it will give a good understanding on the role of web services during the integration of automation systems.

## Task 02

### Goal:

The goal given for this task is to implement a SCADA system, that is capable of storing and processing the data received from the industrial robots. The notifications from the robots will be received via the MQTT. These notifications will provide the states of the robots according to the CAMX nomenclature. With the received notifications, the SCADA should be able to process all received data in the back end and display them in an user-friendly web-ui. The received data must be processed to display real-time data, historical data and alarms generated by the robots.

### Back-End:

The back end system of the SCADA was based on Python. The team used libraries such as Paho-mqtt, Flask, threading, sqlite3 and datetime to receive and process data to send the requested information to the UI to be displayed for a user. The back end can be divided into the three main parts:

APP:

This part of the Backend is responsible for handling all the endpoints, that connect to the server where the front-end SCADA is hosted. When the front- end requests for these endpoints, the handling functions linked to these functions activate and works to complete the user's request. Also, this part of the backend is responsible for subscribing and receiving the data from the robots to be processed and displayed. The main functions that this page completes are as follows;

- Rendering the requested HTML files to be displayed: When the user requests to open a certain page, this system is capable of opening up the requested page on the server to be viewed. Each page included in the groups's system have been defined separately, rather than using one endpoint handler function, as it tend to bring in errors in receiving data from the back end to the HTML.
- Subscription to the broker, which broadcasts the messages on the robot's status in CAMX format and time stamps. When messages are received from the broker, they are converted into python dictionaries from JSON format, with the help of the JSON library. The converted dictionaries will be then sent for processing in sql databases.
- Timestamp conversion: The timestamp is received as a one long string, where the year, month, day, hour, minute, second and millisecond are together and unreadable. This line will be divided into the relevant parts and will be converted into an epoch for easy processing.
- Sending the real time data required for the requested robot to the HTML page, when requested.

- Sending the historical data requested for a user given period along with it’s KPIs calculated to be displayed on the page, for the user govern robot.
- Sending the alarm information for a user-given period to be displayed  for a requested robot.
- Sending the alarm notification when the robot stays in a certain state for too long to the SCADA interface.

One of the main drawbacks faced by the team during the testing phase was the inability to receive the robot states from the backend system. As the html pages kept requesting the wrong endpoint. This was found to be an issue with the browsers that was used. Deleting cache history of the browser seems to solve this problem. But, as a precautionary measure, all the pages required to be opened on the system were defined as separate endpoints.

## KPI Measurements:

This part of the backend system can be defined as the middle man of the back-end. It is responsible for processing the data stored in the database to derive and develop the information to be displayed on the UI, when requested from the user. This part of the system is responsible in providing the following information to the App part of the system;

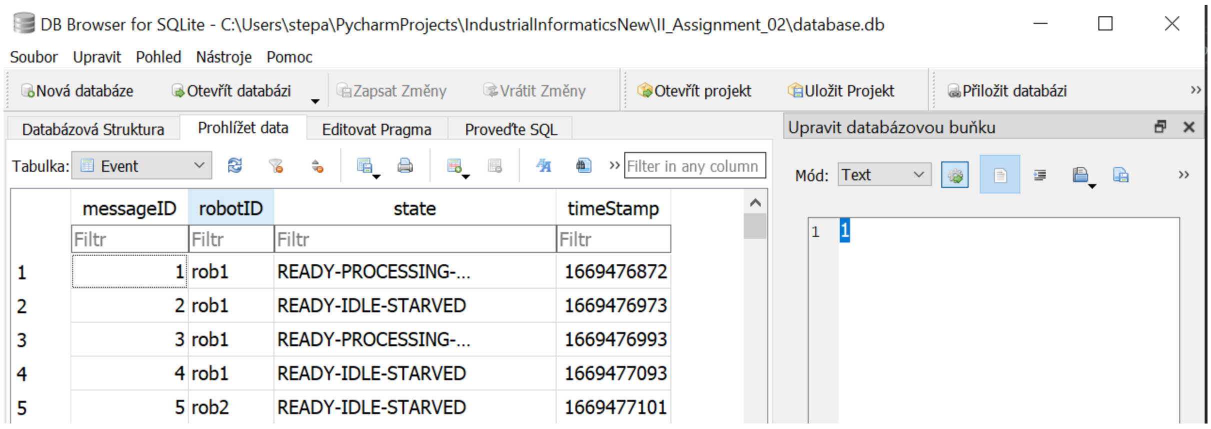
- The percentage values for the duration that robot stayed in each shown state for a given period of time.
- The mean time of failure, the duration between two failures.
- All the instances where the robot passed the given limit for the duration that it can stay in the IDLE or DOWN state. Limits were determined by observing the data as follows: IDLE limit = 20s, DOWN limit = 300 second. IDLE state duration varied starting from 20 seconds. DOWN state lasted 600 seconds constantly. Whenever limit threshold is exceeded, proper notification is displayed at web UI.
- All the data received from a robot for a given period of time in the form of a dictionary to be displayed as a table on the UI.
- The latest state received from the robot with the accurate time and date of receiving.

For the calculation of all the above mentioned information, the KPI part of the system directly communicated with the store data part which handled the database to receive the collected data and processed it in the relevant way to be displayed on the web-ui via the App part. The calculation of the accurate time was an issue that was faced by the team during this phase, as the time zones became a factor when hosting it on the server. This was resolved by adding a time factor that compensated for the difference.

Store Data:
This is the furthest part of the system, where it receives all the python dictionaries from the App part to be saved to a database called „database.db". All the sql queries are defined and executed in this part of the system. The following sql functions are being executed in this part:

- Creating a table named EVENT, with a auto incrementing Message ID, with the robotID, state and timestamp saved in separate columns.
- Saving each received robot state message converted to dictionary into the created database.
- Sending the requested data from the database. This will be done as follows;
    - When a single event is requested, the whole state will be sent back as a dictionary.
    - When a set of events between a given set of arguments is requested, all the events will be sent as a list of dictionaries for processing.
- The events are separated and requested under different arguments as follows;
    - All events on the database.(Implemented in program but not used in our web UI, but for testing purposes.)
    - All events based on the robID.
    - All events between a defined time window. (Implemented in program but not used in our web UI, but for testing purposes.)
    - Latest event saved based on the robID.
    - All events for a robID between a defined time window.

The data extracted using the above given functions are being utilized for the calculations and processing performed in the KPI part, which send the processed information to the App part to be sent to the front-end Web UI.



| | messageID | robotID | state | timeStamp |
|---|---|---|---|---|
| | Filtr | Filtr | Filtr | Filtr |
| 1 | 1 | rob1 | READY-PROCESSING-... | 1669476872 |
| 2 | 2 | rob1 | READY-IDLE-STARVED | 1669476973 |
| 3 | 3 | rob1 | READY-PROCESSING-... | 1669476993 |
| 4 | 4 | rob1 | READY-IDLE-STARVED | 1669477093 |
| 5 | 5 | rob2 | READY-IDLE-STARVED | 1669477101 |

Figure 3: Database used for the saving of the received events

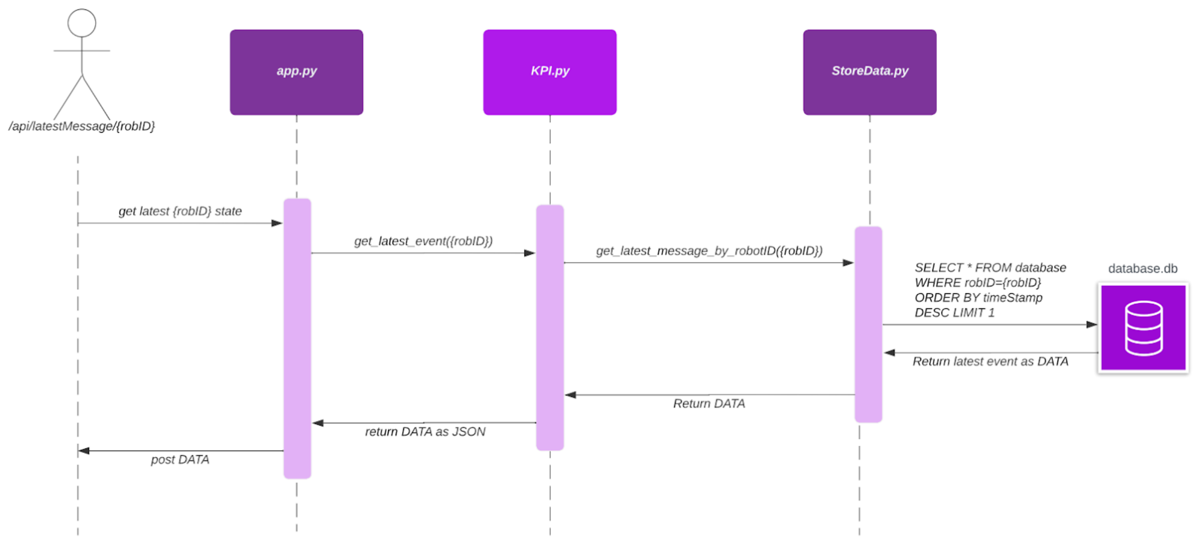

/api/latestMessage/{robID}
app.py
KPI.py
StoreData.py
get latest {robID} state
get_latest_event({robID})
get_latest_message_by_robotID({robID})
SELECT * FROM database WHERE robID={robID} ORDER BY timeStamp DESC LIMIT 1
database.db
Return latest event as DATA
Return DATA
return DATA as JSON
post DATA

Figure 4: Sequential diagram showing how the three parts(App, KPI, store_data) work together to function as the back-end

## Front-End UI:

The front-end of the SCADA consists of the Web UI that displays the requested information about the robots to the user. Our team followed page-divided approach system to fulfill the given requirements, where each requirement was displayed in unique pages. This was done, such that the user is capable of seeing each data separately with no visible restrictions from the other information being cluttered together. But the team made sure that the Current state of the robot and the timestamp at which it was received was displayed on each developed page, to give the user an understanding about the real time state of the robot while surveying through the other requirements. Four pages were defined to be sufficient cover up all the relevant requirements of the assignment;

### Robot Selection:

This page consists of a selection criteria for all given active robots. The addition of this page was to increase the flexibility in adding new robots to the system. When a robot is selected, it will open the dashboard of the selected robot. This is achieved by the robID sent with endpoint to pen the dashboard.

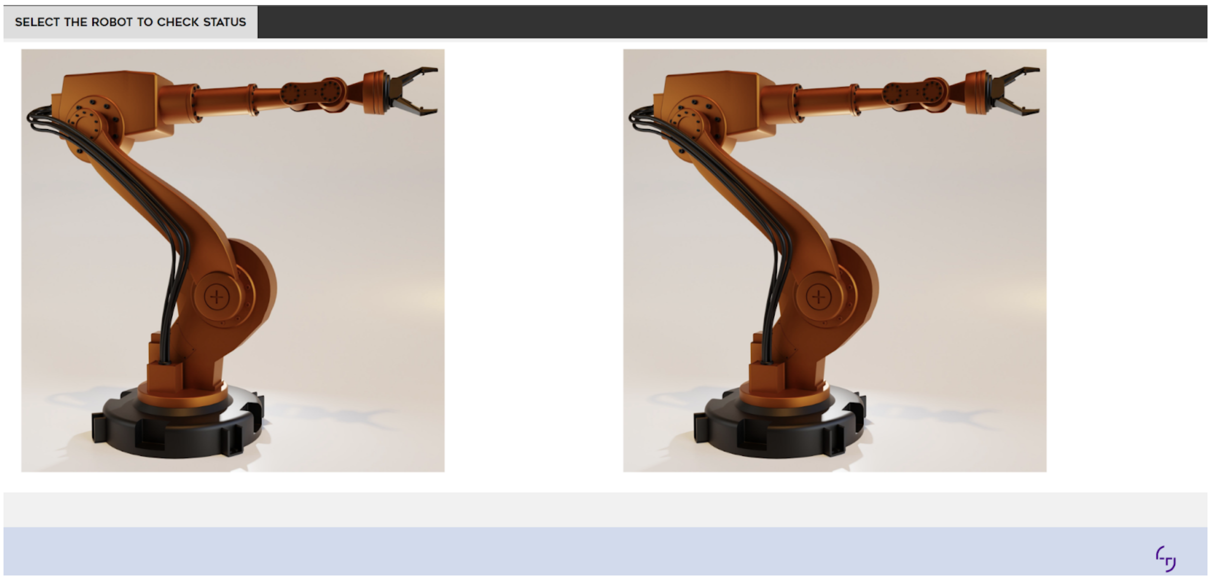
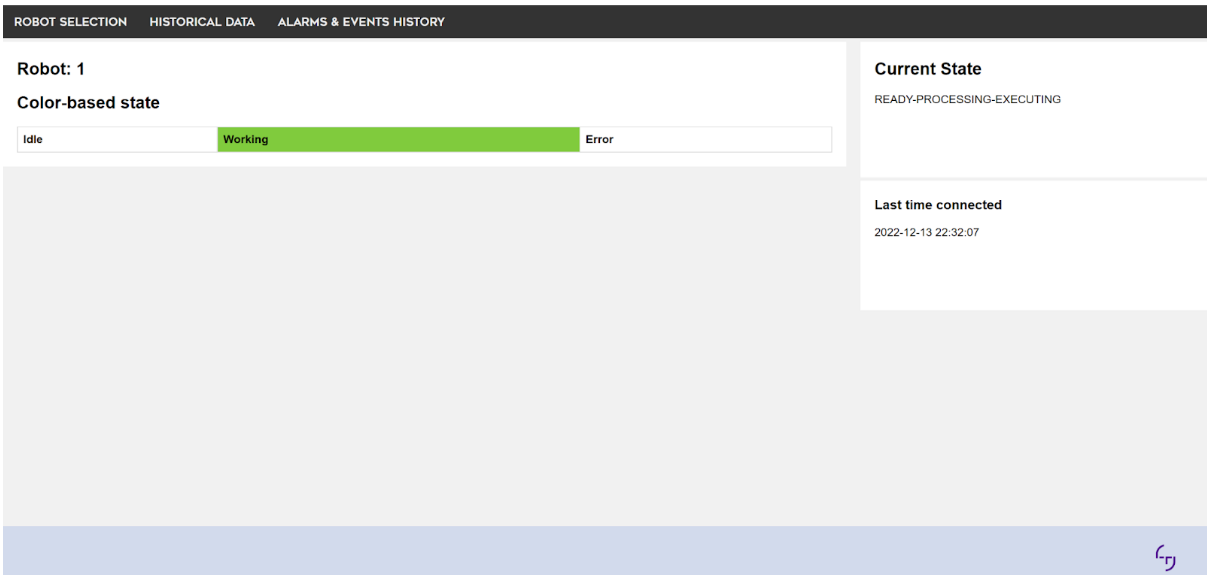


Figure 5: Snapshot of the Robot selection page

## Dashboard:

Contains all the real time data about the robots. This includes the current state, with color definition, Last connected time and date, and also the current state in CAMX format. Other than the state show based on the color, the other two tiles that depict the real time state and the timestamp at which it was received, will be brought forward to all the other pages as explained previously.



Robot: 1

Color-based state

Current State

READY-PROCESSING-EXECUTING

Last time connected

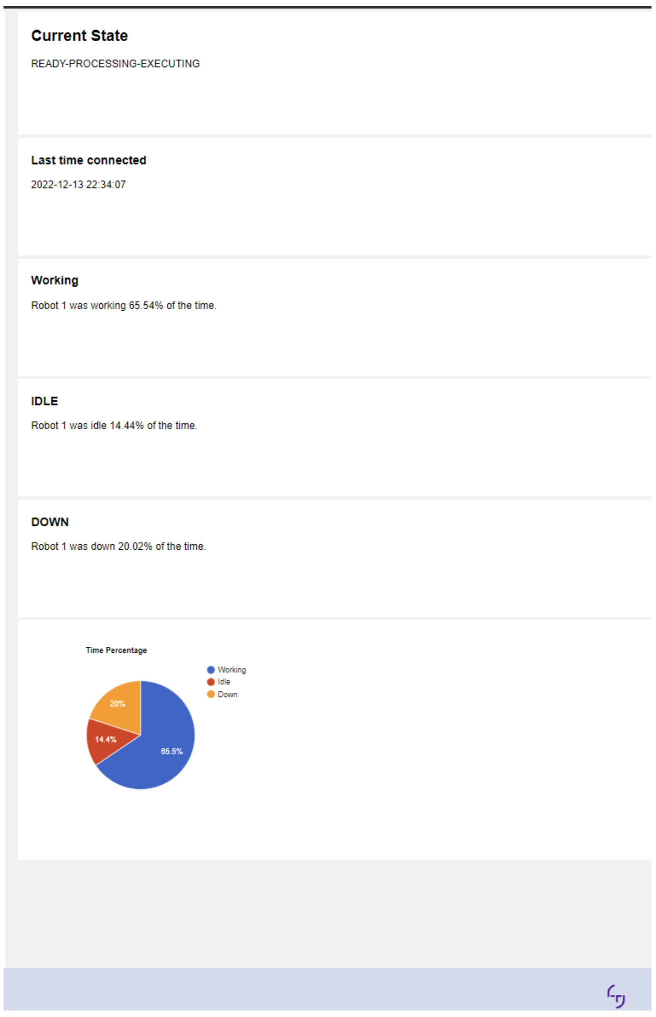
2022-12-13 22:32:07

Figure 6: Snapshot of the Dashboard page

Historical Data:

Fulfills the requirement to get the historical data, all the state notifications of the robot, within a user-defined time period. Also, for the same defined period, the page will display all the required KPIs. The displayed KPIs are as follows;

- The duration of time that the robot stayed in each state as a percentage.
- The state percentages displayed as a pie chart for better understanding.



**Current State**

READY-PROCESSING-EXECUTING

**Last time connected**

2022-12-13 22:34:07

**Working**

Robot 1 was working 65.54% of the time.

**IDLE**

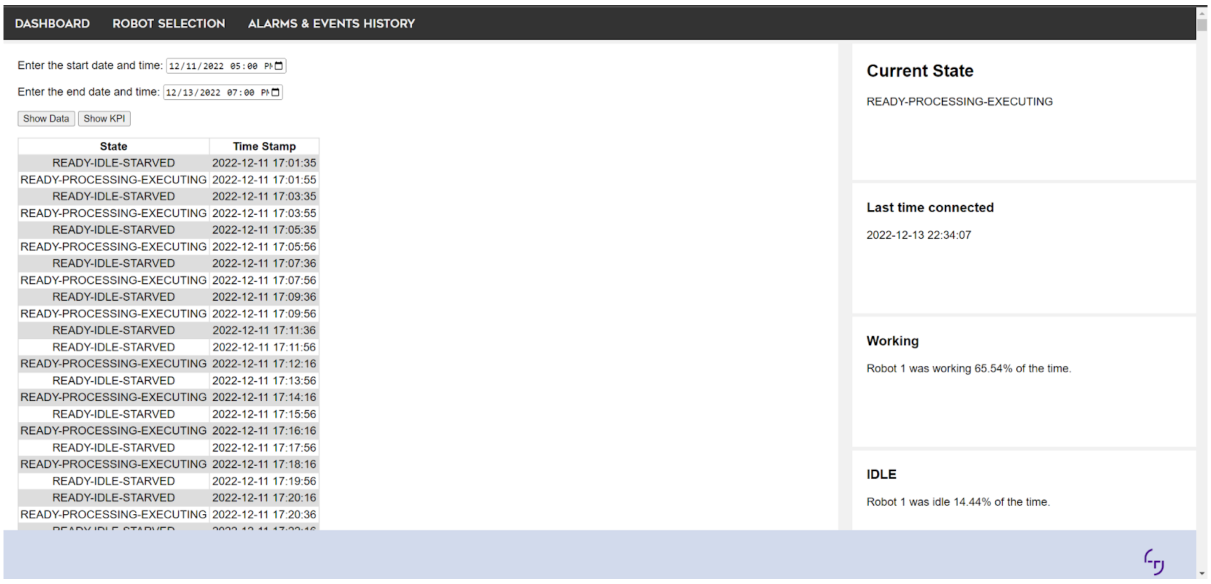
Robot 1 was idle 14.44% of the time.

**DOWN**

Robot 1 was down 20.02% of the time.

Figure 7: Snapshot of the KPIs shown on the Historical Data page

The historical data is displayed as a table for better user understanding.



Enter the start date and time: 12/11/2022 05:00 P

Enter the end date and time: 12/13/2022 07:00 P

| State | Time Stamp |
|---|---|
| READY-IDLE-STARVED | 2022-12-11 17:01:35 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:01:55 |
| READY-IDLE-STARVED | 2022-12-11 17:03:35 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:03:55 |
| READY-IDLE-STARVED | 2022-12-11 17:05:35 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:05:56 |
| READY-IDLE-STARVED | 2022-12-11 17:07:36 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:07:56 |
| READY-IDLE-STARVED | 2022-12-11 17:09:36 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:09:56 |
| READY-IDLE-STARVED | 2022-12-11 17:11:36 |
| READY-IDLE-STARVED | 2022-12-11 17:11:56 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:12:16 |
| READY-IDLE-STARVED | 2022-12-11 17:13:56 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:14:16 |
| READY-IDLE-STARVED | 2022-12-11 17:15:56 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:16:16 |
| READY-IDLE-STARVED | 2022-12-11 17:17:56 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:18:16 |
| READY-IDLE-STARVED | 2022-12-11 17:19:56 |
| READY-IDLE-STARVED | 2022-12-11 17:20:16 |
| READY-PROCESSING-EXECUTING | 2022-12-11 17:20:36 |

**Current State**

READY-PROCESSING-EXECUTING

**Last time connected**

2022-12-13 22:34:07

**Working**

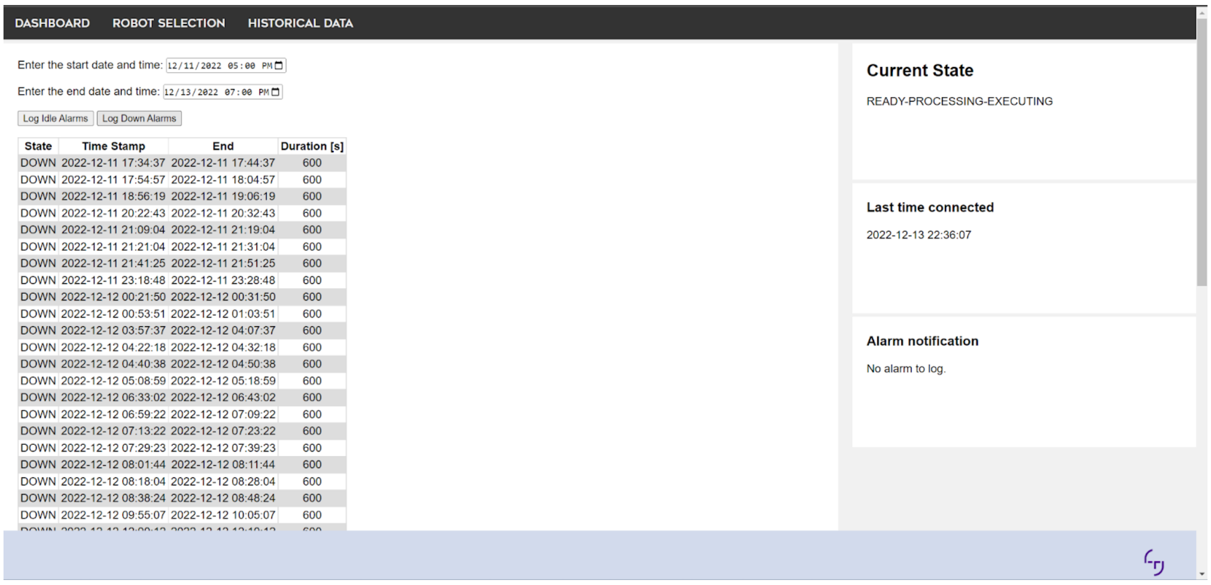
Robot 1 was working 65.54% of the time.

**IDLE**

Robot 1 was idle 14.44% of the time.

Figure 8: Snapshot of the Historical data table shown on the Historical Data page

Alarms and Events generation:
Fullfills the requirement of showing all the events where the robot was either in IDLE or DOWN for an user-given period of time. This page also has a notification tile, where it shows a notification if, the robot is in IDLE or DOWN above the given limit of time, for the user's awareness.



Enter the start date and time: 12/11/2022 05:00 PM

Enter the end date and time: 12/13/2022 07:00 PM

| State | Time Stamp | End | Duration [s] |
|---|---|---|---|
| DOWN | 2022-12-11 17:34:37 | 2022-12-11 17:44:37 | 600 |
| DOWN | 2022-12-11 17:54:57 | 2022-12-11 18:04:57 | 600 |
| DOWN | 2022-12-11 18:56:19 | 2022-12-11 19:06:19 | 600 |
| DOWN | 2022-12-11 20:22:43 | 2022-12-11 20:32:43 | 600 |
| DOWN | 2022-12-11 21:09:04 | 2022-12-11 21:19:04 | 600 |
| DOWN | 2022-12-11 21:21:04 | 2022-12-11 21:31:04 | 600 |
| DOWN | 2022-12-11 21:41:25 | 2022-12-11 21:51:25 | 600 |
| DOWN | 2022-12-11 23:18:48 | 2022-12-11 23:28:48 | 600 |
| DOWN | 2022-12-12 00:21:50 | 2022-12-12 00:31:50 | 600 |
| DOWN | 2022-12-12 00:53:51 | 2022-12-12 01:03:51 | 600 |
| DOWN | 2022-12-12 03:57:37 | 2022-12-12 04:07:37 | 600 |
| DOWN | 2022-12-12 04:22:18 | 2022-12-12 04:32:18 | 600 |
| DOWN | 2022-12-12 04:40:38 | 2022-12-12 04:50:38 | 600 |
| DOWN | 2022-12-12 05:08:59 | 2022-12-12 05:18:59 | 600 |
| DOWN | 2022-12-12 06:33:02 | 2022-12-12 06:43:02 | 600 |
| DOWN | 2022-12-12 06:59:22 | 2022-12-12 07:09:22 | 600 |
| DOWN | 2022-12-12 07:13:22 | 2022-12-12 07:23:22 | 600 |
| DOWN | 2022-12-12 07:29:23 | 2022-12-12 07:39:23 | 600 |
| DOWN | 2022-12-12 08:01:44 | 2022-12-12 08:11:44 | 600 |
| DOWN | 2022-12-12 08:18:04 | 2022-12-12 08:28:04 | 600 |
| DOWN | 2022-12-12 08:38:24 | 2022-12-12 08:48:24 | 600 |
| DOWN | 2022-12-12 09:55:07 | 2022-12-12 10:05:07 | 600 |

**Current State**

READY-PROCESSING-EXECUTING

**Last time connected**

2022-12-13 22:36:07

**Alarm notification**

No alarm to log.

Figure 9: Snapshot of the Historical data table shown on the Historical Data page

**Conclusion:**
Therefore, in conclusion, after many trials and debugging, the team was able to satisfy all the given requirements of the assignment by developing the given system and hosting it on the web. It should be noted that, due to the issues faced when implementing the Database control functions, the team decided to opt away from using classes but rather segregate the functions according to its use as three different files which are connected with each other by importing it when needed. Also since all the desired data is already stored in the database, storing other objects occurs as redundant.

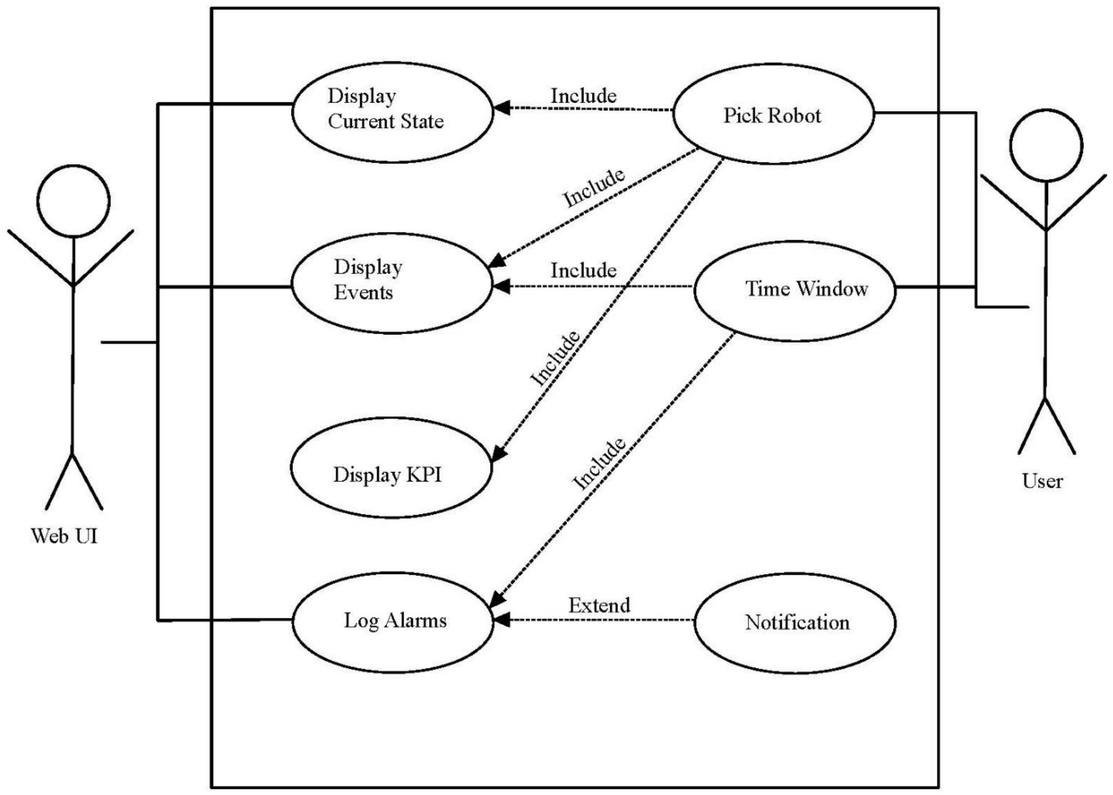


Figure 10: Use Case Diagram to graphical interpret the system

## Appendix

Link: (Azure)
Code: (Github)