University of
South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Development of a predictive maintenance application for packaging machines

USN supervisor: Nils-Olav Skeie

External partner: Goodtech AS

Task background:
Goodtech AS develops machines and lines for handling packaging, for among others the food and pharmaceutical industries. The operation of these packaging machines includes, among others several types of air cylinders and variable-frequency drives (VFDs). These devices are connected to a Programmable Logic Controller (PLC) located in the packing machines. The PLC will record the operations of these devices, including the counting of the stroke of air cylinders. A WizX (IIoT) will be the connection point for the packaging machines and may also have any additional sensors connected. The application will collect this information, store the information in a cloud-based database (intranet or internet) and present maintenance information on a user interface. The application will compare the maintenance information with the specifications and/or requirements, to detect any need for replacement of any devices. The application will detect and inform the user about the status before it needs to be replaced. This will reduce the maintenance cost and risk of unexpected breakdowns. Another benefit of the application is that the process system will change from the need for regular inspections to continuous monitoring.

Task description:
The application has been analysed and a proof of concept (PoC) made from the master project, and the master thesis will be a continuation of this project. The sub tasks for the master thesis will at least be:

- Evaluate solutions for the cloud-based database based on existing vendors. Choose a cloud-based database for your system.
- Give an overview of the software development process at Goodtech AS and compare this process with the Unified Process (UP).
- Develop a data model for the database with functionality for data from both the packing system and additional sensors and include the flexibility to handle several packing systems for each customer, and several customers.
- Discuss the security aspects of this database and suggest any solution for cyber security for the cloud-based database.
- Give an overview of the dataflow in the system with focus on the information needed for the predicted maintenance for the packing machines.
- Design and develop the application using an object-oriented design approach, collecting the information from different WizX systems, store the information in the cloud-based database, and present important maintenance information on the GUI.
- Give an overview of machine learning methods, with focus on neural networks (AN), that can be used for predictive maintenance based on time series data.
- Make a test plan that will be used for testing the application.

Student category: Available for Industry master students at Goodtech AS only (IIA student)

Is the task suitable for online students (not present at the campus)? No

Practical arrangements:
Packing machines with the WizX system will be available at Goodtech AS for testing.

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Signatures:

Supervisor (date and signature): Nhi-Dan Lee 27-JAN-22

Student (write clearly in all capitalized letters):

Student (date and signature): Peshawa J. Saleh 22.Feb-22

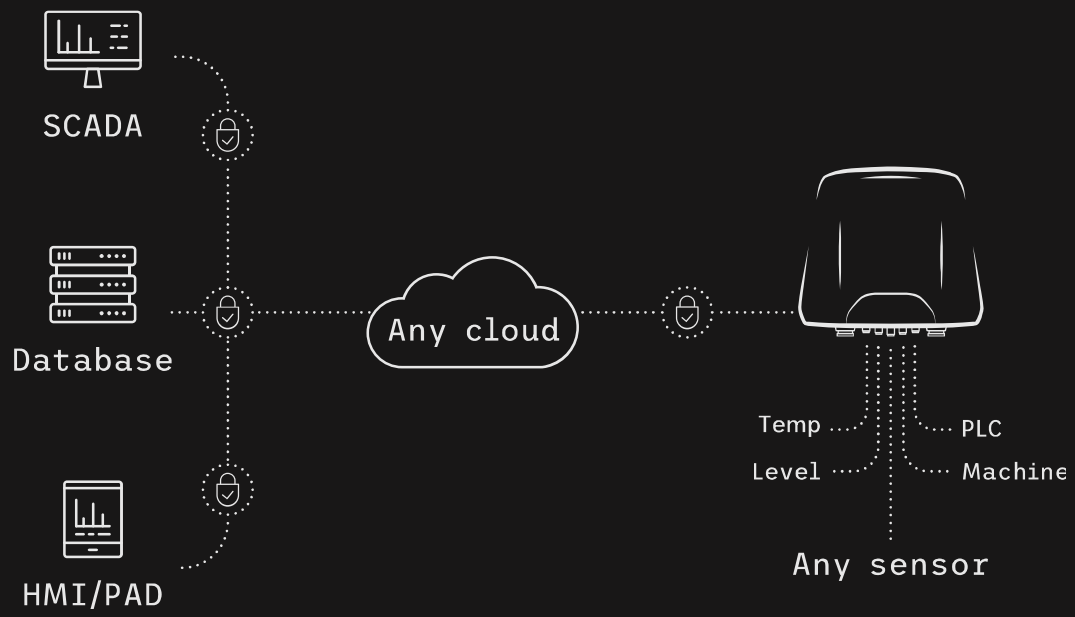
WIZX

THE ULTIMATE IIOT GATEWAY

goodtech

Product: low cost IIoT module

- Measuring, logging and presentation of data
- Minimum Installation Cost
- Easy to configure
- Support large amount of sensor types and communication protocols
- Based on Open Source Software
- Secure Communication Protocols



Product: low cost IIoT module

Sensor Interface support

- ✓ I2C
- ✓ One Wire
- ✓ SPI
- ✓ 0-10V
- ✓ 4-20mA

Visualization and Integration

- ✓ SCADA system
- ✓ IMS/MES systems
- ✓ ERP systems

Communications

- ✓ Wifi
- ✓ Bluetooth Low Energy
- ✓ RF 868/434 MHz
- ✓ 4G
- ✓ Modbus
- ✓ Canbus
- ✓ NB-IOT
- ✓ LoRa
- ✓ OPC UA
- ✓ ProfiBus
- ✓ ProfiNet

Appendix C: This appendix contains SQL syntax used in the local database and SQL script for generating database tables used in Microsoft Azure for MySQL Server.

SQL syntax is used in the local database

The following SQL syntax is used to create a hypertable for the taghistory_data table and column t_stamp in order to make storing and querying time-series data more efficient and efficient:

```
SELECT create_hypertable('taghistory_data', 't_stamp');
```

The taghistory_data table stores data only for one week. This is done by creating a data retention policy for data that is older than one week:

```
SELECT add_retention_policy('taghistory_data', INTERVAL '1 week');
```

Database tables used in Microsoft Azure for MySQL Server

The SQL script is generated using MySQL Workbench, which gives an overview of the different tables with columns and data types used in the Application for storing data in Microsoft Azure for MySQL Server.

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema packagingMachineApp
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `packagingMachineApp` DEFAULT CHARACTER SET utf8 ;
USE `packagingMachineApp` ;

-- -----------------------------------------------------
-- Table `packagingMachineApp`.`packaging_machine`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `packagingMachineApp`.`packaging_machine` (
  `packagingMachine_Id` INT NOT NULL AUTO_INCREMENT,
  `packagingMachine_Tag` VARCHAR(45) NOT NULL,
  `description` VARCHAR(45) NULL,
  UNIQUE INDEX `packagingMachine_Tag_UNIQUE` (`packagingMachine_Tag` ASC) VISIBLE,
  PRIMARY KEY (`packagingMachine_Id`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `packagingMachineApp`.`cylinders`
```

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `packagingMachineApp`.`cylinders` (
  `cylinder_Id` INT NOT NULL AUTO_INCREMENT,
  `cylinder_Tag` VARCHAR(45) NOT NULL,
  `Unit` VARCHAR(45) NULL,
  `description` VARCHAR(45) NULL,
  `packagingMachine_Id` INT NOT NULL,
  UNIQUE INDEX `cylinder_Tag_UNIQUE` (`cylinder_Tag` ASC) VISIBLE,
  PRIMARY KEY (`cylinder_Id`),
  INDEX `fk_cylinders_packaging_machine_idx` (`packagingMachine_Id` ASC) VISIBLE,
  CONSTRAINT `fk_cylinders_packaging_machine`
    FOREIGN KEY (`packagingMachine_Id`)
    REFERENCES `packagingMachineApp`.`packaging_machine` (`packagingMachine_Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `packagingMachineApp`.`vfds`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `packagingMachineApp`.`vfds` (
  `vfd_Id` INT NOT NULL AUTO_INCREMENT,
  `vfd_Tag` VARCHAR(45) NOT NULL,
  `Unit` VARCHAR(45) NULL,
  `description` VARCHAR(45) NULL,
  `packagingMachine_Id` INT NOT NULL,
  UNIQUE INDEX `vfd_Tag_UNIQUE` (`vfd_Tag` ASC) VISIBLE,
  PRIMARY KEY (`vfd_Id`),
  INDEX `fk_vfds_packaging_machine1_idx` (`packagingMachine_Id` ASC) VISIBLE,
  CONSTRAINT `fk_vfds_packaging_machine1`
    FOREIGN KEY (`packagingMachine_Id`)
    REFERENCES `packagingMachineApp`.`packaging_machine` (`packagingMachine_Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `packagingMachineApp`.`sensors`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `packagingMachineApp`.`sensors` (
  `sensor_Id` INT NOT NULL AUTO_INCREMENT,
  `sensor_Tag` VARCHAR(45) NOT NULL,
  `Unit` VARCHAR(45) NULL,
  `description` VARCHAR(45) NULL,
  `packagingMachine_Id` INT NOT NULL,
  UNIQUE INDEX `sensor_Tag_UNIQUE` (`sensor_Tag` ASC) VISIBLE,
```

```
  PRIMARY KEY (`sensor_Id`),
  INDEX `fk_sensors_packaging_machine1_idx` (`packagingMachine_Id` ASC) VISIBLE,
  CONSTRAINT `fk_sensors_packaging_machine1`
    FOREIGN KEY (`packagingMachine_Id`)
    REFERENCES `packagingMachineApp`.`packaging_machine` (`packagingMachine_Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `packagingMachineApp`.`log_data`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `packagingMachineApp`.`log_data` (
  `logData_Id` INT NOT NULL AUTO_INCREMENT,
  `cylinder_Id` INT NOT NULL,
  `cylinder_value` FLOAT NOT NULL,
  `vfd_Id` INT NOT NULL,
  `vfd_value` FLOAT NOT NULL,
  `sensor_Id` INT NOT NULL,
  `sensor_value` FLOAT NOT NULL,
  `timeStamp` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`logData_Id`),
  INDEX `fk_log_data_cylinders1_idx` (`cylinder_Id` ASC) VISIBLE,
  INDEX `fk_log_data_sensors1_idx` (`sensor_Id` ASC) VISIBLE,
  INDEX `fk_log_data_vfds1_idx` (`vfd_Id` ASC) VISIBLE,
  CONSTRAINT `fk_log_data_cylinders1`
    FOREIGN KEY (`cylinder_Id`)
    REFERENCES `packagingMachineApp`.`cylinders` (`cylinder_Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_log_data_sensors1`
    FOREIGN KEY (`sensor_Id`)
    REFERENCES `packagingMachineApp`.`sensors` (`sensor_Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_log_data_vfds1`
    FOREIGN KEY (`vfd_Id`)
    REFERENCES `packagingMachineApp`.`vfds` (`vfd_Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `packagingMachineApp`.`userdatabase`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `packagingMachineApp`.`userdatabase` (
```

```
  `id` INT NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(250) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Appendix D: Application Packaging Machine Sequences Diagrams.

Collect and Store Data Sequence diagram

Figure 1 illustrates the SD diagram for the Collect & Store Data use case. The startup sequence is outside the Application PM, as it is assumed that external modules, The OPC UA server, the MQTT broker, and the Azure cloud exist and run in Docker Containers when the application is running.

After the powring up of the system, the Docker Containers will start, and the Collect & Store Data will start reading data from the PLC and the air pressure sensor every 60 seconds and defines them with the tag name and data type. The collected data are checked for if the value changes, and then they are stored as a global variable using *global.Set()* function. If not, the operation of storing data is stopped. Using an MQTT Client, the collected data (metrics) are formatted and sent with a defined topic namespace (Edge Nodes/WizX/Wizx01/PackgingMchine_id/device_id) to the MQTT Broker within the same loop. The OPC UA Server integrated with the Sparkplug interface subscribes to the same topic in the MQTT Broker, and then the Broker publishes the topic to the OPC UA Server. This operation is executed as an external module of the Collect & Store Data program, which is why it is not illustrated in the SD. Within the loop, there is a 7-second delay. This is to make sure that all the updated metrics are within the OPC UA Server. An MQTT Client uses the browser function, browses the OPC UA Server for all updated metrics, and stores them in the taghistory_te table using *wizxCore Config updater Node* within Timescale DB. A new loop for storing data in the Azure cloud DB starts by first checking the connection with the Azure cloud DB. If the Azure Cloud (MySQL Server) connection is false, the program displayed Azure Cloud DB disconnected on the Node-RED UI. While if the connection is true, the program will retrieve the stored and updated data as a global variable using *global.Set() function* and store data associated with the timestamp in the cloud DB. The sampling time for this loop is 60 seconds.

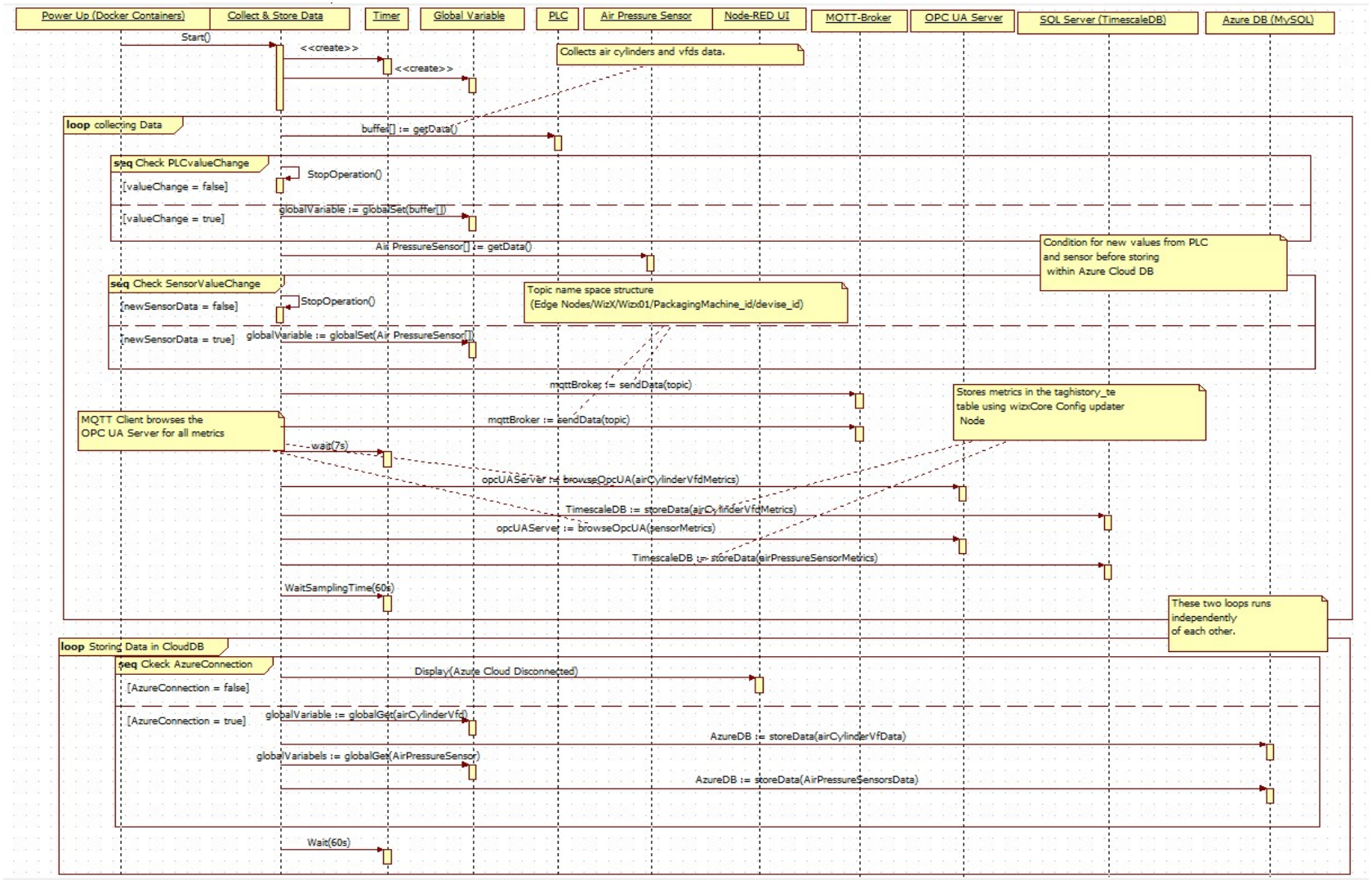


Figure 1: SD for Collect and Store data use case.

Handle Alerts

As stated in the analysis phase, handling alerts are done in the Grafana. Thus creating alerts and notifications channels are defined and configured in Grafana. The SD diagram for the Handle Alerts use case presented in Figure 2 illustrates the program for handling alerts in Grafana. The admin user has the administration role in the Grafana that creates and manages the alerts in the application. Alert states are presented in the Alert list and Graph panel. The program defines and configures alert rules and notifications for each air cylinder, VFD, and air pressure sensor in the graph panel. The alert rules are then stored in the Rule Engine. After configuring the alert rules and email notification channel, the program then gets real-time series data (LastData) in a loop every minute. The program is then getting the configured alerting rules and a notification channel. The program evaluates the rules, in this case, every 20s for executing the alert rules and conditions. If the LastData is less than the threshold value for 10s, the retrieved data state will be set to the 'Ok' alert state in the Alert list and Graph panel. While if the retrieved data is above the threshold value for 10s, then the state will be set to a 'Pending' alert state in the Alert list. If the value of the lastData is still above its threshold for the 20s, the state will change to the 'Alerting' alert state in the Alert list Graph panel. The program sends a notification alert via email. In addition, if lastData has the value of null or if an error occurs in the evaluation rules, the lastData state to set to the 'Alerting' alert state in the Alert list.

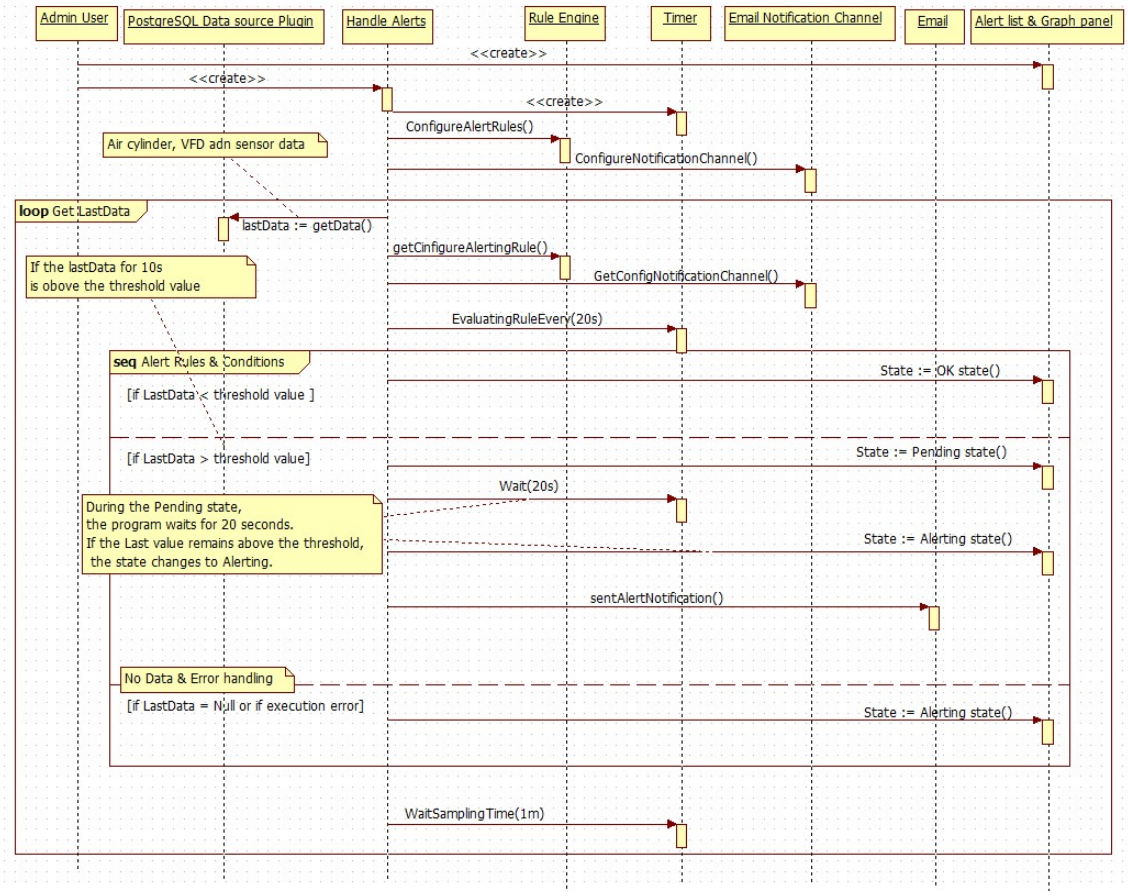


Figure 2: Handle Alerts Sequence Diagram (SD).

Maintain Configuration of Azure Cloud database server

The maintain configuration use case aims to maintain the configuration of the dashboards in Grafana and the cloud-based database in Microsoft Azure Server for MySQL DB. The SD for Maintain Configuration use case is divided into two SDs; Maintain Configuration for cloud-based DB and Maintain Configuration within Grafana.

The configuration of the cloud-based database is performed through the Node-RED dashboard included in the Azure CloudDB flow. As illustrated in Figure 3, the SD of Maintain Configuration for a cloud-based database is shown. Upon powering up the Docker Containers, the maintain configuration program included in the Node-RED container runs the Azure Cloud DB and Dashboard flow. The dashboard can be accessed using the same URL as the Node-RED administrator but with the addition of /ui. After entering the path of the url in a web browser, the Node-RED UI interface appears on the Registration page. For any configuration within the cloud database, a valid user name and password must be entered. A user database table is included in the MySQL cloud database, where the user name and hashed password are stored. The user database table contains admin and operator users, along with their hashed passwords. After a user name and password (userInput) have been entered, the program is able to verify them in the user database table with the registered user. In the event that the username or password is incorrect, a message will appear on the status page of the registration page indicating "Error! Username or password is incorrect". In the event that only the user name is correct, but the password is incorrect, the program displays "Please try again!". Once both username and password are correct, the program displays the status as "Login successful". Users with administrator privileges have full control over the user interface and are able to modify the database by creating, updating, or deleting packing machines, air cylinders, VFDs, and user tables, whereas an operator user is only permitted to modify the packaging machine, air cylinder, VFD, and sensor tables. In order to register a new user, the password must be repeated twice. If the repeated password is not identical, the program displays the status "Password does not match!". The new user is provided with UI control by updating the UI user control flow in the program. When updating a user's password both the old and new passwords must be entered twice. In case the old password is incorrect, the program displays "incorrect old password!" In case the repeated password is incorrect, the program displays "new password does not match".

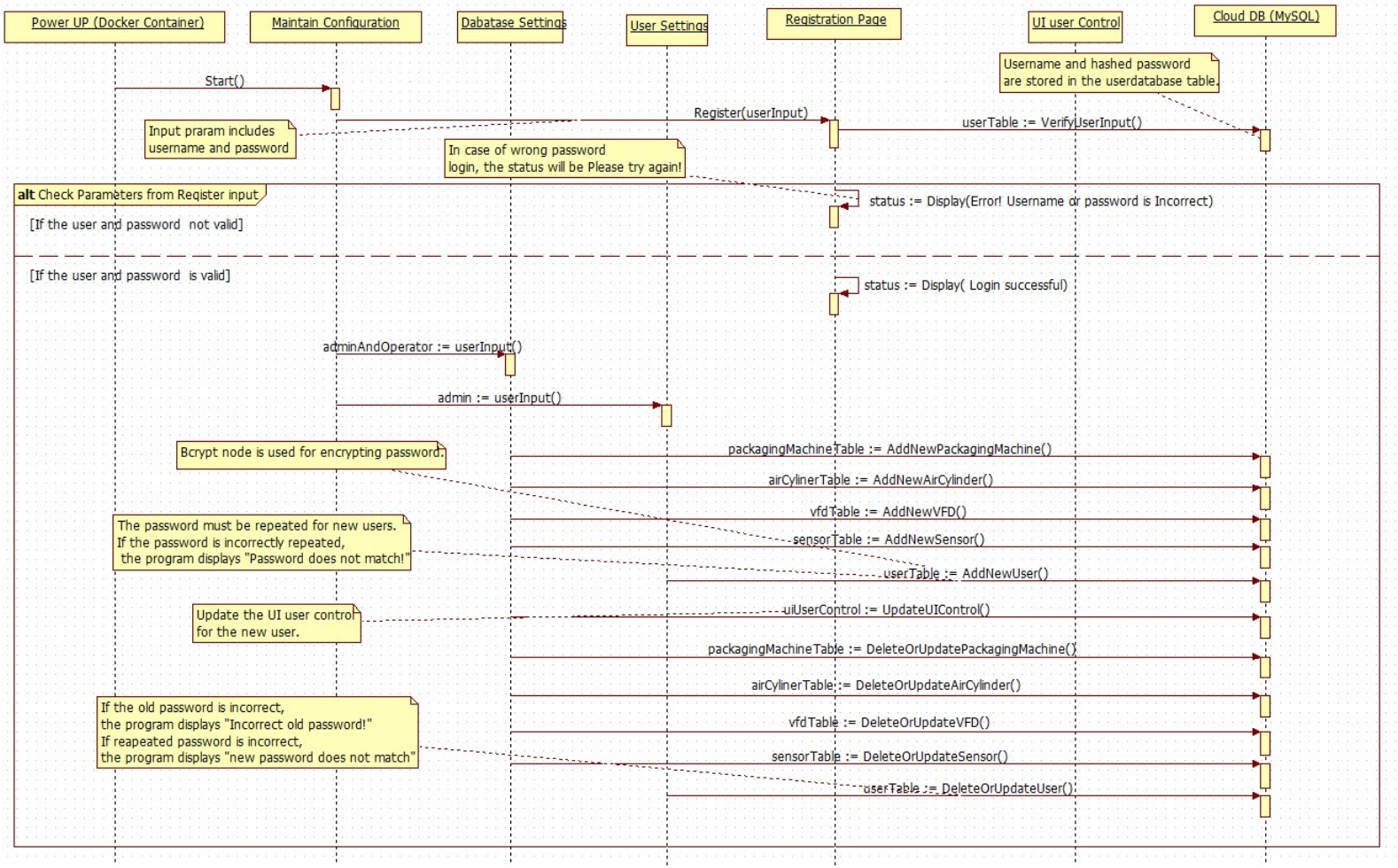


Figure 3: SD for Maintain Configuratin from Node-RED UI.

Maintain Configuration within Garfana

Figure 4 illustrates the SD for Maintaining Configuration within the Grafana. When the Docker Container gets powered up, the Grafana container runs. Grafana can be accessed using any web browser as a client using the IP address followed by port 3000. The log-in page will appear, where a user name and a password must be typed in. If the user name or password is not correct, the registration page displays an incorrect password or user name, while if the correct user name and password are typed in the Home dashboard, it will appear. Currently, two users are registered; admin, operator, and guest. Both the admin has the admin role, and the operator has the editor role, while the guest user has only the viewer role. When it comes to the application configuration within the Grafana, the admin user can configure both users in the Admin Server and the rest of the application dashboards, while the operator can only configure the content within the dashboards. Thus, if the user name is an admin user, it can create, update and delete a user. If the user input is either admin or operator, the application dashboard content such as adding, updating, and deleting a dashboard, air cylinder, VFD, or pressure sensor. In addition, the sampling time range and auto-refresh screen time for the dashboard and panel can be set as default for all dashboards and panel

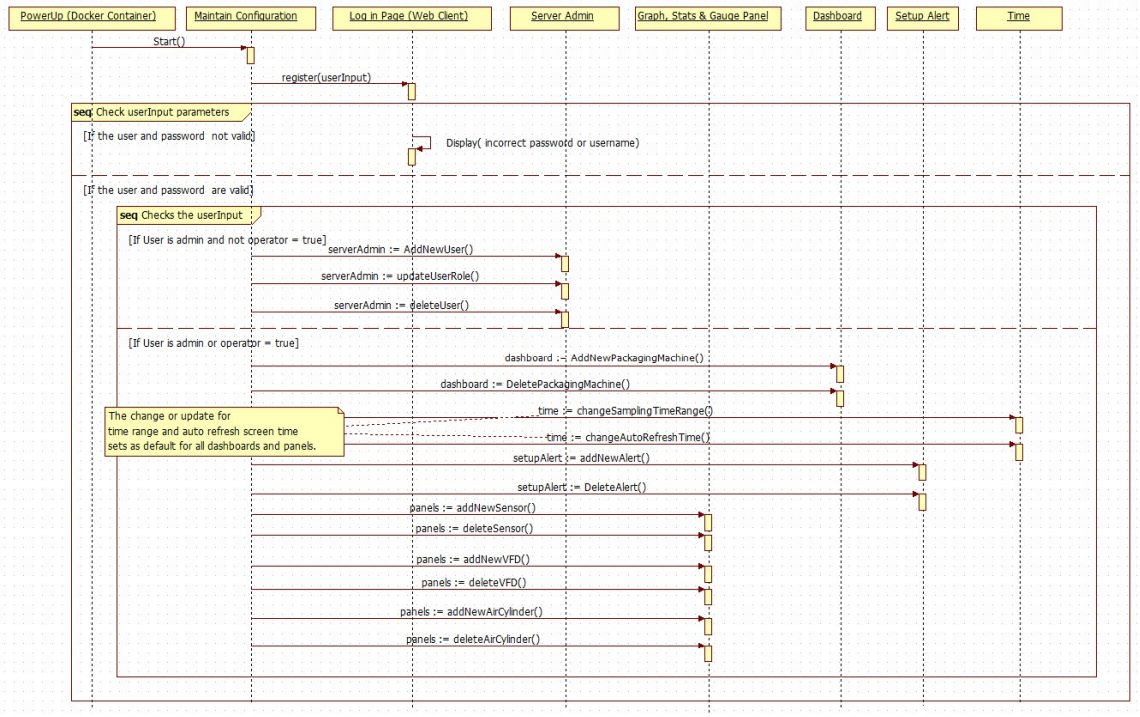


Figure 4: SD for Maintain Configuratin from Grafana dashboards.

Appendix E: Packaging Machine dashboard screenshots

Login page

Once the Docker Container that includes the Grafana server is up and running, one can navigate the Grafana application dashboards using the IP address to the Raspberry Pi followed by the 3000 port number using any web browser. Figure 1 illustrates the login page in Grafana using the Chrome web browser.

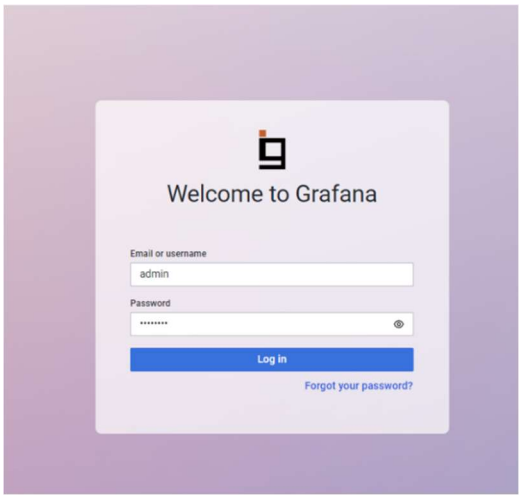


Figure 1: Login to Grafana Packaging Machine Application dashboards.

Dashboards

The packaging machine application consists of three dashboards; Home Dashboard, Packaging Machine 01 Dashboard (Test Celle), and Packaging Machine 02 Dashboard (Simulation Packaging Machine).

Home Dashboard

Once the user logs in successfully, the Home dashboard will appear, as Figure 2 illustrates the Home dashboard. The Home dashboard aims to present and visualize information regarding alerts from the packaging machine dashboards.

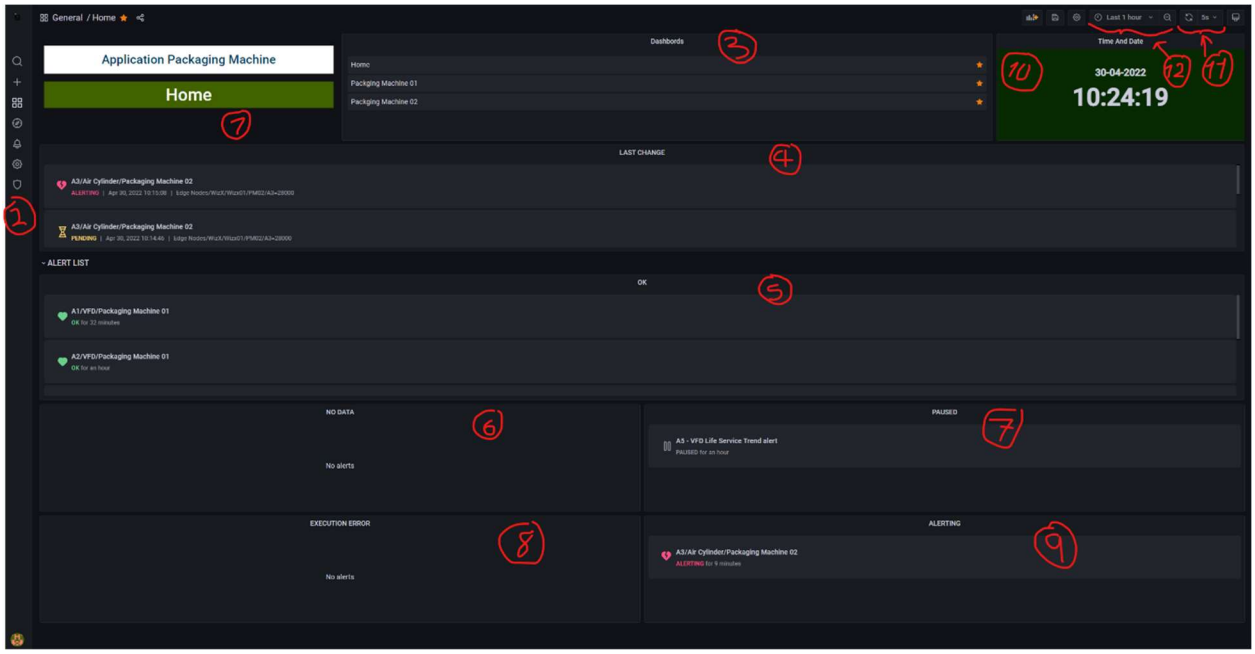


Figure 2: Home Dashboard.

The dashboard has the following section for presenting information regarding alerts:

- Displays the application the dashboard's name in the text panel (1).
- The Grafana menu bar (2).
- Dashboard list (3).
- The last change table panelist. It shows the last change on the dashboards' alerts listed in descending order by timestamp. As it can be seen in the figure, at this moment, A3/Air Cylinder/Packaging machine 02 is in the alerting state with a 'breaking heart' Symbole, which is followed by the pending state of the same cylinder (4).
- Display the 'Ok' alert list panel in descending order by timestamp that is symboled with a green heart symbol (5).
- Present No data list panel. In case of some reason, no data are retrieved (6).
- Paused panel list shows the alerts that are paused. This functionality is useful, especially during maintenance, where an alarm can be paused after maintenance (7).
- Executing error panelists present information during the rule engine evaluation error or timeout errors (8).
- Alerting panelists will contain air cylinders, VFDs, or air pressure sensors in alerting state (9).

- Time and data (10).
- Screen refresh time (11)
- Time picker dropdown (12)

Packaging Machine 01 (Test Celle) Dashboard

Figure 3 shows the screenshot of the dashboard taken under testing the application in the Goodtech department Moss. It displays the real-time data from pneumatic air cylinders, VFDs, and simulated air pressure sensors that simulates the pressure level for central compressed air within the Test Celle machine.

Figure 3: Screenshot of the Packaging Machine 01 (Test Celle) taken under the testing of the applciation in Moss.

Figure 4 shows the actual photo of the machine taken during the testing of the application.

Figure 4: Test Celle machine.

Query real-time data

Once the data source for PostgreSQL is added to the dashboard, the real-time series data must be fetched from the TimescaleDB database tables and visualized in panels. This is done using For SQL syntax query as shown in Figure 5 shows the edit panel window for air cylinder A508. At the bottom of the figure is where the SQL query is written for querying data from the PostgreSQL data source that queries the metrics from the taghistory_te and taghistory_data within the theTimescaleDB. The air cylinders' data type is defined as a real data type from PLC. Thus, the float data type is used within the application, and the approximately operating life service for the air cylinder is set to be 10,000 km stroke distance. To visualize the value to in percentage, the value is divided by 0,10, as can be seen in the query syntax. The tag path *Edge Nodes/Wizx/Wizx01/PM02/A508* for the air cylinder is used that fetch metrics from the database.

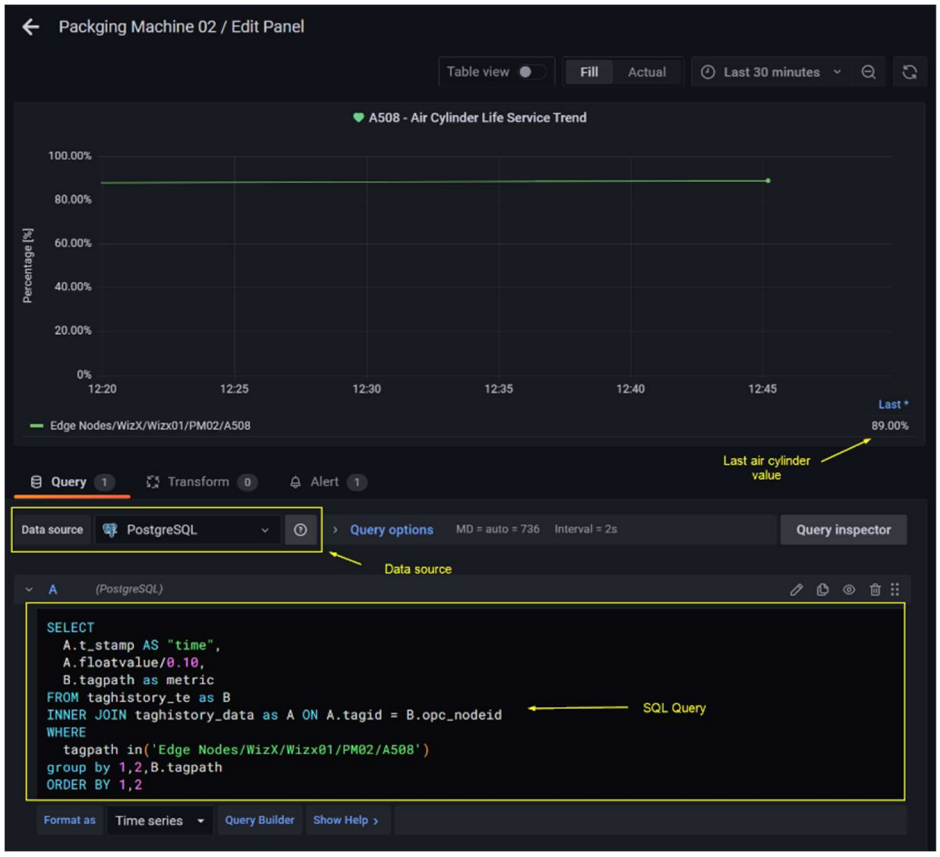


Figure 5: Graph panel for A508 air cylinder shows the query tab window that fetchs real-time data from data source.

Alerting Configuration on Graph panels

Figure 4 shows the alert configuration for the A508 air cylinder on the packaging machine 02. In the Alert tab (1) of the graph panel for A508, alerts are configured by first giving a name to the alert rule (Rule Engine) *Name* field (2) displayed in the Home dashboard. The second field (3) in the alert rule is *Evaluate every*, which in this case is every 20s. This means that the Rule Engine should evaluate the alert rule every 20s. The following configuration field that must be defined is *For* (4). This means for how long the Rule

Engine should evaluate the rule for the query, which in this case is set to be 20sound. The following configuration part that must be set up is the conditions of the alert (5). This condition specifies the threshold for an alert. The threshold in the case of the A508 cylinder is the following; When the last value of the query data from the cylinder is above the threshold (95 %) of its operating service life for 10 seconds for the first time, then the alert state changes from the 'OK' to 'Pending' alert state in the Rule Engine. Suppose the value is still above the threshold for 20 sounds. In that case, the alert changes to the 'Alerting' state and triggers its notifications to the graph panel, alert list in the Home Dashboard, and the email notification channel; Alerts from Grafana (8).

Furthermore, if there is a scenario where the query in the graph panel gets a NULL value or no data value from the data source (PostgreSQL), an alert will arise on the graph panel and the Home dashboard. Likewise, if an execution error or timeout occurs during the Rule Engine evaluation of the alert, an alert will be generated (6) (7). The orange dashed vertical line on the graph panel illustrates the 'Pending' alert state (9), while the red line presents the 'Alerting' alert state (10).

Figure 6: Alert configuration window for A508 air cylinder.

Figure 7 shows the configuration of notification channels implemented for the application.

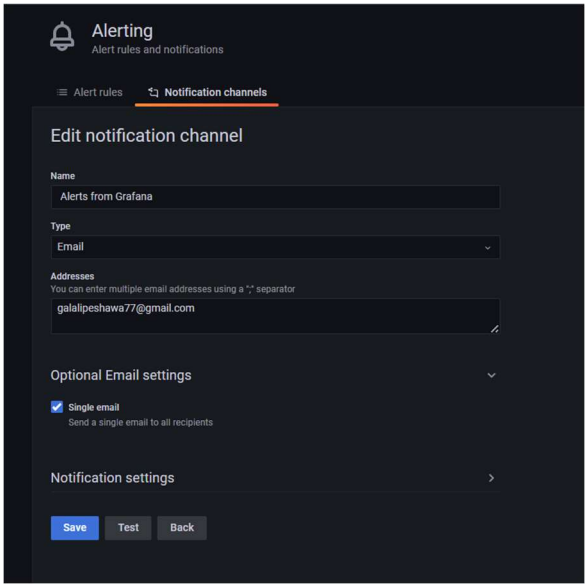


Figure 7: Configuration of Email Notification channel.

Configuration of Organization and

User Authorization within dashboards

Figure 8 shows the Server Admin page, showing the created organization groups. The main organization is owned by Goodtech, while the customer owns the Customer organization.

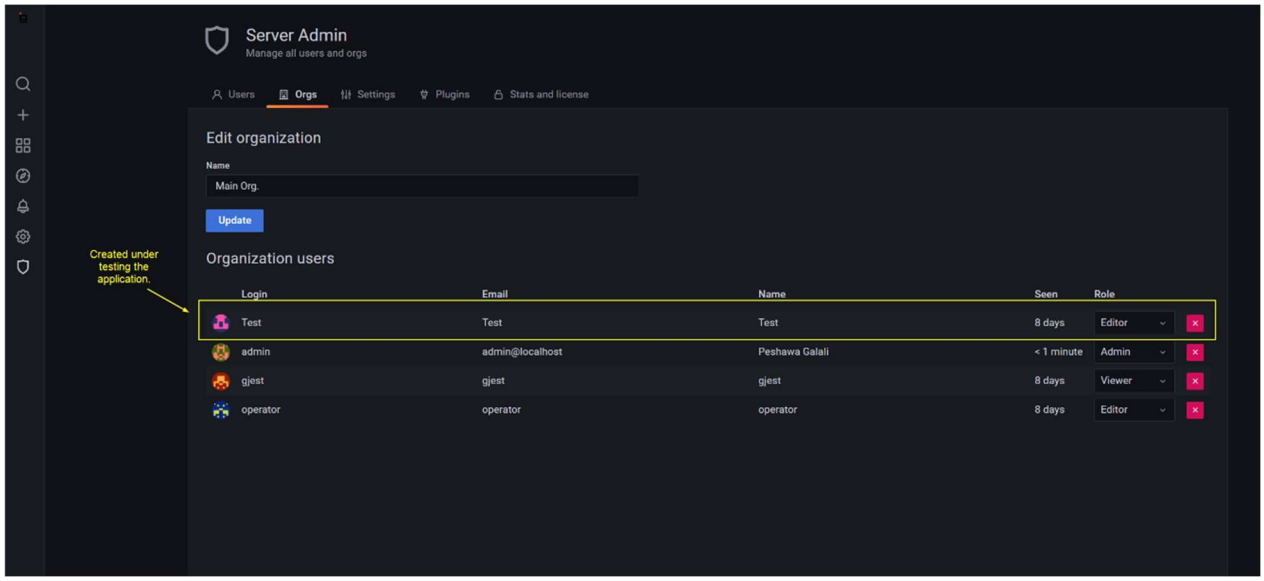


Figure 8: Screenshot of the Server Admin page in Grafana shows the Mian Org. with created users (gjest, operator and admin) and their roles. The test user was created durring the testing in Moss.

Figure 9 illustrates the main organization (Main Org.) and customer for packaging machine applications.

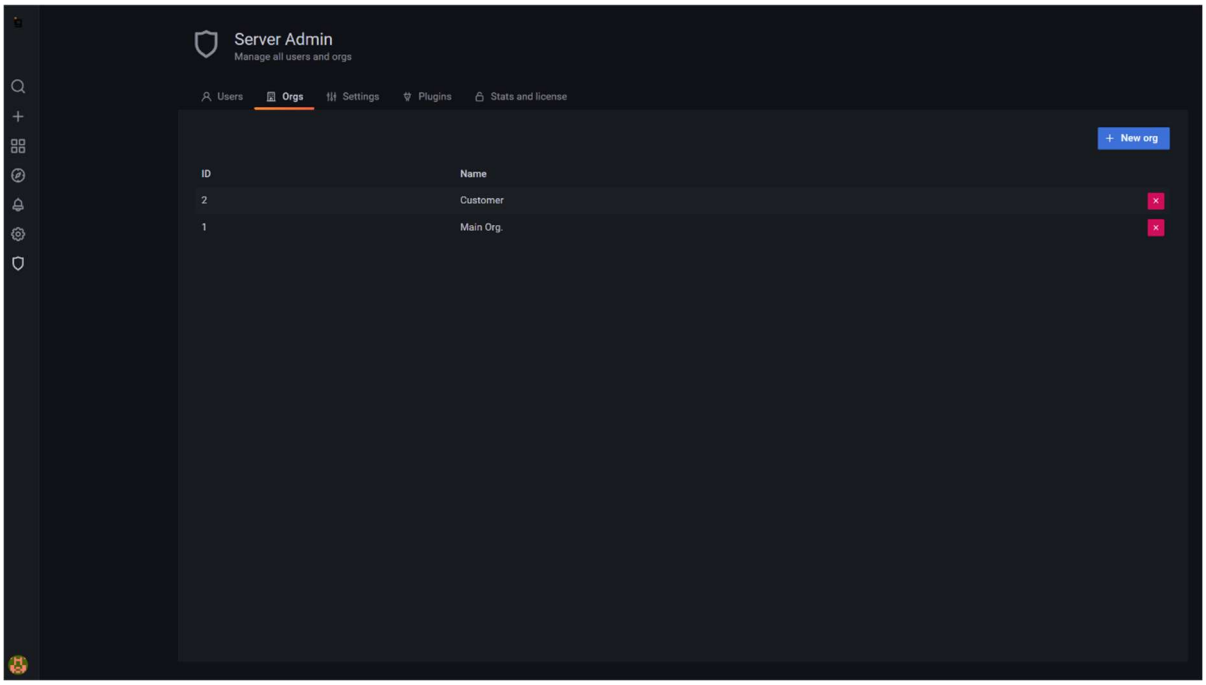


Figure 9: Main org. and Custumer organization created users and their roles.

Appendix F: Node-RED flows.

In Data flow implementation

Figure 1 shows the nodes used in a simulation of a packaging machine within In Data flow, demonstrating nodes for a pressure sensor with the tag name P02, two VFDs with the tag names A3 and A4, and two air cylinders with the tags A506 and A508, which use inject nodes to define tag names and trigger data every minute. A random node simulates the pressure sensor, generating a value between 70.1 and 80.2 when triggered by the trigger node. VFD inject nodes generate integer values from 0 to 30000 and 40000, which simulate the running hours for VFDs. For cylinder simulation, the inject nodes (A506 and A508) simulate float values between 0 and 10 km. This means that those two cylinders have a service life of 10 km. The random node and inject node are connected to the sub-flow PM02 (Packaging Machine 02), where the devices (P02, A3, A4, A506, and A508) are prepared and formatted before being sent to the MQTT broker.

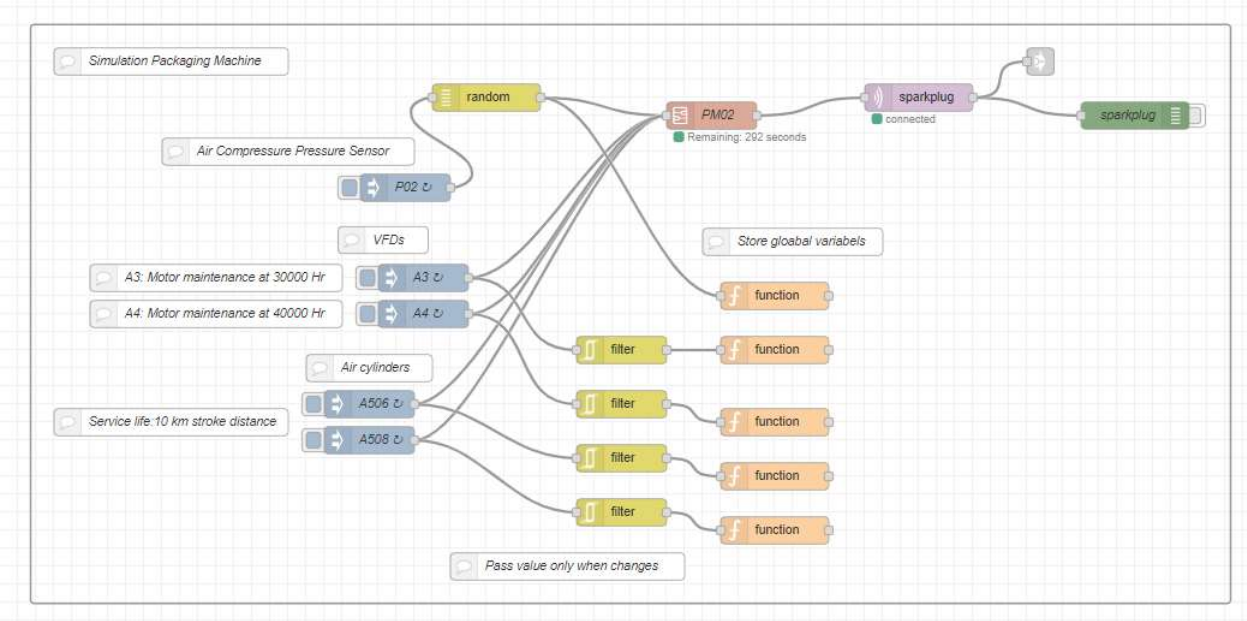


Figure 1: Simulatiuon Packaging machine 02 flow.

Sub-flow is then connected to the Sparkplug node using the MQTT client for the purposes of sending metrics from devices associated with the timestamp of the MQTT broker. The link node transfers metrics to the (admin) Wizxcore flow.

Data from the PLC and air pressure sensor simulator are simultaneously filtered, with changing data passing only if it changes, and stored as global variables using function nodes. At this point, the data is ready to be retrieved by Azure DB flow to be stored within the cloud database.

Figure 2 shows the implemented nodes for collecting data from the PLC and simulated air pressure sensor. For reading data from PLC, the S7-comm node is used. The first step is to define the communication between the PLC and Node-RED (S7-comm node). The S7-comm node uses the RFC 1006 communication protocol to read and write data from the PLC. For this application, only the read node is used. To establish the connection between the S7-comm node and the PLC, the IP address of the PLC and default port, which is 102, must be typed in during the configuration of the S7-comm node. The next step is to define the tag from the PLC (Test Celle machine) that must be read. The tag name for cylinders are A502 and A504 with

float data types, and the tag names for VFDs are A1 and A2 with integers as the data types. The S7-comm is connected to the switch node to switch the metric of the PLC to payload signal into the sub-flow (PL01). Within the sub-flow, they are prepared and formatted before being sent to the MQTT broker.

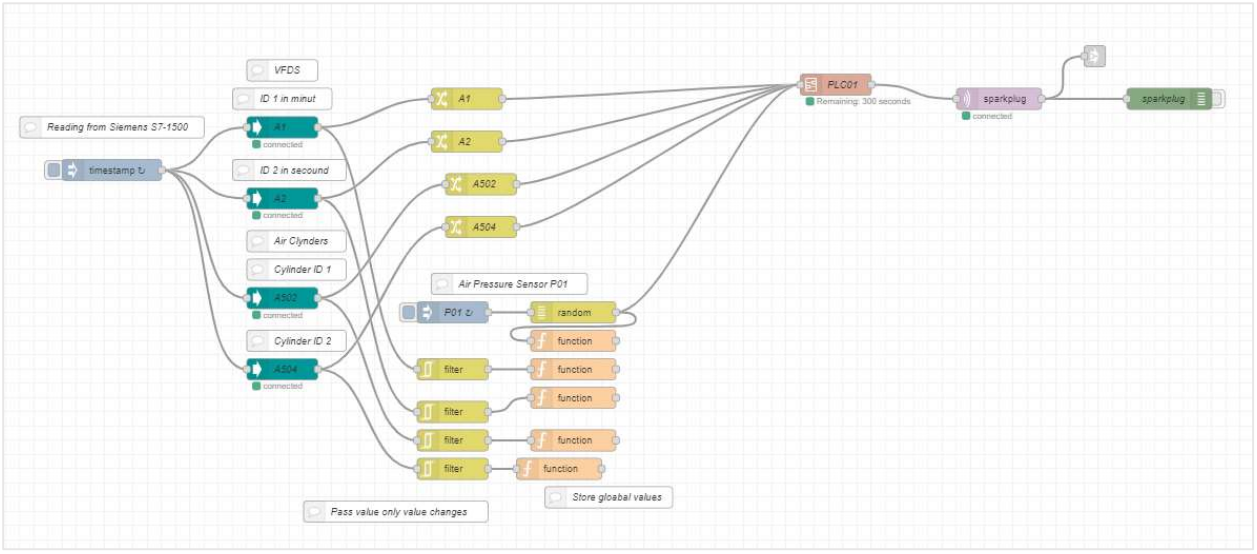


Figure 2: Node-RED flow for collecting data from the PLC, air pressure sensor simulator and sends to MQTT Broker and at the same time stored a global variables.

At the same time, data from the PLC and air pressure sensor simulator are filtered with passing data only if it changes and stored as global variables using function nodes. At this point are prepared to be retrieved by Azure DB flow to be stored within the cloud database.

(admin) wizxcore flow

Figure 3 shows the three separated flows within the (admin) wizxcore flow. In the first flow, where the OPC UA server asks for rebirth from all device metrics, and there is 7s a delay, the OPC UA server gets all the device metrics. Once the OPC UA server has all the device metrics, in the second flow, the OPC UA client scan the OPC UA server, and the wizxCore-Config-updater node updates the local database (TimescaleDB). The third flow deletes all subscriptions for metrics that the OPC UA client has subscribed to before and commands to subscribe to metrics that are history enabled. And then injects the database with metrics history enabled using the WizxCore-inserter node.

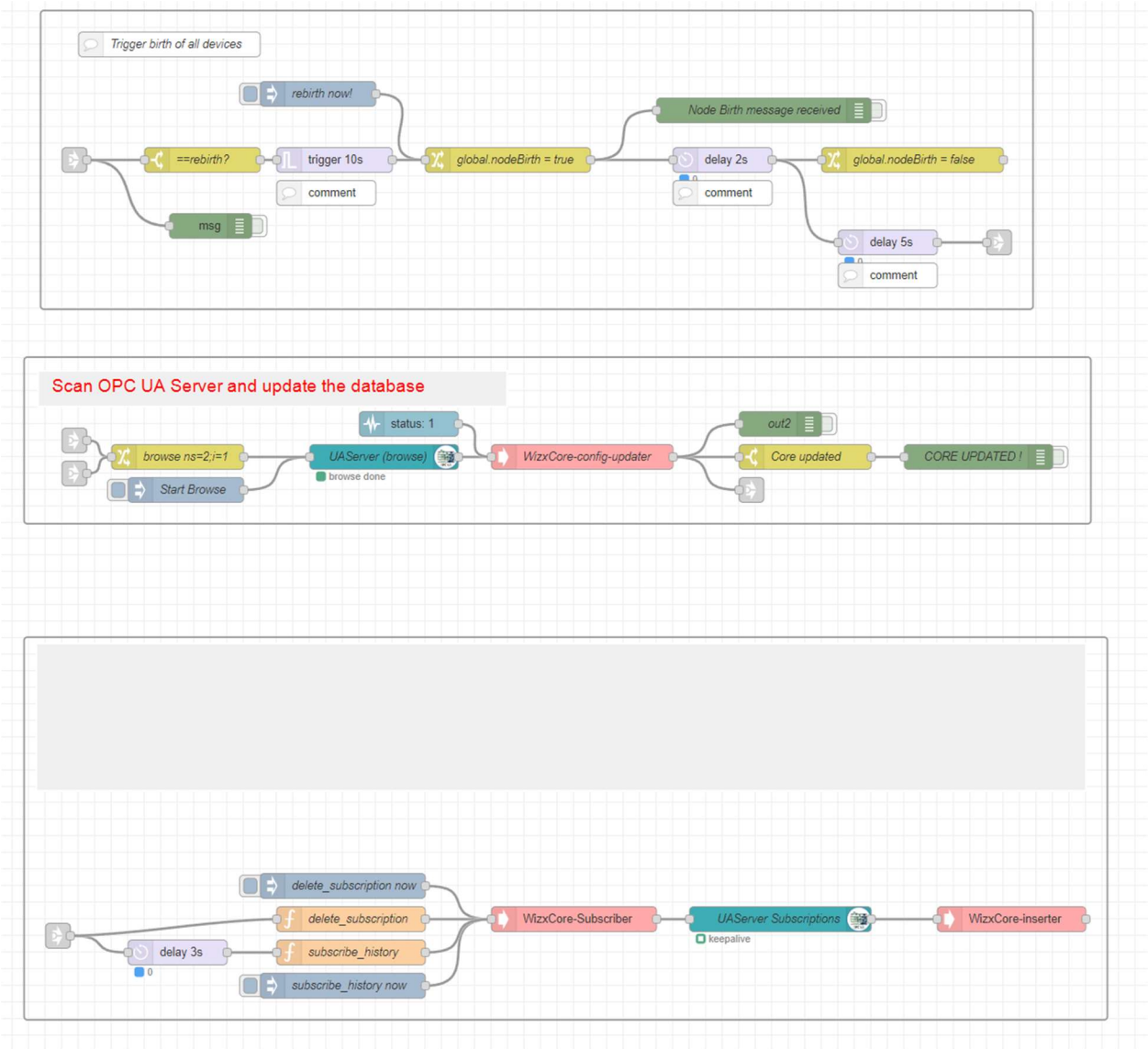


Figure 3: admin wizxcore flow.

Azure DB flow/Node-RED Dashboard

The Azure DB Flow/Node-RED dashboard is divided into the following sub-flows different subflow.

Figure 4 illustrates the flow that checks the connection with the Microsoft Azure cloud every minute. If the database server is not running, a dialog on the screen will pop up with information, and, at the same time, the color of the led will change from green to red. I

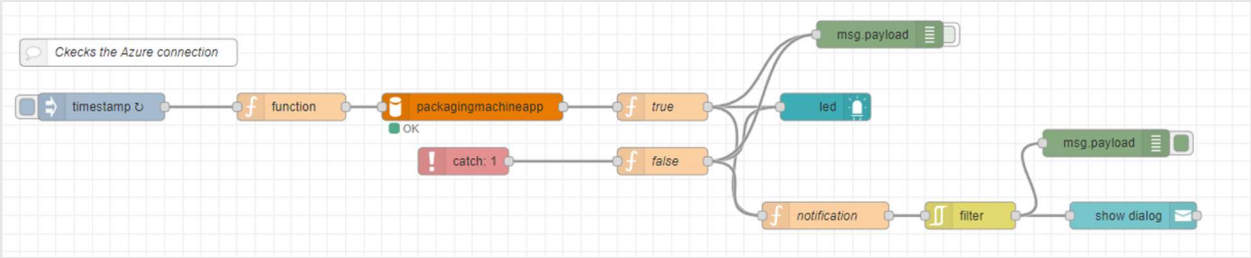


Figure 4: Flow that checks the connection with the database server in Azure.

Store data in the Azure for MySQL DB

Figure 5 illustrates the flow of writing data to logdata tables within the packaging machine cloud database. The inject node (sampling time) is used for triggering the function node every minute. Within the function node, javaScript programming code is used for retrieving the global variables using global.Get() function. Data are then stored in the log data table using SQL syntax script. The function node is connected to the MySQL node (packgingmachineapp) where the connection with the database in the Azure cloud is created.

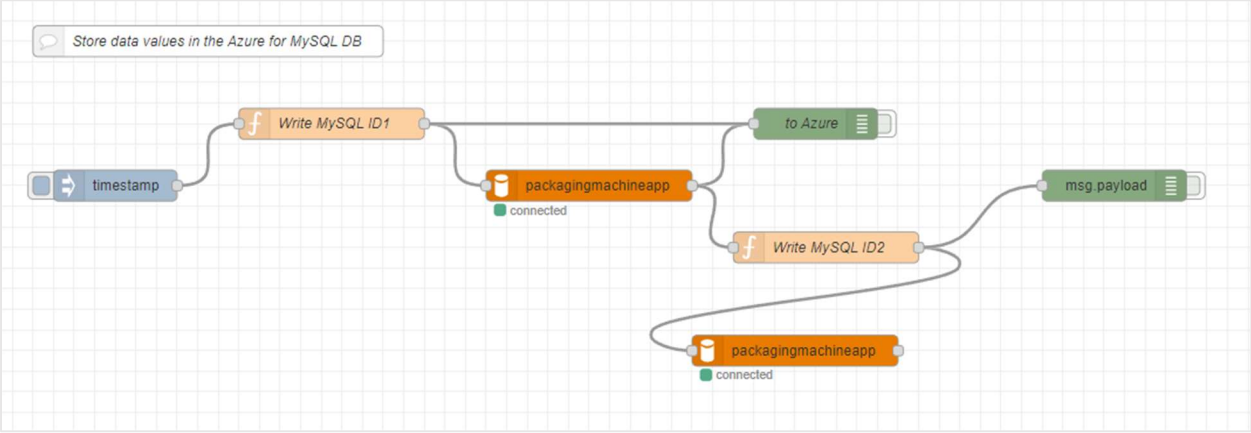


Figure 5: : Storing data in the cloud-based database for the application.

Node-RED dashboard flows

Figure 6 shows the flow for registering a user.

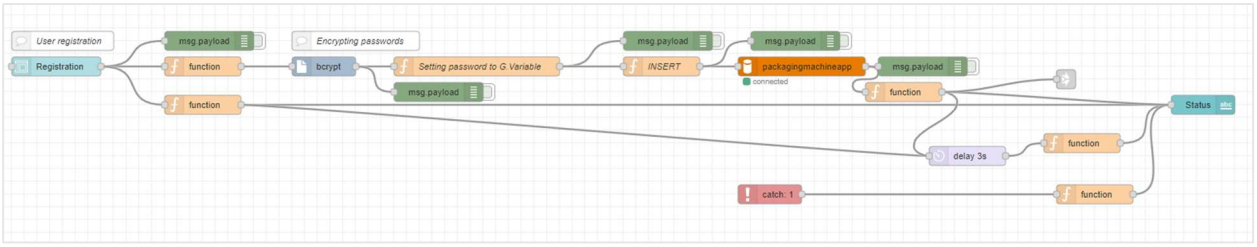


Figure 6: User registreation flow..

Figure 7 shows the flow that verifies the user name and hashed password with the database table.

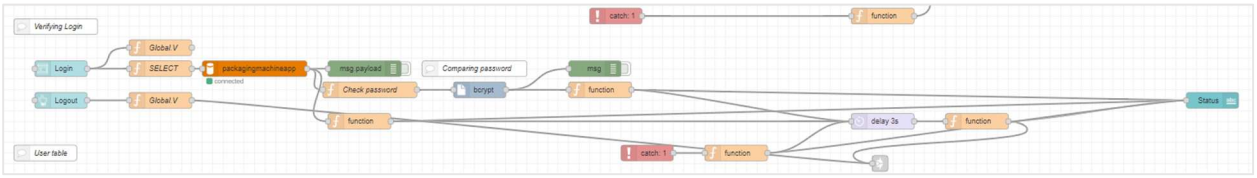


Figure 7: User loging verfication

Figure 8 shows the flow for updating a user password.

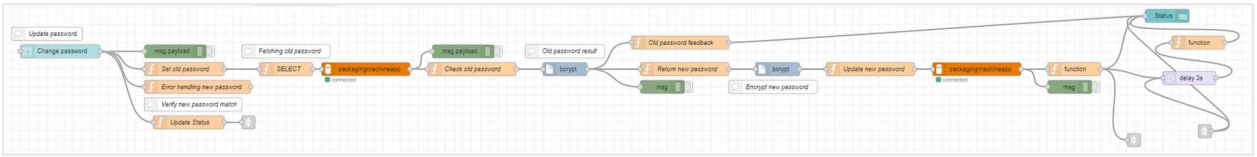


Figure 8: Update user password.

Figure 9 shows the flow for controlling the Node-RED UI for the admin and operator users.

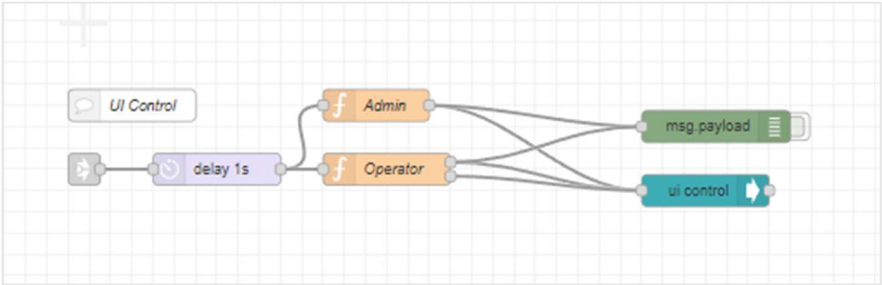


Figure 9: UI control flow.

Figure 10 illustrates the flow for database configuration.

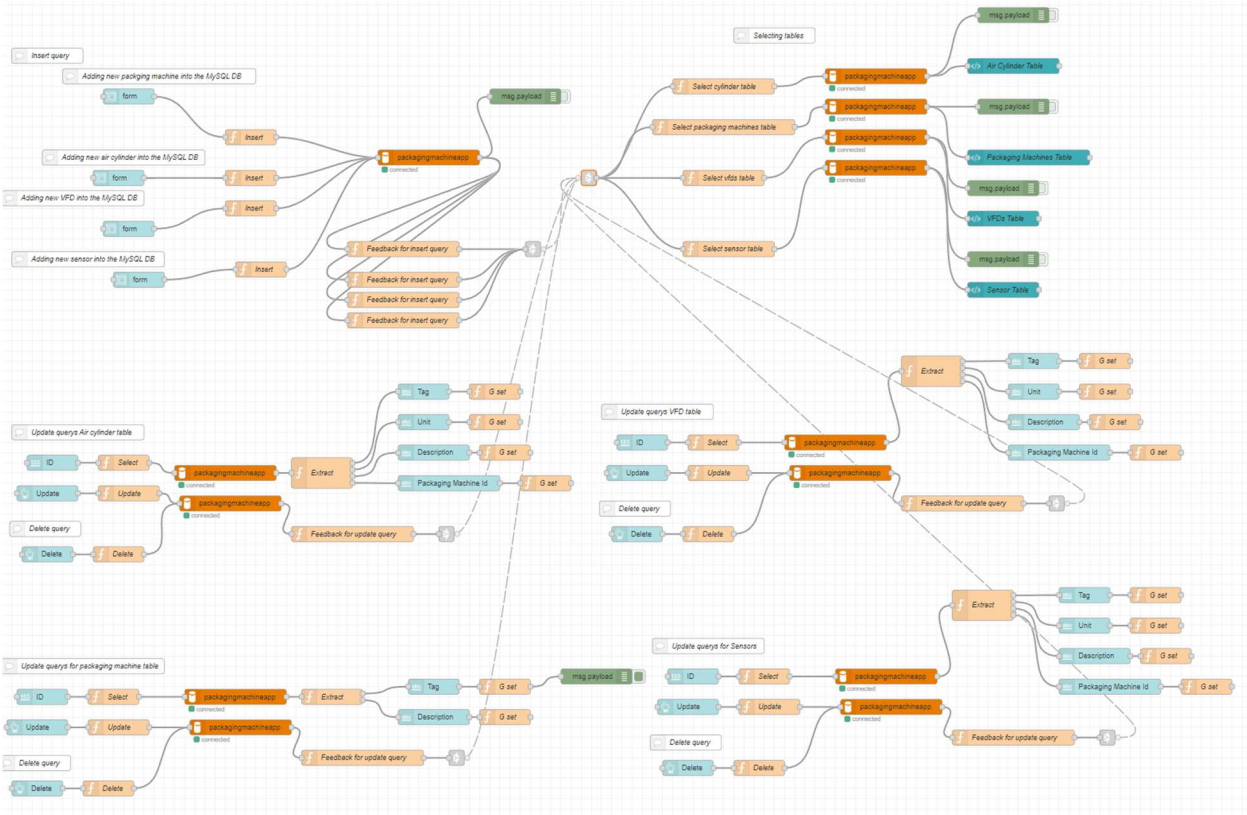


Figure 10: Node-RED flow application packging machine configuration.

Appendix G: Test Case document

Test Cases

Testing of application functionalities of the application for packaging machine.

Login to Grafana

No.	Test description	Comment
1	Dose created users; admin, operator, and guest can log in?	OK
2	Create a new user with a password that will have an editor role.	OK
3	Test the guest user. Can you do anything within the dashboards except view them?	OK

Dashboard (Grafana)

No.	Test description	Comment
1	Does the Home tab in the Grafana Home page link to the application's Home page?	OK
2	By clicking the Packaging Machine 01, does it take to the Test Celle dashboard?	OK
3	By clicking the Packaging Machine 02, does it take to the simulation of a packaging machine dashboard?	OK
4	Can the sampling time of the dashboards be changed to a different sampling time?	OK
5	Does the navigation in the dashboards to the air cylinders, VFDs, and sensors work?	OK
6	Can you resize the stats panels and graph panels and move them?	OK
7	Can the alert rules condition be changed and work?	OK

Alerts in Dashboards

No.	Test description	Comment
1	Is the Home page's alert list synchronized with alerts from Packaging Machine 01 and 02?	OK
2	Does the alert appear on the graph panels in all dashboards?	OK
3	Do the stat panels for air cylinders and VFDs change it cooler when alerting?	OK
4	Can the alert rules condition be changed and work?	OK
5	Dose email notification works?	Yes, the notifications appear in the trach inbox.

Configuration of Cloud-based database using Node-RED UI

No.	Test description	Comment
1	Login into the Node-RED UI using admin and operator user. Does it work?	OK
2	Update the admin and operator passwords. Does it work?	OK
3	Does the refresh button updates the user table?	OK
4	Delete users. Does it work?	OK

5	Log in with the operator user. Does the operator user has access to the user table?	OK
6	Type in wrong user or password. Dose the message Please try again! Pops up on status of log in window?	OK
6	Add new packaging machine, air cylinder, VFD, and sensor. Dose the tables in the database update with new information?	OK
7	Update and delete the new information you just added to the tables. Does it work?	OK
8	Check the above test cases with the MySQL Workbench tool to verify the changes in Azure. Does it work?	OK
9	Disconnect the Azure MySQL Server. Does the LED on the main page of the Node-RED UI gets red, and a notification window pops up with a message; *Azure Cloud DB is disconnected*?	OK

Monitoring live data using pgAdmin 4 and UaExpert software

No.	Test description	Comment
1	Can the live data from the air cylinders, VFDs, and sensor be monitored using the pgAdmin and UaExpert?	

Adding a new simulation packaging machine using Node-RED

No.	Test description	Comment
1	Add a new simulated packaging machine within the Node-RED with one air cylinder using inject node to simulate. Deploy the Node-RED flow with a new packaging machine. Does the program works	OK
2	Create one graph panel with a new dashboard in the Guarana to visualize the data from the air cylinder. Save the dashboard. And refresh the dashboard screen. Can you see the trend of the air cylinder data within the graph panel?	OK
3	Create an alert rule for the air cylinder you just added with the following settings in the Alert Config: • Give the alert a name. • Set evaluate every to 20s and for 20s. • Configure the query condition for the alert to when the last() last value of query(A, 10s, now) is above 20. • Apply the changes, save the dashboard, and refresh. The change the value of the air cylinder in Node-RED to be above the limit value (e,g., 30) in the alert rule. Are you getting any alerts?	OK

Appendix H: Screenshots from the Application PM during the testing.

Figure 1 illustrates the Node-RED flow for Test Celle, and the new simulated packaging machine 03 (PM03) consists of a VFD (A5).

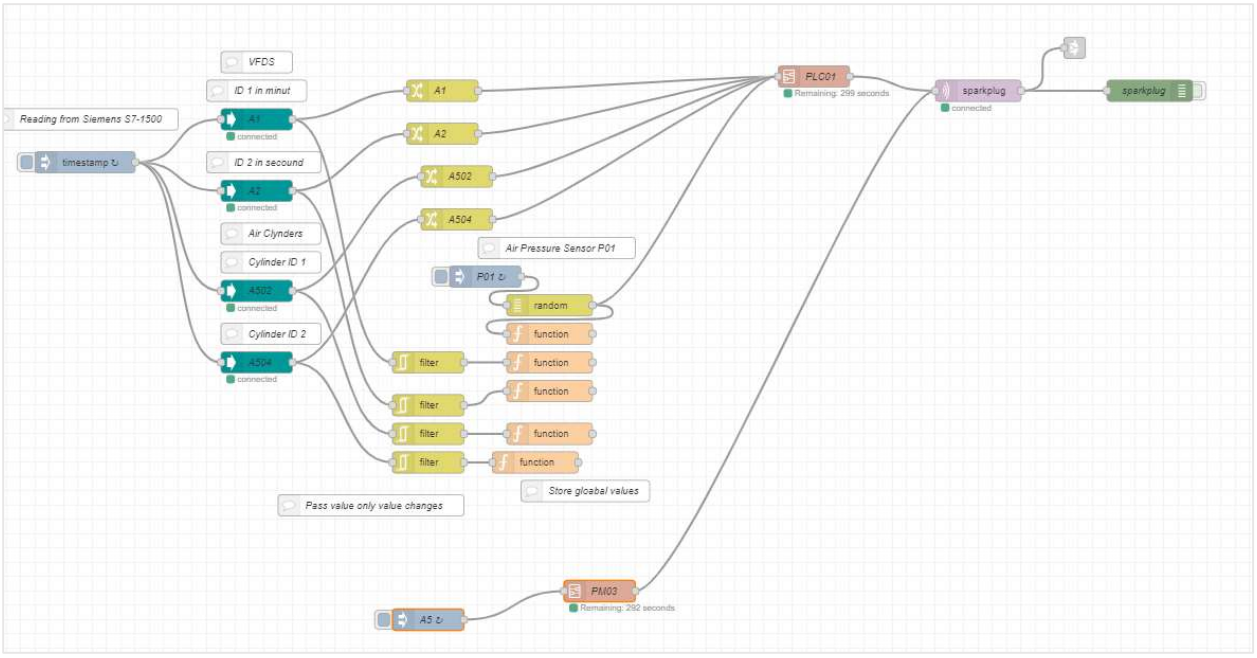


Figure 1: Node-RED flow for Test Celle machine and simulated Packaging Machine 03 (PM03).

Figure 2 displays the SQL query in the graph panel created for A5, used for fetching real-time data from the PostgreSQL data source.

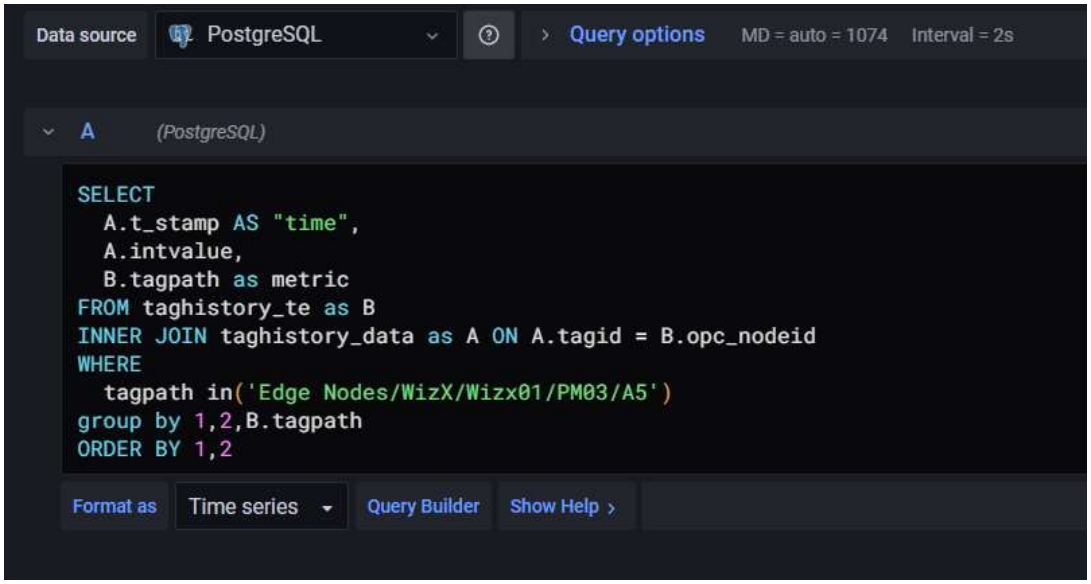


Figure 2: SQL syntax query for fetching real time data from PostgreSQL data source for A5.

To test the alert on the created dashboard, an alert rule with the condition if the last value of A5 is above 20 for 20s seconds, an alert should arise. Figure 3 shows the time-series graph panel for A5 where an alert has arisen.

Figure 3: A5 in Aletring alert state.

Figure 4 illustrates the Home dashboard where the alert on A5 has appeared in both the Last Change and Alerting table.

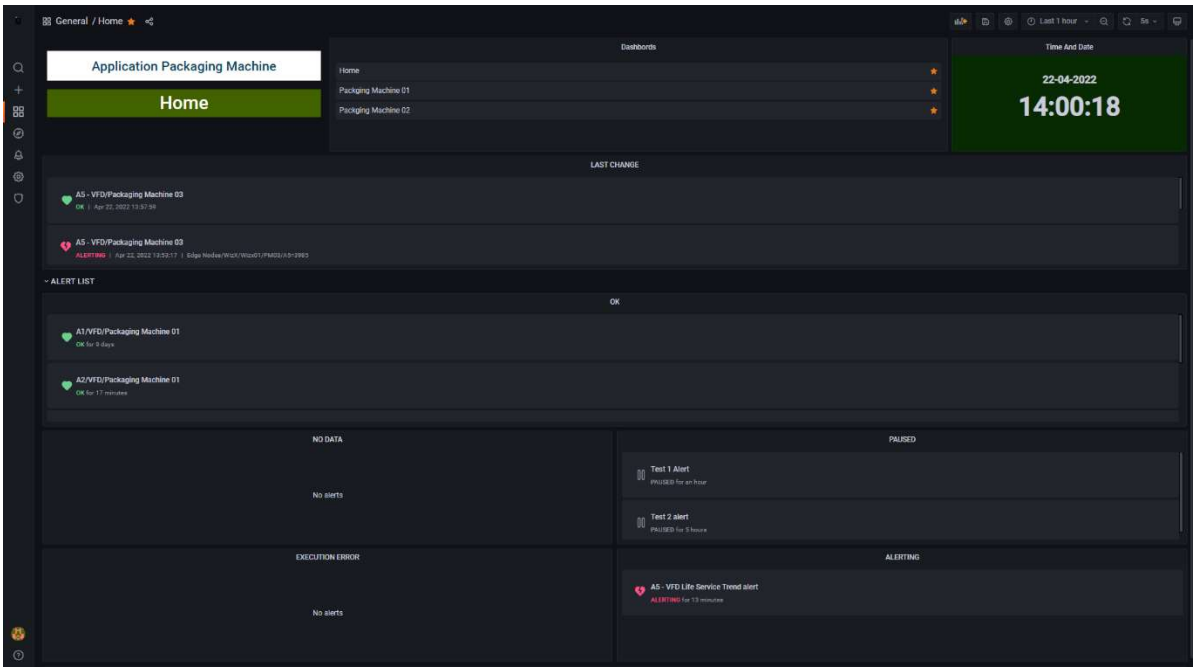


Figure 4: Home dashbord illustatting A5 in alerting state.

Figure 5 illustrates the MySQL Workbench screenshot where the packaging machine (PM03) is added to the packaging machine table within the cloud-based database.

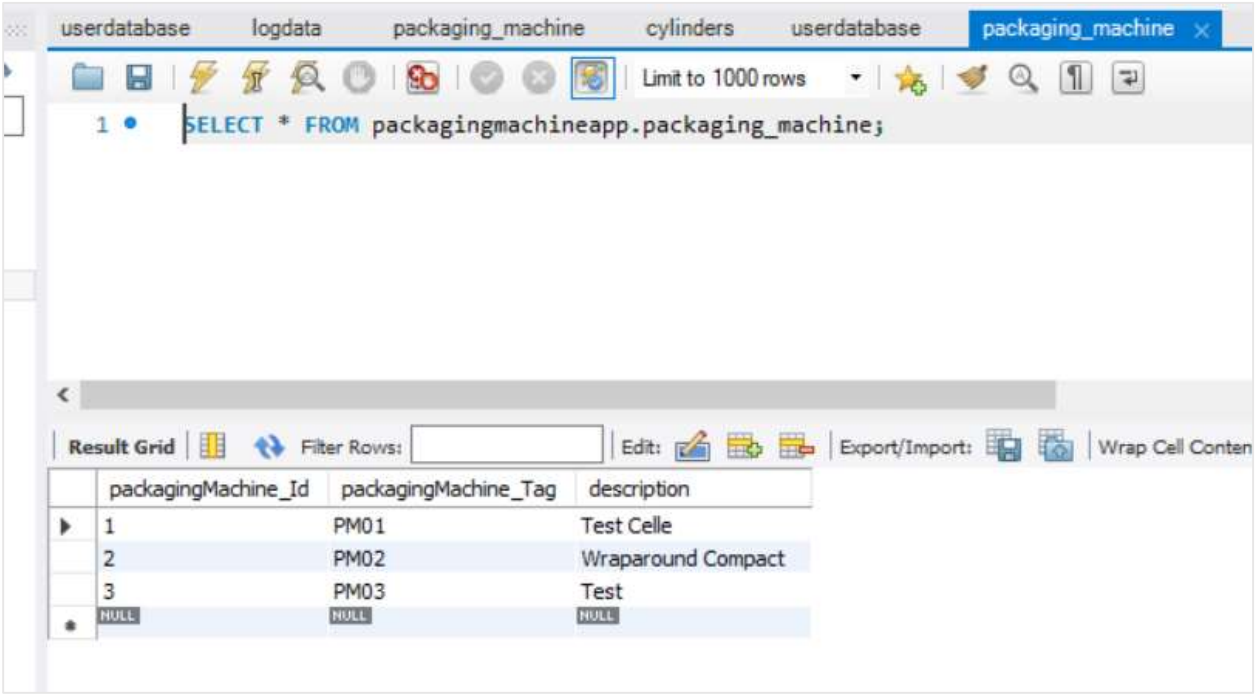


Figure 6: Select SQL query for packaging machine table MySQL Workbench is used.

Figure 6 illustrates the VFD table where A5 is added to the table.

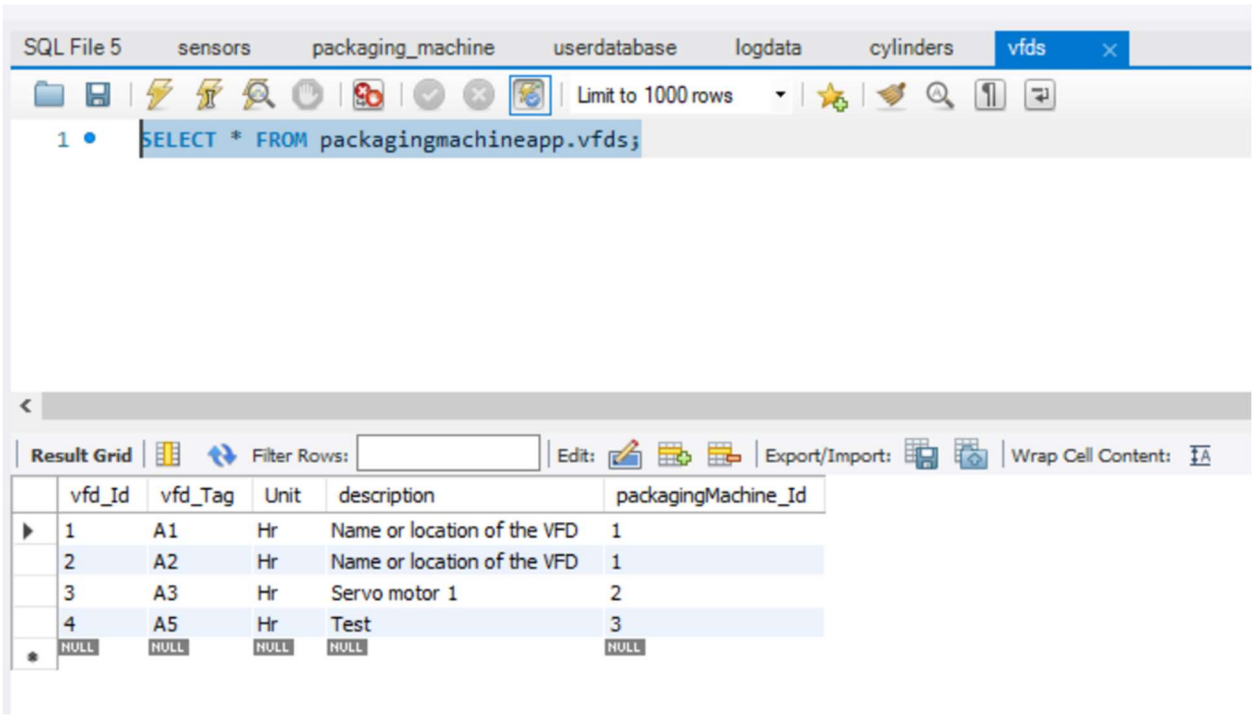


Figure 5: Select SQL query for VFD table MySQL Workbech is used.

Figure 7 illustrates the Node-RED UI when the operator user is logged in. From the figure, it can be seen that only the registration and database setting tabs are available on the UI.

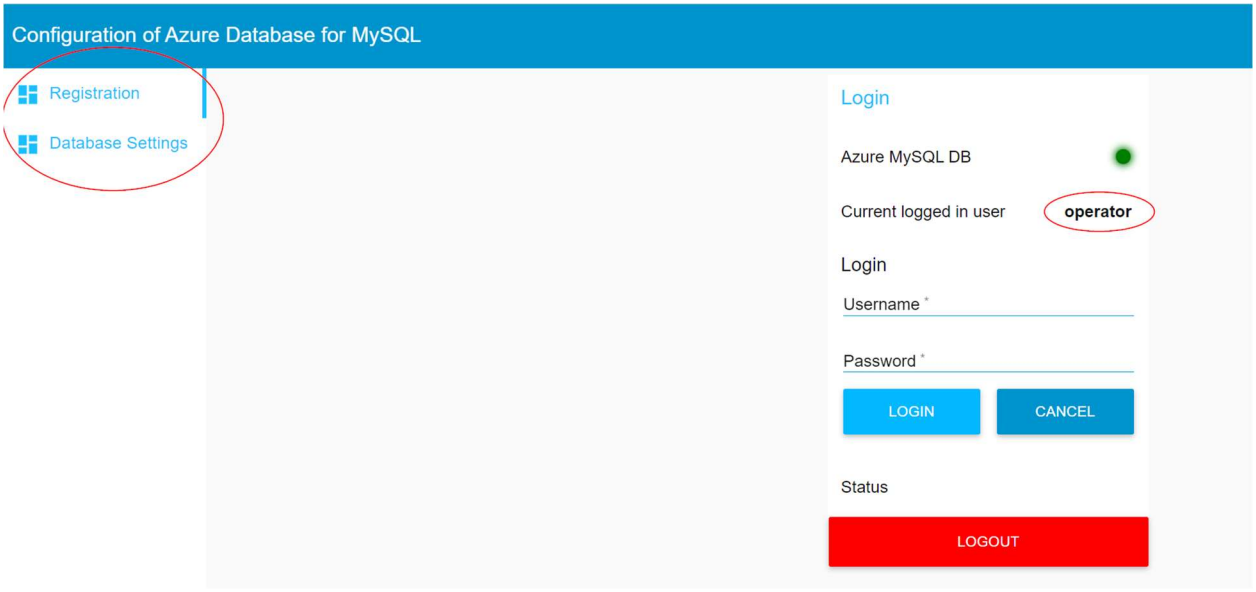


Figure 7: Node-RED UI when operator user is logged in.

Figure 8 shows the Node-RED UI when the admin user is logged in. All the tabs appear on the dashboard when the admin user is logged in.

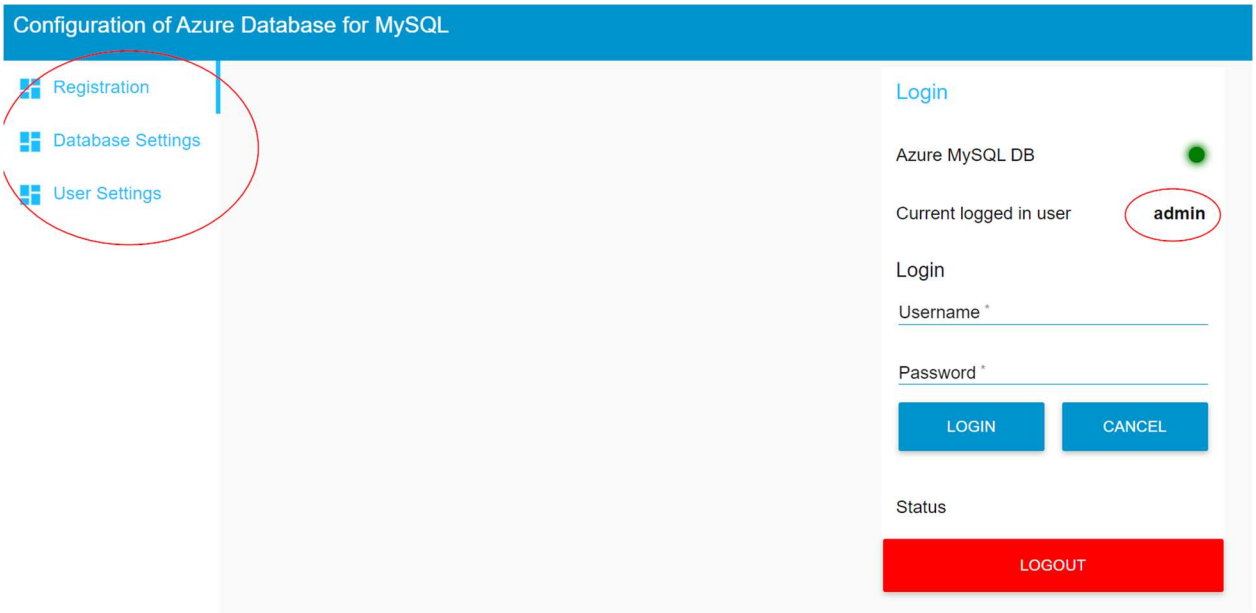


Figure 8: Node-RED UI when the admin user is logged in

Figure 9 shows the screenshot of the User Settings page.

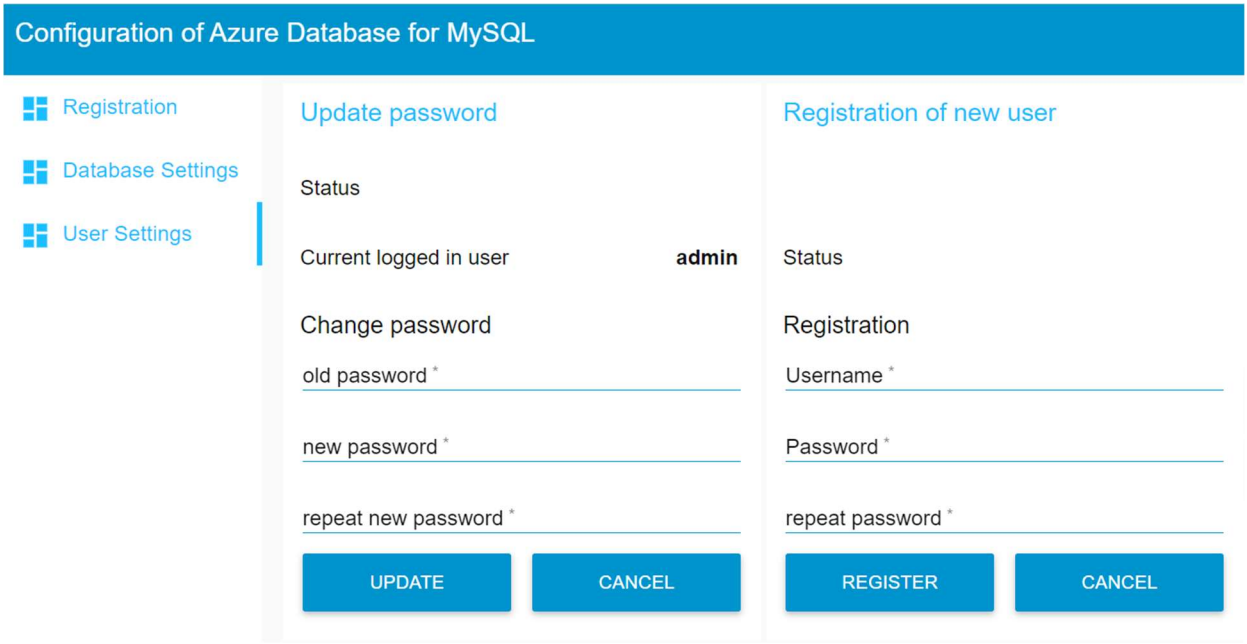


Figure 9: Node-RED UI User Settings window.

Figure 10 shows the screenshot of the Database Settings page.

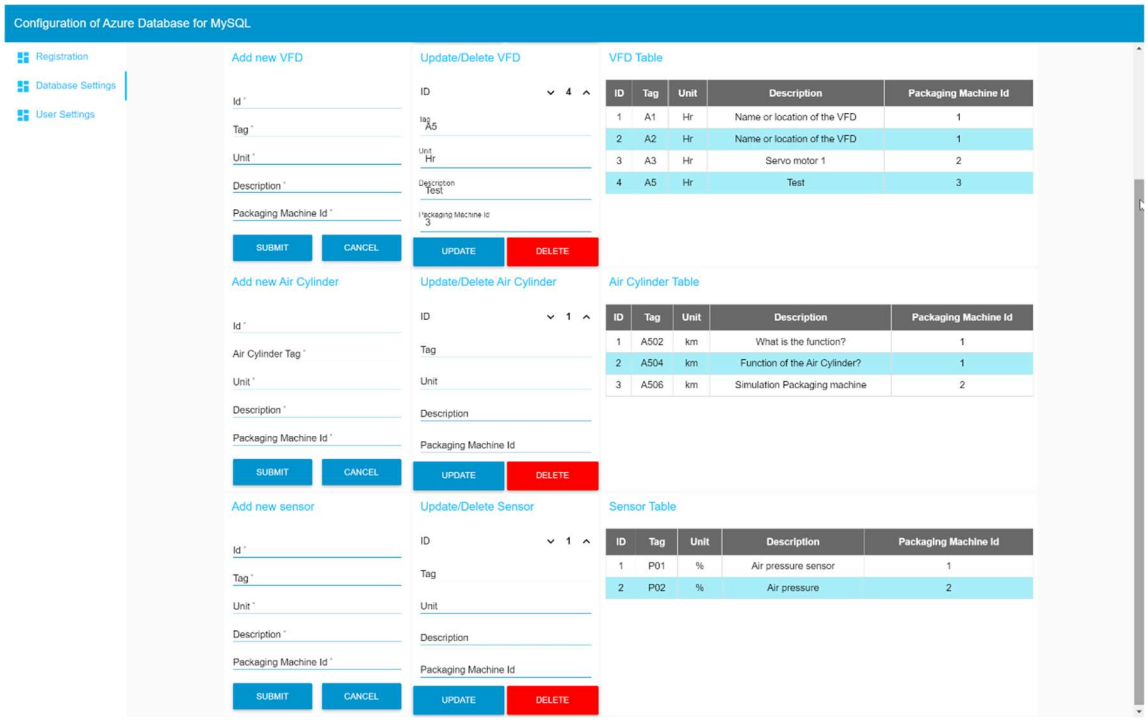


Figure 10: Database Settings Node-RED UI page.

Figure 11 shows when the database server on the Azure is not running, the green led becomes red, and a message window with Azure MySQL Connection is *Disconnected* pops up on the UI.

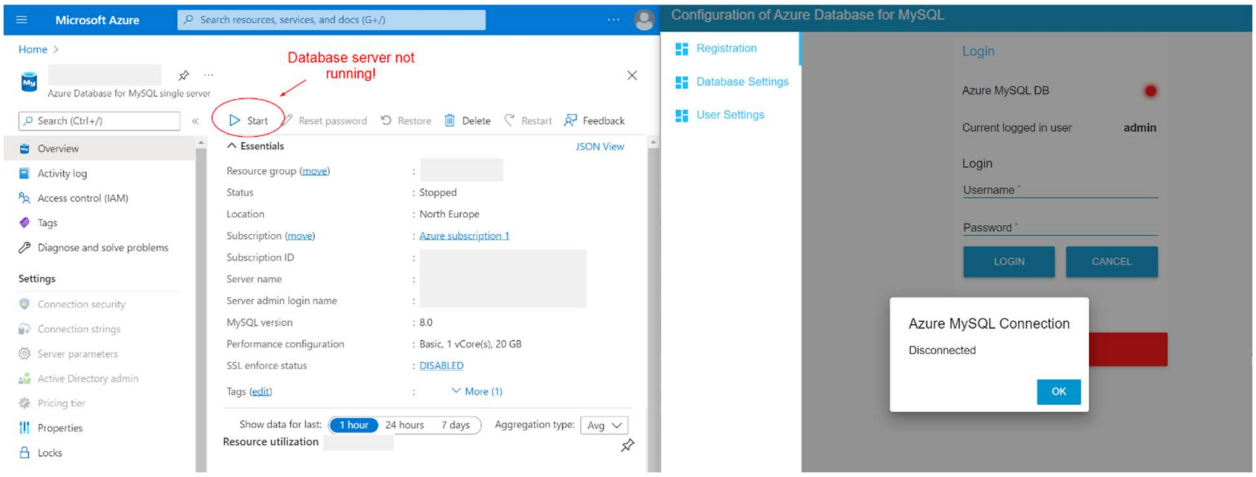


Figure 11: Notification on the Node-RED UI that notifyes that the Azure MySQL Connection status is disconnected.

Figure 12 shows when the database server on Azure is running, and a notification appears on the Node-RED UI with Azure MySQL Connection connected.

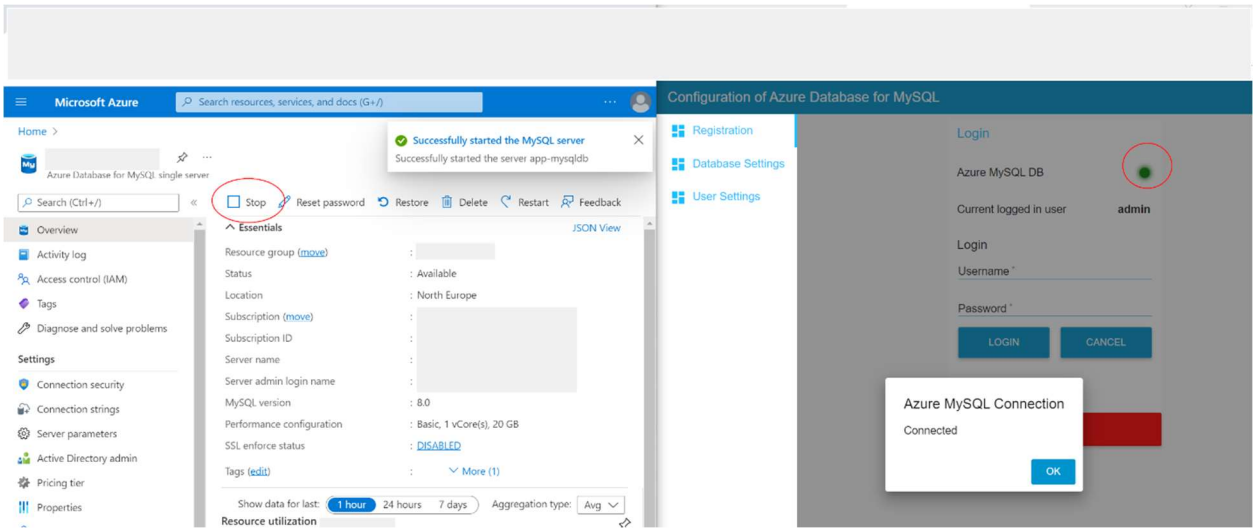


Figure 12: Notification on the Node-RED UI that notifyes that the Azure MySQL Connection status is connected.