Knowledge-based services for and optimization of machines

Deliverable D1.5b

Description of demonstrator for production process optimization

Deliverable type:	Document
Deliverable reference number:	ITEA 18030 | D1.5b
Related Work Package:	WP 1 Use Cases, Requirements & Evaluation
Due date:	M38 (30 November 2022)
Actual submission date:	14.11.2022
Author(s)	Mehmet GÖKÇEDAĞLIOĞLU / Mehmet Cem YILDIZ
Confidentiality	Public
Version	v 1.0

Contributors:

Name	Company
Mehmet GÖKÇEDAĞLIOĞLU	ERMETAL
Mehmet Cem YILDIZ	ERMETAL
Özlem ALBAYRAK	TEKNOPAR
Barış ÖKMEN	DOĞRU
Bilge ÖZDEMİR	ERSTE
Kamer KAYA	DAKİK

Table of Contents

Tables

Figures

1. Revision History

Version	Date	Description
v 1.0	14/11/2022	First version

2. Abstract

This demonstration will be implemented to a 1600T press line in Ermetal press production line. The press line consists of 5 presses (1x1600T press & 4x800T press) 8 transfer robots. Proper devices and software developed by Turkish and other MACHINAIDE partners can be used in the implementation of the demonstration. Aiming at predictive maintenance of the press, digital twin of the press will be created. To do so sensors will be attached on the press. The press will be observed virtually, and data will be collected by the help of the sensors, e.g. pressure, temperature, vibration. These data coming from the sensors will be the input for a learning algorithm. The visualisation and the learning algorithm will enable the operators to view the status of the process.

As a result of the demonstration, Ermetal will obtain several benefits such as determining the wearing effects on press, observing production parameter deviations might cause to breakdowns and increasing efficiency e.g. press shot per minute by avoiding stop of a press line by the help of assumptions of Digital Twin about possible breakdowns.

3. Description of demonstrator for production process optimization (ERMETAL)

3.1. Description and covered Use Cases

In the table below, general information such as use case definitions, responsibilities, possible contributions of partners to the use cases, interaction of use case with other work packages are given (Table 1).

Responsible person:	Mehmet GÖKÇEDAĞLIOĞLU
Main contributor, demonstrator responsibility:	ERMETAL
Additional contributors and needed contribution:	• TEKNOPAR: Use case description, Requirements description, Platform creation, Automation • ERSTE: Use case description, Requirements description, Machine-learning based event estimation, Data collection • DOGRU: Use case description, Requirements description, HMI applications • DAKIK: Use case description, Requirements description, Creation of Digital Twin • Others:
Use case:	• Start Production • Collect and Process Data • Train ML Processes • Conduct Predictive Maintenance • Simulate Product Lifecycle
What is going to be validated with this demonstrator?	• WP1 – Use Cases, requirements and evaluation • WP2 – Interoperability of Digital Twin eco-systems • WP3 – Processing of multiple Digital Twin's data • WP4 – Creating innovative HMIs for Digital Twin based services • WP5 - Information usage across the machine lifecycle
Start of project:	• 10/2019

Table 1: Validation profile of Description of demonstrator for production process optimization

3.2. Outline of the demonstration of ''Production Process Optimization Use Case''

This demonstrator is intended to simulate the MACHINAIDE use cases for Digital Twin applications in a press line which consists 5 presses and 8 robots.

Demonstration will simulate following 5 cases as defined in details in D1.1;

1. Start Production
2. Collect and process data during production
3. Train Macine Learning Process

4. Conduct Predictive Maintenance
5. Simulate Product Lifecycle

The demonstrator is going to be applied to the following 1600T press line of ERMETAL illustrated by pictures below (Figure 1-2 and 3);

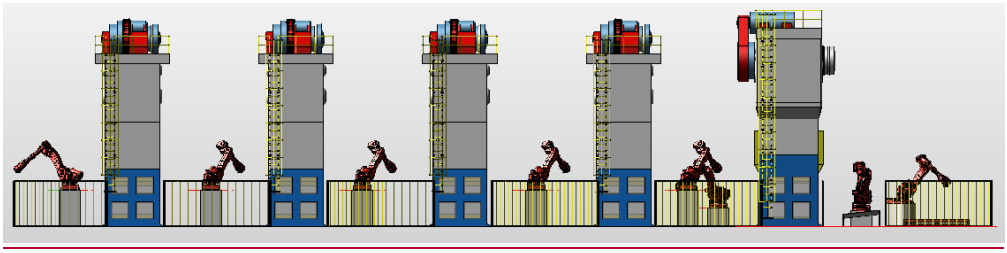


Figure 1: 3D Design of Ermetal Pressline

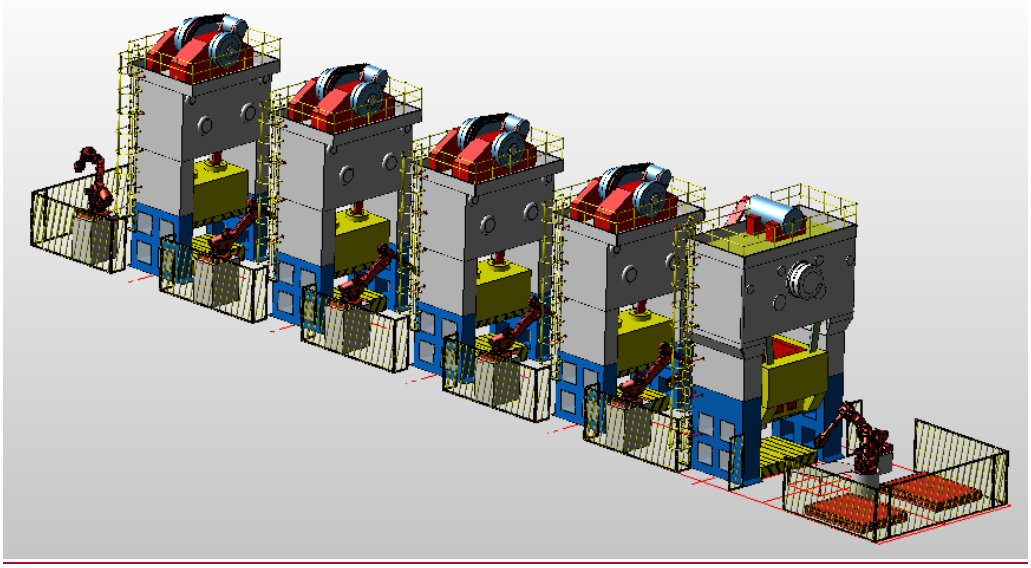


Figure 2: 3D Design of Ermetal Pressline

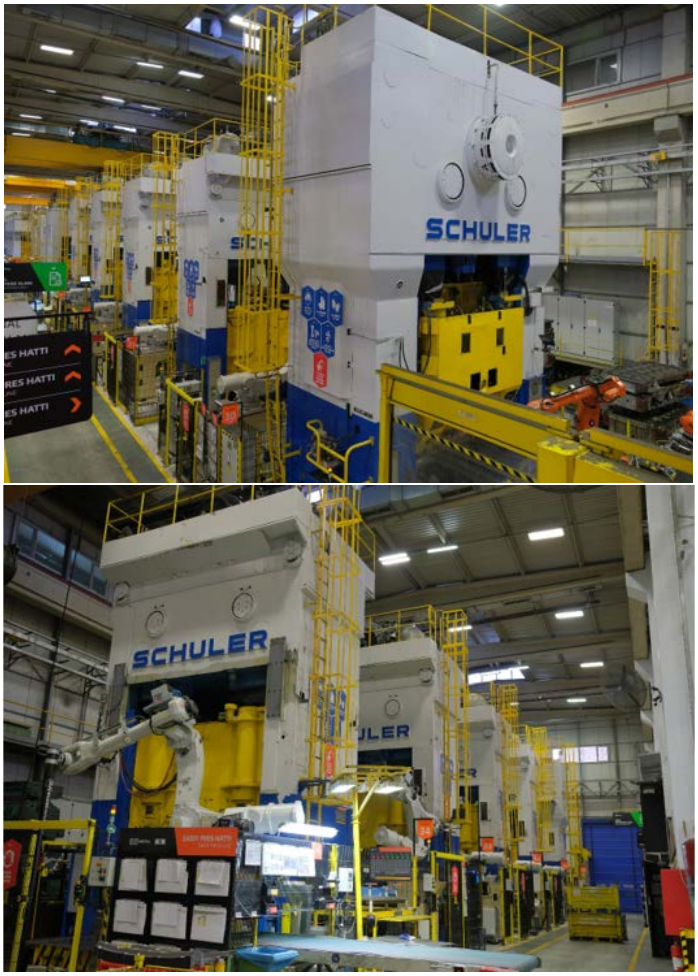


Figure 3: General view of shop floor on which demonstrator to be implemented

Sensor types to be used in demonstration are listed in table below (Table 2);

NO	SENSOR INFORMATION (Pressure sensor, heat sensor, vibration sensor etc.)	QUANTITY
1	VIBRATION SENSOR (Accelermeter 50g.0-6000 hz)	4
2	HEAT SENSOR (PT100 PABUÇ TİPİ)	2
3	HEAT SENSOR 4-20mA OI-LINK CONVERTER	2
4	HEAT SENSOR -50-150 C	1
5	AIR FLOW SENSOR	4
6	PRESSURE SENSOR WITH SCREEN 0-100bar	1
7	PRESSURE SENSOR WITH SCREEN 0-400bar	1
8	ENERGY METER 400 V	5
	TOTAL	20

Table 2: List of Sensors

PLC and other assistance hardwares are listed in the table below (Table 3);

NO	PLC AND ASSISTANCE HARDWARES	QUANTITY
1	PLC CPU 1515-PN	1
2	VIBRATION DIAGNOSTICS UNIT (BETWEEN SENSORS-PLC)	1
3	VIBRATION DIAGNOSTICS UNIT (BETWEEN SENSORS-PLC)	2
4	SIMATIC PLC MEMORY CARD 4MB	1
5	SIMATIC BUSADAPTER BA2XRJ45	2
6	SCALANCE XB008 INDUSTRIAL ETHERNET SWITCH	1
7	SIMATIC PN/PN COUPLER	1
8	SITOP PSU100S 24 V/5 A POWER SUPPLY	1
9	INTERFACE MODULE	1
10	SIMATIC ET 200SP BUSADAPTER	1
11	PLC MODULE SOCKET	6
	TOTAL	18

Table 3: List of PLC and other assistance hardwares

The sensors listed above purchased within the scope of the project and mounted on determinated presses for demonstration. Each press has a total number of 24 sensors and 47 data can be tracked via these sensors. All of these data flows and are ready to store for processing for the purpose of Digital Twin creation etc.

PC information to be used for this demonstration are listed in the table below (Table 4);

NO	PC INFORMATION	QUANTITY
1	DELL OPTIPLEX 5080MT 17-10700/8GB/256SSD/WIN10PRO Computer	1
2	KINGSTON 8GB 2666MHZ CL19 DDR4 PC RAM	1
3	DELL SE2416H 23.8 1920X1080 LED MONITOR	1
	TOTAL	3

Table 4: List of PC information

Softwares and platforms to be used for this demonstration are listed in the table below (Table 5) ;

NO	SOFTWARES AND PLATFORMS	QUANTITY
1	OPC ROUTER 4 - KAFKA PLUG-IN	1
2	OPC ROUTER 4 BASIC	1
3	OPC ROUTER UA/DA CLIENT	1
4	OPC UA Licence (SIMATIC OPC UA S7-1500)	1
	TOTAL	4

Table 5: List of softwares and platforms

By using sensors, PLC, hardwares, PC information, softwares and platforms, mentioned above, following press parameters will be collected;

RELEVANT PRESS SYSTEM	DATA DEFINITON
MAIN ENGINE	MAIN ENGINE RUN TIME
	MAIN ENGINE START TIME
	MAIN ENGINE MINUTE-REVOLUTION NUMBER
	MAIN ENGINE BODY TEMPERATURE
	MAIN ENGINE BACK REGION VIBRATION
	MAIN ENGINE BACK REGION IMBALANCE
	MAIN ENGINE BACK REGION MISALIGNMENT
	MAIN ENGINE BACK REGION AXIAL CLEARENCE
	MAIN ENGINE BACK REGION BEARING
	MAIN ENGINE BACK REGION ACCELERATION
	MAIN ENGINE FRONT REGION VIBRATION
	MAIN ENGINE FRONT REGION IMBALANCE
	MAIN ENGINE FRONT REGION MISALIGNMENT
	MAIN ENGINE FRONT REGION AXIAL CLEARENCE
	MAIN ENGINE BACK REGION BEARING

	MAIN ENGINE FRONT REGION ACCELERATION
LUBRICATION	LUBRICATION PRESSURE
	LUBRICATION OIL ROTATING TEMPERATURE
	LUBRICATION UNIT HYDRAULIC PUMP VIBRATION
	LUBRICATION UNIT HYDRAULIC PUMP IMBALANCE
	LUBRICATION UNIT HYDRAULIC PUMP PALLET
	LUBRICATION UNIT HYDRAULIC PUMP ENGINE BEARING
	LUBRICATION UNIT HYDRAULIC PUMP ACCELERATION
FLYWHEEL	FLYWHEEL BEARING TEMPERATURE
	FLYWHEEL VIBRATION
	FLYHEEL IMBALANCE
	FLYHEEL GEAR
	FLYWHEEL BEARING
	FLYWHEEL ACCELERATION
	FLYWHEEL INTERNAL BEARING
	FLYWHEEL INTERNAL ACCELERATION
BALANCING	BALANCING AIR PRESSURE
	BALANCING AIR CONSUMPTION
	BALANCING AIR TEMPERATURE
GENERAL PRESS	MAIN SUPPLY LINE AIR CONSUMPTION
	MAIN SUPPLY LINE AIR TEMPERATURE
	CLUTCH BRAKING AIR CONSUMPTION
	CLUTCH BRAKING AIR TEMPERATURE
	HYDRAULIC SAFETY OIL PRESSURE
	BRAKING GROUP WORKING TEMPERATURE
	RAM TUNING POSITION
	OPERATING DIE NUMBER OR NAME
	PRESS PRESSURE TONNAGE
	RAM STROKE NUMBER
ROBOT	ROBOT MAIN SUPPLY LINE AIR CONSUMPTION
	ROBOT MAIN SUPPLY LINE AIR TEMPERATURE
	ROBOT CONTROLER UNIT INTERNAL TEMPERATURE MONITORING
MAIN MACHINE ENERGY	VOLTAGE BETWEEN L1-L2 PHASES
	VOLTAGE BETWEEN L2-L3 PHASES
	VOLTAGE BETWEEN L3-L1 PHASES
	VOLTAGE BETWEEN NEUTRAL-L1
	VOLTAGE BETWEEN NEUTRAL-L2

	VOLTAGE BETWEEN NEUTRAL-L3
	L1 PHASE CURRENT
	L2 PHASE CURRENT
	L3 PHASE CURRENT
	NETWORK FREQUENCY
	ACTIVE COUNTER INDEX
	REACTIVE COUNTER INDEX
	TOTAL ACTIVE POWER
	TOTAL REACTIVE POWER
	TOTAL POWER FACTOR POWER
MAIN MOTOR ENERGY	VOLTAGE BETWEEN L1-L2 PHASES
	VOLTAGE BETWEEN L2-L3 PHASES
	VOLTAGE BETWEEN L3-L1 PHASES
	L1 PHASE CURRENT
	L2 PHASE CURRENT
	L3 PHASE CURRENT
	ACTIVE COUNTER INDEX
	REACTIVE COUNTER INDEX
	TOTAL ACTIVE POWER
	TOTAL REACTIVE POWER
	TOTAL POWER FACTOR POWER
RAM TUNING MOTOR ENERGY	VOLTAGE BETWEEN L1-L2 PHASES
	VOLTAGE BETWEEN L2-L3 PHASES
	VOLTAGE BETWEEN L3-L1 PHASES
	L1 PHASE CURRENT
	L2 PHASE CURRENT
	L3 PHASE CURRENT
	ACTIVE COUNTER INDEX
	REACTIVE COUNTER INDEX
	TOTAL ACTIVE POWER
	TOTAL REACTIVE POWER
	TOTAL POWER FACTOR POWER
LUBRICATION HYDRAULIC PUMP MOTOR ENERGY	VOLTAGE BETWEEN L1-L2 PHASES
	VOLTAGE BETWEEN L2-L3 PHASES
	VOLTAGE BETWEEN L3-L1 PHASES
	L1 PHASE CURRENT
	L2 PHASE CURRENT
	L3 PHASE CURRENT

	ACTIVE COUNTER INDEX
	REACTIVE COUNTER INDEX
	TOTAL ACTIVE POWER
	TOTAL REACTIVE POWER
	TOTAL POWER FACTOR POWER
DIE TROLLEY HYDRAULIC PUMP MOTOR ENERGY	VOLTAGE BETWEEN L1-L2 PHASES
	VOLTAGE BETWEEN L2-L3 PHASES
	VOLTAGE BETWEEN L3-L1 PHASES
	L1 PHASE CURRENT
	L2 PHASE CURRENT
	L3 PHASE CURRENT
	ACTIVE COUNTER INDEX
	REACTIVE COUNTER INDEX
	TOTAL ACTIVE POWER
	TOTAL REACTIVE POWER
	TOTAL POWER FACTOR POWER

Table 6: Press parameters to be collected

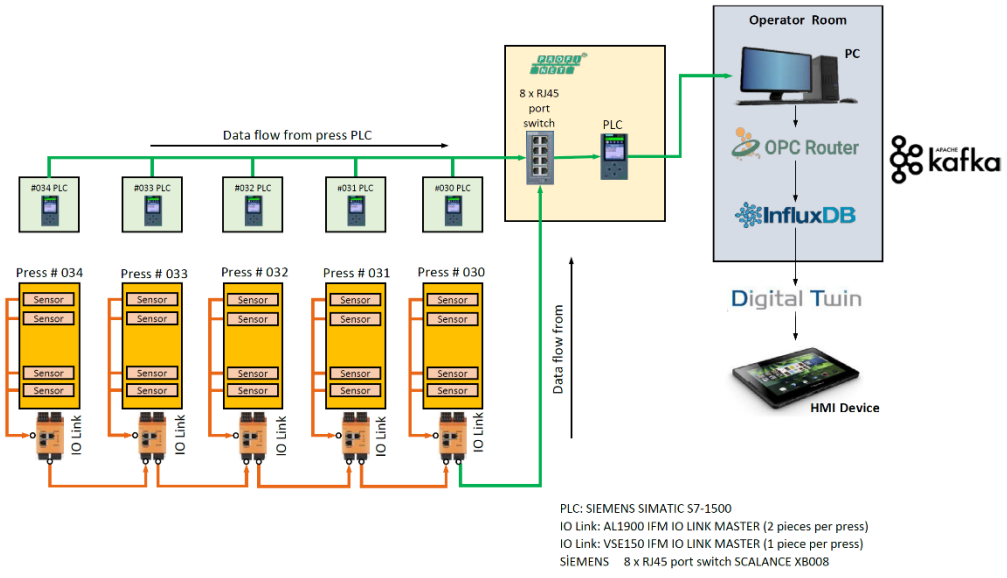


Figure 3: Data Flow Scheme

3.3. Creation of Digital Twin (DAKIK- ERSTE)

3.3.1. DTDL Standards – creation

A key challenge with digital twin technology is understanding how to model industry-specific domain knowledge while enabling different platforms to communicate with each other to make sense of the complex relationships that exist in the physical world. Assuming these twins also interact with each other, a standard digital twin definition language is needed to create and manage the twins in a digitalized ecosystem. However, as industry technology evolves, the languages that support their development must evolve as well. In this project, we have used Azure Digital Twins Definition Language (DTDL). which is implemented and made available by Microsoft as the core DTDL for their Digital Twin applications.[1]

3.3.1.1. Azure DTDL

The key features of Azure DTDL are that you can define your own dictionary and can create your twin graph under automatically defined conditions of your business. This feature is provided through User-defined models. You can think of your models as names in a description of the world.

The model is similar to a class in an object-oriented programming language and defines a data shape for a specific concept in your actual work environment. Models have names (such as a room or temperature sensor) and contain items such as properties, telemetry / events, and commands that describe what this type of entity in your environment can do. You will then use these models to create the digital twin representing specific entities that meet this type of description.[1]

Azure digital twin models are defined using the digital twin definition language (DTDL). DTDL is based on JSON-LD and is programming language independent. It is not exclusive to Azure, but is also used to display device data in other IoT services such as IoT Plug and play.

Within a model definition, the top-level code element is an interface. This encapsulates the entire model and the rest of the model is defined within the interface. A DTDL model interface can contain zero, one, or more of each of the following fields:

- **Property :** properties are data fields that represent the state of an entity (such as properties in many object-oriented programming languages). Features backs up storage and can be read at any time.
- **Telemetry** : telemetry fields represent measurements and events and are often used to show device sensor settings. Unlike features, a telemetry digital twin is not stored; This is a set of time-bound data events that must be processed in the order they occur.

[1] https://docs.microsoft.com/en-us/azure/digital-twins/concepts-models

- **Component :** components allow you to build your model interface as a compilation of other interfaces. An example of a component is a leading Camera interface (and another component interface back camera) used when defining a model for a Phone. First, you need to define an interface for front-camera like its model and then you can refer to it when defining Phone.
- **Relationship :** relationships allow you to represent how to incorporate a digital twin with other digital TWINS. Relationships allow the solution to provide a chart of interrelated entities.

Below is a typical model written as a DTDL interface. The model describes the planners, each having a name, batch, and temperature. Consider that planets can also interact with their moons, the Moons. In the example below, the model refers to the links to these other entities by referring to two external models (and). These models are also defined in the sample code below, but are too simple to stop at the primary example.

```
[
  {
    "@id": "dtmi:com:contoso:Planet;1",
    "@type": "Interface",
    "@context": "dtmi:dtdl:context;2",
    "displayName": "Planet",
    "contents": [
      {
        "@type": "Property",
        "name": "name",
        "schema": "string"
      },
      {
        "@type": "Property",
        "name": "mass",
        "schema": "double"
      },
      {
        "@type": "Telemetry",
        "name": "Temperature",
        "schema": "double"
      },
      {
        "@type": "Relationship",
        "name": "satellites",
        "target": "dtmi:com:contoso:Moon;1"
      },
      {
        "@type": "Component",
        "name": "deepestCrater",
        "schema": "dtmi:com:contoso:Crater;1"
      }
    ]
  },
  {
    "@id": "dtmi:com:contoso:Crater;1",
    "@type": "Interface",
    "@context": "dtmi:dtdl:context;2"
  },
  {
    "@id": "dtmi:com:contoso:Moon;1",
    "@type": "Interface",
    "@context": "dtmi:dtdl:context;2"
  }
]
```

The table below shows some of the keywords and definitions used when creating a digital twin with Azure DTDL. For more details, you can refer to https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v2/dtdlv2.md.

	Explanation

@id	An identifier for the model. It should be in the form. dtmi: <domain> : <unique model identifier> ; <model version number>
@type	Defines the type of information being disclosed. For an interface, the type *interface*.
@context	Sets the JSON document context. Models should use. dtmi:dtdl:context;2
displayName	The selection allows you to give the model an easy name if you want.
contents	All remaining interface data is placed here as an array of attribute definitions. Each attribute must provide a (feature, telemetry, command, relationship, or component) to determine the order of the interface information it describes, followed by a set of properties that define the actual attribute (for example, and define a feature).

```
[
  {
    "@id": "dtmi:com:contoso:CelestialBody;1",
    "@type": "Interface",
    "@context": "dtmi:dtdl:context;2",
    "displayName": "Celestial body",
    "contents": [
      {
        "@type": "Property",
        "name": "name",
        "schema": "string"
      },
      {
        "@type": "Property",
        "name": "mass",
        "schema": "double"
      },
      {
        "@type": "Telemetry",
        "name": "temperature",
        "schema": "double"
      }
    ]
  },
  {
    "@id": "dtmi:com:contoso:Planet;1",
    "@type": "Interface",
    "@context": "dtmi:dtdl:context;2",
    "displayName": "Planet",
    "extends": "dtmi:com:contoso:CelestialBody;1",
    "contents": [
      {
        "@type": "Relationship",
        "name": "satellites",
        "target": "dtmi:com:contoso:Moon;1"
      },
      {
        "@type": "Component",
        "name": "deepestCrater",
        "schema": "dtmi:com:contoso:Crater;1"
      }
    ]
  },
  {
    "@id": "dtmi:com:contoso:Crater;1",
    "@type": "Interface",
    "@context": "dtmi:dtdl:context;2"
  }
]
```

Sometimes you may want to further customize a model. For example, it can be useful to have a general model room and a customized assortment conference-room and a Gym. Azure DTDL supports inheritance in fast customization: interfaces can be inherited from one or more interfaces. The example on the left redisplay the Planet model above as a subtype of a larger model. The "top" model is defined first and then created using the "bottom" model area.

3.3.1.2. Example JSON from the Turkish use-case

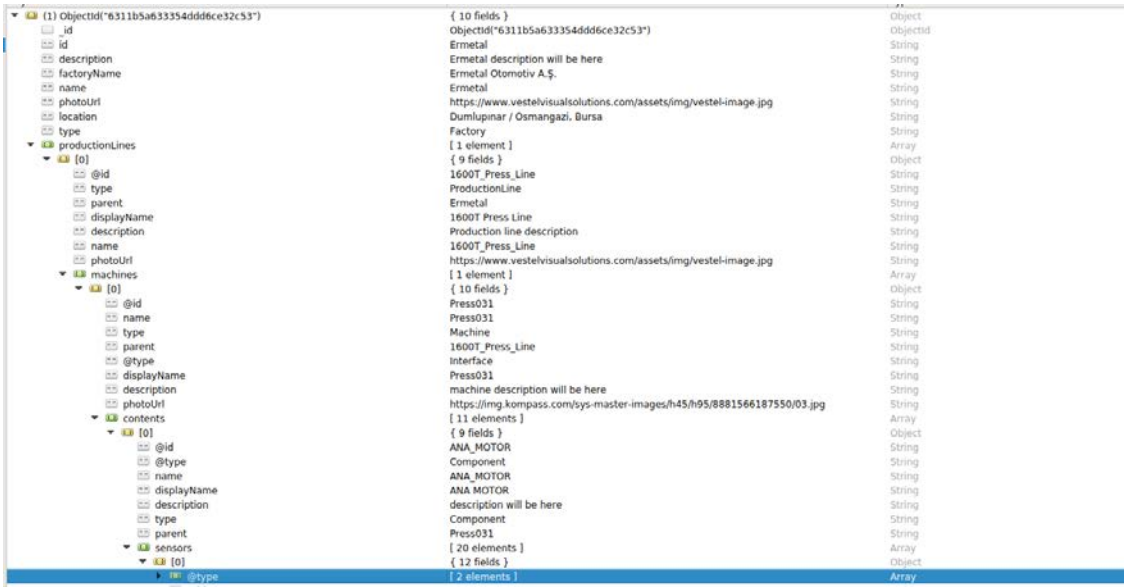
The following file containing the hierarchy of factory, production line, machine, component, sensor, field has been received from ERMETAL. In this file, there is

information that identifies the parent-child relationship among the parts. In addition, metadata information such as name, displayName, description and data source settings about the relevant part are also kept.

	name	parent_id	type	unit	displayName		description
2	Ana_hava_debi_act	GENEL PRES	Debi	Real	Pres ana hava giris debi aktuel		description will be here
3	Ana_hava_sic_act	GENEL PRES	Sıcaklık	Real	Pres ana hava giris sıcaklık aktuel		description will be here
4	Deng_hava_debi_act	DENGELEME	Debi	Real	Koç dengeleme silindirler hava debi aktuel		description will be here
5	Deng_hava_sic_act	DENGELEME	Sıcaklık	Real	Koç dengeleme silindirler hava sıcaklık aktuel		description will be here
6	Robot_hava_debi_act	ROBOT	Debi	Real	Robot hava debi aktuel		description will be here
7	Robot_hava_sic_act	ROBOT	Sıcaklık	Real	Robot hava sıcaklık aktuel		description will be here
8	Kavr_hava_debi_act	GENEL PRES	Debi	Real	Pres kavrama hava debi aktuel		description will be here
9	Kavr_hava_sic_act	GENEL PRES	Sıcaklık	Real	Pres kavrama hava sıcaklık aktuel		description will be here
10	Yaglama_bas_act	YAĞLAMA	Basınç	Real	Pres yağlama ünite çıkış basınç aktuel		description will be here
11	Yaglama_sic_act	YAĞLAMA	Sıcaklık	Real	Pres yağlama yağ dönüş sıcaklık aktuel		description will be here
12	Hid_emn_bas_act	GENEL PRES	Basınç	Real	Koç hidrolik emniyet basınç aktuel		description will be here
13	Deng_hava_bas_act	DENGELEME	Basınç	Real	Koç dengeleme silindirler hava basıncı		description will be here
14	Frn_grup_sic_act	GENEL PRES	Sıcaklık	Real	Frenleme grubu çalışma sıcaklığı		description will be here
15	Reg_poz_act	GENEL PRES	mm	Real	Koç reglaj pozisyon		description will be here
16	Koc_Ton_act	GENEL PRES	Ton	Real	Koç baskı Tonaj		description will be here
17	Rob_ctr_sic_act	ROBOT	Sıcaklık	Real	Robot controller ünite iç sıcaklığı		description will be here
18	AM_rpm_act	ANA MOTOR	rpm/m	Real	Ana motor dak/devir		description will be here
19	Yedek1	GENEL PRES	empty	Real	Yedek		description will be here
20	Yedek2	GENEL PRES	empty	Real	Yedek		description will be here
21	Kalip_No	GENEL PRES	empty	Int	Calisan kalıp numarası		description will be here
22	Koc_Baski_act	GENEL PRES	empty	bit	Pres basıyor		description will be here
23	AM_start_act	ANA MOTOR	empty	bit	Ana motor calisiyor		description will be here
24	Yaglama_act	YAĞLAMA	empty	bit	Yağlama calışıyor		description will be here
25	Pres_baskı_sayisi	GENEL PRES	empty	Lint	Pres baskı sayısı	(kamulatif)	description will be here
26	AM_calisma_suresi_saat	ANA MOTOR	empty	Dint	Pres Ana motor çalışma süresi (saat)	(kamulatif)	description will be here
27	Pres_baskı_sayisi_reset	GENEL PRES	empty	bool	Pres baskı sayısı sıfırlama (reset)		description will be here
28	AM_calisma_suresi_reset	ANA MOTOR	empty	bool	Pres Ana motor süresi sıfırlama (reset)		description will be here
29	Koc_pozisyon	GENEL PRES	° derece	Int	Pres koç pozisyon		description will be here
30	AM_Arka_Vib	ANA MOTOR	mm/s	Real	Pres ana motor arka bölge vibrasyon		description will be here
31	AM_Arka_Balans	ANA MOTOR	mm/s	Real	Pres ana motor arka bölge balanssızlık		description will be here
32	AM_Arka_Eks_kac	ANA MOTOR	mm/s	Real	Pres ana motor arka bölge eksen kaçıklığı		description will be here
33	AM_Arka_Bosluk	ANA MOTOR	mm/s	Real	Pres ana motor arka bölge eksen boşluğu		description will be here
34	AM_Arka_Rulman	ANA MOTOR	m/s2	Real	Pres ana motor arka bölge rulman		description will be here
35	AM_Arka_Acc	ANA MOTOR	m/s2	Real	Pres ana motor arka bölge Acc (ivme)		description will be here

Factory | ProductionLine | Machine | Inter | Sensor | Field

According to this file, the JSON structure shown below has been created. This JSON structure is used in the creation and management of the digital twin in the program.



3.3.1.3. Tree hierarchy and its management/visualization

The digital twin monitor page is where the configurations of the digital twin are made and monitored. This main page consists of 3 parts. In the middle, there is a graph/tree whose nodesand edges generate the factory, production line, machine, component, sensor and field hierarchy. In the left part, the metadata of a selected node on the graph is displayed, and in the right part, a 3-dimensional visual model of a selected node on the graph is displayed.

3.3.1.4. Complex twins, interoperability and TWINAIDE

Twinaide is an open-source platform for managing and creating interoperable digital twins. Twinaide aims to support innovative concepts for accessing, searching, analysing and using multiple Digital Twins data. Twinaide will allow the data of digital twins to be pulled from sources such as Kafka, RabbitMQ, InfluxDB and displayed on appropriate graphics. Using an extended version of Azure DTDL, Twinaide will allow users to flexibly create their own hierarchy and upload them to the system via prepared files. The Twinaide tool consists of 3 separate module: a single page application developed with React, a rest API developed with NodeJS and ExpressJS, and a Flask API developed to interact with third-party services.

- Source codes of API developed using NodeJS, ExpressJS and MongoDB can be accessed at twinaide.
- The source code of the interface developed using ReactJS and various Javascript libraries can be accessed at twinaide-ui.
- The API developed using Python and Flask technologies and communicating with third-party services can be accessed at twinaide-python.

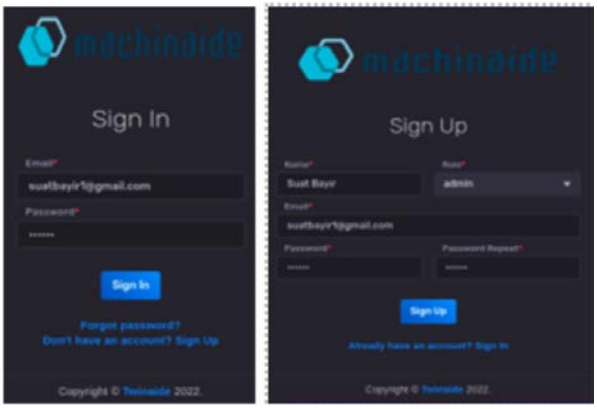
For the implementation of Twinaide, the following technology stack have been used: Overall, we used the traditional MERN (MongoDB, Express.js, React.js, Node.js) stack while developing the web application. On top of MERN, we employed a Javascript library Three.js to render the visual scenes on the browser and d3.js for creating the Digital Factory graph which visualize the relationships, e.g., child/parent, among the multiple twins insider the factory. The integrator currently supports Kafka and MQTT data brokers and using Websocket to connect and retrieve real-time streaming data from the brokers. In addition to these, we used Python and Flask to create a Python API which is used to communicate the external services or third-party libraries. To represent/store/process digital twin metadata, we have used Azure's Digital Twin Description Language (DTDL).

To act as a generic integrator for multiple twins, we have extended Azure DTDL so that users can upload their digital twin hierarchies as flexible as possible. The figure below shows the metadata information of an example digital twin in Twinaide. In order to create the JSON structure that defines the digital twin in a flexible manner, a parent-child relationship has

been created. The data source configurations of the sensors and file information for 3D-display can also be embedded into the the JSON structure which then will be automatically processed by Twinaide.

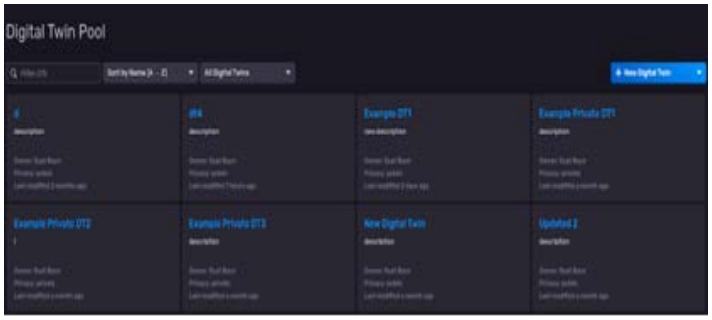
(1) ObjectId("623308c17f89475ac6e2689b")	{ 15 fields }	Object
_id	ObjectId("623308c17f89475ac6e2689b")	ObjectId
version	1.0	String
privacy	public	String
id	5fdf1319-233b-4ea6-8c40-fd7ff84623ee	String
name	dt5	String
description	description	String
displayName	d	String
owner	ObjectId("623080f458d15cb6459112a1")	ObjectId
location	{ 3 fields }	Object
country	Turkey	String
city	Bursa	String
address	Dumplupinar	String
type	Factory	String
manufacturer	{ 2 fields }	Object
name	company	String
website	http://company.com	String
children	[2 elements]	Array
[0]	{ 6 fields }	Object
id	pl1	String
type	ProductionLine	String
displayName	Production Line 1	String
description	lorem ipsum	String
name	pl1	String
children	[1 element]	Array
[1]	{ 6 fields }	Object
id	pl2	String
type	ProductionLine	String
displayName	Production Line 2	String
description	lorem ipsum	String
name	pl2	String
children	[1 element]	Array
createdAt	2022-03-17 10:09:05.738Z	Date
updatedAt	2022-03-30 10:56:31.396Z	Date
__v	0	Int32

The User Interface and Flow



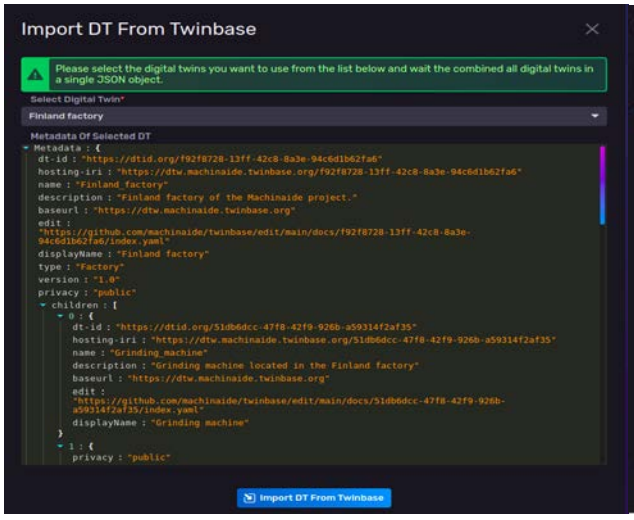
The users of the integrator software will first pass an e-mail and password-based authentication mechanism that has been developed for role-based access and login within the application. The user must first register in the system by entering their information on the sign-up page. Then, they can log in to the system by entering their email and password information on the sign in page. The sign in and sign up pages are shown on the left.

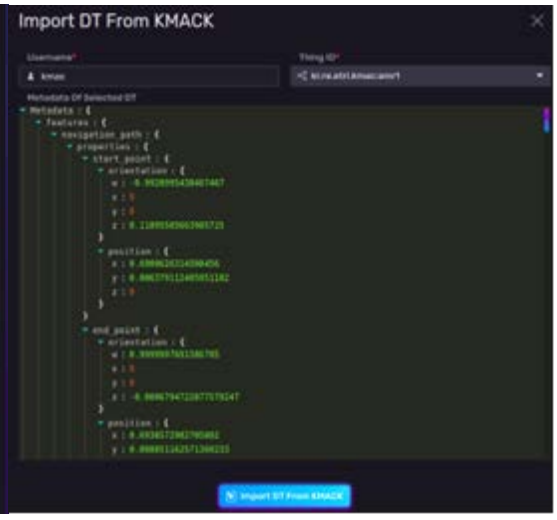
Once the users log in, they can import the metadata of a new digital twin to the integrator via its JSON structure either by downloading it from a file or by coping/pasting it via browser. The user can also view/edit the JSON document via browser with guidance from the interface. The import screen is shown below.



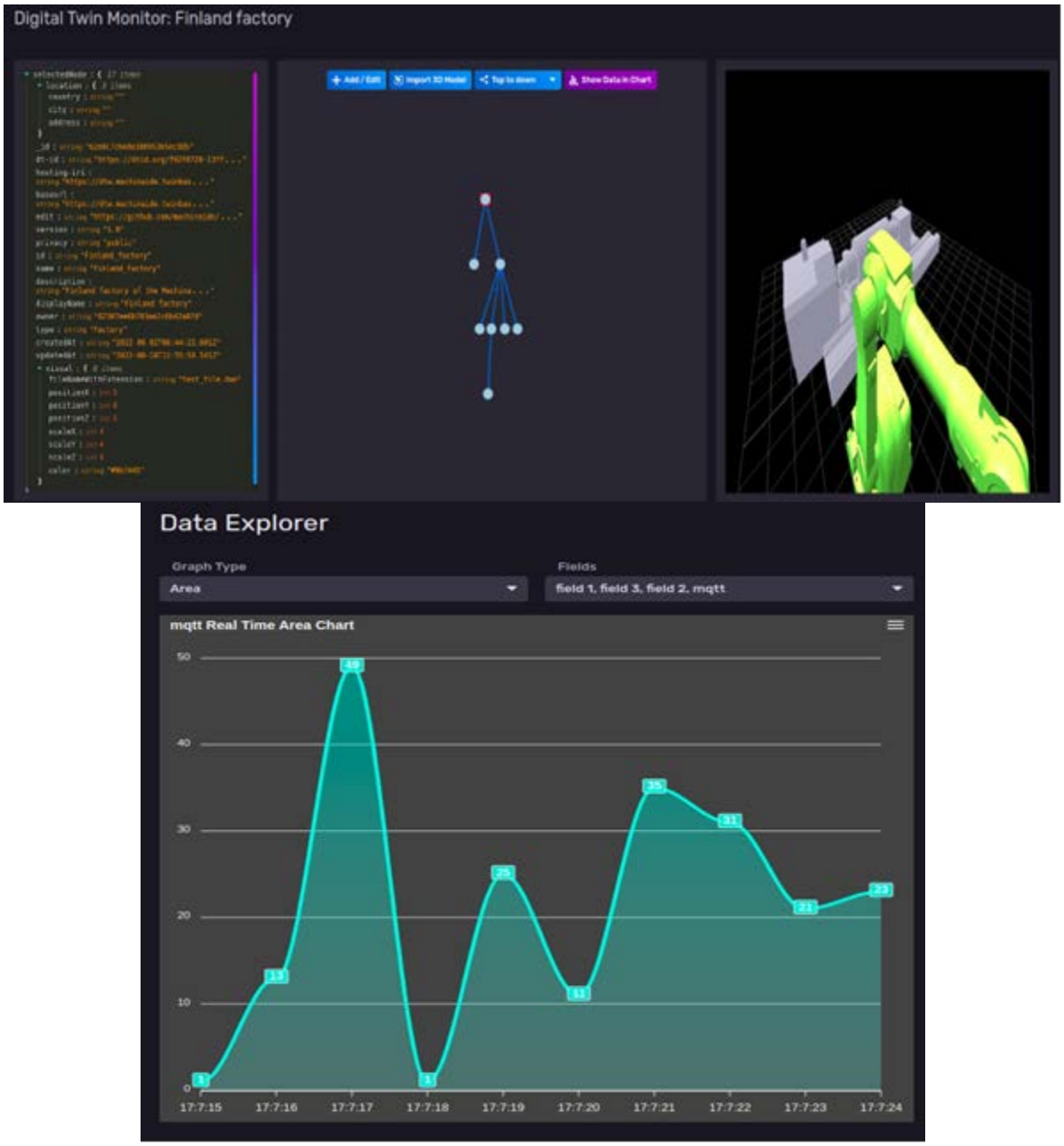
Once the users log in, they can import the metadata of a new digital twin to the integrator via its JSON structure either by downloading it from a file or by coping/pasting it via browser. The user can also view/edit the JSON document via browser with guidance from the interface. The import screen is shown on the left.

In addition to basic copy-paste, as an integrator of DTW, Twinaide currently supports exporting twins from Twinbase and KMAC. The users currently can pull data from these systems. However, more digital twin document servers can be integrated by using the same approach we have used. The following figures below show the screens for DTD export from Twinbase and KMAC, respectively.





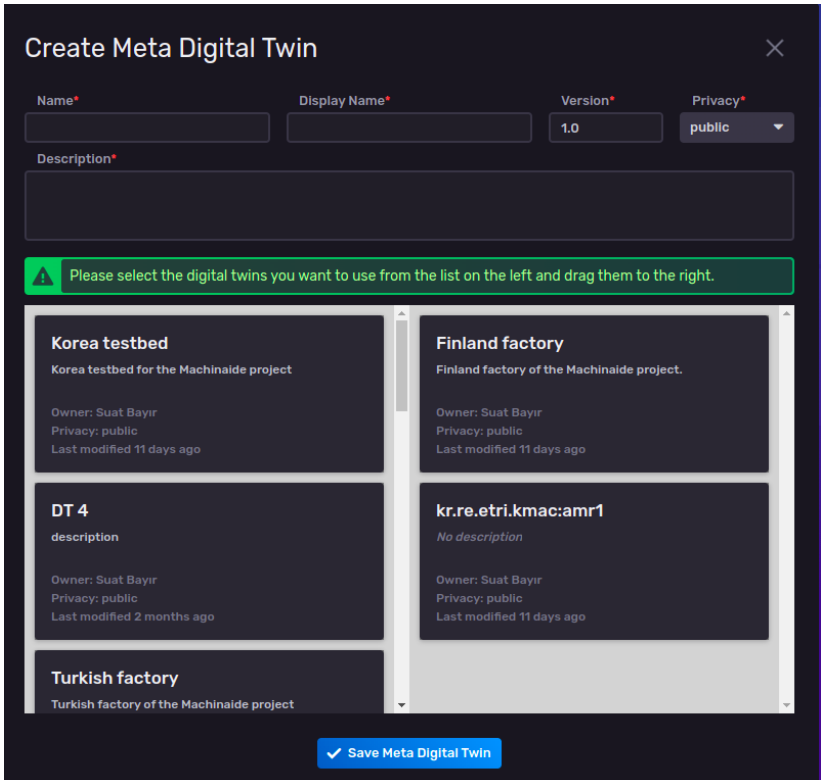
Twinaide allows the user view/monitor a digital twin via browser. The digital twin monitor page consists of three components. On the left, the metadata information of the node selected on the graph appears on a JSON editor and users can update and delete information from that component. In the middle, the hierarchical structure of the digital twin appears on a graph. Users can change the layout format of this graph. The component on the right displays the 3D visual information of the selected node on the graph. By means of the button appearing on the graph, users can upload 3D model files to the system and match them with the relevant element.



After the user selects a node of the field type on the graph, s/he can focus on the sensor data of this field streaming from 3rd party brokers and display them on the graphs by means of a modifiable chart as shown on the left. The stream process can be stopped and resumed from the screen. In addition, the type of the chart can be changed from top-left and multiple fields streaming from different sensors/twins can be displayed.

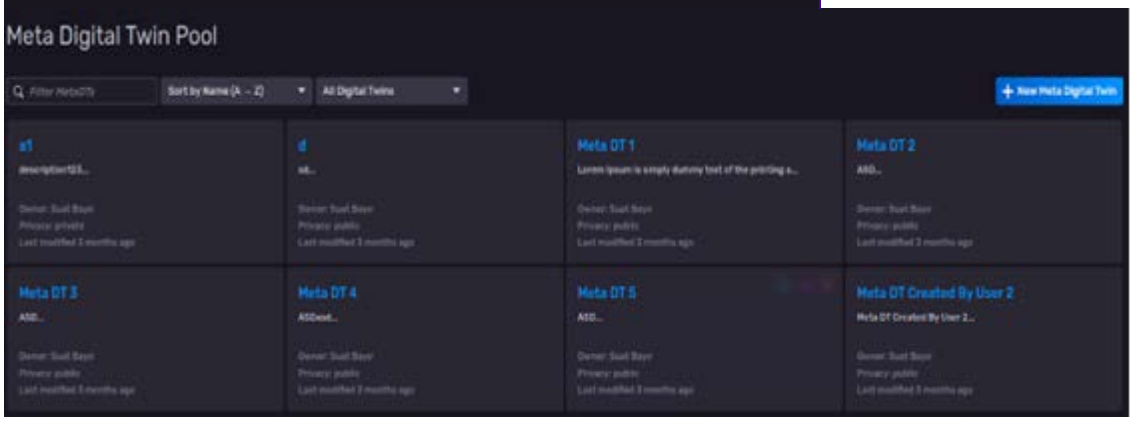
Creating and Managing Meta Digital Twins

As an integrator, Twinaide allows the users to combine multiple Digital Twins and create a Meta Digital Twin (of the virtual factory). To do this, the twins imported to the system can be combined and integrated into a single meta digital twin by using the screen on the left. All the metadata and relevant information can be inserted/edited through the browser. As a small



note, a user can only view/add the digital twins with the appropriate visibility behavior. That is only the public twins and the ones private to the user can be combined within the meta twin.

Similar to the Digital Twin Pool screen, all the meta digital twins registered to the system can be seen through the Meta Digital Twin Pool screen as shown below.



3.3.2. Data Querying and Visualization

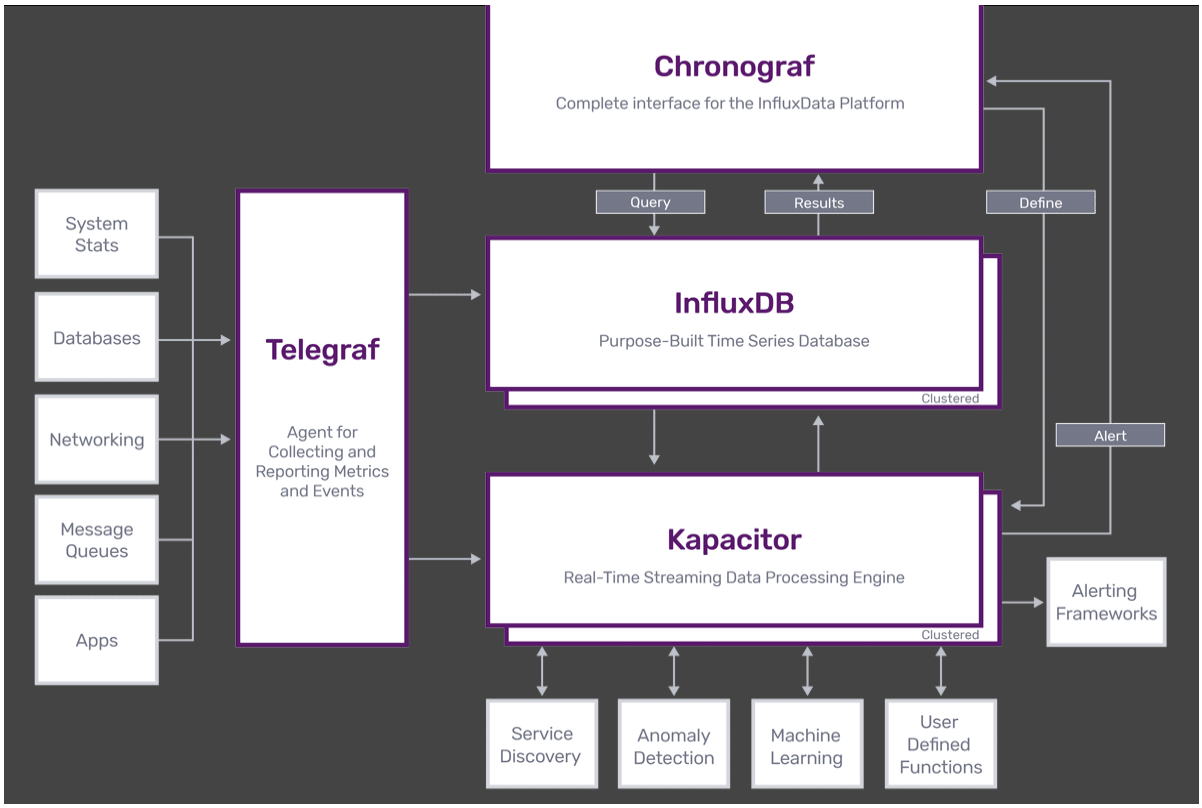
3.3.2.1. Streaming DB – Influx and Flux

InfluxData is an open source time series database built from the ground up to handle high speed write and query loads. It consists of an architecture called TICK Stack (Telegraf, InfluxDB, Chronograph, Kapacitor). InfluxData is a dedicated high-performance data store written specifically for timestamped data and particularly useful for use cases such as DevOps monitoring, IoT monitoring, and real-time analytics. By configuring InfluxDB to store data for a certain period of time and automatically delete unwanted data from the system, storage capacity savings can be achieved on servers. InfluxDB also provides an advanced query language called Flux to query and manage data.

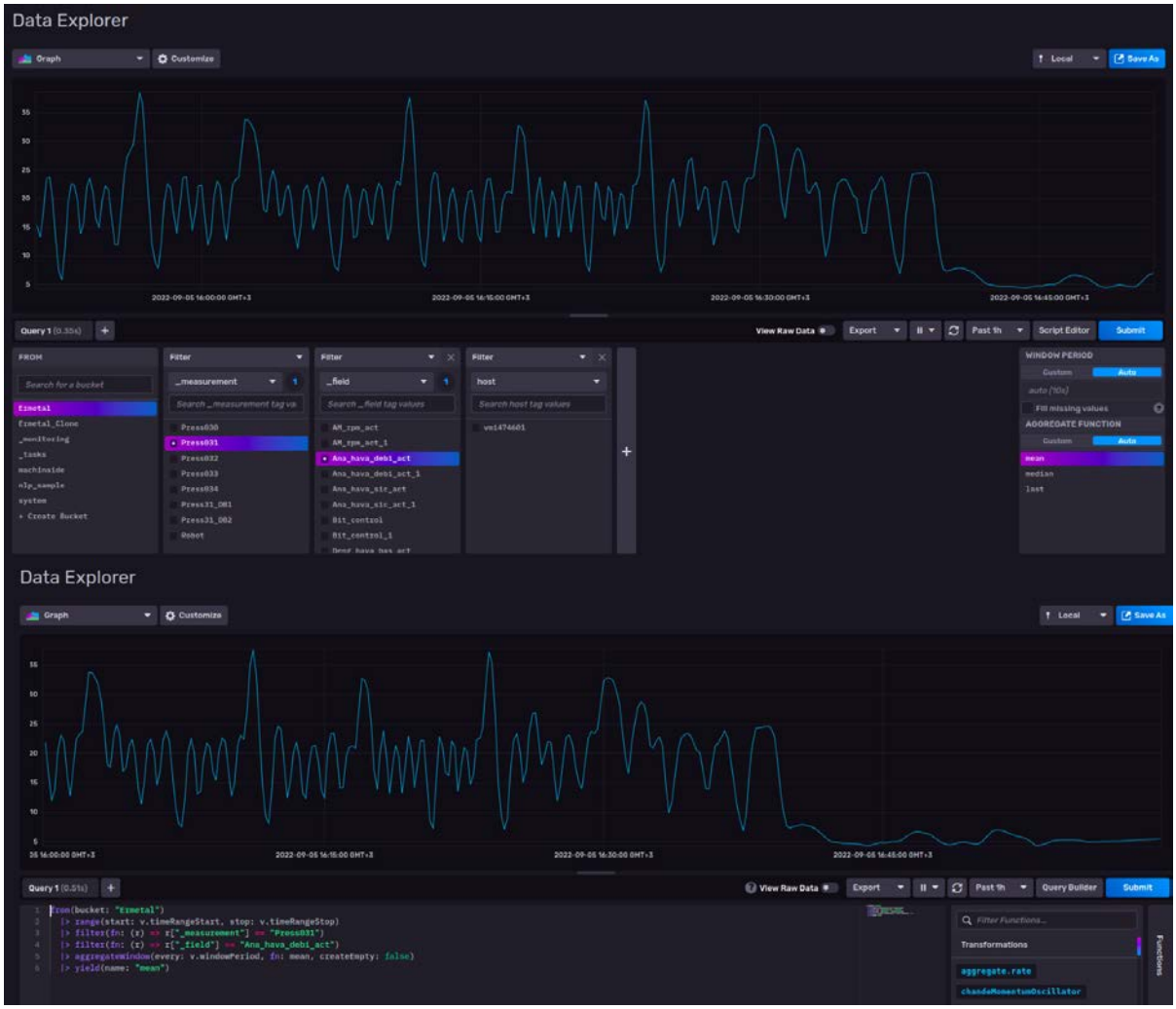
- Telegraf is a plugin-oriented server tool for collecting and reporting metrics/data. Telegraf plugins obtain various metrics directly from the systems they work on, metrics from third-party APIs, and even listen for metrics via a Kafka consumer service. Telegraph, in other words, is a data collection tool.

- Chronograph is the stack's admin user interface and visualization engine. Monitoring data simplifies the creation and maintenance of alerts. It is easy to use and allows real-time visualization of data. Chronograph includes ready-made templates and libraries that allow you to quickly create dashboards, alerts and rules.
- Kapacitor is a native data processing engine. It can handle both streaming and batch data from InfluxDB. Capacitor allows specifying user-defined functions to handle alerts with dynamic thresholds, match metrics for patterns, calculate statistical anomalies, and perform alert-based actions such as dynamic load balancing.

Within the scope of the project, it was decided to use InfluxDB due to the above-mentioned features. For this purpose, source codes were downloaded from InfluxDB's github page and installed on the project's server. It is aimed to store the numerical data coming from the sensors on InfluxDB. Since InfluxDB is an open-source software, we can easily make changes on the Chronograph, which is the user interface, in line with our own needs.

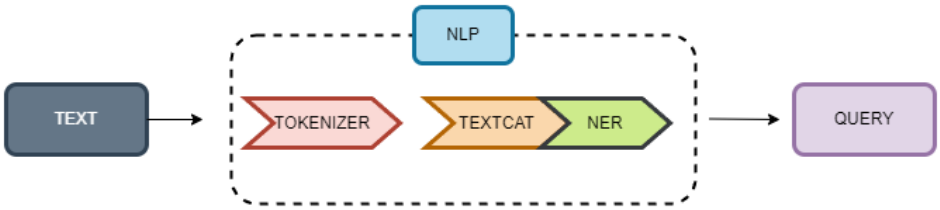


With Flux, the data query language of InfluxDB, queries can be created and matching data can be retrieved. Below is the data explorer page of the chronograph. users can set the data they want to display via this tool. As an additional option, users can filter the data with the flux query language.



3.3.2.2. NLP-based data query

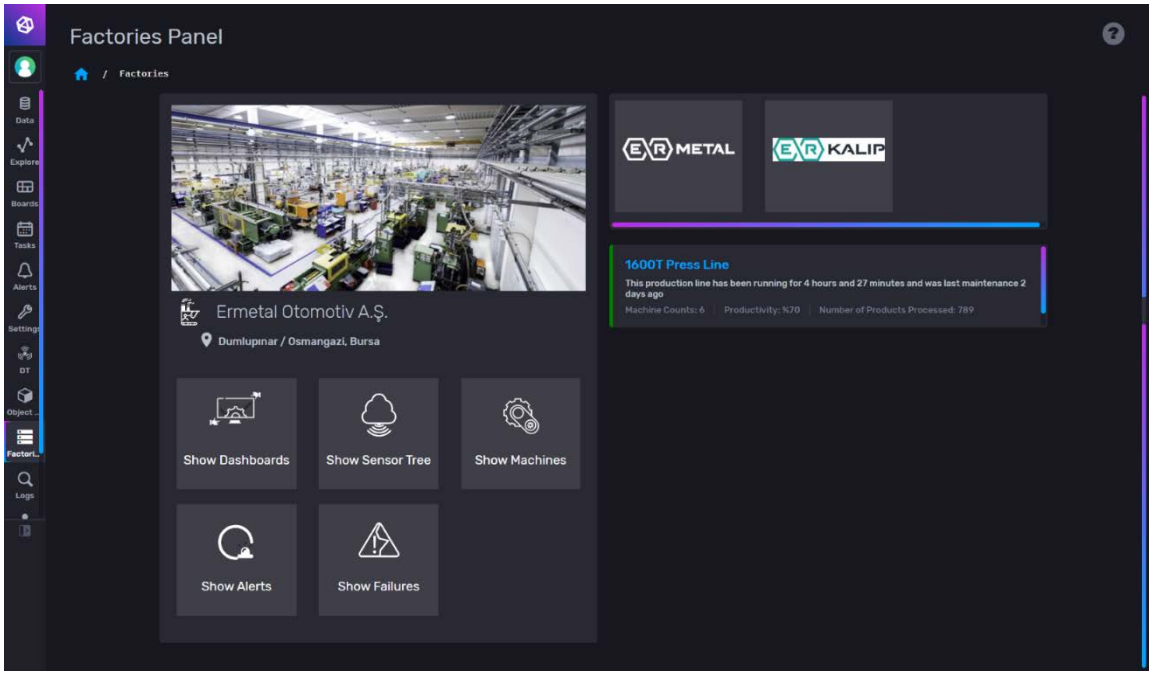
To let users query database using natural language we use spaCy library. Using spaCy's "textcat" pipeline (text categorizer) first we decide which database to query. We use InfluxDB for sensor data and MongoDB for metadata.



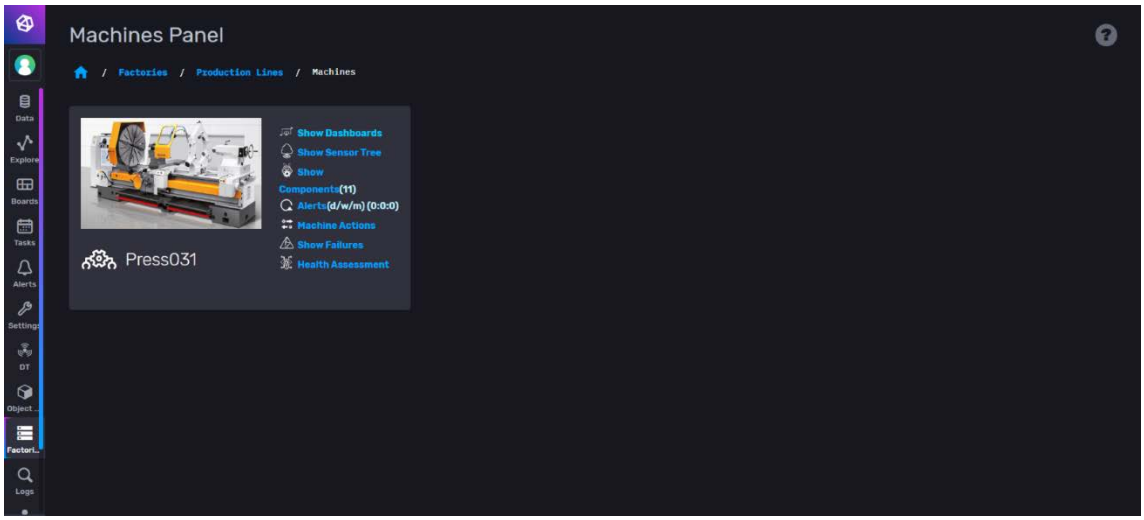
After deciding on the database, using "NER" pipeline (named entity recognizer), we decide machine, component, sensor and the time range to build the flux query or MongoDB query. After creating queries, we query the database and return the results to the user in overlay.

3.3.2.3. Dashboards

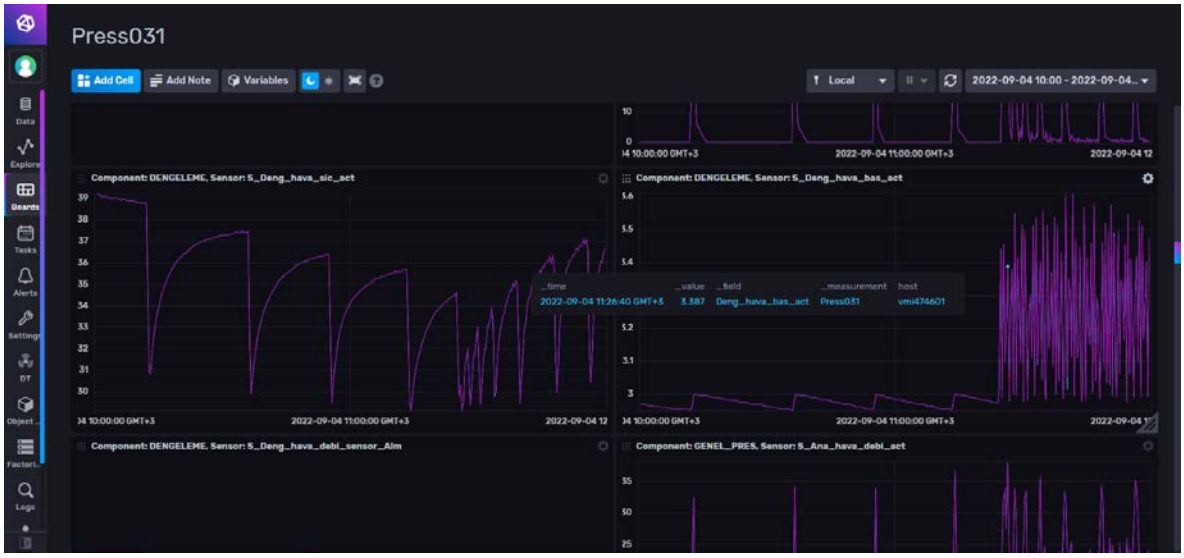
Users can create their own dashboards or we have, also, automatically created dashboards for assets. In these dashboards users can view sensors values in different graphs such as, single stat, scatter, histogram, line or they can see the values in table format.

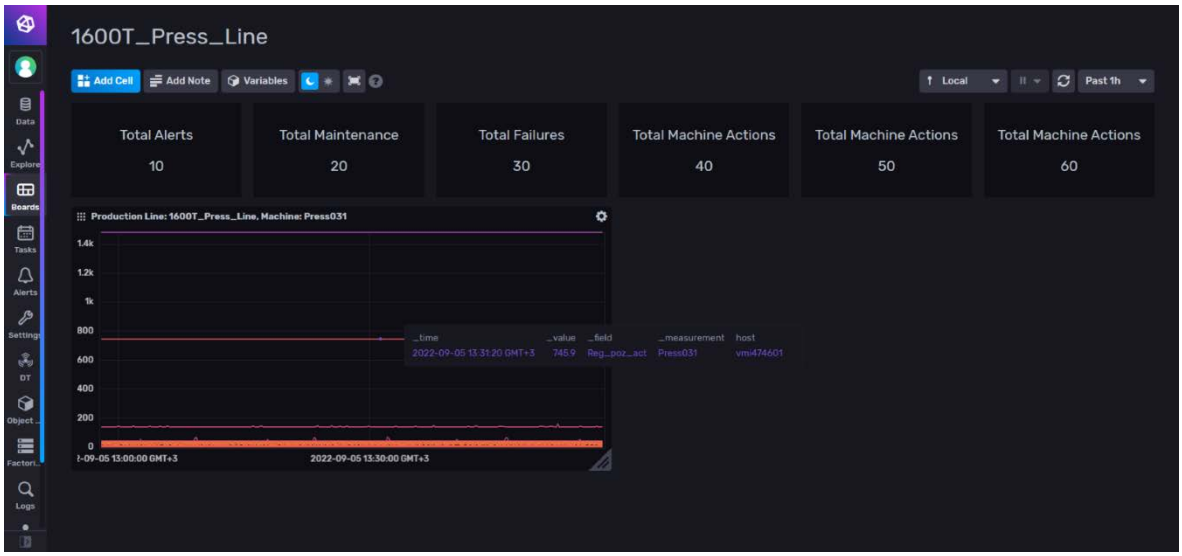


In the above screen shot, we have factory card. By clicking “Show Dashboards” button users can see the automatic dashboard we created for the factory level.

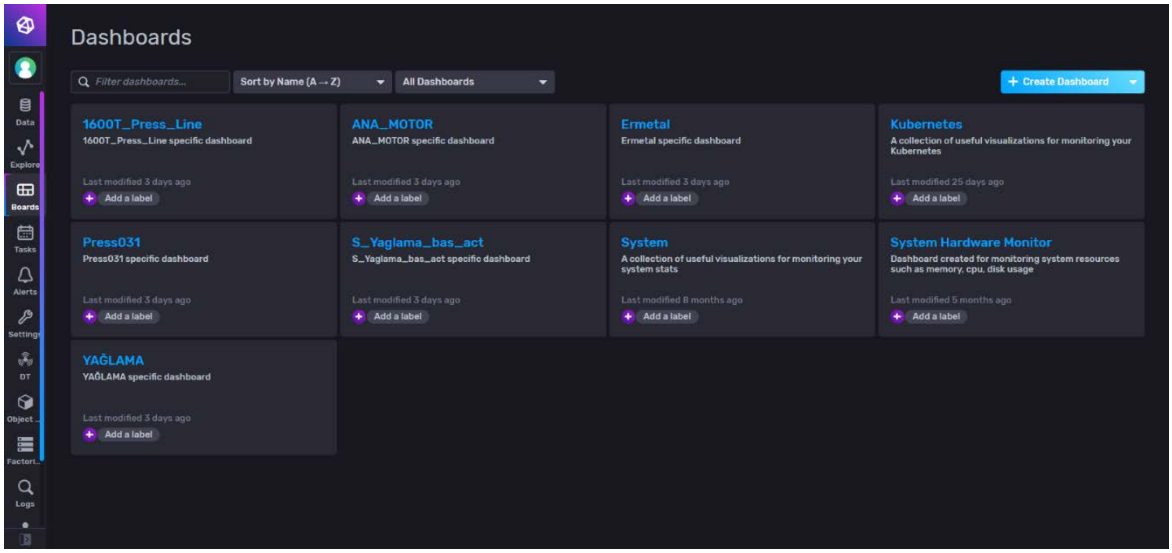


Here we have “Machines Panel” where we list all the machines under the selected production line. Each machine represented as cards. In these cards, by clicking “Show Dashboards” text users can view the created automatic dashboard for the selected machine.

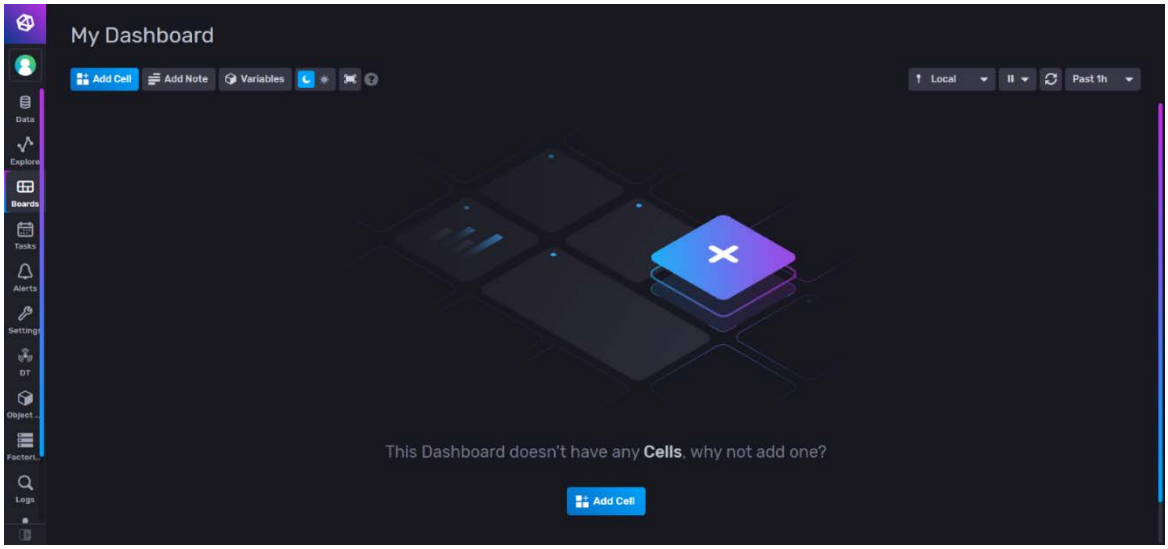




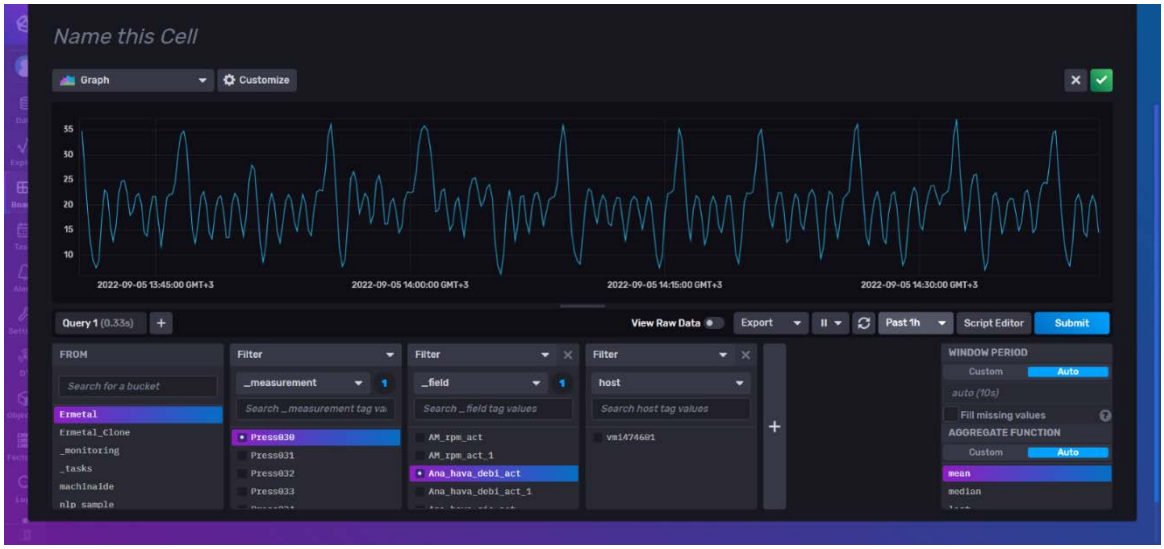
In the above screen shots, we have automatically created dashboards. These dashboards show the sensors values and group them based on the selection level.



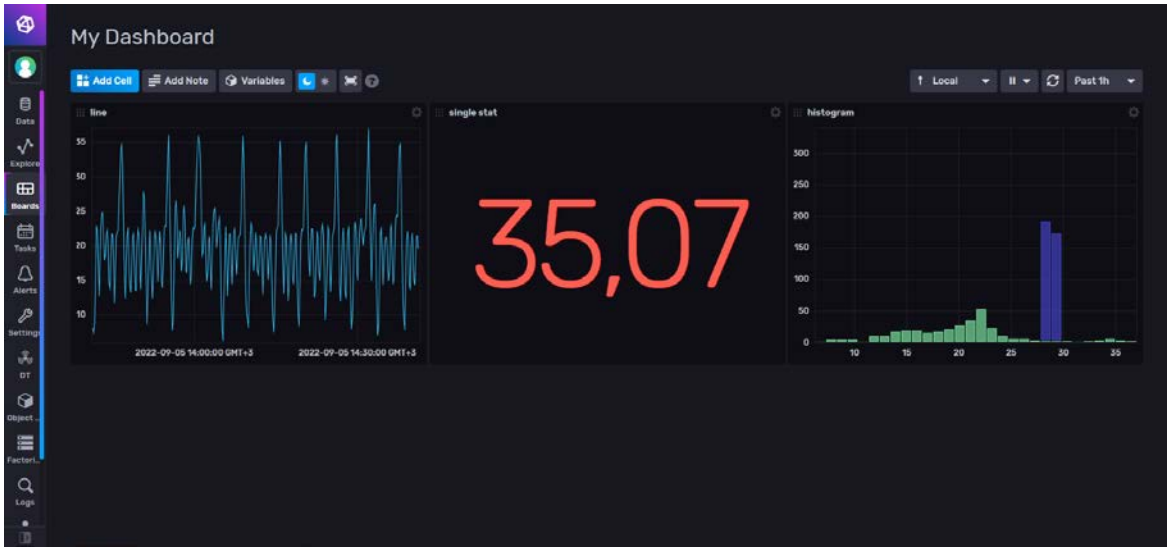
In the “Dashboards” page, by clicking the “Create Dashboard” button users can start to create their own dashboards.



In the new dashboard page, by clicking “Add Cell” button, users can create new graphs.



In the “Add Cell” overlay users can select values they want to show, the graph style, the colors etc.



In the above image, we have a custom dashboard with different graphs.

3.3.3. ML-based from Processing

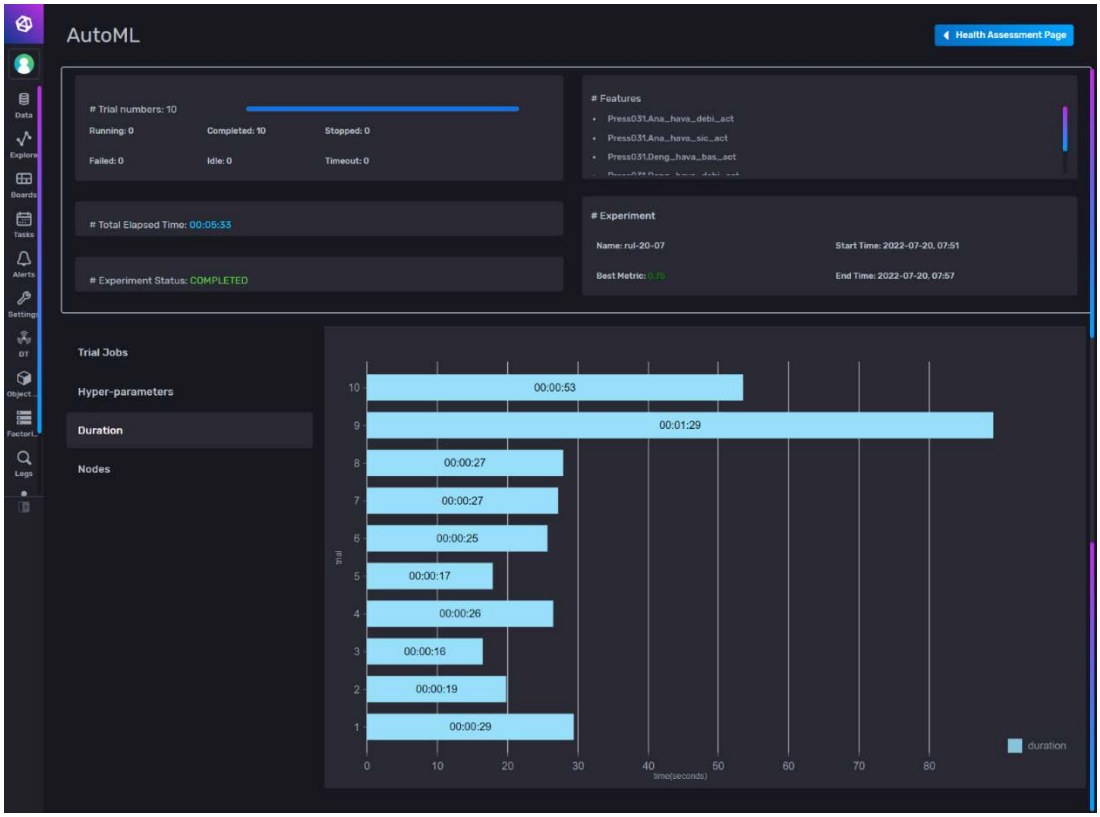
3.3.3.1. Machine Health

For each machine, the frontend has a “Health Assessment” page where users can see all models related to that machine and their logs. For the anomaly detection models, users can see model logs in a data graph, go to a selected timestamp and give feedback to the model logs. Also, users can add any missing anomaly by clicking the graph in selected timestamp.

All the created models are shown under “ML Models” section. Models are represented by cards and their last log is shown in the cards. Cards have buttons that let users to start/stop the model, view AutoML process results, view the model’s logs in a graph, view the model’s

performance report and delete the model.

The users can create new machine learning models using “Add Model” button. They just need to select the task, the data they want to use and the failure records to start a ml process. If users want to change advanced settings of AutoML process, they can change them in the “AutoML Settings” overlay.

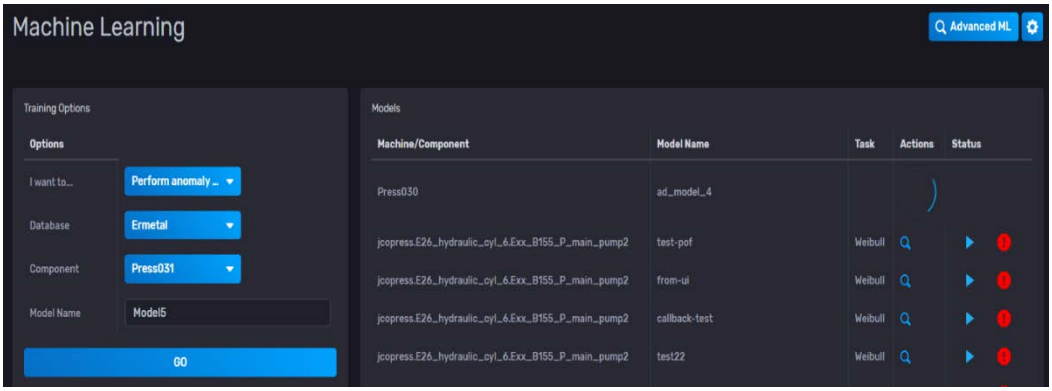


3.3.3.2. Anomaly detection

Anomaly detection tasks are controlled through two different pages. First one is the advanced machine learning page where the users have the option to control training parameters and the data channels to be used in training as well as the time range the data will be coming from. The other way is to use the AutoML page designed for users with minimal machine learning knowledge. On this page users select which components they want to perform these tasks on and then the optimal model is trained with respect to the most recent data. This page is also used for Remaining Useful Lifetime Estimation (RULE) and Probability of Failure (PoF) calculation both of which are detailed later on in the document.

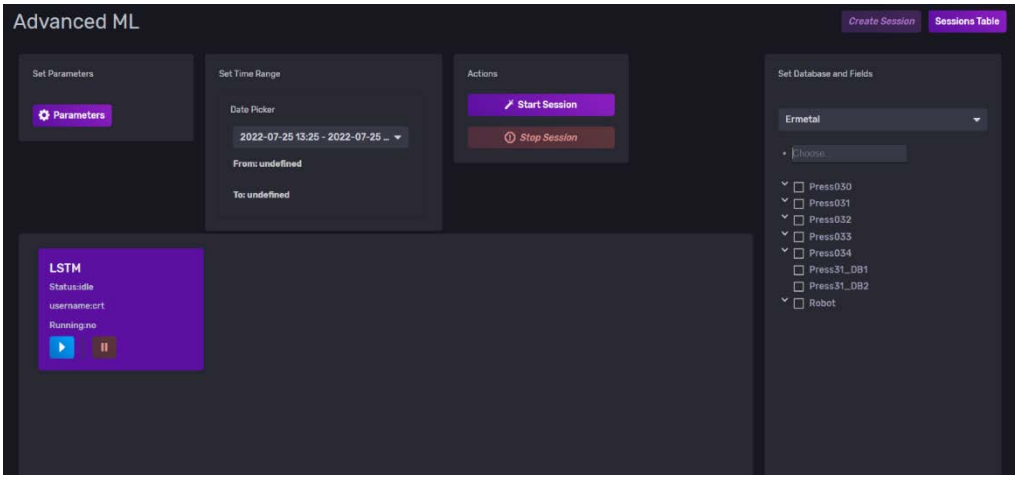
Training Page

The Machine Learning Page is designed for the users with minimal knowledge of the training process for an ML model. The users can select which type of task they want their model to handle, use-case-specific parameters for each task, the components they want to use, and then train their models. When a training job is issued through this page, an automated hyperparameter tuning is performed in the background to be able to train the best possible model with the given resources. These trained models then show up in a table right by the control panel from where the users can engage with the models by enabling/disabling them. They can also observe the metadata for the training process the model went through.

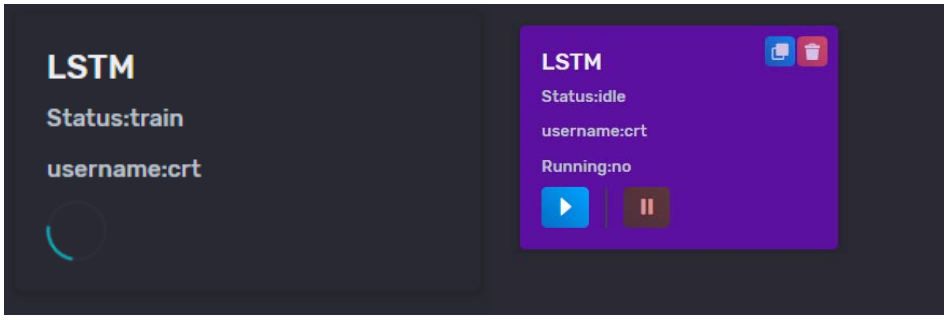


Advanced Training Page

The Advanced Training Page is used for training multiple ML models suited for various jobs and it provides ultimate control over the training parameters. "*Database*" dropdown is for selecting the database name. When the database is selected, the components contained in the database are listed in the section below. All of them are top-level nodes of a dropdown tree, where their children are the sensors installed on those components. The data provided in the given time range by the selected sensors are used in the training process. The "*Selected Models*" button forwards the user to a table where they can interact with the models they have trained and accepted as valid. The "*Train*" button starts the training process.



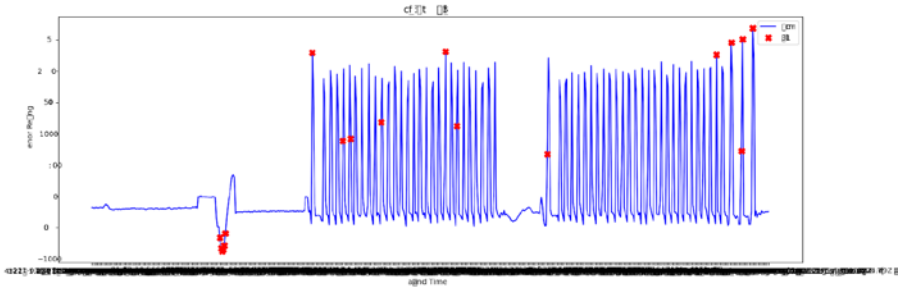
When the training process starts and the related collections are updated, cards representing the models show up. These cards have buttons showing up when hovered over and one shows the parameters being used in training and the training progress (if is applicable) and the other is a "*Cancel*" button that lets the user cancel the training process for that model. After a model is trained, the card is updated with information that could help the user decide the quality, and two buttons that would enable the user to either accept or discard the trained model. Accepting the models enables further interactions with the model which can be accessed from the "*Selected Models*" button. Accepted models can be rejected to be idle again. The "*Discard*" button deletes the model.



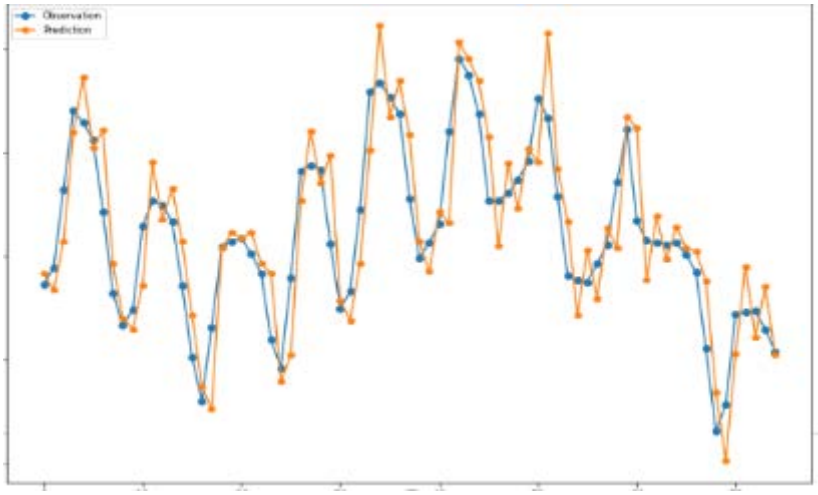
Anomaly Detection Methods

We have used various anomaly detection techniques from literature:

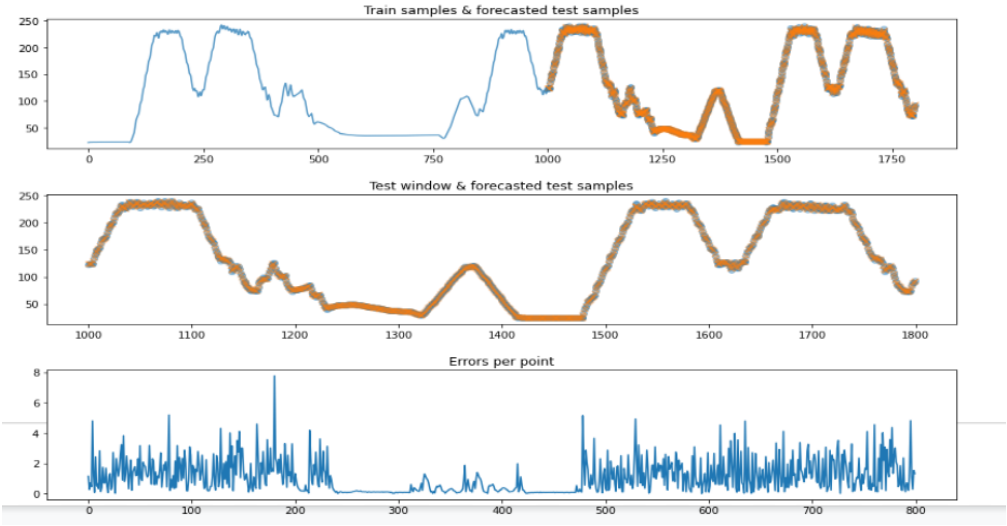
- ***The Mahalanobis Distance*** is a multivariate distance metric between a vector and a distribution. Thresholding the distance between the extracted time-series vectors and the mean value of the overall set is the main idea behind while detecting the anomalies. No training is involved; just the necessary statistical values are calculated which makes this approach lightweight in terms of required resources and the ease of implementation. Our approach is to define a distribution using two or more features and calculate the mean value of this set and use these values to threshold the distances.
- ***Isolation Forests*** are ensembles of isolation trees. An isolation tree is built upon the assumption that anomalous points are rare and few, which means they are susceptible to isolation. When a tree is built, the points which have smaller path lengths have a higher anomaly score. A forest made of these trees is used to choose the anomalous points based on the average anomaly score of a point generated by the trees. We used the *Isolation Forest* implementation found in the *ensemble* module of the widely known *scikit-learn* library.



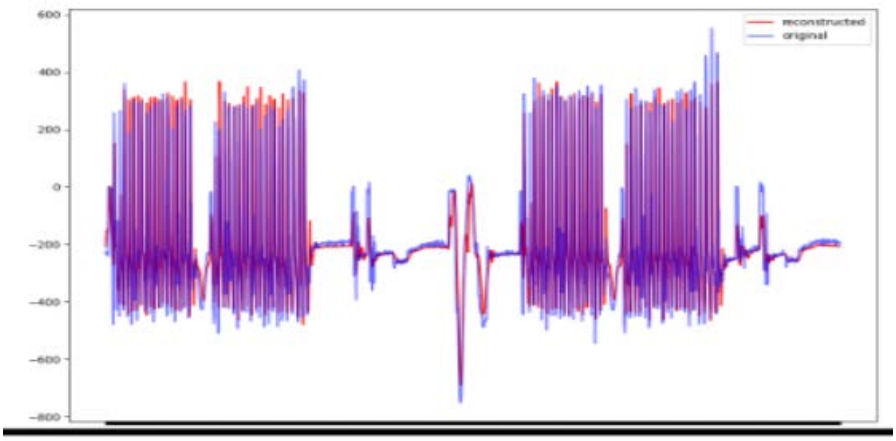
- ***The LSTM network*** is a type of neural network that is good at utilizing the unique aspects of time-series data such as seasonality, non-stationarity, and long and short-term dependencies within data. These networks are used to forecast time-series data to compare the predicted (expected) values to the real-world observations and if the difference between them exceeds a certain threshold an alert is raised. In the picture below, an example of how a finely-tuned LSTM network can forecast sequential data is shown. This picture also implies the significance of the correct hyper-parameter selection.



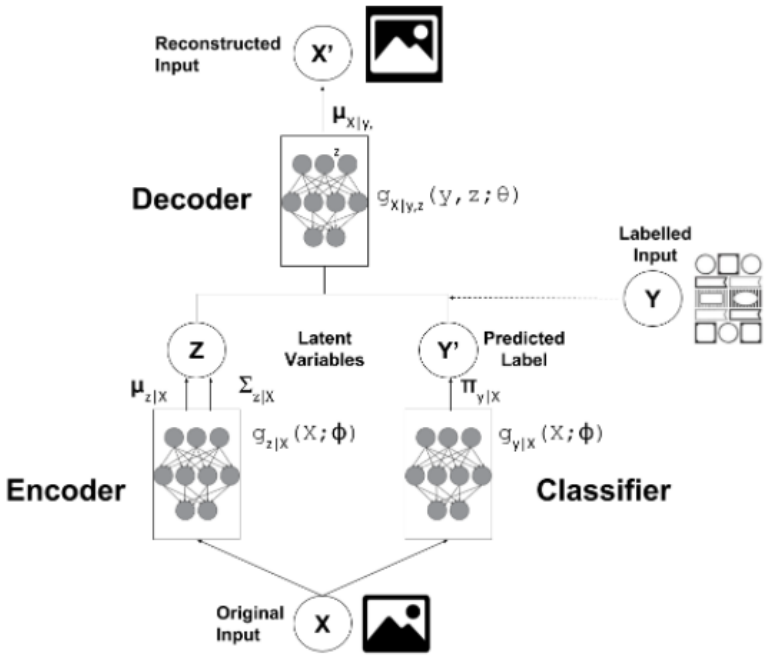
- ***ARIMA*** stands for ***AutoRegressive Integrated Moving Average***. It is a statistical model used to forecast time-series data using past observations. It has a lightweight preparation phase and is controlled by three parameters. One drawback is that the data needs to be stationary because the model involves linear regression. This means before using ARIMA the data should be transformed in a way that it becomes stationary. The anomaly detection process is the same as when LSTM networks are used, meaning points with high forecast errors are flagged as anomaly points. The following image shows how ARIMA performs when parameters are selected correctly, together with the error values and how they behave with respect to the sequential nature of the data.



- ***Variational Autoencoders*** are networks that try to learn compressed representations of original data and reconstruct the data using the learned representations. They are a subclass of autoencoders where instead of learning a direct representation, they learn distributions and reconstruct the data sampling from these distributions. If the margin between the reconstruction and the observation is larger than a set threshold an alert is raised. See the figure on the right for the original and reconstructed data streams via variational autoencoders.



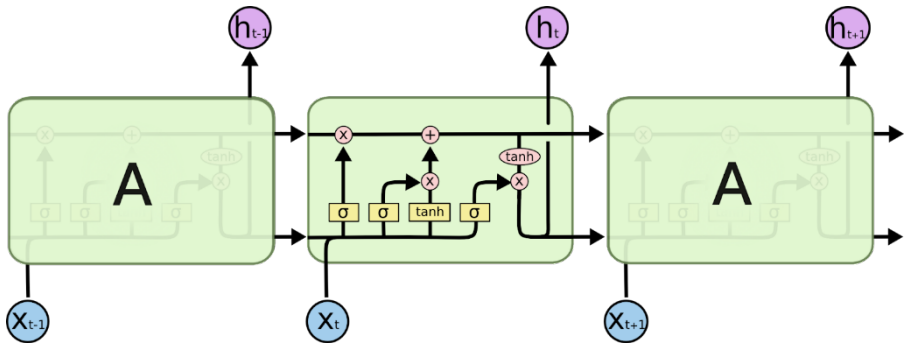
- ***Semi-supervised learning*** is a term used to define methods where unlabeled data is used together with supervised learning techniques. These approaches are very suitable for anomaly detection problems where most of the time unlabeled data is found a lot more than labelled data but where even a small amount of labelled data can provide valuable information for the task. Our approach to semi-supervised learning is based on Variational Autoencoders. We train two different encoders, one of whose output is a regular latent representation obtained from unlabelled samples, and the other label predictions for those samples. The decoder part of the network takes as input these two encoder outputs together with known ground-truth labels, and produces a reconstruction of the input using the knowledge. (https://bjlkeng.github.io/posts/semi-supervised-learning-with-variational-autoencoders/) The image below is a visual representation of the process detailed above, taken from Brian Keng's GitHub page shared.



3.3.3.3. Remaining Useful Lifetime Estimation (RULE)

The Remaining Useful Lifetime is a subjective estimate of the number of remaining cycles that an item, component, or system is estimated to be able to function in accordance with its intended purpose before warranting replacement. We have used deep learning techniques to obtain a cost-saving predictive maintenance schedule. To study the effectiveness of the used techniques, we used Microsoft Azure Team's Predictive Maintenance implementation as our template (published on 31 July 2017).

Among the deep learning methods, we have learned that Long Short Term Memory (LSTM) networks are especially appealing to the predictive maintenance domain due to the fact that they are very good at learning from sequences. Below is a figure of a simple LSTM node.



To test our implementation, we take Microsoft Azure's Deep Learning for Predictive Maintenance template that they used to predict the remaining useful life of aircraft engines using the Turbofan Engine Degradation Simulation Data Set from NASA.

Preparing Data: Data is labeled based on the selected window size. The last *w* days of the engine data are labeled as 1 and 1 and the others are labeled as 0. Also, RULE labels, as cycles, are added to data. As the last step, the data is normalized before the modeling phase.

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21	RUL	label1	cycle_norm
157	1	158	0.367816	0.583333	0.0	0.0	0.503012	0.562023	0.469953	0.0	1.0	0.537842	0.454545	0.123351	0.0	0.494048	0.466951	0.500000	0.136185	0.430550	0.0	0.500000	0.0	0.0	0.472868	0.545706	34	0	0.434903
158	1	159	0.465517	0.583333	0.0	0.0	0.506024	0.403314	0.643484	0.0	1.0	0.375201	0.348485	0.118146	0.0	0.541667	0.417910	0.514706	0.142791	0.444017	0.0	0.500000	0.0	0.0	0.480620	0.495443	33	0	0.437673
159	1	160	0.465517	0.166667	0.0	0.0	0.674699	0.427513	0.604826	0.0	1.0	0.325282	0.469697	0.106300	0.0	0.589286	0.379531	0.500000	0.117350	0.455560	0.0	0.666667	0.0	0.0	0.294574	0.535073	32	0	0.440443
160	1	161	0.545977	0.583333	0.0	0.0	0.539157	0.504905	0.591830	0.0	1.0	0.475040	0.424242	0.118146	0.0	0.547619	0.379531	0.441176	0.139488	0.444786	0.0	0.500000	0.0	0.0	0.403101	0.555095	31	0	0.443213
161	1	162	0.471264	0.833333	0.0	0.0	0.584337	0.461740	0.695982	0.0	1.0	0.455717	0.378788	0.101947	0.0	0.559524	0.373134	0.470588	0.122975	0.679492	0.0	0.500000	0.0	0.0	0.496124	0.358465	30	1	0.445983
162	1	163	0.517241	0.166667	0.0	0.0	0.493976	0.643122	0.656638	0.0	1.0	0.412238	0.454545	0.123441	0.0	0.619048	0.343284	0.470588	0.124471	0.723355	0.0	0.416667	0.0	0.0	0.395349	0.342033	29	1	0.448753
163	1	164	0.528736	0.333333	0.0	0.0	0.590361	0.608677	0.582546	0.0	1.0	0.330113	0.409091	0.122605	0.0	0.505952	0.413646	0.588235	0.127413	0.597922	0.0	0.500000	0.0	0.0	0.372093	0.389257	28	1	0.451524
164	1	165	0.557471	0.833333	0.0	0.0	0.466867	0.566601	0.436192	0.0	1.0	0.473430	0.439394	0.077896	0.0	0.619048	0.426439	0.529412	0.137217	0.643709	0.0	0.416667	0.0	0.0	0.348837	0.440072	27	1	0.454294

Modeling: Since we have the time-series data and have lots of sensor values, using a deep learning algorithm, i.e., LSTM, has certain benefits for us.

- These networks can automatically extract the right features from the data, therefore eliminates the need for manual feature engineering
- LSTMs ability to remember from long-term sequences

We used 2 LSTM layers followed by Dropout layers to control overfitting. The final layer is a Dense output layer with a single unit and sigmoid activation since this is a binary classification problem.

The number of units in the LSTM layer is chosen by our AutoML module. With the AutoML module, we create multiple models with different hyperparameters like units of LSTM

layers, batch size, or number of epochs. This module's main purpose is to let users free from adjusting hyperparameters for the best model outcome. Users don't have to deal with technical stuff. The AutoML module finds the hyperparameters for the best model and creates that model for users.

	id	cycle	s1	s2	s21	RUL	label1	cycle_norm
157	1	158	0.0	0.503012	0.545706	34	0	0.434903
158	1	159	0.0	0.506024	0.495443	33	0	0.437673
159	1	160	0.0	0.674699	0.535073	32	0	0.440443
160	1	161	0.0	0.539157	0.555095	31	0	0.443213
161	1	162	0.0	0.584337	0.358465	30	1	0.445983
162	1	163	0.0	0.493976	0.342033	29	1	0.448753
163	1	164	0.0	0.590361	0.389257	28	1	0.451524
164	1	165	0.0	0.466867	0.440072	27	1	0.454294

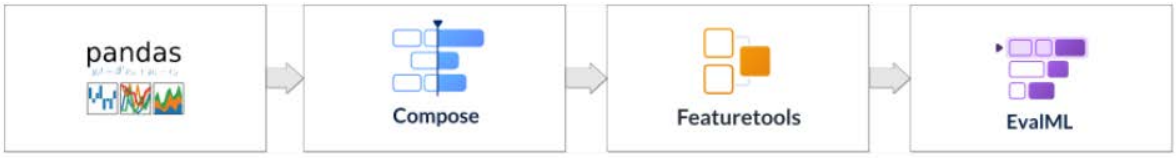
Results on NASA Data: We have a test dataset in NASA data as well. Using this data, we get the following results:

	Accuracy	Precision	Recall	F1-score
LSTM	0.967742	0.892857	1.0	0.943396

In NASA data, our model nearly works perfectly. Recall indicates that the model finds all failure instances. Thus, we are not missing any failures. On the other hand, the precision score indicates the model label some non-failure cycles as a failure.

From these metrics, we can deduce that our model labels cycles as failures more than needed. This way it never misses a failure but gives false alarms sometimes. Depending on the user's objective, not missing any failure but getting some false alarms from time to time can be ideal since these false alarms actually warn the user of a future failure so they can do some maintenance tasks.

In the regression side of this problem, we want to get the number of time cycles left for the selected asset. For the implementation, we used Compose's, open-source ML tool for automated prediction engineering, tutorial for predicting turbofan engine degradation as our template. We implemented a model using Compose, Featuretools, and EvalML. Compose is for structuring data, Featuretools is for feature creation and extraction (feature engineering) and EvalML is for evaluating machine learning pipelines and finding the most optimal one in an AutoML process.



Results on NASA Data: Using the same NASA dataset above, we trained our model and tested its results. Then we get some metric based on the results.

	id	rul	truth	diff	abs_diff	diff/truth
0	47	135.078119	135	-0.078119	0.078119	0.057866
1	43	58.758704	59	0.241296	0.241296	0.408976
2	46	46.731711	47	0.268289	0.268289	0.570828
3	31	8.343472	8	-0.343472	0.343472	4.293404
4	90	27.529558	28	0.470442	0.470442	1.680149
...	...	...	...	...	...	...
95	1	167.549328	112	-55.549328	55.549328	49.597614
96	12	65.520739	124	58.479261	58.479261	47.160694
97	89	75.595901	136	60.404099	60.404099	44.414779
98	67	138.413491	77	-61.413491	61.413491	79.757780
99	10	30.855008	96	65.144992	65.144992	67.859366

100 rows × 6 columns

First, we sorted the results based on the difference in predicted and real data's time cycles. The sorted absolute differences are given on the right.

Here we see that some predictions are really close but we have predictions that are really off to the truth data. To get a better understanding then we split the result based on late (difference<0) and early (difference>0) failure predictions. We got 49 late predictions and 51 early predictions.

When we check the first and last batch of results of late predictions (on the left and right, respectively, below):

	id	rul	truth	diff	abs_diff	diff/truth
0	47	135.078119	135	-0.078119	0.078119	0.057866
1	31	8.343472	8	-0.343472	0.343472	4.293404
2	63	72.506384	72	-0.506384	0.506384	0.703311
3	96	137.783219	137	-0.783219	0.783219	0.571693
4	56	16.001360	15	-1.001360	1.001360	6.675732

	id	rul	truth	diff	abs_diff	diff/truth
44	79	112.589202	63	-49.589202	49.589202	78.713019
45	14	158.699887	107	-51.699887	51.699887	48.317652
46	87	169.047937	116	-53.047937	53.047937	45.730980
47	1	167.549328	112	-55.549328	55.549328	49.597614
48	67	138.413491	77	-61.413491	61.413491	79.757780

Similarly, the first and last batch of results of early predictions are:

	id	rul	truth	diff	abs_diff	diff/truth
0	43	58.758704	59	0.241296	0.241296	0.408976
1	46	46.731711	47	0.268289	0.268289	0.570828
2	90	27.529558	28	0.470442	0.470442	1.680149
3	73	130.344944	131	0.655056	0.655056	0.500043
4	66	12.996875	14	1.003125	1.003125	7.165182

	id	rul	truth	diff	abs_diff	diff/truth
46	93	31.338645	85	53.661355	53.661355	63.131006
47	51	60.029778	114	53.970222	53.970222	47.342300
48	12	65.520739	124	58.479261	58.479261	47.160694
49	89	75.595901	136	60.404099	60.404099	44.414779
50	10	30.855008	96	65.144992	65.144992	67.859366

After that, we calculated some metrics: average distance, minimum distance, maximum distance, mean squared error, root mean squared error, mean absolute error, mean absolute percentage error, first quartile, second quartile (median), third quartile.

predictions	avg_error	min_error	max_error	mse	rmse	mae	mape	q1	q2	q3
all	20.713991	0.078119	65.144992	737.325666	27.153741	20.713991	0.360596	6.359221	16.370499	33.116648
early	20.968502	0.241296	65.144992	768.070949	27.714093	20.968502	0.397589	7.379730	15.744499	33.116648
late	20.449092	0.078119	61.413491	705.325473	26.557964	20.449092	0.322094	6.258553	17.793615	37.106501

The average difference is 20 lifetime cycles whereas the median is 15-17 cycles. Looking at these results, we can say that the model's performance is not excellent. However, looking at the third quartile (~33) and the maximum error (~65) we can say that results are not so bad either. Some improvements are needed here.

	id	rul	truth	diff	abs_diff	diff/truth
0	47	135.078119	135	-0.078119	0.078119	0.057866
1	43	58.758704	59	0.241296	0.241296	0.408976
2	73	130.344944	131	0.655056	0.655056	0.500043
3	46	46.731711	47	0.268289	0.268289	0.570828
4	96	137.783219	137	-0.783219	0.783219	0.571693
...	...	...	...	...	...	...
95	52	65.173428	29	-36.173428	36.173428	124.735960
96	41	41.115567	18	-23.115567	23.115567	128.419815
97	35	-4.459217	11	15.459217	15.459217	140.538332
98	34	-6.296768	7	13.296768	13.296768	189.953834
99	81	-9.366743	8	17.366743	17.366743	217.084287

100 rows × 6 columns

Now let's look at the results based on absolute differences divided by truth data.

The worst results are different from the ones we first looked at. The first and last batch results of late predictions (left) and the first and last results of early predictions (right) are given below:

	id	rul	truth	diff	abs_diff	diff/truth
0	47	135.078119	135	-0.078119	0.078119	0.057866
1	96	137.783219	137	-0.783219	0.783219	0.571693
2	63	72.506384	72	-0.506384	0.506384	0.703311
3	31	8.343472	8	-0.343472	0.343472	4.293404
4	80	94.890254	90	-4.890254	4.890254	5.433616

	id	rul	truth	diff	abs_diff	diff/truth
44	79	112.589202	63	-49.589202	49.589202	78.713019
45	67	138.413491	77	-61.413491	61.413491	79.757780
46	100	36.935315	20	-16.935315	16.935315	84.676577
47	52	65.173428	29	-36.173428	36.173428	124.735960
48	41	41.115567	18	-23.115567	23.115567	128.419815

We then calculated some metrics for absolute_difference/truth data: average absolute_difference/truth, minimum absolute_difference/truth, maximum absolute_difference/truth, first quartile, second quartile (median), third quartile.

predictions	avg_error	min_error	max_error	q1	q2	q3
all	36.059634	0.057866	217.084287	10.418667	25.569810	44.752041
early	39.758910	0.408976	217.084287	9.426975	24.721887	45.455625
late	32.209367	0.057866	128.419815	10.788141	25.569810	47.024316

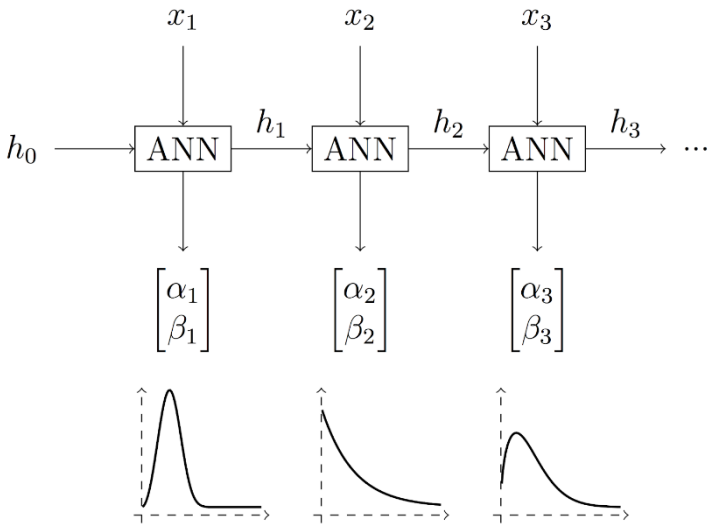
Looking at the results, again the average error is ~35 which is not great for failure prediction. However, looking at third quartile (~45) and maximum errors (120-220) we can say that the errors are not high for ¾ of the results.

This model clearly needs some adjustments and improvements to get better results.

3.3.3.4. Probability of Failure (PoF) estimation

Probability of failure estimation is an important problem for practical predictive maintenance. We used Egil Martinsson's WTTE-RNN : Weibull Time To Event Recurrent Neural Network (published in 2017) paper as our template

Using step-to-step **LSTM-architecture** with a 2-dimensional positive output layer **predict parameters of Weibull Distribution** for engine. Weibull Distribution distribution is widely used in reliability engineering and calculating time to event (TTE) problems. Churn prediction, survival analysis, time-to-failure etc. The structure of the ML model is given below:



We want users to select an asset and enter the lifetime cycles, e.g. days, as input and return a probability of that asset to fail in that given cycle. For this implementation we use Egil Martinsson's paper and this source as our template. We take all sensor data and label them as 1 if the machine is known to fail at the end of the last data point.

The main idea is to design a model that can look at a timeline of historical features (sensor data) leading up to the present, and predict a distribution describing the likelihood that a failure will happen as time moves into the future. The chosen distribution to predict is *Weibull Distribution* which is widely used in survival analysis, reliability engineering and failure analysis to find time to event probabilities. In order to accomplish this, we designed a model that predicts the two parameters, alpha and beta, that control the shape of the Weibull distribution. We used a neural network that can look at some historical data, and output two parameters describing a distribution. In order to train a neural network, yweou need a loss function that lets us evaluate model performance and backpropagate cost information through the network. We have used a *weibull log-likelihood* function for this purpose.

We used 1 LSTM layer followed by a Dense layer with 2 units to output alpha and beta parameters and an Activation layer that uses a custom activation function to find alpha and beta. The LSTM layer's units are determined by our AutoML module. Using the loss function as our metric, AutoML finds the best number of units for the LSTM layer.

$$F(t) = 1 - e^{-\left(\frac{t}{\eta}\right)^{\beta}}$$

Results on NASA Data: Using the same NASA dataset, we trained our model and tested its results. Since our model gives alpha and beta parameters of Weibull Distribution using this formula we found the probabilities:

	TTE	Probability	Test Day	Fail	Distance
0	112.0	0.112345	100	0	0.112345
38	142.0	0.125442	100	0	0.125442
84	118.0	0.127956	100	0	0.127956
21	111.0	0.141482	100	0	0.141482
86	116.0	0.174321	100	0	0.174321
...	...	...	...	...	...
44	114.0	0.805940	100	0	0.805940
29	115.0	0.806820	100	0	0.806820
72	131.0	0.807630	100	0	0.807630
1	98.0	0.189111	100	1	0.810889
88	136.0	0.813068	100	0	0.813068

100 rows × 5 columns

For *t*, we used 100 which means we get the probability of failure in 100 cycles of each engine. Then we calculated a distance for each engine. If the engine is going to fail in 100 days the distance is ***(1-probability)*** and if not, it is ***probability***. Sorting the test results based on this distance we get the following results on the right:

Then we did some calculations on this data: total distance, average distance, minimum distance, maximum distance, first, second and third quartile.

- Total distance: **31.138109183310988**
- Average distance: **0.31138109183310986**
- Minimum distance: **0.11234524020931036**
- Maximum distance: **0.8130683947724363**
- First quartile: **0.1927447331777519**
- Second quartile(median): **0.19765620207882723**
- Third quartile: **0.282159741283545**

The engine with the minimum distance estimation is:

	TTE	Probability	Test Day	Fail	Distance
0	112.0	0.112345	100	0	0.112345

The engine with the maximum distance estimation is:

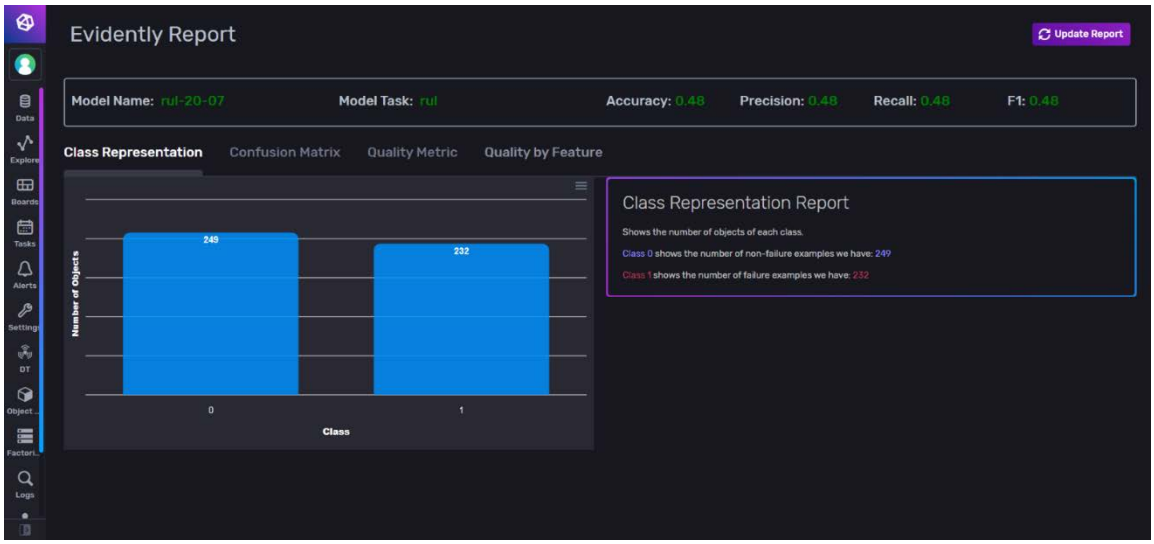
	TTE	Probability	Test Day	Fail	Distance
88	136.0	0.813068	100	0	0.813068

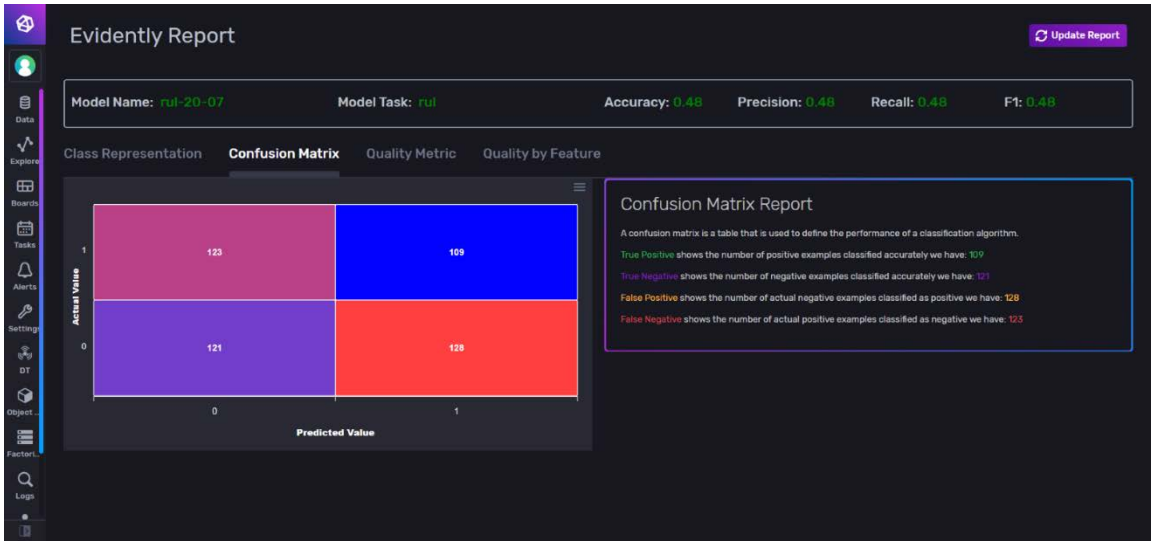
Inspecting these results, we can conclude the following:

- Based on the quartiles, total and average distance, we can say that the model has decent results for estimating the probability of failure.
- The current model doesn't give perfect results but, in the future, we will make it better once we use it with real data. Even if the engine has more lifetime than 100 cycles the probability is not very close to zero or if the engine has definitely failed in 100 cycles the probability is not very close to 1. The minimum distance value of 0.112345 also shows that.
- The model gave high distance results when the failure cycle is above 100 apart from the one engine data where the failure cycle was 98 in which model gives 0.189111 probability to fail in 100 cycles. Estimating engines to fail when they actually have more cycle lifetime can be tolerable based on some cases. However, missing the failure cycle has a more detrimental effect. Our model gives more false alarms than missing failure cycles. If we make the correlation with the RULE binary classification model we can say that the same results can be seen here as well.

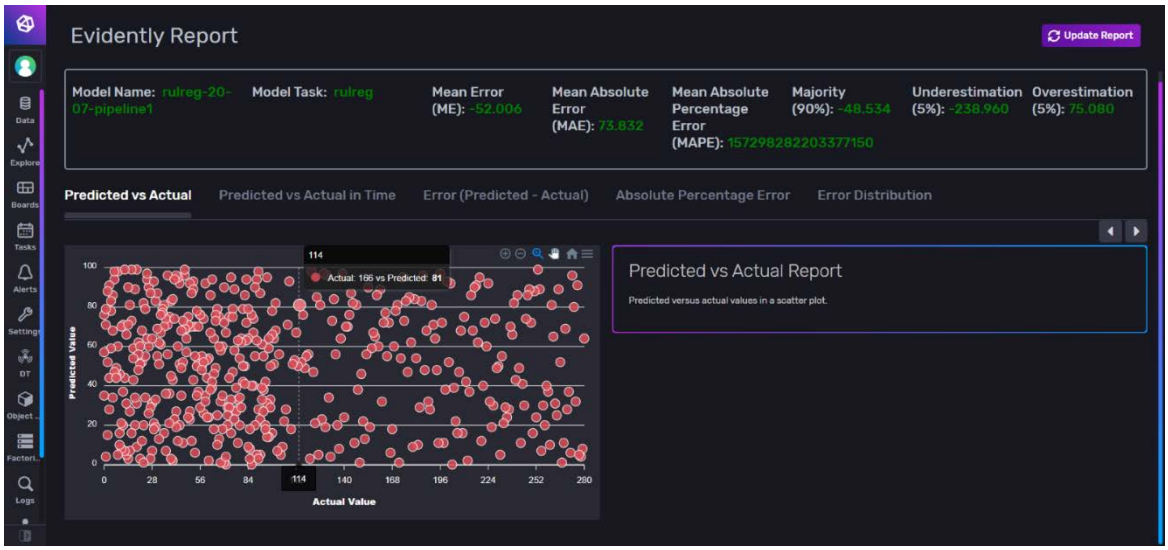
3.3.3.5. Evaluating the success of ML models

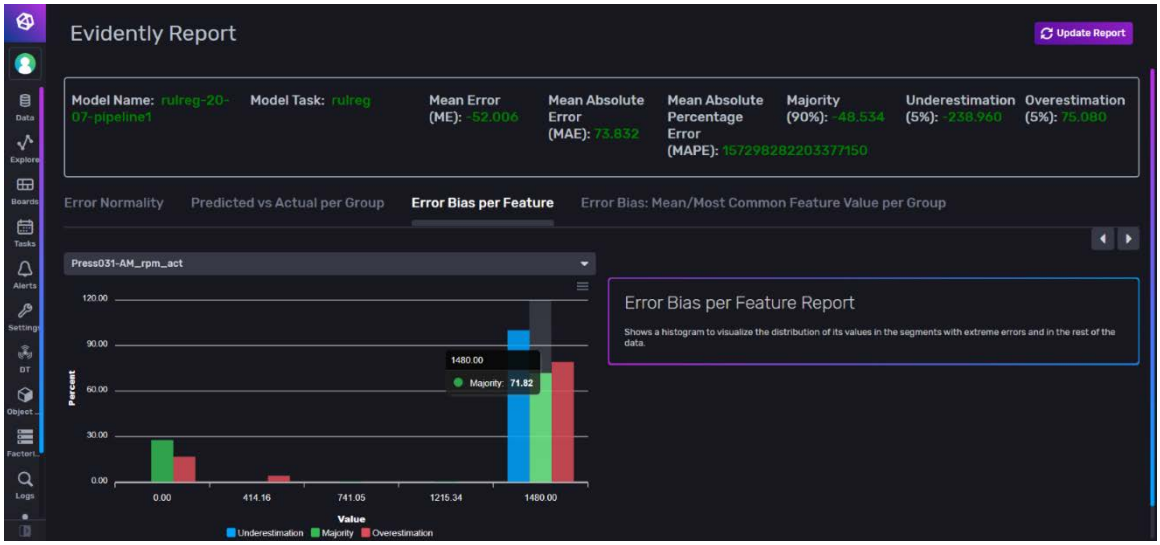
Evaluating the model's success is an important task so that users can decide whether or not to trust the model's logs. For this purpose, using Evidently library we designed a page where users can see model's performance in detailed graphs based on their feedback to the model's logs.





In the above screen shots, we have a RUL model's (binary classification model) performance results.





In the above screen shots, we have a regression model's performance results.

3.4. Data Acquisition and Data Processing (TEKNOPAR)

Data acquisition and processing refers to the studies related to collect, compile and process data. In the Ermetal case, data is collected from the sensors by means of an OPC Router. The fields of the OPC Router are mapped to the related database fields for storage purposes. The main elements for data acquisition and processing are operating systems, message brokers and databases.

The working environment for data acquisition can be built on physical or virtual computers. Before deciding on the environment type to be used for the Ermetal's use case, pros and cons of having virtual or physical environments were studied. The differences between physical and virtual computers have been evaluated, and it was decided to install the system entirely on virtual servers in order to minimize physical failures and possible connection problems.

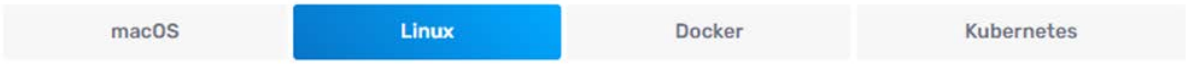
The operating system of the platform has been selected to be the Windows Operating System. The selection of the operating system is based on the OPC Router documentation. The supported Windows Operating systems are listed below:

Operating system

We recommend Microsoft Windows Server operating systems. Client operating systems (64 bit) can also be used for test purposes. The following Windows versions are supported:

- Windows Server 2008 x64
- Windows Server 2008 R2 x64
- Windows Server 2012 x64
- Windows Server 2012 R2 x64
- Windows Server 2016 x64
- Windows Server 2019 x64
- Windows 7 x64
- Windows 8 x64
- Windows 8.1 x64
- Windows 10 x64
- Windows 10 IoT x64

In order to store data, the selected database InfluxDB was installed on the Linux operating system. InfluxDB does not have a Windows installation. For InfluxDB to be installed on the Windows operating system is to use Docker technologies. In the project, instaed of using Docker and Windows, Linux operating system was selected for database installation.



Installation of Database

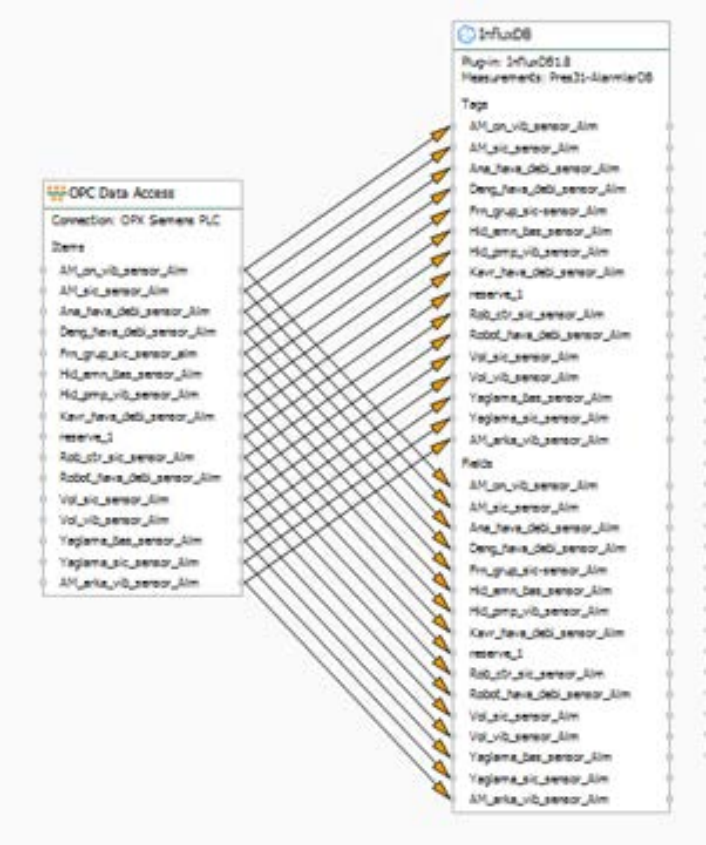
InfluxDB 2.0 was installed on the Linux system using the VPN accounts set by Ermetal. The connection was tested and it was confirmed that the database was successfully installed.

OPC Router was installed on the Windows operating system that is set in a virtual computer. After providing the necessary information for connection to the Ermetal's server, the acquired PLC information fields have been defined to the OPC Router. Thus, the correct operation of the physical installation in the factory has been ensured by providing the connection between the PLC and the OPC Router. After accessing the PLC, the relevant data blocks (sensor and energy analyzer data) are defined to the OPC Router.

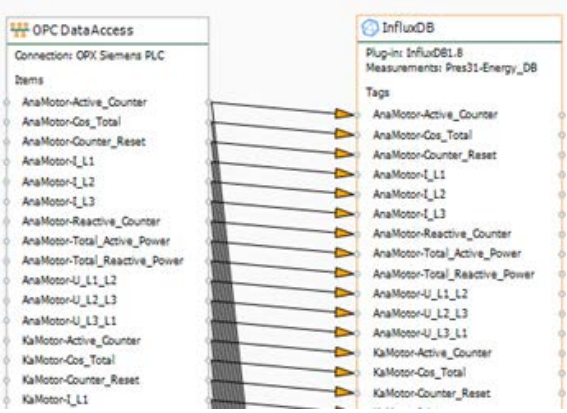
Following the setups in the PLC and the OPC Router, the next step is to install the database. InfluxDB 2.0, has been installed in order to record the sensor data that is flowing from the field into the database. The installation of the InfluxDB 2.0 database was performed on the Linux virtual computer. In order to ensure data flow, the network structure established between 2 virtual computers has been verified. After it was verified that the servers were communicating with each other, the necessary definitions were made for the InfluxDB plugin of the OPC Router. At this stage, the IP and port information of the virtual server where the database is installed are defined. Errors were encountered while connecting to the database by the OPC Router. As a result of the interviews with OPC Turkey and OPC Global, it was determined that the OPC Router-InfluxDB plugin does not support the use of InfluxDB 2.0. Since it was stated by OPC Global that the development of the plugin compatible with InfluxDB 2.0 would take a long time and there might be some errors, it was foreseen that it might cause disruption in the business plan. Thereupon, the alternatives in the system architecture were investigated, and the suitability of the InfluxDB 1.8 database for the solution was evaluated and it was concluded that the use of InfluxDB 1.8 would be sufficient.

The InfluxDB 2.0 database installed on the Linux virtual server was stopped, and the InfluxDB 1.8 database running on the Docker container structure was installed. In the installation of InfluxDB 1.8, virtualization was preferred with the use of Docker container in order to provide easier management, scaling operations and instant log tracking. The path of the files that the database will create is predefined. Thus, in cases where crashes or critical errors are encountered, data is kept separate from the container. Deleting the container will not affect the data, as the data is independent of the container.

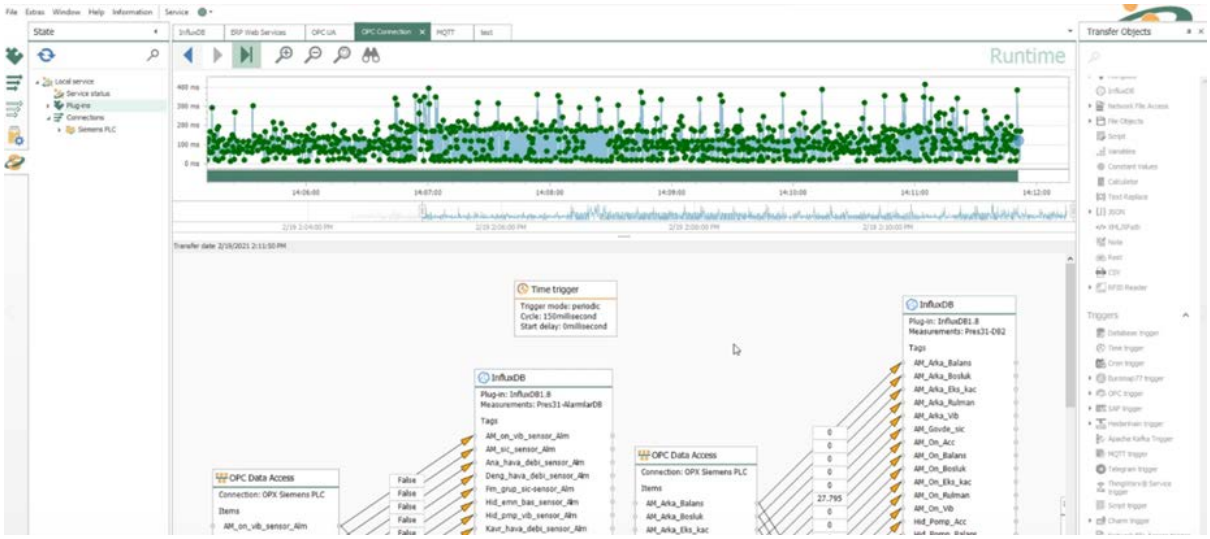
Two users, admin and user, are defined in the database. The main database is named "machinaide". After database operations, the connection to the database was verified using the OPC Router plugin. After this stage, the values from the defined data blocks were recorded in the database.



Recording sensor data into database, items are mapped to Database fields and tags

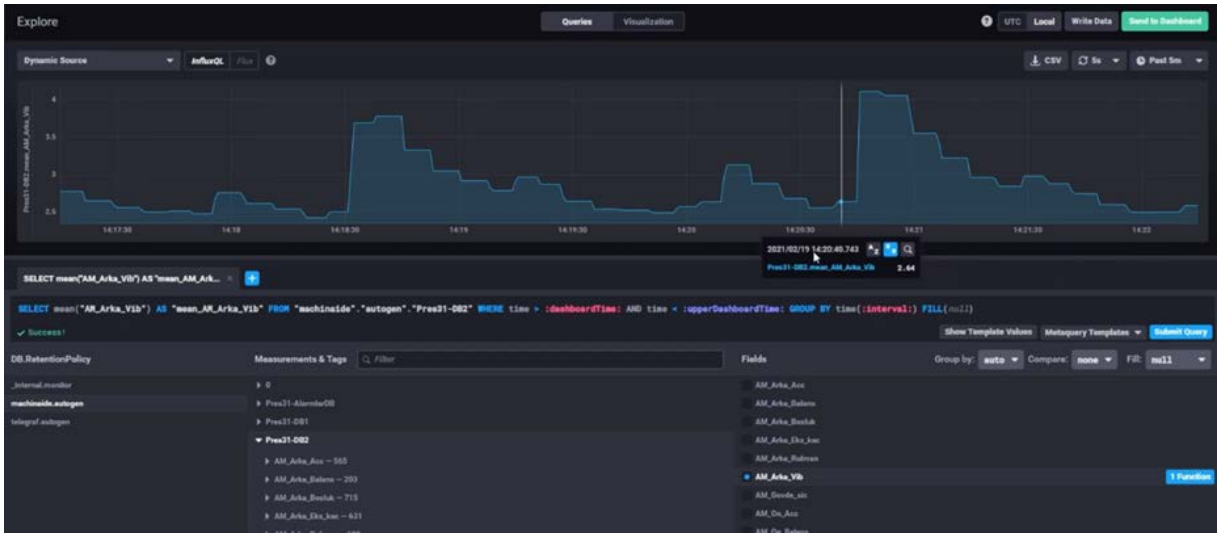


Energy Analyzer data mapped to Database tags



Recording real time data of OPC Router

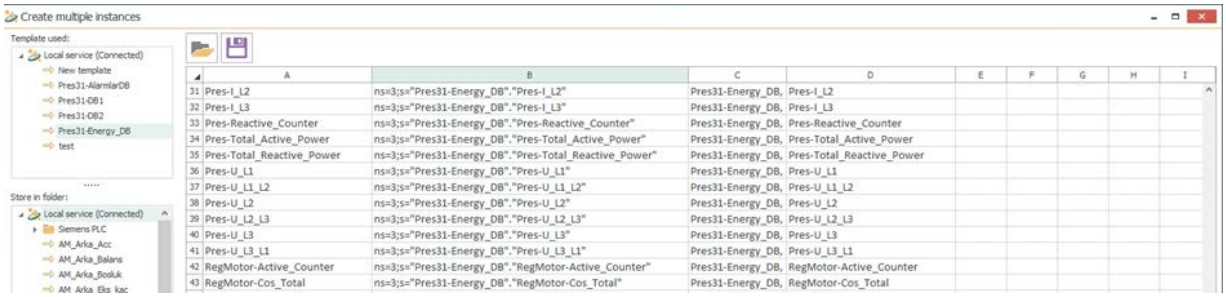
To confirm the collected data and to determine that the data was recorded in the database without any problems, Chronograph was installed. Chronograph enables to monitor the InfluxDB database, view the system status (RAM, CPU, network, disk), query on the database and display the results on the dashboards. Chronograph was also installed on the Docker container structure. By taking Docker containers into the same network, Chronograph has been connected to InluxDB 1.8. A portainer has been installed to monitor and easily manage 2 separate Docker containers. Portainer provides a very useful web interface for management.



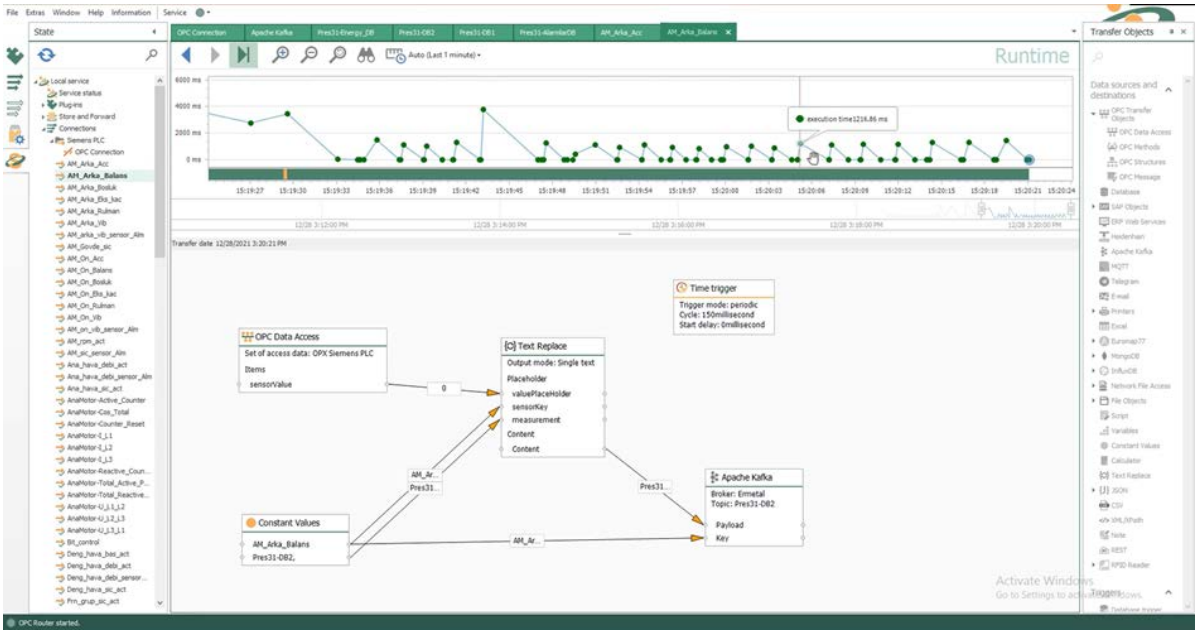
A Screenshot of Chronograph

The connection information of the Cronograph was shared with the Ermetal team to confirm the collected data. After the data was collected, the error “Execution time: Transfer not recorded.“max-series-per-database” limit exceeded (1000000) dropped=1” was encountered. By changing the necessary parameters in the InfluxDB settings file, the data recording limit has been removed. It was decided to use InfluxDB 2.0 with the partners of the Turkish consortium. Since the OPC Router does not have Influx DB 2.0 direct plug-in, it was decided to activate the Kafka plug-in.

Kafka plug-in was purchased. Data has been written to InfluxDB using the Line Protocol over Kafka. The collected data is redefined on the OPC Router in accordance with the Line Protocol structure. Sensor mapping process was performed once again for InfluxDB2.0, and it was ensured that the data were transferred to the appropriate Kafka topics in the desired format with the Key.



Line Protocol Definition



Transferring Data to Kafka

Apache Kafka installation which was also made onto a virtual Windows computer was also made on Linux. During this operation, data connection information was updated for OPC Router.

3.5. HMI Applications (DOĞRU)

Human-machine interaction (HMI) or human-computer interaction (HCI), is the way in which humans interact with machines in order to accomplish certain tasks in a human-machine system consisting of humans i.e. users, machines, interfaces between them and the environment in which they operate. . A sound and functional HMI is a key point in designing such systems and empowers users to perform operations on the machines by designing modes of interactions via instruction and information exchange.

The difficulties and expectations for human operators in the factory space will change as the complexities of manufacturing systems increase with rising autonomous and self-organizing production systems. With this Industry 4.0 transformation, humans need to be integrated into the cyber-physical structure. To facilitate this integration, user interfaces for mobile devices like smartphones, tablets, smart glasses, overhead displays with multi-touch, voice and gesture-based control are already introduced in the industrial domain.

In Industry 4.0, information from multiple components will be aggregated, analyzed and visualized making use of these devices and interfaces. The role of the human operator will be to interpret this information, supervise and monitor the production systems and intervene in potential critical events. Bilberg and Malik, demonstrated how a simulation-based digital-twin can drive a human-robot automated assembly line.

Human-machine interface (HMI) is in a key role for implementing smart manufacturing and optimization process, which comprises the issues of communication, interaction and cooperation between humans and machines. HMIs exist throughout the product lifecycle including product design, manufacturing, and service. Efficiency and safety are primary concerns that increase the requirements. New emerging technologies such as big data, AI, digitalization and augmented reality (AR) take place in HMI activities gradually for smart manufacturing. New technologies used in HMI applications (already widely used in consumer products e.g. smartphones, gaming industry) are gesture recognition, touch interaction, voice interaction, virtual reality and augmented reality. The major expectation from HMI technology is to meet the user needs and provide the latest technological capabilities to facilitate operations/tasks, enhances safety and yield correct decisions (decision support systems and informed decision making).

In this project was investigate the possible usage of the advantage from the know-how and excellent usability reached by the mobile industry (smartphone, tablets, web apps, etc.) of using classical industrial devices. By means of innovative HMIs we have developed so far and that we are developing, will move a huge diversity of interfaces and interaction concepts to a simpler / more intuitive and direct one.

- **Hand Gestures**

Hand gestures are important for our mobile applications that we are developing. Because in the factory environment, personnel work in oily or noisy environments. They can directly switch to the interface they want to access with hand gestures without touching the mobile device.

Hand gestures detected with machine learning in both apps. 21 joint positions in fingers detected by machine learning. As a result of the calculation of the joints according to the Y coordinate, the distance measurement was made against the finger mark and it was displayed to the end user. Hand movements were detected by uploading videos, pictures or camera.

Screenshot 1 - Hand gestures detected with machine learning

- **Blink Gestures**

Also, blink gestures are important for our mobile applications that we are developing. Working machine, vehicle, forklift etc. blink gestures are integrated to make it easy to use. They can directly switch to the interface they want to access with blink gestures without touching the mobile device. Face network line created with machine learning. A total of 30 points in the left and right eyes were located by machine learning algorithm. Distance measurement was calculated according to the Y coordinate.

Screenshot 2 – Blink gestures detected with machine learning

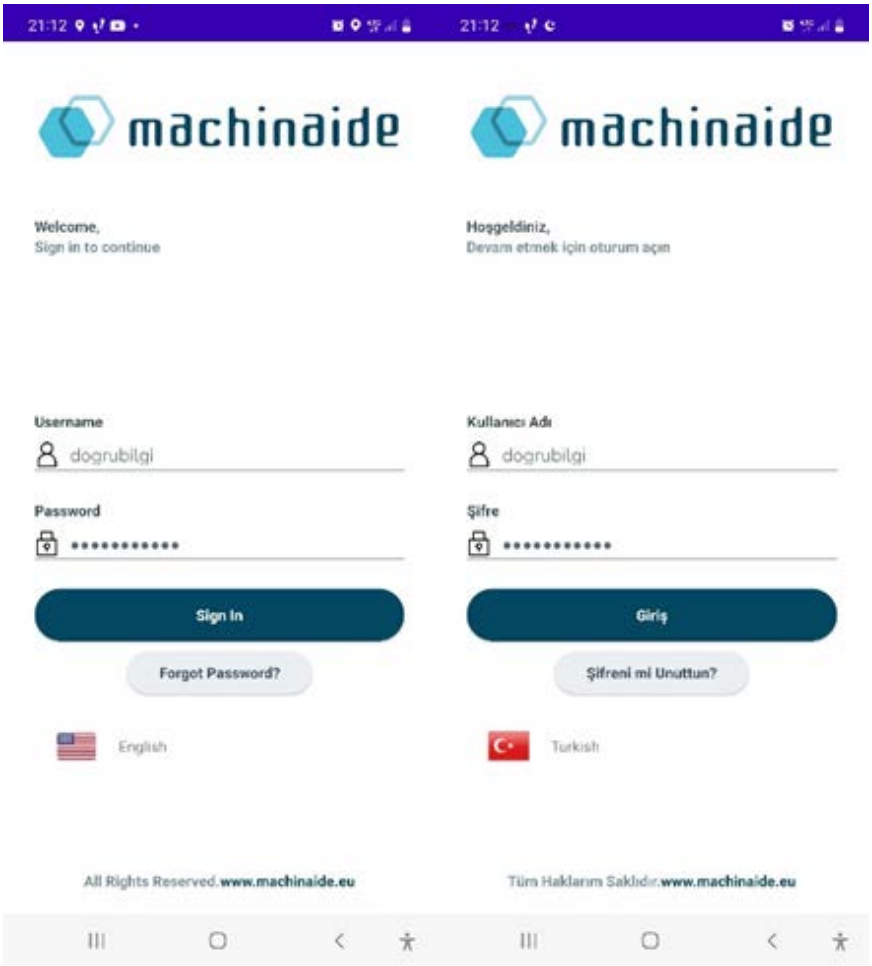
- **Speech To Text**

Another feature our apps have is speech-to-text translation. For Android, we use Google Speech to Text technology and for IOS, Swift Speech technology. The Overlay Foreground service is used. In this way, we can use this service even if the application is not open.

Our mobile applications are developed on 2 different platforms. Native Android and Native IOS. We have done research to ensure that our application is the only platform or cross platform. According to our researched, we realized that cross platforms (flutter etc.) do not support hand gestures or blink gestures. But now, our applications have this features.

3.5.1. Android Application

- Android application was developed in Kotlin language.
- Relevant libraries were included using the MVVM architecture, which is one of the design pattern architectures.
- Room library was used to maintain database independence.
- Improvements were made in accordance with SOLID principles.
- A factory structure was established for the transition between activities in order to provide View-Model communication.
- The design of the mobile application, menu icons and fonts were created.
- We added Turkish and English language packs to our application.
- We provided access to the digital twin API.
- API results are provided with the “Retrofit” library by parsing the final data types.
- User login screen was completed.

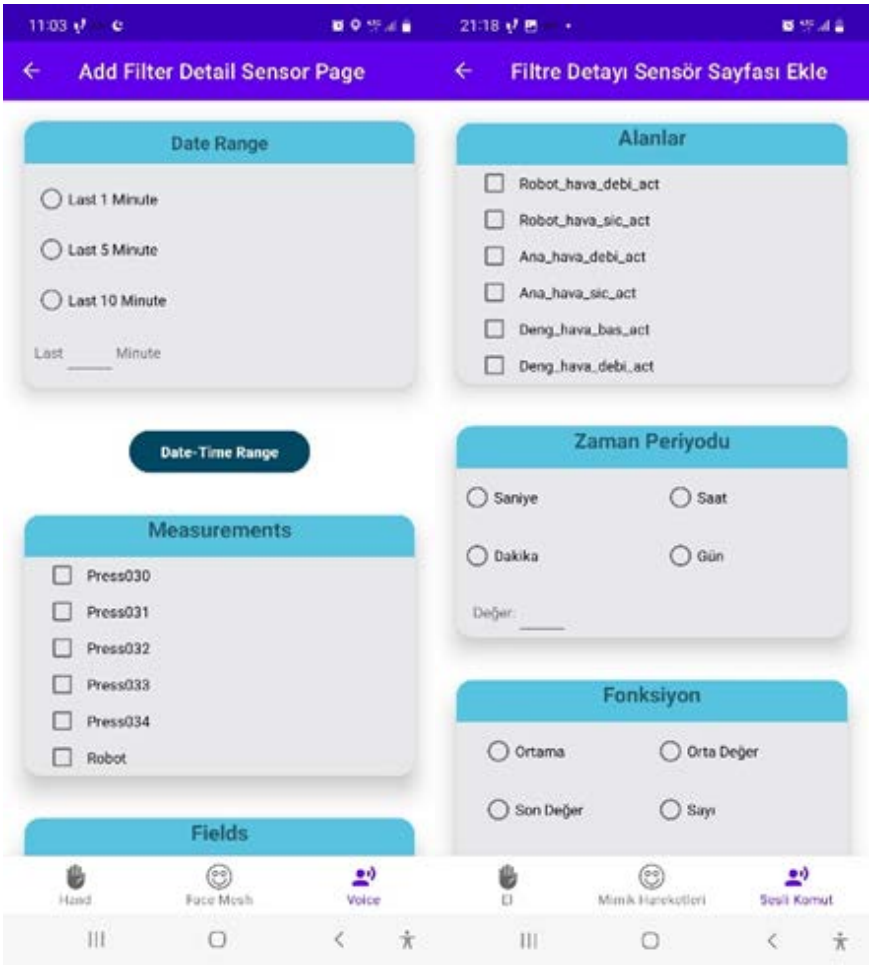


Screenshot 3 - Login (Android)

- The homepage was designed.

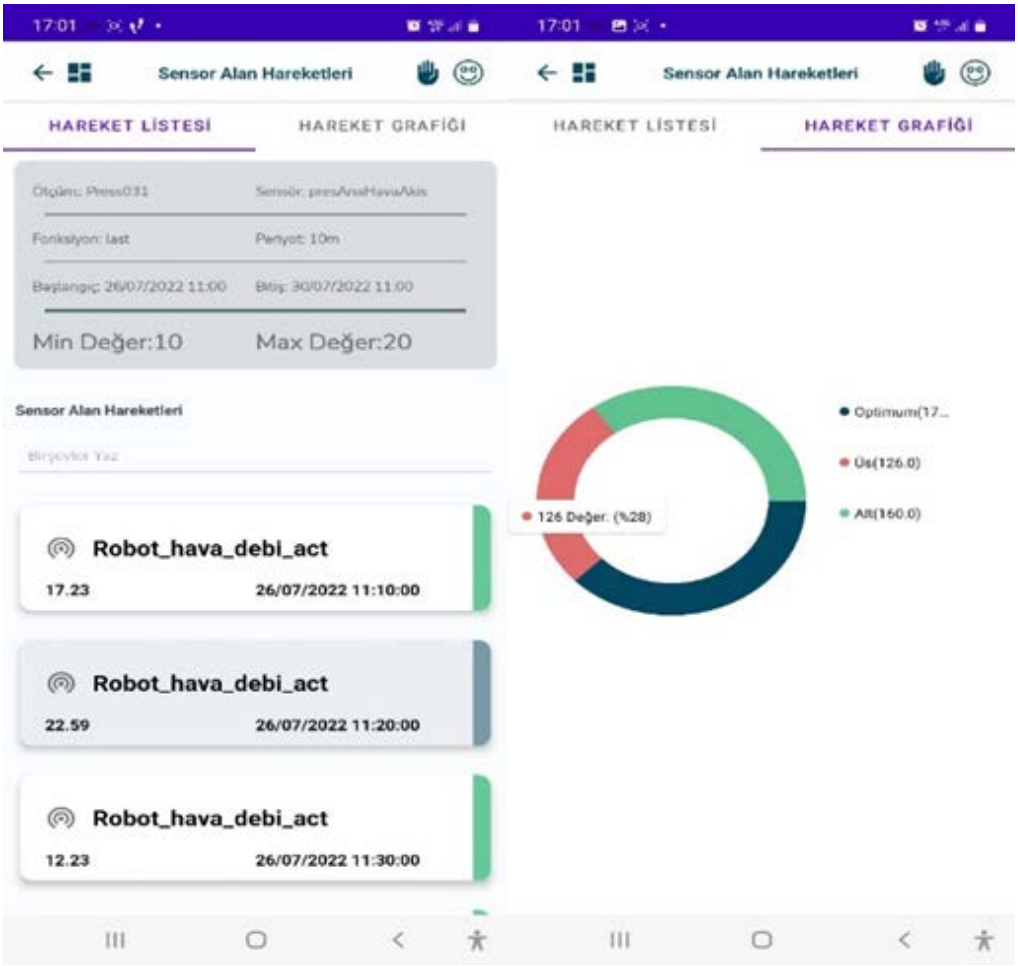
Screenshot 4 – Home Page (Android)

- Detailed Sensor Filtering page has been created. Taking into account the mandatory fields in the API, the details of the sensor's information are displayed. Sensor data is calculated by filling in critical fields. For example; After selection in minutes, hours or days, the request is sent to the API. In this way, the user can easily access the data they want to see without entering the start and end date ranges.



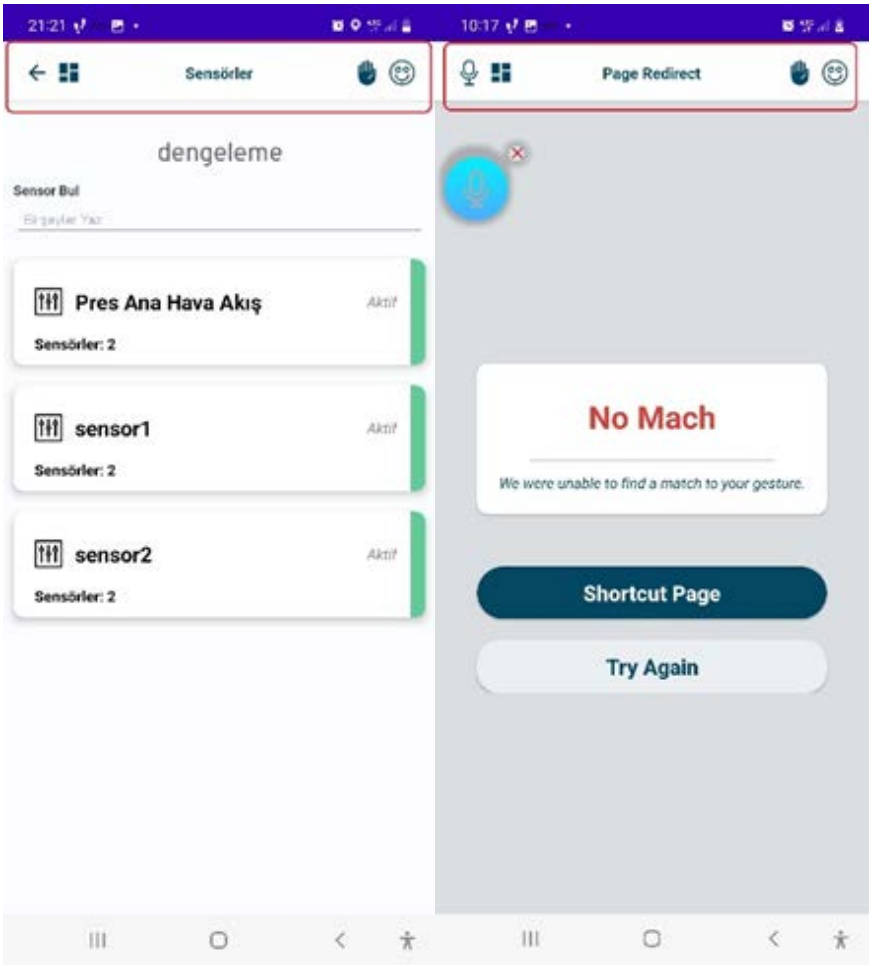
Screenshot 5 - Filter Detail Sensor Page (Android)

- The sensor area movements page has been created. Sensor field movements; It was designed as 2 Tab menus, list and graphic. In the Graphics tab, 3 color types are specified. Grey, green and red. Gray color was selected for data below the threshold value. The green color was chosen for the data in the threshold range. Red color was determined for data above the threshold value. In the motion list tab, the detailed information of the relevant sensor value is listed.



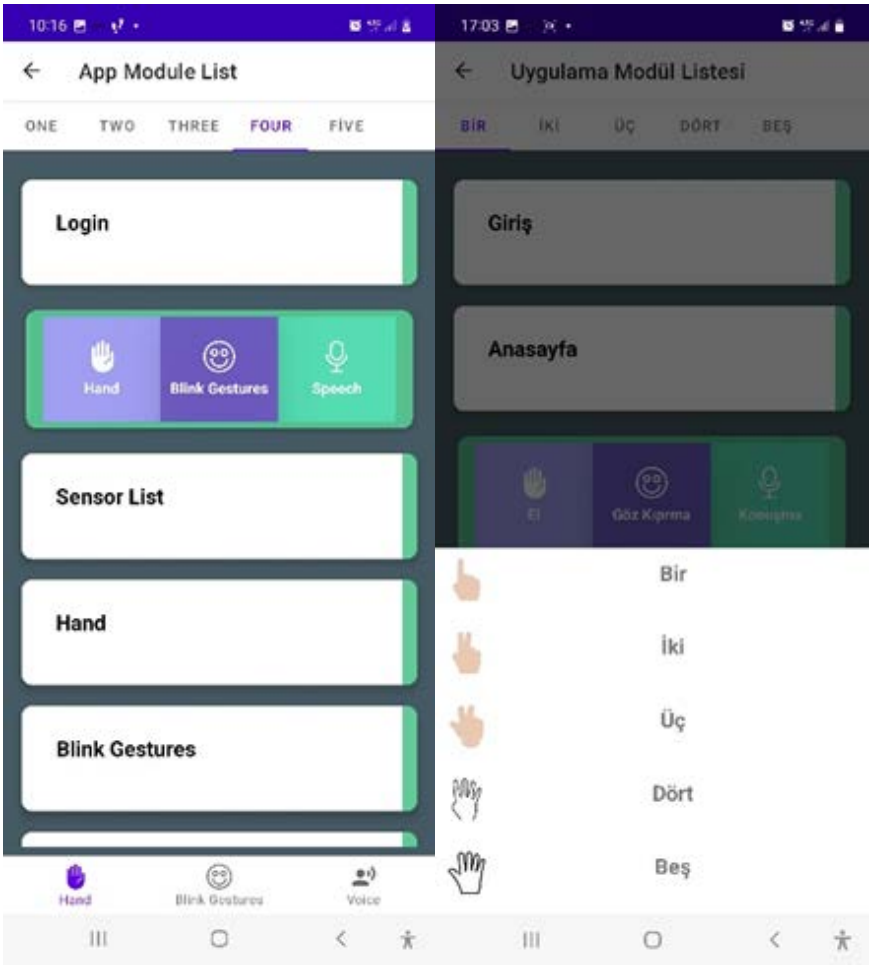
Screenshot 6 - Sensor Field Movements : List and Graphic Page (Android)

- By creating BaseActivity, the top menu is shown in each module. If the current page is Home Page, the back button has been removed. Back button and home button have been activated in all modules except the homepage. Icons related to voice command, hand and blink gestures were displayed in the top menu. Voice command service, hand and blink gestures icons were created and a modular structure was established for direct access.



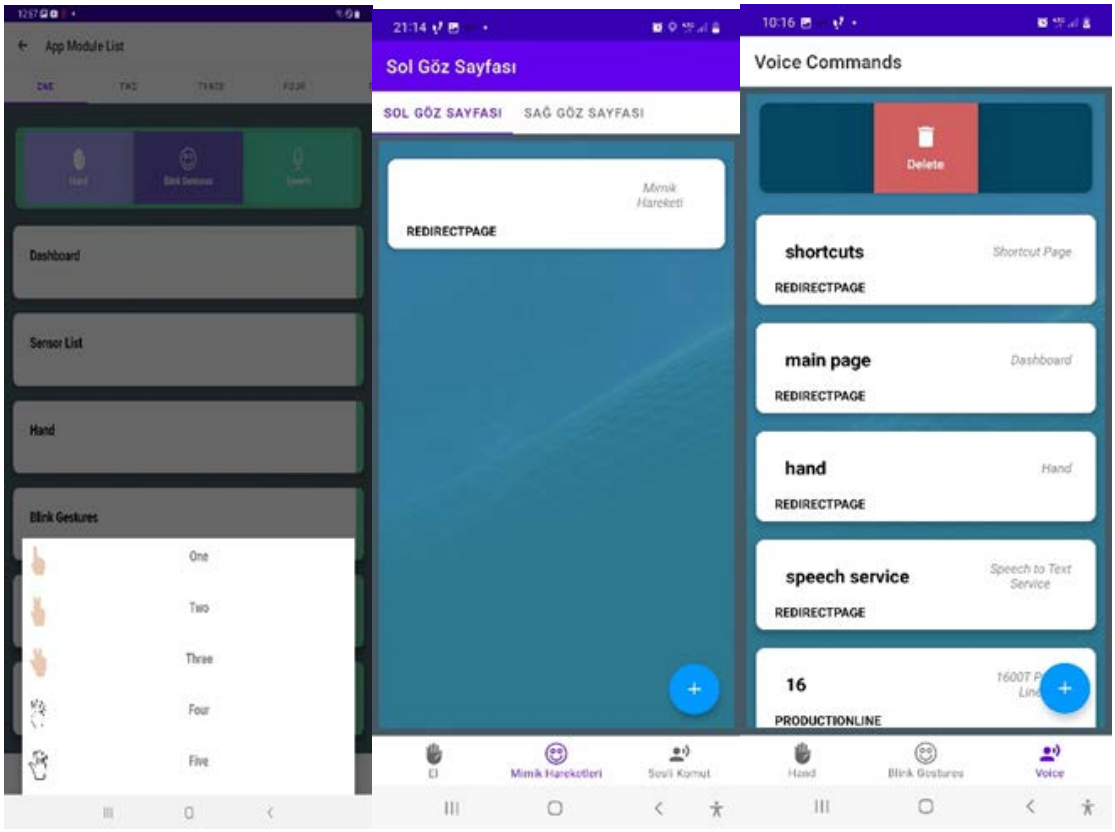
Screenshot 7 – Top Menu (Android)

- Local database is created. In this way, Features can be customized on a user basis. For example, 2 finger gestures can redirect to the settings menu for one user, while 2 finger gestures can redirect to the sensor list for another user.



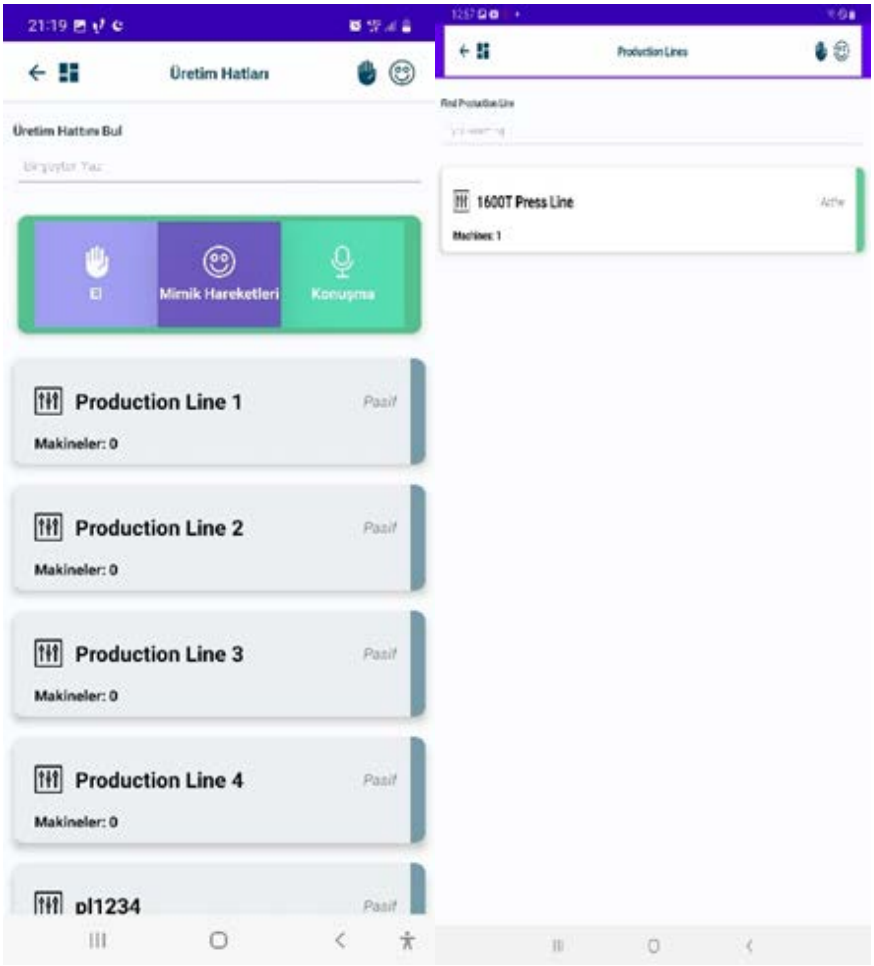
Screenshot 8 – Features Can Be Customized On A User Basis (Android)

- Shortcut page created and Navigation Component yapısı kuruldu. Created 3 submenus on the shortcut page. Hand gestures list, blink gestures list and voice command list. Features available in these menus are associated with pages containing HMI headers.



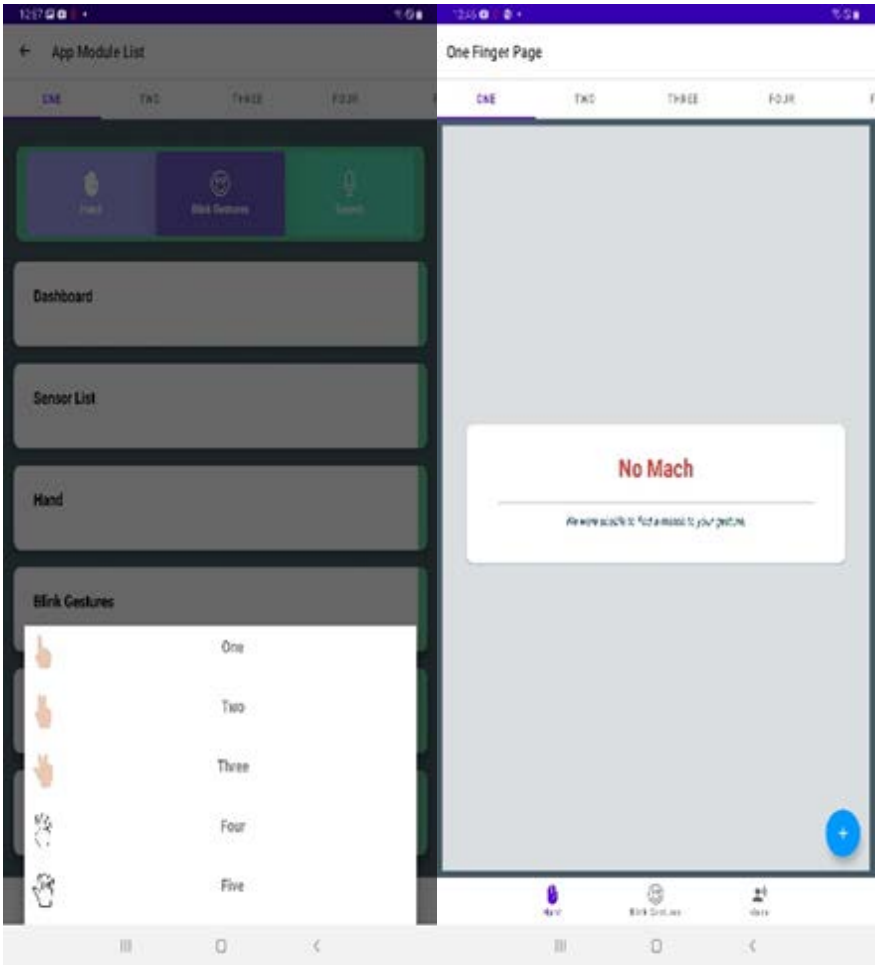
Screenshot 9 – Shortcut assignment pages (Android)

- Another association method is the contents in the Digital Twin hierarchy from the API. In order to define, the list content is moved to the left, and operations are performed with hand, blink and voice command. Digital Twin hierarchy :
  - Production line list
  - Contents list
  - Machine list
  - Sensor list



Screenshot 10 – Production Line List (Android)

- Calculated hand movements based on machine learning. Data associated with finger movements are listed. The top menu corresponding to 5 different finger movements was designed and the list was created with the associated module. If there is no associated module in the list, a "No Match" message has been added to the page.

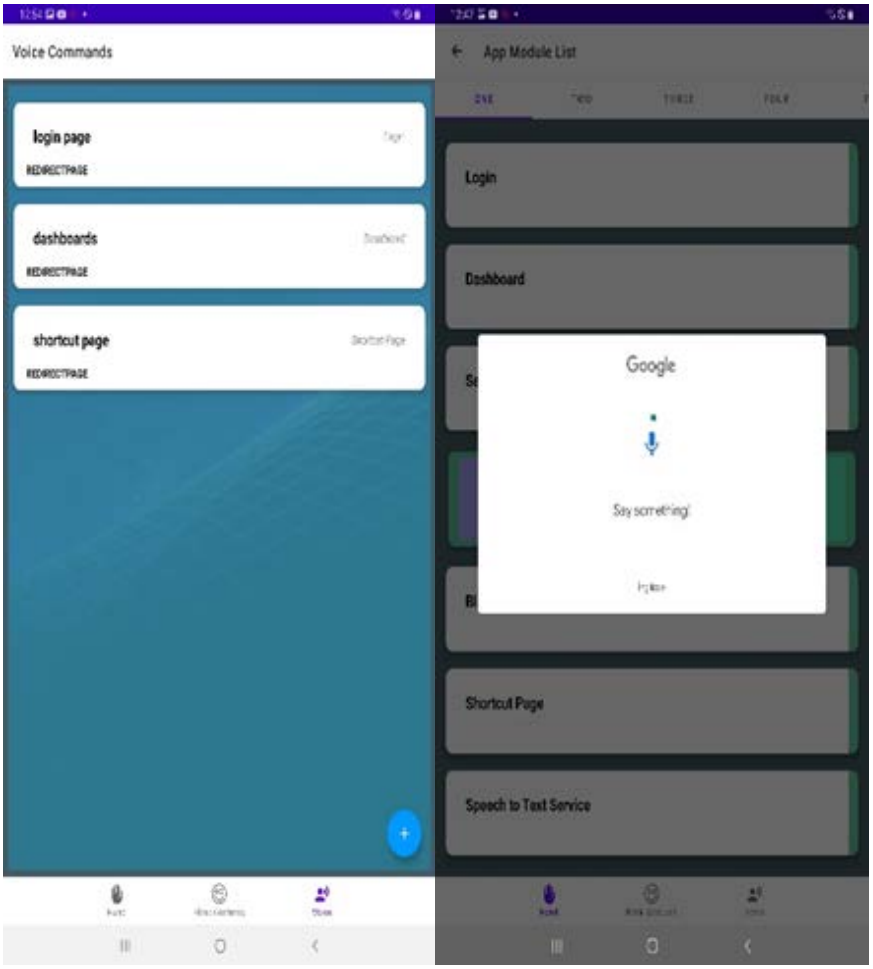


Screenshot 11 – Finger Definitions (Android)

- Blink gestures list page created. The top menu has been designed for the right and left blink movements.

Screenshot 12 – Blink Definitions (Android)

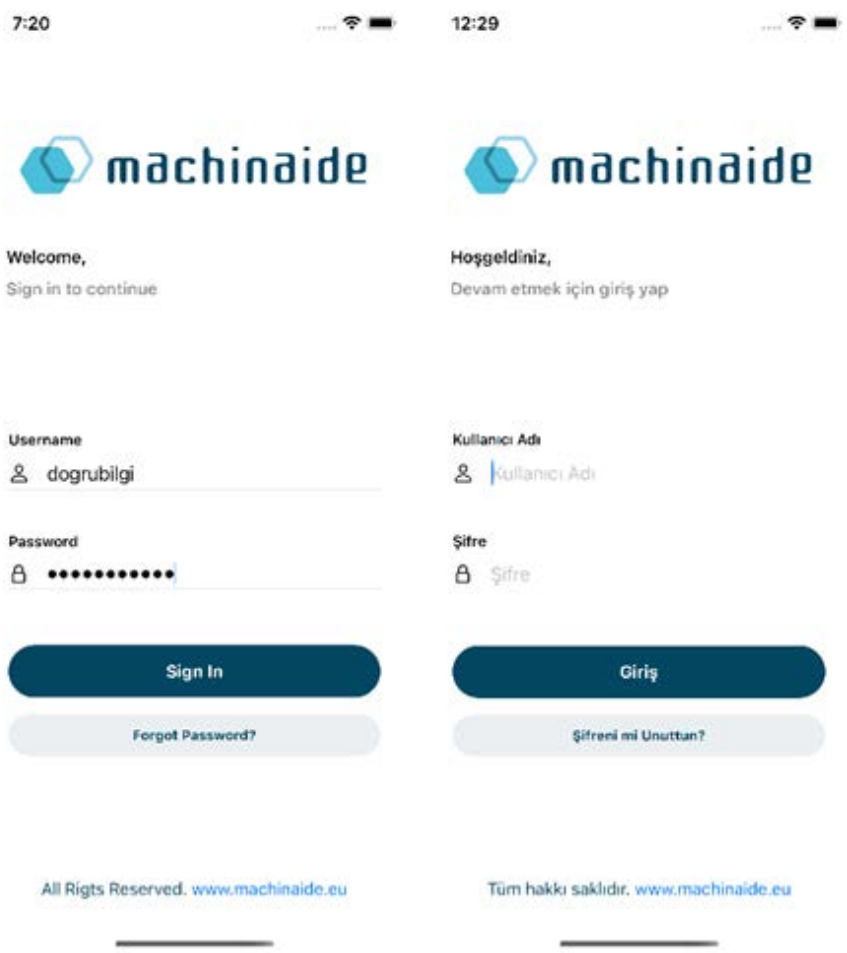
- Voice command service was created and voice analysis was performed. Voice analysis was performed with Google Speech to Text technology. The Overlay Foreground service is started when the voice command icon is selected. A list page of the contents defined using the voice command service has been created.



Screenshot 13 – Voice Definitions (Android)

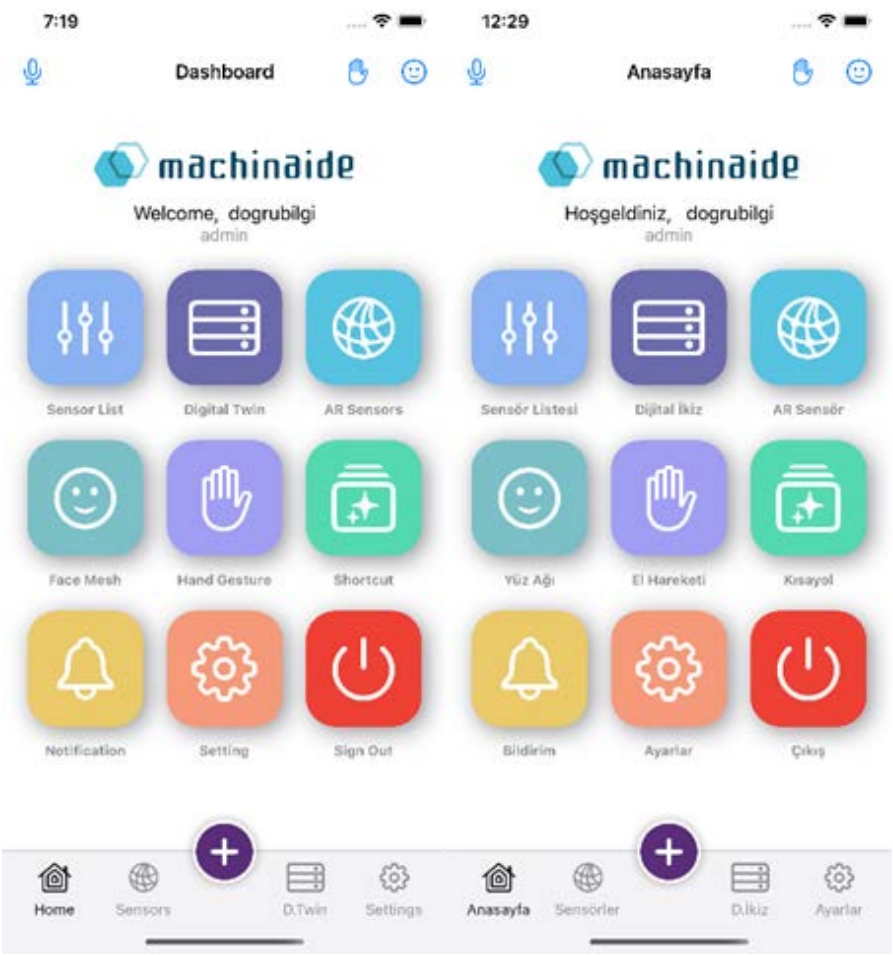
3.5.2. IOS Application

- IOS app developed with SwiftUI. It was preferred because it facilitates the establishment of dynamic structures and data management in the formation of front-end and back-end architectures.
- The MVVM architecture was created. Separation of Business Logic and View module has been ensured.
- The design of the mobile application, menu icons and fonts were created.
- We added Turkish and English language packs to our application. The language option can be changed by selecting the Machinaide application from the settings menu of the device.
- We provided access to the digital twin API.
- Warning messages have been created for mandatory fields.
- User login screen was completed.



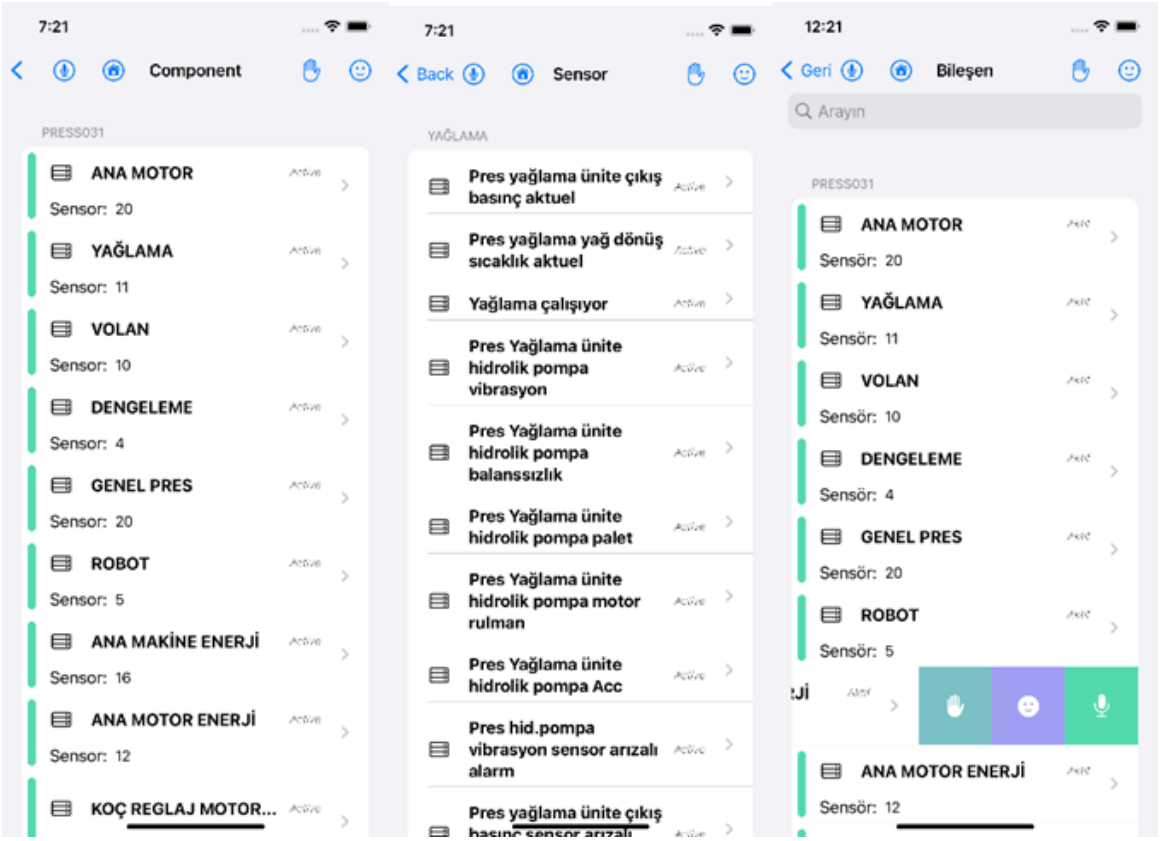
Screenshot 14 - Login (IOS)

- The homepage was designed.



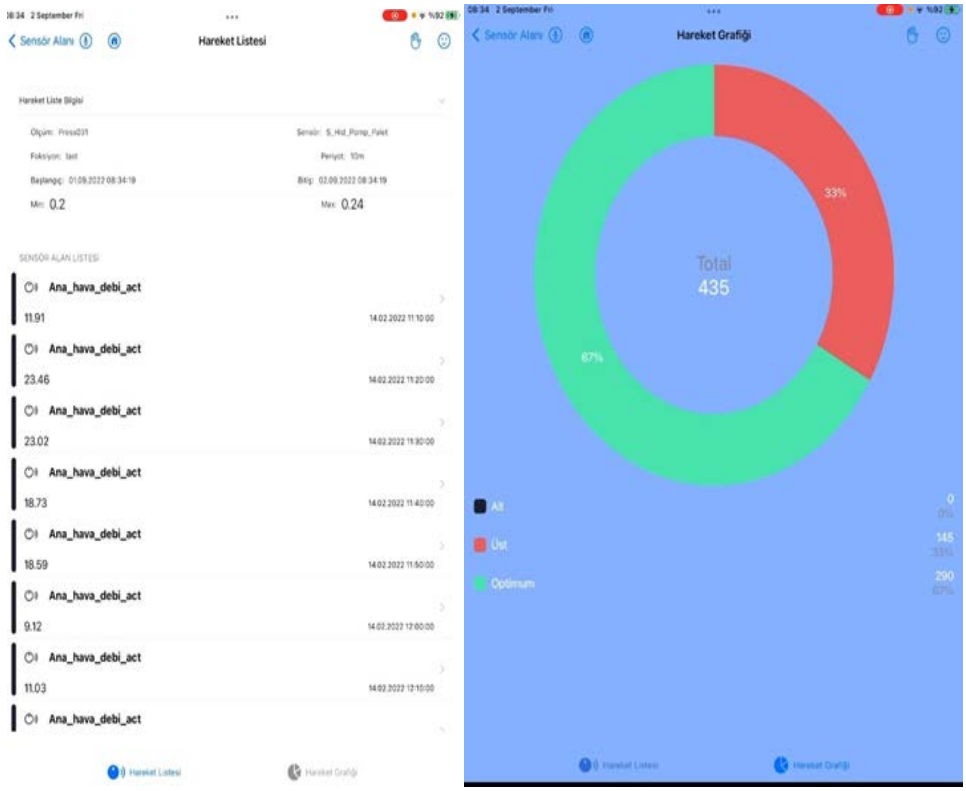
Screenshot 15 – Home Page (IOS)

- Production line, content, machine, sensor, sensor fields pages were designed.



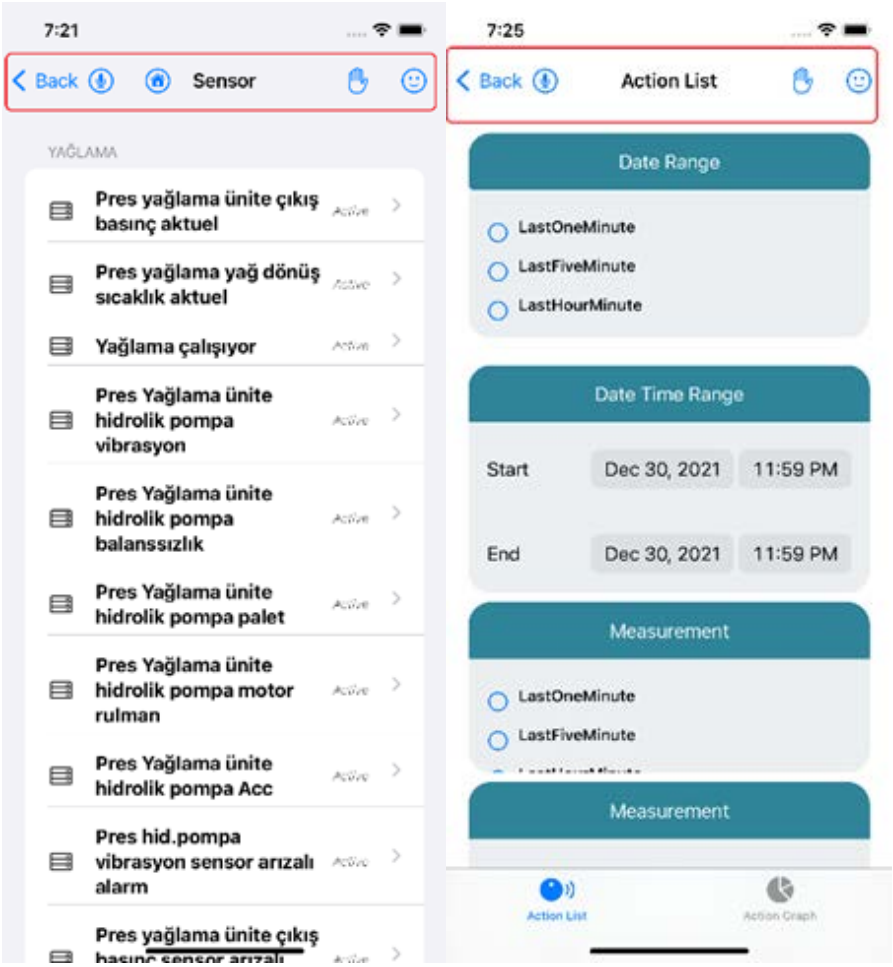
Screenshot 16 – Production Line, Machine and Sensor List (IOS)

- The sensor area movements page has been created. Sensor field movements; It was designed as 2 Tab menus, list and graphic. In the Graphics tab, 3 color types are specified. Grey, green and red. Gray color was selected for data below the threshold value. The green color was chosen for the data in the threshold range. Red color was determined for data above the threshold value. In the motion list tab, the detailed information of the relevant sensor value is listed.



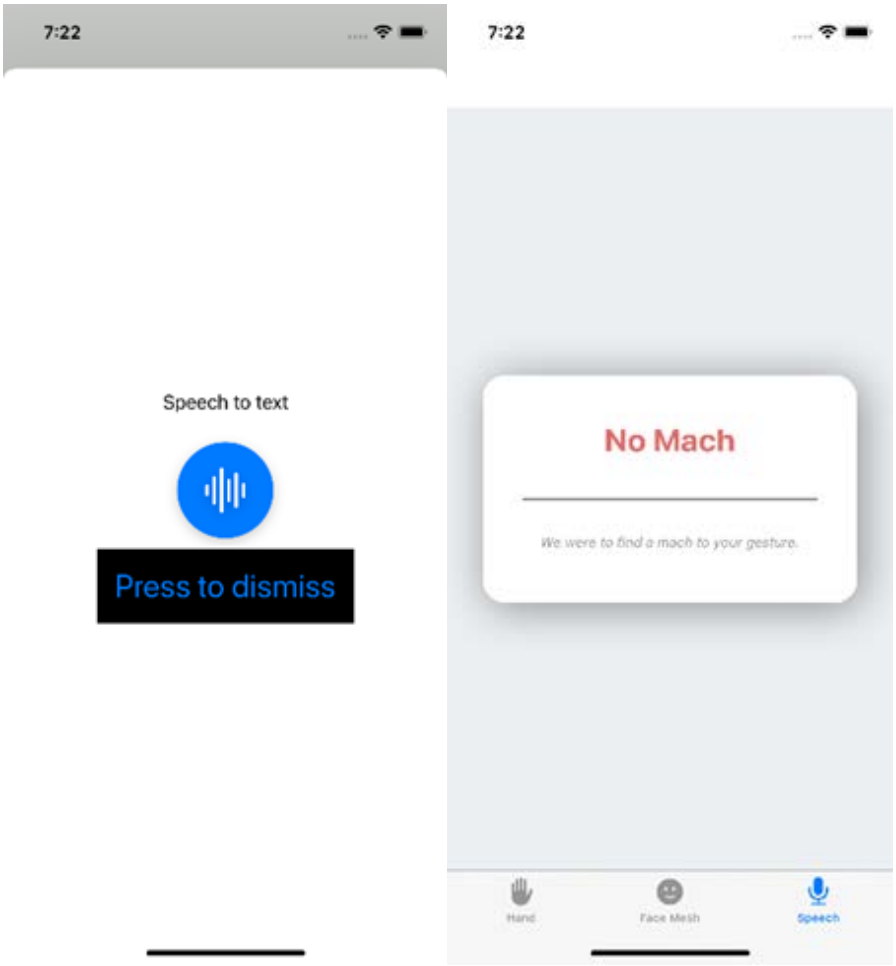
Screenshot 17 – Sensor Field Movements : List and Graphic Page (IOS)

- The top menu is shown in each module. If the current page is Home Page, the back button has been removed. Back button and home button have been activated in all modules except the homepage. Icons related to voice command, hand and blink gestures were displayed in the top menu. Voice command service, hand and blink gestures icons were created and a modular structure was established for direct Access.



Screenshot 18 – Top Menu (IOS)

- Was calculated hand movements based on machine learning with Vision library. Data associated with finger movements are listed. The top menu corresponding to 5 different finger movements was designed and the list was created with the associated module. If there is no associated module in the list, a "No Match" message has been added to the page.
- "Swift Speech" service was used for the 'Speech Textizer' operation and integrated into the top menu search bar.

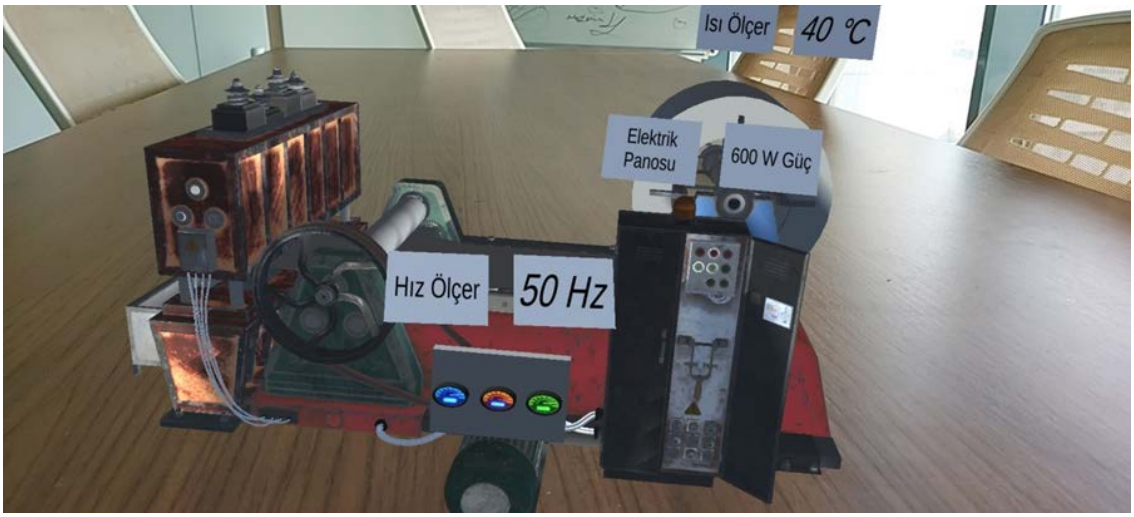


Screenshot 19 – Voice commands (IOS)

3.5.3. Unity

With Unity, it is aimed to display the machine models and the sensors of the models through our mobile applications with augmented reality.

First of all, online trainings on Unity program and machine model design were received. Thanks to these trainings, a machine model on the internet was modeled in the Unity program, for example, and sensor data for testing purposes were added to the machine.

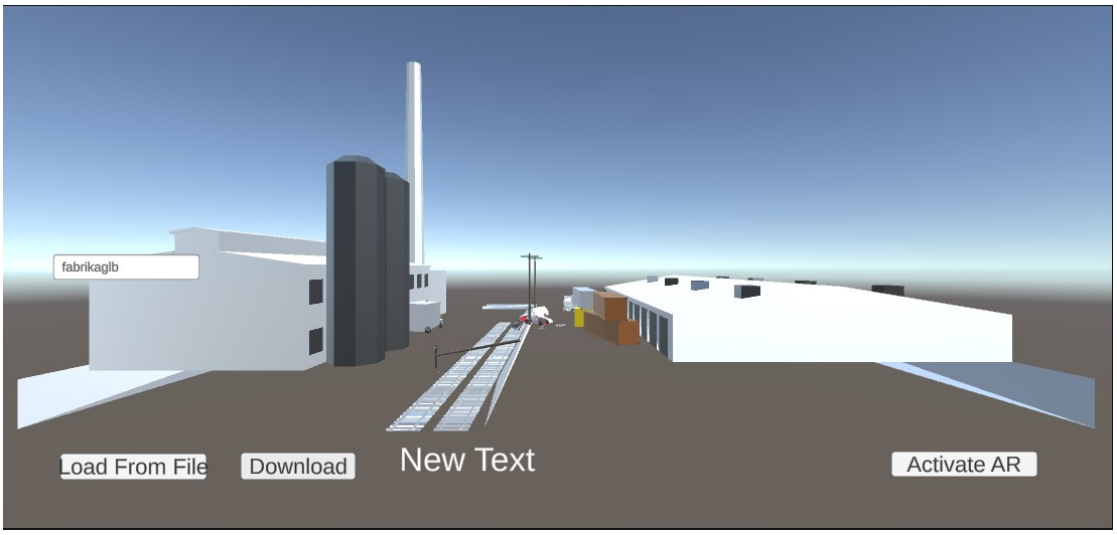


Screenshot 20 – Example Machine Model With AR(Unity)

After understanding the programming logic with Unity, our R&D activities continued as dynamically fetching data from API and creating models of machines imported with Unity. Thus, the data source will be single and we have ensured that the files uploaded by Er-Metal as .dae or .glb format can be opened with the Unity program.

We provided access to GET/POST methods related to machine models in the Machinaide API via Unity. After our meeting with DAKİK, they added a method to the API and enabled this method to send files in .dae format. We downloaded the .dae file from the API and opened it in Unity. However, we have seen that the coating data of the machines (machine paint, color, materials, if any, pictures on it, etc.) is not included in the content of the file with the .dae extension. For this reason, we decided that the file format we will receive is .glb and we asked DAKİK and Er-Metal for help in this regard. They shared their .glb formatted files containing the exterior view of the Er-Metal factory. We enabled this file to be read with Unity and displayed it in our mobile application by providing the output with augmented reality data.

We aim to show the coordinate-based sensor data with augmented reality by ensuring that this work we have done for the Er-Metal factory model is also done for the machines in the Er-Metal company.



Screenshot 21 – Er-Metal Factory Model (Unity)

3.6. Implementation Schedule for Demonstration (ERMETAL)

Action	Start Term	End/Planned Term	Responsible
Usecase and Requirements definition	**2019/2**	**2020/1**	**ERMETAL**
Determination of sensors and data collection equipments	**2019/2**	**2020/1**	**ERMETAL**
Determination of PLC needs	**2019/2**	**2020/1**	**ERMETAL**
Determination of PC needs	**2020/2**	**2020/2**	**ERMETAL, TEKNOPAR**

Determination of required software and platforms and (inc: OPC-UA / Influx DB, plug-in)	**2020/2**	**2020/2**	**ERMETAL, TEKNOPAR**
Purchasing and installation of sensors on one press (1x800T)	**2020/2**	**2020/2**	**ERMETAL**
Purchasing and installation of sensors on one robot	**2020/2**	**2020/2**	**ERMETAL**
Purchasing and installation of required software and platforms (inc: OPC-UA / Influx DB, plug-in)	**2020/2**	**2020/2**	**ERMETAL**
Purchasing and installation of energy modules on one press (1x800T)	**2021/1**	**2021/1**	**ERMETAL**
Establishing data flow from sensors	**2020/2**	**2020/2**	**ERMETAL**
Establishing data flow from energy modules	**2021/1**	**2021/1**	**ERMETAL**
Data collection	**2021/1**	**2023/1**	**ERMETAL, TEKNOPAR**
Creation of Digital Twin	**2020/2**	**2021/1**	**DAKIK,ERSTE**
Development of Digital Twin	**2021/1**	**2023/1**	**DAKIK,ERSTE**
Creation of HMI	**2020/2**	**2021/1**	**DOĞRU**
Development of HMI	**2021/1**	**2023/1**	**DOĞRU**
Machine learning and prediction	**2021/2**	**2023/1**	**DAKIK,ERSTE**
Realization of demonstration	**2022/2**	**2023/1**	**ERMETAL & ALL**
Solving detected problems / Making improvements	**2023/1**	**2023/1**	**ERMETAL & ALL**
Validation of demonstration**	**2023/1**	**2023/1**	**ERMETAL & ALL**

** Validation of demonstration will be evaluated and reported through the deliverable D1.6b ''Evaluation report and lessons learned for production process optimization demonstrator''.

Figure 4: Schedule for Description of demonstrator for production process optimization