Production Management System

Alexandru-Mircea Dima – 266006

Balkis Ibrahim – 260092

Nikola Vasilev – 260099

Supervisor: Knud Erik Rasmussen

VIA University College

Number of characters 96,633

Software Engineering

7th Semester

December 2020

Table of content

Abstract (Alex)

Developing a business intelligence, web-interfaced, monitoring solution for Trifork Smart Devices ApS, with the aim of producing a proof of concept product.

This product's aim is to graphically represent data stored in a cloud database and retrieved via a gateway system from a factory production line. The data is regarding the production quality as the number of operational failures. The performance as the number of produced items vs the planned number of items to be produced and the machine' functioning time in ten minutes units. All these values are reported in regards to an eight-hour working day/shift.

The solution has a strong dependency at the data acquisition layer on the gateway provided by the company and on the user-provided programmable logic controller. Other than that the cloud database provider, business intelligence and the web interface design are at the discretion of the developing team.

The software was developed according to the three tiers architecture using JSON format data objects as a means of data transfer between the tiers.

Acknowledgement

We, the developing team, would like to express our gratitude to Professor Knud Erik Rasmussen and Professor Poul Erik Væggemose, our research supervisors, for their patient guidance, encouragement and useful advice. We would also like to thank Mr Mads Havgaard Mikkelsen, for his advice and hardware and software support.

1 Introduction (Alex)

1.1 Background description

Trifork Smart Devices ApS (former MM Technology) is a division company that operates under the Trifork A/S umbrella and has a particular focus on the continuous development of gateway software and hardware platform known as Smart Interconnect Access (SIA). SIA platform eases the data logging and interconnection between systems and industrial plants.

The SIA platform delivers data solutions for industrial automation. It enables easy access to industrial data monitoring through dashboards and data integration across all the company's systems. In this way, efficiency gains can be implemented based on the right data analysis.

The solution is lightweight and comes as a plug and play device which can easily be connected to existing systems in a non-invasive way.

At the current time, SIA can access and log data from industrial connectors such as BACnet, M-Bus, Modbus, Siemens S7, Beckhoff TwinCAT ADS, Allen-Bradley Ethernet/IP, Universal Robot, KUKA KRC robots. As there is a large variety of industrial connectors in use in the industry, SIA continuously aims to enrich the existing collection of connectors. One of the on purpose limitations of SIA is that the data logged by a connector is directly available only at a local network level.

At the request of clients, a REST API connector can be added in the system so data can be sent through the internet to an endpoint.

1.2 Delimitations

The developing team established the following delimitations that will set the boundaries of the project:

- The data is acquired from only one PLC.
- The communication between the data acquisition and data storage is unidirectional, from acquisition to storage.
- The team can not have direct communication with the clients.
- The user interface language is English.
- This tool is meant to be a proof of concept and is not intended for commercial usage.
- The team does not develop any hardware.
- The team is not responsible for the generation of data at the factory automation level.

2 Analysis (Balkis)

As stated at the Delimitations of the project, the company did not provide communication with real clients. Therefore, the group made an imaginary persona which can be found in Appendix 1. The following requirements were created to meet the persona’s needs.

2.1 Production Monitoring: (Balkis)

The company could not provide us with real production data there for the group had to make up some data. We did some research on what data needed to monitor the production. And we found out that OEE which stands for Overall Equipment Effectiveness is the most interesting parameter to monitor the production.

Defining OEE:

OEE measures the efficiency of a machine. OEE takes into consideration the Availability, Performance, and Quality of a machine.

In mathematical terms, OEE = Availability * Performance * Quality.

Calculating OEE:

For the purposes of calculating OEE, the underlying metrics are as follows:

Availability - Availability can also be referred to as Uptime. This is the amount of time the machine is in cycle, producing parts compared to the amount of time the machine is scheduled to be in the cycle. Any unplanned downtime will result in a lower availability percentage.

Can be calculated from → Run Time / Planned Production Time x 100

Performance - Performance refers to the speed that parts are produced while the machine is in the cycle. Can be calculated from → actual output / theoretical output x 100

Quality - The quality metric is the percentage of good parts out of total parts produced. Any parts rejected during a run will lower the quality percentage of a machine.

Can be calculated from → good product / total product x 100

2.2 Requirements (Balkis)

The following section of the analysis represents the user stories and the non-functional requirements of the system. It clarifies what the system will be able to do when it is implemented.

2.2.1 Functional requirements:

This part of the report shows the functional requirements of the project, which are presented as user stories. The purpose of this section is to help the developers understand what the client wants from the system.

1. As a clerk and production assistant, I want to be able to add new customers so that I can save their contact info.
2. As a clerk and production assistant, I want to be able to delete customers from the system so that I can remove them from the system.
3. As a clerk and production assistant, I want to be able to edit customers so that I can change their data.
4. As a clerk and production assistant, I want to be able to get a list of all customers so I can have answers to specific questions.
5. As a clerk and production assistant, I want to be able to create orders so that I can submit it to the production area.

6 As a clerk and production assistant, I want to be able to delete orders from the system so that I can remove cancelled orders from the system.
7 As a clerk and production assistant, I want to be able to edit orders so that I can make changes to the orders.
8 As a clerk and production assistant, I want to be able to get a list of all orders so that I can do some statistics.
9 As a clerk and production assistant, I want to be able to add new products to the system so that I can decide what mobile cases we can make.
10 As a clerk and production assistant, I want to be able to delete products from the system so that I can remove unwanted mobiles from the system.
11 As a clerk and production assistant, I want to be able to edit products so that I can make a change to the mobiles name.
12 As a clerk and production assistant, I want to be able to get a list of all products so that I can tell what mobile cases we produce.
13 As a clerk and production assistant, I want to be able to add new design for producing a specific mobile case to the system so that I can decide what case design we can produce.
14 As a clerk and production assistant, I want to be able to delete design from the system so that I can remove the unwanted designs from the system.
15 As a clerk and production assistant, I want to be able to edit the design for so that I can make changes to the design description.
16 As a clerk and production assistant, I want to be able to get a list of all mobile case designs so I can tell what designs we make.
17 As a clerk and production assistant, I want to see a chart of the production Availability, so that I can make a timely intervention.
18 As a clerk and production assistant, I want to see a chart of the production performance, so that I can make a timely intervention.
19 As a clerk and production assistant, I want to see a chart of the production quality, so that I can make a timely intervention.
20 As a clerk and production assistant, I want to see a chart of the production OEE, so that I can make a timely intervention.
21 As a management team member, I want to see averages for each production monitoring parameter so I can develop improvement plans.
22 As a management team member, I want to see history for produced items so I can develop improvement plans.
23 As a management team member, I want to have a structured history of data to create reports.

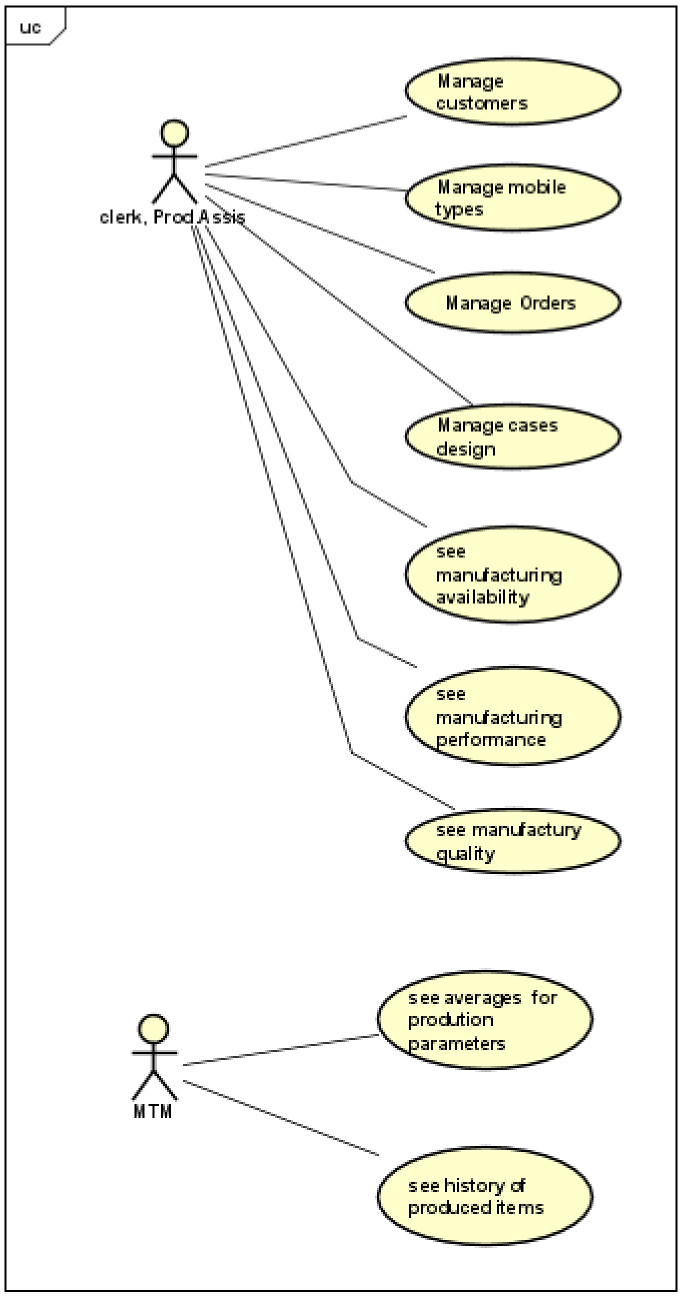
2.2.2 Non-Functional Requirements

This section presents the non-functional requirements for the system and tells how the system should work.

1. The data must be retrieved from the PLC in less than five-second since the request.
2. The system must be able to send data from plc to the database every ten minutes.
3. The system must update the visualisation interface every ten minutes with the data coming from the plc.
4. The tool must be developed in English.
5. The system must use business intelligence tools to analyse data.
6. The system must be able to expose data using APIs.
7. The system must be able to provide a responsive user interface.
8. The project should include all the documentation needed to understand the functionality and system structure for future maintenance and updates.
9. The system must be able to make an analysis of the data and create reports.
10. The system must be compatible with Mitsubishi MELSEC-Q Series PLCs.
11. The system should be able to retrieve the following data types from the PLC:
    a. Integers
    b. Floating point numbers
    c. Boolean values.

2.3 Use case diagram: (Balkis)

The use case diagram is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. It visually clarifies what the system will be able to do when it is implemented.


uc
clerk, Prod.Assis
Manage customers
Manage mobile types
Manage Orders
Manage cases design
see manufacturing availability
see manufacturing performance
see manufactury quality
MTM
see averages for prodution parameters
see history of produced items

Actor descriptions: (Balkis)

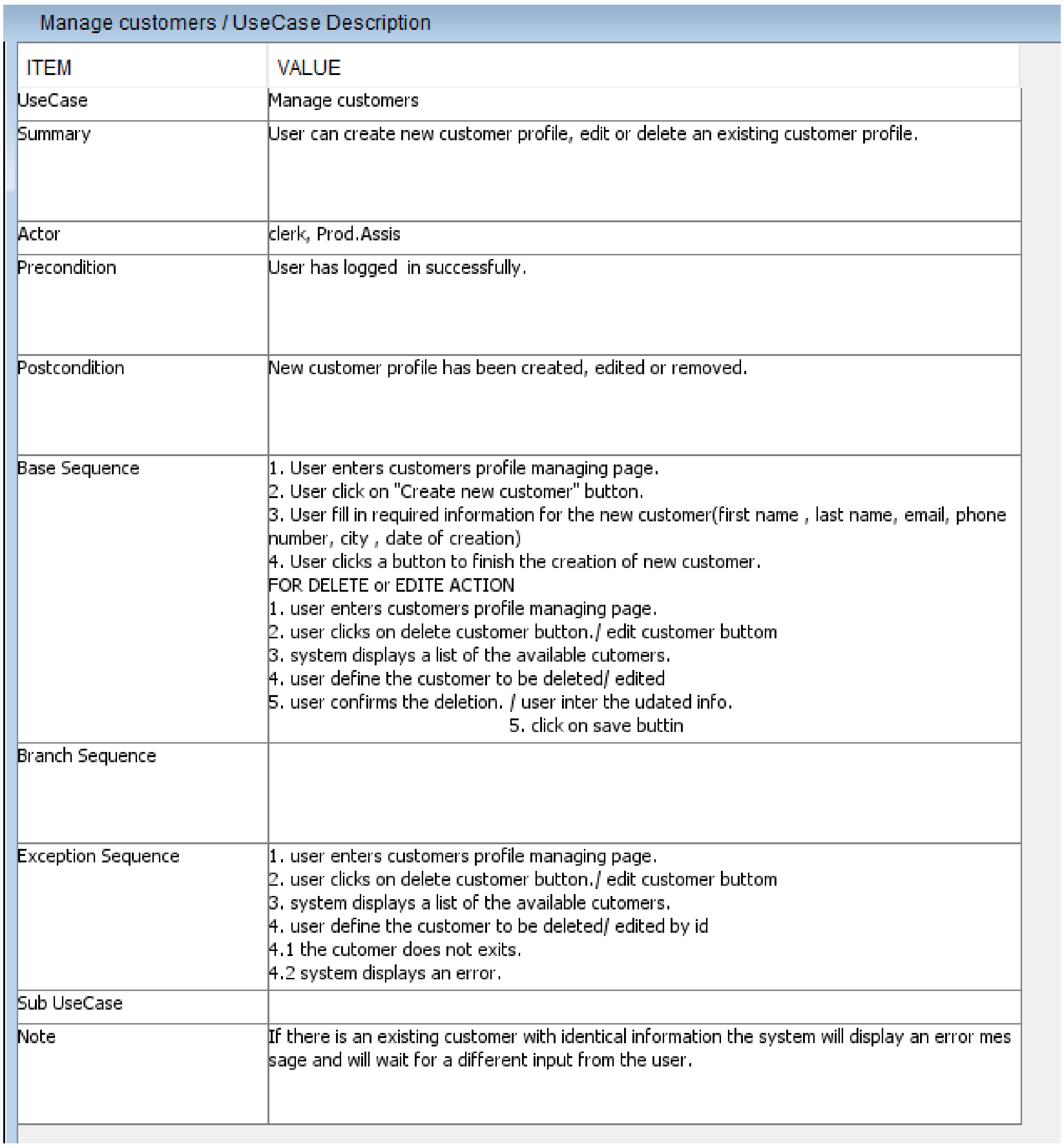
The actors in this project have three primary roles, interacting with the system.
Clerk, production assistant, and management team member.

Clerk and production assistant:
One person is taking two roles; he works as both a clerk and a production assistant.
His job is to manage customer orders and monitor the production line on a daily basis.

Manager team member:
The manager team member represents the person who has access to the data warehouse and the business intelligence tool.

2.4 Use cases description: (Balkis)

The purpose of the use case description is to show additional information to the corresponding use case. It is used as support to track the requirements for the system and give a more descriptive use case scenario. Only one of the use cases descriptions is presented in this section. for more please visit Appendix 3



Manage customers / UseCase Description

ITEM	VALUE
UseCase	Manage customers
Summary	User can create new customer profile, edit or delete an existing customer profile.
Actor	clerk, Prod.Assis
Precondition	User has logged in successfully.
Postcondition	New customer profile has been created, edited or removed.
Base Sequence	1. User enters customers profile managing page. 2. User click on "Create new customer" button. 3. User fill in required information for the new customer(first name , last name, email, phone number, city , date of creation) 4. User clicks a button to finish the creation of new customer. FOR DELETE or EDITE ACTION 1. user enters customers profile managing page. 2. user clicks on delete customer button./ edit customer buttom 3. system displays a list of the available cutomers. 4. user define the customer to be deleted/ edited 5. user confirms the deletion. / user inter the udated info. 5. click on save buttin
Branch Sequence	
Exception Sequence	1. user enters customers profile managing page. 2. user clicks on delete customer button./ edit customer buttom 3. system displays a list of the available cutomers. 4. user define the customer to be deleted/ edited by id 4.1 the cutomer does not exits. 4.2 system displays an error.
Sub UseCase	
Note	If there is an existing customer with identical information the system will display an error mes sage and will wait for a different input from the user.

2.5 Test cases (Nikola)

2.5.1 Functionality testing:

The following table shows test scenarios where different features of the tool are checked.

Test Scenario	Test Case	Test Steps	Test Data	Expected result
Create a new customer	Check response on entering a non-valid first name, last name, email, phone number, address.	1. Launch application 2. Enter first and last name, email, phone number, address 3. Click the "Create" button	First name: George12 Last name: Smith12 Email: george.smith23432gmail.com Phone number: 555ab Address: 4rfe	A new customer is not created.

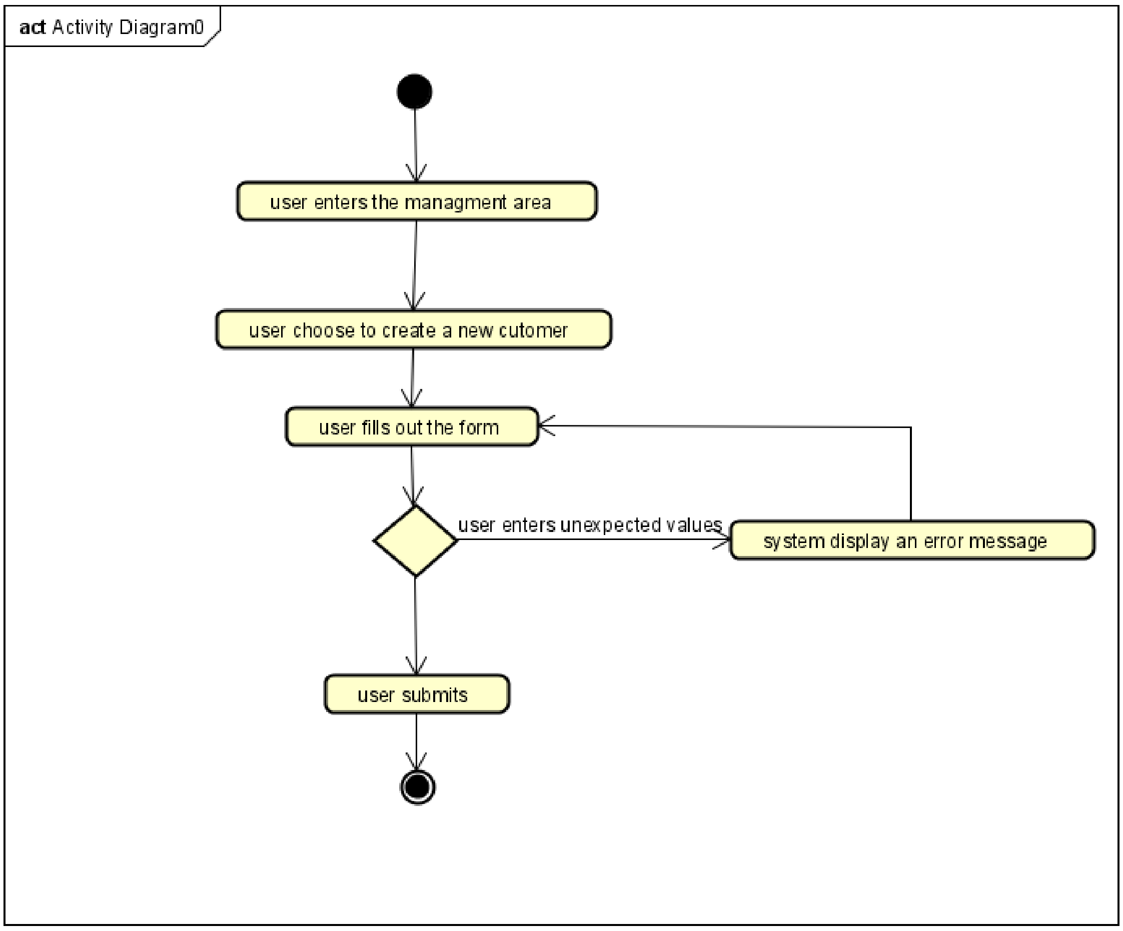
Create a new customer	Check response on entering a valid first name, last name, email, phone number, address.	1. Launch the application 2. Enter first and last name, email, phone number, address 3. Click the "Create" button	First name: George Last name: Smith Email: george.smith23432@gmail.com Phone number: 5555 Address: Horsens, Denmark	Successful creation of new customers.

Test Scenario	Test Case	Pre-Conditions	Test Steps	Test Data	Expected Result
Create a new order	Check response on entering valid order identification number, customer identification number, order due date.	There is an existing customer.	1. Launch application 2. Enter order identification number, customer identification number, order due date. 3. Click “Create button.”	Order Identification number: 7653 Customer identification number: 12345	Successful creation of new order.
Create a new order	Check response on entering non-valid order identification number, customer identification number,	There is an existing customer.	1. Launch application 2. Enter order identification number, customer identification number, order due date.	Order Identification number: 7653! Customer identification number: 12345!	The new order is not created.

	order due date.		Click “Create button.”		
Create a new order	Check response on trying to edit existing order on entering valid data.	There is an existing order. There is an existing customer with id =1234.	1. Launch application 2. Search for an order in the search area by identification number. 3. Click the “Edit existing order” button.	Customer identification number: 1234	The existing order is changed.

2.6 Activity diagrams: (Balkis)

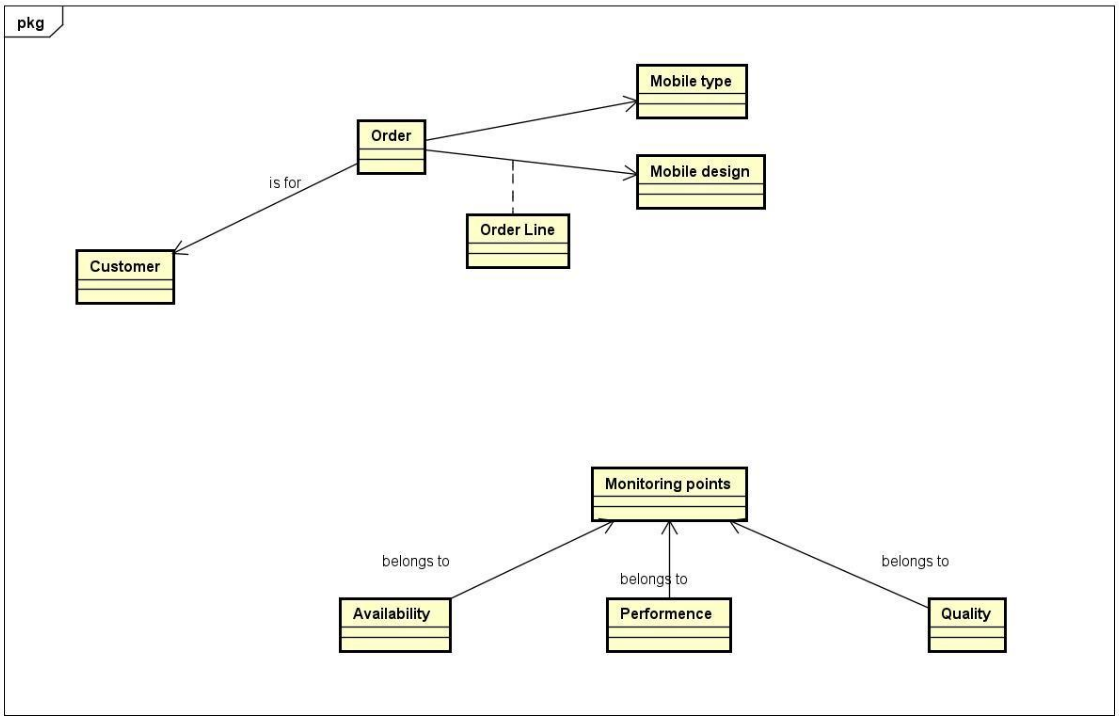
The following section describes the activity diagrams which represent the use cases detailed above. Each of the use cases has different scenarios to avoid complexity the following activity diagram has been created to present one of the scenarios which is create customer. To see the rest of the activity diagrams, please visit Appendix 4.


act Activity Diagram0
user enters the managment area
user choose to create a new cutomer
user fills out the form
user enters unexpected values
system display an error message
user submits

The diagram above shows the steps the user takes in the application in order to achieve the desired goal. He first enters the management area and chooses to create a new customer. The user will fill out a form with the new customer's info.
The system will not allow the user to submit unless the email was from type "example@someting.com" and the mobile is only numbers.

2.7 Domain Model: (Balkis)

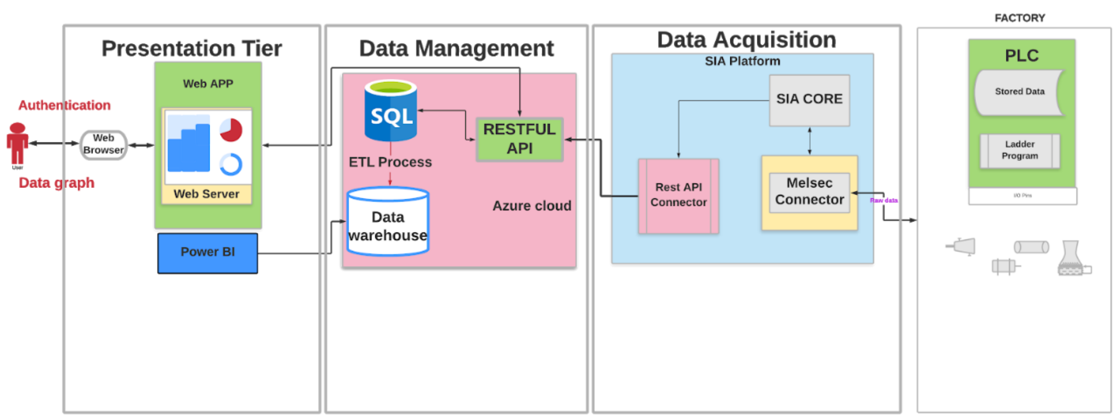
We created the Domain Model to understand the relationships between the objects that take part in the system. It shows an overview of parts connected to what on an abstract level. It’s an important thing to reflect on since it gives the developer a clear overview of how the system is intertwined.
The clerk will email the production area when a new order is created; therefore, there is no connection between the orders and the machines in the factory.



3 Design

This section provides information about the design choices regarding the technical part of the system and the diagrams that are made while designing the system. This chapter shows how the system is structured, and describes the technologies being used to implement it,

- **System architecture overview:**



The figure above represents the system design diagram. It represents three tier architecture of the system. The three tiers are Data Acquisition tier, Data Management tier and Presentation tier. Data is provided by production equipment and retrieved using PLC. PLC can store the data. Ladder Program stands for program logical controller. PLC can transfer raw data to SIA CORE using Melsec Connector. Data acquisition tier communicates with the Data Management tier using the Rest API Connector. Data transfer is one direction. Data Management can not send data to the Data Acquisition tier, it can only receive data. Data Management tier resides on Azure Cloud platform. SQL Database can send and receive data to the RESTFUL API. SQL database can send data to the Data warehouse using ETL Process. Data from the Data warehouse can be displayed in the Presentation tier by Power BI. RESTFUL API can send and receive data from Web APP in the Presentation tier. Web APP sends

Http requests to RESTFUL API. Web APP is hosted on Web Server. Users can see Data graphs by sending and receiving Authentication.

Each member of the group has taken the responsibilities for one the tire.
The data acquisition sup system assigned to Alex.
The data management together with part form the presentation tier (Power BI) is assigned to Balkis.
The web APP is assigned to Nikola.

From this phase, the group members branched out into three; therefore, the rest of the sections will have subsections to better explain each member's relevant parts of the system.

3.1 Front-end design (Nikola)

Angular is framework for building client applications in HTML and TypeScript which compiles to JavaScript. Angular application consists of modules, components, templates, metadata, data binding, directives, services and dependency injection. Components which are basic UI building blocks. Components are subset of directives which are associated with a template. Only one component can be associated with a given template. Angular project has *root component* which connects the component hierarchy with a page document object model (DOM).A module is a mechanism to group different components, directives, pipes and service which are all related in such way that they can be combined with another modules. Each Angular app has a *root module,* called *AppModule*, which provides the bootstrap mechanism which launches the application. An Angular template combines the HTML with Angular markup that can modify HTML elements before they are displayed. Template directives provide program logic. Binding markup connects the application data and the DOM. Angular supports two types of data binding: event binding and property binding. Metadata tells Angular how to process a class. It decorates the class so that it can configure the expected behavior of a class. Services in Angular are used for data or logic that is not associated with a specific view to be shared across components. Dependency injection is used to inject service into a component, giving the component access to that service class.

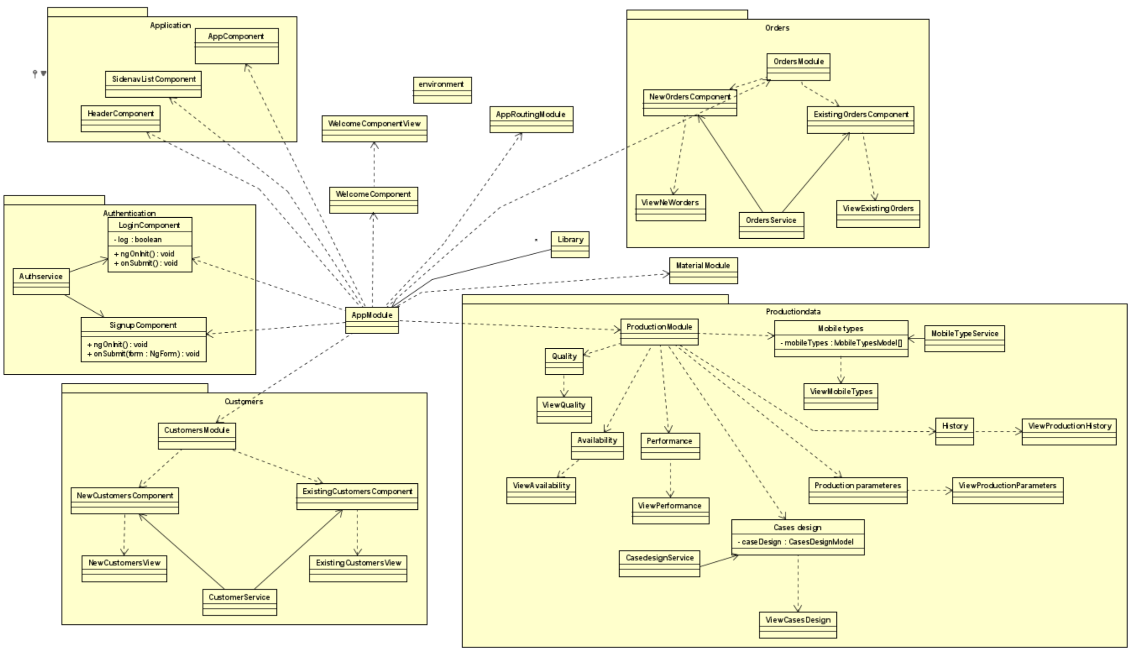
Angular can create Single Page Applications (SPAs). In this case only one page is downloaded from the server and after that everything is handled in the browser. This way response speed is increased because there is no need to wait for server responses in most cases. One single HTML page *index.html* is continuously re-rendered by Angular. This client layer is the presentation level of the system providing end users with an interface in order to display information related to offered services. The client layer communicates with the other system components by sending and receiving responses. Users can request, explore and visualize specific data, perform querying actions in order to receive data, discover and explore data. This application layer contains selected software to and methodologies to be applied on the submitted

data and it establishes the connection and performs the needed data transmissions between client and data layer.
This application uses Angular Material. It is third party package which can be used in Angular projects. In its core it is an Angular Component Suite. It is a pre-build and Styled Angular components, which follows Material Design Spec, created by Google. Angular Material focuses on component building utilities and concrete component implementation (using Material Design Spec). First feature set is bundled in a package called “@angular/cdk”. This does not include any Angular Material styling, but includes a lot of utility and helper methods and classes for rendering overlays, positioning elements etc. Second package contains styled Angular Material components which are built on the first package. It is bundled in “@angular/material”.

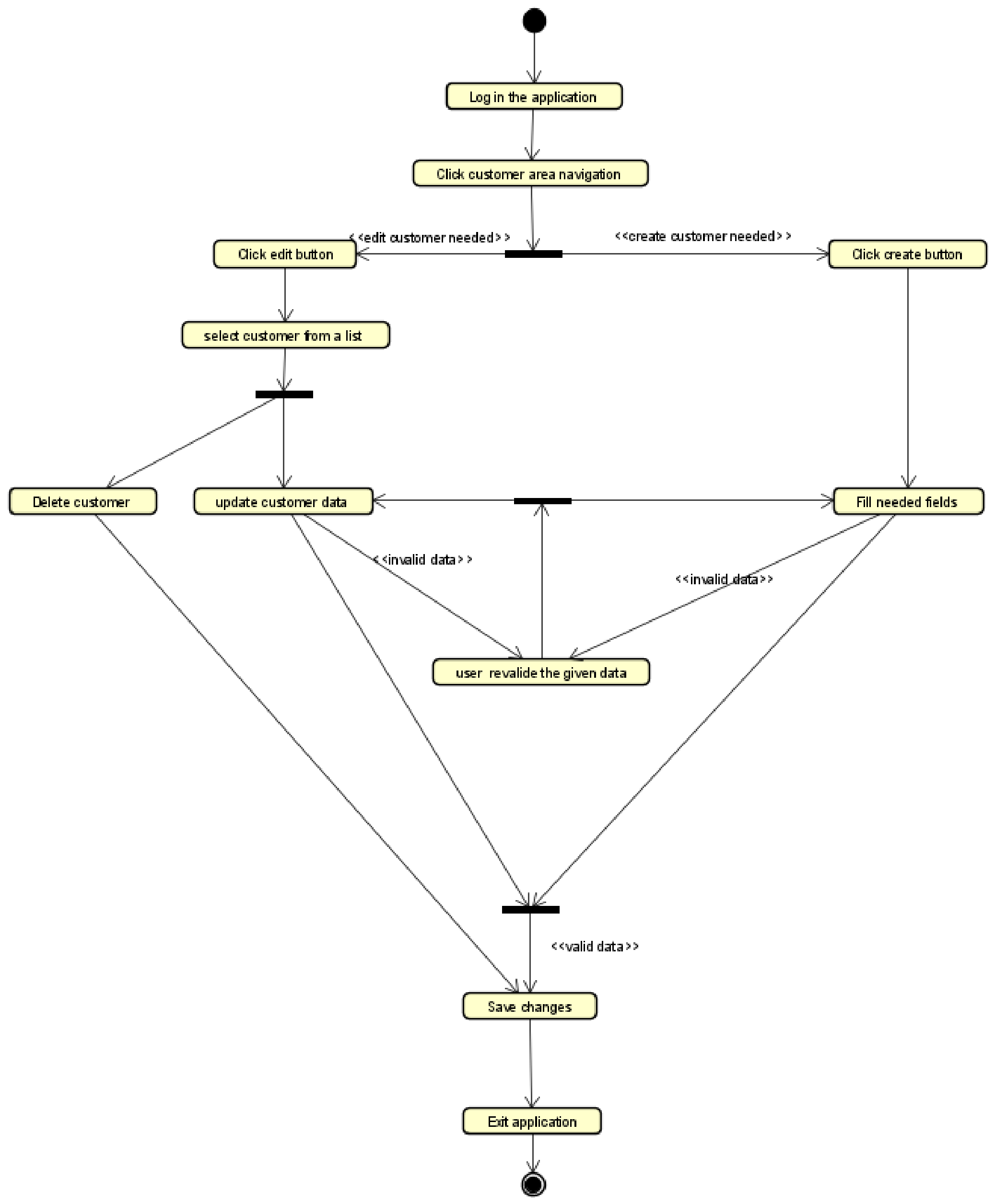
Latest Angular framework version supports TypeScript 2.9 which can read external and local JSON files. Reading local JSON files is out of the scope for this project. This application will read and extract data from an external database in JSON format. Angular provides client HTTP API for Angular applications. This is the *httpClient* service class situated in *@angular/common/http.* This service allows to request typed response objects. *HttpClient* request can declare the type of the response object. This makes consuming the output easier and easier for interpretations. Specifying the response type is a declaration of TypeScript which treats the response as being of given type. This feature is a build-time check and it doesn’t ensure that the server will respond with an object of desired type. It depends on the server API to make sure that desired type is returned. *HttpClient* can also streamline error handling and request and response interception.
Observable pattern is used in Angular to provide support for passing messages between parts of the system. They can be used also for event handling. An object *subject* has list of dependencies which are called *observers* and notifies them automatically of state change. A function is defined for publishing values and it is not executed until consumer is subscribed to it. Observable delivers multiple values of types literals, messages or events. Application code cares only about subscribing to consume values because teardown logic and setup is handled by the observable. When observable is done it unsubscribes.

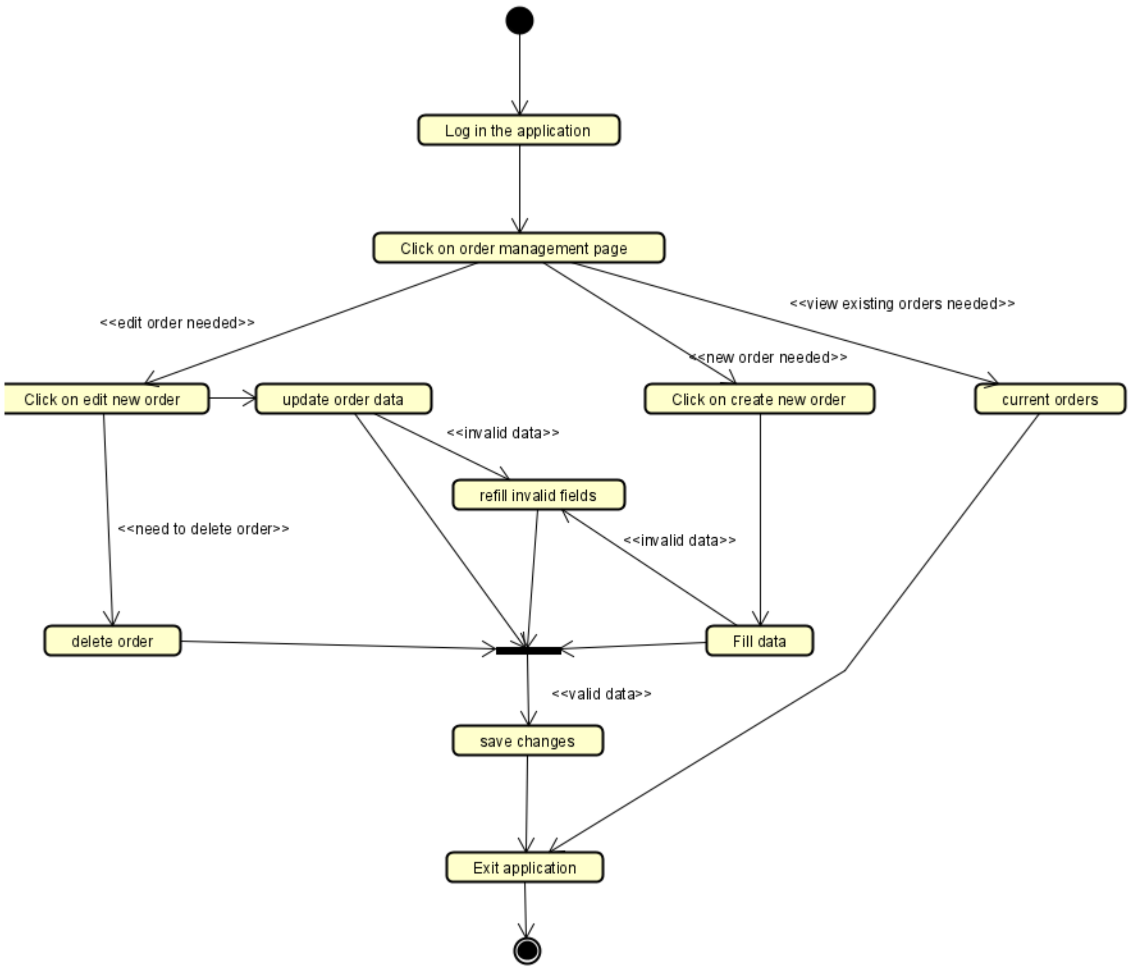
This application consists of multiple building blocks. Upon launching the application, the user views the welcome screen which is part of Welcome Component. From this home page the user can sign in/log in.
In Authentication section users can log in/sign in by filling valid credentials to get access to application functionalities. It contains User model which defines which credentials are valid. Sing in/ log in are divided into two components which use authentication service to authenticate the user.
From home page user can navigate to Customer section where user can create new customer, search through customers to edit their information or delete them if needed. Customer section contains Customer Model which defines the data definition of a Customer. Customer section contains two addition components for creating new customer and second component for searching and editing/deleting customers' information. In order to manage this customers' data a service is needed. Creating new customer and editing/deleting existing one uses service.
From home page user can navigate to Orders section. There user can create a new order, search through existing orders and manipulate their data or delete an order completely. Order section contains order model which defines the data definition for an Order. Order section contains components for creating new order and editing/deleting existing order. These actions are executed with help of service.
Production data section can be accessed from home page after successful sign in/log in. There user can create, edit, delete mobile types and case designs. User can visualize production quality, availability, performance, production parameters and view production history. To manage that production quality, production availability, production performance, production parameters and production history have separated components. Each component (except production history) has a model which defines the data definition for each one of them. This section contains three services: Mobile type service allows the user to add mobile type in database, delete mobile type in database or edit mobile type in database. Case design service allows the user to add case design in database, delete case design in data base or edit case design in database. Production data service allows the user to view production quality, availability, performance, production parameters and view production history.

This application has shared modules, such as UI module which contains features shared by all views in the application. Header and side navigation components ensure that the application will be responsible on different devices and screens. Routing between different components is defined in separated module called application Routing. For better organizing of materials which are imported separated Material module is to be created.
Empty class diagram showed below represents the classes and relationships between then in this Angular application. Full class diagram can be seen in Appendix under *ClassDiagramFrontend*.



Activity diagram below depicts a usual workflow for creating, editing, or deleting customer. First user log in the application. Then navigates to the customer area navigation. Customer area navigation consists of two tabs. One tab is for creating a new customer and the second one is used to filter through array of customers by their identification. If user wants to create a new customer, he/she must navigate to *Create* tab and click create button. Then user is prompt to fill in valid data. If user enters invalid data, he will be asked to revalidate the data. After entering valid credentials user can save changes. I f user wants to delete or edit and already existing customer, he/she must navigate to *Edit* tab in Customer area navigation. There the user filter through existing customers by their *id*. Then user can delete chosen customer from database and save changes. If user wants to edit customer information, he/she must search for the customer by *id* and then fill in the desired fields with valid data. If If user fill in invalid data, he/she will be prompts to enter valid data.


Log in the application
Click customer area navigation
<<edit customer needed>>
<<create customer needed>>
Click edit button
Click create button
select customer from a list
Delete customer
update customer data
Fill needed fields
<<invalid data>>
<<invalid data>>
user revalide the given data
<<valid data>>
Save changes
Exit application

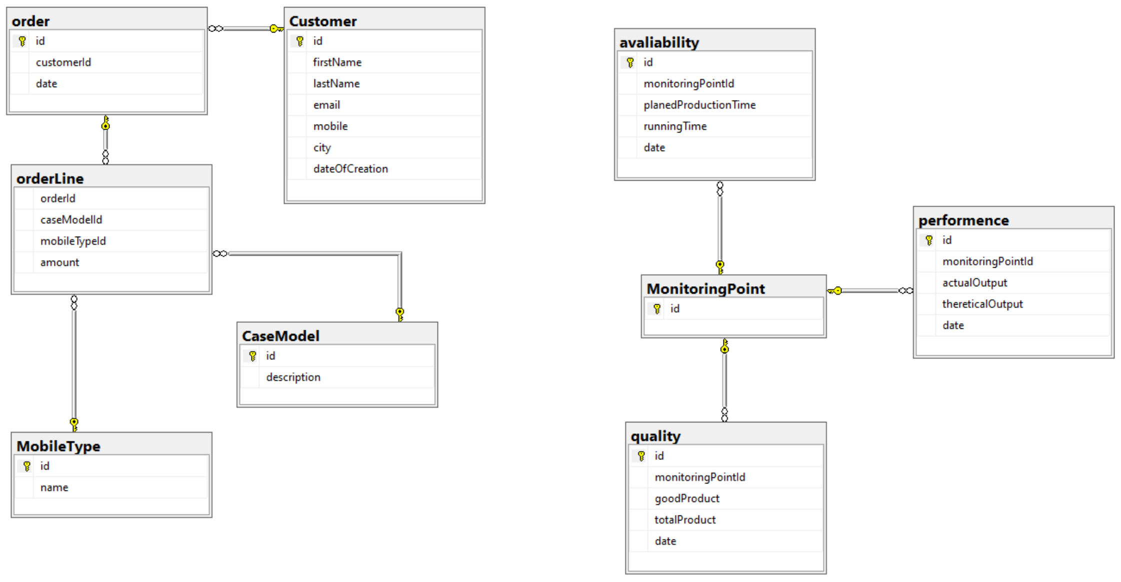


Activity diagram above depicts a case when user wants to create, edit or delete an order or wants to see a list of all existing orders. First user must log in the application. Then user navigates to Orders area navigation. Orders area navigation is divided into three tabs. First tab is responsible for creating orders, second tab is responsible for editing/deleting orders and third tab is responsible for displaying existing orders. If user wants to create an order, he/she must navigate to *Create* tab and click Create button. Then user must fill in valid data, if user fails, he/ she will be prompted to re-enter valid data. After entering valid data user can save changes. If user wants to edit or delete existing order, he/she must navigate to Edit tab. In this tab if user wants to edit an existing order user search through orders by their *id*. Upon choosing desired order user enters valid data. If user enters invalid data, user will be asked to re-enter valid data. Then user can save changes and order will be edited. If user wants to delete an existing order user must navigate to Edit tab and search through orders by their *id*. After selecting desired order, user can delete it and save changes. If user wants to visualise all existing orders, user must navigate to the third tab of this component. There user can see all current orders.

3.2 Backend design (Balkis)

3.2.1 Database:

The Domain Model that we created in the Analysis phase has led us to the following EER diagram.



The EER diagram has the following entities:

- strong entities (customer, case-model, mobile-type, monitoringPoints).
- weak entities (order, performance, quality, availability).
- composite entity(order-line).

Customer entity: keeps essential data about the manufactory Customers (name, email, mobile, address, and when did they become clients of the factory.
Order entity: keeps records about the customer who has made the order (foreign key: customer-id) and on what date the order has been created.
Mobile type entity: save data about the mobile type that the factory makes cases for.
Case-model entity: keeps the information about the different design that the factory produces (number of the case model and a short description of the design).
A bridge entity (order line): has been used to break down the many to many relationships between order with (mobile type and case-model), it has an amount

attribute to keep track of items number that the client ordered from each mobile and case-type.

We thought about making a connection between the orders and machines in the factory. This way we could provide a better experience to the user since it will allow him to see how much left to finish the desired order. but we came to the decision that we wait with that until the next version of the system. bases on this decision the following entities have been added to monitor the production:

Availability entity: plannedProductionTime attribute has int type from 0 to 480 it will be incremented by 10 for each row.
runningTime attribute of int type less than 480. It will store for how long the factory has been working from the beginning of each working day till the end of each working day, the date attribute of type timestamp stores when the value has been retrieved from the monitoring points.
Quality entity: goodProducts attribute stores the number of good items that have been produced during the last period, totalProduct attributes is the number of all the items that the factory-made throughout the day.
Performance entity: theoreticalOutput attribute, is how many items should the manufactory produce in 10 minutes, it will also be incremented by 10 for each row, the actual output is how many items did the manufactured made in total from 8.00 am till the current time.
Monitoring Points entity: has only one attribute to keep track of the monitoring locations.

All the data that is related to production is retrieved from the plc every 10 minutes.

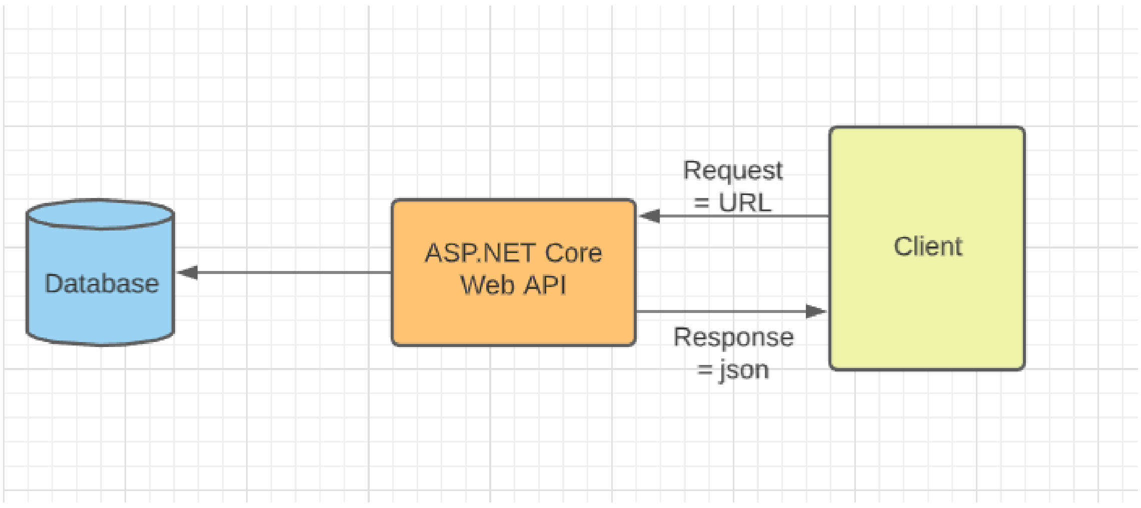
The assumptions made for data testing:

- The system will retrieve data from plc every 10 minutes; therefore, all the parameters will be calculated in a 10-minute interval.
- The planned production time is eight hours per day which means 480 minutes.
- The estimated time to produce one item is one minute.

3.2.2 APIs

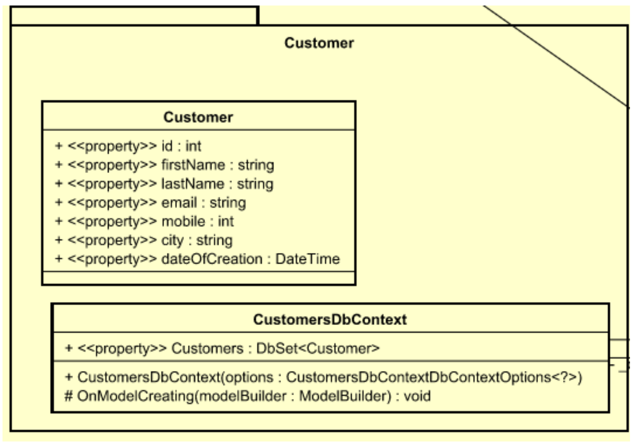
- **Backend Architecture: (Balkis)**

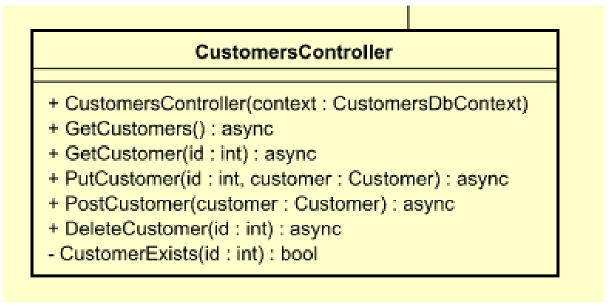
  The rich picture below illustrates the architecture of the backend. The client requests the ASP.NET Core application to either get data or execute some business logic; the application will communicate with the database to read/write. The application will serialize the response and send it to the client.



- **Class Diagram: (Balkis)**

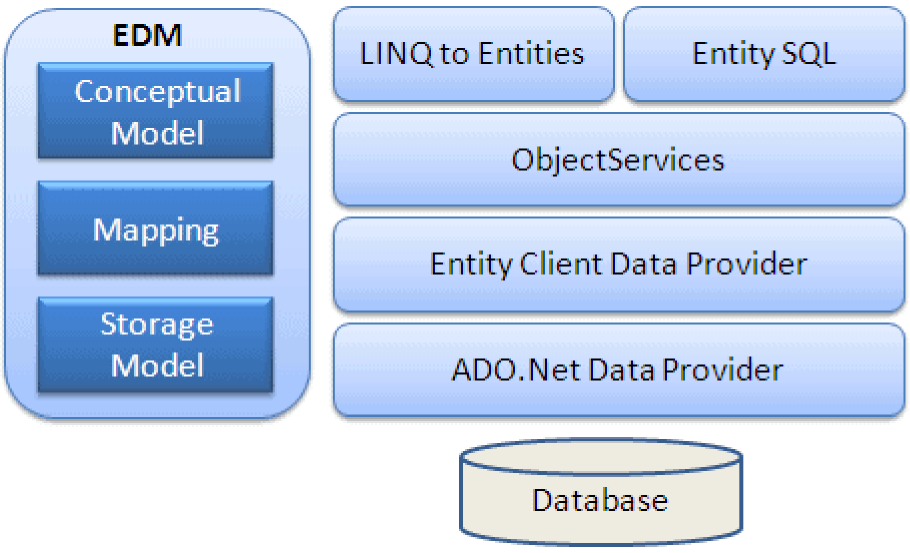
The class diagram provides an overview of the infrastructure of the system.
The API is designed as a RESTful API. The API class diagram consists of two packages: the Model and the controller; The Model package consists of sub-packages for each Entity. The controller package has one controller class for each entity that exists in the database, and the classes are responsible for defining endpoints. Each one of the controllers has the CRUD methods which are handled by the HTTP request. The DBContext classes in the model package inherit the EF core DbContext class, and they hold the dbsets of the tables. When a client sends a request to the API, ASP.Net core route requests to the right controller action, the controller class will communicate with the appropriate DBContext class which, used to query or save the data to the database based on the model. DBContext uses entity framework to read and write from the database. In the next section, I will talk more about the entity framework. The following snips show apart from the class diagram; the full class diagram can be found in
The API class diagram can be found in the Appendix; please visit Backend → Appendix A.


Customer
Customer
+ <<property>> id : int
+ <<property>> firstName : string
+ <<property>> lastName : string
+ <<property>> email : string
+ <<property>> mobile : int
+ <<property>> city : string
+ <<property>> dateOfCreation : DateTime
CustomersDbContext
+ <<property>> Customers : DbSet<Customer>
+ CustomersDbContext(options : CustomersDbContextDbContextOptions<?>)
# OnModelCreating(modelBuilder : ModelBuilder) : void


CustomersController
+ CustomersController(context : CustomersDbContext)
+ GetCustomers() : async
+ GetCustomer(id : int) : async
+ PutCustomer(id : int, customer : Customer) : async
+ PostCustomer(customer : Customer) : async
+ DeleteCustomer(id : int) : async
- CustomerExists(id : int) : bool

- **Entity Framework (EF): (Balkis)**

Entity Framework is an open-source ORM framework for .Net applications supported by Microsoft. It allows the developer to work at a higher level of abstraction when dealing with data. This means that EF will eliminate the need for most of the data-access code that is needed to write; since it enables developers to work with data using objects of domain classes without focusing on the underlying database tables and columns.
The following rich picture shows an overview of the Entity Framework Architecture.



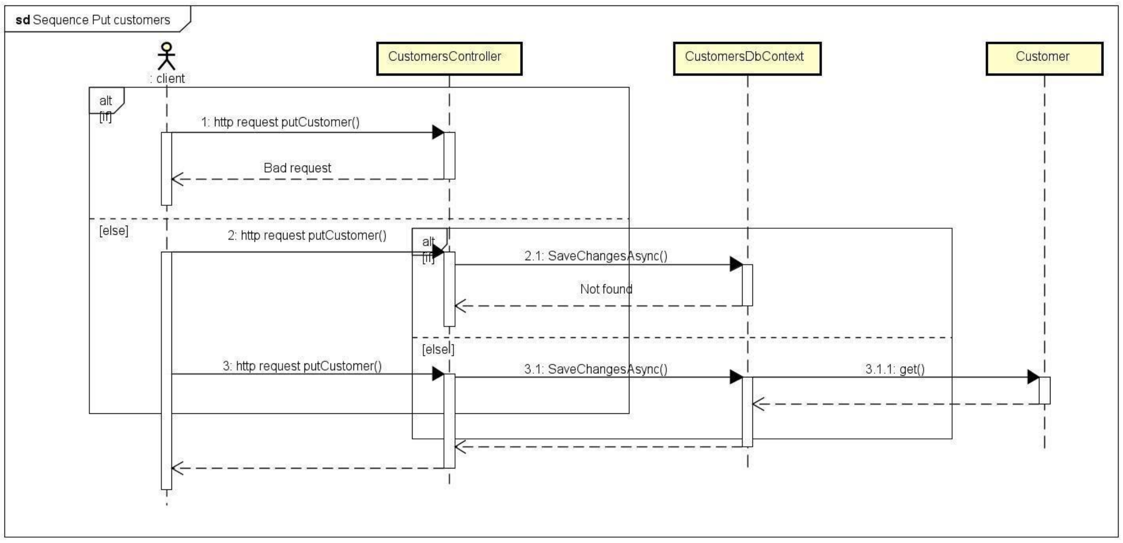
Entity Data Model (EDM) consists of three parts.
Conceptual Model: The conceptual model contains the model classes and their relationships.
Storage Model: The storage model represents the database design model.
Mapping: Mapping consists of information about how the conceptual model is mapped to the storage model.
LINQ to Entities: LINQ-to-Entities (L2E) is the query language used to write queries against the object model. It returns entities.
Entity SQL: Entity SQL is another query language which is not used in the system.
Object Service: Object service is the main entry point for accessing data from the database and returning it. Object service is responsible for converting data returned from an entity client data provider (next layer) to an entity object structure.
Entity Client Data Provider: This layer's primary responsibility is to convert LINQ-to-Entities or Entity SQL queries into a SQL query that is understood by the underlying

database. It communicates with the ADO.Net data provider, which in turn sends or retrieves data from the database.

ADO.Net Data Provider: This layer communicates with the database using standard ADO.Net.

- **Sequence diagram: (Balkis)**

The following graph shows the sequence diagram for the putCustomer method. When the client sends an HTTP put request, the controller class will receive the request and first check the id parameter from the endpoint if it is equal to id from the request body, two scenarios are expected here. One the ids are not the same in this case the controller will return “a bad request message” to the client. The second scenario, the ids are the same. So, it will call the SaveChangesAsync method from the CustomerDbcontext class; this method will check if any of the customers has the requested id. If it did not find any, it would throw back the request with a not found message. If it found the desired customer, it will update his info, send a request to the customer class and get the updated row from the database.



Please visit Backend →Appendix B for sequence diagrams.

- **Design Pattern: (Balkis)**

  SOLID principles have been followed during the design and implementation of the system, making software design more understandable, flexible, and maintainable.

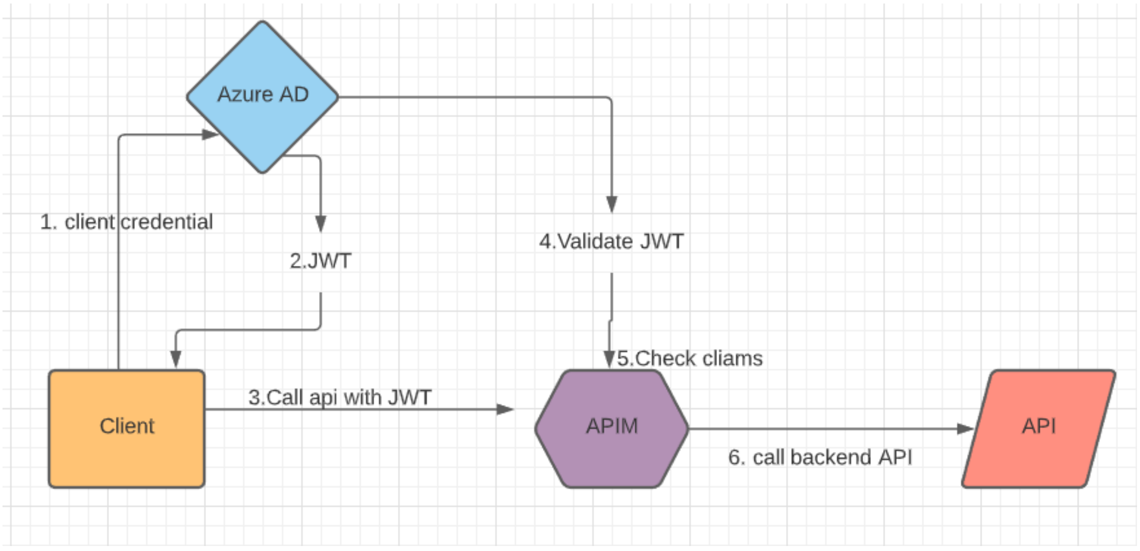
  Single responsibility principle:
  All the classes have only one responsibility, and no responsibilities are separated across multiple courses. The reason is that the more changes requested in the future, the more changes the class need to apply. This makes the classes easier to understand, implement, test and more comfortable to change.

  Open Closed Principle:
  The classes are designed, so they are opened for extensions but closed for modifications. The new features should be implemented using the new code, but not changing existing code, so it reduces the risk of breaking the existing implementation.

  Liskov Substitution Principle:
  When inheriting a class, the derived class can extend its base class without changing its original implementation.

- **APIs Protection: (Balkis)**

The rich picture illustrates an overview of the design for protecting the API.



OAuth 2.0 is applied using Azure API management (APIM) and Azure Active Directory (AAD), which provides an authorisation solution for the API. AAD is acting as the authorisation server that contains the registered applications' information and access control permissions; the APIM is the resource server which administers access to the API.
Azure AD will generate token endpoints that the client can call with his ID and secret. The client will send an Authorization request to AAD token endpoint using his credentials. If the client gets authorised, Azure AD will issue JWT (JSON Web Tokens) and send it back to him. The client then can call the APIM with the JWT passed in the header request. Azure AD will validate the JWT, APIM will check whether the client has the proper claims for the specific APIM operation. The client will get 401 response status with "Unauthorized API call" if he does not have the claim and 200 response along with expected response if he has the proper claim.

3.2.3 Data warehouse **(Balkis)**

- **Dimensional Model**

  Dimensional modelling (DM) is part of the Business Dimensional Lifecycle methodology developed by Ralph Kimball which includes a set of methods, techniques, and concepts for use in data warehouse design (Kimball et al., 2008). Dimensional models are logical and understandable design techniques utilized for data warehouses, and they divide the world into measurements (facts), and context(dimensions).

  The following four steps have reached the dimensional design process:

1. Choosing the business process:

   The business process was defined based on the system requirements, which is monitoring OEE of the factory.
2. Declare the grain:

   The level of the detail in which the information is stored in the data warehouse, and it is the measurement for one day.
3. Identify the dimensions:

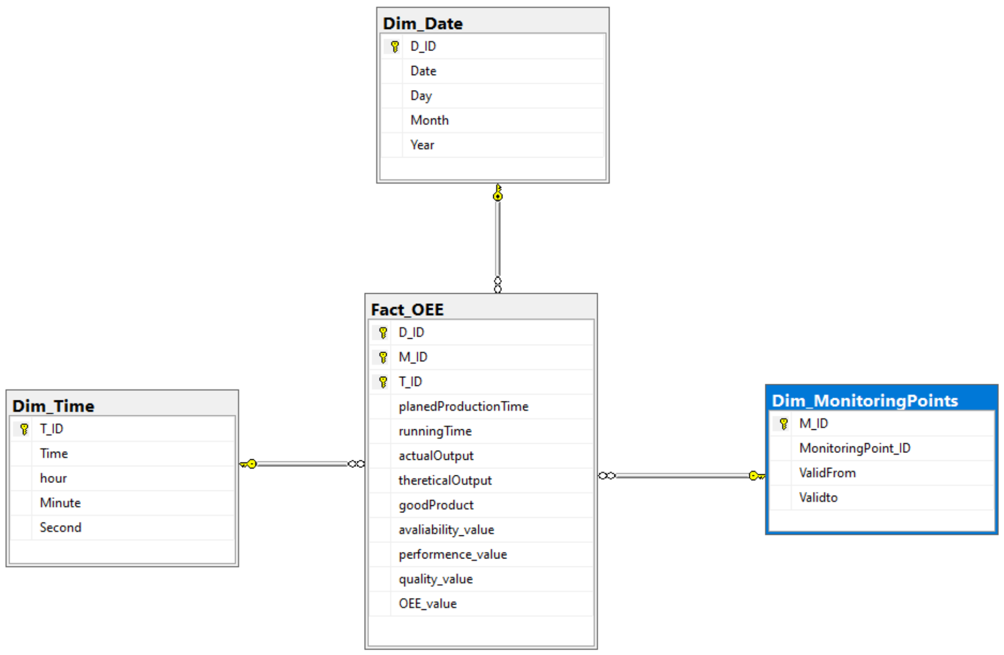
   Two types of dimensions were chosen to cover the business requirements.

- Static dimensions: Time and Date
- Dynamic dimensions: Monitoring Points which is a slowly changing dimension type two since the factory may add a new monitoring location or delete an old one.

4. Identify the facts:

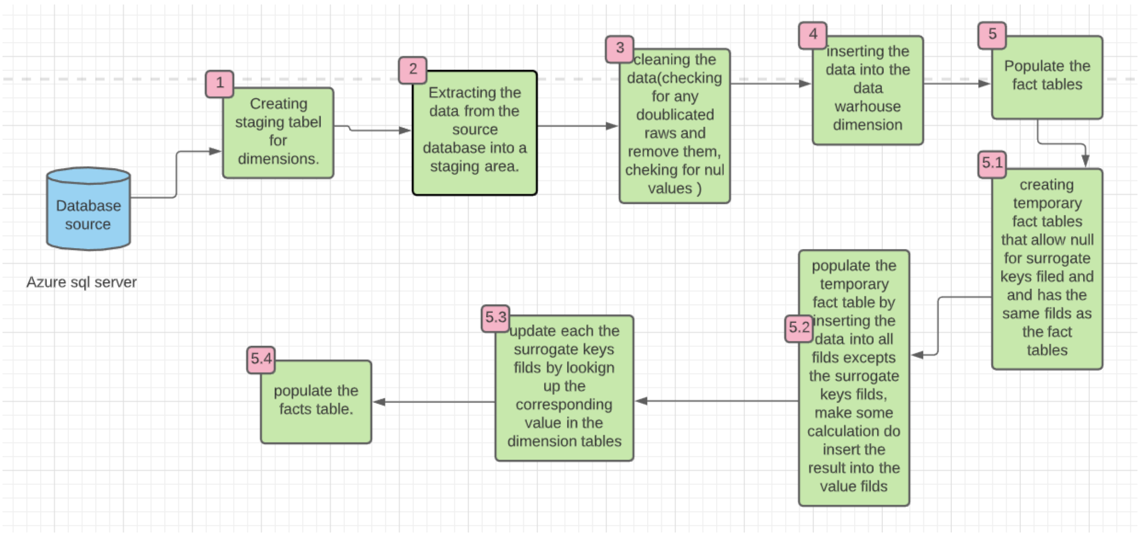
   The data warehouse consists of one semi-additive fact table OEE. The fact table contains foreign keys from all dimensions and all attributes that will be used to calculate the value for the production monitoring parameters (Availability, quality, performance and OEE).

The following figure represents the star schema diagram for the data warehouse.


Dim_Date
D_ID
Date
Day
Month
Year
Fact_OEE
D_ID
M_ID
T_ID
planedProductionTime
runningTime
actualOutput
thereticalOutput
goodProduct
avaliability_value
performence_value
quality_value
OEE_value
Dim_Time
T_ID
Time
hour
Minute
Second
Dim_MonitoringPoints
M_ID
MonitoringPoint_ID
ValidFrom
Validto

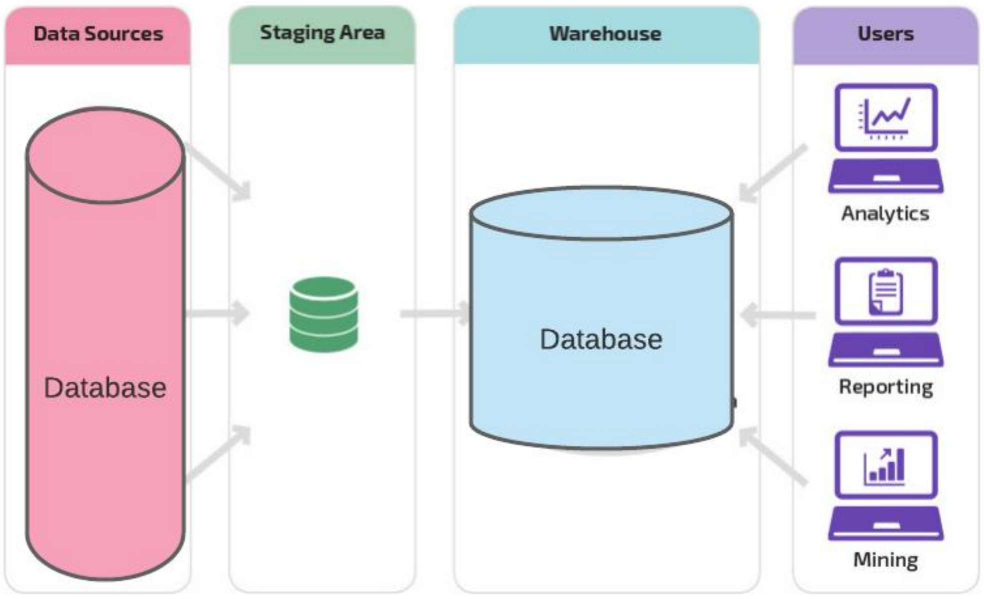
- **ETL Process (Balkis)**

The following diagram illustrates the steps that have to be taken to populate the data warehouse with data from the source database. Multiple databases are created in Azure SQL server one of them is storing the database source, one for the staging area and the other of the Data warehouse. In the real world, the data source and the Data warehouse are located in different places, but for this project's needs and because of the resource lack we decided to put them in the same place.


Database source
Azure sql server
1 Creating staging tabel for dimensions.
2 Extracting the data from the source database into a staging area.
3 cleaning the data(checking for any doublicated raws and remove them, cheking for nul values)
4 inserting the data into the data warhouse dimension
5 Populate the fact tables
5.1 creating temporary fact tables that allow null for surrogate keys filed and and has the same filds as the fact tables
5.2 populate the temporary fact table by inserting the data into all filds excepts the surrogate keys filds, make some calculation do insert the result into the value filds
5.3 update each the surrogate keys filds by lookign up the corresponding value in the dimension tables
5.4 populate the facts table.

- **ETL Architecture (Balkis)**

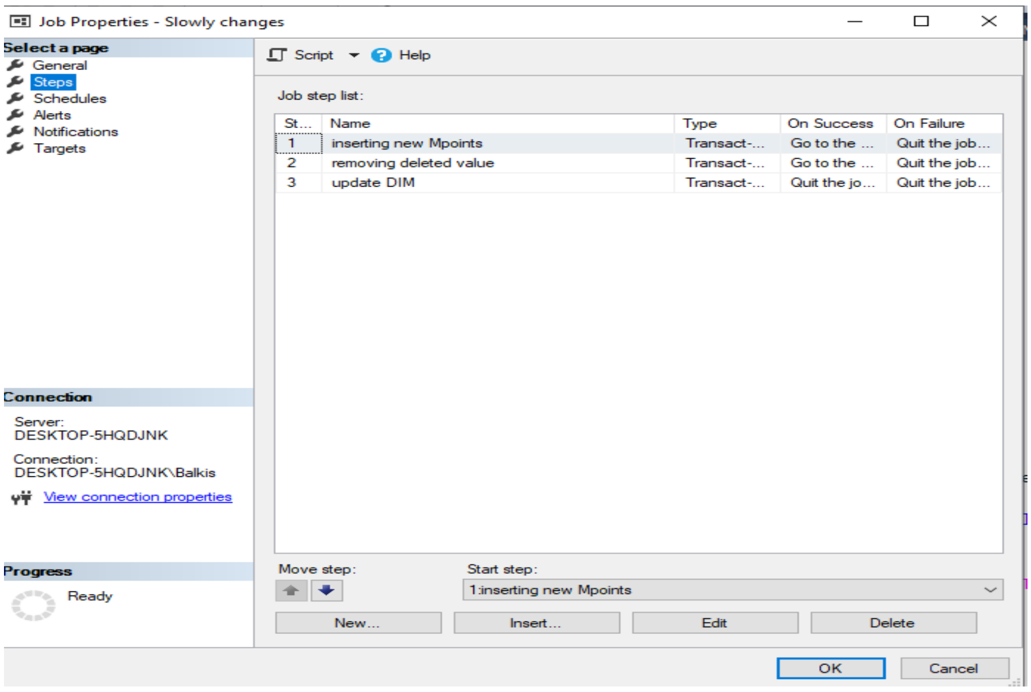
The following rich picture illustrates the architecture of the ETL process.



The data source is located in Azure SQL database. The data from the source is extracted and put in a new database called the staging area. Data cleaning and transformation is performed while the data is on the staging area. The next phase of the process is taking the data from the staging area and loading it into the data warehouse tables, which correspond to the dimensional model tables. Both databases for the staging area and the warehouse are also located in Azure SQL server. A visualisation application is created in Power BI to create data analysis and reports for the users. The application has a direct connection to the data warehouse, which allows him to perform faster queries.

- **SQL Server Job (Scheduling) (Balkis)**

SQL Server job is used to schedule queries to make some amendments in tables or columns. It is a three steps procedure. First, define job where name and description is filled, second, create the new step where write the queries to perform and third is scheduling to create a new schedule when it will start.
The figure represents a server job, and it is starting every night at 00:00. So, data will be inserted from entities to the data warehouse every day.



Job step list:

St...	Name	Type	On Success	On Failure
1	inserting new Mpoints	Transact-...	Go to the ...	Quit the job...
2	removing deleted value	Transact-...	Go to the ...	Quit the job...
3	update DIM	Transact-...	Quit the jo...	Quit the job...

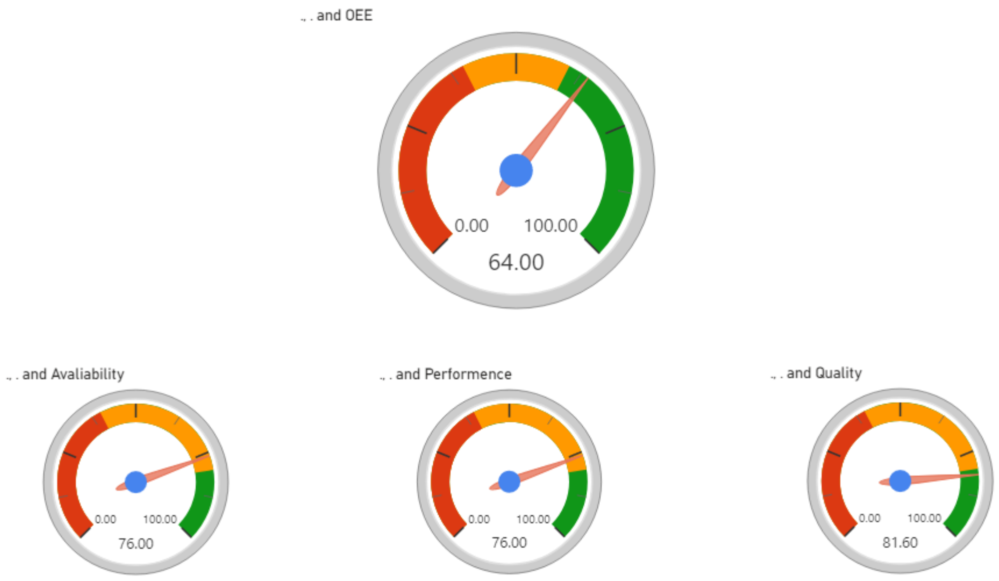
Start step: 1:inserting new Mpoints

Connection
Server:
DESKTOP-5HQDJNK
Connection:
DESKTOP-5HQDJNK\Balkis
View connection properties

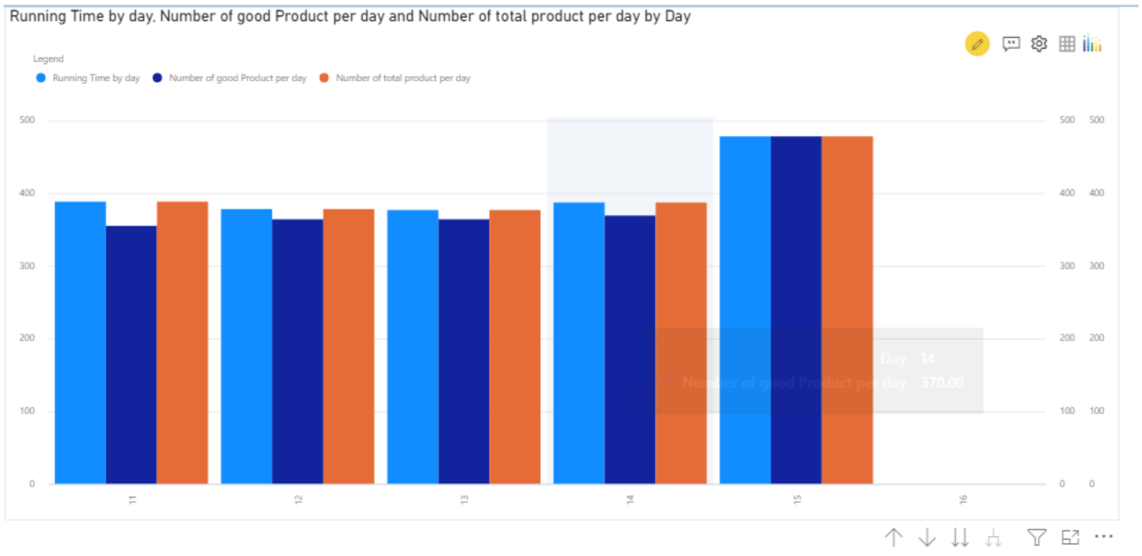
Progress
Ready

3.2.4 Power BI (Balkis)

Power BI is used to visualise data and create Business intelligence reports for the user. The graphs are designed to facilitate reading numbers for the decisions maker and help them make the best decision in no time. I avoided presenting too much data in the same view and tried to show the important number in a simple and clear graph which coloured with comfortable colours. Four DialGauge are designed based on the OEE fact within the same page they show the average for all the production monitoring parameters (availability, performance, quality and OEE). The following graph presents the DialGauge each of the Gauge has three colours to indicate the level of the production. The red means the production at a very low level, the yellow is acceptable production level, and green is good production level.



One more page is designed to present the total produced items, the total of non-defective produced items and the number of minutes that the factory runs during a full day. The following graph shows the chosen chart for visualising the data. The reason for choosing this char is because it is an interaction char which allows the user to hover on one of the coloured columns to get more info, there is also an option for drill-down and up through date hierarchies.



3.2.5 Technologies (Balkis)

The following section describes the technologies that are used to implement the system. The decisions are made based on the project needs and our skills. The technologies used are an essential part of the process as they can increase or decrease the performance. We tried to avoid the technology that needs a subscription. Another reason for choosing most of the technologies is because we were already familiar with them. Therefore, less time is spent to get familiar with the technologies, and more time is used to develop the system.

c#:

C# is a strongly typed object-oriented programming language. C# is open source, simple, modern, flexible, and versatile. (what is c#)
C# is used for the backend side of this project (APIs) to provide a solution for handling requests coming from the client-side. We had to choose between c# and java since those are the languages that we are familiar with them the most. We wanted to get advances in .net core framework features and templates to write less code and focus more on other parts of the system.

Azure cloud:

Azure cloud is Microsoft's public cloud computing platform. It provides a range of cloud services, including compute, analytics, storage and networking. It used to publish the APIs and make available to handle HTTP requests over the internet. It is also used to store the database source and the data warehouse. Most of the tools we used are from Microsoft, so it was more convenient to also host the API on a Microsoft tool. They work better together, and Microsoft has made the process of hosting easy. The fact that we had a free month trial for students has encouraged us the most.

SSMS:

SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure. Use SSMS to access, configure, manage, administer, and develop all SQL Server components, Azure SQL Database, and Azure Synapse Analytics. SSMS provides a single comprehensive utility that combines a broad group of graphical tools with a number of rich script editors to provide access to SQL Server for developers and database administrators of all skill levels (Microsoft Documentation). It is used to manage the Azure SQL Database. We have a local instance from azure SQL server on my machine, which makes it easier to write SQL code and queries the data since we are very familiar with the tool.

Visual Studio:

The Visual Studio integrated development environment is a creative launching pad that you can use to edit, debug, and build code, and then publish an app. An integrated development environment (IDE) is a feature-rich program that can be used for many aspects of software development. Over and above the standard editor and debugger that most IDEs provide, Visual Studio includes compilers, code completion tools, graphic designers, and many more features to ease the software development process (Overview of Visual Studio | Microsoft Docs, no date). It is chosen because we are familiar with this IDE, and it can be used together with Microsoft data tools.

Power BI Desktop:

Microsoft Power BI Desktop is built for the analyst. It combines state-of-the-art interactive visualisations, with industry-leading data query and modelling built-in. Create and publish your reports to Power BI. Power BI Desktop helps you empower others with timely critical insights, anytime, anywhere (Download Microsoft Power BI). The tool is chosen since we have previous experience with it.

Swagger UI:

Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON. Swagger is used together with open-source software tools to design, build, document, and use RESTful web services. We have used to document the APIs so that the API consumers can have a clear overview of what the APIs are capable of doing. The swagger interface describes each API method and tells precisely what parameters it takes and what attributes it contains so you can quickly know how to call it without interfering with the code or the database. The interface also shows the database schema with all the entities and the attributes. The

reasons for choosing swagger are it is free to use; it works perfectly with visual studio.
The swagger UI can be found at Backend→ Appendix C.

Doxygen:

Doxygen is a documentation generator, a tool for writing software reference documentation.
We have used it to document my code, so it is easier to understand.
Doxygen is free to use, and it can do the documentation for c# code also it generates XML files that can be used to create UML diagrams.
The Doxygen documentation can be found at Backend→ Appendix D→ output → HTML → index.

UML modelling tool / Ritch picture:

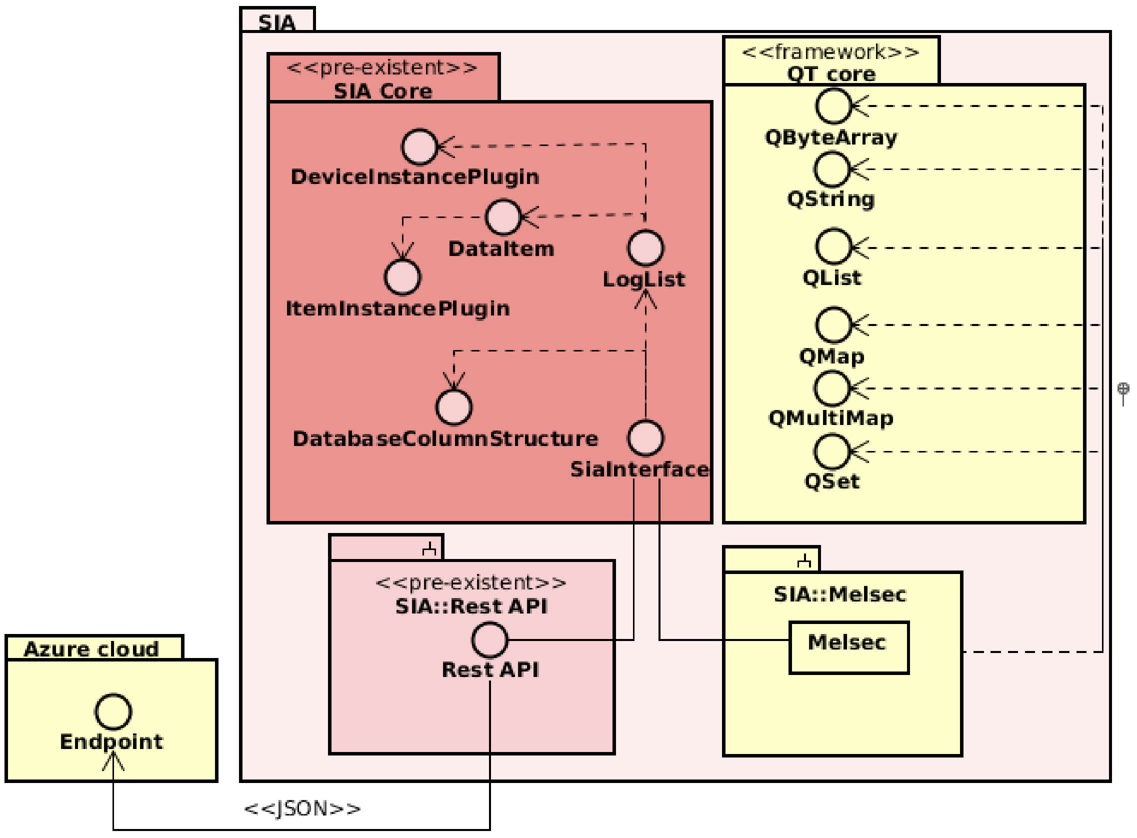
We have used Astah for UML diagram modelling during our education period since the university has provided us with a licence; also, we have been working with it for four years; therefore, we have a good experience with it. We had to look for another tool to make rich pictures. The tool we used is called lucid app. It is free to use, and it gives flexibility in making diagrams.

3.3 Data acquisition design (Alex)

3.3.1 Data acquisition sub-system model (Alex)

The target hardware that holds the data from the production line is a programmable logic controller produced by Mitsubishi, model Q02 and its designed communication protocol is named Melsec. From the range of communication frame types provided by the Melsec communication protocol, this model is compatible with the frames 3E and 4E.

Having taken in consideration the dependencies involved in designing a new connector that will be plugged in the SIA software, and the functionality of the connector the architectural model represented graphically in the diagram below was produced.



The module *Melsec*, named after the PLC's manufacturer original name for the communication protocol, will be responsible with the facilitation of reading, and if needed writing, data to the PLC. Since it is established from the non-functional requirements that the module will be developed using the QT framework, the module will use primitive and non-primitive data types provided by the framework. In the image above only some of these are exemplified and encapsulated in the package *QT core*. This package is included in the SIA package since *SIA core* was compiled against the QT framework libraries, as disclosed by the developers of SIA core.

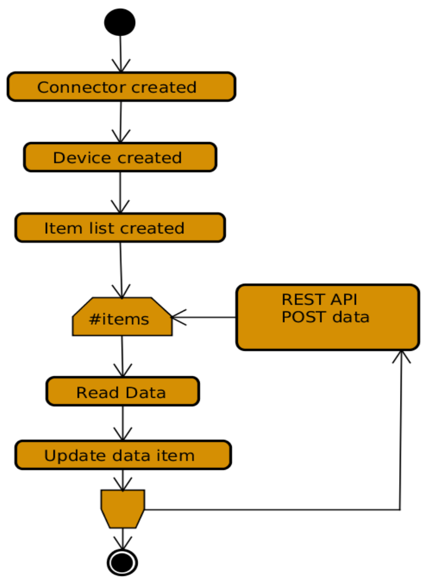
The existent module *Rest API* is provided by the collaborating company, and it facilitates the transmission of data, formatted as JSON, over the internet to an endpoint.

The existent module *SIA core* is also provided by the collaborating company and it facilitates the installation on new communication modules, referred in the module's context as connectors and hence in the rest of this paper, and the running environment for the connectors.

Since both *Rest API* and *SIA core* are of the shelf products, they fall out of the scope of this project and they will be not discussed further than the dependencies that they bring.

3.3.2 Process flow in the data acquisition sub-system (Alex)

Once the connector, the device and list of items are created the system enters a loop that will read the data from the PLC, update the data value in in the item and send the data as JSON using the REST API.
The conditions for exiting the loop are the disabling of all items in the list.
The process flow in the data acquisition layer is represented in the flow chart below.

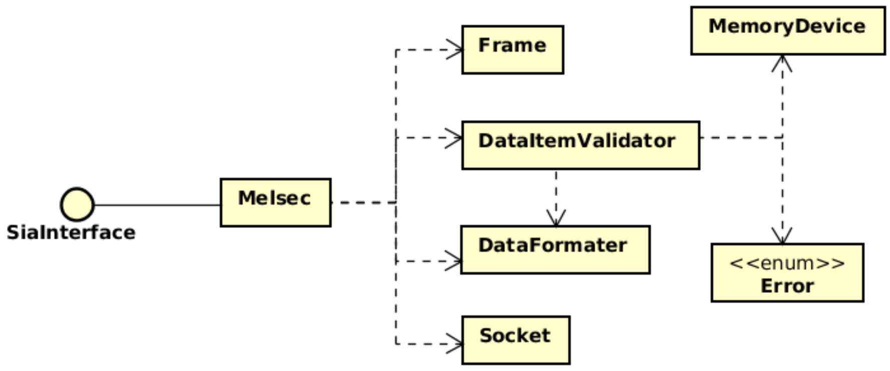


3.3.3 Connector architecture (Alex)

The required functionality of the connector is to access data from the PLC. This can be broken down in several responsibilities:

- check user input
- provide connection to the PLC
- build a communication frame that will be recognized by the PLC as a command
- format the data from the PLC to user specified type

The desired architectural design pattern is a facade since the inner functionality of the connector is irrelevant to the user and the fulfilment of the various responsibilities is delegated to other classes.
For each of these responsibilities a class is designed, and all these classes are private members of the class Melsec.

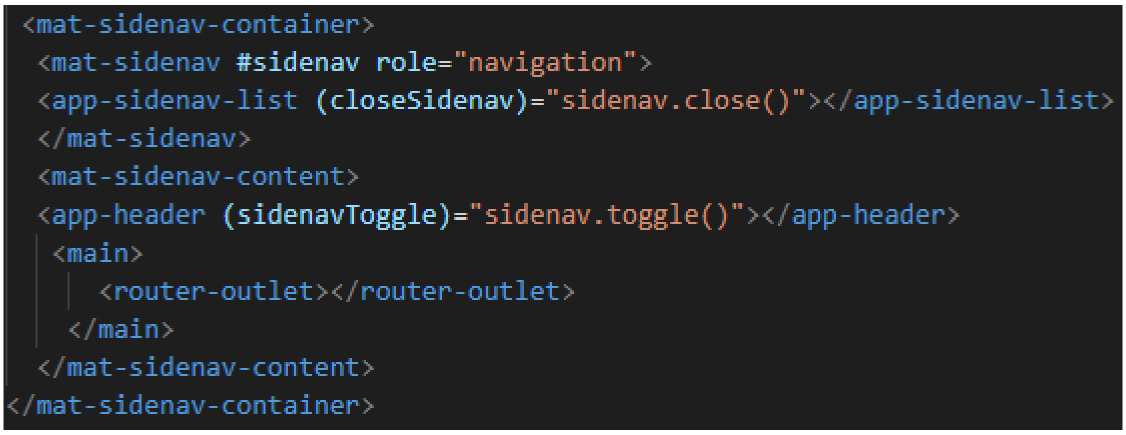


4 Implementation

This section provides details about the implementation of the requirements of the system. Moreover, the most interesting and relevant snippets are being presented, as well.

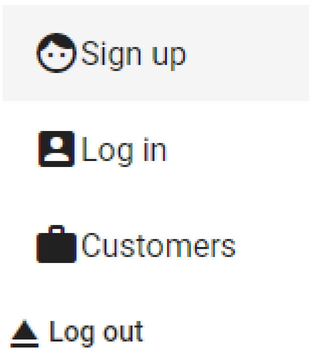
4.1 Front-end Implementation (Nikola)

Initialization of the project
New Angular projects are created using *Angular CLI* tool which is a command line interface for Angular. In order to create project *Node.js* must be installed. First step is to install *Angular CLI* through terminal by typing *npm install -g @angular/cli.* Then new project is created by navigating in the terminal to desired folder and executing *ng new projectname*. To be able to run and test application, development server must start. This is done via the terminal by executing *ng serve.* After these steps application can be open under *localhost:4200.* This application uses Angular Material. It is installed using terminal by executing *npm install –save @angular/material @angular/cdk.*
Navigation and side nav
Navigation and sidenav enable switching between pages without having to manually typing url. To implement them two components from Angular Material are going to be used: Toolbar and Sidenav. Toolbar resembles header component which can be positioned everywhere on the page. Sidenav is responsive side navigation which can be toggle and pushed over other page content. In order to implement pattern which includes Toolbar and Sidenav, Toolbar must be positioned in a container called *<mat-sidenav-container>.* Before using Sidenav module it needs to be imported in *materials.module.ts.* in *imports* and *exports* as *MatSidenavModule.* *<mat-drawer-container>* contains the sidenav navigation which can be pushed in and out and the content of the page which is always visible. *#sidenav* reference can be used on a button to reach sidenav reference to call *toggle()* method which will open sidenav if closed and closed it if open.

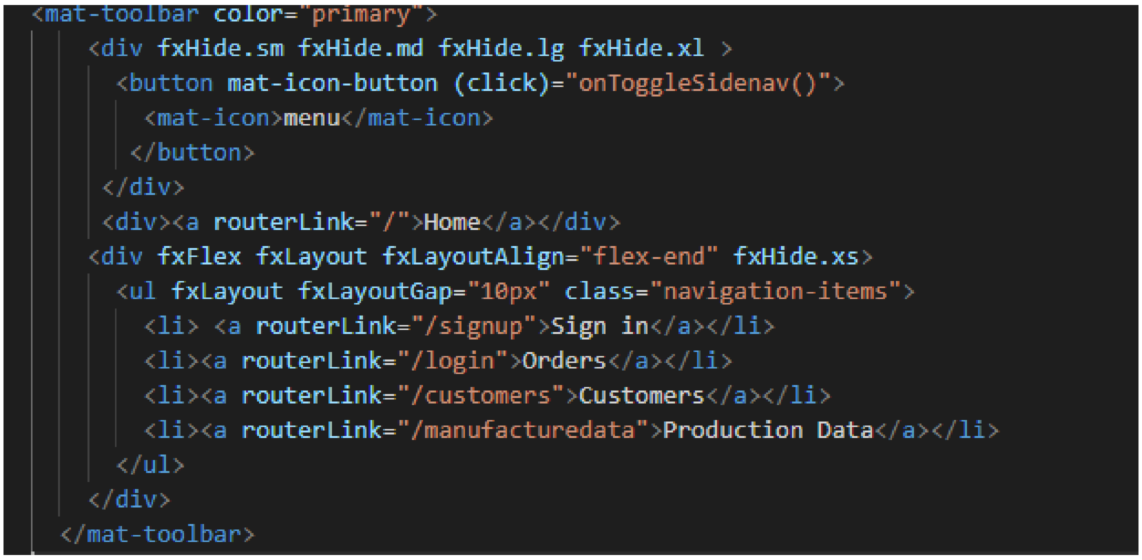


```
<mat-sidenav-container>
  <mat-sidenav #sidenav role="navigation">
  <app-sidenav-list (closeSidenav)="sidenav.close()"></app-sidenav-list>
  </mat-sidenav>
  <mat-sidenav-content>
  <app-header (sidenavToggle)="sidenav.toggle()"></app-header>
    <main>
      <router-outlet></router-outlet>
    </main>
  </mat-sidenav-content>
</mat-sidenav-container>
```

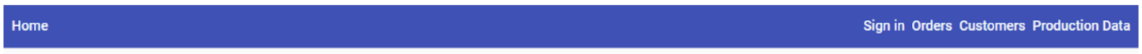
Figure 1 Sidenav code



Toolbar can be added anywhere in the application. It is a block with background colour and items on it. It uses flexbox internally to distribute these items. It needs to be imported in *material. module.ts* in *imports* and *exports* as *MatToolbarModule.* Items on toolbar can be styled using *fxLayoutAlign=" flex-end".* This will push them to the right side of the screen. Items are put in unordered list with *fxlayoutGap="10px"* to ensure there is 10px space between them. Toolbar will disappear on extra small devices.

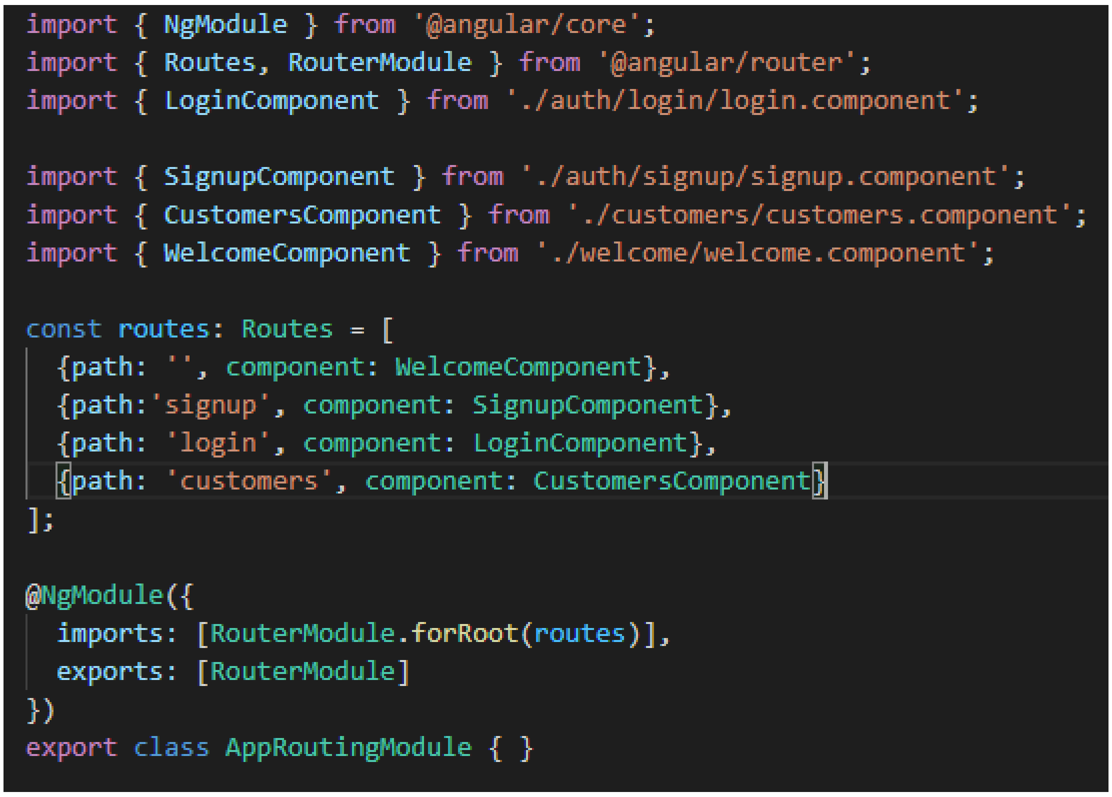


```
<mat-toolbar color="primary">
    <div fxHide.sm fxHide.md fxHide.lg fxHide.xl >
      <button mat-icon-button (click)="onToggleSidenav()">
        <mat-icon>menu</mat-icon>
       </button>
     </div>
     <div><a routerLink="/">Home</a></div>
    <div fxFlex fxLayout fxLayoutAlign="flex-end" fxHide.xs>
      <ul fxLayout fxLayoutGap="10px" class="navigation-items">
        <li> <a routerLink="/signup">Sign in</a></li>
        <li><a routerLink="/login">Orders</a></li>
        <li><a routerLink="/customers">Customers</a></li>
        <li><a routerLink="/manufacturedata">Production Data</a></li>
      </ul>
    </div>
  </mat-toolbar>
```



Routing

Application needs routing to traverse through different pages. *app-routing.module.ts* holds routing logic and set up application routes. In this class *AppRoutingModule* is exported. It does not have a body, but it has *@ngModule({})* decorator. This decorator needs to be imported. Routing is stored in *const routes* of type *Routes.* Type *Routes* is a specific type provided by Angular router package. *Routes* are also imported in the class. Route is a JavaScript object which has path. Path determines when a certain component is going to be loaded. First path is the root path.



```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LoginComponent } from './auth/login/login.component';

import { SignupComponent } from './auth/signup/signup.component';
import { CustomersComponent } from './customers/customers.component';
import { WelcomeComponent } from './welcome/welcome.component';

const routes: Routes = [
  {path: '', component: WelcomeComponent},
  {path:'signup', component: SignupComponent},
  {path: 'login', component: LoginComponent},
  {path: 'customers', component: CustomersComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

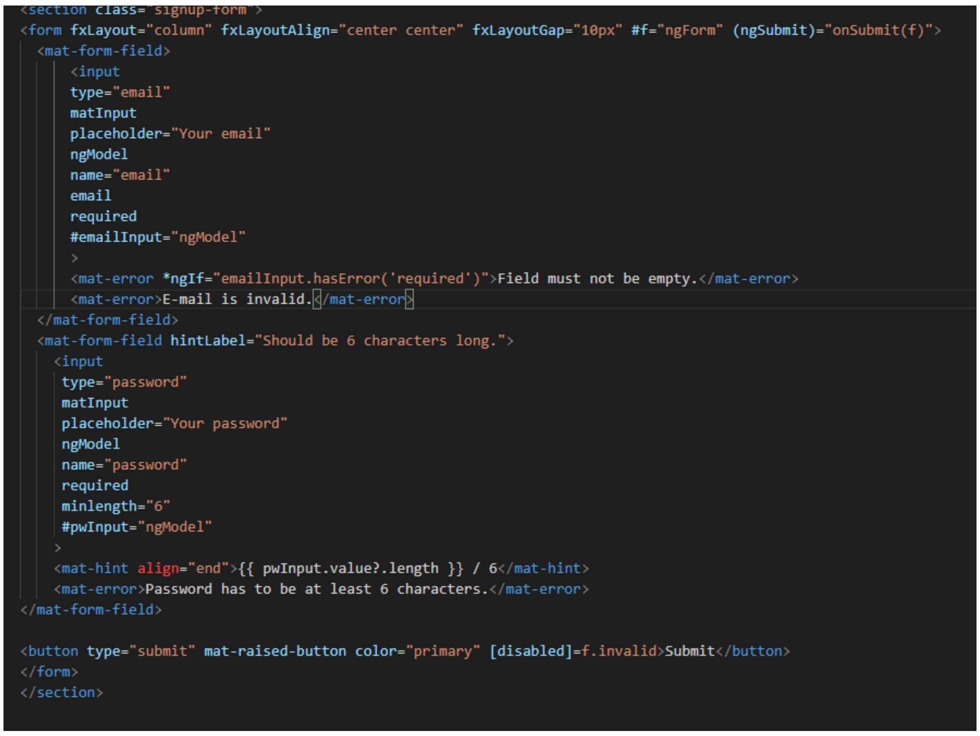
Authentication

Authentication section is using form to consume user credentials. Input and form field are imported from Angular Material. *matInput* is an directive which can be added to normal HTML input to give material design looking field like floating placeholder(text). *matInput* should be wrapped with *<mat-form-field>. <mat-form-field>* is a component, which is a wrapper which allows to use input components in wrapping form element. This approach allows to output hint labels, error messages and floating label.
Grid is not part of angular Material package. Flex layout is a package that uses Flexbox(flexbox.css) and allows for positioning components and elements by using directives. *fxFlex* takes all available space and evenly distributes it across all child elements. *fxLayout* gives responsive API which means that directives can be combined with certain responsive directive additions, where certain rule should apply only to certain screen sizes. *fxLayout* is installed using CLI by executing in the terminal *npm install @angular/flex-layout –save.* Then Angular Flex-Layout is imported into *NgModule.* Form can be submitted by using button. *<mat-form-field>* works with

template driven approach. Register input in angular form in template driven approach can be done by adding directive *ngModel* into the input field and name to the control attribute. Second is to add local reference by adding # and assign in to *ngForm.* This gives access to automatically created object. Object is stored in the local reference, which can be used only in this template. Bind *ngSubmit* event and execute *onSubmit()* and pass the local reference. *onSubmit()* is located in *signup.component.ts.* This method receives a form of type *NgForm* as a parameter.

Validation

Next step is to show error messages and hints in the form if user enters something wrong. This can be accomplish by adding validator *required* to the form inputs(in the email directive). *minlength* specify the minimum number of character which the form will accept. HintLabel is added by using *<mat-form-field>. <mat-hint>* contains second hint which shows how many characters are already typed. Multiple errors can be displayed with *<mat-error>. *ngIf* controls when which error is going to appear.

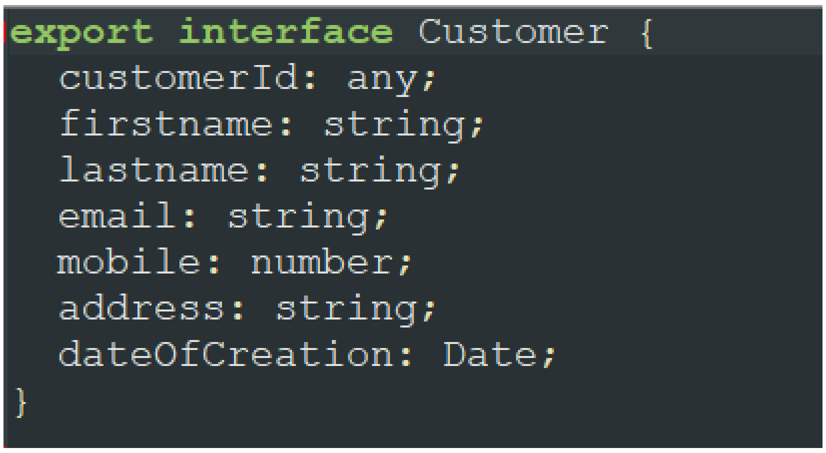


```
<section class="signup-form">
<form fxLayout="column" fxLayoutAlign="center center" fxLayoutGap="10px" #f="ngForm" (ngSubmit)="onSubmit(f)">
  <mat-form-field>
      <input
      type="email"
      matInput
      placeholder="Your email"
      ngModel
      name="email"
      email
      required
      #emailInput="ngModel"
      >
      <mat-error *ngIf="emailInput.hasError('required')">Field must not be empty.</mat-error>
      <mat-error>E-mail is invalid.</mat-error>
  </mat-form-field>
  <mat-form-field hintLabel="Should be 6 characters long.">
    <input
     type="password"
     matInput
     placeholder="Your password"
     ngModel
     name="password"
     required
     minlength="6"
     #pwInput="ngModel"
    >
    <mat-hint align="end">{{ pwInput.value?.length }} / 6</mat-hint>
    <mat-error>Password has to be at least 6 characters.</mat-error>
</mat-form-field>

<button type="submit" mat-raised-button color="primary" [disabled]=f.invalid>Submit</button>
</form>
</section>
```

Customers

Using command *ng g c customer* in terminal new component for new customer managing is created. Another component is generated for editing users.

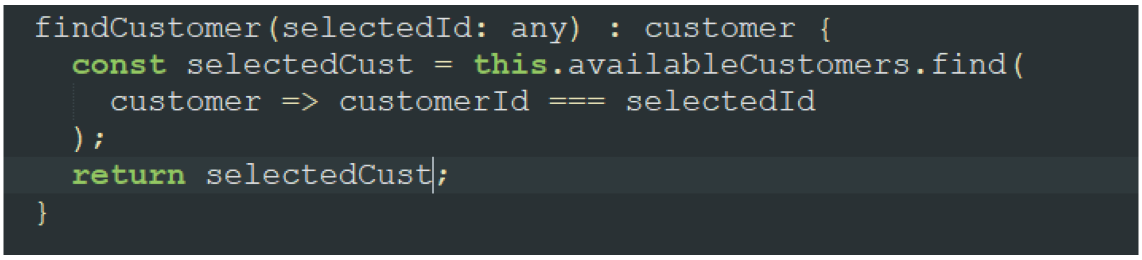
In order to work with customers first data definition of a customer must be provided. This is done in *customers.model.ts.*



```
export interface Customer {
  customerId: any;
  firstname: string;
  lastname: string;
  email: string;
  mobile: number;
  address: string;
  dateOfCreation: Date;
}
```

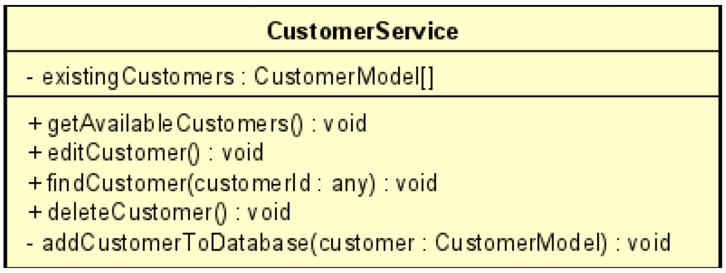
Next step is to create a service which manages customers. The interface *Customer* must be imported in the service. This service must be imported into *app.module.ts*. In *newcustomer.component.ts* training service is inserted into the constructor as parameter of type *Customerservice* . Then property *customers* of type *Customers* is added. Method *ngOnInit()* is the start of the life cycle of this component. Inside this method *customers* are assigned to the service. This will create an empty customer object.

Find customersCusotmers can be traversed by their *customerId. selectedCust* is a temporary constant to store desired customer. Find method will traverse through all available customers and will execute arrow function on every element with matching id.



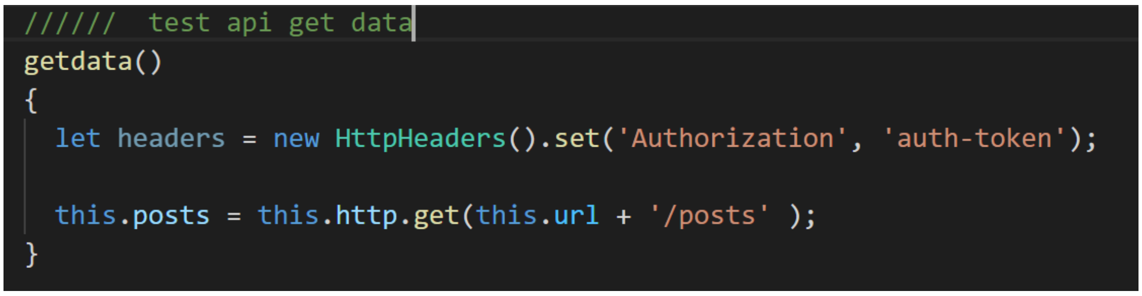
```
findCustomer(selectedId: any) : customer {
  const selectedCust = this.availableCustomers.find(
    customer => customerId === selectedId
  );
  return selectedCust;
}
```

The function to delete a customer follows the same logic. Temporary constant is declared. Then the find method traverse through the array of customers and fetch the customer with matching id.

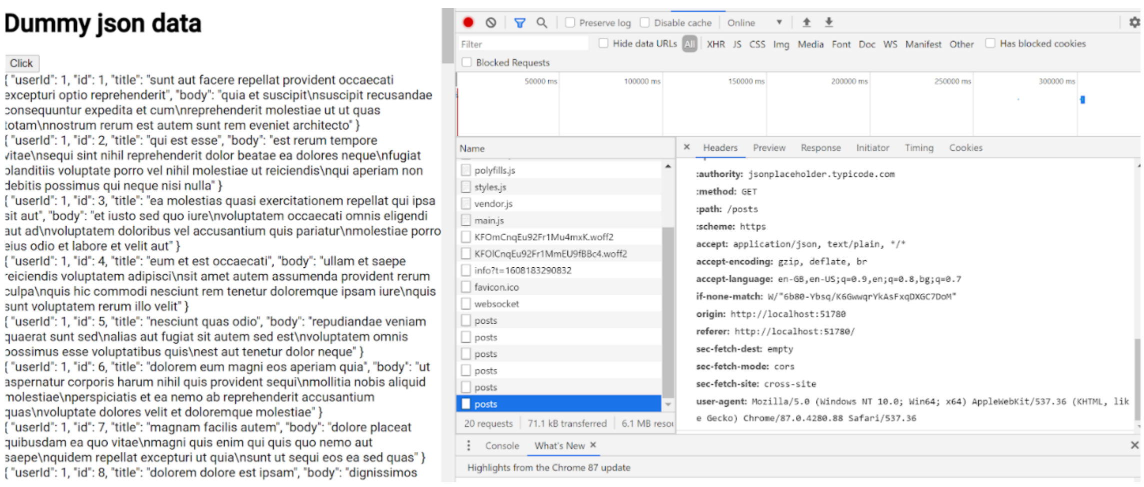

CustomerService
- existingCustomers : CustomerModel[]
+ getAvailableCustomers() : void
+ editCustomer() : void
+ findCustomer(customerId : any) : void
+ deleteCustomer() : void
- addCustomerToDatabase(customer : CustomerModel) : void

Http requests

Client can interact with a server by using specific method. By using GET request client state that he wants to retrieve data, but do not want to modify it. The server will respond with status and requested data. If the request is successful there should be 200 global response. If the request is invalid the global response will be 4XX or 5XX error response. 4XX response means that it is client fault and 5XX response means that server side is faulty. By sending POST request client tell the server that he wants to create new data. POST request also sends the data payload intended to be saved on the server. If server saves it successfully it will response with 201 response message. Other methods which can be used are PUT, POST, DELETE. In order to create a POST request first *HttpClientModule* must be imported in *app.module.ts.* Then in *app.component.ts HttpClient* is imported. Then *readonly* url variable is declared. This is the endpoint which is going to be used. Then post method is defined. Post variable is defined by calling get method on http. This will return an observable with response from the API. In this case this is the url and the posts. In *app.component.html* a button is created to trigger the API call. **ngFor* is used to loop over the observable and unwrapped it with asynchronous pipe. In the end data will be printed as a JSON object. For testing purposes the current end node is https://jsonplaceholder.typicode.com/. This is free to use fake online REST API for testing and prototyping.



```
//////  test api get data
getdata()
{
  let headers = new HttpHeaders().set('Authorization', 'auth-token');

  this.posts = this.http.get(this.url + '/posts' );
}
```



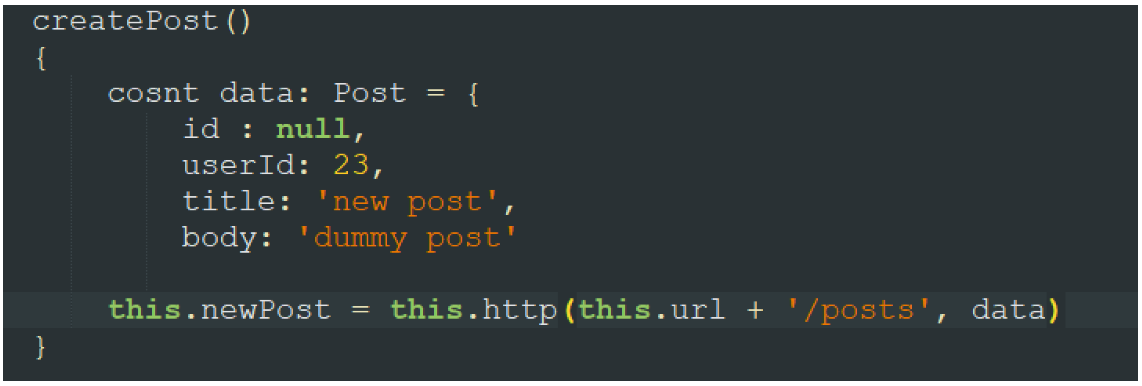
Dummy json data

```
{ "userId": 1, "id": 1, "title": "sunt aut facere repellat provident occaecati
excepturi optio reprehenderit", "body": "quia et suscipit\nsuscipit recusandae
consequuntur expedita et cum\nreprehenderit molestiae ut ut quas
totam\nnostrum rerum est autem sunt rem eveniet architecto" }
{ "userId": 1, "id": 2, "title": "qui est esse", "body": "est rerum tempore
vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat
blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non
debitis possimus qui neque nisi nulla" }
{ "userId": 1, "id": 3, "title": "ea molestias quasi exercitationem repellat qui ipsa
sit aut", "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi
aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro
eius odio et labore et velit aut" }
{ "userId": 1, "id": 4, "title": "eum et est occaecati", "body": "ullam et saepe
reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum
culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis
sunt voluptatem rerum illo velit" }
{ "userId": 1, "id": 5, "title": "nesciunt quas odio", "body": "repudiandae veniam
quaerat sunt sed\nalias aut fugiat sit autem sed est\nvoluptatem omnis
possimus esse voluptatibus quis\nest aut tenetur dolor neque" }
{ "userId": 1, "id": 6, "title": "dolorem eum magni eos aperiam quia", "body": "ut
aspernatur corporis harum nihil quis provident sequi\nmollitia nobis aliquid
molestiae\nperspiciatis et ea nemo ab reprehenderit accusantium
quas\nvoluptate dolores velit et doloremque molestiae" }
{ "userId": 1, "id": 7, "title": "magnam facilis autem", "body": "dolore placeat
quibusdam ea quo vitae\nmagni quis enim qui quis quo nemo aut
saepe\nquidem repellat excepturi ut quia\nsunt ut sequi eos ea sed quas" }
{ "userId": 1, "id": 8, "title": "dolorem dolore est ipsam", "body": "dignissimos
```

```
Name
polyfills.js
styles.js
vendor.js
main.js
KFOmCnqEu92Fr1Mu4mxK.woff2
KFOlCnqEu92Fr1MmEU9fBBc4.woff2
info?t=1608183290832
favicon.ico
websocket
posts
posts
posts
posts
posts
posts
20 requests | 71.1 kB transferred | 6.1 MB resou
```

```
:authority: jsonplaceholder.typicode.com
:method: GET
:path: /posts
:scheme: https
accept: application/json, text/plain, */*
accept-encoding: gzip, deflate, br
accept-language: en-GB,en-US;q=0.9,en;q=0.8,bg;q=0.7
if-none-match: W/"6b80-Ybsq/K6GwwqrYkAsFxqDXGC7DoM"
origin: http://localhost:51780
referer: http://localhost:51780/
sec-fetch-dest: empty
sec-fetch-mode: cors
sec-fetch-site: cross-site
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, lik
e Gecko) Chrome/87.0.4280.88 Safari/537.36
```

In order to filter the data and get only desired data a parameter must be send with the request. This JSON placeholder can be filtered by user id. In this case parameters are created dynamically, and it is not possible to hardcode them directly. To overcome this Angular *HttepParams()* class is used. *Params* are passed as second argument in the get request. If using authentication *HttpHeaders.* Then JSON web token is send authentication details in the header. Then server can decode that token to validate if the user has the right permission to make this request. *Headers* is passed as second argument in get request. POST request is similar. The difference is that in the body of the request it sends the data which is to be modified on the server. The data is sent as a second argument to the POST request.



```
createPost()
{
    cosnt data: Post = {
        id : null,
        userId: 23,
        title: 'new post',
        body: 'dummy post'

    this.newPost = this.http(this.url + '/posts', data)
}
```

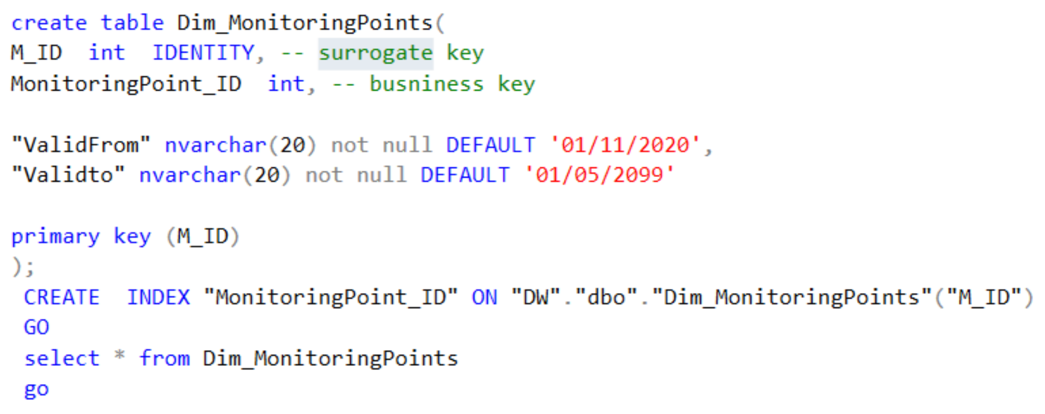
4.2 Backend Implementation(Balkis)

4.2.1 Data Warehouse Implementation (Balkis):

The first part of the section shows the script used for creating the tables designed in the dimensional model. and then to create the table for the staging area.
The second part shows the most important and relevant parts of the ETL process.

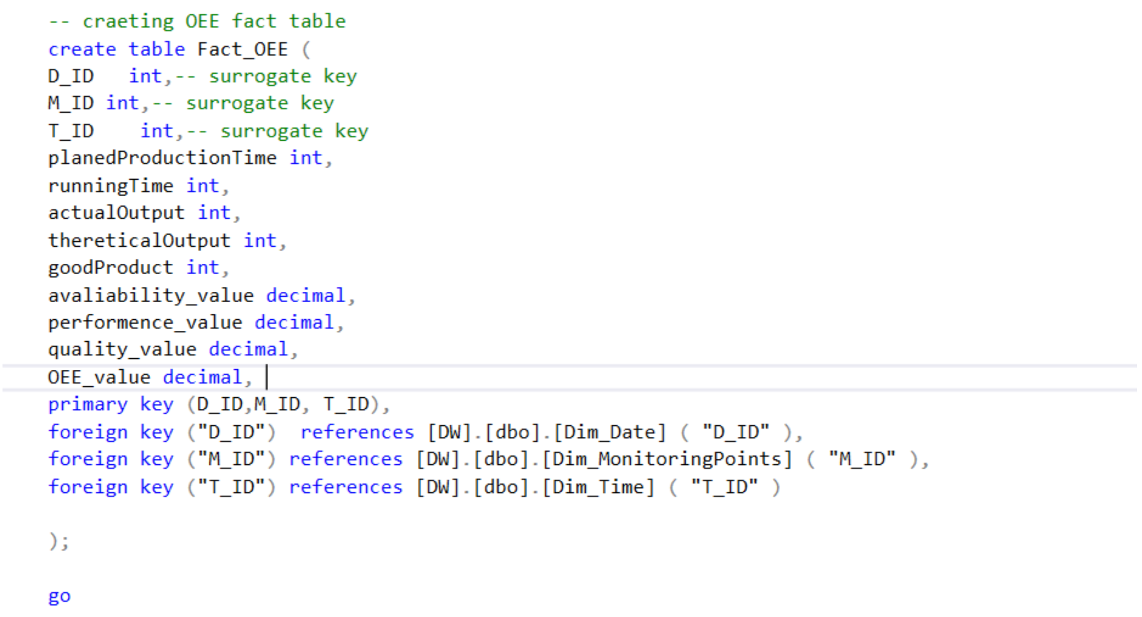
1. Creating the data warehouse and the staging area

In order to create the data warehouse, it is necessary to create all the tables existing in the dimensional model. The dimensions are created first and then fact tables. The following snip is an example of the Monitoring Points dimension:



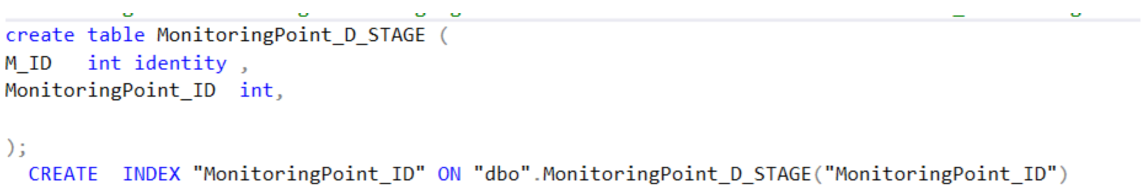
```
create table Dim_MonitoringPoints(
M_ID  int  IDENTITY, -- surrogate key
MonitoringPoint_ID  int, -- busniness key

"ValidFrom" nvarchar(20) not null DEFAULT '01/11/2020',
"Validto" nvarchar(20) not null DEFAULT '01/05/2099'

primary key (M_ID)
);
 CREATE  INDEX "MonitoringPoint_ID" ON "DW"."dbo"."Dim_MonitoringPoints"("M_ID")
 GO
 select * from Dim_MonitoringPoints
 go
```

The following snip shows the SQL code for creating the fact table. It contains the surrogate keys that were generated when populating the dimensions.



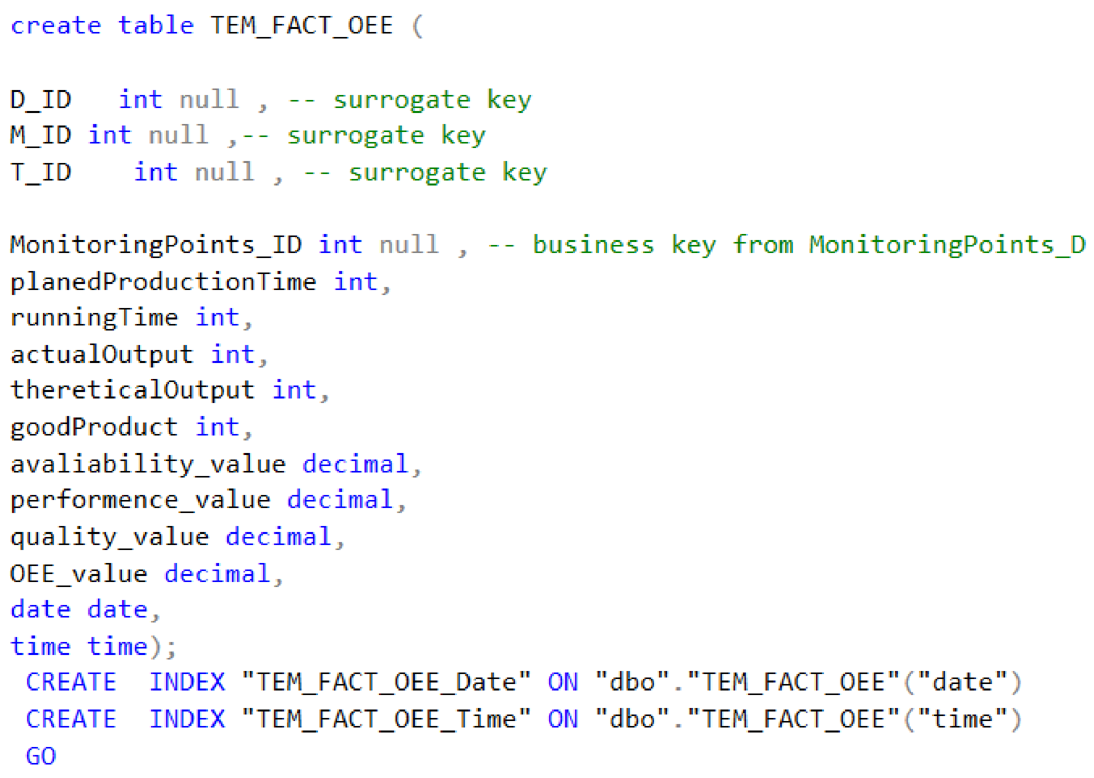
```sql
-- craeting OEE fact table
create table Fact_OEE (
D_ID   int,-- surrogate key
M_ID int,-- surrogate key
T_ID    int,-- surrogate key
planedProductionTime int,
runningTime int,
actualOutput int,
thereticalOutput int,
goodProduct int,
avaliability_value decimal,
performence_value decimal,
quality_value decimal,
OEE_value decimal, 
primary key (D_ID,M_ID, T_ID),
foreign key ("D_ID")  references [DW].[dbo].[Dim_Date] ( "D_ID" ),
foreign key ("M_ID") references [DW].[dbo].[Dim_MonitoringPoints] ( "M_ID" ),
foreign key ("T_ID") references [DW].[dbo].[Dim_Time] ( "T_ID" )

);

go
```

In order to populate the facts, a staging area needed to store the temporary changes and transformations. The following snip corresponds to the staging table for monitoring points dimension.



```sql
create table MonitoringPoint_D_STAGE (
M_ID   int identity ,
MonitoringPoint_ID  int,

);
  CREATE  INDEX "MonitoringPoint_ID" ON "dbo".MonitoringPoint_D_STAGE("MonitoringPoint_ID")
```

The following snip corresponds to the staging table for the fact. It contains both business and surrogate keys. Indexes are used to retrieve data from the database more quickly than otherwise. The indexes can not be seen; they are just used to speed up searches/queries.

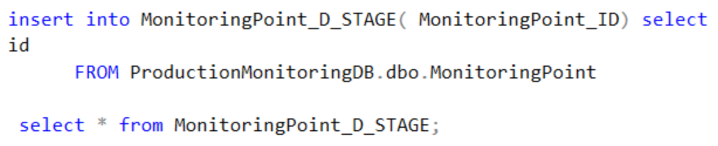


```
create table TEM_FACT_OEE (

D_ID   int null , -- surrogate key
M_ID int null ,-- surrogate key
T_ID    int null , -- surrogate key

MonitoringPoints_ID int null , -- business key from MonitoringPoints_D
planedProductionTime int,
runningTime int,
actualOutput int,
thereticalOutput int,
goodProduct int,
avaliability_value decimal,
performence_value decimal,
quality_value decimal,
OEE_value decimal,
date date,
time time);
 CREATE  INDEX "TEM_FACT_OEE_Date" ON "dbo"."TEM_FACT_OEE"("date")
 CREATE  INDEX "TEM_FACT_OEE_Time" ON "dbo"."TEM_FACT_OEE"("time")
 GO
```

2. ETL implementation (Balkis)

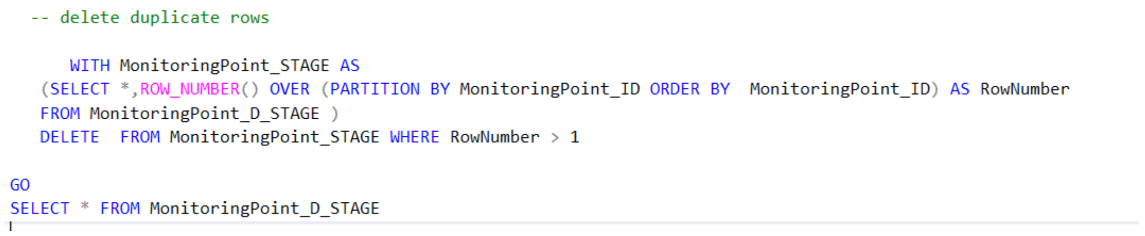
1. Dimensions ETL

The first step is to get the data from the source code (Azure cloud) and insert it into the staging table and generate surrogate keys. In this case, we only have one column to get from the source, the id of each monitoring point.



```
insert into MonitoringPoint_D_STAGE( MonitoringPoint_ID) select
id
      FROM ProductionMonitoringDB.dbo.MonitoringPoint

 select * from MonitoringPoint_D_STAGE;
```

After having the data in the staging area, we start doing data cleaning. The following code snip shows the SQL statements for removing duplicated data.



```
  -- delete duplicate rows

     WITH MonitoringPoint_STAGE AS
   (SELECT *,ROW_NUMBER() OVER (PARTITION BY MonitoringPoint_ID ORDER BY  MonitoringPoint_ID) AS RowNumber
   FROM MonitoringPoint_D_STAGE )
   DELETE  FROM MonitoringPoint_STAGE WHERE RowNumber > 1

GO
SELECT * FROM MonitoringPoint_D_STAGE
```

Before moving on I manually inserted duplicate rows and tested if the SQL statement can remove them. At this point, the data is ready to be loaded into the data warehouse table.

2. Facts ETL:

At first, I extracted the data and loaded into the staging area, and then I did some data transformation which was required for some fields. The source key date is a DateTime type, but in the data warehouse, two separated columns were created one for the time and the other for the date, therefore, the data transformation was needed. The running time and the planned production time are both int type, and some calculation needs to be performed in order to calculate the value for that reason, I needed to cast the type.

The following code snip is an example of the data transformation that I did in the staging area.

```
select  a.monitoringPointId, a.planedProductionTime ,a.runningTime , b.actualOutput, b.thereticalOutput, c.goodProduct,
CAST(a.runningTime AS decimal) / CAST(a.planedProductionTime AS decimal) * 100 ,
CAST(b.actualOutput AS decimal) / CAST(b.thereticalOutput AS decimal) * 100 ,
CAST(goodProduct AS decimal) / CAST(b.actualOutput AS decimal) * 100,
cast(a.date as date) [date], cast(a.date as time) [time]
```

Next step I did is updating the surrogate keys by looking up the business key from all dimensions.

```
UPDATE [TEM_FACT_OEE] SET D_ID =(select D_ID from [DW].[dbo].[Dim_Date] where [DW].[dbo].[Dim_Date].Date = TEM_FACT_OEE.date)
UPDATE [TEM_FACT_OEE] SET T_ID =(select T_ID from [DW].[dbo].[Dim_Time] where [DW].[dbo].[Dim_Time].Time = TEM_FACT_OEE.time)
UPDATE [TEM_FACT_OEE] SET M_ID =(select M_ID from [DW].[dbo].[Dim_MonitoringPoints]
where [DW].[dbo].[Dim_MonitoringPoints].MonitoringPoint_ID = TEM_FACT_OEE.MonitoringPoints_ID)
UPDATE [TEM_FACT_OEE] SET OEE_value = avaliability_value * performence_value * quality_value / 10000
```

The final step is populating the fact tables.

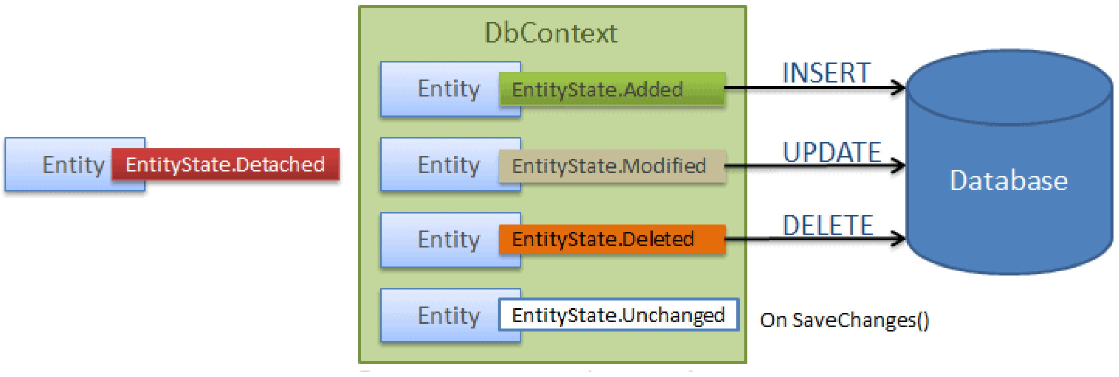
The SQL code can be found at Backend → Appendix E.

4.2.2 APIs Implementation: (Balkis)

Backend Implementation (API)

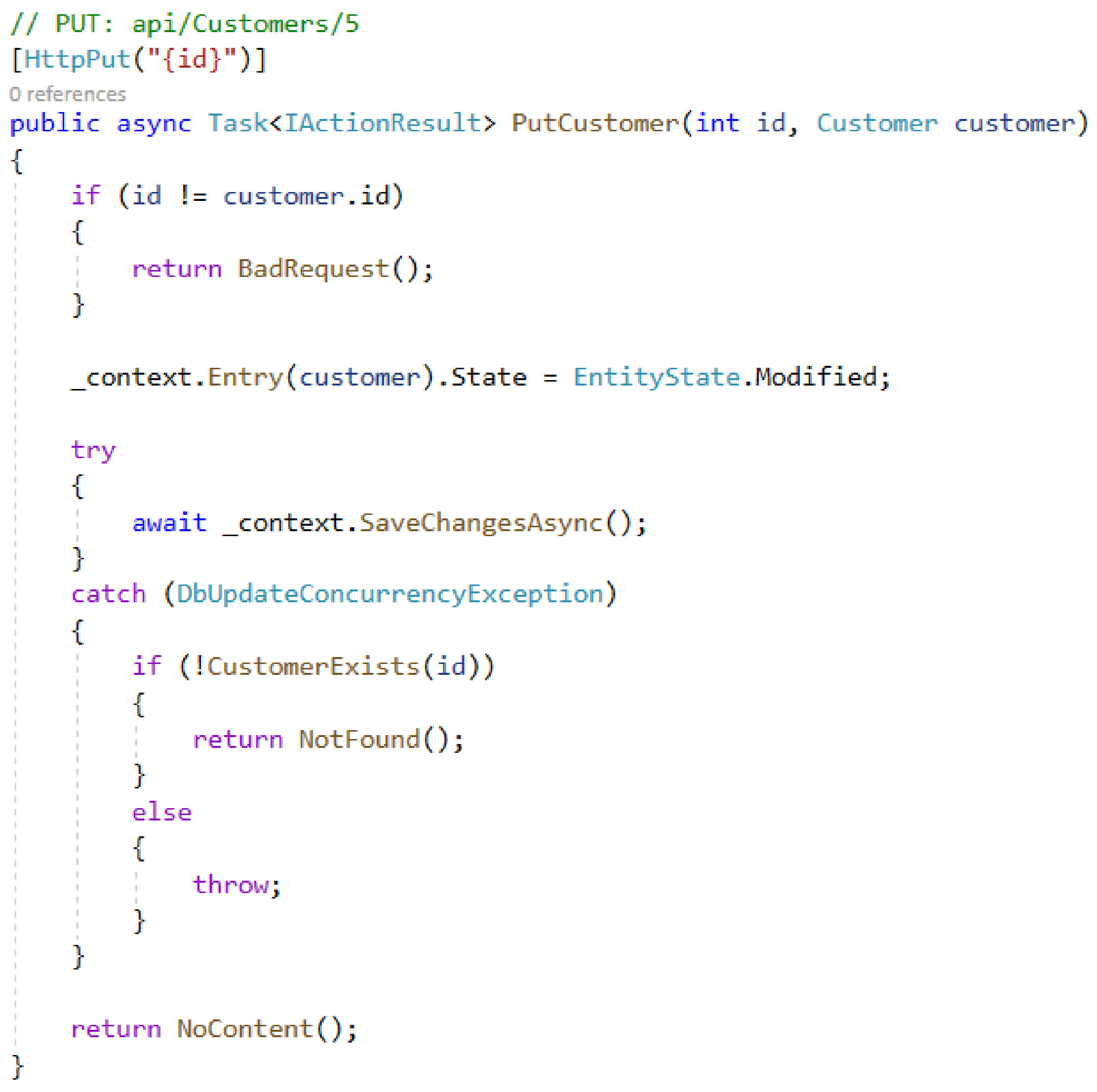
After designing the system (class diagram, sequence diagram, EER diagram), the implementation phase took place. Three classes separated into two packages were needed to implement the functionality of the system for each entity.
First, the domain classes represent a single table in the database and hold the properties that match and map to the table columns.

Second, the DBContext class, basically it is the most important class. It is derived from the EF Core DBContext class. This class holds the DBSets of the tables. It is used to query and save data to the database. It can combine multiple changes under a single database transaction. When the class is called for querying purposes, it will translate LINQ-to-Entities to SQL queries for the relational database using EDM and converts the results back to an entity object. The DbContext class holds a reference to all the entity objects as soon as retrieved from the database to keep track of the entity states and maintain modifications made to the entity's properties. When calling it for inserting, deleting, or updating the database, it will change an entity's state as illustrated in the rich picture.



Third, the controller class handles any incoming HTTP request and routes it to the proper controller action.

The following code snip shows the implementation of PutCutomer method. The method will first check the id it gets from the request URL with the id in the request body. It produces a bad request response if they are not equal. If they are similar, it changes the entity state and tries to save the database's updated entries if it finds the desired customer. So when calling context.SaveChanges() it will build and execute the proper commands based on the state of an entity.

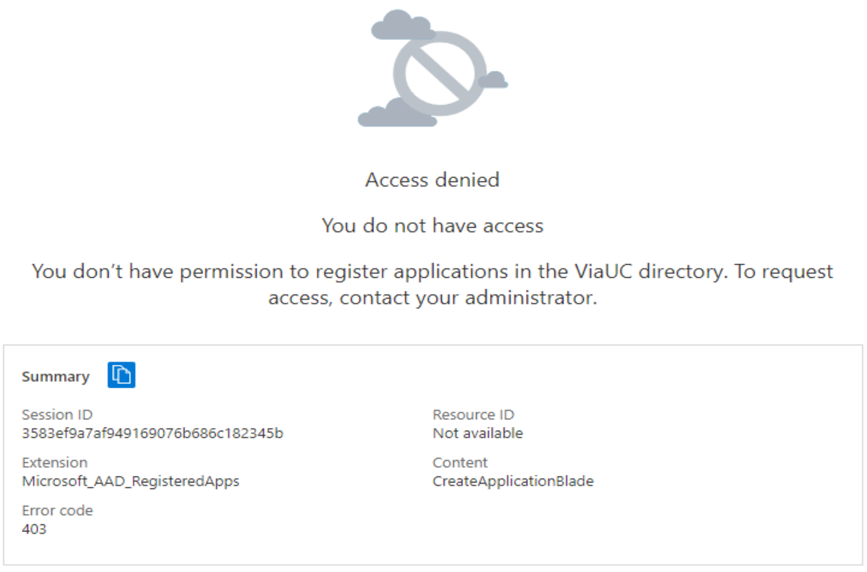


```
// PUT: api/Customers/5
[HttpPut("{id}")]
0 references
public async Task<IActionResult> PutCustomer(int id, Customer customer)
{
    if (id != customer.id)
    {
        return BadRequest();
    }

    _context.Entry(customer).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!CustomerExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}
```

The source code for the APIs code can be found at Backend → Appendix F.

APIs Protection: (Balkis)

NOTE that I could not get the chance to implement this feature since the IT department has shut down access to Azure AD for security purposes.



Access denied

You do not have access

You don't have permission to register applications in the ViaUC directory. To request access, contact your administrator.

Summary

Session ID 3583ef9a7af949169076b686c182345b	Resource ID Not available
Extension Microsoft_AAD_RegisteredApps	Content CreateApplicationBlade
Error code 403	

In the following section, I will concisely go through how I would implement it.

1 I would register the clients and The API resource applications in Azure AD.
2 I would configure the Azure AD.

2.1 Setting access token version in API resource AAD application.
2.2 Expose the API Application and the underlying roles to make them available for client's authorisation request to AAD.
2.3 Setting the client secret in Client AAD application.
2.4 Write code for defiling application roles for the API application.
2.5 Grant the client applications the permissions needed for accessing the API.

- Configure the APIM instance.
- Test the entire flow.

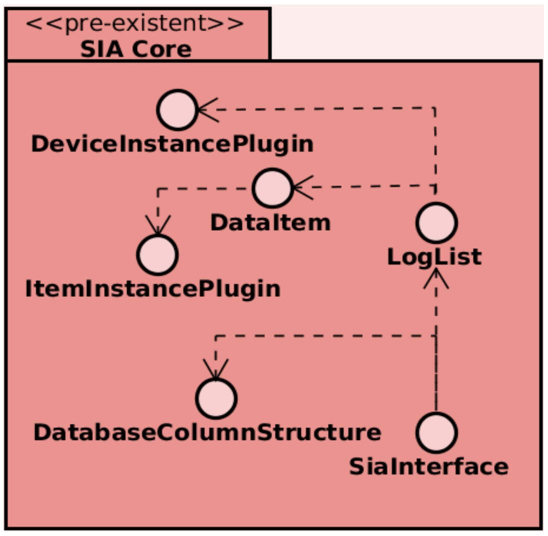
Plan B is applied to secure the API, where the API callers got a subscription key that they can include in the HTTP request header to grand access the APIs.

4.3 Data acquisition sub-system Implementation (Alex)

4.3.1 Prerequisites and dependencies.

4.3.1.1 SIA architecture

For clarification on the SIA software dependencies a brief introduction to two software modules is necessary. *LogList* is a composite data type of type struct as defined by the C/C++ languages ad it has a total of three members out of which two are touching the scope of the project: *DeviceInstancePlugin* and *QList<DataItem>.* The first is a struct that encapsulates the data that defines a device such as internet IP and port, network number, frame type, etc. In this case it refers to the PLC in usage. The latter is a list of struct data types that encapsulate a value as string and a *ItemInstancePlugin* struct. The *ItemInstancePlugin* is a struct that holds data referring to a specific memory cell on the PLC such as the type of the data i.e., integer, character, the name of the memory cell, the reading/writing timeout.
DatabaseColumnStructure is a class that is used in the creation of database tables that will use the information passed to the constructor to create tables in the local database, that in turn, it is used by *SIA core. TableStructure* is a composite data type that encapsulates a list of *DatabaseColumnStructure* items.



To build a proper SIA connector, it must implement the interface *SIAInterface* and implicitly the methods:

- virtual bool init () = 0;
- virtual bool writeData (LogList *data) = 0;
- virtual bool readData (LogList *data) = 0;
- virtual bool stopPlugin () = 0;
- virtual bool getEventData (LogList *data) = 0;
- virtual bool logCallback (LogList *data) = 0;

The last two methods are subject to legacy source code and an empty implementation was advised by the developers of SIA core. The *SIAInterface* is realized by the *Melsec* class.

4.3.2 User settings

By default, after the installation of the connector on SIA platform there are no devices, such as in this case a PLC connection, and no items that can access any PLC memory address. This is done using a local network browser interface accessible at the IP address 10.20.30.40. Upon the adding of devices and items, *SIA core* will create the required struct items necessary for the retrieval and passing of data from and to the local database and user interface. *SIA core* will save the data passed by the user in the local web interface into the struct items used by the connector for reading and writing data to and from the PLC. The user inputted information in the local web interface can be found in the form of a string-to-string map data structure where the key is the name of the text input field and the value is the content of the text input field. This map bears the name *userDefinedFields*. Both struct items *DeviceInstancePlugin* and *ItemInstancePlugin* have such maps. The purpose for these maps is, as follows for each of the two structs, to hold data strings referring to:

DeviceInstancePlugin	- frame type - port - network id - device id
ItemInstancePlugin	- data type - memory address - timeout

In addition to these user defined fields the *DeviceInstancePlugin* has a member called *address* that stores the user inputted IP address and *DataItem* has a member called *value* that stores the value.

4.3.3 Connector initialization

According to information passed by SIA developers, the purpose of the initialization of the connector is to create the tables and column in the local database corresponding to the initialization parameters of the physical PLC. In turn the columns of the database will be used to create user input fields in SIA web interface. This method mirrors the settings made by the user in the PLC.
SIA interface provides a method that initializes the connector:

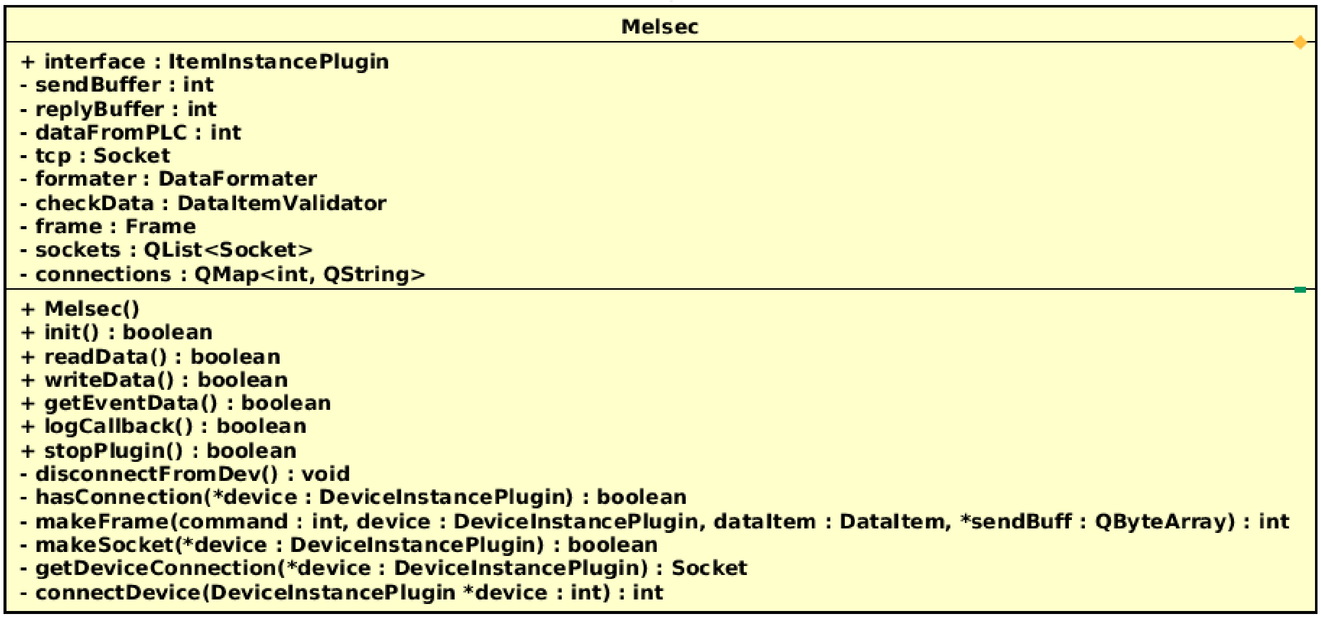
```
interface.initPlugin(NAME, 12, tableDataStructure, tableItemStructure, tableDeviceStructure, {}, false, false);
```

This method will take as parameters three struct of type *TableStructure,* passed by value, and they define the setting parameters of the connector.
These struct items are created and filled with *DatabaseColumnStructure* items before calling the interface initialization, as shown in snippet below.

```
DatabaseColumnStructure deviceIdStructure("device_id", DatabaseColumnStructure::TEXT,3);
tableDeviceStructure.push_back(deviceIdStructure);

DatabaseColumnStructure networkIdStructure("network_id", DatabaseColumnStructure::TEXT,3);
tableDeviceStructure.push_back(networkIdStructure);

DatabaseColumnStructure tcpPortStructure("port",DatabaseColumnStructure::INT, 3);
tcpPortStructure.insertSelectionValues("default" , QString::number(5500));
tableDeviceStructure.push_back(tcpPortStructure);

interface.initPlugin(NAME, 12, tableDataStructure, tableItemStructure, tableDeviceStructure, {}, false, false);
```

4.3.4 Connector classes

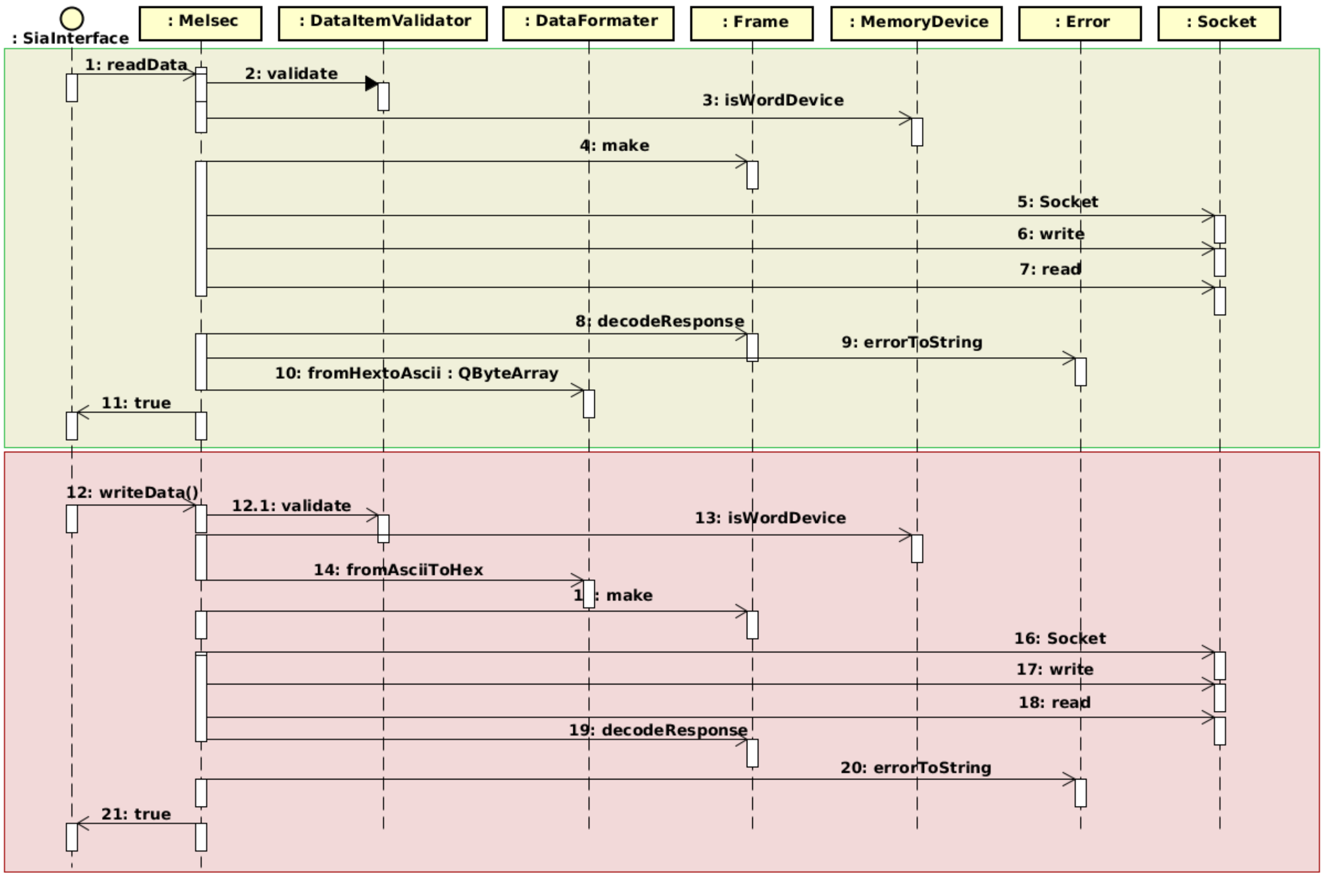
The implementation of the classes followed the design architecture and in the images below there can be seen the UML representation of for the Melsec and Frame classes. Some of the parameters of a few methods were hidden for the sake of aesthetics. For the complete versions and rest of the class diagrams refer to Git repository *Melsec*/SIA platform.asta



```
Melsec

+ interface : ItemInstancePlugin
- sendBuffer : int
- replyBuffer : int
- dataFromPLC : int
- tcp : Socket
- formater : DataFormater
- checkData : DataItemValidator
- frame : Frame
- sockets : QList<Socket>
- connections : QMap<int, QString>

+ Melsec()
+ init() : boolean
+ readData() : boolean
+ writeData() : boolean
+ getEventData() : boolean
+ logCallback() : boolean
+ stopPlugin() : boolean
- disconnectFromDev() : void
- hasConnection(*device : DeviceInstancePlugin) : boolean
- makeFrame(command : int, device : DeviceInstancePlugin, dataItem : DataItem, *sendBuff : QByteArray) : int
- makeSocket(*device : DeviceInstancePlugin) : boolean
- getDeviceConnection(*device : DeviceInstancePlugin) : Socket
- connectDevice(DeviceInstancePlugin *device : int) : int
```

4.3.5 Connector system sequence diagram

The interaction between the classes is depicted in the sequence diagram below.

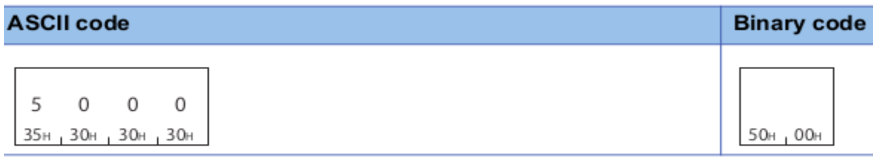


In the both of the cases read data (light green) and write data (pink) the *Melsec* class delegates the validation of user input to the *DataItemValidator* and *MemoryDevice* classes, the construction of the frame to be sent and the decomposition of the frame received from the PLC to the *Frame* class, the frame transmission and response receiving, to and from the PLC, to the *Socket* class, and if the entire process was successful will reply with a *true* value to the calling read or write function, otherwise with *false* and will log an error string.

4.3.6 Reading/writing data

4.3.6.1 Melsec communication protocol

The Melsec communication protocol was designed by Mitsubishi to access data and settings registers in their programmable logic controllers. The protocol is based on two communication mediums: serial and Ethernet communication. Each PLC is equipped with peripherals for one or both mediums. The chosen medium for this project is Ethernet. The communication is achieved by sending to the PLC a frame of characters, that based on their content and position in the frame forms a command. Depending on the PLC model, the frame used to communicate with the PLC could be, as defined by Melsec protocol, either 1E, 3E or 4E. Each one of these have two formats, first as ASCII characters, second as binary representation of the ASCII characters. The ASCII



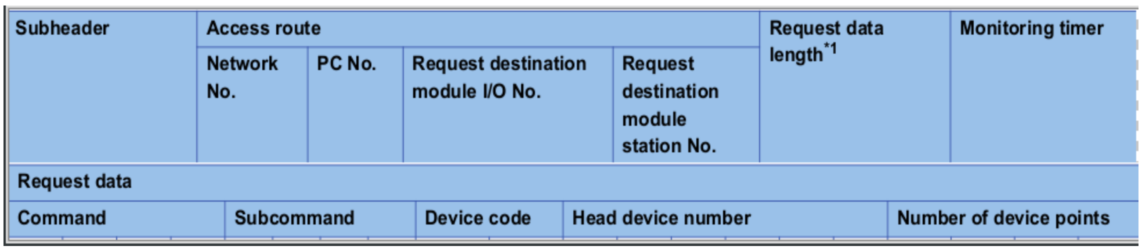
ASCII code	Binary code
5 0 0 0 35H 30H 30H 30H	50H 00H

format, compared to the binary format, is less complex due to its human readable form but it takes double the time for transmission since each ASCII character has double the size in bits of its binary counterpart.

4.3.6.2 Frame composition

As mentioned in the previous paragraph, the model of PLC determines which frame format can be used to achieve communication. The model in this project is Q02 from the Q-series. According to the producer, this model is compatible with the 3E and 4E frames. Further investigation in the official documentation revels that the 4E frame is an extension of the 3E that includes an extra layer of addressing security by including a random serial number on the frame in the sub header block. This will allow the mapping of PLC responses to the queries as the PLC will include the serial number of the frame from the query into the response.
The decided approach was to develop a frame manager based on the 3E format, and later to expand it to 4E format by adding the serial number.

The format of the 3E frame is depicted in the following image.



Subheader	Access route				Request data length[*1]	Monitoring timer
	Network No.	PC No.	Request destination module I/O No.	Request destination module station No.		

Request data				
Command	Subcommand	Device code	Head device number	Number of device points

The main blocks of the frame are sub header, access route, request data length, monitoring timer and request data.

The Frame class is responsible for the composition of communication frames and the input received from the Melsec class will determine the type of the frame to be composed.

The composition of the frame is handles in the class Frame by the method make(). The parameters passed to this method are strings containing the that represent the non-fixed values of the frame. The fixed values of the frame are stored in private members of the class as strings. The method make() returns a byte array that contains the entire frame as requested by the Melsec class.

4.3.6.2.1 Sub header

The sub header refers to the type of frame that the PLC will receive. It is composed of fixed distinct values for 3E and 4E and are defines in the header file of the Frame class as follows:

```
QString subheader_3E = "5000";
QString subheader_4E = "5400";
```

4.3.6.2.2 Access route

The network number refers to internal network settings of the PLC. If the PLC connected to a multi-layered network, this number would represent the identification number of the node network that the PLC is part of. The value is retrieved from DeviceInstancePlugin.userDefinedFields["network_id"]

The PC no. is the number of the PLC in the network. The value is retrieved from DeviceInstancePlugin.userDefinedFields["device_id"]

The request destination module I/O number is the identification number of the input/output extension board attached to the CPU unit of the PLC. The request destination module station number is the identification number of the redundant CPU attached to the main CPU unit. In the case here, the default values for the request destination module I/O and the request destination module station will be used since the main CPU of the PLC is accessed. These values are defined in the header file of the Frame class as follows:

```
QString def_RDM_IO = "03FF"; // default Request destination module I/O No.
QString def_RDM_S = "00"; // default Request destination module station No.
```

4.3.6.2.3 Request data length

Request data length is a number that specifies the sum in bytes of the sizes of monitor timer block and request data block. This number is calculated after the processing of the monitor timer and request data blocks. After adding the size of these two blocks, the result is inserted in front of the monitoring timer block as a byte array containing the ASCII value of the obtained number in base 16. The size of this block is fixed to 4 bytes. For this reason, any non-used byte will be initialized with the ASCII value of character ‘0’. For example, knowing that the size of the monitoring timer is 4 bytes in ASCII frame format (see next paragraphs) and that request data has a size of 20 bytes, the request data length byte array is: “0018” where 18 is the sum of the sizes in base 16. Since this is not a static value, in order to get the right value a call the method mcDataLenght() from the Frame class will return it.

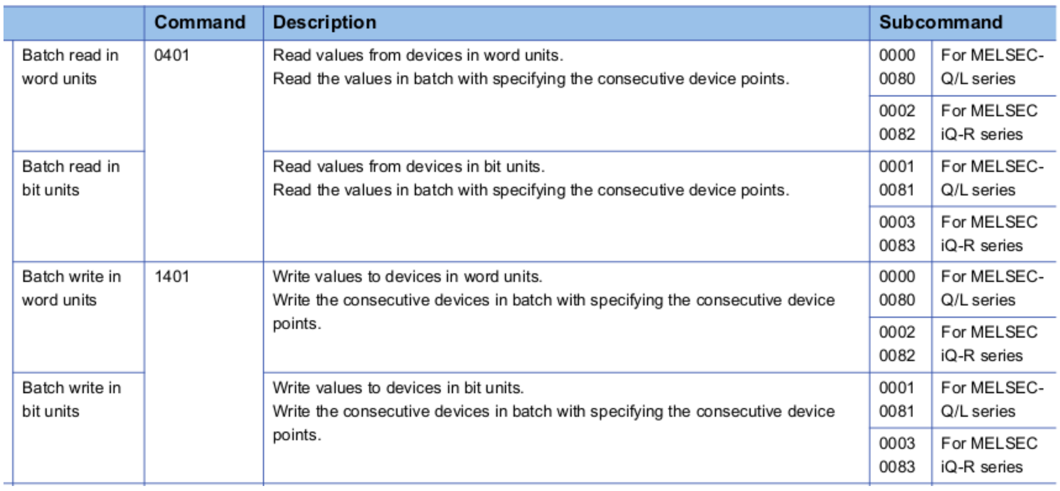
4.3.6.2.4 Monitoring timer

Monitoring timer block is a setting for the wait time up to the completion of reading and writing processing. The range of this value is between 0001 hexadecimal to FFFF hexadecimal (0 to 65535 decimal) and the waiting time unit is 250 milliseconds. If 0000 is set, then the waiting time is set to infinite. The size of this block is 4 bytes for ASCII format and 2 bytes for binary format.
The value for this is provided by ItemInstancePlugin.userDefinedFields["timeout"].

4.3.6.2.5 Request data

Request data block is composed, in this order, from a command, sub command, device code, head device number, number of device points and the actual data, if the command is a write command.

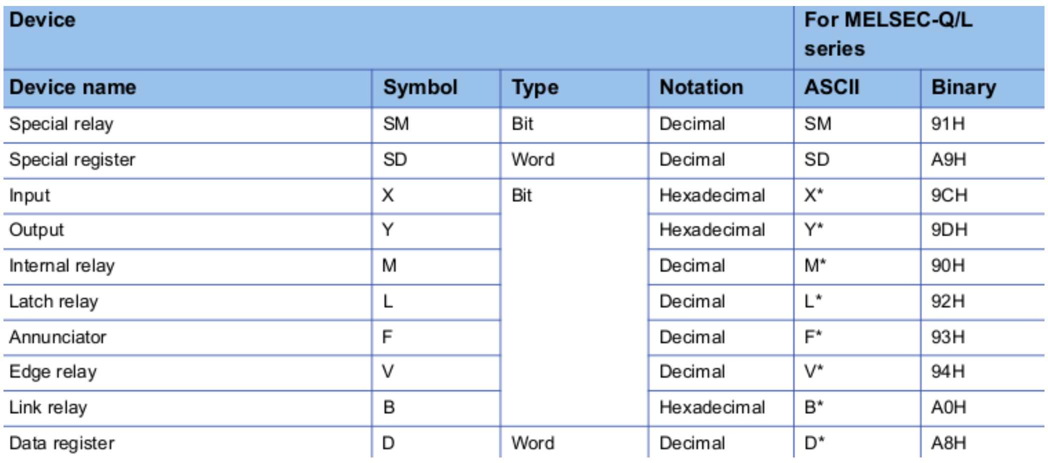
The command and the sub command are defined by the Melsec protocol as it can be seen in the following image:



	Command	Description	Subcommand	
Batch read in word units	0401	Read values from devices in word units. Read the values in batch with specifying the consecutive device points.	0000 0080	For MELSEC-Q/L series
			0002 0082	For MELSEC iQ-R series
Batch read in bit units		Read values from devices in bit units. Read the values in batch with specifying the consecutive device points.	0001 0081	For MELSEC-Q/L series
			0003 0083	For MELSEC iQ-R series
Batch write in word units	1401	Write values to devices in word units. Write the consecutive devices in batch with specifying the consecutive device points.	0000 0080	For MELSEC-Q/L series
			0002 0082	For MELSEC iQ-R series
Batch write in bit units		Write values to devices in bit units. Write the consecutive devices in batch with specifying the consecutive device points.	0001 0081	For MELSEC-Q/L series
			0003 0083	For MELSEC iQ-R series

The value for the command is handled in the *Frame* class, by the method *frameSpecificCmd(),* by prepending the character '0' for when instructed to compose a read frame or '1' for a write frame to the fixed byte array "401". The sub command "0000" is appended to the command since the PLC model in use is from the Q series and only word units are accessed.

The device code refers to the name of a memory area in the internal memory of the PLC. Each area has a purpose and a code as described in the following table:



Device				For MELSEC-Q/L series	
Device name	Symbol	Type	Notation	ASCII	Binary
Special relay	SM	Bit	Decimal	SM	91H
Special register	SD	Word	Decimal	SD	A9H
Input	X	Bit	Hexadecimal	X*	9CH
Output	Y		Hexadecimal	Y*	9DH
Internal relay	M		Decimal	M*	90H
Latch relay	L		Decimal	L*	92H
Annunciator	F		Decimal	F*	93H
Edge relay	V		Decimal	V*	94H
Link relay	B		Hexadecimal	B*	A0H
Data register	D	Word	Decimal	D*	A8H

The class *MemoryDevice* keeps a map of the symbols of these devices to their ASCII code. Furthermore, it provides methods that identify if a memory device is either a bit device (the size of the memory units a bit) or a word device (the size of the memory units a word – 16 bits). *Melsec* class calls the *DataItemValidator* class to retrieve the device ASCII code corresponding to the symbol inputted by the user in the *ItemInstancePlugin.userDefinedFields["memory_address"].* The retrieved ASCII code will be passed to the *Frame* call method *make().*

The head device number refers to the identification number of a memory cell in a designated memory area, e.g., "D 0000" is the first memory cell of 'D' area, "D 0001" is the second memory cell of 'D' area and so on. The range of this number is given by the Mitsubishi is the models corresponding data book and is referred as maximum number of device points. In this case for the targeted model Q02 and for the targeted memory device 'D' the maximum number of device points is 122880. This value is retrieved from the *ItemInstancePlugin.userDefinedFields["memory_address"].*

The number of device points refers to the number of sequential memory cells to be accessed. This is this is required, first, the data access commands sent to the PLC are accessing a batch of memory cells, and the size of the batch needs to be specified and second that there are data types that occupy more than one word.
This number is retrieved from *DataFormater* class by the *getSizeInWords()* method based on the data type passed by the user in the *ItemInstancePlugin.userDefinedFields["data_type"].*.

The data is a formatted byte array of ASCII characters representing hexadecimal values that will be written to the PLC memory. The non-formatted (human readable) data is retrieved from the *DataItem.value["values"]* map. The *DataFormater* class is responsible with the conversion between human readable and PLC accepted values and provides the methods *fromAsciiToHex()* for human readable to PLC accepted values conversion and *fromHexToAscii()* for the inverse.

The available data types can be found in the following table:

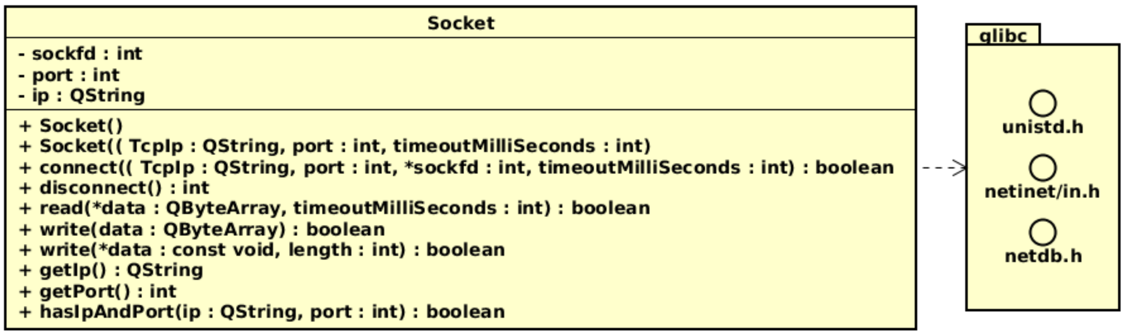
User selection value	Data type
BOOL	Bit
INT16	Signed 16 bits Integer
INT32	Signed 32 bits Integer
INT64	Signed 64 bits Integer
UINT16	Unsigned16 bits Integer
UINT32	Unsigned 32 bits Integer
UINT64	Unsigned 64 bits Integer
REAL	Single precision floating point real
LREAL	Double precision floating point real
WORD	Hexadecimal 16 bits - word

In the cases of real numbers, the IEEE Standard for Floating-Point Arithmetic (IEEE 754) is used.
Prior to the composition of the write command, the *DataItemValidator* class performs a test on the user inputted data, using the *DataFormater* class methods. This test consists of the sequential conversion, based on the data and the data type passed by the user, of the data from string (ASCII characters) to the hexadecimal value and back from the obtained hexadecimal to string. Since these two methods are inverse functions, the result will correspond to the input only in the case of a correct mapping of the inputted value to the data type.

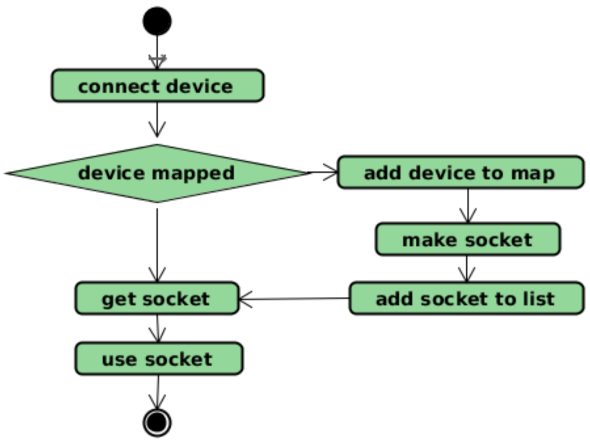
4.3.6.3 Connection management

The TCP connection to the PLC is provided by the Socket class. Since the running system (SIA core) will use this connector in a Linux environment, the connector will use the application binary interfaces (ABI) from the GNU C library (glibc) for the implementation of a TCP connection.

This dependency is depicted in Socket class diagram below:


Socket
- sockfd : int
- port : int
- ip : QString
+ Socket()
+ Socket((TcpIp : QString, port : int, timeoutMilliSeconds : int)
+ connect((TcpIp : QString, port : int, *sockfd : int, timeoutMilliSeconds : int) : boolean
+ disconnect() : int
+ read(*data : QByteArray, timeoutMilliSeconds : int) : boolean
+ write(data : QByteArray) : boolean
+ write(*data : const void, length : int) : boolean
+ getIp() : QString
+ getPort() : int
+ hasIpAndPort(ip : QString, port : int) : boolean
glibc
unistd.h
netinet/in.h
netdb.h

The connector may be used with a variable number of DeviceInstancePlugin items that may be connected to the same or to different PLCs. This implies that a TCP connection must be provided for each of them. This may lead to either a waste of operating system resources or a conflicting call to an already in use socket file descriptor. To avoid this, Melsec class keeps a record of all the DeviceInstancePlugin items in a map where the key is the id of DeviceInstancePlugin and the value is the target IP address, and a record of all the Socket items created. For each new DeviceInstancePlugin that needs to be an inserted in the map, a check is performed if there is already in the list of sockets one that matches the IP destination and port, and if not a new one will be created, as shown in the flowchart below:

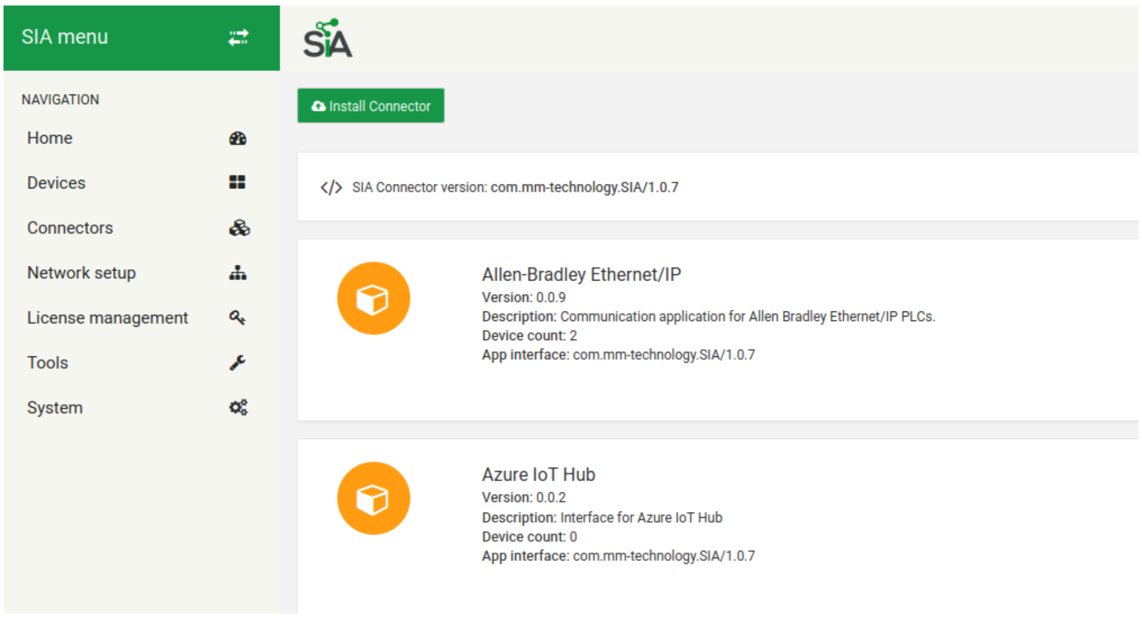


4.3.7 SIA connector integration

The Melsec package was compiled as a library using a Docker container that provides the required SIA libraries, and an ARM GCC compiler, since SIA runs on an embedded Linux platform that is powered by a 32-bit ARM processor. A compressed .sia type file that contains the resulting libmelsec.so library and a .xml info file is the deliverable connector, ready for installation on SIA. The info file contains in a predefined format the name, title, description, and library name of the connector as seen in the snippet below:

```
<?xml version='1.0' encoding='UTF-8'?>
<plugin>
    <name>melsec</name>
    <title>Melsec Connector</title>
    <description>Melsec Connector SIA</description>
    <file>libmelsec.so</file>
</plugin>
```

The installation of a new connector is done by accessing the *Connector* menu from SIA web-interface that can be accessed via the IP address of SIA platform 10.20.30.40. as in the image below:

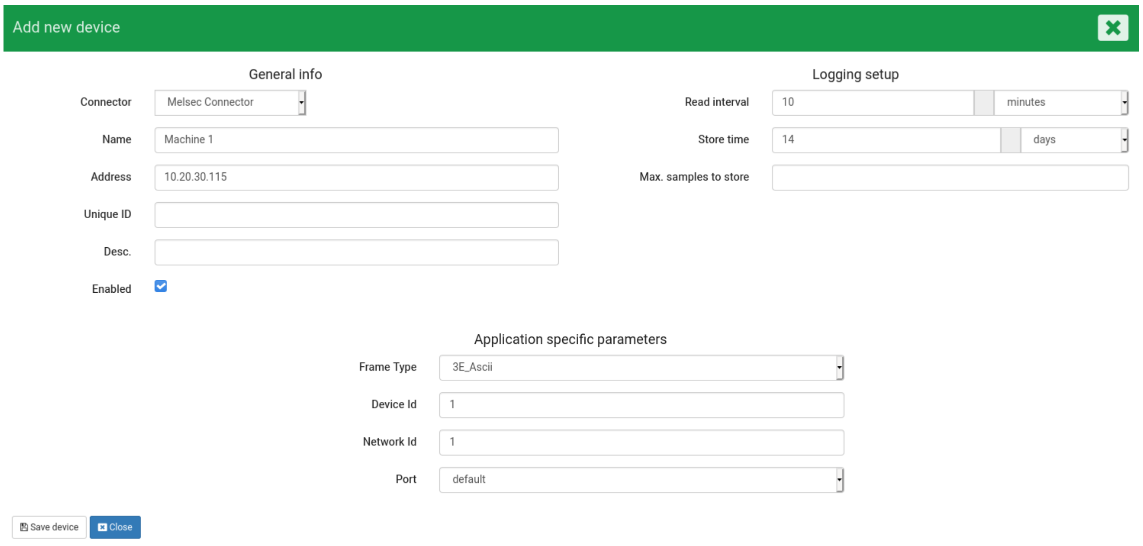


SIA Connector version: com.mm-technology.SIA/1.0.7

Allen-Bradley Ethernet/IP
Version: 0.0.9
Description: Communication application for Allen Bradley Ethernet/IP PLCs.
Device count: 2
App interface: com.mm-technology.SIA/1.0.7

Azure IoT Hub
Version: 0.0.2
Description: Interface for Azure IoT Hub
Device count: 0
App interface: com.mm-technology.SIA/1.0.7

After the installation of the connector, it will appear in the list of connectors and it can be used for the creation of new devices as in the image below.



Add new device

General info

Connector	Melsec Connector
Name	Machine 1
Address	10.20.30.115
Unique ID	
Desc.	
Enabled	✓

Logging setup

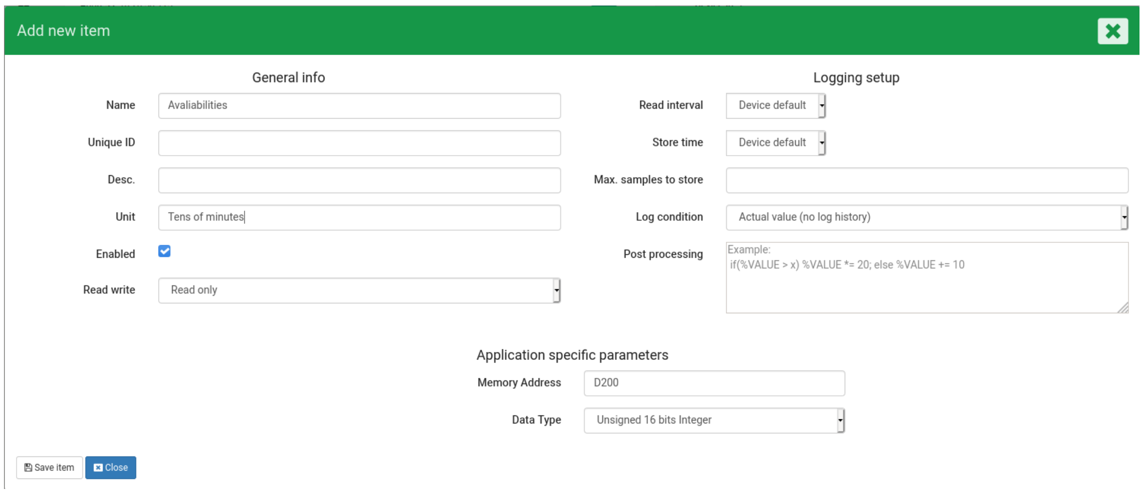
Read interval	10 minutes
Store time	14 days
Max. samples to store	

Application specific parameters

Frame Type	3E_Ascii
Device Id	1
Network Id	1
Port	default

Save device Close

The next step in setting up the running environment is to create data items that will point to the PLC's target memory location:



Add new item

General info

Name	Avaliabilities
Unique ID	
Desc.	
Unit	Tens of minutes
Enabled	✓
Read write	Read only

Logging setup

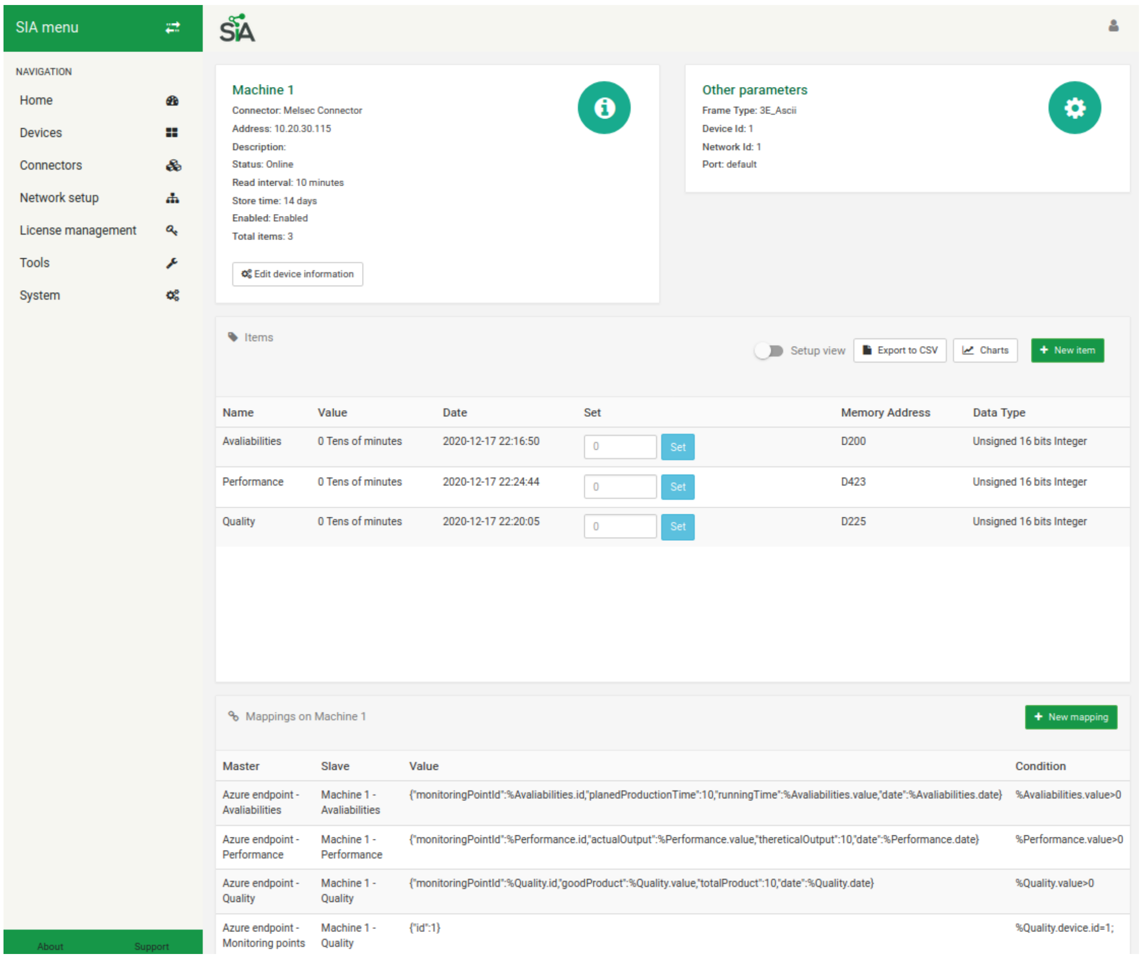
Read interval	Device default
Store time	Device default
Max. samples to store	
Log condition	Actual value (no log history)
Post processing	Example: if(%VALUE > x) %VALUE *= 20; else %VALUE += 10

Application specific parameters

Memory Address	D200
Data Type	Unsigned 16 bits Integer

Save item Close

The final step is to set a mapping relationship between the data access connector and the Rest API connector with the specific JSON format and the acquired data as in the image below:



Machine 1
Connector: Melsec Connector
Address: 10.20.30.115
Description:
Status: Online
Read interval: 10 minutes
Store time: 14 days
Enabled: Enabled
Total items: 3

Other parameters
Frame Type: 3E_Ascii
Device Id: 1
Network Id: 1
Port: default

Name	Value	Date	Set	Memory Address	Data Type
Avaliabilities	0 Tens of minutes	2020-12-17 22:16:50	0 Set	D200	Unsigned 16 bits Integer
Performance	0 Tens of minutes	2020-12-17 22:24:44	0 Set	D423	Unsigned 16 bits Integer
Quality	0 Tens of minutes	2020-12-17 22:20:05	0 Set	D225	Unsigned 16 bits Integer

Master	Slave	Value	Condition
Azure endpoint - Avaliabilities	Machine 1 - Avaliabilities	{"monitoringPointId":%Avaliabilities.id,"planedProductionTime":10,"runningTime":%Avaliabilities.value,"date":%Avaliabilities.date}	%Avaliabilities.value>0
Azure endpoint - Performance	Machine 1 - Performance	{"monitoringPointId":%Performance.id,"actualOutput":%Performance.value,"thereticalOutput":10,"date":%Performance.date}	%Performance.value>0
Azure endpoint - Quality	Machine 1 - Quality	{"monitoringPointId":%Quality.id,"goodProduct":%Quality.value,"totalProduct":10,"date":%Quality.date}	%Quality.value>0
Azure endpoint - Monitoring points	Machine 1 - Quality	{"id":1}	%Quality.device.id=1;

5 Test

5.1 Front-end test (Nikola)

5.2 Backend test (Balkis)

Different testing techniques were used to test all components in the backend.
This section will go through the following tests, unit testing, APIs behaviour testing, ETL process testing, and test specifications. Some tests do not have test cases in the analysis section since they are not directly specified in the requirements.

Unit testing: (Balkis)

The purpose of unit testing is to test the smallest parts – functions of the system. Localizing any theoretical system failures in the smallest possible unites leads to easier problem identification. Also, the bigger the project gets, the harder it is to keep track of the relation between the code and the functionalities, which might result in breaking one functionality by fixing another one. This approach for error detection in early stages can prevent system failures at critical stages.

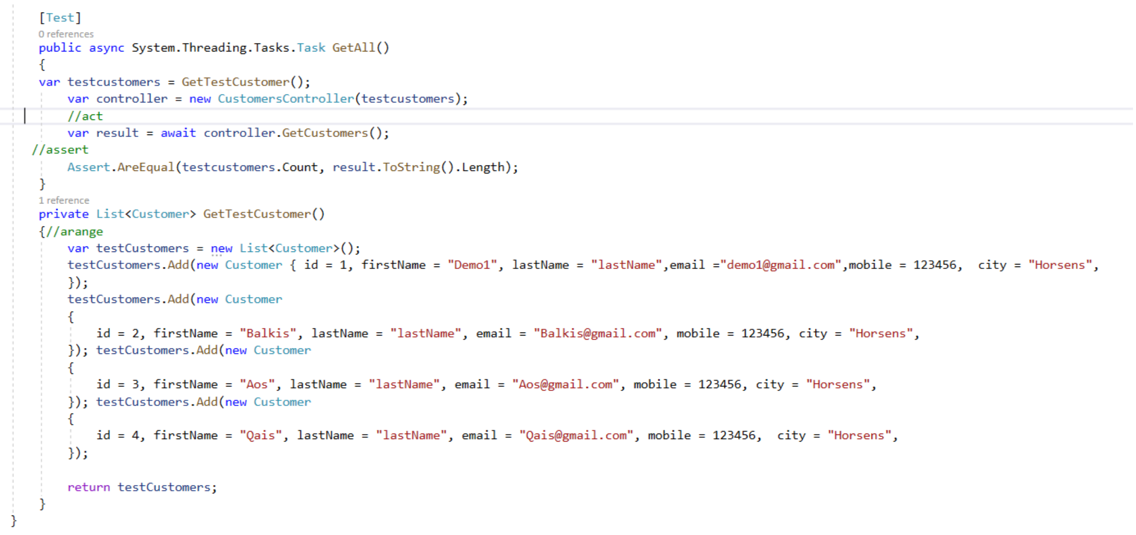
C# unit testing: (Balkis)

Visual Studio had an advanced feature called IntelliTest. It explores your .NET code to generate test data and a suite of unit tests. For every statement in the code, a test input is generated that will execute that statement. A case analysis is performed for every conditional branch in the code. For example, if statements, assertions, and all operations that can throw exceptions are analysed. This analysis is used to generate test data for a parameterised unit test for each of your methods, creating unit tests with high code coverage. When you run IntelliTest, you can easily see which tests are failing and add any necessary code to fix them. You can select which of the generated tests to save into a test project to provide a regression suite. As you change your code, rerun IntelliTest to keep the generated tests in sync with your code changes (IntelliTest).

Unfortunately, this feature was disabled when I wrote my test and Microsoft is still trying to get the functions back. Therefore, I had to manually write the test. The following test shows a unit test I created to test the functionality of GetCustomer from the controller class. I have followed the pattern AAA (Arrange, Act, Assert) pattern is a common way of writing unit tests for a method under test. (unite test pattern)

- The Arrange section of a unit test method initializes objects and sets the value of the data that is passed to the method under test.
- The Act section invokes the method under test with the arranged parameters.
- The Assert section verifies that the action of the method under test behaves as expected.

I first mock some customers' data for a test. And then I invoked the controller class and passed the fake data. I called the method GetCustomers from the controller. And finally, I expect the count from the mock data method is equal to the one from the controller.



```
    [Test]
    0 references
    public async System.Threading.Tasks.Task GetAll()
    {
    var testcustomers = GetTestCustomer();
        var controller = new CustomersController(testcustomers);
        //act
        var result = await controller.GetCustomers();
   //assert
        Assert.AreEqual(testcustomers.Count, result.ToString().Length);
    }
    1 reference
    private List<Customer> GetTestCustomer()
    {//arange
        var testCustomers = new List<Customer>();
        testCustomers.Add(new Customer { id = 1, firstName = "Demo1", lastName = "lastName",email ="demo1@gmail.com",mobile = 123456,  city = "Horsens",
        });
        testCustomers.Add(new Customer
        {
            id = 2, firstName = "Balkis", lastName = "lastName", email = "Balkis@gmail.com", mobile = 123456, city = "Horsens",
        }); testCustomers.Add(new Customer
        {
            id = 3, firstName = "Aos", lastName = "lastName", email = "Aos@gmail.com", mobile = 123456, city = "Horsens",
        }); testCustomers.Add(new Customer
        {
            id = 4, firstName = "Qais", lastName = "lastName", email = "Qais@gmail.com", mobile = 123456,  city = "Horsens",
        });

        return testCustomers;
    }
}
```

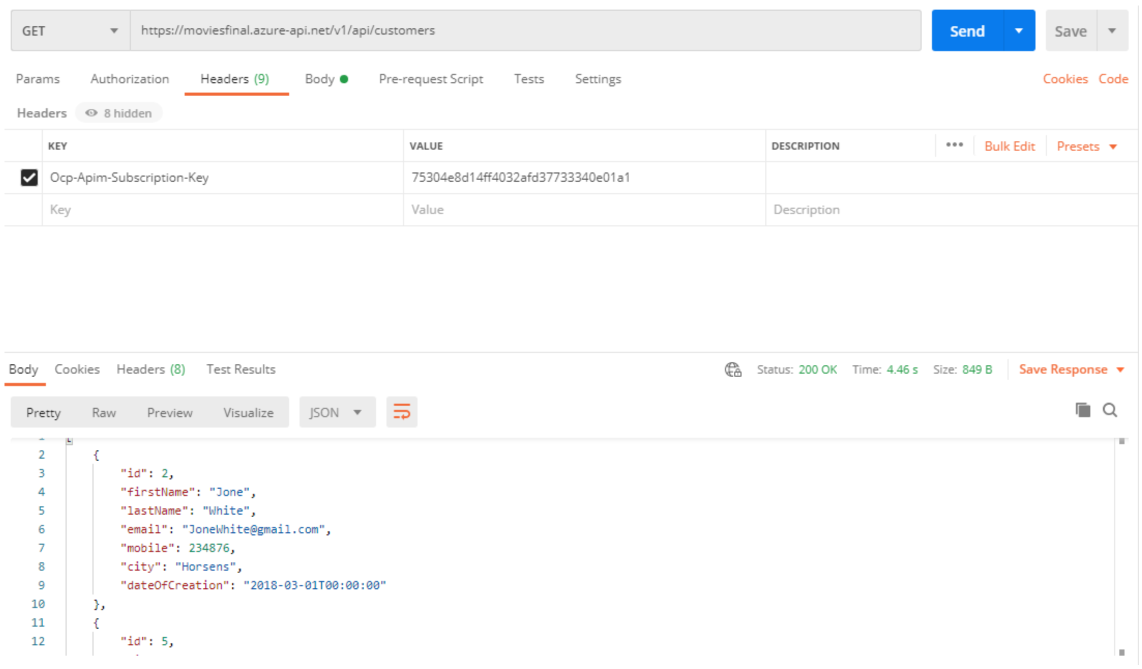
5.2.1 APIs behaviour testing: (Balkis)

Black Box technique was used to test backend system behaviour using Postman, to ensure that the APIs behave the way that is designed. The following test table shows an example of testing one of the APIs.

Test Scenario	**Test case**	**Expected result**	**Actual result**	**pass/ Fail**
Check get customer function.	The API caller enters invalid subscription key.	{ "statusCode": 401, "message": "Access denied due to invalid subscription key. Mak e sure to provide a valid key for an active subscription." }	{ "statusCode": 401, "message": "Access denied due to invalid subscription key. Mak e sure to provide a valid key for an active subscription." }	Passed
Check get customer function.	The API caller enters a valid subscription key.	The API caller receives a list of customers in JSON format.	The API caller got a list of customers in JSON format.	Passed

Checking editing customer info	Edit customer that does not exist.	Not found	Not found	Passed
Checking editing customer info	Edit customer that exists	The customer gets updated in the database.	The customer gets updated in the database.	passed
Checking editing customer info	The id in the endpoints is different than the one in the body	Bad request	Bad request.	passed
Checking adding a new customer	Entering valid data	A new customer is added to the database	A new customer is added to the database	passed
Check to delete customer	Deleting non-Existing customer	Not found message	Not found message	Passed
Check to delete customer	Deleting Exist customer	The customer gets removed.	The customer gets removed.	Passed

The snip below was taken when performing the black box test in Postman. A get request was sent to the customer API URL, inside the request header the subscription key is included. We see a 200 ok response which indicates that the request has succeeded. A list of the customers is returned.



KEY	VALUE	DESCRIPTION
☑ Ocp-Apim-Subscription-Key	75304e8d14ff4032afd37733340e01a1	
Key	Value	Description

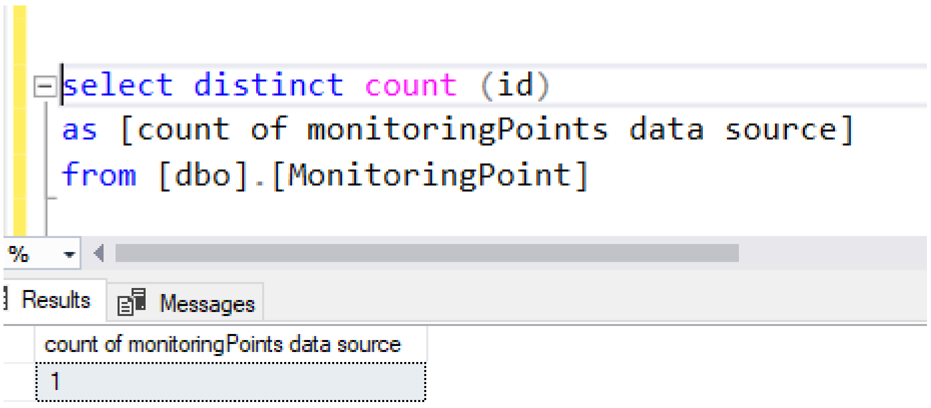
```
    {
        "id": 2,
        "firstName": "Jone",
        "lastName": "White",
        "email": "JoneWhite@gmail.com",
        "mobile": 234876,
        "city": "Horsens",
        "dateOfCreation": "2018-03-01T00:00:00"
    },
    {
        "id": 5,
```

5.2.2 ETL process testing: (Balkis)

- Data completeness test: data completeness testing validates that all the expected source data has been successfully loaded to the target (Data Warehouse Testing 101 | Panoply, no date).

  After I finished the ETL Implementation, I performed this test.

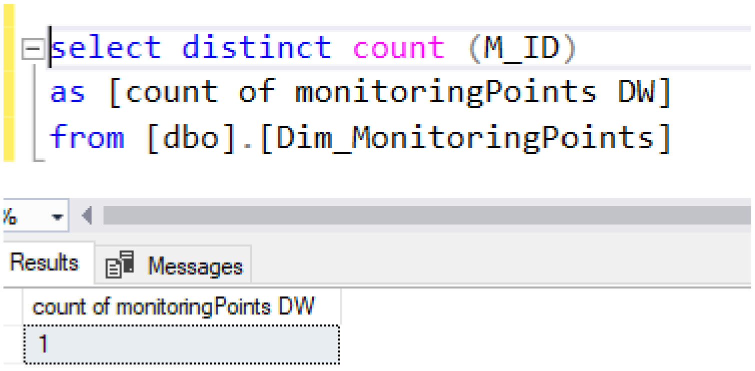
  First, I made sure that all the rows from the data source are loaded in the data destination and. To reach my goal, I run some SQL queries; I will take the monitoring points table as an example. I compared the number of monitoring points in the data source and the Data warehouse. The following two snips show the queries were used. The count should be the same in both tables.



```
select distinct count (id)
as [count of monitoringPoints data source]
from [dbo].[MonitoringPoint]
```

Results | Messages

count of monitoringPoints data source
1

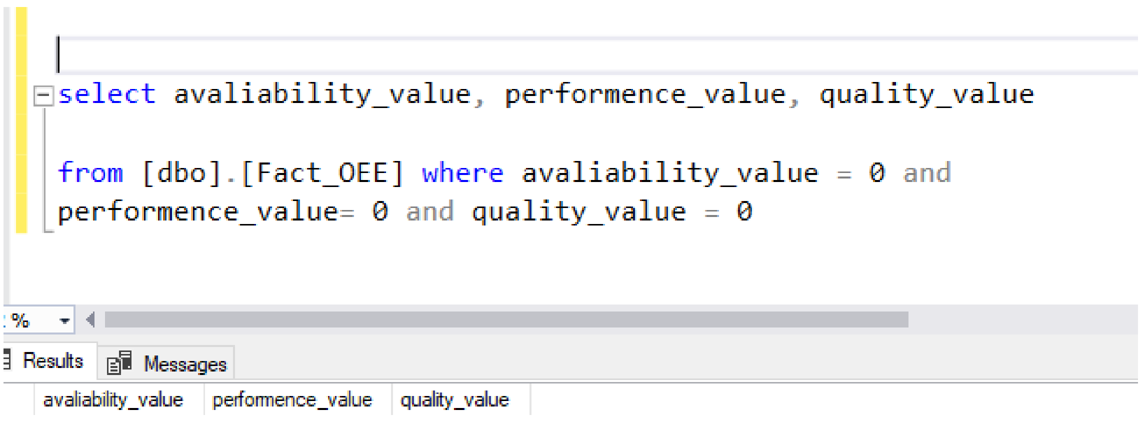


```
select distinct count (M_ID)
as [count of monitoringPoints DW]
from [dbo].[Dim_MonitoringPoints]
```

Results | Messages

count of monitoringPoints DW
1

- Data transformation testing: The attributes in the fact table have a different type than the one in the data source. Therefore, data transformed is performed. To ensure that successfully executed I run an SQL query to check the values of production monitoring parameters (availability, performance, quality) are not zero. When I ran the following SQL Statement, I got no rows back, which meant the test passed.



```
select avaliability_value, performence_value, quality_value

from [dbo].[Fact_OEE] where avaliability_value = 0 and
performence_value= 0 and quality_value = 0
```

```
avaliability_value    performence_value    quality_value
```

- ETL performance testing: ETL performance testing is end-to-end testing to ensure that all ETL process steps are working with expected data volumes (Data Warehouse Testing 101 | Panoply, no date).

  The ETL process is executed in no time since the source code’s size is small.

- Incremental ETL testing: Incremental ETL testing verifies that updates on the sources are getting loaded into the target system correctly (Data Warehouse Testing 101 | Panoply, no date).

  The incremental ETL testing for this project is done by executing the ETL process when the relational database has new, updated, and deleted rows. Then, the data completeness testing needs to be performed again to ensure the ETL is inserting the new and updated rows and setting the deleted rows correctly.

5.2.3 Test Specifications: (Balkis)

The following test table verifies whether the requirements were fulfilled or not from the backend side. This way, testing will cover all the functionalities from the list of the functional requirements.

	Requirement	Implemented
1	As a clerk and production assistant, I want to be able to add new customers so that I can save their contact info.	Yes
2	As a clerk and production assistant, I want to be able to delete customers from the system so that I can remove them from the system.	Yes
3	As a clerk and production assistant, I want to be able to edit customers so that I can change their data.	Yes
4	As a clerk and production assistant, I want to be able to get a list of all customers so I can have answers to certain questions.	Yes
5	As a clerk and production assistant, I want to be able to create orders so that I can submit it to the production area.	Yes
6	As a clerk and production assistant, I want to be able to delete orders from the system so that I can remove cancelled orders from the system.	Yes

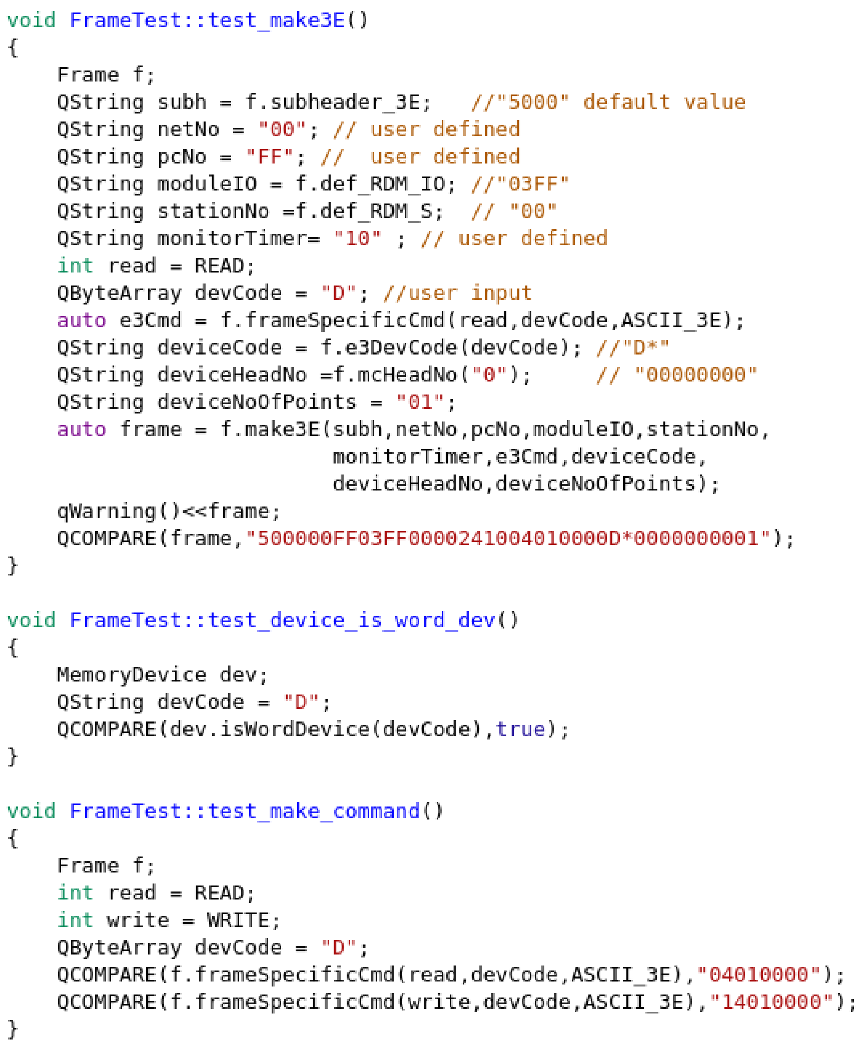
7	As a clerk and production assistant, I want to be able to edit orders so that I can make changes to the orders.	Yes
8	As a clerk and production assistant, I want to be able to get a list of all orders so that I can do some statistics.	Yes
9	As a clerk and production assistant, I want to be able to add new mobiles to the system so that I can decide what mobile cases we can make.	Yes
10	As a clerk and production assistant, I want to be able to delete mobiles from the system so that I can remove unwanted mobiles from the system.	Yes
11	As a clerk and production assistant, I want to be able to edit mobiles so that I can make a change to the mobile's name.	Yes
12	As a clerk and production assistant, I want to be able to get a list of all mobiles so that I can tell what mobile cases we produce.	Yes
13	As a clerk and production assistant, I want to be able to add new mobile cases design to the system so that I can decide what case design we can produce.	Yes
14	As a clerk and production assistant, I want to be able to delete mobile cases design from the system so that I can remove the unwanted designs from the system.	Yes

15	As a clerk and production assistant, I want to be able to edit mobile cases design so that I can make changes to the design description.	Yes
16	As a clerk and production assistant, I want to be able to get a list of all mobile case designs so I can tell what designs we make.	Yes
17	As a clerk and production assistant, I want to see a chart of the production Availability, so that I can make a timely intervention.	Yes
18	As a clerk and production assistant, I want to see a chart of the production performance, so that I can make a timely intervention.	Yes
19	As a clerk and production assistant, I want to see a chart of the production quality, so that I can make a timely intervention.	Yes
20	As a clerk and production assistant, I want to see a chart of the production OEE, so that I can make a timely intervention.	Yes
21	As a management team member, I want to see averages for each production monitoring parameter so I can develop improvement plans.	Yes
22	As a management team member, I want to see history for produced items so I can develop improvement plans.	Yes

23	As a management team member, I want to have a structured history of data in order to create reports.	Yes

5.3 Data acquisition sub-system test (Alex)

Several unit tests were performed on the classes in the package Melsec using the QtTest library provided by the Qt framework. A few of these test cases referring to the *Frame* class are in the image below. The source code test project can be found in



```
void FrameTest::test_make3E()
{
    Frame f;
    QString subh = f.subheader_3E;   //"5000" default value
    QString netNo = "00"; // user defined
    QString pcNo = "FF"; //  user defined
    QString moduleIO = f.def_RDM_IO; //"03FF"
    QString stationNo =f.def_RDM_S;  // "00"
    QString monitorTimer= "10" ; // user defined
    int read = READ;
    QByteArray devCode = "D"; //user input
    auto e3Cmd = f.frameSpecificCmd(read,devCode,ASCII_3E);
    QString deviceCode = f.e3DevCode(devCode); //"D*"
    QString deviceHeadNo =f.mcHeadNo("0");     // "00000000"
    QString deviceNoOfPoints = "01";
    auto frame = f.make3E(subh,netNo,pcNo,moduleIO,stationNo,
                          monitorTimer,e3Cmd,deviceCode,
                          deviceHeadNo,deviceNoOfPoints);
    qWarning()<<frame;
    QCOMPARE(frame,"500000FF03FF000024100401000 0D*0000000001");
}

void FrameTest::test_device_is_word_dev()
{
    MemoryDevice dev;
    QString devCode = "D";
    QCOMPARE(dev.isWordDevice(devCode),true);
}

void FrameTest::test_make_command()
{
    Frame f;
    int read = READ;
    int write = WRITE;
    QByteArray devCode = "D";
    QCOMPARE(f.frameSpecificCmd(read,devCode,ASCII_3E),"04010000");
    QCOMPARE(f.frameSpecificCmd(write,devCode,ASCII_3E),"14010000");
}
```

6 Results and Discussion (Balkis)

The following section presents the outcome and the achieved results of the entire system and discusses the project period's primary cycles.

As a result of the project, The Melsec connector was successfully installed in the SIA platform and thus the system is able to retrieve data from the PLC's memory banks and with the usage of the RESTful API connector to send it to the designated endpoint.

All the functional requirements from the Analysis section were successfully implemented and fully functioning from the backend side, as stated earlier in the report.
The Power BI application is able to visualise the history of the data stored in the Data warehouse.

Regarding the Web application, its current state is still under development. The Angular application is running under a local server for developing and testing purposes. The application can not be accessed from external devices. The application can query data from a web test server.

The rest of this section will discuss the four main part of the project.
Starting from the Analysis to design followed by implementation and ending with the test.
Each of the stated parts is documented and described in details in the previous sections.
The analysis part presents the system requirements and the actor roles of the users. This part of the project was challenging since the group did not have precise requirements for the system.

Talking about the design part, this part of the system shows its structure. The challenges the group had to face are how do we connect the sup-systems we have. Moving to the implementation part, I believe all of us had some challenges during this

phase, and we did our best to solve the issues we faced, most of it was solved, but unfortunately, the implementation of the web app did not go as planned. It ended up with having non-responsive web app.
Finally, the testing part, each of the team used the proper technique to test his part of the system.

From a general point of view, We can not say that the result is satisfied since we still missing the web app, but at least we have the other two parts working properly.

7 Conclusions (Balkis)

This project's primary goal is a real-time visualisation of the data coming for the factory and business intelligence tool that allow the discussion maker to make faster decisions.
This project uses multiple programming languages, farmworks and technologies. (C#, C, C++, JavaScript, power BI, Azure cloud, Microsoft teams, Zoom, GitHub and much more) in order to achieve the stated goal. Looking into all these aspects that should be considered, the project is complex and implementing the desired requirements and making the entire system work together is a lot. Despite the web app not being achieved, all the other parts are fulfilled, making, the system a good base for future developments.

8 Project future

For project future, I think it is really a good idea to work on fo work in a new feature that allows the user to adjust the production amount remotely or being able to see the produced products for a specific order.

The system now only handle data coming from one PLC this can be expanded to multiple PLC in future projects.

Web app application to be implemented entirely and hosted on Firebase Hosting. Firebase is a good choice for hosting Single page applications and is free to use. It has access to Firebase realtime database, which provides real-time access to data on different platforms.

9 Sources of information (Balkis)

what is C#
https://www.c-sharpcorner.com/article/what-is-c-sharp/

Microsoft documentation:
https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15

Welcome to Microsoft Teams - Microsoft Teams | Microsoft Docs (no date)
https://docs.microsoft.com/en-us/microsoftteams/teams-overview?fbclid=IwAR2C9pUpEo8H6SMudc3Q1-RvfM-TwFoRBAkCYlNj-dToUhNTQvLc71bdm-E

Download Power BI
https://www.microsoft.com/en-us/download/details.aspx?id=58494

intelliTest by Microsoft
https://docs.microsoft.com/en-us/visualstudio/test/generate-unit-tests-for-your-code-with-intellitest?view=vs-2019

unit test pattern:
https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019

Data Warehouse Testing 101 | Panoply (no date). Available at: https://panoply.io/data-warehouse-guide/data-warehouse-testing-101/

API protection in Azure API Management by using OAuth 2.0 authorisation with Azure AD.
https://docs.microsoft.com/en-us/azure/api-management/api-management-howto-protect-backend-with-aad

(Kimball et al., 2008)
https://books.google.dk/books?hl=en&lr=&id=XaUV6r2Xy0IC&oi=fnd&pg=PP1&dq=Kimball+et+al.,+2008&ots=H1o9NWVtxE&sig=jiET9MNovhBVu2hDvplGTsK4Sfo&redir_esc=y#v=onepage&q=Kimball%20et%20al.%2C%202008&f=false

Production Monitoring resources. https://www.haldanmes.com/detail/i/practical-example-of-oee-calculation

https://www.machinemetrics.com/blog/how-to-monitor-oee-to-maximize-your-productivity

Entity framework

https://www.entityframeworktutorial.net/what-is-entityframework.aspx

sold principles

https://itnext.io/solid-principles-explanation-and-examples-715b975dcad4

10 Appendices (Balkis)

Appendix 1 – Persona

Appendix 2 – Process Report

Appendix 3 – Use Case diagram and use case descriptions

Appendix 4 – Activity diagrams

Appendix 5 – Group contract

11 Appendix A GitHub repository

Link to GitHub repository:

https://github.com/Balqies/BPR