Ticket and worktime management system on Microsoft PowerApps and Common Data Service

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis
Bachelor's Degree Programme in Business Information Technology
Häme University of Applied Science
2021
Bakr Al-Qassab

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Bachelor's Degree Programme in Business Information Technology
Häme University of Applied Science

Author Bakr Alqassab **Year** 2021

Title Ticket and worktime management system on Microsoft PowerApps and Common Data Service

Supervisor(s) Deepak Kc

TIIVISTELMÄ

Tämän opinnäytetyön tavoitteena oli kehittää digitaalinen ratkaisu tiedon keräämiseksi mobiililaitteilla tehokkaassa työympäristössä. Ratkaisu on integroitu Azure DevOps- ja Common Data Service -ympäristöihin. Järjestelmää käytettiin talteenoton aloittamiseen ja poikkeamien välttämiseen. Järjestelmä oli käyttökelpoinen sekä mobiililaitteissa että työpöytäympäristöissä. Vaaditut toiminnot sisälsivät tietojen keräämisen ja tietojen käsittelyn. Tiedot koostuivat useista sarakkeista, kuten aloitusaika, lopetusaika, kesto ja tietojen haltija. Tiedot tallennettiin tallennettiin komissaarin yhteiseen tietopalvelupilveen lisätarkastusta varten.

Projektin tulos oli työn hallinnan kannalta tehokas ja tarkempi projektiorganisaatiossa. Hanke auttoi työtehtävien jakamisessa työntekijöiden kesken. Projekti toteutettiin A-A White Blue OY Companylle.

Järjestelmää testattiin lyhyessä ajassa tyypillisessä ympäristössä. Järjestelmän kykyä käsitellä paineistettua työtä ei testattu. Järjestelmän tuleva kehitystyö voisi sisältää kattavaa testausta ja jatkokehitystä.

Kehitys voisi koostua järjestelmän laajentamisesta laajentamisesta laajentamisesta mahdollistamalla yhteyden muodostaminen Jira järjestelmään ja Azure DevOps -järjestelmään.

Avainsanat PowerApps, AzureDevOps, CDS, PowerAutomate, Plugins.

Sivut 54 sivua, joista liitteitä 5 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's Degree Programme in Business Information Technology
Häme University of Applied Science

Author	Bakr Alqassab	**Year** 2021
Subject	Ticket and worktime management on Microsoft and common data service	
Supervisor(s)	Deepak Kc	

ABSTRACT

The objective of the thesis was to analyze the data and digitalize it to the user. The data will be gathering from the Employees' mobile devices.
The Data Center is integrated into Azure DevOps and Common Data Service environments. The system was used to start reclamations and avoid mistakes. The system was usable in both mobile devices and Desktop environments. The required functionalities included data gathering and handling the data. The data consisted of several columns such as start time, end time, duration and the holder of the data. The data was stored in the commissioner's Common data service cloud for further inspection.

The outcome of the project enabled efficient work management and more accuracy in the project organization. The project helped in distributing the work among the employees. The project was implemented for A-A White Blue consulting OY Company.

The system was tested under a typical environment for a short period. The system's ability to handle pressured work was not tested. Future work can be comprehensive testing and further developments. The development may consist of expanding the system extension with the possibility to connect with the Jira system and the Azure DevOps system.

Keywords Write PowerApps, AzureDevOps, CDS, PowerAutomate, Plugins.

Pages 52 pages, including appendices 5 pages

CONTENTS

Appendices

Appendix 1 Project Plan Visio
Appendix 2 Table of the User Interface Functions of the canvas app
Appendix 3 Table of canvas app user interface function implementation in backend
Appendix 4 Helper class to handle User inserted data

Terms and Abbreviations

UI	User Interface.
CRM	Customer Relations Management, or Microsoft CRM ERP System.
IDE	Integrated development environment.
Back-end	The functionality done dynamically in the some where else than what visible to the user. Includes things like code, databases.
Front-end	The part of the software that the user sees. Includes things like buttons, input fields, and images.
UML	Unified Modelling Language.
UID	Unique Identifier.
GUID	Globally unique identifier.
IDEs	integrated development environments.
LOOKUP	A variable refers to the common data between two or more CDS entities that can be constructed as a string or integer or Boolean, used when data in one Entity associates to data in another entity.
BPM	Business Process Management.
Boolean	A variable type can either be true or false.
String	A form of text data, consists of multiple characters, usually used as a variable type.
TFVC	Team Foundation Version Control.
SDK	Is a software Development Kit, intended to develop a specific system or program.
Int	Short for integer,represent the whole numbers.

1 INTRODUCTION

The commissioned company needed new systems to transfer from traditional systems of calculating work effort distribution and work tickets like initializing information manually in Azure DevOps, Excel, Paper.

The upsurge of the IT-industry causes companies to transform their strategies in managing their business process and counting work time (McKinsey Company, 2020). The importance of the presence of the mobile in our daily life spunks the company to show interest in involving mobiles in the work environments and using the advantages the mobile can provide like applications.

The System helps in saving the company time with increased accuracy of duties distribution among the employees. The system will help the internal and external temporary employees manage their work smoothly.

This thesis was commissioned by A-A White Blue Consulting Oy. They are a Microsoft partner software development company that creates software projects. For consumers and offers consulting services for businesses. The company is based in Riihimäki. Finland. Founded in 1998 (Commissioned company, 2018).

The company concurred that Microsoft PowerApps and CDS and the other related project's used techniques covering are sufficient for the scopes for this thesis.

2 PROJECT BACKGROUND

The commissioner company handles a lot of data and preparation. Information is stored across various CDS entities. The entity is comparable to a database table that holds data inside. The various attributes in the entities correspond to the table's columns work proceeding. Building upon the required data background each attribute represents certain types of data and helps to determine the information direction inside that entity (Microsoft dynamics entity, 2018).

At the end of 2019, the company started shifting to a new cloud storage system. Common data service (CDS) is the new storage system for the company. The data from Microsoft dynamics 365 apps is also stored in Common Data Service. This feature gives the ability to build projects on different platforms. Users can extend the Power Apps apps by leverage on Microsoft Dynamics 365 data (Common Data Service Definition, 2019).

Common Data service was chosen since it fulfilled company requirements like providing the integration with Azure DevOps. The company prefers Common Data Service because it can be accessed from PowerApps and Microsoft CRM with the ability to use plugins.

The thesis idea introduced when the company started to grow and accordingly needed a flexible system to help in managing the company's projects. The system is required for creating easy access by both the internal and external temporary employees. For ability to manage the project's tickets and initialize the work time automatically or manually. The outcome will show the utilized time for each project separately that the logged-in employees part of. Then billing the customer according to the utilized time and the agreed-upon hourly rate.

3 RESEARCH METHODS

The project is merged between many techniques to fit the business requirements. The thesis will help in extending the knowledge of the PowerApps and CDS and other related techniques fields. During the thesis work used many reliable resources from Microsoft and suggested resources from experts. Microsoft continuously publishes new features to fit the user's requirements (planned features, 2020). Following the latest updates and released features help in carrying out the best performance to make the project work effectively.

One of the Power Apps supported platforms is the Power Apps chat community. The community is helping to solve problems where PowerApps users face. Community helps developers share their experience in the Power Apps system with others. Mostly there is Microsoft employee's participation in helping to solve the common problem (PowerApps Community, 2020).

To extend the level of knowledge and go more deeply into PowerApps need to follow Udemy courses and try to cover most of the features they provide. Courses help Start to learn how to use Microsoft PowerApps and coding in PowerApps plugins in C# coding language. Microsoft CRM system and Microsoft PowerApps have a common way to use code and some common background work process in some aspects.

4 TOOLS

To start implementing the project, it needs to set up the developing environment and CDS that is being used in the back-end storage.

In some development steps, it needs to use Microsoft Power Automate service. Using power Automate will help PowerApps to become a more flexible system. Power Automate will work as a binding tool with other services like Azure DevOps (PowerApps Definition, 2019).

4.1 Microsoft PowerApps

Power Apps is a high-potency enlargement platform for business apps. The system created by Microsoft. The first version was released in April 2016 with basic functions and low User Interface options. It is considered as AppService's connectors and data platforms. It helps to build mobile applications for various types of user needs. It is possible to work with online or on-premises data sources like Share point, ExcelDynamic365, SQL and MySQL. (PowerApps Definition, 2019).

In August 2016 Microsoft released the SharePoint user-friendly version of the SharePoint Online Modern lists. The users can easily create the app from the command bar of the SharePoint online page. The app will show a customizable ready template. Then PowerApps begin to support connecting the on-premises SharePoint list.

CDS becomes a global production and users have increased continuously over the years. The increase of the CDS population makes it the main data storage across PowerApps, Microsoft Flow, and pro-development tools. (Desai, 2016.)

In October 2018, there was a major improvement across the themes in PowerApps. The released version allowed apps makers to create higher quality apps. The update makes the existing contents and concepts more useable and fast to use (Released features from Microsoft PowerApps, 2019).

4.1.1- PowerApps Components

PowerApps components Consisting of three types of platforms namely Canvas app, Model-driven app and Portal app.

Canvas apps: Canvas app can build as Mobile or tablet applications. Canvas app provides ready templates that can evolve with various 200 data sources.

Model-Driven apps: Model-driven apps automatically generate practical useable UI which is responsive across devices. Model-Driven apps can build views, model forms, and other data components from core business data and processes in CDS.

Portal: Portal is support creating external websites that allow the organization's outside users to sign in with a wide variety of identities. It allows creating and viewing data in Common Data Service, or even browse content anonymously. (PowerApps maker, 2020.)

Power Apps uses Microsoft's formula system to perform owned logical tasks. Developers can handle a formula system with insufficient previous involvement in coding. The Canvas app's UI configuration and most of the back-end works are done by the formula's system.

There is a common entering Formula process between Microsoft Excel and Microsoft Power Apps. The Microsoft Excel uses the same formula performance concept. The diversity is the formula system of Power Apps is not just for calculating values and execute other tasks but also responds to the user's input and the app desires (PowerApps formula system, 2020).

A formula is adopted to determine the app's responding action when a user clicks a button, adjust the slider, or utilizes some text (PowerApps formula system, 2020). PowerApps use a Collection function. The collection is an array used to store data in PowerApps internal data storage and load it when they demand it (PowerApps collections, 2018).

In Power Apps You build formulas that apply precisely to applications than of spreadsheets.. The formula itself is attached to PowerApps UI component properties. To more understand the developing platforms (Figure 1) demonstrates the formula structure and how can the user control the inputs to fit the app requirements (PowerApps formula system, 2020).

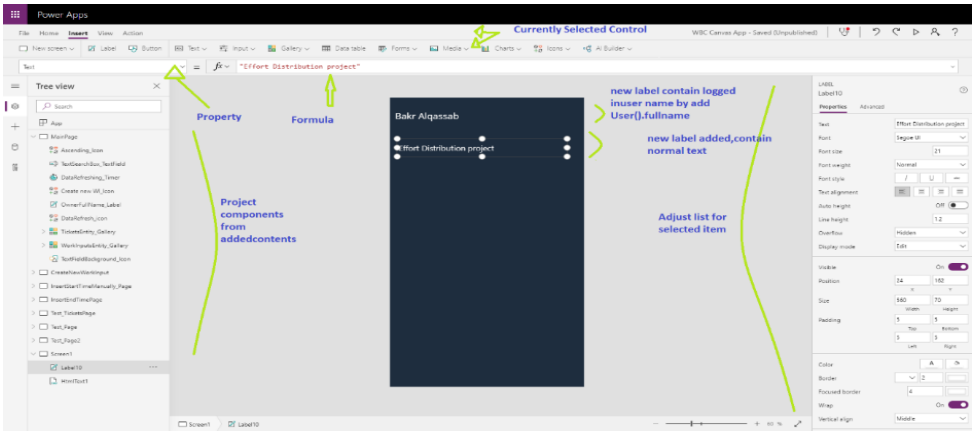

Figure1. The structure of the PowerApps formula system and control list.

4.2 Microsoft Power Automate (previously Flow)

Power Automate is a cloud system that works as a binding tool between services and as a business process management platform. According to the BPM the markets and businesses are growing forward very fast (BPM (Business Process Management), 2020). Business environments need to enhance constantly and need more automating environments between internal and outside systems (Microsoft Power Automate, 2020).

Power Automate aims is to make the user's life easier by automating many techniques. It simplifies the business process and makes it faster and more effective. Basics of business rules and work processes are enough to work PowerAutomate. To fill missing actions need to use the Power Automate and Plugins.

Power Automate allows you to change user focus to different trends where you can get more good results like focusing on the functionality of the system more than focusing on the binding.

Power Automate management grants ready-made templates. The system integrates easily with various types of systems that want to link. During project implementation, Power Automate use as a binding tool between Power Apps and Azure DevOps on one side and Power Apps and CDS on the other side. It helps locate the function that users want to reach automatically. (Microsoft Power Automate, 2020.)

Microsoft Power Automate provides three types of functions, Automated Flow, Scheduled Flow and Button Flow. The Automated Flow is triggered by an update. The Scheduled Flow is effective at a determined time. Button Flow is a flow triggers effect by pressing the Button(Microsoft Power Automate, 2020).

4.3 Common Data Service

CDS is cloud-based storage. CDS stores data in entities. Every entity contains records. Entity work like tables in databases that contain records with various cells types to store the data. CDS provides the ability to create custom entities depending on Business data requirements that are populated with Power Query. Users can prepare the app easily according to the existing data in entities. Many features make the CDS is taken into consideration, managing and accessing data is easy, Data is securely stored.

CDS provides powerful data access data and flexible import and export data options. The CDS offers add-ins that allow accessing the CDS data

easily by Microsoft Excel which makes it easy to show and modify data inside Excel files and publish it in CDS entities(Desai, 2016).

Microsoft Dynamic 365 applications are using CDS. using CDS helps to extend PowerApps and make it cooperate with Microsoft Dynamics 365 because of the common configuration in the data storage aspect as in Figure 2. (Common Data Service Definition, 2019.)

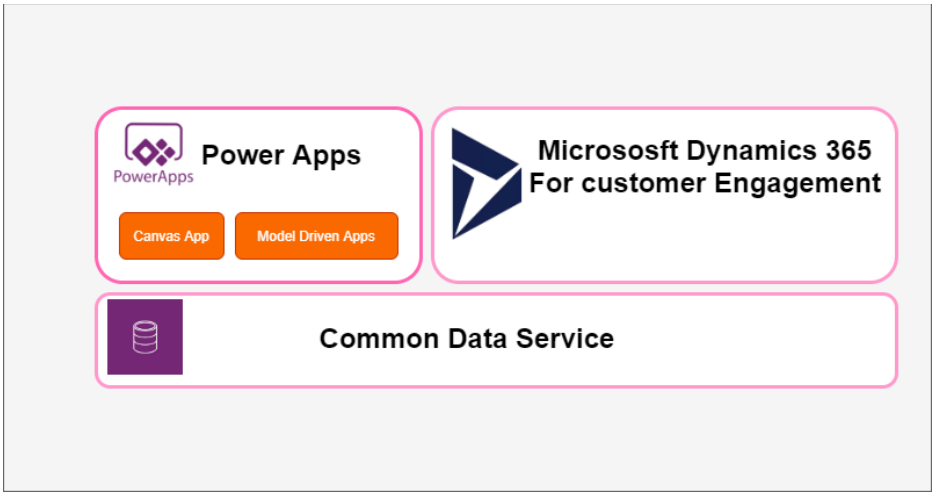

Figure2. Power Apps and Microsoft dynamics 365 storage structure.

4.4 Visual Studio 2019

Visual Studio is a computer application released by Microsoft as an Integrated Development Environment. It is used for various development platforms like computer programs, web apps, websites, mobile apps, and web services. Since 1997, Microsoft has constantly expanded Visual Studio features to be more professional. Microsoft is trying to meet the user's needs and pursue the fast-growing technology. Visual Studio uses Microsoft software development platforms such as API and Windows Forms. These platforms help to utilize either native code or manage code. Native code such as native COM interfaces or the Babel Framework (part of Visual Studio SDK) (Visual Studio 2019 Definition, 2020). Managed Code is the code compiling by a specific time (manage code definition, 2016).

Visual studio is the more practical IDE for coding, debugging and adding extensions to the project. The visual studio offers an easy connection with GitHub Repository and Azure DevOps Repository to upload the project to the cloud (Visual Studio 2019 Definition, 2020).

The Community version of the Visual Studio is open-source with fully IDE features for students and individual developers. There is a various version

of the visual studios like 3.0, 4.0 and the 5.0 support the features of the C# language. Visual Studio has different methods to implement apps by C# such as class designer, Form designer including Data Designer (Visual Studio 2019 definition, 2020).

4.5 **Azure DevOps**

Azure DevOps offers services that support Teamwork for code development. Until 2018 it was formerly named as Visual Studio Team Foundation Server (TFS). It offers collaboration on software development through source control. Azure DevOps offers continuous integration which is a very useful feature in software development.

Azure DevOps integrates features that can be reached easily from the browser or IDE. Azure DevOps includes different features to support teamwork. These features are :

Azure Repos: Azure Repos works the same way as GitHub work but in a more flexible way by participating with other features at the same time and provide TFVC for source control.

Azure Pipelines: Azure pipeline supports continuous integration by getting support from the build and release services.

Azure Boards: Azure Boards is a suite of Agile tools that help to implement scrum processes and work managing and effort distribution.

Azure Test Plans: Azure test feature allows users to test the apps with several tools in the manual or regular testing styles.

Azure Artifacts: is responsible for distributing NPM and NuGet packages from both private and public sources and integrate these packages by sharing them in CI (continuous Integration) / CD (continuous Delivery) Pipeline. (Azure DevOps definition, 2019.

4.6 **Plugin Registration Tool**

The tool is part of the CRM SDK developed by Microsoft, which also includes other documentation, sample code, and tools related to CRM development(Plugin Registration Tool, 2020). The Plugin Registration tool allows you to register and configure plugins for Dynamics 365. The plugin Registration tool is a graphical tool as shown in Figure 3 for Dynamics 365. Plugin Registration tool is a graphical tool as shown in (Figure 3) (CRM, 2016).

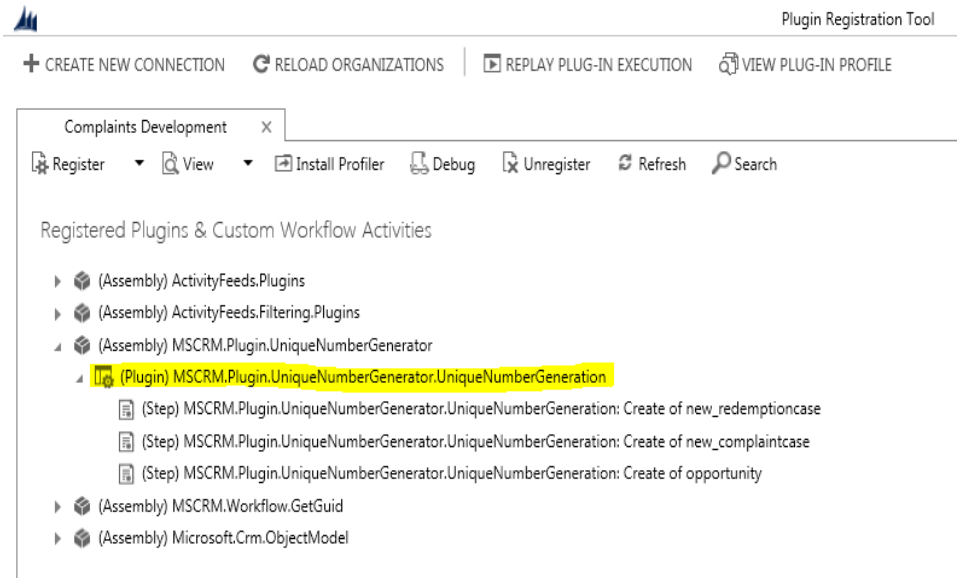

Figure3. Plugin Registration tool UI(CRM, 2016).

At the top of the main view, there are buttons to create a new connection and reload the ongoing connection. At the top, you will also find the “Replay plug-in execution ” and “View plug-in profile ” buttons, which are tools for debugging plugins. Plugin debugging is not essential for the indicated job. The main view also classifies the assemblies registered in the organization. The + button on the left of the assembly opens a list showing the plugins in the Assembly. Furthermore, the list can be expanded if images have been registered for in the plugin.

4.7 **Plugins**

Plugins are a business logic that helps to integrate with Microsoft Dynamics CRM. Plugins help in modifying or changing the system default behavior. It can be defined as the handler of the events of Microsoft Dynamics 365. Plugins execute on a particular event on the server and cannot implement anything that would require an interface. Plugins are written in either C# or VB. Project’s Plugins developed by the default language to use in plugins C#. Plugins can run either in synchronous or asynchronous mode(Plugins Definition, 2020).

Some scenarios of Plugin's handling is, updating a particular field when the specific events happened, calculate data dynamically or send an email to the customer when some adjustments happened in particular positions(Plugins Definition, 2020).

Plugins are very simple classes. To implements Plugin needs to use Pluginbase.cs and extend to the Iplugin file which works on fetching the filled-in data on the interface. The Pluginbase.cs has the same

configuration as the SDK Iplugin file but with more expanded features to reach the highest benefit from the Microsoft SDK components. The (Figure 6) display a simple plugin that does not do any things but have class inherit to the " Pluginbase.cs" file and SPKL code to detect the stage's category that will execute when running the Plugin. execute Plugins with SPKL is more simple than registration the steps manually in the Plugin registration tool.

```
4   using System.Linq;
5   using System.ServiceModel;
6   using System.Text;
7   using System.Threading.Tasks;
8   using System.Globalization;
9   using Microsoft.Xrm.Sdk.Query;
10  using Plugins.helpers;
11
12  namespace WbcErp.Plugins
13  {
14      [CrmPluginRegistration(MessageNameEnum.Create,
15          "wbc_workinput",
16          StageEnum.PreOperation,
17          ExecutionModeEnum.Synchronous, "",
18          "wbc_workinput PreCreate",
19          1,
20          IsolationModeEnum.Sandbox)]
21
22      public class Wbc_WorkInput_PreCreate : PluginBase
23      {
24          public Wbc_WorkInput_PreCreate() : base(typeof(Wbc_WorkInput_PreCreate))
25          {
26
27          }
```

Figure 6. Simple Plugins and SPKL step configuration.

Plugins have diversified phases to implement as drawn in (Figure 7). The stages names are:

Pre-Operation this pipeline phase will be implemented before the main system operation. This Plug-ins stage executes outside the database transaction.

Pre-Validation this pipeline is implemented in the database transaction before the main system operation.

Main-Operation this phase is used for internal use only. Here the main operation of the system is used in functions Like, create, Update, Delete. here no custom Plugins can be registered in this state.

Post-Operation This Stage of plug-ins which are to execute after the main operation. This Plug-ins registered on this phase can execute interior database transactions.

(Plugins Definition, 2020).

There is a similar in work process between CRM and Power Apps especially since both of them are Microsoft Systems and use Common Data Service. The plugins are executing inside the Common Data Service database. The features which apply to CRM Will apply to the Power Apps as well. This one of the components makes Power Apps more expanded and popular.

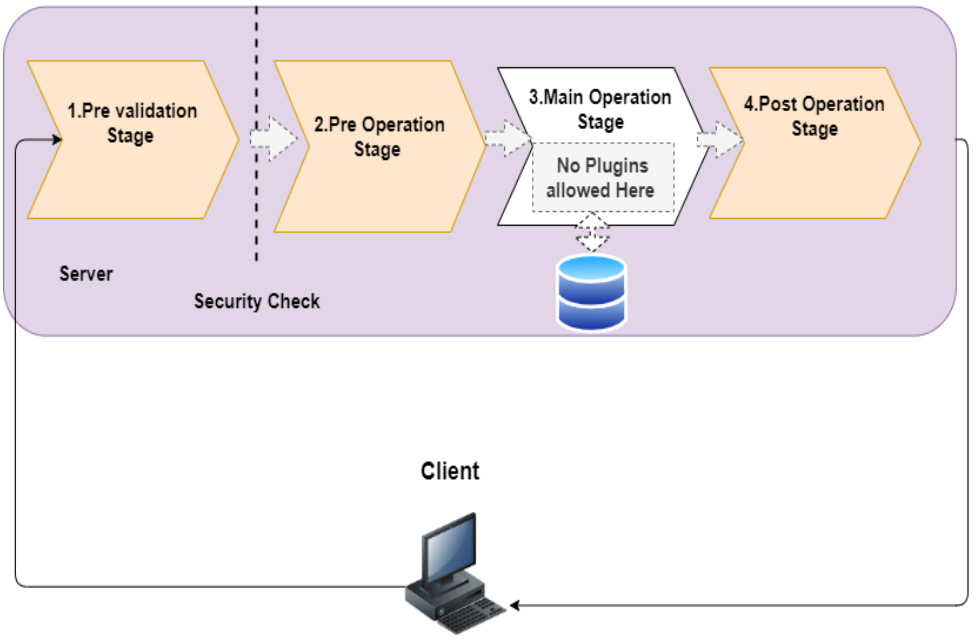

Figure 7. Plugin's stages process workflow.

4.8 **SPKL**

SPKL is a script line tool as demonstrated in (Figure 4). It is a Simple and Lightweight tool. Using SPKL eliminates the use of various processes that were done manually by the plugins registration tool.

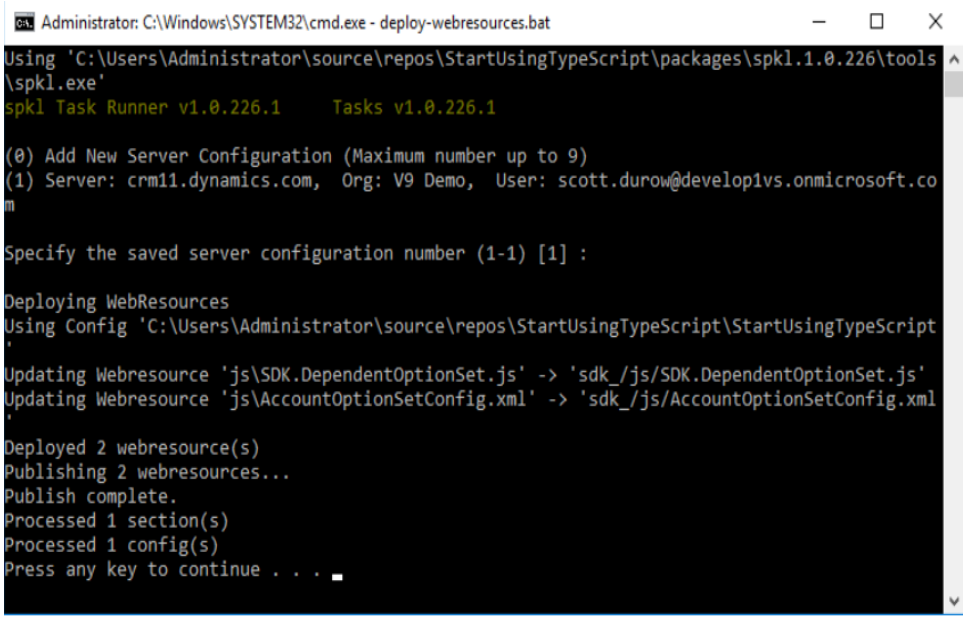

Figure4. SPKL common line configuration (SPKL, 2020).

The tool developed by “Scott Dorow”. It uses as a deployment task runner for Dynamics 365. SPKL will place all registration info on top of the class file and deploy plugins. It will deploy Plugins by executing the “deploy-plugins.bat” file which will handle all the processes. Using SPKL help in saving time and effort.

The SPKL installing can be done from the Nuget package manager. The installed version was 1.0.226”. After prepared the file needs to execute the downloaded file called “-deploy -webresource.bat”. Then log in by Microsoft account and will appear the existing projects in your server to choose the related project. (SPKL, 2020.)

To deploy Plugins, you need to utilize the plugin's name in the “spkl.json” file configuration. Then you can implement the “deploy-plugins.bat” file as shown in (Figure 5). To define the entities that will be generated whenever run the plugins need to use the “spkl.json” file to prescribe the plugins and solution location (SPKL GitHub, 2020).

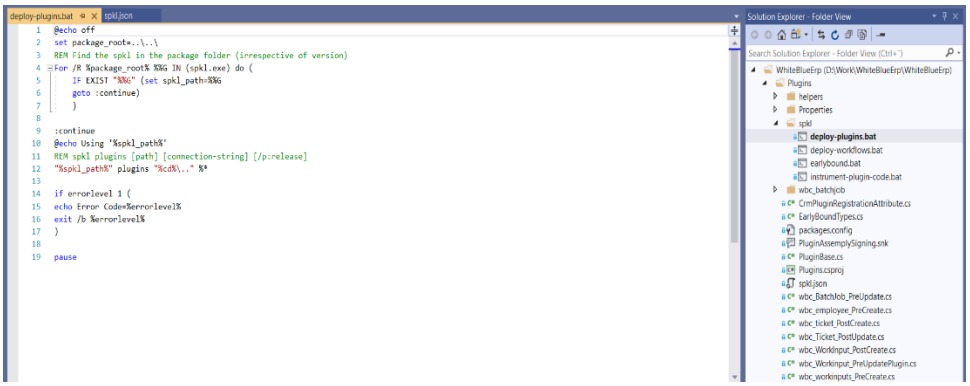

Figure5. Deploy Plugins file configuration.

When working with Organization service Assemblies there are two styles to work with. Late bound & Early bound. Late bound use Entity's class and need to refer to the columns and entities by using their logical name. Early bound programming needs to generate a set of classes based on the existing metadata for the determined organization using the generator tool.

Generation tool is a command-line code generation using with CDS. This tool helps to generate the Early bound .Net Framework classes style. These classes represent the used Entity's data by CDS. The code generation tool (CrmSvcUtil.exe) is classified as part of the Microsoft CrmSdk CoreTools NuGet package.

The major reason for using Early binding is to provide compile-time to checking all types. Late binding checks types particularly when an action is performed on the type or the object is created. There are no implicit casts in Early binding, unlike the Late bound biding. CDS entity class requires specific types to prevent implicitly. That one of the features why SPKL is getting the Early bound pattern. (Late & Early binding Definition, 2020.)

5 COMMON DATA SERVICE (CDS) PREPARATION

The company handled most of the creation of the storage data in CDS. The CDS will be used as a back-end for creating a mobile and computer application. User's operations will need a place to store. The inserted and retrieving data will be stored in their Entity. The entity is a CDS object that consists of data rows. A data row consists of multiple records where have a comparable process to how tables stored data in the database that contain varying types of data. The table's records Data can be int, Varchar, date values, or Lookup.

5.1 - Creating a Common data service database

The first step in preparing the storage data in CDS is to create a new Solution. CDS Solution is like a box that will consist of all the components that you work on. There are two types of CDS Solution default and customized Solution. The difference between them is the customized solution grants more features than the default. Like the user can export the Solution easily to apply the solution publisher name prefix. The publisher's name prefix will be a merge at the beginning of each record's name. Unlike the default solution, the user is not authorized to export the solution to another Environment or create a unique publisher name (CDS Solution, 2020).

To constitute the Solution needs to start selecting the “New Solution” button. Next will display Create a New Solution Window as shown in (Figure 8).

Then start filling the required data. The data window will require to insert the “DisplayName” value. The “Name” field will be generated based on the DisplayName field content. In the “publisher” field you can choose the created publisher and insert a unique term for your data. In the “Version” field I inserted 1.0.0.0 and then selected “Save”.

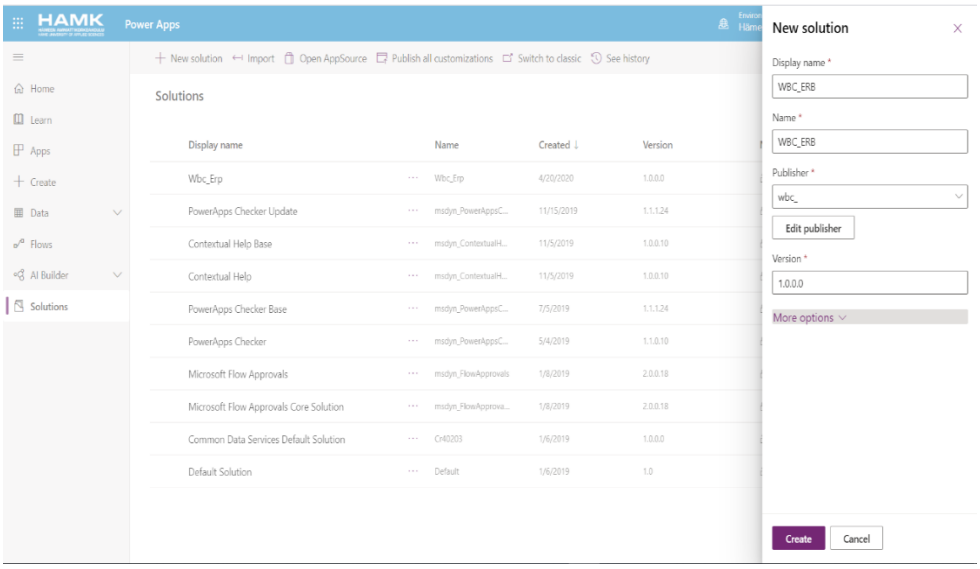

Figure 8. Create a new CDS Solution interface.

5.2 Creating data Entity

The following steps are for creating data entities. The CDS entity will consist of project data. The project's data will be distributed to three entities.

Users, Work Tickets and Work Inputs entities. Creating a new entity start by select “New” then chooses “Entity” from the drop-down list. Then will open create new Entity Window s displayed in (Figure 9). The name field will represent the schema name the first sector of the name is the inserted publisher's name.

The primary field represents the records in Entity. Here the Entity will automatically generate Many of the records.

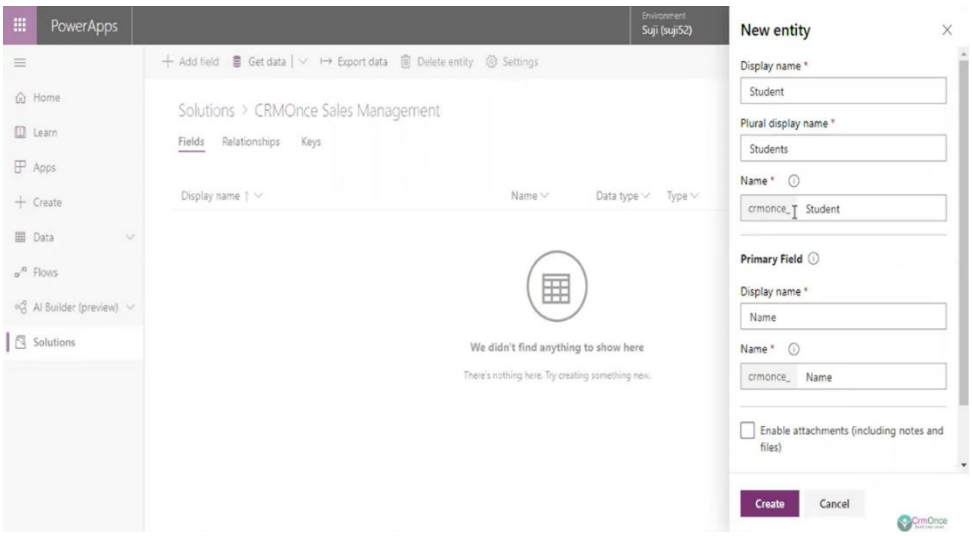

Figure 9. Create a new Entity window.

5.3 Creating data records

After preparing the default entity's form will need to constitute new data records according to the business requirements. For adding new records need to select “+ Add Field”. Then will open create a new record window as marked in (Figure 10). The display name will represent the name on schema and the “Name” will generate automatically by corresponding to the Display name with Publisher name in the beginning. To use and insert the value of this record by calling it by this name. ” The data type” will allow choosing the type of data the record will operate. When you click on the arrow will release the data's types drop-down list like string, int, date, Lookup, image.

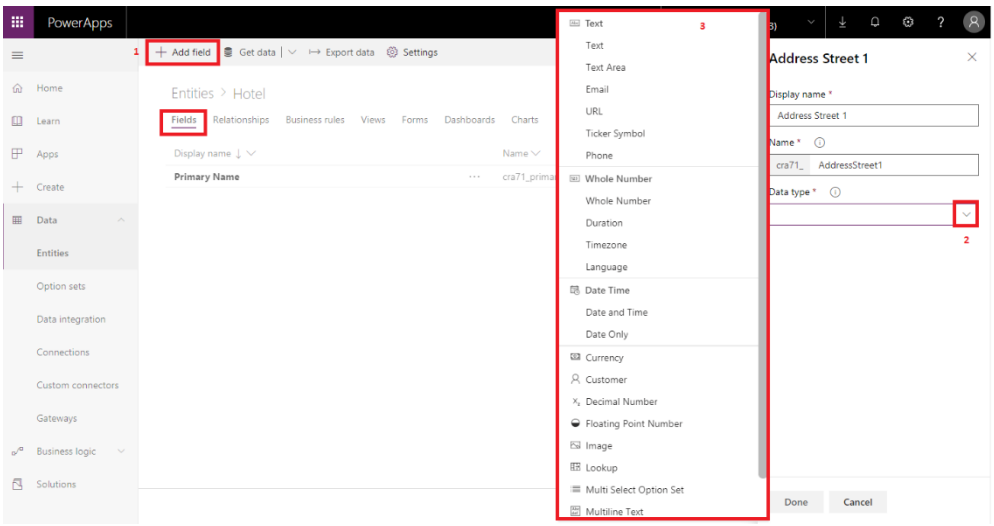

Figure 10. Create new records Window for the selected Entity.

5.4 CDS Data Type

The next step will need to attach between the entities' data. To link data need to use Lookup. Lookup can assign as a unique int or varchar record. When the value is unique then each record in the Entity table has owned value. That will make a smaller chance of duplicate values.

Ticket data will directly be inserted by the employer in Azure DevOps. Data will be stored in CDS. The data will be inserted from Azure DevOps to CDS by fired Microsoft PowerAutomate automatically. The employer who filled the data in Azure DevOps has his/her information already stored in the User's Entity. The UID of the Employer will be inserted automatically when creating a new Ticket. Each created ticket can consist of many Work-Inputs

These work-inputs will be inserted by the allocated Employers for this ticket as drawn on workflow in Appendix 1.

6 MOBILE APPLICATION

The next step after configured the system's back-end needs to starting to implement the front-end. From a technical point, the mobile app seeks an automatic and manual way to manage work tickets. The application needs to be user friendly. Every function in the app has a different duty to do as explained in (Appendix 2) where covered all app's functions. The application needs to be flexible in response to user missteps. Employee mistakes can be missed clicked the Start time button at the appropriate time and need to cancel it or forget to insert the start time when started working on the specific Ticket or forget to click the end time button when he/she conclude work on the selected ticket. This confuses the manual manner can solve it easily. The CDS data are accessible from the Canvas App & Model-Driven App where information will be gathering directly or with helper methods.

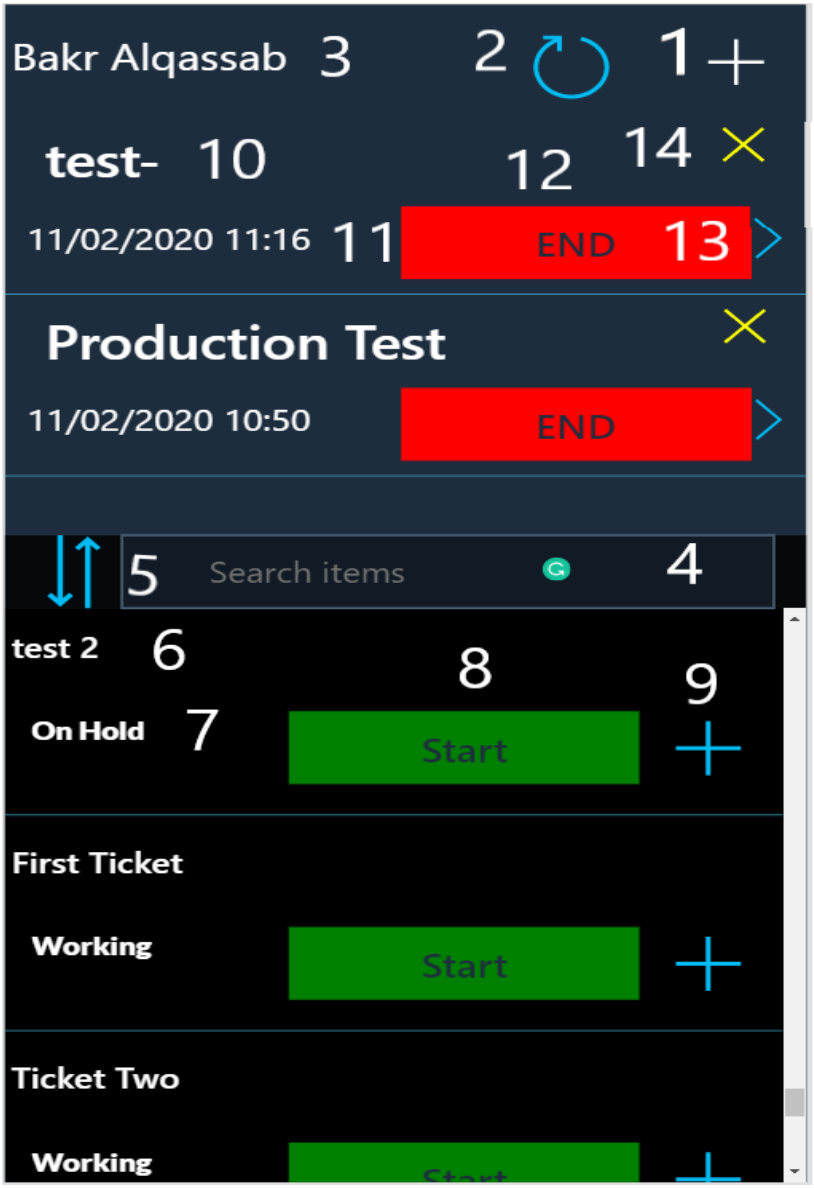

Figure 11. The eventual UI of the application.

Every function from the application is tested by me and other workers. The tested purpose is to make sure the intended data is inserted and retrieved properly from the CDS, To consider the app as effective and reliable to use from the Company.

The application constituting start from the Microsoft Power Apps environment from inside the Solution.

The generated blank screen will include the default aspects ratio of various mobile devices as displayed in figure 12.

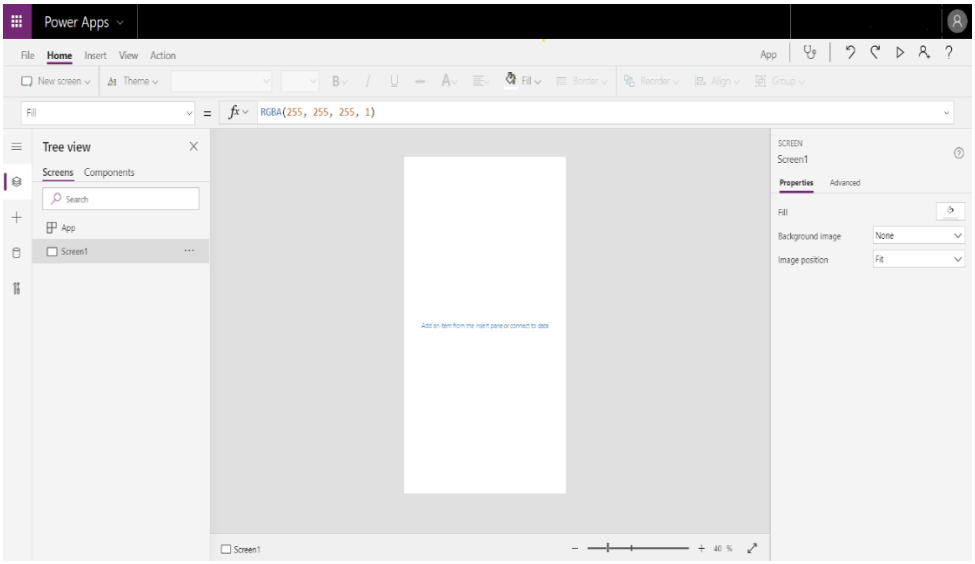

Figure 12. Create a mobile application platform.

The blank field shows what the user will interact with when using the application.

The left area represents the UI configuration elements that will build the app. The up and right sectors as previously mentioned it is the formula area. The formula area is to build the app's operations and elements arrangements.

6.1 **Project's data navigation**

The first step of the app's production is to attach to data storage. Then add the related Entities and other connectors that will be used in the project. These components can be added from the left sector by selecting the " Data Sources" section then can access the added actions from the " In Your app" field.

The project will contain various entities and connectors like " Tickets", "Work Inputs" and "Users" entities. The leading entities' data can be controlled by utilizing a formula system. The Formula System can be modifying the UI and combine the app's fields to the back-end storage as covered in (Appendix3) all the app's filled in orders in the Formula system.

Power Apps provide various styles to display the data at UI. The styles can be ready for Gallery templates or ready empty Gallery frames. In the project's Mobile application main page UI configuration used the ready empty Gallery frame. The frame allows developers to add the data from the online Data Source. The finished gallery's UI shows the added Entity's data elements separately in one row. The gallery's layouts will be editable and duplicate according to the number of rows in the Entity.

Power Apps authorizes filtering Entities and collection of data. The Refining displayed data will help to show the tickets separately according to the logged-in user.

In the system, each employee can be associated with more than one project at the same time. Every company is a set of one or abounding employees working on a specific project. . The displayed information on mobile UI will be filtered according to the logged-in user's full name and the searched text in the search field. The formula system order will be utilized as shown in (Figure 13).

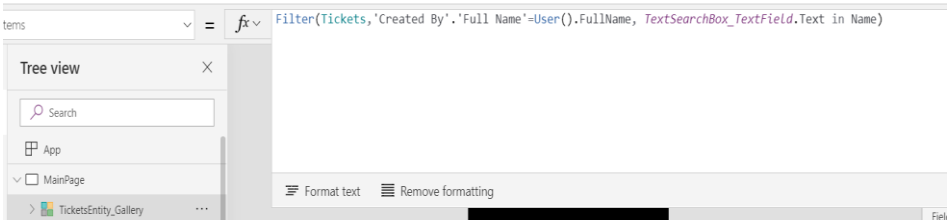

Figure 13. The Tickets Entity displayed a data filtering formula.

The Work Inputs gallery is displayed only the unfinished work-inputs where the end-time or the duration is not entered. It is achievable for the employee to add the duration manually without the start and end times. Then the back-end fired plugin operation will collect them automatically. Additionally, the displayed work-inputs will be related to the logged-in user's full name and the work input's status is active. The entered formula order will be as shown in (Figure 14).

Figure 14. the Work Inputs filtering formula.

The data in Entities is not static because various types of employees work on the same projects with the same back-end storage so it can be changeable constantly. To be assured the employee will have the last updated data will be a refresh button. Data will update manually by refresh button for click it when need to check if there is an update in the Entity's data. The reload icon is added from the icon section where Power Apps offer several shapes of icons to utilize in the app's UI pleasantly as displayed in (Figure 15).

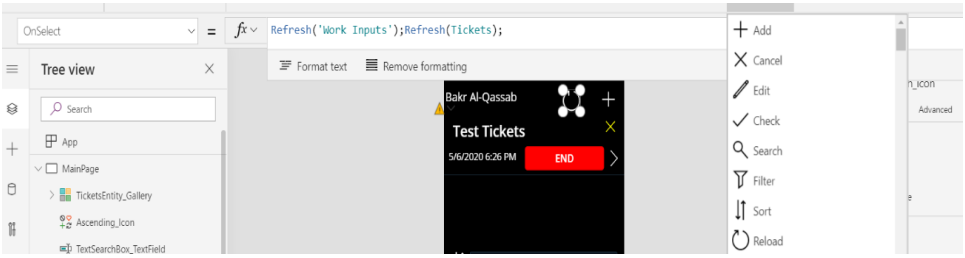

Figure 15. Data reload icon and used formula.

In a certain condition, the data will update automatically like when the employee creates new work Input. There is a timer formulate to update the displayed data after 2.5 seconds. PowerApps offer a timer from the "Input section" then select Timer and will generate a new one. The timer can use to achieve a function when the user wants at a certain time.

I set the timer to work immediately from the “on Start” field when creating new work input and count 2.5 seconds. After the timer is finished then will update the retrieved data and set the timer to false to stop counting until it is true again when new Work Input will be created as in (Figure 16).

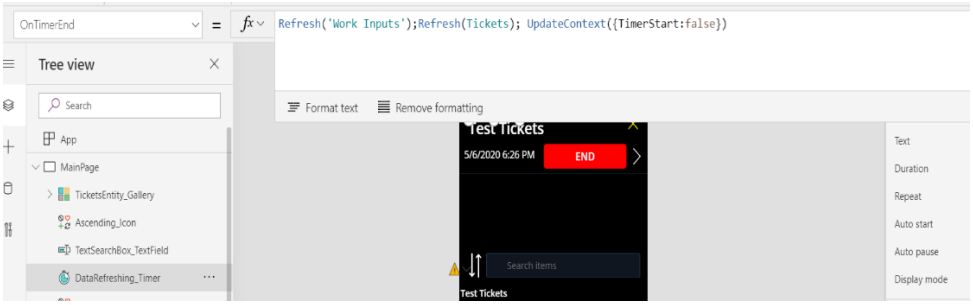

Figure 16. Achieved function timer's calculating is ended.

The manual manner of creating new Work-Input will be by a" + " icon in the main app's UI. The "+ " icon is to navigate the Employee to another page to select the ticket which wants work in. Using the icon can be useful in many instances like, insert the start time& end time || duration to ticket worked already on or just start before a while and forget to insert the started time. This "+ " icon is added from the Power Apps icons list. This icon is navigated to the Power Apps ready Form template as displayed in (Figure 17).

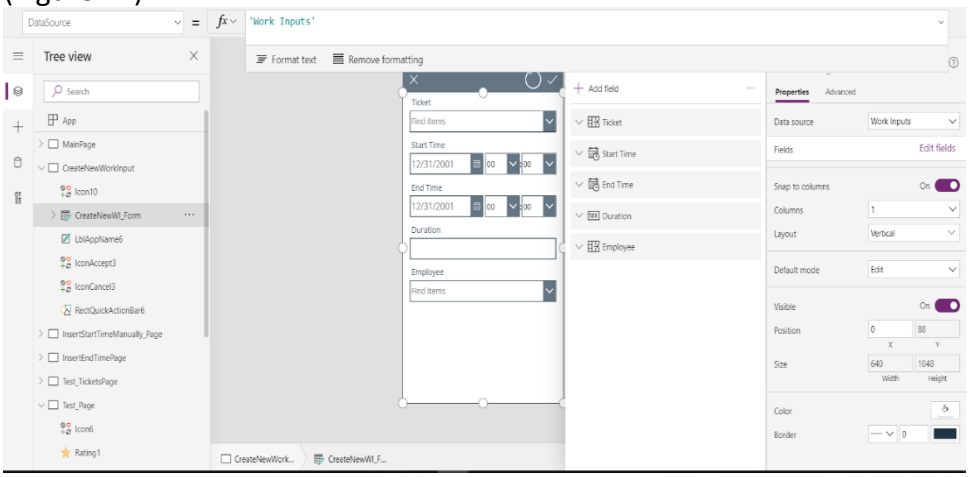

Figure 17. Power Apps Ready Form to insert new work input.

The new form is created to match business needs. The Navigated form is accepted to add the Ticket by the drop-down list and select the start time and end time. It possible to insert just the duration in case the Employee does not want to add the Start and End Time. The back-end Plugins will automatically count start time as the current time and the end time will be the current time subtract the minutes that the Employee adds in the duration field. In some cases, it can be that some external employees or employees work on a certain project temporarily with the company. To add them work time needs to use the manual method of creating new work input and add the employee name from the drop-down list. The Employee field can be empty, not mandatory and there will be considering the logged-in username as the generator of the new Work Input.

6.2 Displayed Data

The main page configuration depends on the desired data. In the left-up corner of the app's UI, as shown in (Figure 18), there is a label to show the current logged in user's full name. The benefit of putting the logged-in username is to strengthen the security of the app by preventing the mixing and be sure that the needed user runs the app.

Figure18. Displaying the logged-in user Full name label and formula.

The app's main page will be partitioned into two components. The Up part will include the in-progress work inputs. The below part will contain the logged-in user-related tickets.

The below part will be the ticket's Gallery components. The displayed information will be in each row contain the ticket name and status as shown in (Figure 19). The rows numbers will be depending on the amount of the tickets The status need to be " Working " status to be displayed. The ticket part will contain an "+ " icon in case the Employee wants to add the start time manually for the selected ticket.

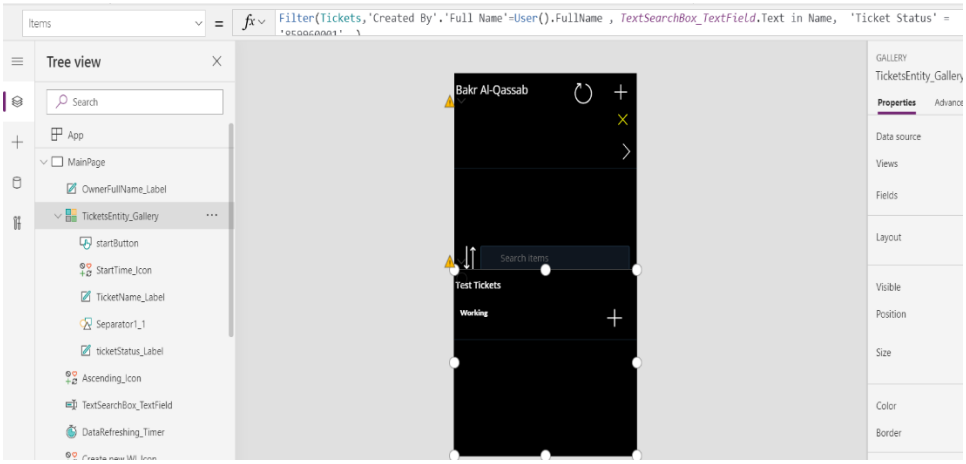

Figure 19. The gallery contains Ticket Entity data.

The navigate icon is intended to lead to another page where the user can add time manually. The add start time manually page configuration is Power Apps ready form. The ticket name and status will be shown for the clicked ticket the employee wants to work on. If the employee inserts only

the start time then will appear on the Work Inputs Gallery field as unfinished Work input and when the user completes then can end it. Employees can insert both start and end times at the same time and the app will calculate the time in the back-end. As previously mentioned it possible to select other employees for which this Work Input will be pointed out or can just be left empty and will automatically insert the logged-in user's name. The app's Ui will be displayed in (Figure 20).

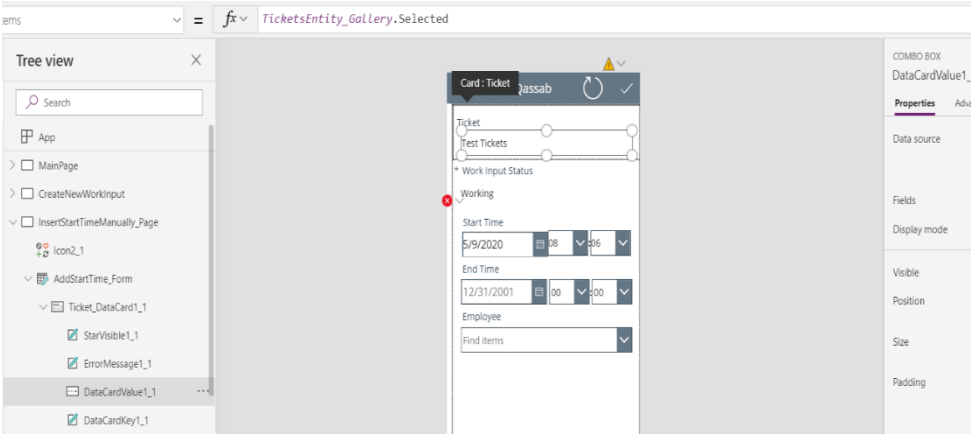

Figure 20. Insert star time manually form.

The up main UI page up section will be the in-progress work input's information. The gallery will contain rows, each row will contain the work input's name, started time, end time button, " X " cancel icon and navigate icon. Work input name to determine which ticket the user works on because the employee can handle more than one ticket at the same time. The label shows the started time for this ticket. The navigate icon will navigate to another form which allows inserting end time manually. The formula order text will be as in (Figure 21). When the user clicks the end time button it will insert the current time as end time and calculate the time in the back-end. The employee can cancel the created work input. To cancel the work input need to click the " X " icon and that will transfer the Work inputs status to "Cancel". After changing the work input's status will disappear from the available Work Inputs on the screen without any calculating for the ticket time.

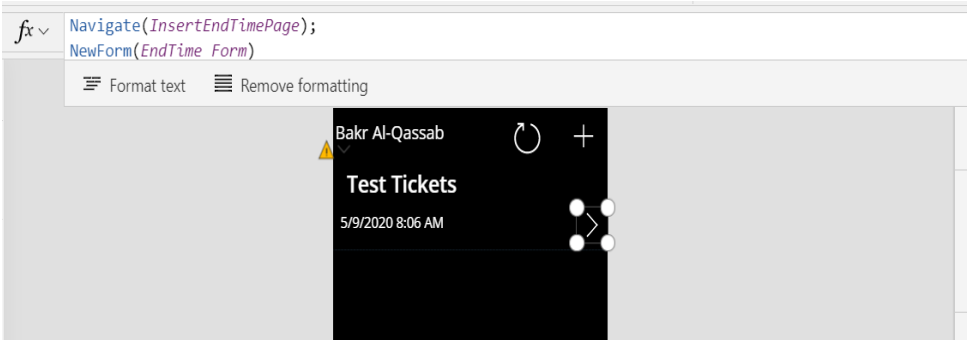

Figure 21. The displayed data of created Work Input.

The data Insert end time Form will allow adding only the end time manually. The other data will show as uneditable as show the configuration in (Figure 22).

The insert end time manually feature gives more flexibility to the app to be more practical to the employee. Usually with a long day of work and with various natures of work's tasks can be a little hard to remember to end the time on the specific tickets.

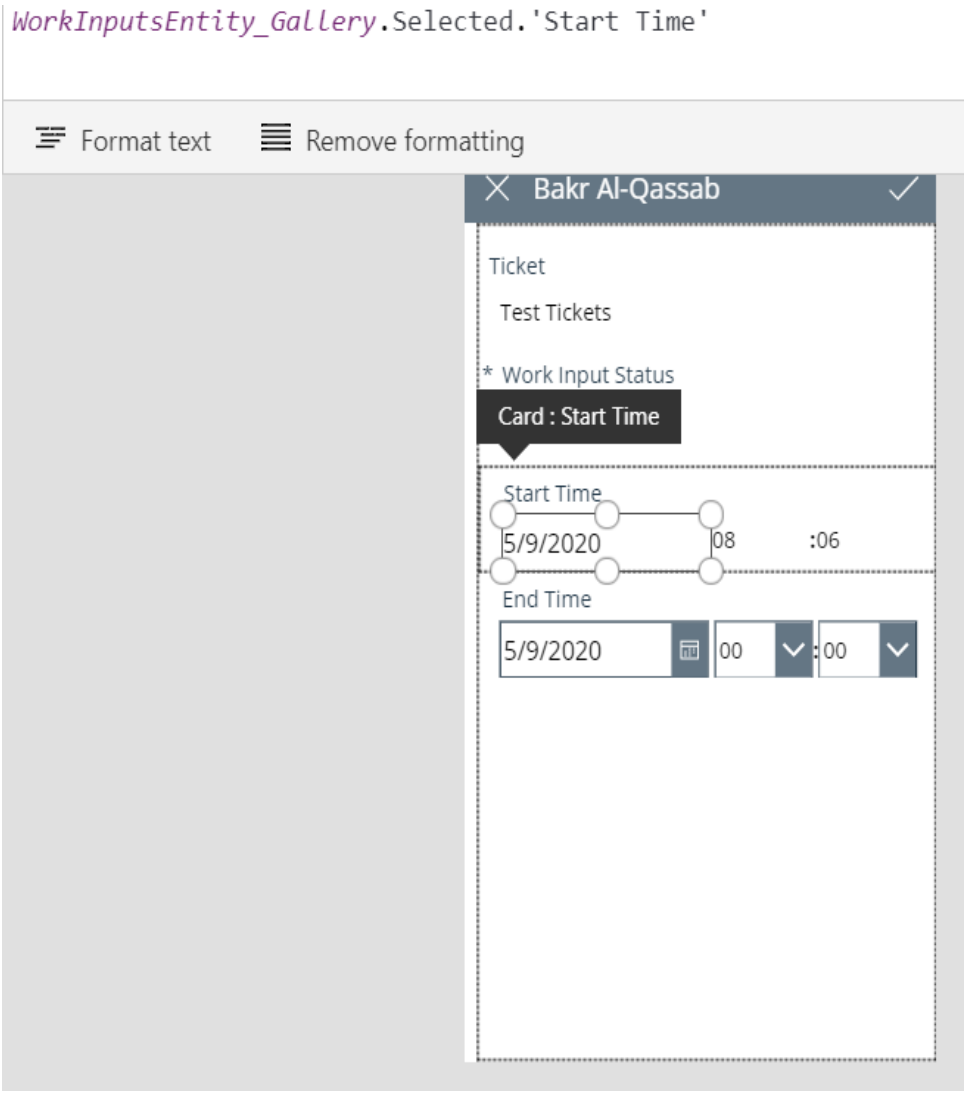

Figure 22. insert end time manually Form.

6.3 **Submit Data with workflow**

After finishing the practical part on the app's UI and formula's side. Then need to combine the data with relevant entity and records to store the data there. The main process in the app is adding the start time

automatically by adding a button. This button appears in each row of data retrieved from the Entity database. This button response is sending the selected ticket name and the current time and the logged-in employer's full name into the Work Input Entity. The process of adding the data to the Entity is done by Microsoft Power Automate (Microsoft Flow).

To join the button with Power Automate need to select the button and go to the “Action” area in the up bar and click on Power Automate and choose to create a new flow as shown in (Figure 23).

The power Automate is used To transport the data between galleries. The use of Power Automate is practical because the latest version of Power Apps not allowing to share two entities' data in the same gallery. The use of the Power Automate is the more efficient way to transfer the data from the Ticket’s gallery to the Work input entity.

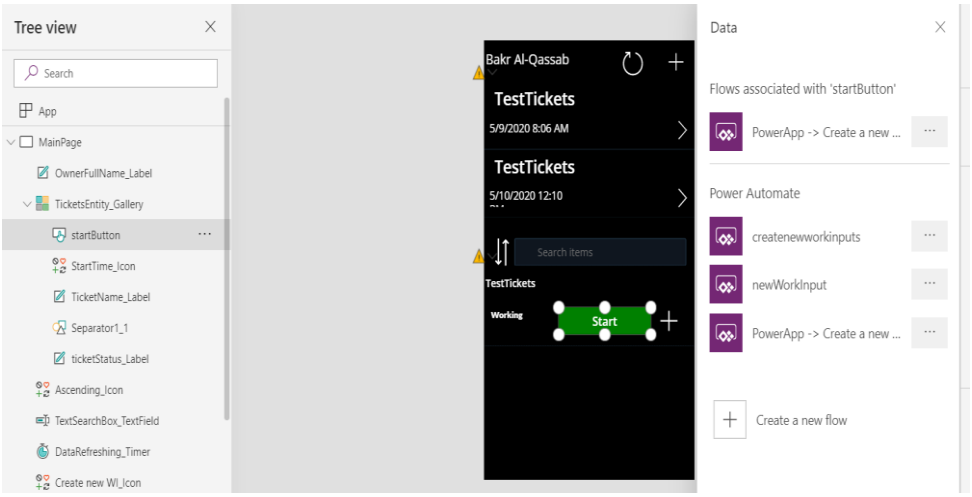

Figure 23. Add Power Automate to the button.

When constitute the Microsoft workflow need to make it send the ticket name, current time, make the status of the new work inputs “Working”. The status value is added directly. The name of the ticket and the time is added from Power Apps. When need some values add from the Power Apps need to select the "Ask in power Apps" option when constituting the Power Automate as show how is filled in (Figure 24).

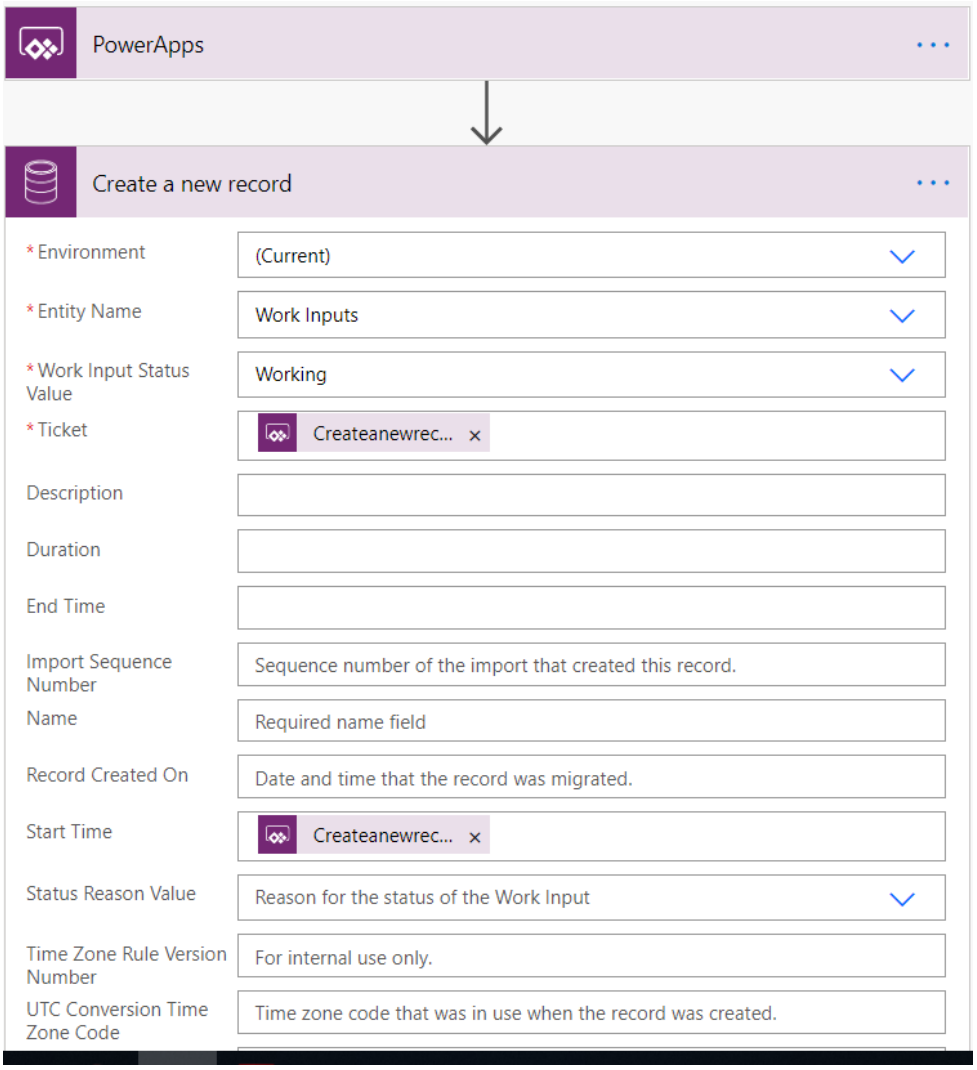

Figure 24. Fill the CDS Entity value in the Power Automate.

After the workflow is added and connected with the button clicked to start processing. Next perform will desire to add the data that choose to add manually from the Power Apps as in (Figure 25).

The added data is the name of the selected ticket. There is one of the elements to use the Gallery is can be referred to as each column's data as " this item" which allows you to reach all the data of the column. To add the current time need to insert the “Now ()” structure in the Formula field.

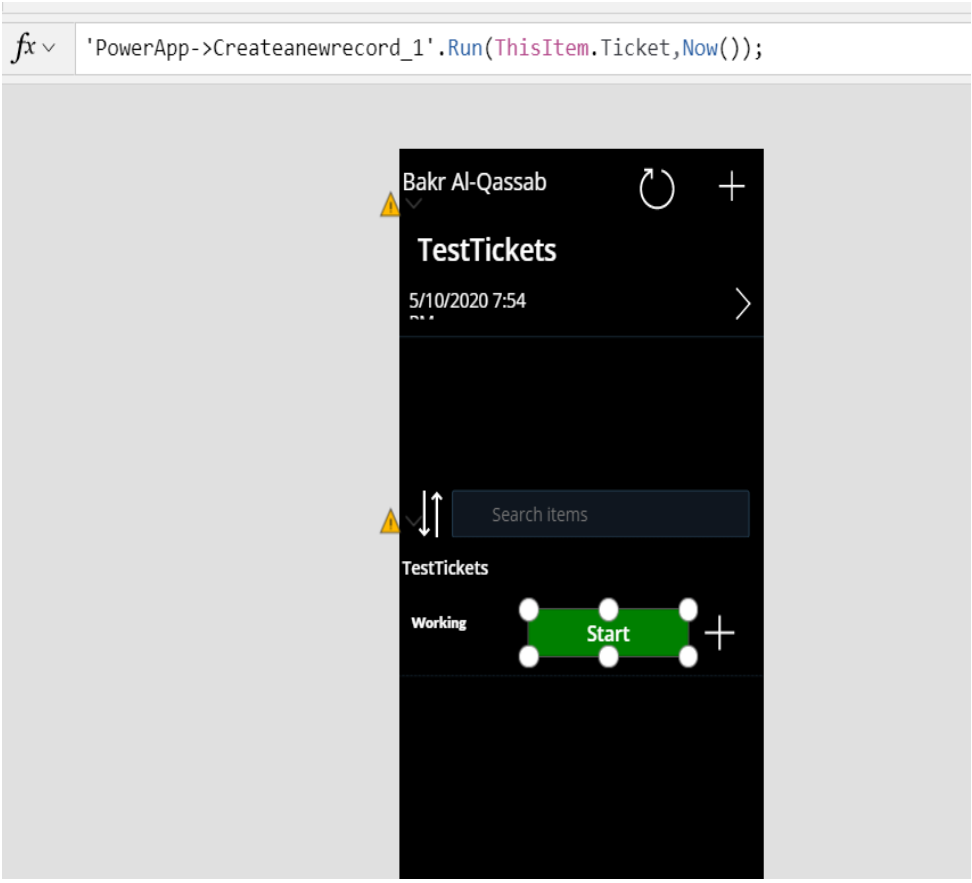

Figure 25. inserted data for the Power Automate formula.

6.4 Submit Data

After finishing creating a new Work Input action along with update the displayed data need to implement the handling the inserting of the new work input's data in the CDS Entity.

In addition to inserting the end time manually. It is accessible to insert it automatically by adding a button on the screen. When the employee ends working on the on-progress work input then will click the end-time button. The back-end will automatically insert the end-time as the current time and the duration will be current time minus inserted start time. The status of the selected work input will be changed to completed as the formula inserted data shown in (Figure 26).

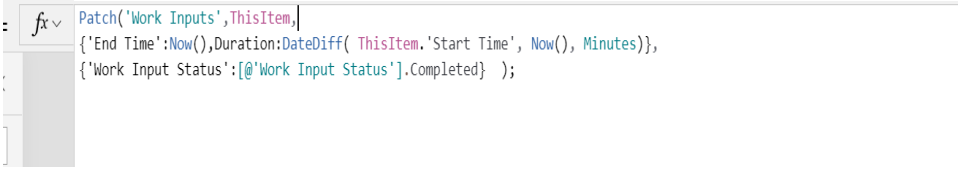

Figure 26. End-time and status submitting formula.

It possible that the Employee accidentally adds new work input or doesn't prefer to calculate the time for the created work input in the system. Then it possible to click the cancel button. The cancel button will reform the work input status to "Cancelled" as the inserted data order shown in (Figure 27). The work Input will disappear from the screen subsequently.

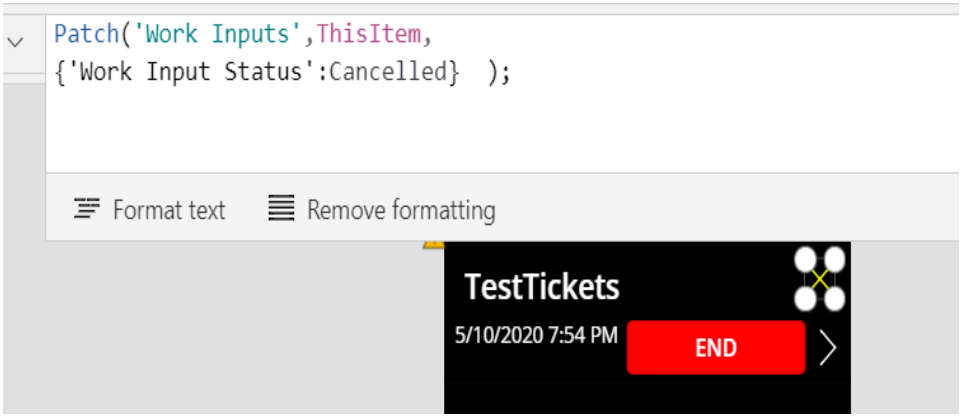

Figure 27. The work input canceling button's formula.

7 PROJECT PARTITIONS

The Application is a complement to a bigger project. The other project parts represent the Computer application as displayed in (Figure 28), Azure DevOps system and partly connecting with the Jira system. Computer application is constituted by Model-Driven App. The application is connecting the system to the Azure DevOps by WorkFlow.

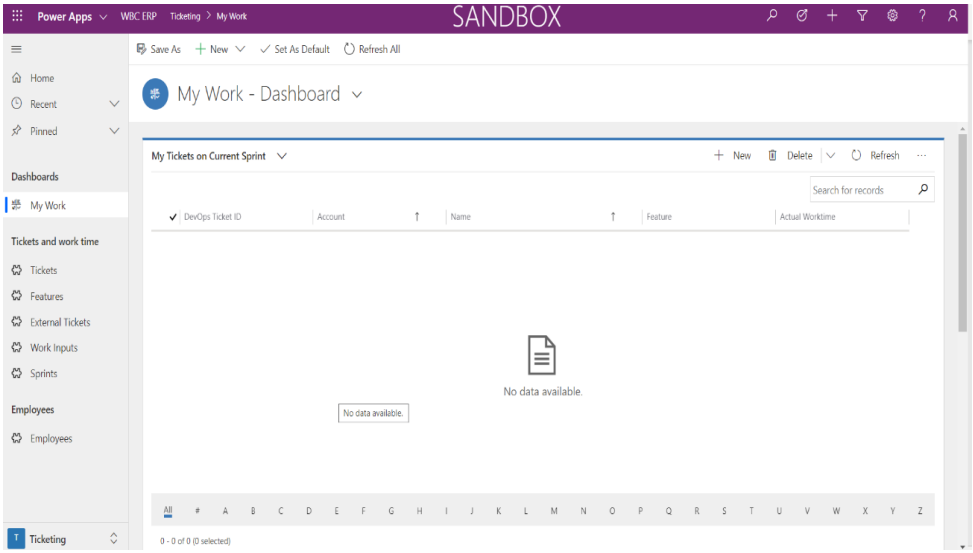

Figure 28. Computer Application UI.

The Computer application is configured by using the Power apps accessible templates. The displayed data generated according to the shape of core business data in the Common Data Service. Every section is created separately. The application templated is flexible to adjust according to business requirements. The computer application is simple as shown in(Figure 29).

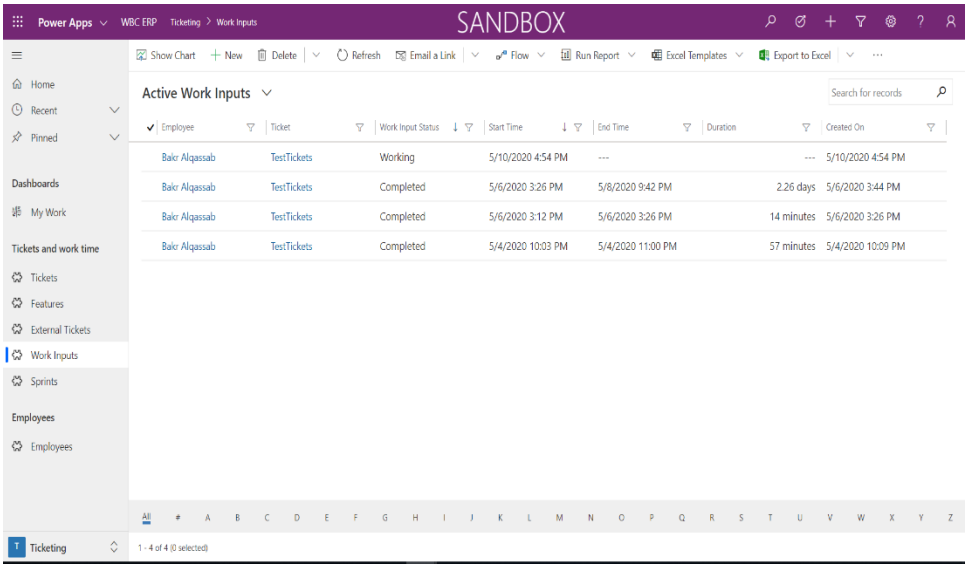

Figure29. Work inputs Entity Displayed data.

In computer application Employers have more control over the project than a mobile application. Computer applications demonstrating more data and give more options like add new users, tickets, work inputs and control everything related to the project.

Another important part of the project is Microsoft Azure DevOps. On Azure, the developers and business side are operating. The project's source code is uploaded in the AzureDevOps cloud.

The computer application authorizes the employee to create new tickets. The project’s parts are complete with each other. The flexibility of binding between Microsoft Power Apps and Microsoft Power Automate's systems simplify reaching the project's goals.

In the coming future, there is a strategy to develop the system to be further qualified to cover all business desires. The main developing strategy will be connecting the system to Jira to calculate the work time when the company's employee works temporally in another company and the host company use the Jira system. Jira is a powerful work management tool suitable for managing teamwork. Jira's work process is like the Azure DevOps Board work Process with little variations and preferences for both of the tools.

8 PLUGINS

The Plugins' integrated development environment was Visual Studio 2017. The coding language is C#. The purpose of use of C# is defined as the common popular coding language in plugins' development and commissioned companies as well. The C# language is familiar to most developers so it will be extremely suitable for further development. The Visual studio using Azure DevOps Server and Visual Studio Team System. These techniques provide various types of tools to support the development environment, like project management, testing, and release management capabilities,... Azure DevOps Server covers the entire project life cycle. Azure DevOps can be used as a back-end to IDEs as you can read more in the 4.5 section. These features are consolidated with the Visual Studio Platform (Azure DevOps/TFS, 2020).

To develop a plugin interface need to use Microsft's IPlugin library. For using IPlugin need to install “Microsoft.Xrm.SDK” (IPlugin interface, 2020).

The IPlugin components are simple to use. It contains the execute method (IServiceProvider). IServiceProvider is a container for service objects. It contains the references for Plug-in execution. The provided refrences like tracing service, organization service, and notification service (IServiceProvider, 2020).

To executes the Iplugin need to create a new file to be more useable. The created file will be called a PluginBase as shown in (Figure 30). The file will consist of the IPlugin. The creating a new file is because additionally of the IPlugin, there are Helper classes that help in reuse the code and covering Organization Service and Execution Contacts. These classes and contexts are created by Microsoft CRM expert “Scott”. Previously in 4.7 and 4.8 sections mentioned more about how Microsoft Power Apps and CRM knowledge are developed during implementing this system.

```
using System;
using System.Collections.ObjectModel;
using System.Globalization;
using System.Linq;
using System.ServiceModel;
using Microsoft.Xrm.Sdk;

namespace WbcErp.Plugins

{
    /// <summary>
    /// Base class for all plug-in classes.
    /// </summary>
    10 references
    public abstract class PluginBase : IPlugin
    {
        /// <summary>
        /// Plug-in context object.
        /// </summary>
        13 references
        protected class LocalPluginContext
        {
            [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Performance", "CA1811:AvoidUncalledPrivateCode", Justification = "LocalPluginContext")]
            0 references
            internal IServiceProvider ServiceProvider { get; private set; }

            /// <summary>
            /// The Microsoft Dynamics 365 organization service.
            /// </summary>
            [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Performance", "CA1811:AvoidUncalledPrivateCode", Justification = "LocalPluginContext")]
            34 references
            internal IOrganizationService OrganizationService { get; private set; }

            /// <summary>
            /// IPluginExecutionContext contains information that describes the run-time environment in which the plug-in executes, information related to the executi
            /// </summary>
            12 references
            internal IPluginExecutionContext PluginExecutionContext { get; private set; }

            /// <summary>
            /// Synchronous registered plug-ins can post the execution context to the Microsoft Azure Service Bus. <br/>
```

Figure 30. PluginBase class.

8.1 **Definitions**

The commissioner company has determined what kind of functions need to implement by plugins. The core idea is to avoid errors and set up dynamics functions that work in the background. The number of plugins per project can be from one to dozen. Plugins help fill the missing actions in the project. The Plugins helps to avoid human error when working on the project. For the first stage There one use case defined to implement in the project. The plugin uses to control the entered data for Work Input's Entity.

The Plugin’s assembly will be about checking the inserted data when creating new work input. The Plugin will consist of various functions. The functions reacting depending on the filled-in data after selecting the intended ticket. If the user inserts only the start time then it will create new work input with start time and the status will be in in-progress. If the inserted data is a start and end time then it will calculate the duration time by subtracting the end time from the start time and status will be Completed. If the inserted data is a start and end time and duration the plugin will calculate the different times between inserted times and corresponding it with the inserted duration and calculate the duration if not matching and the status will be changed to Completed. If the employee gives only the duration the plugin will insert the end time in the time as the current time and the start time will be the current time subtract the duration value and the status will be Completed. The final function will be checking if the start time greater than the end time and warn the user.

8.2 Source Control configuration

The Base of developing Plugins is constructed on the created Solution. The development IDE will be Visual Studio 2019. The created Solution is called “Solution WhiteBlueErp”. Then open the Projects Folder and create a new file called ‘ Solution’ and from the Visual studio program and click right on the project and choose “Add” > “existing Web site”. The benefit of using a web site is to reveal more details of the files than other types. The next step will be to install “ SPKL” and “Microsoft CrmSdk Core Tool” from the Nuget Package as shown in (Figure 31). The Microsoft CrmSdk core tool will be downloaded automatically when installing SPKL because SPKL was established on it.

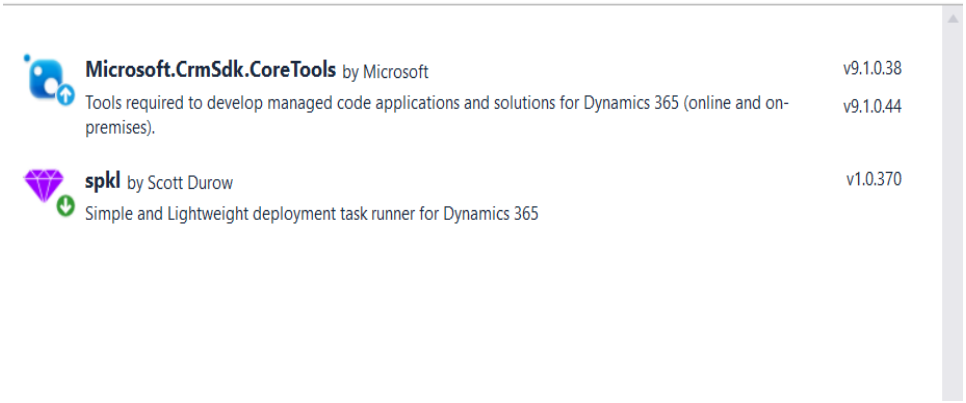

Figure31. SPKL downloading.

Then need to get the Solution's initial metadata internal the source control. This process is done by open the SPKL-bat file and clean the file from over code. SPKL will provide suitable files for various environments like WebResource file. The webResource has controlled the frontend of the project. It’s possible to create the front-end by coding instead of using the ready templates from Microsoft.

Fill the SPKL file to navigate the task runner to look for a solution name in the server and fetch solution data into the project as show how to fill the data in (Figure 32). Then the system will recompose the solution to be a deployable zip file.

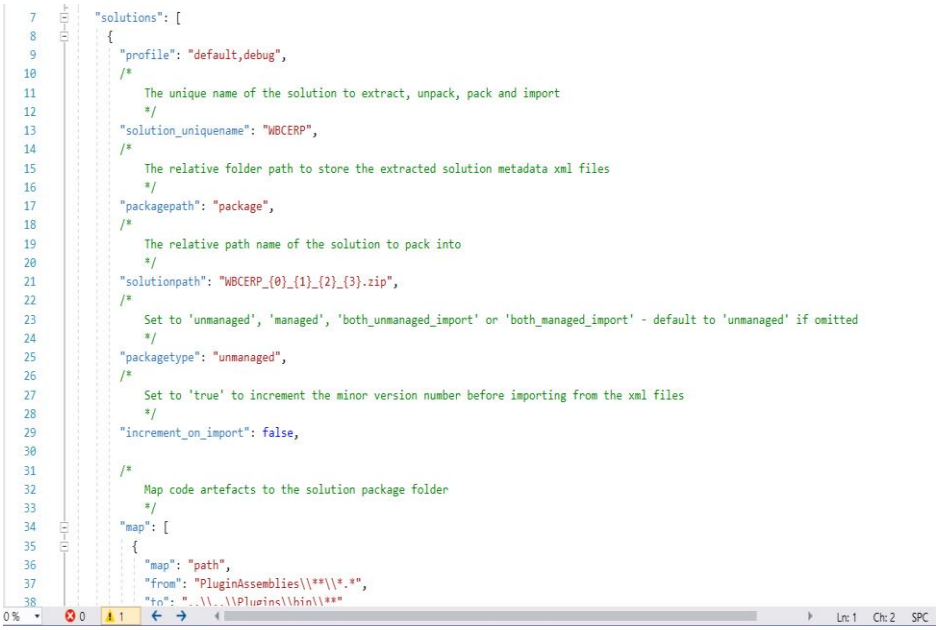

Figure 32. Initialize the SPKL file to fetch solution metadata.

The next phase will be clicking right on the “unpack.bat” file and choose the Execute file. It will pop up the command-line tool where you need to set a new connection by choosing “create a new connection” and insert Y (Yes) answer for the CRM tool. Then answer Y too for the question about is your organization is provisioned in Microsoft office 365. The tool will need to login by your Microsoft account’s User name and Password. The next step will need to choose which instance or the environment is related to the project. Our project environment is called “WBCERP”. After selecting the intended environment then automatically the system will read all metadata in the solution and pull it down locally. The pulling of the metadata will help to check the data in the Source control.

After done need to create the PluginAssemblies phase. Creating PluginAssemblies need to create a new C# class project by clicking right on the project file and select “Add” then “New Project”. Similarly with Solutions need to add Taskrunner by install “SPKL” to help in deploy .Net Plugin code to my development environment. After SPKL needs to install “CRM SDK” by clicking on “Microsoft.CrmSdk.CoreAssemblies”. The CrmSdk helps to combine with Common Data Service.

During builds Plugins and deploy need to give it a strong name as shown in (Figure32). The strong name gives a unique identity to the assembly. The Unique identity prevents the mix among assemblies (Assemblies strong name, 2020).

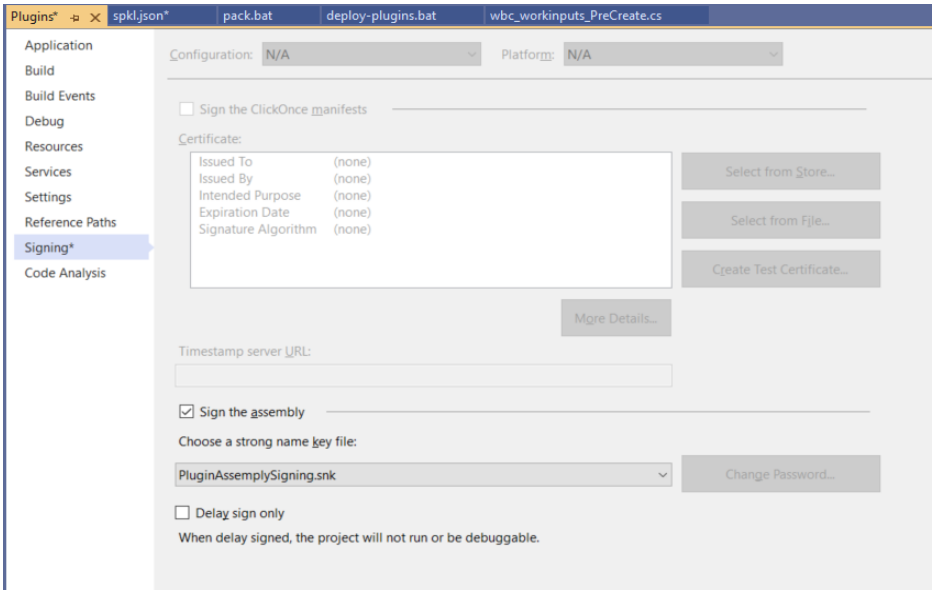

Figure32. giving a strong name to the plugins.

The spkl.json file needs to clear by deleting all over code and leave only the plugin's associated code. Then need to navigate the tool for Solution direction and name to deploy to when executing the “deploy-plugins.bat” file as shown how utilized in (Figure 33).

```
"plugins": [
  {
    /*
          Required - assemblypath - Relative path (from this file or the path parameter above) to the assembies to deply
          Can be either plugins or workflow activities
          Create multiple entries for different profiles
          */
    "solution": "WBCERP",
    "profile": "default,debug",
    "assemblypath": "bin\\Debug"
    /*
          Optional - defines the regex to use to detect a plugin or workflow activity when using a custom base class
          */
    //"classRegex": "((public( sealed)? class (?'class'[\\w]*)[\\W]*?)((?'plugin':[\\W]*?((IPlugin)|(PluginBase)|(Plugin)))|(?'wf':[\\W]*?CodeActivity)))"
  }
],
```

Figure33. Navigate the SPKL file for the solution’s name for deploying it.

As I mentioned in the 4.7 section, SPKL offers EarlyBound Resource. The early Bound code will be downloaded automatically in the “ spkl.json” file. The code structure needs little modification like the add the Solution name and the “EarlyBound.cs” file path as shown in (Figure 34).

```
"earlyboundtypes": [
  {
    /*
          Comma seperate list of entity logical names.
          I've not provided support for -all- entities because this results in unneccessarily large plugins!
          */
    "entities": "wbc_sprint,wbc_batchjob,wbc_workinput,wbc_ticket,wbc_feature,wbc_externalticket,wbc_employee,systemuser,account,contact,task",
    /*
          Comma seperated list of actions request/responses to generate - leave emtpty or ommit for none
          */
    //  "actions": "dev1_simpleaction",
    /*
          Set to 'true' to generate Enums for optionsets
          */
    "generateOptionsetEnums": "true",
    /*
          Set to 'true' to generate Enums for States and Statuses
          */
    "generateStateEnums": "true",
    /*
          Set to 'true' to generate Enums for Global optionsets
          */
    "generateGlobalOptionsets": false,
    /*
          The path (relative to this file) to output
          */
    "filename": "EarlyBoundTypes.cs",
    /*
          Output one file per class/optionset/service/request
          */
    "oneTypePerFile": false,
    /*
          The namespace to put the classes under
          */
    "classNamespace": "WbcErp",
    /*
          The name of the Service context to create - leave blank or ommit for none
          */
    "serviceContextName": "XrmSvc"
  }
]
```

Figure34. SPKL's EarlyBound code.

8.3 Implement the Assemblies

The next phase after prepared the source control fundamentals of the plugin's development will be a practical performance. When installing SPKL then will download the helper method called"CRM Plugin Registration Attribute " as captured in (Figure 35). This helper method will be used to register the plugins.

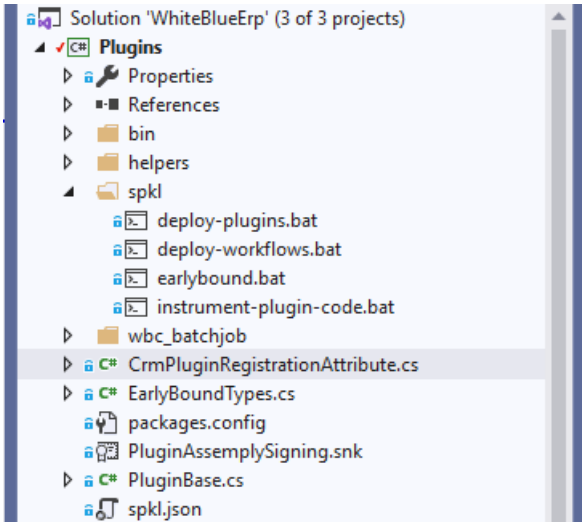

Figure35. SPKL helper method to register the plugins.

Registering the assembly will be done by SPKL because as mentioned in the 4.7 section, most of the operations will be done automatically unlike the plugin registration tool. That will be concluded through navigating the SPKL by inserting the required data as shown in (Figure36). In the previous section, the main steps are explained simply by writing comments above each step.

```
using Plugins.helpers;

namespace WbcErp.Plugins
{
    [CrmPluginRegistration(MessageNameEnum.Create,
        // The Entity Name
        "wbc_workinput",
        // The Stage Type
        StageEnum.PreOperation,
        ExecutionModeEnum.Synchronous,
        // if want to follow only the specific change inside the Entity
        "",
        // Give name to the register Process
        "wbc_workinput PreCreate",
        // Execution Order Number
        1,
        IsolationModeEnum.Sandbox)]

    2 references
    public class Wbc_WorkInput_PreCreate : PluginBase
    {
        0 references
        public Wbc_WorkInput_PreCreate() : base(typeof(Wbc_WorkInput_PreCreate))
        {

        }

        8 references
        protected override void ExecuteCrmPlugin(LocalPluginContext localcontext)
        {
            // Time Checking Variables
```

Figure36. Regestering the plugins by SPKL.

Then need to add new variables to the class to check the inserted data types and fill them in new variables to be ready to add to the CDS concerned entity. To obtain the inserted data in the Entity need to connect the system to the UI fields. Connecting the class to the UI data fields done by IPlugin provided components like in (Figure 37).

```
protected override void ExecuteCrmPlugin(LocalPluginContext localcontext)
{
    // Time Checking Variables

    Boolean isStart;
    Boolean isEnd;
    Boolean isDuration;

    DateTime? start;
    DateTime? end;
    int? duration;

    // Plug-in business logic goes here.
    Entity entity = (Entity)localcontext.PluginExecutionContext.InputParameters["Target"];
```

Figure 37. Create new variables and connect to UI fields.

When constituting new Work Input the Logged in an employee will be determined as the maker of the named work input. In a manual manner, the Work input's creator is not static can be either the logged-in user or another selected employee from the drop-down list.

The code will check if the employee's field is left empty or be filled. This process will be done by the snibbed code in (Figure 38). In case nothing chooses then will insert the logged-in employee name or will insert the chosen Employee.

```
// Check if Employee is given, then just use it

if (entity.Contains("wbc_employee"))
{

    // Do nothing
}
else
{

    // here will obtain the logged user id
    Guid ownerId = ((EntityReference)entity.Attributes["ownerid"]).Id;


    var employees = (from x in new XrmSvc(localcontext.OrganizationService).CreateQuery<wbc_Employee>()

                        // Here will retreive the logged in employee which have Active status
                    where x.wbc_SystemUserid.Id == ownerId && x.statecode.Equals(0)
                    select new wbc_Employee
                    {
                        wbc_EmployeeId = x.wbc_EmployeeId

                    }
                     ).ToList();


    if(employees.Count > 0)
    {
        var employeid = (Guid) employees.FirstOrDefault().wbc_EmployeeId;
        entity.Attributes.Add("wbc_employee", new EntityReference(wbc_Employee.EntityLogicalName, employeid));

    }
```

Figure38. checking the work input's maker field.

To check the inserted data contents that the employee insert needs to going through if statement as shown in (Figure 39). The checking helping to implement the right function to handle the inserted data before pushing it into the CDS. The "isStart" variable will be true if the inserted data contain start time else will be false. When the code checking there is a start time than will the inserted value in the "Start" variable. The same process will be with end time and duration. The next step will check the true fields through the if statement and send the variables to the helper method to handle the data and then send it to the CDS. This proceeding will help to reuse the code.

```
// Calculating Start & End& Duration Time


if (entity.Contains("wbc_starttime") && (entity.GetAttributeValue<DateTime?>("wbc_starttime")) != null)
{

    isStart = true;
    start = entity.GetAttributeValue<DateTime?>("wbc_starttime");

}
else
{

    isStart = false;
    start = null;
}
if (entity.Contains("wbc_endtime") && (entity.GetAttributeValue<DateTime?>("wbc_endtime")) != null)
{

    isEnd = true;
    end = entity.GetAttributeValue<DateTime?>("wbc_endtime");

}
else
{

    isEnd = false;
    end = null;
}
//(int?)entity.Attributes["wbc_duration"] != null

if (entity.Contains("wbc_duration") && (entity.GetAttributeValue<int?>("wbc_duration")) != null)
{

    isDuration = true;
    duration = entity.GetAttributeValue<int?>("wbc_duration");

}
```

Figure39. Go through the inserted data.

After checking the inserted data in the UI need to be sure the inserted start time value not upper than the end time. This step will break the in-progress operation to prevent the error in time calculating and alert the employee to reinsert the time correctly. The used code for executing this process will be as in (Figure 40).

```
// Checking the Start and End time values
if (start > end)
{

    throw new InvalidPluginExecutionException("Start time cannot be after End time!");

}
```

Figure 40: checking the times constrasting.

To avoid code duplicating and make the code easily accessible for further development needs to create a helper class. The helper class code is shown in (Appendix 4). The helper class will handle the confirmed and transferred data from the main class.

The helper class will contain variables and various functions. The variables will be representing start time, end time, duration and status. The concerned function will be executed depending on the transferred values from the main class. If the user inserts only start time then the result will be start time and status will be false. The true/false will help when the data return to the main class to know if the Work Input is on working or completed status. If the user inserts start and end time then the class will calculate the duration and the status will be true. If the user inserts only the duration then the end time will be the current time and the start time will be the end time minus the duration and status will be true.

If the user inserts start&end Time & duration then will confirm if the duration matches the difference between start time and end time. If not then the function will recalculate it again.

After the helper method handles the transferred data from the main class. The main class will take the returned " work input's status" and set or reset the Work Input's status. To publish the status value to the CDS need to be numbered. Every status has a unique number as snipped in (Figure 41). So the fetching value to the function will be True or false if true the status will be “Completed” and if false then it will be “Working”.

```
5 references
private void setWiStatus(bool isCompleted, Entity entity)
{
    if (isCompleted) entity.Attributes["wbc_workinputstatus"] = new OptionSetValue(859960001);
    else entity.Attributes["wbc_workinputstatus"] = new OptionSetValue(859960000);
}
```

Plugin41. Handling the work input’s status value.

After checking the inserted data and settled the status value. This process helps determine which function from the helper method will execute to handle the data and return the results to the main class. The class code will be looking as shown in (Figure 42). The refer to helper method’s functions scenario can be like sending only start time or start and end time or start, end time and duration.

```
if (isStart && isEnd && isDuration)
{

    // throw new InvalidPluginExecutionException("Start Time & End Time & Duration");
    // var weva = new Wbc_ActivityTimeStatus(start, end, duration);
    var wi = new Wbc_ActivityTimeStatus((DateTime)start, (DateTime)end, (int)duration);
    setWiStatus(wi.IsCompleted, entity);

    duration = wi.Duration;

    entity.Attributes["wbc_duration"] = duration; // update to duration!!!!!!!!

}
// && entity.Attributes["wbc_duration"].ToString() == string.Empty
else if (isStart && isEnd)
{


    //   throw new InvalidPluginExecutionException("Start Time & End Time ");
    var wi = new Wbc_ActivityTimeStatus((DateTime)start, (DateTime)end);
    setWiStatus(wi.IsCompleted, entity);

    duration = wi.Duration;

    entity.Attributes["wbc_duration"] = duration;

}
else if (isStart && isDuration)

{
    //  throw new InvalidPluginExecutionException("Start Time & Duration");
    var wi = new Wbc_ActivityTimeStatus((DateTime)start, (int)duration);
    setWiStatus(wi.IsCompleted, entity);

    entity.Attributes["wbc_endtime"] = wi.EndDateTime;
}
```

Figure42. Call the helper class function according to the available data.

After handling the are transferred values from the main class after go through the checking operation. After this process, The final data will be published to the CDS as shown in (Figure 43).

```
else if (isStart)
{
    //  entity.Attributes.Add("wbc_starttime", entity.Contains("wbc_starttime"));
    //throw new InvalidPluginExecutionException("Start Time");
    var wi = new Wbc_ActivityTimeStatus((DateTime)start);
    setWiStatus(wi.IsCompleted, entity);




}
else if (isDuration)
{

    // throw new InvalidPluginExecutionException("Duration");
    var wi = new Wbc_ActivityTimeStatus((int)duration);
    setWiStatus(wi.IsCompleted, entity);


    entity.Attributes["wbc_starttime"] = wi.StartDateTime;
    entity.Attributes["wbc_endtime"] = wi.EndDateTime;


}
```

Figure43. Helper method's functions.

9 CONCLUSION

The mobile application system is implemented as planned. The purpose of the application is to calculate the inserted time for the coming tickets from the cloud. The tickets were created by the computer application or the Azure DevOps system. The displayed tickets in the mobile application are filtered according to the status and the logged-in employee's full name. The status of the displayed ticket should be "Working" so the Employee proceeds with the ongoing work input. The new work input's process is accomplished to be user friendly as much as possible. When the user starts working on a specific ticket can create new work input in two forms. Employees can choose either done the function automatically or manually. The automated creation of the work input just require easily to click the start time button when start and end time when finish. In case need to use the manual then there are various types depends on the generated work input's status. It's possible to create the work inputs automatically and end it manually or the opposite. The manual manner gives more flexibility to the system to fit all the unpredictable situations but the automatic manner more easy and fast to use. The system provides the ability to assign the work input to the temporary worker in the company or temporarily working employees for another company.

The app's functions are tested in daily use in various manners. The test goal is to be sure the program functions will work properly as expected and notice if there are unpredictable errors. The test time was not for a long period. The reason has appeared an emergency crisis when this thesis is written. More deep testing is intended when starting in further development.

Developing apps in Power Apps face some difficulties. The difficulties represent the lack of clear covering the developing guiding and found certified sources. Microsoft tries to reduce the difficulties constantly by developing the system and provides various platforms to share the solutions of Possible encounter difficulties in the newly released and developed features like chat community. Power Apps and CDS powerful increase continuously that help get benefits from other user's experiences. Previously most of the Power Apps users using Sharepoint as back-end storage that leads to being most of the available refrences is talk about SharePoint. Another difficulty when I developed the apps is the deleted features from the last released version of power apps like not to support using more than one Entity in each Gallery, unlike the old version which authorized that. To solve this issue used a Power automate to bind with other Entity.

There are intentions for further development to make the system more qualified to fit business requirements. The conceivable further developing plans is to associate the project to the Jira system, make the system print bills form when needing it and provide the employee easy access for the app by creating the app's icon on the main screen's down bar.

10 REFERENCES & APPENDICES

10.1 - References

Assemblies strong name, M. P. (2020). Strong-named assemblies. Retrieved June 21, 2020, from https://docs.microsoft.com/en-us/dotnet/standard/assembly/strong-named

Azure DevOps/TFS, M. (2020, February 15). Azure DevOps Server. Retrieved April 5, 2020, from https://en.wikipedia.org/wiki/Azure_DevOps_Server

Amit. (2019, December 10). Microsoft Flow Defination (now Power Automate) - And How to Automate Tasks. Retrieved May 21, 2020, from https://tallyfy.com/what-is-microsoft-flow-power-automate/

Azure DevOps, M. (2019). Plan, code, collaborate, ship applications - Azure DevOps definition. Retrieved April 10, 2020, from https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

Canvasapp, M. P. (2020). Get started with formulas in a canvas app - Power Apps. Retrieved May 22, 2020, from https://docs.microsoft.com/en-us/powerapps/maker/canvas-apps/working-with-formulas

CDS, M. (2019). What is Common Data Service? - Power Apps. Retrieved April 10, 2020, from https://docs.microsoft.com/en-us/powerapps/maker/common-data-service/data-platform-intro

CDS Solution, M. P. A. (2020). Use a solution to customize Power Apps - Power Platform. Retrieved February 20, 2020, from https://docs.microsoft.com/en-us/powerapps/maker/common-data-service/use-solutions-for-your-customizations

Company, C. (2018). A-A White Blue Consulting Oy - taloustiedot, Y-tunnus ja päättäjät. Retrieved June 17, 2020, from https://www.finder.fi/Liikkeenjohdon%20konsultointi/A-A%20White%20Blue%20Consulting%20Oy/Riihim%C3%A4ki/yhteystiedot/285865

Desai, D. (2016). Announcing the general availability of PowerApps. Retrieved May 12, 2020, from https://powerapps.microsoft.com/en-us/blog/announcing-general-availability/

Documentation, M. (2020). Microsoft Power Apps Documentation - Power Apps. Retrieved May 21, 2020, from https://docs.microsoft.com/en-us/powerapps/

Dynamics, M. (2020). SPKL definition. Retrieved June 05, 2020, from https://community.dynamics.com/crm/b/develop1/posts/let-s-start-typescript-part-2

Gewarren. (2016). What is a managed code? Retrieved June 01, 2020, from https://docs.microsoft.com/en-us/dotnet/standard/managed-code

IPlugin, M. P. (2020). IPlugin Interface (Microsoft.Xrm.Sdk). Retrieved June 10, 2020, from https://docs.microsoft.com/en-us/dotnet/api/microsoft.xrm.sdk.iplugin?view=dynamics-general-ce-9

Late&Early bound, M. (2020). Late-bound and Early-bound programming using the Organization service (Common Data Service) - Power Apps. Retrieved from https://docs.microsoft.com/en-us/powerapps/developer/common-data-service/org-service/early-bound-programming

Markets, M. A. (2020, January). Business Process Management Market (BPM). Retrieved May 01, 2020, from https://www.marketsandmarkets.com/PressReleases/business-process-management.asp

Microsoft. (2020). New and planned features for Microsoft PowerApps, - Release Notes. Retrieved March 28, 2020, from https://docs.microsoft.com/en-us/business-applications-release-notes/april19/microsoft-powerapps/planned-features

Microsoft dynamics. (2018). What is an Entity in Microsoft Dynamics 365? Retrieved March 20, 2020, from https://www.tutorialkart.com/dynamics365/what-is-an-entity-in-microsoft-dynamics-365/

Microsoft CRM. (2020). Plugin Registration. Retrieved April 1, 2020, from https://www.xrmtoolbox.com/plugins/Xrm.Sdk.PluginRegistration/

Microsoft Dynamics CRM Consultants UK Developer Training Resources. (2016, October 25). Using the Plugin Registration Tool to Add Steps to Plugin. Retrieved May 15, 2020, from http://www.crmconsultants.co.uk/using-the-plugin-registration-tool-to-add-steps-to-plugin/

M. (2020). McKinsey company- The reality of growth in the software industry. Retrieved June 01, 2020, from https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-reality-of-growth-in-the-software-industry

M. (2020). Planned features for Microsoft PowerApps - Release Notes. Retrieved May 11, 2020, from https://docs.microsoft.com/en-us/business-applications-release-notes/october18/powerapps/planned-features

M. (2020). Microsoft Dynamics CRM - Plugins Definition. Retrieved June 16, 2020, from https://www.tutorialspoint.com/microsoft_crm/microsoft_crm_plugins.htm

M. (2020). Plugin registration tool, Register a plug-in (Common Data Service) - Power Apps. Retrieved June 02, 2020, from https://docs.microsoft.com/en-us/powerapps/developer/common-data-service/register-plug-in

PowerApps, M. (2020). Power Apps Community. Retrieved May 10, 2020, from https://powerusers.microsoft.com/t5/Power-Apps-Community/ct-p/PowerApps1

Power Apps Collection. (2018, April 29). What is the collection & How to manage a collection in PowerApps? Retrieved March 12, 2020, from https://indiandotnet.wordpress.com/2018/04/29/what-is-collection-how-to-manage-a-collection-in-powerapps/

Rouse, M. (2016, March 18). What is Microsoft PowerApps? - Definition from WhatIs.com. Retrieved April 20, 2020, from https://searchcontentmanagement.techtarget.com/definition/Microsoft-PowerApps

Scottdurow. (2020). SPKL repository (GitHub) – Scott Durrow/SparkleXrm. Retrieved June 15, 2020, from https://github.com/scottdurow/SparkleXrm/wiki/spkl

Tapanm-MSFT, M. (2020). Overview of creating apps - Power Apps. Retrieved May 25, 2020, from https://docs.microsoft.com/en-us/powerapps/maker/

Udemy. (2020). Online Courses - Learn Anything, On Your Schedule. Retrieved March 01, 2020, from https://www.udemy.com/

us, C. (2020, May 10). Microsoft Canvas & Model-driven Apps: Clients First. Retrieved May 21, 2020, from https://www.clientsfirst-us.com/microsoft-powerapps-canvas-vs-model-driven-apps/

W. (2020). Microsoft Visual Studio 2019. Retrieved May 12, 2020, from https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

10.2 **Appendices**

Visio of project structure Appendix 1

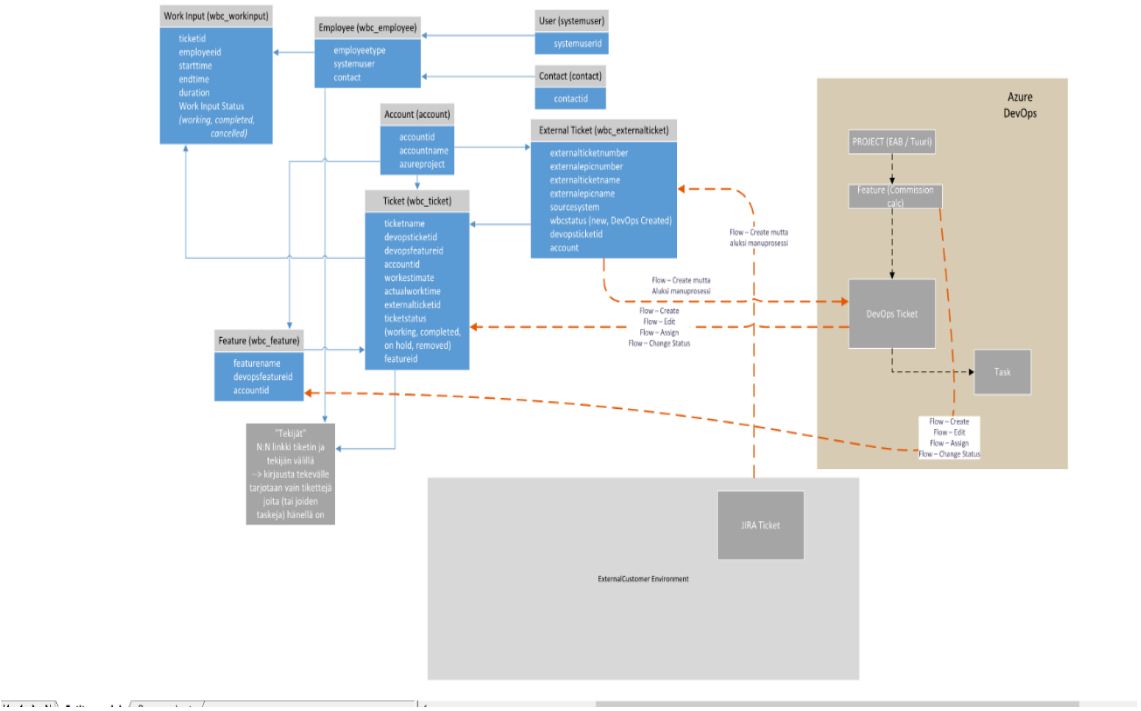

Table of Mobile app's UI Elements **Appendix 2**

No	Button Value	Power Apps Canvas App function implementation
1	New work input Button	This button will show create a new Work Input Page where the user can add information manually
2	Refresh Button	This button will refresh the Common data service data to show the updating information in other employers add something
3	Logged in user label	This Label will show the full name of the logged-in user, which his actions will be saved in CDS

4	Search Text field	Because the work is extending constantly, so the tickets can be too many, for that this search field will help to found a specific ticket easily
5	Organize button	This icon helps to organize the tickets name in an ascending way
6	Ticket name	This label the ticket name, so the user can choose which ticket he/she wants to work on.
7	Status label	This label shows the status of the ticket to the user
8	Start Button	This button will allow starting calculating a time to the specific user in the same field as the Button on.
9	Add Start Time Button	This button allows the user to insert the start time manually I case forget when started
10	Work inputs ticket in process stage label	This label shows which ticket is in the process stage
11	Started time to the work input label	This label showed the started time for the work inputs
12	End time Button	This button allows inserting the current time as the end time for the selected work inputs
13	Add end time manually	This button allows inserting the end time manually in case the user forgets to click the end button when the end.
14	Work input canceling button	In case the user clicked the wrong ticket to work on or no need to work on this ticket can cancel it by clicking the ‘x’ button

Table of canvas app Formula implementation in backend Appendix 3

No	Button Value	Work Process
1	New work input Button	Navigate(CreateNewWorkInput); NewForm(CreateNewWI_Form)
2	Refresh Button	Refresh('Entity1);Refresh(Entity2);
3	Logged in user label	User().FullName

4	Search Text field	Here will be used by the Gallery to get the value from here and insert it in the filter function
5	Organize icon	Sort('Entity1,Name,Ascending)
6	Ticket name	ThisItem.Name
7	Status label	ThisItem.'Ticket Status'
8	Start Button	CreatenewWorkInputs.Run("Entity2","",ThisItem.Ticket,Now()); Refresh('Entity2');Refresh(Tickets); UpdateContext({TimerStart:true}); ((CreatenewWorkInputs is a Workflow function ti insert he data to the CDS Entity))
9	Add Start Time Button	NewForm(AddStartTime_Form); Navigate(InsertStartTimeManually_Page); 'End Time_DataCard5_1'.Update=Today()
10	Work inputs ticket in process stage label	ThisItem.Ticket.Name
11	Started time to the work input label	ThisItem.' Start Time'
12	End time Button	Patch('Entity2',ThisItem, ",Duration:DateDiff(ThisItem.'Start Time', Now(), Minutes)}, {'End Time':Now() ,Duration:DateDiff(ThisItem.'Start Time', Now(), Minutes)}, {'Work Input Status':[@'Work Input Status'].Working});
13	Add end time manually	Navigate(InsertEndTimePage); NewForm(EndTime_Form)
14	Work input canceling button	Patch('Work Inputs',ThisItem, {'Work Input Status':[@'Work Input Status'].Cancelled});
15	Timer	Here I create a timer to update the data after two seconds from I clicked the start button to show the created work inputs in Work Inputs Gallery

Helper class to handle User inserted data Appendix 4

```
using System;

namespace Plugins.helpers
{
    public class Wbc_ActivityTimeStatus
    {
```

```
public DateTime StartDateTime;
public DateTime EndDateTime;
public int? Duration;
public Boolean IsCompleted;
public Wbc_ActivityTimeStatus(DateTime starttime)
{

    StartDateTime = starttime;
    IsCompleted = false;

}

public Wbc_ActivityTimeStatus(DateTime starttime, int duration)
{

    StartDateTime = starttime;

    Duration = duration;
    EndDateTime = StartDateTime.AddMinutes(duration);

    IsCompleted = true;

}

public Wbc_ActivityTimeStatus(DateTime starttime, DateTime endtime)
{

    if (starttime > endtime)
    {
       throw new NotSupportedException("ERROR: star tTime cannot be greater
      than End Time.");

    }
    else
    {
        TimeSpan WorkTime = endtime - starttime;
        StartDateTime = starttime;
        EndDateTime = endtime;
        Duration = (int)WorkTime.TotalMinutes;
        IsCompleted = true;
    }




}
     public Wbc_ActivityTimeStatus(DateTime starttime, DateTime endtime,
     int duration)
{

    StartDateTime = starttime;
    EndDateTime = endtime;

    TimeSpan WorkTime = endtime - starttime;


    if (WorkTime.TotalMinutes != duration)
    {
        Duration = (int)WorkTime.TotalMinutes;
        IsCompleted = true;
    }
```

```
            else
            {
                this.Duration = duration;
                IsCompleted = true;
            }

        }

        public Wbc_ActivityTimeStatus(int duration)
        {

           EndDateTime = DateTime.UtcNow;
           StartDateTime = EndDateTime.AddMinutes(-duration);
            Duration = duration;
            IsCompleted = true;

        }
    }
}
```