Lauri Aho

Mobile data collection and storing solution for Microsoft environments

Bachelor's thesis

Information technology

2018

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	**Tutkinto**	**Aika**
Lauri Aho	Insinööri (AMK)	Marraskuu 2018
Opinnäytetyön nimi Tiedon keräämis- ja tallentamisratkaisu Microsoft-ympäristöön.		40 sivua
Toimeksiantaja T-Base Oy		
Ohjaaja Marko Oras		

Tiivistelmä

Tämän opinnäytetyön tavoite on tuottaa digitalisoitu ratkaisu työmaaympäristössä tapahtuvaa tiedonkeruuta varten mobiilialustoille. Ratkaisu integroidaan Microsoftin Office365-, Azure-, ja SharePoint-ympäristöihin. Ratkaisua käytetään reklamaatioiden aloittamiseen ja turvallisuuspoikkeamien raportointiin. Ratkaisun vaatimuksiin kuuluu toiminta mobiililaitteella, jolla ei välttämättä olet jatkuvaa pääsyä internetiin. Ratkaisun vaadittavat ominaisuudet koostuvat teksti- ja kuvatiedon keruusta, sekä niiden taltiointimahdollisuudesta tulevaa käyttöä ja raportointia varten.

Ensiksi kerättiin ratkaisun vaatimukset, jotka perustuivat asiakkaan tarpeisiin. Kerättyjen vaatimuksien perusteella ratkaisu koostuisi kahdesta mobiilisovelluksesta, Microsoft Azure-pilvessä sijaitsevasta tietokannasta, sekä ohjelmallisesti automatisoidusta työnkulusta. Automatisoitu työnkulku siirtäisi isokokoista dataa tietokannasta SharePointiin. Kun toteutus oli suunniteltu, kehitystyö voitiin aloittaa. Aluksi jo olemassa olevaa tietokantaa päivitettiin ylimääräisillä tietotauluilla ja kuvien käsittelyyn tarkoitetuilla stored procedurella. Tämän jälkeen toteutettiin mobiilisovellukset, joissa olivat vaaditut tiedonkeruuominaisuudet. Ominaisuudet toimivat riippumatta mobiililaitteen nettiyhteydestä. Lopuksi toteutettiin automaattinen työnkulku, joka vastaa kuvien käsittelystä.

Kehitystyö meni hyvin, sillä kirjoittajalla oli aikaisempaa kokemusta samanlaisten järjestelmien toteuttamisesta. Ratkaisu toimi vaatimusten mukaan, ja se tultaisiin viemään tuotantoon aikataulun mukaisesti.

Asiasanat

raportointi, pilvipalvelut, mobiilisovellukset, tietokantaohjelmat, SQL, PowerApps

South-Eastern Finland University of Applied Sciences

Author (authors) Lauri Aho	**Degree** Bachelor of Engineering	**Time** November 2018
Thesis Title Mobile data collection and storing solution for Microsoft environment		40 pages
Commissioned by T-Base Oy		
Supervisor Marko Oras		

Abstract

The objective of this thesis was to produce a digitalized solution for gathering data with mobiled devices, in a construction work environment, integrated into Microsoft's Office365, Azure, and SharePoint environments. This system would be used to start reclamations and report safety anomalies. The system was to be usable on a mobile device without constant access to the Internet. The required functionalities included data gathering in text or image form and the ability to store the data in the commissioner's cloud for further inspection and reports.

First, the requirements for the system were defined in detail, making sure that there was a clear understanding on what the commissioner wanted from it. Based on the requirements, the system would consist of two mobile applications, a database located in the Microsoft Azure cloud, and an automated workflow that would push the pictures from the database into SharePoint, where they would not bloat the database with their large file size. When the implementation was planned, the development was started. First, the currently existing database was updated with additional data tables and a stored procedure for handling the image data. Then the mobile applications were implemented with the required data gathering features which were functional regardless of the device's Internet connection. Finally, the automated workflow was implemented to manage the transfer of the pictures.

The process attained the objective of this thesis, benefitting from the author's previous experience regarding the type of solutions that were commissioned. The system functioned as intended and was to be deployed for production as scheduled.

Keywords

reporting, cloud services, mobile application, database software, SQL, PowerApps

SISÄLLYS

TERMS AND ABBREVIATIONS

Office365	Subscription service provided by Microsoft. Allows usage of Microsoft Office software, such as Outlook and SharePoint and access of other cloud-based software as a service.
SharePoint	Microsoft’s collaborative platform mostly used in document management and data storage.
IDE	Integrated development environment
SQL	Structured Query Language
Back-end	The functionality behind a piece of software, usually not visible to the end user. Includes things like code, and file transfer.
Front-end	The part of software that the the user sees. Includes things like buttons, input fields and images.
UML	Unified Modeling Language
AD	Active Directory
UID	Unique Identifier
GUID	Globally unique identifier
Base64	Binary-to-text encoding scheme that represents binary data in a string format.
Blob	Binary large object, collection of binary data stored as a single entity, typically multimedia objects.
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.
Int	Short for integer, a number that does not have a fractional component. Used a data or variable type.
Ad-hoc testing	A form of software testing that has no planning or documentation.

1 INTRODUCTION

The rise of smart devices has drastically changed our conception of how we communicate, work and entertain ourselves. The power of modern mobile devices allows us to use them, for example as alarm clocks, gaming platforms, cameras, and communication channels. Even companies outside the IT-industry are showing interest in the advantages of using mobile devices in their daily work.

Paperwork is an issue that all companies will have to face eventually. Forms, notes, announcements, data flow; all of these are traditionally handled in paper form. It would only make sense that mobile devices would eventually be harnessed in work environments, increasing the effectiveness of a company's workflow, and eliminating unnecessary paperwork.

1.1 Objective of the thesis

This thesis was a part of a larger project, which was commissioned by a client construction company to the commissioner of this thesis. The purpose of the project was to digitalize their communication, data gathering and reporting systems. The digitalized systems would be used in Microsoft's Office365 and SharePoint online environments. The scope of this thesis consists of the digitalization process of the corporation's data gathering processes.

This thesis is a design-based study that examines the development of two digitalized data gathering solutions. The documentation is divided into three parts. The first part introduces the general requirements which both solutions have been given. After the requirements, the documentation presents the tools used in the implementation of the solution. The third part consists of the actual development of the solutions, ending with the results of the process and conclusions.

1.2 Commissioner

This thesis was commissioned by T-Base Oy, a Microsoft Partner software company, based in Kotka, Finland. Founded in 1999, T-Base provides companies with IT-solutions, based on Microsoft products. Some of T-Base's most commonly known clients include Andritz and ABB. The commissioner considered the digitalization of the data gathering systems a sufficient scope for this thesis.

2 SOFTWARE SPECIFICATIONS

Before even considering starting the development process, both solutions need specifications. Developing a software system without a specification is a random process. The implementation is doomed to be modified, sometimes forever, because it never precisely matches the client's needs (Frappier & Happrias 2006, vii.).

The solution commissioned by the client consists of the digitalization process of several, previously physical forms, used in the client's daily work. Two of the forms include the reclamation data gathering report, and the safety anomaly reporting form. Both forms are to be entirely digitalized into the Office365, SharePoint, and Microsoft Azure environments for reporting, tracking and storing purposes.

2.1 General requirements

The digitalized forms are intended to be used on mobile devices in various locations, some of which have no Internet access. A typical instance of usage consists of multiple sessions with the user's connection quality constantly changing, as is visualized in the UML diagram in Appendix 1. The user must be able to fill out the form throughout multiple sessions and be able to access the data already input in the form during previous sessions.

The data produced by the forms is always related to a project. The forms must grant the user access to a list of his/her active projects, so the gathered data can be related to the appropriate project. The project data needs to be accessible, even though there may not be an Internet connection available. Ideally, the project data will always be available from the user devices memory.

In addition to textual, numeral and date information, the user needs to be able to include images taken with the mobile device’s camera along the gathered data. The data that was collected needs to be stored in a database where it can be accessed by other applications and software. The images must be saved to a document library, located in the projects SharePoint site, where they can then be accessed when other applications review the data with the images. The forms must use Office365 credentials for both data gathering and authentication.

2.2 Reclamation data gathering tool

According to the client, reclamations take place when a subcontractor’s work does not meet the requirements of the contract, be it due to time, safety, or planning issues. Most of the time, these issues are detected during work, where there is no access to a computer, which one could make a reclamation report with. In these cases, the client already has the required information to start a reclamation which could be sent for processing immediately. With a mobile reclamation data gathering tool, the people in charge of the site can start reclamations as soon as the fault in contract terms is detected.

The data required to start a reclamation process consists of

- the project the reclamation is related to.
- the person, corporation, or entity the reclamation is directed towards.
- the product, service, or piece of work that the reclamation targets.
- a description of the situation.
- the first name, last name, and company e-mail of the person filling out the reclamation.

- the date on which the reclamation was filed.

2.3 Safety anomaly reporting tool

While the worksites function under strict safety regulations, according to the client, there is no guarantee that the site is clear of hazards, violations, and other factors that may affect endanger people working on the site. Safety anomalies are something that the client wants reported as soon as possible. However, carrying paper forms around a construction site is not an option. With a mobile reporting tool, the safety anomalies and violations can be reported by people in charge, on the spot, making the client’s work of monitoring the safety ratings of their work sites significantly easier.

An anomaly report consists of

- the location or structure the anomaly affects.
- brief information regarding the aforementioned location or structure, e.g. structure serial number, or road number.
- a description of the anomaly, how it affects security, and what regulations it violates.
- the subtype of the anomaly.
- the project the anomaly is related to.
- the first name, last name, and company e-mail of the person filling out the anomaly report.
- the date on which the anomaly was filed.

The subtypes are defined by the client and should be accessible from e.g. a dropdown menu in the form application.

3 TOOLS

In order to even begin implementing the solutions, a SQL database server is required for the back-end. The front-end requires a mobile application that can transmit both the user-submitted data to the database and the images to SharePoint. The front-end side of both solutions consists only of the mobile user interface found in the data gathering tools. The mobile data gathering solutions will have their front-end applications implemented with Microsoft

PowerApps. The back-end of both solutions will involve handling the data that the user sends from the mobile tool and storing it into the appropriate location. The functionality of the back-end is automated with Microsoft Flow and requires no input from the end user.

3.1 Microsoft PowerApps

Microsoft PowerApps is a suite of apps, services, connectors and data platforms that provides a rapid application development environment to build custom apps for business needs (Microsoft 2018). A public review of PowerApps was released in April 2016 (Desai 2016). With a simple user interface, low learning curve and its large selection of data connectors that allow usage of data outside of the software, PowerApps enables both end-users and developers to build mobile applications for various uses within the Office365 and Azure environments. The IDE was released both as a desktop application and a web-based service.

PowerApps uses Microsoft's formula system to perform its logical tasks. As is displayed in Figure 1, formulas can be used to determine how the application responds when users select a button, adjust a slider, or provide other input (Microsoft 2016). PowerApps also uses global variables to supplement the formula system. The formulas themselves are attached to PowerApps' UI components properties. PowerApps can store variables and other data, such as blobs in Collections. Collections are PowerApps' own data tables that can be saved in the user's mobile devices and loaded on demand (Microsoft 2018).

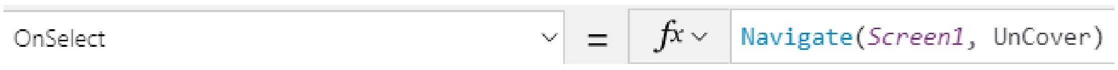

OnSelect =
```
Navigate(Screen1, UnCover)
```

Figure 1: A PowerApps formula, bound to a button's OnSelect-property

3.2 Microsoft Flow

Microsoft Flow is a task automation service, provided by Microsoft. With a public preview released in April 2016 (Microsoft 2018) , Flow allows the user to automate tasks on multiple software, and media platforms, e.g. the end user can send himself a push notification that contains the day's weather every day at a certain time, or receive an e-mail every time someone modifies an Azure SQL Database. A workflow made with Microsoft Flow will be used in the process of storing the data and media into appropriate places in the commissioning corporation's Office365 and Azure environments.

3.3 SQL Server Management Studio (SSMS)

SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure (Microsoft 2018). SSMS has a graphical user interface called Object Explorer (Figure 2) that allows selecting and designing tables without SQL queries.

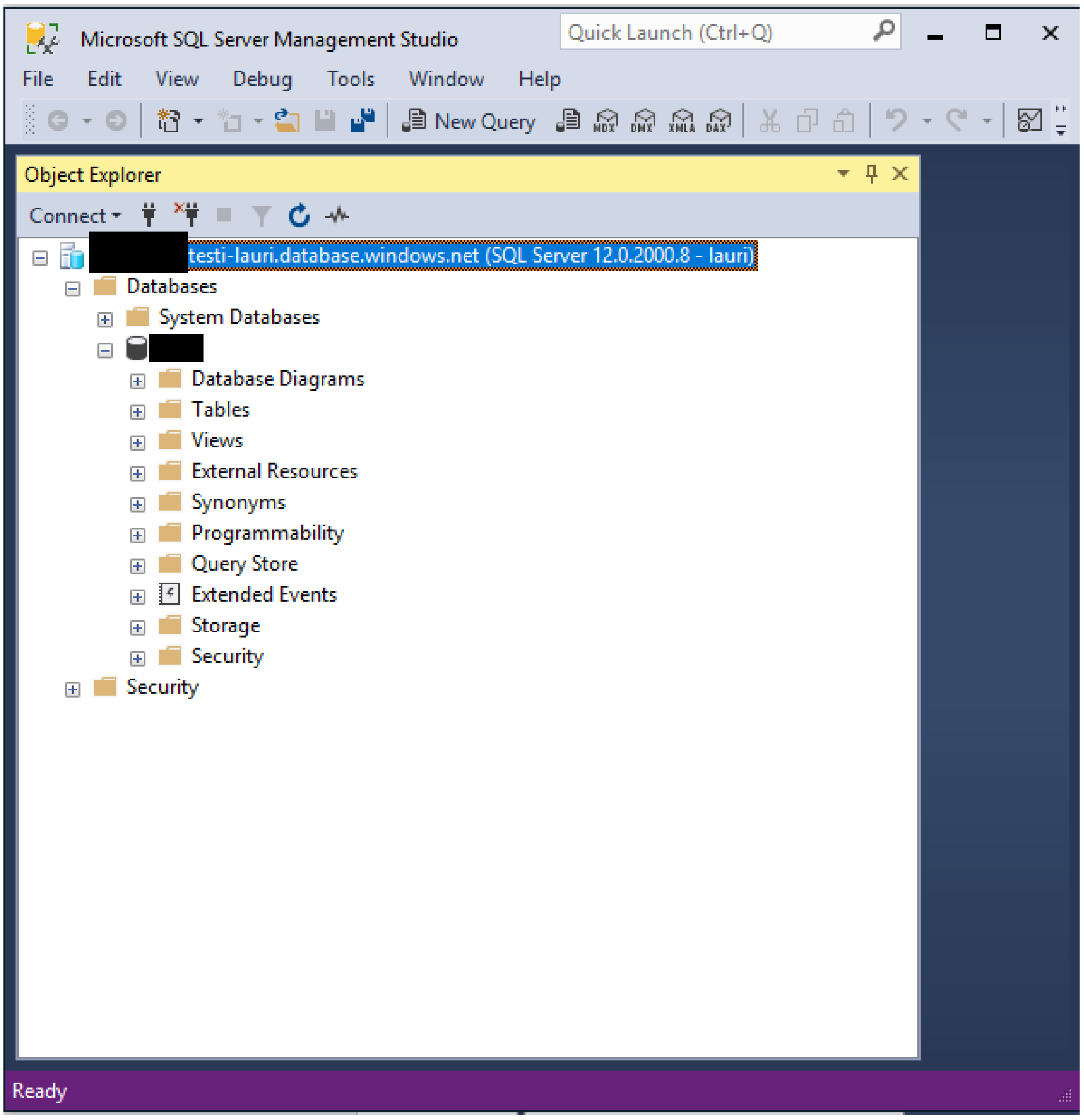


Figure 2: SSMS Studio user interface

The commissioned project includes an already implemented Azure SQL database that contains data which is required for the form’s functionality, such as the safety anomaly subtypes. The data sent from the mobile application will be pushed into an Azure SQL Database with Microsoft Flow. The database design and architecture will be implemented with the use of SSMS.

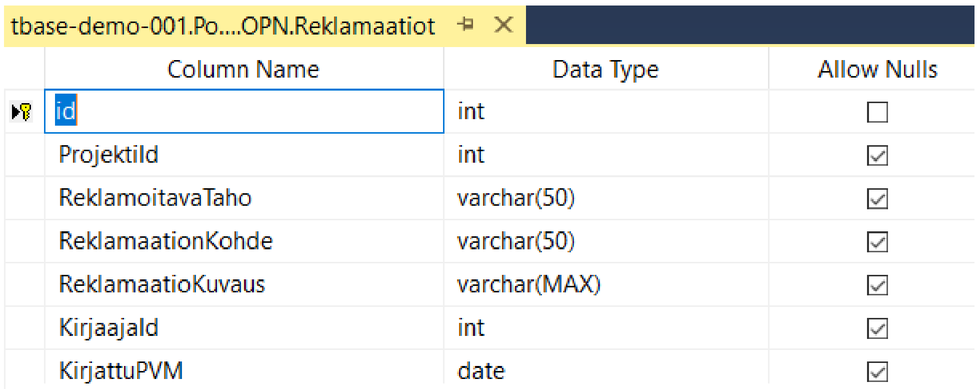
4 DATABASE PREPARATIONS

While the database already contains the required data for the mobile applications to use, the data input by the user that the applications send need a place as well. The reclamations and the anomalies will be stored in their own data tables. A data table is a database object that consists of data rows. A data row consists of multiple columns that contain varying types of data. This data can be, for example numbers, text, or date values.

The already existing data that will be used consists of a data table that has the project data in it, a table of personnel information, a table containing the anomaly types, and a table to temporarily store the image data from the pictures that the users take in the mobile application. The anomalies and reclamations, however, need their own data tables.

4.1 Creating data tables

The first operation is to create a new table. Tables can be created with SQL queries and clauses, but in SSMS they can also be created in the Object Explorer by right-clicking the Tables-folder and selecting “New Table”. This opens the table designer window (Figure 3). From the designer window, columns can be added by typing the name of the column, specifying a data


tbase-demo-001.Po....OPN.Reklamaatiot

Column Name	Data Type	Allow Nulls
id	int	☐
ProjektiId	int	☑
ReklamoitavaTaho	varchar(50)	☑
ReklamaationKohde	varchar(50)	☑
ReklamaatioKuvaus	varchar(MAX)	☑
KirjaajaId	int	☑
KirjattuPVM	date	☑

Figure 3: SSMS database designer interface

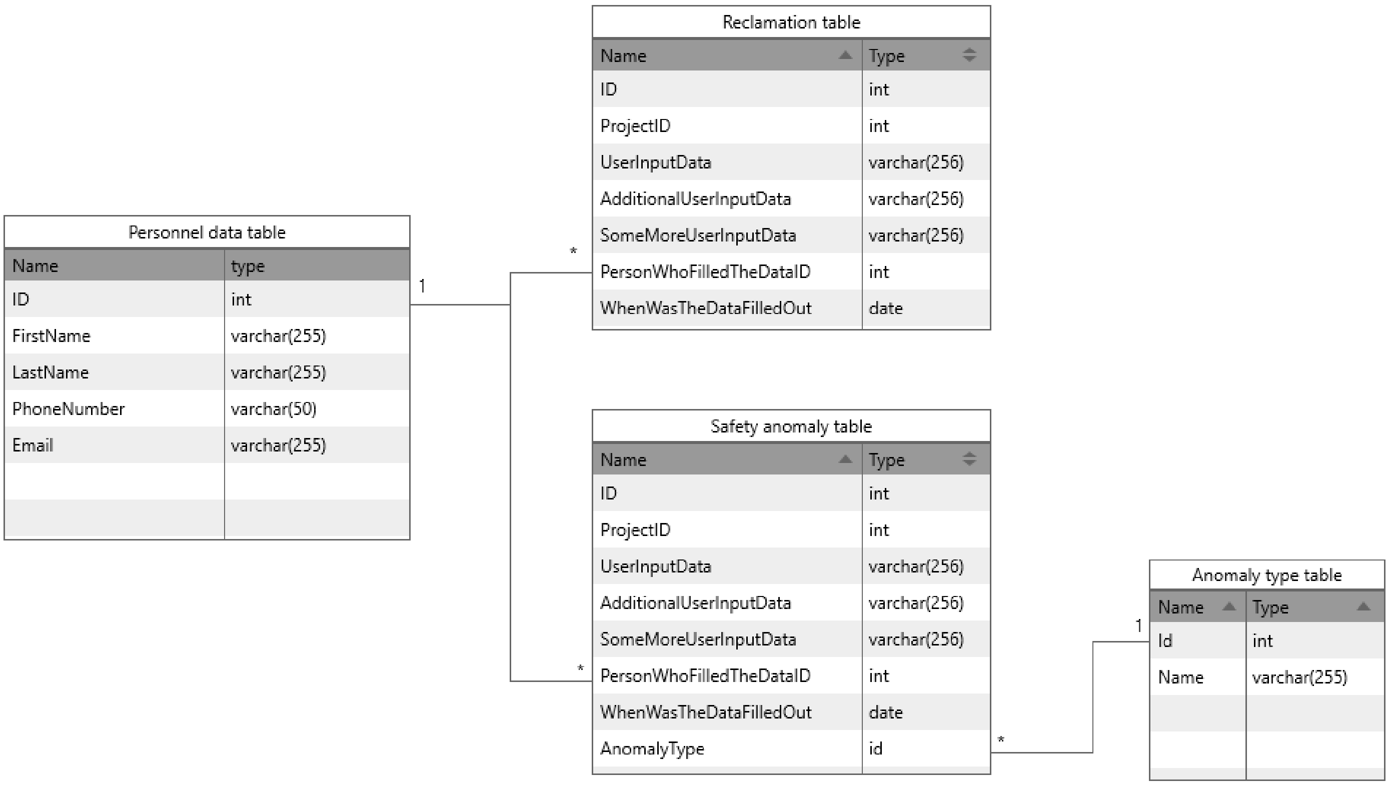
type, e.g. varchar(50). Varchar means that the data is in text form, with the number specifying the maximum character length of the text.

4.2 Data types

Since the reclamations and anomalies will be used in other applications once they are gathered, they need to be labeled somehow. A “ID” – named column that only accepts numbers, is set as a primary key, and is forced to increment each time a new instance is made, will serve as a label. When a column is a primary key, each row in the table must have a unique value in that column. Implementing automatic incrementing makes the column easier to manage as it is not required to be input manually, and there is a smaller chance in duplicate values due to the elimination of human interaction.

In addition to having an own identifier, the reclamations and anomalies are related to the existing projects. Therefore, a “ProjectID”-column is required. The column is functionally identical to the ID-column, apart from not being a primary key, as projects can have multiple reclamations and anomalies related to them. Having this field set as a primary key would make it impossible to make more than one reclamation or anomaly per project.

The data that is directly input from the user is text data and can be of varchar-type. The person who filled up the form, has his/her information already stored in the personnel table. This means that an ID-type will work when the information of the person is needed. The field has a relation to the personnel data table, as displayed in Figure 4. It is possible for one person to have created several reclamations, but only one person can be responsible for a single reclamation.



Reclamation table

Name	Type
ID	int
ProjectID	int
UserInputData	varchar(256)
AdditionalUserInputData	varchar(256)
SomeMoreUserInputData	varchar(256)
PersonWhoFilledTheDataID	int
WhenWasTheDataFilledOut	date

Personnel data table

Name	type
ID	int
FirstName	varchar(255)
LastName	varchar(255)
PhoneNumber	varchar(50)
Email	varchar(255)

Safety anomaly table

Name	Type
ID	int
ProjectID	int
UserInputData	varchar(256)
AdditionalUserInputData	varchar(256)
SomeMoreUserInputData	varchar(256)
PersonWhoFilledTheDataID	int
WhenWasTheDataFilledOut	date
AnomalyType	id

Anomaly type table

Name	Type
Id	int
Name	varchar(255)

Figure 4: A UML class diagram of the reclamation and anomaly tables and their relations

Because safety anomalies are required to have a subtype, the anomaly table requires one specific column, namely the anomaly type ID-column. The anomaly ID also has a relation to the anomaly subtype table. Like the personnel table, only one anomaly type is allowed per anomaly report. Since the types already exist in another table, as is seen in Figure 4, the number of the type will suffice. Finally, the date when the reclamation or anomaly is created, which is a date-type column, accepting only dates, for example, 07-12-1994.

4.3 Stored Procedure

With the data tables in place, the last item to implement into the database, is a stored procedure for image processing. A stored procedure is a piece of SQL code that is saved in the database for reoccurring usage. The procedure can also be invoked outside of the system. A stored procedure is implemented to convert the saved images from binary form to Base64 string data.

First, both incoming and outgoing variables are claimed, like in the procedure that is displayed in Appendix 2, in rows 3-5, with `Base64Data` being the one that is returned from the procedure. `Dataguid` is required for identifying which image is to be converted. Finally, `Nimi` is for parsing a unique name for the image that contains information regarding the image itself and the reclamation or anomaly it was related to. The incoming variable is taken from user input when the procedure is invoked.

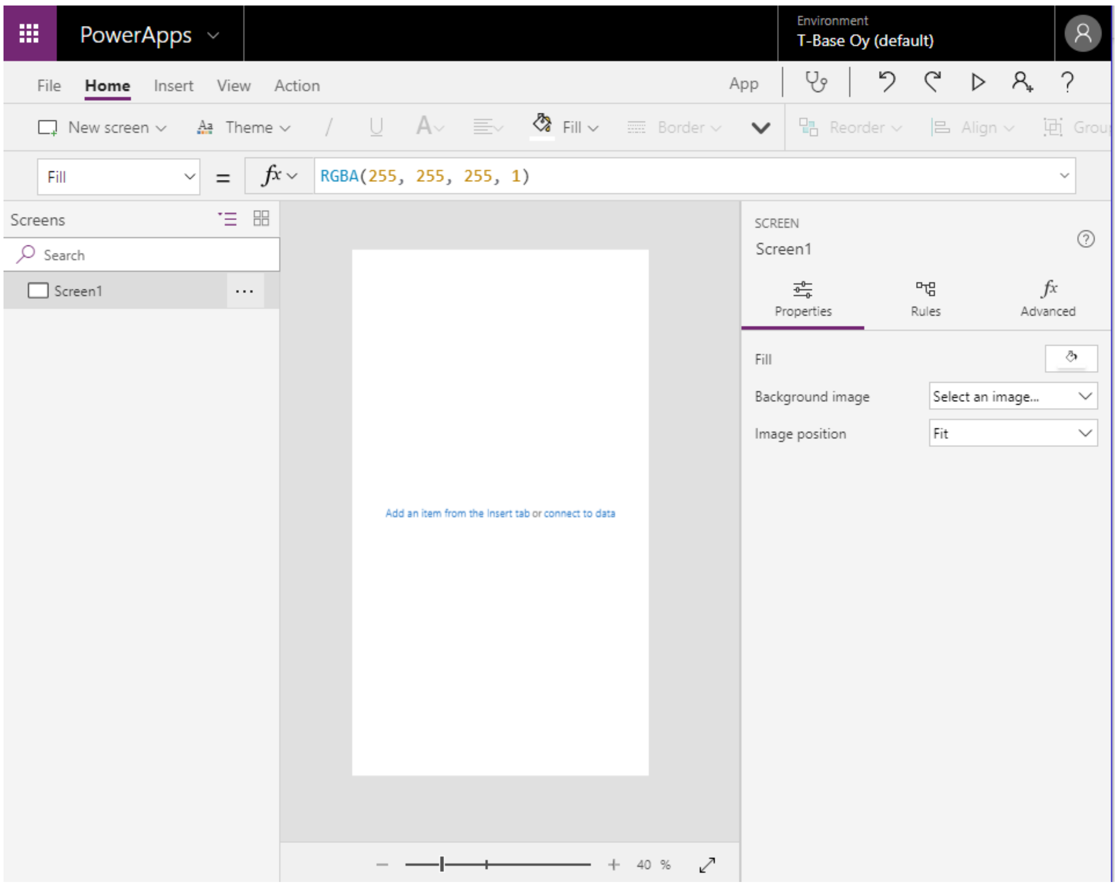
After the variables are declared, NOCOUNT is set to ON at row number 11, stopping the procedure from tracking the affected rows. As the procedure returns a single value, counting the affected rows is pointless. A temporary table is then declared at row 14 of the procedure, which contains columns for an identifier number, name, and binary data. Then, at rows 21-26, data from the image table is retrieved, comparing the GUID-value of the received variable with the ones stored in the table. The matched columns data is then stored into the temporary table. The `Base64Data`-variable then has its value set to the temporary table’s binary data columns value and has its data converted to Base64 with the FOR JSON AUTO -operation at rows 28-35. This operation converts all binary type data in a table into Base64 format. This also changes the datatype to varchar(MAX), which means the data is of text type with no limits in length. At row 38, the image data text has an additional text snippet added in front of it, marking it as Base64 image data.

The `Nimi`-variable is then set at rows 41-44 as a combination of its identifier number and the reclamation/anomaly it is related to. With the procedure finished, the database has everything required for the development of the mobile application and the implementation of the workflow.

5 MOBILE APPLICATION

As can be seen from the specifications, the solutions have very similar implementations from a technical point of view. Both applications require text input and the ability to control photos. After the required data exists in their respective tables in the database, the form applications can have their data gathering functionality and user interface implemented. Lastly, the data processing can be implemented as it is dependent on both the database’s and mobile application’s functionalities. As the client did not require any specific software test plans from the commissioner, both solutions will undergo simple ad-hoc-testing, making sure that the data and images are stored in the intended locations in readable form, and that the mobile applications can be considered usable. The actual software testing will be made by the client.

The development begins by starting the PowerApps development environment from Microsoft’s web service and creating a blank application with default settings for mobile, which leads to a blank screen (Figure 5) that has the default aspect ratio of a typical mobile device.



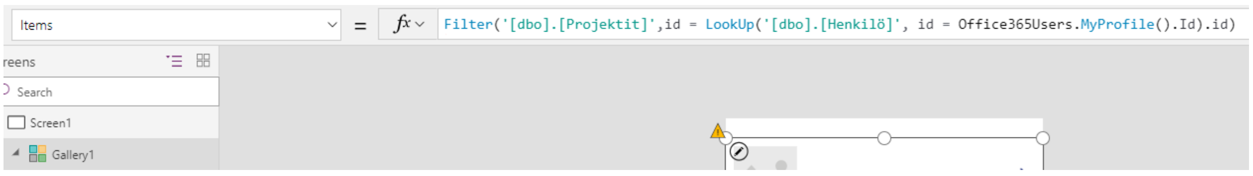
The blank screen represents what the user sees when he/she opens the mobile application. All elements that are implemented into the application are displayed in the left-hand side of the screen. Below the section selection bar is the formula bar, which as previously mentioned, is used for all logical functionality in PowerApps.

5.1 Project selection and navigation

The first feature that both applications need is the project selection screen. Both solutions' data are related to a project, which means that a project must be chosen before data related to it can be gathered. Since the project selection is the first aspect that the user sees upon opening the application, it is very important that the projects are loaded immediately. The projects can be retrieved from the SQL database with the use of a data source connector that can be added to the application from the Insert-tabs‘ Data Sources-section. With the data connector, it is possible to use data tables and views from the database, as data sources in PowerApps.

The projects ,however, need to be presented to the user somehow. This is achieved by implementing a Gallery component into the project selection screen. The Gallery component displays data from online data sources or local data collections from within the application itself. The data is displayed by showing one row of data in one element. The layout of the element is editable and duplicated to each row of the component. By setting the Gallery's Items-property to the project data source that was just retrieved, a list of project elements made from all the projects will be shown to the user.

This leads to the required feature having only the user's own active projects visible. Fortunately, PowerApps' formula system allows the filtering of data sources and collections. As projects always have a set of people in charge whose ID numbers are stored in the personnel data table and connected to the projects, it is possible to check whether the user using the application is associated with a project or not. This requires the implementation of an additional data source connector, this time from the Office365 Users service. The connector provides information regarding the user that is currently using the application, such as name, e-mail, and a unique user identifier. By implementing a Filter-function (Figure 6) that contains a LookUp-function into the project gathering formula, it is possible to compare the user identifier to the ones that are stored in the personnel table, retrieve the user’s personnel identifier number, and gather the projects which associate that number. The function returns the filtered projects.

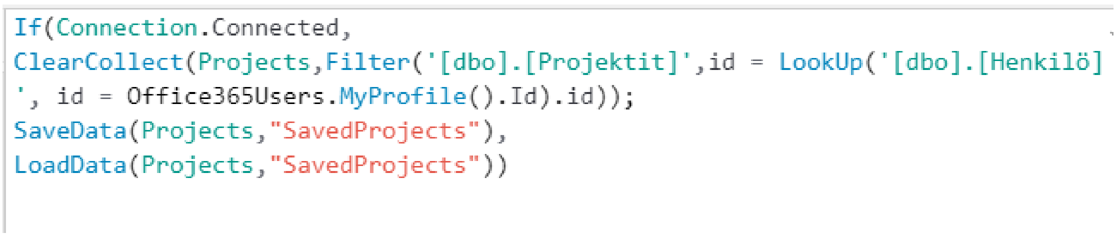


```
Filter('[dbo].[Projektit]',id = LookUp('[dbo].[Henkilö]', id = Office365Users.MyProfile().Id).id)
```

Figure 6: Project filtering function

However, the added feature requirement of having the projects available without having an Internet connection makes it impossible to retrieve the projects from the database every time the form is run. Fortunately, the results of the Filter-function can be inserted into a collection and saved into the user’s mobile device from where it can be accessed without an Internet connection. The projects are gathered into a collection named “Projects” using a ClearCollect-function that creates a collection while clearing it if one with the same name already exists. It is also possible to collect conditionally gathered data, such as the results of Filter. The collection formula is inserted into the project screen’s OnStart-property, making it run as soon as the application is opened. After the collection has its data, the collection can be saved into the mobile devices’ memory using the SaveData-function. The function requires a collection to save and a name for the saved file as its parameters. For the data to be available when the user is offline, it needs to be loaded when the device has no Internet connection. PowerApps has a Connection.Connected-function that can differentiate whether the device is online or not. By wrapping the entire project gathering function inside an If-function, as is done in Figure

7, projects can be gathered from the database, or the device's local memory, depending on the connection.



```
If(Connection.Connected,
ClearCollect(Projects,Filter('[dbo].[Projektit]',id = LookUp('[dbo].[Henkilö]
', id = Office365Users.MyProfile().Id).id));
SaveData(Projects,"SavedProjects"),
LoadData(Projects,"SavedProjects"))
```

Figure 7: The formula used in gathering the user's projects

The data in the database, however, is never static. Changes can happen several times, even during user sessions. A refresh-button for gathering projects again is required. By inserting an icon that represents reloading (Figure 8) and inserting the same startup formula without the If-clause, a refresh button is made. By setting its Visibility-property dependent on Connection.Connected, it can be made sure that the user does not attempt to scan for projects when there is no Internet connection. In order to display the projects to the user, the Gallery has its Items-property set as "Projects". This way the projects are always up to date, and at the same time, available without an online collection.

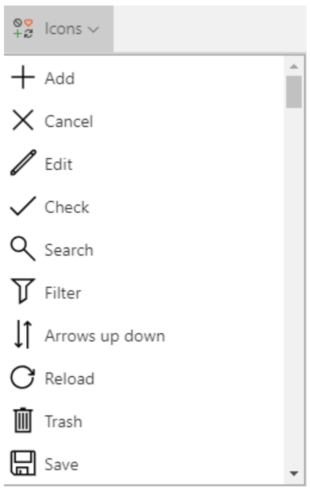
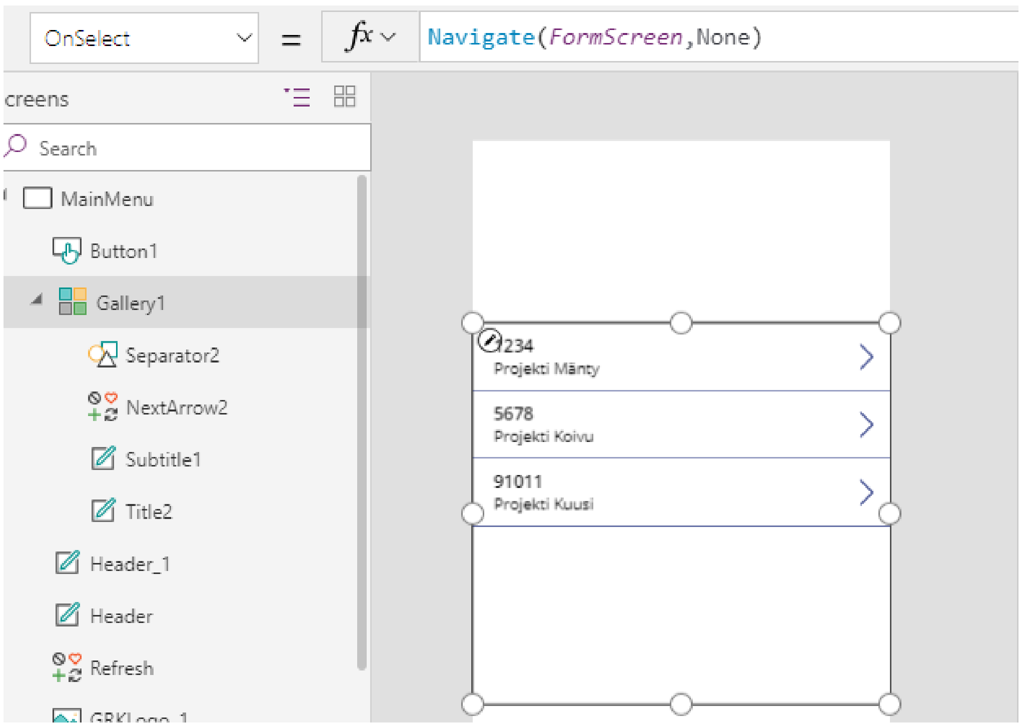


Figure 8: PowerApps's icon selection.

The user needs to navigate to the form page by selecting a project from the project selection menu. By setting the Gallery's OnSelect-property to call the Navigate-function (Figure 9), the user can navigate to the form screen by selecting a project element. In addition to the screen that the user has navigated to, a transition animation type needs to be given as a parameter. In these solutions, the None-parameter is used, which results in no animation in moving between the screens.



```
OnSelect = fx Navigate(FormScreen,None)
```

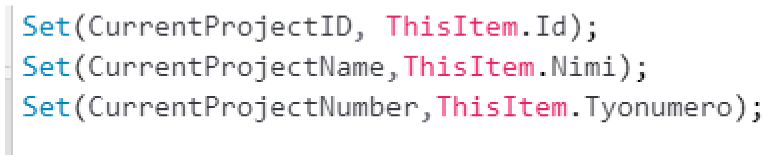
Figure 9: Navigate-function in a Gallery-component

Whilst in the form screen, the user sometimes needs to navigate back to work on another reclamation or anomaly. A Button-element with a similar Navigate-function attached to its OnSelect-property makes for a return button when placed at the top right-hand corner of the screen.

5.2 Displaying project data

While only the database identifier number of the project is necessary in the data gathering process, in order preserve the feeling of filling out a form, some of the project data will be displayed on the top of the data gathering form page. According to Nielsen and Budiu (2012, 113) ,information that does not provide sufficient value-added should be omitted. In order to avoid overbearing the user, only the project’s name and number are shown. The project data is displayed using PowerApps’ Label-component. With the displayed information having to change depending on the selected project, the labels need to have their content manipulated. Unfortunately, PowerApps does not allow manipulating a components properties through another component’s formula.

This issue can be bypassed with the use of variables. As variables are global, they can be modified from any component’s formula. Setting the Text-property of the label to a variable named e.g. `CurrentProjectName`, the text of the Label component changes according to what the variables value is, provided it is compatible with the component which in this case is text. By returning to the project menu and inserting additional Set-functions into the OnSelect-property of the Gallery in the project menu, the project’s name and number can be taken from the collection that the gallery uses by referring to its ThisItem-properties which are column values from the collection that was created for the gallery to display. By having the variables introduced in the formula (Figure 10) as the default value of the labels, the texts change dynamically depending on which project was chosen.



```
Set(CurrentProjectID, ThisItem.Id);
Set(CurrentProjectName,ThisItem.Nimi);
Set(CurrentProjectNumber,ThisItem.Tyonumero);
```

Figure 10: The updated Gallery formula

5.3 Data input

The main functionality of the form consists of input fields and controls with which the user will be able to gather data of the reclamation or safety anomaly at hand. The forms require several text inputs and a control-component which to add images with. In addition, the anomaly form needs a dropdown-component for the anomaly type selection.

From the Insert-tabs’ Text-section, TextInput-components can be added to the application. These components take text input from a user. The input fields are titled with Label-components to inform the user what to input in the field. According to Nielsen and Budiu (2012, 102), it is 108 percent harder to understand information when reading from a mobile screen. In order to clarify to the user what to input, the InputField-components have a HintText-property (Figure 11) that can give the users further information as to what to input in the field. With additional labeled TextInput-components, the required text data can be gathered into three fields.

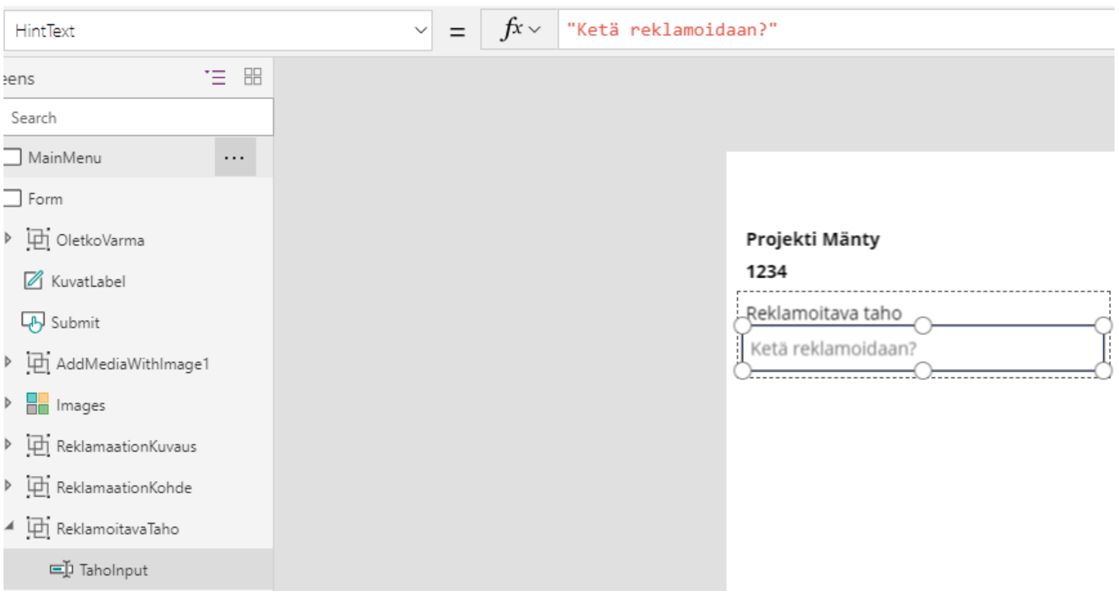
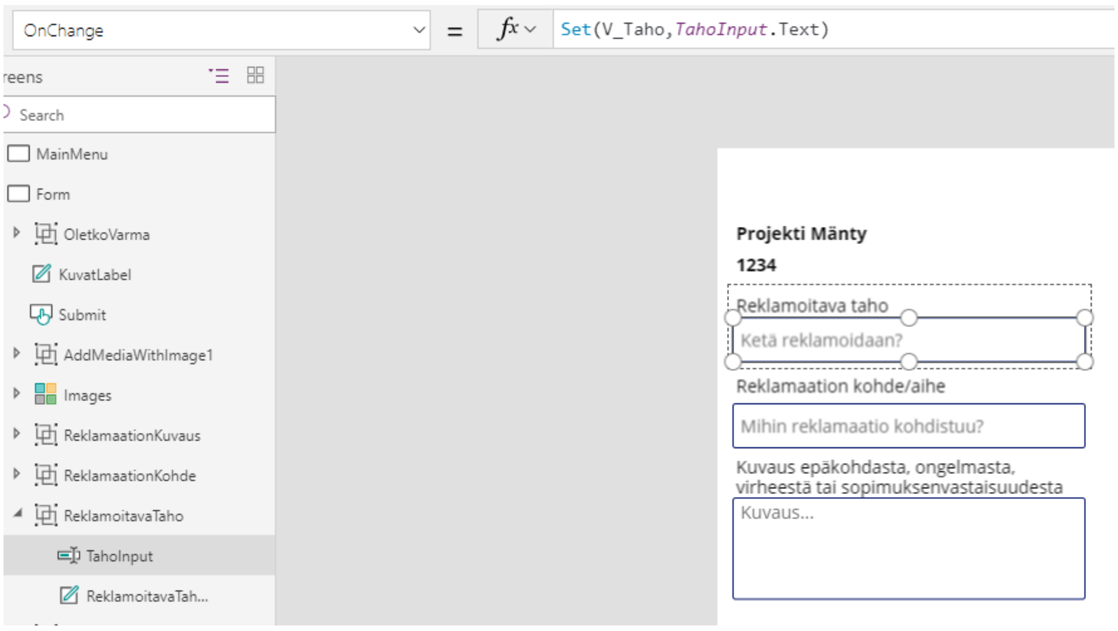


Figure 11: Hint text-property of an Input-component

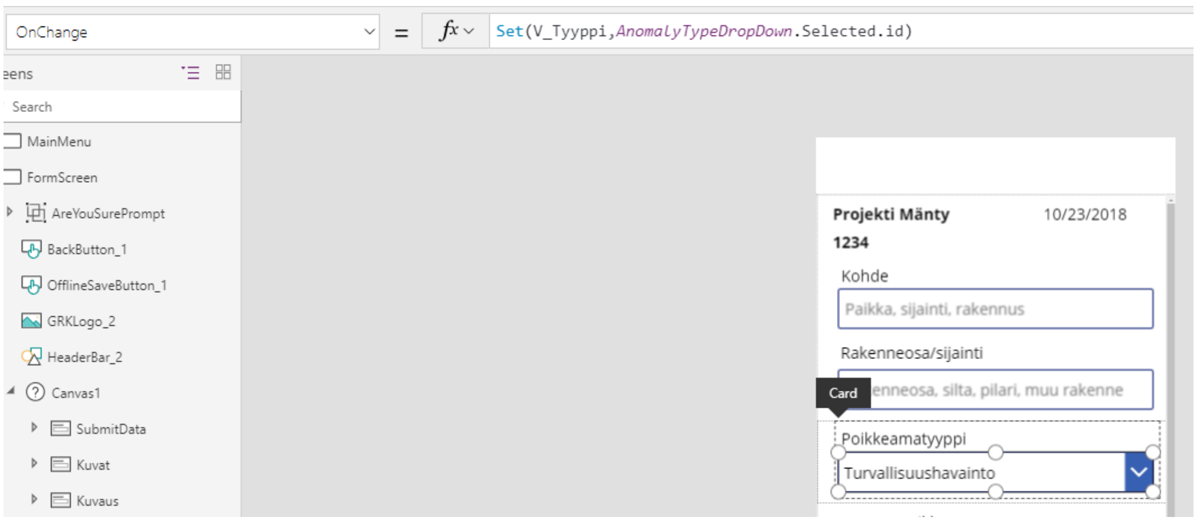
The fields, however, serve no use without some logic applied to them. In order to process the text input into them, their values must be stored in variables through which they can later be gathered into collections. TextInput-components have an OnChange-function that is called each time the user inputs or removes text from in the component. With the use of the Set-function, the current value of an input field can be inserted into a variable. In Figure 12, the TextInputs value is inserted into a variable named `V_Taho`.



```
Set(V_Taho,TahoInput.Text)
```

Figure 12: Changing a variable based on an Input-components value

The DropDown-component that is used for the anomaly type selection is similar to the Gallery-component in that it uses collections as its Item-property, producing the selectable options based on the data in the collection. Its logic operations are identical to the TextInput-component. A Set-function inserted into the OnChange-input (Figure 13) covers the required functionality. The Items-property is set to a collection that will be gathered in the OnStart-function. The collection consists of identifier numbers and the names of the anomaly types.



```
Set(V_Tyyppi,AnomalyTypeDropDown.Selected.id)
```

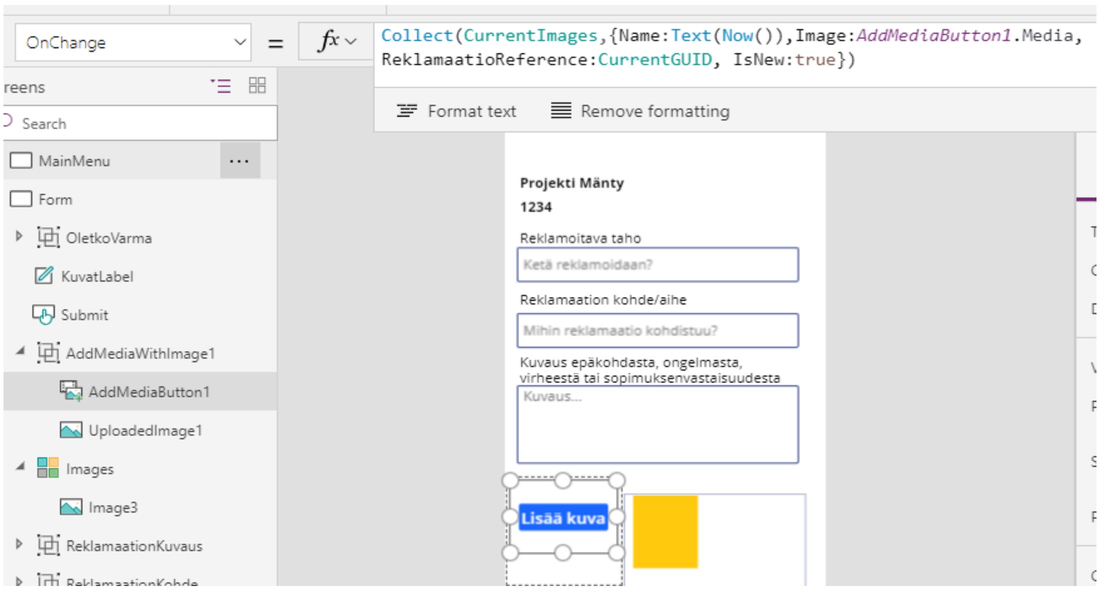
Figure 13: The DropDown-component manipulates the selected type ID number depending on the anomaly type chosen.

The Default-property of the selection is set to `V_Taho`, allowing the dropdown selection to persist even when the user navigates to another screen. This will also make sure that the dropdown remembers the user’s choice through multiple sessions with the requested feature of continued sessions.

5.4 Image controls

One of the factors making data gathering and filing reports on a mobile device unique is the ability to add images along the data to visualize the situation at hand. PowerApps has two ways to add images; a direct camera control component and an image control component. The camera control can be embedded directly to the screen, allowing to take pictures without leaving the form application. The image control uses the native camera application of the mobile device. The image control was chosen to be used in both applications as it is more versatile, and the native camera application will better utilize the device’s camera than the embedded application component.

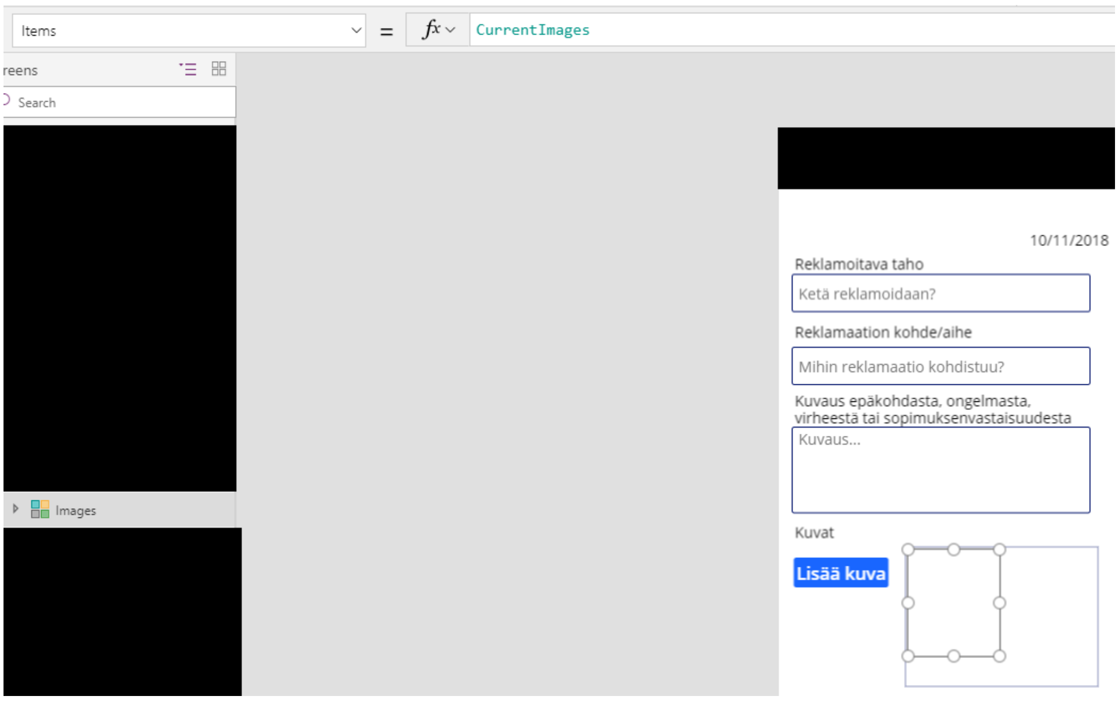
In addition to taking pictures, the user needs to be able to see a list of the taken pictures and be able to remove them from the list. A Gallery-component and a collection will solve both problems. By inserting a Collect-function to the image controls OnChange-property as displayed in Figure 14, the image can be put into a collection after the user has taken it. The image is collected as a blob reference text. The blob reference can be displayed as an image in the application but does not function in other environments.



```
Collect(CurrentImages,{Name:Text(Now()),Image:AddMediaButton1.Media,
ReklamaatioReference:CurrentGUID, IsNew:true})
```

Figure 14: The user can add related items and see them displayed in the gallery as they will be included along with the other data.

Along with the image data, name, unique identifier, that is unique to each reclamation and anomaly, and a boolean value indicating whether the image is new are collected. As unique image names are not mandatory but favorable, the name is given based on the device's clock with the Now()-function. The time value received from Now() is cast into text by encasing it inside Text()-function. The unique identifier is a variable that is unique to a reclamation or a safety anomaly and is used in associating the images to the correct reclamation or anomaly. The gallery in Figure 15 uses the CurrentImages-collections in which the images are collected.


Items = CurrentImages

reens

Search

[redacted]

Images

[redacted]

[redacted]

10/11/2018

Reklamoitava taho

Ketä reklamoidaan?

Reklamaation kohde/aihe

Mihin reklamaatio kohdistuu?

Kuvaus epäkohdasta, ongelmasta, virheestä tai sopimuksenvastaisuudesta

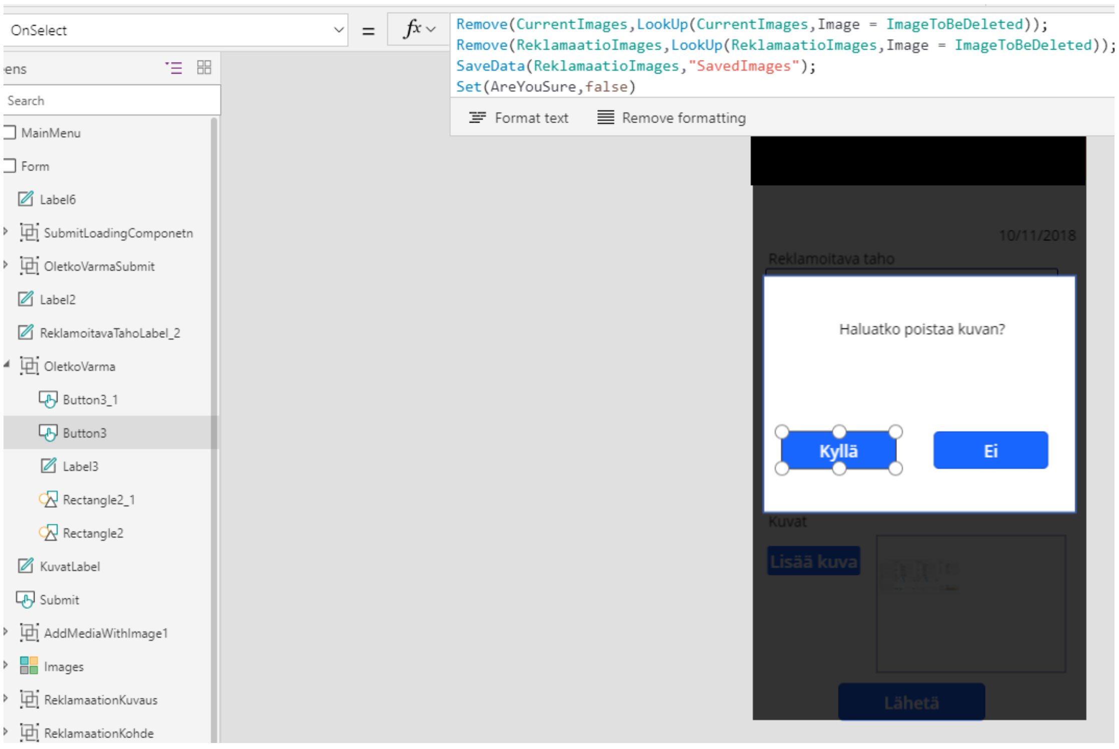
Kuvaus...

Kuvat

Figure 15: Image gallery that uses a collection as its data source

The user needs to be able to delete the added images from the form as well. This process is implemented by setting a formula to the image gallery components OnSelect-property to delete the displayed image from the collection which deletes it from the image gallery as well. Implementing the formula like this, however, leaves a chance for accidental image deletion. The user can unintentionally remove an image if the image is accidentally touched. This can be avoided by implementing a Yes/No-dialogue that prompts the user to select whether he/she actually wants to delete the picture that was selected.

The dialogue consists of a dark see-through rectangle the size of the entire screen. It is put on top of the form, covering it entirely and stopping users from pushing other buttons. On top of it, another rectangle is inserted for the dialogue text and buttons. Inside the boundaries of the rectangle, a label that asks the user if he/she is sure that the image is to be deleted is added. Two buttons, saying Yes and No, are placed at the bottom of the dialogue containing the text. This results in a dialogue screen similar to the one in Figure 16. The components are placed into a group, allowing the manipulation of their visibility through a Group-component. The group's visibility depends on a variable named `AreYouSure`. The image gallery has its OnSelect-formula changed to simply setting the `AreYouSure`-variable to true, which displays the dialogue.



```
Remove(CurrentImages,LookUp(CurrentImages,Image = ImageToBeDeleted));
Remove(ReklamaatioImages,LookUp(ReklamaatioImages,Image = ImageToBeDeleted));
SaveData(ReklamaatioImages,"SavedImages");
Set(AreYouSure,false)
```

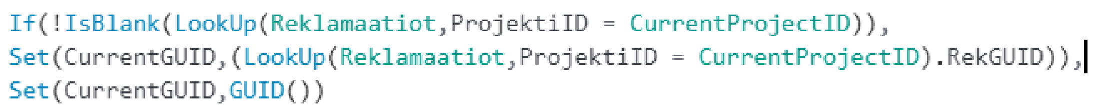
Figure 16: A Yes/No dialogue

In addition to displaying the dialogue, the button sets an `ImageToBeDeleted`-named variable to the unique identifier of the picture that is requested to be deleted which is taken from the ThisItem-properties of the image gallery. The No-button in the dialogue simply sets the variable back to false, hiding the dialogue again. The Yes-button now manages the deleting by locating the from the collection, based on the `ImageToBeDeleted`-variable, and hides the dialogue.

5.5 Unique identifiers

In order to keep track of which images belong to which reclamation or safety anomaly they need a common unique property. PowerApps allows the creation of GUIDs with the GUID-function. A GUID is a 128-bit long string of data that translates to 32 characters that can be considered almost perfectly unique. In a conversation thread with several programming enthusiasts in the website Stack Overflow in 2018, it was deduced that there is a greater chance for a person to be hit by a meteorite than for two GUID-values to be identical.

The reclamations and safety anomalies are given a GUID when the user selects a project (Figure 17), which is why the GUID is inserted into the project gallery's OnSelect-function. If one already exists in the local data, it is loaded from an already existing one's GUID-column. Otherwise the GUID is generated with the GUID-function. The images are given this same GUID when they are added into the form. This way when the reclamation or anomaly is inserted into the database, the application can tell which images are to be patched along with it.



```
If(!IsBlank(LookUp(Reklamaatiot,ProjektiID = CurrentProjectID)),
Set(CurrentGUID,(LookUp(Reklamaatiot,ProjektiID = CurrentProjectID).RekGUID)),
Set(CurrentGUID,GUID())
```

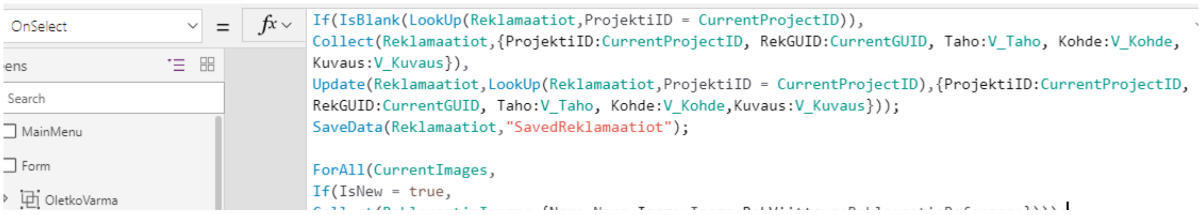
Figure 17: The GUID creation formula

Without a common GUID, an image and a reclamation/anomaly cannot be associated with each other in the mobile application. This would also mean that there would be no way to associate them in the database or reports either.

5.6 Continued sessions

Another notable requirement from the client was the ability to fill out the forms throughout multiple sessions. This means that the data the user inputs in the form application cannot be disposed of when the user closes the application. Everything that the user inputs into the form needs to be saved into the device's memory and loaded back the next time the user opens the application. The SaveData-function that was used in gathering the projects can be used to save the user-inputted data as well. Unfortunately, there is no way to call any functionality when exiting the application. This means that the saving must be done manually by the user, in this case with the use of a save button. The button itself is a simple Button-component added from the Insert-tab. The same button is used for saving everything related to the reclamation or anomaly that the user has input. The saving of the images is done separately due to having to process them when they are inserted into the database.

First, the collection in which the data is gathered needs to be verified on whether data has already been saved on the same reclamation/anomaly or not. Then, either the user data is collected as a new row, or the existing row that has the matching project identifier number has its data updated. The related project's identifier number is taken from the variable that is also used in displaying the project information. The variables that the Input- and Dropdown-components manipulate and the GUID that was given at the start are gathered as well. The entire collection is then saved to a local file, named "SavedReklamaatiot". All of the images in the form that are considered new with the use of the `IsNew`-boolean that was set when the image was taken are gathered into the projects. This is done to avoid duplicate images. The image collection is then saved into a local file as well. The entire process is visualized in Figure 18.

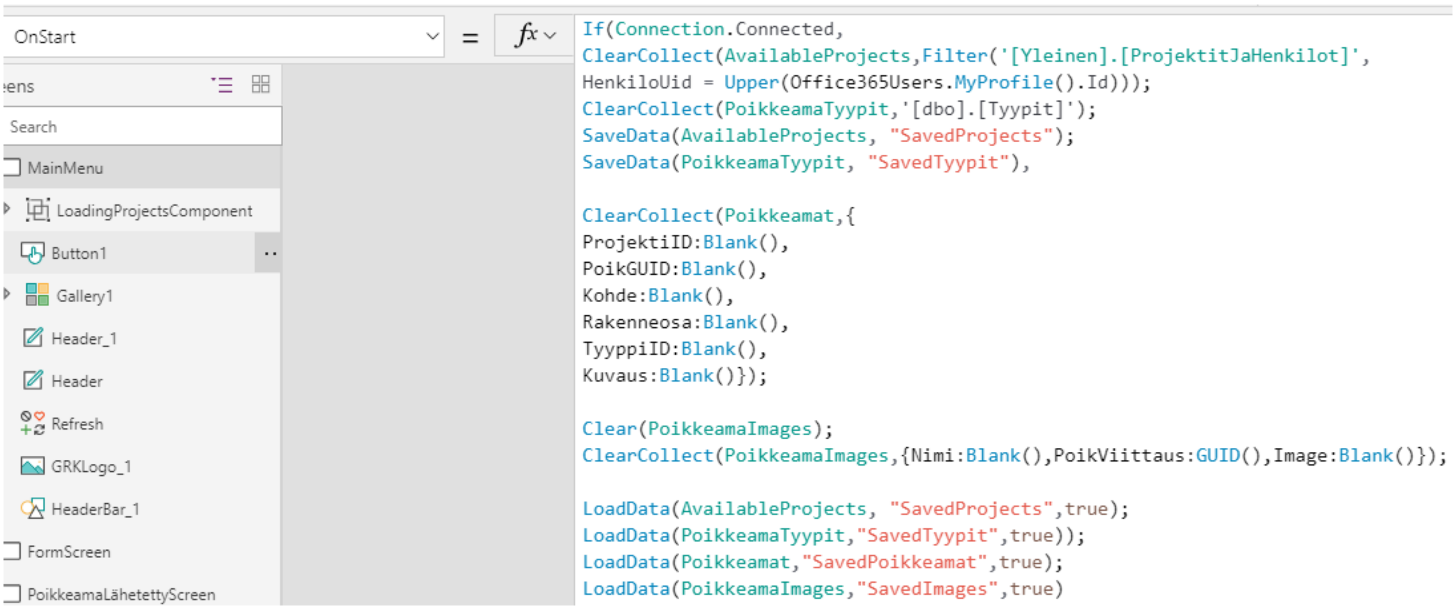


```
If(IsBlank(LookUp(Reklamaatiot,ProjektiID = CurrentProjectID)),
Collect(Reklamaatiot,{ProjektiID:CurrentProjectID, RekGUID:CurrentGUID, Taho:V_Taho, Kohde:V_Kohde,
Kuvaus:V_Kuvaus}),
Update(Reklamaatiot,LookUp(Reklamaatiot,ProjektiID = CurrentProjectID),{ProjektiID:CurrentProjectID,
RekGUID:CurrentGUID, Taho:V_Taho, Kohde:V_Kohde,Kuvaus:V_Kuvaus}));
SaveData(Reklamaatiot,"SavedReklamaatiot");

ForAll(CurrentImages,
If(IsNew = true,
```

Figure 18: The data saving formula used in the Save-button

After the data is saved, a load functionality is implemented. These operations are best done with the OnStart-function immediately when the application is opened. Two LoadData-functions that load the previously saved local files back into the collections will ensure that the saved data is available.

In addition to saving and loading user data, the anomaly form requires the data used in selecting a person responsible for handling the anomaly loaded into the local files so that this data is available without an Internet connection. The process is executed alongside the loading of projects (Figure 19), wrapped in the same If-function. This leads to the personnel data being managed identically to the project data. If there is a connection, the data is brought from the database and saved into the collection. Otherwise it is loaded from the device's memory.



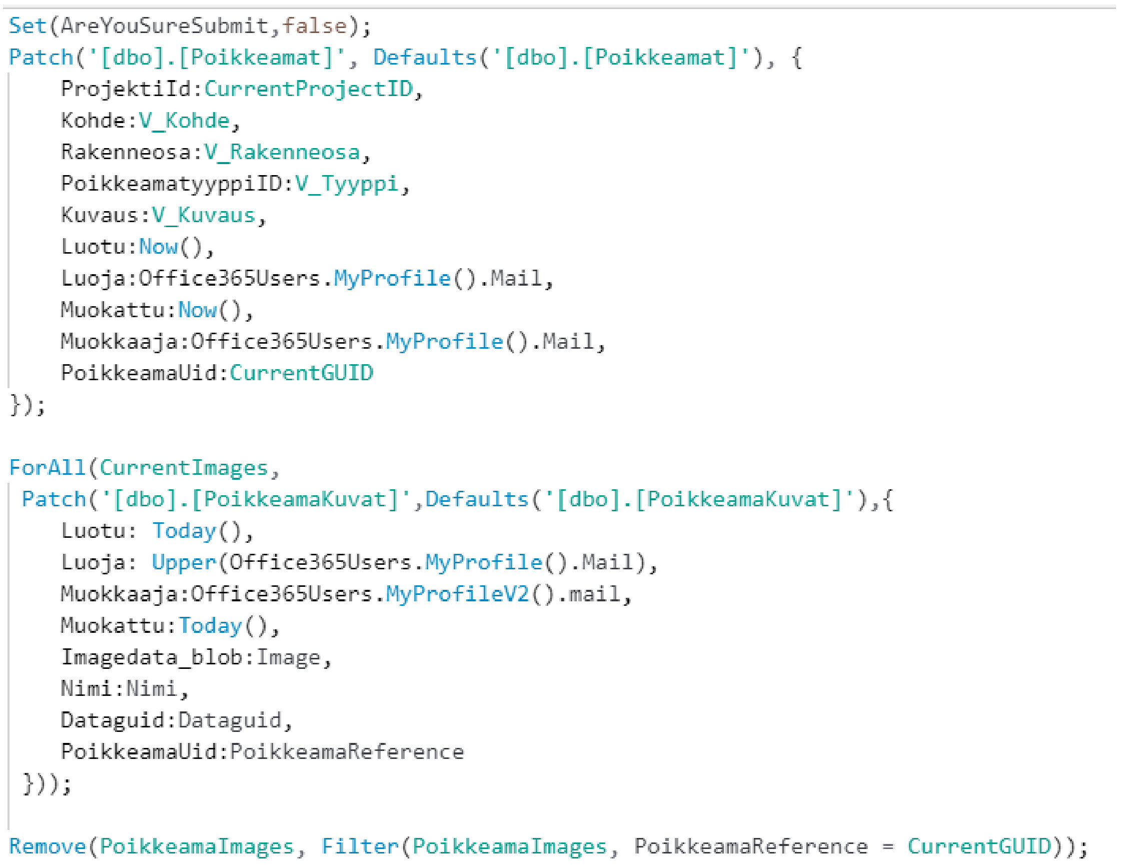
```
If(Connection.Connected,
ClearCollect(AvailableProjects,Filter('[Yleinen].[ProjektitJaHenkilot]',
HenkiloUid = Upper(Office365Users.MyProfile().Id)));
ClearCollect(PoikkeamaTyypit,'[dbo].[Tyypit]');
SaveData(AvailableProjects, "SavedProjects");
SaveData(PoikkeamaTyypit, "SavedTyypit"),

ClearCollect(Poikkeamat,{
ProjektiID:Blank(),
PoikGUID:Blank(),
Kohde:Blank(),
Rakenneosa:Blank(),
TyyppiID:Blank(),
Kuvaus:Blank()});

Clear(PoikkeamaImages);
ClearCollect(PoikkeamaImages,{Nimi:Blank(),PoikViittaus:GUID(),Image:Blank()});

LoadData(AvailableProjects, "SavedProjects",true);
LoadData(PoikkeamaTyypit,"SavedTyypit",true));
LoadData(Poikkeamat,"SavedPoikkeamat",true);
LoadData(PoikkeamaImages,"SavedImages",true)
```

Figure 19: The data loading formula that is run when the application starts.

Since the data mostly consists of numbers and text, it is not expected to take significant amounts of space from the mobile device’s internal memory. However, in order to save data, the data is intended to be deleted when it is submitted into the database.

5.7 Submitting data

With the data input being complete, the data can now be submitted into the database. A Submit-button works best at the bottom of the page. In order to keep users from attempting to submit their data in offline-mode, the buttons Visible-property is set to Connection.Connected, hiding it when the user’s device has no Internet connection. In order to avoid accidental data submits, a Yes/No-prompt similar to the one that is used in managing the images is implemented. The Submit-button sets a variable that the dialogues visibility is dependent on to true, which then displays the dialogue. The No-button sets it back to false, hiding it. The Yes-button runs two Patch-functions (Figure 20).



```
Set(AreYouSureSubmit,false);
Patch('[dbo].[Poikkeamat]', Defaults('[dbo].[Poikkeamat]'), {
    ProjektiId:CurrentProjectID,
    Kohde:V_Kohde,
    Rakenneosa:V_Rakenneosa,
    PoikkeamatyyppiID:V_Tyyppi,
    Kuvaus:V_Kuvaus,
    Luotu:Now(),
    Luoja:Office365Users.MyProfile().Mail,
    Muokattu:Now(),
    Muokkaaja:Office365Users.MyProfile().Mail,
    PoikkeamaUid:CurrentGUID
});

ForAll(CurrentImages,
 Patch('[dbo].[PoikkeamaKuvat]',Defaults('[dbo].[PoikkeamaKuvat]'),{
    Luotu: Today(),
    Luoja: Upper(Office365Users.MyProfile().Mail),
    Muokkaaja:Office365Users.MyProfileV2().mail,
    Muokattu:Today(),
    Imagedata_blob:Image,
    Nimi:Nimi,
    Dataguid:Dataguid,
    PoikkeamaUid:PoikkeamaReference
 }));

Remove(PoikkeamaImages, Filter(PoikkeamaImages, PoikkeamaReference = CurrentGUID));
```

Figure 20: The anomaly form's data submit formula.

The Patch-function can create new rows to a data source if it is allowed to do so. The Patch-function requires the reclamation/anomaly table as a parameter so that it would know in which data source to create the new row. The user-input data is first patched into the database, followed by the related images. The submitted data and images are then removed from the mobile device by locating the appropriate anomalies/reclamations and images based on their unique identifiers.

6 DATA PROCESSING

Once the user data is in the database, it will remain there for reports and other uses as it needs no processing. The images, however, are intended to exist only temporarily in the database. With modern mobile devices being capable of taking pictures of extremely high quality, their size will bloat the database. This means they need to be transferred to another place.

6.1 Image processing solution

The images are taken from the database and stored into a SharePoint library in the cloud by implementing a Microsoft Flow workflow. The reason for storing the images in the database temporarily is based on a compatibility issue. Blob reference images are bound to PowerApps, which means they cannot be used to post images directly to SharePoint as they are not yet in image format. Another issue comes up when the images are posted into Flow as they continue to remain in Blob form even when passed to a workflow. Posting them into the database, however, turns them into binary data that instead of being a reference, represents the image that the user took with the mobile device’s camera. The binary data can then be converted into Base64 data and used in Microsoft Flow to process and post the image into SharePoint

6.2 Saving images with Microsoft Flow

Once logged into Microsoft Flow, a flow needs to be created from blank as the templates Microsoft provides are not useful for this kind of use. The entire process consists of four Flow-components (Figure 21). First, the flow requires a trigger which refers to an action or an event in an application or software that that activates the flow. Since the images are saved in the database in their own table, it is possible to connect the database to Microsoft Flow and have the flow activate when a new row is created in the table. Another potential trigger would have been in PowerApps, as flows can be triggered from the application. Setting the trigger to the database takes additional load away from the form application and lessens the impact of runtime errors by making PowerApps not responsible in any manner for the image processing.

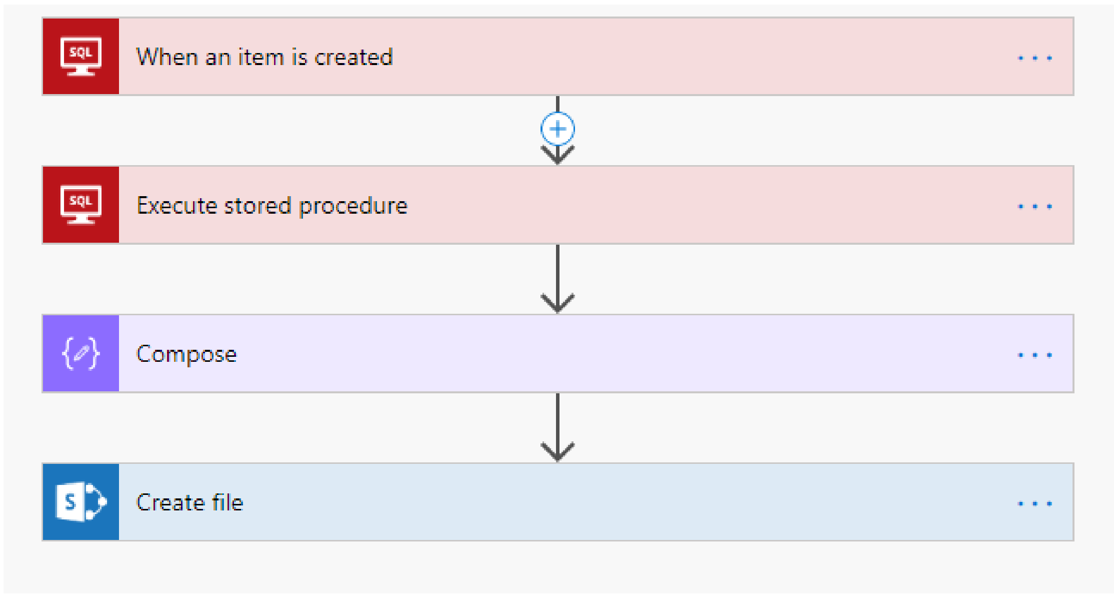


Figure 21: The process of processing a saved image, and saving it into SharePoint, implemented in MS Flow.

After the flow is triggered, it calls the SQL server where the images are kept to perform the stored procedure that was implemented when the database was prepared. The procedure finds the same row of data that was just added and converts the binary data of the image into Base64-format and returns it to Flow.

With the Base64 image data available, a built-in data operation named “Compose” is called. The data operation turns the Base64 string back into a binary string for SharePoint to create a file of it. Afterwards, the converted image can be posted with Flow’s SharePoint connector’s “Create File”-action into a SharePoint site’s image library.

7 CONCLUSION

Both the reclamation reporting application and safety anomaly application were put through ad hoc-testing and were considered functional. Ad-hoc is a form of a black box or behavioral testing performed without any formal process in place (Software Testing Help 2018). The test consisted of simply booting up the PowerApps mobile application, filling up both forms while making sure to include several pictures with them, and submitting the data.

The mobile applications saved the user-input data through multiple sessions and the data was stored successfully into the database with the images available in SharePoint as can be seen in Figure 22. The solutions were set for deployment to the customer according to schedule.

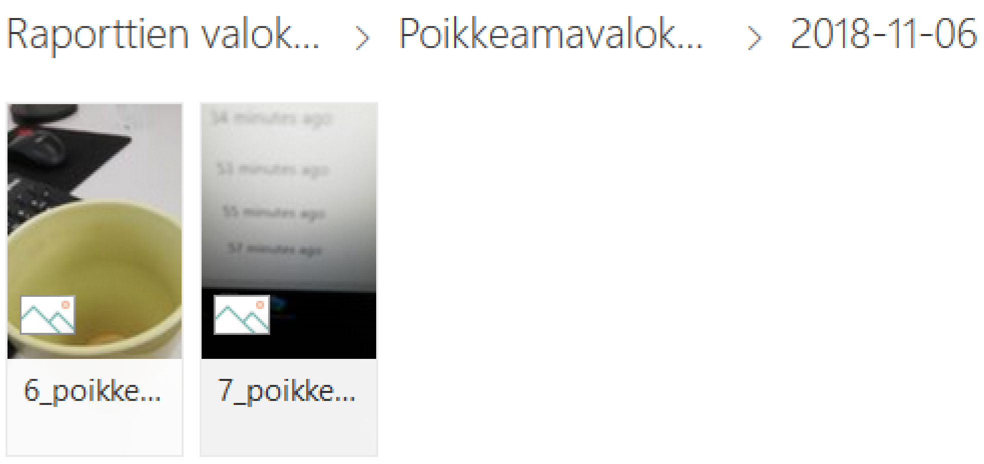


Figure 22: The images taken by the user are now in a SharePoint directory.

The development cycle of the solutions proved to be much faster and smoother than expected. Based on the author’s personal experience, the developing of mobile applications on PowerApps has been difficult due to the restrictions it sets on logical operations and its lack of an actual programming language. A factor regarding the swift development was most likely that a much more complicated PowerApps-system was already being deployed to the customer at the time this study started. This made developing smaller, similar solutions significantly easier.

REFERENCES

Ad-hoc Testing: How to Find Defects Without a Formal Testing Process. 2018. Software Testing Help. WWW-document. Updated 7 Jul 2018. Available at: https://www.softwaretestinghelp.com/ad-hoc-testing/ [Accessed 21 Nov 2018]

Create and update a collection in your app. 2018. Microsoft. WWW-document. Updated 30 Nov 2015. Available at: https://docs.microsoft.com/en-us/powerapps/maker/canvas-apps/create-update-collection [Accessed 10 Nov 2018]

Desai, D. 2016. Announcing general availability of PowerApps. Blog. Available at: https://powerapps.microsoft.com/en-us/blog/announcing-general-availability/ [Accessed 6 Nov 2018]

Frappier, M & Habrias, H. 2006. Software Specification Methods. E-book. John Wiley & Sons.

Get started with canvas-app formulas in PowerApps. 2016. Microsoft. WWW-document. Updated 26 Apr 2016. Available at: https://docs.microsoft.com/en-us/powerapps/maker/canvas-apps/working-with-formulas [Accessed 10 Nov 2018].

Is it safe to assume a guid will always be unique?. 2018. Stack Overflow. WWW-document. Available at: https://stackoverflow.com/questions/2977593/is-it-safe-to-assume-a-guid-will-always-be-unique [Accessed 13 Nov 2018]

Nielsen, J & Budiu, R. 2012. Mobile Usability. E-book. New Riders.

Release notes. 2018. Microsoft. WWW-document. Updated 31 Aug 2018. Available at: https://docs.microsoft.com/en-us/flow/release-notes [Accessed 10 Nov 2018]

SQL Server Management Studio (SSMS). 2018. Microsoft. WWW-document. Updated 16 Oct 2018. Available at:https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-2017 [Accessed 10 Nov 2018]

Understand canvas-app variables in PowerApps. 2018. Microsoft. WWW-document. Updated 6 Jul 2017. Available at: https://docs.microsoft.com/en-us/powerapps/maker/canvas-apps/working-with-variables [Accessed 10 Nov 2018]

What is PowerApps? 2018. Microsoft. WWW-document. Updated 17 Oct 2018. Available at: https://docs.microsoft.com/en-us/powerapps/powerapps-overview [Accessed 10 Nov 2018]

LIST OF FIGURES

Appendix 1

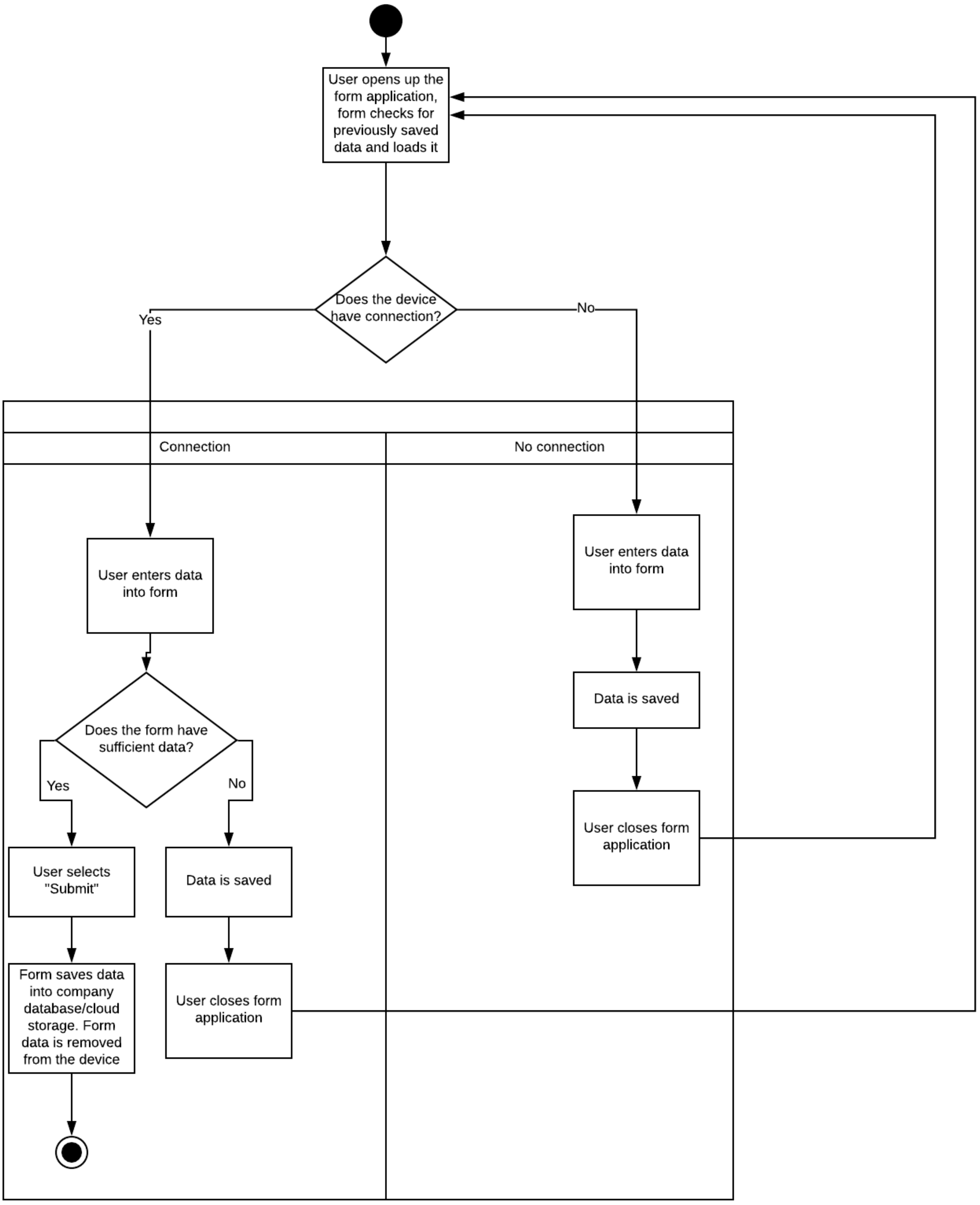
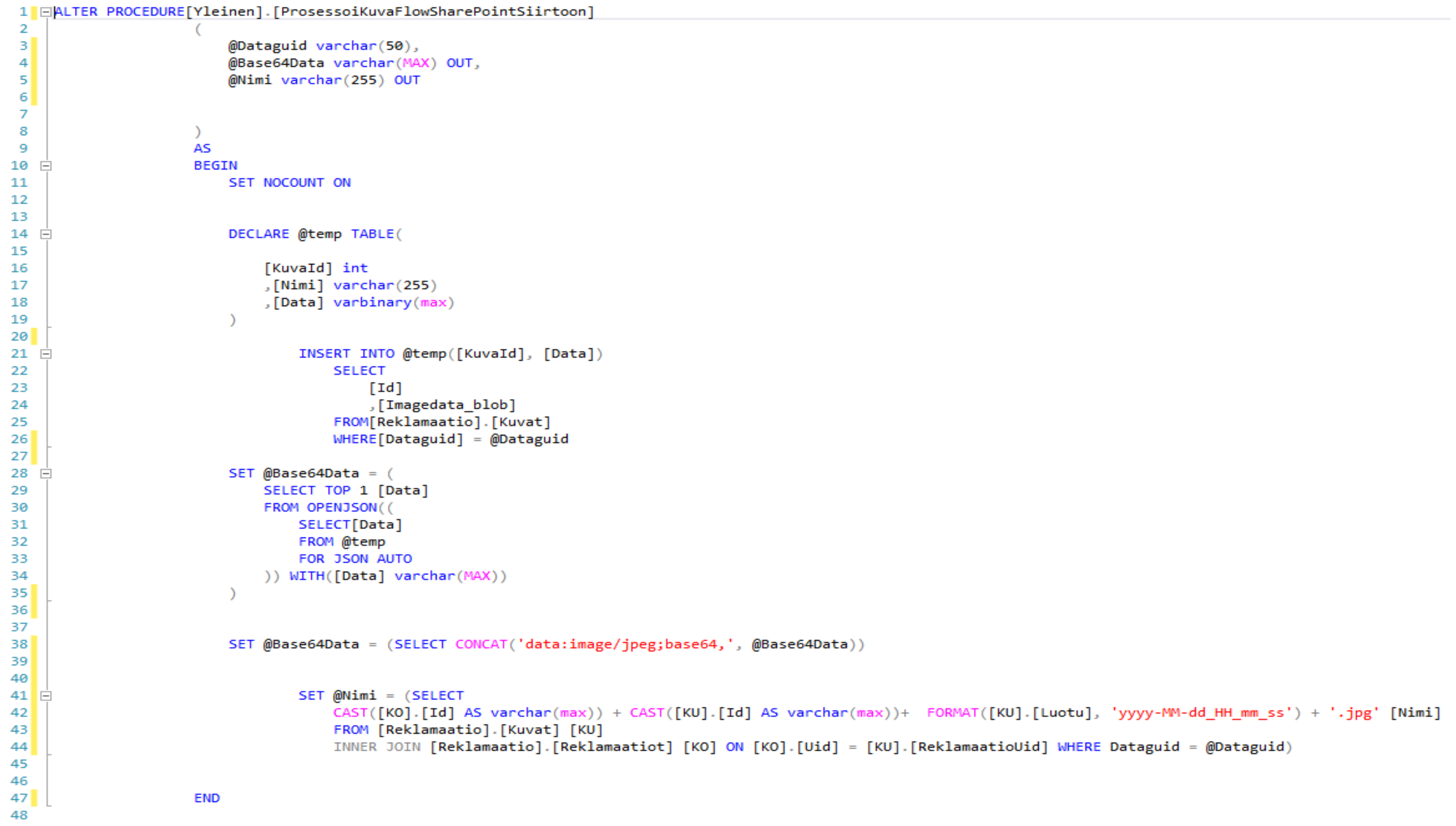

User opens up the form application, form checks for previously saved data and loads it
Does the device have connection?
Yes
No
Connection
No connection
User enters data into form
Does the form have sufficient data?
Yes
No
User selects "Submit"
Data is saved
Form saves data into company database/cloud storage. Form data is removed from the device
User closes form application
User enters data into form
Data is saved
User closes form application

Figure 1: The entire process of filling up a digital form, demonstrated in a UML Activity Diagram.

Appendix 2

Figure 1: A stored procedure built with SQL, used to convert and name images



```sql
ALTER PROCEDURE[Yleinen].[ProsessoiKuvaFlowSharePointSiirtoon]
                (
                    @Dataguid varchar(50),
                    @Base64Data varchar(MAX) OUT,
                    @Nimi varchar(255) OUT


                )
                AS
                BEGIN
                    SET NOCOUNT ON


                    DECLARE @temp TABLE(

                        [KuvaId] int
                        ,[Nimi] varchar(255)
                        ,[Data] varbinary(max)
                    )

                            INSERT INTO @temp([KuvaId], [Data])
                                SELECT
                                    [Id]
                                    ,[Imagedata_blob]
                                FROM[Reklamaatio].[Kuvat]
                                WHERE[Dataguid] = @Dataguid

                    SET @Base64Data = (
                        SELECT TOP 1 [Data]
                        FROM OPENJSON((
                            SELECT[Data]
                            FROM @temp
                            FOR JSON AUTO
                        )) WITH([Data] varchar(MAX))
                    )


                    SET @Base64Data = (SELECT CONCAT('data:image/jpeg;base64,', @Base64Data))


                            SET @Nimi = (SELECT
                                CAST([KO].[Id] AS varchar(max)) + CAST([KU].[Id] AS varchar(max))+  FORMAT([KU].[Luotu], 'yyyy-MM-dd_HH_mm_ss') + '.jpg' [Nimi]
                                FROM [Reklamaatio].[Kuvat] [KU]
                                INNER JOIN [Reklamaatio].[Reklamaatiot] [KO] ON [KO].[Uid] = [KU].[ReklamaatioUid] WHERE Dataguid = @Dataguid)


                END

```
]