Haaga-Helia
University of Applied Sciences

Implementation of a mobile business application built in Microsoft Power Platform

Heejin Moon

Bachelor's Thesis

Degree Programme in Business Information Technology

2020

Haaga-Helia
University of Applied Sciences

Abstract

8.5.2020

Author(s) Heejin Moon	
Degree programme Business Information Technology	
Report/thesis title Implementation of a mobile business application built in Microsoft Power Platform	**Number of pages and appendix pages** **26 + 5**

The thesis explores a relatively new field of business application creation. There are not many resources available on LCDP (Low-Code Development Platform) which is why it has been chosen as the topic for this paper. The thesis mostly discusses the resources available to practice LCDP, how they are applied in the real world, what are the benefits and limitations at their current stage, as well as what future advancements might get introduced later on. As demonstration, a real-life project is introduced mainly to serve as a working example.

Attention is given to the platforms and software utilised in the creation of the example project, namely Microsoft's Power Apps, Power Automate and Azure SQL Database, together with some alternative options. As the above mentioned three products are the most commonly used software in LCDP, the thesis will prove useful for those who are looking to learn more about the topic or are in search of more cost-efficient alternatives to traditional methods.

Furthermore, the thesis details the structure (front-end and back-end both) of a typical mobile business application project. There will be a description of technical know-hows and the real-life agile methodology of a project from a customer-developer relationship's point of view.

Finally, the thesis showcases functions and their deeper workings from the example application itself to show a hands-on parallel with the theory discussed in earlier chapters.

Keywords

Power Platform, Azure, Power Apps, Power Automate, SQL, LCDP, Mobile application

Table of contents

1. Introduction

The objective of this thesis is to introduce Power Apps and other related software as useful new work platforms for app developers, as well as to demonstrate the possibility of building a fully functioning mobile business application with minimal programming skills. For this end, the thesis will include a sample project taken from real life and refer to its lifecycle, design and other elements throughout the chapters to better enable the reader to understand how theory works out technically.

The following chapters discuss in detail a selection of tools and methodologies, including alternatives to the ones used in building the end product. After the thesis, the reader should have a good grasp on where to start with a new idea and how to implement his or her own business application. The thesis, however, will not provide a step-by-step guide on how the sample app was made.

The beneficiary of the thesis is anyone who is interested in modern app development, but mainly people with no coding skills but with great ideas an app, or companies, regardless of size, with restrictions on resources, such as a short time frame, low funds, no skilled and/or specialized personnel, etc. Whether these parties are looking for cheaper options for app development or are simply interested in the topic but do not know where to start, the thesis will provide useful to them.

It is important to note, that most of the tools mentioned in the thesis do not come for free. Certain paid subscriptions are necessary to access them and other services. However, all paid services here have been selected for maximum compatibility and with a tight budget in mind, so they should provide a reliable and cheap option for developers.

The foremost benefit of following the methods discussed in the thesis is overcoming the lack of highly trained professionals and expensive third-party vendors for the creation or acquirement of simple applications. As the user gets more familiar with the building tools, more and more functions become possible for future projects. Also, with the subscriptions in the example project, users have the option to select how many resources their product requires and usually the vendors will bill them accordingly. Other benefits include insight and a deeper understanding of other Microsoft products (e.g. SQL Server, Dynamics, etc.) how Power Apps links work, as well as cooperation with third-party (non-Microsoft) software (e.g. SAP, Salesforce, Pipedrive, Jira, etc.) with a comprehensive description to all of these and others.

1.1 Key concepts and terms explained

Power Platform: Refers to three of Microsoft's products: Power Apps, Power Automate and Power BI, of which Power Apps and Power Automate have leading roles in this thesis.

ERP: Enterprise Resource Planning is the integrated management of main business processes, often in real time, mediated by software and technology, in other words, using computer software to more efficiently carry out day-to-day business processes.

CRM: Customer Relationship Management helps companies build and maintain their relationships with existing or potential customers mainly through software-based means.

Dynamics 365: Microsoft's ERP system and part of the Microsoft Dynamics product line. Dynamics 365 collectively includes a variety of ERP and CRM applications.

On-demand: Service provided on-demand comes down to an agreement between two parties, where one buys the right to use the service or product which the other party offers for previously agreed periods of time. The agreement also includes the complexity of the service, often tailored to be exactly what the tenant party needs.

Azure Functions App / Azure Functions: A compute-on-demand service expanding the already existing Azure platform with additional capabilities. It allows the developer to connect multiple data sources and run event-triggered code on an existing infrastructure.

JSON: JavaScript Object Notation is the open standard file format that conducts communication between a browser and a server by transforming the data into human-readable text and vice versa. Essentially, it is a text format translation of any JavaScript object.

Binary data: The most basic data type in computing. Binary data is most often represented by 0s and 1s, following the binary numeral system and Boolean algebra. Binary files, unlike text files, are not human-readable.

Primary Key: The unique identifier in a database relationship model. If we consider a table with rows of data in it, the primary key will be represented by a column where every row contains a unique id with which all other rows on the same level can be quickly and easily identified by.

Foreign Key: The link between two tables in a database relationship model. Foreign keys are non-unique identifiers referring to another table's primary key, thus forming a logical connection between the two tables which the database software can recognize.

2 Structure and Tools

Below are described the key tools of the thesis regarding software and applications. These are provided to give a better overall understanding of how the application was built. Furthermore, this chapter will also discuss additional information regarding, for example, the backend development of the application, further insight into the creative choices, as well as alternative product options for the implementation of similar projects.

2.1 Overall structure of the application

Figure 1 below illustrates the three main components used to create the application. In no particular order these are:

- Power Apps, representing the frontend development part, where the user inputs, reviews and edits data through a comprehensible visual interface.
- Power Automation stands for backend as it automatically deals with the user inputs and executes the desired functions without the need of coding from the user's part.
- Azure SQL is the database where core data as well as new data get stored.

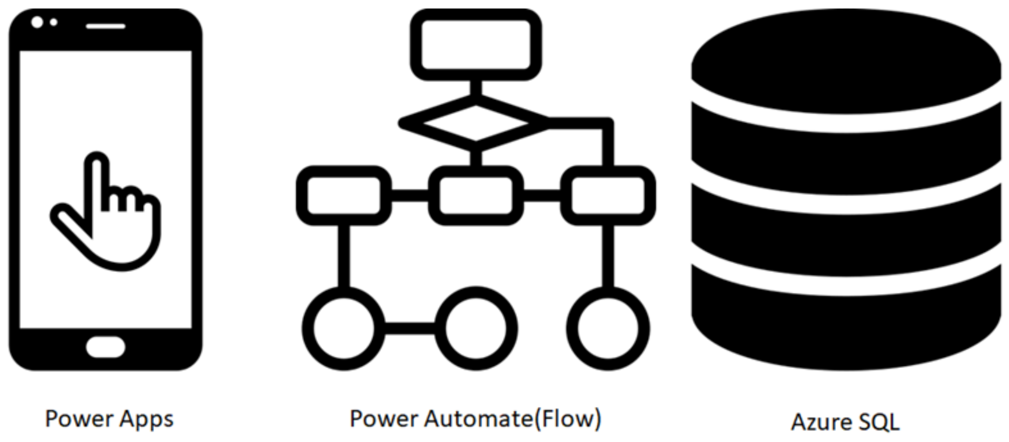


Figure 1. Structure of development

2.2 Frontend development solutions

2.2.1 LCDP

LCDP is the acronym for Low-Code Development Platform(s). In other words, software which provide an application creation environment with minimal coding requirements. (Marvin 2014) LCDPs are based on model-driven design principles so they work with a pre-set logic structure to create elements and populate them with functions. They utilize automated code generation based on said templates to make the visual programming possible. App developers only see a list of items from which they can chose from and the software will automatically write the code when the element is used, e.g. by drag-and-dropping it from its list field into the canvas field. The platforms are capable of delivering fully functional applications like this and they do not require additional computer programming skills, except in case of adding unique features which the base product does not provide. Most LCDP-based services include such self-development capabilities not only as an added feature, but simply due to their nature and logic relying on open-source coding principles.

Not as wide-ranged as traditional programming platforms, hence their limitations, LCDPs usually specialize in one particular kind of application (e.g. databases, web application user interfaces, etc.), meaning that choosing the right LCDP platform for a project is crucial. With the elimination of hand coding from the creation process, LCDPs significantly reduce the risk of human-error bugs (e.g. typos, logic errors, etc.), reduce setup, personnel training, and deployment costs significantly, but most importantly they allow more people to contribute to application development and. (Richardson and Rymer, 2016)

2.2.2 Microsoft Power Apps

Microsoft Power Apps is a platform for building business applications. The apps created on it run through web browsers. This feature boosts the apps compatibility by bypassing mainstream hardware requirements. By not downloading the full software and therefore storing and updating its data on-device, the app becomes accessible online for any hardware with the proper internet and display capacity (however, smartphones and tablets stand as preferred). This also allows for constant in-house updating for Power Apps. On average, there is at least one update released every month, but sometimes there are multiple updates within the same month. The updates vary between small tweaks to significant changes therefore an on-device setup would make this configuration extremely heavy on the user hardware and difficult to professionally maintain.

Microsoft Power Apps follows LCDP principles in a way that it requires very little coding knowledge and instead allows the developer to build applications through visual methods. Microsoft Power Apps uses cloud-based data storing, allowing it to access data from OneDrive, Dropbox, Dynamics 365 and other compatible databases from Microsoft as well as other publishers. (Microsoft 2020b)

Using Power Apps requires a valid subscription to the service. There are two kinds of subscriptions, both options basically including the same features, but the pricing varies depending on the client's plans on the scale and complexity of the application they want to create.

- Plan A (Per App) subscription costs 10$/month/user and is valid for a single app. It is most suitable for companies with only a few running app services with lots of users.
- Plan B (Per User) option costs 40$/month/user and includes an unlimited number of creatable apps, making it a better choice for companies with a lot of applications.

Subscription to a MS Office 365 Outlook account on top of the Power Apps account can be helpful to use Power Apps in terms of login as well as added features (e.g. email automation). In this project, using an Outlook account was necessary to include certain features into the app. A Power Apps account in itself was intended for internal use only, in other words, only people who are part of the user organization can access its features and services, which is sometimes considered a drawback.

Subscriptions can be switched mid-development. This is true to most other Microsoft platforms. In case of smaller projects, these changes can be quickly overcome by simply moving the app or app-in-progress into the new system. Using new features on the other hand can and usually will cause problems. Power Apps has an in-built feature for version control and reversion which is a useful addition, but it is not failproof.

2.2.3 Types of Power Apps

Power Apps have 3 types (Canvas, Model-driven, Portals) each designed to support a different approach to the development process, mainly regarding the needs, type and scale of the project at hand. The main difference between the three is how they approach the overall design of the app.

Canvas apps

As its name suggests, it is likely the most visually straightforward type of Power Apps development type. It starts by building a detailed and personalized interface from a blank canvas with most of the focus spent on usability.

The items are selected from the provided menus. Developers can drag-and-drop them into place, very similarly to how building a PowerPoint slide or the standard model of most Prototyping tools work. When the interface is done, it can connect to a huge number of data sources. For added functionality, Canvas allows for Excel-like expressions to build specific logic pipelines for data.

Products made with Canvas are suitable for mobile devices or PC as they run through browsers. They can be embedded into other apps, so that users can run them from Power BI, Teams, etc. Canvas is most suitable for apps with very specific user interface choices or those with a variety of independent data sources. (Microsoft 2020d)

This project was made using Canvas, however, there will be a short explanation for the other two types as well for better contrast.

Model-driven apps

While Canvas starts from the user interface, Model-driven, very descriptively, starts from the application's core data model. Instead of the developer tailoring the interface, the app itself takes care of that part and only requires the developer to work with business data connections and define core business processes. This Power Apps type is most suitable for database specific projects, where the quality of the data is very important, its complexity is high, however, the user interface (also known as the front-end deliverables for the user) are not as important or complex. (Microsoft 2020d)

Power Apps Portals

The third type allows for a new kind of experience for its users and is the newest addition to Power Apps, launched in 2019. By creating external-facing websites with Portals, the future visitors to that website do not need to be a part of the user company. Instead, they are allowed access with a variety of predefined identifiers (e.g. login credentials). Even though these visitors will not have company credentials, they will be able to browse and even modify certain data which they have access to. This might not seem like a novelty but considering that still almost no coding skills are required, it is a significant step forward in application development to allow for such complicated infrastructures to be made. (Microsoft 2020d)

Common Data Service

CDS (Common Data Service) is not a type of Power Apps builder, but rather a backend component that makes its overall versatility possible. CDS comes with Power Apps by default. It handles storage and modelling of business data and forms the core for Dynamics 365. All Dynamics users already have their data in CDS meaning, that if they decide to use Power Apps in the future, they will not need to bother with risky data transactions, since everything will be already on hand. CDS stores data mainly in common scenario entities (it already has populatable table-like subcategories which are most commonly found in business databases), but, according to need, users can create their own entities. (Microsoft 2020d)

2.2.4 Alternative products

OutSystems

An LCDP platform for the development of enterprise web and mobile applications. These applications can be run in the cloud, on-premises, or in hybrid environments. (OutSystems, 2020)

Salesforce Lightning

Another LCDP software which allows visual-based application creation for multiple devices. An added plus to Lightning is that if a user creates a custom component and decides to share it, others can access and use the same component or even modify it to their own liking. This means that with Lightning, one can build an app using both off-the-shelf and custom-made parts. (Salesforce Lightning, 2020)

Mendix

Mendix, yet another LCDP-minded developer, offers a solution that focuses on the average user. Outside the standard LCDP model, Mendix puts extra effort into cloud-based team functionality betterment and testing capabilities. Mendix is most suitable for small teams with short projects. (Mendix, 2020)

2.3 Backend development solutions

2.3.1 Azure SQL

Azure SQL offers database as a service within the Microsoft Azure environment. Applications built with Azure SQL are easily transferred to other databases (and vice-versa) if the developer changes platforms at any time during development or after. Azure SQL provides the necessary background components of an application database (data security, performance, storage space etc.), while requiring very little micromanagement from the user’s part. (Mazumdar, Agarwal & Banerjee, 2016)

Other than its versatility with data exchange, Azure SQL was chosen for this project by the specific request of the customer who the application was built for due to its price options.

2.3.2 Microsoft Power Automate (previously named: Flow)

Microsoft Power Automate is a cloud-based workflow automation service. The idea behind Power Automate is to help users improve productivity and lessen the workload by creating flows based on certain actions or triggers. In the simplest of terms, when a user clicks a button, the software recognizes it as a predetermined trigger. It then proceeds to carry out a series of successive actions in a chain reaction-like fashion, until it reaches the endpoint where a controlled termination takes place and no further actions are carried out.

Power Automate can connect multiple platforms and applications together, allowing them to communicate and carry out complicated functions without the need of human input. Like in the rest of the project, setting up flows requires no coding skills whatsoever. The platform is accessible through an internet browser, namely Edge, Chrome, or Safari. (Microsoft 2020c)

2.3.3 Alternative products for Power Automate

IFTTT

IFTTT, also known as If This Then That is a free web-based service to create chains of simple conditional statements. These statements function similarly to Microsoft Flow’s flows in that they are set up to be triggered by an action (e.g. a keyword) and execute a reaction (e.g. sending email) in specified ways. IFTTT is a web-based mobile application that allows its users to come up with recipes for automation (mostly focusing on social media platforms) and share these recipes with other users. Like Flow, it does not require coding skills to set up and use. (IFTTT, 2020)

Zapier

An online automation tool that again requires no coding, Zapier can connect a wide range of apps and run automated tasks between them very similarly to previously mentioned tools. (Zapier, 2020)

Build on Standard Library

A more professional approach to the standard workflow automation. It works similarly to the above-mentioned platforms, however, it allows for more complex workflow designs (depending on the developer’s skill) as every custom created schema is also available as raw code which can be modified and reinserted for the desired effect. Build on Standard Library also provides tools for version control, rollbacks, etc. (Build on Standard Library, 2020)

3 Development of the application

This section is going to discuss the development of the sample product and its deeper workings in detail. The goal is to highlight the standard procedure of modern app development with a valid example at hand. Since the application of the thesis was made by the request of a client company, many of the choices were pre-decided by them and not the developer. However, there have been plenty of opportunities for creative development from both functionality and usability sides. The chapters below are going to follow an abridged explanation about how the application was designed and made mainly from a customer-developer relationship point of view.

3.1 Case introduction

The customer was a small company in the waste management sector who needed a business application for their on-site workers. The problem was that the original method of documentation was paper-based and full of potential risks (e.g. running out of forms on-site, the paper could get easily damaged or go missing, processing was slow, manual and troublesome, etc.). According to the customer's summary, the workers (end users) were mostly middle-aged men, inexperienced with software and applications.

Before the application, the workers would carry out their task on-site, after which they would mark on the paper form what type of work they carried out, get it signed by the customer and later submit it for processing. The customer company prioritized three things: mobility, usability, simplicity. Their idea was some sort of mobile application so the workers can always have the form ready on their mobile devices. The app would eliminate physical damage and make processing easier and possibly automated. Finally, the app had to resemble the original paper form, at least in its standard functions. Also, it would have to be easy to use. Additional requests included a digital signature function and an automated email function.

The customer company was working with a tight budget and short timeframe, so the solution had to be a simple one. The end-product was an easy to use mobile app, with all the functionalities listed above. During development, the functionality testers were the real end users to make sure that the app was well tailored to their needs.

3.2 Development plan

The product development consists of five main phases: initiation, design, development, testing, implementation. Each of these phases are distinct in their function within the project's creation and have been implemented accordingly. They each mark a milestone which is to be reached before moving on to the next part. Time is an important factor throughout the whole development process, so proper scheduling and time management are crucial to handle.

The development process chosen for the project was agile. In the agile methodology, small iterations were agreed upon by both developer and customer. Each iteration consisted of four of the abovementioned five phases, excluding implementation, where the focus was on a set deliverable or closely related deliverables which were to be done and added to the rest of the project by the end of the iteration cycle, or run. Each run was discussed with the supervisor of the project who kept contact with the customer. In agile in general, an iteration should not take longer than a weak, allowing for quick changes and more direct customer-company communication.

When an iteration was carried out it was first evaluated by the supervisor before finally submitted for approval to the customer. Once the approval was received, the next iteration commenced. Note, however, that the individual runtimes often vary due to certain conditions. One main factor in this is the problem to be tackled in the given run. If the problem is difficult, or the app development has reached a certain complexity, or in this case, when the developer asks for things to be added, deleted or changed, it makes the developer's work harder and therefore prolongs the iteration itself. Another reason for prolongment or delay is simple communication with the customer. Customers in general tend to take their time with the testing for reasons of their own and the developer effectively cannot move on until the customer has evaluated their previous work.

3.2.1 Initiation

Initiation means the beginning of the project. The client company contacted the developer and after negotiations, the request for a product was made. At this point, the product plan was just an outline of requirements rather than an actual idea. Clarifying requirements is important to do early on in the development process. It determines whether the project is doable with the developer's skills and/or resources, as well as giving an idea about the needs, functions and scope of the final deliverable. Discussions, both legal and technical, while part of such endeavours, will not be included in the thesis as they are not within scope.

Every iteration also begins with an initiation phase. As such, it marks the beginning of the creation of a new function or piece added to the final product, or the changing of an existing part. Initiation is always carried out by the customer after careful evaluation of the previous run’s results, and then approved by the developer. Sometimes the initiation poses problems, such as disagreements or unclear requests in which case careful communication is needed until the idea is clearly outlined and both parties are in agreement with one another.

In this project, the customer company provided information on what functionalities they were primarily looking for, what systems they have been using in the past and what systems they plan on using this time. The customer also provided requests on interface layouts. To help the developers better understand their needs, the customer described in detail the average qualification and experience of the end users, giving the developer a solid basis for user case build scenarios. Some of the major points the customer company had requested included a user friendly interface, a product focused on data input, review and editing capabilities, simplicity for non-IT personnel and resemblance to the existing, but not software-based solution which the app was designed to replace.

3.2.2 Design

The design phase is where the developer decides how to include the new deliverables or how to modify what needs to be modified in the project. The sample application of the thesis is considered to be a small endeavour, but no matter the size, the design phase is extremely important for a smooth iteration and good end results.

After the initiation meetings, the developer always builds up a solid and logical approach on how to tackle the task at hand. This plan is then to be followed thoroughly until the end of the run or, in case of the first run, the design is meant for the overall project. Without a proper design plan, developers are more prone to error, and a project is more likely to fail.

As the customer in this project’s case had very specific requests, the developer had to focus on delivering those first. The design phase is meant to streamline the development process, serve as a trusty guideline and it’s a good way for backing up the creation process by keeping track of what has been done, when and how and what comes next. Constant communication with the customer is still very important, as they can add or change certain things according to their needs or serve with extra information should the need arise.

One distinguishing feature of LCDP type development and visual development in general is its strong compatibility with creativity. As the developer does not necessary have to understand every function that goes on in the background of the platform while at the same time using it to its full extent and building up the app, more focus can be given on what the end user will see and experience. Working out how to best implement the required qualities into a product often comes in the middle of the development process and since the LCDP allows for spontaneous ideas, it is well suited for the task. If something new, unplanned seems to work fine, the developer simply shares it with the customer who can then decide if they go forward with the solution or not. LCDP also eliminates the need for strict hierarchies within the app's functions. In this case, the developer focused on creating the most important aspects of the app first so the customer could evaluate and change it if necessary. It was the most crucial part of the project, tackling it first meant that later during development the frequency of sudden change requests would be lower, allowing for an overall better development experience. These features adda more flexibility to the already very flexible agile methodology. However, the benefits and reasonings provided here are mainly true in case of smaller projects only.

3.2.3 Development

The development phase is where the designs are executed. In this project, it started with the building of the database in Azure SQL. Based on the customer's needs, data tables were created with label names picked out prior to the initiation of this phase. This has been a very important step in the development which will be further discussed in chapter 4.

After the data structure was done, it was time to create the interface and all the other seen-by-the-user parts of the application. An initial design plan was followed and improved after every iteration, with the overall product broken into smaller partitions based on individual deliverables. Power Apps' Canvas type creator was used for the interface design with added focus on usability details. Connecting the two parts for the functional or semi-functional model could have been done either during the Power Apps interface development phase or after. Since the product was made in small, co-reliant iterations, the automation part was done after the interfacing was ready.

There have been constant changes during development. Sometimes these changes interfere with other parts of the app, rendering them faulty or useless, therefore the developers must keep a well-documented change log. Power Apps, as it was mentioned before in chapter 2, has its own in-built changelog where developers can reverse actions or roll back the app completely to a previous version, given that a save exists of it.

3.2.4 Testing

Testing is necessary to see if a product or service works as intended. Does it execute the tasks it was meant to execute? If yes, is it doing it in the proper way? Some testing has been done partially during the development phase. This was necessary in the final runs, where the automation was set up. These small-scale functionality tests were carried out to see if the individual components were connecting and working as intended. This part was done by the developer as it qualifies as part of the backend development. However, the overall testing at the end of each run was meant for the testing of the results of the iterations and as such, they were carried out by the customer. After each of these phases, the customer was to approve or submit change requests, including design or functionality discrepancies. The testing phase was crucial in the development of the final application, after which the product was ready to be launched.

3.2.5 Implementation

After many rounds of testing and refining, the product stood ready to be launched. In this phase, the customer gets the product and the users begin using it as intended. It is up to the customer and the developer to agree on a short period of time during the early life of the product when customer training and/or support is available. In case of this application, such support was given until the users learnt how to use the app, which took only a few days. This phase, however, does not allow further changes to the application, only customer support.

4 The three main parts of the application

As mentioned in the previous chapter on the design phase of the development, the application consists of three main parts: Database, Interface, Automation. This chapter will take a closer look on how these parts appear in a functioning app, using the sample product. The introductions of these parts will be carried out in the same order as it was done in the real-life implementation. This order is by no means particular and was setup during the initial designing of the project as a whole. As discussed earlier, most LCDP platforms allow for a flexible approach towards tackling a project. It is usually up to the developer to decide in which order they will set up the components for the final product. That is unless the customer has specific requests for a given order.

4.1 The database of the product: Azure SQL

The application could have been connected to various data storages, such as the standard SQL database, Sharepoint List, Excel files, ERP systems, etc. Among these, SQL database, Sharepoint List and Dynamics 365 are the most commonly used ones in real user cases. Deciding on the database provider is an important part in a project's development which largely comes down to the customer's preferences and/or previous implementations. For example, if the company uses Dynamics 365 for database as default within their organization, Power Apps can be connected to it in later projects as well. Not to mention pricing problems, where using or even extending an existing subscription is often cheaper comparing to registering to a new provider. As the customer company in this case had no such prior implementations, the choice of using Azure SQL was made for functionality and budgeting reasons mainly, however, there were other factors taken into consideration too.

Sharepoint list is limited to 5000 rows (items), making it insufficient for storing large quantities of data, which was expected to be the case with this product. Dynamics has no such drawbacks, however, the users could face complications if they, later in the application's life cycle, decide to change the original data structure. It cannot be stressed enough that a clear data structure plan is one of the most important parts of app development as changes within data structure may cause bugs or even a complete crash. Meanwhile, Azure SQL allows for such changes, while also offering a very cost-efficient service with no limit to the number of users.

4.1.1 Data structure

The database is built in a simple and comprehensive way: figure 2 illustrates it below. The database consists of only two tables. The first table, labeled “form” contains information from the customer, the service provider, together with the time frame of the service carried out. The other table, labeled “Service_items” has information about the service provided to the customer, including offers and descriptions. The two tables are connected by a “formID” column as primary key in the first table, and “ID” column as foreign key in the second table.

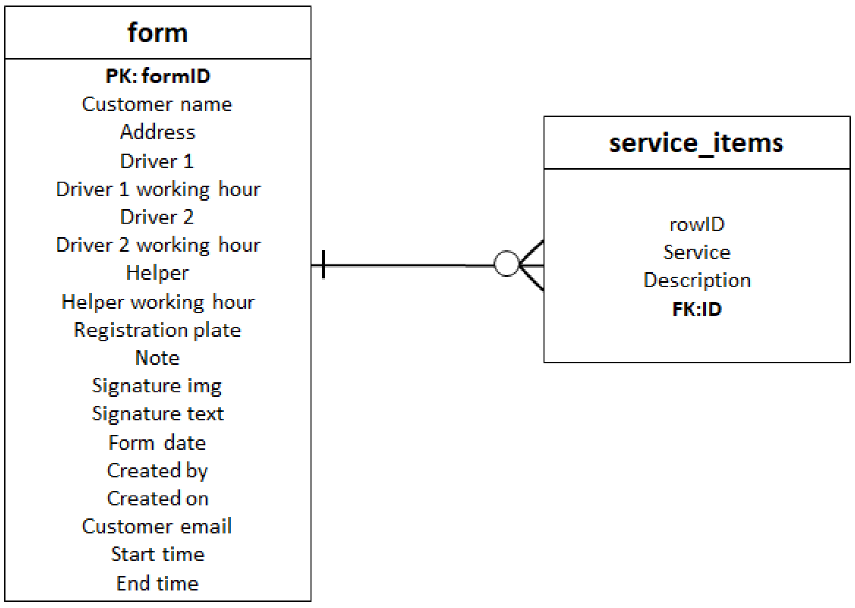
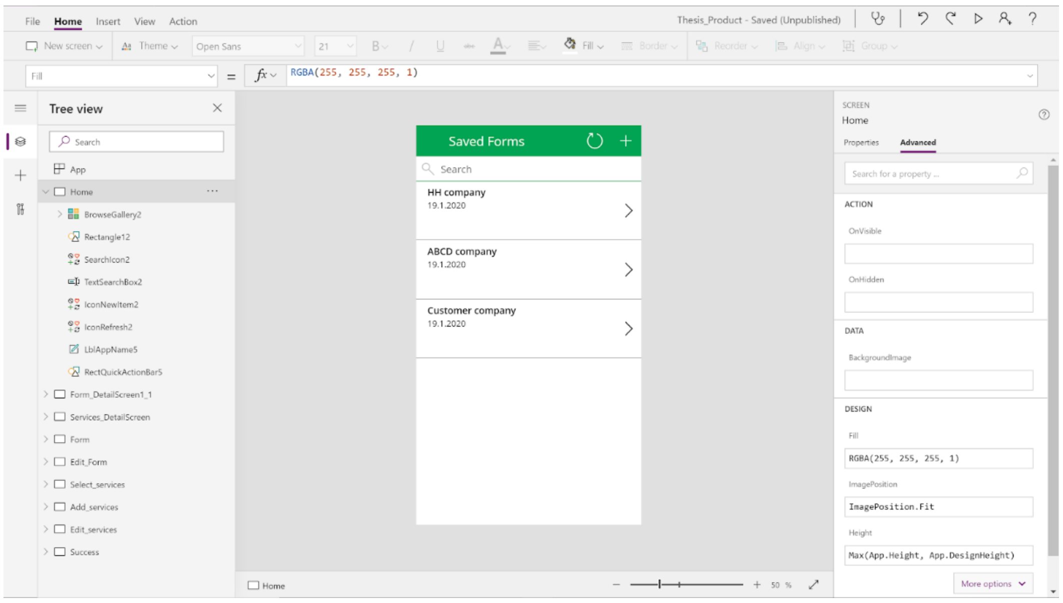


Figure 2. Data structure

4.2 Frontend of the product: Power Apps

Power Apps was used to create the interface of the application. This part is what the users see and interact with, and it was a very specific request of the customer to make this part as comprehensible and familiar to the users as possible. Things to keep in mind during the building of an app include what mainstream programmers often do not think about, for example colour theory, layout, etc. The visual-based coding interface helps a lot in this sense. As these requirements differ from project to project so they will not be discussed in further detail.



Fill = RGBA(255, 255, 255, 1)

Fill: RGBA(255, 255, 255, 1)

ImagePosition: ImagePosition.Fit

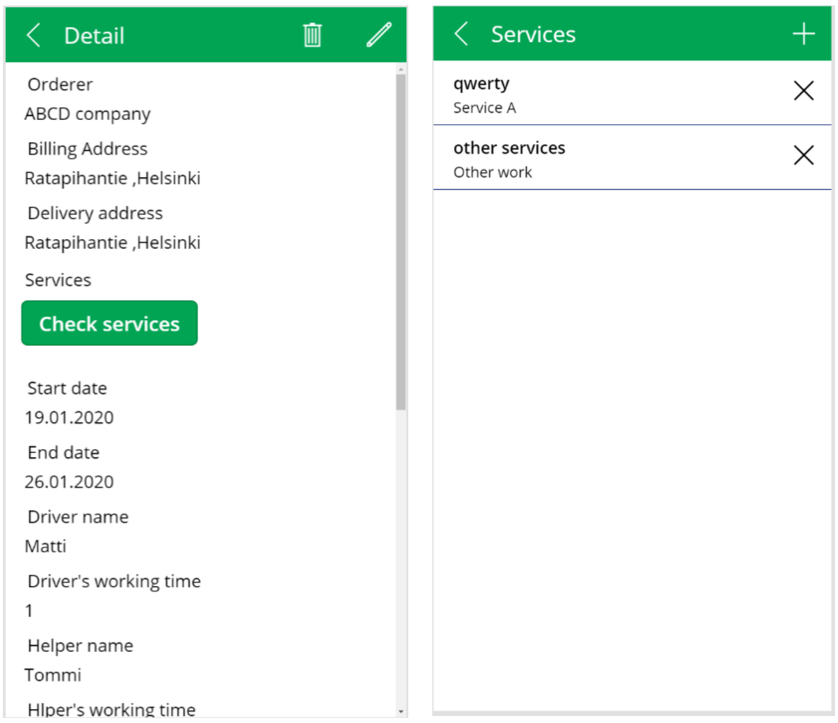
Height: Max(App.Height, App.DesignHeight)

Figure 3. Example image from Power Apps

Power Apps’ interface consists of nine screens. Figure 3 depicts the Power Apps Editor. It uses a drag-and-drop method for interface building. Each element can be selected from a menu and then assigned to the right fields or data sources. Careful planning can save the developer lots of trouble by predetermining the right names for the fields, because, for example, if a column is renamed in the database, it does not automatically change in Power Apps.

- From the tree view on the left side, the user can navigate to the other screens created for their application-to-be.
- The ribbon on the top contains actions that the user can choose from (e.g. text fonts, colouring, etc.)
- On the right side of the screen, the user can modify each individual element on the currently open interface tab.

Figure 3 shows the application’s “Home” tab. This is what the users of the final product will see if they open up the application. Users can see all their saved forms, add new ones, delete or modify existing ones, or simply select one for reviewing by pressing buttons.


Detail

Orderer
ABCD company
Billing Address
Ratapihantie ,Helsinki
Delivery address
Ratapihantie ,Helsinki
Services

Check services

Start date
19.01.2020
End date
26.01.2020
Driver name
Matti
Driver's working time
1
Helper name
Tommi
Hlper's working time

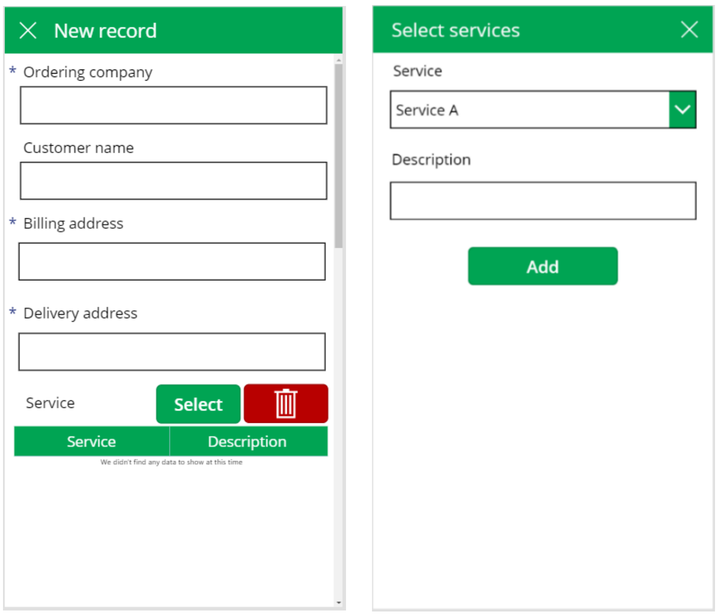
Services

qwerty
Service A

other services
Other work

Figure 4. “Detail” and “Services” pages of the application

Figure 4 shows the detail tab and services tab. On the “Detail” page, users can open saved forms, or see saved services and their descriptions by simply tapping the “Check services” button on the left picture of figure 4. Services and descriptions can be added, edited, or deleted. The functions of Detail and Services are based on different data tables. In this case, the separation speeds up the application and prevents too much information to be crammed into a single tab, thus improving usability. It also resembles the old standard paper for of which the users were most accustomed to.


New record

* Ordering company
Customer name
* Billing address
* Delivery address
Service
Select

Service	Description

We didn't find any data to show at this time

Select services

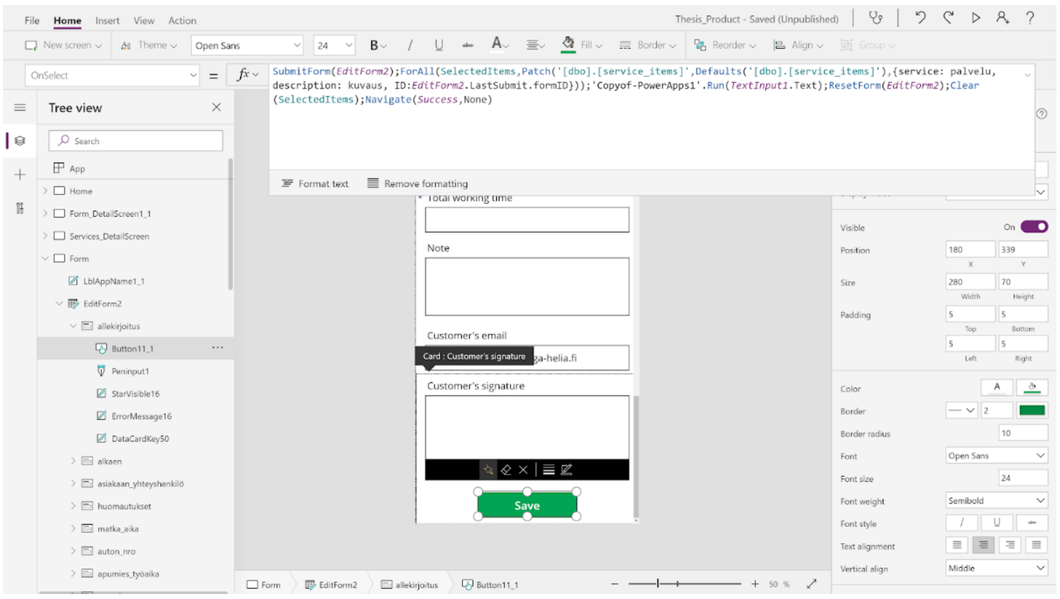
Service
Service A
Description

Add

Figure 5. “New record” and “Selected services” tabs

New records can be added in the “New records” tab. Tapping the “Select” button navigates the user to the “Select services” tab where they can add a new service along with a short description attached to it. Clicking the “X” button will close the tab. The “Select service” and the previously described “Services” tabs illustrate a common problem which developers often face. The two tabs are very similar. In fact, they were made by copy-pasting and then modifying one’s functions. The “Select function” tab allows for the creation of new entries, while the “Services” tab only allows to view, modify, or delete descriptions. There is a fundamental workflow behind both tabs which distinguishes them from each other.

- On Power Apps, every form is assigned a data source and an item. Data source equals data table from the database which the form is connected to, while item is the data row within that table which the user can select and modify through using the app’s interface.
- A new form is assigned to the default item from its data source (e.g. Defaults(‘form’)). The new form is unique in what it contains, but not in properties and therefore it allows a fast trackability within the database.
- For updating/deleting an existing record, the item assigned is more specific (e.g. Gallery1.Selected.item, Gallery1.selected.id=’2’). This is to prevent multiple, unrelated records to be modified at once by accident.



```
SubmitForm(EditForm2);ForAll(SelectedItems,Patch('[dbo].[service_items]',Defaults('[dbo].[service_items]'),{service: palvelu,
description: kuvaus, ID:EditForm2.LastSubmit.formID}));'Copyof-PowerApps1'.Run(TextInput1.Text);ResetForm(EditForm2);Clear
(SelectedItems);Navigate(Success,None)
```

Figure 6. Power Apps buttons and functions

Every button press executes an automated response within the application. The button responses communicate with each other and with the SQL database through Power Automate (more on this in the next chapter). A few key example functions and how they work are listed below:

- When the user saves a form record by tapping the “Save” button: it triggers Power Automate which then updates the appropriate data source within the database.
- Submit Form: This function records into the “Form” section of the app and the database.
- For All: This function saves all the selected services, along with descriptions, to a table within the database called “service_items”.
- Run: Triggers Power Automate and sends the form ID to Power Automate. Based on this form ID, Power Automate gets the correct row from SQL database and updates the changes to the database.
- Reset Form: As its name implies, it clears all previous user input from the fields in the Power Apps form so that fresh records can be created. It is designed so that users will not have to close every form and open new ones in case they want to create multiple submissions. Instead, they can complete one form, save it, and hit Reset Form to automatically get a clean slate for the next input.
- Clear: When a user selects a service, this selection is firstly stored into a collection (array) named “SelectedItems” within the database. The “Clear” function empties this array, so that old records would not remain in the system unnecessarily.
- Navigate: Like a simple search menu, this function redirects users to their desired tabs. (Microsoft 2020a)

4.3 Process automation: Power Automate

The chapter covers the basic process with which Power Automate operates. Power Automate gets inputs or triggers from Power Apps and follows a series of steps to carry out the appropriate function. An example is shown below in the form of a process map. To help understand its workings, every step down the line will be explained in more detail.

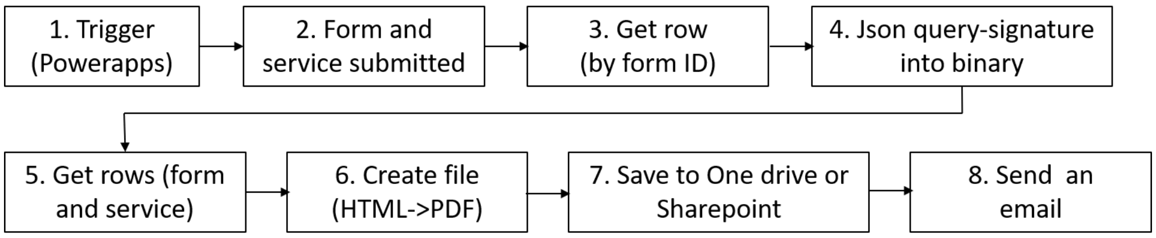

1. Trigger (Powerapps)
2. Form and service submitted
3. Get row (by form ID)
4. Json query-signature into binary
5. Get rows (form and service)
6. Create file (HTML->PDF)
7. Save to One drive or Sharepoint
8. Send an email

Figure 7. process map for Power Automate process execution

Power Automate is part of the backend development. It automates a process to minimise input demand from the users' side and to prevent users from having to repeat procedures every time they use the app. It also makes functionality coding more reliable by eliminating most human interactions.

Trigger

Power Automate is activated by a trigger. A trigger is a piece of code that, when detected, makes Power Automate run a series of steps which relate to each other in a set chain. There are three categories of triggers:

- Automated: triggered by a designated event (e.g. arrival of a new email). It adds significantly to the automation of different work processes by cutting out user input requirements.
- Instant: triggered manually as needed (e.g. pressing a button). The most common trigger and possibly the only one a user is aware of while using the app.
- Scheduled: triggered according to a fixed time schedule (e.g. every day, every five minutes, etc.). Depending on the requirements, developers can set any time frame to carry out necessary events, such as clearing unnecessary data at the end of the day or updating a menu with new information every few minutes.

Triggers are to be selected according to the function that needs to be carried out. In Power Automate it is also possible to use API calls instead of triggers, but it needs more advanced IT skills and thus would defeat the purpose of the thesis's main subject. The sample application largely uses button presses (instant triggers).

Form and service submission

Power Automate triggered when the user pressed a specific button on the interface in Power Apps. It prompted the execution of a number of functions designated to that button. The first step (after triggering) is the submission of the form which the user filled out. To do this, the app inserts the new records into the database. With every new record, in this example with a new form, a form ID is generated and stored together with the record. In case of a new service, a row ID would be created since these two functions use different data sources. The two ID types, however, serve the same purpose, namely, to accurately identify their own objects within the database.

Get row (by form ID)

After adding it to the database, Power Automate immediately recalls the new record by its form ID. This step is necessary for carrying out further functions associated with the original trigger. The original save of the form into the database itself only serves to safekeep the user's inputs. Every modification is later going to be carried out in the same form, instead of creating new ones, by simply updating the appropriate rows, hence the recalling in this step.

Json query-signature into binary

Power Automate executes an SQL query (compatible with Azure SQL database) to convert the user's electronic signature from an image file into binary data. It is highly recommended to store image files in binary format in SQL as storing images may severely slow down the system. The binary data can be recalled and converted back into an image file later on if needed.

Get rows (form and service)

The app draws records from both the form and the service tables. In the case of a form insertion, Power Automate uses the previously recalled form ID to find the linked records from the service table based on their row ID. Form ID and row ID connect on a one-to-many principal, meaning that while one form ID is generated per form, that form may contain more than one service, so it could be linked to more than one row ID. This depends on the submitted form itself.

Create file (HTML to PDF)

This section is an added function for the application and it requires slightly more advanced knowledge about Microsoft Function App and the C# language. Therefore, the exact details of how this automation process is carried out will not be discussed, instead a short and comprehensive summary is explored.

The application is accessed through an internet browser and thus the user has the option to recreate the file using HTML tags to modify the form later. An additional function app was created and linked with the example process through API that is then called upon via an HTTP connector. Using previously extracted data, the app converts the now HTML form into PDF format.

If the user does not mind manually converting to PDF, this step can be completely abandoned. Originally it was invented to further eliminate user input, as the end users were described as generally being unskilled in IT.

Saving on OneDrive or Sharepoint

The generated PDF file (created either manually or automatically via the previous step) is then stored to a web-drive service (usually either to OneDrive or SharePoint as both are Microsoft services).

Send an email

Finally, an email is sent to the designated receiver named in the form itself, with the PDF file attached. This stage allows developers to use SMTP (Simple Mail Transfer Protocol), which would eliminate the need of every user having their own email address and would instead use a globally setup one. It is not necessary as long as the end users do not mind sending the email from their own Outlook email addresses.

5 Result and Discussion

5.1 Power Apps and low-code development

Power Apps can be very efficient in cases when the app is meant to be simple, with only a couple of functionalities and a clear, easy-to-use interface. Depending on the requirements of the customer, a fully functioning app can be done in a matter of weeks. In fact, Microsoft has readymade templates for Power Apps, which only requires to be populated and properly set up with data sources, and it is ready to go as a fully functioning mobile application. Using these templates, it is not uncommon to deliver products within 1-2 business days. The whole point of the thesis was to introduce this relatively new (and constantly under development) method of mobile app development. It is a perfectly valid solution for less tech savvy people, new/unskilled developers, or start-up companies with low budgets, short time frames, low level of requirements, etc.
If the developers have some knowledge of power apps, they can change things relatively easily by themselves, especially from a design point of view. Power Apps, like all LCDP-s are currently very strict with their design options, allowing only for simplistic layouts due to the code templates and other automated code generation functions. Small tweaks in e.g. colour, size of the font, etc. can be easily done. While these can be changed from the source code, the logic itself should not be modified as it will most likely lead to crashes.

5.2 Microsoft and other providers

Power Apps is not the only low-code development tool on the market and this thesis was not meant to advertise it in any way. However, for the purposes of this and similar projects, Power Apps delivers well and within a reasonable price range. Alternative platforms usually offer tiers to their users. The more functionality a tier has, the higher the price. Out of flexibility reasons, the price is often not even included in the general information on their websites. Microsoft kept its prices adaptive and easy to understand, as it was discussed in chapter 2. On the other hand, while most competitors offer a low functionality but free version of their software, Microsoft only offers a 30-day trial for Power Apps.

Undeniably, Microsoft has huge influence in technology, and they are clearly taking advantage of this fact. Would it be worth for companies with existing MS subscriptions to use Power Apps, Power Automate and other MS products? Apparently yes, as Microsoft offers subscriptions (MS Office 365 E3/F3) which include both Power Apps and Power Automate, meaning that all standard functions are included in one package. They are also leaders within the market by a large margin and pioneers in further development, so in the end, it is almost inevitable to use one of their products, and they are building on this idea very consciously.

5.3 Limitations and future improvements

There is still lots of room for improvement:

- The error detection system works very well in Power Apps (it underlines with red the logic errors, typos, etc.) and it has a reliable version control system for safe-guarding against human or computer errors. However, while adding new thing to the existing solution is extremely simple, changing something that has been already made is not recommended, because it can disrupt the entire app's functionality, often leading to the need to completely reverse to a previous version.
- Power Apps is still very limited, especially design wise. Text font, container shapes and many more features are within strict limits and can only be changed by manually tempering with the code. On the one hand, this limits the developer's creative impulse, a thing on which low-code development heavily builds. On the other, Power Apps and others like it are a new concept and their creators are constantly updating, releasing new patches, etc. Microsoft has an update at least every month.
- Power Apps is for internal users only. Although Microsoft released Power Apps Portal in 2019, which enables external users to access the apps, it is still almost just a prototype rather than a fully functioning platform. It lacks basic functionalities and it is not compatible with Canvas which was the basis of the thesis's sample project.

5.4 About low-code development

It is essential to understand by anyone who is interested in the topic, that low-code development is not a new idea, only by relative terms. However, the solutions which are currently available for developers are quite simplistic, but powerful. Another important note is that not many resources are available on the topic yet, mainly due to its novelty. As it is the product of the 21st century, there are quite few academic records discussing low-code development in detail. Also, due to the constant updates, all LCDP type products are prone to change in both looks and logic. If the developer faces a problem, the best they can hope for is that someone else has faced the same problem recently, has specified that problem on internet forums and somebody gave a valid answer. Otherwise it is pure trial-error and logical thinking.

5.5 Evaluation of the thesis project and own learning

The experience from both the project and the writing of the thesis has been positive. The project itself was a milestone as it was a level above everything I have achieved so far at that point of my career. It was educational to try out more things on Power Apps development (e.g. converting image to binary data, using Function App to convert binary data to pdf, etc.). The experience from this project is helping me in the more complicated next steps in app development.

Writing the thesis, explaining how things work, is much different than doing them. The same logic applies to the development process itself. During the making of the application, I was not required to know so much about Microsoft's different platforms, how they work internally, what kind of changes have been done to them since their release, etc. The thesis made me research these things and as a result, many functions and other things which I did not fully understand at the time (even if I used them effectively) became clear. While it is not a prerequisite to know the software in such detail for a developer, it is certainly useful to look into things deeper than base functionality.

References

Build on Standard Library O Official Documentation
URL:https://docs.stdlib.com/overview/introduction/ Accessed: 12 February 2020

Chowhan, K. 2018. Hands-On Serverless Computing: Build, Run and Orchestrate Serverless Applications Using AWS Lambda, Microsoft Azure Functions, and Google Cloud Functions. Packt Publishing, Limited. (ISBN: 9781788836654)

Clay Richardson and John R. Rymer 2016. "Vendor Landscape: The Fractured, Fertile Terrain Of Low-code Application Platforms" (PDF). Forrester Research. Archived from the original (PDF) on 2017-08-09. (Retrieved 2017-01-25)
URL: https://web.archive.org/web/20170809060147/http://informationsecurity.report/Resources/Whitepapers/0eb07c59-b01c-4399-9022-dfc297487060_Forrester%20Vendor%20Landscape%20The%20Fractured,%20Fertile%20Terrain.pdf
Accessed: 10 February 2020

IFTTT Official Documentation

URL: https://platform.ifttt.com/docs Accessed: 11 February 2020

Mazumdar, P., Agarwal, S. & Banerjee, A. 2016. Pro SQL Server on Microsoft Azure. Apress.(ISBN: 9781484220825)

Mendix Official Documentation
URL: https://www.mendix.com/low-code-guide/ Accessed: 11 February 2020

Microsoft 2020a: Power Apps Official Documentation – Formula Reference
URL: https://docs.microsoft.com/en-us/Power Apps/maker/canvas-apps/formula-reference
Accessed:01 March 2020

Microsoft 2020b: Power Apps Official Documentation – What are Power Apps?
URL: https://docs.microsoft.com/en-us/Power Apps/Power Apps-overview Accessed:01 March 2020

Microsoft 2020c: Power Automate Official Documentation. URL: https://docs.microsoft.com/en-us/power-automate/ Accessed:02 March 2020

Microsoft 2020d: Power Apps Official Documentation -Types of Power Apps
URL: https://docs.microsoft.com/en-us/Power Apps/maker/ Accessed: 02 May 2020

Outsystems Official Documentation
URL: https://www.outsystems.com/platform/ Accessed: 10 February 2020

Rob Marvin (12 August 2014) "How low-code development seeks to accelerate software delivery - SD Times". SD Times. San Diego Times. (Retrieved 18 November 2016) https://sdtimes.com/application-development/low-code-development-seeks-accelerate-software-delivery/ Accessed: 10 February 2020

Salesforce Lightning Official Documentation
URL: https://www.salesforce.com/campaign/lightning/ Accessed: 11 February 2020

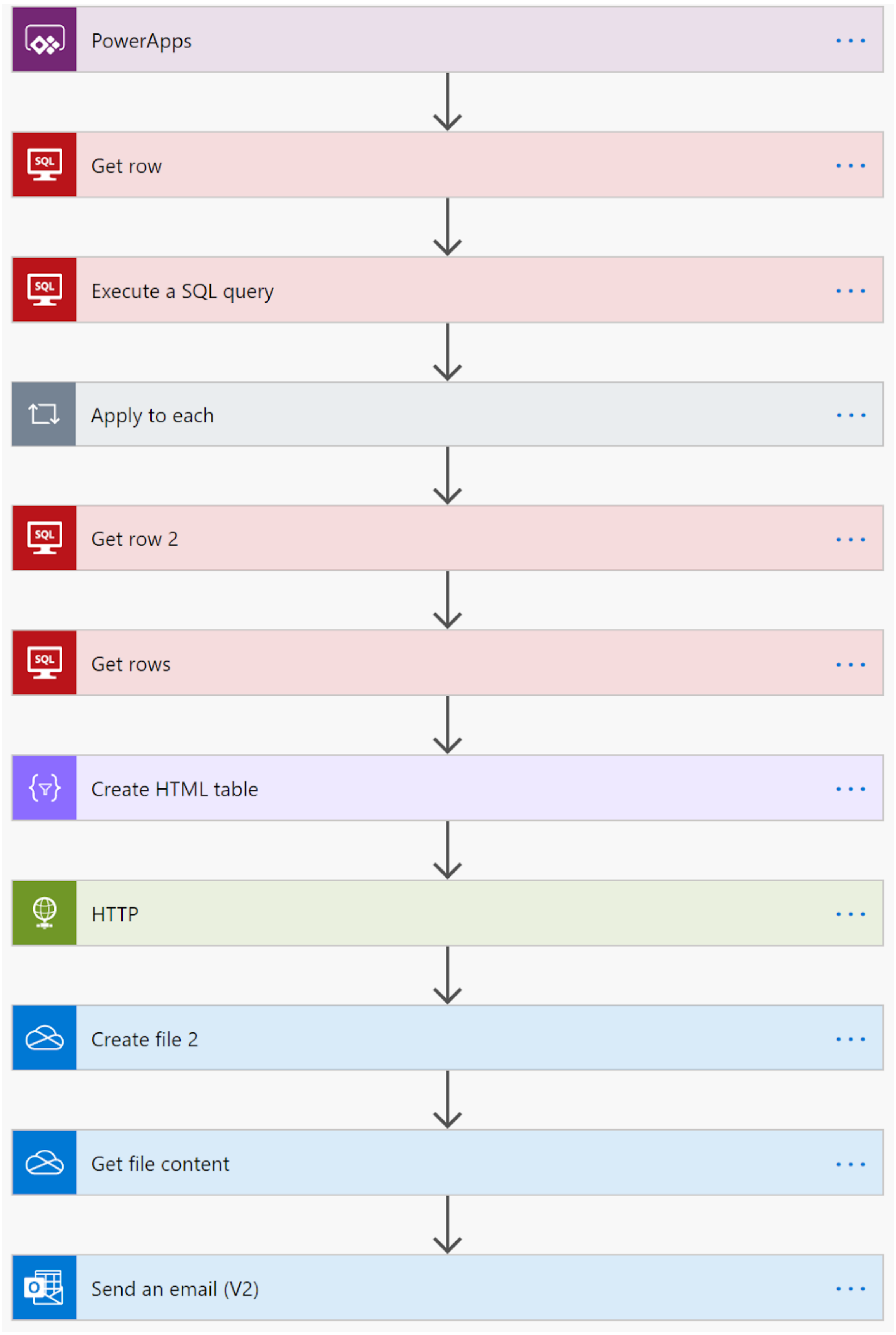
Tim Leung. 2017. Beginning Power Apps: The non-developers guide to building business mobile applications.(ISBN: 9781484230039)

Vijai Anand Ramalingam.2018. Introducing Microsoft Flow: Automating workflows between apps and services.(ISBN: 9781484236307)

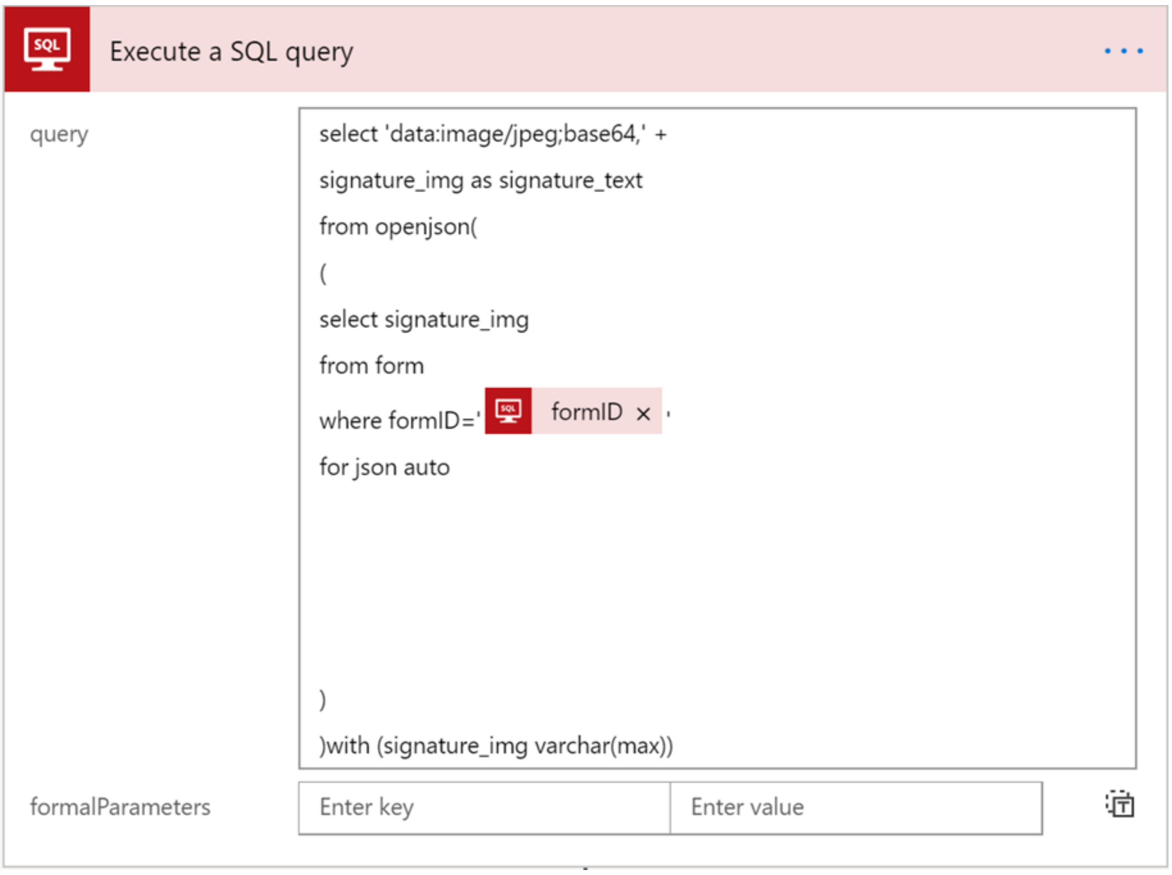
Zapier Official Documentation
URL: https://zapier.com/learn/getting-started-guide/what-is-zapier/
Accessed: 12 February 2020

Appendices

Appendix 1. Power Automate Structure



Appendix 2. JSON formatting for image transformation



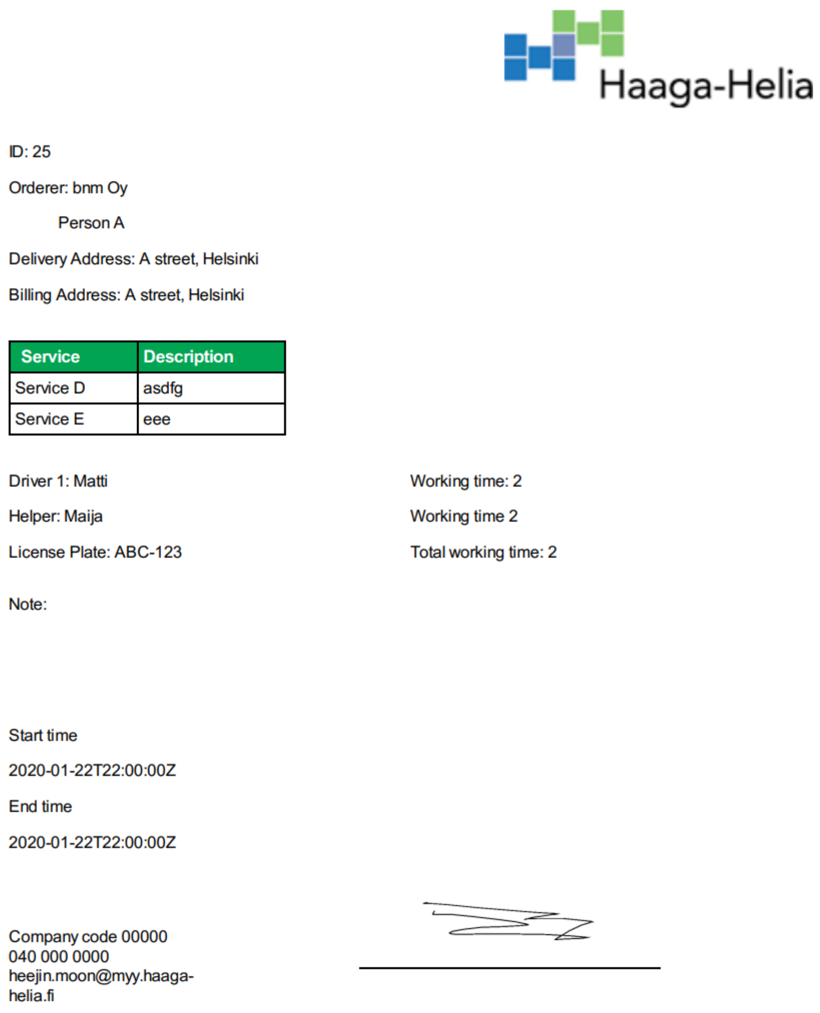
Execute a SQL query

query

```
select 'data:image/jpeg;base64,' +
signature_img as signature_text
from openjson(
(
select signature_img
from form
where formID=' formID '
for json auto
)
)with (signature_img varchar(max))
```

formalParameters | Enter key | Enter value

Appendix 3. Generated PDF form example



Haaga-Helia

ID: 25

Orderer: bnm Oy

Person A

Delivery Address: A street, Helsinki

Billing Address: A street, Helsinki

Service	Description
Service D	asdfg
Service E	eee

Driver 1: Matti	Working time: 2
Helper: Maija	Working time 2
License Plate: ABC-123	Total working time: 2

Note:

Start time

2020-01-22T22:00:00Z

End time

2020-01-22T22:00:00Z

Company code 00000
040 000 0000
heejin.moon@myy.haaga-
helia.fi

[illegible]