Xiaoxiao Wu

Android Mobile Application Using Azure Mobile Services

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

28 October 2018

Metropolia

Author Title Number of Pages Date	Xiaoxiao Wu Android Mobile Application Using Azure Mobile Services 24 pages 28.10.2018
Degree	Bachelor of Engineering
Degree Program	Information Technology
Professional Major	Computer Networks
Instructors	Markku Nuutinen, Senior Lecturer Zhongliang Hu, Development Manager

The main objective of this thesis was to develop an application on Android platform which is connected to Azure mobile services. This application has three features: live chat, backup file sharing and remote drive parameter monitoring. Part of this project was to gather information on how external libraries are used in this application. Also, the workflow of the application and the user interfaces were examined. Moreover, ideas for improvement in future were analyzed.

As the result of this project, an Android application was developed, it followed MVC architectural pattern and three external libraries were imported. With this application user can send and receive chat messages, upload parameter backups to Azure cloud from phone or vice versa. Also, when the application is connected to drive via Bluetooth and to web application via internet at the same time, it is able to send parameter values to web application in real time.

Keywords	Android, Mobile Application, Azure Mobile Services

Tekijä Otsikko Sivumäärä Aika	Xiaoxiao Wu Android Mobile Application Using Azure Mobile Services 24 sivua 28.10.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Tietoverkko
Ohjaajat	Markku Nuutinen, Yliopettaja Zhongliang Hu, Development Manager

Tämän opinnäytetyön päätavoite on kehittää Android-sovellus, joka on kytketty Azure-mobiilipalveluihin. Sovellus käyttää puhelimen internetyhteyttä. Sovelluksella on kolme ominaisuutta: pikaviestin lähettäminen, varmuuskopion tiedoston vaihtaminen mobiilisovelluksen ja web-sovelluksen välillä ja taajuusmuuttajan parametrin seuranta etänä web-sovelluksen kautta. Opinnäytetyössä tutkitaan myös kolmannen osapuolen kirjaston käyttämistä tässä Android-sovelluksessa. Myös tämän sovelluksen työkulut ja käyttöliittymät on tutkittu. Lisäksi mahdolliset parannusideat tulevaisuudessa on analysoitu.

Työn tuloksena saatiin tämä Android-sovellus tehtyä. Sovellus käyttää MVC-arkkitehtuuria ja kolmannen osapuolen kirjastoa apuna. Sovelluskäyttäjä voi lähettää ja vastaanottaa viestin, ladata parametrin varmuuskopion puhelimesta Azure-pilveen tai pilvestä puhelimen muistiin. Lisäksi kun tämä sovellus on kytketty Bluetooth-yhteyden kautta taajuusmuuttajaan ja internetyhteyden kautta web-sovellukseen samaan aikaan, se voi lähettää parametrin arvot reaaliaikaisesti.

Keywords	Android, Mobiilisovellus, Azure Mobile Services

Metropolia

Contents

List of Abbreviations

JSON	Stands for JavaScript Object Notation, is lightweight data-interchange format. It is human readable, easy to parse or generate and language independent text format[9].
Custom API	A custom API enables you to define custom endpoints that expose server functionality that does not map to an insert, update, delete or read operation. By using custom API, you can have more control over messaging, including reading and setting HTTP message headers and defining a message body format other than JSON[6].
ADFS	Active directory federation services is a standards-based service that allows the secure sharing od identity information between trusted business partners across an extranet[10].
MVC	Model-view-controller is an architectural pattern used for developing user interfaces that divides an application into three interconnected parts[13].

1 Introduction

A mobile application or app is a software used on mobile devices such as smartphone or tablet. The official channel for mobile application purchasing depends on mobile platforms. For iOS users it is App Store and for Android users it is Google Play Store. Mobile applications have gained attention at a global level due to their ease of use and user-friendly interface. There is increased demand of smartphones over feature phones partly because of a boost in the mobile app industry growth. Furthermore, a lot of mobile applications are designed to be part of commercial and industrial merchandises and they could be a key factor for consumer when choosing a product.

ABB, as an industrial manufacturer company, has focused on digitalization for quite some time. Currently, several types of PC tools and mobile applications have been developed to supplement and support their products. For instance, Drivetune[4], as a mobile application, allows user to observe drive status and modify drive settings in distance by using Bluetooth connection.

Despite the fast growth and wider variety of mobile application, users are more demanding than ever. To stand out from similar apps, new features needs to be continuously added to the application and connection availability must be high as well. With Drivetune, user has the possibility to control their drives over Bluetooth; the drive can be up to 70 meters away from the mobile device. But when customers need a help drive, the traditional way of support is phone call or email exchange. If that is not enough, ABB local office or partners need to provide on-site support, which is time and cost consuming. To improve support availability, a new feature has been now added to Drivetune iOS app.

Remote Express is an ad-hoc communication tool. In addition to all the existing functionalities of Drivetune, the goal is to provide remote technical support 24/7. There are three ways of support Remote Express can offer:

1. Chat with ABB support team.
2. Share backups back and forth with ABB support team.
3. Give permission to ABB support team and allow them to view drive data in real time and make parameter change proposal.

The current status of Remote Express development has been implemented on iOS and pilot testing started last year. Based on the feedback from pilot testing and published statistical articles, it is important that Remote Express should support Android platform beside iOS platform. Research results and projections are listed as following:

1. The most recent statistic from Statista.com shows that there are around 3.3 million of available apps in the Google Play Store[1].
2. Another statistic from the same organization recognizes Android as the leading smartphone OS since 2011, the market share has grown rapidly since then and the most recent number from Q1 2018 is above 85%[2].
3. There are approximately 17k new Android mobile applications being ported from iOS[3] in 2017.

Not only is Remote Express a faster and more economical supporting method compared to traditional support, but it is also a safety issue as described below:

1. It is an ad-hoc tool so when mobile app is not connected to web portal, drive data cannot be monitored by anybody through the application.
2. When end user is connected to a support user in Remote Express support case, he or she has the control of when and how long the support user has access to drive data.
3. When support user sends a parameter change proposal, end user again has the power to decide if the proposal should be applied and when to apply it to drive.

All these safety protocols can make sure that sensitive information is not accessible without permission.

2 Usage of External Libraries

Before Remote Express features, Drivetune is fully functional offline because it relies on Bluetooth communication. But for Remote Express, it needs to talk with the mobile backend which is in the cloud and provide services to the application that is running on the phone. This makes Remote Express a cloud connected mobile application. Examples of such application are Instagram and Facebook where cloud is used for photo and news feeds storage. For Remote Express, chat messages and backups files are stored in the cloud. Azure Mobile Services has been chosen as the backend because it encapsulates all the features of mobile service, such as push notification services, API hosting and provide mobile SDKs. To talk with backend, new external libraries must be imported.

2.1 Azure client SDK

When this project began, the first thing to study was how the application is running on iOS platform, and what external libraries are used. The conclusion is that Remote Express on iOS uses MVC model[16] and Azure client SDK[5] for all API requests. Azure client SDK also supports Android platform so ideally all API requests can be made with this library. The official tutorial[6] explains how to communicate between app and Azure Mobile services, the best practice is to have a singleton class and initialize a mobile service client once the app is launched. The advantage of Azure Mobile App SDK library is that when data class is defined corresponding to the tables in the mobile app backend, the response from backend would be converted automatically to the data model that is defined in the app (class name is the same as SQL Azure table name).

The first step is importing the dependency in Android build.gradle file and adding a line to enable internet permission to AndroidManifest.xml. The next step is to create a class to initialize the mobile service adapter giving the URL of mobile backend. This adapter is initialized as a singleton and to obtain a reference of it getInstance() function must be called every time, as shown in figure 1.

```
//To initialized an Azure service adapter

private AzureServiceAdapter(Context context) {
    mContext = context;
    mClient = new MobileServiceClient(
        mMobileBackendUrl, // The URL of Remote Express backend
        mContext);         // Application Context
}

public static void Initialize(Context context) {
    if (mInstance == null) {
        mInstance = new AzureServiceAdapter(context);
    } else {
        throw new IllegalStateException("AzureServiceAdapter is already
        initialized");
    }
}

public static AzureServiceAdapter getInstance() {
    if (mInstance == null) {
        throw new IllegalStateException("AzureServiceAdapter is not
initialized");
    }
    return mInstance;
}
```

Figure 1. How Azure service adapter is initialized and used.

Here is the Enduser data class, the serialized names “id”, “name” and “email” are also the column names in EndUser table. Similarly, SupportUser, SupportCase and ChatMessage data classes are created as demonstrated in figure 2.

```
public class EndUser
{
    @com.google.gson.annotations.SerializedName("id")
    private String mId;

    @com.google.gson.annotations.SerializedName("name")
    private String mName;

    @com.google.gson.annotations.SerializedName("email")
    private String mEmail;

    public EndUser() { }

    public EndUser(String id, String email) {
        this.setId(id);
        this.setEmail(email);
    }

    public String getId() { return mId; }
    public final void setId(String id) { mId = id; }


    public String getName() { return mName; }
    public final void setName(String name) { mName = name; }


    public String getEmail() { return mEmail; }
    public final void setEmail(String email) { mEmail = email; }

}
```

Figure 2. EndUser data class.

In Enduser controller the application can execute queries on the table reference.

```
//To get a SQL table

MobileServiceTable<EndUser> mEndUserTable =
mClient.getTable(EndUser.class);

//A query without filter

List<EndUser> results = mEndUserTable
    .execute()          // Returns a ListenableFuture<E>
    .get()              // Converts the async into a sync result

//A query with filter:

List<EndUser> results = mEndUserTable
    .where()
    .field("email").eq("example@example.com")
    .execute()          // Returns a ListenableFuture<E>
    .get()              // Converts the async into a sync result
```

Figure 3. End user query using Azure client SDK.

Because the response from backend is already converted to the data class that is defined, it is very convenient to use. For support case to be created, POST query has to be called but a problem was found which made Android Azure client SDK incompatible with Remote Express backend. Azure mobile apps backend table defines five special fields: “id”, “updatedAt”, “createdAt”, “version” and “deleted”. While the first four are available to clients, the last field should not be defined by client, thus “deleted” should not be a property in any of the data class. But Remote Express backend is just a proto type and it requires “deleted” to be included in the POST query, furthermore the reason why Remote Express on iOS platform could use Azure client SDK is because it sends JSON object instead of data object. Therefore, for this project another library that could send JSON as a query body was needed.

2.2 Fast Android Networking

Fast Android Networking library is an open source library on Github[7] and it is built on top of OkHttp Networking Layer[8]. The advantage of using Fast Android Networking over Azure client SDK is that the developer does not need to create asynchronous callback to each HTTP query because this library handles it in the background by default. Another

advantage is that Fast Android Networking can send any object as header or body of the query. In the case of Remote Express, “deleted” field can thus be included in an JSON object and backend accepts it. For each data class, two additional functions are needed, one to convert an object to JSON and another one for JSON to object conversion. To improve the reusability of the code, an abstract class called DataModel was created. It has properties that each Remote Express data class has. In each data class, additional properties are added. E.g. Email is a unique property of end user and support user. See figure 4 below:

```
public abstract class DataModel {

    JSONObject json;
    DataModel(JSONObject json){
        this.json = json;
        try {
            if(this.json == null)
                this.json = new JSONObject();
            if(!json.has("createdAt"))
                json.put("createdAt", Calendar.getInstance().getTime());
            if(!json.has("deleted"))
                json.put("deleted",false);
        }catch (Exception e){}
    }

    public String getId() {
        return json.optString("id");
    }

    public String getCreatedAt(){
        return json.optString("createdAt");
    }

    public String getUpdatedAt(){
        return json.optString("updatedAt");
    }

    public boolean isDeleted(){
        return json.optBoolean("deleted");
    }

    public JSONObject toJSON(){
        return json;
    }

    public void fromJSON(JSONObject json) {
        this.json = json;
    }
}
```

Figure 4. The Java extension class that is used by all Remote Express data class.

For each GET and POST query, the following things need to be specified:

1. The full URL (include mobile backend URL, the table name and filter if necessary)
2. Request header (ZUMO-API-VERSION, Content-Type, etc.)
3. Callback to respond to backend response and trigger UI changes

```
//To specify the http request header

public JsonObject getRequestHeaders() {
    JsonObject requestHeader = new JsonObject();
    requestHeader.addProperty("ZUMO-API-VERSION", "2.0.0");
    requestHeader.addProperty("Content-Type", "application/json");
    requestHeader.addProperty("Accept", "application/json");
    return requestHeader;
}

//GET request function

public void getRequest(String path, JsonObject header, final
MobileServiceRequestCallback callback) {
    AndroidNetworking.get(mMobileBackendUrl + path).addHeaders(header)
            .build().getAsJSONArray(new JSONArrayRequestListener() {
        @Override
        public void onResponse(JSONArray response) {
            if(callback != null)
                callback.onRequestSuccess(response);
        }

        @Override
        public void onError(ANError error) {
            if(callback != null)
                callback.onRequestFailed(error);
        }
    });
}
```

Figure 5. An example of get request header and get request using Fast Android Networking.

The challenging side of using Fast Android Networking is to find the full URL, when retrieving the end user from backend. In addition to backend URL and path to the end user table, a filter must be specified as well, e.g. ?$filter=Email eq ‘email’ is the filter for email address. See figure 5 above.

2.3 Apache HttpComponents Client Library

Apache HttpComponents client library[15] is used in this project parallel to Fast Android Networking is because of PATCH request. For some chat messages, PATCH request is used to update some fields in the ChatMessage table after user. After the user interacts with clickable chat message (connection request message) in Remote Express, the visibility field of ChatMessage table needs to be updated using PATCH request. In Fast Android Networking, although it states that all types of HTTP/HTTPS request like GET, POST, DELETE, HEAD, PUT, PATCH are supported. In this project it was realized that PATCH request is an extension of POST request. Remote Express backend requires PATCH request for updating a table, so this library is imported in addition to Fast Android Networking. Like Fast Android Networking, Apache HttpClient also needs full backend URL and specified request header. Similar to Azure client SDK, this library requires asynchronous callback to handle HTTP request.

2.4 Azure Storage SDK

Azure Blob storage[14] is a storage solution for cloud. It is optimized for storing unstructured data, such as binary data[17]. Azure Storage SDK is used for handling upload and download backups to/from Azure Blob storage. When a backup is shared, a new entry in backup SQL table is created first. Information such as created time, drive model is saved, but the actual backup file is not saved in the table. One of the fields in backup table is called data and it contains the unique reference generated for that backup. After the application gets a call back from the POST request, it sends the unique reference and backup bytes to Azure Blob storage.

Later when the user downloads a backup from Azure Blob storage, it first finds the unique reference in the data field from the backup table and then goes to the blob storage to look for the file that is corresponding to the reference found in the backup table. To connect application with blob storage, connection string and blob storage container name are also needed.

3 Application UI

3.1 Remote Express Workflow

Remote Express service contains two parts, mobile application for ABB customers with malfunctioning drives and web portal for ABB support team. The main goal of this project is to transform Remote Express iOS app to Android platform and study how to make it better. Functionalities worth mentioning are live chat, backup sharing and parameter exploring on web portal. The connection from app to web portal is handled by Azure mobile services, and all communications such chat messages and any other messages with parameter values are stored in Azure SQL database for quality assurance and future service improvement analysis.

The following paragraphs will explain the workflow of Remote Express briefly. It begins when issues with drive are detected by an on-site engineer who monitors and controls the drive, he or she opens Remote Express on the phone, becomes an end user and gets remote support from one or more support users. Once end user is authenticated, a support case ID will be created, and a chat view is shown to end user. At the same, app starts to send heartbeats to web portal every five seconds to notify the support user that this end user is online until the case is closed. This helps the support user to discover if any issue, such as a networking problem, is blocking the system or the end user leaves from Remote Express. Inside the chat view two people can write messages to each other. Photo and voice message sharing is not included in this project, but they will also be supported in the future, as shown below in figure 6.

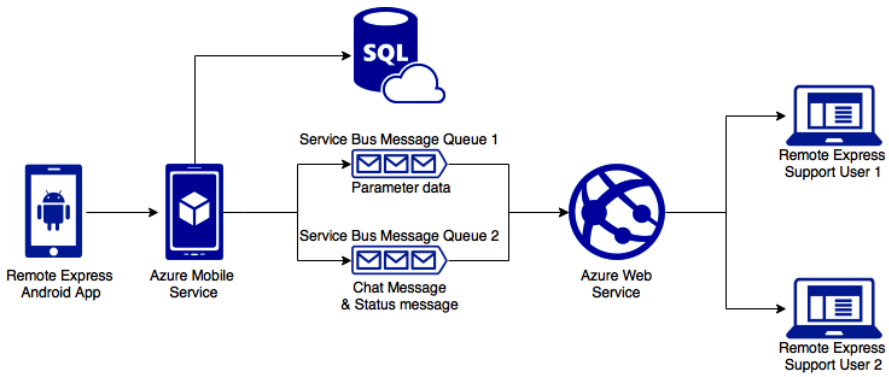

Figure 6. When mobile is sending message to Web.

If the drive issue cannot be solved by message exchanging, backup sharing is another functionality Remote Express provides. For backup sharing, user needs to create a parameter backup file and upload it to the cloud. When the backup file is successfully uploaded, support user will be notified. Similarly support user can also upload backups and when that happens, end user will be notified as well. Once a backup is uploaded, it can be downloaded on both ends at any time. This backup file is compatible with ABB PC tool (Drive composer). After a support user downloads it, he or she can restore it to another drive and simulate the issues that the end user is facing. This solution is good for users who do not have internet connection near the area where the drives are, or if the problem is complex and requires further investigation. Once a solution is found, the support team can then send the working backup file to end user and this backup will be restored to the original drive.

When internet speed is reasonably fast, the alternative way to provide support is real time parameter monitoring on web portal and parameter change proposal to end user. Real time parameter monitoring is covered in this project, but parameter change proposal is not in the scope. When ABB support user sends a connection request from web portal to mobile, the end user can choose to grant the connection permission or deny it. Once the permission is given, the application will send another type of heartbeat every five seconds to web portal to notify that web portal can access parameters. At the beginning, drive type, serial number and a list of parameter group names will be sent to web portal and displayed to the support user. Once the support user clicks on any group name, the mobile device will send all the parameter details within that group to web portal as indicated in figure 7.

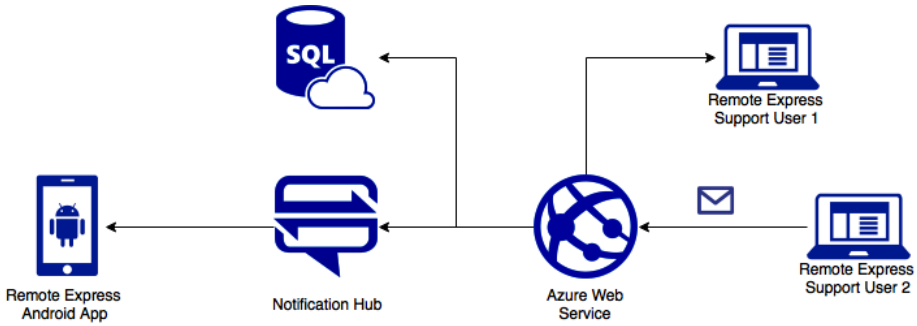

Figure 7. When web is sending message to mobile.

Support user can add specific parameters to favorite list and enable auto-updating to monitor them in real-time. Support person can also send modified parameter values back to mobile and the mobile user can decide if he or she wants to apply the new values to drive. Once the support is complete and mobile user exits from Remote Express, his or her status will change to offline.

3.2 Log in

The user interface of log in page is shown below in figure 8. End user can go to login view after a single click on Remote Express icon in home view, log in page is very simple, there are two editable text field where end user should put name and email address. At the moment there is no authentication, the only check is to compare whether that email exists in EndUser table. If it does, app will remember it and user does not need to write it next time when log in. If the email does not exist in the database, an error prompt will appear.

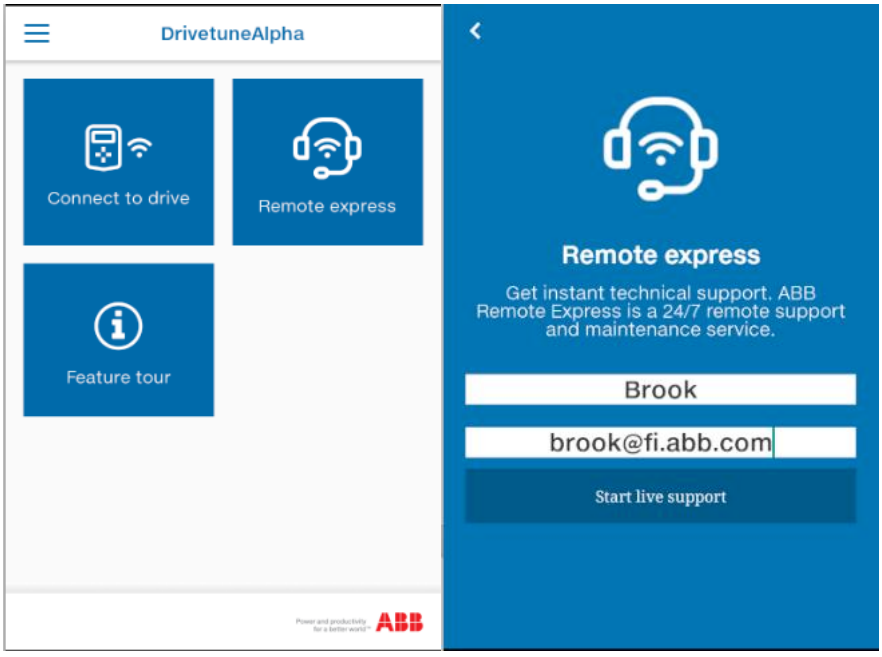

Figure 8. When web is sending message to mobile.

A total of five HTTP requests are sent in log in view, they are listed in the table below.

Table 1. HTTP request in log in view.

Request type	**purpose**
Get	Check user email is valid
Get	Assign a support user to
Post	Create a support session between the end user and mobile user
Post	Post a welcome/session start message
Post	Start to send heart beat to web portal, this indicates that end user is online

As soon as the last request is sent, support user can notice mobile user status change from offline to online. When heartbeats arrive continuously to web portal, support user can have the awareness whether a mobile user is online. Heartbeat message will not be posted to database.

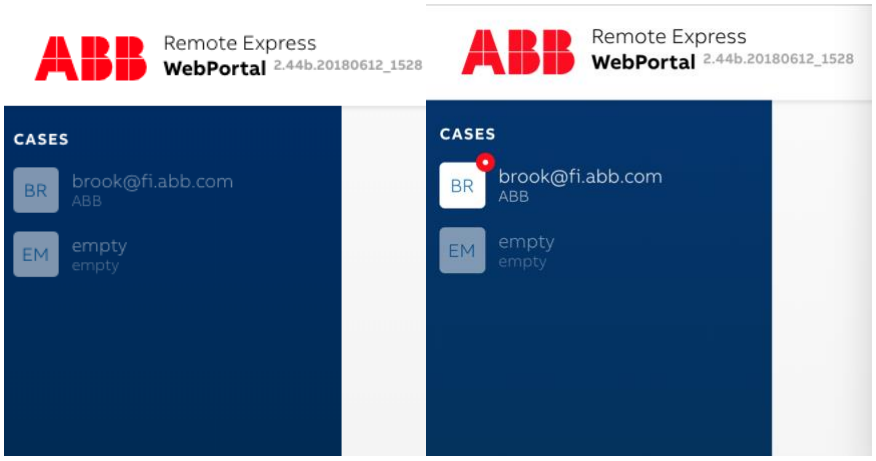

Figure 9. Mobile user status: offline and online.

3.3 Chat View

Once a support case is created, user will be redirect to dashboard view, where there are three tabs. By default, when user enters dashboard view chat tab is selected, the first chat message is always a welcome message.

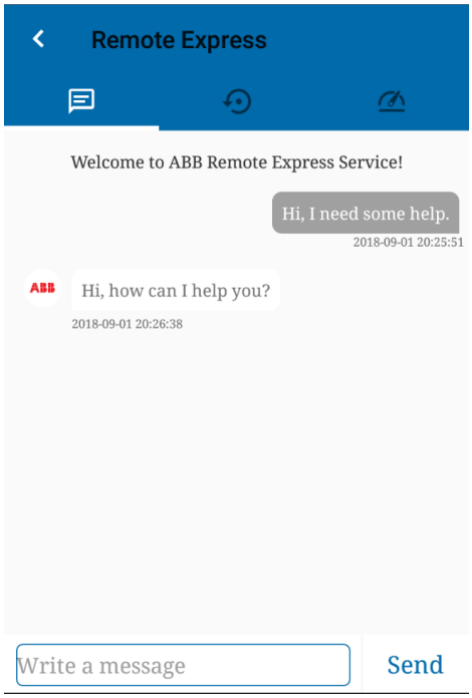

Figure 10. Chat Message example.

Chat view UI is similar to other popular chat apps, with a recycle view to display messages and one input field at the bottom. Chat messages bubbles are on different side and have different color for received message and sent message.

Different message types are defined for Remote Express, message type determines whether message is visible to end user, how message is displayed and if user can click on the message. In ideal situation, the end user should be notified as soon as any message arrives from a support user by push notification, but for this project an alternative way is used. A periodic task runs every five seconds for retrieving new message from ChatMessage table. See table 2 below.

Table 2. Message types.

Message type	Purpose
19	Welcome message
1	Normal chat message
3	Request drive connect from web portal
4	Grant drive access from mobile to web portal
15	Deny drive access from mobile to web portal
12	Support user ask for parameter from a specific group
13	Mobile user sends parameters requested by message type 12
16	Backup uploaded to cloud
17	Stop connection between web portal and drive

Backup sharing is can also be reflected in chat view, once end user clicks upload button of a specific backup, a message will also be sent to support user that a new backup is shared. The same message can also be sent from support user to mobile user from web portal. This message is clickable and end user will be redirected to backup detail view. See figure 11 below.

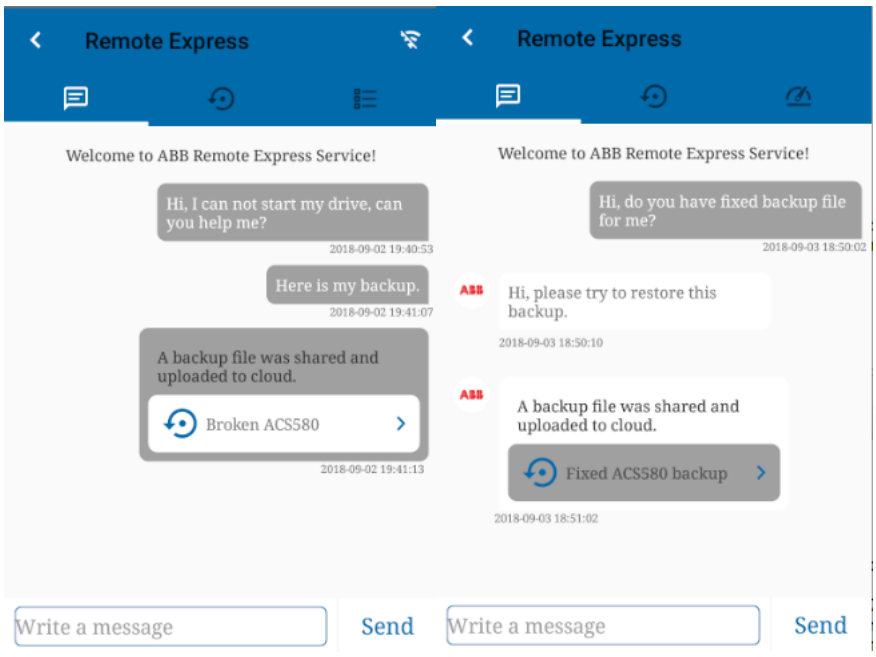

Figure 11. Backup message example.

The second way of support is to read drive parameters on web portal in real time. This requests the end user to give drive access when support user sends a connection request. The advantage of this method is that it does not require additional tool to open parameter backup file and it shows real time value. When the end user receives a connection request, he/she can accept or deny it. The support user can not send a second connection request until one of the following situations happen as shown below:

1. End user denies the request.
2. Support user cancels the first request.
3. End user does not deny or accept the active request within two minutes.

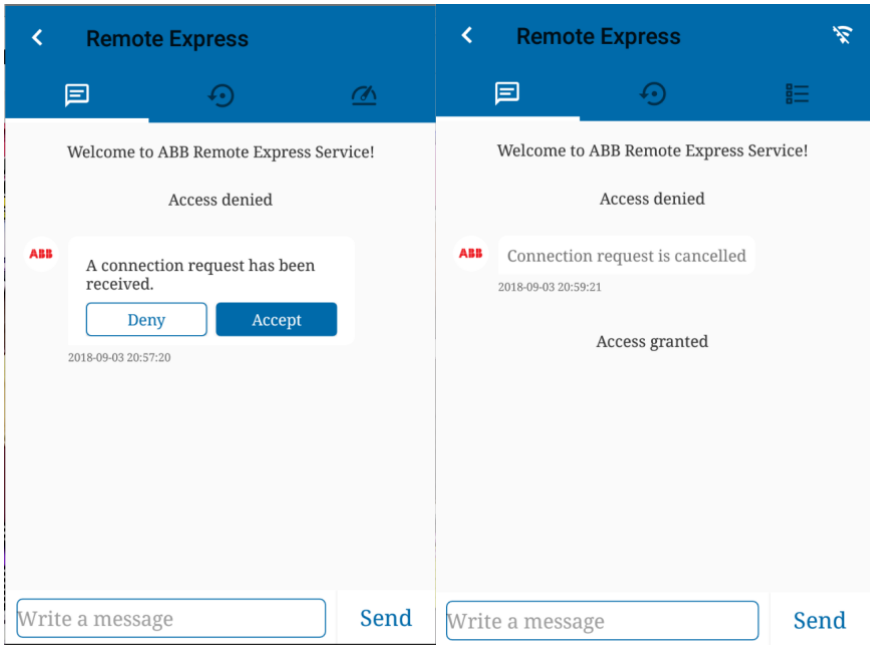

Figure 12. Examples of connection related message.

As soon as the end user grants drive access to the support user, the application will send a message to web portal. It includes drive information such as name and serial number, and most importantly the name of each parameter group's name. And as mentioned above app sends drive heartbeats in the background every five seconds until the support session ends or one side stops the connection. See figures 13 and 14 below.

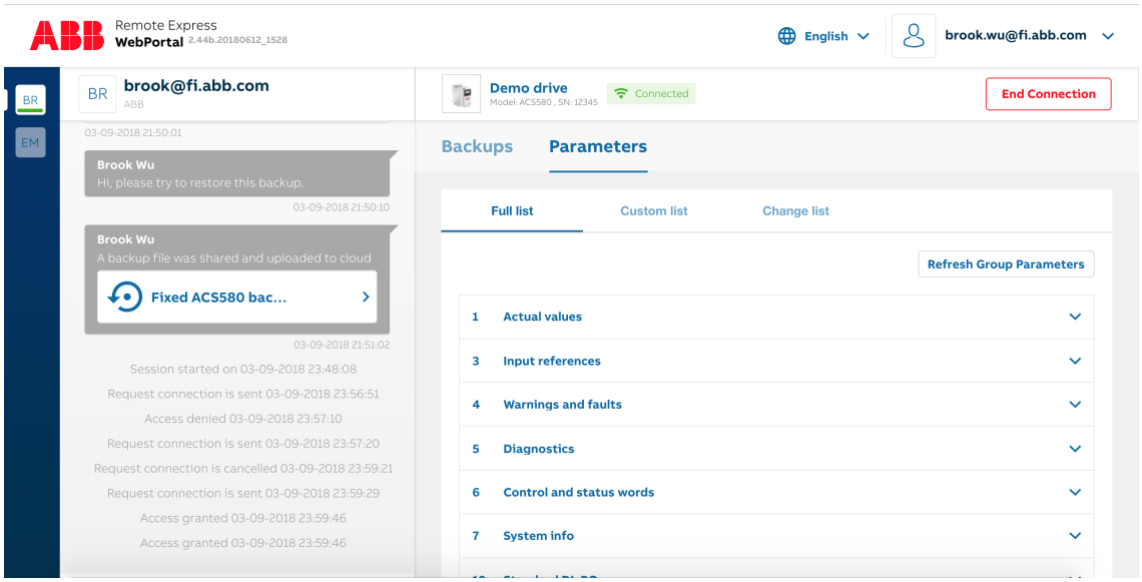

Figure 13. When end user grant access to support user.

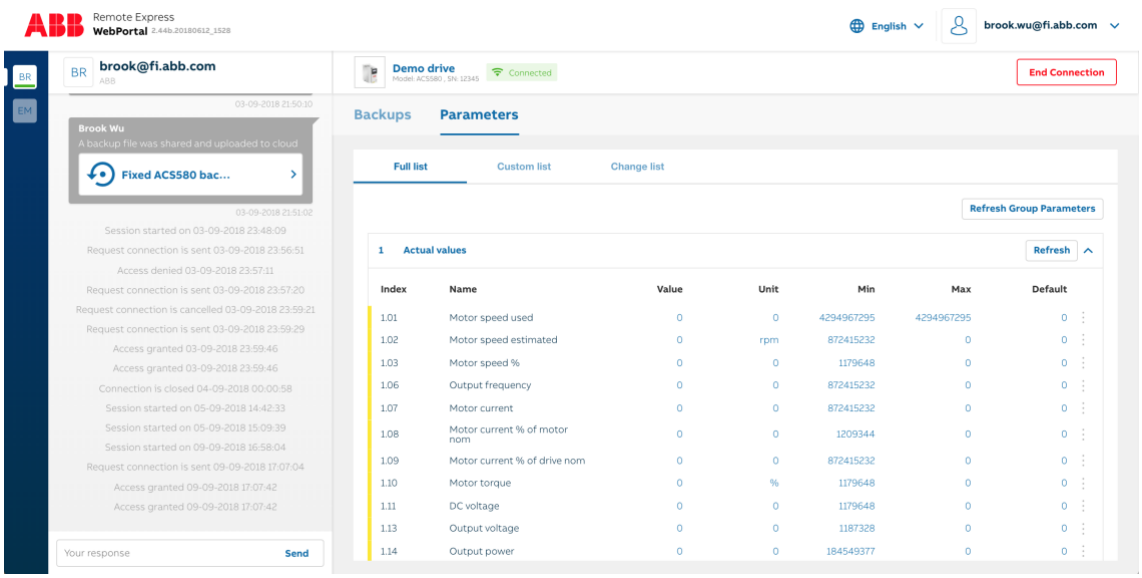

Figure 14. Example of parameters in group 1 of demo drive.

When the support user wants to read the value of specific parameters, he or she can click on the parameter group name to open parameter list within that group. When a group is clicked, the web portal sends a message to mobile with the group index and the app will reply with a message with all the details of that groups.

3.4 Backup View

As mentioned in the previous section, backup sharing and restoring is an important part of Remote Express. In Remote Express backup view, mobile user can see backups from three sources:

1. Local backups stored in mobile phone.
2. Online backups uploaded from mobile to cloud.
3. Online backups uploaded from web portal to cloud.

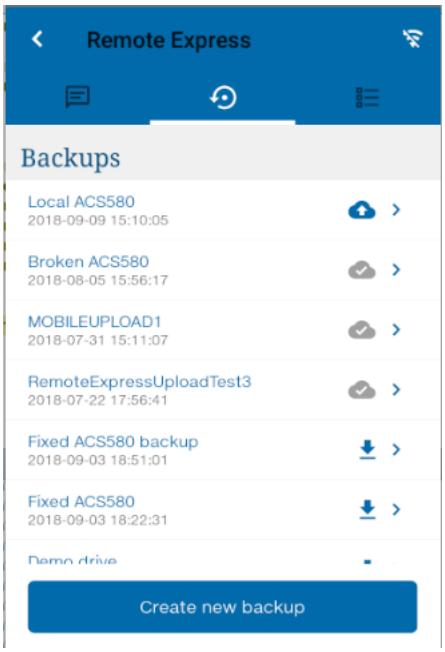

Figure 15. Remote Express backup view.

As indicated above in figure 15, local backup could be a backup that is created before the Remote Express session has started or after the session has begun when support

user is requesting one. As long as the backup is not uploaded to the cloud, it is only visible to the end user. In other words, support user can only see backups from second and third sources.

In the list view, next to the name of each backup, there is an icon for indicating the source of that backup file. There can be three kinds of status:

1. means backup exists only in mobile phone
2. means backup exists only on the cloud
3. means backup exists both in the mobile and on the cloud

In the first and second case, the user can tap on the icon, app will access the blob storage and either uploads or downloads the selected backup. Create new backup button at the bottom of this view leads mobile user to another activity and make a local backup.

3.5 Backup Detail View

When mobile user selects a backup from the list, backup detail view will be opened. This is demonstrated in figure 16 below.

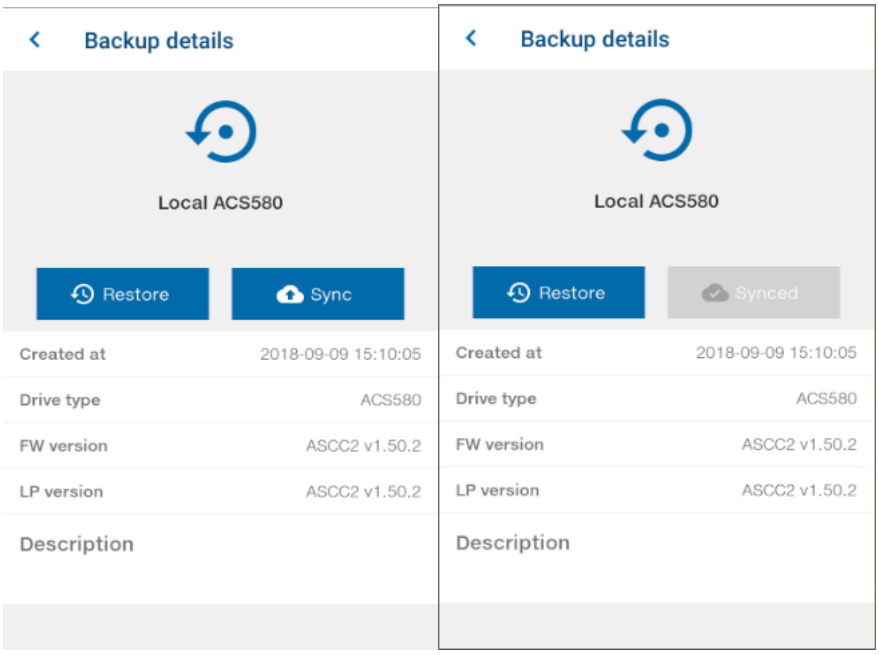

Figure 16. Remote Express Backup Detail View.

Button on the left is a restore button, it is active only when backup is stored locally and is compatible with the connected drive. Button on the right is a sync button which is disabled when the backup is stored both locally and remotely.

3.6 Applied History View

This is the last view in Remote Express in this project, as shown in figure 9, when support user can read drive parameters on web portal. If the support user notices that one or more parameter values should be changed, he/she can send a parameter change proposal to the end user who will decide if the change should be applied. Due to time limit, the apply change proposal functionality is not implemented, but the previous change proposal history is displayed when the app is paired with drive. Parameter change proposal history is serial number specific. Therefore, suggestions sent from any support user to the same drive will be displayed here as shown in figure 17.

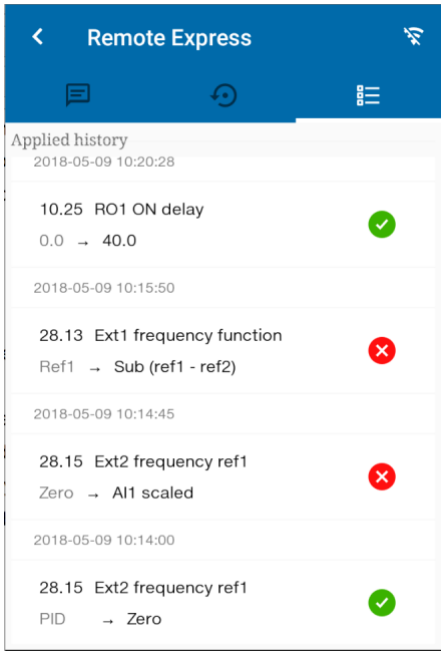

Figure 17. Remote Express Backup Detail View.

4 Future Studies

4.1 User authentication, authorization and API access token

Because in Remote Express the user will be browsing sensitive information such as drive parameter backups and parameter change history, user authentication and authorization is definitely one thing that must be implemented.

On Remote Express web portal, Active Directory (ABB's ADFS) is used for authentication, and therefore, the same functionality should be implemented in mobile app as well. When the end user wants to access Remote Express on mobile, he/she is redirected to ABB's ADFS page and log in is required. Once the user is authenticated, his or her email will be checked in Remote Express EndUser table. If this end user is found in the table, access is then granted, and a new support case is created as described in the earlier chapter.

Another thing that could be improved is to take access token into use. At the moment, the API can be called from anywhere by anybody. Access token can be used to restrict who, and which application is authorized to request from the database.

4.2 Push Notification

In the iOS application, push notification was implemented, so as soon as a new message is entered into ChatMessage table, it is pushed to end user (mobile app) as the workflow in figure 2 demonstrates. This is a missing functionality in this project. When Remote Express app is in front, it may not be so critical that the app sends a request every five seconds to backend and checks if there are any new messages. But if the end user puts the phone aside and puts the app to background, push notification can help to notify the end user that a new message has arrived. Also, when the app is used by more and more users, sending an API call every five seconds from every phone could affect the performance of backend. Firebase cloud messaging is a widely used for instant messaging[11]. Azure notification hub can be connected to FCM and deliver messages instantly to mobile users[12]. With Firebase cloud messaging, the application can send a push notification to a single user, a group of users or all users.

4.3 API calls

When implementing POST request, I noticed that in order for web portal to receive the message from mobile app, two API calls are sent as shown in figure one: one for database entry and one for notifying web portal with the same content. The second API call is a custom API call trigged by the callback of first request, which tells that the data has been successfully entered into the table. This could be optimized so one API call could handle both data entry and a web portal notification task.

When using Azure database tables, besides the column created by the database administrator, Azure mobile application's backend table defines five special fields, those are ”id”, “updatedAt”, “createdAt”, “version” and “deleted”. As mentioned in section 2.1, “deleted” should not be a property of any data class. If the backend could be improved so that it is compatible with Azure client SDK on Android, and API call would be much easier. For example, the request header is not needed, and it supports all HTTP request types that are needed for this application. As a result, only a single library is needed for this application.

4.4 Unit Test and User Interface Test

Quality assurance is an important part of application development, and unit test is the fundamental part of application testing strategy, because it helps to verify that the logic of each unit is correct. UI test, on the other hand, tests the quality of the app by performing a set of user operations. One challenging UI test could be useful for Remote Express: running mobile UI and web UI test simultaneously. Two ends perform actions alternately; for example a mobile UI test sends a chat message and soon after a web UI test to check that this message has arrived to the browser.

5 Conclusion

In recent decades, digitalization has been considered as another revolution in manufacturing industry since it may automate processes, measure different aspects of business that were previously unmeasurable and even cross reference them. With the help of digitalization we can also apply information from a wide variety of sources to provide insight into decision making. In addition, digitalization can bring new opportunities, new markets and new levels of efficiency and cost reduction from a manufacturer's perspective. ABB being one of the traditional manufacturers, realized the trend early on and has been willing to embrace the change.

Based on ABB's requirements, instant chat, backup sharing as well as parameter browsing on web portal have been implemented for this project. Yet some Remote Express iOS features are missing, such as push notification and parameter change proposal from support user to end user. Due to easy cross-platform application development and maintenance reasons, the final application will be using JavaScript language. However, it was a valuable experience to have the opportunity to complete this project. I believe Remote Express can significantly speed up technical support and reduce the cost of on-site support. Through completing this project, I have learned a lot. I have a much better understanding of MVC model and design patterns such as singleton. The structure of this application follows MVC models, where a model such as end user is singleton which can only be initiated once. Another thing I learned a lot about is API calls and how to troubleshoot them; in fact, this was the biggest issue I faced when working on this project. It was time-consuming to send requests using Azure client SDK, which was not compatible with Remote Express backend API interface. However, this issue was solved by using different external libraries. There are still many areas that can be improved to optimize the performance of Remote Express.

References

1 Google Play: number of available apps 2009-2018: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

2 Global market share held by smartphone operating systems 2009-2018: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

3 https://blog.appfigures.com/ios-developers-ship-less-apps-for-first-time/

4 Drivetune Android application: https://play.google.com/store/apps/details?id=com.abb.spider

5 Android Azure mobile services: https://github.com/Azure/azure-mobile-apps-android-client

6 Android Azure mobile services tutorial: https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-android-how-to-use-client-library

7 Fast Android Networking: https://github.com/amitshekhariitbhu/Fast-Android-Networking

8 OkHttp Networking Layer: http://square.github.io/okhttp/

9 Introducing JSON: http://www.json.org/

10 ADFS introduction: https://msdn.microsoft.com/en-us/library/bb897402.aspx

11 Firebase cloud messaging introduction: https://firebase.google.com/docs/cloud-messaging/

12 Push notifications to Android devices by using Azure notification hubs and Firebase cloud messaging: https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-android-push-notification-google-fcm-get-started

13 Model-view-controller: https://en.wikipedia.org/wiki/Model–view–controller

14 Azure Blob storage: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction

15 Apache HttpComponents Client: https://github.com/apache/httpcomponents-client