Muhammed Masum Islam

Android Communication Over Azure Cloud

A project from Espoo City Corporation

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

15 October 2018

Helsinki Metropolia University of Applied Sciences

PREFACE

I would like to express my sincere gratitude to Mr. Kari Salo, Principle Lecturer at Metropolia University of Applied Sciences for supervising this thesis and for all the helpful criticism and advice. His incredible knowledge of the different aspects of Information Technologies as well as his profound academic approach to research problems was inspiring. Also, I am thankful to Ms. Anne Perälampi, Senior Lecturer at Metropolia University of Applied Sciences. Her English language guidance and advice was quite helpful.

I would like to express my appreciation to my co-workers at my workplace who have always shared their experience regarding the different perspectives of the subject area.

Last but not the least I want to thank my family for their support, especially my elder brother Mr. Muhammad Khurshed Islam and my wife Mrs. Sabrina Islam. Both of you offered continued support and encouragement during my entire Master's studies. And finally, my lovely daughter Mahi Islam's birth on 29.04.2018 highly inspired me to complete this thesis as well the degree in the same year. Year 2018 will be the most memorable year for our family.

Espoo, 15 October 2018
Muhammed Masum Islam

Author(s) Title Number of Pages Date	Muhammed Masum Islam Android Communication Over Azure Cloud 58 pages + 1 appendix 15th October 2018
Degree	Master of Engineering
Degree Programme	Master in Information Technology
Instructor(s)	Pekka Pakkala (Project Manager) Kari Salo (Supervisor)

Espoo City Corporation's (Espoon kaupunki) ground staff always needs to survey various locations for construction, renovation and cleaning. To complete a location survey there are several steps required. First, ground staff has to visit a location and take several pictures and note down what is required to be done for that location. Second, the staff must store pictures and data in database. Third, the managers need to be notified because they view data and picture as well as accept or reject the survey. Often managers ask for more data or clarification.

Because the process has always inaccurate and faulty, Espoo City Corporation managers decided to modernize their conventional survey. After long planning, they decided to use mobile devices such as smart phone and tablet as well as cloud technologies for the smart survey service.

The main target of this project was to simplify and automate the tedious and time-consuming manual process. It was assumed that if survey data and pictures are stored instantly from the survey site, it may make this process more accurate and faster.

After reviewing their process, problem and requirements, the proposal was to use Android devices and Azure cloud service, namely a client Android app for mobile phone or tablet and web services running in the Azure cloud. Azure cloud is the storage for all the data, pictures and maintains all relational databases. Also, Azure cloud is responsible for notifying any update automatically to all subscribers.

To achieve this goal, a client-server application was developed. The client Android app was built for location, push notifications, multimedia, current location, map awareness and the Azure cloud connection orientated. Azure cloud's binary large object (Blob) Storage was used to store pictures, and a relational database management system (RDBMS) was used to store survey data and Notification Hubs to manage push notifications.

As a result of this thesis project, an Android app (app name Photo location note) which usages Azure backend to perform survey digitally was developed. This app assists Espoo City Corporation staff to perform their survey rapidly in a smarter way.

Keywords	Android, Azure, Cloud, Push notification, Blob Storage, SQL

Table of Contents

List of Tables:-

List of Figures:-

Abbreviations

AWS	Amazon Web Services
Blob	Binary Large Object
CRM	Customer relationship monument
RDBMS	Relational Database Management System
IoT	Internet of Things
IaaS	Infrastructure as a Service
App	Application
MBaaS	Mobile Backend as a Service
APNS	Apple Push Notification Services
GCP	Google Cloud Platform
GCM	Google Cloud Message
WNS	Windows Push Notification Services
MPNS	Microsoft Push Notification Service
IDE	Integrated Development Environment.
HAL	Hardware Abstraction Layer
GPS	Global Positioning System
PaaS	Platform as a Service
NIST	National Institute of Standards and Technology
UI	User Interface
SDK	Software Development Kit
IDE	Integrated Development Environment
API	Application Programming Interface
OS	Operating System
ADT	Android Development Tool
ISO	International Organization for Standardization
HTML	Hypertext Markup Language
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SaaS	Software as a Service
LBS	Location-based services
APK	Android Package
AD	Active Directory
SDLC	Software Development Life Cycle

1 Introduction

The primary objective of this project was to create an Android app integrated with Microsoft Azure cloud service. This app was developed for Espoo City Corporation (Espoon kaupunki). Espoo City Corporation provides various services for Espoo city inhabitants such as constructions (e.g. houses, schools, roads), renovation, gardening, garbage collection and recycling. Before providing these services, there are lots of surveys, paper work and approval requirements to be taken care of.

Usually Espoo City Corporation staff performs survey by using digital camera, pen, pepper, etc. To present a real-life survey, let's imagine that they have to renovate a school. First, Espoo City Corporation's ground staff visits the renovation area for initial survey. During the survey they take pictures by digital cameras and write notes in paper. Usually the notes contain what needs to be done for that particular place or picture. After the survey, the staff returns to office to transfer photos from camera to computer. Then they try to match each photo and paper note into a database. This process is very tedious and erroneous. When all survey data has been submitted on the database, they notify their managers. This communication is done usually by email or instant message by messenger. When the managers check the data, it often contains errors and they must ask for more calcification. Again, the communication between managers and employees is carried out by email or instant messages. The problem is that this kind of communication cannot be stored and used for future reference although it would be necessary. For example, the final reports are generated based on this information.

After refining the survey, managers send it to the higher-level managers for approval. When a survey approved, the actual work is carried out. Bookkeeping and tracking all these papers manually is extremely difficult, time-consuming and error-prone. Espoo City Corporation is always trying to improve their service. That is why they want to survey, audit and tack by computers, smart phones and tablets.

The traditional working method is tedious, backward, and sluggish. This process consumes more time and delays the actual work. Obviously, there is much room for improvement and consequently, the managers decided to digitize this manual work by using smart phone or tablet.

In order to digitize the survey, the survey data has to be stored in the database. This requires technology such as web servers, database servers, smart phones, tables etc. Espoo City Corporation is not an ICT company and they do not have enough money to hire a new server administrator to maintain those servers. Also, over time this kind of server hardware and software needs to be upgraded. To avoid this kind of maintenances, managers decided to use cloud service.

2 Background

Manual surveys contain lots of human errors which eventually produce inaccurate surveys. When Espoo City Corporation's staff performs surveying to renovate streets, it is a tedious task. First, they have to find out about the condition of a particular street which requires in-depth analysis. This analysis always has many paper notes, location photos and budget estimations. It is very hard to keep track of the paper notes and digital camera photos and as a result, there is lots of errors. Eventually they have to produce a final report based on the data. If a report is based on erroneous data, the report will be incorrect. Table 2.1 shows how manual survey can be improved by digital survey:

Table 2.1: Manual survey and digital survey comparisons

Manual survey	Digital Survey	Improvement
Survey done using digital camera, pen, paper, etc.	Survey done using Android application either by smart phone or tablet.	No need to match paper note and digital photo if we use Android app. Because app combines both photo and note together.
It requires photo transfer from digital camera to computer.	Android app automatically uploads photo to sever.	This saves time and avoids the risk of data corruption.
It requires manual data entry into database.	Android app automatically inset data into database.	Again it saves time and avoids the human mistake.
Staffs needs to inform the mangers about the survey by email or instant message.	Android app is smart enough to broadcast push notification to all interested user.	No need to spend extra time for notifications. Notification done automatically.
It is hardly possible to store the communication between staffs and managers. It is important to store any communication done between them.	Android app can store all communication logically. It is organized structurally which is easy to follow in future.	All communication saved automatically for future reference.
Sometimes it is required to	Android app can instantly	It is inconvenient to store

contact the manager form the survey location with survey data. Nowadays it is done by instant messengers such as Skype, WhatsApp, etc.	update all survey data to Azure cloud and send notification to everybody who is subscribed for that particular survey.	data which is sent over instant messengers. There is no need to store data if Android app is used. All data uploaded to Azure cloud automatically.

It would be wonderful if digital camera had an option to enter text note together with photo. That is why Espoo City Corporation management decided to improve this process by using modern technology where there will be possibilities to take a photo together with a note. To keep track of the note and photo, the photo should stamp the current location and date. Also an option to save all data and photo in a central location is necessary. It is mandatory that important data can be shared by authorities and everyone has access to it in the organization.

2.1 Android Application Development

Android application development is a process where a new application is developed for the Android operating system (OS). Application is usually developed in Java programming language and Kotlin programming language using the Android software development kit (SDK). Other development environments are also available, such as Xamarin android, Apache Cordova, etc. Also nowadays cross platform hydride Android application which is mostly developed by JavaScript is famous. In this project, Java programming language, Android Studio Integrated Development Environment (IDE) were used.

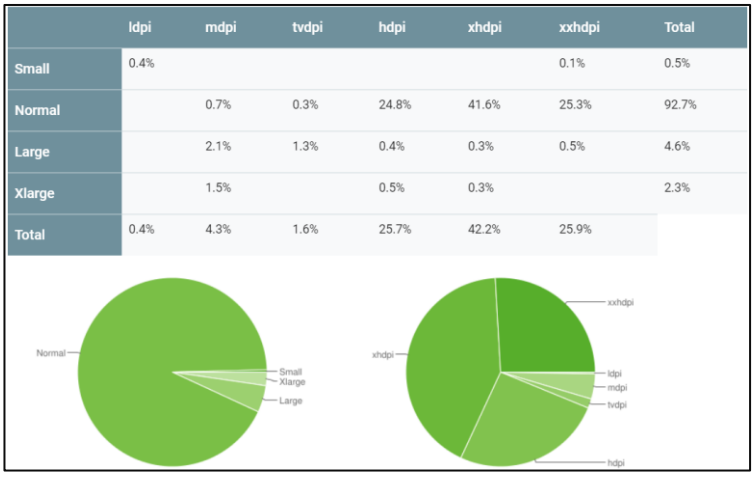
The target of the development is to support the latest devices and screen size in the market. Android executes on a many types of devices that have different pixel densities and screen sizes. The Android system performs resizing and scaling to make it possible to process material to different screens. However, more work to ensure UI for each type of screen is needed.

2.2 Screen Sizes and Densities

Android devices come in all shapes and sizes, so the app's layout needs to be flexible. That is, instead of defining the layout with rigid dimensions that assume a certain screen size and aspect ratio, the layout should gracefully respond to different screen sizes and orientations.

By supporting as many screens as possible, our app can be made available to the greatest number of users with different devices, using a single APK. Additionally, making our app flexible for different screen sizes ensures that the app can handle window configuration changes on the device.

This section provides data about the relative number of devices that have a particular screen configuration, defined by a combination of screen size and density. To simplify the way developers design the user interfaces for different screen configurations, Android divides the range of actual screen sizes and densities as expressed by the figure 2.2 below. Data collected during a 7-day period ending on September 28, 2018.



	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	0.4%					0.1%	0.5%
Normal		0.7%	0.3%	24.8%	41.6%	25.3%	92.7%
Large		2.1%	1.3%	0.4%	0.3%	0.5%	4.6%
Xlarge		1.5%		0.5%	0.3%		2.3%
Total	0.4%	4.3%	1.6%	25.7%	42.2%	25.9%	

Figure 2.2 Screen sizes and densities

On the above picture we can see the ration of devices depending of the screen size and densities.

2.3 Current Application Support

Currently this application supports following resolution, density, and size:

- Resolution = number of pixels available in the display, scale-independent pixel = sp.
- Density = how many pixels appear within a constant area, dots per inch = dpi.
- Size = amount of physical space available for displaying an interface.
- Density-independent pixel = virtual pixel that is independent of the screen density, dp.

Table 2.3 Density classes [1]

Class	Name	Density	Factor
Mdpi	Medium density	160 dpi	sp = dp
Hdpi	high density	240 dpi	sp = 1.5 x dp
xhdpi	extra high density	320 dpi	sp = 2 x dp
xxhdpi	extra extra high density	480 dpi	sp = 3 x dp

We can see on table 2.3 [1] various supported screen size and density devices range. Following table 2.3 [2] shows the screen types, size, respective folder with typical device examples.

Table 2.3 Screen size classes [2]

Class	Size in dp	Layout Folder	Examples
Normal	470x320 dp	Layout	typical phone screen (480x800 hdpi)
Large	640x480 dp	layout-large	tweener tablet like the Streak (480x800 mdpi), 7" tablet (600x1024 mdpi)
xlarge	960x720 dp	layout-xlarge	tablet (720x1280 mdpi, 800x1280 mdpi, etc.)

To support various screen size in Android application, we have to create layout folder under the resource folder. Above table Screen Size Classes [2] shows Android application layout fold names. Also we have to add all icons with respective regulations.

The following API versions are supported:

- **Minimum API level**
  - API level: 19
  - Code name: Kit Kat
  - Version: 4.4

- **Maximum API level**
  - API level: 27
  - Code name: Oreo
  - Version: 8.0

2.4 Azure Cloud Service

For digital survey service, data needs to be stored in the server. A decision to use Azure cloud service as storage was made. There are several reasons for choosing Azure cloud such as binary large object (Blob) storage to store picture, relational database management system (RDBMS) and deliver push notifications to the users. Azure provides many features but, in this project, only the following features from Azure cloud will be used:

- **Azure SQL Database:** Azure SQL Database is an organized cloud database for software developers that allow developing and maintenance applications easier and constructive way. Azure SQL Database is service that allows speedily creates, enlarge and extent relational applications into the cloud system. Azure SQL Database automatically expends spaces, security, backup, replication, etc.
- **Azure Storage:** Azure Storage offers following types of storages:
  - Binary Large Object (BLOB): Massively extensible object repository for unstructured data such as audio, video, picture, etc.
  - Table: Key-value table repository.
  - Queue: Strong queues for massive-volume cloud services.
  - Disk: Premium storage for input/output exhaustive applications.
  - File: Manageable, distributed cross-platform files system

  For survey service I will use Binary Large Object (BLOB) to store digital photo.

- **Azure DocumentDB:** DocumentDB is Microsoft's newest NoSQL document database platform that runs on Azure. DocumentDB is designed keeping in mind the requirements of managing data for latest applications. DocumentDB is an organized unstructured object database that retrieving and updating over the schema free data, certain and durable achievement and speedy development. DocumentDB is used to store final survey report. This is the central location for report. These reports are accessible from the web application. It certainly can be characterized as a typical NoSQL document database. It's massively scalable, and it works with schema-free JSON documents.

- **Azure Mobile Services:** Azure mobile services backend is for mobile applications. Mobile backend as a service (MBaaS) intend to minimize the complication of mobile application development by the contribution of cloud repository, authentication, push notification and similar type of cloud services. They depend on RESTful programming interface to access data. Access data means creating, reading, updating, deleting (crud) operations based on HTTP methods GET, PUT, POST, and DELETE. It also offers authentication services using OAuth protocol. Azure mobile services are used to access data from Azure. Mobile service allows secure access to Azure backend.

- **Azure Service Bus:** Service Bus allows couple of message-oriented middleware mechanism includes dependable message queuing and reliable publish or subscribe messaging. These brokered messaging possibilities can be seen as decoupled messaging option that supports publish subscribe and load balancing scenarios using the Service Bus messaging fabric. Loosely coupled communication has much superiority. For example, clients and servers can connect as requited and execute their operations in an asynchronous way. There are 3 major messaging services such as queues, topics and subscriptions.

- **Azure Notification Hubs:** Azure Notification Hubs is an extensively expandable mobile push notification engine for rapidly delivering trillions of notifications to iOS, Android, Windows working with GCM (Google Cloud Message), APNS (Apple Push Notification Services), MPNS (Microsoft Push Notification Service) and more. Notification Hubs is adjustable enough to plug in to any backend – Java, PHP, .NET, and Node.js – whether it’s located in the cloud or on-premises. This makes it effortless to immediately update the mobile apps and

capture the user's attention. Notification Hubs can be easily configured with Google Cloud Messaging (GCM) for Android deices.

- **Azure Blob Storage:** Azure Binary Large Object (BLOB) saves any type of unstructured data – audio, videos, images, documents and more. Azure Blob storage grasps millions of stored objects with thousands of requests per second for users all over the world. Strong stability if an object is updated, it's verified everywhere for super data purity, grantee us always have access to the latest version. Gives us the resilience to operate edits in place, which can improve our application performance and reduce bandwidth consumption. Binary data stored as a single entity in a database management system. Representational state transfer-based object storage for unstructured data. I will use Azure blob storage to store images. I will only store image path in the SQL table so that I can refer which image belongs to which note.

- **Azure Web Apps:** Web Apps option in Azure App Service allows developers quickly develop, deploy and manage robust web service and web apps. Build standards web apps and APIs using Java, .NET, Node.js, PHP and Python. Deliver both web and mobile apps for staff members or customers using a single backend. Securely deliver APIs that allow additional apps and devices. We need a web app to access Azure backend data. It would be implemented with by c# or Node.js.

- **Azure Active Directory:** Azure Active Directory (AD) offers identity management and access control for the cloud services. To reduce user access to cloud services, we can replicate on-premises identities and enable single sign-on. Espoo City Corporation staffs have local Active Directory (AD) on premise credential. I have to synchronize local AD to the Azure AD for so that everybody can access Azure service using the same credentials.

2.5 Background Conclusion

I have discussed the required technologies for the digital survey to argue that digital survey is more reliable and more efficient than manual survey. Thus by using digital survey by Android device and Azure cloud, survey date can be submitted instantly from the survey location. In this project, the only thing left for the staff to do in the office, is to write a report for the higher-level authority approval. As soon as an approval is granted, the actual task is carried out. As a result, Espoo City Corporation is able to provide service faster.

3 Digital Survey Requirements

To perform the survey digitally, both hardware and software upgrading is necessary. In this chapter hardware and software requirements are discussed in depth.

3.1 Hardware Requirements

Nowadays there many smart phones and tablets available. Among them Apple's iPhone and Google's Android phones are the most popular ones. The same applies for the tablets, Google's Android tablets and Apple's iPad are widely used.

Usually Apple's devices are expensive and in terms of operation system, it uses iOS which is a closed source and exclusive to the vendor only. By being a close source operating system, it lacks free software and it is not flexible; this restricts third party developers.

On the other hand Google's Android phones are more reasonably priced. There are varieties of Android phone available in the market form high end to low end. Also, Android is an open source operating system, meaning there are many kinds of free software available. It is very flexible for the third-party developer.

Espoo City Corporation wants to provide maximum service within a limited budget. Because of cost and software flexibility, a decision was made to use Android phone and tablets as hardware for survey service.

3.2 Software Requirements

Because Android phone and tablets were chosen for this project, an application for Android operating system had to be developed. There are large varieties of Android devices available in the market. Android devices are mainly launched with specific code name, version number and API level as shown in figure 3.2 [1.]

Figure 3.2: Android platform codes, names and version numbers [1]

The recent versions of Android always provide significant APIs for the app. I should try to support both the latest as well as older versions of Android. However, it is not possible to support all Android platforms. In order to develop an Android app, we had to select lowest and highest API level. Currently active Android device distribution is shown in table 3.2 and figure 3.2. A library which only supports minimum API level 19 had to be used. Table 3.2 shows the respective number of devices running a specific version of the Android platform.

Table 3.2 Distribution of android platform

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x – 4.3	Jelly Bean	16 – 18	3.2%
4.4	KitKat	19	7.8%
5.0 – 5.1	Lollipop	21 - 22	18.3%
6.0	Marshmallow	23	29.3%
7.0 – 7.1	Nougat	24 - 25	0.7 %
8.0 – 8.1		26 - 27	19.2 %

On the above table we can see the proportion of Android device distribution. Below is a pie chart of the same Android device distribution.

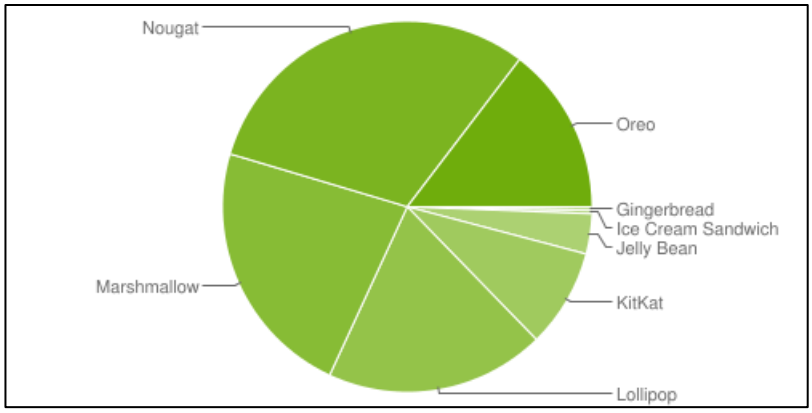


Figure 3.2: Android platform distribution percentage [2]

As we can see in figure 3.2 [2] when the share of the latest Android devices increased in the market, proportionally the share of older devices decreases. Therefore, our main focus in this study is to support the latest devices as much as possible.

3.3 Cloud Computing Services

There are several cloud services available in the market. Among them the most famous are Google Cloud Platform (GCP), Amazon Web Services (AWS) and Microsoft Azure.

Amazon has been leading in Cloud computing services but other players such as Microsoft, Google, Alibaba, IBM, Fujitsu, and Oracle are catching up fast.

Espoo City Corporation IT infrastructures are based on Microsoft technologies such as SharePoint, Dynamics CRM (customer relationship monument), etc. Also all local workstations are Microsoft Windows. To achieve seamless integration with the existing infrastructures, the obvious choice was to select Microsoft Azure. Microsoft Azure can offer seamless integration with existing technologies.

Cloud computing offers the following benefits:

- **Broad network and device explosion:** Offers accessibility over the network and also offers benefit for verity clients like tablets or laptops, smart phone.

- **On demand service:** The cloud provider is required when the customer needs to use cloud resources.

- **Resource allocation with multi occupancy:** Capability like memory or storage is accrued to different client on demand.

- **Speedy elasticity:** The cloud system can be organized based on demand scale up or down.

- **Data retrieval:** There is always risk of losing important data in case of hardware failure, user error and server failure. So server data should be backed-up frequently. All data in cloud computing is automatically replicate data. Replication means that transfer the data to a remote copy either immediately or with a short time delay. Secondary copy of data is identical to the primary copy.

3.4 Azure Features Consumed

For this survey service and Android application the following Azure cloud features listed in table 3.4 are needed:

Table 3.4 Applied Azure features

Azure Feature Name	Consumed Counter Part
Azure Storage Account (Blob)	To store images
Azure SQL database	To store relational data
Azure Mobile app service	For authentication and secure access
Azure Web app service	To access data by web browser
Azure Notification Hub	To send push notification

Since application or service always needs to update according to latest requirements, some extra features not included in the list above might be needed in the future.

- **Azure Storage Account:** Azure Storage data contains blobs, files, queues, tables, and disks. For this study, only blobs are needed. Blob Storage can handle all your unstructured data such as audio, video, image, animation, documents, presentation, etc. In this study blobs will be used to store images. How to implement blobs in the backend and how to utilize blobs in the Android application will be explained later.

- **Azure SQL Database:** Microsoft Azure SQL Database is a managed cloud database. Meaning database ser-vices take care of scalability, backup, and high availability of the database. No additional database administrator requited. We know SQL is a sequel relational database and we must store survey report data into SQL database table. Complete SQL database table structure will be discussed later.

- **Azure Mobile App Service:** It is highly recommended to use Azure Mobile app service if a developer wants to communicate with Azure with any mobile device. Currently Azure support following mobile platforms:
  - Android
  - iOS (Objective-C)
  - iOS (Swift)
  - Windows Phone (C#)
  - Xamarin.Android
  - Xamarin.iOS
  - Xamarin.Forms
  - Cordova

  It is recommended to use Azure Mobile app because of secure and easy user authentications. It is extremely easy to use Azure Mobile app to identify the user. Microsoft provides API for Azure Mobile app. The implementations of Azure Mobile app will be explained later.

- **Azure Web App Service:** Azure Web App service is hosting web applications, REST APIs and mobile backend. We can develop it by using any programming language such as Java, Ruby, Node.js, .NET core, PHP or Python. This survey

service will have both a web client and a Mobile client. As a result, Azure Web App service will be used.

- **Azure Notification Hub:** Azure notification hub is mainly used to broadcast push notification to mobile device and broadcast them fast to millions of devices. In this study, Azure Notification hub is used to notify as soon as any survey report is submitted to the Azure database. The implementation of Azure notification hub will be explained later.

In a nutshell above features will be used for the following purposes:

- Azure's Storage account service (Blob storage) to store survey image.
- Azure SQL database to store survey report's relational data.
- Azure Mobile app service to authentic users to access Azure features.
- Azure Web app service to access data from Azure cloud.
- Azure Notification Hub to send push notification to all mobile clients.

3.5 Android Application Requirements

The Android application (app) requirements are outlined by the Espoo City Corporation's managers. App should include features listed in table 3.5 [1]

Table 3.5 Android app feature requirements [1]

Android feature	App requirement
GPS (location)	Able to find current location in the map.
Camera	Able to take photo
GPS + Camera	Each photo should stamp current address and date
Note editor	All photos should have ability to write note for that particular image.
Note editor	User can edit current address if displayed address is incorrect.
Gallery + SQLite	All photos and notes should store locally in the Android device.
Internet	Able to upload photo and note to Azure cloud storage.
Push notification	After the successful upload, it notifies all interested persons.
Internet	User can retrieve data from cloud to their local device.
Local storage	Users are allowed to delete data from their local mobile device.

To implement above features Android app must requires user permission mentioned in table 3.5 [2]:

Table 3.5 [2] Android permission name

Feature	Permission name
GPS	android.permission.ACCESS_FINE_LOCATION
Internet	android.permission.INTERNET
Network status	android.permission.ACCESS_NETWORK_STATE
Storage write	android.permission.WRITE_EXTERNAL_STORAGE
Storage read	android.permission.READ_EXTERNAL_STORAGE
Map	android.MAPS_RECEIVE
Power manager	android.permission.WAKE_LOCK

Android app permissions must be defined in app AndroidManifest.xml. Only dangerous permissions require user agreement. The way Android asks the user to grant dangerous permissions depends on the version of Android running on the user's device, and the system version targeted by the app.

If the device is running Android 6.0 (API level 23) or higher, the user is not notified of any app permissions at install time. The app must ask the user to grant the dangerous permissions at runtime. When your app requests permission, the user sees a system dialog telling the user which permission group the app is trying to access. The dialog includes a Deny and Allow button.

4 Android Application Development

The target of the development is to support the latest devices in the market and also to support various screen size and orientation. Android executes on many types of devices that have different pixel densities and screen sizes. As in the previous chapter, minimum target API 19 and maximum target API 27 will be used in this study.

4.1 Supported Devices Coverage

According to the Android studio if minimum target API is 19 selected then 95.3% available devices are supported. Because the following library is used, the lowest possible API has to be 19:

- A client library for Android that makes it easy to consume Microsoft Azure Storage services: com.microsoft.azure.android:azure-storage-android:2.0.0@aar. Microsoft released this open source library at GitHub. It can be freely used by any application.

- Analytics to your Android app to measure user activity to named screens: com.google.android.gms:play-services-analytics:16.0.0

- Windows Azure Notification Hubs we can easily send push notifications to our mobile apps (Windows 10, iOS, Android). Broadcast notifications to millions of devices: com.microsoft.azure:notification-hubs-android-sdk. Microsoft released this open source library at GitHub. It can be freely used by any application.

- Google HTTP Client Library for Java is a flexible, efficient, and powerful Java library for accessing any resource on the web via HTTP: com.google.http-client:google-http-client-android

- The Google API Client Library for Java provides functionality common to all Google APIs. Easily access Google APIs from Java: com.google.api-client:google-api-client-android

- Microsoft Azure Mobile Apps you can add a scalable backend to your connected client applications: com.microsoft.azure:azure.mobileandroid:3.2.0@aar Microsoft released this open source library at GitHub. It can be freely used by any application.

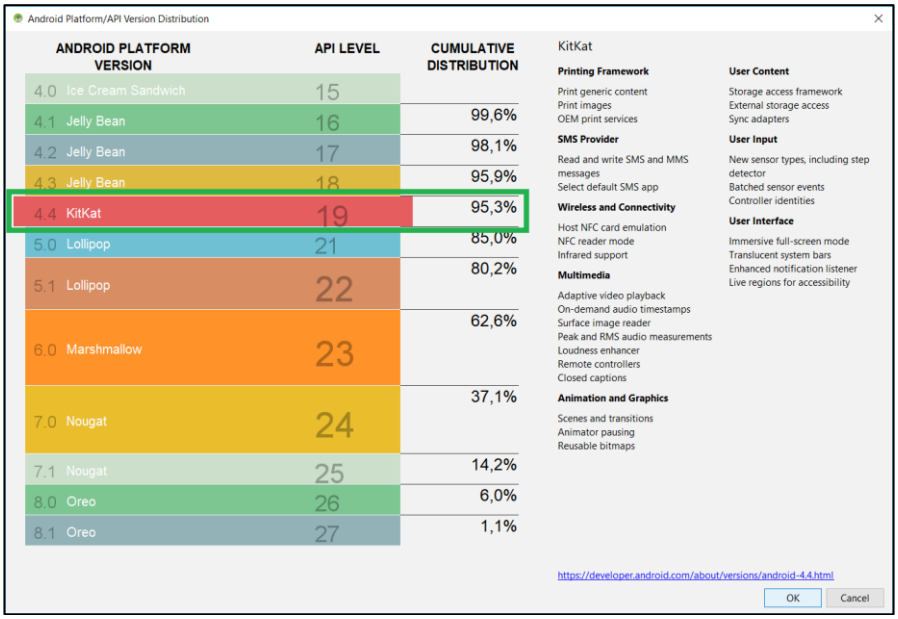


Figure 4.1 Android platform/api version distributions

Figure 4.1 is taken from latest Android Studio. That shows the device percentage which is supported.

4.2 Application Classification

This application has 2 parts client and server. For the better understanding client and server implantations will be discussed in separate chapters. This application has 6 views or activities:

- Current Location
- Camera
- New Note
- All Notes
- Local Gallery
- Azure Photos

Current Location is used as launcher activity. These application usages DrawerLayout. The DrawerLayout is a good Android widget for creating sliding menu drawer navigation. The drawer is a UI panel that shows app's main navigation menu.

5 Application Client Implementation

This app is mainly used for the location survey with photos. So I named this application as **“Photo Location Note”**. As previously mentioned, this app has 6 views or activities: Current Location, Camera, New Note, All Notes, Local Gallery, and Azure Photos.

5.1 Current Location

Current Location is the launcher view or activity shown in figure 5.1. It shows the user’s current location on map. Before showing the current location on map, it asks user permission to use GPS and data connectivity.

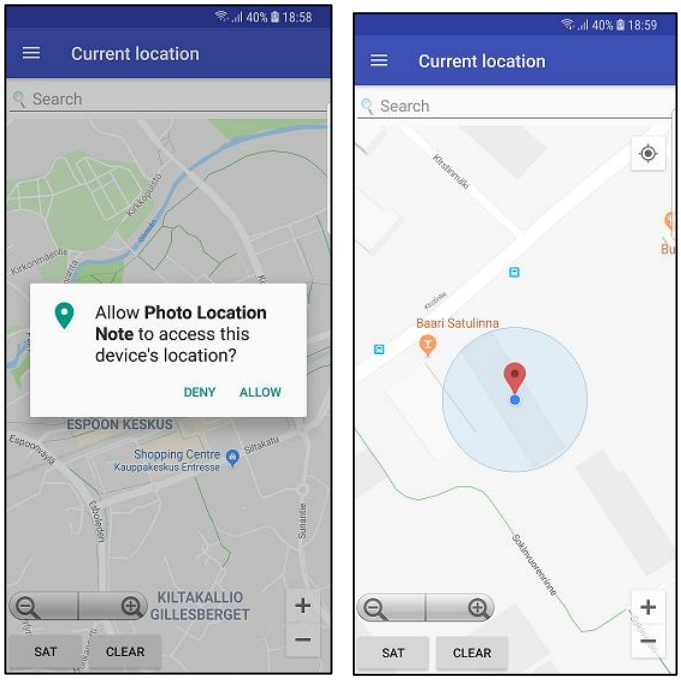


Figure 5.1 Location access permission and current location activity

In this study, the latest FusedLocationProviderClient API for the location was used. FusedLocationProviderClient is the optimized way to get the user’s current location. It consumes least possible power and the accuracy of 5 meters radius. This activity has also an option to search location by the address. There are also user options to choose street map view or satellite map view. Moreover, the user can zoom in or out and set marker on various locations. All source code is available at GitHub.

5.2 Camera

The camera view is mainly used to take a snapshot by using the camera. As soon as the photo is taken, this view redirects to New Note view or activity. Figure 5.2 shows screenshot of Camera activity.

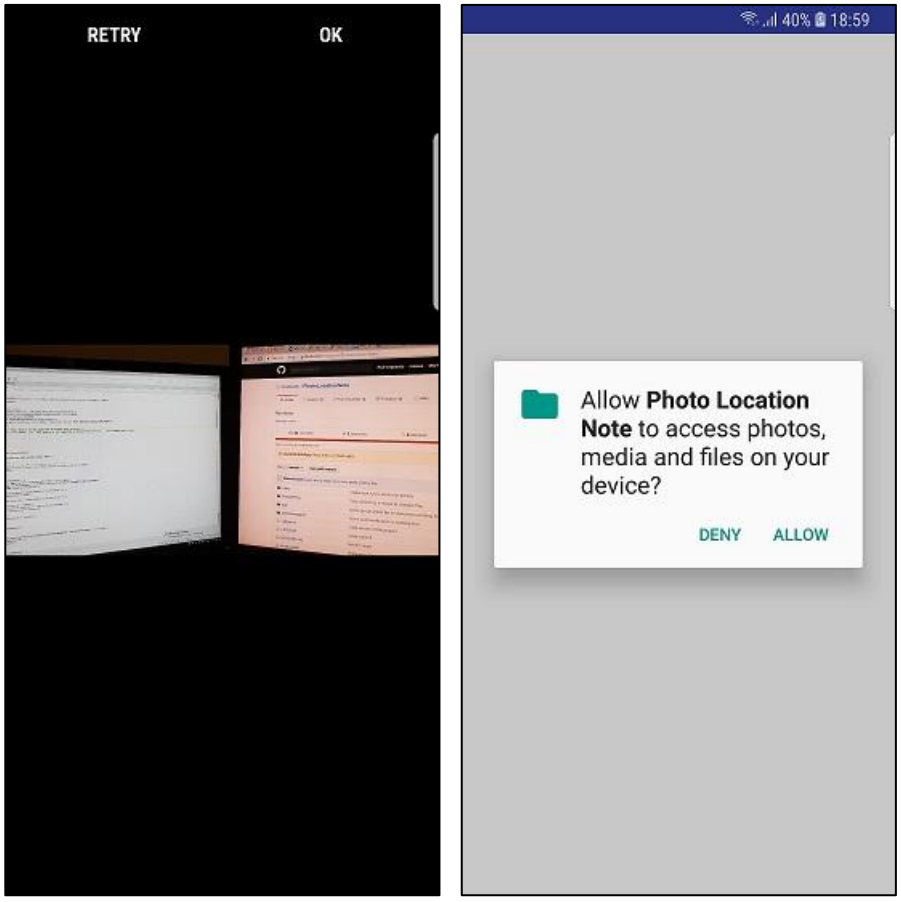


Figure 5.2 Camera access permission and camera activity

The latest Android forces user permission to various resources such as camera and media files. In this study the latest Android API implementation was used. It has several steps to ask for user permissions and has to act accordingly. When the user permits, it opens the viewfinder. Once the user takes a picture, it previews the photo. If user accepts, it saves photo in the application's private album. This app can be sent for devices supporting API level 28 and older versions of devices up to API level 19. Again all implemented source codes are found at GitHub.

5.3 New Note

New Note activity is used to write key notes for the photo. When this view is accessed, current address and current date is added. If this view is accessed from camera view, the photo is embedded with note. Figure 5.3 shows the screenshot's new note.

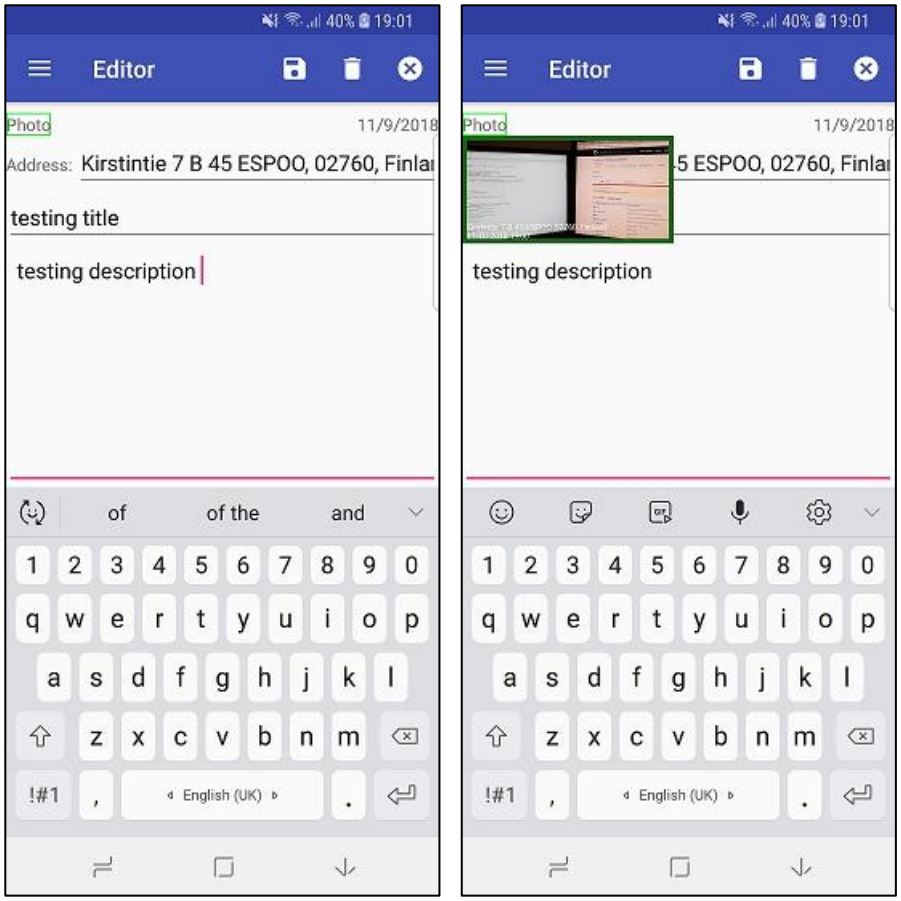


Figure 5.3 New note activity

Also the photo is stamped with current address and date. This view is used to make a new note or edit any existing note. The view is redirect to All Notes view or activity. This app also has a local SQLite database and it writes data in the database. This local database has a table called NoteTable shown in table 5.3. It has following columns:

Table 5.3: NoteTable table in local SQLite database

Column name	Data type.
id	INTEGER PRIMARY KEY AUTOINCREMENT

title	VARCHAR(255)
description	VARCHAR(255)
date	VARCHAR(255)
latitude	REAL
longitude	REAL
address	VARCHAR(255)
image	VARCHAR(255) (it is the image path)

Azure SQL database has almost similar table with some extra columns.

5.4 All Notes

All Notes activity contains saved notes in list view. If the list item is single tapped, it opens all data in Note editor view. If it pressed long, it shows a context sensitive menu. Context menu has 2 options: upload and delete. If upload is selected it uploads the selected item to Azure SQL database and Azure Blob. This upload speed is a time-consuming process and is, therefore, implemented by AsyncTask. During the upload a progress dialog is continuously spinning until upload has been done successfully. Figure 5.4 shows all note activities or view:

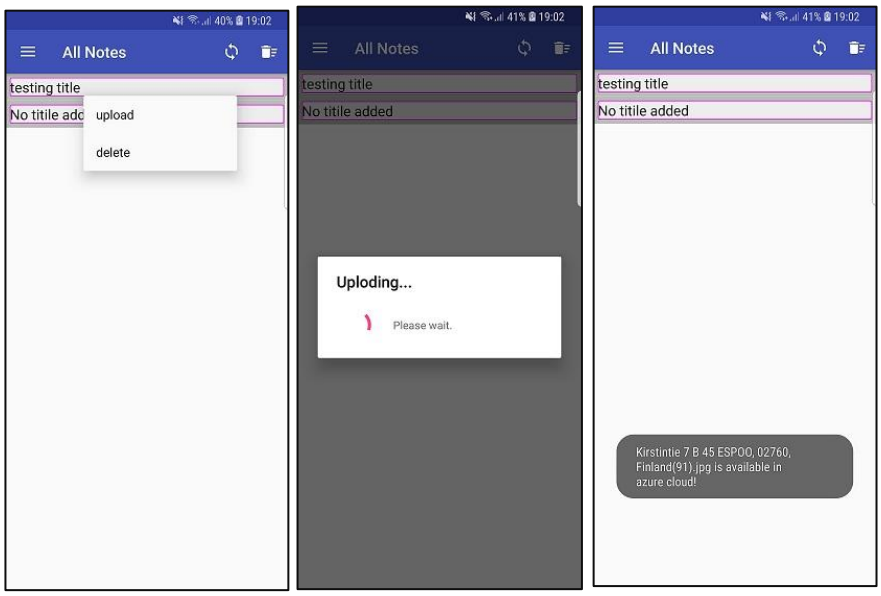


Figure 5.4 All notes activity

Also it sends a push notification. This push notification is broadcast to all registered clients. If delete is selected from the context menu, it deletes the selected item. It has 2

action menus, sync and delete all. They perform the action as their name implies. Sync uploads all items to azure and deletes all delete all items from local SQLite database.

5.5 Local Gallery

Local Gallery shows all snapshots taken by the user. This application saves all snapshots in Local Gallery. This Local Gallery has all functionalities such as share, delete, edit, and options like standard gallery of the device. It comes with tons of features, but it is one of the easiest and most preferred ones. The app comes pre-installed in nearly every smart phone. Figure 5.5 shows screenshot of Local Gallery:

Figure 5.5 Local gallery activity

This local gallery is also accessible from the device’s common gallery. App makes a separate album named Photo Location Note in the device gallery.

5.6 Azure Photos

Azure Photos activity shows all upload photos in Azure Blobs. In Azure we have Azure SQL database where all String, Date and Number data types are stored. Since image

is a binary file, it is recommended not to store any images in SQL database if image size is 256 kilo bytes (KB). This recommendation is to achieve best performance. In this study, the image name was only saved in the SQL database and it saves the same image in the Azure Blob. Figure 5.6 shows screenshot of Azure Photos activity:

Figure 5.6 Azure Photos activity

This activity has MobileServiceClient to retrieve all image names from Azure blob. MobileServiceClient API is from Microsoft Azure Mobile app open source library freely available at GitHub. Azure Mobile Apps provides data access and offline synchronization, Authentication with Azure App service authentication and authorization, Push Notification Registration with Notification Hubs. MobileServiceClient provides secure access to Azure backend. It can read write data to Azure backend. Azure Photos activity only shows the name of the image. Once user long presses, it shows a context sensitive menu with the option to manipulate that image from device to Azure. It has an option to delete the photo or open the photo in full screen. As soon as user clicks any image name it opens the full image in Azure Image activity.

5.7 Azure Image

When any photo selected from Azure Photos activity it opens in Azure Image activity. Figure 5.7 shows a screenshot of Azure Image activity:

Figure 5.7 Azure image activity

Azure Photos Activity is basic activity only to display image. If any image drag left of right it shows the next images or previous image. When the user clicks back, It goes back to Azure Photos activity.

5.8 Activity Class Diagram

We have seen all app's activities' screenshots. Now, their class relationship between each other will be explained. Modern Android app's activity extends AppCompatActivity to support ActionBar. We can add an ActionBar to our activity when running on API level 7 or higher by extending AppCompatActivity class for our activity and setting the activity theme to Theme.AppCompat or a similar theme.

Figure 5.8 shows that all our activities are inheriting or extending from AppCompatActivity. Also all activity has a member Globals class.

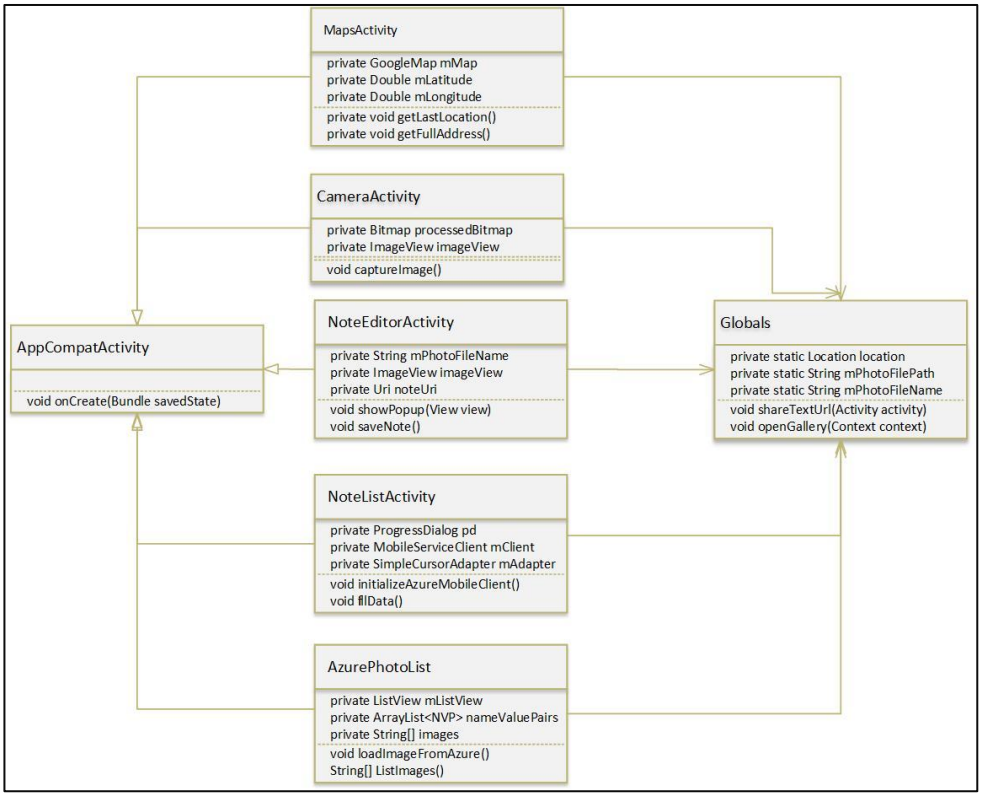


Figure 5.8 App Activities class diagram

Globals is singleton class. It has many commonly uses functionalities. MapsActivity is the launcher activity configured in the Android Manifest file. MapsActivity has FusedLocationProviderClient API to get users current location.

5.9 NoteEditorActivity Class Diagram

NoteEditorActivity extends AppCompatActivity and it has Globals and NoteDatabaseHelper class member. NoteDatabaseHelper extends SQLiteOpenHelper. SQLiteOpenHelper is helper class to manage database creation and version management. We have to extend SQLiteOpenHelper implement onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int) and optionally onOpen(SQLiteDatabase), and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.

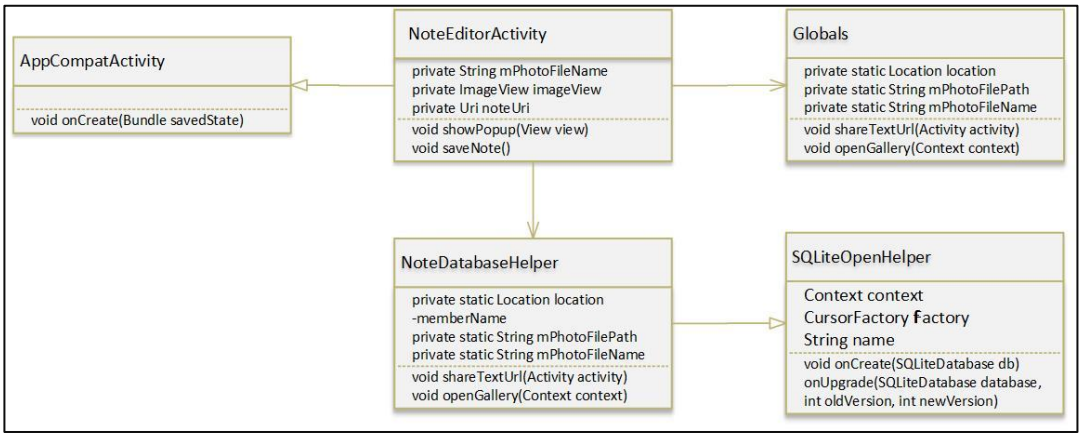


Figure 5.9 NoteEditorActivity class diagram

NoteEditorActivity is responsible for database creation and all CURD (create, update, read, delete) operations.

5.10 NoteListActivity Class Diagram

NoteListActivity has MobileServiceTable and MobileServiceClient class as member. MobileServiceClient provides secure access to Azure backend data.

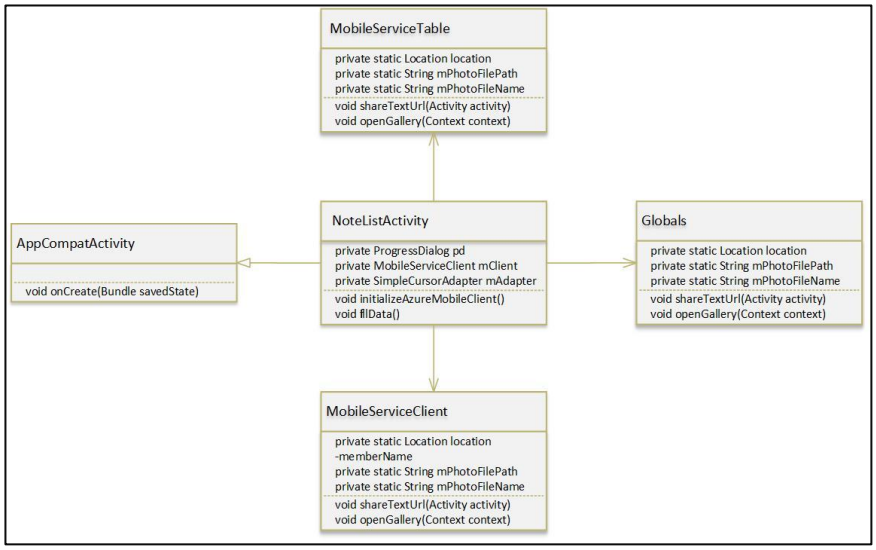


Figure 5.10 NoteListActivity class diagram

MobileServiceTable is responsible for database creation and all CURD (create, update, read, delete) operations.

6 Application Server Implementation

Espoon Kaupunki's information technology (IT) infrastructure is based on Microsoft technology. Espoon Kaupunki is using Microsoft Dynamics and Microsoft Sharepoint. To achieve seamless integration I decided to use Microsoft Azure as backend for this application. Azure has many well know features such as Azure's storage service, Azure Recovery, Azure Express Route, Azure virtual machine extension, Azure portal, Networking feature, Azure Active Directory (AD), Cloud App Discovery, Azure Remote App, Storage account, Notification Hub, App Service, Mobile App, SQL database, Azure Blobs. In this project, I utilized Storage account, Notification Hub, App Service, Mobile App, SQL database and Azure Blob.

6.1 Storage Account

Azure storage account provides 3 types of storage. Each has a different feature and its own pricing structure. I had to consider these differences before I was able to determine which type is the suitable for survey service. The types of the accounts are General-purpose v2 accounts (recommended for most scenarios), General-purpose v1 accounts and Blob storage accounts. After reading all feature and offering I found Blob storage is the most suitable for survey service. Because blob storage support unstructured non-SQL data such as audio, video, image, animation, document, permeation, and pdf.

In this project, both non-SQL data and SQL data had to be saved. Non-SQL data is the image of the survey. And SQL data is the actual survey data that includes title as String type, description as String type, date as Date type, address as String type, image (path) as String, latitude as Number and longitude as Number.

Non-SQL data, in other words images, will be stored in blob storage and SQL database. It is also possible to store image in the SQL database, but it is not recommended.

After performing several tests and analysis, it appeared that if the image size below 256 kilo byte (KB), then SQL database table column type VARBINARY which represents binary data type is more efficient than blob storage. If image size is over 256 kilo byte (KB), storing them in the blob storage is more efficient.

Nowadays, nobody takes a photo in such a low regulation that the image size would be below 256 KB, because of high regulation camera it is very common even in the low-end phone. That is why I decided to use blob storage to store image to store only the image path in the SQL database to maintain the relationship of each SQL table row data and image.

6.2 Blob Storage Accounts

Blob storage is the most suitable account to store photos, because it is suitable for unstructured data such as audio, video, image and animation. Therefore I used Blob storage account in this application to store image. Every storage account must belong to an Azure resource group. A resource group is a logical container for grouping all Azure services. When we create a storage account, we have the option to either create a new resource group or use an existing resource group. We can see in figure 6.2 [1] where to find Blob storage account.

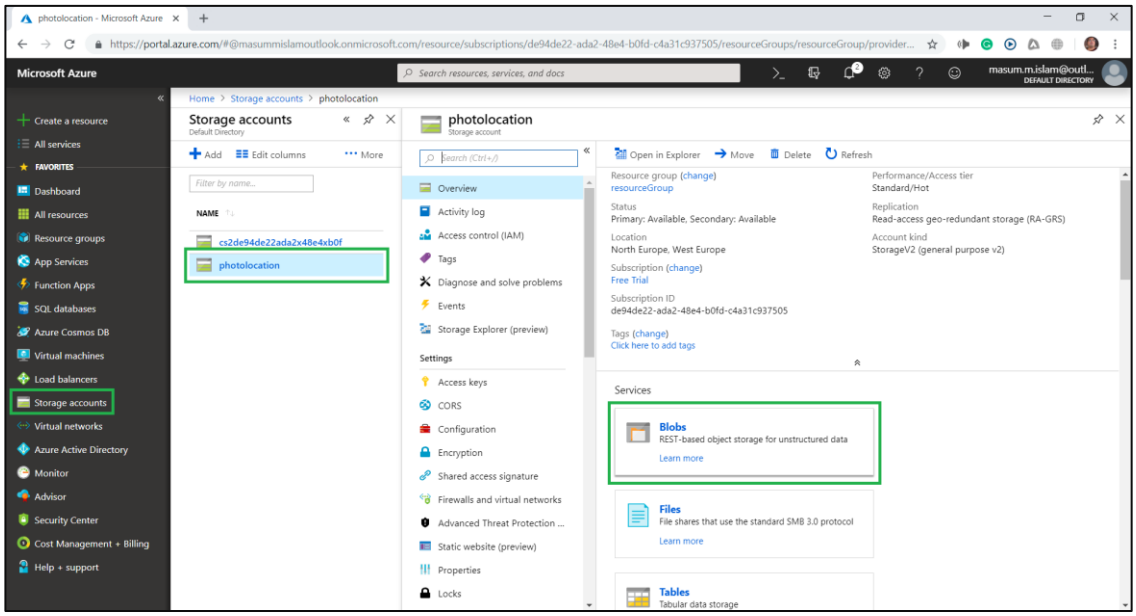


Figure 6.2 Blob storage account [1]

It is very straightforward to create Blob storage account in Azure portal. Following figure 6.2 [2] screenshot explains how to create Blob storage account steps. Step 1 click Add then step 2 fill subscription type and resource group. Step 3 enter storage name.

Step 4 select performance types, select Blob storage form account kind dropdown and replication type from the drop down.

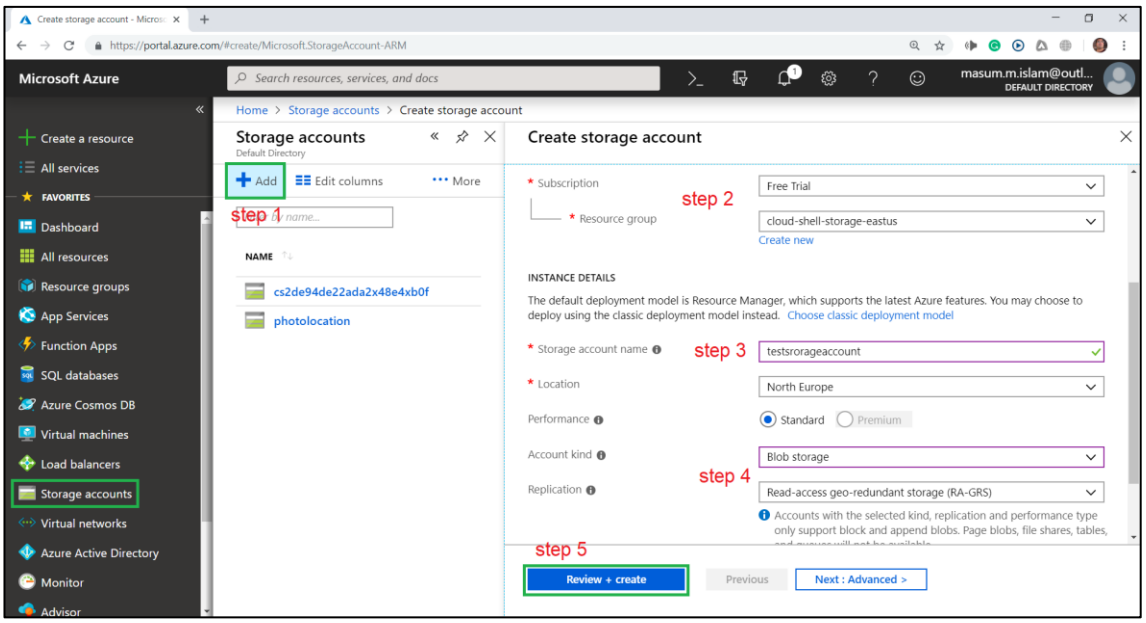


Figure 6.2 Blob storage account creation steps [2]

Select options step by step and then fill all required information. Finally click Review + create button. After Blob storage account creation is done successfully. We need Access keys for our Android application. It is found in figure 6.2 [3] screenshot:

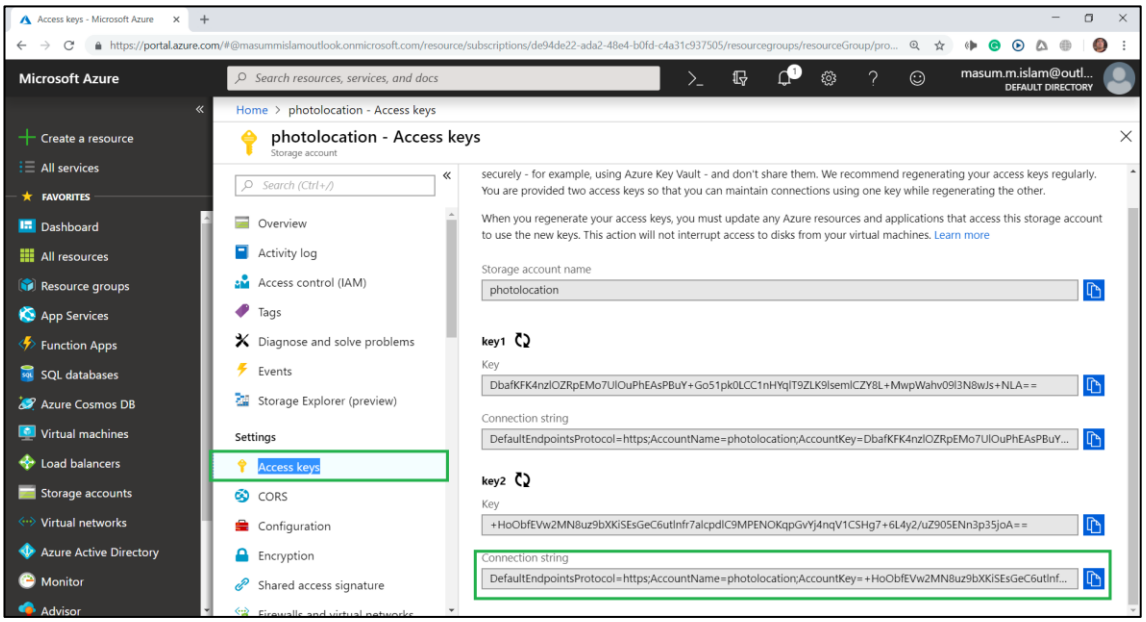


Figure 6.2 Storage account connection string [3]

We have to copy connection string and save it somewhere. Also, we need a connection string in our Android app.

6.3 Connect Azure Blob from Android App

It is straightforward to communicate between Android app and Azure Blob storage. For data exchange between Azure Blob storage and Android app the main requirement is the storage account connection string. I implemented a class ImageManager in my application; I think developers prefer explanation from the source code. Here is the implantation of the ImageManager class:

```
public class ImageManager {

  // storage account connection string
   public static final String storageConnectionString = "DefaultEndpointsProto-
col=https;AccountName=photolocation;
"+"AccountKey=DbafKFK4nzlOZRpEMo7UlOuhEAsPBuY+Go51pk0LCC1nHYqlT9ZLK9lsemlCZY8L+Mwp
Wahv09l3N8wJs+NLA==;"+"EndpointSuffix=core.windows.net";

private static CloudBlobContainer getContainer(){
      // Retrieve storage account from connection-string.
      try {
         CloudStorageAccount storageAccount = CloudStorageAccount
               .parse(storageConnectionString);
         // Create the blob client.
         CloudBlobClient blobClient = storageAc-
        count.createCloudBlobClient();
   // Get a reference to a container. The container name must be lower case
      CloudBlobContainer container = blobCent.
               getContainerReference("images");
         return container;
      }catch (Exception ex){
         System.out.printf(ex.getMessage());
      }
      return null;
   }

// upload an image
public static String UploadImage(InputStream image, int imageLength, String imageName ) throws Exception
{
```

```
    CloudBlockBlob imageBlob = container.
                getBlockBlobReference(imageName);
        imageBlob.upload(image, imageLength);
        return imageName;
    }

// get all image name from the Azure blob
    public static String[] ListImages() throws Exception{
        CloudBlobContainer container = getContainer();
        Iterable<ListBlobItem> blobs = container.listBlobs();
        LinkedList<String> blobNames = new LinkedList<>();
        for(ListBlobItem blob: blobs) {
            blobNames.add(((CloudBlockBlob) blob).getName());
        }
        return blobNames.toArray(new String[blobNames.size()]);
    }

// get an image base on the given image name
    public static void GetImage(String name, OutputStream imageStream,
   long imageLength) throws Exception {
        CloudBlobContainer container = getContainer();
        CloudBlockBlob blob = container.
           getBlockBlobReference(name);
        if(blob.exists()){
            blob.downloadAttributes();
```

We need Azure storage library to access 'com.microsoft.azure.android:azure-storage-android:2.0.0@aar'. We have to add this library in the app's build.gradle as follows in this section:

```
dependencies {
//... other library
implementation   'com.microsoft.azure.android:azure-storage-android:2.0.0@aar'
// .. other library
}
```

6.4 Notification Hub

As explained earlier, the feature of Azure notification hub is mainly used to broadcast push notification to mobile device. It is not simple to implement Azure notification in Android platform because there are several steps involved. Also it requires advanced Android development knowledge.

There are several steps in various sections inside Google console project and Azure portal and consequently, it is hard to follow the text descriptions.

6.4.1 Google API Key Generation

To generate a google API key, we need an API key from Google Cloud. To achieve the API key we have to make a project in the Google Cloud Consoleby using https://console.cloud.google.com/ and sign in by using Google ID. After signing in, we can see the following figure 6.4.1 [1] screenshot:

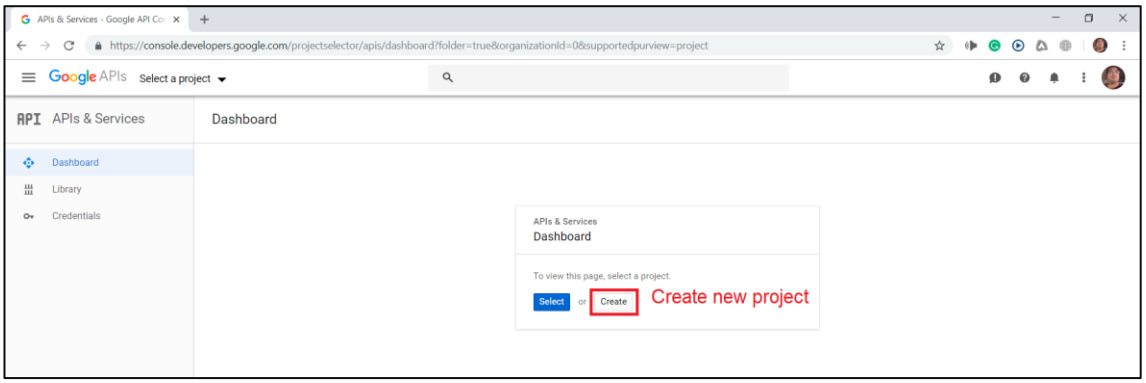


Figure 6.4.1 Google console [1]

We have to click to create new project. Because Google changes their UI frequently, it may not be exactly same UI. However, the idea is to create a new project in the Google console. In order to make a Google console project, you have to sign in by using Google ID. In Google console, we can create a limited number of projects for free. After we click create, we can see figure 6.4.1 [2] screenshot. The name has to be a unique name. As soon as the project is created, it will show the project name, number and location. Google API key mainly used by Google to track the application communications with Google servers. Once we generate an API key, we can add it to our mobile

app, website, or web server. The API key is used to track API requests associated with the project for usage and billing.

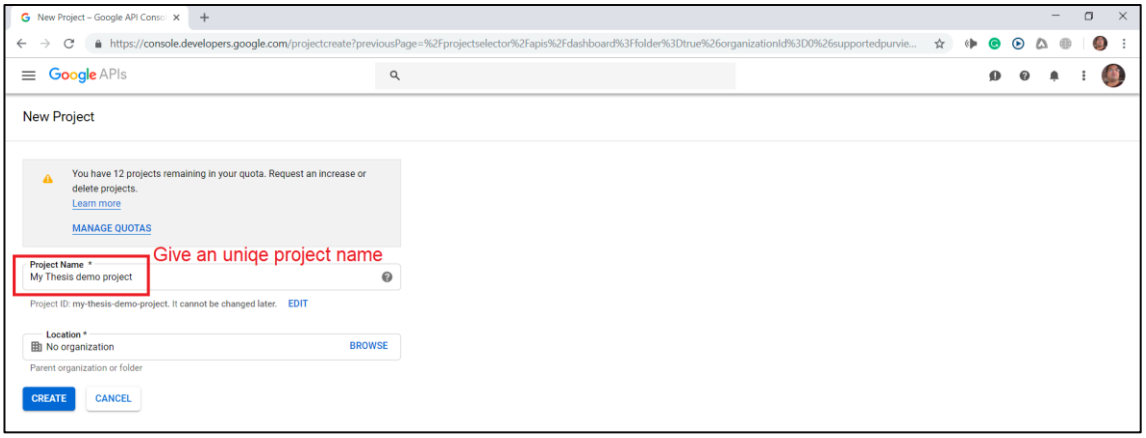


Figure 6.4.1 Google Console New Project [2]

In figure 6.4.1 [2], we can see the new project called **My Thesis demo project** when the creation has been done. When a new project is created successfully, we can see how it looks in figure 6.4.1 [3] screenshot. Google Cloud Platform (GCP) projects are the basis for creating, enabling, and using all GCP services including managing APIs, enabling billing, adding and removing collaborators, and managing permissions for GCP resources.

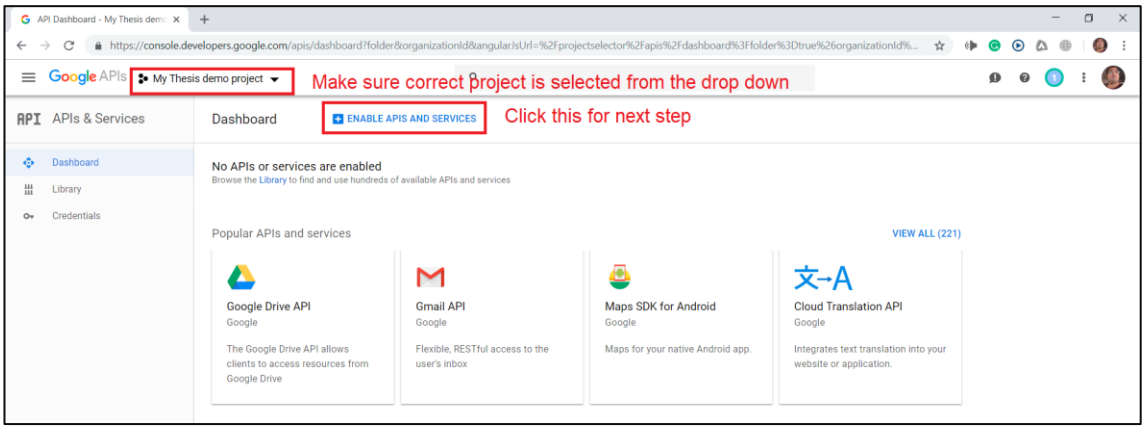


Figure 6.4.1 Google console new project setup [3]

A project ID is different from a project name. The project name is a human-readable way to identify the projects, but it is not used by any Google APIs. We have to make

sure that the correct project is selected from the drop down in case if there are more than one project. After that, click blue EXPEND APIS AND SERVICES. After this we can see the following figure 6.4.1 [4] screenshot. We have to search for Google Cloud Messaging:-

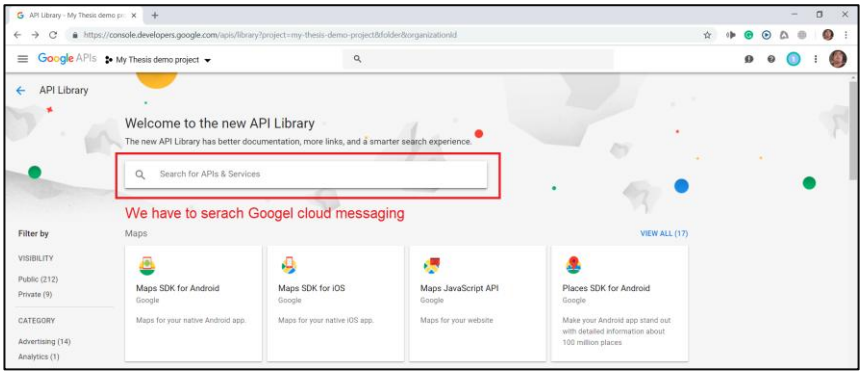


Figure 6.4.1 Search Google cloud messaging [4]

When we search Google Cloud Messaging, it may show up many search results. One should only select the right one shown in figure 6.4.1 [5]:

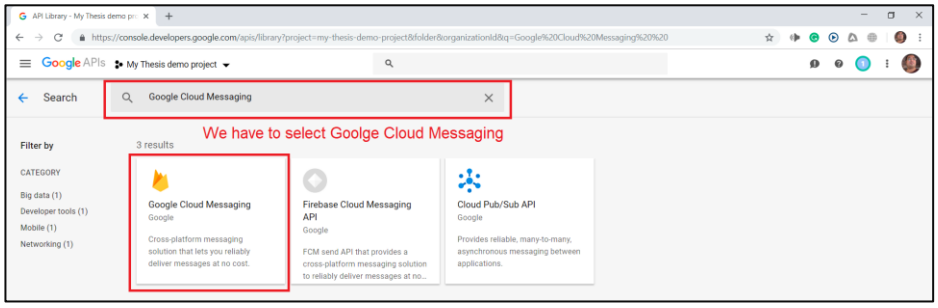


Figure 6.4.1 Select Google Cloud Messaging [5]

After selecting GCM, we can see the following figure 6.4.1 [6]:

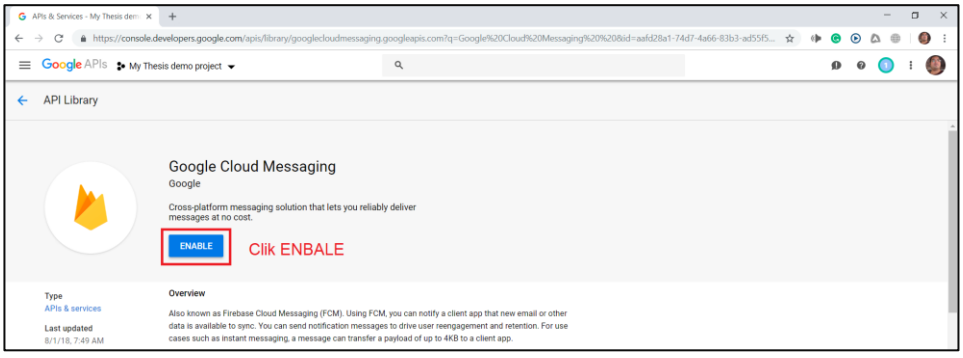


Figure 6.4.1 Enable Google Cloud Messaging [6]

We have to enable GCM. After we enable the Google Clod Messaging (GCM), we have to create API key credentials for this GCM API. There are multiple types of credentials. We should only enable API key credentials for the GCM as shown in figure 6.4.1 [7]:

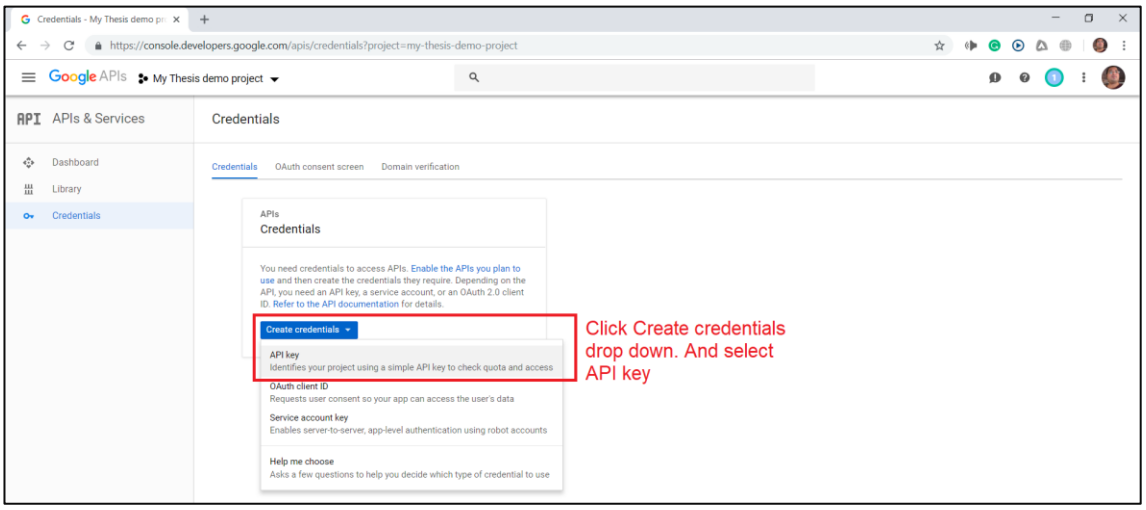


Figure 6.4.1 Enable Google Cloud Messaging [7]

Once we click API key following figure 6.4.1 [8] will appear. It must be configured further before we can use this API key.

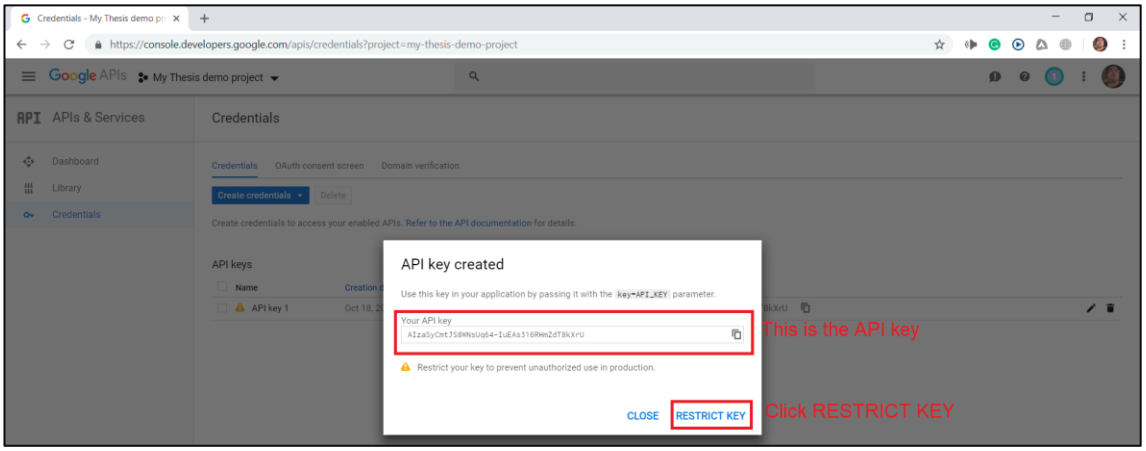


Figure 6.4.1 Enable Google Cloud Messaging [8]

In the figure above, we can see an API key, which must be restricted by clicking RESTRICT KEY. After the click we will see this figure 6.4.1 [9] in the screenshot:

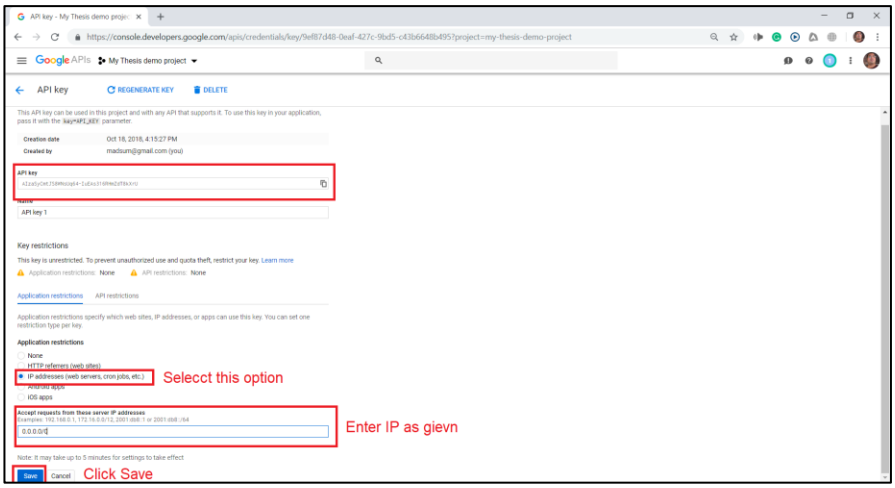


Figure 6.4.1 Restrict API key [9]

An API key is unrestricted by default. Unrestricted keys are insecure because they can be viewed publicly, such as from within a browser, or they can be accessed on a device where the key resides. Application restrictions specify which web sites, IP addresses, or apps can use an API key. Add application restrictions based on your application type. We can only set one restriction type per API key. We restrict it by selecting IP addresses (web servers, cron jobs, etc.) option and enter 0.0.0.0/0 and clicks save. After the save button make sure the following screen shows that it's restricted as following figure 6.4.1 [10] screenshot:

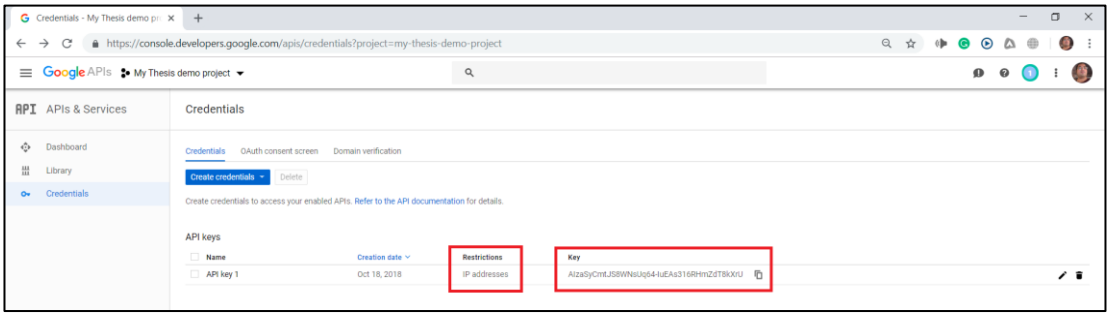


Figure 6.4.1 Restrict API key saved [10]

We have to copy and save the API key somewhere to use it later in the Azure Notification Hub. Also we need project number. Project number is found in the project setting page shown in the following figure 6.4.1 [11]:

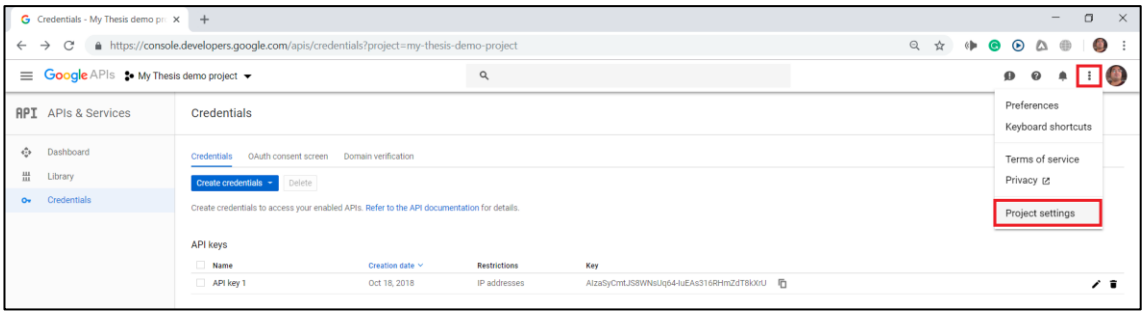


Figure 6.4.1 Google Console Project setting [11]

Once we click Project setting as shown in figure 6.4.1 [11], we will see following figure 6.4.1 [12] screenshot. There we can see project number:

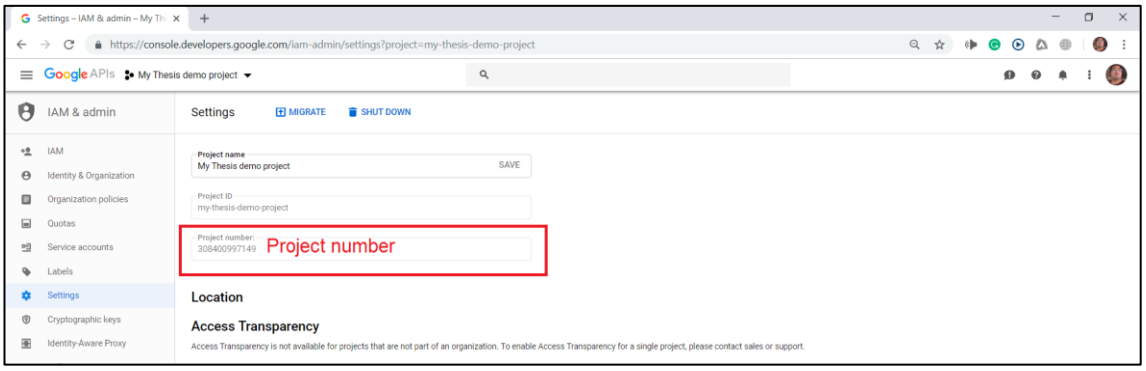


Figure 6.4.1 Google Console Project number [12]

We have to copy project number and save it for later use in the Android project.

6.4.2 Notification Hub Implementation

Azure Notification Hubs provides an easy-to-use, multi-platform, scalable infrastructure for sending push notifications to mobile devices. With Notification Hubs we can easily send cross-platform, personalized push notifications, abstracting the details of the different platform notification systems (PNSs). With a single API call, you can target individual users or entire audience segments containing millions of users, across all their devices. To setup Notification Hub, simply we have to go to the Azure Portal create a new resource and choose the Mobile App. This will contain a set of services including the Notification Hub. It is easy to create Notification Hub. Following Figure 6.3.2 [2] shows the steps how to create Notification Hub.

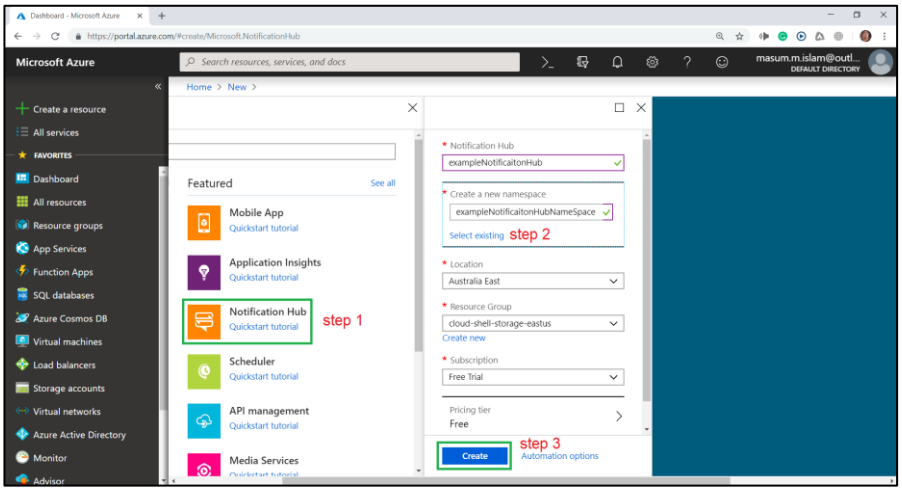


Figure 6.4.2 Azure portal Notification Hub creation steps [1]

After creating the Notification Hub we need Access Policies connection strings. There are two kinds of access policies listen and full access policy. They are named as follows:

Table 6.4.2 Notification Hub Access Policy

Policy Name	Permission	Connection String
DefaultListenSharedAccessSignature	Listen	Endpoint=sb://xxxxxx
DefaultFullSharedAccessSignature	Listen,Manage,Send	Endpoint=sb://xxxxxx

Push notifications are delivered through platform-specific infrastructures called Platform Notification Systems (PNS). A PNS offers barebones functions (that is, no support for broadcast or personalization) and the platform-specific PNSs have no common interface. Notification Hubs eliminate major complexity; we do not have to manage the challenges of push notifications. Instead, we can use a Notification Hub. Notification Hubs use a full multiplatform, scaled-out push notification infrastructure, and considerably reduce the push-specific code that runs in the app backend. We need only connection string for our android application. We have to use DefaultListenSharedAccessSignature permission to receive push notification. And we have to use DefaultFullSharedAccessSignature to send and manage push notification. Following figure 6.4.2 [2] shows where to locate Notification Hub access policy:

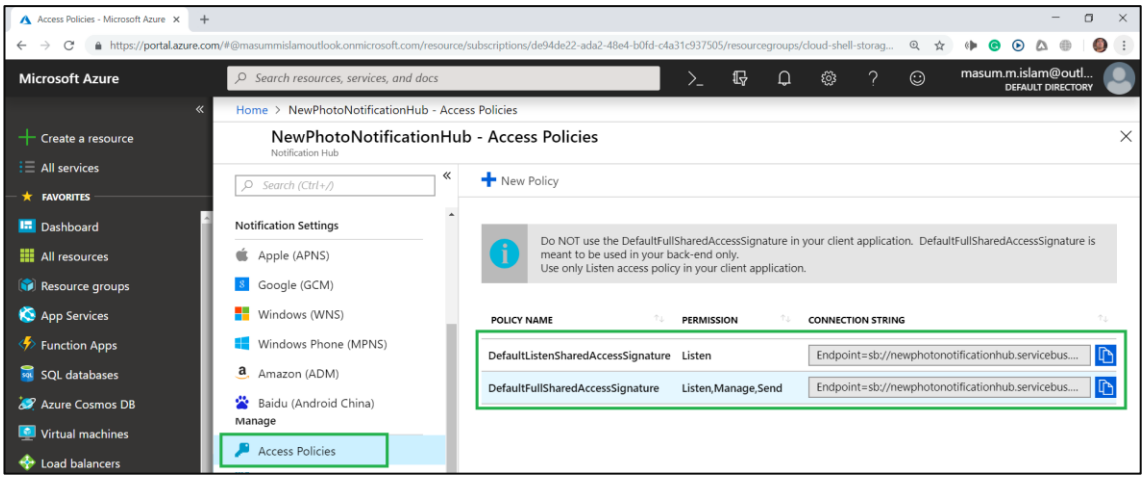


Figure 6.4.2 Notification Hub Access Policies [2]

We have to use the API key we generated at the Google Console Project. API key must be inserted at the following figure 6.4.2 [3] screenshot. Since we are using Android platform, we need Google platform. Respectively if we need to configure for iOS or Windows(C#), etc if we need those platforms.

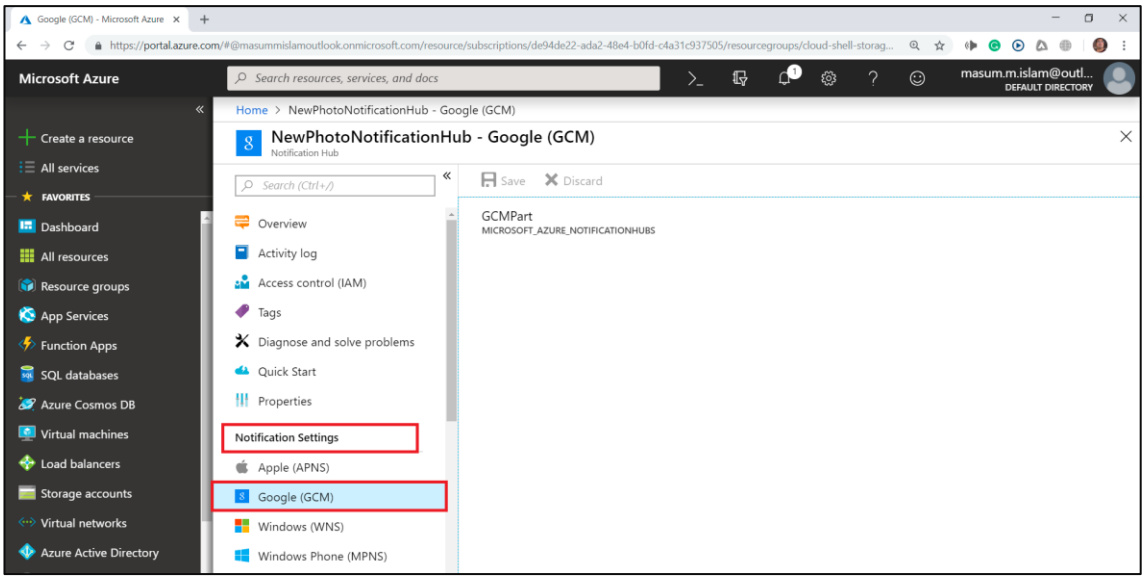


Figure 6.4.2 Notification Hub Google (GCM) key [3]

After Notification Hub setup is completed, we have to save Notification Hub name and Notification Hub Access Policy.

6.5 App Service

When we make a new Mobile service at the Azure, it appears in App service section. App service runs effortlessly and securely in the cloud as well as on primes. It ensures secure access to all Azure resource from web and mobile client. Here we can see figure 6.5 [1] where to find App service in Azure portal:

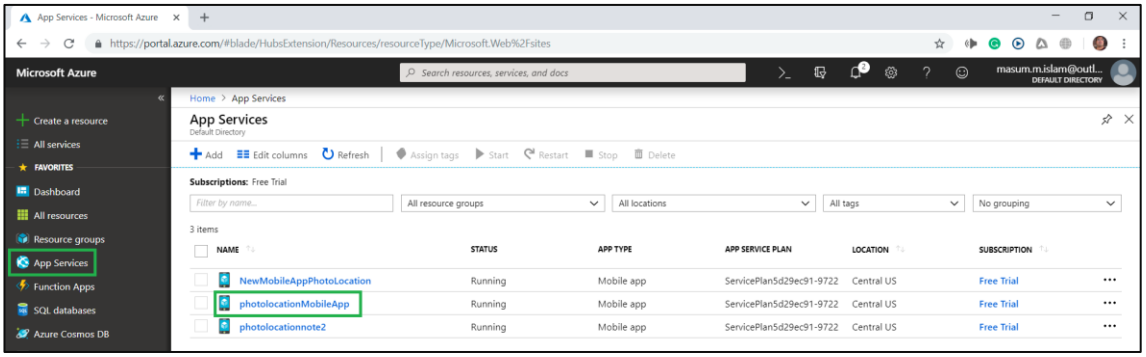


Figure 6.5 Azure App Service [1]

By App service also allowed authentication by Azure Active Directory more securely. We can use Mobile Apps to offer native sync experience across the Android, iOS and Windows Phone apps. Again it is simple to create Mobile app. Figure 6.5 [2] screenshot shows how to create Mobile App step by step in Azure portal:

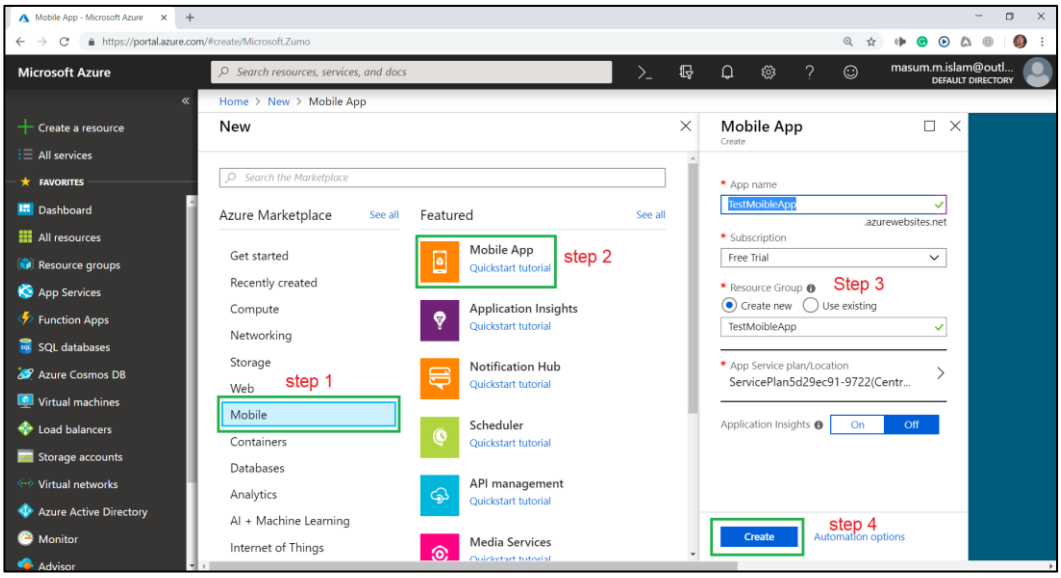


Figure 6.5 Azure Mobile App creation steps [2]

When Mobile App is successfully created, we need URL for our application. The URL is found in the overview. Following screenshot shows where to find it:

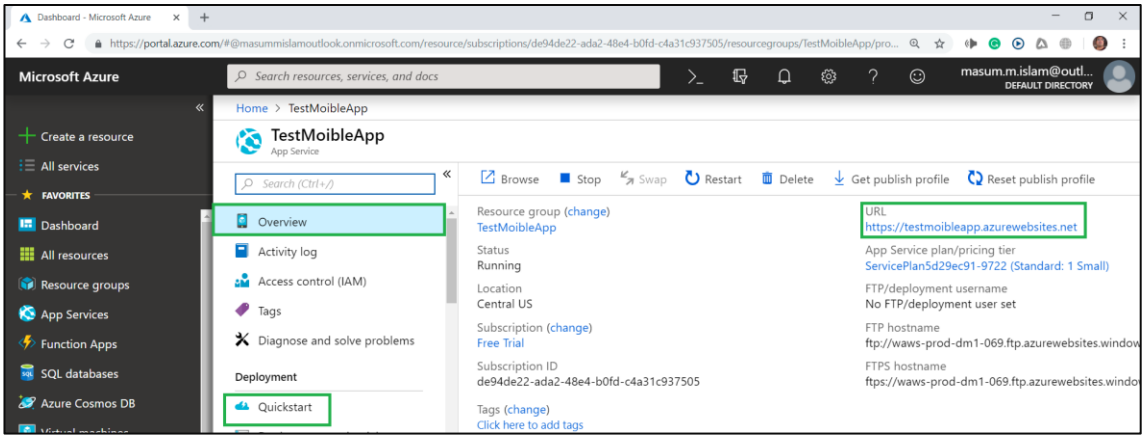


Figure 6.5 Azure Mobile App URL [3]

We can see QuickStart in the above figure 6.5 [3]. QuickStart guides us how to quickly build a client app on various platform. Currently it supports the platform shown in figure 6.5 [4] For this application, I had to select Android.

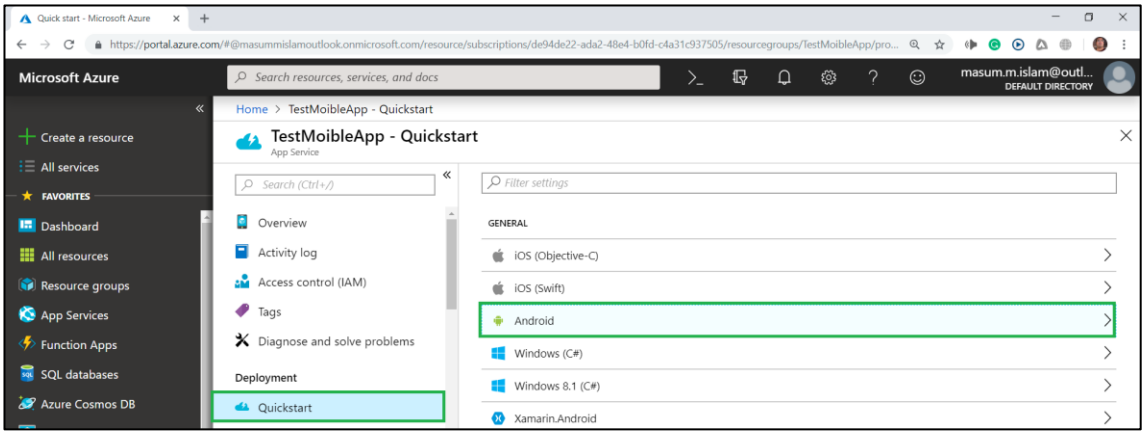


Figure 6.5 Azure mobile app QuickStart [4]

After successful creation of Mobile App we have to save the Mobile App URL. We have to use Mobile App URL in Android Application and authenticate our customers with Azure Active Directory for a more secure connect to on-premises. When robust apps that remain useful when there are network issues are created, it means that our customers can create and modify data when they are offline. Improve app responsiveness is maintained by caching server data.

6.6 Azure SQL Database

Azure SQL Database is a self-maintain type of database system. Usually to maintain any database, we need an expert database administrator to perform various database-related tasks. Common database admin task includes the following:

- Performance Monitoring.
- Capacity Planning.
- Security and Authentications.
- Database Backup and Recovery.
- Software Upgrade.
- Troubleshooting.

This is a fulltime job for a System or Database administrator. Azure SQL Database offers everything and more additional support together with the SQL Database. Basically we do not need to employ any additional Database administrator. The amount of data generated today can be huge. It is a challenge to face the mounting pressure of database management. Azure is able to manage everything round the clock 24/7. After using Azure SQL Database it seemed there will not be any job title for Database administrator, because if we use Azure or any other cloud Database system, we do not need extra Database administrator. Figure 6.6 shows where to find Mobile App service in Azure portal:

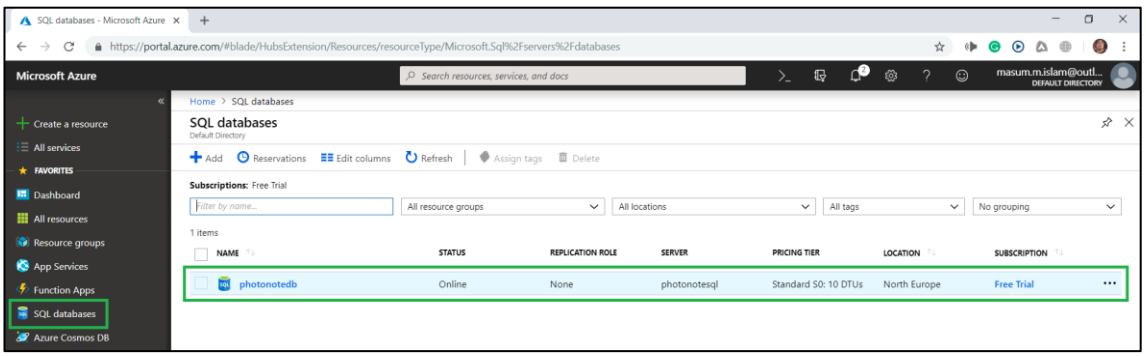


Figure 6.6 Azure SQL databases

Azure SQL Database is a relational database-as-a-service (DBaaS) based on the latest stable version of Microsoft SQL Server Database Engine.

6.7 Azure SQL Database Table

For this application I designed a table named PhotoLocationNote where to store title, description, date, address, image, latitude and longitude. Thus I have to design table column as shown in table 6.7 [1]:

Table 6.7 Table columns app's own data [1]

Column name	Data type	Required
title	String	no
description	String	no
date	Date	yes (current date inserted)
address	String	no (current address inserted)
image	String	no (blob link is save)
latitude	Number	no (current latitude inserted)
longitude	Number	no (current latitude inserted)

Azure SQL table always generates following extra column shown in table 6.7 [2]. These columns are used by the Azure cloud. Mostly used for the following purpose:

- Manage data migration
- Avoid data corruption
- Concurrent modification conflicts

Table 6.7 Table generated columns [2]

Column name	Data type	Required
Id	String	yes
createAt	Date	yes
updatedAt	Date	yes
version	Version	yes
deleted	Boolean	yes

Consequently, together with my required columns and generated column, the total table looks like the following table 6.7 [3]:

Table 6.7 Azure SQL Database Table columns [3]

Column name	Data type	Required
Id	String	yes
createAt	Date	yes
updatedAt	Date	yes
version	Version	yes
deleted	Boolean	yes
title	String	no
description	String	no
date	Date	yes (current date inserted)
address	String	no (current address inserted)
image	String	no (blob link is save)
latitude	Number	no (current latitude inserted)
longitude	Number	no (current latitude inserted)

Note that I don't need to write any raw SQL query to create this table. As soon I try to write any data by the Azure's MobileServiceTable API, this table is generated

Figure 6.7 shows a screenshot of the table's schema from Azure portal:

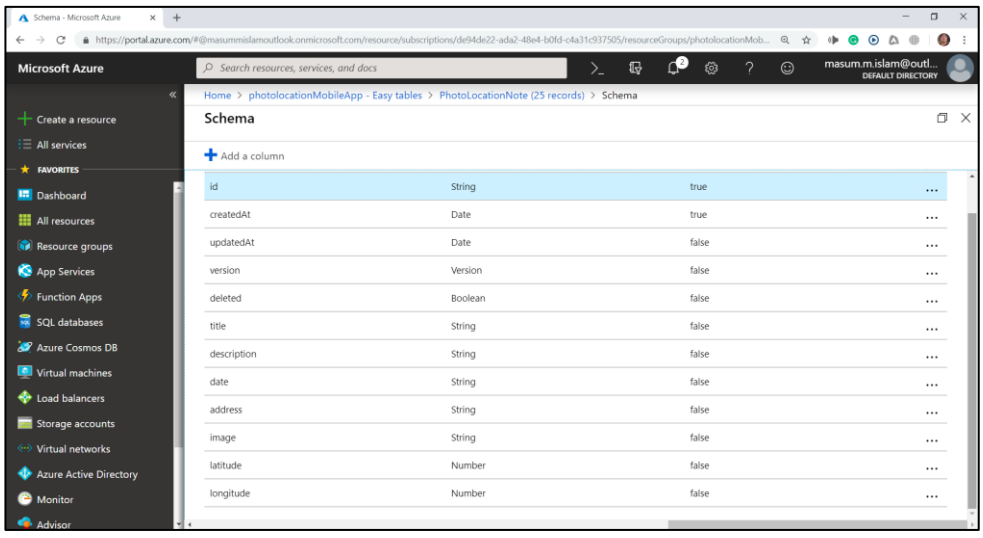


Figure 6.7 Azure SQL database table

Figure 6.7 shows complete Azure SQL database table with data type.

6.8 Azure Configures Summery

It is very hard and confusing to keep track of all the setting from the Azure. All settings are scarred in various features, because we are using couple features from the Azure. Below are all the required settings from Azure.

1. Storage account connection string. Azure Blob storage and Android app requires this connection string. Found at section 6.1.2

2. We need API key from Google Console Project to use it later in the Azure Notification Hub. Found at section 6.2.1
3. We need Google Console Project number. We need it for the Android application. Found at section 6.2.1

4. We need Azure Notification Hub name and access string. Found at 6..2.2

5. We need Azure Mobile App URL to access Azure data securely. Found at section 6.3

6.9 Azure Libraries

First of all we need a few Azure libraries to call Azure APIs from Android application. My app's build.gradle file's dependencies section will clarify this to a developer. Below are my app's build.grdle file's dependencies:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    androidTestImplementation('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    // Azure storage account library
    implementation 'com.microsoft.azure.android:azure-storage-android:2.0.0@aar'
    implementation 'javax.xml.stream:stax-api:1.0-2'
    implementation 'org.apache.commons:commons-lang3:3.0'
```

```
implementation 'com.google.firebase:firebase-core:16.0.1'
implementation 'com.google.android.gms:play-services-analytics:16.0.0'
implementation 'com.google.android.gms:play-services-maps:16.0.0'
implementation 'com.google.android.gms:play-services-location:16.0.0'
implementation 'com.google.android.gms:play-services-gcm:16.0.0'
// Azure notification hub library
implementation 'com.microsoft.azure:notification-hubs-android-sdk:0.4@aar'
// Azure notification hub push notification handler library
implementation 'com.microsoft.azure:azure-notifications-handler:1.0.1@aar'
// Azure mobile app library
implementation 'com.microsoft.azure:azure-mobile-android:3.2.0@aar'
implementation 'com.squareup.okhttp:okhttp:2.5.0'
implementation 'com.google.http-client:google-http-client-android:+'
implementation 'com.google.api-client:google-api-client-android:+'
implementation 'com.google.api-client:google-api-client-gson:+'
implementation 'com.squareup.okhttp:okhttp:2.5.0'
implementation 'com.google.guava:guava:18.0'
testImplementation 'junit:junit:4.12'
```

I have a class NotificationSettings that keeps all Azure Notification Hub setting. These settings are used all over the application. Class implementation is shown below:

```
public class NotificationSettings {
    // Google console project number
    public static String SenderId = "622396949720";
    // Notification hub name
    public static String HubName = "PhotoLocationNote2NotificationHub";
    // Notification listen connection string to listen for push notification
    public static String HubListenString = "End
     point=sb://photolocationnote2notificationhubnamespace.servicebus.windows.net/;"+
     "SharedAccessKeyName=DefaultListenSharedAccessSignature;"+
      "SharedAccessKey=+R9mhNtMtpihM+fBSDmD4hShSF86aXJgtSSPdX0QqjQ=";
     // Notification full access connection string to send or manage push notification
       public static String HubFullAccessString = "End
        point=sb://photolocationnote2notificationhubnamespace.servicebus.windows.net/;"+
        "SharedAccessKeyName=DefaultFullSharedAccessSignature;"+
         "SharedAccessKey=rNiMaoRsAOyKMKDT6B4sSrc5osnd6SqUW7K8kVpq01U=";
}
```

7 Testing

The positioning capabilities of wireless applications such as Location-based services (LBS) are used to collect the user's current geographical location. This kind of app provides the users information based on their current location’s information. Location data is huge and it requires quality a network to assured quality of the LBS. That is why testing is necessary to ensure the quality validation. The main purpose of the LBS testing is to deuce erroneous behaviour of LBS apps.

I took the above fact into account while I tested Photo Location Note app. First of all I planned to test this app in a few odd locations such as in the forest, sea coastal area and higher altitudes. Thus I chose Nuuksio national park, Haukilahti sea coast, and Leppävaara panorama tower.

7.1 Outdoor Network Connectivity Test

I was expecting less cellular signals at Nuuksio national park, but my assumption was wrong. I always had 3/4 bars signal there. Thus the app worked perfectly. I tested this app using Samsung 9S and Samsung Galaxy Tab A. The cellular provider was Telia and Elisa. Next in my search of low cellular signals I went to costal area of Haukilahti. Again my expectation to find low cellular signals at Haukilahti sea coast was wrong. I got full a full signal of 5 bars. Again the app worked perfectly. It is hard to find a low signal zone in the capital area of Finland. Again, I tested this app using similar device and cellular provider.

7.2 Low Cellular Signal Zone

When travelling by commuter train between Pasila railway station and Helsinki central station, there is most of the time a very low signal or no signal while using Telia cellular provider. I was happy to find such location to test my app. I found a few irregularities and bugs when there is no network. This application needs an internet access, so the only option for me to show the following warning signal is shown in figure 7.2. As soon as MOBILE DATA ENABLE or WIFI ENABL is selected, I open respective settings and kill this app.

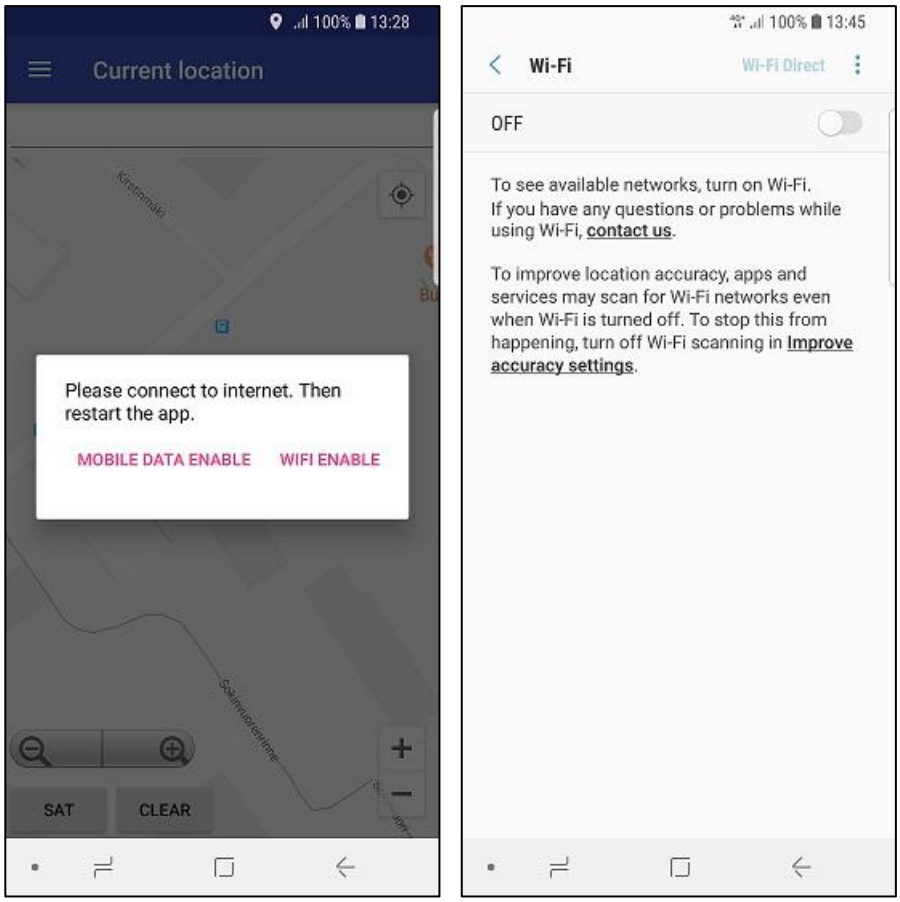


Figure 7.2 No network dialog

When there are any network connectivity issues, I show the dialog in figure 7.2. When clicking a button, it opens the respective setting.

7.3 Test Processes

A successful test planning contains step of test activities. That helps the test engineers to execute the test plan. Eventually it helps the test outcomes. There are many requirements. The testing steps may satisfy the quality of mobile LBS apps. In this process, the major necessities are mentioned below from the LBS's perspective:

- Functionality Test
- Network Connectivity Test
- Scalability Test
- Usability Test

- Performance Test
- Security and Privacy Test
- Context Change Test
- User location Test
- Cloud-Based Services Test

7.4 Test Conclusion

LBS became one of the major features that mobile devices bring. To figure out the functionality and correctness of the LBS mobile apps, an analysis of the performance is required. Deep detail testing is required to ensure the quality. However, testing these apps is difficult task because multiple techniques such as including positioning systems, the mobile devices limitations, internet, change of context and great amount of location data are applied.

8 Further Development

Espoo City Corporation manages user credentials in Windows Active Directory (AD). My intention was to synchronize local AD with Azure Active Directory to ensure that Espoo City Corporation's employee can log into this application using the same credentials. I could not implement this feature due to job alteration. Therefore, at this moment there is no authentication feature in this app.

I wanted to implement an extra feature as Internet of Things (IoT) communication over the cloud by using Raspberry Pi 3 B+. It was not required by the Espoo City Corporation. I just wanted to implement it as an extra feature. However, due to time limitations, I am leaving it for my fellow students to implement.

Every software/app has a version. Gradually I have added features for the applications created for this project. The current version of this app is 7.0. I expect there will be more features requested in the future. Thus it is a continuous development process.

8.1 Digital Survey Service Benefits

Before I published this app to the Google Play Store, I locally installed this app for a few Espoo City Corporation ground stuff member's handset mostly for testing in various device ranges. They used this app for their professional work for a few days. They found some minor bugs and suggest some ideas for improvement. Mostly they were pleased with the application. According to them, it reduced survey time about 95% and it also reduces costs.

A manual survey always has two parts. First, pictures are taken from the ground, and notes are written. After that the staff return to the office to transfer pictures to a local workstation and then transfer them to the local database. The staff tries to match paper the notes with the pictures.

It was previously discussed that a lot of human errors often occur in this process. With the help of the new application, they can instantly take a picture and write a note in the same app from the ground and transfer it to the Azure cloud directly. As soon as one photo and note is uploaded to the cloud, every member of the staff who requires the

picture, get an instant notification. This will not require more work and consequently, a survey that used to take 1000 minutes, needs only 50 minutes with the new application.

8.2 Unimplemented Features

Espoo City Corporation staff wanted to have Finnish language for this app, but unfortunately, I was already working for another company. Luckily, Espoo City Corporation staff managed to use it without any authentication as I could not synchronize on premise AD to Azure AD. There was also another request to implement a messenger service similar to WhatsApp to enable all communication between employees and managers to be saved in a single place. Unfortunately I could not do this because of my new work.

8.3 Project Outcome

The targets set for this project were successfully achieved. Most of the required features were implemented. Basically **Photo Location Note** was my first professional Android App done for a company and I learned a lot about Android application development as well as Azure cloud services. I believe these valuable skills will be beneficial for my professional career.

I strongly support open source approaches because they can always be improved by other developers. That is why I published all source code at GitHub. Anybody is welcome to make suggestion and make a pull request and fork in the GitHub link:
https://github.com/madsum/PhotoLocationNote

The application created for this thesis was published at Google Play Store by the name “Photo Location Note” Link:
https://play.google.com/store/apps/details?id=com.home.ma.photolocationnote

9 Conclusion

The objective of the project was to develop an Android mobile application for a type of survey that communicates with Azure could backend. The goal was met and the application exploits verities features of the device such as camera, gallery, SQLite database and GPS.

The project gave me an opportunity to explore cloud computing and I got a clear understanding of cloud computing, Internet of Things (IoT) communication over the clouds as well as the process of developing an application for Android with Azure cloud backend.

The application created for this thesis project has many common features such as map, current location and a photo snapshot. It also enables to store data in local SQLite database as well as in cloud backend. Although there are many applications available which have these common features, this app has one unique feature, namely embedding current address and time together. So far, I have not found a single app that has this feature. This feature really makes it stand out from the crowd.

During the project, I was the only developer in Espoo City Corporation and I had to do complete the application from scratch. As a result, I had to involve all the Software Development Life Cycle (SDLC) phases. I am proud to have successfully completed the project on schedule under very limited resources. The desired functionalities of this application are working as required by Espoo City Corporation. This application has been successfully introduced to Espoo City Corporation.

References

1. Recombu. P. 4. Available at:
URL: https://recombu.com
Accessed: 04.02.2018

2. Android developer console dashboard. P. 4. Available at:
URL: https://developer.android.com/about/dashboards/index.html
Accessed: 05.02.2019

3. Microsoft Azure Documentations. P. 9. Available at:
URL: https://docs.microsoft.com/en-us/azure/index
Accessed: 06.02.2018

4. Installing Microsoft Azure Pack P. 14. Available at:
https://blogs.technet.microsoft.com/canitpro/2015/07/22/step-by-step-installing-microsoft-azure-pack-for-windows-server/
Accessed: 17.02.2018

5. Microsoft Azure P. 13 Available at:
https://azure.microsoft.com/en-us/overview/what-is-iaas/
Accessed: 30.03.2018

6. Microsoft TechNet P. 14 Available at:
https://blogs.technet.microsoft.com/canitpro/2015/07/22/step-by-step-installing-microsoft-azure-pack-for-windows-server/
Accessed: 31.03.2018

7. Microsoft Azure hybrid-integration P. 14 Available at:
https://iamnetworks.net/cloud/cloud-network-infrastructure-services/microsoft-

azure-hybrid-integration/
Accessed: 31.03.2018

8. IEEE Institute of Electrical and Electronics Engineers P. 10 Available at: https://www.ieee.org/index.html
Accessed: 03.04.2018

9. Android Screen Sizes P. 34 Available at: https://developer.android.com/training/multiscreen/screensizes
Accessed: 20.09.2018

10. Android Screen and Resolution P. 35 at: http://www.webs.co.kr/index.php?mid=adnroid&document_srl=20138
Accessed: 20.09.2018

11. Android Navigation drawer P. 38 at: https://developer.android.com/training/implementing-navigation/nav-drawer
Accessed: 20.09.2018

12. Manage storage account settings P. 45 Available at: https://docs.microsoft.com/en-us/azure/storage/common/storage-account-manage?toc=%2fazure%2fstorage%2ffiles%2ftoc.json
Accessed: 25.09.2018

13. Azure Notification Hubs P.49 Available at: https://azure.microsoft.com/en-us/services/notification-hubs/
Accessed: 26.09.2018

14. Tutorial: Push notifications to Android P.50 Available at: https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-android

push-notification-google-gcm-get-started
Accessed: 27.09.2018

15. Azure Mobile Apps P. 52 Available at:
https://azure.microsoft.com/en-au/services/app-service/mobile/
Accessed: 28.09.2018

16. Azure SQL Server databases P. 54 Available at:
https://azure.microsoft.com/en-us/services/sql-database/
Accessed: 02.10.2018

17. Azure Serverless Compute P. 56 Available at:
https://azure.microsoft.com/en-us/services/functions/
Accessed: 04.10.2018

18. Azure Serverless Compute with Functions P. 57 Available at:
https://azure.microsoft.com/en-us/services/functions/
Accessed: 04.10.2018

19. A Viewpoint on Location-Based Mobile Apps Testing P. 58 Available at
https://www.computer.org/csdl/proceedings/bigdataservice/
2017/6318/00/07944956.pdf
Accessed: 10.10.2018

Google Play Store Link:-

https://play.google.com/store/apps/details?id=com.home.ma.photolocationnote

Source repository GitHub link:-

https://github.com/madsum/PhotoLocationNote