

**Czech University of Life Sciences Prague**  
**Faculty of Economics and Management**  
**Department of Information Engineering**



**Diploma Thesis**

**Application programming for Android platform using a cloud database**

**Mayur Sanpurkar**

## DIPLOMA THESIS ASSIGNMENT

Bc. Mayur Sanpurkar

Informatics

Thesis title

**Application programming for Android platform using a cloud database**

---

### Objectives of thesis

The goal of this thesis is to demonstrate application programming for the Android mobile platform in the Java programming environment using a cloud database on a small practical example about restaurant market where people can choose the best restaurant based on various parameters such as price, popularity, location, etc. Users can also write their experience into the connected database.

### Methodology

First part of thesis will contain a theoretical review of all tools and techniques used for the development. Next, functional requirement will be specified. Then, software implementation in Java language and Android Studio IDE will follow. This program will be tested and uploaded into the Google Play store service.

## The proposed extent of the thesis

60-70 pages

## Keywords

Android; Java; mobile application programming

---

## Recommended information sources

Griffiths, D. and Griffiths, D. (2017). Head first Android development. 2nd ed. Sebastopol, CA: O'Reilly Media.

Madusanka, I. and Fernando Pulle, R. (2016). Busy Programmer's Guide To Firebase with Android Learn by doing: less theory, more results. 1st ed. Amazon.

Moroney, L. (2017). The definitive guide to firebase. 1st ed. Seattle, Washington: Apress.

Phillips, B., Marsicano, K. and Stewart, C. (2018). Android programming. 3rd ed. Atlanta: Big Nerd Ranch.

---

## Expected date of thesis defence

2019/20 WS – FEM (February 2020)

## The Diploma Thesis Supervisor

doc. Ing. Vojtěch Merunka, Ph.D.

## Supervising department

Department of Information Engineering

Electronic approval: 19. 12. 2019

**Ing. Martin Pelikán, Ph.D.**

Head of department

Electronic approval: 19. 12. 2019

**Ing. Martin Pelikán, Ph.D.**

Dean

Prague on 03. 01. 2020

## **Declaration**

I declare that I have worked on my diploma thesis titled “Application programming for Android platform using a cloud database” by myself and I have used only the sources mentioned at the end of the thesis. As the author of the diploma thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on .....

---

**Mayur Sanpurkar**

## Acknowledgement

I would like to thank my supervisor doc. Ing. Vojtěch Merunka, Ph.D. for his support and guidance while working on this thesis. Additionally, I would like to thank my family and close friends for their encouragement, support and belief in me.

# Application programming for Android platform using a cloud database

## Abstract

Nowadays, with increased usage of mobile devices in human life, many of the day to day tasks are done using different mobile apps, such as online shopping, payments, etc. Before buying anything, most of the people check its reviews online, which helps them to make a better choice. Same principle applies when people want to go to restaurants for lunch or dinner with friends and family.

The purpose of this thesis was to develop an Android application called “Places to Eat”. It uses cloud-based database, which allows users to search for restaurants based on filters, such as cuisine type, location and price range. There is also an option to sort the restaurants based on price range, popularity or ratings given by other users.

Google Firebase is used for services and database. Data will be stored in Firestore cloud-based database and Firebase services will be used for user authentication and data storage and retrieval to and from the database. In order to learn more about the data and users, Google Analytics is used for data analysis.

Reviews and ratings about restaurants help customers to choose a better restaurant or the one which is very popular for its food and quality. While choosing restaurant, price and variety of food is also important for many people. Keeping all these things in mind, this application is developed. As a result, the application is fully functional and suggests users a list of restaurants from the database and allows a search functionality, where user can filter restaurants based on different parameters.

**Keywords:** Android, Java, mobile application, Firebase, restaurants, food, reviews and ratings, cloud storage.

# Programovací aplikace pro platformu Android využívající cloudové databáze

## Abstrakt

V dnešní době se zvýšeným využíváním mobilních zařízení v lidském životě se mnoho každodenních úkolů provádí pomocí různých mobilních aplikací, jako je online nakupování, platby atd. Před nákupem čehokoli si většina lidí přečte recenze online, aby se mohli lépe rozhodnout. Stejný princip platí, když lidé chtějí jít do restaurace na oběd nebo večeři s přáteli a rodinou.

Účelem této práce bylo vyvinout aplikaci pro Android nazvanou „místa k jídlu“. Ta využívá databáze založené na cloudu, které umožňují uživatelům vyhledávat restaurace na základě vybraných parametrů, jako je například typ kuchyně, její umístění nebo cenové rozpětí. Dále je možné seřadit restaurace podle ceny, popularity nebo hodnocení uživatelů.

Google Firebase je využíván pro služby a databáze. Data budou uložena v cloudové databázi Firestore a služby Firebase budou použity pro ověřování uživatelů, ukládání a načítání dat do a z databáze.

Google Analytics je používán za účelem analýzy informací o datech a uživatelích.

Recenze a hodnocení restaurací pomáhají uživatelům vybrat lepší restauraci nebo restauraci, která je velmi oblíbená pro své jídlo a kvalitu. Při výběru restaurace je pro mnoho lidí také důležitá cena a rozmanitost jídel. Tato aplikace je vyvinuta s ohledem na všechny tyto věci. Výsledkem je, že aplikace je plně funkční. Navrhuje uživatelům seznam restaurací z databáze a dává jim možnost vyhledávání restaurace podle různých parametrů.

**Klíčová slova:** Android, Java, mobilní aplikace, Firebase, restaurace, jídlo, recenze a hodnocení, cloudové úložiště.

# Table of Contents

<b>1 INTRODUCTION.....</b>	<b>11</b>
1.1 Comparison of Android and iOS .....	12
<b>2. Objectives and Methodology.....</b>	<b>14</b>
2.1 Objectives .....	14
2.2 Methodology.....	14
<b>3 Literature Review.....</b>	<b>15</b>
3.1 Market Research .....	15
3.2 Android Operating System Overview.....	18
3.2.1 Android Platform Architecture.....	19
3.2.2 Application Fundamentals.....	22
3.2.3 App components.....	23
3.2.4 The Manifest File .....	29
3.2.5 App Resources.....	30
3.2.6 Android Studio and Development Environment .....	31
3.3 Firebase and Firestore.....	37
3.3.1 Firebase .....	37
3.2.2 Firestore.....	38
<b>4 Practical part .....</b>	<b>40</b>
4.1 Application Architecture.....	40
4.2 The Implementation of the Application.....	41
4.2.1. Application Layer.....	41
4.2.2 Services .....	54
4.2.3 Cloud Database .....	64
<b>5 Result and Discussion .....</b>	<b>67</b>
5.1 Google Play Store.....	67
<b>6 Conclusion.....</b>	<b>69</b>
6.1 Future Enhancements .....	69
<b>7 REFERENCES.....</b>	<b>70</b>



# List of Pictures

Figure 1 - Android Architecture .....	19
Figure 2 - Activity Life Cycle .....	24
Figure 3 - Android Studio .....	32
Figure 4 - Project Structure .....	34
Figure 5 - App Architecture .....	40
Figure 6 - Sign in Form.....	41
Figure 7 - Sign in with Email.....	41
Figure 8 - Sign up.....	42
Figure 9 - Registered User's password.....	43
Figure 10 - Registered User's Email.....	43
Figure 11 - onCreate () Method .....	43
Figure 12 - Activity_main.xml.....	44
Figure 13 - onStart () Method .....	45
Figure 14 - startSignin () Method .....	45
Figure 15 - Restaurants List .....	46
Figure 16 - Firestore update .....	47
Figure 17 - SnapshotListener .....	48
Figure 18 - onEvent () Method .....	48
Figure 19 - Document and Collection.....	49
Figure 20 - Filters.....	50
Figure 21 - onFilter () method.....	51
Figure 22 - Filter Results .....	52
Figure 23 - Ratings and Review 2.....	53
Figure 24 - Ratings and Review.....	53
Figure 25 - Firebase Services.....	55
Figure 27 - Firebase New Project .....	56
Figure 28 - Firebase Google Analytics .....	57
Figure 29 - Firebase Select Account .....	57
Figure 30 - Firebase SDK Setup .....	58
Figure 32 - Build.gradle 1 .....	59

Figure 31 - Firebase JSON download .....	59
Figure 33 - Build.gradle 2 .....	60
Figure 34 - Firebase SDK verification .....	60
Figure 35 - Firebase Console .....	61
Figure 36 - Authentication .....	62
Figure 37 - Sign in Providers .....	62
Figure 38 - Email Template.....	63
Figure 39 - Database .....	65
Figure 40 - Restaurants collection .....	65
Figure 41 - Rules.....	66

## List of Tables

Table 1 - Firebase Services .....	55
-----------------------------------	----

# 1 INTRODUCTION

Since the invention of telephone in the late 18th century, the field of telecommunications had numerous major advances in modulation techniques, information theory, coding, etc. As the communication theory grew, the newer telephones had the need to be connected by wires and mobile telephones were born. These 21st century mobile telephones allowed users to make phone calls from anywhere, as long as they are in the radio coverage of their base stations and able to maintain an active link (commonly referred to as cells). However, these mobile phones were incapable of doing anything other than making phone calls and storing the user's contact list.

With the explosive growth of the World Wide Web, the people's need to be connected to the Internet all the time increased and these devices proved to be extremely restrictive. As the variety of such needs increased, the new breed of smart devices was needed that had capabilities to run customized/specialized applications to fulfill a user's computing and connectivity needs. These new devices were called Smartphones.

With time, the smartphone matured to be a small computer on which the mobile telephone was also nothing but one of many other applications. They featured music players, camera, photo/video surfing programs, games, etc., as other possible applications. These smartphones are low powered, energy constrained computers which required a light-weight but highly efficient operating system to run all of a user's applications.

Many efficient solutions were devised over time by companies such as RIM, Palm, etc. However, only two solutions stood above the pack due to their efficiency and ease of usage: Google's Android Operating System and Apple's iOS running on iPhones. Mobile applications (commonly referred as "apps"), are considered to be one of the fastest growing trends in Information Systems industry (**Eddy, 2011**). Users enjoy the variety of features that mobile apps can provide quickly and without introducing unnecessary complexity into their designs. As a result, mobile apps present a more popular interface for interaction with business systems than using web applications via Web Browser.

## 1.1 Comparison of Android and iOS

While Android and iOS provide efficient solutions for fulfilling the user's computing needs on resource constrained devices, there are some key differences. At the time of writing this document iOS version 6 is the latest on iPhone and version 4.0 on Android. One of the primary differences between the two is in the hardware platforms on which they run. While iOS is tailored to run on Apple manufactured smart devices such as iPhones, iPads, etc. only, Android, being based on open sourced Linux operating system can be run on a variety of devices.

As a result, a number of OEM (Original Equipment Manufacturer) companies choose Android for their devices, thereby boosting its market share. However, due to this very reason, the Android operating system as well as any application written for it needs to be tested on a variety of devices before being released. Secondly, the Android operating system is open sourced, supports much more open development platforms and ability to connect/run third party tools to enhance functionality of applications.

Apple, on the other hand, imposes very strict application development guidelines and screening procedure to allow any application to be made available widely. The tools used for any app development on iOS are very strictly controlled by Apple. Also, the testing and debugging tools available on Android via its IDE are very mature compared to iOS's Xcode.

The idea behind this thesis was to build a simple Android application or app that would help users to search restaurants based on criteria such as cuisine, location, price and last option is sort: can be based on price, popularity and reviews. People like to know best places to eat around the city so that they can enjoy great food and drinks. Rating of the restaurants helps user to choose the best place.

Also, price is very important factor for many people, because they don't want to spend more money for some food which is cheaper and has better quality at some other place. Reviews help both: customers to find good place based on their needs and restaurant owners to get feedback on their service. Keeping these things in mind, the application was developed.

One of the goals of the thesis was to implement the application using the latest technologies that are nowadays on the market. It is based on the features of the newest Android update — the Material Design and Firebase which is Google's cloud storage service.

The focus is on User Experience and seamless interaction with cloud storage which provides real time data changes to the user. This topic is often omitted during the design and also implementation phase. That is why a big part of the development was focused on the User Interface. Another goal of the thesis was to create an application that can be commercially useful in the practice.

## **2. Objectives and Methodology**

### **2.1 Objectives**

The main goal of the thesis is to develop an application that would help users to find restaurant and coffee shops in cities so that they can enjoy delicious food and drinks. The application would recommend restaurants based on cuisine, location and price. Sorting of restaurants is based on price or rating or popularity. The use of cloud-based database or storage is another goal of this thesis.

### **2.2 Methodology**

The functional requirements will be designed and developed based on the requirement analysis and data collected.

Functionalities requirements of the application are following:

- 1) The application should have functionality so that user can register and login using email address
- 2) The application should have connectivity with cloud-based database where data will be stored and retrieved.
- 3) The application should have search feature where user can search for restaurant
- 4) The application should have rating functionality so that user can rate and write review about a restaurant
- 5) The application should have sort function so that user can have tailored results based on his/her requirements.
- 6) The application should have logout function for the user.

## 3 Literature Review

### 3.1 Market Research

In 2018 the number of food and drink apps downloads grew by 115% comparing to 2016. Today more and more people prefer to check the place and its menu before actually going there. Restaurant and food apps encourage client's interaction, grow their loyalty and increase brand recognition. Just like e-commerce has changed how we shop, food apps are now changing how we eat. As some restaurants are taking out tables and expanding their kitchens to meet the food delivery trend, others are developing their own apps to change the experience inside the restaurants **(Cheng, 2018)**

According to a report from the investment bank UBS, the global value of this market is set to increase from 35 to 365 billion dollars per year by 2030. In the US, there are already reports of restaurants reducing the number of seats and expanding their kitchens to keep up with this development. This development shares a lot of similarities with how e-commerce has changed the conditions for the retail industry in the past few decades, but there are also clear differences. One such difference is that food and drinks, unlike clothes, electronics or books, still rely on geographical proximity. **(Cheng, 2018)**

For restaurants it is still vital to remain in attractive locations close to where their customers live. Because of this, the UBS report makes the prediction that food delivery apps won't kill local restaurant, instead it's the home cooked meals that they think run the highest risk of being reduced to history. But even if delivery apps won't replace your local restaurant, they do give a clear indication that customer demands are changing. Just like online shopping led to demands of greater convenience within retail, restaurant guests are starting to expect a greater amount of flexibility and speed. **(Cheng, 2018)**

A clear effect of this is that we're also seeing innovations when it comes to how we order and pay for food in restaurants. Perhaps the best example of this is the success of Starbucks' app. Their app allows customers to order and pay for drinks and food in advance and then simply pick it up at their local Starbucks cafe. Over 23 million Americans are estimated to have made a purchase through this app, making it the most popular payment app in the US, ahead of both Apple Pay and Google Pay. **(Cheng, 2018)**

In the Nordics we can also see examples of how restaurant apps are changing how we order and pay. One of Bambara's customers, the Swedish tapas restaurant Pinchos, has allowed their customers to order and pay through the restaurant's app since 2012 and Pinchos is now one of Sweden's fastest growing restaurant chains. In Norway, another one of Bambara's customers, OrderX, has shown how app-based ordering and payments can reduce lines and increase sales at events and festivals. **(Cheng 2018)**

This thesis discusses technologies incorporated in Android application development and how they apply to the research problem. As the official Android website describes this platform, “Android is a software stack for mobile devices that includes an operating system, middleware and key applications” Android provides the “core set of applications including an email client, SMS program, calendar, maps, browser, contacts, and others”, while additional applications can be downloaded through Google Play service **(Bishop, 2012)**.

Android powers millions of phones, tablets and other devices. Phones and tablets are mobile devices that can have Android applications installed on them. These applications are written in Java programming language and they are called mobile device applications or apps. Development techniques for apps are structured sets of Java code focused on implementing particular task that provides content for a mobile device application.

Although Java programming language includes a broad variety of topics, this thesis focuses on development techniques required for successful implementation of Restaurant Recommendation app. The following paragraphs analyze research efforts that addressed these techniques in the past. Many authors described Android application development fundamentals, which include setting up Android development environment on the machine, AndroidManifest.xml file, Activities, Intents, and XML layouts.

**Jackson (2011)** outlines “three major components of an Android development environment: Java, Eclipse, Android” and provides instructions on how to download and install necessary files to establish this environment. **Felker (2011)** does not explicitly state the components but rather points out that Java JDK, Android SDK, Eclipse IDE, and Android ADT need to be installed and configured on a machine.



The steps provided by these two authors are standard. They appear in many books written on Android development and are also presented on official Android website (**Ableson, King, and Sen (2011)**) present “four primary components of Android applications”: Activity, Service, BroadcastReceiver, and ContentProvider. It is noted that “a particular Android application might not contain all of these elements, but will have at least one of these elements” (**Ableson, King, and Sen, 2011**).

Since Activity “displays a UI (user interface) and responds to system and user initiated events” (**Ableson, King, and Sen, 2011**), it is used very frequently for Android applications. These Activities are declared in AndroidManifest.xml file, which provides “the foundation for any Android application” (**Phillips, Marsicano and Stewart, 2018**). Activities present their views through XML layouts and “communicate” with each other through Intents. Clear understanding of these concepts and Java programming language is a prerequisite to start implementing the development techniques used in Android applications.

## 3.2 Android Operating System Overview

**Android** is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance, with the main contributor and commercial marketer being Google. **(King, 2016)** Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, with the first commercial Android device launched in September 2008.

The current stable version is Android 10, released on September 3, 2019. The core Android source code is known as Android Open Source Project (AOSP), which is primarily licensed under the Apache License. This has allowed variants of Android to be developed on a range of other electronics, such as game consoles, digital cameras, PCs and others, each with a specialized user interface. Some well-known derivatives include Android TV for televisions and Wear OS for wearables, both developed by Google.

Android's source code has been used as the basis of different ecosystems, most notably that of Google which is associated with a suite of proprietary software called Google Mobile Services (GMS)**(Android, 2018)**, that frequently comes pre-installed on said devices. This includes core apps such as Gmail, the digital distribution platform Google Play and associated Google Play Services development platform, and usually apps such as the Google Chrome web browser. These apps are licensed by manufacturers of Android devices certified under standards imposed by Google.

Other competing Android ecosystems include Amazon.com's Fire OS, or LineageOS. Software distribution is generally offered through proprietary application stores like Google Play Store or Samsung Galaxy Store, or open source platforms like Aptoide or F-Droid, which utilize software packages in the APK format. Android has been the best-selling OS worldwide on smartphones since 2011 and on tablets since 2013.

As of September 2019, it has over two billion monthly active users, the largest installed base of any operating system, and the Google Play Store features over 2.8 million apps. **(Clement, 2019)**

### 3.2.1 Android Platform Architecture

Android is an open source, Linux-based software stack created for a wide array of devices and form factors. The following diagram shows the major components of the Android platform.

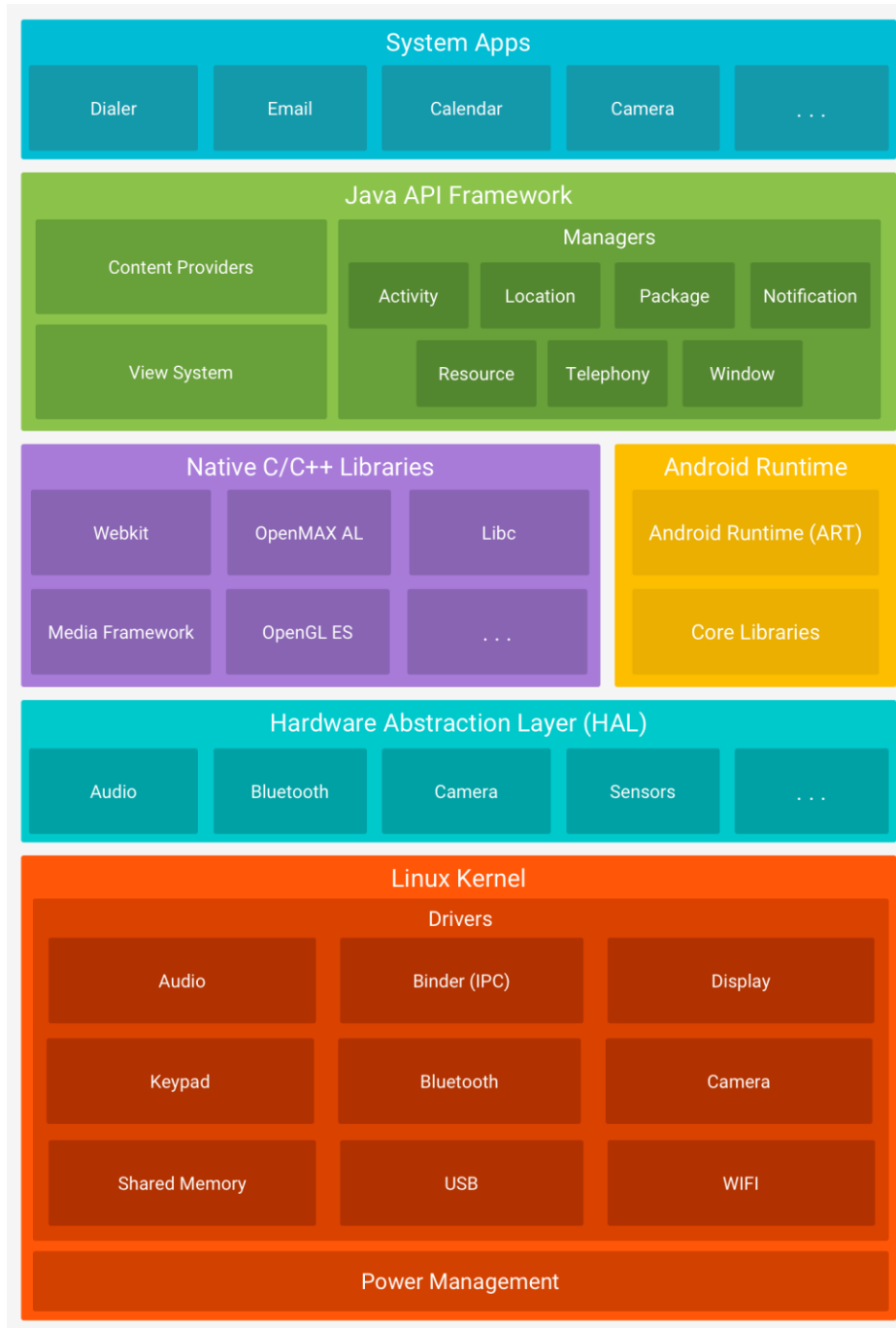


Figure 1 - Android Architecture

## **The Linux Kernel**

The foundation of the Android platform is the Linux kernel. For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management. Using a Linux kernel allows Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel. **(Android Developers, 2010)**

## **Hardware Abstraction Layer (HAL)**

The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component. **(Android Developers, 2010)**

## **Android Runtime**

For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART). ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed especially for Android that's optimized for minimal memory footprint. Build toolchains, such as Jack, compile Java sources into DEX bytecode, which can run on the Android platform. **(Android Developers, 2010)**

Some of the major features of ART include the following:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation
- Optimized garbage collection (GC)
- On Android 9 (API level 28) and higher, conversion of an app package's Dalvik Executable format (DEX) files to more compact machine code.
- Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watch points to monitor specific fields
- Prior to Android version 5.0 (API level 21), Dalvik was the Android runtime. If the app runs well on ART, then it should work on Dalvik as well, but the reverse may not be true.**(Android Developers, 2010)**

Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features that the Java API framework uses.

### **Native C/C++ Libraries**

Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++. The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps. For example, user can access OpenGL ES through the Android framework's Java OpenGL API to add support for drawing and manipulating 2D and 3D graphics in the app. If developing an app that requires C or C++ code, user can use the Android NDK to access some of these native platform libraries directly from user's native code. (Android Developers, 2010)

### **Java API Framework**

The entire feature-set of the Android OS is available to user through APIs written in the Java language. These APIs form the building blocks user needs to create Android apps by simplifying the reuse of core, modular system components and services, which include the following(Android Developers, 2010) :

- A rich and extensible View System to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all apps to display custom alerts in the status bar
- An Activity Manager that manages the life cycle of apps and provides a common navigation back stack
- Content Providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data Developers have full access to the same framework APIs that Android system apps use.

## System Apps

Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. Apps included with the platform have no special status among the apps the user chooses to install. So a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard (some exceptions apply, such as the system's Settings app).

The system apps function both as apps for users and to provide key capabilities that developers can access from their own app. For example, if an app would like to deliver an SMS message, user doesn't need to build that functionality by himself he can instead invoke whichever SMS app is already installed to deliver a message to the recipient user specify. (**Android Developers, 2010**)

### 3.2.2 Application Fundamentals

Android apps can be written using Kotlin, Java, and C++ languages. The Android SDK tools compile code along with any data and resource files into an APK, an *Android package*, which is an archive file with an **.apk** suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app. (**Android Developers, 2019**)

Each Android app lives in its own security sandbox, protected by the following Android security features:

- The Android operating system is a multi-user Linux system in which each app is a different user.
- By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. The Android system starts the process when any of the app's components need to be executed, and then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

The Android system implements the *principle of least privilege*. That is, each app, by default, has access only to the components that it requires to do its work and no more. This creates a very secure

environment in which an app cannot access parts of the system for which it is not given permission. However, there are ways for an app to share data with other apps and for an app to access system services:

- It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM. The apps must also be signed with the same certificate.
- An app can request permission to access device data such as the device's location, camera, and Bluetooth connection. The user has to explicitly grant these permissions.

### 3.2.3 App components

App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter the app. Some components depend on others.

There are four different types of app components:

- Activities
- Services
- Broadcast receivers
- Content providers

Each type serves a distinct purpose and has a distinct life cycle that defines how the component is created and destroyed. The following sections describe the four types of app components. (**Android Developers, 2019**)

#### 3.2.3.1 Activities

An activity is the entry point for interacting with the user. It represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities if the email app allows it. (**Android Developers, 2019**)

An activity facilitates the following key interactions between system and app:

- Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
- Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
- Helping the app handle having its process killed so the user can return to activities with their previous state restored.
- Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.) An activity is implemented as a subclass of the Activity class.

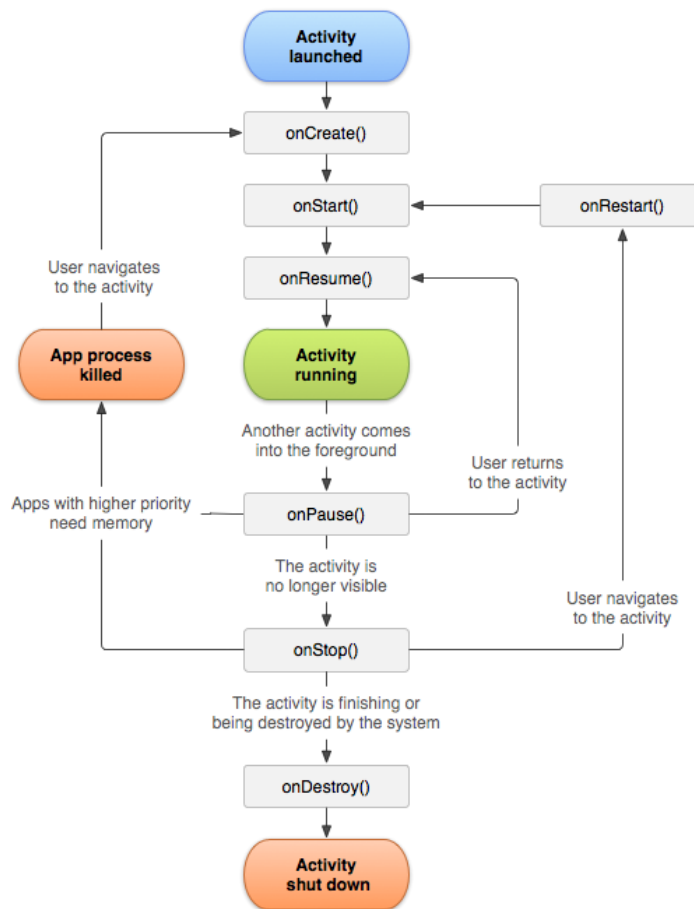


Figure 2 - Activity Life Cycle



The Activity class provides a core set of six callbacks: onCreate(), onStart(), onResume(), onPause(), onStop() and onDestroy(). The system invokes each of these callbacks as an activity enters a new state.

### **onCreate()**

On activity creation, the activity enters the Created state. In the onCreate() method, there should be basic application startup logic that should happen only once for the entire life of the activity.

### **onStart()**

When the activity enters the Started state, the system invokes this callback. The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. For example, this method is where the app initializes the code that maintains the UI.

### **onResume()**

When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the onResume() callback. This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app. Such an event might be, for instance, receiving a phone call, the user's navigating to another activity, or the device screen's turning off.

### **onPause()**

The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode).

### **onStop()**

When activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the onStop() callback. This may occur, for example, when a newly launched activity covers the entire screen. The system may also call onStop() when the activity has finished running, and is about to be terminated.

## **OnDestroy()**

OnDestroy() is called before the activity is destroyed. The system invokes this callback either because:

- 1.the activity is finishing (due to the user completely dismissing the activity or due to finish() being called on the activity), or
- 2.the system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)

When the activity moves to the destroyed state, any lifecycle-aware component tied to the activity's lifecycle will receive the ON\_DESTROY event. This is where the lifecycle components can clean up anything it needs to before the Activity is destroyed. (**Android Developers, 2019**)

### **3.2.3.2 Services**

A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity.

Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. There are actually two very distinct semantics services tell the system about how to manage an app: Started services tell the system to keep them running until their work is completed. This could be to sync some data in the background or play music even after the user leaves the app. Syncing data in the background or playing music also represent two different types of started services that modify how the system handles them:

- Music playback is something the user is directly aware of, so the app tells the system this by saying it wants to be foreground with a notification to tell the user about it; in this case the system knows that it should try really hard to keep that service's process running, because the user will be unhappy if it goes away.
- A regular background service is not something the user is directly aware as running, so the system has more freedom in managing its process. It may allow it to be killed (and then restarting

the service sometime later) if it needs RAM for things that are of more immediate concern to the user.

Bound services run because some other app (or the system) has said that it wants to make use of the service. This is basically the service providing an API to another process. The system thus knows there is a dependency between these processes, so if process A is bound to a service in process B, it knows that it needs to keep process B (and its service) running for A. Further, if process A is something the user cares about, then it also knows to treat process B as something the user also cares about. Because of their flexibility (for better or worse), services have turned out to be a really useful building block for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they should be running. (**Android Developers, 2019**)

### **3.2.3.3 Broadcast receivers**

A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running. So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event... and by delivering that alarm to a Broadcast Receiver of the app, there is no need for the app to remain running until the alarm goes off.

Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.

More commonly, though, a broadcast receiver is just a *gateway* to other components and is intended to do a very minimal amount of work. For instance, it might schedule a `JobService` to perform some work based on the event with `JobScheduler`. A broadcast receiver is implemented as a subclass of `BroadcastReceiver` and each broadcast is delivered as an `Intent` object. (**Android Developers, 2019**)

### 3.2.3.4 Content providers

A content provider manages a shared set of app data that user can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that an app can access. Through the content provider, other apps can query or modify the data if the content provider allows it. For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query the content provider, such as `ContactContract.Data`, to read and write information about a particular person.

It is tempting to think of a content provider as an abstraction on a database, because there is a lot of API and support built in to them for that common case. However, they have a different core purpose from a system-design perspective. To the system, a content provider is an entry point into an app for publishing named data items, identified by a URI scheme.

Thus an app can decide how it wants to map the data it contains to a URI namespace, handing out those URIs to other entities which can in turn use them to access the data. There are a few particular things this allows the system to do in managing an app:

- Assigning a URI doesn't require that the app remain running, so URIs can persist after their owning apps have exited. The system only needs to make sure that an owning app is still running when it has to retrieve the app's data from the corresponding URI.
- These URIs also provide an important fine-grained security model. For example, an app can place the URI for an image it has on the clipboard, but leave its content provider locked up so that other apps cannot freely access it. When a second app attempts to access that URI on the clipboard, the system can allow that app to access the data via a temporary *URI permission grant* so that it is allowed to access the data only behind that URI, but nothing else in the second app.

Content providers are also useful for reading and writing data that is private to user's app and not shared. A content provider is implemented as a subclass of Content Provider and must implement a standard set of APIs that enable other apps to perform transactions. For more information, see the Content Providers developer guide.

A unique aspect of the Android system design is that any app can start another app's component. For example, if user want to capture a photo with the device camera, there's probably another app that does that and user's app can use it instead of developing an activity to capture a photo.

User don't need to incorporate or even link to the code from the camera app. Instead, user can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to user's app so he can use it. To the user, it seems as if the camera is actually a part of user's app.

When the system starts a component, it starts the process for that app if it's not already running and instantiates the classes needed for the component. For example, if user app starts the activity in the camera app that captures a photo, that activity runs in the process that belongs to the camera app, not in user's app's process. Therefore, unlike apps on most other systems, Android apps don't have a single entry point (there's no main () function).

Because the system runs each app in a separate process with file permissions that restrict access to other apps, user's app cannot directly activate a component from another app. However, the Android system can. To activate a component in another app, deliver a message to the system that specifies user's intent to start a particular component. The system then activates the component for user. **(Android Developers, 2019)**

### **3.2.4 The Manifest File**

Every app project must have an AndroidManifest.xml file (with precisely that name) at the root of the project source set. The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play. Before the Android system can start an app component, the system must know that the component exists by reading the app's manifest file, AndroidManifest.xml. App must declare all its components in this file, which must be at the root of the app project directory. **(Android Developers, 2019)**

Among many other things, the manifest file is required to declare the following:

- The app's package name, which usually matches your code's namespace. The Android build tools use this to determine the location of code entities when building project. When packaging the app, the build tools replace this value with the application ID from the Gradle build files, which is used as the unique app identifier on the system and on Google Play.
- The components of the app, which include all activities, services, broadcast receivers, and content providers. Each component must define basic properties such as the name of its Kotlin or Java class. It can also declare capabilities such as which device configurations it can handle, and intent filters that describe how the component can be started.

- The permissions that the app needs in order to access protected parts of the system or other apps. It also declares any permissions that other apps must have if they want to access content from this app.
- The hardware and software features the app requires, which affects which devices can install the app from Google Play.

The manifest does a number of things in addition to declaring the app's components, such as the following:

- Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declares the minimum API level required by the app, based on which APIs the app uses.
- Declares hardware and software features used or required by the app, such as a camera, Bluetooth services, or a multi touch screen.
- Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.(**Android Developers, 2019**)

### **3.2.5 App Resources**

An Android app is composed of more than just code—it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the app. For example, user can define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using app resources makes it easy to update various characteristics of the app without modifying code. Providing sets of alternative resources enables user to optimize app for a variety of device configurations, such as different languages and screen sizes. (**Android Developers, 2019**)

For every resource that is included in Android project, the SDK build tools define a unique integer ID, which user can use to reference the resource from app code or from other resources defined in XML. For example, if an app contains an image file named logo.png (saved in the res/drawable/ directory), the SDK tools generate a resource ID named R.drawable.logo. This ID maps to an app-specific integer, which user can use to reference the image and insert it in your user interface.

One of the most important aspects of providing resources separate from your source code is the ability to provide alternative resources for different device configurations. For example, by defining UI strings

in XML, user can translate the strings into other languages and save those strings in separate files. Then Android applies the appropriate language strings to user's UI based on a language qualifier that user append to the resource directory's name (such as res/values-fr/ for French string values) and the user's language setting.

Android supports many different qualifiers for user's alternative resources. The qualifier is a short string that user include in the name of resource directories in order to define the device configuration for which those resources should be used. For example, when the device screen is in portrait orientation (tall), user might want a layout with buttons to be vertical, but when the screen is in landscape orientation (wide), the buttons could be aligned horizontally.

To change the layout depending on the orientation, user can define two different layouts and apply the appropriate qualifier to each layout's directory name. Then, the system automatically applies the appropriate layout depending on the current device orientation. (**Android Developers, 2019**)

### **3.2.6 Android Studio and Development Environment**

To build an Android application Integrated Development Environment or IDE is used. There are many IDEs available which can be used to develop Android application such as Android Studio, Eclipse, IntelliJIdea, DroidEdit and many more but the official IDE supported by Google is Android Studio. Before Android Studio, Eclipse was the official Android development environment. Used to code Java but can be expanded to other languages via plugins, it is still a powerful tool. By installing ADT or Android Development Tools plugin in Eclipse it was possible to develop application for Android. (**Android Developers, 2019**)

#### **Android Studio**

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development.

Android Studio was announced on May 16, 2013 at the Google I/O conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0.

Since May 7, 2019, Kotlin is Google’s preferred language for Android app development. Still, other programming languages are supported by Android Studio, such as Java and C++.

On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Apply Changes to push code and resource changes to your running app without restarting your app
- Code templates and GitHub integration to help you build common app features and import sample code
- Lint tools to catch performance, usability, version compatibility, and other problems
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

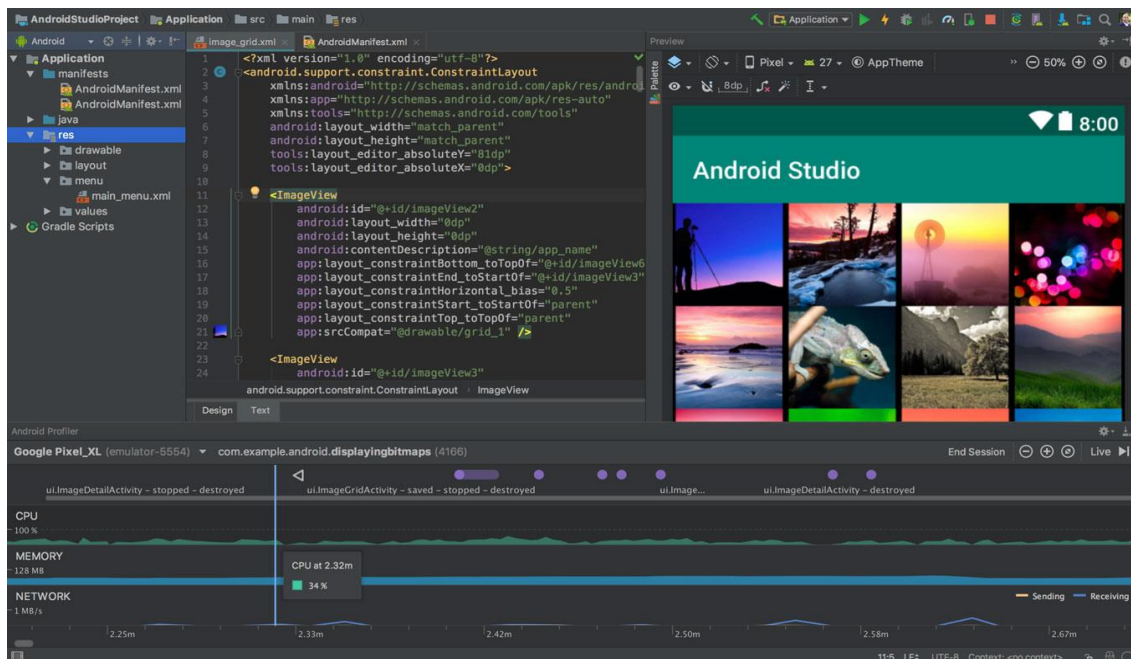


Figure 3 - Android Studio



## Project Structure

Each project in Android Studio contains one or more modules with source code files and resource files.

Types of modules include:

- **Android app modules**
- **Library modules**
- **Google App Engine modules**

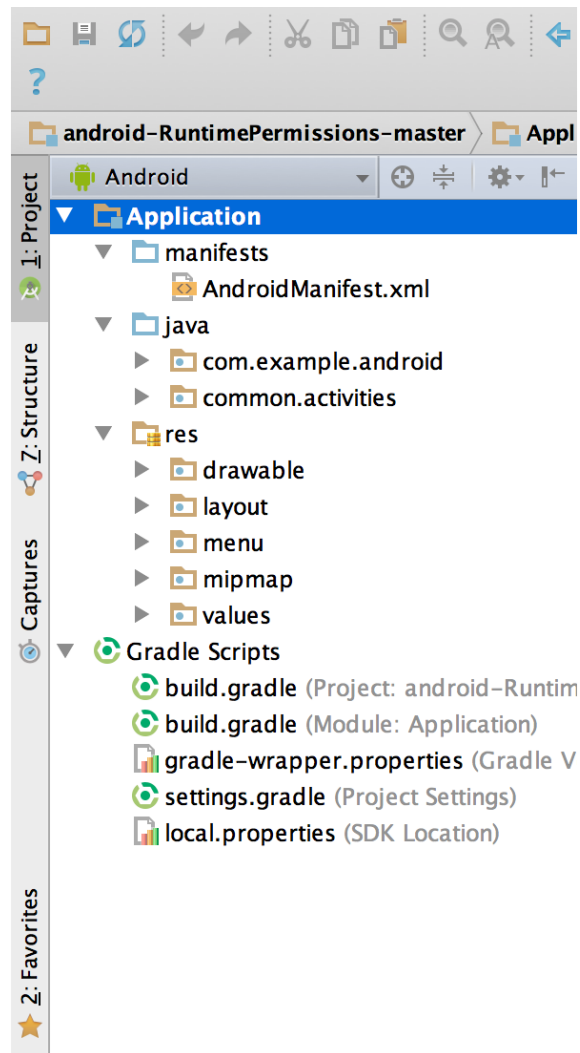
By default, Android Studio displays your project files in the Android project view, as shown in figure 1.

This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under Gradle Scripts and each app module contains the following folders:

- **Manifests:** Contains the AndroidManifest.xml file.
- **Java:** Contains the Java source code files, including JUnit test code.
- **Res:** Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select Project from the Project dropdown (in figure, it is showing as Android)



*Figure 4 - Project Structure*

### **System Requirements for Android Development:**

You won't need any high-end expensive stuff to begin with Android Development. In fact, you can even start developing Android apps even if you don't have an Android device.

Having said that following are resources required to begin with Android development

- Windows (XP or later), Linux (any recent Linux distribution) and Mac OS X (10.4.9 or later)
- Android Studio or Eclipse
- Android SDK
- Java

## Windows

- Microsoft® Windows® 7/8/10 (32- or 64-bit)
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,  
4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution
- For accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

## Mac

- Mac® OS X® 10.10 (Yosemite) or higher, up to 10.12 (macOS Sierra)
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,  
4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

## Linux

- GNOME or KDE desktop *Tested on Ubuntu® 14.04 LTS, Trusty Tahr (64-bit distribution capable of running 32-bit applications)*
- 64-bit distribution capable of running 32-bit applications
- GNU C Library (glibc) 2.19 or later
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,  
4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

- For accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality, or AMD processor with support for AMD Virtualization™ (AMD-V™) **Kumail, I. (2016)**

## **IDE (Eclipse or Android Studio)**

### **Android SDK**

The Android SDK (software development kit) is a set of development tools used to develop applications for Android platform. The Android SDK includes the following:

- Required libraries
- Debugger
- An emulator
- Relevant documentation for the Android application program interfaces (APIs)
- Sample source code
- Tutorials for the Android OS

### **Java**

A Java Development Kit (JDK) 8 for developing on Android.

## 3.3 Firebase and Firestore

As mentioned earlier during the development of this Android application cloud-based database Firestore and data storage is used which is Firebase. The main purpose of using these is to have real time data updates to the database and fetching those changes to the user. Firebase does this very fast as it is a Backend as a Service which uses WebSocket which are faster than normal HTTP to connect to the database. **(Rouse, 2019)**

### 3.3.1 Firebase

Google Firebase is a Google-backed application development software that enables developers to develop iOS, Android and Web apps. Firebase provides tools for tracking analytics, reporting and fixing app crashes, creating marketing and product experiment. It is a Backend-as-a-Service (BaaS). **(Rouse, 2019)**

Firebase offers a number of services, including:

- Analytics – Google Analytics for Firebase offers free, unlimited reporting on as many as 500 separate events. Analytics presents data about user behavior in iOS and Android apps, enabling better decision-making about improving performance and app marketing.
- Authentication – Firebase Authentication makes it easy for developers to build secure authentication systems and enhances the sign-in and onboarding experience for users. This feature offers a complete identity solution, supporting email and password accounts, phone auth, as well as Google, Facebook, GitHub, Twitter login and more.
- Cloud messaging – Firebase Cloud Messaging (FCM) is a cross-platform messaging tool that lets companies reliably receive and deliver messages on iOS, Android and the web at no cost.
- Real-time database – the Firebase Real-time Database is a cloud-hosted NoSQL database that enables data to be stored and synced between users in real time. The data is synced across all clients in real time and is still available when an app goes offline.
- Crashlytics – Firebase Crashlytics is a real-time crash reporter that helps developers track, prioritize and fix stability issues that reduce the quality of their apps. With crashlytics, developers

spend less time organizing and troubleshooting crashes and more time building features for their apps.

- Performance – Firebase Performance Monitoring service gives developers insight into the performance characteristics of their iOS and Android apps to help them determine where and when the performance of their apps can be improved.
- Test lab – Firebase Test Lab is a cloud-based app-testing infrastructure. With one operation, developers can test their iOS or Android apps across a variety of devices and device configurations. They can see the results, including videos, screenshots and logs, in the Firebase console. **(Rouse, 2019)**

### 3.2.2 Firestore

Cloud Firestore is a fast, fully managed, server less, cloud-native NoSQL document database that simplifies storing, syncing, and querying data for your mobile, web, and IoT apps at global scale. Its client libraries provide live synchronization and offline support, while its security features and integrations with Firebase and Google Cloud Platform (GCP) accelerate building truly server less apps.

It provides an automatically scaling solution built from the ground up to take advantage of Google Cloud Platform's powerful infrastructure. It is designed to provide a great developer experience and simplify app development with live synchronization, offline support, and ACID transactions across hundreds of documents and collections. Cloud Firestore is integrated with both Google Cloud Platform (GCP) and Firebase, Google's mobile development platform.

You can talk directly to Cloud Firestore from your mobile or web clients for a truly serverless solution. No need to set up an intermediary server to manage access to your data. This is great for prototyping, iterating, and getting a production system up and running quickly.

With Cloud Firestore, your applications can be updated in near real time when data on the back end changes. This is not only great for building collaborative multi-user mobile applications, but also means you can keep your data in sync with individual users who might want to use your app from multiple devices. **(Google Cloud, 2017)**

## **Features**

### **Security**

Cloud Firestore has built-in security access controls for data and enables simple data validation via a configuration language.

### **Data store mode**

Cloud Firestore supports the Data store API. You won't need to make any changes to your existing Data store apps, and you can expect the same performance characteristics and pricing with the added benefit of strong consistency.

### **Automatic upgrade**

Your Cloud Data store databases will be upgraded seamlessly and automatically shortly after the GA release of Cloud Firestore. No code changes required, and no downtime for your app.

### **ACID transactions**

Cloud Firestore has support for transactions, so if any of the operations in the transaction fails (and cannot be retried) the whole transaction will fail.

### **Multi-region replication**

With automatic multi-region replication and strong consistency, your data is safe and available, even when disasters strike.

### **Powerful query engine**

Cloud Firestore allows you to run sophisticated queries against your NoSQL data without any degradation in performance. This gives you more flexibility in the way you structure your data.

### **Built for cloud-native applications**

Typical workloads include mobile and web applications, collaborative multi-user applications, IoT asset tracking and real-time analytics, retail product catalogs, communications, social user profiles and activity, and gaming leaderboards. (**Google Cloud, 2017**)

## 4 Practical part

The practical part describes the details of the implementation of the application. It includes technologies used while developing the application, architecture of the application, flow of data in the application, programs, databases, clients, server, diagrams etc.

### 4.1 Application Architecture

Application architecture describes the behavior of the application or how the application components interact with each other, with databases and with users. It is more focused on how data is consumed and produced. Application architecture helps to maintain clear structure of the system and allows to modify based on the future requirements. Dividing application components into layers based on their functionality helps to have a simple architecture of the application.

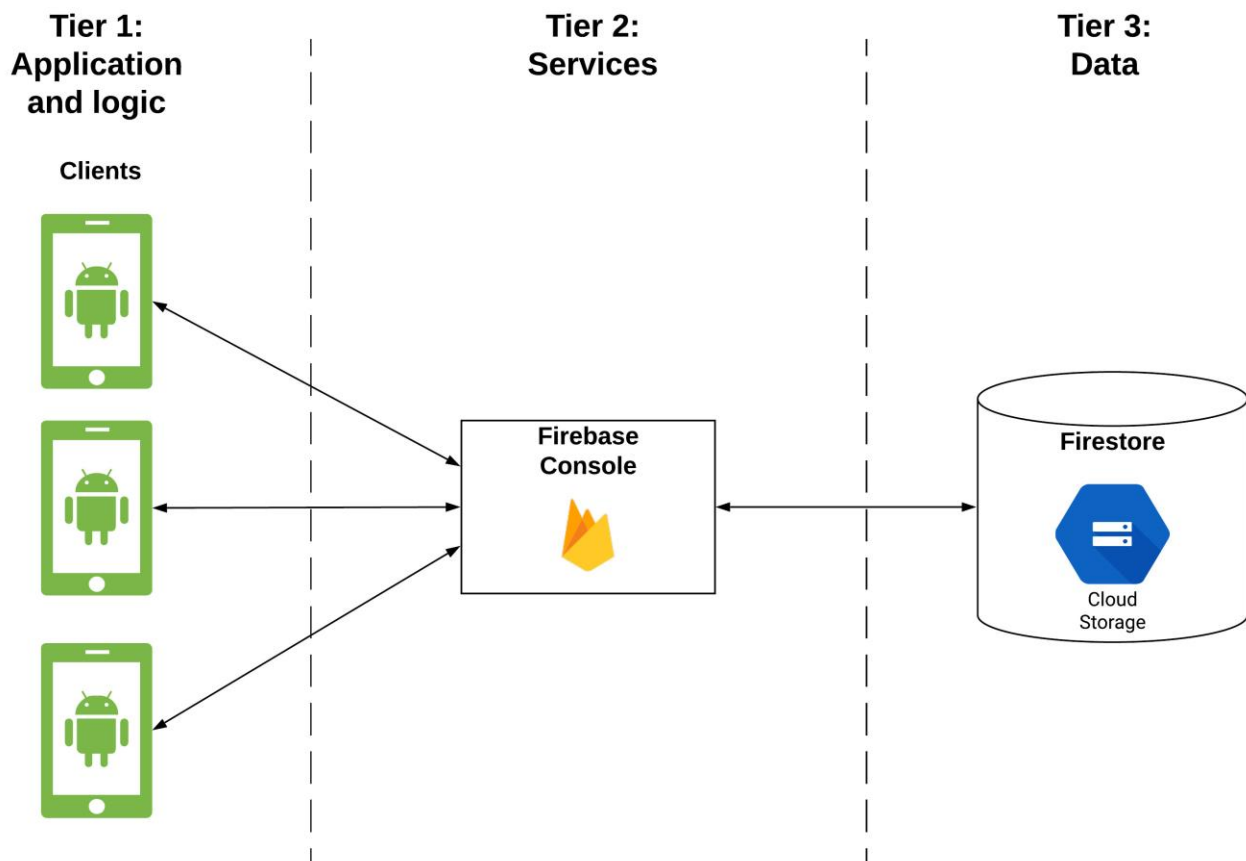


Figure 5 - App Architecture



## 4.2 The Implementation of the Application

1. Application and the logic
2. Services
3. Cloud database

### 4.2.1. Application Layer

The first layer is the Application layer and business logic, which is also client side of the application. It consist of the application and the application or business logic. This is the layer where user interacts with the system. This layer takes input from the user and sends it to the services. For example user enters data into a text box or text field and clicks on the submit button which send the data to services and which maps the data received from the user to appropriate keys or fields in the database and stores the data.

When user wants some information from the database then he/she enters the text into text field and requests for the data and when he receives the data from the database, it is displayed in a text view to the user. So, basically the application layer works as presentation layer where user send and receives data to and from the database. In this case the Android application acts as application layer where user interacts with it and performs necessary tasks. Let's take a detailed technical view at the application.

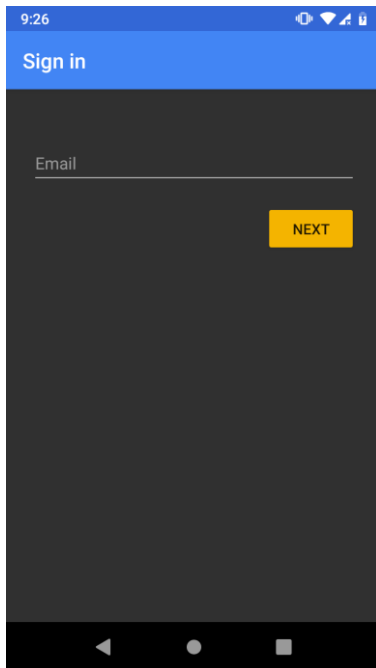


Figure 6 - Sign in Form

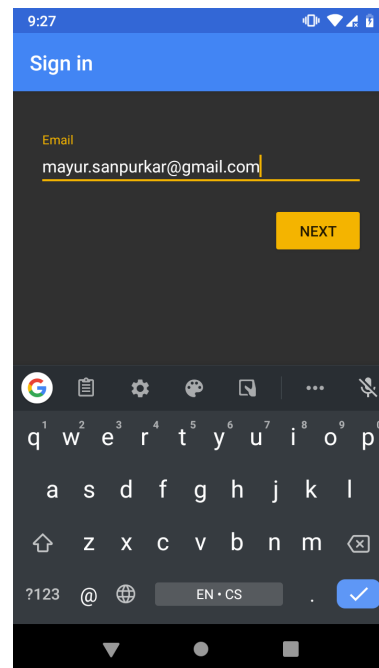
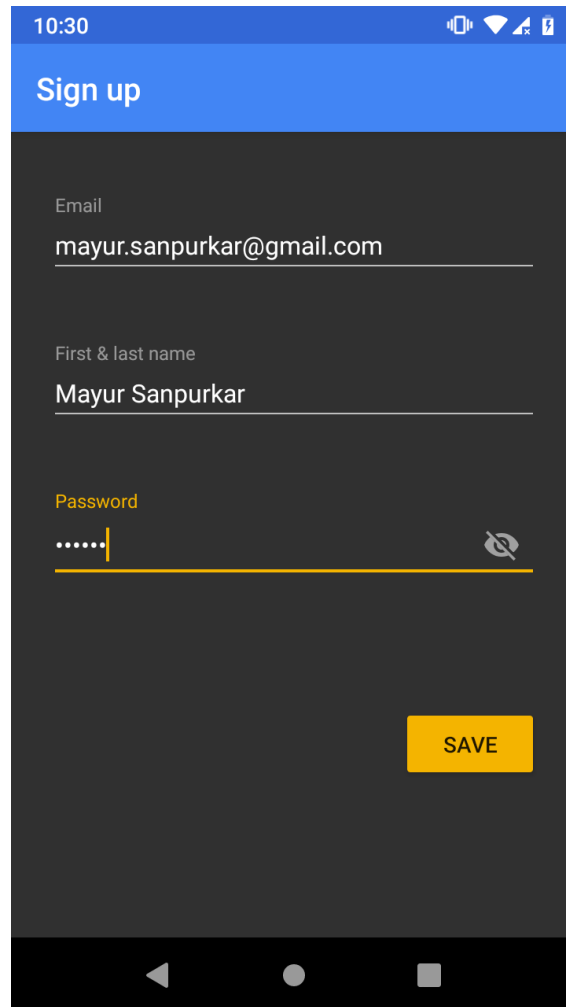


Figure 7 - Sign in with Email

As we can see above is the Sign in window that appears when the application is started. User has to enter the email address and click on **Next** button.



*Figure 8 - Sign up*

After that user will be presented with sign up form where he/she needs to enter details such as **First name** and **Last name** and **password**. After the when user clicks on the **SAVE** button he/she will be registered. This is the sign in process for anyone who is new user.

However, the screens presented for someone who is already a registered user are different as shown below.

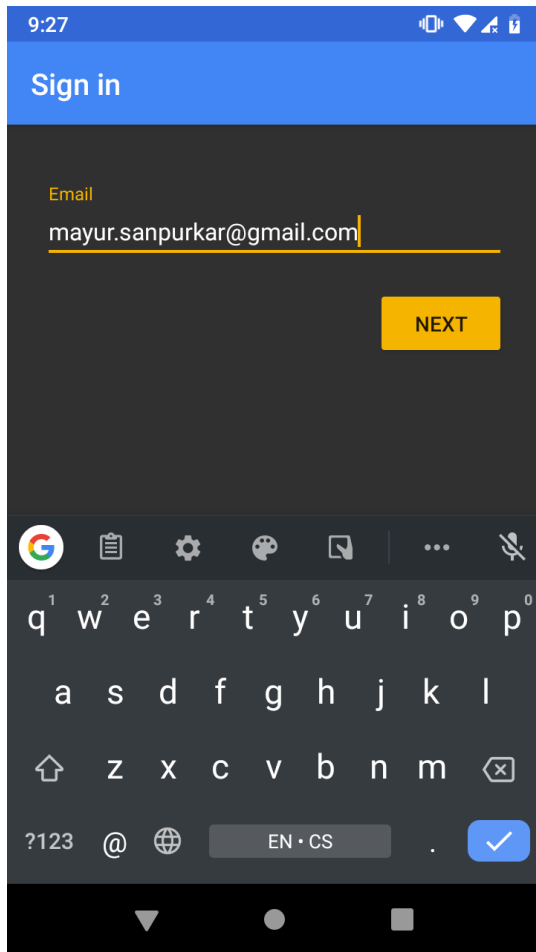


Figure 10 - Registered User's Email

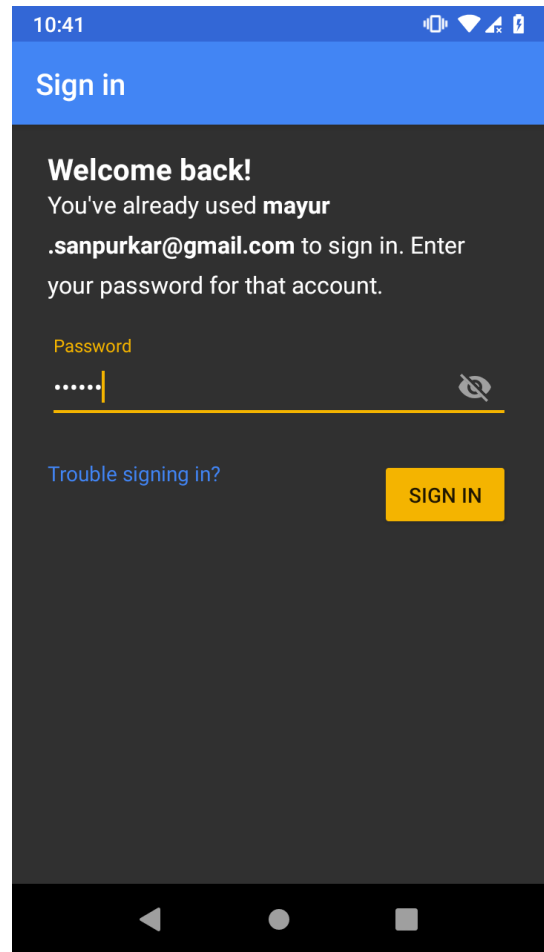


Figure 9 - Registered User's password

If a user is already registered then he will be presented with screen to enter his/her password along with message which shows that user is already registered in the database.

Since the beginning of the sign in process the application starts to interact with the Firebase and Firestore database to check if the user is new or already registered user. When the application is launched the code present in the **MainActivity.Java** starts to execute.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

Figure 11 - onCreate () Method

The code present in the onCreate () method above is responsible for presenting user with Graphical User Interface (GUI). The line **setContentView** method which takes layout file as an argument is responsible for presenting user with layout specified in the **activity\_main.xml** file.

```
</xml version="1.0" encoding="utf-8" ?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#E0E0E0">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:theme="@style/AppTheme"
        app:logo="@drawable/ic_restaurant_white_24px"
        app:popupTheme="@style/Theme.AppCompat.Light.DarkActionBar"
        app:title="Places to Eat"
        app:titleMarginStart="24dp"
        app:titleTextColor="@android:color/white" />

    <FrameLayout
        android:id="@+id/filterBarContainer"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/toolbar"
        android:background="@color/colorPrimary"
        android:paddingBottom="12dp"
        android:paddingLeft="12dp"
        android:paddingRight="12dp">
```

*Figure 12 - Activity\_main.xml*

Above is the typical structure of the **activity\_main.xml** file which contains the graphical elements such as **TextView, Button, Toolbar, ImageView, ProgressBar** etc. that can be placed together in different types of layouts such as **FrameLayout, LinearLayout, RelativeLayout** etc. Grouping all these graphical elements helps to create clear User Interface.

Also as we can see these graphical elements have different properties such as **id, layout\_width, layout\_height** which is to have width and height relative to the parent layout. Other properties such as **background, Padding, logo, title, titleTextColor** and many more. Changing element's properties through xml file is a lot easier than changing it in Java file. Android studio presents lots of properties for every graphical element.

The login process happens with the help of Firebase. The code present in the MainActivity executes so quickly and it establishes connection with Firebase and initializes Firestore database.

```

@Override
public void onStart() {
    super.onStart();

    // Start sign in if necessary
    if (shouldStartSignIn()) {
        startSignIn();
        return;
    }

    // Apply filters
    onFilter(mViewModel.getFilters());

    // Start listening for Firestore updates
    if (mAdapter != null) {
        mAdapter.startListening();
    }
}

```

*Figure 13 - onStart() Method*

One of the methods of Activity class are onStart() method which executes after onCreate() method. In onStart() method as name suggests user needs to specify what should happen or which method should be called. As we can see the code is to check with **IF statement** if user is already signed in or not. The shouldStartSignIn() method will be checked for its output whether it will be true or false and based on that it will be decided whether to call startSignIn() method present in the IF block or to skip and go to next line which will apply filters and start to listening to the Firestore database for any new updates.

```

private boolean shouldStartSignIn() {
    return (!mViewModel.getIsSigningIn() && FirebaseAuth.getInstance().getCurrentUser() == null);
}

private void startSignIn() {
    // Sign in with FirebaseUI
    Intent intent = AuthUI.getInstance().createSignInIntentBuilder()
        .setAvailableProviders(Collections.singletonList(
            new AuthUI.IdpConfig.EmailBuilder().build()))
        .setIsSmartLockEnabled(false)
        .build();

    startActivityForResult(intent, RC_SIGN_IN);
    mViewModel.setIsSigningIn(true);
}

```

*Figure 14 - startSignIn() Method*

When the IF block in above picture is executed to check the condition the program flow goes to the **shoudStartSignIn()** method. Here the method returns Boolean value which is either true or false. It check if value of **!mViewModel.getIsSigningIn()** is not equal to its original value and using Firebase authentication value of **getCurrentUser()** method is null. If both values are true then it executes code present in the IF block which is call to **startSignIn ()** method.

The **startSignIn ()** method is provided by **FirestoreUI** library which is built on the top of **Firestore Authentication SDK** (Software Development Kit). It is easiest way to implement authentication process for the new user using Firestore.

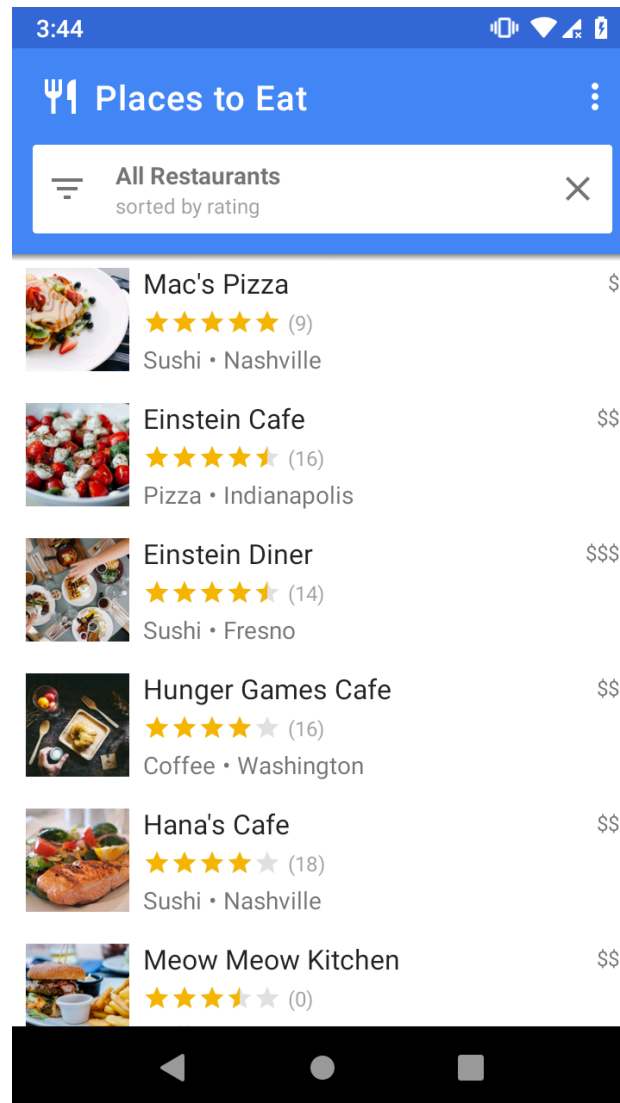


Figure 15 - Restaurants List

When user logs in then he/she is presented with the screen which contains many things. At the top is the name of the application which is “Places to Eat” along with image of knife and fork. On the right side there is a three dot icon or menu icon by clicking on which user will have some menu option to perform some tasks. Below the name the of the restaurant there is a search bar which allows user to sort or filter restaurants based on some criteria which will be explained later and below that is list of restaurants along with one of their menu picture, name of the restaurant, ratings given by the customers, popular menu at the restaurant, location or city where the restaurant is present and on the right side of the layout there is \$ sign to represent the price. One Dollar sign (\$) means restaurant is cheap, two Dollar signs (\$\$) mean restaurant is with average prices and three Dollar signs (\$\$\$) mean restaurant is with high prices or it is expensive.

The list of the restaurants is presented using RecyclerView. RecyclerView widget is part of Android’s User Interface elements. It is mainly used to display list of elements based on large data sets. It is more advanced and flexible version the ListView.

The data about restaurant is present in the Firestore databse. To fetch that data and populate the RecyclerView with it RestaurantAdapter class is used. In Android OS **Adapter** is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to an Adapter view then view can takes the data from the adapter view and shows the data on different views like as RecyclerView, ListView, GridView, Spinner etc. For more customization in Views we uses the base adapter or custom adapters.

```
// Start listening for Firestore updates
if (mAdapter != null) {
    mAdapter.startListening();
}
```

*Figure 16 - Firestore update*

The above code from the MainActivity file calls the starListening () of the FirestoreAdapter class which is extended by RecyclerViewAdapter class. So, RecyclerViewAdapter class is implementing methods from the FirestoreAdapter class. It is doing this so that it can fetch the data from the Firestore database and fill the elements present in the RecyclerView in activity\_main.xml.

The data which will be fetched from the Firestore database will be all information about the restaurant and if any data is updated. The data will be fetched without FirebaseUI so **addSnapshotListener** is used.

```
public void startListening() {
    if (mQuery != null && mRegistration == null){
        mRegistration = mQuery.addSnapshotListener(this);
    }
}
```

*Figure 17 - SnapshotListener*

In the above figure the by calling addSnapshotListener() method, SnapshotListener Starts listening to the document referenced by this DocumentReference with the given options using an Activity-scoped listener.

```
@Override
public void onEvent(@Nullable QuerySnapshot documentSnapshots, @Nullable FirebaseFirestoreException e) {

    // Handle Errors
    if (e != null){
        Log.w(TAG, msg: "onEvent:error",e);
        return;
    }

    // Dispatch the Event
    for (DocumentChange change : documentSnapshots.getDocumentChanges()){

        // Snapshot of the changed document
        DocumentSnapshot snapshot = change.getDocument();

        switch (change.getType()){

            case ADDED:
                onDocumentAdded(change);
                break;

            case MODIFIED:
                onDocumentModified(change);
                break;

            case REMOVED:
                onDocumentRemoved(change);
                break;

        }
    }
}
```

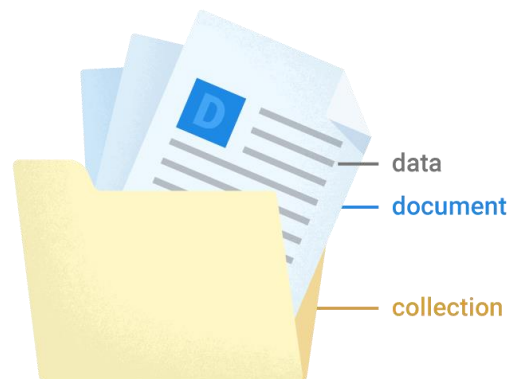
*Figure 18 - onEvent() Method*



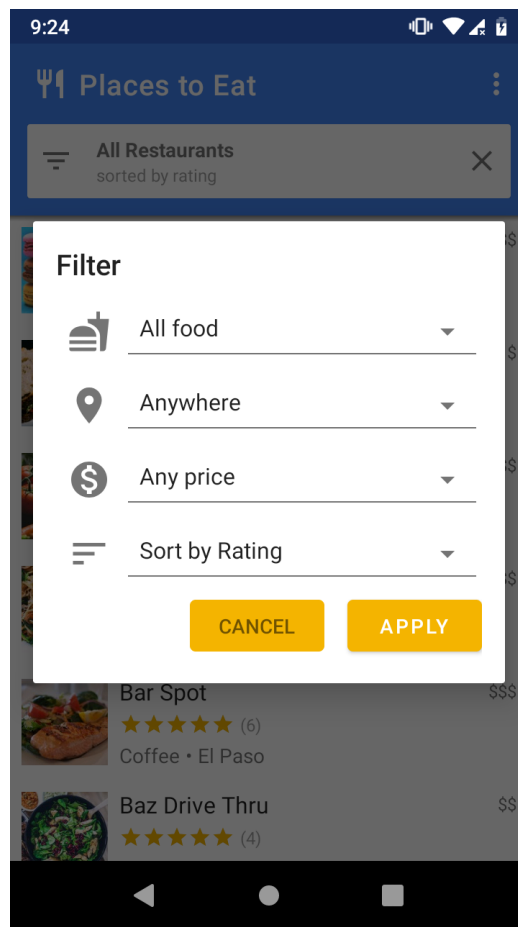
If there are some changes to the database such as adding new data, modifying existing data or removing some data then update to the database and notifying to the user about the changes in the database are implemented in `onEvent()` method. It takes the Snapshot of the changes in the documents then it updates the database and notifies the user about the changes in the documents. All these is done in real-time.

In the above code as we can see there is a term Document. In Firestore database the unit of storage is the document. A document is a lightweight record that contains fields, which map to values. Each document is identified by a name. Each document contains a set of key-value pairs. Cloud Firestore is optimized for storing large collections of small documents.

All documents must be stored in collections. Documents can contain sub collections and nested objects, both of which can include primitive fields like strings or complex objects like lists. Collections and documents are created implicitly in Cloud Firestore. Simply assign data to a document within a collection. If either the collection or document does not exist, Cloud Firestore creates it.



*Figure 19 - Document and Collection*



*Figure 20 - Filters*

When user clicks on the Search bar then a dialog appears with filters to choose restaurant based on user's choice from available filters. As we can see there are four filters available for the user.

- 1) All Food – Type of food
- 2) Anywhere - Location
- 3) Any Price – Prices (cheap, medium and expensive)
- 4) Sort by – Ratings, Price and Popularity

When users clicks on type of food he want he will have options like Italian, Mexican, Indian, burgers, sushi and many more and he/she can choose whichever he/she wants. After that there is option for location which means city. User can choose any city which is available in the database.

The price option specifies 3 types of Dollar sign (\$, \$\$, \$\$\$). Each sign represents how much could be the price range. Single Dollar sign means the restaurant is cheap. Two Dollar signs mean the restaurant

is in medium price range and three Dollar signs mean the restaurant is expensive or with high price range, so user can choose based on his willingness to spend money.

The last option is sort by in which user can sort restaurants based on Price to have restaurants with lower price and then gradually increasing with price. Another option to sort is by rating, in this restaurants are sorted based on their ratings given by customers and last option is based on popularity which displays restaurants which are with highest ratings and reviews by customers.

```
@Override
public void onFilter(Filters filters) {
    // Construct query basic query
    Query query = mFirestore.collection( collectionPath: "restaurants");

    // Category(equality filter)
    if(filters.hasCategory()){
        query = query.whereEqualTo( field: "category", filters.getCategory());
    }

    // City (equality filter)
    if(filters.hasCity()) {
        query = query.whereEqualTo( field: "city", filters.getCity());
    }

    // Price (equality filter)
    if(filters.hasPrice()) {
        query = query.whereEqualTo( field: "price", filters.getPrice());
    }
    // Sort by(OrderBy with direction)
    if (filters.hasSortBy()){
        query = query.orderBy(filters.getSortBy(),filters.getSortDirection());
    }

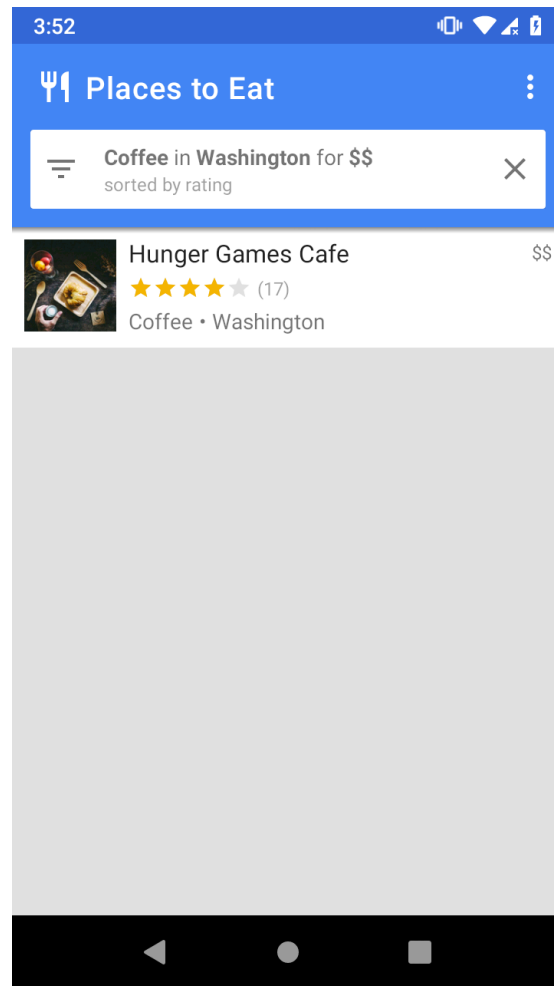
    // Limit items
    query = query.limit(LIMIT);

    // Update the query
    mQuery = query;
    mAdapter.setQuery(query);
}
```

*Figure 21 - onFilter() method*

As we can see in the above picture the onFilter() method does the job of filtering the restaurants based on user's preferences. In the first line the basic query is created which will be fired on Collection "restaurants". It is collection of all restaurants where each restaurant details are stored as document.

Further code in the method checks if the specified filter is available or not. If it is present then it fires the query to fetch the data. After that it updates the database and update the RecyclerView with the data read after firing the query. The query.LIMIT option reads the data from the database upto the records set by the LIMIT. After execution of filters set by user, he/she is presented with the updated result based on his preferences.



*Figure 22 - Filter Results*

As we can see the user searched for coffee cafe in Washington for average price range which are sorted by rating. The user got one restaurant named Hunger Games Cafe in search results.

If user clicks on one of the restaurants from the list then he will be presented with a screen which contain details reviews and ratings about the restaurant.

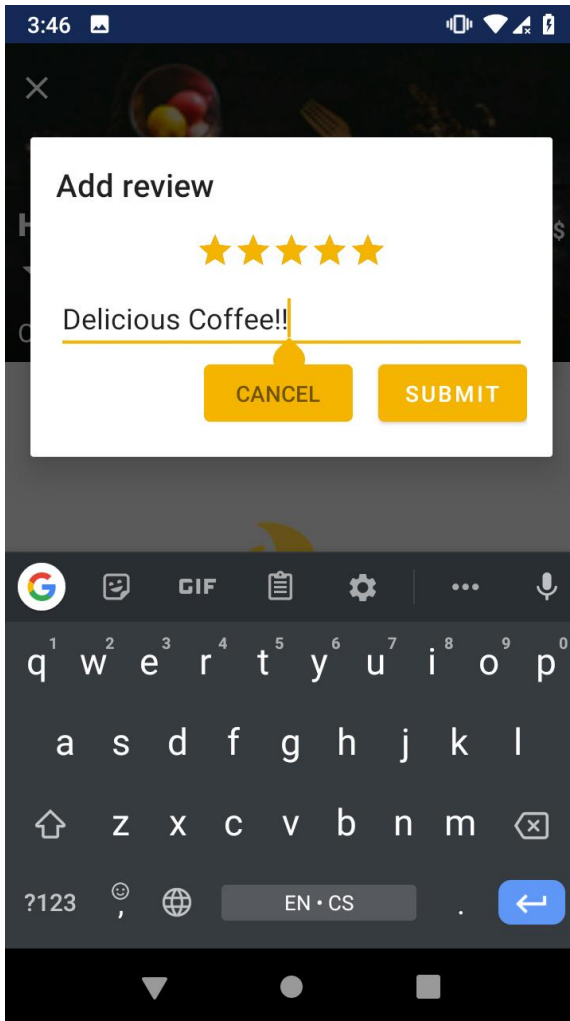


Figure 24 - Ratings and Review

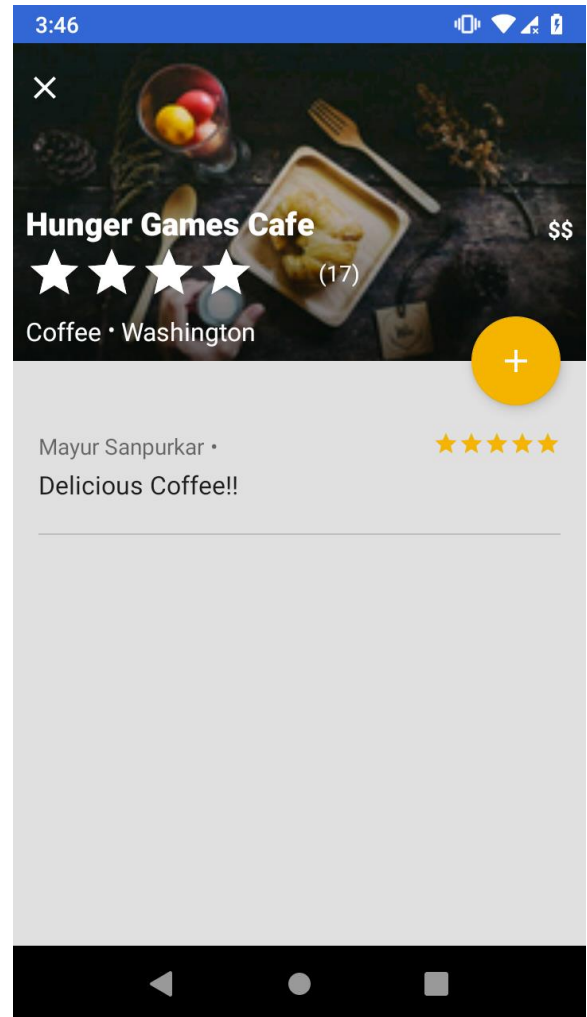


Figure 23 - Ratings and Review 2

As we can see in the Ratings figure, there is Ratings and reviews given by other users. A user can add give ratings and write review about his/her experience at the restaurant by clicking on the add button (+). The rating for the restaurant is average rating based on the ratings given by all the customers.

## 4.2.2 Services

In this app the services are provided by Firebase which interacts with the cloud storage, so let's get introduced to cloud. In simple term cloud means software or services delivered over the Internet. The first sense of cloud services covers a wide range of resources that a service provider delivers to customers via the internet, which, in this context, has broadly become known as the cloud. Characteristics of cloud services include self-provisioning and elasticity; that is, customers can provision services on an on-demand basis and shut them down when no longer necessary.

In addition, customers typically subscribe to cloud services, under a monthly billing arrangement, for example, rather than pay for software licenses and supporting server and network infrastructure upfront. In many transactions, this approach makes a cloud-based technology an operational expense, rather than a capital expense.

From a management standpoint, cloud-based technology lets organizations access software, storage, compute and other IT infrastructure elements without the burden of maintaining and upgrading them. The usage of cloud services has become closely associated with common cloud offerings, such as software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS).

The Services tier provides different kinds of services to the user based on the requirements. It acts as the middleware between user and the database. It processes user's requests and reads and writes data to the database. The Firebase is Backend as a service (BaaS). In this there is no need to have an application server which would act as business logic and handle user's requests. As the application consist of business logic and services is the main layer here, so basically this application is a 2-tier architecture.

Every operation that user has to perform is associated with the cloud services and database. Firebase provides many kinds of services as follows

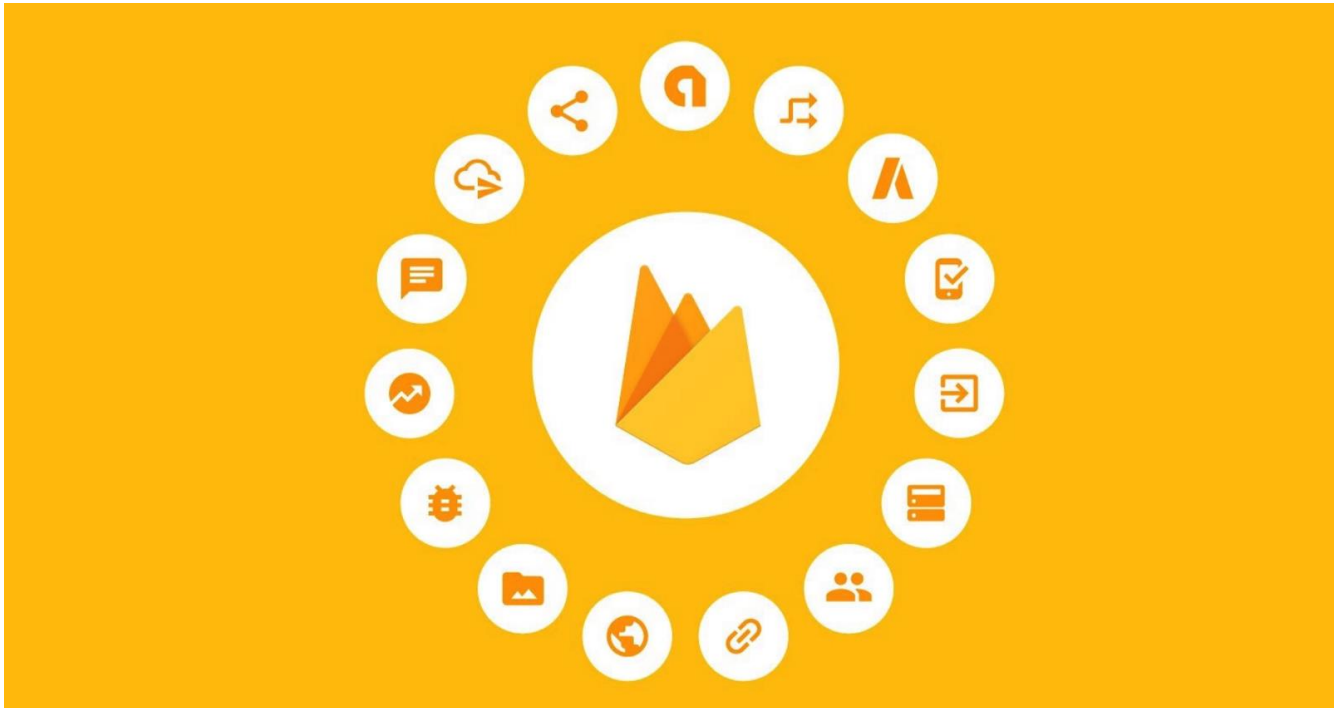


Figure 25 - Firebase Services

As we can see in the picture, Firebase provides many kinds of services such as:

<b>Build better apps</b>	<b>Improve app quality</b>	<b>Grow your business</b>
Cloud Firestore	Crashlytics	In-App Messaging
ML Kit	Performance Monitoring	Google Analytics
Cloud Functions	Test Lab	Predictions
Authentication	App Distribution	A/B Testing
Hosting		Cloud Messaging
Cloud Storage		Remote Config
Realtime Database		Dynamic Links

Table 1 - Firebase Services

These services are grouped according to their functionality. So Build better apps services are mainly to help developers while developing the app. Improve app quality services are as name suggests to improve the quality of the app and Grow your business services are for analysis and marketing and predicting future behavior of the app based on the current data. While developing this application we used Cloud Firestore, Authentication, Real time Database and cloud storage and google analytics to analyze the data so that quality of the app can be improved based on the gathered data.

To interact with the cloud Firestore we extend the FirestoreAdapter class through RestaurantAdapter class so that we can fetch the data from the Firestore Database. In fact FirestoreAdapter class is by every other class which interacts with the database in some way such as ratings and reviews.

The Firebase allows to set up and operate many things from its console. When user clicks on Create New Project button then process begins to create new project starting with the name first as shown below.

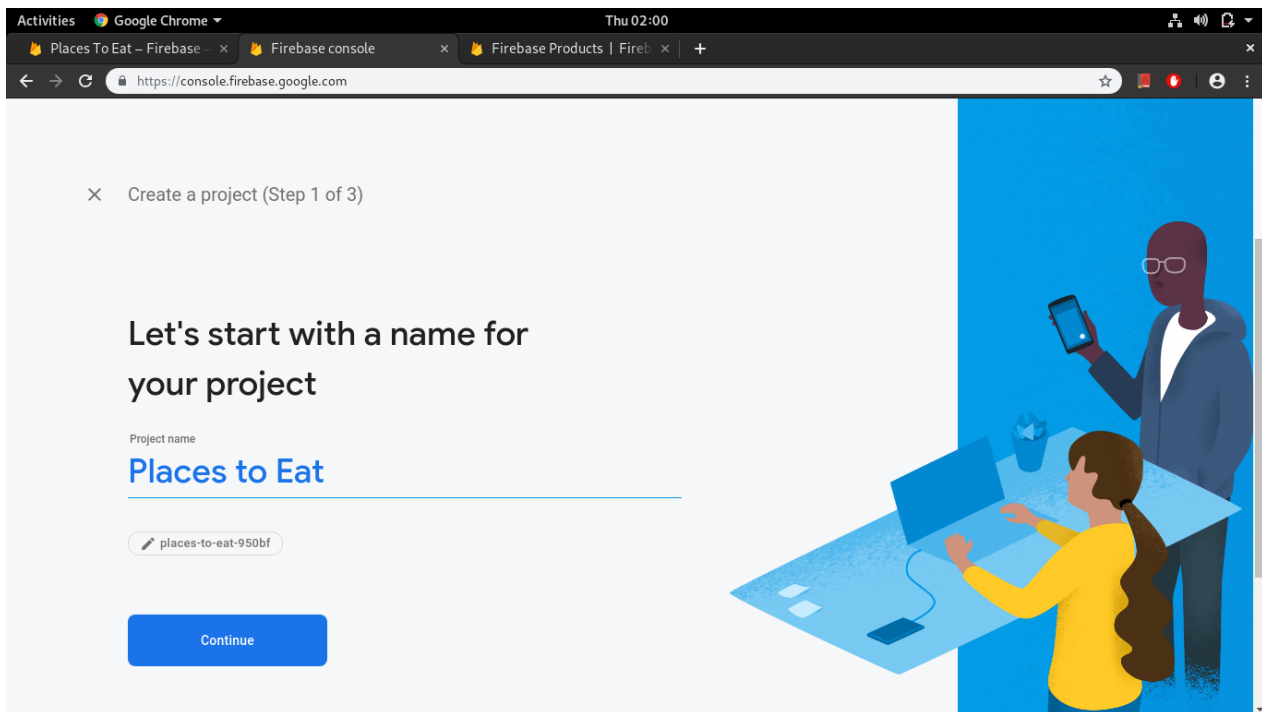


Figure 26 - Firebase New Project



After this user will be asked if he/she would like to enable Google Analytics for his project as it would help in many things such as analysis, reporting, targeting specific users etc.

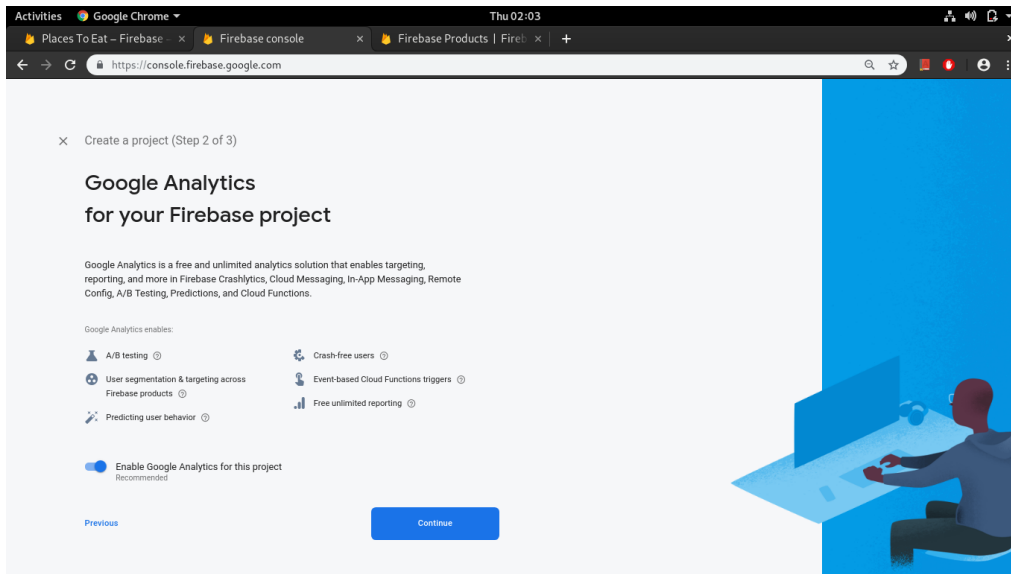


Figure 27 - Firebase Google Analytics

After that user has to select account for Google Analytics, which is final step for creating the project in Firebase.

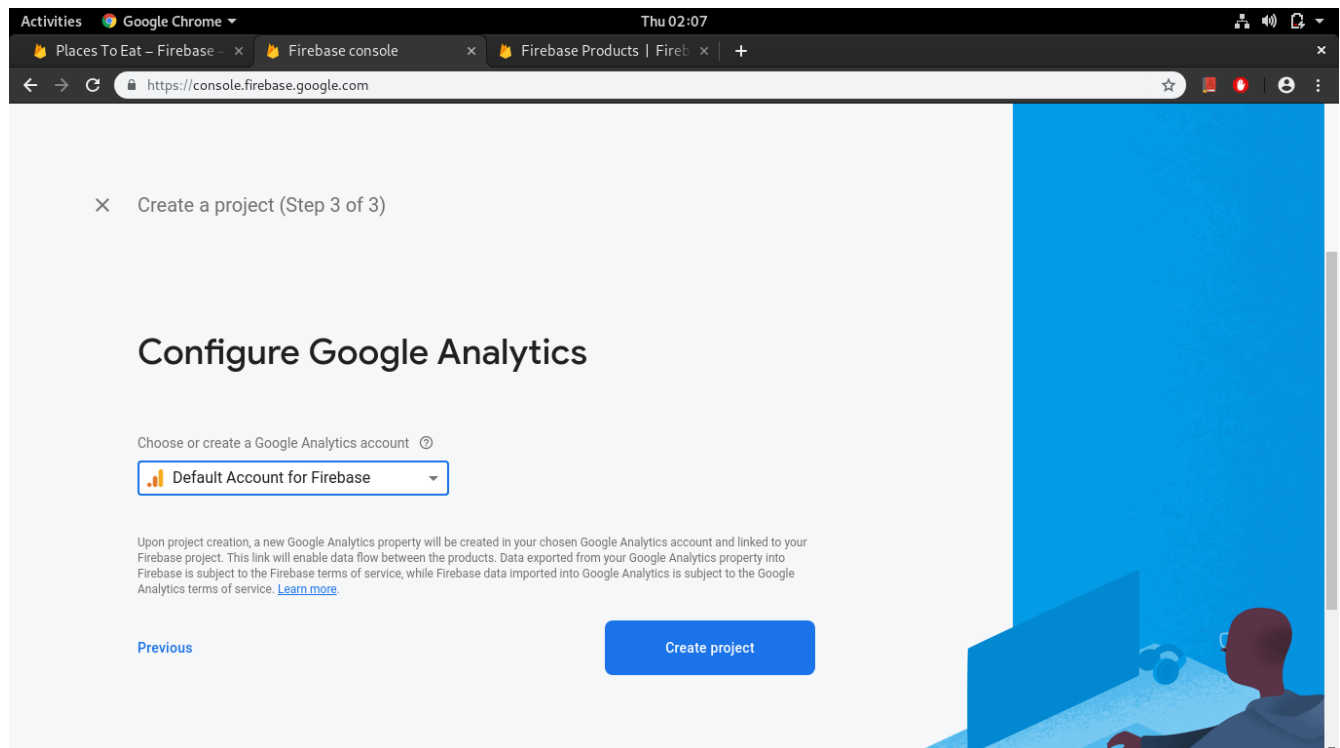
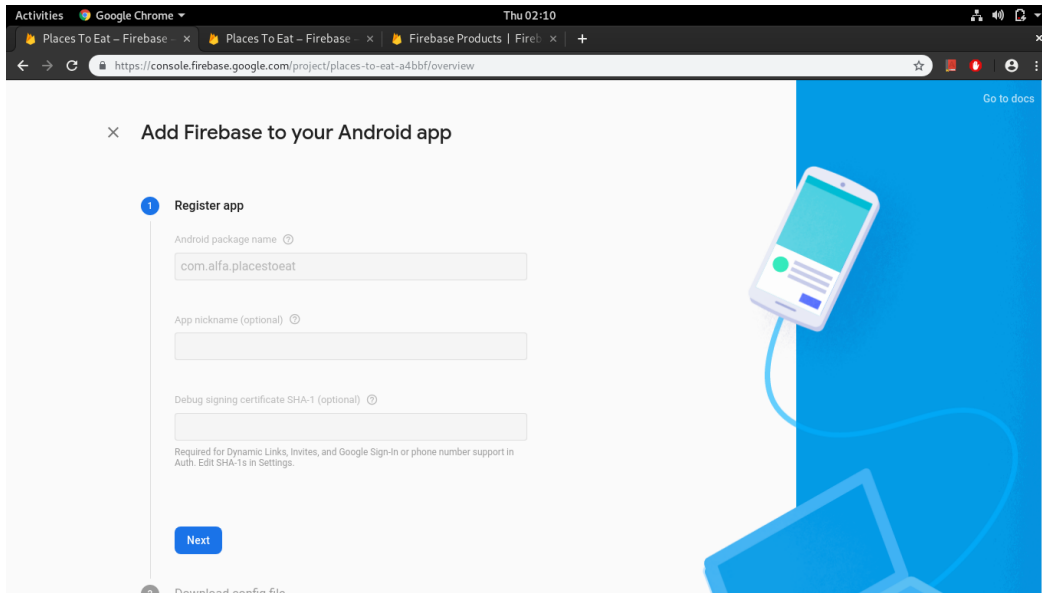


Figure 28 - Firebase Select Account

After this project will be created and then user will be redirected to set up the SDK (Software Development Kit).



*Figure 29 - Firebase SDK Setup*

Here user has to register the app and for that he/she has to enter the package name for the app such as for example **com.example.appname**. This format for the package name is usually preferred where com.example is usually company name and then followed by app name. This package name should be similar to the package name for the project in Android Studio otherwise there will be error while syncing the data.

After this to use the google-services.json file user has to modify the Build.gradle file in the project directory and in the app directory also

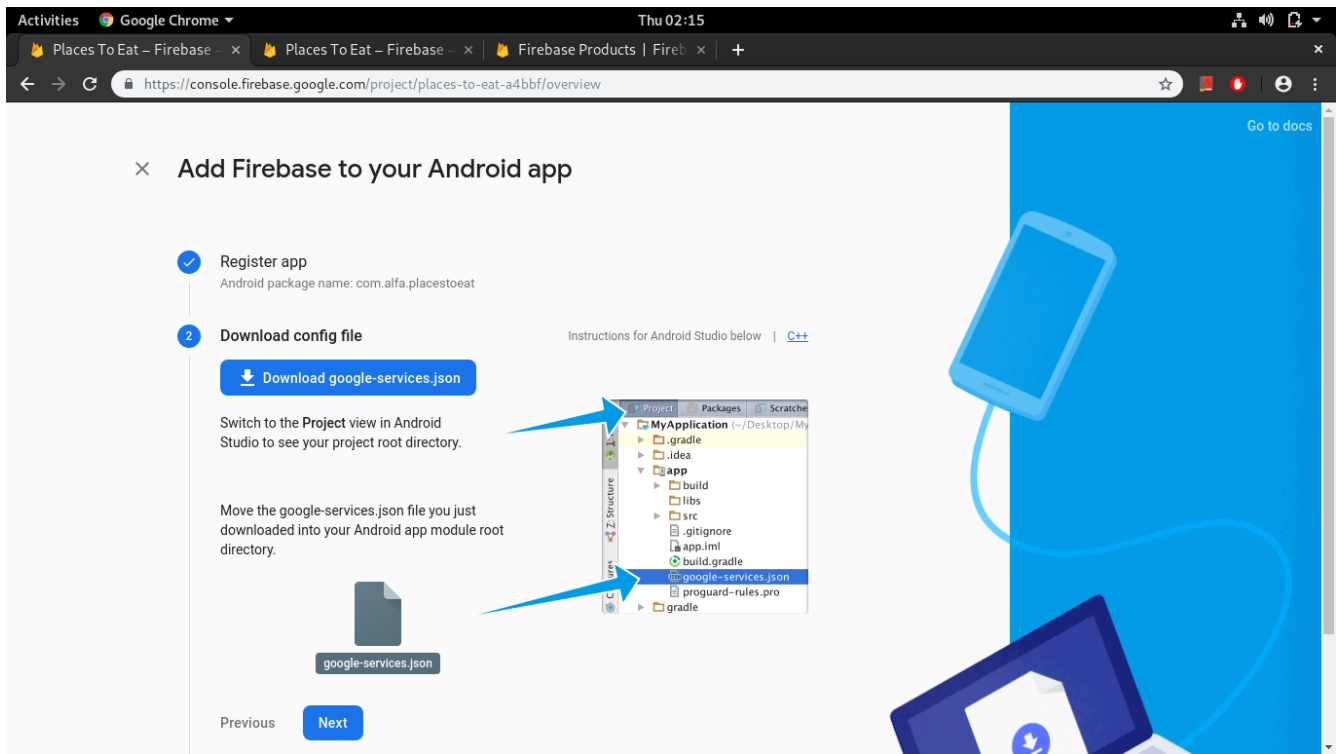


Figure 31 - Firebase JSON download

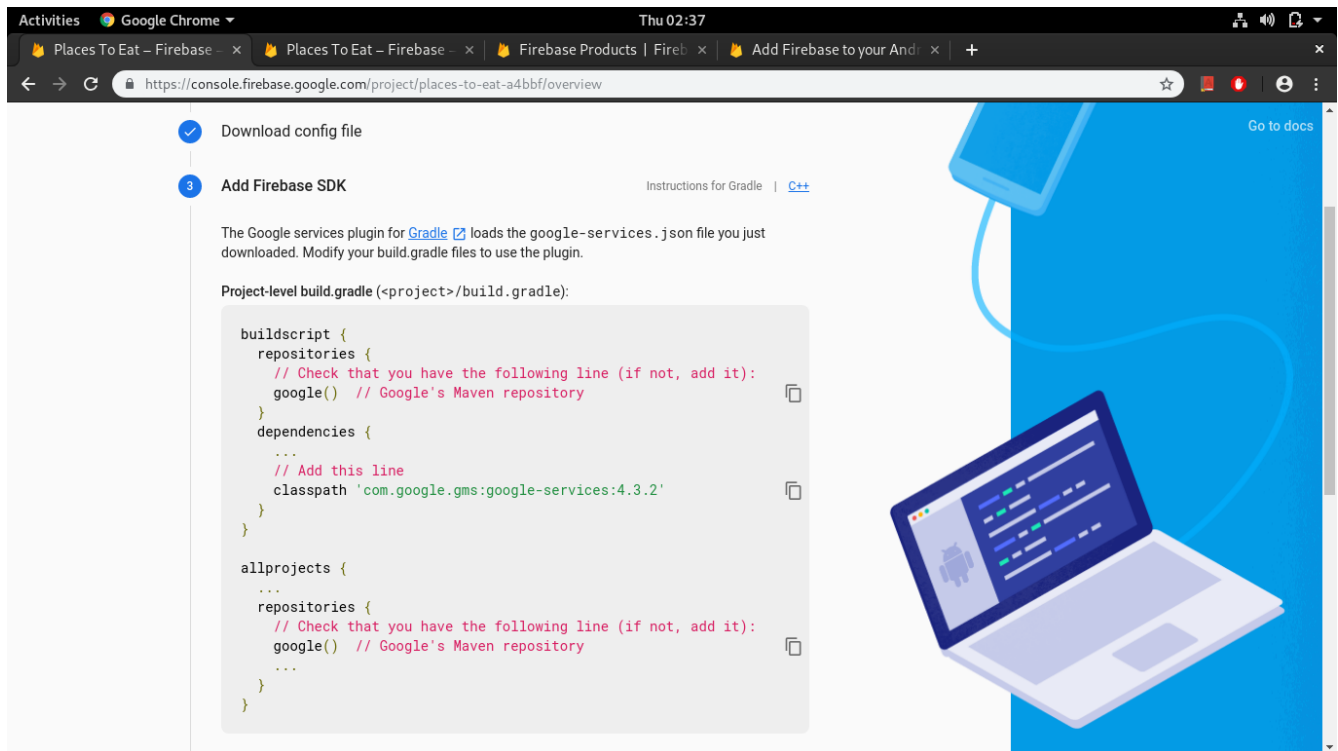


Figure 30 - Build.gradle 1

App-level build.gradle (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'

dependencies {
    // add the Firebase SDK for Google Analytics
    implementation 'com.google.firebase:firebase-analytics:17.2.0'
    // add SDKs for any other desired Firebase products
    // https://firebase.google.com/docs/android/setup#available-libraries
}
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

Finally, press "Sync now" in the bar that appears in the IDE:

Gradle files have changed since last sync. [Sync now](#)

Figure 32 - Build.gradle 2

The final step is to run the app to verify the Firebase SDK (Software Development Kit) installation in the Android Studio.

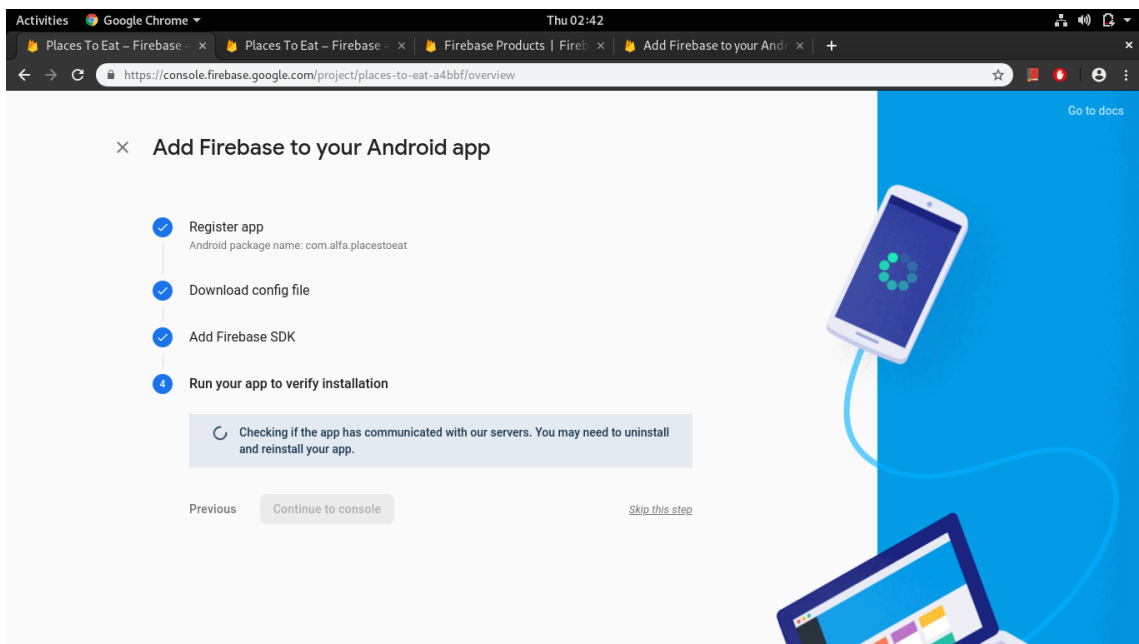


Figure 33 - Firebase SDK verification

Once the installation process is complete user will be directed to the console page or Dashboard which presents overview of the application.

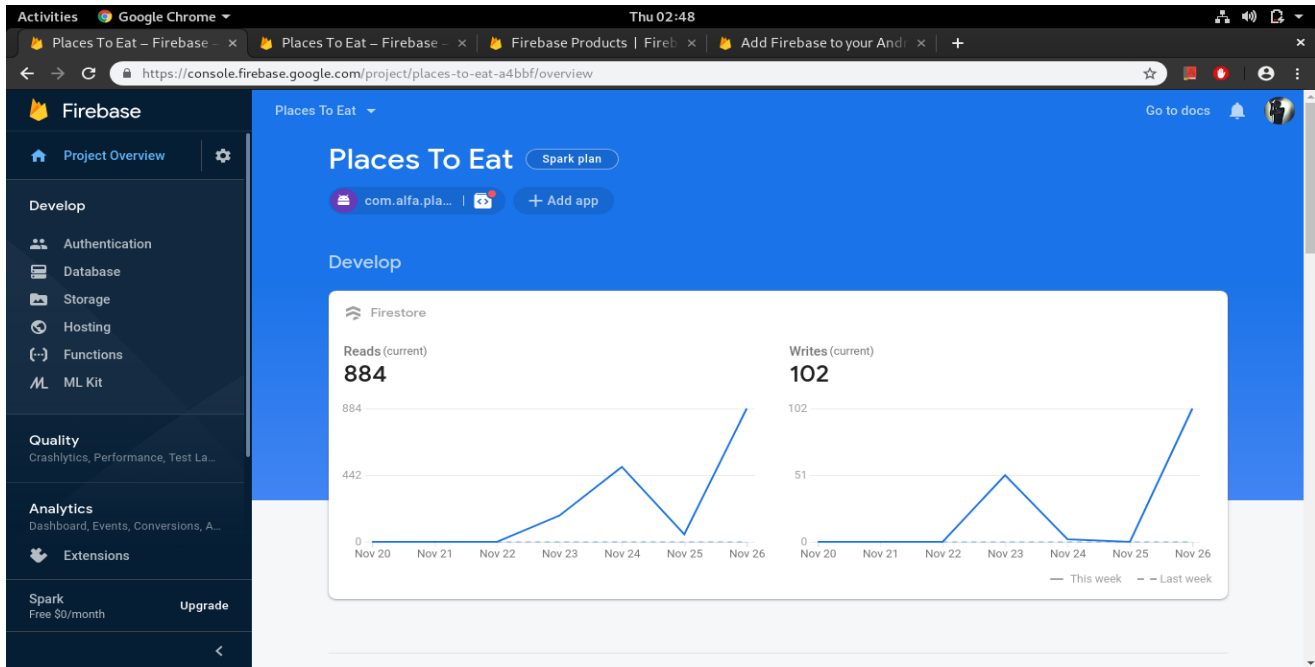


Figure 34 - Firebase Console

Here we can see Google Analytics service showing how many times the data was read and how many times there was write operation to the database. On the left side of the screen is a panel where we have links to other services which are used or which can be used in the project such as Authentication, Database, Storage, Hosting, Functions and ML Kit (Machine Learning Kit).

Below that are services related to the quality and analytics and some other extensions. In this app we have user Authentication and Database and Google Analytics which we can see on the Dashboard.

The Authentication service shows the information about the authenticated users or users which are registered. It also allows to add new user from the console. There are other tabs such as sign-in method, templates and usage. The sign-in method provides different kinds of providers for signing in such as email and password, phone, Google, Facebook, Twitter, GitHub, Yahoo, Microsoft and Apple etc.

The Templates tabs provides different kinds of templates which can be used for example when user signs in using email address then a template can be used to send user a confirmation email to verify the registered email address. Another one is when user forgets the password then a template can be used to help the user set new password. Others are for change of email address, verification through SMS and SMTP settings.

# Authentication

Users | Sign-in method | Templates | Usage

Search by email address, phone number, or user UID Add user Refresh More

Identifier	Providers	Created	Signed In	User UID ↑
hanahaskovic@gmail.com		Nov 26, 2019	Nov 26, 2019	1Xg8C1rFXCNMes6OnSwneC4fOL...
mayur.sanpurkar@gmail.com		Nov 26, 2019	Nov 27, 2019	z0JMMgNXghZcUVXIZr500Vald0...

Rows per page: 50 | 1-2 of 2

Figure 35 - Authentication

### Sign-in providers

Provider	Status
Email/Password	Enabled
Phone	Disabled
Google	Disabled
Play Games	Disabled
Game Center <span>Beta</span>	Disabled
Facebook	Disabled
Twitter	Disabled
GitHub	Disabled
Yahoo	Disabled
Microsoft	Disabled
Apple <span>Beta</span>	Disabled

Figure 36 - Sign in Providers

## Templates

### Email



#### Email address verification



#### Password reset



#### Email address change




#### SMTP settings

### SMS



#### SMS verification

### Email address verification

When a user signs up using an email address and password, you can send them a confirmation email to verify their registered email address. [Learn more](#) 

Sender name

not provided

From

noreply@places-to-eat-a4bbf.firebaseio.com 

Reply to

not provided

Subject

Verify your email for project-1079864651955

Message

Hello %DISPLAY\_NAME%,

Follow this link to verify your email address.

[https://places-to-eat-a4bbf.firebaseio.com/\\_\\_/auth/action?mode=<action>&oobCode=<code>](https://places-to-eat-a4bbf.firebaseio.com/__/auth/action?mode=<action>&oobCode=<code>)

If you didn't ask to verify this address, you can ignore this email.

Thanks,

*Figure 37 - Email Template*

### 4.2.3 Cloud Database

Cloud storage involves storing data on hardware in a remote physical location, which can be accessed from any device via the internet. Clients send files to a data server maintained by a cloud provider instead of (or as well as) storing it on their own hard drives. Google Cloud Storage, which lets users store and share files, is a good example. Cloud storage systems generally encompass hundreds of data servers linked together by a master control server, but the simplest system might involve just one.

The cloud database Firestore is a fast, fully managed, serverless cloud-native NoSQL document database that simplifies storing, syncing, and querying data for mobile, web, and IoT apps at global scale. Cloud Firestore is a flexible, scalable, realtime database. It's simple enough for rapid prototyping yet scalable and flexible enough to grow with user to any size.

Cloud Firestore is a realtime database, meaning that clients can listen to data in Cloud Firestore and be notified in real time as it changes. This feature lets user build responsive apps that work regardless of network latency or Internet connectivity.

Cloud Firestore is a cloud-hosted NoSQL database. User stores data in *documents*, which contain fields mapping to values. These documents are stored in *collections*, which are simply containers that help user organize and query his/her documents.

It takes care of all the functionality that a server provides so user don't need to worry about it and he/she can focus on the app development and improvements. Its client libraries provide live synchronization and offline support, while its security features and integration with Firebase and Google Cloud Platform (GCP) accelerate building truly serverless apps.



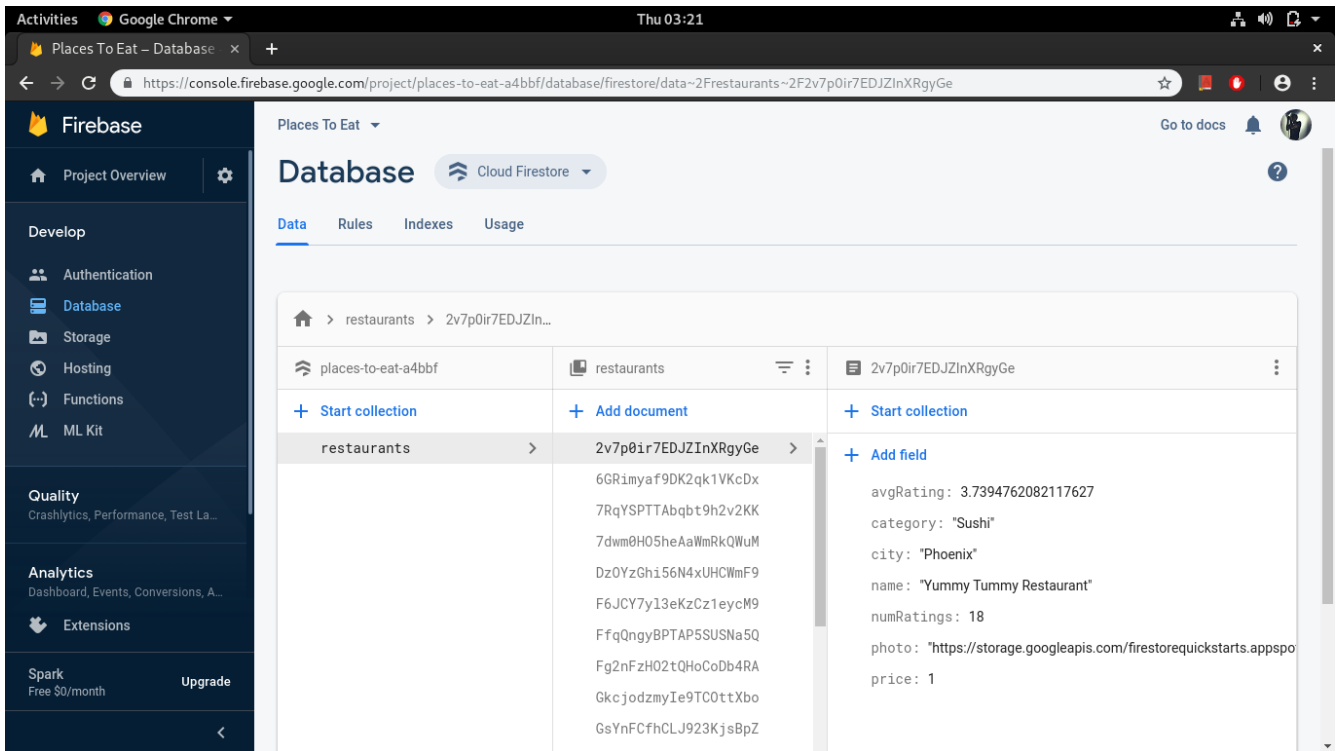


Figure 38 - Database

As we can, here we have a collection called ‘restaurants’ and it contains multiple documents. Each document represents a restaurant and on the right side we can see the data stored in each collection, which is in key-value format. User can create new collection from here or from the code also.

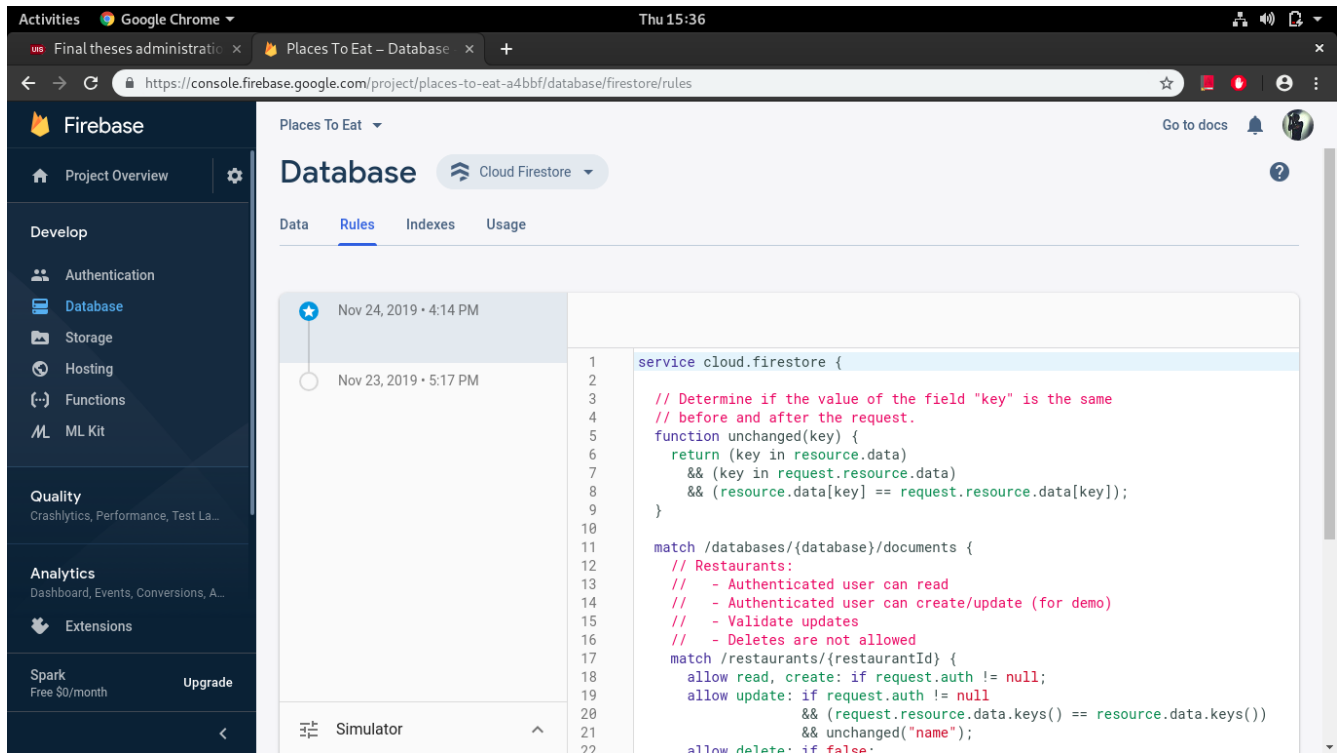
```
private void onAddItemsClicked() {
    // Get a reference to the restaurants collection
    CollectionReference restaurants = mFirestore.collection("restaurants");

    for (int i = 0; i < 10; i++) {
        // Get random restaurants POJO
        Restaurant restaurant = RestaurantUtil.getRandom(context, this);

        // Add a new document to the restaurants collection
        restaurants.add(restaurant);
    }
}
```

Figure 39 - Restaurants collection

Above code gets reference to the restaurants collection and below that is a for loop which adds documents to that collection in a random manner.



*Figure 40 - Rules*

The Database also contains rules tabs where user can enter rules which are useful for verification of the data, user such as if user is registered or not and what kind of access level user has for example read only or read and write to the database. Other things such as only registered users are allowed to rate and write a review about the restaurant.

## **5 Result and Discussion**

The result of the development of the application is as expected: a working application, which suggests user restaurants. Restaurants are pulled from the cloud database, based on users' parameters or search criteria. During the functional testing of the application with several users, it was observed that the application performed well and it showed user data from the database and also real time updates to the view, if there were any updates.

The decision to use cloud-based database was made because also other Firebase services are useful and integrating them into the application is easy and convenient. Also, all data is secured in the cloud database. The connection with the Firebase is using WebSockets and it is faster than usual HTTP connection. There are many other services which Firebase provides and they can be used in the app for the future improvements of the app.

The app user interface is very simple and any new user can use it without the help of any introductory animation or guides. The User Interface elements are developed using Google's Material design.

The application will be uploaded to the Google Play Store which is Google's official store for the apps.

### **5.1 Google Play Store**

Google Play, formerly Android Market, is a digital distribution service operated and developed by Google. It serves as the official app store for the Android operating system, allowing users to browse and download applications developed with the Android software development kit (SDK) and published through Google. Google Play also serves as a digital media store, offering music, books, movies, and television programs. It previously offered Google hardware devices for purchase until the introduction of a separate online hardware retailer, Google Store, on March 11, 2015, and it also offered news publications and magazines before the revamp of Google News on May 15, 2018.

Applications are available through Google Play either free of charge or at a cost. They can be downloaded directly on an Android device through the Play Store mobile app or by deploying the application to a device from the Google Play website. Applications exploiting hardware capabilities of a device can be targeted to users of devices with specific hardware components, such as a motion sensor (for motion-dependent games) or a front-facing camera (for online video calling). The Google Play store had over 82 billion app downloads in 2016 and has reached over 3.5 million apps published in 2017. It has been the subject of multiple issues concerning security, in which malicious software has been approved and uploaded to the store and downloaded by users, with varying degrees of severity.

Google Play was launched on March 6, 2012, bringing together Android Market, Google Music, and the Google eBookstore under one brand, marking a shift in Google's digital distribution strategy. The services included in Google Play are Google Play Books, Google Play Games, Google Play Movies & TV and Google Play Music. Following their re-branding, Google has gradually expanded the geographical support for each of the services. As of September 2019 there are 2.8 million apps in the Google Play Store. **(En.wikipedia.org, 2019)**

## **6 Conclusion**

The main goal of this thesis was to create an application based on the Android operating system using cloud-based database Firestore, which is part of Google's Backend as a Service (BaaS) Firebase. The application will suggest user restaurants from the Firebase database. Users can search restaurant with specific conditions such as price, location, cuisine, ratings, reviews and popularity of the restaurant. Users can write reviews and rate the restaurants, which will be helpful to other users.

Keeping authenticity of the reviews in the mind, app will allow only registered users to search for restaurants and post reviews and ratings for restaurants. This will help them to see how many registered users there are. Internet connectivity is a must for the app to provide its functionality to the users.

The app was designed and developed in Android Studio. Before the development of the application, market analysis was done to see how people choose restaurants. Based on this, requirements were gathered and architecture was designed. After that, the application was implemented in Android Studio using Java programming language and Firebase cloud-based Backend as a Service, along with its different services such as authentication, real-time database, cloud storage and Google analytics.

### **6.1 Future Enhancements**

The application has many opportunities for the enhancements and build-ups. First enhancement could be to use other services provided by Firebase for improvements of the app and user experience, such as predictions, Machine Learning Kit and cloud messaging - user can send message to their friends and families about the restaurants or meeting point.

Second possibility is to add maps and real time location of the user, so that he/she can share it with other users in order to let them know his/her current location.

So far I came up with these two enhancements in the application. There could be many more and I would try to implement them gradually. The app will be released and distributed through Google Play Store for users to download and it will be for free.

## 7 REFERENCES

**Griffiths, D. and Griffiths, D. (2017).***Head first Android development*. 2nd ed. Sebastopol, CA: O'Reilly Media.

**Murphy, M. (2011).** *Beginning Android 3*. New York, NY. Apress.

**Cheng, A. (2018).***Millennials Are Ordering More Food Delivery, But Are They Killing The Kitchen, Too?*. [online] Forbes.com. Available at:

<https://www.forbes.com/sites/andriacheng/2018/06/26/millennials-are-ordering-food-for-delivery-more-but-are-they-killing-the-kitchen-too/#2bcbfab3393e>

**En.wikipedia.org. (2019).***Google Play*. [online] Available at:

[https://en.wikipedia.org/wiki/Google\\_Play](https://en.wikipedia.org/wiki/Google_Play)

**En.wikipedia.org. (2019).***Smartphone*. [online] Available at:

<https://en.wikipedia.org/wiki/Smartphone#History>

**Kumail, I. (2016).***Android Development System Requirements*. [online] AndroidClarified. Available at:

<https://androidclarified.com/android-development-system-requirements/>

**Esplin, C. (2016).**What is Firebase?. [online] Medium. Available at: <https://howtofirebase.com/what-is-firebase-fcb8614ba442>

**Medium. (2016).**Firebase Developers – Medium. [online] Available at: <https://medium.com/firebase-developers>

**Google Cloud. (2017).**Cloud Firestore | Google Cloud. [online] Available at:

<https://cloud.google.com/firestore/>

**Firestore. (2017).**Cloud Firestore Data model | Firestore. [online] Available at:

<https://firebase.google.com/docs/firestore/data-model>

**Firestore. (2017).**Firestore Products | Firestore. [online] Available at:

<https://firebase.google.com/products>

**Bishop, T. (2012).**‘Google Play’ replaces Android Market, consolidates Google’s media marketplaces.

[online] GeekWire. Available at: <https://www.geekwire.com/2012/google-play-replaces-android-market-consolidates-googles-media-marketplaces/>

**Sites.google.com. (2013).***Android - BB102 Jan2013Miracle*. [online] Available at:

<https://sites.google.com/site/morehandsomethenyou/chapter-5/android>

**Jackson, W. (2011).** *Android Apps for Absolute Beginners*. New York, NY. Apress.

- Felker, D. (2011).** *Android Application Development For Dummies*. Hoboken, NJ. Wiley Publishing, Inc.
- Ableson, W.F., Sen, R. and King, C. (2011).** *Android in Action*. Stamford, CT. Manning Publications Co.
- Phillips, B., Marsicano, K. and Stewart, C. (2018).** *Android programming*. 3rd ed. Atlanta: Big Nerd Ranch.
- Eddy, N. (2011).** *Slide 1: Low-Cost Cloud Services: 2015*. [online] eWEEK. Available at: <https://www.eweek.com/cloud/slide-1-low-cost-cloud-services-2015>
- King, B. (2016).** *Is Android Really Open Source? And Does It Even Matter?*. [online] MakeUseOf. Available at: <https://www.makeuseof.com/tag/android-really-open-source-matter/>
- Android. (2018).** *Android – Google Mobile Services*. [online] Available at: <https://www.android.com/gms/>
- Clement, J. (2019).** *Google Play Store: number of apps 2019 | Statista*. [online] Statista. Available at: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>
- En.wikipedia.org. (2019).** *Android (operating system)*. [online] Available at: [https://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](https://en.wikipedia.org/wiki/Android_(operating_system))
- Moroney, L. (2017).** *The definitive guide to firebase*. 1st ed. Seattle, Washington: Apress.
- Android Developers. (2010).** *Platform Architecture | Android Developers*. [online] Available at: <https://developer.android.com/guide/platform>
- Rouse, M. (2019).** *What is Google Firebase? - Definition from WhatIs.com*. [online] SearchMobileComputing. Available at: <https://searchmobilecomputing.techtarget.com/definition/Google-Firebase>
- S, R. (2017).** *Introduction to Firebase*. [online] Hackernoon.com. Available at: <https://hackernoon.com/introduction-to-firebase-218a23186cd7>