

INTERNSHIP REPORT

FEB 2021 – JUNE 2021

Internship report submitted in partial fulfilment of the requirement for
the degree of Bachelor of Technology

In

COMPUTER SCIENCE ENGINEERING

By:

Divij Gupta (171268)

To



Department of Computer Science & Engineering and Information
Technology

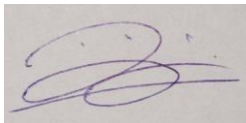
**Jaypee University of Information Technology Wahnaghat, Solan-
173234, Himachal Pradesh**

TABLE OF CONTENTS

Chapter No.	Topics	Page No.
	Student Declaration	2
	Certificate from the Supervisor	3
	Acknowledgement	4
	Summary	5
	List of Figures	6
	Glossary of terms used	7
Chapter 1	Company's Profile	8-9
Chapter 2	Introduction to the Project	10
Chapter 3	Implementation details	11-38
Chapter 4	Results & Conclusion	39
	References	40

DECLARATION

I hereby declare that this submission is my own work carried out at **Paymentus Corporation, Mohali** from **Feb, 2021** to **June, 2021** and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma from a university or other institute of higher learning, except where due acknowledgment has been made in the text.



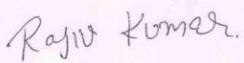
Signature

Name: **Divij Gupta**

Date: **22-05-2021**

CERTIFICATE

This is to certify that **Mr. Divij Gupta** of Jaypee University of Information Technology carried out the internship under my supervision at **Paymentus Corporation** from **Feb, 2021** to **June, 2021**. His efforts in the development of this internship were satisfactory.

A handwritten signature in black ink on a light purple background, reading "Rajiv Kumar".

Rajiv Kumar

Team Lead (Android)

Paymentus Corporation

Date: 22 May, 2021

ACKNOWLEDGEMENT

I take this opportunity to express my sincere thanks and deep gratitude to all those people who extended their wholehearted cooperation and have helped me in completing this internship successfully.

First of all, I would like to thank **Mr. Rajiv Kumar**, who mentored me, guided me and challenged me.

I also thank my family and friends who greatly supported me during the course of the internship.

Last but not the least, I would like to thank our founders for considering me a part of the organization and provide such a great Platform to learn and enhance my skills.

A very special thanks goes to all the faculties of Jaypee University of Information Technology under whom guidance I have been able to excel in my career and become a part of the Paymentus family.

Divij Gupta

171268

Jaypee University of Information Technology

SUMMARY

This report is all about what I learned as an intern and the work I carried out in Paymentus Corporation, Mohali during my internship period from Feb, 2021 to June, 2021.

Paymentus is the industry's fastest growing and most complete billing and payment network — powering the next generation of electronic bill payments.

In 2004, Paymentus was born from a desire to improve the way bills get paid. Vision, innovation and exemplary service have propelled Paymentus to become the leading paperless electronic billing and payment solution on the market, resulting in 1,300 clients including some of the largest billers in North America.

Working here has taught me that a project is not only a piece of code, it is a compilation of uncountable number of modules and a process behind building these modules. Writing code is just a small fraction of making an application. Planning, assigning, reviewing, fixing, testing, compiling and tracking all this process are some other fractions of developing an application.

During this internship, I was trained on various modern and best practices used in Android Development using Kotlin. I got hands on experience on Material Design, various Android components, MVVM architecture, Kotlin Coroutines, Kotlin Flows, etc. Using all the acquired knowledge, I was able to create a **Card Scanner** using ML kit and Camera X which is used to extract card details instantly.

Divij Gupta

May 22, 2021

LIST OF FIGURES

1.	<i>Material Components</i>	13
2.	<i>Permissions in Android</i>	15
3.	<i>Locations in Android</i>	16
4.	<i>MVC Architecture</i>	18
5.	<i>MVP Architecture</i>	19
6.	<i>MVVM Architecture</i>	20

ABBREVIATIONS

1. **MVVM** – Model View ViewModel
2. **MVC** – Model View Controller
3. **MVP** – Model View Presenter
4. **VCS** – Version Control System
5. **KTX** – Kotlin Extensions
6. **ML** – Machine Learning

Chapter - 1

COMPANY'S PROFILE

1.1. Summary

Website:

<https://www.paymentus.com/>

Facebook Page:

<https://www.facebook.com/Paymentus>

Linkedin Page:

<https://www.linkedin.com/company/paymentus/mycompany/>

Twitter Page:

<https://twitter.com/PaymentusCorp/>

1.2. About Us

In 2004, Paymentus was born from a desire to improve the way bills get paid. Vision, innovation and exemplary service have propelled Paymentus to become the leading paperless electronic billing and payment solution on the market, resulting in 1,300 clients including some of the largest billers in North America.

We know that in order to keep our solutions current and relevant, we need people with the know-how, drive and proclivity for fostering a supremely happy customer experience. Our highly committed, creative employees turned an idea into a secure, SAAS-based Customer Engagement and Payment Platform; one that enables direct-bill organizations to provide a unified customer experience and boost adoption of cost-saving electronic billing and payment services.

Recognized by Deloitte to be among the fastest growing North American companies in 2011, 2013, 2014, and 2016, Paymentus consistently strives to develop better, faster, more secure, cost-efficient billing and payment platforms. We continually seek higher value for our customers, in both solutions and service.

It's what has led to our remarkable growth in the last decade. We succeed when our clients succeed. They succeed when their customer relationships are enhanced and, in turn, their customers participate in these cost-saving electronic services at high rates.

Chapter - 2

INTRODUCTION TO THE PROJECT

Card Scanner is a Debit / Credit card scanning app which was created using Google's ML – Kit and Camera X Android Library. It is able to extract card numbers and expiry date within a fraction of second. It gives a hassle free experience to the users as they won't have to waste time typing long card numbers.

The different technologies used are:

- 1) **Material Design:** For the design of the app
- 2) **Lottie:** For the animation used for scanning
- 3) **Camera X:** For real-time scanning of the card
- 4) **ML Kit:** For extracting text from the frames received from the camera
- 5) **LiveData:** For asynchronously observing the results
- 6) **MVVM:** The architecture of the app

Chapter - 3

IMPLEMENTATION DETAILS

- **Android Studio**



Android Studio is the authority IDE for Android application improvement, in light of IntelliJ IDEA. On the abilities you anticipate from IntelliJ, Android Studio offers:

- Adaptable Gradle-based form the framework Construct variations and numerous apk record age
- Code layouts to assist you with building the regular application highlights
- Rich format proofreader with help for simplified topic altering
- Build up devices to get execution, ease of use, form similarity, and different issues ProGuard and application marking capacities
- Worked in help for Google Cloud Platform, making it simple to incorporate Google Cloud Messaging and App Engine

What's more, substantially more Android Studio bolsters no different programming dialects of IntelliJ for example Java, C++, and more with augmentations, for example, Go; and Android Studio 3.0 or later backings Kotlin and "all Java 7 language highlights and a subset of Java 8 language includes that change by stage rendition." External activities backport some Java 9 highlights. While IntelliJ that Android Studio is based on bolsters all discharged Java forms, and Java 12, it's not satisfactory to what level Android Studio underpins Java forms up to Java 12 (the

documentation makes reference to fractional Java 8 help). Probably some new dialect includes up to Java 12 are usable in Android.

- **Material Design**

Material Design (codenamed Quantum Paper) is a design language developed by Google in 2014. Expanding on the "card" motifs that debuted in Google Now, Material Design uses more grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows.

Google announced Material Design on June 25, 2014, at the 2014 Google I/O conference.

The main purpose of material design is creation of new visual language that combines principles of good design with technical and scientific innovation. Designer Matías Duarte explained that, "unlike real paper, our digital material can expand and reform intelligently. Material has physical surfaces and edges. Seams and shadows provide meaning about what you can touch." Google states that their new design language is based on paper and ink but implementation takes place in an advanced manner.

In 2018, Google detailed a revamp of the language, with a focus on providing more flexibility for designers to create custom "themes" with varying geometry, colors, and typography. Google released Material Theme Editor exclusively for the macOS design application Sketch.

UI Components

TEXT BUTTON OUTLINED BUTTON


CONTAINED BUTTON

BUTTON 1 BUTTON 2 BUTTON 3

Chip 1 Chip 2 Chip 3 Chip 4

- Radio button 1
- Radio button 2
- Radio button 3

- Checkbox 1
- Checkbox 2
- Checkbox 3
- Checkbox 4



This is a card
Material Card
This is subtitle

BUTTON 1 BUTTON 2

- **Permissions in Android**

Every Android app runs in a limited-access sandbox. If your app needs to use resources or information outside of its own sandbox, you can and set up a permission request that provides this access. These steps are part of the .

If you declare any , and if your app is installed on a device that runs Android 6.0 (API level 23) or higher, you must request the dangerous permissions at runtime by following the steps in this guide.

If you don't declare any dangerous permissions, or if your app is installed on a device that runs Android 5.1 (API level 22) or lower, the permissions are automatically granted, and you don't need to complete any of the remaining steps on this page.

Basic Principles:

The basic principles for requesting permissions at runtime are as follows:

- Ask for permissions in context, when the user starts to interact with the feature that requires it.
- Don't block the user. Always provide the option to cancel an educational UI flow related to permissions.
- If the user denies or revokes a permission that a feature needs, gracefully degrade your app so that the user can continue using your app, possibly by disabling the feature that requires the permission.
- Don't assume any system behavior. For example, don't assume that permissions appear in the same permission group. A permission group merely helps the system minimize the number of system dialogs that are presented to the user when an app requests closely-related permissions.

CAMERA PERMISSION

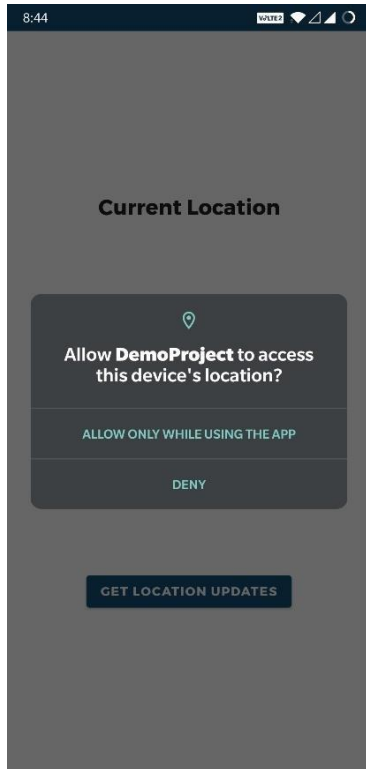
FILES PERMISSION

LOCATION PERMISSION

CALLING PERMISSION

- **Location in Android**

One of the unique features of mobile applications is location awareness. Mobile users take their devices with them everywhere, and adding location awareness to your app offers users a more contextual experience. The location APIs available in Google Play services facilitate adding location awareness to your app with automated location tracking, wrong-side-of-the-street detection, geofencing, and activity recognition.



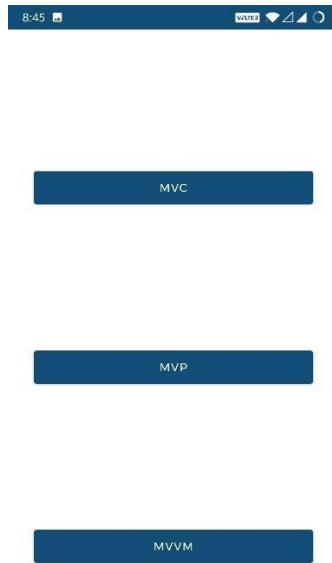
Current Location

Pacca Danga, Afghan Town, Pakki
Dhaki, Old Heritage City, Jammu,
180001

GET LOCATION UPDATES

Location update received

- **Architectures in Android**



- **MVC**

Developing an application by applying a software architecture pattern is always preferred by the developers. It gives modularity to the project files and assures that all the codes get covered in Unit testing. It makes the task easy for developers to maintain the software and to expand the features of the application in the future. There are some architectures that are very popular among developers and one of them is the Model—View—Controller(MVC) Pattern. The MVC pattern suggests splitting the code into 3 components. While creating the class/file of the application, the developer must categorize it into one of the following three layers:

- **Model:** This component stores the application data. It has no knowledge about the interface. The model is responsible for handling the domain logic(real-world business rules) and communication with the database and network layers.
- **View:** It is the UI(User Interface) layer that holds components that are visible on the screen. Moreover, it provides the visualization of the data stored in the Model and offers interaction to the user.
- **Controller:** This component establishes the relationship between the View and the Model.

MVC Demo

Count: 1

Count: 3

Count: 2

- **MVP**

MVP (Model — View —

Presenter) comes into the picture as an alternative to the traditional MVC (Model — View —

Controller) architecture pattern. Using MVC as the software architecture, developers end up with the following difficulties:

Most of the core business logic resides in Controller. During the lifetime of an application, this file grows bigger and it becomes difficult to maintain the code.

Because of tightly-

coupled UI and data access mechanisms, both Controller and View layer falls in the same activity or fragment. This cause problem in making changes in the features of the application.

It becomes hard to carry out Unit testing of the different layer as most of the part which are under testing needs Android SDK components.

MVP Demo

Counter value = 4

Increase Counter

○ MVVM

MVVM suggests separating the data presentation logic(Views or UI) from the core business logic part of the application.

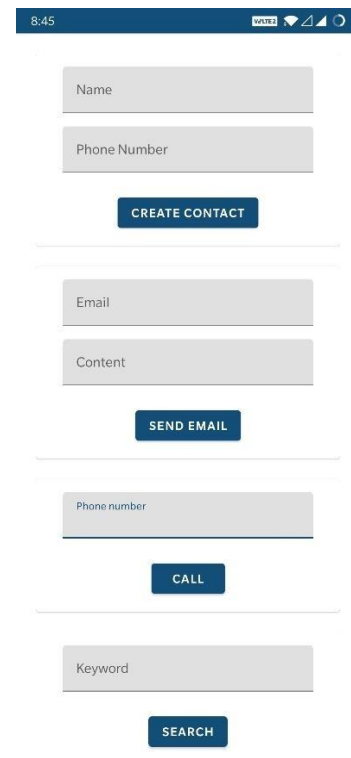
The separate code layers of MVVM are:

- **Model:** This layer is responsible for the abstraction of the data sources. Model and ViewModel work together to get and save the data.
- **View:** The purpose of this layer is to inform the ViewModel about the user's action. This layer observes the ViewModel and does not contain any kind of application logic.
- **ViewModel:** It exposes those data streams which are relevant to the View. Moreover, it serves as a link between the Model and the View.



- **Intents in Android**

An intent is to perform an action on the screen. It is mostly used to start activity, send broadcast receiver, start services and send message between two activities. There are two intents available in android as Implicit Intents and Explicit Intents.



- **Datastore**

Jetpack DataStore is a data storage solution that allows you to store key-value pairs or typed objects with `String`. DataStore uses Kotlin coroutines and Flow to store data asynchronously, consistently, and transactionally.

DataStore provides two different implementations: Preferences DataStore and Proto DataStore. Preferences DataStore stores and accesses data using keys. This implementation does not require a predefined schema, and it does not provide type safety.

Proto DataStore stores data as instances of a custom data type. This implementation requires you to define a schema using `ProtoDataStore`, but it provides type safety.



SET USERNAME

GET USERNAME

RESET

- **Room Database**

Jetpack DataStore is a data storage solution that allows you to store key-value pairs or typed objects with `LiveData`. DataStore uses Kotlin coroutines and Flow to store data asynchronously, consistently, and transactionally.

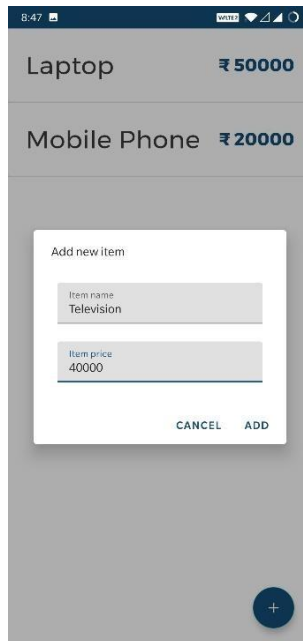
DataStore provides two different implementations: Preferences DataStore and Proto DataStore. Preferences DataStore stores and accesses data using keys. This implementation does not require a predefined schema, and it does not provide type safety.

Proto DataStore stores data as instances of a custom data type. This implementation requires you to define a schema using `ProtoDataStore`, but it provides type safety.



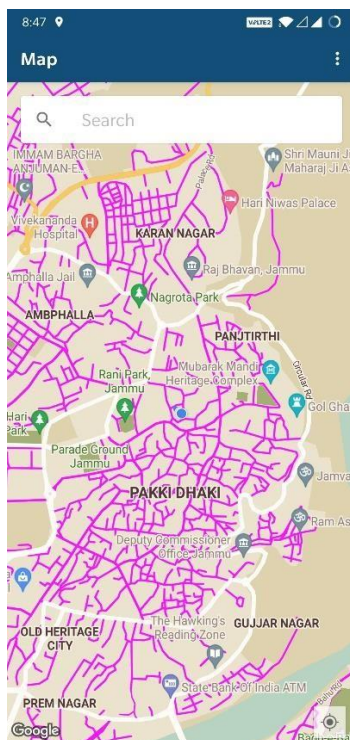
Laptop	₹ 50000
Mobile Phone	₹ 20000

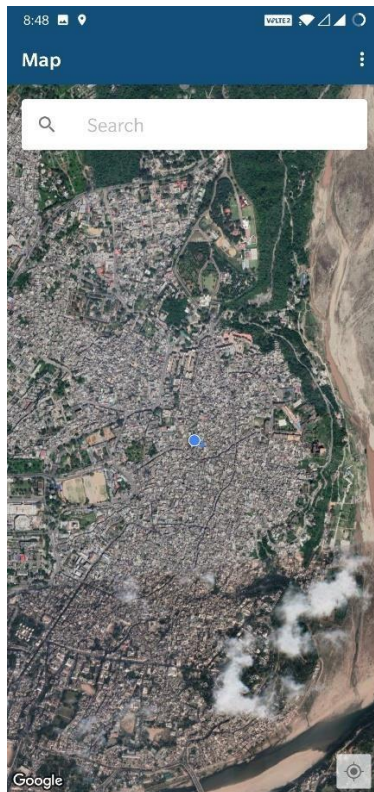
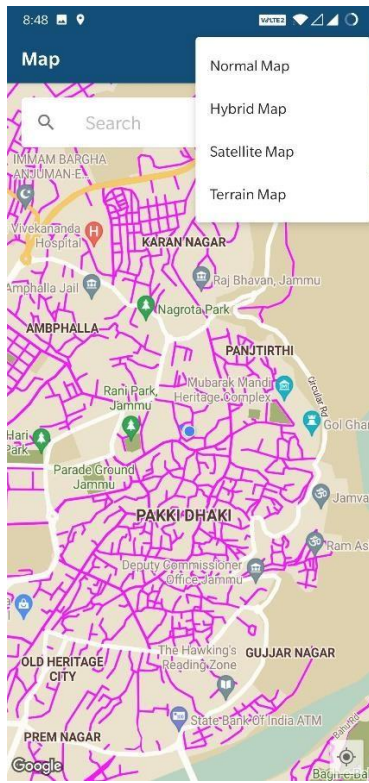




- **Maps SDK**

With the Maps SDK for Android, add maps to your including apps using Google Maps data, map displays, and map gesture responses. You can also provide additional information for map locations and support user interaction by adding markers, polygons, and overlays to your map. The SDK supports both the and Java programming languages and provides additional libraries and extensions for features and programming techniques.





- **Retrofit**

Retrofit is type-

safe REST client for Android and Java which aims to make it easier to consume RESTful web services. We'll not go into the details of Retrofit 1.x versions and jump onto Retrofit 2 directly which has a lot of new features and a changed internal API compared to the previous versions.

Retrofit 2 by default leverages OkHttp as the networking layer and is built on top of it.

Retrofit automatically serialises the JSON response using a POJO(Plain Old Java Object) which must be defined in advanced for the JSON Structure. To serialise JSON we need a converter to convert it into Gson first. We need to add the following dependencies in our build.gradle file.

Coroutines

A coroutine is a concurrency design pattern that you can use on Android to simplify code that executes asynchronously. were added to Kotlin in version 1.3 and are based on established concepts from other languages.

On Android, coroutines help to manage long-running tasks that might otherwise block the main thread and cause your app to become unresponsive. Over 50% of professional developers who use coroutines have reported seeing increased productivity. This topic describes how you can use Kotlin coroutines to address these problems, enabling you to write cleaner and more concise app code.

Features

- Coroutines is our recommended solution for asynchronous programming on Android. Noteworthy features include the following:
- Lightweight: You can run many coroutines on a single thread due to support for `yield`, which doesn't block the thread where the coroutine is running. Suspending saves memory over blocking while supporting many concurrent operations.
- Fewer memory leaks: Use `useContext` to run operations within a scope.
- Builtin cancellation support: `cancel` is propagated automatically through the running coroutine hierarchy.
- Jetpack integration: Many Jetpack libraries include that provide full coroutines support. Some libraries also provide their own `CoroutineScope` that you can use for structured concurrency.



Welcome to Paymentus

The Most Complete eBilling and
Payment Solution

Login

Register



Login

Please login to your account.

Email

Password



LOGIN

Don't have an account? Register

Register

Please enter the details

First Name

Last Name

Email

Phone Number

Zip Code

Security Question 1 

Answer

Security Question 2 

Answer

Password 

Confirm Password 

REGISTER

Already have an account? [Login](#)

- **Scoped Storage in Android**

Android 11 (API level 30) further enhances the platform, giving better protection to app and user data on external storage. This release introduces several enhancements, such as raw file path access, batch edit operations for media, and an updated UI for the Storage Access Framework.

The release also offers improvements to `requestLegacyExternalStorage`, which makes it easier for developers to fulfill their needs after they migrate to using this storage model.

Scoped storage enforcement

Apps that run on Android 11 but target Android 10 (API level 29) can still request the `requestLegacyExternalStorage` attribute. This flag allows apps to access associated with scoped storage, such as granting access to different directories and different types of media files.



CREATE FILE

OPEN FILE

DOWNLOAD FILE

DOWNLOAD IMAGE TO DOWNLOADS

SAVE IMAGE TO INTERNAL

- **Notifications in Android**

Notifications provide short, timely information about events in your app while it's not in use. This page teaches you how to create a notification with various features for Android 4.0 (API level 14) and higher. For an introduction to how notifications appear on Android, see the . For sample code that uses notifications, see the .

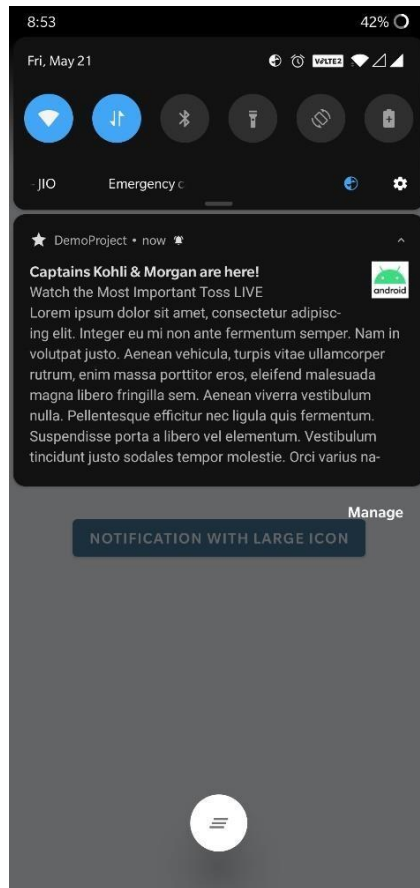
Notice that the code on this page uses the NotificationCompat APIs from the Android support library. These APIs allow you to add features available only on newer versions of Android while still providing compatibility back to Android 4.0 (API level 14). However, some new features such as the inline reply action result in a no-op on older versions.



EXPANDABLE NOTIFICATION

CUSTOM NOTIFICATION

NOTIFICATION WITH LARGE ICON



- **ML Kit**



ML Kit is a mobile SDK that brings Google's on-device machine learning expertise to Android and iOS apps. Use our powerful yet easy to use Vision and Natural Language APIs to solve common challenges in your apps or create brand-new user experiences. All are powered by Google's best-in-class ML models and offered to you at no cost.

ML Kit's APIs all run on-device, allowing for real time use cases where you want to process a live camera stream for example. This also means that the functionality is available offline.

○ **Text Recognition**

With ML Kit's text recognition APIs can recognize text in any Latin-based character set. They can also be used to automate data-entry tasks such as processing credit cards, receipts, and business cards.

Input image guidelines

- For ML Kit to accurately recognize text, input images must contain text that is represented by sufficient pixel data. Ideally, each character should be at least 16x16 pixels. There is generally no accuracy benefit for characters to be larger than 24x24 pixels.

So, for example, a 640x480 image might work well to scan a business card that occupies the full width of the image. To scan a document printed on letter-sized paper, a 720x1280 pixel image might be required.

- Poor image focus can affect text recognition accuracy. If you aren't getting acceptable results, try asking the user to recapture the image.
- If you are recognizing text in a real-time application, you should consider the overall dimensions of the input images. Smaller images can be processed faster. To reduce latency, ensure that the text occupies as much of the image as possible, and capture images at lower resolutions (keeping in mind the accuracy requirements mentioned above)

Recognize text in images

To recognize text in an image, run the text recognizer as described below.

Prepare the input image

To recognize text in an image, create an `InputImage` object from either a `Bitmap`, `media.Image`, `ByteBuffer`, byte array, or a file on the device. Then, pass the `InputImage` object to the `TextRecognizer`'s `processImage` method.

You can create an `InputImage` from different sources, each is explained below.

Using a media.Image

To create an InputImage object from a media.Image object, such as when you capture an image from a device's camera, pass the media.Image object and the image's rotation to InputImage.fromMediaImage().

Get an instance of TextRecognizer

```
val recognizer = TextRecognition.getClient()
```

Process the image

Pass the image to the process method:

```
val result = recognizer.process(image)
    .addOnSuccessListener { visionText ->
        // Task completed successfully
        // ...
    }
    .addOnFailureListener { e ->
        // Task failed with an exception
        // ...
    }
```

Extract text from blocks of recognized text

If the text recognition operation succeeds, a Text object is passed to the success listener. A Text object contains the full text recognized in the image and zero or more TextBlock objects.

Each TextBlock represents a rectangular block of text, which contains zero or more objects. Each Line object contains zero or more objects, which represent words and word-like entities such as dates and numbers.

For each TextBlock, Line, and Element object, you can get the text recognized in the region and the bounding coordinates of the region.

- **Camera X**

CameraX is a Jetpack support library, built to help you make camera app development easier. It provides a consistent and easy-to-use API surface that works across most Android devices, with backward-compatibility to Android 5.0 (API level 21).

While CameraX leverages the capabilities of camera2, it uses a simpler approach that is lifecycle-aware and is based on use cases. It also resolves device compatibility issues for you so that you don't have to include device-specific code in your code base. These features reduce the amount of code you need to write when adding camera capabilities to your app.

Lastly, CameraX enables developers to leverage the same camera experiences and features that pre-installed camera apps provide, with as little as two lines of code. CameraX Extensions are optional additions that enable you to add effects on supported devices. These effects include Portrait, HDR, Night, and Beauty.

Primary benefits

CameraX improves the developer experience in the following ways:

Ease of use

CameraX introduces use cases, which allow you to focus on the task you need to get done instead of spending time managing device-specific nuances. There are several basic use cases:

: get an image on the display

: access a buffer seamlessly for use in your algorithms, such as to pass into MLKit

: save high-quality images

These use cases work across all devices running Android 5.0 (API level 21) or higher, ensuring that the same code works on most devices in the market.

Consistency across devices

Managing consistent camera behavior across apps is hard. There is a lot to account for, including aspect ratio, orientation, rotation, preview size, and high-resolution image size. With CameraX, these basic behaviors just work.

We're investing in an automated CameraX test lab that tests a variety of camera behaviors across all operating system flavors since Android 5.0 (API level 21). These tests are run on an ongoing basis to identify and fix a wide range of issues

Our aim is to, over time, significantly reduce your test burden.

New camera experiences

CameraX has an optional add-on, called `CameraX.Experimental`, which allow you to access the same features and capabilities as those in the native camera app that ships with the device, with just two lines of code.

The first set of capabilities available include Portrait, HDR, Night, and Beauty. These capabilities are available on supported devices.

CameraX Architecture

CameraX is an addition to Jetpack that makes it easier to leverage the capabilities of the camera hardware. This topic covers the architecture of CameraX, including its structure, how to work with the API, how to work with lifecycles, and how to combine use cases.

CameraX structure

Developers use CameraX to interface with a device's camera through an abstraction called a use case. The following use cases are currently available:

Preview: accepts a surface for displaying a preview, such as a `PreviewView`.

Image analysis: provides CPU-accessible buffers for analysis, such as for machine learning.

Image capture: captures and saves a photo.

Use cases can be combined and active concurrently. For example, an app can let the user view the image that the camera sees using a preview use case, have an image analysis use case that determ

ines whether the people in the photo are smiling, and include an image capture use case to take a picture once they are.

API model

To work with the library, you specify the following things:

The desired use case with configuration options.

What to do with output data by attaching listeners.

The intended flow, such as when to enable cameras and when to produce data, by binding the use case to .

You configure use cases using `set()` methods and finalize them with the `build()` method. Each use case object provides a set of use case-specific APIs. For example, the image capture use case provides a `takePicture()` method call.

Instead of an application placing specific start and stop method calls in `onResume()` and `onPause()`, the application specifies a lifecycle to associate the camera with, using `cameraProvider.bindToLifecycle()`. That lifecycle then informs CameraX when to configure the camera capture session and ensures camera state changes appropriately to match lifecycle transitions.

CameraX Lifecycles

CameraX observes a lifecycle to determine when to open the camera, when to create a capture session, and when to stop and shut down. Use case APIs provide method calls and callbacks to monitor progress.

As explained in , you can bind some mixes of use cases to a single lifecycle. When your app needs to support use cases that can't be combined, you can do one of the following:

Group compatible use cases together into more than one and then switch between fragments

Create a custom lifecycle component and use it to manually control the camera lifecycle

If you decouple your view and camera use cases' Lifecycle owners (for example, if you use a custom lifecycle or a), then you must ensure that all use cases are unbound from CameraX by using `ProcessCameraProvider.unbindAll()` or by unbinding each use case individually. Alternatively, when

When you bind use cases to a Lifecycle, you can let CameraX manage opening and closing the capture session and unbinding the use cases.

If all of your camera functionality corresponds to the lifecycle of a single lifecycle-aware component such as an AppCompatActivity or an AppCompatActivity fragment, then using the lifecycle of that component when binding all the desired use cases will ensure that the camera functionality is ready when the lifecycle-aware component is active, and safely disposed of, not consuming any resources, otherwise.

Analyze images

The image analysis use case provides your app with a CPU-accessible image to perform image processing, computer vision, or machine learning inference on. The application implements an analyze method that is run on each frame.

Implementation

Images are processed by passing an executor in which the image analysis is run and an ImageAnalysis.Analyzer parameter to the setAnalyzer() method.

Image analysis can work in two modes: blocking and non-blocking. Blocking mode is enabled by calling setBackpressureStrategy() with . In this mode, the executor receives frames from the camera in sequential order; this means that, if the analyze() method takes longer than the latency of a single frame at the current framerate, the frames may no longer be current since new frames are blocked from entering the pipeline until the method returns.

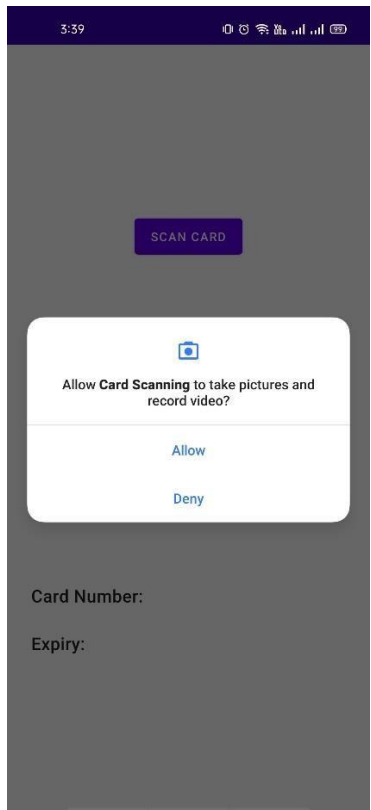
Non-blocking mode is enabled by calling setBackpressureStrategy() with . In this mode, the executor receives the last available frame from the camera at the time that the analyze() method is called. If the method takes longer than the latency of a single frame at the current framerate, some frames might be skipped so that the next time analyze() receives data, it gets the last frame available in the camera pipeline.

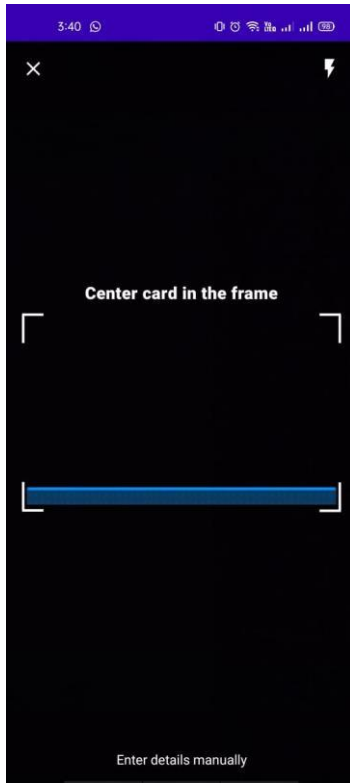
Before returning from analyze(), close the image reference by calling image.close() to avoid blocking the production of further images (causing the preview to stall) and to avoid potentially dropping images. The method must complete analysis or make a copy instead of passing the image reference beyond the analysis method.

SCAN CARD

Card Number:

Expiry:





SCAN CARD

Card Number: 510372 [REDACTED]

Expiry Month: 11 Expiry Year: 23



Chapter - 4

RESULTS AND CONCLUSION

This internship was indeed a pool of knowledge, not only have I gained knowledge in Android Development but I have also learned about how development of any project takes place, how team works, how the work of each employee is tracked, how work is distributed between different team mates, what are the different stages of development, what are the technical problems that one faces in the development of any project, what all things are required before the development of any project, what the code base should be like and what norms need to be followed in the development.

The card scanner is able to scan various types of debit / credit cards. It easily scans non embossed cards but takes some time to scan the embossed ones. This can be improved by further improvements in the scanning algorithm used.

References

- [1] <https://developer.android.com/>
- [2] <https://developer.android.com/training/camerax>
- [3] <https://developers.google.com/ml-kit/vision/text-recognition>
- [4] <https://material.io/design>
- [5] <https://github.com/googlesamples/mlkit>