**LICENSE PLATE SCANNER**

**(ANDROID)**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| SERIAL NO. | TOPIC |
| Chapter 1 | Introduction |
| Chapter 2 | Optical Character Recognition |
| Chapter 3 | OCR Integration in android |
| Chapter 4 | Database Systems |
| Chapter 5 | Design and Modelling |
| Chapter 6 | Application Overview |
| VIII | Conclusion |
| IX | Future Work |
| X | Reference |

**CHAPTER 1**

**INTRODUCTION**

Automatic number plate recognition (ANPR) is an image processing technology which uses number (license) plate to identify the vehicle. The objective is to design an efficient automatic authorized vehicle identification system by using the vehicle number plate. The system is implemented on the entrance for security control of a highly restricted area like military zones or area around top government offices e.g. Parliament, Supreme Court etc. This system can be used by every man for their security purpose. For general public an android application will installed on their mobile phone. After that whenever he wants to know the details of any vehicle he just have to capture the image of licence plate and then that image will be processed and he will get the desired information about that vehicle. This system is very important and must needed. For defence purposes the developed system first detects the vehicle and then captures the vehicle image. Vehicle number plate region is extracted using the image segmentation in an image. Optical character recognition technique (OCR) is used for the character recognition. The resulting data is then used to compare with the records on a database so as to come up with the specific information like the identity of owner, place of registration, address, etc. The system is implemented and simulated in JAVA, and it performance is tested on real image.

**Chapter 2**

**OPTICAL CHARACTER RECOGNITION**

**2.1Introduction**

Optical Character Recognition, or OCR, is a technology that enables us to convert different types of documents, such as scanned paper documents, PDF files or images captured by a digital camera into editable and searchable data.

Imagine you’ve got a paper document - for example, magazine article, brochure, or PDF contract your partner sent to you by email. Obviously, a scanner is not enough to make this information available for editing, say in Microsoft Word. All a scanner can do is create an image or a snapshot of the document that is nothing more than a collection of black and white or colour dots, known as a raster image. In order to extract and repurpose data from scanned documents, camera images or image-only PDFs, you need an OCR software that would single out letters on the image, put them into words and then - words into sentences, thus enabling you to access and edit the content of the original document.

**2.2 HISTORY OF OCR**

Most people think getting machines to read human text is a relatively recent innovation, but it's older than we might suppose. Here's a whistle-stop tour through OCR history:

* 1928/9: Gustav Tauschek of Vienna, Austria patents a basic OCR "reading machine." Paul Handel of General Electric files a patent for a similar system in the United States in April 1931. Both are based on the idea of using light-detecting photocells to recognize patterns on paper or card.
* 1949: L.E. Flory and W.S. Pike of RCA Laboratories develop a photocell-based machine that can read text to blind people at a rate of 60 words per minute. (Read all about it in the [February 1949 issue of Popular Science](http://books.google.com/books?id=pCQDAAAAMBAJ&lpg=PA125&ots=3cnACr-VKH&dq=1949%20rca%20scanner%20blind&pg=PA125#v=onepage&q&f=false).)
* 1950: [David H. Shepard](https://en.wikipedia.org/wiki/David_H._Shepard) develops machines that can turn printed information into machine-readable form for the US military and later founds a pioneering OCR company called Intelligent Machines Research (IMR). Shepherd also develops a machine-readable font called Farrington B (also called OCR-7B and 7B-OCR), now widely used to print the embossed numbers on credit cards.
* 1960: [Lawrence (Larry) Roberts](https://en.wikipedia.org/wiki/Lawrence_Roberts_(scientist)), a computer graphics researcher working at MIT, develops early text recognition using specially simplified fonts such as OCR-A. He later becomes one of the founding fathers of the [Internet](https://www.explainthatstuff.com/internet.html).
* 1950s/1960s: Reader's Digest and RCA work together to develop some of the first commercial OCR systems.
* 1960s: Postal services around the world begin to use OCR technology for mail-sorting. They include the US Postal Service, Britain's General Post Office (GPO, now called Royal Mail), Canada Post, and the German Deutsche Post. Helped by companies such as Lockheed Martin, postal services remain at the forefront of OCR research to this day.
* 1974: [Raymond Kurzweil](http://www.kurzweilai.net/) develops the Kurzweil Reading Machine (KRM) that combines a flatbed scanner and speech synthesizer in a machine that can read printed pages aloud to blind people. Kurzweil's OCR software is acquired by Xerox and marketed under the names ScanSoft and (later) Nuance Communications.
* 1993: The [Apple Newton MessagePad (PDA)](https://en.wikipedia.org/wiki/MessagePad) is one of the first handheld computers to feature handwriting recognition on a touch-sensitive screen. During the 1990s, handwriting recognition becomes an increasingly popular feature on cellphones, PDAs (notably the pioneering Palm and PalmPilot), and other handhelds.
* 2000: Researchers at Carnegie Mellon University flip the problem of developing a good OCR system on its head—and develop a spam-busting system called CAPTCHA

**2.3 HOW DOES OCR WORK?**

Let's suppose life was really simple and there was only one letter in the alphabet: A. Even then, you can probably see that OCR would be quite a tricky problem—because every single person writes the letter A in a slightly different way. Even with printed text, there's an issue, because books and other documents are printed in many different typefaces (fonts) and the letter A can be printed in many subtly different forms.

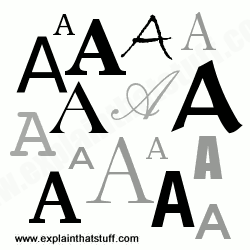
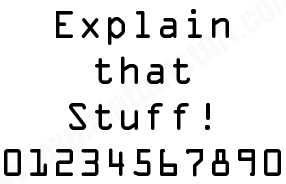


Figure 2.1 There's a fair bit of variation between these different versions of a capital letter A, printed in different computer fonts, but there's also a basic similarity: you can see that almost all of them are made from two angled lines that meet in the middle at the top, with a horizontal line between.

Broadly speaking, there are two different ways to solve this problem, either by recognizing characters in their entirety (pattern recognition) or by detecting the individual lines and strokes characters are made from (feature detection) and identifying them that way.

**2.3.1 Pattern recognition**

If everyone wrote the letter A exactly the same way, getting a computer to recognize it would be easy. You'd just compare your scanned image with a stored version of the letter A and, if the two matched, that would be that. So how do you get everyone to write the same way? Back in the 1960s, a special font called **OCR-A** was developed that could be used on things like bank checks and so on. Every letter was exactly the same width (so this was an example of what's called a monospace font) and the strokes were carefully designed so each letter could easily be distinguished from all the others. Check-printers were designed so they all used that font, and OCR equipment was designed to recognize it too. By standardizing on one simple font, OCR became a relatively easy problem to solve. The only trouble is, most of what the world prints isn't written in OCR-A—and no-one uses that font for their handwriting! So the next step was to teach OCR programs to recognize letters written in a number of very common fonts (ones like Times, Helvetica, Courier, and so on). That meant they could recognize quite a lot of printed text, but there was still no guarantee they could recognize any font you might send their way.

Figure 2.2 OCR-A font: Designed to be read by computers as well as people. We might not recognize the style of text, but the numbers probably do look familiar to you from checks and computer printouts. Note that similar-looking characters (like the lowercase "l" in Explain and the number "1" at the bottom) have been designed so computers can easily tell them apart.

**2.3.2 Feature detection**

Also known as feature extraction or intelligent character recognition (ICR), this is a much more sophisticated way of spotting characters. Suppose you're an OCR computer program presented with lots of different letters written in lots of different fonts; how do you pick out all the letter As if they all look slightly different? You could use a rule like this: If you see two angled lines that meet in a point at the top, in the center, and there's a horizontal line between them about halfway down, that's a letter A. Apply that rule and you'll recognize most capital letter As, no matter what font they're written in. Instead of recognizing the complete pattern of an A, you're detecting the individual component features (angled lines, crossed lines, or whatever) from which the character is made. Most modern **omni font** OCR programs (ones that can recognize printed text in any font) work by feature detection rather than pattern recognition. Some use [neural networks](https://www.explainthatstuff.com/introduction-to-neural-networks.html) (computer programs that automatically extract patterns in a brain-like way).

****

Figure 2.3 Feature detection: We can be pretty confident that we're looking at a capital letter A if we can identify these three component parts joined together in the correct way.

**2.4 ADVANTAGES OF OCR**

Probably the most well-known use case for OCR is converting printed paper documents into machine-readable text documents. Once a scanned paper document went through OCR processing, the text of the document can be edited with word processors like Microsoft Word or Google Docs. Before OCR technology was available, the only option to digitise printed paper documents was to manually re-typing the text. Not only was this massively time consuming, it also came with inaccuracy and typing errors.

OCR is often used as a “hidden” technology, powering many well known systems and services in our daily life. Less known, but as important, use cases for OCR technology include data entry automation, indexing documents for search engines, automatic number plate recognition, as well as assisting blind and visually impaired persons.

OCR technology has proven immensely useful in digitising historic newspapers and texts that have now been converted into fully searchable formats and had made accessing those earlier texts easier and faster.

Machine-readable text can also be decoded by screen readers, tools that use speech synthesizers (computerized voices, like the one Professor [Stephen Hawking](https://en.wikipedia.org/wiki/Stephen_Hawking) used) to read out the words on a screen so blind and visually impaired people can understand them. (Back in the 1970s, one of the first major uses of OCR was in a [photocopier](https://www.explainthatstuff.com/photocopier.html)-like device called the **Kurzweil Reading Machine**, which could read printed books out loud to blind people.)

**2.5 TABULAR COMPARISION OF EXISTING OCRs**



Table 2.1. Tabular comparison of technologies

**2.6 WHAT ARE THE LIMITATIONS OF OCR?**

Although Optical Character Recognition (OCR) scanning technology has increased rapidly over the years, there are, however, limitations in regards to the source materials and character formatting. Following are some of them:

* Text from a source with a font size of less than 12 points will results in more errors.
* Often the target text is the one which is selected from the bottom in case of multiple sentences in front of the scanner.
* Most document formatting is lost during text scanning, except for paragraph marks and tab stops. Sometimes bold, italics and underline are recognised, depending on your software.
* The output from a finished text scan will be a single column editable computer file. This computer file will always require spellchecking and proofreading as well as reformatting to desired final layout.
* Scanning of plain text files or spreadsheet print outs usually work, however the data needs to be reformatted to match the original.
* Source materials that often cause issues are:
* Forms
* Small text
* Blurry copies
* Mathematical formulas
* Draft copy
* Coloured paper
* Handwritten text
* Unusual or script-type fonts
* Document formatting may be lost during text scanning (bold, italic & underline are not always recognized).
* Output from a finished text scan may be a single column editable text file. Text file will always require spellchecking and proofreading as well as reformatting to desired final layout

**Chapter 3**

**OCR INTEGRATION IN ANDROID**

We have used the **Google Mobile Vision API** to read texts via the smartphone’s camera and generate the desired result in a text format.

**3.1 GOOGLE MOBILE VISION API**

The Mobile Vision API provides a framework for finding objects in photos and video. The framework includes **detectors**, which locate and describe visual objects in images or video frames, and an event driven API that tracks the position of those objects in video.

Currently, the Mobile Vision API includes **face**, **barcode**, and **text detectors, which can be applied separately or together.** The vision package includes a framework of common base functionality, and sub-packages for specific detector implementations such as the **TEXT RECOGNITION API** for text detection.

# 3.2 TEXT RECOGNITION API

Text recognition is the process of detecting text in images and video streams and recognizing the text contained therein. Once detected, the recognizer then determines the actual text in each block and segments it into lines and words. The Text API detects text in Latin based languages (French, German, English, etc.), in real-time, on device.

The Text API can recognize text in any Latin-based language. This includes, but is not limited to:

Catalan Danish Dutch English

Hungarian Italian Latin Norwegian

Romanian Spanish Swedish Tagalog

German French Portugese Turkish

Finnish Polish

## **3.2.1 TEXT STRUCTURE**

The Text Recognizer segments text into blocks, lines, and words. Roughly speaking:

* a **Block** is a contiguous set of text lines, such as a paragraph or column,
* a **Line** is a contiguous set of words on the same vertical axis, and
* a **Word** is a contiguous set of alphanumeric characters on the same vertical axis.

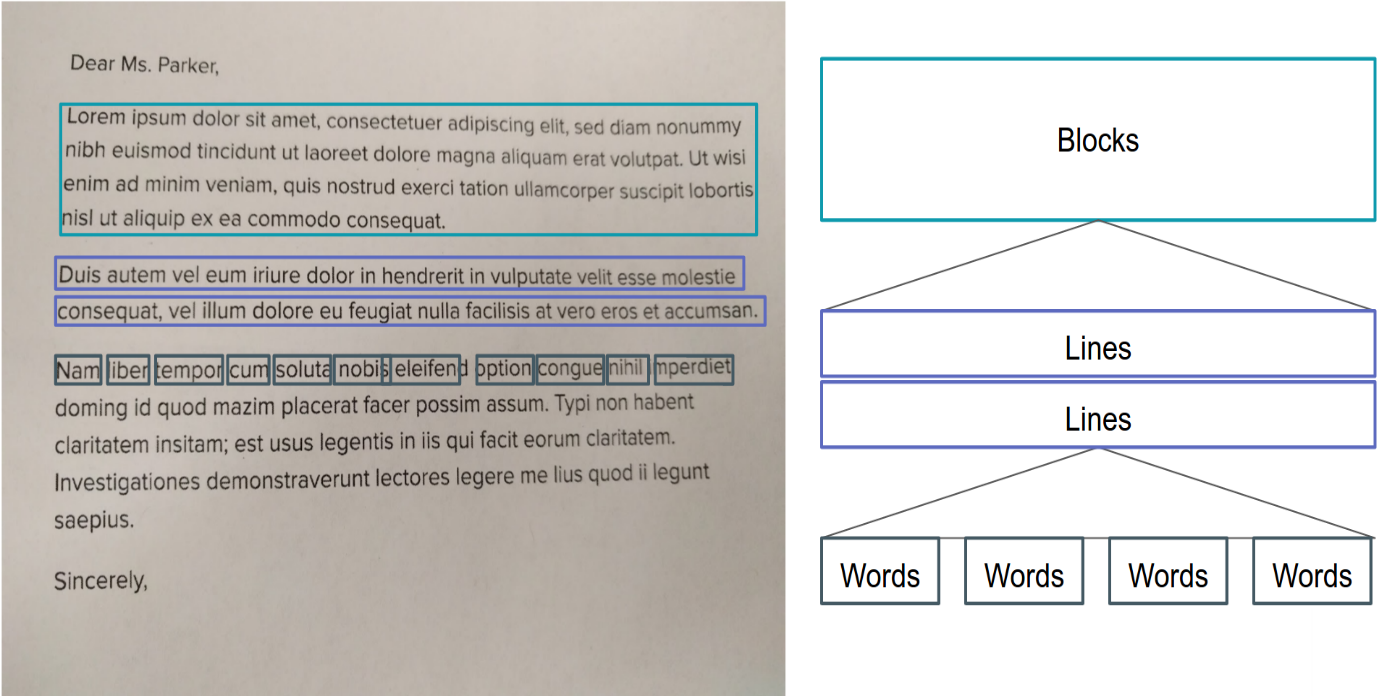
The image below highlights examples of each of these in descending order. The first highlighted block, in cyan, is a Block of text. The second set of highlighted blocks, in blue, are Lines of text. Finally, the third set of highlighted blocks, in dark blue, are Words.

Figure 3.1. Segmentation process by text recognizer.

**3.3 IMPLEMENTATION OF THE API**

**3.3.1 MINIMUM REQUIREMENTS**

* Android Studio version 3.1.
* A test device with Android 4.1+ and a rear-facing camera.
* A USB micro/USB-C cable.
* Google Repository should be up to date. It should be at least version 26.

**3.3.2 PHASES OF IMPLEMENTATION**

* The build.gradle file is opened in the app module and the dependencies block are changed to include the play-services-vision dependency. When we’re done, it should look like this:

dependencies {

implementation fileTree(dir: 'libs', include: ['\*.jar'])

implementation 'com.android.support:support-v4:26.1.0'

implementation 'com.android.support:design:26.1.0'

implementation 'com.google.android.gms:play-services-vision:15.0.0'

}

* **Setting Up TextRecognizer and CameraSource :**

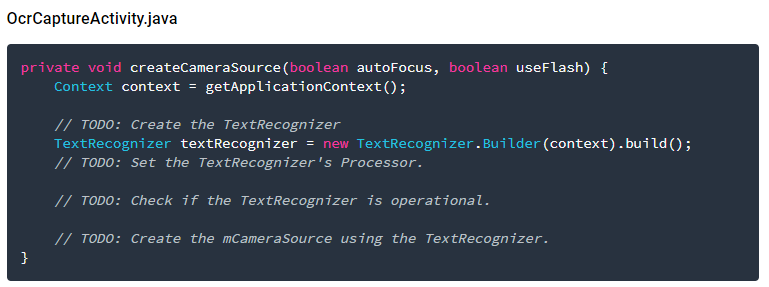
To start things out, we're going to create our TextRecognizer. This detector object processes images and determines what text appears within them. Once it's initialized, a TextRecognizer can be used to detect text in all types of images. Find the createCameraSource method and build a TextRecognizer.

Figure 3.2. Initializing TextRecognizer

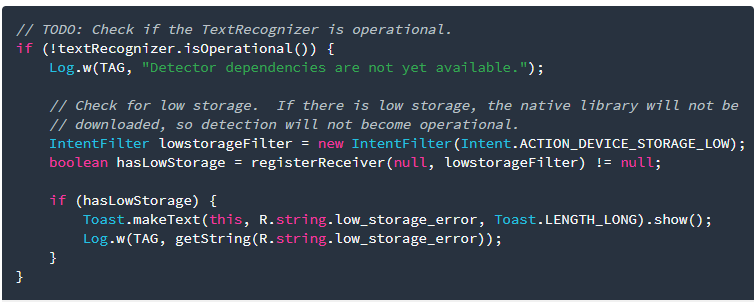
 Now, the TextRecognizer is built. However, it might not work yet. If the device does not have enough storage, or Google Play Services can't download the OCR dependencies, the TextRecognizer object may not be operational. Before we start using it to recognize text, we should check that it's ready. We'll add this check to createCameraSource after we initialized the TextRecognizer

Figure 3.3. Checking for operational TextRecognizer

Now that we've checked that the TextRecognizer is operational, we could use it to detect individual frames. But we want to do something a little more interesting: read text live in the camera view. In order to do that, we'll create a CameraSource, which is a camera manager pre-configured for Vision processing. We're going to set the resolution high and turn autofocus on, because that's a good match for recognizing small text. If we knew our users would be looking at large blocks of text, like signage, we might use a lower resolution, which would be able to process frames more quickly.If we build the app now, we should see a live camera view! But in order to process images from the camera, we need to handle that last TODO in createCameraSource: create a Processor to handle the text detections as they come in. We'll do that in the next step.

* **Creating the OcrDetectorProcessor :**

By now, our app could detect text on individual frames using the detect method on the TextRecognizer. That's what we would do if we wanted to find text in a photograph or other image file. But in order to read text straight from the camera, it's useful to implement a Processor, which will handle detections as often as they become available. Thus, we go to the OcrDetectorProcessor class and have it implement Detector.Processor<TextBlock>.

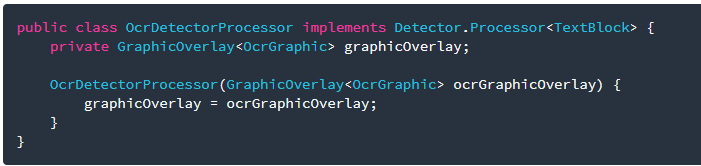


Figure 3.4. Creating OcrDetectorProcessor

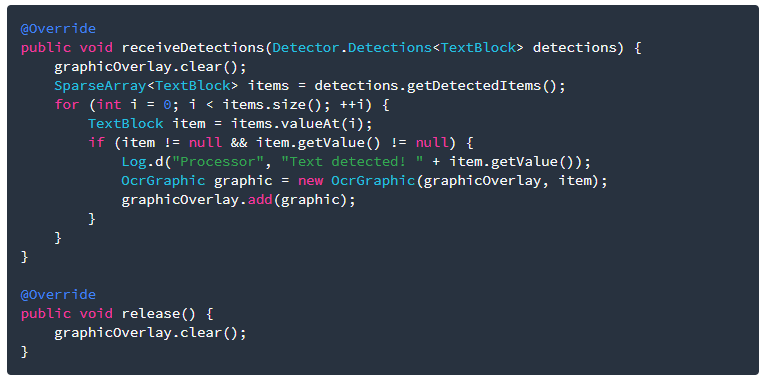
That interface requires two methods be implemented. The first, receiveDetections, will receive TextBlocks from the TextRecognizer as they become available. The second, release, can be used to cleanly get rid of resources when the TextRecognizer is disposed of. In this case, we only have to clear the graphic overlay, which cleans up all the OcrGraphic objects.We'll get the TextBlocks from the detection and create OcrGraphic objects for each text block that the processor detects. For now, they won't render; we'll implement their draw behavior in the next step.

Figure 3.5. Implementing interface methods

Now that the processor is ready, we have to set the textRecognizer to use it. So, we head back to the last remaining TODO in the createCameraSource method in OcrCaptureActivity, where TextRecognizer is present.

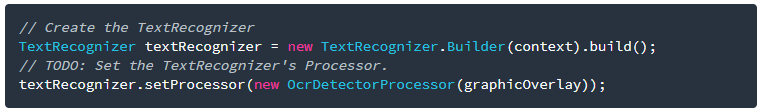


Figure 3.6. Setting the OcrDetectorProcessor in TextRecognizer.

Now we build and run the app. At this point, we should be able to point the camera at text and see the "Text detected!". We can see the debug message in logcat to verify the same.

* **Drawing the graphics to the screen:**

The debug message tells us that text is being recognized. We'd like the user to see that, too, by drawing the text on top of the camera preview.

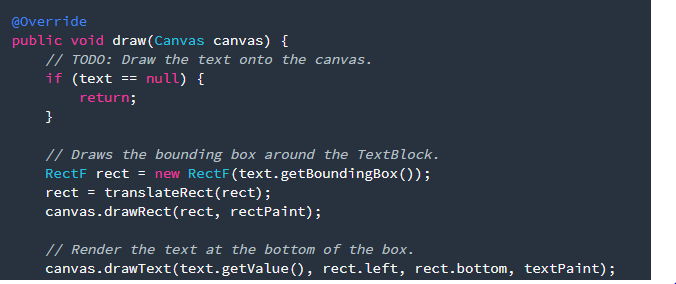
Let's implement the OcrGraphic draw method. We want to see if the graphic has text, translate its bounding box to the appropriate coordinates for the canvas, and then draw the box and text.

Figure 3.7. Implementing draw method

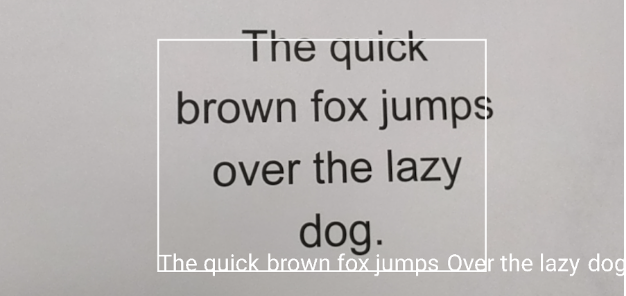


Figure 3.8. OCR recognition

In this example,the bounding box looks right, but the text is all at the bottom of the box.

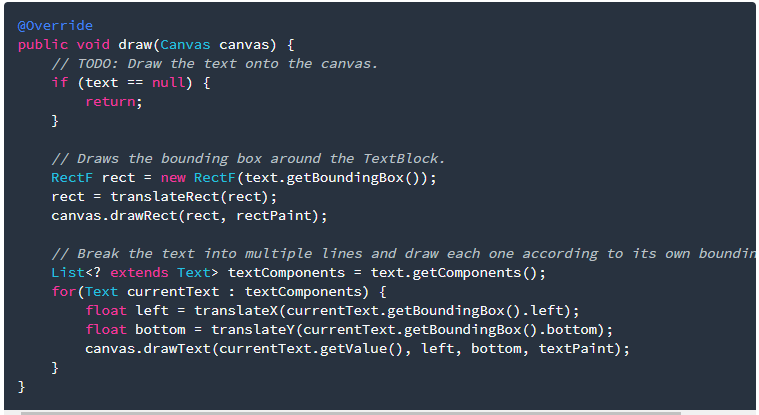
That's because the engine puts all the text it recognizes in a TextBlock into one complete sentence, even if it sees the sentence broken over multiple lines. If we want the complete sentence, that's very useful. But what if we want to know where each individual line of text actually is? We can get the Lines from a TextBlock by calling getComponents, and then we can iterate over each line to get the location and values of the text within it. This lets us put the text in the place it actually appears.

Figure 3.9. Using getComponents to fetch each sentence

We can choose how granular you want to go based on your application's needs. If you like, you can call getComponents on each Line and get the positions of the actual Elements (words, in Latin languages). You can also customize the textSize to fill as much space as the actual text does on screen.

 We can choose how granular you want to go based on your application's needs. If you like, you can call getComponents on each Line and get the positions of the actual Elements (words, in Latin languages). You can also customize the textSize to fill as much space as the actual text does on screen.

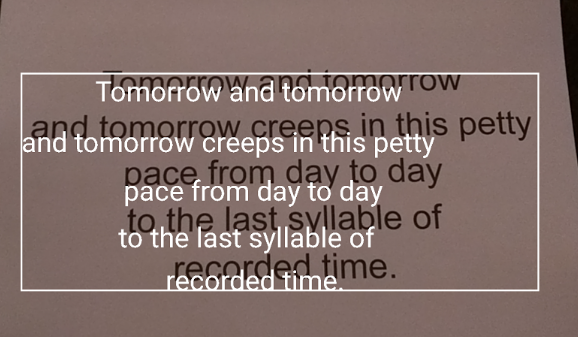


Figure 3.10. Fetching one sentence at a time

**CHAPTER 4: DATABASE SYSTEMS**

**4.1 FIREBASE**

Firebase the latest platform by Google.Inc which has effective tools to develop Mobile and Web Applications. It is built to help developers to share various features between cross-platform apps related Database, Config, and Notifications. You can perform various complex operation easily with firebase to build high-quality and bug-free apps for Android, iOS, and The Web. With Firebase, you can integrate authentication functionality in the application, create storage functionality for an app to collect and store pictures, audio, and video without using the server-side code.

Firebase provides fast, secure, static, and production-grade hosting for developers. It allows developers to efficiently deploy web apps and static content to a CDN(Content Delivery Network).

It is another best advantage of Firebase. It is very useful when you want to create an app for storing and serving files of users such and images and videos.

Firebase notification is a free service which allows targeted user notifications for mobile app developers. It has the notification console GUI where you can create and send notifications to targeted users.

This feature is used to index application in Google search results. After app indexing, If a user search related to your app, it will start the app installed in user’s device directly from the search result.

It gives a basic and unified platform with so many Google features packed-in. You don't need to configure your server when you use Firebase. Everything will be taken care of by Firebase automatically.

**4.1.1 A COMPARISION WITH POPULAR DATABASES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Firebase Realtime Database** | **MySQL** | **SQLite** |
| **Description** | **Cloud-hosted realtime document store. iOS, Android, and JavaScript clients share one Realtime Database instance and automatically receive updates with the newest data.** | **Widely used open source RDBMS** | **Widely used in-process RDBMS** |
| **Primary database model** | **Document store** | **Relational DBMS** | **Relational DBMS** |
| **Developer** | **Google** | **Oracle** | **Dwayne Richard Hipp** |
| **Initial release** | **2012** | **1995** | **2000** |
| **Current release** |  | **8.0.16, April 2019** | **3.28.0, April 2019** |
| **License** | **commercial** | **Open Source** | **Open Source** |
| **Cloud-based only** | **yes** | **no** | **no** |
| **Implementation language** |  | **C and C++** | **C** |
| **Server operating systems** | **hosted** | **FreeBSD Linux OS X Solaris Windows** | **server-less** |
| **Data scheme** | **schema-free** | **yes** | **yes** |
| **Typing** | **yes** | **yes** | **yes** |
| **XML support** | **no** | **yes** | **no** |
| **Secondary indexes** | **yes** | **yes** | **yes** |
| **SQL** | **no** | **yes** | **yes** |
| **APIs and other access methods** | **Android iOS JavaScript API RESTful HTTP API** | **Proprietary native API ADO.NET JDBC ODBC** | **ADO.NET  JDBC  ODBC** |
| **Server-side scripts** | **limited functionality with using 'rules'** | **yes** | **no** |
| **Triggers** | **Callbacks are triggered when data changes** | **yes** | **yes** |
| **Partitioning methods** |  | **Horizontal partitioning, sharding with MySQL Cluster or MySQL Fabric** | **none** |
| **Replication methods** |  | **Master-master replication Master-slave replication** | **none** |
| **MapReduce** | **no** | **no** | **no** |
| **Consistency concepts** | **Eventual Consistency  Immediate Consistency** | **Immediate Consistency** |  |
| **Foreign keys** | **no** | **yes** | **yes** |
| **Transaction concepts** | **yes** | **ACID** | **ACID** |
| **Concurrency** | **yes** | **yes** | **yes** |
| **Durability** | **yes** | **yes** | **yes** |
| **In-memory capabilities** |  | **yes** | **yes** |
| **User concepts** | **yes, based on authentication and database rules** | **Users with fine-grained authorization concept** | **no** |

Table 4.1. A comparison between Firebase and other popular databases

**4.2 Key Features Used:**

**4.2.1 Real-time Database**:

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

Store and sync data with our NoSQL cloud database. Data is synced across all clients in realtime and remains available when your app goes offline.

**Key Capabilities**:

**Realtime**: Instead of typical HTTP requests, the Firebase Realtime Database uses data synchronization—every time data changes, any connected device receives that update within milliseconds. Provide collaborative and immersive experiences without thinking about the networking code.

**Offline**: Firebase apps remain responsive even when offline because the Firebase Realtime Database SDK persists your data to disk. Once connectivity is reestablished, the client device receives any changes it missed, synchronizing it with the current server state.

**Accessible from Client Devices**: The Firebase Realtime Database can be accessed directly from a mobile device or web browser; there’s no need for an application server. Security and data validation are available through the Firebase Realtime Database Security Rules, expression-based rules that are executed when data is read or written.

**Scale across multiple databases**: With Firebase Realtime Database on the Blaze pricing plan, you can support your app's data needs at scale by splitting your data across multiple database instances in the same Firebase project. Streamline authentication with Firebase Authentication on your project and authenticate users across your database instances. Control access to the data in each database with custom Firebase Realtime Database Rules for each database instance.

**How does it work?**

The Firebase Realtime Database lets us build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, realtime events continue to fire, giving the end user a responsive experience. When the device regains connection, the Realtime Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

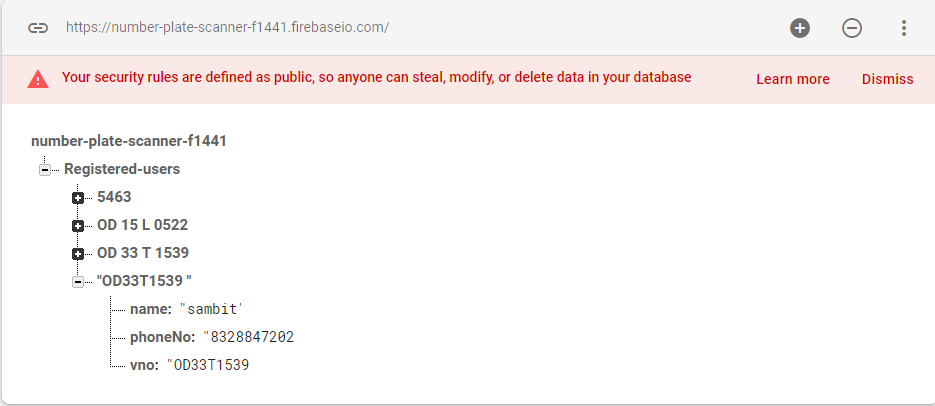
The Realtime Database is a NoSQL database and as such has different optimizations and functionality compared to a relational database. The Realtime Database API is designed to only allow operations that can be executed quickly. This enables you to build a great realtime experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access your data and then [structure it accordingly](https://firebase.google.com/docs/database/web/structure-data).

Figure 4.1. Storing user information in Realtime Database.

**4.2.2 Firebase Authentication:**

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with your custom backend.

**Key Capabilities**:

**FirebaseUI Auth**:

FirebaseUI provides a drop-in auth solution that handles the UI flows for signing in users with email addresses and passwords, phone numbers, and with popular federated identity providers, including Google Sign-In and Facebook Login.

The FirebaseUI Auth component implements best practices for authentication on mobile devices and websites, which can maximize sign-in and sign-up conversion for your app. It also handles edge cases like account recovery and account linking that can be security sensitive and error-prone to handle correctly.

**Firebase SDK Authentication**:

**Email and password based authentication**: Authenticate users with their email addresses and passwords. The Firebase Authentication SDK provides methods to create and manage users that use their email addresses and passwords to sign in. Firebase Authentication also handles sending password reset emails.

**Federated identity provider integration**: Authenticate users by integrating with federated identity providers. The Firebase Authentication SDK provides methods that allow users to sign in with their Google, Facebook, Twitter, and GitHub accounts.

**Phone number authentication**:Authenticate users by sending SMS messages to their phones.

**How does it work?**

To sign a user into your app, you first get authentication credentials from the user. These credentials can be the user's email address and password, or an OAuth token from a federated identity provider. Then, you pass these credentials to the Firebase Authentication SDK. Our backend services will then verify those credentials and return a response to the client.

After a successful sign in, you can access the user's basic profile information, and you can control the user's access to data stored in other Firebase products. We can also use the provided authentication token to verify the identity of users in your own backend services.

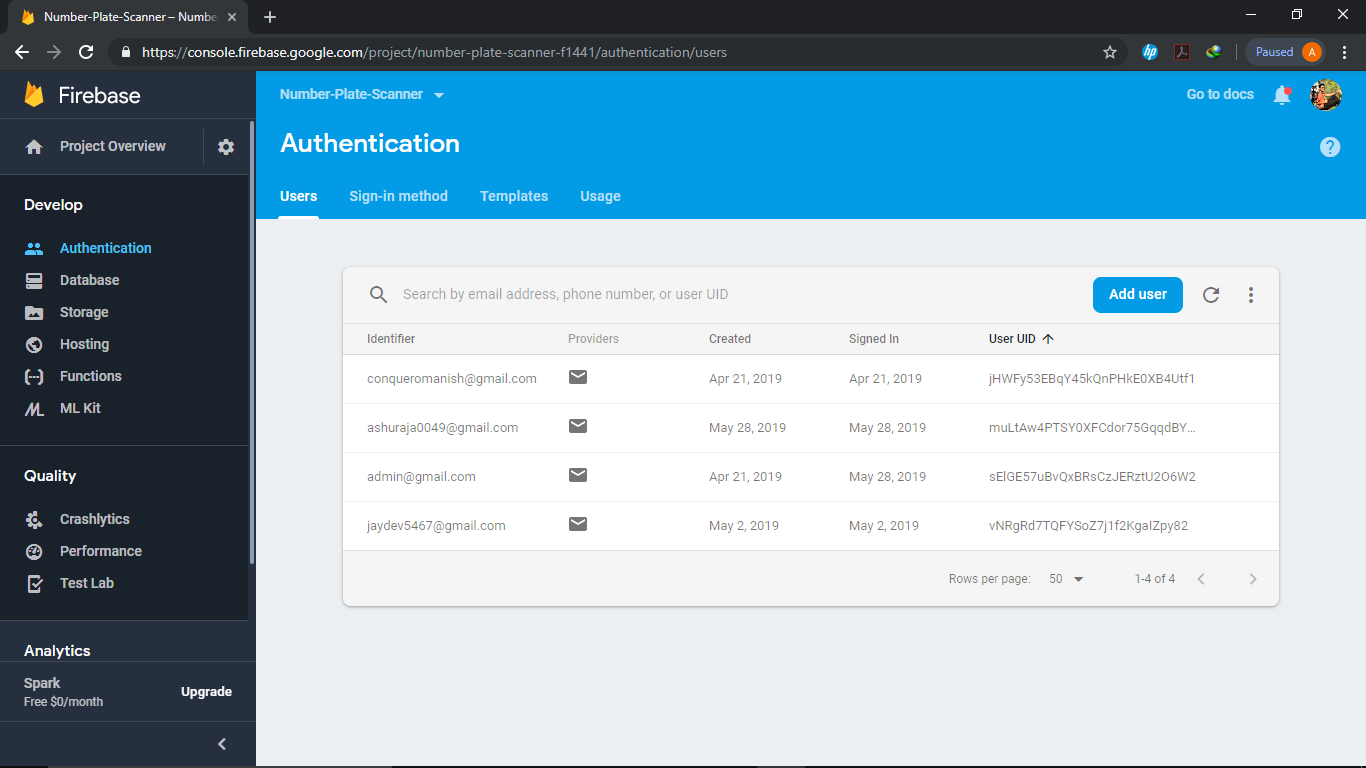


Figure 4.2. Admin authentication page

**CHAPTER 5: DESIGN AND MODELLING**

**5.1 DFD FOR LICENSE PLATE RECOGNITION**

Many businesses use data flow diagrams every day to analyze existing systems to see where road blocks exist and to create new business processes. Data Flow Diagram (DFD) illustrates the flow and transformation of data for a particular business process. It’s a visual representation of how data flows through a system, so you can clearly see where the data comes from, where it goes, how it gets stored. It provides a simple intuitive method for describing business process without focusing on the details of the computer systems. DFDs are attractive technique because they provide what users do rather than what computers do.

Vehicle Details

Image of license plate

LICENSE PLATE RECOGNITION SOFTWARE

0

ADMIN

USER

License holder Information

LEVEL-0

(CONTEXT DIAGRAM)

Image

Unregistered number

Details

Display

0.5

Driver’s details

Fetch corresponding details

0.4

0.3

Valid license

Extracted word

Validation of License number

0.3

Extracted text

Segmentation of text

0.2

Detect Text in Images

0.1

LEVEL -1

**5.2 USE-CASE DIAGRAMS**

## **What is a Use Case Diagram?**

A use case diagram is a dynamic or behavior diagram in [UML](https://www.smartdraw.com/uml-diagram/). Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

**Why Make Use Case Diagrams?**

Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities. They also help identify any internal or external factors that may influence the system and should be taken into consideration.

They provide a good high level analysis from outside the system. Use case diagrams specify how the system interacts with actors without worrying about the details of how that functionality is implemented

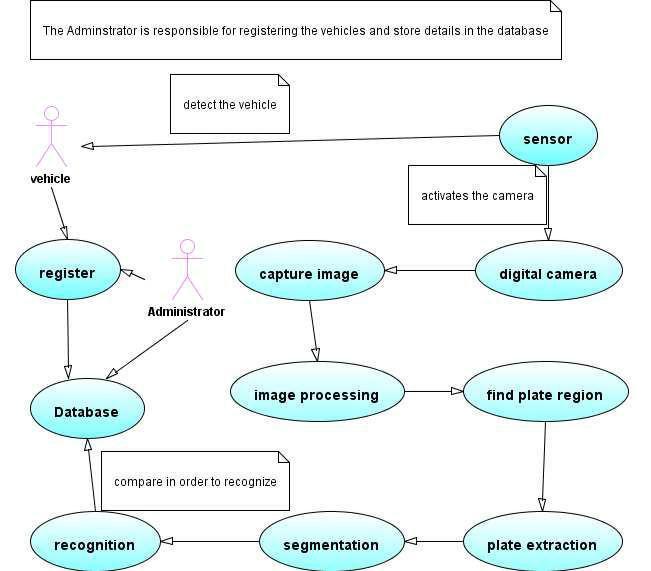


Figure 5.1. Use case diagram for license plate recognition.

**CHAPTER 6**

**APPLICATION OVERVIEW**

**6.1 Overview:**

We propose an efficient automatic real-time license plate recognition framework, particularly designed for the safety and well conduct of automobiles in Institutions or Societies. As common as it is, stored and well-organized data is undeniably, a useful aspect in any kind of work. Thus, with the help of the android application our team has developed, the safe and legal conduct of day-to-day increasing number of automobiles will definitely benefit.

**6.2 Working:**

The Organization will be provided with an Administrator account for the application. The Admin will have access to all the information registered in the app under the same organization name.  

On opening the app, the user will see the home page. On the home page, there are three buttons at the bottom most part namely ‘Home’, ‘Register’ and ‘Our Team’.

**6.3** **Home Button:**

The ‘Home’ button will bring you back to the home page of the app if or when you are doing something else. The user will also find a ‘Scan’ button on the home page, which will access the camera of the device so as to proceed with the capturing of digital image of the vehicle which will be inside the range of the camera. The application uses Text Recognition API to detect the License Plate number from the digital image of the vehicle. Once scanned and detected, the number will be provided with a ‘Fetch’ button below it, which will check and verify the number with real-time database used by the app. After verification of the number, the permission to enter or not will be granted to the owner of the vehicle.

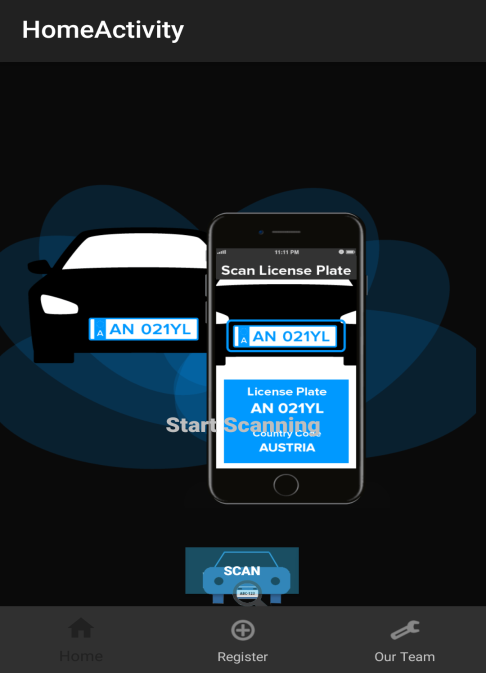


       Figure 6.1. Home Page

**6.4 Register Button:**

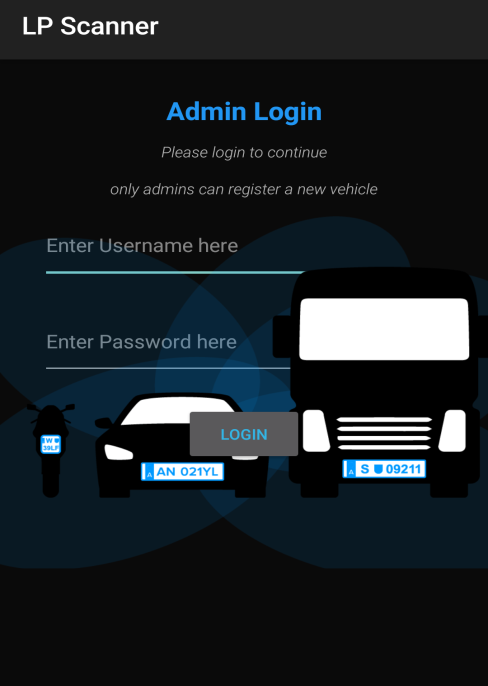
The ‘Register’ button will direct the user to a Login page, where the administrator account given to the organization will be logged in by providing the appropriate Username and Password. After logging in the owner of the vehicles will be required to enter their details in the text fields provided and click on the register button below the text fields.

   Figure 6.2. Admin Login Page

The Vehicle number, the Owner’s name and contact number will be provided by the user and then registered in our database for further use. Real-time database is used with the implementation of Firebase. After being registered successfully, the owner of the vehicle will be notified by a sms sent to their registered mobile number. The sms is sent using smsmanager API.

Default admin username : [admin@gmail.com](mailto:admin@gmail.com)

Password 123456

**6.5 smsmanager API:**

smsmanager API is used to communicate between the users’ messaging application and our application. Its main function is to send messages to the registered participants. Main classes used are Activity and SmsManager. Main methods used are getDefault(), sendTextMessage(), makeText() and getApplicationContext().

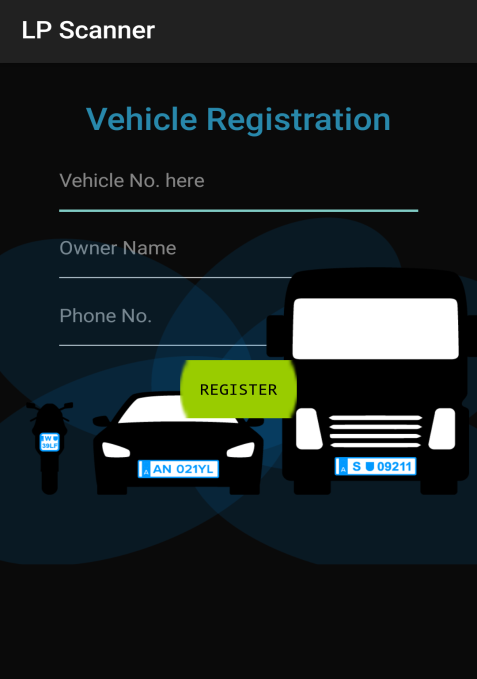


Figure 6.3. Vehicle Registration Page

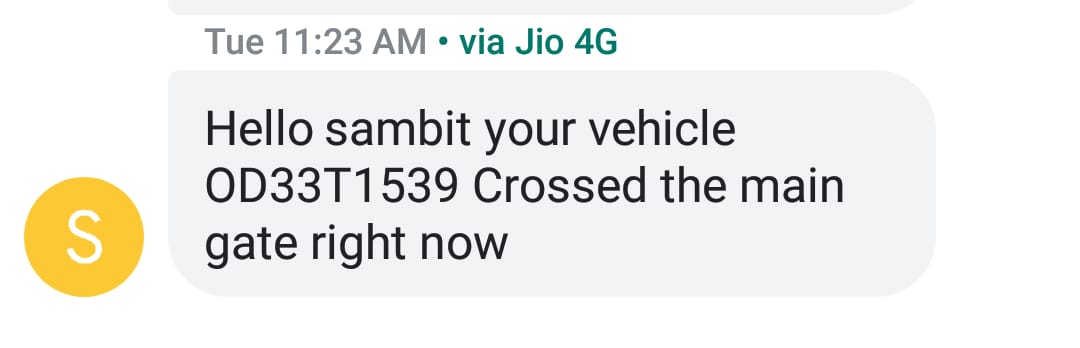


Figure 6.4. SMS received after registration

**6.6 Support and Feedback:**

The last button ‘Our Team’ will direct the user to the support and feedback page of our app. There the user will find the details of the developers of this application and contact our members with the mobile number provided there. There is also a feedback option where the user can send email to any member of the team to give their valuable suggestions which might help us in updating and improving the app further.



Figure 6.5. Support and Feedback Page

The proposed idea is for the guards to possess devices to use this application and scan the vehicles entering or leaving the organization’s area. The app will automatically update the database and will notify the user as well about the status of the vehicle. This way there will always be a count of the number of vehicles inside any campus and notifying the user will let him know the status and thus being certain about the whereabouts of his vehicle.

The application asks for email and mobile number of the users. The necessary information is send to the user through the application and called if necessary. Classes used are Activity and Intent.

The intent fields that are used by the application for a call are:

ACTION\_CALL

The intent fields that are used by the application for an email are:

ACTION\_SEND

EXTRA\_EMAIL

EXTRA\_SUBJECT

EXTRA\_TEXT

**Conclusion**

In this project, we aimed to develop a plate recognition system using OCR in Android Platform .We got images by capturing images through a camera without noise and tried to read characters from that image . Android platform is used because of its availability. Every person nowadays has android cell phone and anyone can use this application.Firebase is a free database service by Google i.e used as a realtime database for storing the information for every vehicle registered .

By implementing this idea we are offering advantages in law enforcement, security and vehicle  access under most ordinary driving condition .We are aiming for a system for secure & gated entrances , for those who need their entrance fully secured, LPR can offer a good benefit.

**Future Work**

**Electronic Toll Collection**

Electronic toll collection aims to eliminate the delay on [toll roads](https://en.wikipedia.org/wiki/Toll_road), [HOV lanes](https://en.wikipedia.org/wiki/HOV_lane), [toll bridges](https://en.wikipedia.org/wiki/Toll_bridge), and [toll tunnels](https://en.wikipedia.org/wiki/Toll_tunnel) by collecting [tolls](https://en.wikipedia.org/wiki/Toll_(road_usage)) without cash and without requiring cars to stop.

Electronic toll booths may operate alongside cash lanes so that drivers who do not have the medium to pay cashless or are not able do a cashless transaction.

As there is always frequent traffic jam at a toll booth area,we can implement the use of License Plate Recognition to identify the vehicle and process a payment request to the registered user .

There must be a speed limit in a toll lane for successful capture of the license plate.

After the License Plate is scanned ,a bill may be mailed to the address where the car's license plate number is registered, or drivers may have a certain amount of time to pay with a credit card by phone.

**In-App Developments**

This application will be implemented in Cross-Platform in near future ,so that it will be easy for all different OS users for successful operation.

Advanced cameras will used for capturing the License Plate Number from a moving vehicle.

The application will also support blurry images or not so good quality images.

More detailed information regarding the number plate will be available.

REFERENCE

[1] https://firebase.google.com/docs/database

[2] https://www.visual-paradigm.com/tutorials/data-flow-diagram-dfd.jsp

[3] [https://www.omicsonline.org/open-access/performance-analysis-of-vehicle-number- plate-](https://www.omicsonline.org/open-access/performance-analysis-of-vehicle-number-%20%20%20%20plate-) recognition-system-using-template-matching-techniques-2165-7866-1000232-101211.html

[4] https://www.abbyy.com/en-eu/finereader/what-is-ocr/

[5] https://www.explainthatstuff.com/how-ocr-works.html

[6] https://developers.google.com/vision/android/text-overview

[7] <https://codelabs.developers.google.com/codelabs/mobile-vision-ocr/#4>

[8] <https://www.slideshare.net>

[9] <https://db-engines.com/en/system/Firebase+Realtime+Database%3BMySQL%3BSQLite>