INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 7, Issue 5 - V7I5-1164)
Available online at: https://www.ijariit.com

Rake- A cross-platform digital product

Ganavi J.
ganavijayaram2050@gmail.com
Altiostar, Bangalore, Karnataka

Amith S. Bharadwaj
amithsb21@gmail.com
Sakha Global, Bangalore, Karnataka

ABSTRACT

Rake is a cross-platform release-ready digital product that is a production-level app, making use of all the latest Firebase technologies. This app uses Firebase MLKit, Firebase Firestore & Firebase Authentication for its core functionality. It also uses libraries like Retrofit, Glide, GSON, and more third-party libraries for other functionalities. Firebase MLKit is used to recognize handwritings and text in images, objects in images and also scan barcodes. This contributes to the main purpose of our app, which labels images and provides the user with context-aware actions of what the user can do next. Firebase Authentication is used to authenticate and help users login & signup in the app. This uses OAuth 2.0 under the hood, so it is a very secure mechanism for login. Firebase Firestore is used as a database to store the user's scanned data so that he can refer to it for later use. We're also using our own custom TensorFlow Lite model, named 'mobilenet', to detect images. The app can be accessed via the web portal as well. Firebase Authentication lets the user login on the web portal, which can be found at rake.now.sh and give the user access to his data on Firebase Firestore. This extended functionality where the user can access his data enhances the ease-of-use and makes sure that the user doesn't always need a mobile device to see his previously-scanned data.

Keywords**:** *Image Classification, Text Recognition, Barcode Scanning, Reverse Image Search, REST API, Android, Web Mining, Cross-Platform, Firebase*.

1. INTRODUCTION

In today's fast-paced world, it's not only important to have a product that runs on a web browser but also one that extends to devices that we carry around in our pockets: our phones.

Because the abilities of a web browser and an application running on it are limited, the need to build a native mobile app for the very same is more than imminent. Probably the most obvious feature that the web platform is missing, which the mobile platform has is something we depend on in our day-to-day usage of phones: Notifications.

We're building a product that is start-up-ready, cross-platform, and one that brings the full immersive experience of the web along with enhanced features from the mobile platform. We are going to implement this by following Design Thinking principles and we aim to make sure that the entire product is based on User-Centered Design.

2. PROBLEM STATEMENT

2.1 Existing System

Most of the current systems are web based and applications such as google lens aren't compatible on devices with lower OS versions. Our product overcomes these compatibility and non-native issues by making it accessible on most of the online platforms. Existing systems use raw data to train models to analyze the image which requires a lot of human effort and time.

2.2 Limitations of existing system

Existing systems have the following limitations.

- Browser abilities are limited.
- Web applications do not utilize much of the native features of the device such as push notifications.
- Most of the current systems are not compatible with mobile devices with lower OS versions such as Android 5.0

2.3 Proposed System

The proposed system architecture (Fig 2.a) will utilize the features of Android devices, Web Platforms and Firebase products for its core functionalities.

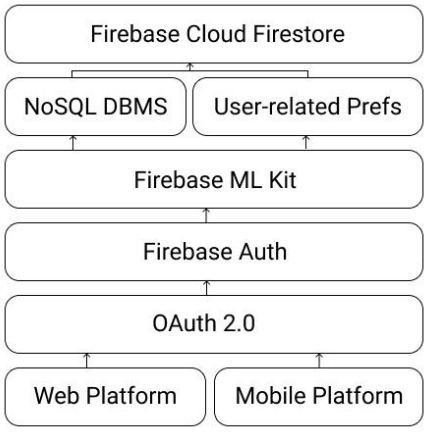

Fig-2.a: Proposed System Architecture.

2.4 Advantages of the Proposed System

The proposed system has the following advantages over the existing systems.

- Reduces the human effort required to manually annotate tags and labels in an image.
- Reduces the human errors that occur when manually annotating tags and labels in an image.
- The reverse image search API uses web mining technologies to crawl the web pages quickly to provide the desired results.
- The users can view and control their data using an easy-to-use dashboard interface designed using a RESTful API.

3. SYSTEM DESIGN

The data flows through our application as shown in Fig 3.a Fig 3.b and Fig 3.c. Users are authenticated via Google Auth Provider. Once authenticated, core functionalities such as text recognition, barcode scanning, image labelling can be performed, and the results are stored with the help of a non-relational DBMS such as Firebase Firestore database. The stored data can be further utilized to recommend similar products or images and provide information about the same.

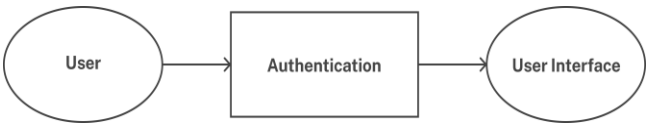

Fig-3.a: Data Flow Diagram for Authentication

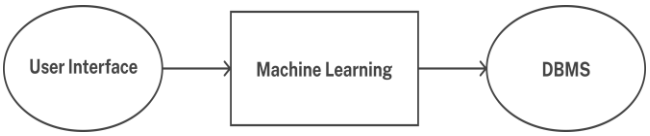

Fig-3.b: Data Flow Diagram for Text Recognition, Image Labelling and Barcode Scanning

Fig-3.c: Data Flow Diagram for Reverse Image Search

4. REQUIREMENTS

4.1 Hardware Requirements

- Desktop/PC - Minimum 4GB RAM
- Android Device – OS v5.0 and above, Minimum 2GB RAM.

4.2 Software Requirements

- Android SDK
- Android Studio
- Web Browsers – Safari, Chrome, IE
- Package Installers – NPM, Yarn
- Node.js Runtime environment.
- Libraries – React.js, Webpack, Babel, css-modules, Flask, Beautiful Soup.

5. IMPLEMENTATION

The image from the mobile camera is converted into a Bitmap or a Byte Buffer before it is sent to Firebase Vision Image for preprocessing. The preprocessed image is then sent to the MLKit model for text recognition / image labeling as shown in Fig 5.a

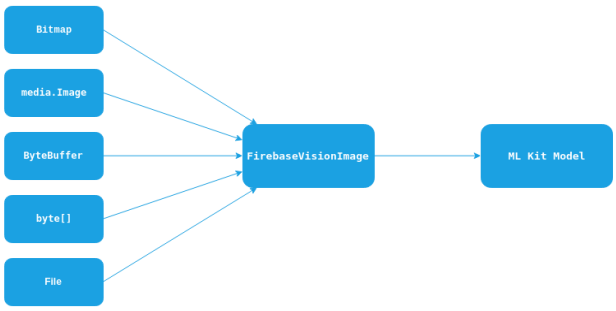

Fig-5.a: Pre-processing of the input image.

MLKit provides both on-device and cloud-based models for Text Recognition. The on-device model uses the text recognizer feature of the user device to recognize text which can be helpful for offline use. The cloud-based model can be instantiated as follows.

5.1 Pseudocode for Text Recognition

```
val textRecognizer =
FirebaseVision.getInstance().cloudTextRecognizer
```

The preprocessed image is then passed to the model for Text Recognition.

5.2 Pseudocode for Processing of the Image

```
textRecognizer.processImage(image).
  addOnSuccessListener {
   // Task completed successfully
  }
  .addOnFailureListener {
   // Task failed with an exception
  }
```

If the text recognition was successful, a FirebaseVisionText object is returned in the success listener. The MLKit's Text Recognizer segments the text into blocks, lines and elements from which the information can be extracted. The image contains zero or more Text Block objects. Text Block objects consists of zero or more Line objects and the Line object contains zero or more Element objects.

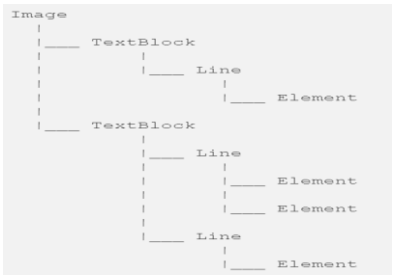

Fig-5.b: Text Segmentation.

For Image Recognition, a custom TensorFlow-lite model named MobileNet is used, which implements a convolutional neural network [3] with several datasets such as Open Images Dataset which includes approximately 9 million annotated images [1]. MLKit provides both on-device and cloud-based models for image labeling. The on-device recognizer does not require an internet connection. The cloud-based recognizer, however, requires an internet connection and is more accurate. The preprocessed image is passed through this model. The cloud-based image labeling API can be instantiated as follows.

5.3 Pseudocode for Label Detection

```
val detector =
FirebaseVision.getInstance().visionCloudLabelDetector
```


The preprocessed image can now be passed for image labeling as follows.

5.4 Pseudocode for Image Labelling

```
detector.detectInImage(image).
  addOnSuccessListener {
    // Task completed successfully
  }
  .addOnFailureListener {
    // Task failed with an exception
  }
```

If the image labeling was successful, the success listener will receive a list of FirebaseVisionLabel Objects.

Each FirebaseVisionLabel object represents the entity that was labeled and contains all the information related to it.

The reverse image search API parses web pages for selected attributes using beautiful soup, a python library used to parse html and web documents. The input can either be one of the labels from image recognition or an image from the device's camera. The results can be stored in the cloud via Firebase Firestore, for further reference using the web application.

6. RESULT/ TESTING ANALYSIS

6.1 Unit Testing

The unit testing results for the Reverse Image Search and the Web Application are shown below.

Table-6.a: Unit testing results for the web application

Function	Description	Expected Ouput	Actual Output
User Authentication	Show Google Auth Pop-up	Show a pop-up where the user can login via google account	Show a pop-up where the user can login via google account
	Show Google Auth Pop-up	Show a pop-up where the user can login via google account	Login failed due to network error or user disabled pop-ups
	Redirect authenticated user to the dashboard	Redirects the user to the dashboard to select the features they want	Redirects the user to the dashboard to select the features they want
	Redirect authenticated user to the dashboard	Redirects the user to the dashboard to select the features they want	Redirect failed due to network error.
Dashboard	Fetch recognized text data	Fetch the required data and render a list of cards with the	Fetches the required data and render a list of cards with the
	Fetch scanned barcodes	Fetch the required data and render a list of images alongside the scanned url, email, UID or phone number	Fetches the required data and render a list of images alongside the scanned url, email, UID or phone number or renders a message saying there is no data.
	Fetch labelled images	Fetch the required data and render a list of images alongside a list of labels	Fetches the required data and render a list of images alongside a list of labels
	Reverse Image Search	Makes an API call with the image as input and renders the response in a dialog box.	Makes an API call with the image as input and renders the response in a dialog box.

Table-6.b: Unit testing results for the reverse image search API.

Function	Description	Expected Ouput	Actual Output
ReverseImage search	Perform image search without header	Request must be in JSON format. Please make sure header is json/application and json is valid	Request must be in JSON format. Please make sure header is json/application and json is valid
	Perform image search with incorrect key	Key Error	Key Error
	Invalid url-incorrect value or non-public url	500- Internal server Error. The server encountered an internal error or was unable to complete your request.	500- Internal server Error. The server encountered an internal error or was unable to complete your request.
	Perform image image search with correct input	JSON output that contains various keys-link to similar images,link to where the product can be bought	JSON output that contains various keys-link to similar images,link to where the product can be bought
Label Search	Perform label search without header	Request must be in JSON format. Please make sure header is json/application and json is valid	Request must be in JSON format. Please make sure header is json/application and json is valid
	Perform image search with incorrect key	Key Error	Key Error
	Perform image image search with correct input	JSON output that contains various keys-link to similar images,link to where the product can be bought	JSON output that contains various keys-link to similar images,link to where the product can be bought

6.2 Performance Metrics

The Android application Rake's performance metrics in comparison with other applications of similar domain can be found in Table 6.c.

Table-6.c: Performance Metrics of the Android Application.

Performance Metrics for Rake Android

Product	Text Recognition Latency	Barcode Scanning Latency	Image Labelling Latency
The-Dagger App	320ms	1160ms	980ms
LabelMe	280ms	NA	781ms
Rake (Our App)	175ms	823ms	450ms

Similarly, the web application's performance can be measured against similar products for factors such as Accessibility, performance, speed index, *and* Time to interactive as shown in Table 6.d

Table-6.d: Performance Metrics of the Web Application.

Performance Metrics for Rake Web App

Product	Accessibility	Performance	Time to Interactive	Speed Index
LabelMe	67/100	97/100	2.9s	2.0s
Diffgram	55/100	55/100	15.7s	9.5s
Rake (Our App)	100/100	70/100	1.45s	3.3s

7. CONCLUSION

Since the product is cross platform, there's no doubt that it is far superior to applications that are only available on web or applications that only available on mobile devices. Cross-platform applications are what makes a product versatile, and convenient for users to use. User retention is higher in such products. Language Recognition is one of the features that can be implemented in the future. The user interface also requires few enhancements. An iOS application is also under consideration. Flutter or React Native can be used to write an application which is compatible with both Android and iOS devices.

8. REFERENCES

[1] Yashaswi Verma, "Image Annotation using Metric Learning" in Semantic Neighborhoods Proceedings of 12th European Conference on Computer Vision, 7-13 Oct. 2012, Print ISBN 978-3-642-33711--6, Vol. ECCV 2012, Part-III, LNCS 7574, pp. 114-128, Firenze, Italy.

[2] Bruce A. Draper and J Ross Beveridge, "Efficient Label Collection for Unlabeled Image Datasets", in Colorado State University Fort, 2017.

[3] Tianmei Guo, Jiwen Dong, Henjian Li, Yunxing Gao, "Simple Convolutional Neural Network on Image Classification", in Department of Computer Science and Technology, Shandong Provincial Key Laboratory of Network based Intelligent Computing University of Jinan, Jinan . 2017.

