2. Internship activities

My supervisor gave me a project in the first day of my internship and I developed this project throughout my internship. Before he gave me the project he talked about software design patterns, mainly MVC(Model-View-Controller) design pattern. They used this approach one of their commercial projects and he said these are important concepts and it will be better if you learn these software design patterns. After he gave me the project I looked through these patterns and used one of them in my project. Now let's look at the project:

- ### Project Description:

Create an Android application that takes an image from the user and searches this image in Google Images and returns the similar images to the input and best guess about the content of the image. Input can be given in 3 different ways: Giving an image url, taking a photo from the camera and choosing and image from the gallery. Similar images must be displayed in a list view and the best guess must be displayed as a string.

- ### Introduction to The Project:

The project described above is named as "**Reverse Image Search**". Google has a service which enables users to search images by images. This concept is very different from searching images by text. While search image by text reads the texts of the websites looks for the input text and returns images accordingly, search image by image looks images of the websites in the internet and compares with input image using machine learning algorithms. We can manually do reverse image search by our browsers. An example input and output of manual search is in the Appendix 1. The task is doing this search through our Android mobile application.

I developed 2 different solutions to this project. First one uses one of the Google's relevant APIs and the second one uses web scraping techniques. But my supervisor said that the web scraping solution is not healthy, because we cannot present the product without getting a valid data from Google. So that I constructed my project using Google Cloud Vision API. First let's look at the GUI of the program:

- ### Graphical User Interface

This is an Android application and in order to tell about the project I should first talk about the Android. I used "**Android Studio**" to create the mobile part of the application. Android uses Java language in the backend. I learned to use Android Studio before my internship and used those skills in this project. Before talking about the backend of the application let's first look at the appearance of the application. The opening screen is in the Appendix 2. There are various buttons and a text in this screen. The wanted functionalities of this widget are:

Enqura Bilgi Teknolojileri A.Ş.
Monumento Plaza Esentepe Mah.
Milangaz Cad. No:75/3 Kartal / İST.
Tel:+90(216)987 05 05 Fax:+90(216)380 60 13
Kartal V.D.: 335 044 3933
Mersis No: 0335044393300018

[illegible]

1. _**"Search by Url" Button**: When the user clicks this button an edit text widget appears and user types the url of the image input.

2. **"Take Photo" Button**: Opens up the camera and after user takes photo, transmits this photo to back end of the program.

3. **"Choose an Image" Button**: Opens up the gallery and after uses selects a photo, transmits it to the back end of the program.

4. **"Upload to Server" Button**: Uploads the image after uses takes or picks one. Server side of the program will be discussed later.

5. **"Search the image" Button**: After the image is uploaded or url is typed, main part of the program runs when user clicks this button. The request to the google is sent and the response is taken. Best guess appears at the top and similar images can be seen by clicking similar images button. If something goes wrong then a message at the bottom saying that an error occurred.

6. **"Similar Images" Button**: After the search is ended, similar images can be seen in a list view by clicking this button.

7. **"Best Guess" TextView:** If the search is successful, best guess appears in this text view.

Now let us elaborate the 1st solution .

- ## Solution to The Project: Google Cloud Vision API

Most reliable and convenient way to achieve this project is to use a relevant Google API for this task. Google Cloud Vision API is the suitable one for this project. It takes an image and returns lots of fields about the image. It returns a big response in JSON response. We are interested in only a small portion of the response, which is displayed in the Appendix-3. Everything seems easy so far but we have a problem. When we look at the documentation of the API, we see a warning: "Cloud Java client libraries do not currently support Android." Screenshot of this warning is in Appendix-3. This warning means that our Android application cannot communicate directly to Google Cloud Vision API. It turns out that if we cannot communicate directly, we must do it indirectly.

Enqura Bilgi Teknolojileri A.Ş.
Monumento Plaza Esentepe Mah.
Milangaz Cad. No:[illegible] Kartal / İST.
Tel:+90(216)[illegible] Fax:+90(216)380 60 13
Kartal V.D.: 335 044 3933
Mersis No: 0335044393300018
[illegible]

- **Server Side:**

  How could we make a request and take a response from API. The answer I came up with is that I must have set up a server which would deliver data between my application and the API. This server must have been able to send and take data successfully with both my app and the API. I chose "**Python**" language and "**Flask**" framework in order to create this server. Python is a convenient language to write code and Google Cloud Vision API supports python language. Flask framework helps to create a web server and it is easy to use. In short our Android app will send image to server, server will make request to API, take a response and deliver it to the Android app and the app will display the output. This is the main usage of the server but it has one more usage.

- **Uploading images to the server:**

  API accepts image urls that must be accessable publicly. But what if user wants to search a local image in his/her mobile phone's hard disk. Those photos are not publicly available. In order to make those images public and give them a web url, we should upload those images to our flask server. Flask supports uploading images. First step to upload a photo to server is to create a folder named static in the server directory. Uploaded photos will be stored in this file. The related code snippet for this task is in Appendix-4.

- **Android Application Classes:**

  We defined the widgets of the application's GUI. Now we will look at the code behind those widgets and the code behind the communication between server and app. All the classes and the methods inside those classes will be examined in detail. First class we will look at is MainActivity.java Class in which the main process runs:

1. **MainActivity.java:**

This is the class where the android application is initialized and governed. All the code of this class is appended to Appendix 5. Firstly let's discuss what external libraries I am used to in this class. We need to send and receive **Http** requests and responses. There are several libraries available to achieve this task, such as **okHttp**, **HttpUrlConnection** and **Volley.** I used Volley library that was developed by Google for Android. It is easy to use and suitable for our task. Lets start to examine this class with onCreate method.

1.1 onCreate:

When the application is started, this method is called. Firstly layout of the opening screen is set and then variables that are defined globally is initialized in this method. Afterwards it is tested that if the device has any camera. Because camera is used as a way of input. If it is not available, then the user is informed by a toast.

Enqura Bilgi Teknolojileri A.Ş.
Monumento Plaza Esentepe Mah.
Milangaz Cad. No: 75/3 Kartal / İST.
Tel:+90(216)987 05 05 Fax:+90(216)380 60 13
Kartal V.D.: 335 044 3933
Mersis No: 0335044393300018

Toast is the short messages that pops up in the screen and disappears after some time. Then we set a click listener object for the "search the image" button. After that button is clicked relevant method is invoked.

1.2 onClickUrl:

This method is called when the "search by url" button is clicked. Right after the button is clicked the button disappears and an edit text field appears to enable user to type image url.

1.3 hasCamera:

This method returns a Boolean value indicating whether the device has a camera or not by using relevant system functions.

1.4 onClickLaunchCamera:

In Android applications, in order to trigger a different process a special class is used called **Intent**. We must create an intent to launch the camera. After the intent is created a special method is called **"startActivityForResult"**. This method launches the camera. Image taken is passed to another method named "**onActivityResult**".

1.5 onClickImagesButton:

This method is called when the "Similar Images" button is pressed. This button is pressed after the search is ended. User clicks this button to see visually similar images to input. Images are displayed in a list view in another class, we will discuss it later. Firstly an intent is created for this class and then an arraylist called "**imageLinks**" which contains the image urls passed along this intent. Then list view class is launched and the arraylist is cleared.

1.6 onClickChoose:

When user clicks "Choose an Image" button, this method is called. Then an intent to the gallery of the device is created and launched accordingly.

1.7 onActivityResult:

Mission of this method is to take the image taken from camera or chosen from gallery and set it to a global variable called "**bitmap**". Bitmap is an instance of class "Bitmap" which is a format for using images. There are two if clauses in this method to detect whether the incoming image is from gallery or camera. Once this is classified bitmap is set to the image. After the bitmap is set, user is informed with a toast.

Enqura Bilgi Teknolojileri A.Ş.
Monumento Plaza Esentepe Mah.
Milangaz Cad. No: 75/3 Kartal / İST.
Tel:+90(216)987 05 05 Fax:+90(216)380 60 13
Kartal V.D.: 335 044 3933
Mersis No: 0335044393300018

1.8 imageToString:

In order to send the image to the server, first I converted the image to a string value. To achieve this, I used a “binary to text encoding” scheme called **Base64**. This method converts the bitmap which is a byte-like object to a string value. The reason for conversion is that it is easy to send a string along with a http request. I could not find a better way, so that I chose this method. Afterwards this encoded image is decoded in the server side by using the decode method of Base64 library in python.

1.9 onClickUpload:

As I mentioned earlier, the reason to upload the image to the server is that Google Cloud Vision API accepts image urls. If we want to get an url for input image, we should put it to a web server. In this method we send our image as a string which is encoded using Base64 to the server. The server returns the name of the image to the application. In this way we can create an url by concatenating the server url and image name. Server side code and the details of this process will be discussed later.

I used Volley library to communicate with the server. The data format that I used to make a request and take a response is **JSON(JavaScript Object Notation)**. In the method I created a json object using the class **JSONObject**. Json data format mainly includes key-value pairs. I put 2 data in my json object: Name and the image string. The volley class to send and receive json data is **JSONObjectRequest**. There are several methods in http communication. The one that I used is **POST** request that supports additional data along with the request. I chose post method because of the fact that we want to send an image. In the constructor of JSONObjectRequest class, we set the method the url of the server, json data object, a response listener, an error listener and a method to set header of the request. We talked about the method and the url of the server will be discussed later. In the response listener method we set take the response and set the global variable **uploadedImageName** to the taken response. We make a toast indicating the error message, in the error listener method. Lastly we put the our headers by **getHeaders** method which indicates that this http request is in json format. Http headers are the information that characterizes the request. The receiver first looks the headers and behave accordingly. After the request is set, it is added to the request queue.

1.10 sendPostRequestAndPrintResponse:

After uploading the image to the server, we are ready to search the image. In this method we send the image url as a json object similar to the description above and take the response. Firstly we put the image url which is obtained by uploading the chosen image to server or entry from the user to the json object we send a volley request to the server. Server returns the best guess and visually similar links as a json object. An example of this data is in the Appendix 3. In order to get the specific data out of this json object we should first parse it.

Enqura Bilgi Teknolojileri A.Ş.
Monumento Plaza Esentepe Mah.
Milangaz Cad. No: 75/3 Kartal / İST.
Tel: +90(216)387 75 03 Fax: +90(216)380 60 13
Kartal V.D.: 335 044 3933
Mersis No: 0335044393300018

In order to parse a json response, we should understand the incoming data. I fetched the needed best guess string and the image links according to their key values in the data. After reaching those values, I set the best guess text view described in the GUI section accordingly. Afterwards I append the image links to the arraylist that is created globally. This arraylist is then passed to the list view class with intent. The structure of list view will be described later.

Different from the upload method, I added a **Retry Policy.** The reason for this is that if the respond does not come in a small time interval, volley throws an **Timeout Error**. To avoid this error I set a retry policy dictating the program wait 10 seconds. If there is no response after 10 seconds then program throws Timeout Error.

After setting all parameters I added this request to the request queue. This is because of the Volley's structure. User is informed in each step of the process whether the process is successful or not by toast messages.

2. **CustomListAdapter Class**

   Android lets user to custom list views apart from its default one. In order to do that I created this class to customize my list view. There is a java class called **ArrayAdapter<T>** helping to create custom adapters. In this class the images are downloaded from the given links and set to the imageView widgets. There is an external library called **Picasso** enables you to do this task in one line of code. It takes an url, downloads it and sets it to the given imageView. All code of this class is available in Appendix 6.

3. **ListView Class**

   This class creates the list view to demonstrate the visually similar images. How the list view is customized is determined my the custom list adapter. An instance of CustomListAdapter class is created and the applied to the list view with the function **setAdapter**. Besides this class handles the incoming imagelinks that is passed along the intent. All code of this class is available in Appendix 7.

4. **RequestQueueSingleton Class**

   Singleton is one of the software design patterns described in the introduction. One example of singleton pattern is this class. Volley requires an request queue for storing requests and managing it. This class handles the incoming requests. All code of this class is available in Appendix 8.

5. **SimilarImages Class**

   While creating a custom list view, we should describe the each item of the list. Every item in the list is an instance of SimilarImages class. For this project it only includes the image itself. However anything could be added to this class and so to list such as text, button, etc. All code of this class is available in Appendix 9.

- ## Server-side Code and Functions

We have made an introduction to server above and now let's dive into the details of the it. As mentioned

Enqura Bilgi Teknolojileri A.Ş.
Monumento Plaza Esentepe Mah.
Milangaz Cad. No: 75/3 Kartal / İST.
Tel:+90(216)987 05 65 / Fax:+90(216)380 60 13
Kartal V.D. 335 044 3933
Mersis No: 0335044393300018

above I used flask framework inside the python programming language to create this server. We have covered how to send requests from android app to server. Now we will examine how server takes the request, makes requests to Google Cloud Vision API and delivers the response to the application. Before talking about the functions of this server. I should mention another topic: **Port forwarding**

While I was creating this application, at first I used my computer's local IP number as the address of server, i.g., **https://192.168.1.45/** . But then I faced with a problem. When I make a request from application to server I got successful, however making request to google resulted with failure. Because this is my local IP, it is not a public address so that Google cannot access. The solution I came up with was port forwarding. It is the concept of making your local server publicly available. When I asked this issue to my supervisor he talked about several tools which enables port forwarding such as **ngrok** and **ssh tunneling serveo.net**. Usages of these tools are very similar. Both of them can be run in command line interface by passing the port number as an argument The way I run those tools are listed below:

- **ngrok: ./ngrok htttp 5000**
- **serveo: ssh -R 80:localhost:5000 serveo.net**

After those commands are run program gives a publicly available url. I used the given url as an address of my server. e.g. **https://peius.serveo.net**. Subdomains of the server can be reached by concatenating this address with "/" character. e. g. **https://peius.serveo.net/upload , https://peius.serveo.net/search or https://peius.serveo.net/<image_name>.jpg**.

There are several functions in the server code. All code of these functions is available in the Appendix 10. Lets explain those functions:

1. **Main Function:**

   Main function is one that is called firstly when program starts to run. In this function firstly we parse the argument given to the program. To do this, I imported **argparse** library and added one argument using its **add_argument** function. The argument to the program is the port number on which the server runs. Default port number is 5000. After the argument is parsed app is started with the host address "0.0.0.0" and the given port number. Once the server begins to run, it starts to listen for requests in the given port.

2. **name_generator Function:**

   After we uploaded images to server, those images go the static folder which is declared globally. We need to create unique names for these images. Taking image names from the user is unnecessary and inconvenient. Instead I created this name_generator function in which a random string is generated and returned. I used the library called **random** to achieve this task. This function is called in the upload function and the return value is set as the name of the uploaded image.

3. **send_url Function:**

   This method is used in creating image urls. In detail, flask has a function called **send_from_directory**

Enqura Bilgi Teknolojileri A.Ş.
Monumento Plaza Esentepe Mah.
Milangaz Cad. No: 75/3 Kartal / İST.
Tel: +90(216)987 65 65 Fax: +90(216)380 60 13
Kartal V.D.: 335 044 3933
Mersis No: 0335044393300018

which looks the static folder in the root directory of the server and makes the static files publicly available. An example of this is **https://peius.serveo.net/<image_name>.jpg**. This is a very convenient way to reach images.

4. **upload Function:**

This function handles the incoming upload requests from the application. At first it looks the headers of the post request. As mentioned above, headers contains descriptive information about the request. The function first checks whether the incoming request is an json request or not. If not it returns a feedback message saying that it should be a json request. After this control function takes the json data and converts it a python dictionary which is a data structure containing key and value pairs. It is very similar to "Map" in the Java language. After this conversion it reaches the encoded bitmap value using the key "bitmap". This is an encoded string value which represents the image. The function decodes this string value to an image file using the decode method of the Base64 library in python. Then the upload folder whose path is defined globaly is opened and the decoded image file is written to this folder. It is important that after the conversion the data is a byte-like object, not a string. So that we opened the upload folder with the argument "**wb**" indicating that we will **W**rite a **B**yte-like object to this folder. Before putting the image to the folder I created a name for the image using name generator function and set it to a variable called **currentName**, so that I can be able to reach the image with its name. This name is returned to the application as json data.

5. **search Function:**

This function is called when the **/search** subdomain is invoked. It checks whether the incoming request is a json request or not similar to the upload function. Afterwards it converts the json data to python dictionary and passes it to the leading function called **vision_image_manager** and returns the incoming answer to application as json data.

6. **vision_image_manager Function:**

This is the function which communicates with the Google Cloud Vision API. In this function I used several libraries of Google called **vision, types** and **discovery.** In the documentation of Google API, they instructs how to send a request and take a response from the API. I used those instructions and edited them suitable for my purpose. First of all, API returns a big json data including lots of features about the image. However I just need two of these features: Visually similar images and best guess. Therefore I shaped my request according to this purpose. The features that I said is under the **web detection** section of the data and I wanted just this section from the API. The request is made using the image url as described earlier and response is taken in json data format. This json data is parsed in search function. Google Cloud Vision API is free in 1000 requests per month but in any case it requires to be recorded to Google and take an API key. My supervisor took an API key and gave me. In order to make authorization I had to set an environment variable. I did this with the following command:

export GOOGLE_APPLICATION_CREDENTIALS=PATH_TO_KEY_FILE

[illegible]

Enqura Bilgi Teknolojileri A.Ş.
Monumento Plaza Esentepe Mah.
Milangaz Cad. No:75/3 Kartal / İST.
Tel:+90(216)387 15 00 Fax:+90(216)380 60 13
Kartal V.D.: 335 044 3933
Mersis No: 0335044393300018