Bilkent University

Department of Computer Engineering

Senior Design Project

SEPS

Final Report

Taner Durmaz, Samir Ibrahimzade, Mehmet Erkin Şahsuvaroğlu, Alperen Koca, Burak Yeni

Supervisor: Shervin Rahimzadeh Arashloo
Jury Members:

Final Report and User's Manual

April 30, 2021

1. Introduction

This report presents the requirement details, final architecture design and its details, development / implementation elaborations, testing details, maintenance plan details and comments on conclusion and future work of the project SEPS.

To briefly re-introduce the “social event photo sharing” project, SEPS (Social event photo sharing) is a mobile application which creates a comfortable environment for the event organizers and attendants to share and access the photos of certain events while securing their privacy. SEPS aims to enable users to share their photos in a social media environment, while taking advantage of cutting-edge security technologies, such as QR protocols and image processing solutions.

2. Requirements Details

2.1 Functional Requirements

- User can create individualized accounts with their email address.
- User can login with their email or username, and password.
- User can logout.
- User can create events and send invitations to other accounts.
- User can send QR code to event participants of the created event of the user.
- User can display QR code from their screen.
- User can upload photographs to a database.
- User can tag photographs with words and events.
- User can access photographs taken during the event they participated in.
- User can download accessible photographs.
- User can access the information about the event they are participating in.

2.2 Nonfunctional Requirements

Privacy: The non attendant users will not be able to upload and access to event photos.

Security: Users are required to be authorized by QR protocol in order to be tagged to an event tag.

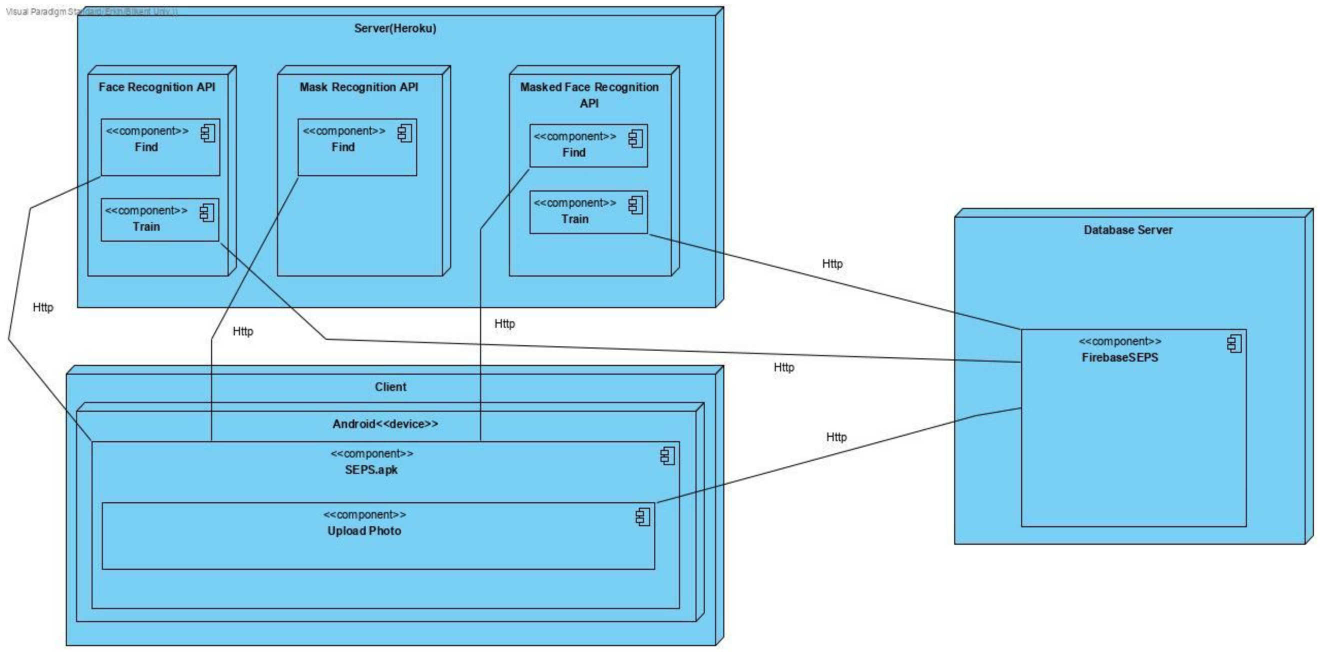
Reliability: The untagged photos, which are also associated to a particular event by a machine learning model, will be offered to attendant users of the particular event.

3. Final Architecture and Design Details

Our architectural style has not changed but we needed to do some alterations for subsystems of our architecture. We selected client/server architectural style as final. We used the Java language for Android devices to develop the client side of our application. In the client side we have all the UIs of our application. On the server side, we used the Heroku platform. Our APIs are working on the Heroku cloud server. For storing data of our users we used Firebase realtime database to keep sustainability of our database related interactions.

3.1 Overview

Our design and architecture overview consist of three parts: Heroku server, mobile application and database server. Since the database and APIs work on different servers we divided the server side into two.



3.2 API

SEPS uses algorithms which use gpu for image recognition tasks. Thus server - client architecture is a need, SEPS serves servers data via API since the end user is not directly involved with servers. Documentation about how to use API as follows, Server will be deployed on Heroku.com and will be written with python3. Choice of library is flask.

3.2.1. Face Recognition API

Route: https://seps-api-face.herokuapp.com

Methods:

train(url image, profile id): Train model with image in url and profile id. Returns success or failure.

Format: https://seps-api-face.herokuapp.com/train/**<url>**/**<id>**

Example request:

https://seps-api-face.herokuapp.com/train/https://seps.page.link/Qy7XGPe6NN93JiMt5/1002

find(url image): Find which profile image belongs to. Returns json object with profile id and face location.

Format: https://seps-api-face.herokuapp.com/find/**<url>**

Example request:

https://seps-api-face.herokuapp.com/find/https://seps.page.link/Qy7XGPe6NN93JiMt5

GitHub: https://github.com/tanerdurmaz/seps-api-face-recognition

3.2.2. Mask Recognition API

Route: https://seps-api-mask.herokuapp.com

Methods:

find(url image): Finds face in photo has mask or not then returns “Mask” or “No Mask”

Format: https://seps-api-mask.herokuapp.com/find/**<url>**

Example request:

https://seps-api-mask.herokuapp.com/find/https://seps.page.link/Qy7XGPe6NN93JiMt5

GitHub: https://github.com/tanerdurmaz/seps-api-mask-recognition

3.2.3. Masked Face Recognition API

Route: https://seps-api-masked-face.herokuapp.com

Methods:

train(url image, profile id): Train model with image in url and profile id. Returns success or failure.

Format: https://seps-api-masked-face.herokuapp.com/train/**<url>**/**<id>**

Example request:

https://seps-api-masked-face.herokuapp.com/train/https://seps.page.link/Qy7XGPe6NN93JiMt5/1002

find(url image): Find which profile image belongs to. Returns json object with profile id and face location.

Format: https://seps-api-masked-face.herokuapp.com/find/**<url>**

Example request:

https://seps-api-masked-face.herokuapp.com/find/https://seps.page.link/Qy7XGPe6NN93JiMt5

GitHub: https://github.com/tanerdurmaz/seps-api-masked-face-recognition

4. Development/Implementation Details

4.1 Mobile Application

The Android application was implemented by using Android Studio IDE which is built on JetBrains' IntelliJ IDEA. It is the tool that is built with the aim of only to operate for Android development. Java language was used to create an android app and Extensible Markup Language (XML) was serviced to build the user interface.

Activities and Fragments

Activities are the one focused thing that users can do in the Android operating system. It enables the user interface of the according pages by its setContentView(View) method. Compared with activities, fragments cannot be created without host activity or another fragment. They help to improve modularity of the application by its reusability and enabling to manage all parts of the screen. Therefore, fragments help to adjust the UI according to the different screen sizes.

In the SEPS mobile application, we used MainActivity to put bottom navigation, action bar, settings icon and profile items, which are the global components of the app's UI. It hosts three fragments: MyEvents, JoinEvent and CreateEvent, which can be navigated by bottom navigation.

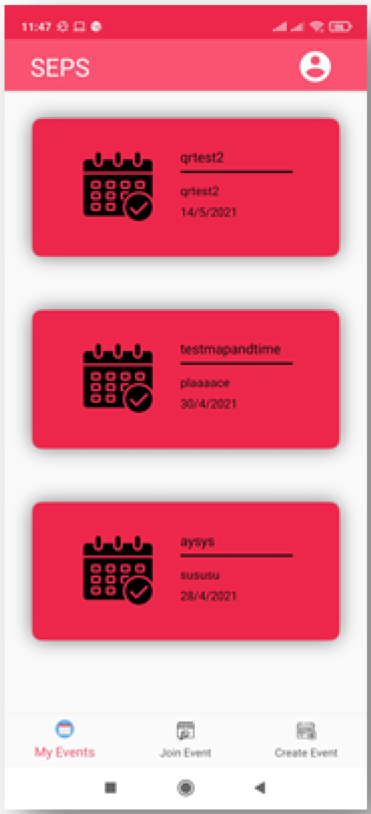
In addition, the app contains Login and Register activities for managing user authentication and PhotosActivity to handle the recycler view with the event gallery.

MainActivity

MainActivity is the host activity for MyEvents, JoinEvent and CreateEvent fragments. It also includes a bottom navigation menu and action bar, the elements that can be accessed from each page.

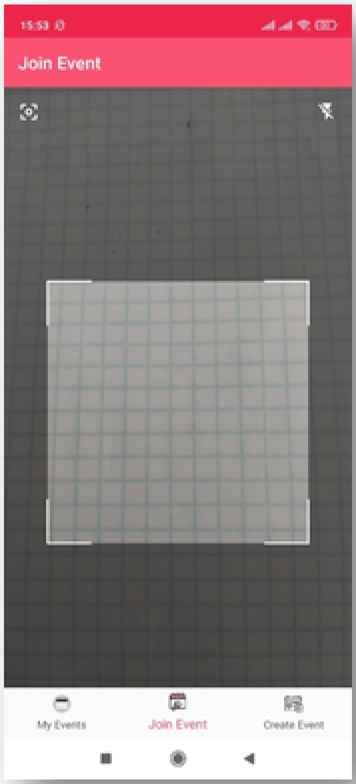
MyEventsFragment

This fragment contains the recycler view, which shows all the events of the current user. The recycler view items are clickable and redirects users to according event's page.



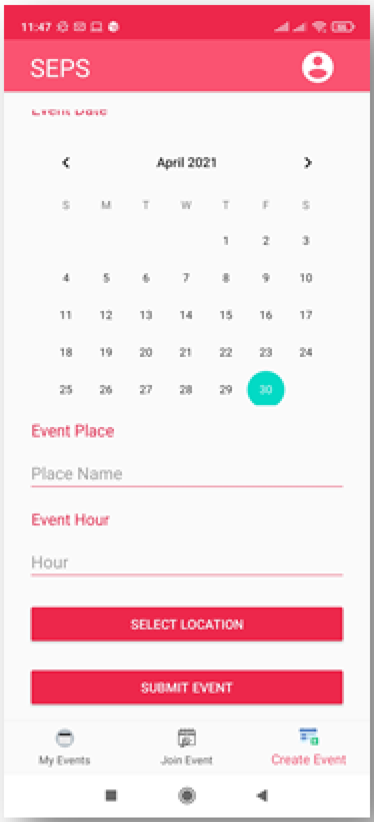
JoinEventFragment

This fragment shows the view from the camera, it waits for the user to scan the QR code of the event.



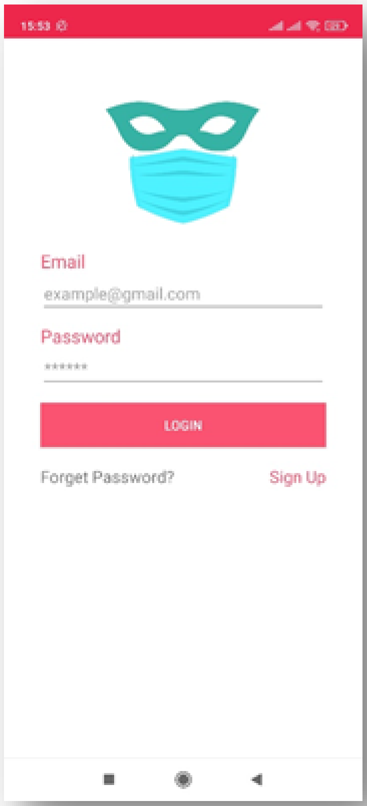
CreateEventFragment

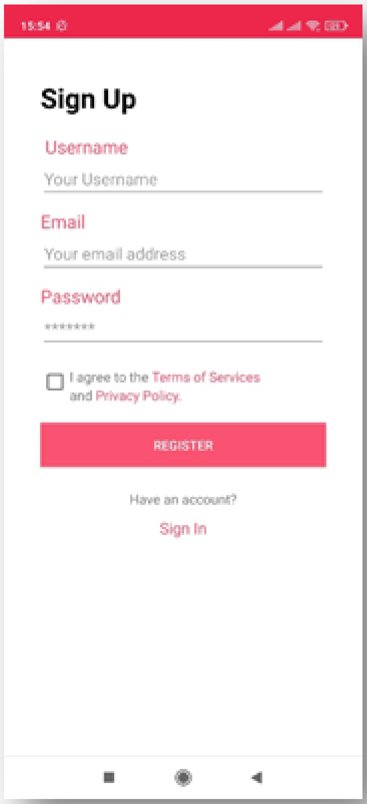
This fragment contains edittexts and calendarview; it waits for the user to fill the edittexts and submit the event.



Login and Register Activities

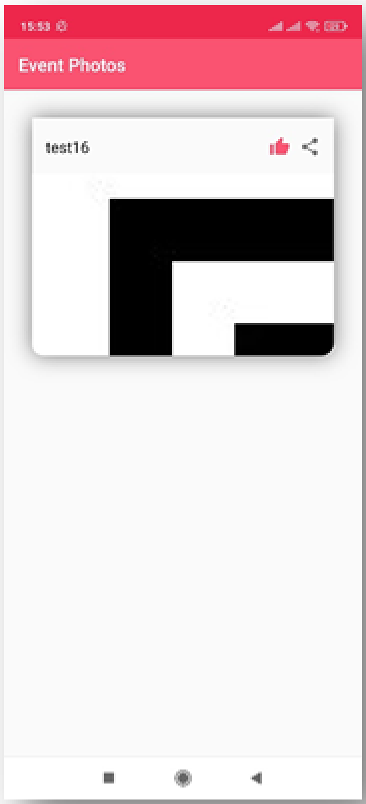
These activities are aimed to get the data from user to respectively sign in or sign up the user. The user has to check the box that confirms the privacy policy had been read in order to register. In addition if the user cannot remember the password, the forgot password option is included.





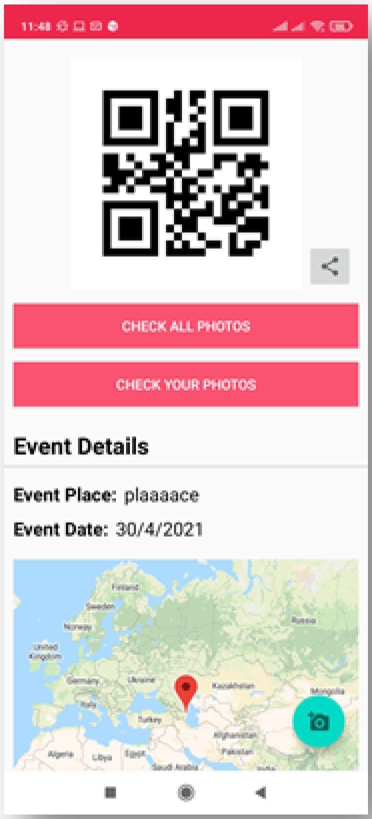
PhotosActivity

This activity contains event photos (inside event Photo relative layout) which are placed in recycler view. The user can like or share the photos.



EventActivity

Event Activity contains QR code of the event, share button for QR, check all/check user's photos button, event details and upload a photo button. The creators of the event also see the button for editing event place and date.



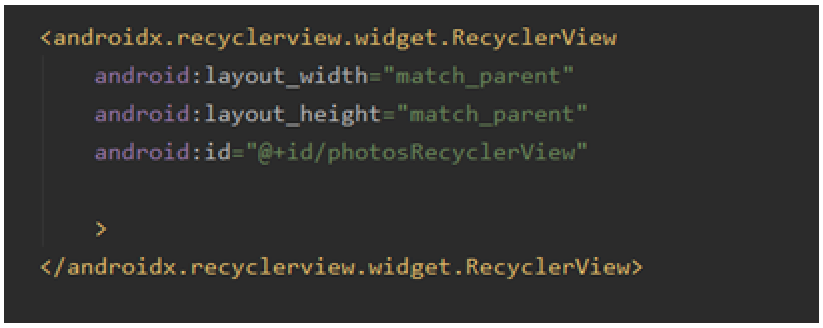
Event Details

Event Place: plaaaace

Event Date: 30/4/2021

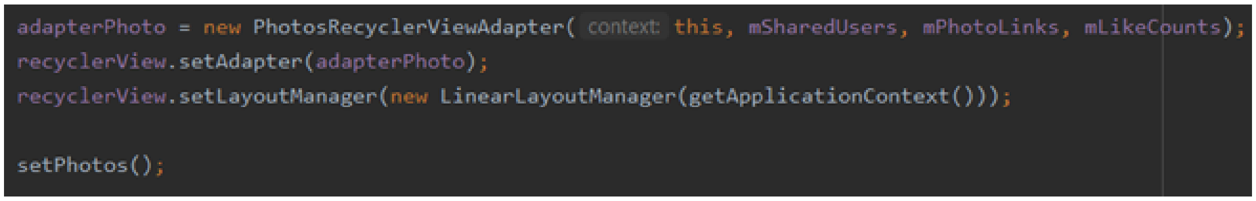
RecyclerView – Adapter

RecyclerView was used in SEPS to store the list items in MyEvents fragment and Photos Activity. The reason for deciding to use RecyclerView instead of List View was the need of keeping dynamic lists. To implement the view, we first created its adapter file and added its UI components, consequently we created a RecyclerView object and we set it to its adapter.



```
<androidx.recyclerview.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/photosRecyclerView"

    >
</androidx.recyclerview.widget.RecyclerView>
```

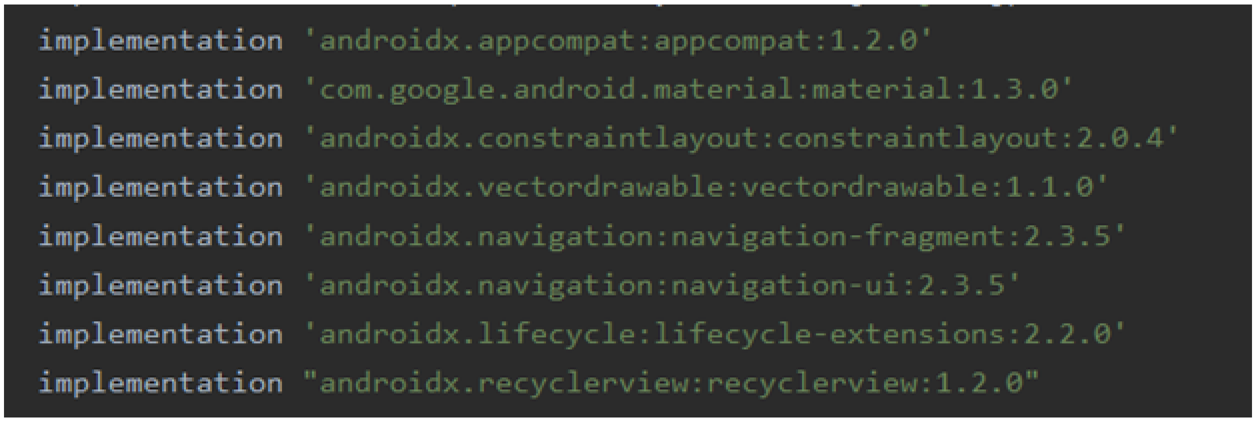


```
adapterPhoto = new PhotosRecyclerViewAdapter( context: this, mSharedUsers, mPhotoLinks, mLikeCounts);
recyclerView.setAdapter(adapterPhoto);
recyclerView.setLayoutManager(new LinearLayoutManager(getApplicationContext()));

setPhotos();
```

Libraries

- We used androidx libraries to add android view components to the user interface of the application.
- For creating QRs and scanning them we used the zxing library[https://github.com/zxing/zxing].
- To have more customizable and user-friendly toasts we used pranavpandey dynamic toast library[https://github.com/pranavpandey/dynamic-toasts].
- We used a fresco library – which is created by Facebook – to manage the images in the UI [https://frescolib.org/docs/].
- To enable cropping the image before uploading we used theartofdev library [https://github.com/ArthurHub/Android-Image-Cropper].

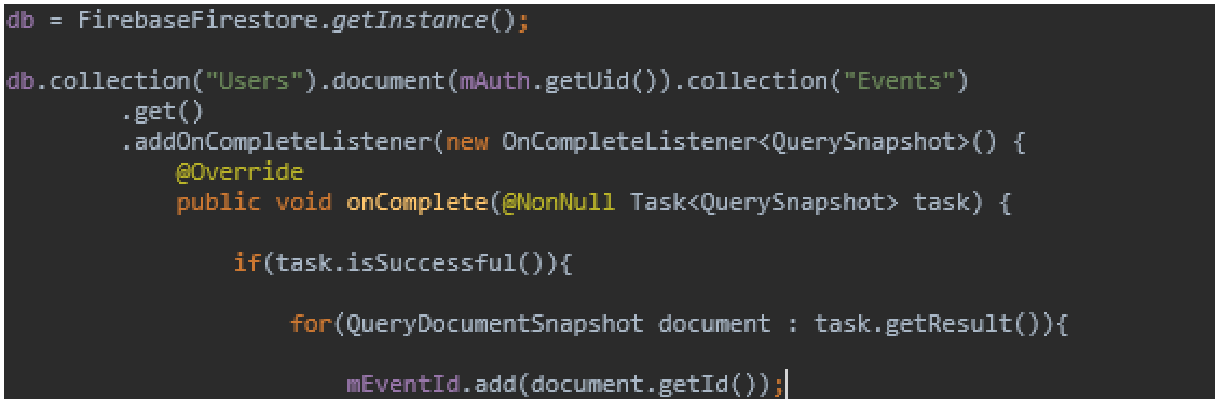


```
implementation 'androidx.appcompat:appcompat:1.2.0'
implementation 'com.google.android.material:material:1.3.0'
implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
implementation 'androidx.vectordrawable:vectordrawable:1.1.0'
implementation 'androidx.navigation:navigation-fragment:2.3.5'
implementation 'androidx.navigation:navigation-ui:2.3.5'
implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
implementation "androidx.recyclerview:recyclerview:1.2.0"
```

Firebase

To store the photos and user/event data we used Firebase, which is Google's Backend-as-a-Service (BaaS). Our current plan is Spark Plan – includes free A/B testing, analytics, app distribution and app indexing. It permits to write 20K/day and read 50K/day documents and it gives 10 GB storage with 360 MB/day data transfer [https://firebase.google.com/pricing?authuser=0]. Therefore, after investigating Firebase's Spark plan and concluding that it fits all SEPS's requirements, we decided to use it.

- The Authentication was used for storing the user data.
- The Firestore was used for storing the event data.
- The Storage was used for storing the uploaded events.



```
db = FirebaseFirestore.getInstance();

db.collection("Users").document(mAuth.getUid()).collection("Events")
        .get()
        .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
            @Override
            public void onComplete(@NonNull Task<QuerySnapshot> task) {

                if(task.isSuccessful()){

                    for(QueryDocumentSnapshot document : task.getResult()){

                        mEventId.add(document.getId());
```

4.2 Server & Machine Learning

The face recognition task aims to identify whether an inputed face image belongs to some users of the SEPS. To do it, SEPS compares the input image with a face schema which was constructed by previous images.

Python was used during the implementation and integration phase. The used libraries include libraries used by the API's and models. The "Heroku" API is used for cloud and server operations on the machine learning side. The retrieval of uploaded images are done via Heroku and Firebase API's, by accessing them by their corresponding url's.

Face Recognition API's

Machine learning architecture utilizes pre-trained models for decision-making. The implementation of the machine learning side uses plug-in API's for model operations. Usage and update status of the learning model is managed by three API's, namely "Face Recognition API", "Mask Recognition API", and "Masked Face Recognition API" . These two distinct modules were trained by using corresponding datasets, which can also be found at glossary.

Face Image Processing Modules

In order to parse face-images from a given picture, SEPS ML architecture rectangles all the faces within a picture. The pictures are already cropped by the application side when they arrive to the server from the firebase database. For detecting each face structure SEPS uses single shot detection(SSD). All the extracted faces within a picture are inputed to mask and face recognition models respectively. There are train and find functions for all the modeles. The architecture of the used learning network is 300x300 res10 DNN. The preference for using these API's and models is our assumption such that it would be faster by these compact blocks. There were also implementation concerns on workload and necessity.

Utilization

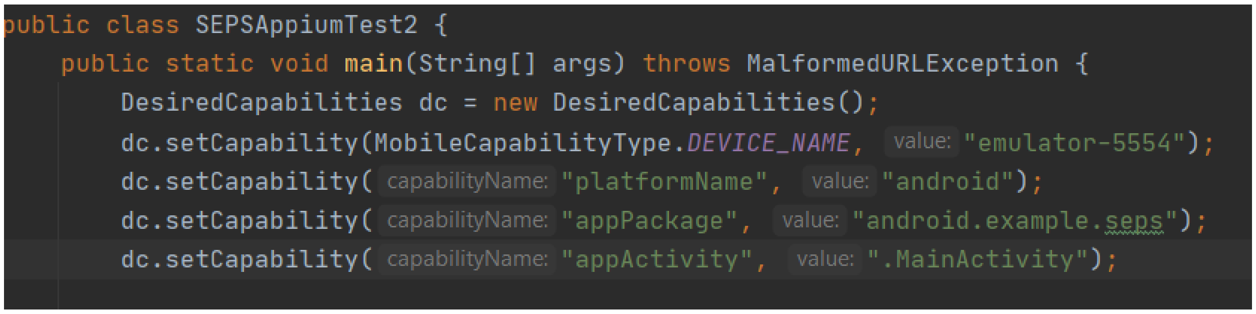
A single model is trained by all the face images and corresponding labels of names, surnames, etc., for each purpose. If an input image becomes recognized as a face of a user, and the user is the same as the uploader; then the system of SEPS awards a badge for implying that the image is properly uploaded. There is also a "mask" badge, indicating that all the faces within a picture are masked. These operations are done by "find" requests for the input images. The badge awarding logic is also handled by the server side, outputting a message to be used by database and users. This output contains the meta-data for an uploaded image, with the mask and face verification information.

5 Testing Details

Testing is a very important aspect of software development. End product must satisfy customers which is one of the most crucial stakeholders in the whole process. Since we have developed an Android application we selected blackbox and whitebox testing. Our blackbox testing is done by automation testing. Testing the functionality of our application depends on the scenario that is conducted automatically. In order to do whitebox testing we picked unit testing and these activities were done manually.

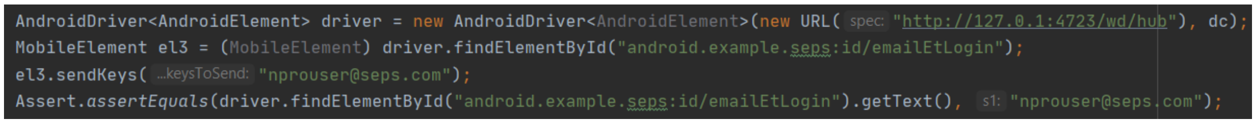
5.1 Black Box Testing

Blackbox testing is a very necessary technique to do verification of the application. In this technique, we test functionalities of our program and we can see our application from the perspective of our customers. UI features and performance can be traced well via using this method. To conduct blackbox testing, we decided to use Appium which is an automation tool for mobile applications. We used Java for the test codes. We have imported Appium and Selenium packages to conduct our automation test. Therefore we used a virtual Android device from Android Studio. Thus we wrote device capabilities and features in our Java code. Finally with Appium methods we implemented our test scenario in Java code. By running this code all functionalities will be tested automatically. At the end our assertions will be displayed in the console. Automation testing will reduce the time consumption if we have numerous functionalities in the future by comparison to manual testing.



```java
public class SEPSAppiumTest2 {
    public static void main(String[] args) throws MalformedURLException {
        DesiredCapabilities dc = new DesiredCapabilities();
        dc.setCapability(MobileCapabilityType.DEVICE_NAME,  value: "emulator-5554");
        dc.setCapability( capabilityName: "platformName",  value: "android");
        dc.setCapability( capabilityName: "appPackage",  value: "android.example.seps");
        dc.setCapability( capabilityName: "appActivity",  value: ".MainActivity");
```

Figure: Desired capabilities of our virtual test device

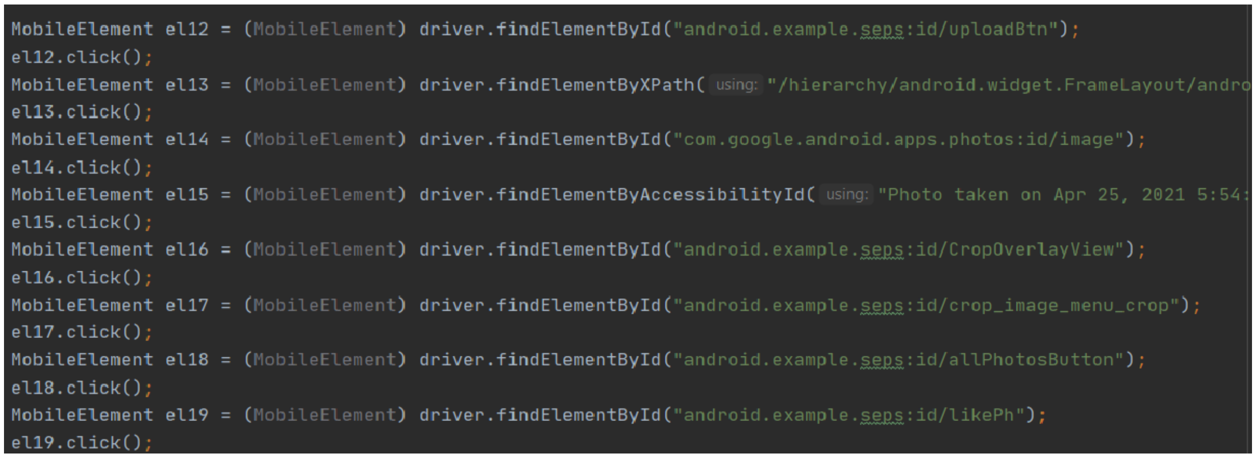


```java
AndroidDriver<AndroidElement> driver = new AndroidDriver<AndroidElement>(new URL( spec: "http://127.0.1:4723/wd/hub"), dc);
MobileElement el3 = (MobileElement) driver.findElementById("android.example.seps:id/emailEtLogin");
el3.sendKeys( ...keysToSend: "nprouser@seps.com");
Assert.assertEquals(driver.findElementById("android.example.seps:id/emailEtLogin").getText(),  s1: "nprouser@seps.com");
```

Figure: Sample automation run for logging to SEPS

In the figure above, we created an android driver which is named as “driver”. Therefore by finding an element by id in the main activity page our program fills the user email in the desired blank. Then by Assert class in the TestNG library we check whether the entry is valid.

This is our automation code for uploading images to the selected event. First, the upload button is clicked then the image will be taken from the photos folder from our emulator. This image will be cropped then will be uploaded. After that automation process will be continued with clicking all photos button in the event screen then the uploaded photo will be liked.



```
MobileElement el12 = (MobileElement) driver.findElementById("android.example.seps:id/uploadBtn");
el12.click();
MobileElement el13 = (MobileElement) driver.findElementByXPath( using: "/hierarchy/android.widget.FrameLayout/andro
el13.click();
MobileElement el14 = (MobileElement) driver.findElementById("com.google.android.apps.photos:id/image");
el14.click();
MobileElement el15 = (MobileElement) driver.findElementByAccessibilityId( using: "Photo taken on Apr 25, 2021 5:54:
el15.click();
MobileElement el16 = (MobileElement) driver.findElementById("android.example.seps:id/CropOverlayView");
el16.click();
MobileElement el17 = (MobileElement) driver.findElementById("android.example.seps:id/crop_image_menu_crop");
el17.click();
MobileElement el18 = (MobileElement) driver.findElementById("android.example.seps:id/allPhotosButton");
el18.click();
MobileElement el19 = (MobileElement) driver.findElementById("android.example.seps:id/likePh");
el19.click();
```

Figure: Sample photo uploading automation process

5.2 White Box Testing

Whitebox testing is the technique of verifying source code blocks working well enough. There are two types of whitebox testing one of them is unit testing and other one is memory leak testing. Since we used Java and xml languages we did not care much about memory leaks. We used unit testing with code reviews as a combination. Our code blocks reviewed one by one and for each code block we conducted a unit testing. We checked coverage of the conditions and complexities of our algorithms. This test will help our application work efficiently if our application reaches excessive users.

6. Maintenance Plan and Details

Maintenance is essential for SEPS application because SEPS depends on both server and model dependent application. Since we use face and mask recognition models and event handlers in our application, we must keep our server stable to prevent possible problems such as storage problems or model training problems. Meanwhile, we must also keep our android application up to date to ensure stable connections with both phone and network. For this neccesities of maintenance, we divide the maintenance into 3 parts:

6.1 Application Maintenance

Our application is on an Android platform so there are a lot of different combinations of Android phones used and each has different screen size, hardware types and softwares so there may be bugs about UI, connection and compatibility. We have to make sure to fix the possible problems and make the application stable to use.

6.2 Server Maintenance

We have an application that is heavily based on an online database, Heroku. We use a free version of it and this gives us limited low storage, 500 MB. This can be a problem if the user database increases in time, so we could move to a different database that gives us bigger storage. Heroku also gives us high latency and no opportunity to restore data. This situation is enough right now, but this is not enough for an increased number of users, so we may change the database to keep the application stable.

6.3 Model Maintenance

Since our application uses the trained models of face and mask recognition, we have to make sure that the model works properly in different conditions. There can be camera related problems, light problems, weather related problems of photos so this could cause the malfunction of our models. We need to keep our models stable to make sure that they work stable and give precise results under extraordinary situations.

7. Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

Public Health

- The application's face recognition model should take COVID-19 mask regulations into consideration.
- Public safety The application should guarantee security of the event zone by the usage of QR code.

Global Factors

- The application’s implementation will take global trends on software solutions into account. The most trending photograph sharing applications will be analyzed in order to ensure the quality desired by users.

Cultural Factors

- The application will provide users with the opportunity to easily find photographs that they do not even know but they are in. Moreover, the users will not search for their photos from other attendants, they will be able to access them directly via the application. The face recognition algorithm should be unbiased for any kind of race, religion and other cultural values.

Social Factors

- The application should use ethical constraints for determining uploadable photographs. Any abusive usage should be reported and deleted from the database. 18Environmental Factors The devices being used in the design process should be efficient in terms of electric consumption.

Economic Factors

- The application should be free of charge.

Factors	Priority Point(out of 10)
Public Health	4
Public Safety	10
Global Factors	6
Cultural Factors	7
Social Factors	5
Environmental Factors	3
Economic Factors	2

7.2 Ethics and Professional Responsibilities

- In the ethical scope, our system respects the privacy of users, so our system must keep and protect confidential and private information of users. Since our program collects the photos which are the most valued information of the users, the program must store these photos privately, and must not make these photos available publicly without permission of users who are in photos.[1]
- Users are stakeholders in computing ethics, so the stake of every user must be kept and protected by our program. In some cases, users may accidentally log in the wrong event and then he/she realises that situation and log out from the event; if this kind of situation occurred, the system would delete this user's information from the event. By doing that, our system protects the private information of the users.[1]
- In ethical view, the users must be kept distant from harm caused by computer professionals. In some cases, if any threat or action happens during the event that may cause harm to the user's information, users would inform the system; according to this information, the system will inform other user's in this event. [1]
- In order to respect the privacy of users, users won't be asked to get irrelevant information. Only the necessary information will be asked to users in our program.[1]

7.3 Judgments and Impacts to Various Contexts

SEPS aims to decrease the chaos while getting photos in social events, and unite people who attend the same event in one roof to access their photos. In order to use our application, people must sign up, join or add events to the app. If it is noticed that the number of users of our application starts increasing, more and more people will use it. Today, people often take photos during events and they also want to share these photos so they must feel to use SEPS to share photos among themselves easily. This is the main idea behind SEPS.
Our judgement: People should feel to use SEPS to easily share the photos taken during the event.

- Impact in Global Context is high because SEPS is generated to be used anywhere in Worldwide.
- Impact in Economic Context is low because users already do not spend money on getting photos, SEPS just make it much easier for users.

- Impact in Environmental Context is medium because SEPS is safe to use and practical app in event environments.
- Impact in Societal Context is High because accessibility of SEPS has a huge impact on users and this affects the growing of the SEPS community positively.

7.4 Teamwork Details

7.4.1 Contributing and Functioning Effectively on the Team

The developer team had distributed design roles to each member of SEPS. Each of the members had contributed to every section of the design, although a leader for a specific role had always been assigned. Briefly, Alperen contributed to specification of design goals and purposes and of the SEPS system, while considering other factors for the design. Burak planned the software/hardware mapping and subsystem decomposition. Erkin elaborated the data management and control scheme of the SEPS's system. Taner prepared the design plan for the server side of the subsystems. Samir designed the SEPS's client side subsystems.

7.4.2 Helping Creating and a Collaborative and Inclusive Environment

We have used several tools for proper communication among us, the number of such communication tools also increased as time went on throughout the semester. We used Whatsapp messages, e-mail messages, Zoom calling and phone calling for basic communications. However, since file sharing and editing is too hard for teamwork use on these communication tools, we used shared Google Drive folders and Google Docs for this purpose. We also divided projects into parts and distributed them among us and set due dates. so everyone in the group had certain work and time to do their work. For the, mainly implementation parts, we used GitHub to share and synchronise project's separate code segments, which was constructed already.

7.4.3 Taking Lead Role and Sharing Leadership on the Team

Our team utilizes delegated leadership, in which a different leader is elected for every stage. The workload for our project is in below.

7.4.3.1 Work Package 1

Leader: Samir Ibrahimzade

Major milestones and deliverables:Front-End Implementation, High-Level Design Report

Start Date: November, 21, 2020

End Date: Jan 21, 2021

7.4.3.2 Work Package 2

Leader: Alperen Koca

Major milestones and deliverables: Back-End Implementation, Low-Level Design Report

Start Date: December, 21, 2020

End Date: Feb 8, 2021

7.4.3.3 Work Package 3

Leader: Mehmet Erkin Şahsuvaroğlu

Major milestones and deliverables: Database & Server Installation, Testing

Start Date: Feb 9, 2021

End Date: Apr 28, 2021

7.4.3.4 Work Package 4

Leader: Burak Yeni

Major milestones and deliverables: Final Reporting, Presentation & Demo

Start Date: March 15, 2021

End Date: April 30, 2021

7.4.4 Meeting Objectives

In this section, we will explain what we have done so far for the requirements.

7.4.4.1 Functional Requirements Met

- Users can create individual addresses with their email.
- Users can login with their email and password.

- Users can logout from their account.
- Users can create events and share the QR code of the event.
- Users can upload photos to the database.
- Users can download photos from selected events.
- Users can see the detailed information of the events.

7.4.4.2 Non-Functional Requirements Met

- Privacy: Now, users who do not attend the event cannot upload photos to this event.

7.5 New Knowledge Acquired and Applied

Our prior knowledge was not adequate to implement our project because we used tools and technologies which we had not learned yet. Our application is an Android type of application and it targets many users from different places, so we need to have a real time database and it should have worked online. In addition to this our database should have been scalable. To obtain this we decided to use Firebase technology from Google. Firebase works with JSON messages and it is for NoSQL type databases. Furthermore, to process images uploaded by users we needed to know machine learning algorithms such as deep neural networks and data analyzing. To do this we needed a robust server to keep data and process images efficiently and fast, so we chose Heroku cloud platform because it is free to use for a limited data size. We learned about creating APIs for our machine learning actions and we learned how to connect these APIs to our database and Android side of our application. In addition, for testing our application we acquired Appium and Selenium automation tools to conduct our automated black box testing

8. Conclusion and Future Work

8.1 Conclusion

After all work, we are satisfied with our application done so far. Our requirements are nearly finished and they are working well. Our most machine learning models train data well and give accurate results to users. Our face recognition API processes profile pictures and finds photos of the user in the events which the user is in them. In addition, our android side of application is also working well and all functionalities in the UI side are satisfied and up. By doing this project, we have improved our collaboration and communication skills a lot. We also developed our learning skills because we used new technologies and frameworks which we did not know before the phase of this project. However we still need to improve our performance for the future projects we will be in.

8.2 Future Work

In the future we will design and implement our application for ios devices. This leads us to get more users. By getting more users we can improve our face recognition models by training more data. It will also be beneficial to optimize the learning model according to increasing account numbers. Mask detection system can also be improved by re-evaluating its approach for genuine mask violations. Our other goal for the next decade is having short videos for events and labeling each person with their names. Gallery only web page is also our consideration to implement in the future.

9. Glossary

Face Mask Detection Image Dataset (Kaggle):

Dataset containing 843 masked face images with labels

Accessible at: https://www.kaggle.com/andrewmvd/face-mask-detection?select=images

Recognition API's:

Mask detection API and Face recognition API can be found at:

https://github.com/tanerdurmaz/seps-api-face-recognition

https://github.com/tanerdurmaz/seps-api-mask-recognition

Black Box Testing:

Automated test source codes can be available at:

https://github.com/Aerk1996/SEPSAppiumTest

Android Image Cropper Library:

A library for cropping an image at the mobile side. Available at:

https://github.com/ArthurHub/Android-Image-Cropper

Tufts-Face-Dataset:

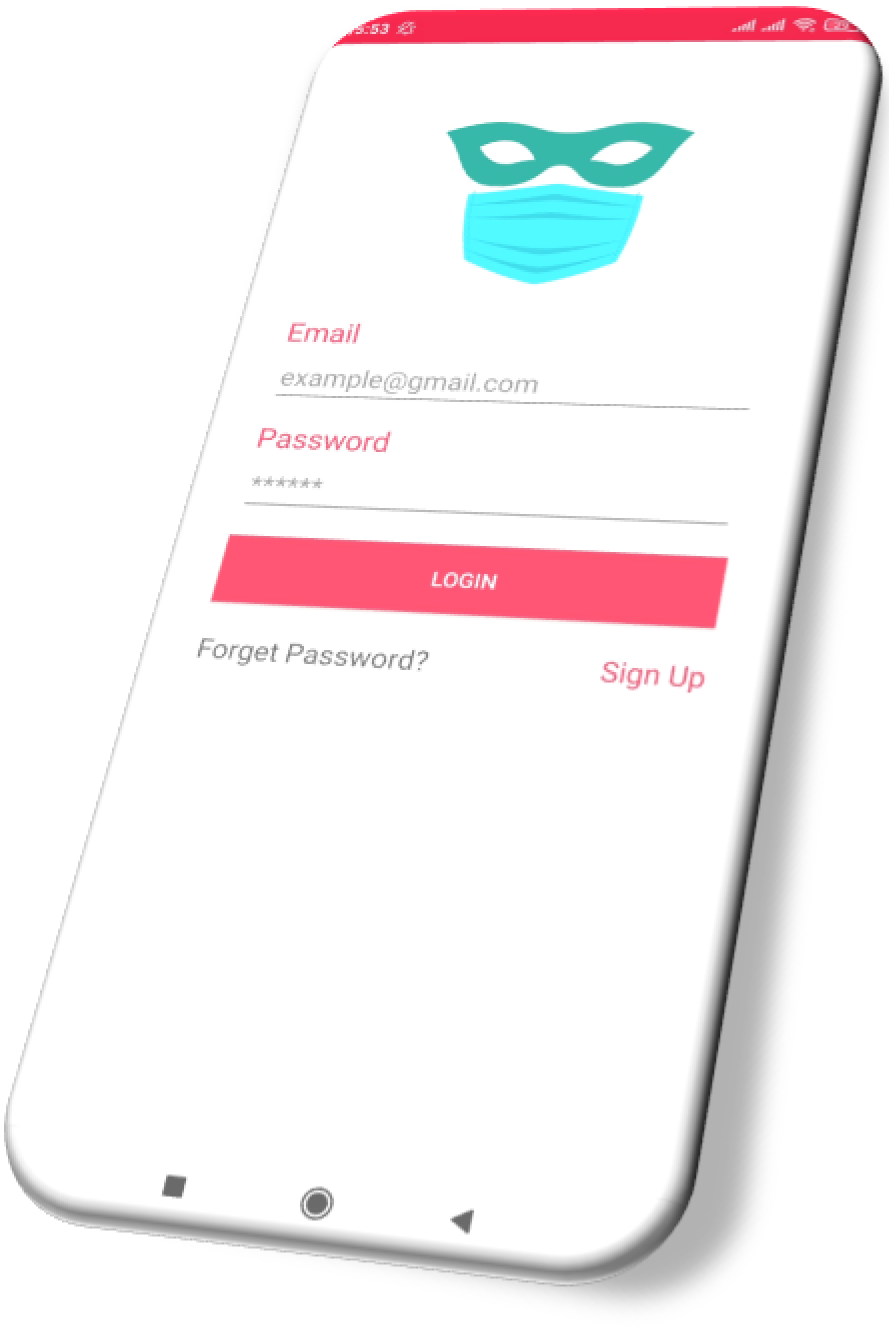
A dataset containing more than 10.000 images with labels. Accessible at:

https://www.kaggle.com/kpvisionlab/tufts-face-database

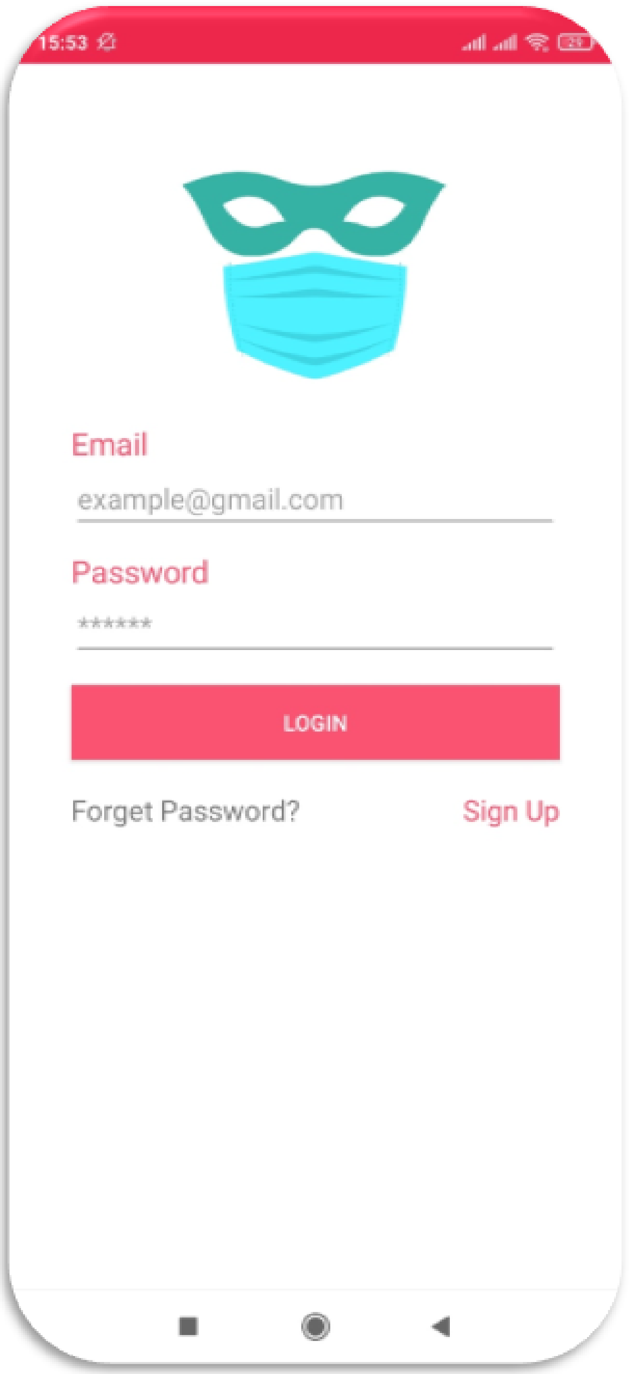
10. References

[1] The Code affirms an obligation of computing professionals to use their skills for the benefit of society. (n.d.). Retrieved November 1, 2020, from https://www.acm.org/code-of-ethics.

SEPS – User's Manual



1.1Login

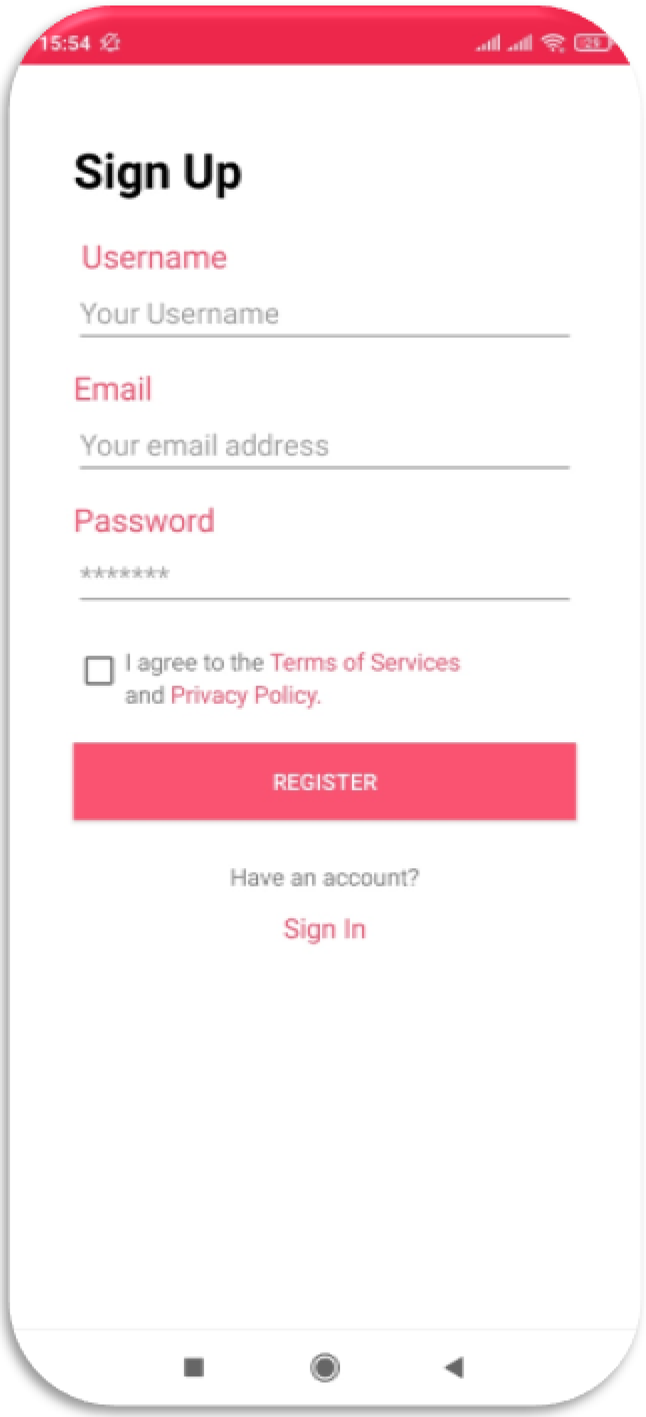


IN THIS PAGE, USERS CAN LOGIN SEPS VIA EMAIL AND PASSWORD

IF THEY DO NOT KNOW THEIR PASSWORD, THEY CAN CLICK 'FORGET PASSWORD' BUTTON TO CHANGE THEIR PASSWORD (1.3)

IF THEY ARE NOT SIGNED IN SEPS, THEY CAN GO SIGN UP PAGE BY CLICKING 'SIGN UP' BUTTON (1.2)

1.2Registration


15:54

Sign Up

Username

Your Username

Email

Your email address

Password

****

I agree to the Terms of Services and Privacy Policy.

REGISTER

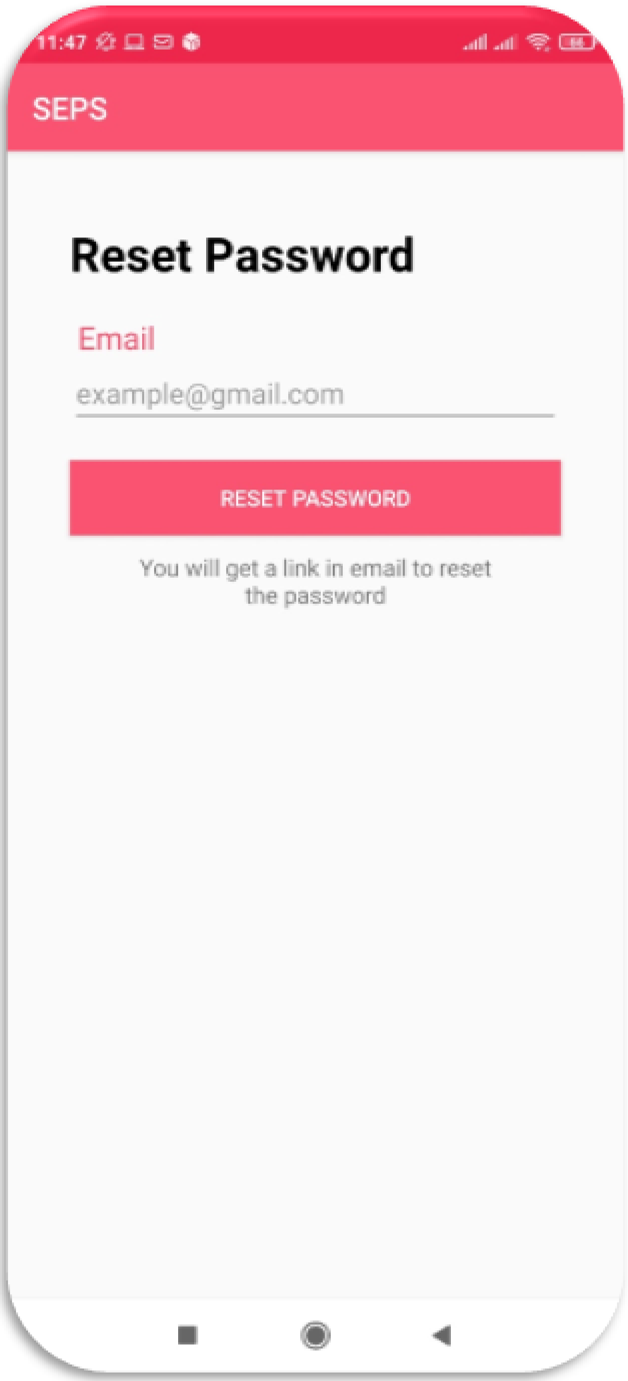
Have an account?

Sign In

IN THIS PAGE, USERS CAN SIGN UP SEPS BY FILLING TABLES (USERNAME, EMAIL, PASSWORD) AND ACCEPT THE TERMS OF SERVICES AND PRIVACY POLICY PAGE

IF THEY REMEMBER THEY HAVE ALREADY SIGNED IN SEPS, THEY CAN GO SIGN IN PAGE DIRECTLY

1.3Forgot Password?


11:47

SEPS

Reset Password

Email

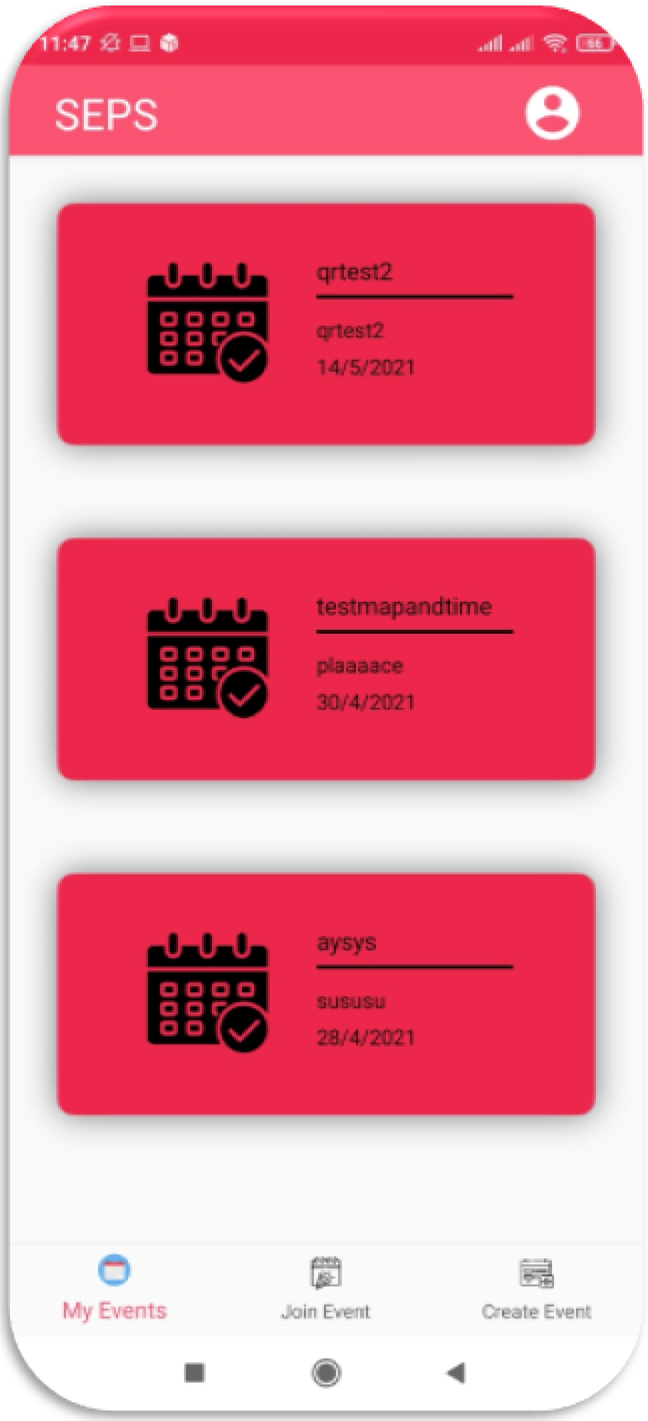
example@gmail.com

RESET PASSWORD

You will get a link in email to reset the password

USERS CAN RESET THEIR PASSWORD BY GETTING CONFIRMATION MAIL TO THEIR EMAIL ADRESS

2.1My Events


11:47

SEPS

qrtest2

qrtest2

14/5/2021

testmapandtime

plaaaace

30/4/2021

aysys

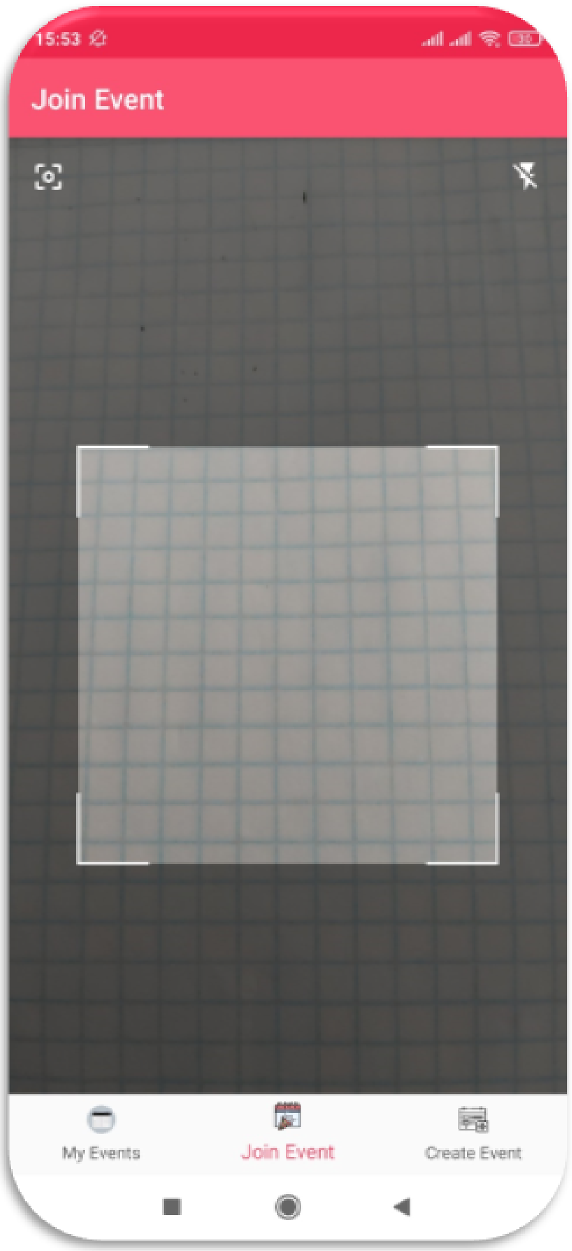
sususu

28/4/2021

USERS CAN VIEW THEIR UPCOMING AND PREVIOUS EVENTS IN THIS PAGE

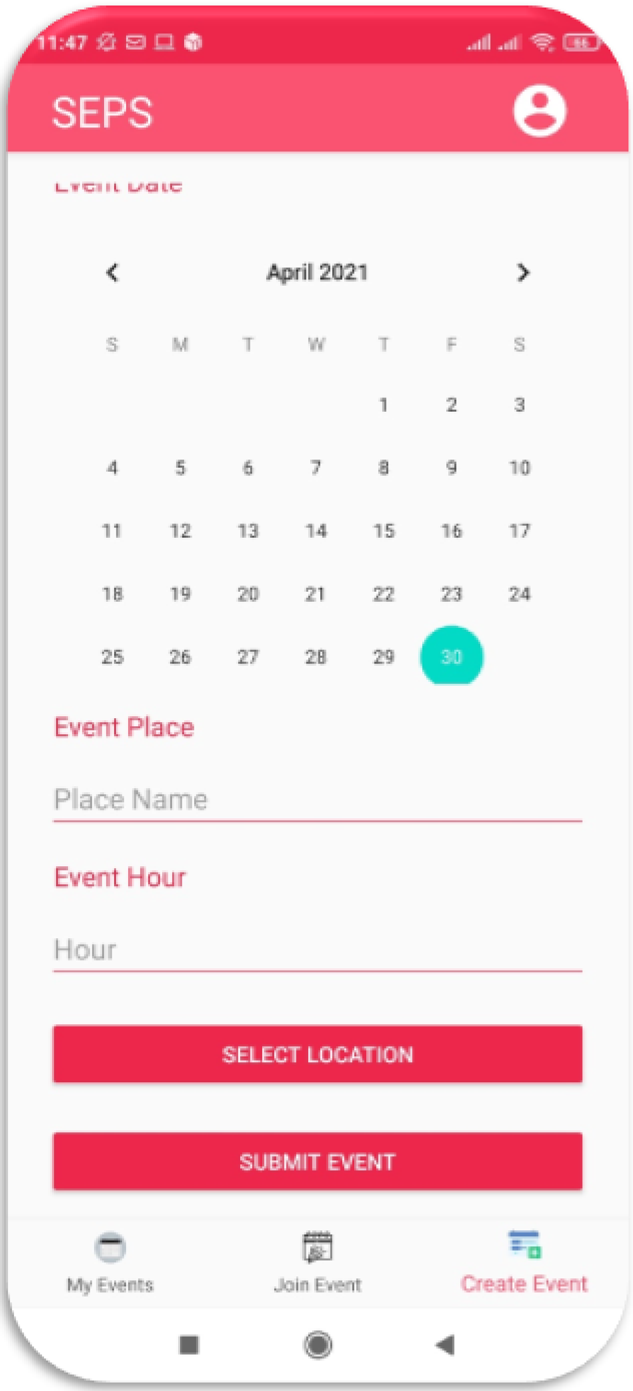
THEY CAN GO JOIN EVENT PAGE, CREATE EVENT PAGE AND SPECIFIC EVENT PAGE FROM THIS SCREEN

2.2 Join Event



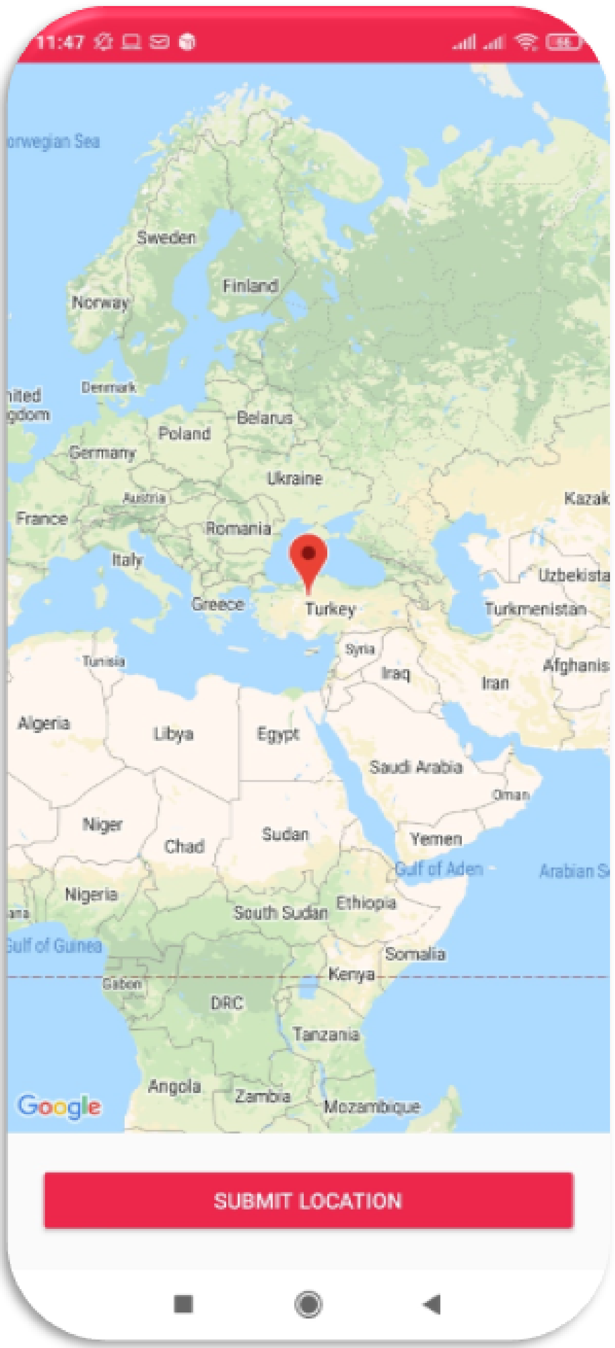
USERS CAN JOIN EVENT BY SCANNING QR CODE OF THE EVENT

2.3Create Event



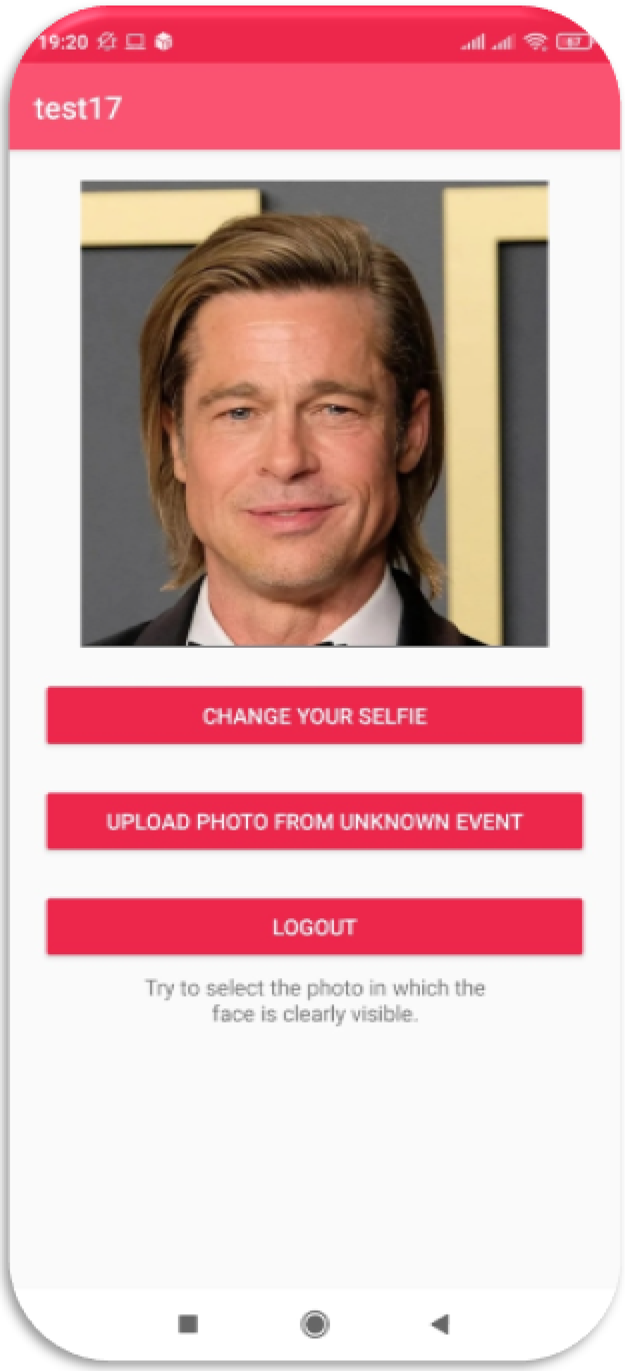
USERS CAN CREATE EVENT BY SELECTING DATE, PLACE, TIME AND LOCATION, THEN SUBMIT IT FOR OTHER USERS

2.4Select Location



USERS CAN SELECT LOCATION OF EVENT USING PRACTICAL MAP IN THIS PAGE

3 Profile Page



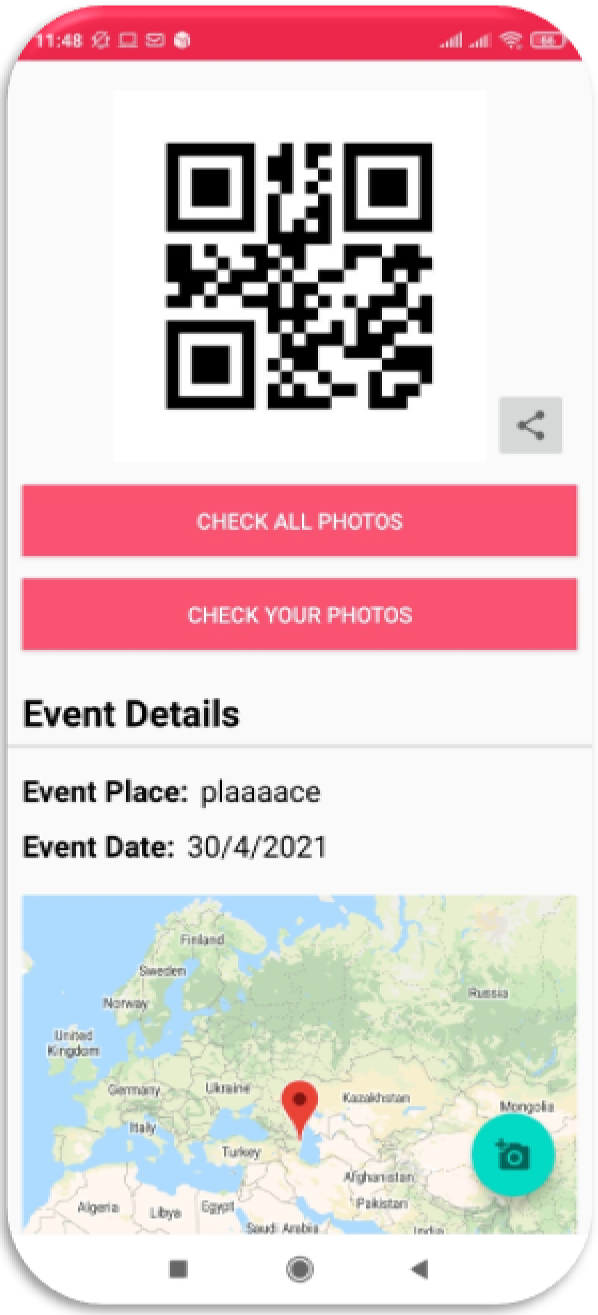
Try to select the photo in which the face is clearly visible.

USERS CAN CUSTOMIZE THEIR PROFILE PICTURE BY TAKING NEW SELFIE

ANOTHER OPTION IS TO UPLOAD UNKNOWN EVENT PHOTO

USERS CAN ALSO LOG OUT FROM SEPS IN THIS PAGE

4.1Event Page



Event Details

Event Place: plaaaace

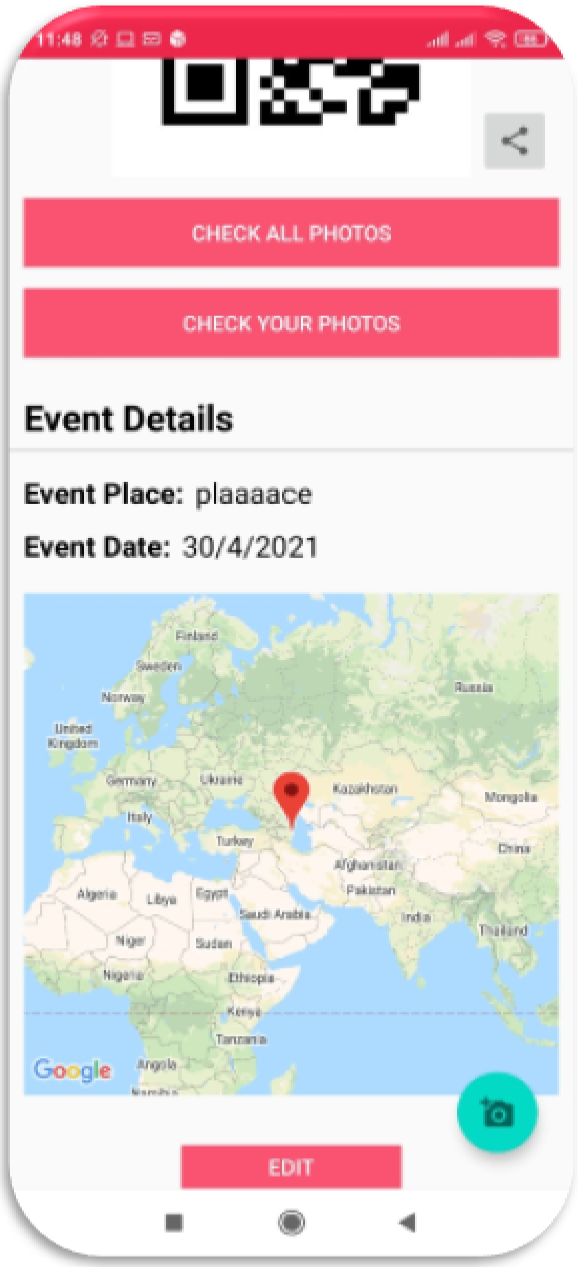
Event Date: 30/4/2021

USERS CAN VIEW QR CODE OF EVENT

THEY CAN CHECK ALL PHOTOS OR THEIR PHOTOS

THEY CAN ALSO VIEW EVENT DETAILS

4.2Event as admin



CHECK ALL PHOTOS

CHECK YOUR PHOTOS

Event Details

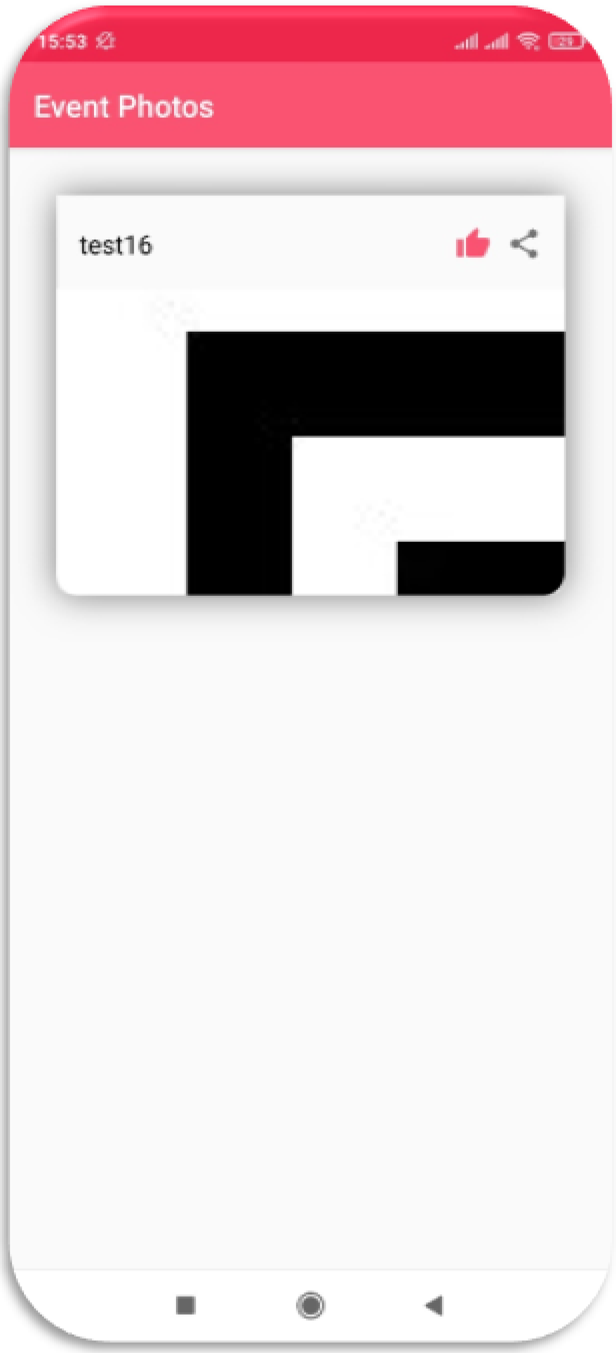
Event Place: plaaaace

Event Date: 30/4/2021

EDIT

IF USER IS ADMIN OF THE EVENT, USER CAN ALSO EDIT EVENT DETAILS

5 Event Photos



USERS CAN VIEW PHOTOS TAKEN IN EVENT THEY SELECTED AND THEY CAN ALSO SHARE OR LIKE THESE PHOTOS