AOSSIE

Google Summer of Code 2021 Proposal Idea (Monument App)

PROJECT GOAL:

The main objective of the project is to create a similar codebase for iOS and create a social media app for travellers where they can share their photos, experiences and also be able to do like, comment. Along with adding new features that enhance the User Experience and productivity of the app.

BASIC INFORMATION :

Contact Information:

- Name: Avinash Kumar.
- Email:avinashkumar.met18@itbhu.ac.in
- IRC/Gitter/Github: Kumaravinash9
- Phone number: +91 6204840373
- Country/ Region:Varanasi, India
- University : Indian Institute of Technology, Varanasi .

Skills:

- Depth Knowledge on Flutter, Firebase .
- Intermediate in Kotlin & Swift.
- Strong command on Backend (FrameWork : NodeJs (express)) .
- Strong Command on HTML and CSS (FrontEnd) .
- Proficient in Data Structures and Algorithms.
- Fluent in C++ .
- Basic in Java.

Version Control

- Strong concepts of Git. I used Github and Gitlab a lot.

PRE-GSOC INVOLVEMENTS :

Contribution to Monumento :

By thoroughly going through the code base for the **Monumento**, I found some bugs, fixed some additional things that should be integrated

into the app on an as-needed basis. To make them and to overcome the issues I have presented various MRs.Also, I tried to actively review MRs presented by my fellow co-developers and tried my level best that I could help them.

Some of my contributions are:

Monumento :

Merge Requests:

- 1.1. #54 :(**merged**) Feature Request:Added Google Maps screen, which takes the user to the location of the monument with the appropriate animation when tapping the location icon in the details screen using the **google_map** package. For the conversion of address to latlng, I used a **geocoder package** that returns latlng on passing address. Fix issue: #58.

  1.2. Fix bug: Each popular monument was being saved in the bookmark collection as many times as you tap the bookmark icon in the details Screen. I fixed that issue and also implemented a logic code to toggle (add /remove) the bookmark items from its collections. Fix issue : #46.

- #55 :(**merged**) Feature Request: Added visibility icon and its logic code in the password textField that toggles the visibility of the password in both the signup screen and the login screen. Fix issue: #60.

- #44 :(**merged**) Feature Request: WebView takes some time to load the Url. so, I added CircularProgressIndicator Until the Url gets reloaded. Fix issue: #52.

- #59:(**merged)** Fix bug : Fixed pixel overflow error in the HomeScreen by Using **FittedBox Widge**t . Fix issue : #61.

- #60 :(**merged)** Fix bug: Fixed pixel overflow error in the details screen that was appearing in the original container of the webview. Fix issue : #62.

- #62 :(**mergeable**) Feature Request: Implemented a profile change screen to remove and add a profile picture using **ImagePicker** (for getting the user's image through the camera, gallery) and **Firebase_Storage.** (storing the user's image). Fix issue: #50.

- #46 :(**mergeable**) Feature Request: Previously forget password logic was not defined in the source code. So, I implemented a forget password logic code along with designing and creating a forget password Screen. Fix issue : #44.

- #47 :(**mergeable**) Feature Request: Added animation to the carousel of popular monuments providing a better user experience (UI) using the **carousel_sider** package.

- #58 :(**mergeable**)Feature Request : Added merge_request template and issue template.

- #64 :(**mergeable**) Fix: The previously plain password was being stored in the firestore user collection. To make it more private, First I hashed the plain password then stored it in the Firestore User collection by using a **decrypt** package (creating a hashed password with a salt). Fix issue : #68**.**

- #48 :**(mergeable)** Feature Request: Fix: Previously remember me checkbox logic was not defined in the source code in the login screen. So I implemented the logic code by using the **Shared Preference package.** Fix issue : #45.

ISSUES CREATED :

(Note all the issues which are mentioned above in MR requests were also created by me.)

- #55 : **(Issue)**: In the Login Screen & Sign Up screen, Form validation is to be there for the validation of the textField like email, name and password.

- #59 : **(Issue)**: App is not specifying the proper reason why login fails. for example: “User already exists ”, “Password is wrong”.

- #66 : **(Issue):** Users are not able to edit their profile credentials like name, about, status and profile picture.

- #73 : **(Issue):** Push notification feature is not defined in the source code.

ASSOIE SCHOLAR PROJECT :

 # MERGE REQUEST :

- #104 :(**merged**): Added loading spinner in the popup.html until the web scraping is completed. changes I committed in popup.html,style.css and popup.js. Fix issue : #52**.**

- #97 : (**merged**) : Assoie Scholar extension is not working with url having **.co.in** domain or **.ca** domain . so i fixed it.Fix issue : #45.

- #103 : **(merged):** On the starred page, the "Google Scholar Page" is not completely inside its respective container. So, I fixed this by doing some changes in style.css. Fix issue : #51.

- #96 :(**merged):** Added loading spinner for the profile, starred and Search tab until the page completely reloaded. Fix issue : #43.

- #101 : (**merged**): On the scholar profile page, "Profile pic of the scholar is not completely inside its tag". So, I made some changes to the profile.css file and fixed it. Fix issue : #49.

- #100 :(**merged):** Added document title for all the tabs.

- #99 : (**merged**): Added description tag in manifest.json file which provides a brief description to the user about the project's goal in Chrome extension tab. Fix issue : #47

ISSUE CREATED :

- (Note all the issues mentioned above in each MR was also created by me.)

- #54 : (**Issue**): UI improvement is needed on the starred page.

- #50 : (**Issue**): showing an error that module is not defined in the profile .js which is showing in the chrome extension debugging.

In addition to these, I will keep on contributing to the repository before the official time for the project starts.

PROJECT SUMMARY :

The project aims to provide users with a unique experience of learning and learning more about the various monumental structures around the world within the app by Using AR (Augmented Reality) and Wikipedia content. In future, it will also provide a social media platform for travellers where users can share their photos, experiences.

This project is made on native android which is bound with the flutter module and they are communicating through channels. In this app, all the functionality is written in dart (Flutter) except AR and LandMark Detection which is written in kotlin and java. Firestore is used as a database for storing user's information, user's bookmarks as well as popular monuments.

DETAILED DESCRIPTION :

- **CREATING APP MODULE AS WELL AS METHODE CHANNEL :**

  For getting started, we will have to create an iOS app that will be binded with a flutter module. To do so, we will need to create a flutter module which is created by simply running the **flutter create my path** command in the terminal. After creating a flutter module, I will need to add a flutter module in CocoaPods(podFile.pd) and then run the podFile.pd .

```
flutter_application_path = '../my_flutter'
load File.join(flutter_application_path, '.ios', 'Flutter', 'podhelper.rb')

// integrate the flutter module with the PodFile

target 'MyApp' do
  install_all_flutter_pods(flutter_application_path)
end
```

PodFile.pd and run it

Creating Channels :

For creating a communication line, I will have to create methods channels. For that, I will have to write some codes in the flutter module as well as in AppDelegate.swift. Below I have added some pseudo-code on how I will create methods channels.

- Calling the host from the Flutter (client) through MethodeChannel :

```
static const platform = const MethodChannel("monument_detector");

_navToMonumentDetector() async {
  try {
    await platform.invokeMethod(
        "navMonumentDetector", {"monumentsList": monumentMapList});
  } on PlatformException catch (e) {
    print("Failed to navigate to Monument Detector: '${e.message}'.");
  }
}
```

Codes Methode channel in the Flutter dart file

```
onPressed: () async { _navToMonumentDetector();},
```

Invoke the Method call by calling the method channel function

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) ->
        Bool {

    let controller : FlutterViewController = window?.rootViewController as!
        FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "monument_detector",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in

      batteryChannel.setMethodCallHandler({
          [weak self] (call: FlutterMethodCall, result: FlutterResult) -> Void in
  // Note: this method is invoked on the UI thread.
           guard call.method == "navMonumentDetector" else {
          result(FlutterMethodNotImplemented)
           return
  }
  // define the function/methods for Landmark detection , WebView and AR view.
})
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

Listening to the methodChannel call along with handling the function call in AppDelegate.swift

ADDING AR FUNCTIONALITY :

For creating AR functionality, I will have to enable camera permission usage in info.plist. After that, I will drag and drop an ARKit SceneKit View on the ViewController.And I will also have to define @IBOutlet class, add the ARWorldTrackingConfiguration () and assign it in a variable that is responsible for established an AR config. Then after I will define the viewDidLoad() function which does the same work that initstate does in a flutter. Inside the viewDidLoad() function I will add some code for the configuration of the plane as well as running the scene kit session. On moving the camera, the AR kit starts to find different ARAnchors in the surroundings. After finding new ARAnchors, Arkit starts to render the 3D models. So, I will write some functions which delete all the previous Anchors and add our 3D models with the new AR anchor position. For that, I will write three important functions which take 3 parameters SCNSceneRenderer, SCNNode and anchor. Below, I have written pseudocode for the complete implementation.

```
import UI Kit
import AR Kit


class ViewController : UIViewController{

    @IBOutlet var sceneView: ARSCNView! let config = ARWorldTrackingConfiguration()
    override func viewDidLoad() {
        super.viewDidLoad()

        sceneView.delegate = self
        sceneView.scene = SCNScene()
        config.planeDetection = .horizontal

        //ARSCNDebugOptions.showFeaturePoints will display all the points on the screen which the AR camera has recognised in the surroundings.

        //ARSCNDebugOptions.showWorldOrigin will display world origin on the screen

        sceneView.debugOptions = [SCNDebugOptions.showFeaturePoints,ARSCNDebugOptions.showWorldOrigin]

        sceneView.session.run(config)
    }
}

 fileprivate func createFloorNode(anchor:ARPlaneAnchor) ->  Func()//function for the models


extension ViewController:ARSCNViewDelegate{
// add the anchor when we move the camera
    func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
        guard let planeAnchor = anchor as? ARPlaneAnchor else {return}
        let planeNode = createFloorNode(anchor: planeAnchor)
        //
    }
// update the new anchor and delete the all the node of the pervious position
    func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode, for anchor: ARAnchor) {
        guard let planeAnchor = anchor as? ARPlaneAnchor else {return}
        node.enumerateChildNodes { (node, _) in
            node.removeFromParentNode()
        }
        //write the function for the providing the 3D models
        let planeNode = createFloorNode(anchor: planeAnchor)
        // add the model
        node.addChildNode(planeNode)
    }
    // delect all the nodes
    func renderer(_ renderer: SCNSceneRenderer, didRemove node: SCNNode, for anchor: ARAnchor) {
        guard let _ = anchor as? ARPlaneAnchor else {return}
        node.enumerateChildNodes { (node, _) in
            node.removeFromParentNode()
        }
    }
}
```

Codes for the AR view:

After fetching the Landmark by using landmark detection, we render the respective models in AR view by doing some changes in the above code. (We will use the same logic that is defined in the Monumento app)

Adding a Cloud Vision Api in Flutter:(iOS & Android) :

Now, landmark detection is integrable with flutter. So, I have decided to use flutter for the implementation of LandMark detection functionality.

- By using googleapis & googleapis_auth plugin, I will implement the cloud vision functionality. I have created some snapshot which supports how will I implement Cloud vision API in flutter itself :

  Googleapis_auth package is used for checking the cloud vision API credentials.Googleapis_auth provides a clientViaServiceAccount function which takes two parameters, the first one is for credentials details and the second one is for visionApi Scope and after passing these parameters, the function will return the instance of AutoRefreshingAuthClient. Later, It is used for Calling landmark detection API calls.

```
class CredentialsProvider {
  CredentialsProvider();

  Future<ServiceAccountCredentials> get _credentials async {
    return _credentiales;
  }

  Future<AutoRefreshingAuthClient> get client async {
    AutoRefreshingAuthClient _client = await clientViaServiceAccount(
        await _credentials, [VisionApi.cloudVisionScope]).then((c) => c);
    return _client;
  }
}
class RekognizeProvider {
  var _client = CredentialsProvider().client;

  Future<String> search(String image) async {
    var _vision = VisionApi(await _client);
    var _api = _vision.images;
    var _response = await _api.annotate(BatchAnnotateImagesRequest.fromJson({
      "requests": [
        {
          "image": {"content": image},
          "features": [
            {"maxResults": 10, "type": "LAND_DETECTION"}
          ]
        }
      ]
    }));
    //print(_response.responses);
    String _bestGuessLabel;
    _response.responses.forEach((data) {
      var _label = data.webDetection.bestGuessLabels;
      //_bestGuessLabel = _label.single.label;
      _bestGuessLabel = data.landmarkAnnotations.single.description;
    });

    return _bestGuessLabel;
  }
}
```

Pseudocode for the landmark detection using google API _auth as well as googl_apis package

- Cloud Vision API requires an image file in **the base64 image** string. For that, I will have to convert the clicked image into bytes and

then convert it into a base64 image string and pass it as a parameter of the **RecognizedProvider().search** function. **RecognizedProvider().search** function does API call on Cloud vision API and returns the name of the Monumento.

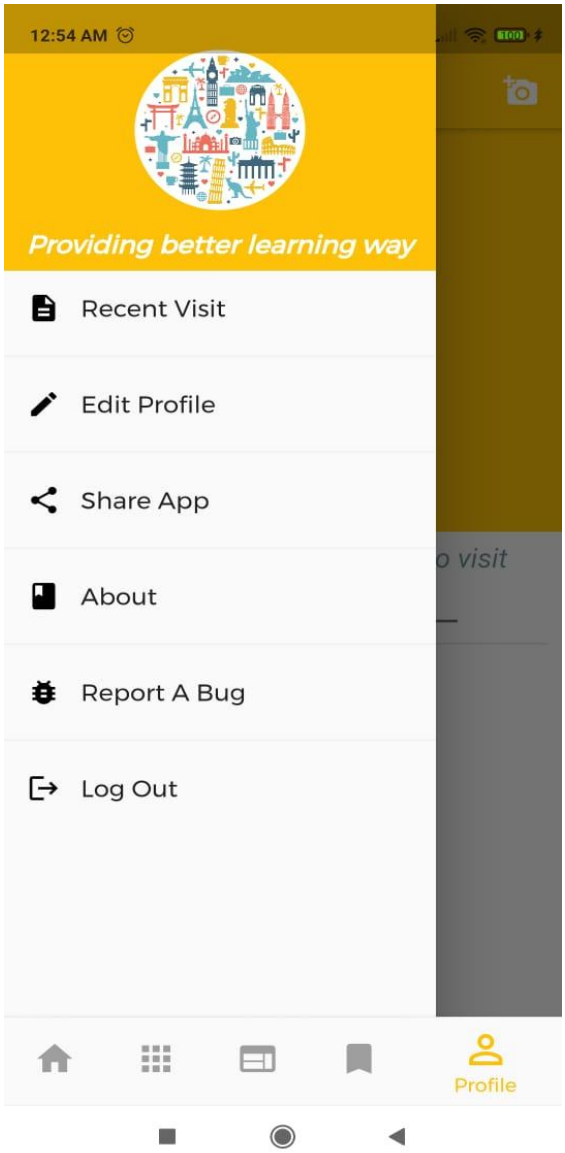
```
Future<dynamic> getlandmark() async {
  List<int> imageBytes = await widget.image.readAsBytesSync();
  print(imageBytes);
  String base64Image = base64Encode(imageBytes);
  var value = await  RekognizeProvider().search(base64Image);
  return value;
}
```

pseudo-code for converting image file into base64 string as well as calling Landmark function.

In this way, we will implement the cloud Vision Api (Landmark detection) in android and iOS.

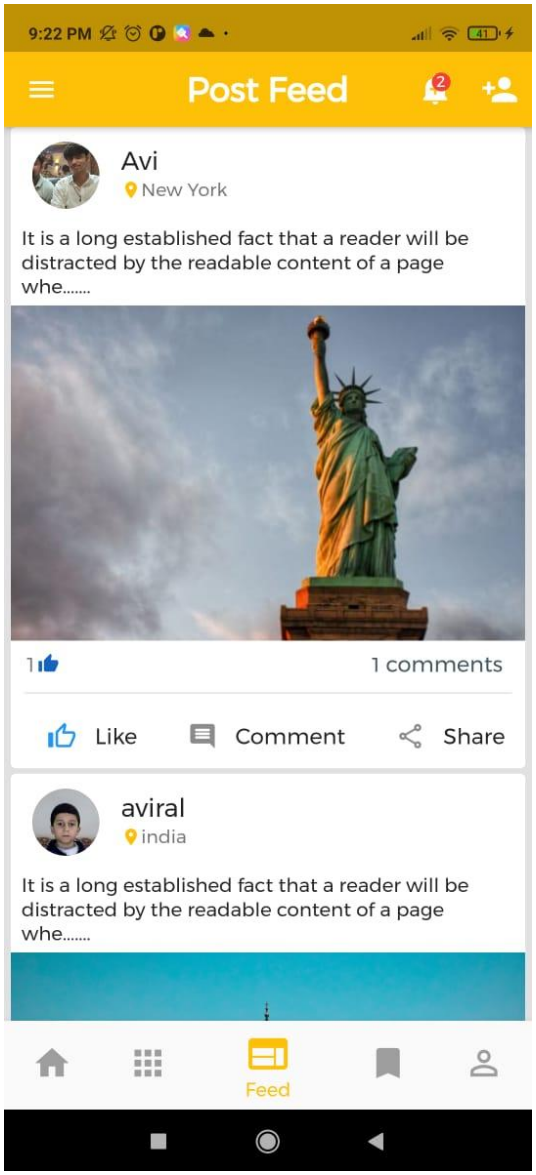
Adding More Screens :

- App Drawer provides a better UI as well as route which will help users to navigate different Screens. We can implement a drawer by Using the Drawer Widget.



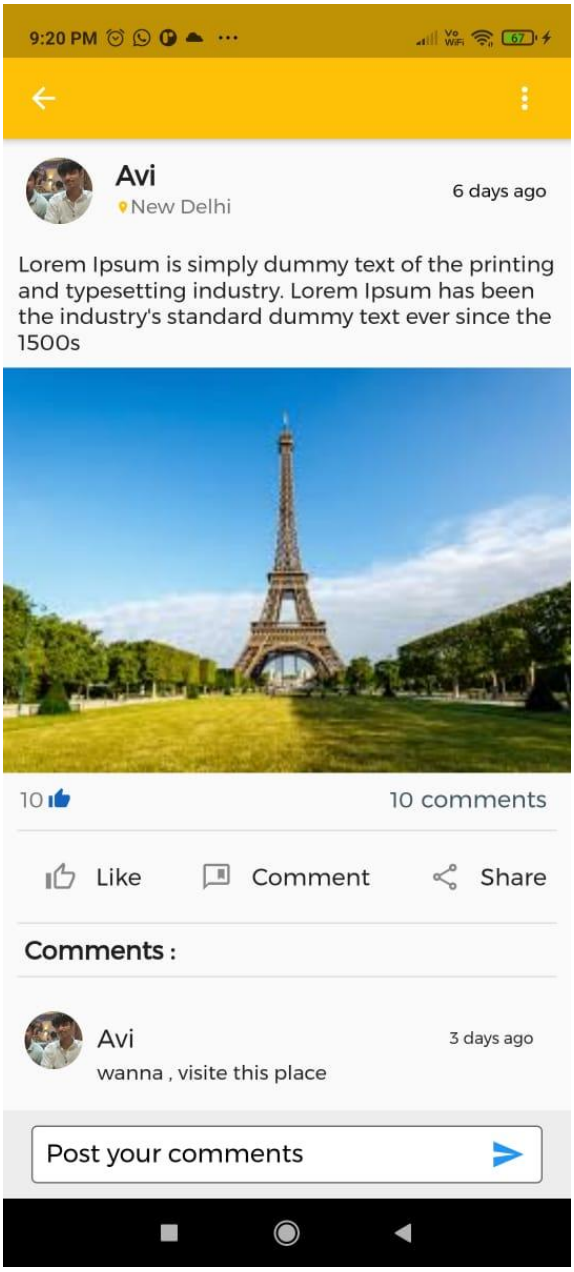
- **Feed screen**: It contains all the posts in a time sorted manner that is posted by the users. In each post, users are able to do like, share and comment. This screen contains one drawer as well as a

notification icon that takes the user to the notification Screen. In this screen, we will retrieve all the posts from the post-collection using the stream builder widget & FutureBuilder Widget.



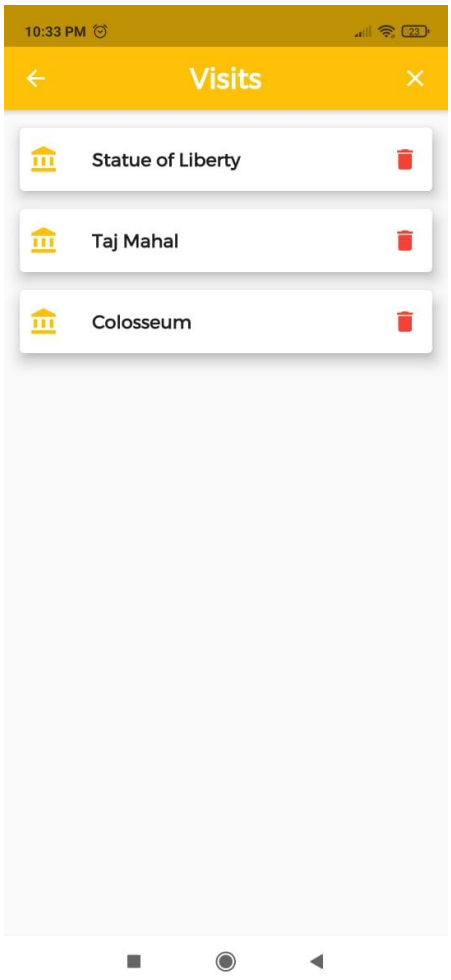
- **View post Screen**: Here, users can edit, delete and drop their comment. On this screen, we will retrieve post data by simply

passing the postid in the firebase Query. We will handle that data by using StreamBuilder. When the user typed their comment and taped the send button then the comment_message,auth_id and time will store in the comments collection as well as comment_id will also store in this post's comments list.

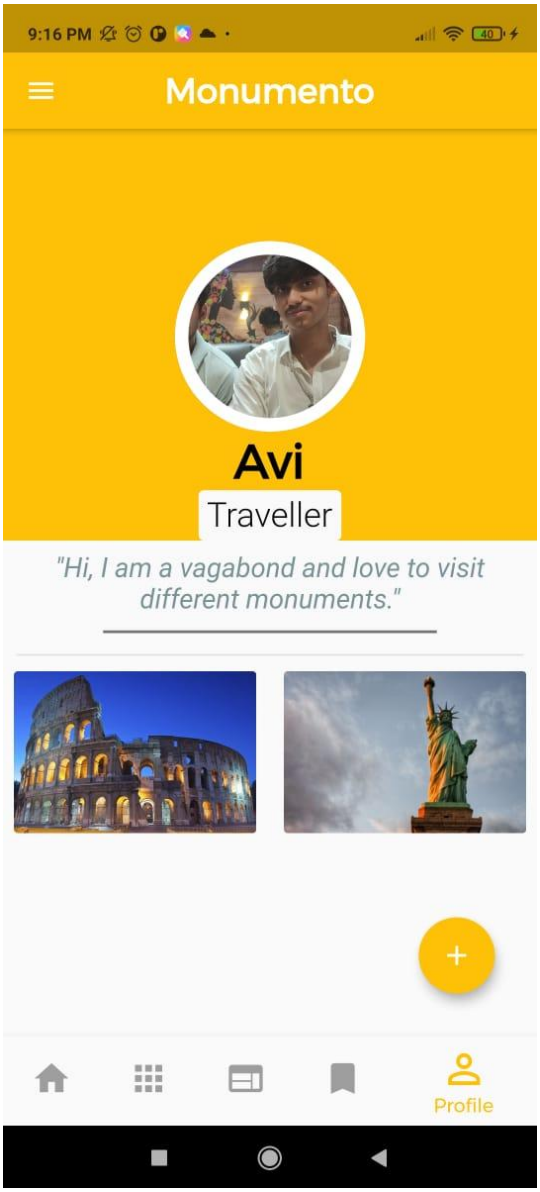


- **Recent Visit screen**: It contains all information that we have visited. By using the post's id, we can navigate to that post again.

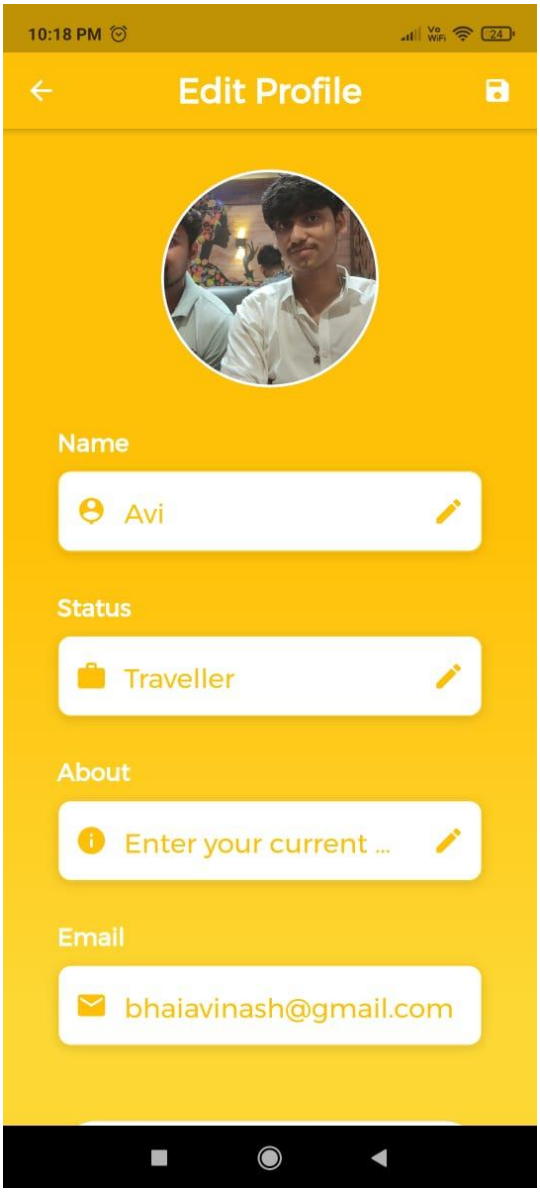
Users are also able to delete the history of each post by simply Swiping left or tapping the delete icon. It will be implemented by using **Dismissible Widget** (which provides a better swiping animation). Using steam builder & Firebase Query, we will retrieve all the visited posts from the history collection and render them to the screen.



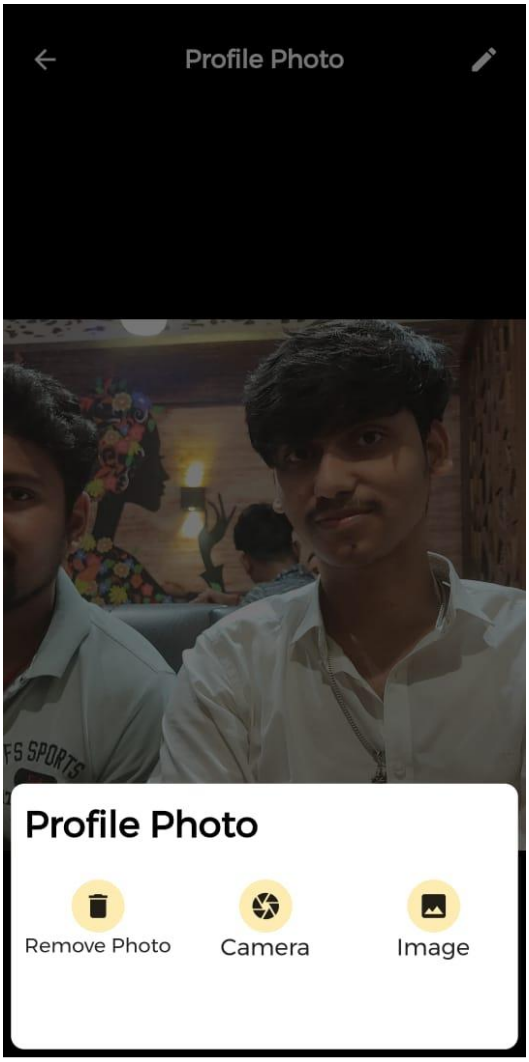
- **Profile Screen**: In this Screen, Users can view their profile, posts and also be able to share their travelled photos.



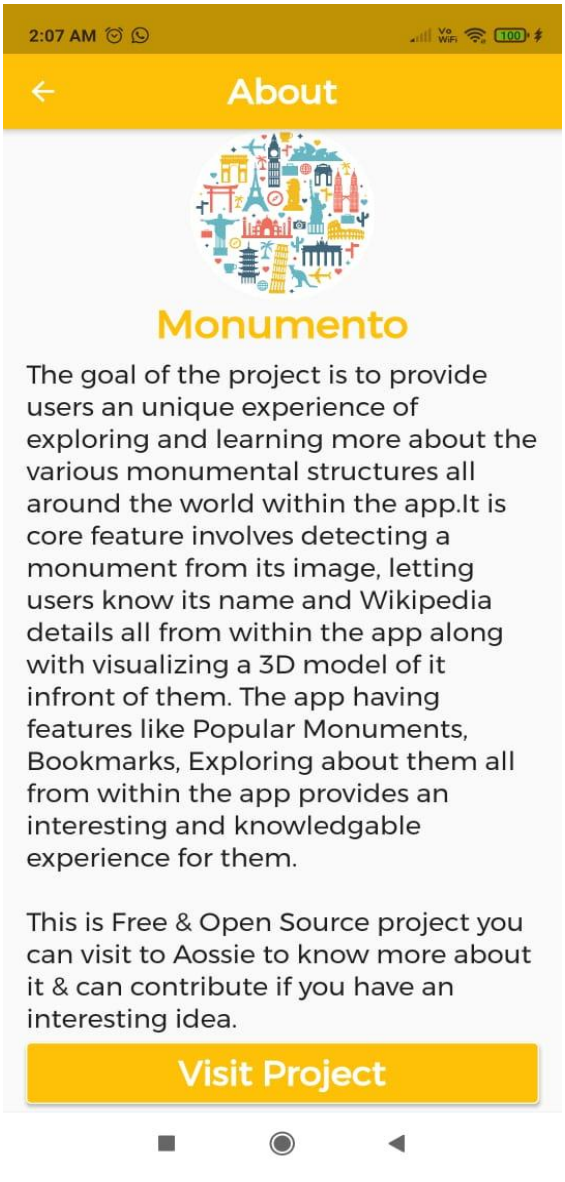
- **Edit Profile Screen**: In this Screen, Users can update their profile name, status, about and also profile pic. For changing the profile picture, Users need to tap the profile picture, it takes the user to the new View Profile Screen where users are able to change /remove their profile picture. User's profile information will be easily updated in the database by using the **cloud_firestore** query.



- **Edit Profile Picture Screen**: In this screen, users are able to change/remove/view their profile picture. After tapping the edit icon button on the top right, showBottomSheet will trigger that show options to remove or update the profile pic through camera or gallery. By using the image_picker package, we can access the camera as well as the gallery of a phone. After getting the photo we will update or store the picture in the **Firebase storage** and also save the URL in the user collection document.



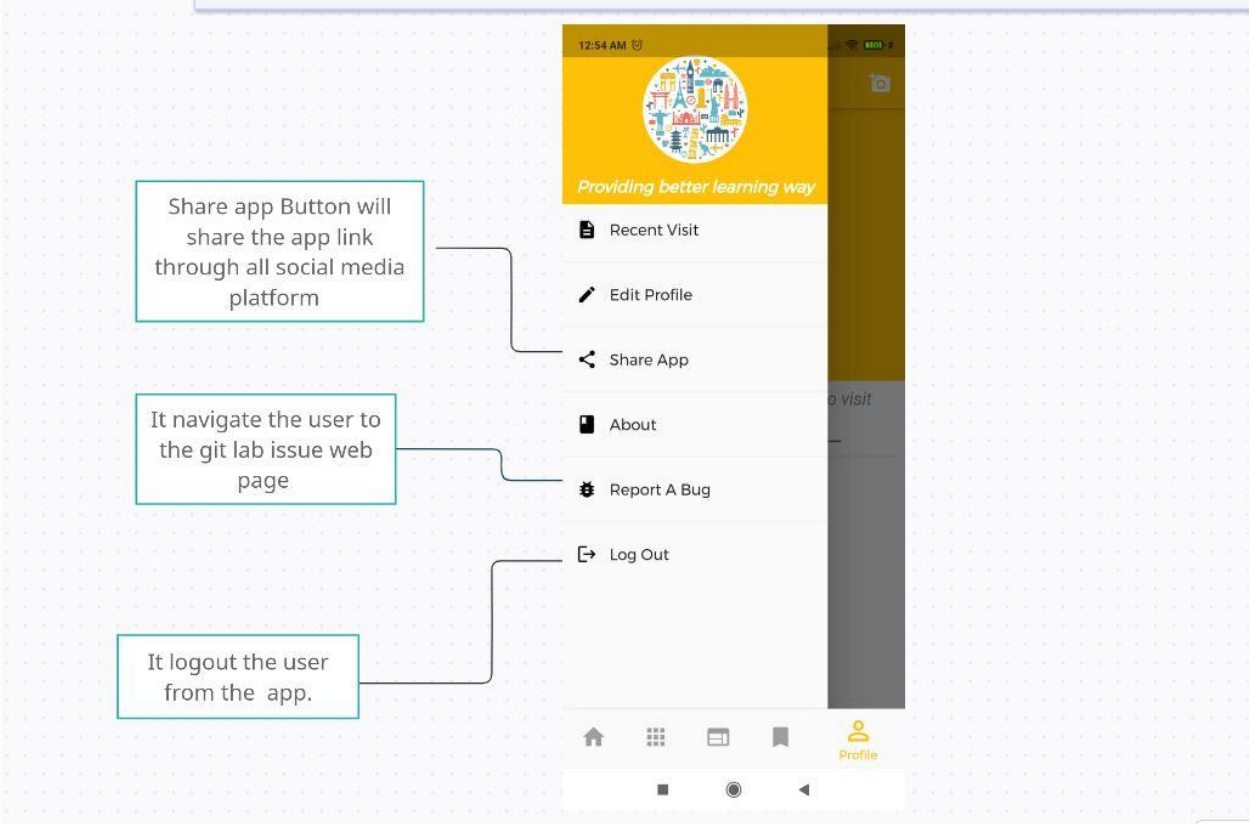
- **About Screen**: In this Screen, Users will be able to know about the Monumento Project and also be able to navigate to its git lab repository by tapping **Visite Project Button.** It will be implemented by using the **Url_launcher** package.



- **Forget Password Screen**: In this Screen, Users will be able to recover their password. For changing the password, users must have to enter their registered email. For checking, we simply do a firebase query on users collection. If the firebase query operation returns any document then we simply call the forget password

Query that sends the mail to the user's registered email. I have already implemented this screen in [merge request #46](#) .

- **Some More Functionality In App Drawer** :



- **Share App**: On Tapping Share App, It shows all the social media apps on which we can share the Apk Link to our friends. This will be implemented by using flutter **Share** Package.

- **Report A Bug** : On Tapping Report Bug, It navigates the user to its git lab repository issues web page. We can implement it by using the Url **Launcher** package .

- **LogOut** : On Tapping LogOut , It signout the user and takes the user to the LogIn Screen.

DATABASE DESIGNING :

The reason for using Firebase is very clear because Firebase itself provides a lot of features like push notification, Firebase provides authentication service, Firebase Analytics, Cloud Messaging, Real-Time Database, Firebase Storage and also provides Firebase ML kit. All these functionalities are easily compatible with the flutter.

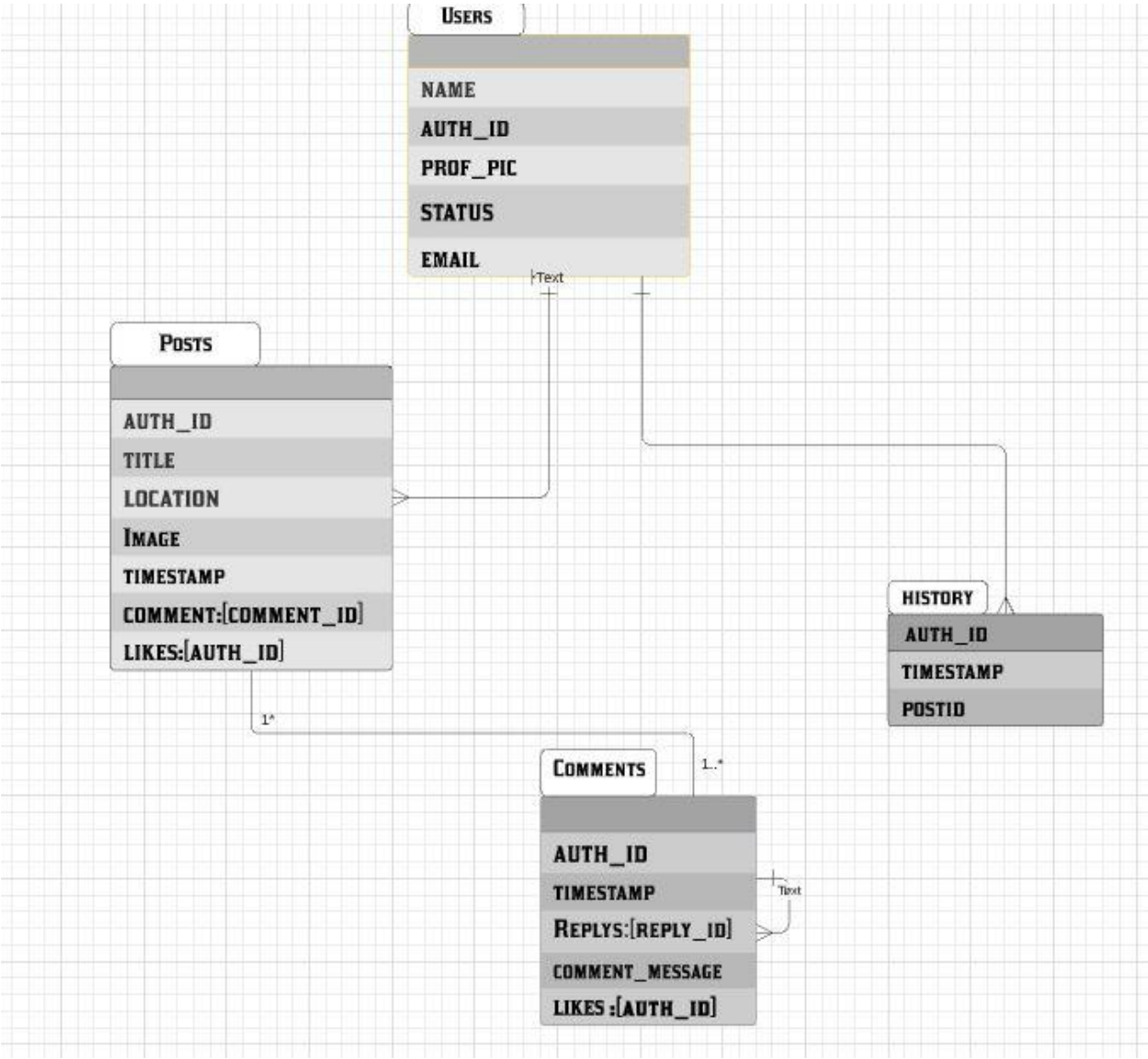
The app is already using Firebase. So , I will also align with firebase. After going through the whole source code, I got to know that there are 3 collections already present in the Firebase cloud storage .

- **Users collection**: Containing auth_id, name, email, prof_pic and status.

- **BookMarks collection**: Containing name, city, country, wiki URL, image.

- **popular_monuments collection** : Same data as BookMark collection.

For adding some more features (like, share and comment) , I need to do database management and Database designing for efficiently handling large amounts and multiple types of data.



Schematic diagram

- **Users collections**: Containing all the information regarding users like **auth_id**, **name**, **status**, **prof_pic**, **status**, **about** and **email**.

- **Posts collections** : It contain all the necessary data related to each post like **upload time**, **User auth_id** (who uploaded the post), **location**(address where photo has been uploaded), **image**(containing the url of the post), **title**(about the post), **Comments**(array of comment_id), **likes**(array of user's auth_id who likes the photos).

- **Comments collection**: It contains all the information regarding each comment like **comment_message**, **auth_id**(users who commented), **timestamp**(commented time), **Replies**(array of comment_id that replied on it), **Comments**(array of comment_id who like this comment). Each comment is related to its respective post.

- **History collection**: It contains all the necessary data which helps to retrieve all the information about visited posts . This will be implemented by simply saving the **PostID**(document id of the post that the user has visited in the past), **auth_id**, **timestamp**(time when we visited the post) in history collections. Each history is related to their respective users.

For creating a connection with the Firebase , I will add some **Flutter released plugin** :

- **Cloud_firestore plugin :** It helps to create live connection with firebase cloud messaging.It also provides some Firebase query

operation by which we can retrieve, update, create and delete data from the database collections.

- **firebase_auth plugin:** It helps to create local authentication and also contains login sessions of the user. It provides Query like createUserWithEmailAndPassword(), signInWithCredential(), signInWithEmailAndPassword(), currentUser() that helps to authenticate the user through email, password, credentials and also helps to get all the meta-data information about the user when the user is logged in.

- **firebase_storage plugin :** It helps to create a live connection with firebase storage. It also provides some queries operation by which we can store, delete and update the image(.jpg), video(.mp4) in the storage bucket. The Firebase storage will provide the image URL which we will save in the respective collection .

ADDING SOME IMPROVEMENT IDEA & APPROACH :

- I have some improvement requests and want to add these improvements to the app. We already know that In the app, the AR View option appears only when we snap the image via camera and if the image matches .But we should provide one more option to the user that they can select the image from the gallery and if the chosen image matches then they will get the option of AR view. In this way the user will not have to snap the photo again and again to get the AR view.

Implementation :

.After the user selects the image from the gallery , I will pass that image in the landmark detection function that will return the detected monument's name . In flutter, By the help imagepicker package we can easily select the image from the gallery/ Camera.After choosing image from the gallery , we will do the same thing that We are already doing for the camera option.

- Apart from this above improvement idea, we should do one more improvement that If the picked image via camera or gallery does not match with our app's popular monuments, then at least we should show wikipedia content of the response that comes from the landmark detection. In this way we can provide a new learning way to the user. We know providing an AR view of all the landmarks is not possible but providing wikipedia content is possible. I think we should add this improvement on the needed basics because the main purpose of this project is to provide a new learning opportunity about the monuments to the user by using AR and cloud vision api.

Implementation:

After getting the name of the landmark and if the landmark doesn't match with 3D models (.gltf URL) then we simply pass Wikipedia URL into Webview that renders the Wikipedia Contents.

Two approaches for getting Wikipedia URL :

- https://it.wikipedia.org/w/api.php?action=query&generator=search&gsrsearch=**response**&format=json&gsrprop=snippet&prop=info&inprop=url

We simply pass the response name at the of gsrsearch parameters that simply return a JSON file that contains all the Wikipedia content URL that is related to the landmark detection response.

- Most probably the Landmark is quite famous. so, if we simply pass the response in https://it.wikipedia.org/wiki/**response** that will also do our work.

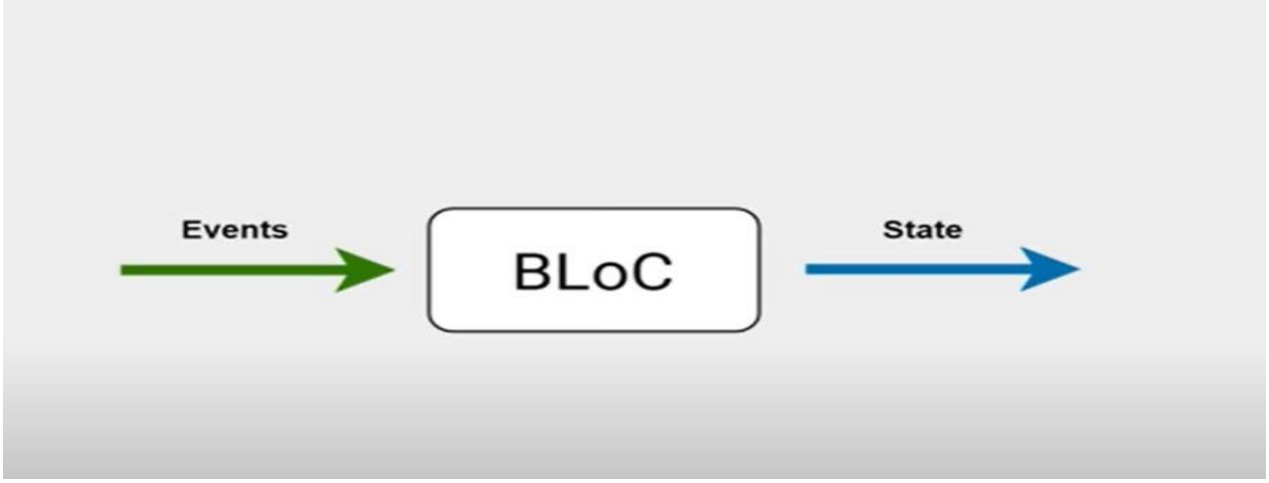
ADDING BLoC Models (state Management):

After adding features like share, comment and like photos, our app becomes more complex. Then we will have to use some state management plugin. There are too many plugins out there that do state management like getX, BLoC, provider package and so on . We can use any of them but the most popular plugin is BLoC models which is used by many companies. So, we should also use this plugin in this app.

Purpose of Using State Management :

- On calling set state function inside the Screen will result in re-rendering the whole screen. That will impact the performance of the app. By using the state Management plugin, it will re-render only those Widget that needs to be re-rendered.
- It also helps to refactor the codes as well as provides a clean code syntax pattern that will help future developers to understand the code base easily.

BLoC (Business Logic Components) pattern provides a unique way of state management in which we trigger the event in it and it returns the state of the event. BLoC flutter package link: flutter_bloc



(BLoC)

ADDING SIMMER EFFECT :

Till now, the app is using a circular progress indicator until the data gets reloaded. I want to add the SHIMMER effect at the place of the CircularProgressIndicator for the improvement of a better User Interface. There is a flutter package called Shimmer which can easily provide the shimmer effect into the screen. Flutter Package Link: shimmer

ADDING SOME IMPORTANT PACKAGE:

- **Intl package** :

  This package will be used for date conversion.

- **Flutter toast package** :

  This package provides different types of show dialogue box with better UI.

- **Cached_network_image package** :

  NetworkImage widget takes some time to reload the URL. So this package provides a placeholder Widget until the URL gets reloaded.

- **Shared_preference package** :

  This package is used as storage and the stored data can be handled/used on any of the screens.

- **Image_picker package :**

This package will be used for picking & clicking images from the gallery & camera.

SOME ISSUES AND ITS IMPLEMENTATION :

The ploy is going to Shut Down :

In the app, we are using gltf URL which is hosted on Poly. Now, poly is going to shut down. so we need to find alternative solutions.

APPROACHES:

(a). We can store the gltf models in our database like firebase Storage or S3 bucket and be able to use its hosted URL.

(b). We can upload the gltf/glb models on Github/ git lab and be able to use its GitHub/GitLab location raw URL: For Example :

https://raw.githubusercontent.com/Kumaravinash9/gltf/master/scene1.gltf.

TIMELINE :

1 . **April 13,2021 - May, 17,2021 : (application Review Period)**

a. Learning more about swift and Cloud Vision API.

2. **May 18, 2021 -June 1, 2021:**

a. Community bonding
b. Start working on implementing the project.
c. Discussing with the mentor the possible features that we can add to the project.
d. Solve issues and bugs if any.
e . Start implementing AR in swift.

3. **June 1, 2021 -June 31, 2021: (coding period start)**

a. Implementing cloud Vision API in a flutter.
b. Adding more screens.
c. Integrating cloud vision API and AR.
d. Adding More Screens For User profile like Edit profile, View Profile and Forget Password Screen.

4. **July 1, 2021 -July 28,2021 :**

a. Adding more screen and logics for comment, like and Share feature.
b. Connecting database with the screen.

5. **July 29, 2021 - August 20,2021**

a. Polishing UI

b. Learning and Writing test.
c. Working on pending work if any.

Final evaluation: August 20, 2021

ABOUT :

I am a third-year undergraduate student pursuing a B.Tech from the Indian Institute of Technology, BHU.

I have been doing web Development & App Development since my first Year. I have had experience in working with large codebases as well as making something from scratch through previous internships. Along with that, I have a strong hold on Data Structures and Algorithms.

I believe I am the right choice for this project and will work on this full time over the summer. I will give weekly updates about my progress and ensure I deliver according to the timeline set. I have started my open source contribution 3 months back from the same Organization and continuously try to solve issues as well as find bugs.

Some of my other relevant *achievements/ experience* are:

- Worked as a **Flutter developer intern** at Jashara Private Limited :

  During this internship, my work is to understand the whole source code, find bugs, their solutions, Writing logic part, refactor codes, improving UI and helping the team to deploy the application successfully on the Playstore. Link :

- Worked as a **Backend Developer Intern** at Billbolo Private Limited (January 2021 to current) :

  During this internship, my job is to create Apis for the merchant/customer android app in which I had created authentication APIs, payment-related APIs, merchant & user banks related APIs as well as store this information into MongoDB database and successfully deploying these APIs on the AWS E2C instance.

- Successfully cleared Uber HashTag First Round.

- Institute Rank under 30 on Geeks for Geeks.

MOTIVATION :

My inspiration for GSoC this year is making myself more familiar with open source Organization. GSoC is a great program to start organizations with future Contributors and when I saw this project, I felt that it was something I could do. I believe this project can definitely help me expand my boundaries, as this is the first time I will be working on a large project with a real-world impact. This possibility is very exciting for me.

I chose this project because it is using AR which always fascinates me and when I went through the entire source code, I found out how AR and Cloud Vision API (“Landmark Detection”) integrated into this project using channels that inspired me to choose this project and it is also

related to things I have done in the past. I am pretty sure that this project provides a good opportunity to apply my learning on a practical scale.

AVAILABILITY :

The official GSoC period is from 17 May to 23 August. I can easily devote 25-35 hours a week till my college reopens and 20-25 hours per week after that I'm free on weekends. I intend to complete most of the work before my college reopens.

Other than this project, I have no commitments/ vacations planned for the summer. Also, I don't plan on doing any internships this summer. I shall keep my status posted to all the community members on a weekly basis and maintain transparency in the project.

AFTER GSOC :

Being a part of such a vast community is a great opportunity in itself and I would love to collaborate with others throughout my project timeline and even after that, as this is the true essence of Open Source culture. I'll be an active member of the community and keep contributing. My motivation would always be that I'd be able to contribute to something big and widely in use. This gives me a lot of satisfaction.

REFERENCES:

- https://firebase.google.com/docs/ml-kit/android/recognize-landmarks.
- https://www.raywenderlich.com/6986535-arcore-with-kotlin-getting-started.

- https://pub.dev/packages/firebase_ml_vision.
- https://www.raywenderlich.com/6986535-arcore-with-kotlin-getting-started.
- https://github.com/kgediya/ArCore-Sceneform-Kotlin/tree/master/app/sampledata.
- https://developers.google.com/sceneform/develop.
- https://github.com/ranadhirdey/Home-Decor.
- https://firebase.google.com/docs/ml/ios/recognize-landmarks