**Milestone Report Document**

Final Year Project

**Technical Document**

For

**XPK**

BSCS

By

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S# | **Name** | **Registration #/Roll #/Section** | **Mobile #** | **E-Mail** |
| 1. | Abdulmanan Nazir | 17277/FA-2021/BSCS/287#/F | 03174689617 | mrmanan143@gmail.com |
| 2. | Iqra Fateh | 17294/FA-2021/BSCS/304#/F | 03097002398 | iqra133660@gmail.com |

**Supervised by:**

Ma'am Farwa Javad **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (**Signature**)**



Department of Computer Science

Lahore Garrison University

Lahore

**Milestone Report (XPK)**

## **1. Introduction**

XPK (Explore Pakistan) is a tourism-based mobile application that enables users to explore Pakistan's tourist spots based on their available time. Users can also upload vlogs, blogs, and share experiences. This report details the progress of Milestone 4, covering implemented features, UI development, API integration, and testing.

## **2. Project Architecture**

### 2.1 Domain Layer

* Contains business logic and entities.
* Defines use cases for user authentication, fetching places, and managing posts.

### 2.2 Presentation Layer

* Handles UI and user interactions.
* Implements screens such as Login, Home, and Profile using Flutter.

### 2.3 Infrastructure Layer

* Manages external dependencies like Firebase Authentication, Firestore, and Google Places API.
* Handles API requests, Firebase operations, and data storage.

## **3. Implemented Features**

### 3.1 Splash Screen

* Displays for 3 seconds.
* Navigates to the Home Screen if the user is authenticated.
* If the user is not authenticated, it navigates to the Get Started Screen.

### 3.2 Get Started Screen

* Displays the app logo and welcome text.
* Animated transition to the Login Screen.

### 3.3 Sign-Up Screen

* Fields: Username, Email, Password, Gender, Date of Birth, Profile Image.
* Validation checks ensure all fields are filled correctly.
* After clicking Sign Up, an email verification link is sent.
* Upon verification, user data is stored in Firebase Firestore.

**Code of Sign-Up Screen**

Future<void> signup() async {

check(true);

try {

UserCredential userCredential =

await \_auth.createUserWithEmailAndPassword(

email: email.value,

password: newPassword.value,

);

await userCredential.user?.sendEmailVerification();

User user = userCredential.user!;

await UploadImage();

\_saveUserToFirestore(user);

check(false);

} catch (e) {

check(false);

Get.snackbar("Error", e.toString());

}

check(false);

}

void checkEmailVerification() async {

User? user = \_auth.currentUser;

await user?.reload();

if (user != null && user.emailVerified) {

// Get.offAllNamed(Routes.home);

} else {

Get.snackbar("Error", "Email not verified yet.");

}

}

**3.4 Verify Email**

* Verify email after sign-up in the App.

void checkEmailVerification() async {

User? user = \_auth.currentUser;

await user?.reload();

if (user != null && user.emailVerified) {

// Get.offAllNamed(Routes.home);

} else {

Get.snackbar("Error", "Email not verified yet.");

}

}

### 3.5 Login Screen

* Allows users to enter a valid email and password.
* Authentication is handled using Firebase Authentication.
* Implemented proper field validation for email format and password strength.
* Users must verify their email before logging in.
* Only verified users can log in.

#### Code Implementation for Login

//%%%%%%%%%%%%%%%%%%%%%%%%%%% firebse functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

final FirebaseAuth \_auth = FirebaseAuth.instance;

final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

final FirebaseStorage \_storage = FirebaseStorage.instance;

var check = false.obs;

void login() async {

check(true);

debugPrint(

"Trying to login with Email: ${email.value}, Password: ${newPassword.value}");

try {

await \_auth.signInWithEmailAndPassword(

email: email.value.trim(),

password: newPassword.value.trim(),

);

check(false);

} catch (e) {

check(false);

Get.snackbar("Error", "Invalid email or password.");

}

}

**3.6 Forget Password Screen**

* If users not enter a valid email and password, then forget password screen appear.
* Verifies that password reset email is sent successfully.

**Code for** **Forget Password Screen**

void resetPassword() async {

if (email.value.isEmpty) {

Get.snackbar("Error", "Please enter your email address.");

return;

}

try {

await \_auth.sendPasswordResetEmail(email: email.value);

Get.snackbar(

"Success",

"Password reset email sent. Please check your inbox.",

);

Get.offAllNamed(AppRoutes.login);

} catch (e) {

Get.snackbar("Error", e.toString());

}

}

Get.offAllNamed(AppRoutes.login);

} catch (e) {

Get.snackbar("Error", e.toString());

}

}

### 3.7 Profile Screen

* Fetches and displays user profile data from Firebase.
* Currently UI is static (API integration pending).

### 3.8 Home Screen

* Fetches places using Google Places API.
* Displays place details, including title, description, images, rating, etc.
* Implemented filters for available time options (2, 4, 6 hours, 1, 2 days).
* Added search functionality using Google Places API.
* Auto-suggestions provided when searching for places.

**Code for Home Screen Search Place**

static Future<List<PlaceModel>> searchPlaces(

String query, Position position) async {

final searchUrl = Uri.parse(

'$\_baseUrl/textsearch/json?query=$query&location=${position.latitude},${position.longitude}&radius=50000&key=${ApiConstant.googleApikey}');

return await \_fetchFullPlaceDetailsFromSearch(searchUrl);

}

**Radius**

* Radius from duration static,implemented filters for available time options (2, 4, 6 hours, 1, 2 days).

// Get radius from duration

static int \_getRadiusFromDuration(String duration) {

switch (duration) {

case '2 Hours':

case '4 Hours':

return 15000;

case '8 Hours':

case '12 Hours':

return 35000;

case '1 Day':

return 50000;

case '2 Day':

case '3 Day':

case '5 Day':

return -1; // Indicates multi-day trip

default:

return 20000;

}

}

### 3.9 Detail Screen

* UI is **completed**.
* API integration **pending**.

// Get full place details

static Future<PlaceModel?> getPlaceDetails(String placeId) async {

final url = Uri.parse(

'$\_baseUrl/details/json?place\_id=$placeId&fields=name,rating,user\_ratings\_total,formatted\_phone\_number,formatted\_address,geometry,photo,review,url,website,price\_level,opening\_hours,current\_opening\_hours,editorial\_summary,types,permanently\_closed,business\_status&key=${ApiConstant.googleApikey}');

try {

final response = await http.get(url);

if (response.statusCode == 200) {

final data = json.decode(response.body);

if (data['status'] == 'OK' && data['result'] != null) {

return PlaceModel.fromJson(data['result']);

}

}

} catch (e) {

log('Error fetching place details: $e');

}

return null;

}

if (response.statusCode == 200) {

final data = json.decode(response.body);

if (data['status'] == 'OK' && data['result'] != null) {

return PlaceModel.fromJson(data['result']);

}

}

} catch (e) {

log('Error fetching place details: $e');

}

return null;

}

### 3.10 Add Post Screen

* UI is **completed**.
* API integration **pending**.

**3.11 Store data into FireBase**

**Code**

void \_saveUserToFirestore(User user) async {

await \_firestore.collection('users').doc(user.uid).set({

'username': userName.value,

'email': user.email,

"password": newPassword.value,

"birthday": birthday.value,

"gender": selectedGender.value,

"profileImageUrl": profileImageUrl.value,

"uid": user.uid,

"createdAt": FieldValue.serverTimestamp(),

});

}

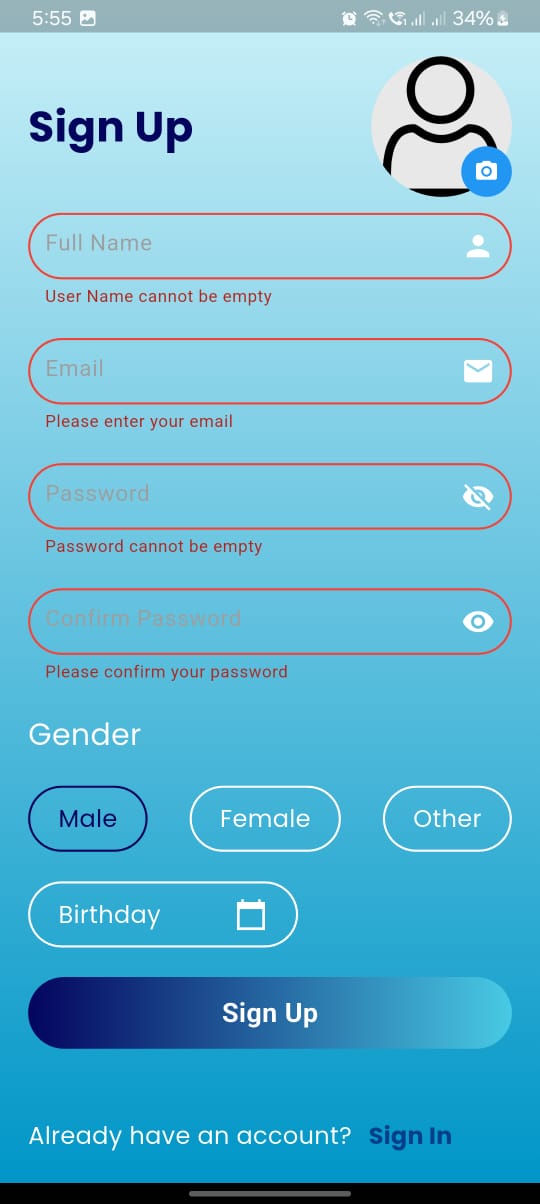
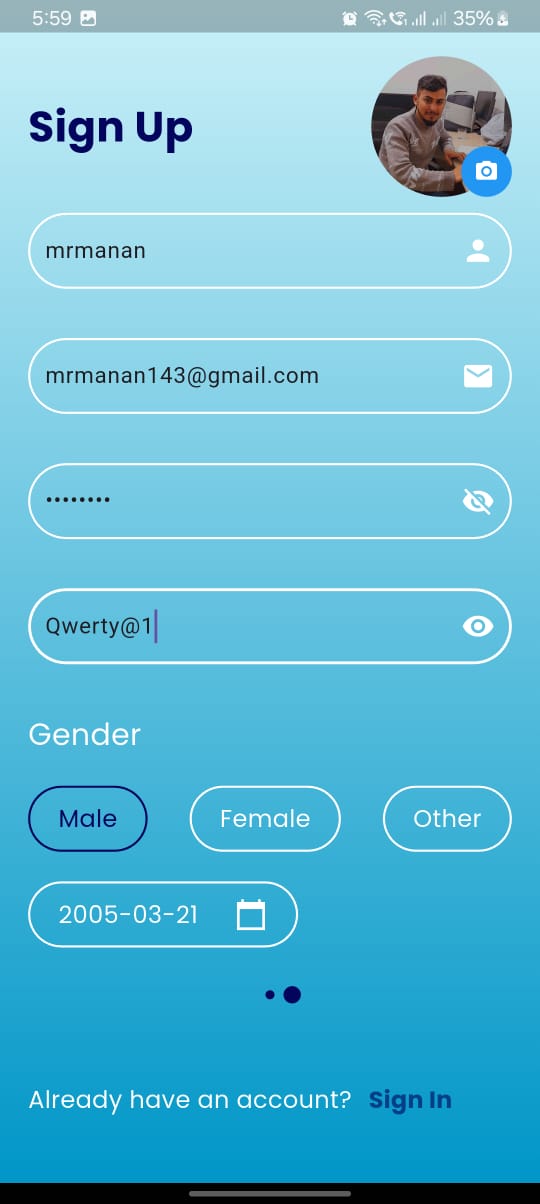
## **4. Testing**

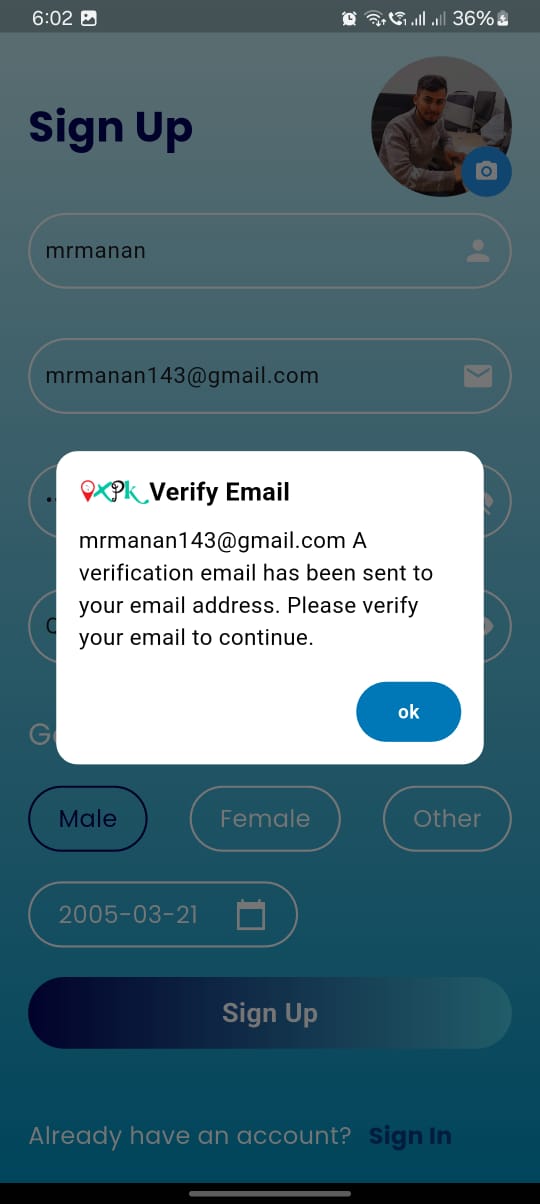
### Splash Screen and Get Started Screen: Navigates to the Home Screen,display the app logo and welcome text.

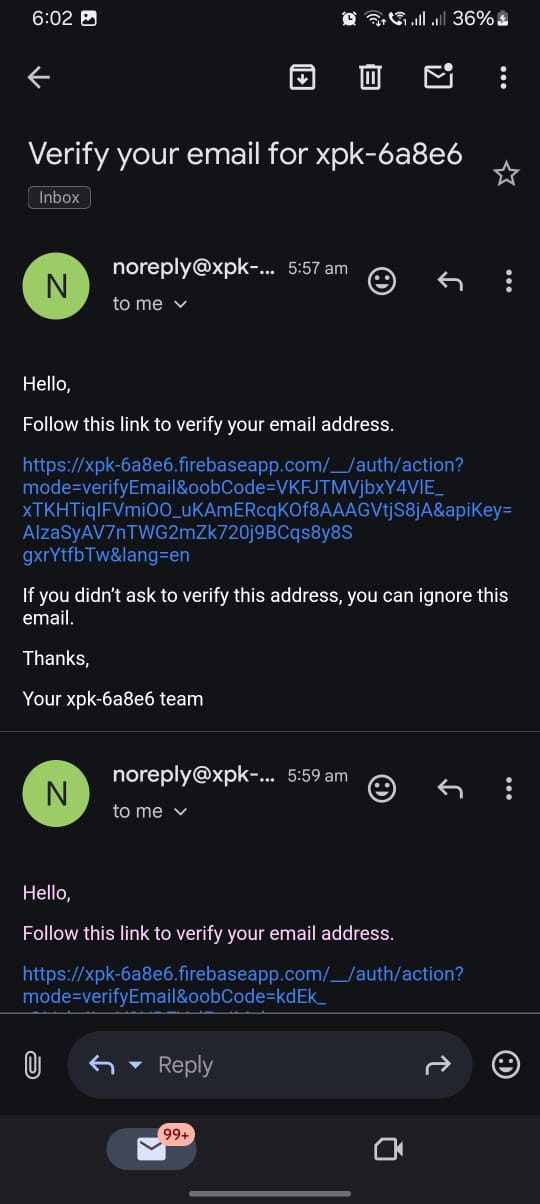




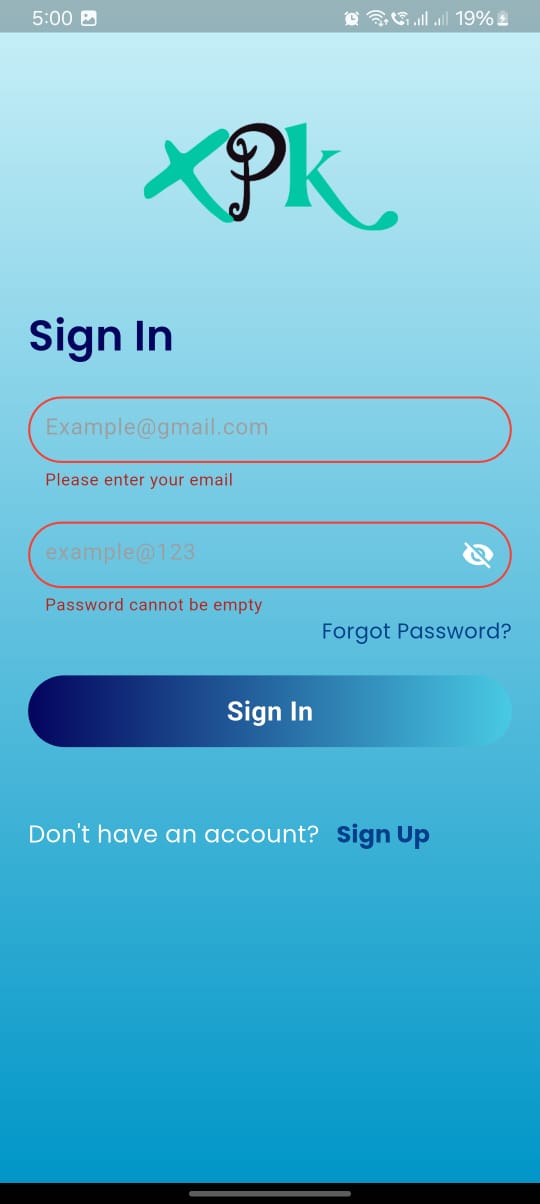
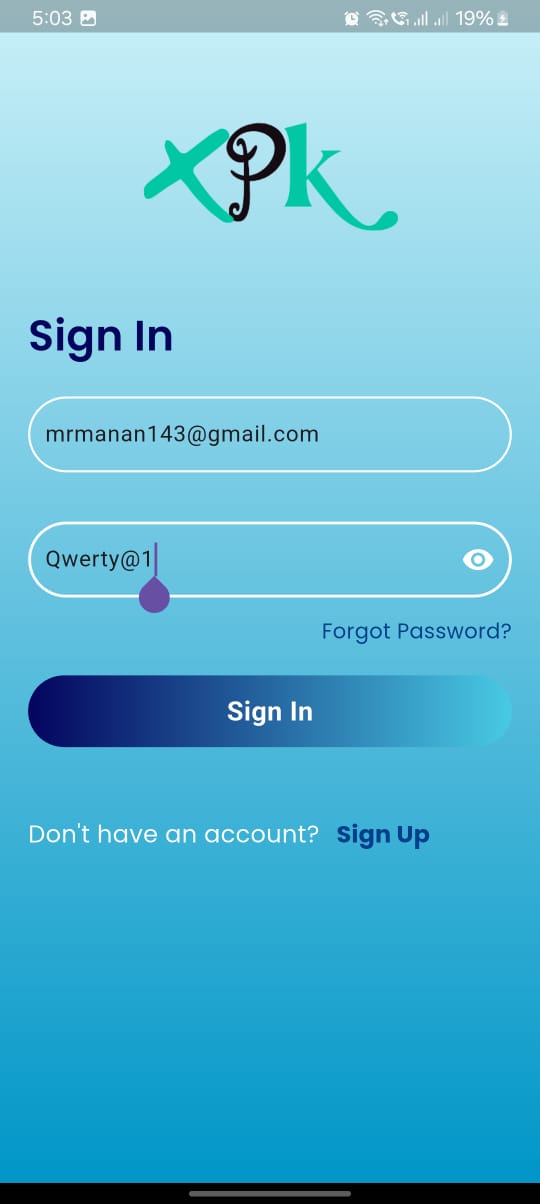
* **Sign-Up Module:** Tests field validation and email verification flow.

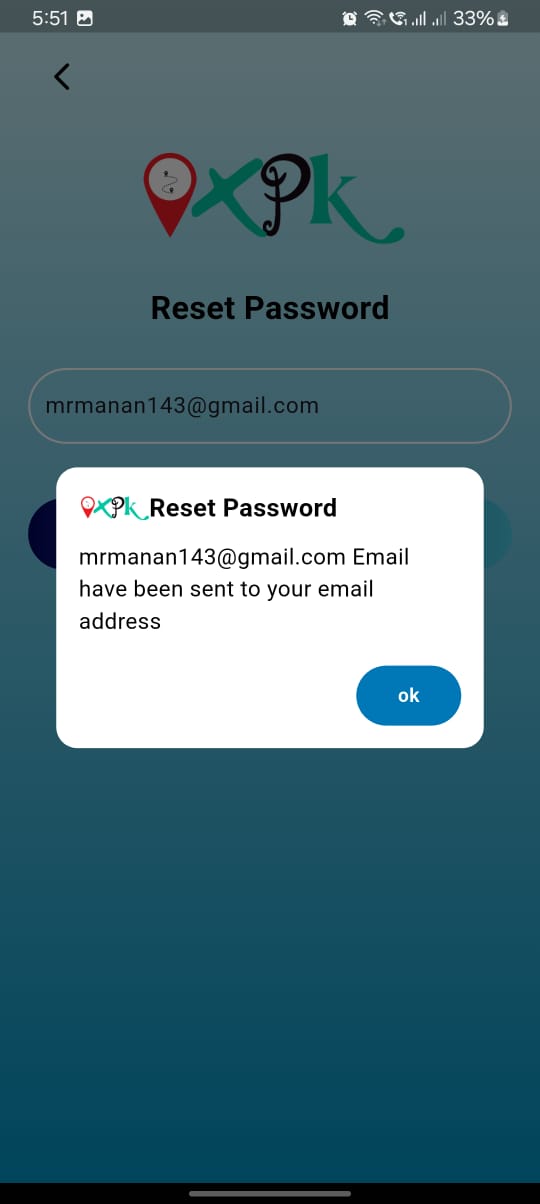
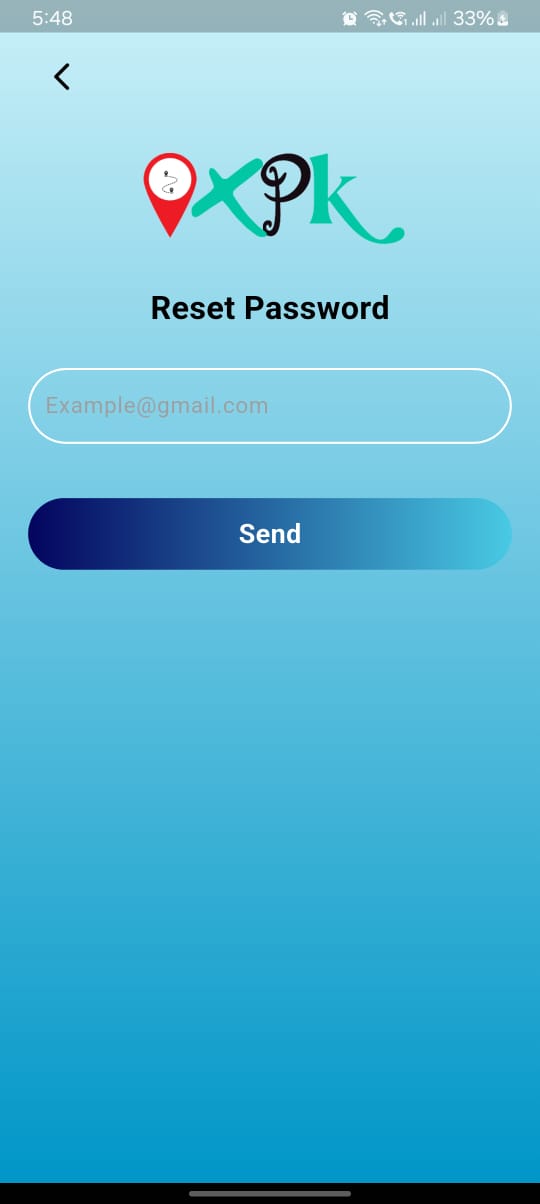
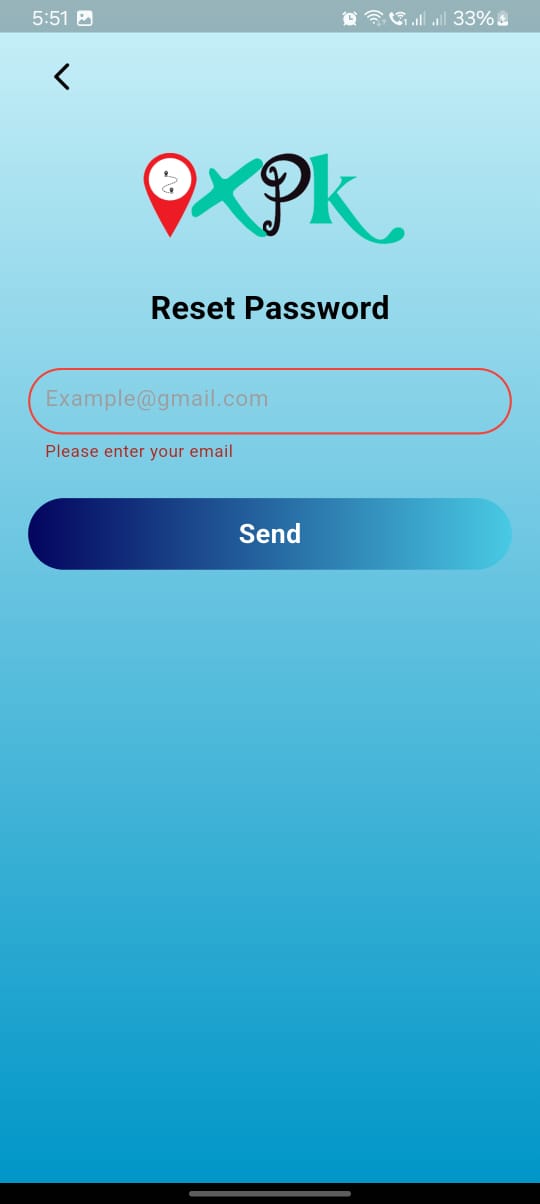


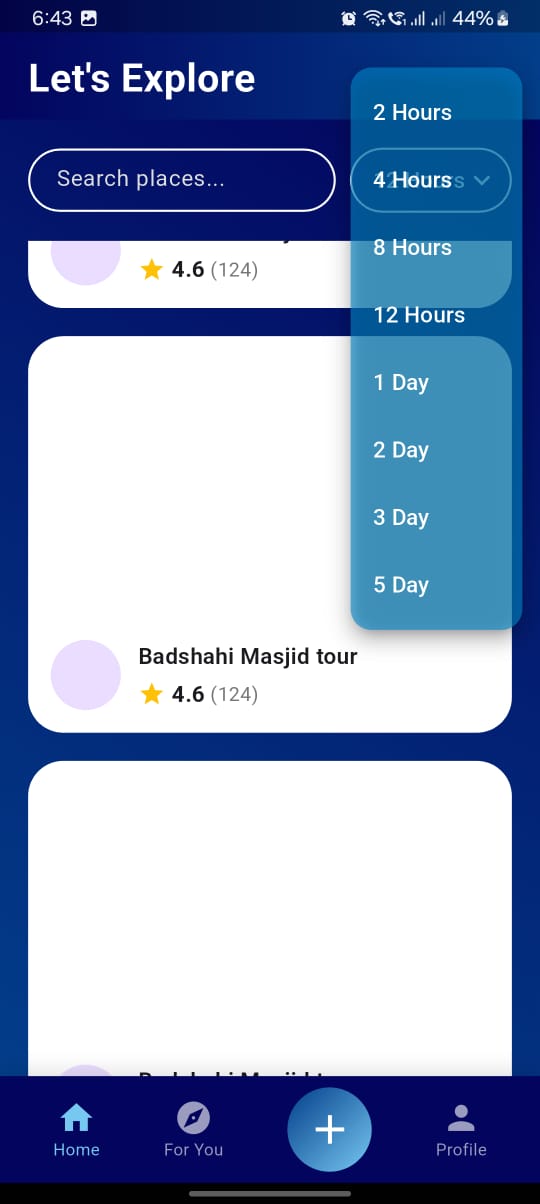
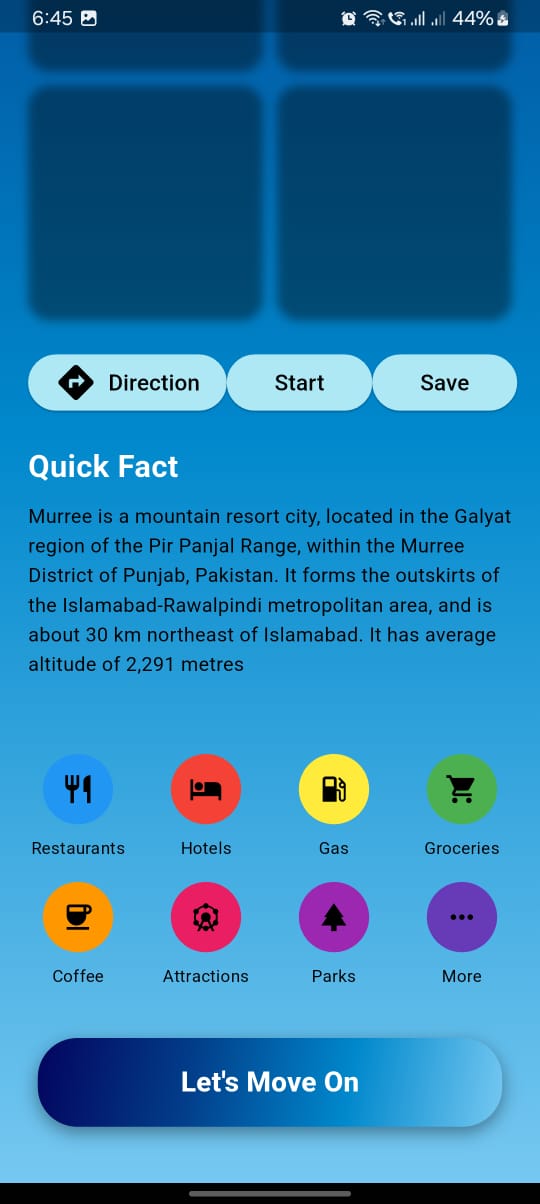
* **Verify Email:** After sign up, verify the email.



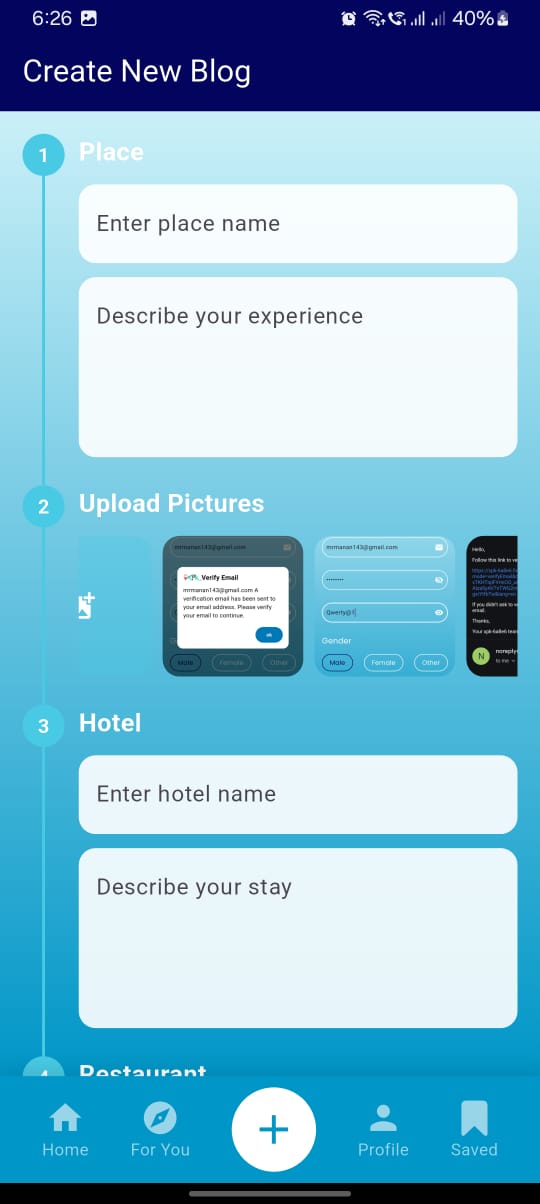
* **Login Module:** Ensures correct email/password validation and successful authentication.



* **Forgot Password:** Verifies that password reset email is sent successfully.
* **Home Screen & APIs:** Ensures correct place details are fetched from Google Places API.



* **Add Blogs:** We add the blog of our vacations in our profile

****