Test Document

for

Chalo Kart

(Golf Cart Management)

Prepared by

Group 13: **Group Name:** EE Got Latent

Trijal Srivastava	221144	trijals22@iitk.ac.in
Snehasis Satapathy	221070	ssnishasis22@iitk.ac.in
Gautam Arora	220405	garora22@iitk.ac.in
Naman Gupta	220686	namangupta22@iitk.ac.in
Divyam Agarwal	220376	divyamag22@iitk.ac.in
Udbhav Agarwal	221149	audbhav22@iitk.ac.in
Deham Rajvanshi	220335	dehamr22@iitk.ac.in
Saksham Parihar	220939	psaksham22@iitk.ac.in
Dharvi Singhal	220354	dharvis22@iitk.ac.in

Course: CS253

Mentor TA: Mr. Vikrant Chauhan

Date: April 5, 2025

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.0	Trijal Srivastava Snehasis Satapathy Gautam Arora Naman Gupta Divyam Agarwal Udbhav Agarwal Deham Rajvanshi Saksham Parihar Dharvi Singhal	The first version of the Software Test Document	05/04/2025

Introduction

2.1 Test Strategy

We utilized Flutter Test for unit and widget testing, ensuring that individual components of our app function correctly in isolation. This automated testing process helped us validate core functionalities and UI elements efficiently.

The integration testing was done manually. We carefully checked all parts of the application, ensuring everything worked well together and addressing potential user problems upfront.

2.2 Testing Timeline

We mainly tested our software after the implementation phase, especially using the Flutter testing framework for automation. However, we also did some manual testing while working on the implementation.

2.3 Testers

The developers themselves conducted the testing process. However, we made sure that the individual responsible for implementing a specific functionality did not perform the testing for that functionality.

2.4 Coverage Criteria

We used branch coverage for unit testing.

2.5 Tools used for testing

For unit and widget testing, we leveraged the flutter_test package, while the integration_test package was used for end-to-end testing of the app's functionality. To simulate external dependencies and isolate components during testing, we extensively used the Mockito framework. This included creating mock classes such as MockFirebaseAuth, MockDataSnapshot, and MockDatabaseReference to mock Firebase services and other dependencies effectively.

Unit Testing

3.1 Authentication

3.1.1 Logging in a User

Class: AuthService

Function: signIn()

Test Owner: Naman Gupta
Date: 3/04/2025
Test Description: Tests basic authentication flows, including sign-in with email/password, email verification, sign-out, and current user state management.
Test Results: The user can sign in with valid credentials and see clear messages for empty fields, wrong password, or non-existent email.

```dart
group('AuthService - Sign In', () {
    test('should return success when sign in is successful', () async {
      when(mockFirebaseAuth.signInWithEmailAndPassword(
        email: 'test@example.com',
        password: 'password123',
      )).thenAnswer((_) async => mockUserCredential);

      final result = await authService.signIn('test@example.com', 'password123');

      expect(result['success'], true);
      expect(result['data']['user_id'], 'test-uid');
      expect(result['data']['email'], 'test@example.com');
    });

    test('should return error when email is empty', () async {
      final result = await authService.signIn('', 'password123');

      expect(result['success'], false);
      expect(result['message'], 'Email and password cannot be empty');
    });

    test('should return error when password is empty', () async {
      final result = await authService.signIn('test@example.com', '');

      expect(result['success'], false);
      expect(result['message'], 'Email and password cannot be empty');
    });

    test('should handle FirebaseAuthException - user-not-found', () async {
      when(mockFirebaseAuth.signInWithEmailAndPassword(
        email: 'test@example.com',
        password: 'password123',
      )).thenThrow(
        FirebaseAuthException(code: 'user-not-found'),
      );

      final result = await authService.signIn('test@example.com', 'password123');

      expect(result['success'], false);
      expect(result['message'], 'No user found with this email.');
    });

    test('should handle FirebaseAuthException - wrong-password', () async {
      when(mockFirebaseAuth.signInWithEmailAndPassword(
        email: 'test@example.com',
        password: 'wrongpassword',
      )).thenThrow(
        FirebaseAuthException(code: 'wrong-password'),
      );

      final result = await authService.signIn('test@example.com', 'wrongpassword');

      expect(result['success'], false);
      expect(result['message'], 'Wrong password provided.');
    });
  });
```

3.1.2 Registering in a User

Class: AuthService

Function: signUp()

Test Owner: Gautam Arora
Date: 3/04/2025
Test Description: Tests user registration flow, including successful account creation, duplicate email handling, and password strength validation.
Test Results: The user can create an account with name, email, and password and gets notified for duplicate email, weak password, or invalid email format.

```
group('AuthService - Sign Up', () {
  test('should successfully create new user account', () async {
    when(mockFirebaseAuth.createUserWithEmailAndPassword(
      email: 'test@example.com',
      password: 'password123',
    )).thenAnswer((_) async => mockUserCredential);

    when(mockUser.updateDisplayName(any)).thenAnswer((_) async => {});
    when(mockUser.sendEmailVerification()).thenAnswer((_) async => {});

    final result = await authService.signUp(
      name: 'Test User',
      email: 'test@example.com',
      mobile: '1234567890',
      password: 'password123',
    );

    expect(result['success'], true);
    expect(result['data']['uid'], 'test-uid');
    expect(result['data']['email'], 'test@example.com');
    expect(result['data']['name'], 'Test User');
  });

  test('should handle email already in use', () async {
    when(mockFirebaseAuth.createUserWithEmailAndPassword(
      email: 'existing@example.com',
      password: 'password123',
    )).thenThrow(
      FirebaseAuthException(code: 'email-already-in-use'),
    );
```

```
    final result = await authService.signUp(
      name: 'Test User',
      email: 'existing@example.com',
      mobile: '1234567890',
      password: 'password123',
    );

    expect(result['success'], false);
    expect(result['message'], 'Email is already in use.');
  });

  test('should handle weak password', () async {
    when(mockFirebaseAuth.createUserWithEmailAndPassword(
      email: 'test@example.com',
      password: 'weak',
    )).thenThrow(
      FirebaseAuthException(code: 'weak-password'),
    );

    final result = await authService.signUp(
      name: 'Test User',
      email: 'test@example.com',
      mobile: '1234567890',
      password: 'weak',
    );

    expect(result['success'], false);
    expect(result['message'], 'Password is too weak.');
  });

  test('should handle invalid email', () async {
    when(mockFirebaseAuth.createUserWithEmailAndPassword(
      email: 'invalid-email',
      password: 'password123',
    )).thenThrow(
      FirebaseAuthException(code: 'invalid-email'),
    );

    final result = await authService.signUp(
      name: 'Test User',
      email: 'invalid-email',
      mobile: '1234567890',
      password: 'password123',
    );

    expect(result['success'], false);
    expect(result['message'], 'Invalid email address.');
  });
});
```

3.1.3 Email Verification

Class: AuthService
Function:sendEmailVerification(), verifyEmail(), resendVerification()
Location: test/services/auth_service_test.dart

Test Owner: Trijal Srivastava
Date: 3/04/2025
Test Description: Tests that users receive verification emails after signup, can verify their email addresses with verification codes, and can request new verification emails if needed, with appropriate success and error handling.
Test Results: User receives a verification email after signing up, can verify with the code, and request a new one if needed.

```dart
group('AuthService - Email Verification', () {
  test('should successfully send verification email', () async {
    final result = await authService.sendEmailVerification('test@example.com');

    expect(result['success'], true);
    expect(result['message'], 'Verification email sent successfully');
  });

  test('should successfully verify email', () async {
    final result = await authService.verifyEmail('test@example.com', '123456');

    expect(result['success'], true);
    expect(result['message'], 'Email verified successfully');
  });

  test('should successfully resend verification', () async {
    final result = await authService.resendVerification('test@example.com');

    expect(result['success'], true);
    expect(result['message'], 'Verification email sent successfully');
  });
});
```

3.1.4 Phone Verification

Class: AuthService
Function: verifyPhoneNumber(), verifyOTP()
Location: test/services/phone_verification_test.dart

Test Owner: Udbhav Agarwal

Date: 3/04/2025
Test Description: Tests OTP-based phone verification including successful verification and invalid code handling.
Test Results: User can verify phone number with a correct OTP and gets notified for an invalid code.

```dart
group('AuthService - Phone Verification', () {
  test('should successfully verify OTP', () async {
    final PhoneAuthCredential credential = PhoneAuthProvider.credential(
      verificationId: 'test-verification-id',
      smsCode: '123456',
    );

    when(mockFirebaseAuth.signInWithCredential(any))
        .thenAnswer((_) async => MockUserCredential());

    final result = await authService.verifyOTP('test-verification-id', '123456');

    expect(result['success'], true);
    expect(result['message'], 'OTP verified successfully');
  });

  test('should handle invalid OTP', () async {
    when(mockFirebaseAuth.signInWithCredential(any))
        .thenThrow(FirebaseAuthException(
      code: 'invalid-verification-code',
      message: 'Invalid OTP',
    ));

    final result = await authService.verifyOTP('test-verification-id', 'invalid');

    expect(result['success'], false);
    expect(result['message'], 'Invalid OTP');
  });
});
```

3.1.5 Password Reset

Class: AuthService
Function: requestPasswordReset()
Location: test/services/password_reset_test.dart

Test Owner: Divyam Agarwal

Date: 3/04/2025
Test Description: Tests password reset functionality including sending reset emails and handling various error cases like non-existent users.
Test Results: User can request a password reset email and gets notified for non-existent accounts or invalid email formats.

```dart
group('AuthService – Password Reset', () {
  test('should successfully send password reset email', () async {
    when(mockFirebaseAuth.sendPasswordResetEmail(
      email: 'test@example.com',
    )).thenAnswer((_) async => {});

    final result = await authService.requestPasswordReset('test@example.com');

    expect(result['success'], true);
    expect(result['message'], 'Password reset email sent successfully');
  });

  test('should handle non-existent user', () async {
    when(mockFirebaseAuth.sendPasswordResetEmail(
      email: 'nonexistent@example.com',
    )).thenThrow(
      FirebaseAuthException(code: 'user-not-found'),
    );

    final result = await authService.requestPasswordReset('nonexistent@example.com');

    expect(result['success'], false);
    expect(result['message'], 'No user found with this email.');
  });

  test('should handle invalid email', () async {
    when(mockFirebaseAuth.sendPasswordResetEmail(
      email: 'invalid-email',
    )).thenThrow(
      FirebaseAuthException(code: 'invalid-email'),
    );

    final result = await authService.requestPasswordReset('invalid-email');

    expect(result['success'], false);
    expect(result['message'], 'Invalid email address.');
  });
});
```

3.2 Cart Bookings

3.2.1 Ride Booking

Function: calculateFareAmountFromOriginToDestination() , obtainOriginToDestinationDirectionDetails(), sendNotificationToDriverNow(), saveRideRequestInformation()
Location: test/screens/booking_test.dart

Test Owner: Saksham Parihar

Date: 3/04/2025
Test Description: Tests verify that users can book rides with accurate fare calculations, get proper route details, and drivers receive notifications,
Test Results: The system calculates fares accurately, plans routes with directions and travel time, notifies drivers with ride details, and stores bookings with user info and timestamps.

```dart
test('Fare calculation test for different distances', () {
    // Test case 1: Short distance
    final shortTrip = DirectionDetailsWithPolyline(
      distance_value_in_meters: 2000, // 2 km
      duration_text_in_s: '600s', // 10 minutes
      e_points: 'test_points'
    );

    final shortTripFare = AssistantMethods.calculateFareAmountFromOriginToDestination(shortTrip);
    expect(shortTripFare, 3.5); // (2000/60 * 0.1) + (2000/1000 * 0.1) = 3.33 + 0.2 = 3.5

    // Test case 2: Medium distance
    final mediumTrip = DirectionDetailsWithPolyline(
      distance_value_in_meters: 5000, // 5 km
      duration_text_in_s: '900s', // 15 minutes
      e_points: 'test_points'
    );

    final mediumTripFare = AssistantMethods.calculateFareAmountFromOriginToDestination(mediumTrip);
    expect(mediumTripFare, 8.8); // (5000/60 * 0.1) + (5000/1000 * 0.1) = 8.33 + 0.5 = 8.8

    // Test case 3: Long distance
    final longTrip = DirectionDetailsWithPolyline(
      distance_value_in_meters: 10000, // 10 km
      duration_text_in_s: '1800s', // 30 minutes
      e_points: 'test_points'
    );

    final longTripFare = AssistantMethods.calculateFareAmountFromOriginToDestination(longTrip);
    expect(longTripFare, 17.7); // (10000/60 * 0.1) + (10000/1000 * 0.1) = 16.67 + 1 = 17.7
  });
```

Fare Calculation for Different Distance

```dart
test('Fare calculation handles zero distance', () {
    final zeroTrip = DirectionDetailsWithPolyline(
      distance_value_in_meters: 0,
      duration_text_in_s: '0s',
      e_points: ''
    );

    final fare = AssistantMethods.calculateFareAmountFromOriginToDestination(zeroTrip);
    expect(fare, 0.0);
  });

  test('Fare calculation handles null values', () {
    final nullTrip = DirectionDetailsWithPolyline(
      distance_value_in_meters: null,
      duration_text_in_s: null,
      e_points: null
    );

    expect(
      () => AssistantMethods.calculateFareAmountFromOriginToDestination(nullTrip),
      throwsA(isA<TypeError>()),
      reason: 'Should throw TypeError when distance is null'
    );
  });
```

Edge Case: Fare Calculation Handling for Zero Distance

```dart
test('Direction details calculation test', () async {
    final origin = LatLng(26.512507, 80.233355);
    final destination = LatLng(26.522507, 80.243355);

    final (directionDetails, polyline) = await AssistantMethods.obtainOriginToDestinationDirectionDetails(
      origin,
      destination
    );

    expect(directionDetails, isA<DirectionDetailsWithPolyline>());
    expect(polyline, isA<String>());
  });
```

Direction details calculation tests

```dart
test('Ride request information is saved correctly', () async {
      final mockUserModel = UserModel(
        name: "Test User",
        phone: "1234567890",
        email: "test@example.com"
      );

      // Verify pickup location is set
      expect(appInfo.userPickupLocation, isNotNull);
      expect(appInfo.userPickupLocation!.locationName, "Test Pickup");
      expect(appInfo.userPickupLocation!.locationLatitude, 26.512507);
      expect(appInfo.userPickupLocation!.locationLongitude, 80.233355);

      // Verify dropoff location is set
      expect(appInfo.userDropOffLocation, isNotNull);
      expect(appInfo.userDropOffLocation!.locationName, "Test Dropoff");
      expect(appInfo.userDropOffLocation!.locationLatitude, 26.522507);
      expect(appInfo.userDropOffLocation!.locationLongitude, 80.243355);

      // Verify database reference key is generated
      expect(mockDatabaseRef.key, isNotNull);
      expect(mockDatabaseRef.key, "test_ride_request");
    });
```

Testing for Ride Request Info being saved correctly

```dart
testWidgets('Driver notification is sent correctly', (WidgetTester tester) async {
    final mockClient = MockClient();
    final driverToken = "test_driver_token";
    final rideRequestId = "test_ride_request";

    await tester.pumpWidget(
      MaterialApp(
        home: ChangeNotifierProvider.value(
          value: appInfo,
          child: Container(),
        ),
      ),
    );

    final BuildContext context = tester.element(find.byType(Container));

    when(mockClient.post(
      any,
      headers: anyNamed('headers'),
      body: anyNamed('body'),
      encoding: anyNamed('encoding'),
    )).thenAnswer((_) async => http.Response('{"success": true}', 200));

    AssistantMethods.httpClient = mockClient;

    await AssistantMethods.sendNotificationToDriverNow(
      driverToken,
      rideRequestId,
      context
    );

    verify(mockClient.post(
      any,
      headers: anyNamed('headers'),
      body: anyNamed('body'),
      encoding: anyNamed('encoding'),
    )).called(1);
  });
```

Testing that Driver Receives the correct notification

3.3 User Analytics

3.3.1 Ride History

Function: fromSnapShot()
Test Owner: Deham Rajvanshi

Date: 3/04/2025
Test Description: Constructs a TripsHistoryModel from Firebase snapshot data.
Test Results: Trip details are saved accurately with all relevant info, and the app handles complete, partial, empty, and invalid data scenarios properly.

3.3.2 Viewing Past Bookings

Function: TripsHistoryScreen Widget Tests
Test Owner: Trijal Srivastava
Date: 03/04/2025
Test Description: Verifies the functionality of the trips history screen widget, including display, navigation, and theme handling.
Test Results: The user can view past rides with full details, sees a proper screen when no rides exist, can close the history screen easily, with support for light/dark mode and readable date formats.

```dart
testWidgets('TripsHistoryScreen shows list of trips', (WidgetTester tester) async {
    when(mockAppInfo.allTripsHistoryInformationList).thenReturn(mockTrips);

    await tester.pumpWidget(
      MaterialApp(
        home: ChangeNotifierProvider<AppInfo>.value(
          value: mockAppInfo,
          child: const TripsHistoryScreen(),
        ),
      ),
    );

    // Verify app bar
    expect(find.text('Trips History'), findsOneWidget);
    expect(find.byIcon(Icons.close), findsOneWidget);

    // Verify trips are displayed
    expect(find.text('Udbhav'), findsOneWidget);
    expect(find.text('Naman'), findsOneWidget);
    expect(find.text('Test Origin 1'), findsOneWidget);
    expect(find.text('Test Origin 2'), findsOneWidget);

    // Verify trip details
    expect(find.text('₹100'), findsOneWidget);
    expect(find.text('₹150'), findsOneWidget);
    expect(find.text('completed'), findsOneWidget);
    expect(find.text('cancelled'), findsOneWidget);
  });

```

Testing Widgets

```dart
testWidgets('TripsHistoryScreen handles empty trip list', (WidgetTester tester) async {
    when(mockAppInfo.allTripsHistoryInformationList).thenReturn([]);

    await tester.pumpWidget(
      MaterialApp(
        home: ChangeNotifierProvider<AppInfo>.value(
          value: mockAppInfo,
          child: const TripsHistoryScreen(),
        ),
      ),
    );

    // Verify app bar still shows
    expect(find.text('Trips History'), findsOneWidget);

    // Verify no trip items are shown
    expect(find.byType(Card), findsNothing);
  });

```

Edge Case : Empty Trip List

```dart
testWidgets('TripsHistoryScreen formats date correctly', (WidgetTester tester) async {
    when(mockAppInfo.allTripsHistoryInformationList).thenReturn(mockTrips);

    await tester.pumpWidget(
      MaterialApp(
        home: ChangeNotifierProvider<AppInfo>.value(
          value: mockAppInfo,
          child: const TripsHistoryScreen(),
        ),
      ),
    );

    // Verify date format (Mar 4, 2024 - 10:00 AM)
    expect(find.textContaining('Mar 4, 2024'), findsWidgets);
  });
```

Testing human Readable date formats

```dart
testWidgets('TripsHistoryScreen handles dark theme', (WidgetTester tester) async {
    when(mockAppInfo.allTripsHistoryInformationList).thenReturn(mockTrips);

    await tester.pumpWidget(
      MaterialApp(
        home: MediaQuery(
          data: MediaQueryData(platformBrightness: Brightness.dark),
          child: ChangeNotifierProvider<AppInfo>.value(
            value: mockAppInfo,
            child: const TripsHistoryScreen(),
          ),
        ),
      ),
    );

    // Verify dark theme colors are applied
    final scaffold = tester.widget<Scaffold>(find.byType(Scaffold));
    expect(scaffold.backgroundColor, Colors.black);
  });
```

Testing Dark Theme in the widget

3.4 Payment

3.4.1 Getting Wallet Balance

Function: Get Wallet Balance
Test Owner: Dharvi Singhal
Date: 05/04/2025

Test Description: Verifies the functionality of retrieving and handling wallet balance from user data.
Test Results: User is able to get their wallet balance successfully. With edge case handled as well

```
group('Wallet Balance Tests', () {
    test('Get wallet balance from valid user data', () {
      final userData = {
        'name': 'Test User',
        'email': 'test@example.com',
        'phone': '1234567890',
        'wallet_amount': '500'
      };

      when(mockSnapshot.value).thenReturn(userData);
      when(mockSnapshot.key).thenReturn('test-user-id');

      final userModel = UserModel.fromSnapshot(mockSnapshot);

      expect(userModel.wallet_amount, '500');
      expect(userModel.name, 'Test User');
      expect(userModel.email, 'test@example.com');
    });


```

```
test('Handle missing wallet balance', () {
      final userData = {
        'name': 'Test User',
        'email': 'test@example.com',
        'phone': '1234567890',
        // wallet_amount field missing
      };

      when(mockSnapshot.value).thenReturn(userData);
      when(mockSnapshot.key).thenReturn('test-user-id');

      final userModel = UserModel.fromSnapshot(mockSnapshot);

      expect(userModel.wallet_amount, null);
      expect(userModel.name, 'Test User');
      expect(userModel.email, 'test@example.com');
    });

```

Edge Case : Handle missing wallet balance

```
test('Handle empty wallet balance', () {
      final userData = {
        'name': 'Test User',
        'email': 'test@example.com',
        'phone': '1234567890',
        'wallet_amount': ''
      };

      when(mockSnapshot.value).thenReturn(userData);
      when(mockSnapshot.key).thenReturn('test-user-id');

      final userModel = UserModel.fromSnapshot(mockSnapshot);

      expect(userModel.wallet_amount, '');
      expect(userModel.name, 'Test User');
      expect(userModel.email, 'test@example.com');
    });
```

Edge Case : Handle Empty Wallet Balance

3.4.2 Updating Wallet Balance

Test Owner: Snehasis Satapathy
Date: 05/04/2025
Test Description: This test suite verifies the functionality of wallet balance updates in the Firebase Realtime Database.
Test Results: The user is able to update their wallet balance successfully., with edge cases also handled gracefully

```dart
group('Wallet Balance Update Tests', () {
    test('Successfully update wallet balance', () async {
      const String userId = 'test-user-id';
      const String newBalance = '1000';

      // Setup mock behavior
      when(mockReference.set(any)).thenAnswer((_) => Future.value());
      when(mockReference.once()).thenAnswer((_) => Future.value(mockEvent));

      // Initial user data
      final userData = {
        'name': 'Test User',
        'email': 'test@example.com',
        'phone': '1234567890',
        'wallet_amount': '500'
      };

      when(mockSnapshot.value).thenReturn(userData);
      when(mockSnapshot.key).thenReturn(userId);

      // Attempt to update wallet balance
      await mockReference
          .child('users')
          .child(userId)
          .child('wallet_amount')
          .set(newBalance);

      // Verify the update was called with correct parameters
      verify(mockReference.set(newBalance)).called(1);
    });

```

```
test('Handle invalid wallet balance update', () async {
      const String userId = 'test-user-id';
      const String invalidBalance = 'invalid_amount';

      when(mockReference.set(any))
          .thenThrow(Exception('Invalid wallet balance format'));

      // Attempt to update with invalid balance
      expect(
        () => mockReference
            .child('users')
            .child(userId)
            .child('wallet_amount')
            .set(invalidBalance),
        throwsException,
      );
    });
```

Edge Case : Invalid Wallet Balance Update

```
test('Update wallet balance with zero amount', () async {
      const String userId = 'test-user-id';
      const String zeroBalance = '0';

      when(mockReference.set(any)).thenAnswer((_) => Future.value());

      // Attempt to update with zero balance
      await mockReference
          .child('users')
          .child(userId)
          .child('wallet_amount')
          .set(zeroBalance);

      verify(mockReference.set(zeroBalance)).called(1);
    });
```

Edge Case: Wallet balance update with zero amount

3.4.3 Viewing Transaction History

Test Owner: Udbhav Agarwal
Date: 05/04/2025
Test Description: This test suite verifies the functionality of wallet transaction history in the Firebase Realtime Database.
Test Results: Transaction history is successfully created and correctly parsed

```dart
group('Wallet Transaction History Tests', () {
  test('Successfully fetch transaction history', () async {
    const String userId = 'test-user-id';

    // Mock transaction data
    final transactionData = [
      {
        'amount': '500',
        'type': 'credit',
        'timestamp': '2024-03-14 10:00:00',
        'description': 'Wallet recharge',
        'status': 'completed'
      },
      {
        'amount': '200',
        'type': 'debit',
        'timestamp': '2024-03-14 11:00:00',
        'description': 'Ride payment',
        'status': 'completed'
      }
    ];

    when(mockSnapshot.value).thenReturn(transactionData);
    when(mockReference.once()).thenAnswer((_) => Future.value(mockEvent));

    // Attempt to fetch transactions
    final event = await mockReference
        .child('users')
        .child(userId)
        .child('transactions')
        .once();

    final transactions = (event.snapshot.value as List).map((data) {
      final mockTransactionSnapshot = MockDataSnapshot();
      when(mockTransactionSnapshot.value).thenReturn(data);
      return WalletTransactionModel.fromSnapshot(mockTransactionSnapshot);
    }).toList();

    expect(transactions.length, 2);
    expect(transactions[0].amount, '500');
    expect(transactions[0].type, 'credit');
    expect(transactions[1].amount, '200');
    expect(transactions[1].type, 'debit');
  });
```

```
test('Handle empty transaction history', () async {
      const String userId = 'test-user-id';

      when(mockSnapshot.value).thenReturn([]);
      when(mockReference.once()).thenAnswer((_) => Future.value(mockEvent));

      final event = await mockReference
          .child('users')
          .child(userId)
          .child('transactions')
          .once();

      final transactions = (event.snapshot.value as List).map((data) {
        final mockTransactionSnapshot = MockDataSnapshot();
        when(mockTransactionSnapshot.value).thenReturn(data);
        return WalletTransactionModel.fromSnapshot(mockTransactionSnapshot);
      }).toList();

      expect(transactions.length, 0);
    });
```

Integration Testing

4.1 Authentication

4.1.1 Registering a User

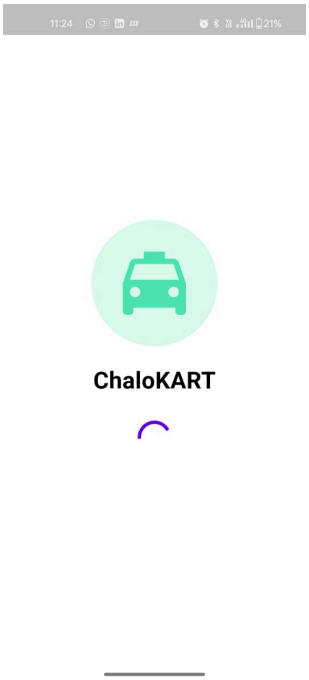
Module Details: Here we test the integration of the registration process for new users, providing the necessary functionality to create an account.
Test Owner: Saksham Parihar
Test Date: 01/04/2025
Test Results:

- **Test Case 1:** Invalid phone number or email address is entered.
  Result: Error messages are displayed in red text below the respective entry fields:
  - ”Please enter a valid phone number” is displayed if an invalid phone number format is entered.
  - ”Please enter a valid email” is displayed if an invalid email address format is entered.
- **Test Case 2:** Valid name, phone number, and already registered email address are entered in the respective fields.
  Result: An alert dialog is triggered, notifying the user of an existing profile associated with the provided email address.
- **Test Case 3:** Valid name, phone number, and unregistered email address are entered in the respective fields.
  Result: Upon successful registration, the user is directed to the Sign in page. Also, an alert is shown “Account created successfully”

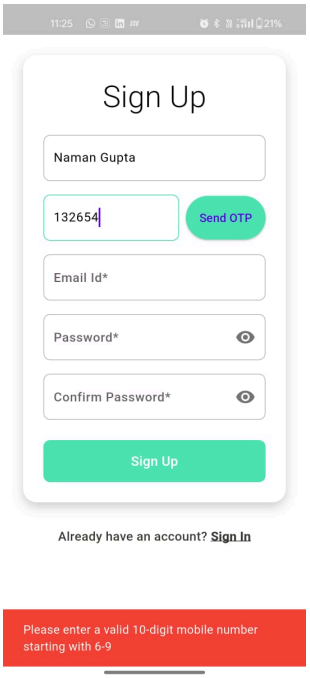
4.1.2 Logging in a User / Driver

Module Details: This module is the integration of all the components of the login page.
Test Owner: Gautam Arora
Test Date: 03/04/2025
Test Results:

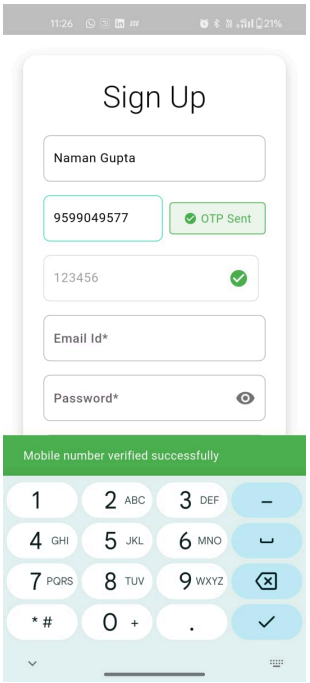
- **Test Case 1:** If the user has not signed up and attempts to log in with an unregistered email ID and password.
  Result: A flutter error is displayed that conveys that no such user exists.
- **Test Case 2:** If the user has not Signed up and attempts to Sign up.
  Result: The user can click on the “First time here? Create Account” link at the bottom of the screen, redirecting them to the registration page.
- **Test Case 3:** Invalid email ID is entered, such as incorrect formatting or other issues.



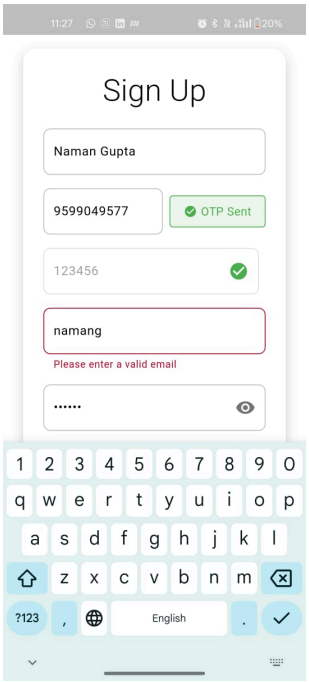
(a) Loading Screen



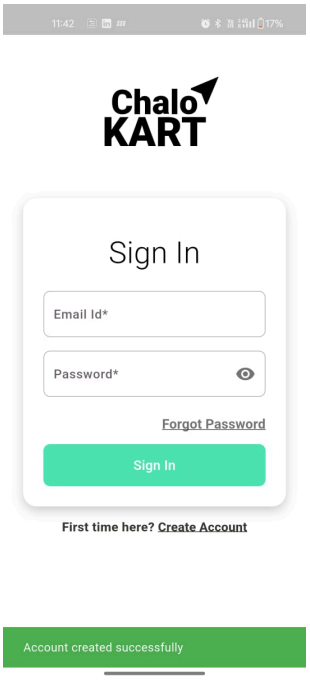
(b) Invalid Phone number



(c) Phone number verified via OTP



(d) Invalid Email ID



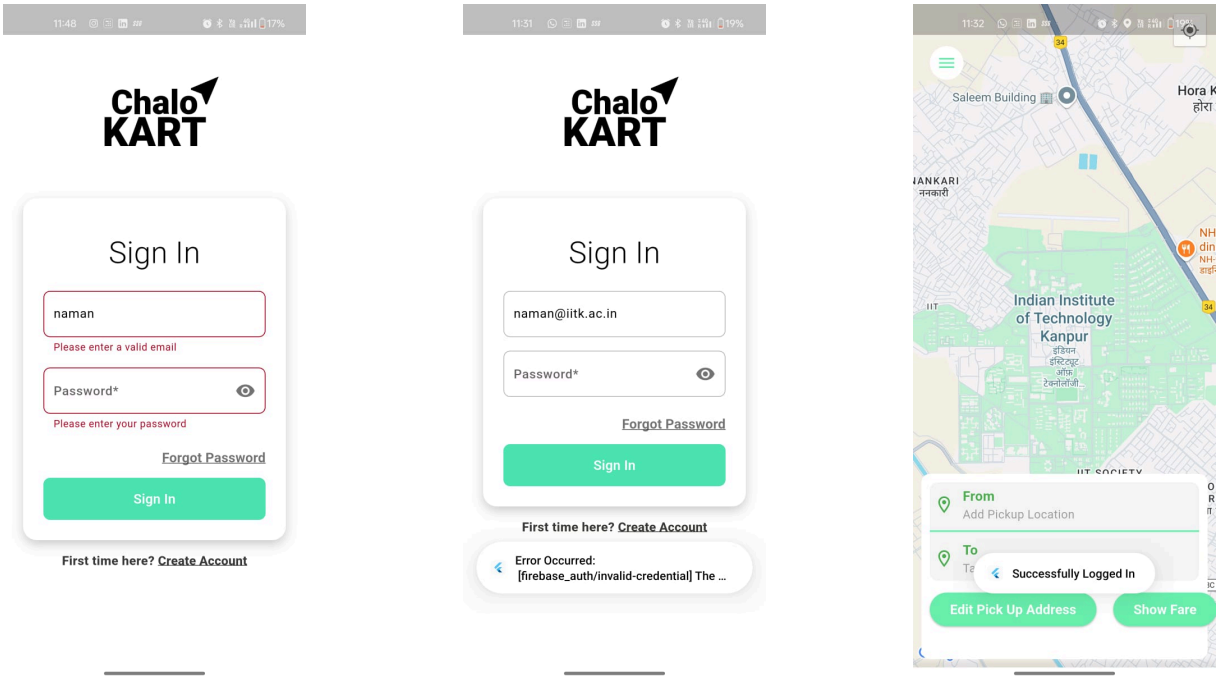
(b) Successfully Registered

Figure 1: Registration

Result: An error message in red text is displayed below the email input field, indicating ”Please enter a valid email”

- **Test Case 4:** Incorrect password is entered with a registered email ID.
  Result: A flutter error is displayed.
- **Test Case 5:** The user doesn’t remember the password.
  Result: Clicking on ”Forgot Password” redirects to the Forgot Password page. On entering the correct email address, the user receives a email on the registered email id to reset the password.
- **Test Case 6:** Registered email ID and password are entered, and the Login button is clicked.
  Result: The user is successfully redirected to the Homepage of ChaloKart.

Additional Comments: Test case 1, Test case 4 show flutter errors. They should be changed with explicit user-friendly error messages on screen.



(a) Invalid Email or Password (b) Incorrect Password (c) Successfully logged in

Figure 2: Logging In

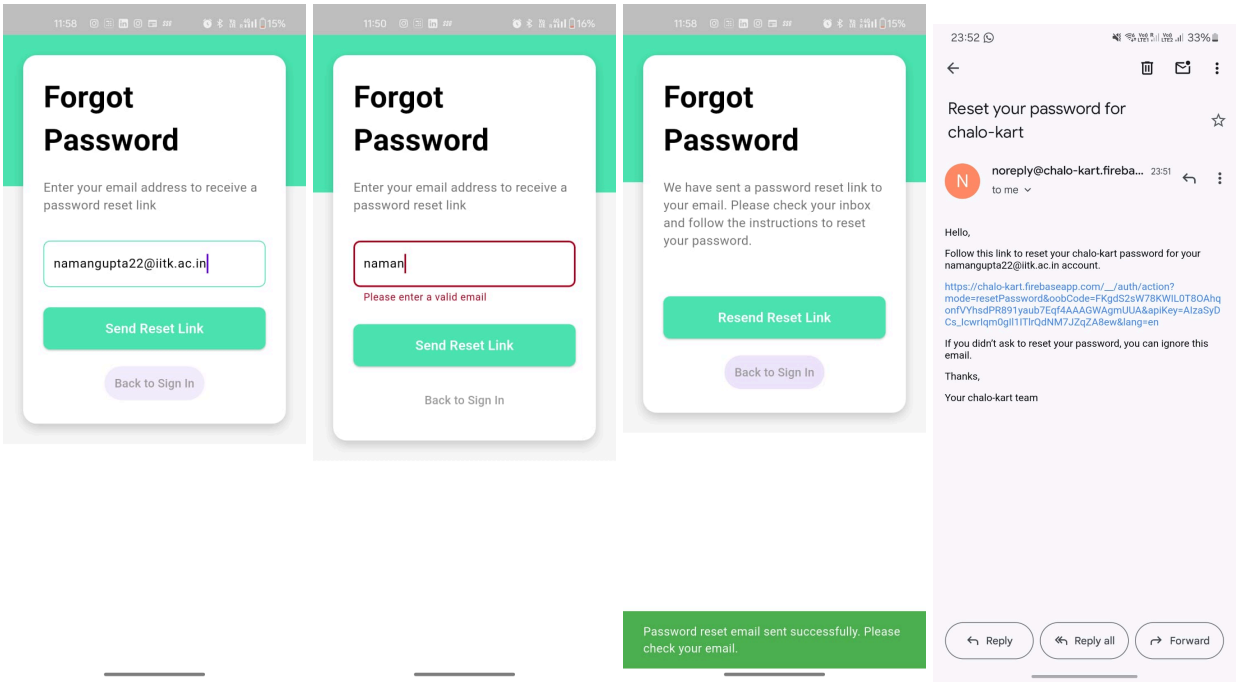
4.1.3 Forgot Password

Module Details: This module integrates the forgot password page and all the components that are necessary for the user to update their password.
Test Owner: Naman Gupta
Test Date: 03/04/2025
Test Results:

- **Test Case 1:** User initiates a password reset by providing a valid email address and clicking the ”Send a Link” button.
  Result: After submitting the request, the user is redirected to a confirmation page that advises them to check their email for a password reset link. At the same time, an email is sent to the specified address, containing a secure link for resetting the password. When the user clicks this link, they are taken to a dedicated page where they can enter a new password. Once submitted, the system updates the user's password accordingly.
- **Test Case 2:** User enters an invalid email address format

  Result: An error message in red is displayed below the entry field, alerting the user to enter a valid email address.

Additional Comments: None



(a) Forget Password Page (b) Invalid email (c) Link Sent Alert (d) Email received

Figure 3: Forgot Password

4.2 Ride

4.2.1 Start Ride

Module Details: This module integrates all the components necessary for starting a ride.
Test Owner: Divyam Agarwal
Test Date: 03/04/2025
Test Results:

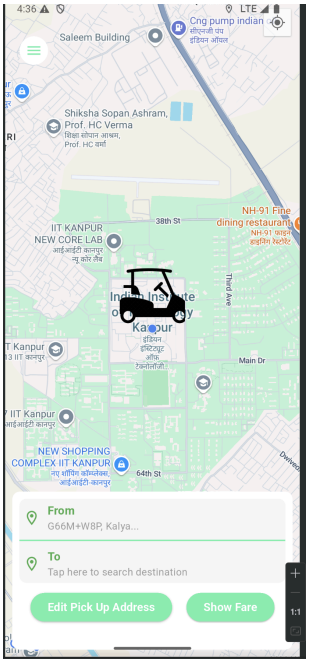
- **Test Case 1:** Upon selecting the pickup and drop location functionality from the home screen of the application, ride fare and choose cart will be shown
  Result: It works as expected. We see options to choose a cart.

- **Test Case 2:** On selecting the cart, it will show the expected time of arrival of the cart

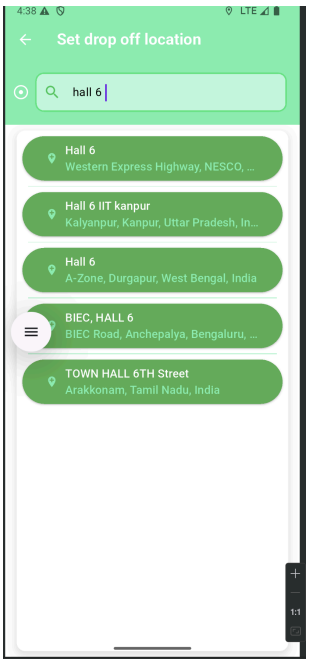
  Result: The driver receives an alert of the ride, and the user is able to see the cart on the map.

- **Test Case 3:** Once the ride is started, it should display the time of arrival.

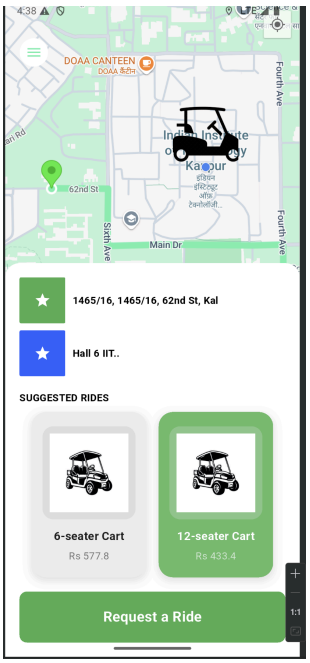
  Result: The screen displays the time of arrival correctly



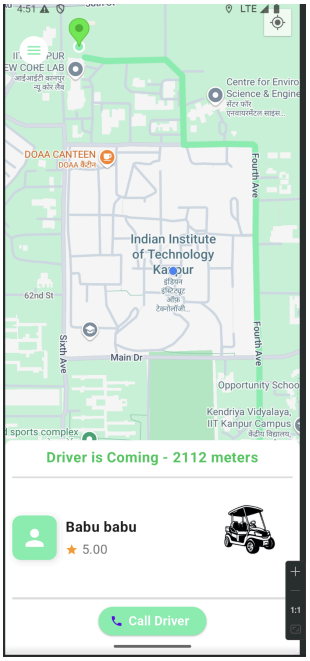
(a) Driver carts are shown



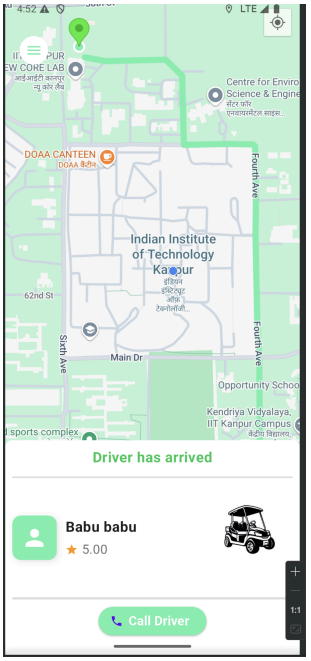
(b) Locations are chosen



(c) Cart is chosen



(a) The driver coming is shown

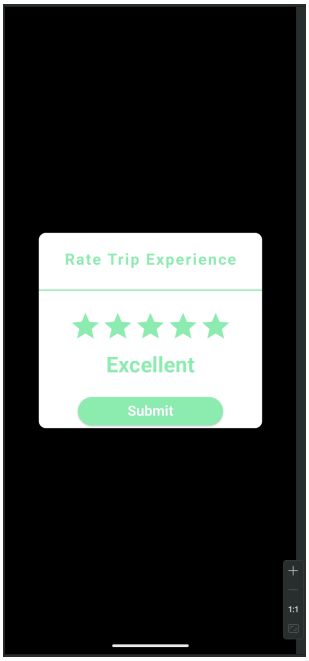


(b) Driver arrived updated.

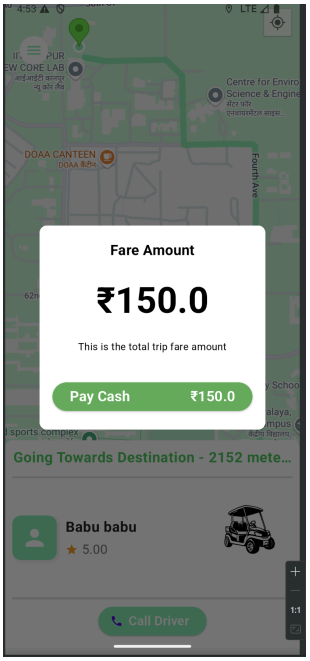
4.2.2 End Ride

Module Details: This module integrates all the components necessary for ending a ride.
Test Owner: Snehasis Satapathy
Test Date: 04/04/2024
Test Results:

- **Test Case 1:** Upon clicking the “End Ride” button from the drivers side, user is shown the fare.
  Result: The fare amount is displayed on the screen and this amount can be deducted from the wallet of the user.
- **Test Case 2:** Driver rating page shows up
  Result: The user can give rating to the driver of the just ended ride.



(a) Trip Rating Page



(b) Trip fare page

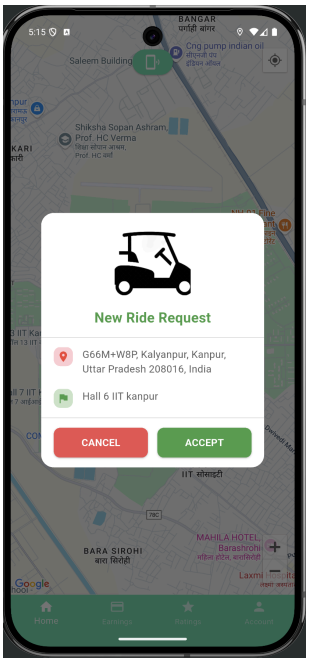
4.3 Driver Side Ride Request / End

Module Details: This module integrates the driver side functionality.

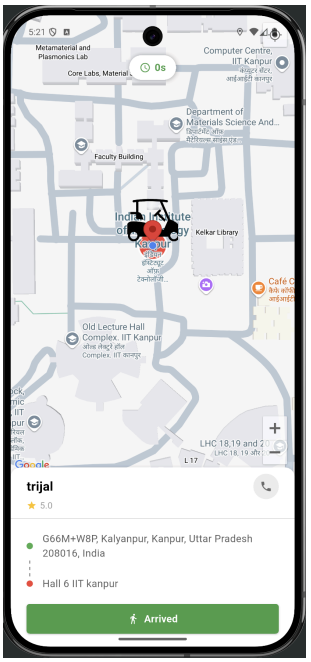
Test Owner: Deham Rajvanshi
Test Date: 03/04/2025
Test Results:

- **Test case 1:** Driver can accept or reject ride request.
  Result: Driver is shown with a prompt every time there is a new ride request. On accepting he is shown where to head.
- **Test case 2:** Driver can end the ride and is shown the amount.
  Result: The driver is shown with the trip fare. And then he is redirected to the home page.
- **Test case 3:** Driver can see the amount earned till now.
  Result: The driver can navigate to the page to see his earnings.

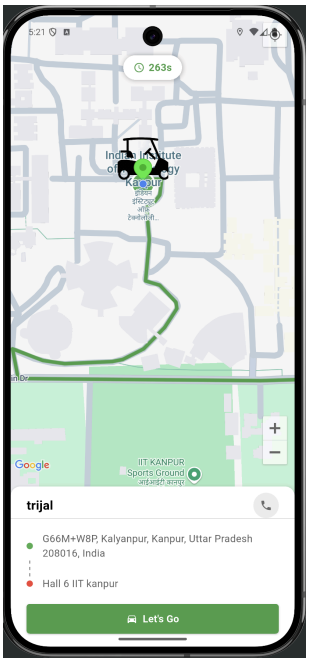
Additional Comments: Recent Transactions unable to load.



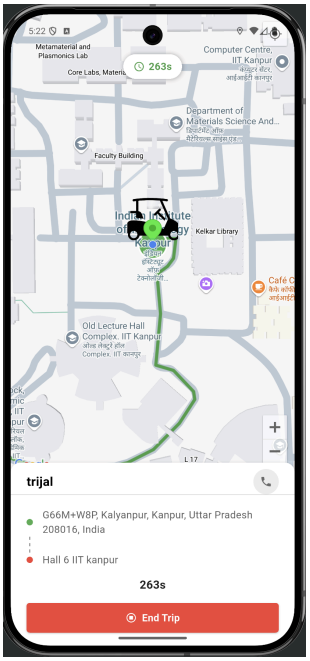
(a) Ride Request Page



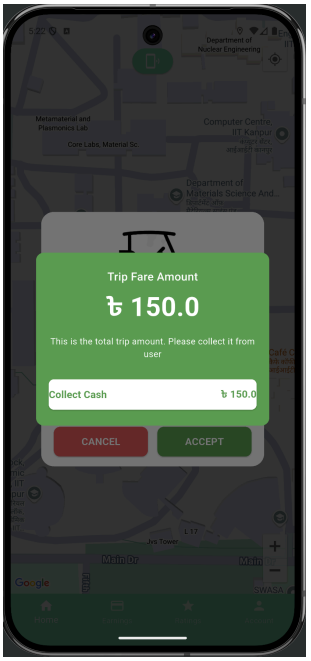
(b) Drivers current location



(c) Where driver is headed.



(d) End Trip Prompt



(e) Driver earning for the ride

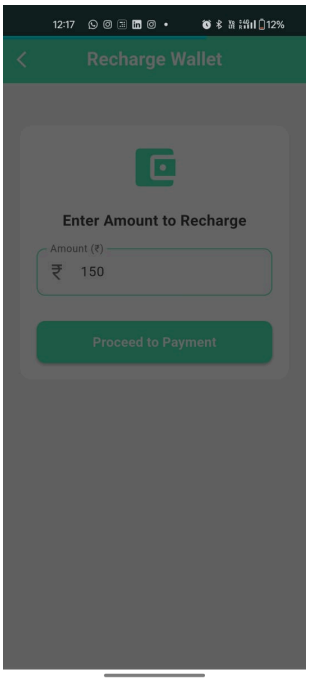
4.4 Wallet: View Balance and Add Funds

Module Details: This module integrates the wallet functionality crucial to making payments for each ride.

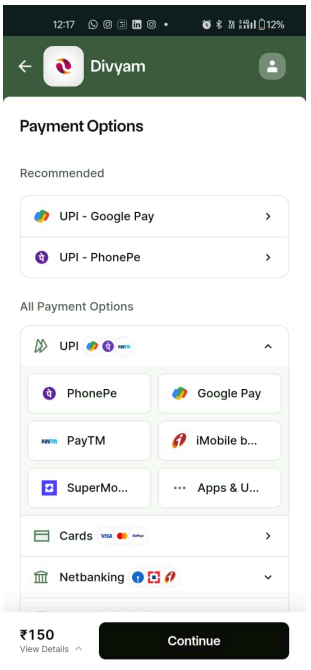
Test Owner: Naman Gupta
Test Date: 02/04/2025
Test Results:

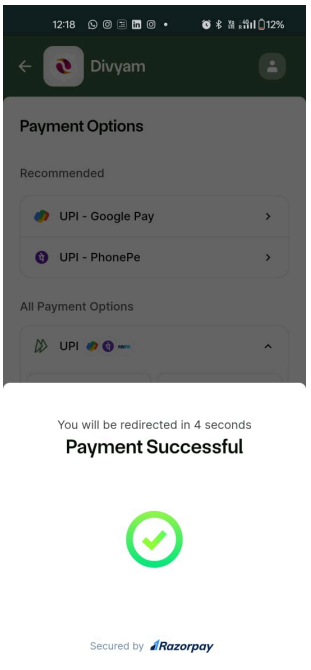
- **Test case 1:** User navigates to the wallet section from the sidebar.
  Result: User is shown with the current balance in his wallet.
- **Test case 2:** User wishes to add funds to his wallet.
  Result: A payment gateway via 'Razorpay' is shown through which he can add funds to the wallet. There are options to use UPI, Cards and Net Banking

Additional Comments: Recent Transactions unable to load.

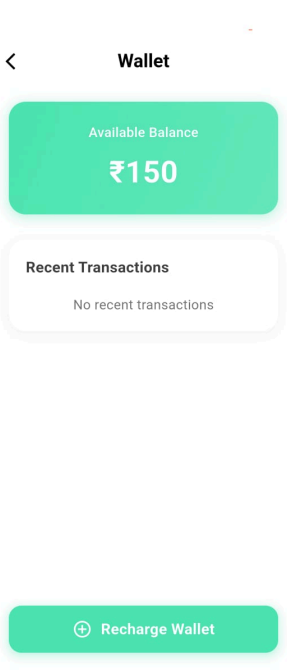


(a) Adding Rs150 to wallet





(b) and (c) Payment Gateway Pages



(d) Wallet added with amount

4.5 Ride History Page

Module Details: This module integrates all components for viewing past ride logs.
Test Owner: Divyam Agarwal
Test Date: 03/04/2025
Test Results:

- **Test Case 1:** A user needs to be shown all the rides he has taken in the past.
  Result: History Page is displayed with brief information of all the past rides taken by the specific user.

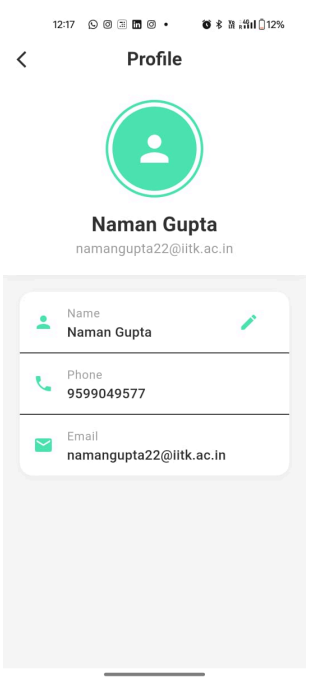
Additional Comments: None

4.6 View Profile Page and Edit Profile

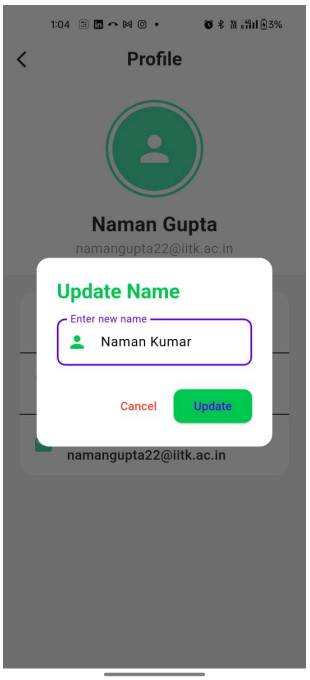
Module Details: This module integrates a method for accessing information about a user and editing it.
Test Owner: Trijal Srivastava
Test Date: 02-04-2025
Test Results:

- **Test Case 1:** A user navigates to the Edit Profile page.
  Result: Profile Page is displayed with their Name, Email Id and Phone number.
  Test Case 2: A user attempts to change their Name.

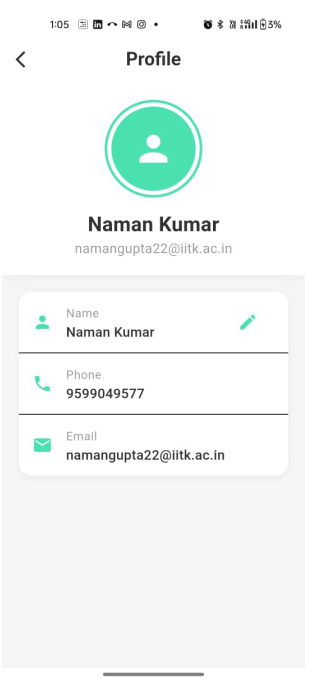
Result: The feature to change Name can be accessed by clicking besides the Name.



(a) Profile Page View



(b) Update Name Prompt



(c) Name Updated Successfully

System Testing

5.1 Functional Requirements

5.1.1 Launches Successfully

Requirement: App launches properly, and all images and text are rendered correctly.
Test Owner: Udbhav Agarwal
Date: 02/04/2025
Test Results: App launched properly with all text and images rendered, along with a map of IIT Kanpur on Home page as expected.
Additional Comments: It might be needed to clear the cache/clear data of the app when testing on an actual android device.

5.1.2 Registration and Authentication of User/Driver

Requirement: App allows new users to register while enabling existing users to log in.
Test Owner: Saksham Parihar
Date: 01/04/2025
Test Results: The login feature was tested with existing user credentials, successfully granting access when valid details were provided. The “forgot password” function allowed login with a newly set password and correctly identified unregistered emails. The “sign-up” process redirected new users to the register page. Attempts to register with an existing email triggered an alert stating, “Profile with this email ID already exists.”

5.1.3 In-App Wallet Service

Requirement: An In-App Wallet service for users to pay their fare and booking fee.
Test Owner: Deham Rajvanshi
Date: 30/03/2025
Test Results: For Users, the wallet is displayed successfully, showing the correct balance along with options to add funds or view transaction history. We were able to successfully add funds to the wallet by selecting the ”Add Balance” option, entering a valid amount in the input box, and completing the payment through the Razorpay Payment gateway.
Additionally, after any transaction, whether it’s adding funds or paying fare, the transaction log was NOT accurately updated. It needs to be fixed.
Additional Comments: None

5.1.4 Trip History

Requirement: App correctly displays the past ride records of the user.
Test Owner: Snehasis Satapathy
Date: 02/04/2025
Test Results: After the ride ends, its information is automatically stored in the past rides database on firebase, and the same is displayed on the trip history section accessed by the user.
Additional Comments: None

5.1.5 User Profile and Logout

Requirement: Users can see their Information, Edit it and Logout.
Test Owner: Dharvi Singhal
Date: 01/04/2025
Test Results: We were able to view the correct Name, registered Email ID, Phone Number of the user at My Profile page. Logging out was also successful, as we were redirected to the ”Login” or ”Register” page. Edit profile also worked as expected.
Additional Comments: None

5.2 Non-Functional Requirements

5.2.1 Performance

Requirements Test Owner: Dharvi Singhal
Date: 02/05/2025
Test Description: This test case is used to check the response time of the ChaloKart backend server for driver operations.
Test Results: The response time of critical driver-related APIs (login, order acceptance, location updates) is less than 1 second.

Additional Comments: The response time of the backend server APIs is less than 1 second only when the server is already active. After cold starts or during peak usage periods, initial response times may increase to 2-3 seconds.

5.2.2 Security Requirements

Requirements Test Owner: Gautam Arora
Date: 02/05/2025

- All user passwords are encrypted before storing them in the database. The user's password is never stored in plain text.
- All payments are processed through Razorpay's secure payment gateway to ensure transaction security.
- User authentication implements multi-factor authentication for sensitive operations.
- Data at rest is encrypted using industry-standard encryption algorithms implemented in Firebase.

Conclusion

How Effective and exhaustive was the testing?

The testing process for ChaloKart was comprehensive, covering multiple layers of the application:

- Unit tests achieved thorough coverage across models, services, and utility functions, with robust mocking of external dependencies including Firebase services.
- Integration tests successfully validated critical flows including authentication, ride booking/completion, wallet functionality, and user profile management.
- System testing confirmed that all functional requirements were met, including successful app launch, user registration, in-app wallet services, ride history tracking, and profile management.
- Test cases were well-structured and covered both ideal scenarios and edge cases such as invalid inputs, partial data, and various state transitions.

While the testing was extensive, there are areas for future expansion, particularly in automated end-to-end testing of complete user journeys.

Which components have not been tested adequately?

- Some of the components which involved mocking have not been tested using an automation suite.
- We haven't tested our required hardware components, ie. drivers actual location and cart tracking completely.

What difficulties have you faced during testing?

- The high degree of interconnection between different modules and views made integration testing challenging.
- Team members' limited familiarity with testing tools caused initial delays as time was needed to research and learn effective testing methodologies.
- Some error messages in the authentication process showed Flutter errors rather than user-friendly messages, requiring additional refinement.

How could the testing process be improved?

- Testing process could be improved by using automated testing for the frontend, as we did for the backend.
- Begin testing as early as possible in the development process to identify issues early on, allowing more time for fixing them before release.

Group Log

NOTE: Every team member worked regularly on assigned tasks, collaborating whenever needed. This log only covers online/in-person meetings and significant discussions.

S.No	Date	Timings	Venue	Description
1	28/03/2025	21:30 to 23:00	Google Meet	Discussed about possible bugs in the app and features we need to update.
2	31/03/2025	11:00 to 14:00	RM Building	Discussed about the problems we were facing in implementing the added features.
3	01/04/2025	21:30 to 22:00	Discord	Shared progress with each other and decided what to do next.
4	02/04/2025	14:30 to 17:00	RM Building	Started the making of the 2 documents
5	03/04/2025	17:30 to 18:30	Google Meet	Fixed basic bugs encountered and moved towards finishing the documents.
6	04/04/2025	21:00 to 23:30	Library	Finalised the documents