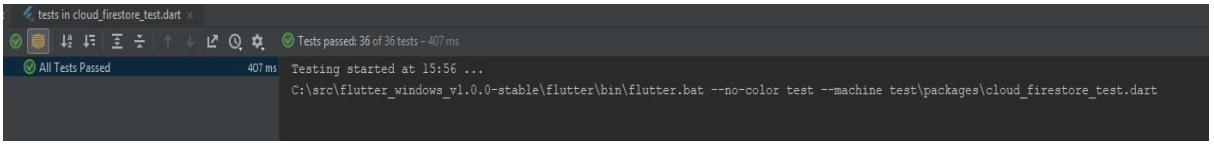
Testing Documentation

UI / Unit Testing - Mockito/Widget tester
User Acceptance Testing - Focus Group

Mockito/Widget testing

1. Cloud_Firestore: 36/36 tests passed



This test ensures transactions are recorded on the Cloud database.

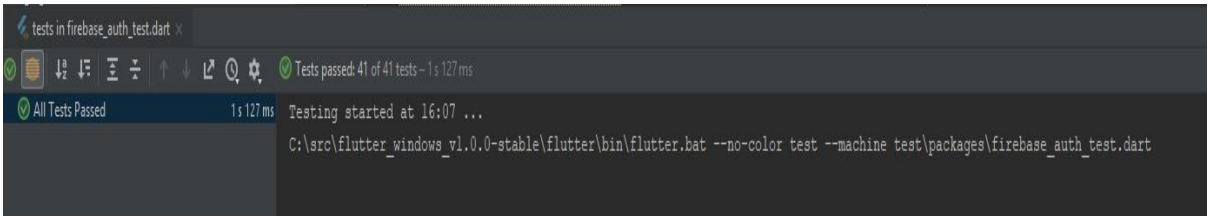
```
test('multiple apps', () async {
  expect(Firestore.instance, equals(Firestore()));
  final FirebaseApp app = FirebaseApp(name: firestore.app.name);
  expect(firestore, equals(Firestore(app: app)));
});

group('Transaction', () {
  test('runTransaction', () async {
    final Map<String, dynamic> result = await firestore.runTransaction(
        (Transaction tx) async {},
        timeout: const Duration(seconds: 3));

    expect(log, <Matcher>[
      isMethodCall('Firestore#runTransaction', arguments: <String, dynamic>{
        'app': app.name,
        'transactionId': 0,
        'transactionTimeout': 3000
      }),
    ]);
    expect(result, equals(<String, dynamic>{'1': 3}));
  });
```

```dart
test('get', () async {
  final DocumentReference documentReference =
      firestore.document('foo/bar');
  final DocumentSnapshot snapshot =
      await transaction.get(documentReference);
  expect(snapshot.reference.firestore, firestore);
  expect(log, <Matcher>[
    isMethodCall('Transaction#get', arguments: <String, dynamic>{
      'app': app.name,
      'transactionId': 0,
      'path': documentReference.path
    })
  ]);
});
```

2. Firebase_Auth: 41/41 tests passed



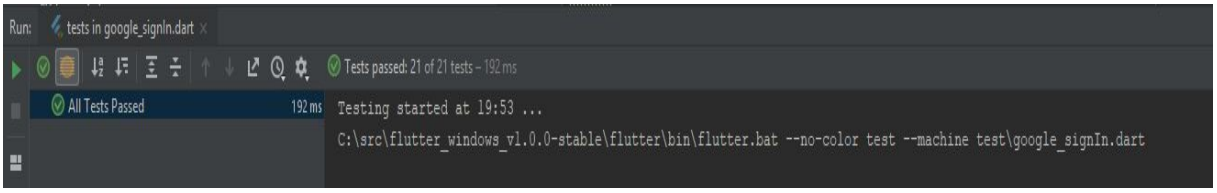
In these tests, we made sure our firebase registration is working correctly along with the login in page for returning users.

```
test('fetchSignInMethodsForEmail', () async {
  final List<String> providers =
  await auth.fetchSignInMethodsForEmail(email: kMockEmail);
  expect(providers, isNotNull);
  expect(providers.length, 0);
  expect(
    log,
    <Matcher>[
      isMethodCall(
        'fetchSignInMethodsForEmail',
        arguments: <String, String>{
          'email': kMockEmail,
          'app': auth.app.name
        },
        ),
    ],
    );
});
```

```
test('EmailAuthProvider reauthenticateWithCredential', () async {
  final FirebaseUser user = await auth.currentUser();
  log.clear();
  final AuthCredential credential = EmailAuthProvider.getCredential(
    email: kMockEmail,
    password: kMockPassword,
    );
  await user.reauthenticateWithCredential(credential);
  expect(
    log,
    <Matcher>[
      isMethodCall(
        'reauthenticateWithCredential',
        arguments: <String, dynamic>{
          'app': auth.app.name,
          'provider': 'password',
          'data': <String, String>{
            'email': kMockEmail,
            'password': kMockPassword,
          }
        },
        ),
    ],
    );
```

```dart
test('createUserWithEmailAndPassword', () async {
  final FirebaseUser user = await auth.createUserWithEmailAndPassword(
    email: kMockEmail,
    password: kMockPassword,
    );
  verifyUser(user);
  expect(
    log,
    <Matcher>[
      isMethodCall(
        'createUserWithEmailAndPassword',
        arguments: <String, String>{
          'email': kMockEmail,
          'password': kMockPassword,
          'app': auth.app.name,
        },
        ),
    ],
    );
});
```

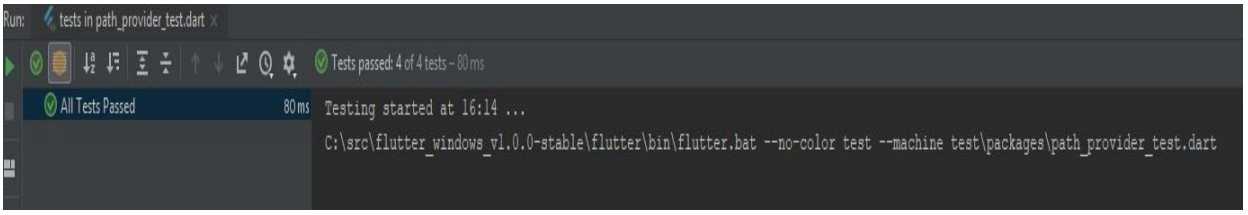
3. Google_SignIn: 21/21 tests passed



Our tests for google sign in ensured a valid google address was given, Sign in could be completed.

```dart
test('signIn', () async {
  await googleSignIn.signIn();
  expect(googleSignIn.currentUser, isNotNull);
  expect(
    log,
    <Matcher>[
      isMethodCall('init', arguments: <String, dynamic>{
        'signInOption': 'SignInOption.standard',
        'scopes': <String>[],
        'hostedDomain': null,
      }),
      isMethodCall('signIn', arguments: null),
    ], // <Matcher>[]
  );
});
```

4. Path_provider: 4/4 tests passed

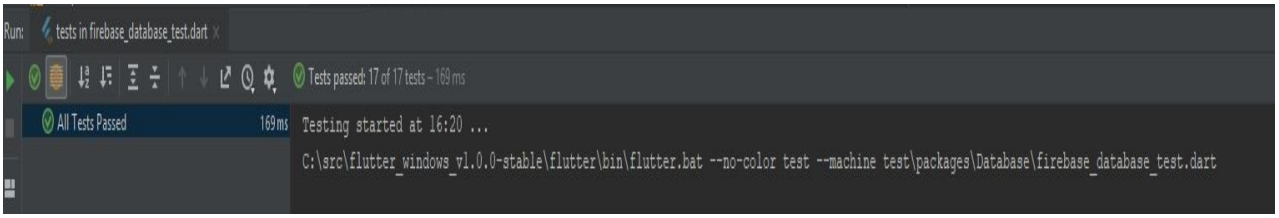


We tested our path provider to ensure the location of file systems were retrieved correctly.

```dart
test('getApplicationDocumentsDirectory test', () async {
  response = null;
  final Directory directory = await getApplicationDocumentsDirectory();
  expect(
    log,
    <Matcher>[
      isMethodCall('getApplicationDocumentsDirectory', arguments: null)
    ],
    );
  expect(directory, isNull);
});

test('TemporaryDirectory path test', () async {
  final String fakePath = "/foo/bar/baz";
  response = fakePath;
  final Directory directory = await getTemporaryDirectory();
  expect(directory.path, equals(fakePath));
});
```

5. Firebase_database



We conducted more tests for our database to see how it handled errors while also testing the reliability of updating it.

```
test('update', () async {
  final dynamic value = <String, dynamic>{'hello': 'world'};
  await database.reference().child("foo").update(value);
  expect(
    log,
    <Matcher>[
      isMethodCall(
        'DatabaseReference#update',
        arguments: <String, dynamic>{
          'app': app.name,
          'databaseURL': databaseURL,
          'path': 'foo',
          'value': value,
        },
        ),
    ],
    );
```

```
});
test('observing error events', () async {
  mockHandleId = 99;
  const int errorCode = 12;
  const String errorDetails = 'Some details';
  final Query query = database.reference().child('some path');
  Future<void> simulateError(String errorMessage) async {
    await BinaryMessages.handlePlatformMessage(
      channel.name,
      channel.codec.encodeMethodCall(
        MethodCall('Error', <String, dynamic>{
          'handle': 99,
          'error': <String, dynamic>{
            'code': errorCode,
            'message': errorMessage,
            'details': errorDetails,
          },
        }),
        ),
          (_) {},
```

User Testing With Focus Group

In keeping with the ISO / IEEE standards for user testing, we decided to carry out a focus group our target audience. Our focus group was comprised of four students along with 2 people between the ages of 30-50 and another participant who was 65+. All participants used public transport regularly and used a leap card as payment.

Each user was given dummy data to input when logging in and synced fake test accounts. The test was carried out to gauge the reaction of users and gain valuable feedback which could be used to improve our performance and design.

Task 1:
Users are asked to register an account with the dummy data provided, describe first impressions on opening the app and interacting with the login page.

Feedback:
Overall feedback was very positive. Users liked the minimalist UI design and the simplicity of the login and registration process. One user recommended using a larger font for the text fields during the login and registration process.

Changes:
After receiving the above feedback we increased the size of the text font to ensure our application was accessible to a wide of a range of users as possible.

Task 2:
Users were asked to attempt to purchase their ticket from the home page of the application. They were given test payment information to complete this transaction.

Feedback:

Users enjoyed the layout and the colour scheme of the user interface of the main menu. A few users found it difficult to navigate to the payment screen at first but commented they would know what to do the next time they attempted to purchase a ticket.

Many users were not fans of our original NFC screen. They remarked the layout ‘looked a bit clunky’.

Users were impressed by the layout of the addCard feature commenting on the style of the user interface and how intuitive it was to use.

Changes:
We tidied up the NFC screen and gave it a new design layout.
We added icons to the buttons to make it clearer what buttons were used to navigate to the payment section.

Task 3:
Users were asked to plan their route using the maps feature in the application.

Feedback:
Users liked how easy it was to navigate to the maps screen. A few users commented that they did not find it very easy to plan their route using the marker. Others were impressed with the option to drop multiple markers and the fact they were able to plan their route and buy their ticket from within the same app.

Changes:
We added a live location feature after our focus group meeting took place. The flutter framework only recently added the google maps plugin. It does not yet have the functionality to add an in-app search bar.

Task 4:
Users were asked to find the ticket they had previously purchased in the application.

Feedback:
All the users found this convenient and easy to find.

Changes:
We added an icon to the view ticket button.

Task 5:
Users we asked to purchase a new ticket using Google Pay using test information that was provided to them.

Feedback:
Users were impressed by the Google Pay feature commenting on the style of the user interface and how easy it was to use.

Changes:
No changes were needed.

Task 6:
Users were asked to View the blog and user manual.

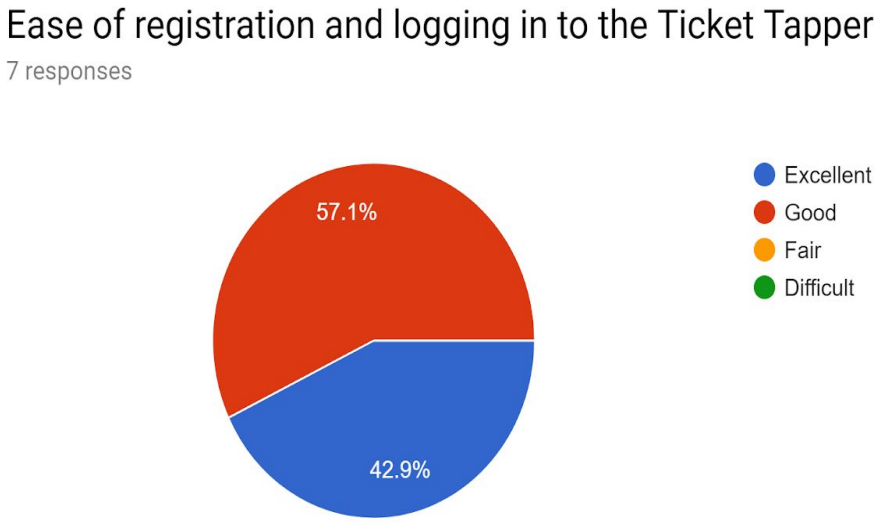
Feedback:
Users enjoyed viewing the blog and liked the images.

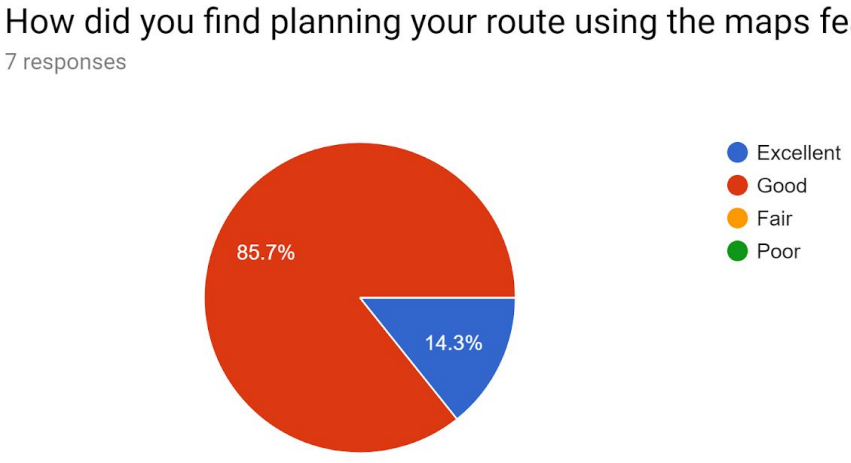
Overall the users were impressed with our application and its performance. They liked the overall theme and design apart from the layout of the NFC screen.

Changes:
Since the focus group, we changed up the design of the of the NFC screen, increased the font size and added icons to the buttons on the

main menu screen. In the future, we hope to add a search bar to our map.

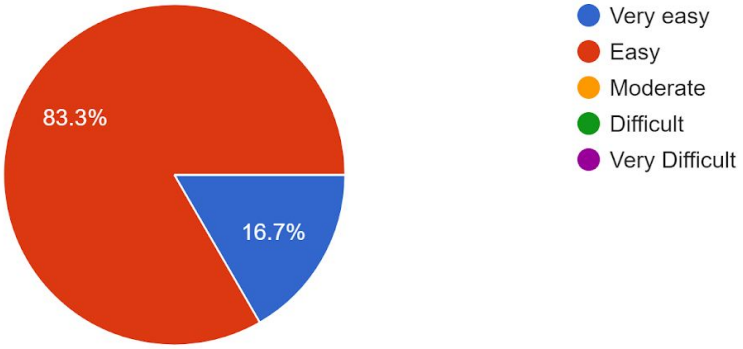
User Testing Survey Results





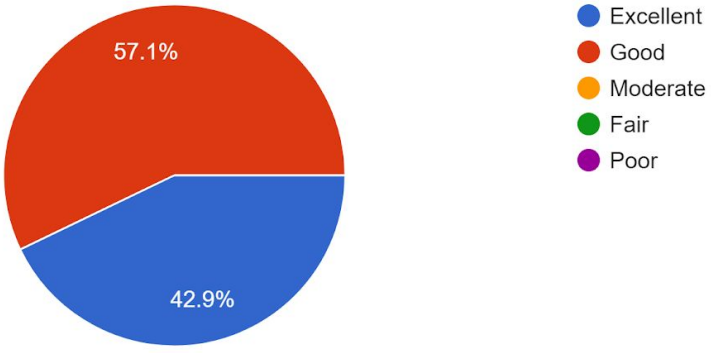
How easy was it to find a ticket that they had previously bought.

6 responses



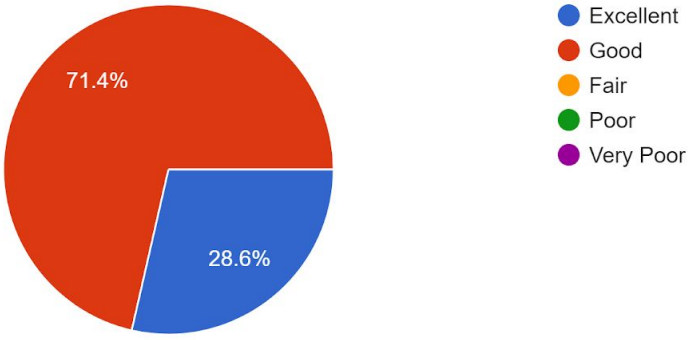
How did you like look of the app overall.

7 responses



Overall Satisfaction with application

7 responses



Any other comments are greatly appreciated.

5 responses

Very good app. Would recommend.
Nice App
The maps were very easy to navigate
NFC screen looks a bit clunky
Liked the add card feature. It was very nice