MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Midterm Report
of
Practice School

Cross-platform Mobile Application for an online on-demand service provider

SUBMITTED BY

AKSHIT SAXENA 170905430

Under the Guidance of:

Ganesh Babu C

Assistant Professor Senior Scale

Computer Science & Engineering
Manipal Institute of Technology

ABSTRACT

In today’s fast paced world where time-to-market can make or break a promising venture, businesses rely on cross-platforms apps more than ever to undertake this labyrinthine task. These businesses would not want to risk missing their presence on either popular platform: App Store or the Google Play Store.

However for many of these new ventures there are many financial restrictions imposed, as a result of which it isn’t always feasible to dedicate entirely separate development teams on two versions of the identical app just to cater to the wider market, this is where cross-platform development frameworks come in.

This project covers in detail one such framework - Flutter, and the language that drives it - Dart.

Furthermore, the actual journey of a Flutter app from its inception to production is shown along with many of the technology stacks which were imperative to scale this application for market requirements.

CHAPTER 1
INTRODUCTION

The goal set by Flutter, the cross-platform UI toolkit is to enable developers to deliver high-performance apps that feel natively compiled on different platforms, appreciating the contrasts where they arise and reusing as much code as possible.

Flutter prides itself on being open-source, having a permissive BSD license, and having a thriving ecosystem of third-party packages that supplement the core library functionality.

1.1 GENERAL INTRODUCTION

The framework is built as a layered system, it is analogous to a series of autonomous libraries that each depend on the layer beneath it. None of the layers have privileged access to the layer below it, and all the parts of this series are designed in a way to be optional and more importantly, replaceable.

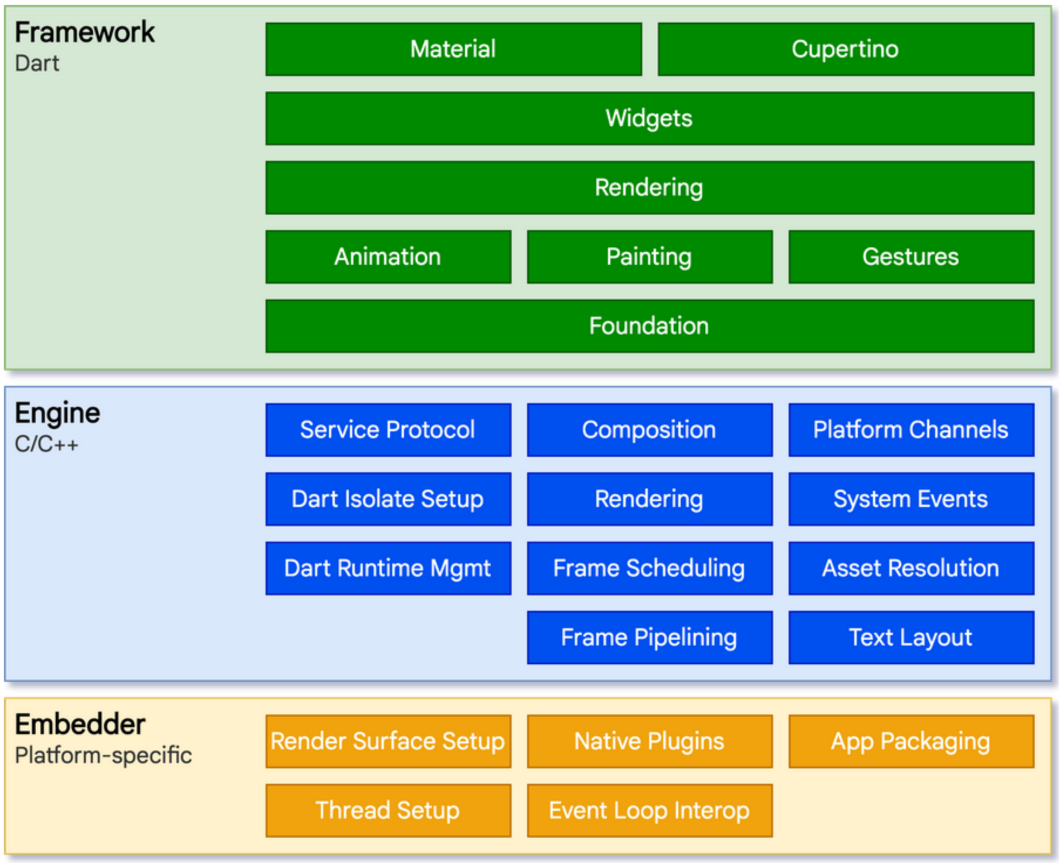

Fig 1.1

1.2 AREA OF COMPUTER SCIENCE

For the OS used to run it, Flutter applications are packaged similar to how various native applications are, using a platform specific embedder which manages entry points, asks for things like rendering surfaces, input/output etc and coordinates the message event loop.

Native application development for various platforms still holds the crown for efficiency and performance, for this very reason the flutter embedder is written in a language suitable for the respective platform.

- Java and C++ for Android
- Objective-C and Objective-C++ for iOS and macOS
- C++ for Windows and Linux

At the core of Flutter is the **Flutter engine**, which is mostly written in C++ and supports the primitives necessary to support all Flutter applications. The engine is responsible for rasterizing composited scenes whenever a new frame needs to be painted. It provides the low-level implementation of Flutter's core API, including graphics (through Skia), text layout, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain.

The engine is exposed to the Flutter framework through `dart:ui`, which wraps the underlying C++ code in Dart classes. This library exposes the lowest-level primitives, such as classes for driving input, graphics, and text rendering subsystems. [1]

1.2.1 Basic Building Block of Flutter

State, also known as Application State represents the totality of everything necessary to keep your application running. When we refer to the application state we are normally referring to the state of the program as it exists in the contents of its memory. [2]

A simple way to understand how the UI is rendered on a Flutter app would be by looking at figure 1.2

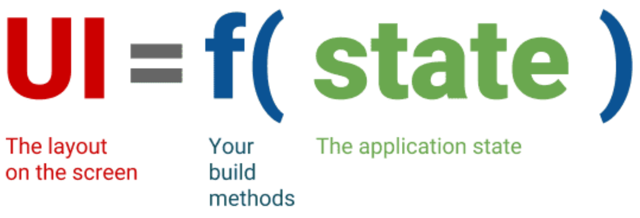

Fig 1.2

This figure condenses down what Flutter tries to achieve, the UI (layout on the screen) is essentially just a function of the state. Where the functions are the build methods specifying what to do with all the data available to the app.

For instance, a typical `Hello World!` Application in Flutter looks something like this.

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(
    Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
}
```

This is a simple application where the build function is `runApp()` which takes the `Widget Center` as an argument and displays the text in a left to right direction. The output of this code snippet is given in figure 1.3

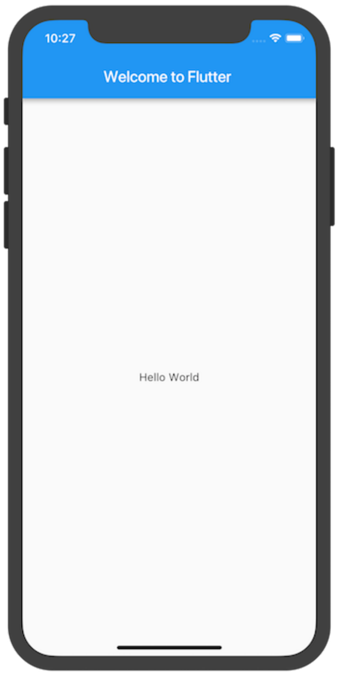

Fig 1.3

Another important thing to keep in mind while working with Flutter widgets is the concept of Stateful and Stateless Widgets.

Stateful widgets are those which have a state of their own (data of their own) which they can mutate and re-render themselves by using the method `setState()`

Stateless widgets are those who do not have a state of their own and only render once based on the data provided to them by the parent widget.

CHAPTER 2
PROBLEM DEFINITION

This project is based on a cross-platform application called ***What U Want.*** It is an online on-demand service provider which provides a wide range of services form House cleaning, Medicine delivery to Pick up & drop of everyday items.

An Android only app already exists on the Google Play Store however in order to add further features and cater to a small but significant portion of the iOS users, a decision was taken to rewrite the entire app in Flutter with all the additional features.

In addition to having a cross-platform app for the end users of these varied services, there is also a companion app which is to be used by the delivery people of ***What U Want***,
Similar to popular companion apps by companies like Zomato, Swiggy, Uber - This companion app also serves as a medium to further provide a better user experience to the end user.

The app comes with features such as location tracking for a seamless order delivery system, order acceptance/rejection for an efficient integration with partners etc.

CHAPTER 3
OBJECTIVES

The objectives of this project as specified as follows:

- Build an intuitive UI design which focuses on ease-of-use and accessibility, understand the psychology of different color patterns and choose appropriate ones.
- Connect the frontend of the Application to a backend or a typical BaaS (Backend-as-a-Service) such as Firebase.
- Building an Authentication system using a phoneNumber-OTP token to prevent misuse and ill-intentioned use.
- Maintain a stream-based open connection with Firebase Cloud Firestore to update the items on the app in realtime.
- Configuring realtime push notification using FCM (Firebase Cloud Messaging) for timely updates to the users regarding the status of their orders.
- Build a robust system for handling payments made by the user using a popular payment gateway such as Razorpay.
- Build Map support and location tracking functionality to serve relevant orders efficiently.
- Package the app for both Google Play Store and App Store by acquiring the proper security files needed, including obfuscating the codebase, creating keystore and configuring gradle signing.

CHAPTER 4
BACKGROUND

There were a number of technologies needed in order to build the entire infrastructure of the app, two of the important ones were:

1) Firebase
2) Google Cloud Platform

4.1 FIREBASE

Firebase is Google's platform for developing mobile and web applications, it is essentially a Baas (Backend-as-a-Service) which handles many of the typical challenges that novice developers face while scaling their apps for a larger market in a simple and intuitive manner.

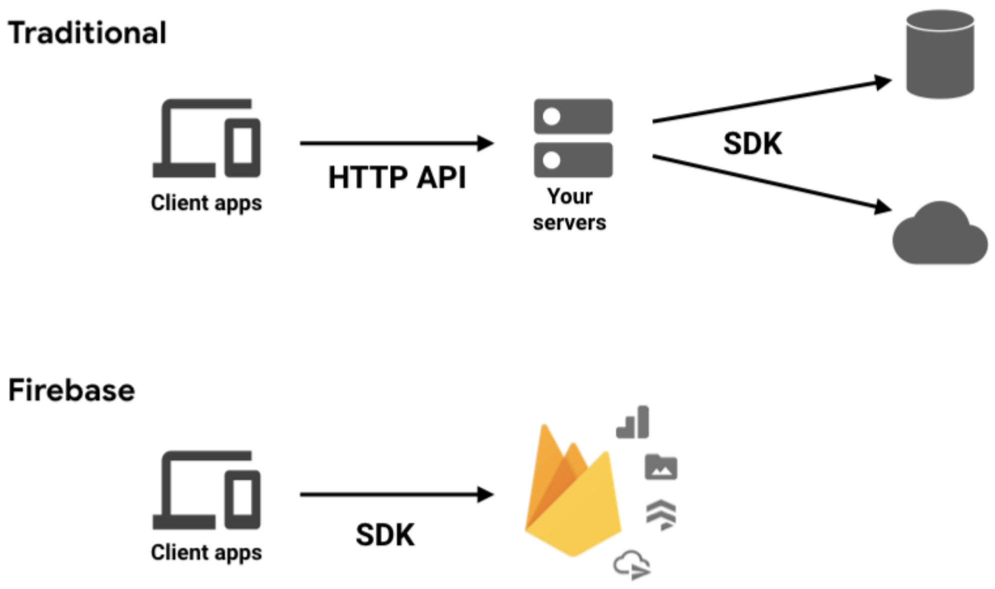

Fig 4.1

We can see how Firebase manages many tasks needing attention in a traditional backend, on its own. It also offers multiple other services which are listed as follows.

4.1.1 Authentication

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to the app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more [3]

On launch, the user is asked for his credentials which in this case is a Name and Phone Number, these are then sent to the Firebase Authentication SDK which verifies these credentials (using an OTP) and further returns a unique ID associated with each user. This ID can be used to target individual users.

4.1.2 Cloud Firestore

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. It keeps data in sync across client apps through real-time listeners and offers offline support for mobile and web. [4]

Cloud Firestore is a NoSQL database which translates into it being an excellent choice for novice developers who want to get an app to production without spending too much time designing relational schemas.

Unlike a traditional relational database, Cloud Firestore offers a document based storage solution which supports querying. Data is stored using key-value pairs which allows for quick read/writes and the value can be any data type unlike an SQL table.

```json
{
 "id": "32hzb02983",
 "firstName": "Akshit",
 "lastName": "Saxena",
 "address": {
   "street": "DM 22/4 Navri",
   "city": "Bhopal",
   "state": "Madhya Pradesh"
 },
 "hobbies": ["reading", "cooking"]
}
```

(example of a typical document based database entry)

4.2 GOOGLE CLOUD PLATFORM

GCP is Google's cloud computing solution which offers many services for all development needs. It provides developers with Kubernetes Engine, DevOps support, Cloud functions etc.

4.2.1 Cloud Functions

Cloud Functions is a serverless execution environment for building and connecting cloud services. With Cloud Functions we can write simple, single-purpose functions that are attached to events emitted from the cloud infrastructure and services. The function is triggered when an event being watched is fired. [5]

A typical Node.js function returning a message is as follows.

```
exports.helloWorld = (req, res) => {
  let message = req.query.message || req.body.message || 'Hello
World!';
  res.status(200).send(message);
};
```

4.2.2 Google Maps API

Google provides an SDK for Maps functionality in an app, apps can use map displays, respond to gestures and also provide additional information for locations and adding markers, polygons and overlays.

This was needed specifically to know the location of the user, and to calculate the correct smallest path from the vendor shop to the user. In addition the companion app makes use of the Maps SDK to calculate which delivery guy is closest to the vendor from which the user has requested to make an order.

In order to have Maps SDK functionality in an app, a GEO KEY is needed which is used for verification and eschew spam use, it then needs to be integrated in the app as such.

```
<manifest ...
  <application ...
    <meta-data android:name="com.google.android.geo.API_KEY"
               android:value="YOUR KEY HERE"/>
```

CHAPTER 5
METHODOLOGY & IMPLEMENTATION

5.1 INITIALIZE THE APP AND ENVIRONMENT

Flutter apps need to first be initialised, this can be done by going to any appropriate directory and then running `flutter create app_name` ,this command sets up the flutter environment along with all the necessary files needed for its execution.

In addition, this also creates a `pubspec.yaml` file which contains all the meta-data and miscellaneous application information.

The boilerplate code for the base flutter app simply provides us with a “Hello World” app similar to the one we saw above.

5.2 INSTANTIATE DATA MODELS

An app of this scale clearly needs to work with a lot of data that is stored on the backend, using Cloud Firestore which stores data in a JSON format.

All this data needs to be fetched on the app start, and in order to ensure proper serialization of said data many exceptions need to be handled. The fetched data needs to instantiate proper data classes as they will be used throughout the lifecycle of the app. An example of an EssentialItem data class is given below.

```
class Essential {
 final String name;
 final String id;
 final bool enabled;
 final List<String> images;
 final String bottomBanner;
 final List<Benefits> benefits;
 final List<FAQ> faq;
 final String subtitle;

 factory Essential.fromJson(Map<dynamic, dynamic> json, String id) {
   List<FAQ> temp = [];
   print(json['name']);
   try {
     json['faq'].forEach((e) {
       try {
         temp.add(FAQ.fromJson(e));
       } catch (e) {}
     });
   } catch (e) {}

   List<Benefits> tempBen = [];

   try {
     json['benefits'].forEach((e) {
       try {
         tempBen.add(Benefits.fromJson(e));
       } catch (e) {}
     });
   } catch (e) {}

   List<String> tempImages = [];

   try {
     json['image'].forEach((e) {
       try {
         e.values.forEach((ele) {
           tempImages.add(ele);
         });
       } catch (err) {
         print("Error parsing essential images inner loop $err");
       }
     });
   } catch (error) {
     print("Error parsing essential images inner loop $error");
   }

   return Essential(
       name: json['name'],
       enabled: json['enabled'] ?? true,
       id: id,
       benefits: tempBen,
       images: tempImages,
       bottomBanner: json['bottom-banner'],
       faq: temp,
       subtitle: json['subtitle'] ?? "");
 }
}
```

The code snippet given is an example of the just one of the classes which is needed. It shows the data attributes of the class and also the correct way to parse them given the input of a JSON fetched straight from the backend.

It also has multiple `try catch` blocks which are necessary to handle various random situations that can arise when fetching raw data from the server.

5.3 USER AUTHENTICATION

First a Firebase Auth object is instantiated as such

```
FirebaseAuth auth = FirebaseAuth.instance;
```

Then a method is invoked which is provided with the phoneAuthCredentials of the user, this method returns a user object which can be further manipulated to obtain all information about the user.

```
auth.signInWithCredential(phoneAuthCredential).then((value) {
User = value.user; })
```

A number of errors can also arise depending what we pass to the Firebase SDK, the error codes for some them are:

1) `ERROR_CREDENTIAL_ALREADY_IN_USE`: This arises when the phone number given is already in use with another account.

2) `ERROR_SESSION_EXPIRED`: This arises when the session of this particular user has expired.

3) `ERROR_INVALID_VERIFICATION_CODE`: This arises when the OTP entered by the user is incorrect.

5.4 PLACING ORDERS

A REST API has been put in place which is hit every time the user needs to make an order, a POST request is sent to the API which creates the order, and then the backend further processes that request. Working of a typical POST request has been shown in figure 5.1

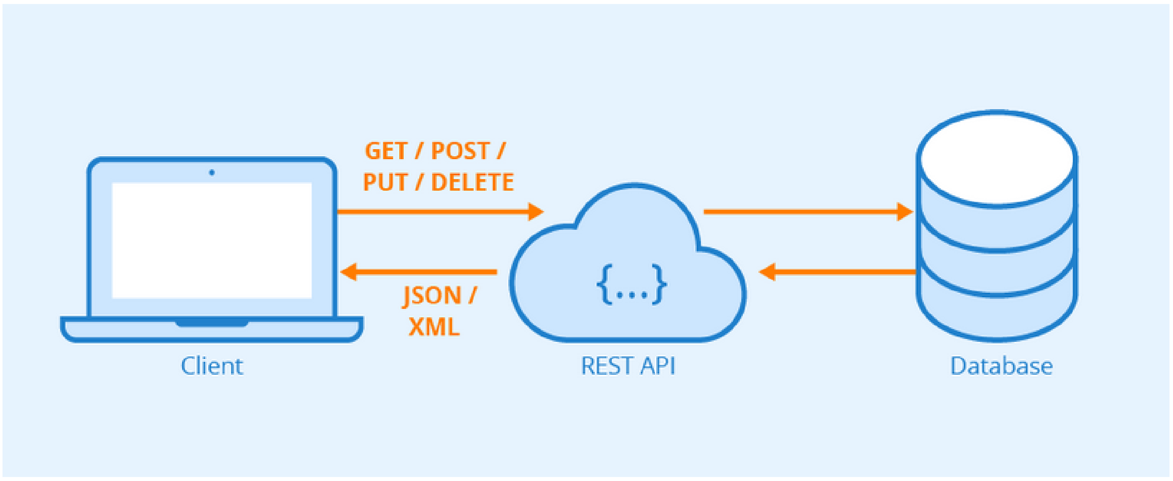

Fig 5.1

The payload given as the body of the POST request is something as follows:

```
payload = {
     "uid": user.uid,
     "type": widget.category,
     "payment": payMode,
     "amount": {
       "subTotal": widget.subtotal,
       "taxes": widget.taxes,
       "coupon": widget.coupon == "" ? null : widget.coupon,
       "discount": widget.discount == "null" ? null :
widget.discount,
       "tip": widget.tip,
       "total": widget.total
     },
     "instructions": widget.instructions == "" ? null :
widget.instructions,
   };
```

The actual request in this manner:

```
      String body = json.encode(payload);

      http.Response response = await http.post(
        createOrderUrl,
        headers: {"Content-Type": "application/json"},
        body: body,
      );

      var result = json.decode(response.body);
```

Depending on the payment method chosen by the user (Cash On Delivery or Online), the response from the request contains necessary information to further complete the order for the user.

Razorpay is used as a payment gateway in order to facilitate online payment, it is a startup which provides an easy to use SDK for integrating payments in a Flutter application. It also offers predefined methods to handle the various outcomes of the transaction (Success/Failure)

```
handleOnlinePayment(String razorpayId) {
   _razorpay = Razorpay();

   _razorpay.on(Razorpay.EVENT_PAYMENT_SUCCESS, _handlePaymentSuccess);
   _razorpay.on(Razorpay.EVENT_PAYMENT_ERROR, _handlePaymentError);
   _razorpay.on(Razorpay.EVENT_EXTERNAL_WALLET, _handleExternalWallet);

   var options = {
     'key': _razorpayKeyID,
     'name': 'What U Want',
     'order_id': razorpayId,
     'description': 'Service',
     'prefill': {
       'contact': widget.phoneNumber,
     },
     'external': {
       'wallets': ['paytm'],
     }

     _razorpay.open(options);
```

.

CHAPTER 6
REMAINING WORK

Mainly **Two** important aspects of this project are left.

6.1 CONFIGURING PICKUP & DROP OFF

An important feature of this app is the ability to have any item(s) of yours dropped off to any other place, this feature remains to be added. Similar to how other companies have brought a feature like this to market, it will essentially track the location of the origin and select an idle delivery man to assign him the job.

The drop off location and its coordinates will also be provided by the user and based on the coordinate distance between the origin and the destination, a delivery charge will be calculated and charged accordingly.

6.2 PUBLISHING THE APP

At the end of the development process the only thing remaining would be to publish the app on the Google Play Store and App Store,

In order to publish the app, it goes through a series of rigorous standard checks by the two platforms to ensure it does not contain unsafe or malicious content. These review processes can notoriously take up to 1-2 weeks.

Both the platforms also offer console dashboards for the app in which the developer can see the unexpected crashes or any relevant information for the app. In addition, they also provide easy ways to roll out further updates of the app.

CHAPTER 7
REFERENCES

[1] https://flutter.dev/docs/resources/architectural-overview

[2] https://thedaylightstudio.com/blog/2018/03/14/what-is-state-in-web-application-development

[3] https://firebase.google.com/docs/auth

[4] https://firebase.google.com/docs/firestore

[5] https://cloud.google.com/functions/docs/quickstart-nodejs

CHAPTER 8
PROJECT DETAILS

Student Details

Student Name	**Akshit Saxena**
Registration Number	170905430
Email Address	akshit.razor@gmail.com
Section/ Roll no.	A - 55
Phone No.	+91 7507161007

Project Details

Project Details	**Cross-platform Mobile Application for an on-demand service provider**
Project Duration	4 months

Internal Guide Details

Faculty Name	**Ganesh Babu C**
Contact address	Dept. of Computer Science & Engg, Manipal Institute of Technology, Manipal - 576 104 (Karnataka State), INDIA
Email Address	ganeshbabu.c@manipal.edu