An Industrial Training Report

on
Flutter internship

A Report Submitted

in Partial Fulfilment of the Requirements

for the

Diploma in

in

Information Technology

by

Pratyay Prasad Dhond

(Roll No. 1907011)

Department of Information Technology

Government Polytechnic, Nagpur

Near Mangalwari Bazar,Sadar, Nagpur 440 001 (M.S.)

[A Y 2021-22]

DECLARATION

I, **Pratyay Prasad Dhond (Roll No: 1907011)**, hereby declare that, this report entitled **"Flutter Internship Report"** submitted to Department of Information Technology, Government Polytechnic Nagpur towards partial requirement of **Diploma** in **Information Technology.**

Industrial Training work carried out by me under the supervision of **Prof. S J Pukale** Sir, and **Soumitra Mahajan** sir, Whenever an external information or statement or result is used then, that have been duly acknowledged and cited.

Place:-Government Polytechnic, Nagpur **Pratyay Prasad Dhond**

Date:-[December 23, 2021]

CERTIFICATE

This is to certify that the Industrial Training Report entitled **Flutter training** Submitted by **Pratyay Prasad Dhond (Roll No: 1907011)**, is a bonafide work carried out under the supervision of **Soumitra Mahajan sir and Prof. S J Pukale** and it is submitted towards the partial fulfilment of the requirement Government Polytechnic,Nagpur for the award of the Diploma in Information Technology.

Prof.Pukale Sachin J	**Dr.A R Mahajan**
Lecturer	Head of the Department,
Department of Information Technology	Department of Information Technology
.	**Dr. M. B. Daigavane**
Seal/Stamp of the College	Principal

Place:-Government Polytechnic, Nagpur

Date:-[December 23, 2021]

ABSTRACT

Flutter is a cross-platform UI development framework that allows developers to create cross-platform apps with a single code base.

Flutter may be used to create stunning natively built apps for iOS and Android. Android, iOS, Windows, Mac, Linux, Web, and more platforms are all supported. a single codebase is used The Flutter Framework is known for its speed and ease of use.

the Native, the Hot Reload, and the Expressive and Flexible User Interface (UI) Performance.

Flutter's rapid reload feature allows you to develop quickly and effortlessly. Experiment, design user interfaces, add functionality, and address errors more quickly. On emulators, simulations, and games, experience subsecond reload times without losing state. hardware.

Flutter comes with a set of easy-to-build, highly customizable UI widgets that may be utilised for quick development. Flutter Supports the creation of bespoke widgets from other widgets. Flutter's widgets take into account all important platform differences, such as to give features like as scrolling, navigation, icons, and fonts

Contents

List of Figures

List of Tables

Chapter 1

Introduction

You'll probably get ten different replies if you ask ten different mobile developers how they design their apps for Android or iOS devices. Because different languages are used for different operating systems, development teams for each operating system, such as Android, iOS, Windows, Mac, Linux, and web-apps, are required. This has a significant impact on the project's construction costs. As with every project, multiple meetings, development phases, testing phases, designing phases, and different codebases are required, and these softwares are difficult to manage because each change in the software must be reflected in each and every code base individually by the various teams.

To avoid this reason there was a need for cross platform development from the old-fashioned porting of apps. For solving this problem, Google developed the flutter framework to make cross platform development

easier, with the view of - 'One codebase, multiple platforms!'.

1.1 History

The most popular version of Flutter was codenamed "Sky" and operated on the Android operating system. It was announced during the 2015 Dart engineer summit with the stated goal of being able to dependably deliver at 120 frames per second. During the September 2018 Google Developer Days in Shanghai, Google announced Flutter Release Preview 2, the final major release before Flutter 1.0. Flutter 1.0, the main "stable" edition of the Framework, was released on December 4th of that year at the Flutter Live event. At the Flutter Interactive event on December 11, 2019, Flutter 1.12 was released.

The Dart programming advancement unit (SDK) in version 2.8 and Flutter in version 1.17.0 were released on May 6, 2020, with support for the Metal API, which improved execution on iOS devices (by about half), new Material gadgets, and new organisation following.

Google released Flutter 2 on March 3rd, 2021, during an online Flutter Engage event. This major release added official support for online apps, including a new Canvas Kit renderer and web explicit gadgets, as well as early-access desktop application support for Windows, macOS, and Linux, and improved Add-to-App APIs. The Flutter group supplied directions to relieve these progressions as well. This delivery incorporated sound invalid

wellness, which created several breaking changes and troubles with multiple exterior bundles, but the Flutter group gave directions to alleviate these progressions as well.

Google released the Dart SDK in version 2.14 and Flutter adaption 2.5 on September 8, 2021. The update included improvements to Android Full-Screen mode and Material You, Google's most recent Material Design version. Dart received two new upgrades, with the most recent build conditions being normalised and set as the defaults. Dart for Apple Silicon is currently stable.

Chapter 2

Introduction to Dart

Dart is a programming language designed mainly for client development, such as for the web and mobile apps. Developed by Google, it is gaining popularity quite fast and it can also be used to build server and desktop applications.

Dart is an Object-oriented, class based, garbage collected language with C like syntax.

Dart can either compile to either native code or to JavaScript

It supports interfaces, mixins, abstract classes, reified generics, and type inference.

Figure 2.1: Dart Language

2.1 History

- Unveiled at GOTO conference in Aarhus, Denmark, on October 10-12 2012.
- The dart project was founded by Lars Bak and Kasper Lund.
- Dart 1.0 was released on November 14,2013.
- Dart 1.9 released in 2015 to focus on compiling Dart to JavaScript.
- Dart 2.0 released in August 2018
- Dart 2.6 introduced new extension, dart2native

Chapter 3

Introduction to Firebase

Firebase is a platform developed by Google for creating mobile and web applications. It was originally an independent company founded in 2011. In 2014, Google acquired the platformabstract classes, reified generics, and type inference.
and it is now their flagship offering for app development.

Figure 3.1: Firebase

3.1 History

- Firebase evolved from Envolve, a prior startup founded by James Tamplin and Andrew Lee in 2011.
- Involve provided developers an API that enables the integration of online chat functionality into their websites. After releasing the chat service, Tamplin and Lee found that it was being used to pass application data that were not chat messages.
- In May 2012, a month after the beta launch, Firebase raised $1.1 million in seed funding from venture
- They founded Firebase as a separate company in 2011 and it launched to the public in April 2012.
- In October 2014, Firebase was acquired by Google
- July 2016, Google announced that it was acquiring the mobile developer platform LaunchKit, which specialized in app developer marketing, and would be folding it into the Firebase Growth Tools team.
- In October 2017, Firebase launched Cloud Firestore, a real-time document database as the successor product to the original Firebase Realtime Database.

Chapter 4

Architecture

The Flutter framework is organized into a series of layers, with each layer building upon the previous layer

4.1 Dart Framework

Flutter apps are created in the Dart programming language and take advantage of many of the language's more advanced features. Flutter operates in the Dart virtual machine, which contains an in-the-moment execution engine, on Android, as well as Windows, macOS, and Linux via the semi-official Flutter Work area Embedding project. Flutter apps for iOS

Figure 4.1: Architecture of Flutter Framework

employ ahead-of-time (AOT) compilation due to App Store constraints on unique code execution. The Dart stage's support for "hot reload," which allows changes to source documents to be injected into a running application, is a standout feature. Flutter extends this by providing support for stateful hot reload, which allows changes to source code to be reflected in the current application without needing a restart or any loss of state.

4.2 Flutter's Engine

Flutter's Engine, which is based on C++, supports low-level rendering with Google's Skia Graphics package. It also interacts with platform-specific

SDKs, such as those provided by Android and iOS. The Flutter Engine is a small runtime that makes it easier to create Flutter applications. Flutter's key libraries, such as activity and designs, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compilation toolchain, are all executed by it. Most developers will be familiar with Flutter through the Flutter Framework, which provides a cutting-edge, responsive framework as well as a diverse set of staging, design, and setup tools.

4.3 Platform

Flutter provides a Shell with the Dart Virtual Machine at the platform level. The Shell is platform agnostic, allowing access to local APIs and easing the creation of stage-ready content. There's also an embedder API, which lets you use Flutter as a library rather than a framework for executing apps. The Shells also aid in establishing contact with the appropriate IMEs and the framework's application lifecycle events.

Chapter 5

Widgets

Everything in Flutter is a Widget. This incorporates UI components, like ListView, TextBox, and Image, just as different parts of the system, including format, activity, motion acknowledgment, and subjects, to give some examples.

Widgets are fundamental for an application's view and interface. They should have a characteristic look and feel paying little heed to screen size. They likewise should be quick, extensible, and adjustable. Flutter takes the Everything is a Widget approach. It has a rich arrangement of Widgets and broad abilities for making complex custom Widgets. In Flutter, Widgets aren't just utilized for views. They're additionally utilized for whole screens and in any event, for the actual application.

As Flutter's documentation puts it, every Widget is an unchanging presentation of part of the UI. Different structures separate perspectives, view regulators, designs, and different properties. Flutter, then again, has a

predictable, consistent, brought together item model: the Widget. A Widget can characterize an underlying component (like a button or menu); a complex component (like a textual style or on the other hand color scheme); a part of the format (like padding, etc. Widgets structure a chain of command dependent on their piece. Every Widget settles within and acquires properties from its parent. There's no different application object. All things considered, the root Widget serves this job.
Flutter has a full arrangement of Widgets in Google's Material Design and in Apple's style with the Cupertino pack. Widget delivering happens straightforwardly in the Skia engine without utilizing Original Equipment Manufacturer Widgets. So we get a smoother UI experience contrasted and other cross platform structures.

Widgets in Flutter are divided into two sub categories:
1) StateLess Widgets
2) StateFul Widgets

5.1 StateLess Widgets

A stateless Widget is a Widget that portrays part of the UI by building a network of different Widgets that portray the UI all the more solidly. The building process proceeds recursively until the description of the UI is completely concrete.

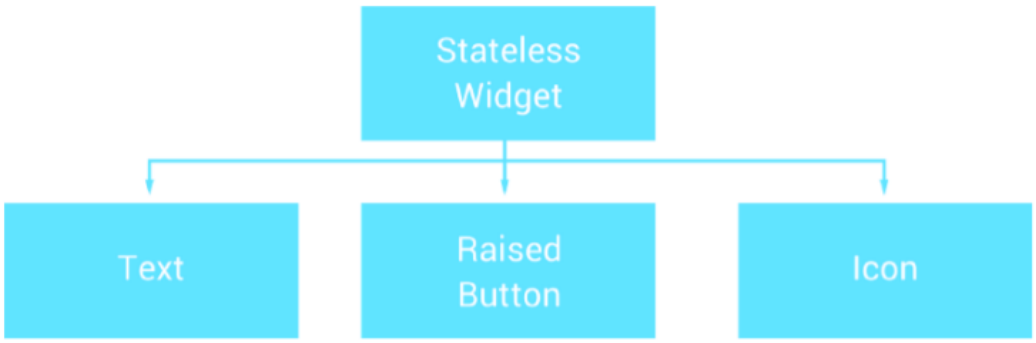

Figure 3.1.1 Stateless Widgets

Figure 5.1: Stateless Widget

Stateless Widget are valuable when the piece of the UI you are portraying doesn't rely upon something besides the arrangement data in the actual article and the BuildContext in which the Widget is expanded. For compositions that can change dynamically, for example due to having an inside clock-driven state, or relying upon some system state, think about utilizing StatefulWidget.

5.2 StateFul Widget

A stateful widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface

more concretely. The building process continues recursively until the description of the user interface is fully concrete (e.g., consists entirely of RenderObjectWidgets, which describe concrete RenderObjects).

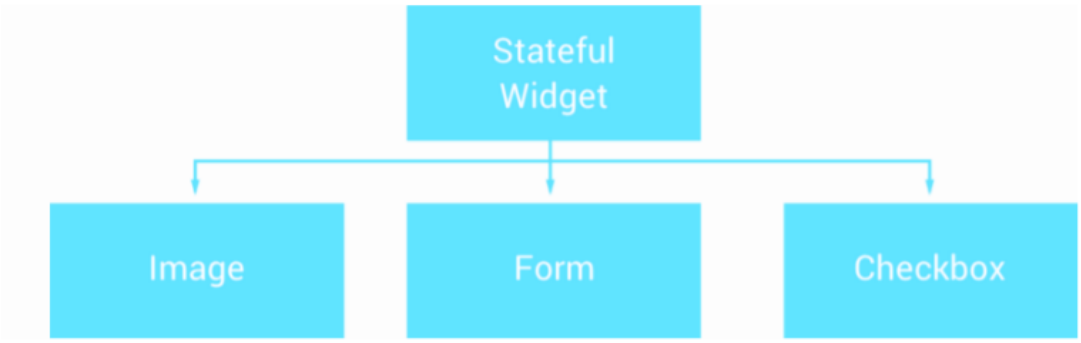

Figure 5.2: Stateful Widget

Stateful widgets are useful when the part of the user interface you are describing can change dynamically, e.g. due to having an internal clock-driven state, or depending on some system state. For compositions that depend only on the configuration information in the object itself and the BuildContext in which the widget is inflated, consider using StatelessWidget.

Chapter 6

Widget Tree

6.1 Summary

Developers utilise the widget tree to design user interfaces; they place widgets within each other to create basic and complicated layouts. Because almost everything in the Flutter framework is a widget, the code might get difficult to read when they start layering them. Developers can divide widgets into their own widget class, resulting in a shorter widget tree, which improves code readability and administration. Developers may take advantage of Flutter's subtree rebuilding, which enhances efficiency, by refactoring using a widget class. The widget is initialised to a final variable when you refactor with a constant. The chapter delves into refactoring with a method rather than a constant in depth. By invoking the method name, refactoring with a method returns the widget.

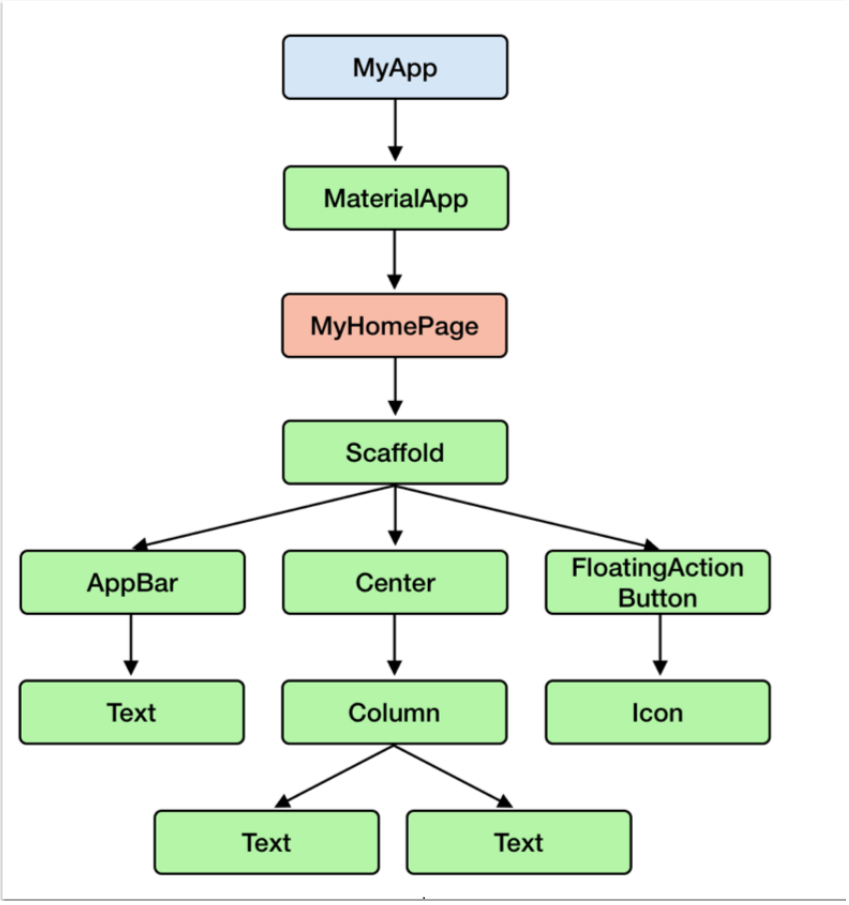

Figure 6.1: Stateless Widget

Chapter 7

Flutter Provider Widget

7.1 What is Flutter Provider?

The provider package is an easy to use package which is basically a wrapper around the InheritedWidgets that makes it easier to use and manage. It provides a state management technique that is used for managing a piece of data around the app.

7.2 basic classes that are provided with flutter provider

The basic classes available in the provider package are –
ChangeNotifierProvider¡T extends ChangeNotifier¿ It listens to a ChangeNotifier extended by the model class, exposes it to its children and descendants, and rebuilds depends whenever notifyListeners is called.
Consumer - It obtains the provider from its ancestors and passes the value obtained to the builder.
FutureProvider - This class listens for a Future and then passes its values to its children and descendants.
InheritedProvider - The InheritedProvider provides a general implementation of the InheritedWidget.
MultiProvider A provider that is used to provide more than one class at the same time.
Provider - It is the basic provider.
ProxyProvider - This provider depends on other providers for value. The value can be used by create or update.

Chapter 8

Project

At the end of the internship, we were assigned a task to develop an application using Flutter. I made a to-do List application, with the following features:

- Login and Signup with Firestore authentication as backend
- Two user friendly themes (Light theme and dark theme)
- Minimalistic Design
- Data synchronisation on Firestore firebase cloud

Figure 8.1: Firebase Authentication Logo

8.1 Login and Signup with Firestore Authentication

In the present era, user authentication is one of the most important requirements for Android apps. It is essential to authenticate users, and it is much harder if we have to write all this code on our own. This is done very easily with the help of Firebase.

- Being able to authenticate our users securely, it offers a customized experience to them based on their interests and preferences.
- We can ensure that they have no problems accessing their private

data while using our app from multiple devices.

- Firebase Authentication provides all the server-side stuff for authenticating the user. Firebase Authentication becomes easy with SDK. It makes API easy to use.
- Firebase Authentication also provides some user interface libraries which enable screens for us when we are logging it.
- Firebase authentication supports authentication using a password, phone numbers, popular identity provider like Google, Facebook, and Twitter, etc.
- We can sign in users to our app by using the FirebaseUI.
- It handles the UI flows for signing in user with an email address and password, phone numbers, and popular providers, including Google sign-In and Facebook Login
- It can also handle cases like account recovery.
- It is not required to design a UI since it is already provided for us. It means we don't have to write the activities.
- We can also sign-in users using the Firebase Authentication SDK to integrate one or several sign-in methods into our app manually.

8.2 Two user friendly themes

In the To-Do List app i created, i had two themes i.e. dark mode and light mode for the users to have good user experience.

8.2.1 Light Mode

- White background
- Grey Buttons (HexCode : #EAEAEA)
- Black icons
- Green Update and Complete task button

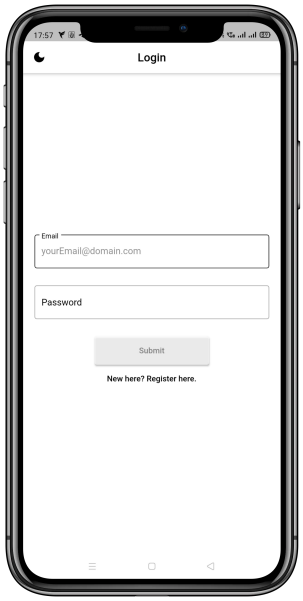

Figure 8.2: Login Page

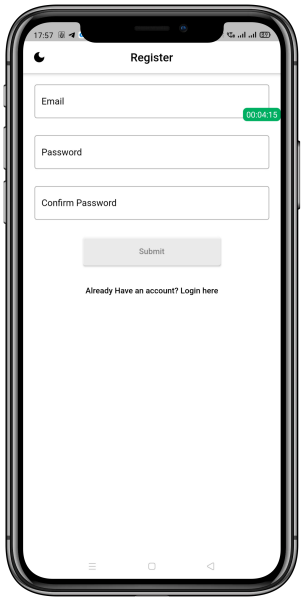

Figure 8.3: Registration page

Figure 8.4: Add Task Page

Figure 8.5: Home page

Figure 8.6: Update Task Page

8.2.2 Dark Mode

- An eye friendly shade of black as background of app (Hexcode: #1A1A1A)
- Black buttons with a silver-white outline
- White icons
- Green add task icon
- Yellow color edit task icon

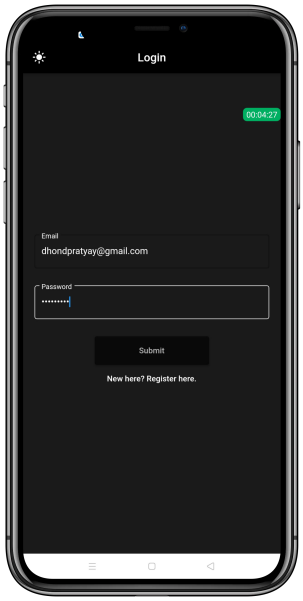

Figure 8.7: Login Page

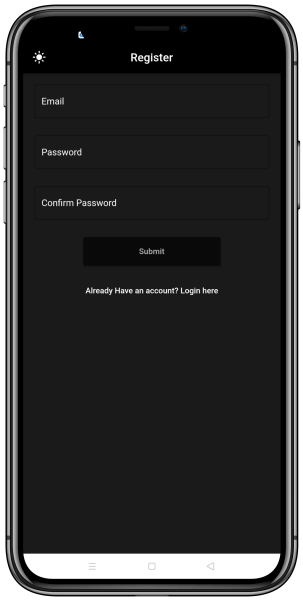

Figure 8.8: Registration page

Figure 8.9: Home Page

Figure 8.10: Add Task Page

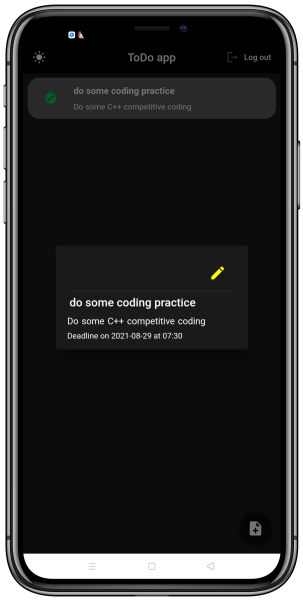

Figure 8.11: Edit Task Page

8.3 Minimalistic Design

I developed the application with the goal of keeping it minimalistic, making it simpler and usable.

- Minimal function
- productivity oriented
- Easy to understand

8.4 Data Synchronsiation with firestore database

All the data of the application such as tasks, task due date, task creation date, etc. will be uploaded to the Firestore Firebase database so that it could be accessed by the user through any android device meeting the minimum requirements.

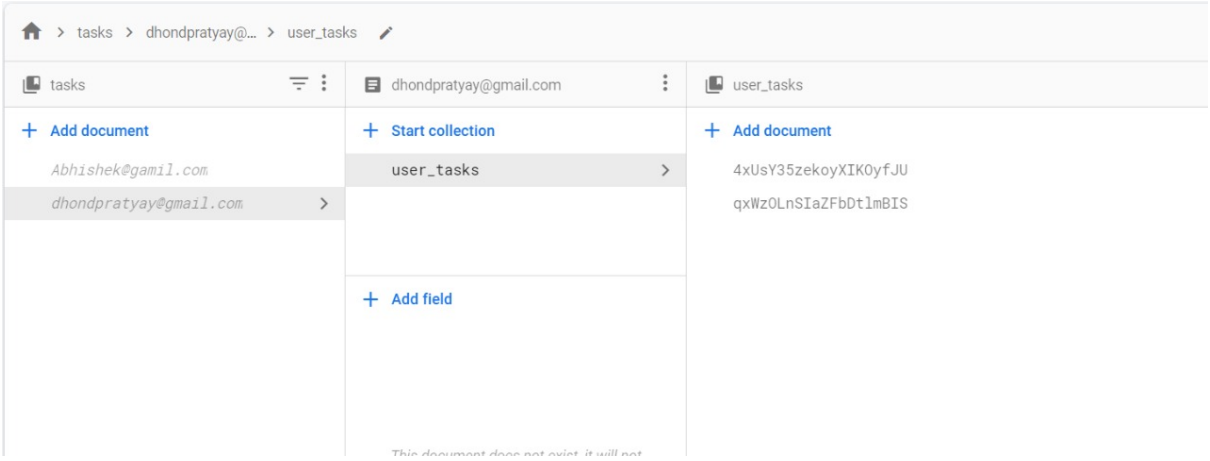

Figure 8.12: Firestore Database View

8.5 Result

Thus, I made the To-Do List application using flutter framework as a project for the internship.

Chapter 9

Conclusion

In the Industrial training in flutter, I learnt about the Flutter framework and the dart programming language, Firestore Firebase database and also about the CRUD operations. I also created an ToDo List app as a project for this internship with two themes and cloud storage so that the tasks can be accessible from any device.

The training was an enriching experience for me as I got to know about a lot of industrial practices like 'never to store a password in the database, it should be encrypted' and also 'Not using users gmail id's as an database name but their uid in database' to maintain their anonimity.

The main things i have learned through this trainging is time management, working with a team and also Developing cross platform applica-

tions using flutter. Working with Flutter has increased my interest in mobile development even more as I was always into android development. Flutter was the spark i needed to enlighten myself and get my interest in development.

Chapter 10

References

1. https://flutter.dev/
   "This is the official website of Flutter."
2. https://pub.dev/
   "This is a website where developers can access various open source dependencies and implement them in their project."
3. https://dart.dev/
   "This is the official website for Dart programming language."
4. Practical Flutter - Frank Zammetti
   "This is a great book for anyone who wants to learn to code using flutter framework and build cross platform apps."
5. https://www.youtube.com/playlist?list=PLOU2XLYxmsIJ7dsVN4iRuA7BT8XHzGtCr
   "This is a link to a flutter playlist on youtube made by people working on flutter at google themselves."

6. https://www.reddit.com/r/FlutterDev/

   "Flutter forum on reddit contains various doubts, release notes , bugs, bug fixes, etc."

7. https://discord.com/invite/N7Yshp4

   "Flutter community on discord is a great place to chat with other flutter developers, post your doubts, solve other people's doubts and to grow better as a developer."