**PROJECT REPORT**

**ON**

**“Messager-Chatting Application”**

Submitted in partial fulfilment of the requirements for the award of degree of

**MASTER’S OF COMPUTER APPLICATION**

**To**

**DEPARTMENT OF COMPUTER APPLICATIONS**

**UTTRANCHAL SCHOOL OF COMPUTING SCIENCES**

**UTTRANCHAL UNIVERSITY, DEHRADUN**



**(Session: 2022-2023)**

**Under the guidance of: Submitted by:**

**Dr. Monisha Awasthi Vaibhav Vats**

**Assistant Professor (MCA-3-B)**

**UU2220000094**

**ACKNOWLEDGEMENT**

There are many people who helped directly or indirectly on the successful completion of my mini project. I would like to thank Dean of USCS “**Dr. Sonal Sharma**” for providing us with all the necessary resources required to complete the project. We profusely are thankful to the program coordinator of the department “**Mr. Sameer Sharma**” for their valuable guidance, inspiring guide and committed caretaker for her unflinching devotion. The encouragement and support by her, especially in carrying out this project motivated me to complete this project. I would like to express deep sense of gratitude to our staff members of Dept. of Computer Applications for their co-operation, which has given in the congruency to build up this project. I would like to thank all our friends for their help and constructive criticism during my project period.

Finally, I am very much indebted to my parents for moral support and encouragement to achieve goals. I have no words to express my gratitude and still I am very thankful to my parents who have shown me this world and for

every support they gave me.

**VAIBHAV VATS**

**MCA SEMESTER-(III)**

**DECLARATION**

I hereby declare that the project report entitled “**Messager”** submitted by **Vaibhav Vats** to Uttaranchal Institute of Computing Sciences**.** I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this university or any other university or institute.

**VAIBHAV VATS**

**MCA SEMESTER-(III)**

**CERTIFICATE**

This is to certify that the project entitled “**Messager”** submitted by **Vaibhav Vats** and has been submitted in the partial fulfilment of the requirements for the award of the degree of MCA from Uttaranchal University, Dehradun. The results embodied in this project have not been submitted to any other University or Institution for the record of any degree.

**Under the guidance of:**

Dr. Monisha Awasthi

(Assistant Professor)

**CONTENTS**

[1.INTRODUCTION 6](#_Toc133266572)

[2.OBJECTIVE 7](#_Toc133266573)

[2.1PURPOSE, SCOPE AND APPILCABILITY 8](#_Toc133266574)

[3.SYSTEM ANALYSIS 9](#_Toc133266575)

[3.1 PROBLEM DEFINITION 9](#_Toc133266576)

[3.2 FEASIBILITY STUDY 9](#_Toc133266577)

[3.3 HARDWARE AND SOFTWARE REQUIREMENTS 10](#_Toc133266578)

[4.SYSTEM DESIGN 11](#_Toc133266579)

[4.1 PROJECT PLANNING 11](#_Toc133266580)

[4.2 METHODOLOGY 11](#_Toc133266580)

[4.3 REQUIREMENT SPECIFICATION(SRS) 11](#_Toc133266582)

[4.4 PROJECT SCHEDULING 14](#_Toc133266583)

[4.4.1 GANTT CHART 14](#_Toc133266584)

[5.DATA MODELS 15](#_Toc133266586)

[5.1](#_Toc133266587) [ACTIVITY DIAGRAM 15](#_Toc133266588)

[5.3 USE CASE DIAGRAM 16](#_Toc133266589)

[5.4 SEQUENCE DIAGRAM 19](#_Toc133266590)

[5.5](#_Toc133266591) [COMPONENT DIAGRAM 17](#_Toc133266592)

[6.TESTING 17](#_Toc133266593)

[6.1 TEST CASE DESIGN](#_Toc133266594) 18

[7. COST ESTIMATION 19](#_Toc133266595)

[7.1 Project Estimation](#_Toc133266596) 20

[7.2 Project Estimation Methods](#_Toc133266597) 21

[7.3 Future Scope Of This Project 23](#_Toc133266598)2

[8. APPENDICES](#_Toc133266599) 24

[8.1 Source Code:](#_Toc133266600) 24

[9. CONCLUSION 3](#_Toc133266601)5

[10.REFERENCE & BIBLIOGRAPHY](#_Toc133266602) 36

# INTRODUCTION

In the ever-evolving landscape of digital communication, the Messager app emerges as a testament to innovation and user-centric design. As we navigate through the intricacies of modern connectivity, the need for a messaging platform that seamlessly combines functionality with an aesthetically pleasing interface becomes increasingly pronounced. Enter Messager, a cross-platform marvel crafted with Flutter—a powerful UI toolkit known for its ability to transcend the limitations of traditional development.

Bridging Platforms with Flutter's Versatility

Flutter, with its promise of natively compiled applications for mobile, web, and desktop from a single codebase, stood out as the ideal framework for our endeavour. The decision to utilize Flutter is rooted in the desire to offer a consistent and reliable messaging experience, regardless of the device in use. This versatility not only streamlines development but also ensures that Messager is a truly cross-platform solution.

Real-Time Conversations in the Digital Realm

At the heart of Messager lies the ambition to facilitate real-time communication. Leveraging the robust Firebase services, the app synchronizes messages instantaneously, transforming conversations into dynamic exchanges that transcend the limitations of traditional messaging. The result is an environment where users can engage in lively and responsive dialogues, blurring the lines between physical distances.

iOS Aesthetics with the Cupertino Package

While the technical foundation of Messager is rooted in Flutter, the design philosophy takes a distinct turn towards the elegance of iOS. Enter the Cupertino package—a collection of Flutter widgets designed to emulate the visual aesthetics of Apple's operating system. By incorporating Cupertino design elements, Messager seamlessly integrates into the iOS ecosystem, providing users with a sense of familiarity and a visually cohesive experience.

# 2.OBJECTIVE

# 2.1.1 Create a User-Friendly Messaging App

# The primary objective of Messager is to deliver a messaging application that prioritizes user-friendliness at its core. This encompasses a seamless onboarding process, intuitive navigation, and an interface that allows users to effortlessly engage in conversations.

# 2.1.2 Intuitive User Interface

# The user interface is designed with a focus on simplicity and clarity, utilizing familiar patterns to make interactions instinctive. Through thoughtful layout and responsive design, Messager aims to create an environment where users feel comfortable expressing themselves without unnecessary complexity.

# 2.1.3 Effortless Onboarding

# The onboarding process is crafted to be smooth and engaging, guiding users through the initial setup seamlessly. Clear instructions, minimalistic design, and an emphasis on user preferences contribute to an onboarding experience that sets the tone for a user-friendly journey within the app.

# 2.1.4 Implement Real-Time Messaging Features

# Messager distinguishes itself by embracing the immediacy of communication through the implementation of robust real-time messaging features.

# 2.1.5 Dynamic Conversations

# Real-time messaging extends beyond the conventional, creating an environment where conversations become dynamic exchanges. Users experience instant updates, ensuring that they are always in sync with the latest messages and developments within their chats.

## 2.1PURPOSE, SCOPE AND APPILCABILITY

**Purpose**

Purpose of Messager is to provide all the features a messaging app can do along with maintaining the privacy of its users.

**Scope**

Messager, a cross-platform messaging app developed with Flutter, aims to deliver a user-friendly and versatile communication experience. Its functional scope includes real-time one-on-one and group text communication, multimedia sharing capabilities, and push notifications for timely message alerts. The app ensures cross-platform compatibility, leveraging Firebase for user authentication and real-time messaging, and incorporates the Cupertino package to achieve a visually cohesive and minimalistic design in alignment with iOS principles. Design considerations focus on iOS aesthetics and responsiveness across various iOS devices. While acknowledging limitations such as potential variances in platform-specific features and network dependency, Messager envisions future enhancements including end-to-end encryption, additional multimedia features, and user customization options, setting the stage for a dynamic and evolving messaging platform.

**Applicability**

Messager's applicability is broad and significant, catering to users seeking a seamless and user-friendly messaging experience. Its cross-platform nature, developed with Flutter, ensures accessibility on both iOS and Android devices, addressing a diverse user base.

# 3.SYSTEM ANALYSIS

System Analysis is about complete understanding of existing systems and finding where the existing system fails. The solution is determined to resolve issues in the proposed system. It defines the system. The system is divided into smaller parts. Their functions and inter relation of these modules are studied in system analysis. The complete analysis is followed below.

## 3.1 PROBLEM DEFINITION

Messager addresses the challenges of existing messaging apps by focusing on two key problems: a lack of user-friendly interfaces and limited real-time communication features. Many platforms struggle with unintuitive designs, leading to user frustration. Additionally, the absence of real-time messaging inhibits spontaneous conversations. Messager aims to solve these issues by providing a user-friendly app with a seamless iOS-inspired interface, utilizing the Cupertino package. The goal is to redefine messaging by offering a cohesive and dynamic platform that enhances the overall user experience.

## 3.2 FEASIBILITY STUDY

Feasibility study can help you determine whether or not you should proceed with your project. It is essential to evaluate cost and benefit. It is essential to evaluate cost and benefit of the proposed system. Three types of feasibility study are taken into consideration.

1. **Technical feasibility:** Messager demonstrates strong technical feasibility due to the use of Flutter, which allows for cross-platform development, ensuring accessibility on both iOS and Android devices. The integration of Firebase for real-time messaging and authentication enhances the technical robustness of the application. The choice of technologies aligns with industry standards, making it technically sound.
2. **Operational feasibility:** Operationally, Messager is feasible as it aligns with user expectations by providing a seamless user interface and real-time messaging features. The app's operational aspects are designed to enhance user experience, fostering dynamic and engaging conversations. The user-friendly design and features contribute to the operational feasibility of the messaging platform.
3. **Economic feasibility:** Economically, Messager is feasible due to the cost-effectiveness of using Flutter, an open-source framework that reduces development costs. The potential for future enhancements, such as premium features, provides avenues for revenue generation, ensuring economic viability. The economic feasibility of Messager is strengthened by the consideration of sustainable revenue models.

## 3.3 HARDWARE AND SOFTWARE REQUIREMENTS

The software is designed to be light-weighted so that it doesn’t be a burden on the machine running it. This system is being build keeping in mind the generally available hardware and software compatibility. Here are the minimum hardware and software requirement for virtual assistant.

**Hardware:**

Any Device which can Run Android OS or IOS.

Internet Connectivity. (Network Card)

**Software:**

Android 6 (Min.)

IOS 4(Min.)

Development Environment:

1.Flutter SDK:

Installation of Flutter SDK for cross-platform app development.

Integrated Development Environment (IDE):

2. Version Control:

Implementation of version control systems (e.g., Git) for collaborative development.

3. Dependency Management:

Use of package managers like pub (for Dart/Flutter) to manage project dependencies.

Testing Frameworks:

Integration of testing frameworks for unit and widget testing.

# 4.SYSTEM DESIGN

## 4.1 PROJECT PLANNING

Project planning for Messager involves a systematic approach to organize tasks, allocate resources, and establish timelines to ensure the successful development and deployment of the messaging application. This phase encompasses defining project scope, setting milestones, and allocating responsibilities to create a well-structured development roadmap.

## 4.2 Methodology

The Agile methodology is adopted, allowing flexibility and continuous feedback throughout the development process.

## 4.3 SYSTEM REQUIREMENT SPECIFICATION(SRS)

**1**. Introduction

Messager is a messaging application designed to offer a seamless and intuitive user experience on both iOS and Android platforms. This document outlines the functional and non-functional requirements to guide the development of the application.

2. System Overview

2.1 Purpose

Messager aims to provide real-time communication with a user-friendly design inspired by iOS, utilizing the Cupertino package. The application caters to users seeking dynamic and visually cohesive messaging experiences.

2.2 Scope

The application encompasses messaging features, multimedia sharing, and a responsive interface for diverse iOS and Android devices.

3. Functional Requirements

3.1 User Authentication

1. User Registration:

- Users can create accounts securely with valid email addresses.

2. Login:

- Authenticated users can securely log in.

3.2 Messaging Features

1. Real-Time Messaging:

- Users engage in instant one-on-one and group text conversations.

2. Multimedia Sharing:

- Support for sharing images and videos within conversations.

3.3 User Interface

1. Cupertino Design:

- Integration of Cupertino package for iOS-inspired design.

- Consistent navigation bars, buttons, and UI components.

2. Responsiveness:

- Interface adapts seamlessly to various iOS and Android devices.

4. Non-Functional Requirements

4.1 Performance

1. Real-Time Responsiveness:

- Messages delivered and received instantly.

2. Scalability:

- System handles a scalable number of concurrent users.

4.2 Security

1. Data Encryption:

- User data, including messages, encrypted for security.

2. Authentication Measures:

- Robust mechanisms prevent unauthorized access.

4.3 Usability

1. User-Friendly Interface:

- Intuitive design for ease of use.

- Minimalistic and clear layout.

2. Accessibility:

- Application accessible to users with varying technical proficiencies.

5. Constraints

1. Platform Dependency:

- Application dependent on iOS and Android platforms.

2. Network Dependency:

- Real-time features reliant on a stable internet connection.

6. Future Enhancements

1. End-to-End Encryption:

- Implementation of enhanced security measures.

2. Additional Multimedia Features:

- Integration of voice messaging and document sharing.

3. User Customization Options:

- Allow users to personalize themes and chat settings.

7. Conclusion

This System Requirements Specification serves as a guide for the development of Messager, ensuring a clear understanding of its functionalities, constraints, and potential enhancements.

## 4.4 PROJECT SCHEDULING

### 4.4.1 GANTT CHART

**TABLE 1: GANTT CHART**

****

# 5.DATA MODELS

## 5.1 ACTIVITY DIAGRAM

Start

|

v

Project Kick-off

|

v

Define Project Scope

|

v

Set Milestones

|

v

Resource Allocation

|

v

Timeline Creation

|

v

Risk Assessment

|

v

Development Phase

|

v

Testing and QA

|

v

Deployment

|

v

Future Enhancements

|

v

End-to-End Encryption

|

v

Multimedia Features

|

v

User Customization

|

v

Final Testing

|

v

Project Completion

|

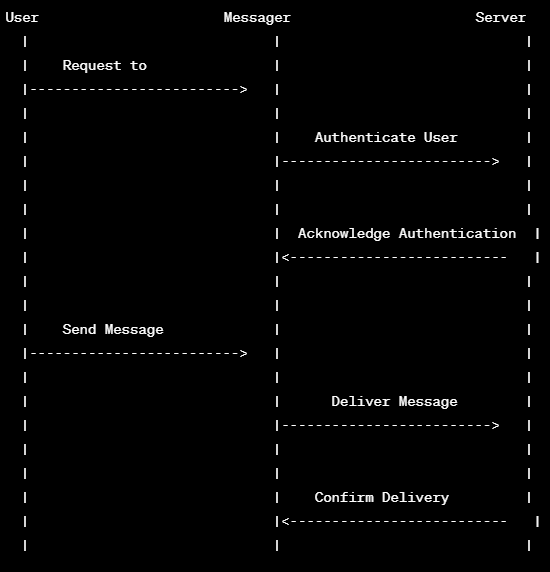
v

End

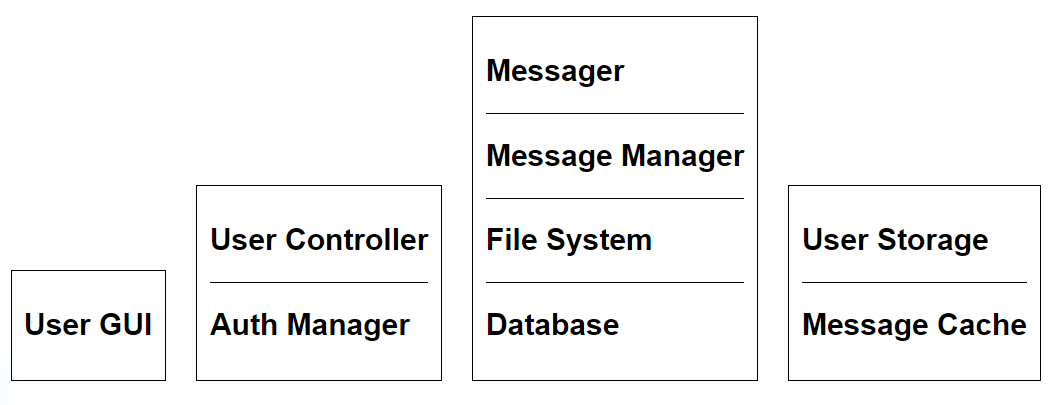
## 5.2 USE CASE DIAGRAM



## 5.3 SEQUENCE DIAGRAM



## 5.4 COMPONENT DIAGRAM



# 6. TESTING: TESTING TECHNIQUES AND TESTING STRATEGIES

**Unit Testing:**   
Unit Testing is the type of [software testing](https://www.geeksforgeeks.org/software-testing-basics/) level in which each individual component of the software is tested. Unit Testing is generally performed by the developer. Unit Testing can’t be used for those systems which have a lot of interdependence between different modules. It does not allow for parallel testing.

**System Testing:**   
System testing is done to check whether the software or product meets the specified requirements or not. It is done by both testers and developers. It contains System testing and Integration testing. It is done through more positive and negative test cases.

Testing Messager to iterate and improve the NLU model is a critical step in the virtual assistant life cycle. Before you launch your virtual assistant to production, before folks are actually using it, you want to make sure that it works. It's pretty simple, but testing is so important.

There are a variety of ways in which you can go about your testing because ultimately when you do get to production you want to deliver an experience that people love and really enjoy using.

## 6.1 TEST CASE DESIGN

**Test Case 1**

Test Title: Response Time

Test ID: T1

Test Priority: High

Test Objective: To make sure that the system respond back time is efficient.

Description: Time is very critical in a voice-based system. As we are not typing inputs, we are speaking them. The system must also reply in a moment. User must get instant response of the query made.

**Test Case 2**

Test Title: Accuracy

Test ID: T2

Test Priority: High

Test Objective: To assure that answers retrieved by system are accurate as per gathered data.

Description: A virtual assistant system is mainly used to get precise answers to any question asked. Getting answer in a moment is of no use if the answer is not correct. Accuracy is of utmost importance in a virtual assistant system.

**Test Case 3**

Test Title: Approximation

Test ID: t3

Test priority: Moderate

Test Objective: To check approximate answers about calculations.

Description: There are times when mathematical calculation requires approximate value. For example, if someone asks for value of PI the system must respond with approximate value and not the accurate value. Getting exact value in such cases is undesirable. Note: There might include a few more test cases and these test cases are also subject to change with the final software development.

# 7. COST ESTIMATION

## 7.1 Project Estimation

There are several common IT project cost estimation models, also called methods. There is no one-size-fits-all solution; rather, select a method by identifying your project’s needs using IT heuristics, such as project size, complexity, necessary resources, and preparation time.

Some models base the estimate on the smallest building block, such as a line of code, while others compare the average size of previous projects. Use the following heuristics as you consider which model best serves your project:

**Complexity:** Estimate the project’s complexity. Measure both subjective (based on your past experience) and objective (such as limited resources) factors.

**Method Reliability:** While there is not necessarily a preferred method, some methods are more reliable than others in their estimated accuracy.

**Necessary Resources:** Some projects have a wealth of historical data that is easily accessible through public databases. Others may have inconsistent or incomplete data, which will impact your ability to conduct thorough research. You may also run into personnel ability issues, such as working with a new or inexperienced team.

**Preparation Time:** Some methods are more time-consuming to prepare. Your project schedule, size, and complexity influence operational time constraints within your allotted schedule.

**Project Size:** Measure the project size using one of several common methods, such as function points (a unit of measurement to approximate the functional business value of a product), T-shirt sizing (an estimation tool defining the relative size of work), or lines of code.

## 7.2 Project Estimation Methods

There are two categories of IT project estimation methods: algorithmic and non-algorithmic. Algorithmic methods involve an equation to quantify the effort. Non-algorithmic methods use data and analysis. Use at least two methods in your cost estimation to increase accuracy.

**Algorithmic Methods**

Algorithmic estimation methods are formalized estimation models. These methods use a mechanical process, most often a formula created from patterns in historic data, and are the most standardized.

**Constructive Cost Model (COCOMO):** This method estimates effort, cost, and schedule using a three-tiered process: basic, intermediate, and detailed. It uses complex algorithmic formulas derived from historic project data to construct. This parametric method is one of the most accurate techniques for estimating project costs and often used for incremental software development.

**Function Point (FP):** This is the industry standard for sizing software. FPs are a measurement unit to show how functional an information system is to the user in the business. It uses past projects to calculate the hourly or monetary cost.

**Enhanced Version of Intermediate COCOMO (REVIC)**:Revised This method modifies the COCOMO process by including testing, integration, and maintenance in the publicly available copyrighted program. It is primarily used for military projects.

**Source Lines of Code (SLOC):** Also called lines of code (LOC), this method counts the number of all source code command lines as the base unit metric of the project size. Many complex software methods use this method as the base.

**Weighted Micro Function Points (WMFP):** This method calculates the size of the project by parsing the program code into micro functions to create volume metrics and code complexity. Teams calculate a final effort score with the WMFPs. This method is compatible with Waterfall software development lifecycle, Agile, and Six Sigma. The estimate results are more accurate than traditional sizing methods.

**Non-Algorithmic Methods**

Non-algorithmic estimations analyse assembled historic data to infer the best estimate.

**Activity-Based Costing**: This method considers the activities of the people and equipment necessary to deliver the product. This style evaluates the indirect costs in proportion to the type of work, as well as the resource demand of each activity.

**Analogy:** This method uses previous projects to compare and estimate the cost, size, complexity, or schedule. The method takes into account the similarities, differences, and actual results. You do not need an expert for this method, but must have fully accurate historic data.

**Bottom-Up:** This technique starts the estimate from the low-level details of the work breakdown structure (WBS) and builds up to a higher level. This method provides a high level of accuracy due to its attention to detail.

**Delphi:** This is a structured and systematic Expert Judgment method subset to forecast with an expert panel. The panel weighs in with opinions and debates the factors that influence the estimate. The method assumes that the accuracy from structured group interactions is more reliable than from unstructured group interactions. Other subset adaptations of Delphi are available, like the mini-Delphi or the estimate-talk-estimate (ETE). This method removes politics and bias from the estimate, but it may be time-consuming.

**Expert Judgment (EJ):** In this method, you consult experts who have extensive experience and have completed similar projects. This method can be highly subjective, inconsistent, and unstructured, which poses risks to accuracy.

Planning Poker: In this gamified method, teams use a card deck to build consensus. This helps to avoid anchoring, which is a bias toward the first estimate. This method is a variation of the Wideband Delphi, most often used in Scrum and XP in Agile development.

**Process Group:** This is another Wideband Delphi variant. Here, the estimation team collectively creates a WBS and details any assumptions. From this meeting, individual team members estimate effort, and use these projections in a second meeting, where they reach a consensus.

**Relative Sizing (T-Shirt Sizing and Fibonacci Sequence):** Here, the team breaks down work tasks and sizes them in relative values to each other. These methods are popular in Agile software development, especially when there is a significant amount of ambiguity. These methods use an iterative process to quickly assemble historic data from the team’s past projects to improve estimate accuracy over time. The most common methods are T-Shirt sizing and Fibonacci sequence sizing, which estimate the relative sizes of T-shirts (XS, S, M, L, XL) or the Fibonacci sequence (1, 2, 3, 5, 8, 13). This method provides a quick estimate, which allows the team to get started more quickly. However, this method requires a greater degree of management trust up front, since accuracy only improves by continually using this method over time.

Teams break down their work into tasks, or story points, and then place them on an empty chart or graph based on the amount of effort and risk in relation to the other tasks. At this stage, only the facilitator knows the specific sizing numbers to keep the team focused on the relative size to each other. Once all story points are placed on the chart, the facilitator reveals the sizing underneath. The team then organizes the assigned points and estimates how long the project will take (or how much work can be completed in a given cycle, or sprint) based on the team’s capacity.

## 

## 7.3 Future Scope of This Project

1.End-to-End Encryption: Strengthen the security of the messaging app by

implementing end-to-end encryption to ensure the privacy and confidentiality of user

communications.

2.Multimedia Features: Integrate additional multimedia features, such as image and

video sharing, voice messages, and file attachments, to enhance the richness of

communication.

3.User Customization: Provide users with more customization options, allowing them to

personalize their app experience, such as choosing themes, customizing chat

backgrounds, or selecting notification preferences.

4.Advanced Search and Organization: Enhance search capabilities within the app,

allowing users to easily find and organize messages, contacts, and media. Implement

features like message filtering, sorting, and labelling.

5.Cross-Platform Compatibility: Develop the app for various platforms, ensure ng a

consistent and seamless user experience across different devices and operating

systems, including iOS, Android, and web platforms.

6.Conaboration Tools: Integrate collaboration tools such as shared calendars, task lists, and document collaboration, making the messaging app a central hub for both

personal and professional communication. Features: Implement artificial intelligence (AI) and machine learning (ML) algorithms to enhance user experience, such as smart replies, predictive typing, and

personalized content recommendations.

8.Voice and Video Caning: Extend the communication capabilities by incorporating

voice and video calling features, enabling users to make high-quality calls directly

within the app.

# 8. APPENDICES

## 8.1 Source Code:

Main.dart

import 'dart:developer';

import 'package:Messager/screens/splash\_screen.dart';

import 'package:Messager/theme/dark\_theme.dart';

import 'package:Messager/theme/light\_theme.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:firebase\_core/firebase\_core.dart';

import 'package:flutter/material.dart';

import 'package:flutter/services.dart';

import 'package:flutter\_notification\_channel/flutter\_notification\_channel.dart';

import 'package:flutter\_notification\_channel/notification\_importance.dart';

import 'firebase\_options.dart';

//global object for accessing device screen size

late Size mq;

void main() {

WidgetsFlutterBinding.ensureInitialized();

//enter full-screen

SystemChrome.setEnabledSystemUIMode(SystemUiMode.edgeToEdge);

//for setting orientation to portrait only

SystemChrome.setPreferredOrientations(

[DeviceOrientation.portraitUp, DeviceOrientation.portraitDown])

.then((value) {

\_initializeFirebase();

runApp(const MyApp());

});

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Messager',

debugShowCheckedModeBanner: false,

theme: lightTheme,

darkTheme: darkTheme,

home: const SplashScreen(),

);

}

}

\_initializeFirebase() async {

await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);

var result = await FlutterNotificationChannel.registerNotificationChannel(

description: 'For Showing Message Notification',

id: 'chats',

importance: NotificationImportance.IMPORTANCE\_HIGH,

name: 'Chats');

log('\nNotification Channel Result: $result');

FirebaseAuth.instance.setPersistence(Persistence.SESSION);

}

Homescreen.dart

import 'dart:developer';

import 'dart:io';

import 'dart:ui';

import 'package:Messager/screens/groups.dart';

import 'package:cached\_network\_image/cached\_network\_image.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:flutter/cupertino.dart';

import 'package:flutter/material.dart';

import 'package:flutter/services.dart';

import 'package:google\_sign\_in/google\_sign\_in.dart';

import 'package:liquid\_pull\_to\_refresh/liquid\_pull\_to\_refresh.dart';

import '../api/apis.dart';

import '../helper/dialogs.dart';

import '../main.dart';

import '../models/chat\_user.dart';

import '../widgets/chat\_user\_card.dart';

import 'auth/login\_screen.dart';

import 'profile\_screen.dart';

class HomeScreen extends StatefulWidget {

final ChatUser user; // Add this line to accept a ChatUser object

const HomeScreen({Key? key, required this.user}) : super(key: key);

@override

State<HomeScreen> createState() => \_HomeScreenState();

}

class \_HomeScreenState extends State<HomeScreen> {

// for storing all users

List<ChatUser> \_list = [];

int \_currentIndex = 0;

final \_formKey = GlobalKey<FormState>();

String? \_image;

// for storing searched items

final List<ChatUser> \_searchList = [];

// for storing search status

bool \_isSearching = false;

Future<void> \_handleRefresh() async {

return await Future.delayed(const Duration(seconds: 2));

}

@override

void initState() {

super.initState();

APIs.getSelfInfo();

//for updating user active status according to lifecycle events

//resume -- active or online

//pause -- inactive or offline

SystemChannels.lifecycle.setMessageHandler((message) {

log('Message: $message');

if (APIs.auth.currentUser != null) {

if (message.toString().contains('resume')) {

APIs.updateActiveStatus(true);

}

if (message.toString().contains('pause')) {

APIs.updateActiveStatus(false);

}

}

return Future.value(message);

});

}

@override

Widget build(BuildContext context) {

mq = MediaQuery.of(context).size;

return GestureDetector(

onTap: () => FocusScope.of(context).unfocus(),

child: WillPopScope(

onWillPop: () {

if (\_isSearching) {

setState(() {

\_isSearching = !\_isSearching;

});

return Future.value(false);

} else {

return Future.value(true);

}

},

child: Scaffold(

backgroundColor: Theme.of(context).colorScheme.background,

drawer: Drawer(

backgroundColor: Theme.of(context).colorScheme.primary,

elevation: 50,

child: Column(

crossAxisAlignment: CrossAxisAlignment.stretch,

children: <Widget>[

const SizedBox(

height: 70,

),

Center(

child: Form(

key: \_formKey,

child: Container(

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(20),

color: Theme.of(context).colorScheme.background,

),

child: Padding(

padding: const EdgeInsets.all(10.0),

child: Stack(

children: [

//profile picture

\_image != null

?

//local image

Container(

decoration: BoxDecoration(

borderRadius:

BorderRadius.circular(mq.height \* .2),

border: Border.all(

color: Colors

.white, // Choose your desired border color

width:

5.0, // Choose your desired border width

),

),

child: ClipRRect(

borderRadius: BorderRadius.circular(

mq.height \* .1),

child: Image.file(File(\_image!),

width: mq.height \* .1,

height: mq.height \* .1,

fit: BoxFit.cover)),

)

:

//image from server

Container(

decoration: BoxDecoration(

borderRadius:

BorderRadius.circular(mq.height \* .2),

border: Border.all(

color: Colors

.deepPurpleAccent, // Choose your desired border color

width:

5.0, // Choose your desired border width

),

),

child: ClipRRect(

borderRadius:

BorderRadius.circular(mq.height \* .1),

child: CachedNetworkImage(

width: mq.height \* .1,

height: mq.height \* .1,

fit: BoxFit.cover,

imageUrl: widget.user.image,

errorWidget: (context, url, error) =>

const CircleAvatar(

child: Icon(CupertinoIcons

.person\_alt\_circle)),

),

),

),

],

),

),

),

),

),

const SizedBox(

height: 30,

),

Container(

height: 40,

margin: const EdgeInsets.only(left: 10, right: 10),

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(15),

color: Theme.of(context).cardColor,

),

child: Center(

child: Text(

widget.user.name,

style: Theme.of(context).textTheme.titleLarge,

),

),

),

const SizedBox(

height: 50,

),

const Divider(

thickness: 2,

),

Container(

margin: const EdgeInsets.only(left: 10, right: 10),

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(15),

color: Theme.of(context).cardColor,

),

child: CupertinoListTile.notched(

leading: const Icon(

CupertinoIcons.group\_solid,

color: CupertinoColors.white,

size: 30,

),

title: Text(

'Groups',

style: Theme.of(context).textTheme.titleLarge,

),

onTap: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (\_) => const GroupsPage()));

},

),

),

const Divider(

thickness: 2,

),

Container(

margin: const EdgeInsets.only(left: 10, right: 10),

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(15),

color: Theme.of(context).cardColor,

),

child: CupertinoListTile.notched(

leading: const Icon(

CupertinoIcons.settings,

color: CupertinoColors.white,

),

title: Text(

'Settings',

style: Theme.of(context).textTheme.titleLarge,

),

onTap: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (\_) => ProfileScreen(user: APIs.me)));

},

),

),

const Divider(

thickness: 2,

),

Container(

margin: const EdgeInsets.only(left: 10, right: 10),

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(15),

color: Theme.of(context).cardColor,

),

child: CupertinoListTile.notched(

leading: const Icon(

CupertinoIcons.square\_arrow\_left,

color: CupertinoColors.systemRed,

),

title: const Text(

'Logout',

style: TextStyle(

color: CupertinoColors.systemRed,

),

),

onTap: () async {

//for showing progress dialog

Dialogs.showProgressBar(context);

await APIs.updateActiveStatus(false);

//sign out from app

await APIs.auth.signOut().then((value) async {

await GoogleSignIn().signOut().then((value) {

//for hiding progress dialog

Navigator.pop(context);

//for moving to home screen

Navigator.pop(context);

APIs.auth = FirebaseAuth.instance;

//replacing home screen with login screen

Navigator.pushReplacement(

context,

MaterialPageRoute(

builder: (\_) => const LoginScreen()));

});

});

},

),

),

const Divider(

thickness: 2,

),

],

),

),

//app bar

appBar: AppBar(

title: \_isSearching

? CupertinoTextField(

cursorColor: Theme.of(context).colorScheme.tertiary,

placeholder: " Search..",

showCursor: true,

autofocus: true,

style: Theme.of(context).textTheme.titleMedium,

onChanged: (val) {

//search logic

\_searchList.clear();

for (var i in \_list) {

if (i.name.toLowerCase().contains(val.toLowerCase()) ||

i.email.toLowerCase().contains(val.toLowerCase())) {

\_searchList.add(i);

setState(() {

\_searchList;

});

}

}

},

)

: const Text('Chats'),

actions: [

//search user button

IconButton(

onPressed: () {

setState(() {

\_isSearching = !\_isSearching;

});

},

icon: Icon(

\_isSearching

? CupertinoIcons.xmark\_square\_fill

: CupertinoIcons.search,

)),

],

),

// ignore: sized\_box\_for\_whitespace

floatingActionButton: FloatingActionButton(

backgroundColor: Colors.blueAccent,

onPressed: () {

\_addChatUserDialog();

},

child: const Icon(CupertinoIcons.person\_add\_solid,

color: Colors.white),

),

bottomNavigationBar: CupertinoTabBar(

backgroundColor: Theme.of(context).colorScheme.primary,

activeColor: Colors.white,

inactiveColor: Colors.grey.shade500,

currentIndex:

\_currentIndex, // You need to maintain a currentIndex variable

onTap: (int index) {

// Handle navigation based on the tapped index

setState(() {

\_currentIndex = index; // Update the currentIndex

});

switch (index) {

case 0:

// Navigate to the Groups page

Navigator.push(

context,

MaterialPageRoute(

builder: (\_) => HomeScreen(user: APIs.me),

),

);

break;

case 1:

// Navigate to the Settings page

Navigator.push(

context,

MaterialPageRoute(

builder: (\_) => const GroupsPage(),

),

);

break;

case 2:

Navigator.push(

context,

MaterialPageRoute(

builder: (\_) => ProfileScreen(user: APIs.me),

),

);

break;

}

},

items: const [

BottomNavigationBarItem(

icon: Icon(

CupertinoIcons.chat\_bubble\_fill,

),

label: 'Chats',

),

BottomNavigationBarItem(

icon: Icon(

CupertinoIcons.group\_solid,

size: 40,

),

label: 'Groups',

),

BottomNavigationBarItem(

icon: Icon(CupertinoIcons.settings\_solid),

label: 'Settings',

),

],

),

//body

body: StreamBuilder(

stream: APIs.getMyUsersId(),

//get id of only known users

builder: (context, snapshot) {

switch (snapshot.connectionState) {

//if data is loading

case ConnectionState.waiting:

case ConnectionState.none:

return const Center(

child: CupertinoActivityIndicator(

radius: 25,

color: Colors.grey,

));

//if some or all data is loaded then show it

case ConnectionState.active:

case ConnectionState.done:

return StreamBuilder(

stream: APIs.getAllUsers(

snapshot.data?.docs.map((e) => e.id).toList() ?? []),

//get only those user, who's ids are provided

builder: (context, snapshot) {

switch (snapshot.connectionState) {

//if data is loading

case ConnectionState.waiting:

case ConnectionState.none:

// return const Center(

// child: CircularProgressIndicator());

case ConnectionState.active:

case ConnectionState.done:

final data = snapshot.data?.docs;

\_list = data

?.map((e) => ChatUser.fromJson(e.data()))

.toList() ??

[];

if (\_list.isNotEmpty) {

return LiquidPullToRefresh(

onRefresh: \_handleRefresh,

color: Theme.of(context).colorScheme.primary,

backgroundColor:

Theme.of(context).colorScheme.tertiary,

height: 100,

animSpeedFactor: 10,

showChildOpacityTransition: false,

springAnimationDurationInMilliseconds: 300,

child: ListView.builder(

itemCount: \_isSearching

? \_searchList.length

: \_list.length,

padding:

EdgeInsets.only(top: mq.height \* .01),

physics: const BouncingScrollPhysics(

parent: AlwaysScrollableScrollPhysics()),

itemBuilder: (context, index) {

return ChatUserCard(

user: \_isSearching

? \_searchList[index]

: \_list[index]);

}),

);

} else {

return const Center(

child: Text(' ', style: TextStyle(fontSize: 20)),

);

}

}

},

);

}

},

),

),

),

);

}

// for adding new chat user

void \_addChatUserDialog() {

String email = '';

showCupertinoDialog(

context: context,

builder: (context) => Stack(

children: [

Positioned.fill(

child: BackdropFilter(

filter: ImageFilter.blur(

sigmaX: 5, sigmaY: 5), // Adjust the blur intensity as needed

child: Container(

color: Colors

.transparent, // You can set a background color here if needed

),

),

),

CupertinoAlertDialog(

// No direct equivalent for contentPadding, use padding for content spacing

title: const Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Icon(

CupertinoIcons.person\_crop\_circle\_fill\_badge\_plus,

color: Colors.blueAccent,

//size: 28,

),

Text(

' Add User',

style: TextStyle(color: Colors.blueAccent),

)

],

),

content: Padding(

padding: const EdgeInsets.fromLTRB(5, 10, 10, 5),

child: CupertinoTextField(

maxLines: null,

onChanged: (value) => email = value,

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(10),

border: Border.all(color: CupertinoColors.systemGrey),

),

placeholder: 'Email Id',

prefix: const Padding(

padding: EdgeInsets.only(left: 5),

child: Icon(

CupertinoIcons.mail\_solid,

color: Colors.blueAccent,

),

),

),

),

actions: [

CupertinoDialogAction(

onPressed: () {

// Hide alert dialog

Navigator.pop(context);

},

child:

const Text('Cancel', style: TextStyle(color: Colors.red)),

),

# 9. CONCLUSION

Creating a comprehensive messaging app involves a multi-faceted approach, including front-end development, back-end logic, real-time communication, and data storage. While the provided example demonstrates a basic front-end interface using HTML, CSS, and JavaScript, a complete messaging app would require much more sophisticated development, including:

1. Back-End Development:

- User authentication and authorization.

- Database integration to store user data and messages.

- Server logic to handle message routing and storage.

- Real-time communication using technologies like WebSockets.

2. Front-End Development:

- A more complex user interface with features like chat history, multimedia sharing, and user customization.

- Integration with back-end services to fetch and display data.

- Consideration of accessibility and responsive design.

3. Security Considerations:

- Implementation of secure user authentication and authorization.

- End-to-end encryption for message privacy.

- Protection against common security threats such as cross-site scripting (XSS) and cross-site request forgery (CSRF).

4. Testing:

- Rigorous testing for both front-end and back-end components.

- Consideration of unit tests, integration tests, and end-to-end tests.

- Load testing for assessing system performance under heavy usage.

5. Deployment:

- Deployment of the application on servers or cloud platforms.

- Configuration of domain names, SSL certificates, and other deployment-related tasks.

6. Maintenance and Updates:

- Regular monitoring and maintenance to ensure the app's continued functionality and security

# REFERENCE and BIBLIOGRAPHY

* Flutter Documentation: The official documentation for Flutter provides a comprehensive guide, tutorials, and samples to get started with Flutter development.
* Firebase Documentation: The official Firebase documentation offers detailed information on various Firebase services, including Firestore (real-time NoSQL database), Firebase Authentication, Cloud Functions, and more.
* FlutterFire GitHub Repository: The FlutterFire repository on GitHub contains a collection of Flutter plugins for Firebase services. You can find plugins for authentication, Firestore, Cloud Functions, Cloud Messaging, and other Firebase features.
* Firebase in Flutter Codelab: This codelab guides you through the process of integrating Firebase services into a Flutter app. It covers Firebase Authentication, Firestore, and Cloud Functions.
* Official Flutter YouTube Channel: The official Flutter YouTube channel provides a variety of videos, including tutorials and discussions about Flutter and its integration with Firebase.
* Firebase Flutter Community on GitHub: Join the Firebase Flutter Community on GitHub for discussions, issue tracking, and community-driven development of Firebase plugins for Flutter.