COMPANION BOT USING FLUTTER FRAMEWORK

A MINI PROJECT REPORT

Submitted by

SHARUN R	**[711720104086]**
SUJITHA K	**[711720104094]**
TAMILARASI P	**[711720104099]**
VIGNESWARAN S	**[711720104104]**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KGiSL INSTITUTE OF TECHNOLOGY, SARAVANAMPATTI

ANNA UNIVERSITY: CHENNAI 600 025

May 2023

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**COMPANION BOT USING FLUTTER FRAMEWORK**" is the bonafide work of "**SHARUN R, SUJITHA K, TAMILARASI P, VIGNESWARAN S**" who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. THENMOZHI T	**Ms. AARTHI T**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
PROFESSOR	ASSISTANT PROFESSOR
Computer Science and Engineering	Computer Science and Engineering
KGiSL Institute of Technology	KGiSL Institute of Technology
Coimbatore-641035	Coimbatore-641035

Submitted for the Anna University Viva-Voce examination held on ______________

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

We express our deepest gratitude to our **Chairman and Managing Director Dr. Ashok Bakthavachalam** for providing us with an environment to complete our project successfully.

We are grateful to our CEO, Academic Initiatives **Mr. Aravind Kumar Rajendran,** our beloved Director-Academics **Dr. P. Shankar** and our honorable Secretary **Dr. N. Rajkumar**. We are thankful to our beloved Principal **Dr. S. Suresh Kumar** for his support, valuable guidance and blessings.

We would like to thank **Dr. T. Thenmozhi, M.E.,M.B.A., Ph. D,** Head of the Department, Department of Computer Science and Engineering for her unwavering support during the entire course of this project work. We express our sincere thanks to **Ms. Aruna T N, M.E.,**, our Project Coordinator, Assistant Professor, Department of Computer Science and Engineering who modelled us both technically and morally for achieving greater success in completing this project work.

We express our sincere and heartful thanks to our faculty guide **Ms.Aarthi T, M.E.,** Assistant Professor, Department of Computer Science and Engineering for her constant encouragement and support throughout our course, especially for the useful suggestions given during the course of the project period and being instrumental in the completion of our project with her complete guidance.

We also thank all the **Faculty members** of our department and finally, we take this opportunity to extend our deep appreciation to our **Family** and **Friends**, for all they meant to us during the crucial times of the completion of our project.

ABSTRACT

Mental health is an important issue in the world today. With a large population now working from home and staying away from loved ones, the mental health situation has deteriorated. As such, it becomes important to track and remedy any problems before they get too serious. Companion robot is a robot created for the purposes of creating real or apparent companionship for human beings. Target markets for companion robots include the elderly and single children. A robot that supports a person's everyday life. Social robots range from slightly animated stuffed animals to intelligent android-like devices that function as real companions. Companionbot is an intelligent and versatile personal digital assistant designed to enhance your daily life. Built using the Flutter framework, this cross-platform companion bot provides a seamless user experience on both Android and iOS devices. With its intuitive interface and advanced features, Companionbot aims to be your trusted companion for various tasks and interactions. It is capable of speech and movement and can detect and track people at home.In current world it helps to comeout from over mental pressure. And it will try to get an idea of the mental state of your user (in the least intrusive ways), find out if they are suffering and then suggest measures they can take to get out of their present condition. A user answers some questions and based on the answers that they provide, you will suggest tasks to them and maintain a record of their mental state for displaying on a dashboard. Due to the deployment of this system, the chances of mentally suffered people's situation can be minimized.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ML	Machine Learning
AI	Artificial Intelligence
VS	Visual Studio
iOS	iPhone Operating System
QA	Question and Answer
AUT	Automated Ultrasonic Testing
IDE	Integrated Development Environment

CHAPTER 1

INTRODUCTION

Mental health is an important issue in the world today. With a large population now working from home and staying away from loved ones, the mental health situation has deteriorated. As such, it becomes important to track and remedy any problems before they get too serious. We try achieving this using the Companion bot.

This project involves building a simple app in Flutter that tracks the mental health of its users and tries to help them get through their condition by suggesting tasks and keeping record of their progress. Interactions were recorded, participants' comments and observations were transcribed, and content was analysed.

PROBLEM DEFINITION

- It demonstrate stark differences in preferences and requirement between mentally affected people and bots, suggesting that engaging the end user in the design and development of companion robots is essential.
- Companion robots may reduce agitation and depression for mentally affected people.

1.1 OBJECTIVE OF THE PROJECT

- Mental health is an important issue in the world today. With a large population now working from home and staying away from loved ones, the mental health situation has deteriorated. As such, it becomes important to track and remedy any problems before they get too serious. We try achieving this using the Companion Bot.
- The project focuses on building a mental health tracker. You will try to get an idea of the mental state of your user, find out if they are suffering and then suggest measures they can take to get out of their present condition.

- **Algorithm:** In this project, we use three types of algorithm User Input Processing, Natural Language Processing (NLP), Decision-making and Response Generation.
- Utilize NLP techniques to analyze the user input.
- This can involve tracking user interactions
- **Interface:** Many companion bots use a chat-based interface to simulate a conversation with the user. The interface typically consists of a chat window or chat bubbles where the user's messages and the bot's responses are displayed.

1.2 SIGNIFICANCE OF THE PROJECT

- They are used to recognize people's mental stage.
- Helps the people with communication of their present state to solve the problem.
- Performing the tasks that helps mentally affected people.

1.3 OUTLINE OF THE PROJECT

- The system evaluates the mental health situation has deteriorated. As such, it becomes important to track and remedy any problems before they get too serious. We try achieving this using the Companion bot.
- Later it initiates through using flutter framework and it get developed.
- This type of protocol is used to develops the app for tracking user's information.

CHAPTER 2

LITERATURE REVIEW

Mental health is an important issue in the world today and people facing many problems. With the help of a literature survey, we realized that the basic steps in performing the tasks are:

- Set up Flutter
- Design the User Interface (UI):
- Implement Natural Language Processing (NLP):
- Generate Bot Responses

2.1 Data Acquisition

The different approaches to acquiring data about companion bot can be done in the following ways:

2.1.1 Analytics and Usage Tracking:

Utilize analytics tools to track user interactions and behaviors within the companion bot application. This can include tracking the paths users take, the frequency of certain actions, or the time spent in different sections of the app. Analyzing this data provides insights into user engagement and usage patterns.

2.1.2 Surveys and Questionnaires:

Implementing surveys or questionnaires within the companion bot can allow users to provide structured feedback and input. By designing targeted questions, you can gather specific information about user preferences, satisfaction levels, or suggestions for improvement.

2.2 Build screen and UI refinement:

- We will set up the basic skeleton of the app first. Use your creativity to build an intuitive app and design the elements in such a way that they are appealing to people of all backgrounds. Make sure the workflow is easy to understand and follow. Ditch any design choice that would require the user to take a complex route to achieving a goal that can be done in a simpler way.
- All the screens should be built as shown in the images. The app doesn't have any logic implemented yet, just the facade. Dashboard.dart, Tasks.dart and CustomTabBar.dart should be written by this stage.
- This task focuses on making the app friendly to its target audience. You have to remember that potential users of this app would be suffering from mental illness and as such it won't be wrong to assume that asking a lot of questions won't be the best choice. In addition, the mechanism adopted for asking said questions need to be the least intrusive one you can think of. That way, we get the required information at minimum inconvenience to the end-user. Another detail worth mentioning is the double-tap to disappear feature. Once the user is done with a task, the user can get rid of it from the screen using just a double-tap. Easy, intuitive and satisfying **-** just the way we want it! An important thing to note while building the screens and using the variables that power your app, is state management

2.3 Functional Classification:

Personal Assistants: These companion bots focus on assisting users with various tasks and providing information or recommendations. Examples include Apple's Siri, Google Assistant, or Amazon's Alexa.

Entertainment Bots: These bots are designed to provide entertainment, engage in fun conversations, or offer games and interactive experiences. Examples include chatbots like Cleverbot or Mitsuku, which aim to entertain users through engaging conversations.

Emotional Support Bots: These bots are designed to provide emotional support, companionship, or act as virtual friends. They may engage in empathetic conversations, offer motivational messages, or provide a listening ear. Examples include Replika or Woebot.

D**omain-specific Bots:** These bots are built for specific domains or industries and focus on providing specialized assistance or information. Examples include customer service bots, language learning bots, or healthcare bots.

Platform Classification:

Mobile Companion Apps: These bots are developed as mobile applications, designed to run on smartphones or tablets. They typically offer a range of functionalities and engage users through a mobile interface.

Text-based Bots: These bots primarily engage with users through text-based conversations. Users interact with the bot by typing messages, and the bot responds with text-based replies.

Voice-based Bots: These bots rely on voice interactions, where users speak commands or questions, and the bot responds using text-to-speech or voice-based responses.

Multimodal Bots: These bots combine both text-based and voice-based interactions, allowing users to switch between typing and speaking, depending on their preference or convenience.

Rule-based Bots: These bots follow predefined rules and patterns to generate responses based on specific triggers or keywords. They have limited capabilities for understanding and adapting to user input.

Machine Learning Bots: These bots leverage machine learning techniques to improve their responses and behavior over time. They can learn from user interactions, adapt to user preferences, and generate more contextually relevant responses.

Deep Learning Bots: These bots utilize deep learning algorithms, such as neural networks, to enhance their understanding of user input and generate more sophisticated and natural responses. They can learn from large datasets and make more complex decisions.

These classification models provide a way to categorize companion bots based on their functionalities, platforms, interaction modalities, or underlying AI techniques. However, it's important to note that many companion bots may fall into multiple categories, as they can have overlapping features or functionalities. The classification depends on the specific characteristics and goals of the companion bot in question.

CHAPTER 3

SYSTEM ANALYSIS

System analysis is a problem-solving technique that decays a system into component pieces to study how well those parts work and interact to accomplish their purpose. The following chapter provides a detailed description of the existing system. It also provides an overview of the proposed system and the feasibility of companion bot using flutter.

3.1 EXISTING SYSTEM

The Existing System which is also built to support the elderly, helps its owner interact with smart home environment as well as care givers. It is capable of speech and movement and can detect and track people at home.

3.2 DRAWBACKS

- Performance limitations: This can be noticeable in complex or graphics-intensive applications where the performance requirements are high.
- Limited access to native APIs.
- Larger app size: Flutter apps tend to have a larger file size compared to their native counterparts.

3.3 PROPOSED SYSTEM

Our proposed system is a companion bot using flutter framework which recognize the companion bot can have a user-friendly interface accessible through a mobile app or web portal. The interface should provide easy navigation and clear communication channels between the user and the bot.

The bot can be designed to learn and understand the user's preferences, habits, and interests over time. This enables it to provide personalized recommendations, suggestions, and content based on the user's individual needs and preferences.

3.4 FEASIBILITY STUDY

An analysis and evaluation of a proposed project to determine if it is technically feasible within the estimated cost and will be profitable. Feasibility studies are almostalways conducted where large sums are at stake. A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of existing sign language applications and threats present in the environment, the resources required to carry through, and ultimately the prospects for success related Sign language.

3.4.1 Tests of Feasibility

- A feasibility study for a companion bot using Flutter would assess the practicality and viability of developing such a bot using the Flutter framework. Here are some key aspects to consider in the feasibility study:

3.4.1.1 Technical Feasibility

Flutter's cross-platform capabilities:

Evaluate whether Flutter's cross-platform nature is suitable for the companion bot, allowing it to be developed for both Android and iOS platforms with a single codebase.

Required integrations:

Determine if Flutter provides the necessary plugins or libraries to integrate with the required APIs, services, and hardware features essential for the companion bot's functionality.

Performance requirements:

Consider the performance needs of the bot and assess whether Flutter's rendering engine can meet those requirements, especially for resource-intensive features like real-time interactions or complex animations.

The essential questions that help in testing the technical feasibility of a system include the following:

- Is the project feasible within the limits of current technology?
- Does the technology exist at all?
- Is it available within given resource constraints

3.4.1.2 Operational Feasibility

Operational feasibility is dependent on human resources available for the project and involves projecting whether the system will be used if it is developed and implemented. Operational feasibility is a measure of how well a proposed system in companion bot using flutter framework solves the problems and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of companion bot.

The essential questions that help in testing the operational feasibility of a system include the following:

- Does the current mode of operation provide adequate throughput and response time?
- Does the current mode provide end-users and managers with timely, accurate, and useful formatted information?

3.4.1.3 Economical Feasibility

The economic analysis could also be referred to as cost/benefit analysis. It is the most frequently used method for evaluating the effectiveness of a new system. In economic analysis, the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs.

Possible questions raised in economic analysis are:

- Is the system cost-effective?
- Do benefits outweigh costs and system study?

CHAPTER 4

SYSTEM SPECIFICATION

4.1 HARDWARE REQUIREMENTS

Operating system	: Windows / Mac OS / Linux
Ram	: Minimum 8GB
Processor	: 64 bit processor intel core i3 or higher
Disk space	: 4 GB
Screen Resolution	: 1200 x 800 pixels

4.2 SOFTWARE REQUIREMENTS

Visual Studio Code	: Version (1.78.2)
Flutter	: Version (3.7.8)
Android studio	: Version (4.2.2)

CHAPTER 5

SOFTWARE DESCRIPTION

5.1 VISUAL STUDIO CODE

Visual Studio Code (VS Code) is a lightweight, cross-platform code editor developed by Microsoft. Although it's not an IDE specifically tailored for mobile app development, VS Code has become a popular choice for Flutter development due to its versatility and extensive ecosystem of extensions. With the help of the Flutter and Dart extensions available for VS Code, developers can benefit from features like code completion, linting, debugging, and hot-reloading while working on Flutter projects. VS Code also offers integration with version control systems and a customizable interface to suit individual preferences.

5.2 FLUTTER

Flutter is an open-source UI software development kit created by Google. It allows developers to build cross-platform applications for mobile, web, and desktop using a single codebase. Flutter uses the Dart programming language and provides a rich set of pre-built UI components and tools for creating visually appealing and performant applications

5.3 ANDROID STUDIO

Android Studio is the official integrated development environment (IDE) for Android app development. It is based on JetBrains' IntelliJ IDEA and provides a comprehensive set of tools and features specifically designed for building Android applications. Android Studio includes a robust code editor, a visual layout editor, debugging tools, and support for testing and profiling Android apps. It also integrates seamlessly with the Android SDK and offers tools for managing dependencies and building app bundles or APKs.

CHAPTER 6

PROJECT DESCRIPTION

A Companion Bot is an innovative and versatile mobile application developed using the Flutter framework. It serves as your personal digital assistant, providing a wide range of features and functionalities to simplify your daily life and enhance your productivity. Track your daily steps, set fitness goals, and receive personalized exercise and nutrition recommendations.

6.1 OVERVIEW OF THE PROJECT

In companion bot using flutter framework, the system evaluates the Companion Bot helps mentally suffered people to balance in day today life. And it will try to get an idea of the mental state of your user (in the least intrusive ways), find out if they are suffering and then suggest measures they can take to get out of their present condition. A user answers some questions and based on the answers that they provide, you will suggest tasks to them and maintain a record of their mental state for displaying on a dashboard.

6.2 MODULE DESCRIPTION

6.2.1 BUILDING A SCREEN

The First Step of building this project was building a screen with the custom tab bar and the Tasks, Dashboard. And to build an intuitive app and design the elements in such a way that they are appealing to people of all backgrounds. Flutter provides a rich set of widgets and customization options to build beautiful and interactive screens for your app.

6.2.2 QUESTIONS SCREEN USING UI REFINEMENT

This task focuses on making the app friendly to its target audience. And to remember that potential users of this app would be suffering from mental illness and as such it won't be wrong to assume that asking a lot of questions won't be the best choice. That way, we get the required information at minimum inconvenience to the end-user. Another detail worth mentioning is the double-tap to disappear feature. Once the user is done with a task, the user can get rid of it from the screen using just a double-tap. Easy, intuitive and satisfying - just the way we want it! An important thing to note while building the screens and using the variables that power your app, is state management.

6.2.3 SETTING UP FIREBASE

We need a backend for our application to provide services like authentication, cloud storage etc. Firebase is one of the easiest ways to get started with building a backend. Depending on the platform or programming language you're using, you'll need to install the Firebase SDKs or libraries. Firebase supports various platforms like web, Android, iOS, Unity, and more. Refer to the Firebase documentation for the specific platform you're targeting. With Firebase set up and the SDKs installed, you can start using Firebase services in your application. The Firebase documentation provides detailed guides and examples for each product, including how to authenticate users, read and write data, store files, and more.

6.2.4 ADDING GOOGLE SIGN IN FEATURE

Once Firebase is set up, we can use one of the authentication providers available (like Google) to add the login feature to our app. This way, user data can be stored and tracked for use across the app. We can also customise suggestions for the user and control access to the app as whole. User information procured from the Google account can also be used for setting up communication with the doctors to prevent users from entering information again. Remember to consult the Firebase documentation and guides specific to your platform for detailed instructions and code samples on implementing Google Sign-In using Firebase Authentication.

6.3 DATA FLOW DIAGRAM

A data flow diagram is used to describe how the information is processed and stored and identifies how the information flows through the processes. The data flow diagram illustrates how the data is processed by a system in terms of inputs and outputs. The data flow diagram also depicts the flow of the process and it has various levels. The initial level is the context level which describes the entire system functionality and the next level describes every sub module in the main system as a separate process or describes all the processes involved in the system separately.

6.4 ARCHITECTURE DIAGRAM

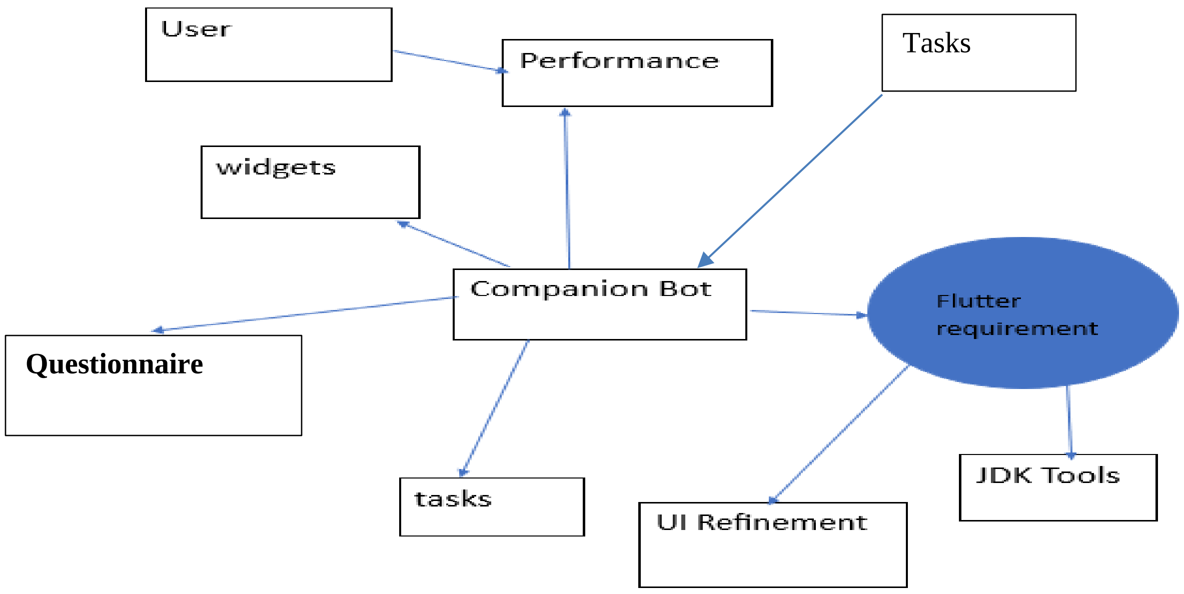


Figure.6.4 Architecture Diagram

6.4 OUTPUT DESIGN

Quality output is one, which meets the requirements of the end-user and presents the information. In any system results of processing are communicated to the users and another system through outputs. In output design, it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source of information for the user. Efficient and intelligent output design improves the system’s relationship to help user decision-making.

CHAPTER 7

SYSTEM TESTING

System Testing is a level of software testing where complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. By definition, ISTQB system testing is the process of testing an integrated systemto verify that it meets specified requirements.

7.1 TESTING METHODS

Software Testing Type is a classification of different testing activities into categories, each having a defined test objective, test strategy, and test deliverables. The goal of having a testing type is to validate the Application under Test for the defined Test Objective.

For instance, the goal of Accessibility testing is to validate that AUT is accessible to disabled people. So, if your Software solution must be disabled-friendly, you check it against Accessibility Test Cases.

7.2 TYPES OF TESTING

7.2.1 Unit Testing

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

In this sign language, every unit of code is tested and the correctness of every module is ensured.

7.2.2 Integration Testing

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them into larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing. In this sign language detection, the units were tested as a whole and the testing was successful.

7.2.3 Functional Testing

Functional testing is a quality assurance (QA) process and a type of black-box testing that basesits test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (unlike white-box testing). Functional testing usually describes what the system does.Functional testing does not imply that you are testing a function (method) of your module or class. Functional testing tests a slice of the functionality of the whole system.

Functional testing has many types:

- Smoke testing
- Sanity testing
- Regression testing
- Usability testing

7.2.4 Stress Testing

Stress testing is a Non-Functional testing technique that is performed as part of performance testing. During stress testing, the system is monitored after subjecting the system to overload to ensure that the system can sustain the stress.

Reasons can include:

- to determine breaking points or safe usage limits
- to confirm the mathematical model is accurate enough in predicting breaking points or safe usage limits
- to confirm intended specifications are being met
- to determine modes of failure (how exactly a system fails)
- to test stable operation of a part or system outside standard usage

The recovery of the system from such a phase (after stress) is very critical as it is highly likely tohappen in a production environment. In this sign language detection, all of the modules are being tested and it has safe usage measures.

7.2.5 Acceptance Testing

Acceptance Testing is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

Formal testing concerning user needs, requirements, and business processes is conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers,or other authorized entity to determine whether or not to accept the system.

In this sign language detection, the system satisfies all the acceptance criteria.

7.2.6 White Box Testing

White Box Testing is the testing of a software solution's internal coding and infrastructure. It focuses primarily on strengthening security, the flow of inputs and outputs through the application,and improving design and usability. White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing.It is one of two parts of the "box testing" approach of software testing. Its counterpart, black box testing, involves testing from an external or end-user type perspective. On the other hand, White boxtesting is based on the inner workings of an application and revolves around internal testing.

The term "white box" was used because of the see-through box concept. The clear box or white box name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "black box testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

In this sign language detection, all the inner functionality is tested and it is correctly implemented.

7.2.7 BlackBox Testing

Black box testing is a software testing technique in which the functionality of the software under test (SUT) is tested without looking at the internal code structure, implementation details, and knowledge of the internal paths of the software. This type of testing is based entirely on the software requirements and specifications. In this sign language detection, the implementation part has been checked for its correctness.

7.2.7.1 Methods of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones -

- Functional testing - This black box testing type is related to the functional requirements of a system; it is done by software testers.
- Non-functional testing - This type of black-box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, and usability.

7.2 TESTING STRATEGY

Test Strategy is also known as the test approach defines how testing would be carried out. The test approach has two techniques:

- Proactive - An approach in which the test design process is initiated as early as possible to find and fix the defects before the build is created.
- Reactive - An approach in which the testing is not started until after design and coding is completed.

Test strategy calls for implementing two entirely different methodologies for testing this project. Sign language detection includes a fair amount of manual UI-based testing.

In this sign language detection, a proactive approach is being used for testing. Since the proactive approach is efficient t has been used in this language.

CHAPTER 8

SYSTEM IMPLEMENTATION

8.1 Design the UI

For the project, we tried to setup the requirements that statisfied. All we could find were the designing of User Interface. Hence, we decided to create our own app by setting. The steps we followed to create our setup are as follows. Design the user interface for your companion bot using Flutter widgets. Consider the various screens and components you'll need, such as chat bubbles, input fields, and buttons.

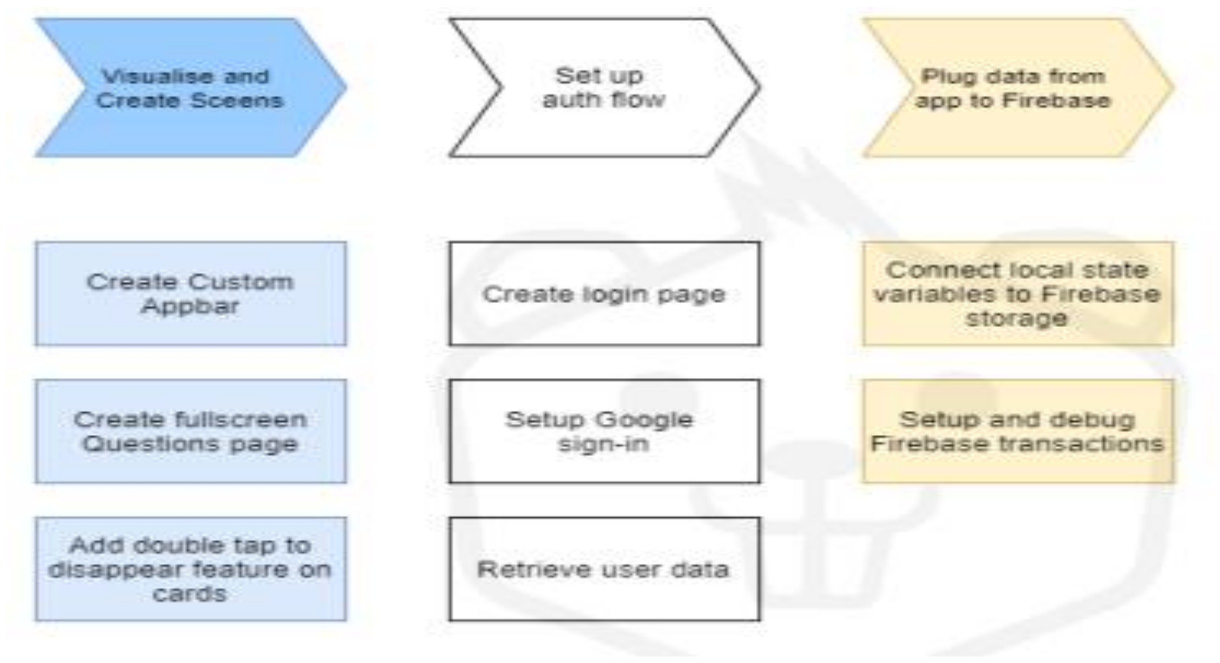

Visualise and Create Sceens
Set up auth flow
Plug data from app to Firebase
Create Custom Appbar
Create login page
Connect local state variables to Firebase storage
Create fullscreen Questions page
Setup Google sign-in
Setup and debug Firebase transactions
Add double tap to disappear feature on cards
Retrieve user data

Figure.8.1 Setting Environment

Algorithm Layer 1

- **Choosing a Design Style**: Determining the overall design style of the app, such as material design or iOS-style design.
- selecting Flutter widgets and designing the app's visual elements

Algorithm Layer 2

- **Using Widgets for Structure and Layout**: Flutter provides a wide range of built-in widgets for structuring and laying out your app's user interface.
- Some commonly used layout widgets include Container, Row, Column, Stack, and List View.
- Using these widgets to arrange and organize your UI elements.

Algorithm Layer 3

- Creating UI Components: App's UI into smaller reusable components. For example, a chat screen, might create separate components for chat bubbles, input fields, and buttons.
- Use Flutter widgets to build these components and apply appropriate styling.

8.2 IMPLEMENT CHAT FUNCTIONALITY:

- Creating a chat screen where users can interact with the companion bot. Using Flutter widgets like ListView and TextField to display messages and capture user input.
- Managing the state of the conversation, including storing and displaying previous messages.
- Adding the necessary dependencies to the Flutter project. It might considered using packages like flutter_bloc or provider for state management and socket_io_client or firebase_core for real-time communication. Updating the pubspec.yaml file and running flutter pub to fetch the dependencies.
- Designing the chat user interface using Flutter widgets. BY using a ListView to display the chat messages, TextFormField for the message input field, and a send button to submit messages. Bubble or Card widgets to style and display the chat message can be used.

8.3 INTEGRATE NATURAL LANGUAGE PROCESSING(NLP)

- To understand and respond to user inputs, integrate a natural language processing library or service. Popular options include Dialogflow, Wit.ai, or Rasa. These services can analyze user messages and extract intent and entities

8.4 GENERATE BOT RESPONSES:

- Based on the user's intent, entities, and any additional context, generate appropriate bot responses. These responses can be predefined templates, dynamically generated based on data or APIs, or a combination of both. Use Flutter widgets to display the bot's responses in the chat interface.

8.5 TRAINING AND TESTING:

- Test the Natural Language Processing(NLP) integration thoroughly, including different user inputs and scenarios. Verifying that the intents and entities are correctly identified, and the bot responses align with the user's expectations. Continuously iterate and refine your NLP implementation based on user feedback and usage patterns. Testing and refining a companion bot implemented in Flutter involves ensuring its functionality, conversation flow, and overall user experience.

CHAPTER 9

CONCLUSION & FUTURE ENHANCEMENT

9.1 CONCLUSION

- In conclusion, developing a companion bot using Flutter can provide users with a valuable and interactive experience. Throughout the project, various features and enhancements can be implemented to create a more robust and engaging companion bot.
- We have provided empirical support for the necessity and designed in the development of companion bots targeted for mentally affected people. Our results demonstrate stark differences in preferences and requirement between mentally affected people and bots, suggesting that engaging the end user in the design and development of companion robots is essential.
- By incorporating natural language processing (NLP) capabilities, the companion bot can better understand user queries and conversations, leading to more accurate and relevant responses.

9.2 FUTURE ENHANCEMENT

- In the future, there are several potential enhancements for a companion bot developed using Flutter, a popular framework for building cross-platform mobile applications.
- Natural Language Processing (NLP): Implementing advanced NLP capabilities will enable the companion bot to better understand and respond to user queries and conversations. This could involve integrating machine learning models for sentiment analysis, entity recognition, intent classification, and dialog management.
- Voice Recognition and Synthesis: Integrating speech recognition and synthesis technologies will allow users to interact with the companion bot using voice commands and receive spoken responses. Flutter provides plugins for working with speech recognition and synthesis, such as the speech_to_text and flutter_tts packages.

CHAPTER10

APPENDIX

10.1. SOURCE CODE

Main.dart

```
Import 'package:flutter/material.dart';
import 'CustomTabBar.dart';
import "Screens/Tasks.dart";
import 'Screens/Dashboard.dart';
import 'Screens/AccountPage.dart';
import 'Screens/DoctorPage.dart';
import 'Screens/SplashPage.dart';
import 'authentication.dart';


void main() =>runApp(MyApp());


 class MyApp extends StatelessWidget {
 final String uid;
 MyApp({this.uid=""});
 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
   return  MaterialApp(
     home: SplashPage(),
     routes:
     {
       "tasks":(context)=>Tasks(uid: uid),

       "main" :(context)=>MyHomePage(uid:uid),
       "doctor": (context)=>DoctorPage(),
       "account": (context)=>AccountPage(),
     },
     title: 'companion',
     theme: ThemeData(
       // This is the theme of your application.
       //
       // Try running your application with "flutter run". You'll see the
       // application has a blue toolbar. Then, without quitting the app, try
       // changing the primarySwatch below to Colors.green and then invoke
```

```
        // "hot reload" (press "r" in the console where you ran "flutter run",
        // or simply save your changes to "hot reload" in a Flutter IDE).
        // Notice that the counter didn't reset back to zero; the application
        // is not restarted.
        primarySwatch: Colors.blue,
      ),

    );
  }
}

class MyHomePage extends StatefulWidget {
  final String uid;
  MyHomePage({Key key, this.uid}) : super(key: key);

  // This widget is the home page of your application. It is stateful, meaning
  // that it has a State object (defined below) that contains fields that affect
  // how it looks.

  // This class is the configuration for the state. It holds the values (in this
  // case the title) provided by the parent (in this case the App widget) and
  // used by the build method of the State. Fields in a Widget subclass are
  // always marked "final".

  final String title="Your Companion";

  @override
  _MyHomePageState createState() => _MyHomePageState(uid:uid);
}

class _MyHomePageState extends State<MyHomePage> {
PageController pageController=PageController(initialPage: 0);
_MyHomePageState({this.uid});
String uid;

Map<String,Widget> pages;
@override
void initState(){
pages=<String,Widget>{
    "Tasks" : Tasks(uid:uid),
    "Dashboard" : Dashboard(),
  };
  super.initState();
}
```

```
@override
Widget build(BuildContext context) {

  return Scaffold(
    appBar: AppBar(
      // Here we take the value from the MyHomePage object that was created by
      // the App.build method, and use it to set our appbar title.
      backgroundColor:
      Colors.deepOrange,
      title: Text(
        widget.title,
        textAlign: TextAlign.left,
        style: TextStyle(fontStyle: FontStyle.normal,fontWeight: FontWeight.bold)
        ),

      bottom: CustomTabBar(pageController: pageController, pageNames: pages.keys.toList(),),
    ),
    drawer: Drawer(
      elevation: 16.0,
      child: ListView(
      padding: EdgeInsets.zero,
      children: [
        DrawerHeader(
          margin:EdgeInsets.all(10.0) ,
          child: Text.rich(TextSpan(text: "JAYESH")),
        ),
      ListTile(
      leading: Image.asset("doctor"),
      title: Text('Get Professional Help',
        style: TextStyle(fontWeight: FontWeight.bold,fontStyle: FontStyle.normal),),
      onTap: () {
      Navigator.pushNamed(context, "doctor");
      Navigator.pop(context);
      },
      ),
      ListTile(
      leading: Image.asset("Account"),
      title: Text('Your Account',
      style: TextStyle(fontWeight: FontWeight.bold,fontStyle: FontStyle.normal),),
      onTap: () {
      Navigator.pushNamed(context, "account");
      Navigator.pop(context);
      },),
      ListTile(
              title: new Text('Logout', textAlign: TextAlign.right,
              style: TextStyle(fontWeight: FontWeight.bold,fontStyle: FontStyle.normal),),
              trailing: new Icon(Icons.exit_to_app),
              onTap: () async {
                await signOutWithGoogle();
                Navigator.of(context).pushReplacementNamed('/');
              },
            ),
        ]
        )
      ),
      body:
        PageView(
        controller: pageController,
        children: pages.values.toList(),
        )

    );

  }
}
```

Accountpage.dart

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

class AccountPage extends StatefulWidget{
  AccountPage();

@override
AccountPageState createState()=> AccountPageState();
}

class AccountPageState extends State<AccountPage>{

  @override
  Widget build(BuildContext context){
    return Container();
  }
}
```

Dashboard.dart

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

  class Dashboard extends StatefulWidget {
  Dashboard();

  @override
  DashboardState createState() => DashboardState();
}

  class DashboardState extends State<Dashboard> {
  int progress;
  bool show = true;
  String userId;
  FirebaseUser user;
  FirebaseAuth auth = FirebaseAuth.instance;
  DocumentSnapshot document;

  DocumentReference ansRef;

  void getUser() async {
    var user1 = await auth.currentUser();
    user = user1;
    userId = user.uid;
  }

  @override
  void initState() {
    getUser();
    ansRef = Firestore.instance.collection("answers").document(userId);
    super.initState();
    getDocument();
    updateProgress();
  }

  void getDocument() async {
    document = await ansRef.get();
  }

  void updateProgress() {
    progress = document['q1'] - document['q1old'];
```

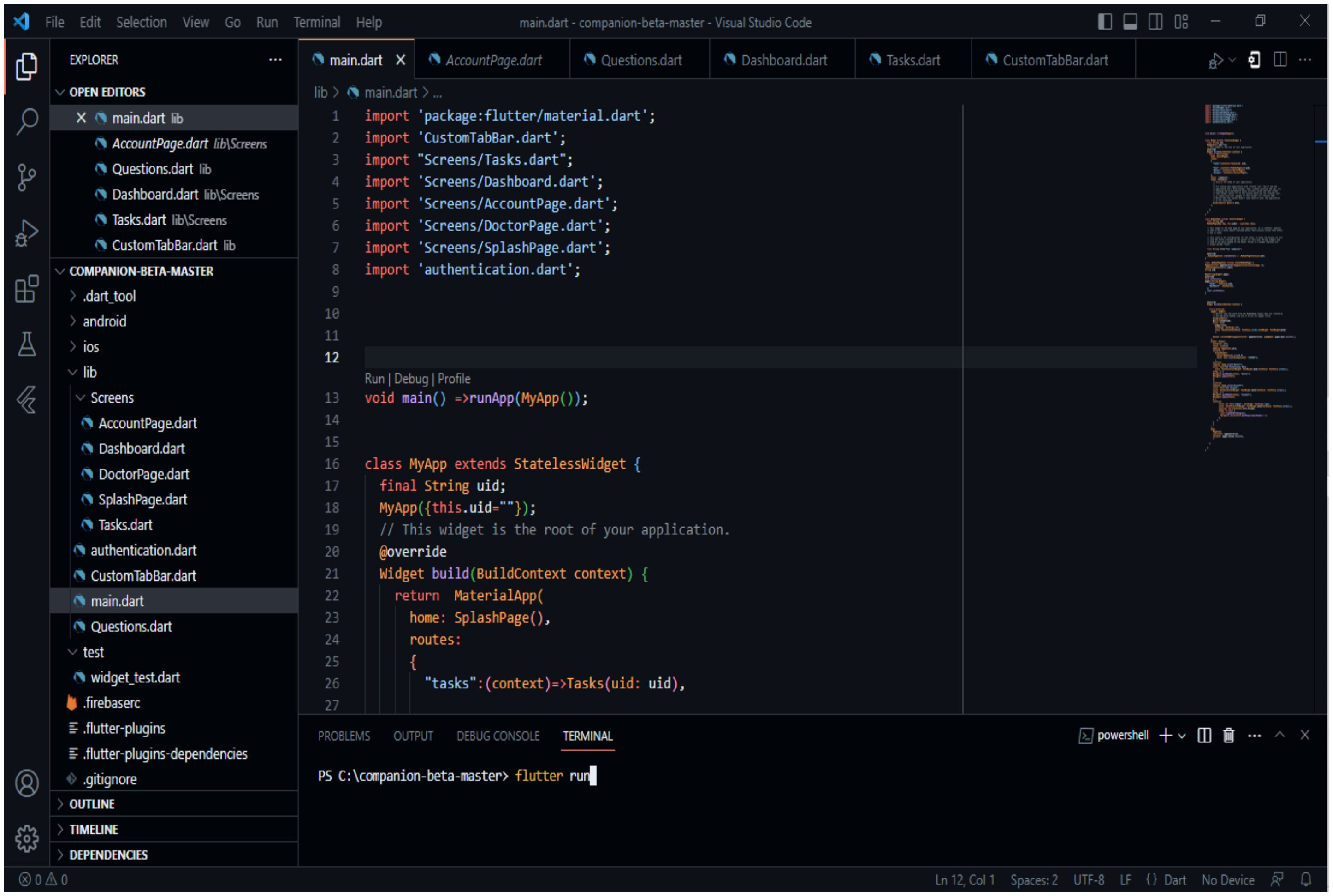
```
}

@override
Widget build(BuildContext context) {
  return Column(children: <Widget>[
    Flexible(
      child: (!show)
          ? Container()
          : Column(
              children: <Widget>[
                (progress > 0)
                    ? ListTile(
                        leading: Image.asset("community"),
                        title: Text("Congratulations!",
                            style: TextStyle(fontWeight: FontWeight.bold)),
                        subtitle: Text(
                            "You met $progress more people today. Keep it going."),
                      )
                    : ListTile(
                        leading: Image.asset("encourage"),
                        title: Text("You are yet to make progress"),
                      ),
                ButtonTheme(
                  // makes buttons use the appropriate styles for cards
                  child: ButtonBar(
                    children: <Widget>[
                      TextButton(
                        child: const Text('Share'),
                        onPressed: () {},
                      ),
                      TextButton(
                        child: const Text('Dismiss'),
                        onPressed: () {
                          setState(() {
                            show = false;
                          });
                        },
                      ),
                    ],
                  ),
                )
              ],
            ),
    )
    ]);
  }
}
```

Tasks.dart

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:companion_beta/Questions.dart';

class Tasks extends StatefulWidget {
  Tasks({this.uid = ""});
  final String uid;
  @override
  TasksState createState() => TasksState(uid);
}

class TasksState extends State<Tasks> {
  var now = DateTime.now();
  bool timeUp = true;
  TasksState(this.userId);

  Map<String, dynamic> identifiers = Map.from({
    "answered": false,
    "value": false,
    "notDone1": true,
    "notDone2": true,
    "defNumber": -1,
    "hour": null
  });
  FirebaseAuth auth = FirebaseAuth.instance;

  String userId;
  DocumentReference ansRef;

  Future<void> configure() async {
    final FirebaseApp app = await FirebaseApp.configure(
        name: 'companionbeta',
        options: const FirebaseOptions(
            googleAppID: '1:933028870179:android:ede06df604d42475',
            apiKey: 'AIzaSyCdSfHQoXvBPNyBd-NtXO4RB3MVB-gUE6A ',
          projectID: 'companionbeta'));
  final Firestore firestore = Firestore(app: app);
  await firestore.settings(timestampsInSnapshotsEnabled: true);
}

bool isDepressed(DocumentSnapshot document) {
  document.data.putIfAbsent("q1", document["defNumber"]);
  if (document["q1"] < 3) return true;
  return false;
}

@override
void initState() {
  configure();
  ansRef = Firestore.instance.collection("answers").document(userId);
  addData();
  super.initState();
}

void updateTime(DocumentSnapshot document) async {
  if (document["hour"] == null)
    await document.reference.updateData({"hour": now.hour});
  assert(now.hour != null);
  if (document["hour"] < 24) {
    setState(() {
      timeUp = true;
    });
    await document.reference.updateData({"hour": document["hour"] + 24});
  }
}

void addData() async {
  ansRef.get().then((DocumentSnapshot ds) {
    ds.data.addAll(identifiers);
    updateTime(ds);
    isDepressed(ds);
  });
}

@override
Widget build(BuildContext context) {
  return Material(
      child: StreamBuilder(
          stream: Firestore.instance.collection("answers").snapshots(),
          builder: (context, snapshot) {
            if (!snapshot.hasData) return Text("Loading..");
            DocumentSnapshot doc = snapshot.data.documents[0];
            Map<String, bool> check = Map.from({
              "answered": (doc["answered"] == null) ? false : true,
              "notDone1": (doc["notDone1"] == null) ? false : true,
              "notDone2": (doc["notDone2"] == null) ? false : true,
              "q1": (doc["q1"] == null) ? false : true,
            });
            return builder(context, doc, check);
          }));
}

Widget builder(BuildContext context, DocumentSnapshot document,
    Map<String, bool> check) {
  return Column(mainAxisSize: MainAxisSize.min, children: <Widget>[
    (check["answered"] && timeUp && !document["answered"])
        ? askQuestion(document)
        : Container(),
    Flexible(
        child: Card(
      child: ListView(
        children: <Widget>[
          (check["notDone1"] &&
                  check["q1"] &&
                  isDepressed(document) &&
                  document["notDone1"])
              ? ListTile(
                  contentPadding: EdgeInsets.all(10.0),
                  leading: Icon(Icons.people),
                  title: Text("Meet more people"),
                  subtitle: Text("uno dos tres"),
                  trailing: Checkbox(
                    value: false,
                    onChanged: (value) {
                      setState(() {
                        if (value)
                          document.reference.updateData({"notDone1": false});
                      });
                    },
                  ),
                )
              : Container(),
          (check["notDone2"] &&
                check["q1"] &&
                isDepressed(document) &&
                document["notDone2"])
              ? ListTile(
                contentPadding: EdgeInsets.symmetric(horizontal: 10.0),
                leading: Icon(Icons.library_music),
                title: Text("Listen to tunes"),
                subtitle: Text("uno dos tres"),
                trailing: Checkbox(
                  value: document["value"],
                  onChanged: (value) {
                    setState(() {
                      if (value)
                        document.reference.updateData({"notDone2": false});
                    });
                  },
                ),
              )
              : Container(),
            (check["q1"] &&
                check["notDone1"] &&
                isDepressed(document) &&
                !document["notDone1"])
              ? Card(
                child: Expanded(
                child: Text("Yay! We are happy today"),
              ))
              : Container()
          ],
        ),
        elevation: 2.0,
        margin: EdgeInsets.symmetric(horizontal: 12.0, vertical: 6.0),
      ))
    ]);
  }

  Widget askQuestion(DocumentSnapshot document) {
    return Card(
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: <Widget>[
          const ListTile(
            leading: Icon(Icons.question_answer),
            title: Text('Ready to answer a few questions?'),
          subtitle: Text('It will only take a minute'),
        ),
        ButtonTheme(
          // make buttons use the appropriate styles for cards
          child: ButtonBar(
            children: <Widget>[
              TextButton(
                child: const Text('Yes!'),
                onPressed: () {
                  Navigator.of(context).push(MaterialPageRoute(
                      builder: (context) => Questions(document)));
                },
              ),
              TextButton(
                child: const Text('Not now'),
                onPressed: () {
                  setState(() {
                    document.reference.updateData({"answered": true});
                  });
                },
              ),
            ],
          ),
        ),
      ],
    ),
    elevation: 2.0,
    margin: EdgeInsets.all(12.0),
  );
 }
}
```

Custombar.dart

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

class CustomTabBar extends AnimatedWidget implements PreferredSizeWidget {
  CustomTabBar({this.pageController, this.pageNames})
      : super(listenable: pageController);

  final PageController pageController;
  final List<String> pageNames;
  @override
  final Size preferredSize = Size(0.0, 70.0);

  @override
  Widget build(BuildContext context) {
    return Container(
      height: 40.0,
      margin: EdgeInsets.all(10.0),
      padding: EdgeInsets.symmetric(horizontal: 40.0),
      decoration: BoxDecoration(
        color: Colors.black.withOpacity(0.3),
        borderRadius: BorderRadius.circular(20.0),
      ),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: List.generate(pageNames.length, (int index) {
          return InkWell(
              child: Text(pageNames[index],
                  style: TextStyle(
                    fontSize: 17.0,
                    color: Colors.white
                        .withOpacity(index == pageController.page ? 1.0 : 0.6),
                    fontWeight: FontWeight.bold,
                  )),
              onTap: () {
                pageController.animateToPage(index,
                    curve: Curves.easeOut,
                    duration: const Duration(milliseconds: 300));
              });
        }).toList(),
      ),
    );
  }
}
```

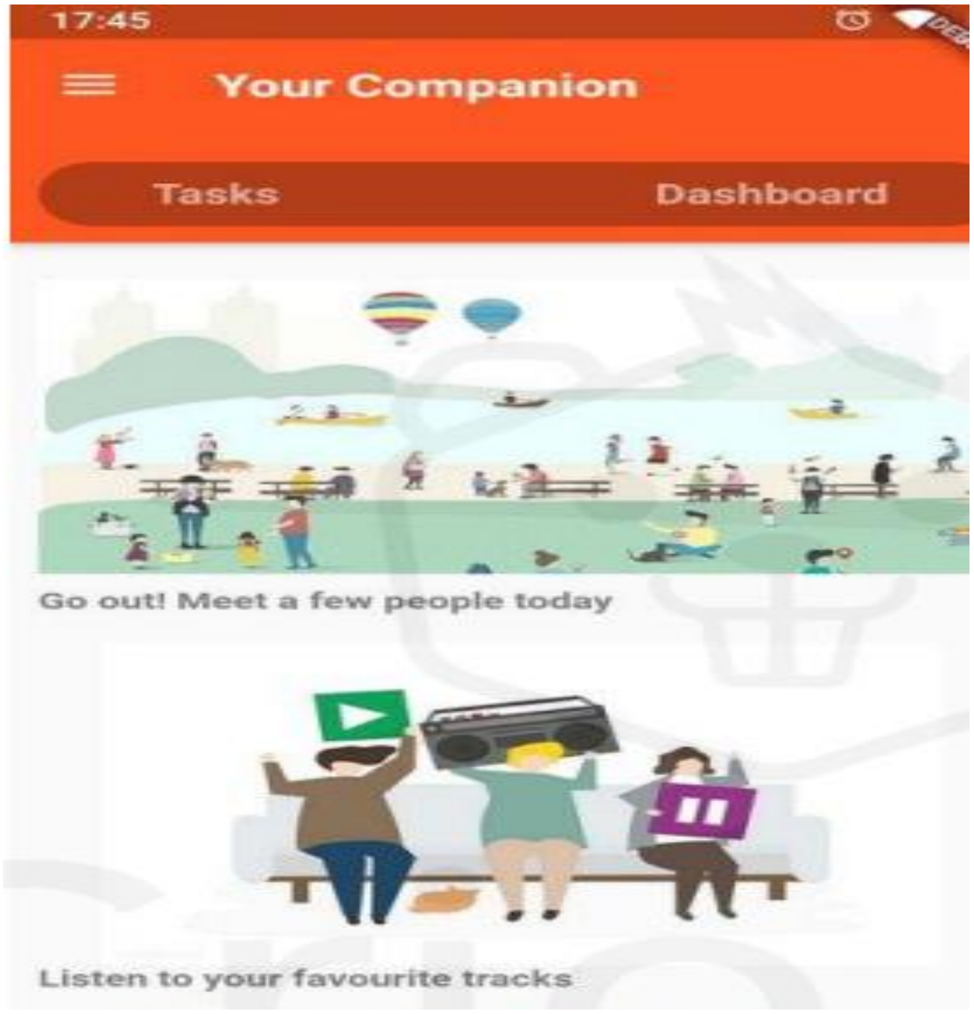


```
import 'package:flutter/material.dart';
import 'CustomTabBar.dart';
import "Screens/Tasks.dart";
import 'Screens/Dashboard.dart';
import 'Screens/AccountPage.dart';
import 'Screens/DoctorPage.dart';
import 'Screens/SplashPage.dart';
import 'authentication.dart';




Run | Debug | Profile
void main() =>runApp(MyApp());


class MyApp extends StatelessWidget {
  final String uid;
  MyApp({this.uid=""});
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return  MaterialApp(
      home: SplashPage(),
      routes:
      {
        "tasks":(context)=>Tasks(uid: uid),
```

```
PS C:\companion-beta-master> flutter run
```

Figure.10.2 Executing Stage

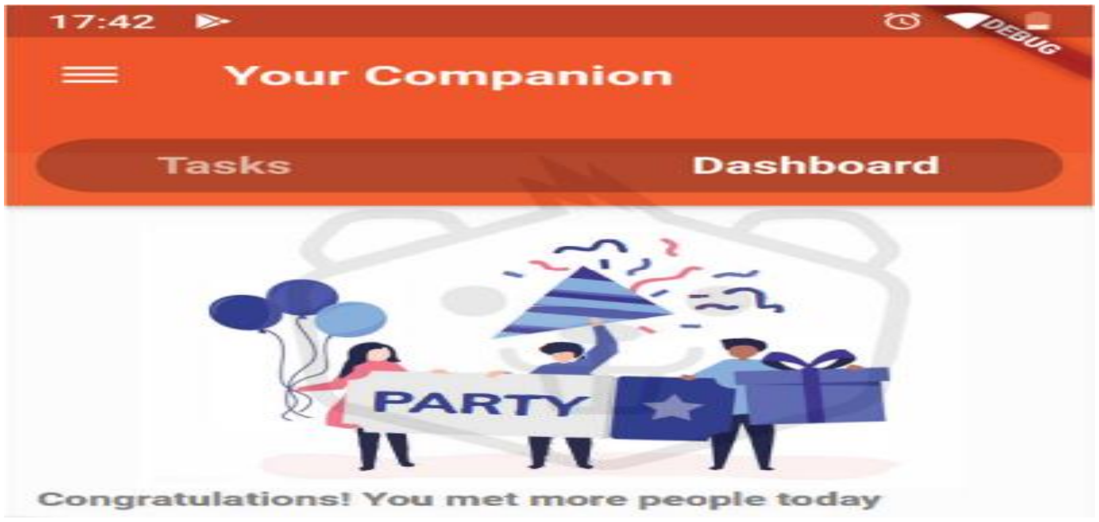
10.2 SCREENSHOTS



Go out! Meet a few people today

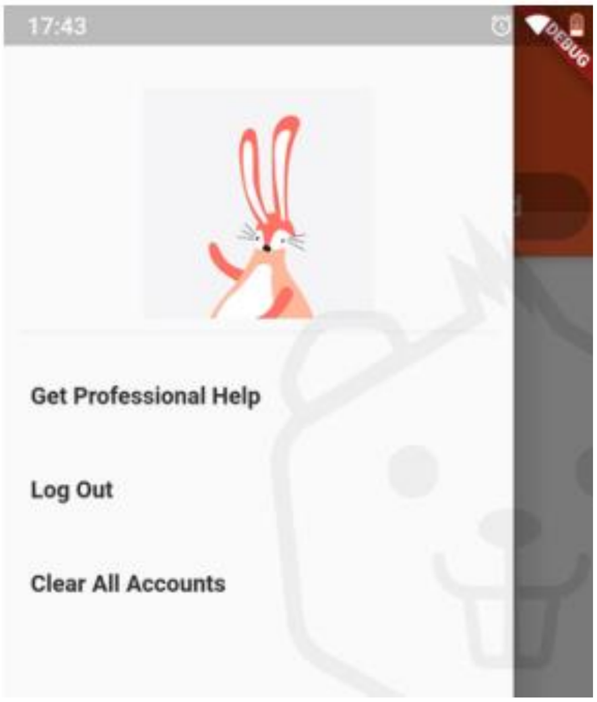
Listen to your favourite tracks

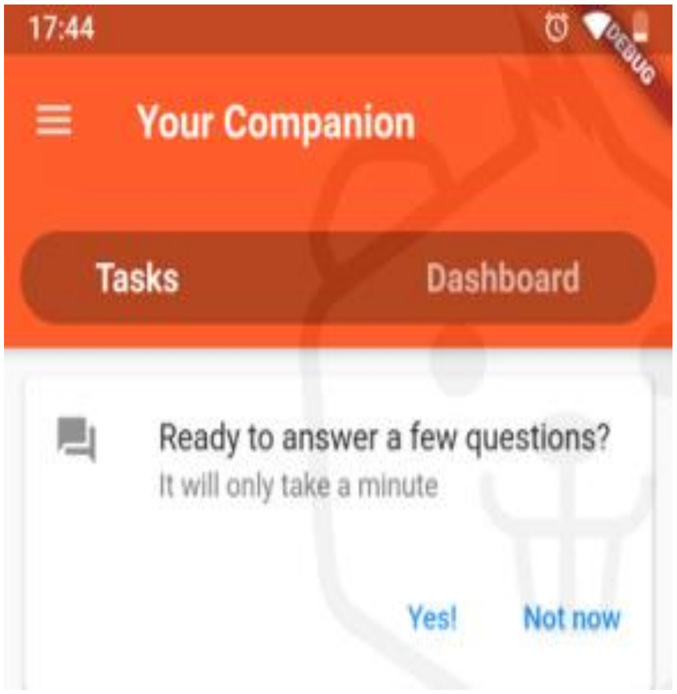
Fig 10.4 Front page



Congratulations! You met more people today

Fig 10.5 Dashboard





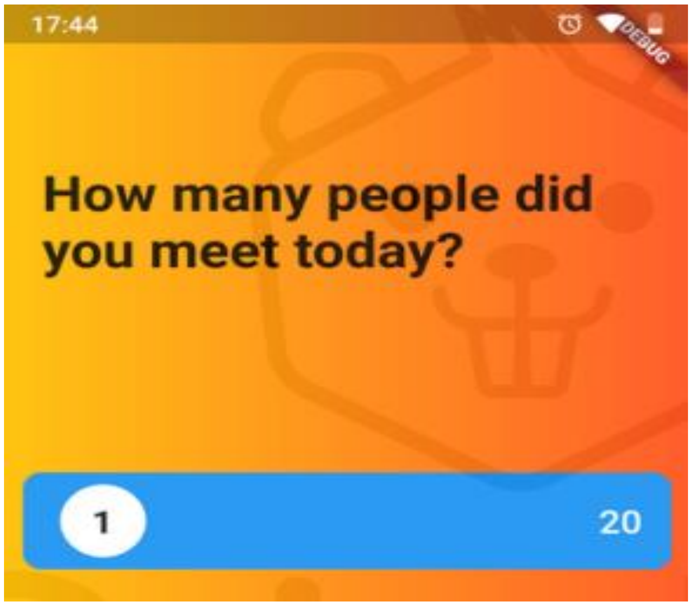


Fig 10.6 Ask Questions

CHAPTER 11

REFERENCES

[1] Xiong, Caihua; Huang, Yongan; Xiong, Youlun (2008). Intelligent Robotics and Applications: First International Conference, ICIRA 2008 Wuhan, China, October 15-17, 2008

[2] Sojka, Petr; Horak, Aleš; Kopecek, Ivan (2008). Text, Speech and Dialogue: 11th International Conference

[3] Wang, Kelly. "'iPal' robot companion for China's lonely children". phys.org. Retrieved 10 December 2021.

[4] Lamers, Maarten H.; Verbeek, Fons J. (2011). Human-Robot Personal Relationships: Third International Conference, HRPR 2010, Leiden.

[5] "NICOBO, a robot born out of empathy with consumers and sense of mission". *Panasonic*. Panasonic. Retrieved 2 May 2023

[6] A multimodal face detection system for elderly companion robot Yong Tao; Hongxing Wei; Tianmiao Wang; Diansheng Chen

[7] My Chatbot Companion - a Study of Human-Chatbot Relationships Marita Skjuve a, Asbjørn Følstad a, Knut Inge Fostervold b, Petter Bae Brandtzaeg

[8] Evaluation of a companion robot based on field tests with single older adults intheir homes Katalin Zsiga, MDa,b, András Tóth

[9] .A Review on the Use of Mobile Service Robots in Elderly Care Pouyan Asgharian Adina M. Panchea

[10] Assistive robotic systems in nursing care: a scoping review Christoph Ohneberg Nicole Stöbich Angelika Warmbein Inge Eberl