The Egyptian E-Learning University (EELU)

Faculty of Computer & Information Technology

" Yuna Pet Application "

By:

No.	Name	ID
1	Malk Haitham Mostafa	20-01137
2	Shahd Abdel Moez	20-00088
3	Aya Yasser Helmy	20-01114
4	Sara Rabie Ahmed	20-01186
5	Doaa Anter Kotb	20-00532
6	Alaa Khaled Rashed	20-00536
7	Soha Nagy Khedr	20-00552

Under supervision of:

DR. Rodaina Abdelsalam	ENG. Nader Emad

EELU-Assiut “2024”

Acknowledgment

All Praises and Thanks Be to Allah for the success and his help for us to finish the project this year. During this year we had to take advice and support from a boom of respectable people and we had to thank them for their time and effort and we would like to express our gratitude and thanks to them and we would like to show this to our

supervisors:

Dr. Rodina

(Dean of Computer Science and Information Technology

Egyptian E-Learning University)

Eng. Nader Emad

(Teaching Assistant in Computer and Information Technology faculty At Egyptian E Learning University)

For giving us a good guideline for the Project throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in doing this Project. Many people, especially our classmates and team members itself, have made valuable comment suggestions on this proposal which gave us an inspiration to improve our project. We thank all the people for their help directly and indirectly to complete our project.

Abstract

Yuna Application is a revolutionary pet application designed to streamline pet care and health management through the integration of artificial intelligence (AI) technology. The application offers a myriad of features aimed at simplifying the lives of pet owners while ensuring the well-being of their furry companions. One of Yuna Application's standout features is its image recognition capability, which allows users to capture photos of their pets and receive instant identification of the species and breed. Leveraging AI algorithms, the application provides comprehensive information about the identified animal, including its characteristics and care requirements. Moreover, Yuna Application serves as a proactive health monitoring tool by facilitating the detection of skin conditions and gastrointestinal ailments in pets. Through the simple process of capturing and uploading images of affected skin or feces, the AI-powered system swiftly identifies the presence of diseases and offers detailed insights into their nature and symptoms. Furthermore, the application boasts a convenient reminder feature, aiding users in managing their pet's feeding schedules, vaccination appointments, and medical reservations. This integrated reminder system ensures that pet owners never miss crucial events related to their pet's health and well-being. In summary, Yuna Application emerges as a comprehensive solution for pet owners, combining advanced AI technology with user-friendly functionalities to enhance pet care and health management.

By empowering users with instant identification, proactive health monitoring, and timely reminders, Yuna Application revolutionizes the way we care for our beloved animal companions.

CONTENTS

Chapter 1 (Introduction)..... 6

Introduction..... 7

Problem Definition 8

Project Objectives..... 8

Motivation 10

System Architecture..... 11

Related Work..... 12:13

Chapter 2 (Methodologies and Techniques) 14

Adobi XD 15:16

Front End..... 17:34

Web Scrapping 34:37

Back End..... 37:43

Machine Learning..... 44:68

Chapter 3 (Implementation and Coding) 69

Software Techniques..... 70

Analysing System 70:76

Application UI..... 77:92

Coding 93:200

Chapter 4 (Summary, Conclusion and Future Work)..... 201

Summary..... 202

Conclusion..... 202

Challenges..... 203

Future Work..... 203

Chapter 5(References)..... 204

References..... 205:207

Chapter 1: - Introduction

1.1 Introduction:

As the world increasingly embraces digital solutions for various needs, the realm of pet care is no exception. With the advent of innovative technologies, pet owners now have access to unprecedented levels of assistance and guidance in nurturing their beloved companions. In today's fast-paced world, ensuring the well-being of our pets requires more than just basic care routines. Yuna Application recognizes this need and strives to empower pet owners with advanced tools and insights to make informed decisions about their pet's health and happiness. Whether you're a seasoned pet parent or embarking on your first pet ownership journey, Yuna Application is here to simplify and enhance every aspect of caring for your furry friend. With Yuna Application, pet owners gain access to a wealth of features that streamline pet care tasks and provide big support. From capturing a simple snapshot of your pet to leveraging artificial intelligence for breed identification and health analysis, Yuna Application equips users with the knowledge and resources needed to ensure optimal pet wellness. Whether you're navigating the complexities of pet ownership for the first time or seeking to enhance your existing pet care routine, Yuna Application is your trusted companion every step of the way. Join us as we embark on a journey to revolutionize pet care with Yuna Application, where innovation meets compassion in the service of our beloved animal companions.

1.2 Problem definition:

1-Lack of accessible and detailed information about various pet species and their specific needs, leading to uninformed pet care practices.

2-Identifying skin diseases in dogs is challenging, potentially leading to delayed treatment. Pet owners may struggle to recognize signs of disease in their pets' stool."

3-Busy schedules can result in overlooked pet care responsibilities.

1.3 Project objectives:

- Our main goal is to develop an innovative pet care application, such as Yuan, that addresses the identified challenges in pet care.

❖ The main proposed functions of the system are as follows:

1-Providing detailed and accessible information:

- Develop a database containing information about types of pets, including their needs, characteristics, and care requirements.
- Design user-friendly interfaces that allow pet owners to access and navigate this information easily.
- the information provided is comprehensive, accurate, up-to-date and meets the diverse needs of different pet owners.

2-Facilitating early detection of skin diseases and fecal diseases:

- Integrating advanced image recognition technology into the app to identify and analyze skin conditions in dogs based on uploaded images.
- Implementing algorithms that can accurately detect common skin conditions, such as dermatitis or fungal infections, to provide early warnings to pet owners.
- Provide detailed information about specific skin diseases and fecal diseases, including symptoms, treatment options, and preventive measures.

3- Streamline Pet Care Management:

- Develop features that allow pet owners to create and manage customized schedules for pet care activities, such as feeding, grooming, and veterinary appointments.
- Implement reminder notifications to alert users of upcoming tasks and appointments, ensuring that important responsibilities are not overlooked.
- Integrate functionality that enables users to track their pets' health records, including vaccination histories, medication schedules and vet visits.

1.4 Motivation:

the motivation behind developing this pet care application stems from a deep-seated passion for improving the well-being of pets and their owners. As pet lovers ourselves, we understand the joys and challenges that come with pet ownership. However, we also recognize the gaps and inefficiencies present in traditional pet care practices.

The lack of easily accessible and detailed information about various pet species and their specific needs often leaves pet owners feeling overwhelmed and uncertain about how to provide the best care for their furry companions. Additionally, the difficulty in identifying skin diseases in dogs and cats, and the potential consequence of delayed treatment highlights the pressing need for innovative solutions in pet healthcare.

Furthermore, the busy lifestyles of modern pet owners can result in overlooked pet care responsibilities, leading to compromised health and happiness for pets. This underscores the importance of developing tools that streamline pet care management and provide support to pet owners, allowing them to fulfil their responsibilities without sacrificing their hectic schedules.

By addressing these challenges through the development of our pet care application, we aim to empower pet owners with the knowledge, resources, and tools needed to ensure the optimal health and happiness of their pets. Our ultimate motivation is to make pet care more accessible, efficient, and enjoyable for pet owners worldwide, fostering stronger bonds and happier lives between pets and their human companions.

1.5 System Architecture:

The application process begins with the user or pet owner logging in. If it's their first time using the app, they can create an account. Upon logging in, they are directed to the home page of the application, where they can add their pet by either capturing a photo of the pet using the camera or selecting a photo from their phone's gallery. Artificial intelligence then classifies the pet in the image, identifying its species and breed, and provides information about it. The pet is then added to its own page called the profile, where its name, photo, breed, weight, age, and other information are displayed. Users can also detect early signs of illness or symptoms on their pet's skin or in its feces by capturing a photo of the skin or feces, or by adding a photo from their phone's gallery. The AI recognizes the illness and provides information about it, including symptoms and preventive measures. Additionally, users can visit the reminder page to add tasks, such as reminders for feeding times, vaccination schedules, and more. Notifications are then sent to remind them of these events.

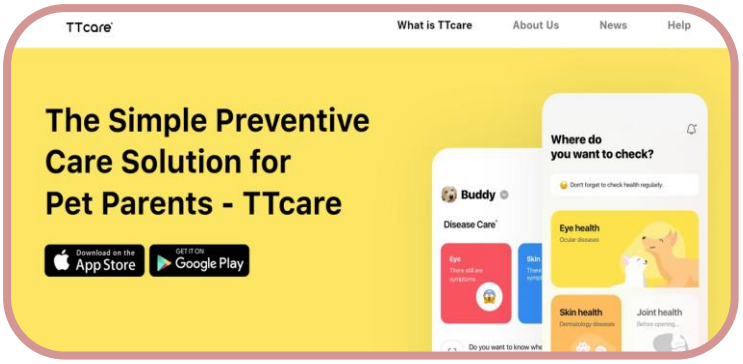
1.6 Related Work:

After searching in many local and international applications, we found:

1. TTCARE APP:

- **HOW TTCARE WORKS:**

  Simply take pictures of your pet's eyes and skin to check their health with the help of our AI.



- **ADVANTAGES**
  - SIMPLE

    Just download the app and you are then

    ready to check the health of your dog(s) or cat(s).

  - Smart

    AI will detect abnormal symptoms early

    with an accuracy level of over 90%.

  - Innovative

    World's first mobile application that

    identifies symptoms of diseases.

  - Recognized

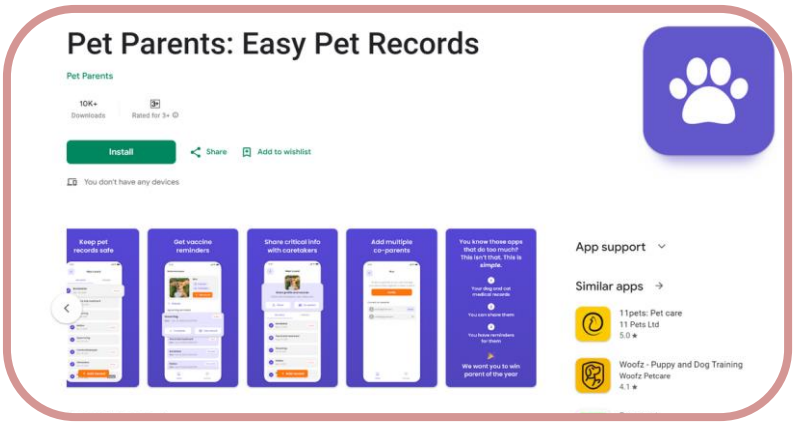
Patent registered medical device software approved by the Korean Ministry of Agriculture, Food and Rural Affairs .

2. PET PARENTS

It is an application that treats the pet owner like a father and provides important reminders regarding the pet.

- **ADVANTAGES**

- ADDS VACCINE REMINDERS.
- AND GIVES YOU EASY WAYS TO SHARE CRITICAL INFO WITH CARETAKERS



Chapter 2 :
Methodologies and Techniques

2.1 Adobe XD

Adobe XD is a powerful design and prototyping tool used primarily by UI and UX designers to create interactive and high-fidelity prototypes for websites, mobile applications, and other digital products.

What is adobe XD used for?

What's important to remember is that Adobe XD addresses the two main problems Photoshop and other graphics applications couldn't:

1. nteraction design isn't static A designer cannot communicate a fluid and dynamic design using pixels alone.

2. A modern design process involves more than a polished finished article.

Wireframing, iteration, and behavior are all part of the collaborative decision making with UI and UX design.

Adobe XD is ideal for vector-based UI design, wireframing, interactive design, prototyping, and hi-fidelity web/app design, for solo designers or whole teams.

What are the features of Adobe XD ?

Adobe XD offers a range of features tailored to streamline the user experience (UX) and user interface (UI) design process. Here are some key features:

- **Artboards:** Designers can work on multiple screens or layouts within a single document, facilitating the creation of designs for various devices and screen sizes.

- **Responsive Resizing:** Designs can be easily adapted to different screen sizes and orientations, ensuring consistency across various devices.

- **Repeat Grids:** Simplifies the process of creating repeating elements such as lists or grids by allowing designers to duplicate and maintain consistency across multiple instances.

- **Prototyping and Animation:** Enables designers to create interactive prototypes with animations and transitions to simulate the user experience and demonstrate functionality.

- **Interoperability:** Seamless integration with other Adobe Creative Cloud applications such as Photoshop and Illustrator, allowing for easy import of assets and designs .

- **Content-Aware Layout:** Automatically adjusts content layout based on the changes made to the design, ensuring consistency and efficiency .

- **Voice Design:** Supports designing voice-enabled experiences by providing tools to visualize and prototype voice interactions .

- **Components:** Create reusable design elements that can be easily updated across multiple artboards, ensuring consistency and efficiency in the design.

These features collectively empower designers to create, prototype, and share engaging and interactive user experiences efficiently.

- To get hold of it visit : https://www.adobe.com/products/xd or if you have Creative Cloud app installed and you have a valid subscription you can download it directly from there .

2.2 Front end

Front-end development primarily focuses on user experience . Using the related coding and design techniques, you as front-end developers build the elements of an application that are directly accessed by end-users with a goal of rendering the entire interface elegant, easy to use, fast, and secure, fostering user engagement and interaction.

As part of creating an engaging user interface, front end app development often focus on specific design elements such as text colors and styles, images, graphs and tables, buttons, and overall color schemes. These elements play a crucial role in enhancing the visual appeal and user-friendliness of the application.

Front end app development encompasses various interactive elements like sliders, pop-up forms, and custom interactive maps. An essential part of a front end application are navigational menus, which guide users through the application, enhancing their overall experience and interaction with the website or application. The creation of intuitive and user-friendly navigational menus is a key skill for front-end developers.
Front-end developers require a specific set of skills to effectively create user interfaces.

Front end applications, also known as the “client side” of an application, are what users see and interact with. They differ from the backend, which is like the hidden machinery behind the scenes. In this context, APIs act as translators, ensuring seamless communication between the visually rich front-end and the complex backend.

What is the technique we used in front end ?

We used Flutter in our front end. Flutter is an open-source framework developed by Google for building cross-platform applications from a single codebase. It uses the Dart programming language and allows developers to create natively compiled applications for mobile (iOS and Android), web, and desktop from a single codebase .

Why we choose flutter ?

Flutter offers several advantages for cross-platform app development:

1. Cross-Platform Development: Flutter enables developers to write a single codebase that can be used to create apps for multiple platforms, including iOS, Android, web, and desktop. This reduces development time and costs significantly.

2. Fast Development: Flutter's "hot reload" feature allows developers to see changes in real-time without restarting the app, speeding up the development process.
3. High Performance: Flutter apps are compiled directly into native machine code, which ensures high performance and smooth animations .

4. Customizable Widgets: Flutter provides a rich set of customizable widgets that conform to specific design guidelines, making it easy to create visually attractive and functional UIs .

5. Unified UI and Business Logic: The same UI and business logic can be applied across all platforms, ensuring consistency and reducing the complexity of maintaining separate codebases .

6. Cost Efficiency: Using Flutter for cross-platform development can be more cost-effective compared to developing and maintaining separate native apps for each platform.

7. Community and Support: Flutter is backed by Google and has a growing community of developers, ensuring continuous improvement and a wealth of resources for learning and problem-solving .

Flutter architectural overview :

Flutter is a cross-platform UI toolkit that is designed to allow code reuse across operating systems such as iOS and Android, while also allowing applications to interface directly with underlying platform services. The goal is to enable developers to deliver high-performance apps that feel natural on different platforms, embracing differences where they exist while sharing as much code as possible.

This overview is divided into a number of sections:

1. **The layer model:** The pieces from which Flutter is constructed.

2. **Reactive user interfaces:** A core concept for Flutter user interface development.

3. **An introduction to widgets :** The fundamental building blocks of Flutter user interfaces.

4. **The rendering process:** How Flutter turns UI code into pixels.

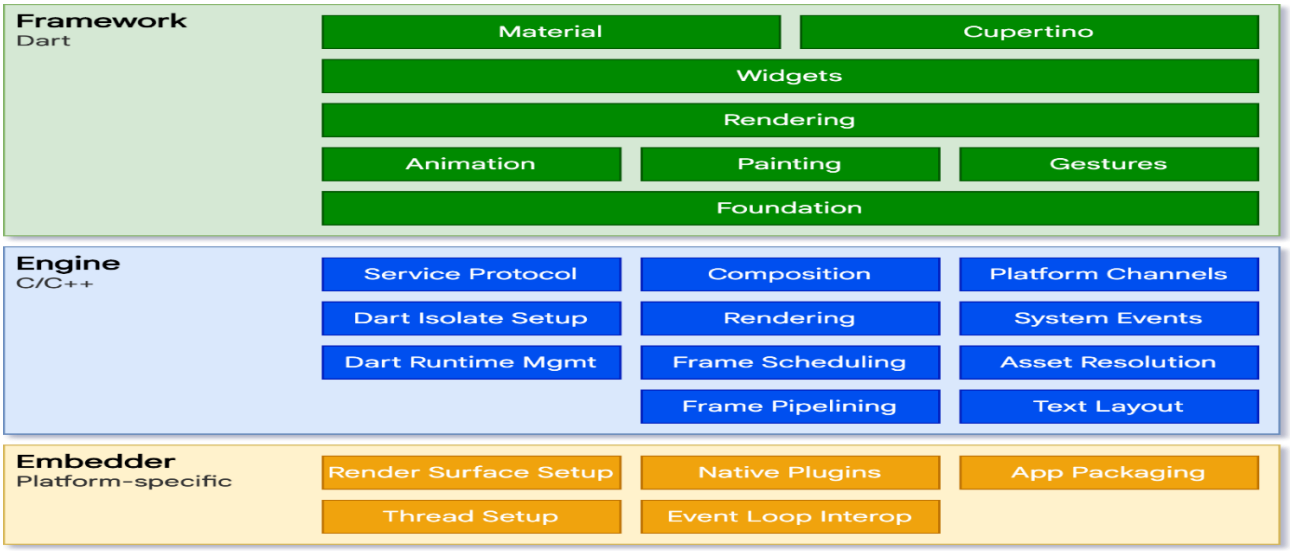
5. **An overview of the platform embedders**: The code that lets mobile and desktop OSes execute Flutter apps.

6. **Integrating Flutter with other code**: Information about different techniques available to Flutter apps.

7. **Support for the web:** Concluding remarks about the characteristics of Flutter in a browser environment.

1. Architectural layers :

Flutter is designed as an extensible, layered system. It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable.



To the underlying operating system, Flutter applications are packaged in the same way as any other native application. A platform-specific embedder provides an entrypoint; coordinates with the underlying operating system for access to services like rendering surfaces, accessibility, and input; and manages the message event loop. The embedder is written in a language that is appropriate for the platform: currently Java and C++ for Android, Objective-C/Objective-C++ for iOS and macOS, and C++ for Windows and Linux. Using the embedder, Flutter code can be integrated into an existing application as a module, or the code might be the entire content of the application. Flutter includes a number of embedders for common target platforms, but other embedders also exist.

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

The engine is exposed to the Flutter framework through dart:ui, which wraps the underlying C++ code in Dart classes. This library exposes the lowest-level primitives, such as classes for driving input, graphics, and text rendering subsystems.
Typically, developers interact with Flutter through the Flutter framework, which provides a modern, reactive framework written in the Dart language. It includes a rich set of platform, layout, and foundational libraries, composed of a series of layers.

Working from the bottom to the top, we have:

- Basic foundational classes, and building block services such as animation, painting, and gestures that offer commonly used abstractions over the underlying foundation.
- The rendering layer provides an abstraction for dealing with layout. With this layer, you can build a tree of renderable objects. You can manipulate these objects dynamically, with the tree automatically updating the layout to reflect your changes.

- The widgets layer is a composition abstraction. Each render object in the rendering layer has a corresponding class in the widgets layer. In addition, the widgets layer allows you to define combinations of classes that you can reuse. This is the layer at which the reactive programming model is introduced.
- The Material and Cupertino libraries offer comprehensive sets of controls that use the widget layer's composition primitives to implement the Material or iOS design languages.

The Flutter framework is relatively small; many higher-level features that developers might use are implemented as packages, including platform plugins like camera and webview, as well as platform-agnostic features like characters, http, and animations that build upon the core Dart and Flutter libraries. Some of these packages come from the broader ecosystem, covering services like in-app payments, Apple authentication, and animations.
The rest of this overview broadly navigates down the layers, starting with the reactive paradigm of UI development. Then, we describe how widgets are composed together and converted into objects that can be rendered as part of an application. We describe how Flutter interoperates with other code at a platform level, before giving a brief summary of how Flutter's web support differs from other targets.

2-Reactive user interfaces

On the surface, Flutter is a reactive, pseudo-declarative UI framework, in which the developer provides a mapping from application state to interface state, and the framework takes on the task of updating the interface at runtime when the application state changes. This model is inspired by work that came from Facebook for their own React framework, which includes a rethinking of many traditional design principles.

In most traditional UI frameworks, the user interface’s initial state is described once and then separately updated by user code at runtime, in response to events. One challenge of this approach is that, as the application grows in complexity, the developer needs to be aware of how state changes cascade throughout the entire UI. here are many places where the state can be changed: the color box, the hue slider, the radio buttons. As the user interacts with the UI, changes must be reflected in every other place. Worse, unless care is taken, a minor change to one part of the user interface can cause ripple effects to seemingly unrelated pieces of code.

One solution to this is an approach like MVC, where you push data changes to the model via the controller, and then the model pushes the new state to the view via the controller. However, this also is problematic, since creating and updating UI elements are two separate steps that can easily get out of sync.
Flutter, along with other reactive frameworks, takes an alternative approach to this problem, by explicitly decoupling the user interface from its underlying state. With React-style APIs, you only create the UI description, and the framework takes care of using that one configuration to both create and/or update the user interface as appropriate.
In Flutter, widgets (akin to components in React) are represented by immutable classes that are used to configure a tree of objects. These widgets are used to manage a separate tree of objects for layout, which is then used to manage a separate tree of objects for compositing. Flutter is, at its core, a series of mechanisms for efficiently walking the modified parts of trees, converting trees of objects into lower-level trees of objects, and propagating changes across these trees.
A widget declares its user interface by overriding the build() method, which is a function that converts state to UI .

3- Widgets :

As mentioned, Flutter emphasizes widgets as a unit of composition. Widgets are the building blocks of a Flutter app's user interface, and each widget is an immutable declaration of part of the user interface.

Widgets form a hierarchy based on composition. Each widget nests inside its parent and can receive context from the parent. This structure carries all the way up to the root widget (the container that hosts the Flutter app, typically MaterialApp or CupertinoApp), as this trivial example shows:

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
 void main() => runApp(const MyApp());
class MyApp extends StatelessWidget {
const MyApp({super.key});

@override Widget build(BuildContext context) {
return MaterialApp(
home: Scaffold(
appBar: AppBar(
title: const Text('My Home Page'),
),
 body: Center(
child: Builder(
 builder: (context) {
 return Column(
children: [
const Text('Hello World'),
const SizedBox(height: 20),
ElevatedButton(
 onPressed: () {
print('Click!');
},
child: const Text('A button'),
),
                                    ],
    );
          },
      ),
            ),
      ),
     );
 }
}
```

In the preceding code, all instantiated classes are widgets.
Apps update their user interface in response to events (such as a user interaction) by telling the framework to replace a widget in the hierarchy with another widget. The framework then compares the new and old widgets, and efficiently updates the user interface.
Flutter has its own implementations of each UI control, rather than deferring to those provided by the system: for example, there is a pure Dart implementation of both the iOS Toggle control and the one for the Android equivalent.

This approach provides several benefits:

- Provides for unlimited extensibility. A developer who wants a variant of the Switch control can create one in any arbitrary way, and is not limited to the extension points provided by the OS.

- Avoids a significant performance bottleneck by allowing Flutter to composite the entire scene at once, without transitioning back and forth between Flutter code and platform code.

- Decouples the application behavior from any operating system dependencies. The application looks and feels the same on all versions of the OS, even if the OS changed the implementations of its controls.

Building widgets

As mentioned earlier, you determine the visual representation of a widget by overriding the build() function to return a new element tree. This tree represents the widget's part of the user interface in more concrete terms. For example, a toolbar widget might have a build function that returns a horizontal layout of some text and various buttons. As needed, the framework recursively asks each widget to build until the tree is entirely described by concrete renderable objects. The framework then stitches together the renderable objects into a renderable object tree.

A widget's build function should be free of side effects. Whenever the function is asked to build, the widget should return a new tree of widgets[1], regardless of what the widget previously returned. The framework does the heavy lifting work to determine which build methods need to be called based on the render object tree (described in more detail later). More information about this process can be found in the Inside Flutter topic.

On each rendered frame, Flutter can recreate just the parts of the UI where the state has changed by calling that widget's build() method. Therefore it is important that build methods should return quickly, and heavy computational work should be done in some asynchronous manner and then stored as part of the state to be used by a build method.

While relatively naïve in approach, this automated comparison is quite effective, enabling high-performance, interactive apps. And, the design of the build function simplifies your code by focusing on declaring what a widget is made of, rather than the complexities of updating the user interface from one state to another.

Widget state

The framework introduces two major classes of widget: *stateful* and *stateless* widgets.
Many widgets have no mutable state: they don't have any properties that change over time (for example, an icon or a label). These widgets subclass StatelessWidget.

However, if the unique characteristics of a widget need to change based on user interaction or other factors, that widget is *stateful*. For example, if a widget has a counter that increments whenever the user taps a button, then the value of the counter is the state for that widget. When that value changes, the widget needs to be rebuilt to update its part of the UI. These widgets subclass StatefulWidget, and (because the widget itself is immutable) they store mutable state in a separate class that subclasses State. StatefulWidgets don't have a build method; instead, their user interface is built through their State object.

Whenever you mutate a State object (for example, by incrementing the counter), you must call setState() to signal the framework to update the user interface by calling the State's build method again.

Having separate state and widget objects lets other widgets treat both stateless and stateful widgets in exactly the same way, without being concerned about losing state. Instead of needing to hold on to a child to preserve its state, the parent can create a new instance of the child at any time without losing the child's persistent state. The framework does all the work of finding and reusing existing state objects when appropriate.

4-Rendering processing:

Rendering and layout :

This section describes the rendering pipeline, which is the series of steps that Flutter takes to convert a hierarchy of widgets into the actual pixels painted onto a screen.

Flutter's rendering model :

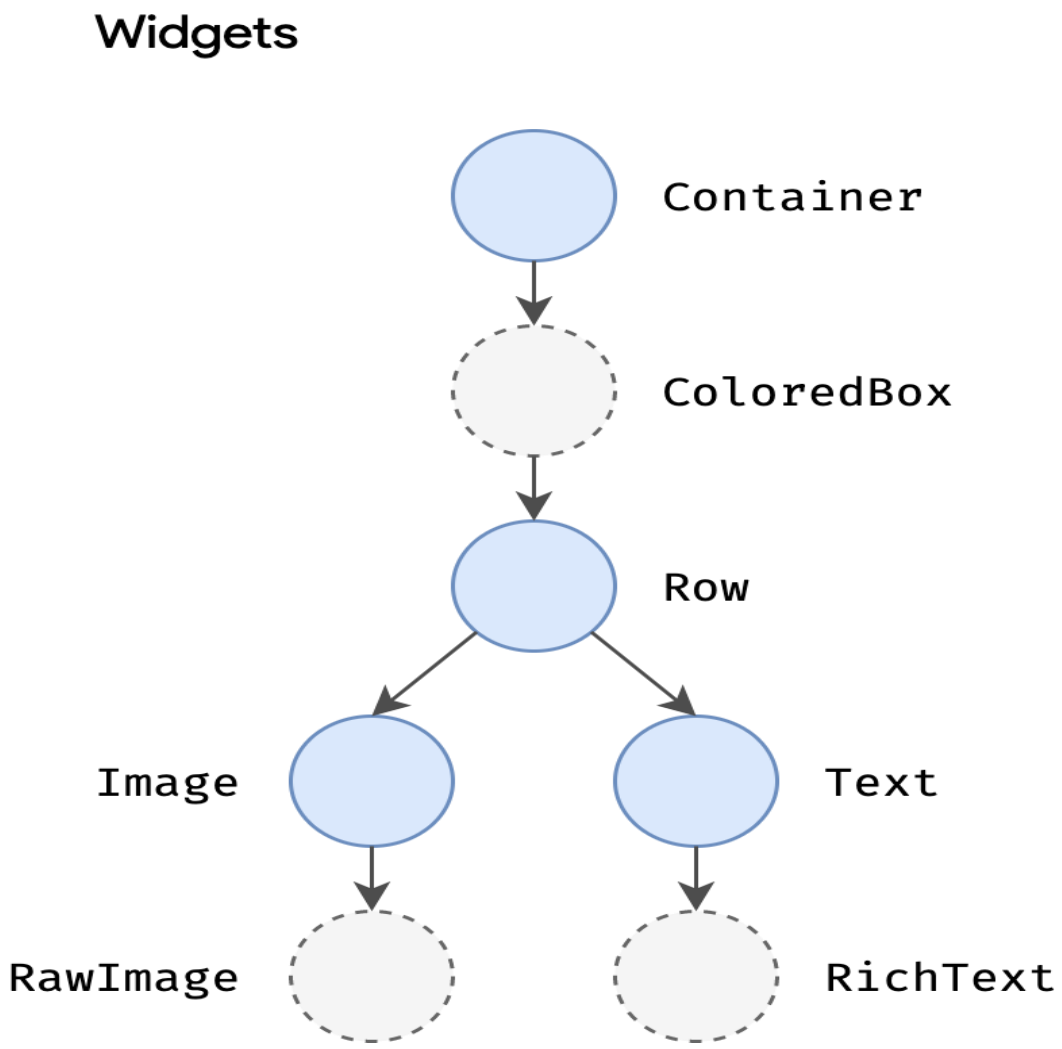
You might be wondering: if Flutter is a cross-platform framework, then how can it offer comparable performance to single-platform frameworks?

It's useful to start by thinking about how traditional Android apps work. When drawing, you first call the Java code of the Android framework. The Android system libraries provide the components responsible for drawing themselves to a Canvas object, which Android can then render with Skia, a graphics

engine written in C/C++ that calls the CPU or GPU to complete the drawing on the device.

Cross-platform frameworks *typically* work by creating an abstraction layer over the underlying native Android and iOS UI libraries, attempting to smooth out the inconsistencies of each platform representation. App code is often written in an interpreted language like JavaScript, which must in turn interact with the Java-based Android or Objective-C-based iOS system libraries to display UI. All this adds overhead that can be significant, particularly where there is a lot of interaction between the UI and the app logic.

By contrast, Flutter minimizes those abstractions, bypassing the system UI widget libraries in favor of its own widget set. The Dart code that paints Flutter's visuals is compiled into native code, which uses Skia (or, in the future, Impeller) for rendering. Flutter also embeds its own copy of Skia as part of the engine, allowing the developer to upgrade their app to stay updated with the latest performance improvements even if the phone hasn't been updated with a new Android version. The same is true for Flutter on other native platforms, such as Windows or macOS.



This explains why, when you examine the tree through a debug tool such as the Flutter inspector, part of the Flutter/Dart DevTools, you

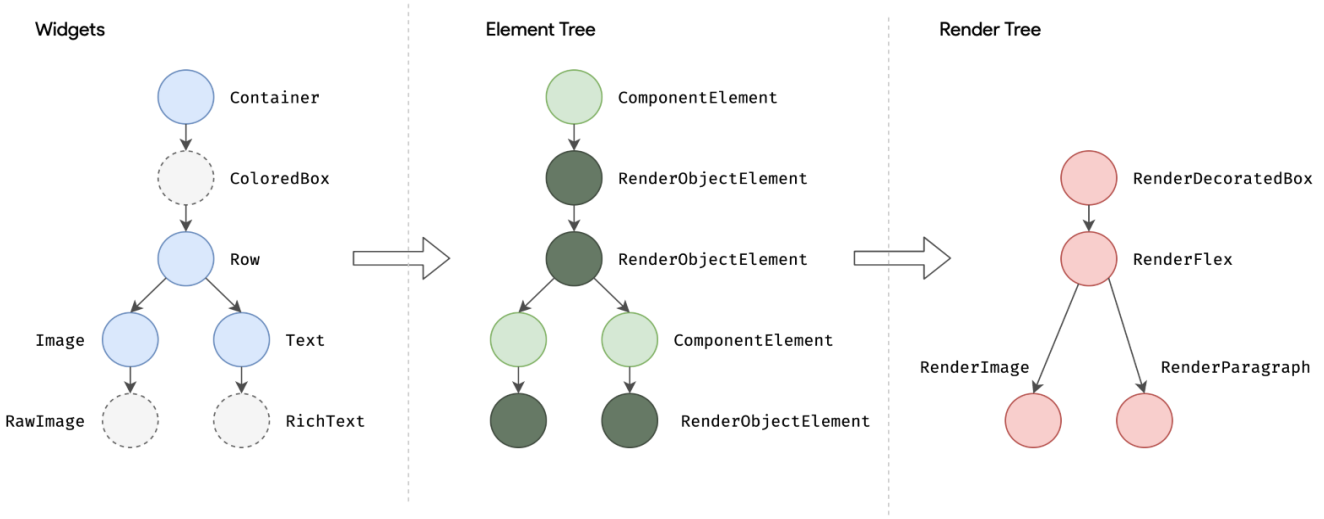
might see a structure that is considerably deeper than what is in your original code.
During the build phase, Flutter translates the widgets expressed in code into a corresponding **element tree**, with one element for every widget. Each element represents a specific instance of a widget in a given location of the tree hierarchy. There are two basic types of elements:

- ComponentElement, a host for other elements.
- RenderObjectElement, an element that participates in the layout or paint phases

It would be a rare application that drew only a single widget. An important part of any UI framework is therefore the ability to efficiently lay out a hierarchy of widgets, determining the size and position of each element before they are rendered on the screen. The base class for every node in the render tree is RenderObject, which defines an abstract model for layout and painting. This is extremely general: it does not commit to a fixed number of dimensions or even a Cartesian coordinate system (demonstrated by this example of a polar coordinate system). Each RenderObject knows its parent, but knows little about its children other than how to *visit* them and their constraints.

This provides RenderObject with sufficient abstraction to be able to handle a variety of use cases.

During the build phase, Flutter creates or updates an object that inherits from RenderObject for each RenderObjectElement in the element tree. RenderObjects are primitives: RenderParagraph renders text, RenderImage renders an image, and RenderTransform applies a transformation before painting its child.



Most Flutter widgets are rendered by an object that inherits from the RenderBox subclass, which represents a RenderObject of fixed size in a 2D Cartesian space. RenderBox provides the basis of a *box constraint model*, establishing a minimum and maximum width and height for each widget to be rendered.

5-Platform embedding :

As we've seen, rather than being translated into the equivalent OS widgets, Flutter user interfaces are built, laid out, composited, and painted by Flutter itself. The mechanism for obtaining the texture and participating in the app lifecycle of the underlying operating system inevitably varies depending on the unique concerns of that platform. The engine is platform-agnostic, presenting a stable ABI (Application Binary Interface) that provides a *platform embedder* with a way to set up and use Flutter.

The platform embedder is the native OS application that hosts all Flutter content, and acts as the glue between the host operating system and Flutter. When you start a Flutter app, the embedder provides the entrypoint, initializes the Flutter engine, obtains threads for UI and rastering, and creates

a texture that Flutter can write to. The embedder is also responsible for the app lifecycle, including input gestures (such as mouse, keyboard, touch), window sizing, thread management, and platform messages. Flutter includes platform embedders for Android, iOS, Windows, macOS, and Linux; you can also create a custom platform embedder, as in this worked example that supports remoting Flutter sessions through a VNC-style framebuffer or this worked example for Raspberry Pi.

Each platform has its own set of APIs and constraints. Some brief platform-specific notes:

- On iOS and macOS, Flutter is loaded into the embedder as a UIViewController or NSViewController, respectively. The platform embedder creates a FlutterEngine, which serves as a host to the Dart VM and your Flutter runtime, and a FlutterViewController, which attaches to the FlutterEngine to pass UIKit or Cocoa input events into Flutter and to display frames rendered by the FlutterEngine using Metal or OpenGL.
- On Android, Flutter is, by default, loaded into the embedder as an Activity. The view is controlled by a FlutterView, which renders Flutter content either as a view or a texture, depending on the composition and z-ordering requirements of the Flutter content.

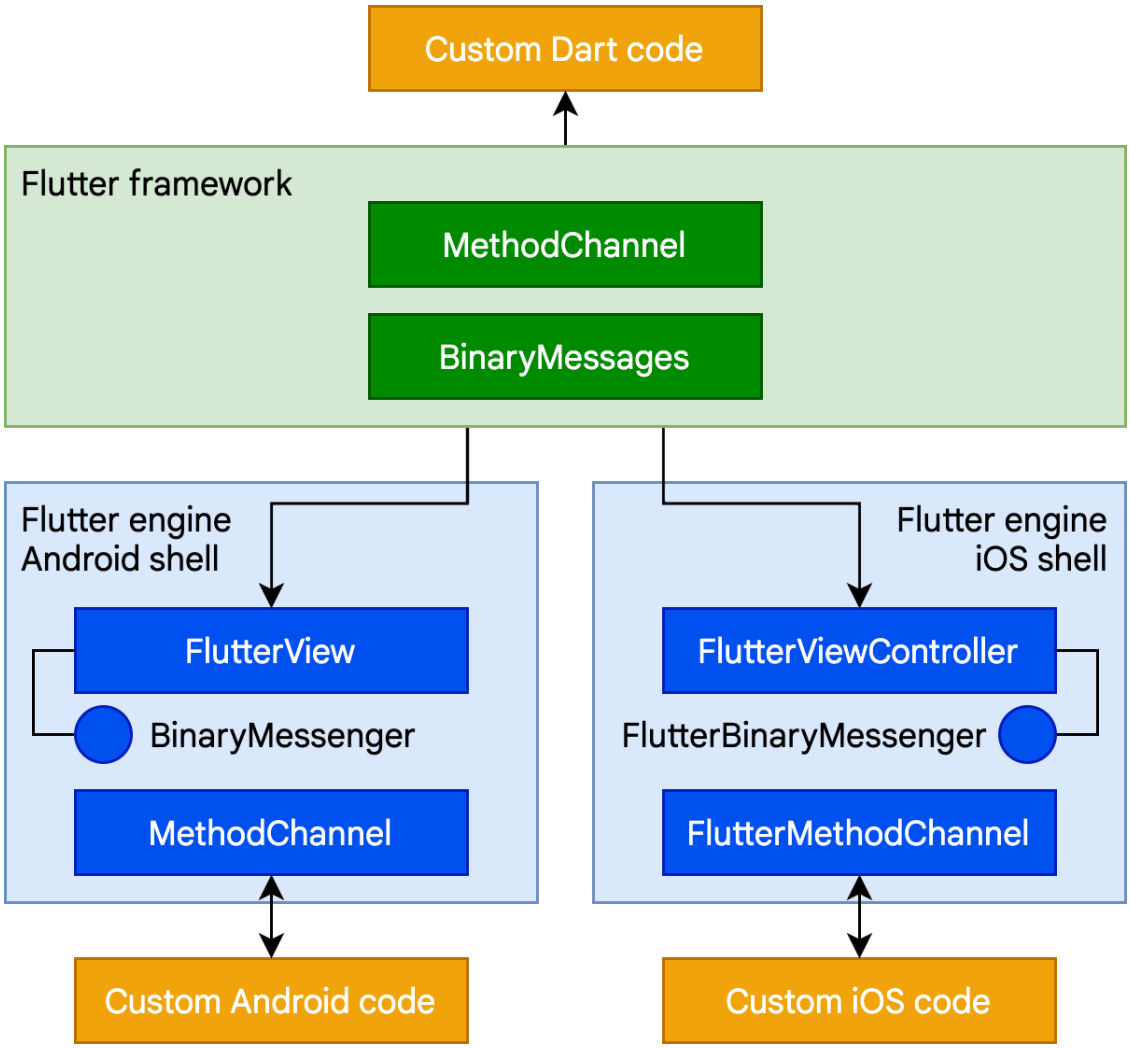
- On Windows, Flutter is hosted in a traditional Win32 app, and content is rendered using ANGLE, a library that translates OpenGL API calls to the DirectX 11 equivalents.

6-Integrating with other code:

Flutter provides a variety of interoperability mechanisms, whether you're accessing code or APIs written in a language like Kotlin or Swift, calling a native C-based API, embedding native controls in a Flutter app, or embedding Flutter in an existing application.
Platform channels

For mobile and desktop apps, Flutter allows you to call into custom code through a *platform channel*, which is a mechanism for communicating between your Dart code and the platform-specific code of your host app. By creating a common channel (encapsulating a name and a codec), you can send and receive messages between Dart and a platform component written

in a language like Kotlin or Swift. Data is serialized from a Dart type like Map into a standard format, and then deserialized into an equivalent representation in Kotlin (such as HashMap) or Swift (such as Dictionary).

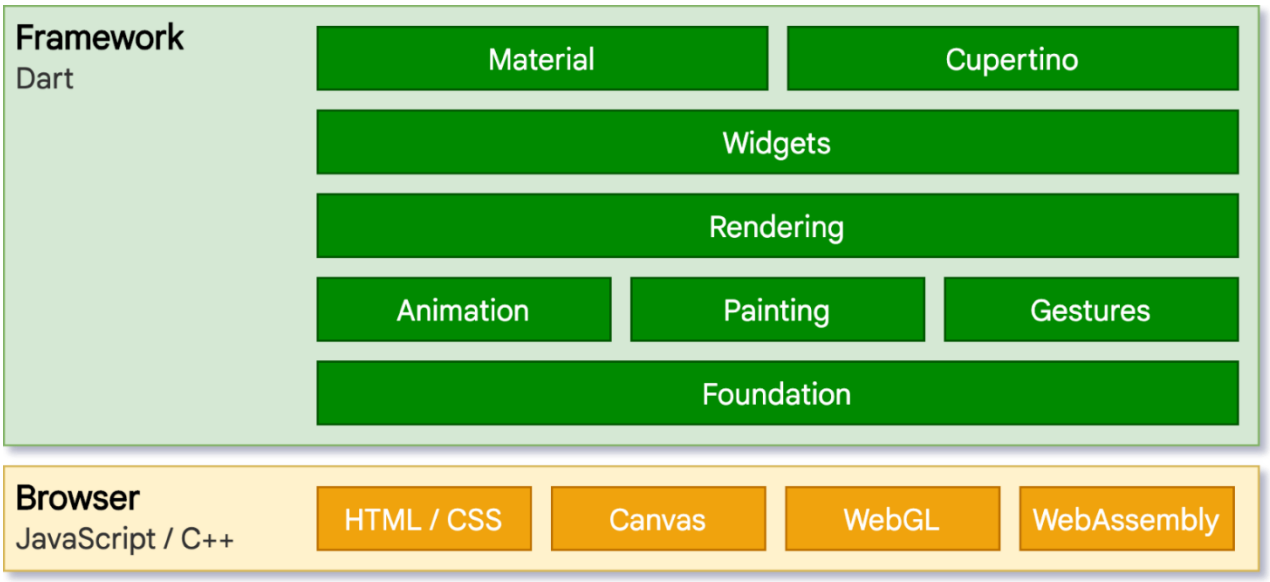


7-Flutter web support :

While the general architectural concepts apply to all platforms that Flutter supports, there are some unique characteristics of Flutter's web support that are worthy of comment.

Dart has been compiling to JavaScript for as long as the language has existed, with a toolchain optimized for both development and production purposes. Many important apps compile from Dart to JavaScript and run in production today, including the advertiser tooling for Google Ads. Because the Flutter framework is written in Dart, compiling it to JavaScript was relatively straightforward.

However, the Flutter engine, written in C++, is designed to interface with the underlying operating system rather than a web browser. A different approach is therefore required. On the web, Flutter provides a reimplementation of the engine on top of standard browser APIs. We currently have two options for rendering Flutter content on the web: HTML and WebGL. In HTML mode, Flutter uses HTML, CSS, Canvas, and SVG. To render to WebGL, Flutter uses a version of Skia compiled to Web Assembly called CanvasKit. While HTML mode offers the best code size characteristics, CanvasKit provides the fastest path to the browser's graphics stack, and offers somewhat higher graphical fidelity with the native mobile targets. The web version of the architectural layer diagram is as follows:



During development time, Flutter web uses dartdevc, a compiler that supports incremental compilation and therefore allows hot restart (although not currently hot reload) for apps. Conversely, when you are ready to create a production app for the web, dart2js, Dart's highly-optimized production JavaScript compiler is used, packaging the Flutter core and framework along with your application into a minified source file that can be deployed to any web server. Code can be offered in a single file or split into multiple files through deferred imports .

2.3 Web Scrapping

Web scraping is an automated method used to extract large amounts of data from websites. This process involves fetching the web pages, parsing the HTML content, and extracting the desired information, which is then often structured into a more usable format, such as a spreadsheet or database.
Web scraping is widely used for various purposes, including:

Data Collection: Gathering data from different websites for market research, price comparison, and competitive analysis.
Content Aggregation: Compiling information from multiple sources into a single platform, such as news aggregators or job listing sites.

Research: Collecting data for academic, scientific, or business research.
Monitoring: Keeping track of changes on websites, such as price fluctuations or stock availability.

The main techniques used in web scraping include parsing HTML, using APIs provided by websites, and employing web scraping tools and libraries such as BeautifulSoup, Scrapy, and Selenium.

Why to use web scrapping ?

Web scraping offers numerous advantages for businesses, researchers, and developers by providing efficient and effective ways to gather large amounts of data from websites. Here are some key reasons to use web scraping:

1. **Cost-Effective**: Web scraping services can be a cost-effective method to collect data, eliminating the need for manual data entry or expensive data collection processes .

2. **Time-Saving**: Automating data extraction significantly reduces the time required to gather and process data, enabling faster decision-making and project completion .

3. **Accurate and Up-to-Date Data**: Web scraping ensures access to accurate and real-time data, which is crucial for tasks such as market analysis, competitive monitoring, and trend tracking .

4. **Customization and Flexibility**: The process can be tailored to meet specific data needs, allowing users to extract exactly the information they require from targeted sources .

5. **Scalability**: Web scraping tools can handle large volumes of data, making it possible to scale up data collection efforts as needed without significant additional costs.

6. **Improved Decision Making:** Access to comprehensive and up-to-date data supports better decision-making in areas such as pricing strategies, market trends, and customer insights .

7. **Automation:** Web scraping automates the data collection process, which reduces the likelihood of human error and ensures consistent data quality .

8. **Competitive Monitoring:** Businesses can use web scraping to monitor competitors' activities, such as pricing, product offerings, and marketing strategies, to stay ahead in the market .

Overall, web scraping provides a powerful tool for accessing and leveraging web data to gain insights, drive business strategies, and streamline data collection processes.

BeautifulSoup

BeautifulSoup is a popular Python library used for web scraping purposes to extract data from HTML and XML documents. BeautifulSoup is designed to parse HTML and XML documents and extract data from them. It creates a parse tree for parsed pages, which can be used to extract data easily.

BeautifulSoup is a powerful tool for web scraping that simplifies the process of extracting and manipulating data from web pages.

HTTP Requests

HTTP (Hypertext Transfer Protocol) is the foundation of data communication on the World Wide Web. HTTP requests are used to communicate between clients (like web browsers) and servers. An HTTP request is a message sent by a client to a server, initiating a

transaction. It includes a request method, URI, HTTP version, headers, and sometimes a body.
HTTP requests are fundamental to web development and API integration, enabling efficient and standardized communication across the web.

2.4 Back end

Back-end Development: refers to the server-side development. It focuses on databases, scripting, website architecture. It contains behind-the-scene activities that occur when performing any action on a website. It can be an account login or making a purchase from an online store. Code written by back-end developers helps browsers to communicate with database information.

What are reasons for using the backend ?
Using a backend for web and mobile applications provides numerous benefits:

1. **Data Management and Storage:** A backend is essential for handling data storage, ensuring data integrity, and managing database interactions. This allows for efficient storage and retrieval of user data, enhancing the application's functionality .

2. **Security:** Backend systems can implement robust security measures to protect sensitive data, such as encryption, authentication, and authorization mechanisms. This is crucial for safeguarding user information and maintaining privacy .

3. **Scalability:** Backends enable applications to scale effectively, handling increased loads and user requests without compromising performance. Technologies like Node.js facilitate horizontal and vertical scaling, making it easier to manage growing application demands .

4. **Performance Optimization:** A backend can offload resource-intensive tasks from the frontend, such as complex computations, file processing, and data analytics, improving overall performance and user experience .

5. **API Integration:** Backends provide a way to integrate third-party APIs and services, enabling applications to leverage external functionalities and data sources seamlessly .

6. **Business Logic Implementation:** The backend is responsible for implementing core business logic, processing user input, and making decisions based on predefined rules. This ensures that the application behaves correctly and meets business requirements.

7. **Hosting and Deployment:** The backend provides the infrastructure necessary to host and deploy web applications, making them accessible to users. This includes managing servers, databases, and other resources required for the application to run smoothly.

Backend Types and Their Use Cases

Backend development involves various types of backend systems, each suited to different use cases. Here are the primary types and their use cases:

1. Relational Databases (RDBMS)

- Use Cases: Ideal for applications requiring complex queries and transactions, such as financial systems, e-commerce platforms, and enterprise applications.
- Examples: MySQL, PostgreSQL, Oracle Database.
- Benefits: Strong ACID (Atomicity, Consistency, Isolation, Durability) properties ensure data integrity and reliability .

2. NoSQL Databases

- Use Cases: Suitable for applications with large volumes of unstructured data, real-time web apps, and big data analytics.
- Examples: MongoDB, Cassandra, Couchbase.
- Benefits: Scalability, flexibility in data models (document, key-value, graph, column-family)

3. Backend as a Service (BaaS)

- Use Cases: Best for mobile and web applications requiring rapid development without managing backend infrastructure. Examples include social media apps, small-scale e-commerce sites, and MVPs (Minimum Viable Products).
- Examples: Firebase, AWS Amplify, Backendless.
- Benefits: Accelerated development, reduced need for backend management, built-in features like authentication and real-time databases .

4. Microservices

- Use Cases: Complex, large-scale applications where services need to be independently deployable and scalable. Examples include streaming services, large-scale e-commerce platforms, and financial systems.
- Examples: Docker, Kubernetes, Spring Boot.
- Benefits: Scalability, resilience, flexibility in development and deployment .

5. Backend for Frontend (BFF)

- Use Cases: Applications with multiple front-end interfaces needing different backend interactions. Examples include complex web and mobile applications with varied user experiences.
- Examples: Custom backend services tailored for specific frontends.
- Benefits: Optimized performance and development efficiency for diverse client needs .

6. GraphQL Backend

- Use Cases: Applications requiring efficient data fetching, flexible queries, and reduced over-fetching of data. Examples include social networks, data-driven dashboards, and developer APIs.
  - Examples: Apollo Server, Hasura.
  - Benefits: Efficient data fetching, flexible query capabilities, and improved performance for client applications .

What is Google Firebase?

Google Firebase is a comprehensive platform for mobile and web application development. It provides a suite of tools and services designed to help developers build, improve, and grow their apps efficiently. Key features of Firebase include:

1. Backend as a Service (BaaS): Firebase offers cloud-based backend services such as real-time databases, cloud storage, and authentication, eliminating the need for developers to manage server infrastructure.

2. Real-time Database: Allows for data to be synced in real-time across all clients, enabling features like live chat and real-time notifications.

3. Authentication: Simplifies user authentication using a variety of methods, including email/password, phone, and social media logins.
4. Analytics: Provides in-depth analytics to help developers understand user behavior and measure app performance.

5. Cloud Functions: Enables developers to run server-side code in response to events triggered by Firebase features or HTTPS requests.

6. Hosting: Firebase Hosting provides fast and secure hosting for web apps, static and dynamic content, and microservices.

7. Machine Learning: Includes ML Kit, which brings machine learning capabilities to apps with ready-to-use APIs for common tasks like text recognition and image labeling.

Firebase is designed to streamline the app development process, allowing developers to focus more on building great user experiences and less on managing infrastructure.

Firebase History:

Firebase history is quite fascinating, and as many startups has a lot of ups and downs. It originated from Envolve, a startup company established in 2011 by Andrew Lee and James Tamplin. The company offered an API for developers to facilitate online chat integration for websites.

The founders of Envolve discovered that their chat service was being utilized for relaying non-chat messages. Developers were relying on the platform for real-time application data syncing. Lee and Tamplin decided to differentiate the real-time architecture from the chat system, a move which led to Firebase being founded in 2011. The Firebase backend as a service platform was publicly launched in April 2012. The first Firebase product launched was the Realtime Database. It is an API for application data synchronization across Android, web, and iOS devices. Application developers can rely on the platform for creating collaborative real-time applications.

The company accumulated seed funding of more than $1 million in 2012 from contributors including Greylock Partners, New Enterprise Associates, Flybridge Capital Partners, and Founder Collective. The company also raised series A funding of $5.6 million in June 2013 from Flybridge Capital Partners and Union Square Ventures. Firebase Authentication and Firebase Hosting were launched in 2014 by Firebase, establishing the company as a leading mobile backend as a service (MbaaS). Firebase became a part of Google in October 2014, and it is now the Google BaaS platform. The technology giant then acquired Divshot, a web hosting platform that was then merged with Firebase.

Here are some examples of Firebase's client showcase:

- Twitch
- Trustpilot
- Lyft
- Venmo
- Duolingo
- The Economist

What is Firebase used for?

It is a pretty comprehensive and flexible platform. It allows users to develop the following categories of applications:

- Android
- iOS
- Web

Firebase use cases:

The Firebase use cases are pretty broad and include:

- MVPs – Minimum viable products
- Real-time applications
- Chat and messaging apps
- Ads optimization based on user behavior
- Sharing photos

- User retention optimization via machine learning
- Business applications

Firebase Advantages:

- Free to start
- Development speed
- End-to-end app development platform
- Powered by Google
- Developers can focus on frontend development
- It’s serverless
- It offers machine learning capabilities
- Generates traffic to your apps
- Error monitoring
- Security

2.5 AI & Machine learning

2.5.1 What is machine learning ?

Machine learning (ML) is a subset of artificial intelligence (AI) focused on developing algorithms that allow computers to learn from data and make predictions or decisions without being explicitly programmed for each task. ML algorithms learn from patterns and trends in data to improve their performance over time.
Key components of machine learning include:

Data: ML algorithms require data to learn and make predictions. This data can be structured or unstructured, and it's used to train the algorithms.

Algorithms: ML algorithms process the data, identify patterns, and extract insights. These algorithms can be supervised, unsupervised, semi-supervised, or reinforcement learning, depending on the learning approach.

Model Training: ML models are trained using labeled data (in supervised learning) or unlabeled data (in unsupervised learning). During training, the model adjusts its parameters to minimize errors and improve performance.

Model Evaluation: After training, ML models are evaluated using validation data to assess their performance and generalization ability. This step helps ensure that the model can make accurate predictions on new, unseen data.

Deployment: Once trained and evaluated, ML models can be deployed in real-world applications to automate tasks, make predictions, or assist in decision-making processes.

Machine learning has numerous applications across various industries, including healthcare, finance, e-commerce, marketing,

and more, revolutionizing how businesses leverage data to gain insights and drive innovation.

What is AI?

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision.

Why is artificial intelligence important?

AI is important for its potential to change how we live, work and play. It has been effectively used in business to automate tasks done by humans, including customer service work, lead generation, fraud detection and quality control. In a number of areas, AI can perform tasks much better than humans. Particularly when it comes to repetitive, detail-oriented tasks, such as analyzing large numbers of legal documents to ensure relevant fields are filled in properly, AI tools often complete jobs quickly and with relatively few errors. Because of the massive data sets it can process, AI can also give enterprises insights into their operations they might not have been aware of. The rapidly expanding population of generative AI tools will be important in fields ranging from education and marketing to product design.

Indeed, advances in AI techniques have not only helped fuel an explosion in efficiency, but opened the door to entirely new business opportunities for some larger enterprises. Prior to the current wave of AI, it would have been hard to imagine using computer software to connect riders to taxis, but Uber has become a Fortune 500 company by doing just that.

AI has become central to many of today's largest and most successful companies, including Alphabet, Apple, Microsoft and Meta, where AI technologies are used to improve operations

and outpace competitors. At Alphabet subsidiary Google, for example, AI is central to its search engine, Waymo's self-driving cars and Google Brain, which invented

the transformer neural network architecture that underpins the recent breakthroughs in natural language processing.

What are the advantages and disadvantages of artificial intelligence?

Artificial neural networks and deep learning AI technologies are quickly evolving, primarily because AI can process large amounts of data much faster and make predictions more accurately than humanly possible.

While the huge volume of data created on a daily basis would bury a human researcher, AI applications using machine learning can take that data and quickly turn it into actionable information. As of this writing, a primary disadvantage of AI is that it is expensive to process the large amounts of data AI programming requires. As AI techniques are incorporated into more products and services, organizations must also be attuned to AI's potential to create biased and discriminatory systems, intentionally or inadvertently.

Advantages of AI:

The following are some advantages of AI.

- **Good at detail-oriented jobs:** AI has proven to be just as good, if not better than doctors at diagnosing certain cancers, including breast cancer and melanoma.
- **Reduced time for data-heavy tasks:** AI is widely used in data-heavy industries, including banking and securities, pharma and insurance, to reduce the time it takes to analyze big data sets. Financial services, for example, routinely use AI to process loan applications and detect fraud.

- **Saves labor and increases productivity:** An example here is the use of warehouse automation, which grew during the pandemic and is expected to increase with the integration of AI and machine learning.
- **Delivers consistent results:** The best AI translation tools deliver high levels of consistency, offering even small businesses the ability to reach customers in their native language.

- **Can improve customer satisfaction through personalization:** AI can personalize content, messaging, ads, recommendations and websites to individual customers.
- **AI-powered virtual agents are always available:** AI programs do not need to sleep or take breaks, providing 24/7 service.

Disadvantages of AI :

The following are some disadvantages of AI:

- Expensive.
- Requires deep technical expertise.
- Limited supply of qualified workers to build AI tools.
- Reflects the biases of its training data, at scale.
- Lack of ability to generalize from one task to another.
- Eliminates human jobs, increasing unemployment rates.

Strong AI vs. weak AI:

AI can be categorized as weak or strong.

- **Weak AI**, also known as narrow AI, is designed and trained to complete a specific task. Industrial robots and virtual personal assistants, such as Apple's Siri, use weak *AI*.

- **Strong AI**, also known as artificial general intelligence (AGI), describes programming that can replicate the cognitive abilities of the human brain. When presented with an unfamiliar task, a strong AI system can use fuzzy logic to apply knowledge from one domain to another and find a solution
- autonomously. In theory, a strong AI program should be able to pass both a Turing test and the Chinese Room argument.

What are the 4 types of artificial intelligence?

Arend Hintze, an assistant professor of integrative biology and computer science and engineering at Michigan State University, explained that AI can be categorized into four types, beginning with the task-specific intelligent systems in wide use today and progressing to sentient systems, which do not yet exist. The categories are as follows.

- Type 1: Reactive machines: These AI systems have no memory and are task-specific. An example is Deep Blue, the IBM chess program that beat Garry Kasparov in the 1990s. Deep Blue can identify pieces on a chessboard and make predictions, but because it has no memory, it cannot use past experiences to inform future ones.

- Type 2: Limited memory: These AI systems have memory, so they can use past experiences to inform future decisions. Some of the decision-making functions in self-driving cars are designed this way.
- Type 3: Theory of mind: Theory of mind is a psychology term. When applied to AI, it means the system would have the social intelligence to understand emotions. This type of AI will be able to infer human intentions and predict behavior, a necessary skill for AI systems to become integral members of human teams.

- Type 4: Self-awareness: In this category, AI systems have a sense of self, which gives them consciousness. Machines with self-awareness understand their own current state. This type of AI does not yet exist.

What are examples of AI technology and how is it used today?

AI is incorporated into a variety of different types of technology. Here are seven examples.

Automation: When paired with AI technologies, automation tools can expand the volume and types of tasks performed. An example is robotic process automation (RPA), a type of software that automates repetitive, rules-based data processing tasks traditionally done by humans. When combined with machine learning and emerging AI tools, RPA can automate bigger portions of enterprise jobs, enabling RPA's tactical bots to pass along intelligence from AI and respond to process changes.

Machine learning: This is the science of getting a computer to act without programming. Deep learning is a subset of machine learning that, in very simple terms, can be thought of as the automation of predictive analytics. There are three types of machine learning algorithms:

- Supervised learning: Data sets are labeled so that patterns can be detected and used to label new data sets.
- Unsupervised learning**:** Data sets aren't labeled and are sorted according to similarities or differences.
- Reinforcement learning: Data sets aren't labeled but, after performing an action or several actions, the AI system is given feedback.

Machine vision: This technology gives a machine the ability to see. Machine vision captures and analyzes visual information using a camera, analog-to-digital conversion and digital signal processing. It is often compared to human eyesight, but machine vision isn't bound by biology and can be programmed to see through walls, for example. It is used in a range of applications from signature identification to medical image analysis. Computer vision, which is focused on machine-based image processing, is often conflated with machine vision.

Natural language processing (NLP): This is the processing of human language by a computer program. One of the older and best-known examples of NLP is spam detection, which looks at the subject line and text of an email and decides if it's junk.

Current approaches to NLP are based on machine learning. NLP tasks include text translation, sentiment analysis and speech recognition.

Robotics: This field of engineering focuses on the design and manufacturing of robots. Robots are often used to perform tasks that are difficult for humans to perform or perform consistently. For example, robots are used in car production assembly lines or by NASA to move large objects in space. Researchers also use machine learning to build robots that can interact in social settings.

Self-driving cars: Autonomous vehicles use a combination of computer vision, image recognition and deep learning to build automated skills to pilot a vehicle while staying in a given lane and avoiding unexpected obstructions, such as pedestrians.

Text, image and audio generation: Generative AI techniques, which create various types of media from text prompts, are being applied extensively across businesses to create a seemingly limitless range of content types from photorealistic art to email responses and screenplays.

What are the applications of AI?

Artificial intelligence has made its way into a wide variety of markets. Here are 11 examples.

AI in healthcare: The biggest bets are on improving patient outcomes and reducing costs. Companies are applying machine learning to make better and faster medical diagnoses than humans. One of the best-known healthcare technologies is IBM Watson. It understands natural language and can respond to questions asked of it. The system mines patient data and other available data sources to form a hypothesis, which it then presents with a confidence scoring schema. Other AI applications include using online virtual health assistants and chatbots to help patients and healthcare customers find medical information, schedule appointments, understand the billing process and complete other administrative processes. An array of AI technologies is also being used to predict, fight and understand pandemics such as COVID-19.

AI in business: Machine learning algorithms are being integrated into analytics and customer relationship management (CRM) platforms to uncover information on how to better serve customers. Chatbots have been incorporated into websites to provide immediate service to customers. The rapid advancement of generative AI technology such as ChatGPT is expected to have far-reaching consequences: eliminating jobs, revolutionizing product design and disrupting business models.

AI in education: AI can automate grading, giving educators more time for other tasks. It can assess students and adapt to their needs, helping them work at their own pace. AI tutors can provide additional support to students, ensuring they stay on track. The technology could also change where and how students learn, perhaps even replacing some teachers. As demonstrated by ChatGPT, Google Bard and other large language models, generative AI can help educators craft course work and other teaching materials and engage students in new ways. The advent of these tools also

forces educators to rethink student homework and testing and revise policies on plagiarism.

AI in finance: AI in personal finance applications, such as Intuit Mint or TurboTax, is disrupting financial institutions. Applications such as these collect personal data and provide financial advice. Other programs, such as IBM Watson, have been applied to the process of buying a home. Today, artificial intelligence software performs much of the trading on Wall Street.

AI in law: The discovery process -- sifting through documents -- in law is often overwhelming for humans. Using AI to help automate the legal industry's labor-intensive processes is saving time and improving client service. Law firms use machine learning to describe data and predict outcomes, computer vision to classify and extract information from documents, and NLP to interpret requests for information.

AI in entertainment and media: The entertainment business uses AI techniques for targeted advertising, recommending content, distribution, detecting fraud, creating scripts and making movies. Automated journalism helps newsrooms streamline media workflows reducing time, costs and complexity. Newsrooms use AI to

automate routine tasks, such as data entry and proofreading; and to research topics and assist with headlines. How journalism can reliably use ChatGPT and other generative AI to generate content is open to question.

AI in software coding and IT processes: New generative AI tools can be used to produce application code based on natural language prompts, but it is early days for these tools and unlikely they will replace software engineers soon. AI is also being used to automate many IT processes, including data entry, fraud detection, customer service, and predictive maintenance and security.

Security: AI and machine learning are at the top of the buzzword list security vendors use to market their products, so buyers should approach with caution. Still, AI techniques are being successfully applied to multiple aspects of cybersecurity, including anomaly detection, solving the false-positive problem and conducting behavioral threat analytics. Organizations use machine learning in security information and event management (SIEM) software and related areas to detect anomalies and identify suspicious activities that indicate threats. By analyzing data and using logic to identify similarities to known malicious code, AI can provide alerts to new and emerging attacks much sooner than human employees and previous technology iterations.

AI in manufacturing: Manufacturing has been at the forefront of incorporating robots into the workflow. For example, the industrial robots that were at one time programmed to perform single tasks and separated from human workers, increasingly function as cobots: Smaller, multitasking robots that collaborate with humans and take on responsibility for more parts of the job in warehouses, factory floors and other workspaces.

AI in banking: Banks are successfully employing chatbots to make their customers aware of services and offerings and to handle transactions that don't require human intervention. AI virtual assistants are used to improve and cut the costs of compliance with banking regulations. Banking organizations use AI to improve their decision-making for loans, set credit limits and identify investment opportunities.

AI in transportation: In addition to AI's fundamental role in operating autonomous vehicles, AI technologies are used in transportation to manage traffic, predict flight delays, and make ocean shipping safer and more efficient. In supply chains, AI is replacing traditional methods of forecasting demand and predicting disruptions, a trend accelerated by COVID-19 when many companies were caught off guard by the effects of a global pandemic on the supply and demand of goods.

What Is Machine Learning?

Machine learning (ML) is a discipline of artificial intelligence (AI) that provides machines with the ability to automatically learn from data and past experiences while identifying patterns to make predictions with minimal human intervention.

Machine learning methods enable computers to operate autonomously without explicit programming. ML applications are fed with new data, and they can independently learn, grow, develop, and adapt.

Machine learning derives insightful information from large volumes of data by leveraging algorithms to identify patterns and learn in an iterative process. ML algorithms use computation methods to learn directly from data instead of relying on any predetermined equation that may serve as a model.

The performance of ML algorithms adaptively improves with an increase in the number of available samples during the 'learning' processes. For example, deep learning is a sub-domain of machine learning that trains computers to imitate natural human traits like learning from examples. It offers better performance parameters than conventional ML algorithms.

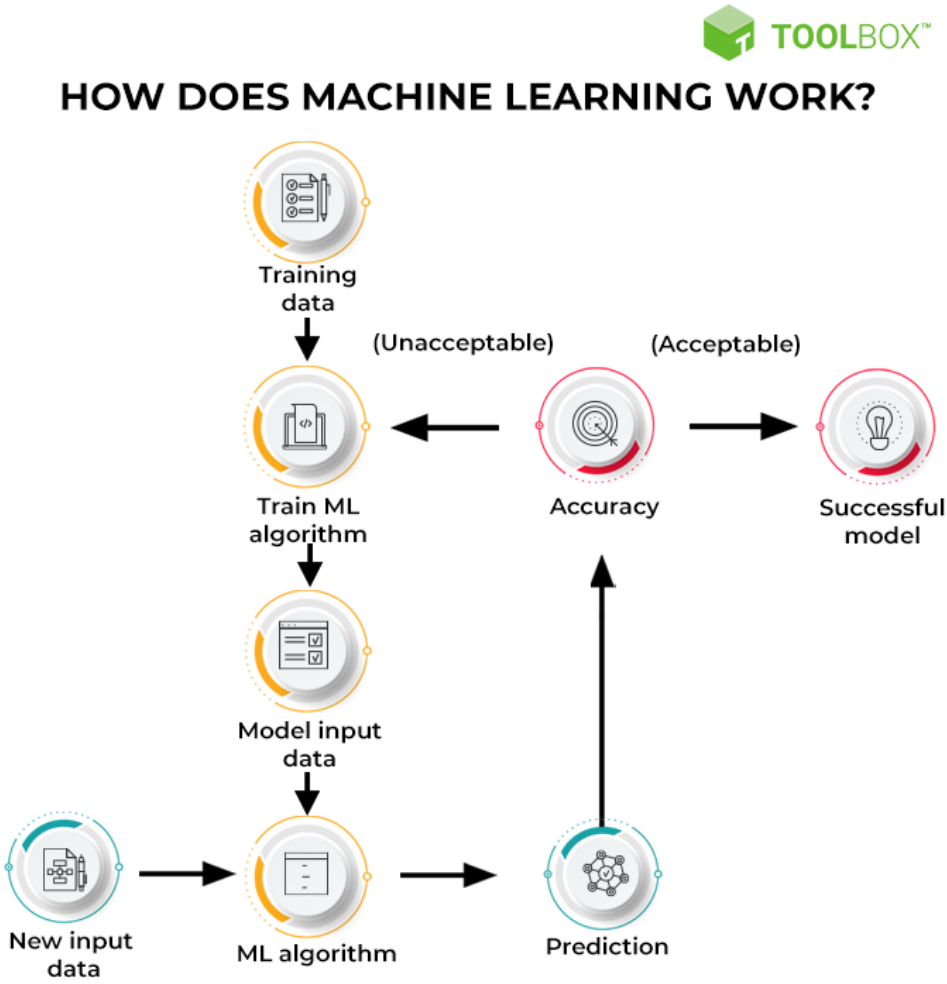
While machine learning is not a new concept – dating back to World War II when the Enigma Machine was used – the ability to apply complex mathematical calculations automatically to growing volumes and varieties of available data is a relatively recent development.

Today, with the rise of big data, IoT, and ubiquitous computing, machine learning has become essential for solving problems across numerous areas, such as

- Computational finance (credit scoring, algorithmic trading)
- Computer vision (facial recognition, motion tracking, object detection)
- Computational biology (DNA sequencing, brain tumor detection, drug discovery)
- Automotive, aerospace, and manufacturing (predictive maintenance)
- Natural language processing (voice recognition)

How does machine learning work?

Machine learning algorithms are molded on a training dataset to create a model. As new input data is introduced to the trained ML algorithm, it uses the developed model to make a prediction.



How Machine Learning Works

***Note**: The above illustration discloses a high-level use case scenario. However, typical machine learning examples may involve many other factors, variables, and steps.*

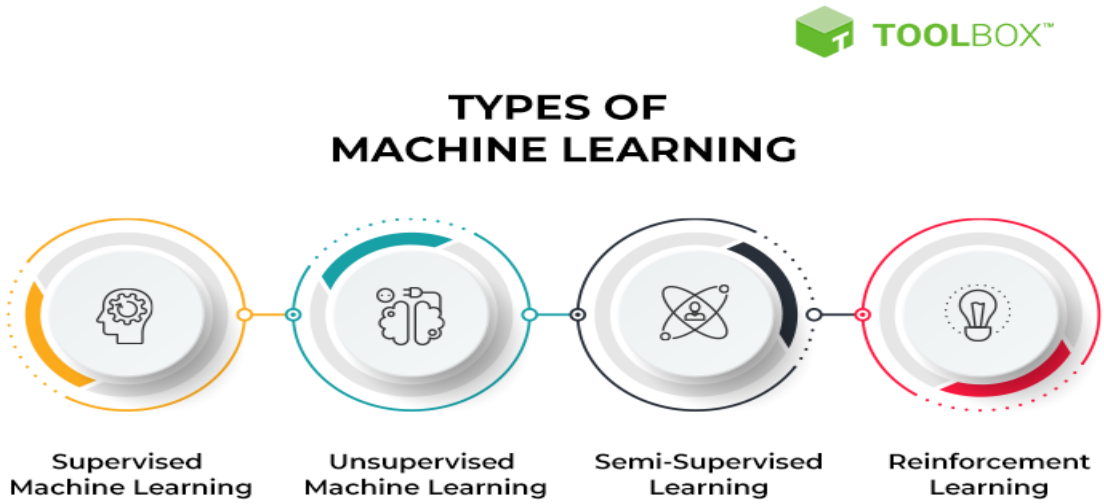
Further, the prediction is checked for accuracy. Based on its accuracy, the ML algorithm is either deployed or trained repeatedly with an augmented training dataset until the desired accuracy is achieved.

See More: **What Is Artificial Intelligence (AI) as a Service? Definition, Architecture, and Trends**

Types of Machine Learning

Machine learning algorithms can be trained in many ways, with each method having its pros and cons. Based on these methods and ways of learning, machine learning is broadly categorized into four main types:

Types of Machine Learning



What is Deep Learning?

Deep learning is a method in artificial intelligence (AI) that teaches computers to process data in a way that is inspired by the human brain. Deep learning models can recognize complex patterns in pictures, text, sounds, and other data to produce accurate insights and predictions. You can use deep learning methods to automate tasks that typically require human intelligence, such as describing images or transcribing a sound file into text.

Why is deep learning important?

Artificial intelligence (AI) attempts to train computers to think and learn as humans do. Deep learning technology drives many AI applications used in everyday products, such as the following:

- Digital assistants
- Voice-activated television remotes
- Fraud detection
- Automatic facial recognition

It is also a critical component of emerging technologies such as self-driving cars, virtual reality, and more.

Deep learning models are computer files that data scientists have trained to perform tasks using an algorithm or a predefined set of steps. Businesses use deep learning models to analyze data and make predictions in various applications.

How does deep learning work?

Deep learning algorithms are neural networks that are modeled after the human brain. For example, a human brain contains millions of interconnected neurons that work together to learn and process information. Similarly, deep learning neural networks, or artificial neural networks, are made of many layers of artificial neurons that work together inside the computer.

Artificial neurons are software modules called nodes, which use mathematical calculations to process data. Artificial neural networks are deep learning algorithms that use these nodes to solve complex problems.

What are the components of a deep learning network?

The components of a deep neural network are the following.

Input layer

An artificial neural network has several nodes that input data into it. These nodes make up the input layer of the system.

Hidden layer

The input layer processes and passes the data to layers further in the neural network. These hidden layers process information at different levels, adapting their behavior as they receive new information. Deep learning networks have hundreds of hidden layers that they can use to analyze a problem from several different angles.

For example, if you were given an image of an unknown animal that you had to classify, you would compare it with animals you already know. For example, you would look at the shape of its eyes and ears, its size, the number of legs, and its fur pattern. You would try to identify patterns, such as the following:

- The animal has hooves, so it could be a cow or deer.
- The animal has cat eyes, so it could be some type of wild cat.

The hidden layers in deep neural networks work in the same way. If a deep learning algorithm is trying to classify an animal image, each of its hidden layers processes a different feature of the animal and tries to accurately categorize it.

Output layer

The output layer consists of the nodes that output the data. Deep learning models that output "yes" or "no" answers have only two nodes in the output layer. On the other hand, those that output a wider range of answers have more nodes.

What is deep learning in the context of machine learning?

Deep learning is a subset of machine learning. Deep learning algorithms emerged in an attempt to make traditional machine learning techniques more efficient. Traditional machine learning methods require significant human effort to train the software.

For example, in animal image recognition, you need to do the following:

- Manually label hundreds of thousands of animal images.
- Make the machine learning algorithms process those images.
- Test those algorithms on a set of unknown images.
- Identify why some results are inaccurate.
- Improve the dataset by labeling new images to improve result accuracy.

This process is called supervised learning. In supervised learning, result accuracy improves only when you have a broad and sufficiently varied dataset. For instance, the algorithm might accurately identify black cats but not white cats because the training dataset had more images of black cats. In that case, you would need to label more white cat images and train the machine learning models once again.

What are the benefits of deep learning over machine learning?

A deep learning network has the following benefits over traditional machine learning.

Efficient processing of unstructured data

Machine learning methods find unstructured data, such as text documents, challenging to process because the training dataset can have infinite variations. On the other hand, deep learning models can comprehend unstructured data and make general observations without manual feature extraction. For instance, a neural network can recognize that these two different input sentences have the same meaning:

- Can you tell me how to make the payment?
- How do I transfer money?

Hidden relationships and pattern discovery

A deep learning application can analyze large amounts of data more deeply and reveal new insights for which it might not have been trained. For example, consider a deep learning model that is trained to analyze consumer purchases. The model has data only for the items you have already purchased. However, the artificial neural network can suggest new items that you haven't bought by comparing your buying patterns to those of other similar customers.

Unsupervised learning

Deep learning models can learn and improve over time based on user behavior. They do not require large variations of labeled datasets. For example, consider a neural network that automatically corrects or suggests words by analyzing your typing behavior. Let's assume it was trained in the English language and can spell-check English words. However, if you frequently type non-English words, such as *danke*, the neural network automatically learns and autocorrects these words too.

Volatile data processing:

Volatile datasets have large variations. One example is loan repayment amounts in a bank. A deep learning neural network can categorize and sort that data as well, such as by analyzing financial transactions and flagging some of them for fraud detection.

What are the challenges of deep learning?

As deep learning is a relatively new technology, certain challenges come with its practical implementation.

Large quantities of high-quality data

Deep learning algorithms give better results when you train them on large amounts of high-quality data. Outliers or mistakes in your input dataset can significantly affect the deep learning process. For instance, in our animal image example, the deep learning model might classify an airplane as a turtle if non-animal images were accidentally introduced in the dataset.

To avoid such inaccuracies, you must clean and process large amounts of data before you can train deep learning models. The input data preprocessing requires large amounts of data storage capacity.

Large processing power

Deep learning algorithms are compute-intensive and require infrastructure with sufficient compute capacity to properly function. Otherwise, they take a long time to process results.

Pre-Trained Model

Encord Computer Vision Glossary

Pre-trained model

A pre-trained model is a machine learning (ML) model that has been trained on a large dataset and can be fine-tuned for a specific task. Pre-trained models are often used as a starting point for developing ML models, as they provide a set of initial weights and biases that can be fine-tuned for a specific task.

There are several advantages to using pre-trained models, including the ability to leverage the knowledge and experience of others, the ability to save time and resources, and the ability to improve model

performance.

Pre-trained models are often trained on large, diverse datasets and have been trained to recognize a wide range of patterns and features. As a result, they can provide a strong foundation for fine-tuning and can significantly improve the performance of the model.

Pre-trained models come in a variety of forms, such as language models, object detection models, and picture classification models. Convolutional neural networks are frequently used as the foundation for image classification models, which are trained to categorize images into predetermined categories (CNNs).

CNNs or region-based convolutional neural networks are frequently used as the foundation for object recognition models, which are taught to recognise and categorize items in photos or videos (R-CNNs). Recurrent neural networks (RNNs) or transformers are frequently used as the foundation for language models, which are trained to predict the next word in a sequence.

Overall, pre-trained models are a useful tool in ML, and can be used as a starting point for developing ML models. They provide a set of initial weights and biases that can be fine-tuned for a specific task and can significantly improve the performance of the model.

Why Are Pretrained AI Models Used?

Instead of building an AI model from scratch, developers can use pretrained models and customize them to meet their requirements.

To build an AI application, developers first need an AI model that can accomplish a particular task, whether that’s identifying a mythical horse, detecting a safety hazard for an autonomous vehicle or diagnosing a cancer based on medical imaging. That model needs a lot of representative data to learn from.

This learning process entails going through several layers of incoming data and emphasizing goals-relevant characteristics at each layer.

To create a model that can recognize a unicorn, for example, one might first feed it images of unicorns, horses, cats, tigers and other animals. This is the incoming data.

Then, layers of representative data traits are constructed, beginning with the simple — like lines and colors — and advancing to complex structural features. These characteristics are assigned varying degrees of relevance by calculating probabilities.

As opposed to a cat or tiger, for example, the more like a horse a creature appears, the greater the likelihood that it is a unicorn. Such probabilistic values are stored at each neural network layer in the AI model, and as layers are added, its understanding of the representation improves.

To create such a model from scratch, developers require enormous datasets, often with billions of rows of data. These can be pricey and challenging to obtain, but compromising on data can lead to poor performance of the model.

Precomputed probabilistic representations — known as weights — save time, money and effort. A pretrained model is already built and trained with these weights.

Using a high-quality pretrained model with a large number of accurate representative weights leads to higher chances of success for AI deployment. Weights can be modified, and more data can be added to the model to further customize or fine-tune it.

Developers building on pretrained models can create AI applications faster, without having to worry about handling

mountains of input data or computing probabilities for dense layers.In other words, using a pretrained AI model is like getting a dress or a shirt and then tailoring it to fit your needs, rather than starting with fabric, thread and needle.

Pretrained AI models are often used for transfer learning and can be based on several model architecture types. One popular architecture type is the transformer model, a neural network that learns context and meaning by tracking relationships in sequential data.

According to Alfredo Ramos, senior vice president of platform at AI company Clarifai — a Premier partner in the NVIDIA Inception program for startups — pretrained models can cut AI application development time by up to a year and lead to cost savings of hundreds of thousands of dollars.

How Are Pretrained Models Advancing AI?

Since pretrained models simplify and quicken AI development, many developers and companies use them to accelerate various AI use cases.

Top areas in which pretrained models are advancing AI include:

- **Natural language processing:** Pretrained models are used for translation, chatbots and other natural language processing applications. Large language models, often based on the transformer model architecture, are an extension of pretrained models. One example of a pretrained LLM is NVIDIA NeMo Megatron, one of the world's largest AI models.

- **Speech AI:** Pretrained models can help speech AI applications plug and play across different languages. Use cases include call center automation, AI assistants and voice-recognition technologies.
- **Computer vision:** Like in the unicorn example above, pretrained models can help AI quickly recognize creatures — or objects, places and people. In this way, pretrained models accelerate computer vision, giving applications human-like vision capabilities across sports, smart cities and more.
- **Healthcare:** For healthcare applications, pretrained AI models like MegaMolBART — part of the NVIDIA BioNeMo service and framework — can understand the language of chemistry and learn the relationships between atoms in real-world molecules, giving the scientific community a powerful tool for faster drug discovery.
- **Cybersecurity:** Pretrained models provide a starting point to implement AI-based cybersecurity solutions and extend the capabilities of human security analysts to detect threats faster. Examples include digital fingerprinting of humans and machines, and detection of anomalies, sensitive information and phishing.
- **Art and creative workflows:** Bolstering the recent wave of AI art, pretrained models can help accelerate creative workflows through tools like GauGAN and NVIDIA Canvas.

Pretrained AI models can be applied across industries beyond these, as their customization and fine-tuning can lead to infinite possibilities for use cases.

Mobilenet V2 :

MobileNetV2 is a classification model (distinct from MobileNetSSDv2) developed by Google. It provides real-time classification capabilities under computing constraints in devices like smartphones. This implementation leverages transfer learning from

ImageNet to your dataset.

MobileNetV2 is an enhancement of the original MobileNet architecture, which is a neural network model designed for mobile and embedded vision applications. Developed by a research team at Google, MobileNetV2 aims to improve performance efficiency compared to the original MobileNet architecture.

Key features of MobileNetV2 include:

Depthwise Separable Convolution: MobileNetV2 extensively uses depthwise separable convolution, which decomposes the standard convolution into depthwise convolution followed by pointwise convolution. This significantly reduces the number of parameters and computational cost while preserving representational capacity.

Inverted Residuals with Linear Bottlenecks: It introduces the concept of inverted residuals where the input and output of the residual block have the same number of channels. Linear bottlenecks are employed to avoid non-linearity in the bottleneck layers, which improves training stability and efficiency.

Linear Bottlenecks: Instead of using non-linear activation functions in the bottleneck layers, MobileNetV2 employs linear activations. This simplifies the model and enhances information flow through the network.

Width and Resolution Multipliers: MobileNetV2 introduces two hyperparameters, width multiplier and resolution multiplier, which allow the

model to be scaled up or down to meet various performance and resource constraints.

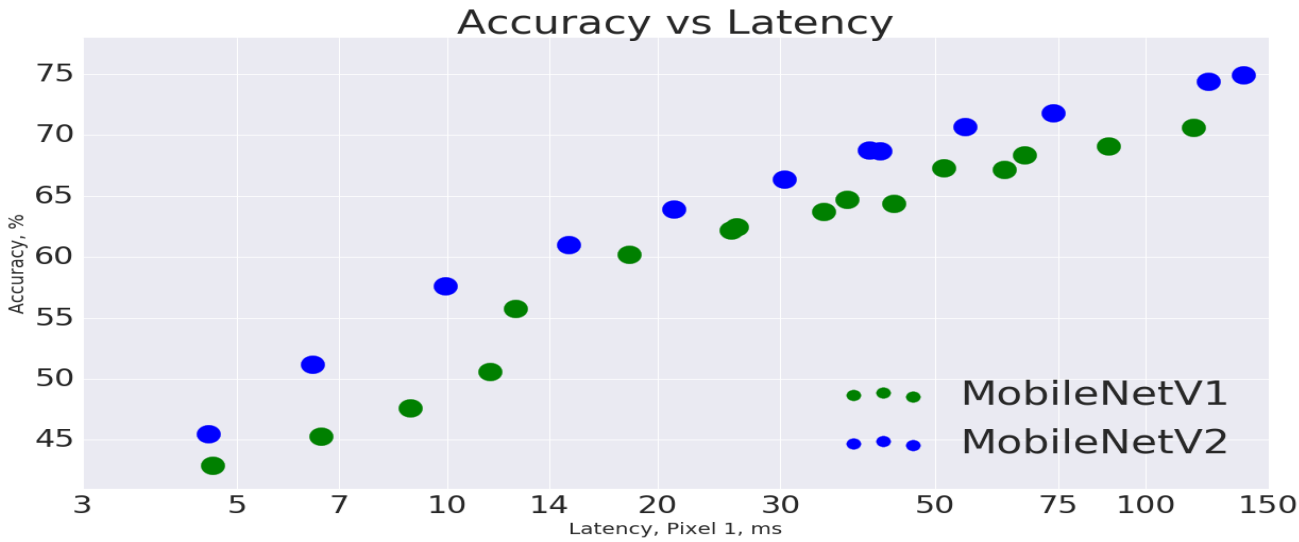
Pre-trained Implementation: MobileNetV2 has been implemented in popular deep learning frameworks such as TensorFlow and PyTorch and is available as a pre-trained model for direct use in various vision applications.

MobileNetV2 Architecture

The MobileNetV2 architecture utilizes an inverted residual structure where the input and output of the residual blocks are thin bottleneck layers. MobileNetV2 also uses lightweight convolutions to filter features in the expansion layer. Finally, it removes non-linearities in the narrow layers.

MobileNetV2 Results

MobileNet V2 outperforms MobileNet V1 with higher accuracies and lower latencies.



Chapter 3: Implementation and Coding

Software Techniques:

Design

we use Adobe XD for the design of the prototype

Front End

flutter to create a basic application interface that interacts with the user

Back End

we use firebase, It offers services such as real-time databases, cloud storage, authentication

Analyzing the system:

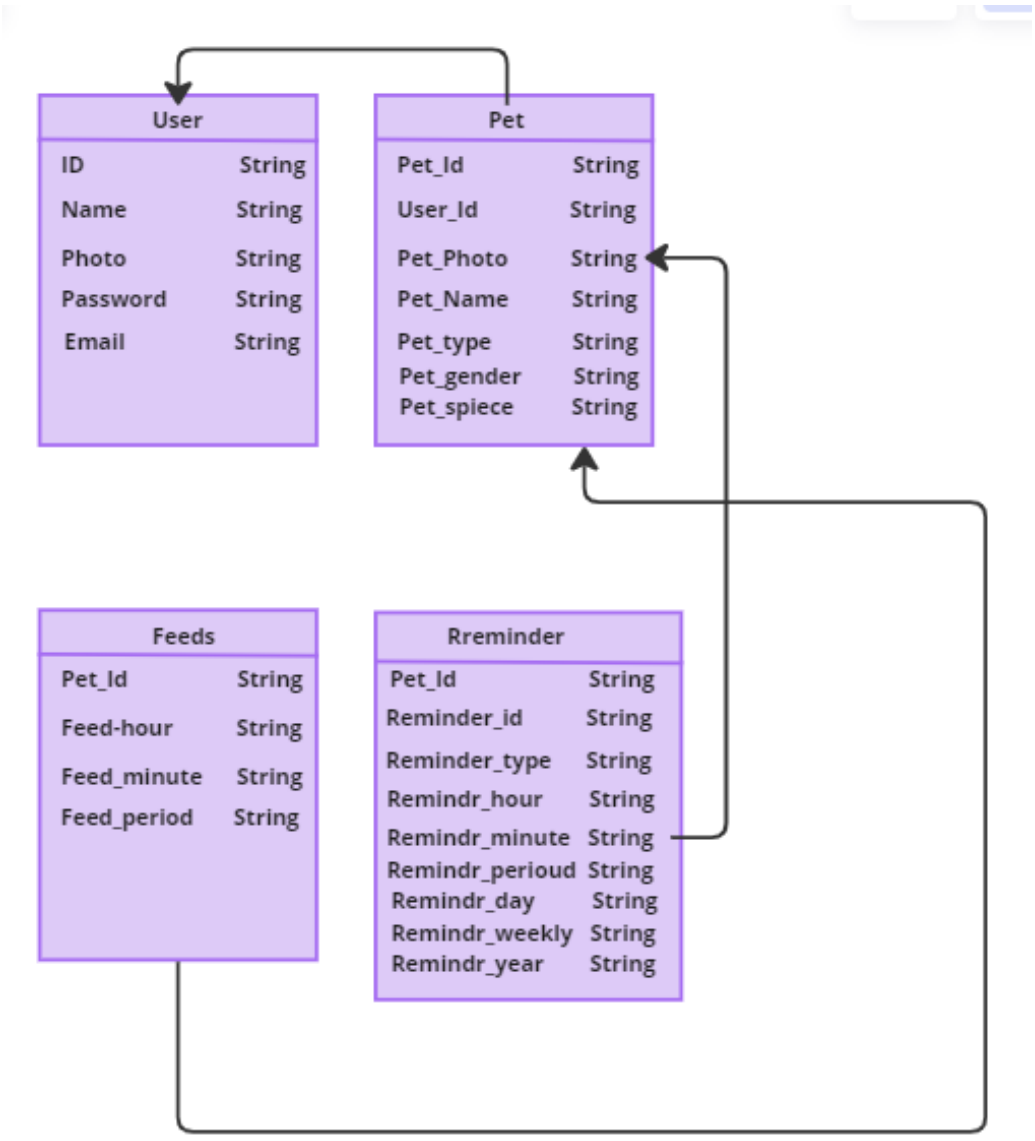
the pet application system entails a thorough examination of its key features and user interface. The system is equipped with sophisticated image recognition capabilities that accurately detect and classify various pets, ensuring a personalized user experience. It also includes a comprehensive reminder functionality, allowing users to set and manage reminders for various pet-related tasks such as feeding, grooming, and medical appointments. By utilizing cloud-based services like Firebase, the application ensures real-time data synchronization, secure user authentication, and reliable storage solutions.

The importance of system analysis is to divide the complex system in its structure into its main components in a logical manner, taking into account the scope of the system, its objectives, and the organizational framework of the system as a whole, and this comes on the sidelines of the analyst's attempt to seek to develop and improve the system. Below we try to explain some steps of our system with (ERD Diagram - Network Diagram - Activity Diagram) ,

because that will be able to help the readers to understand the new system more easily.

Entity Relationship Diagram (ERD) :

An entity–relationship model (or ER model) describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between entities (instances of those entity types)



Entity Relationship Diagram (ERD)

- **User Attributes:**
    1. ID
    2. Name
    3. Email
    4. Password
    5. Photo

- **Pet Attributes:**
    1. Pet_photo
    2. Pet_Name
    3. Pet_gender
    4. Pet_type
    5. Pet_piece
    6. User_id
    7. Pet_id

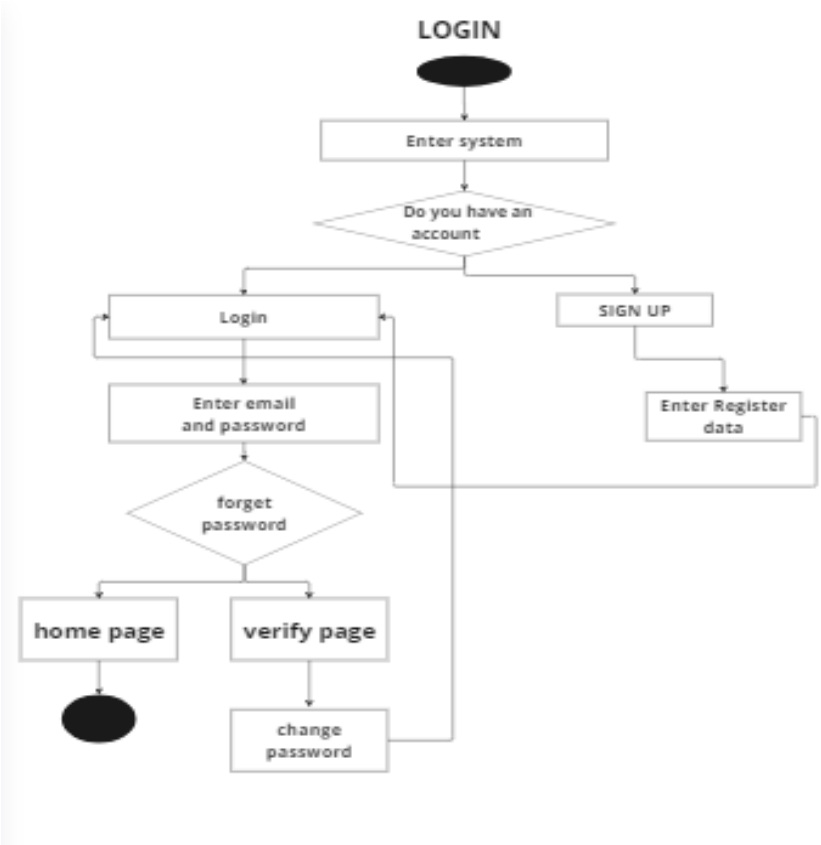
- **Feed Attributes:**
    1. Pet_id
    2. User_id
    3. Feed_hour
    4. Feed_minute
    5. Feed_perioud

- **Reminder Attributes:**
    1. Pet_id
    2. User_id
    3. Reminder_id
    4. Remindr_name
    5. Reminder_type
    6. Reminder_hour
    7. Reminder_perioud
    8. Reminder_minute

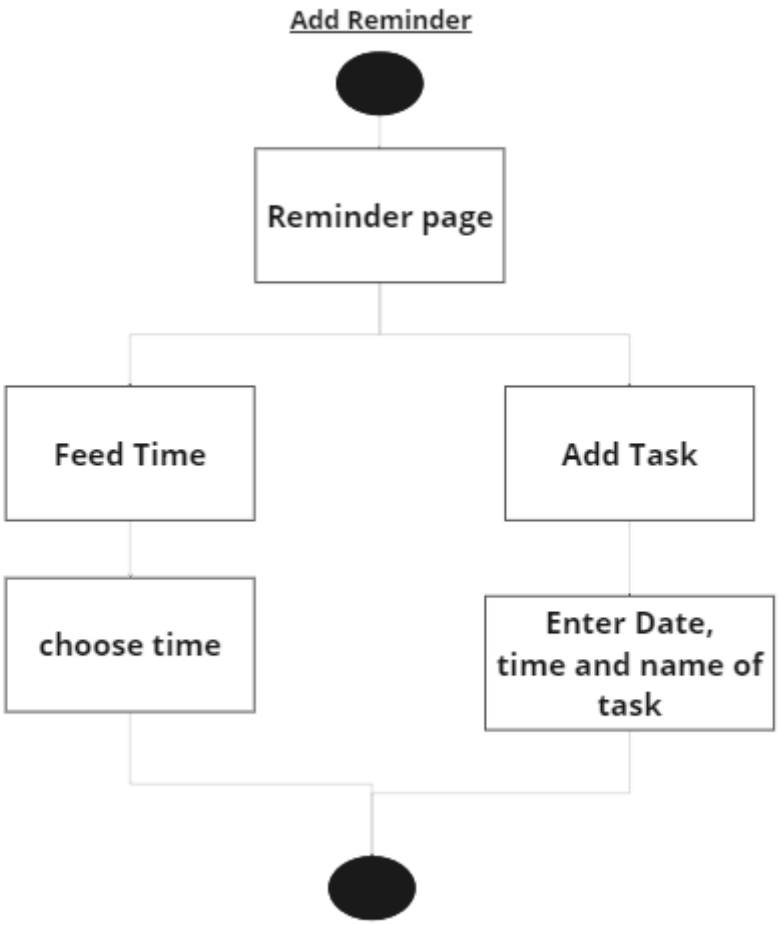
9. Reminder_weekly
10. Reminder_month
11. Reminder_day

Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores.



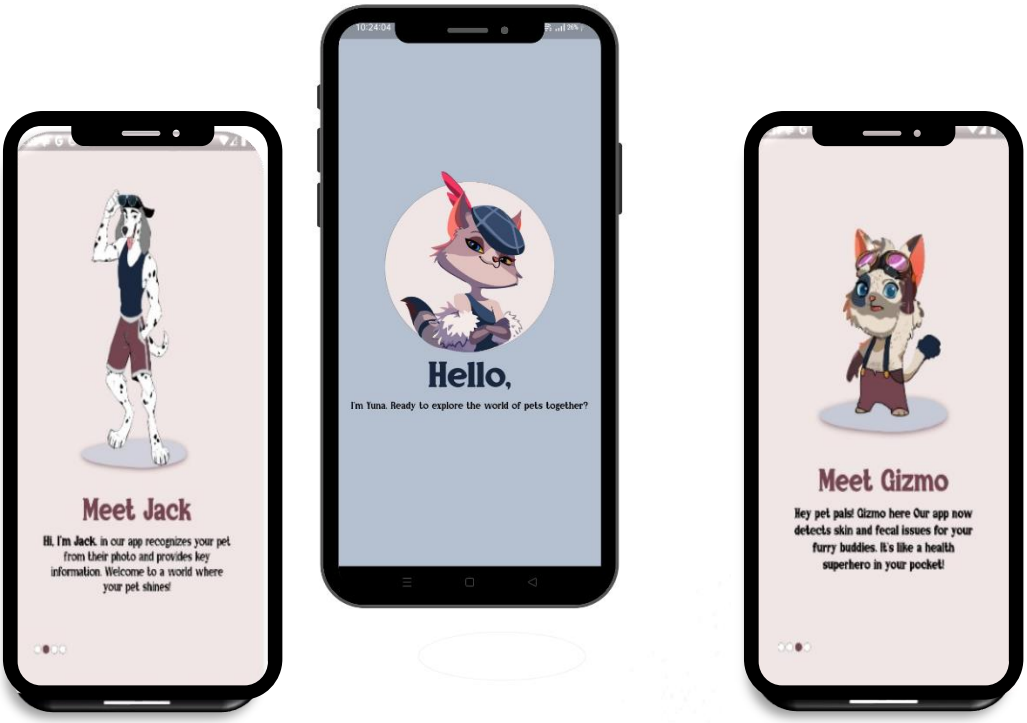

Add pet
Add pet
Detect
Add pet manually
detect page
camera page
take photo
gallery
add pet info
choose photo
detect for type
Result page of type detection
pet page


Add Reminder
Reminder page
Feed Time
Add Task
choose time
Enter Date, time and name of task

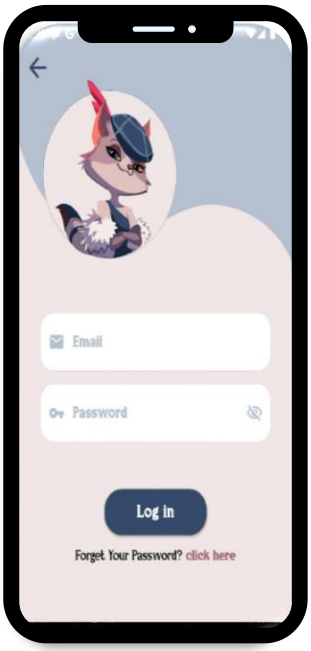

Detect Diseases
choose pet for detection
disease detect page
stole diseases
skin diseases
camera pge
take photo
camera page
take photo
gallery
choose photo
detect the disease
Result page of detection
about disease
symptoms
prevention
treatment

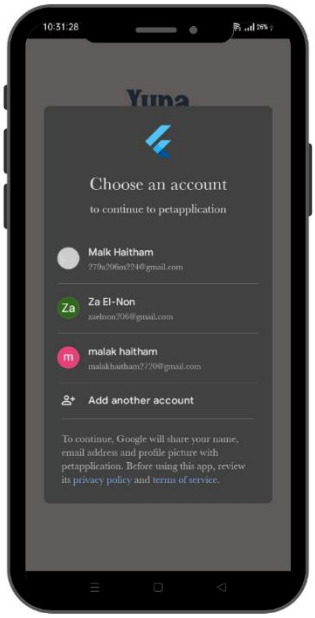
Application UI (Phone view):

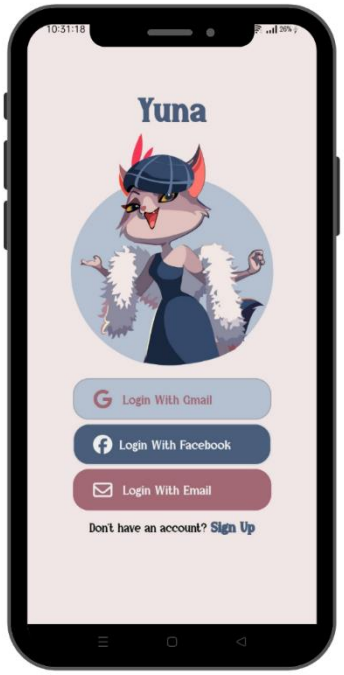
Welcome page



Login pages

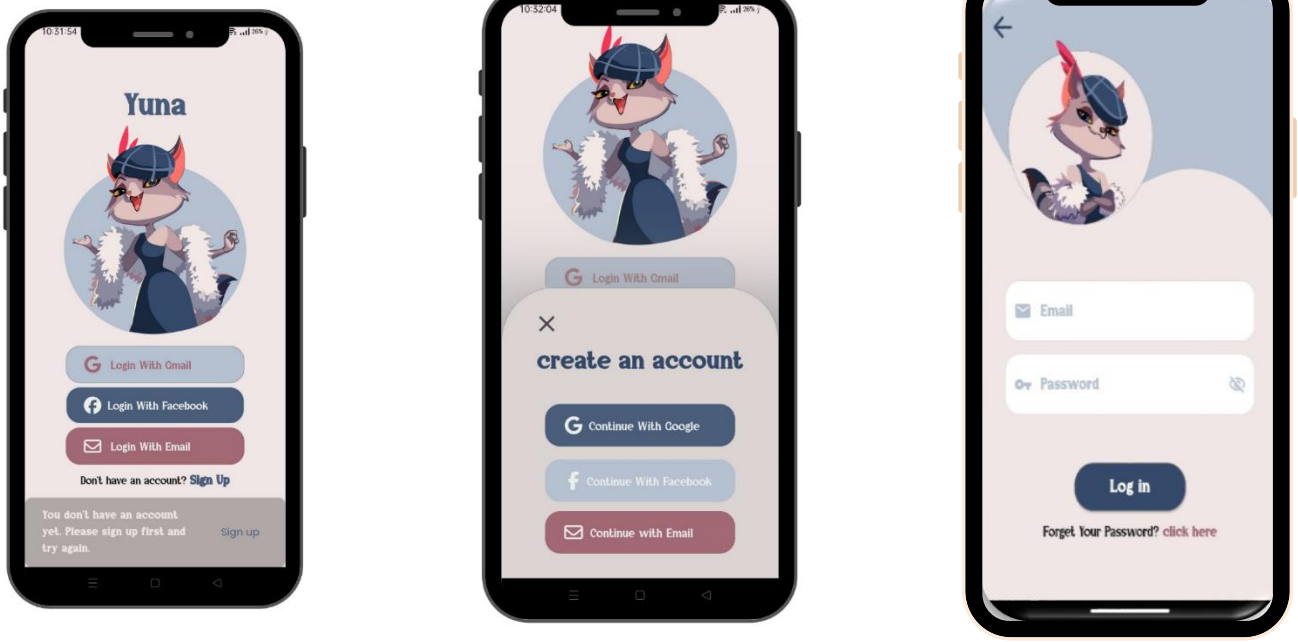






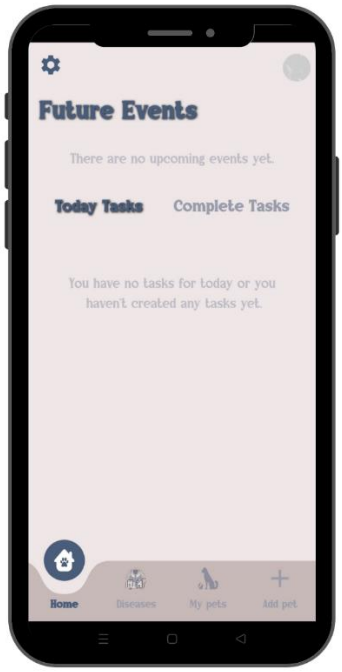
The login page allows the user to log into the application with his own account which he previously registered. If it is his first time using the app, he can click "Sign up" to send it to the new account registration page

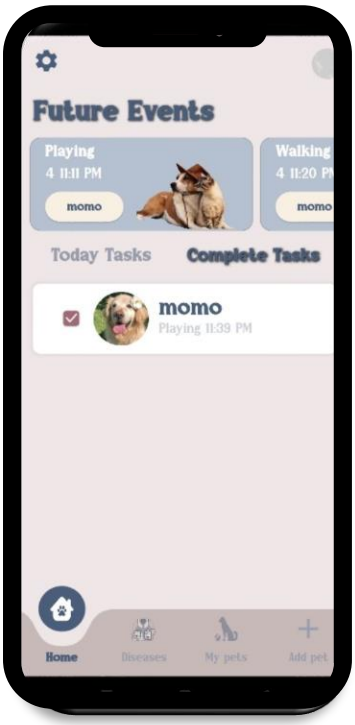
SignUp pages



The Sign up page through which the user can register a new account in the application in order to be able to use the application and log into the main page, and when the Sign up button is pressed, the data is sent to firebase and thus he is able to log in to the application and on this page the user is required Enter each of (name - email - password) ,the user can register using Gmail of Facebook account

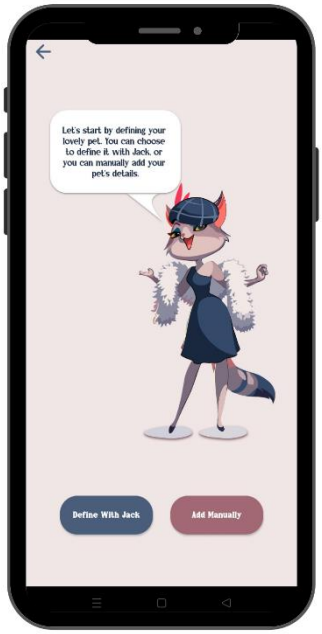
Home page

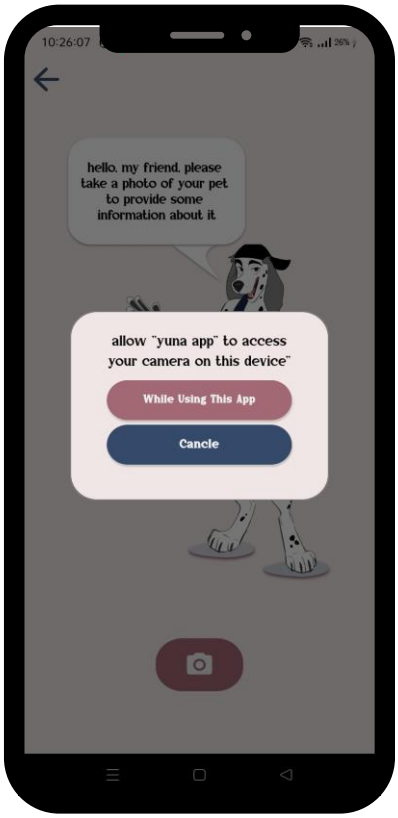




Home page that contains future events and today tasks that user put them .there are button that enables users to add pet.

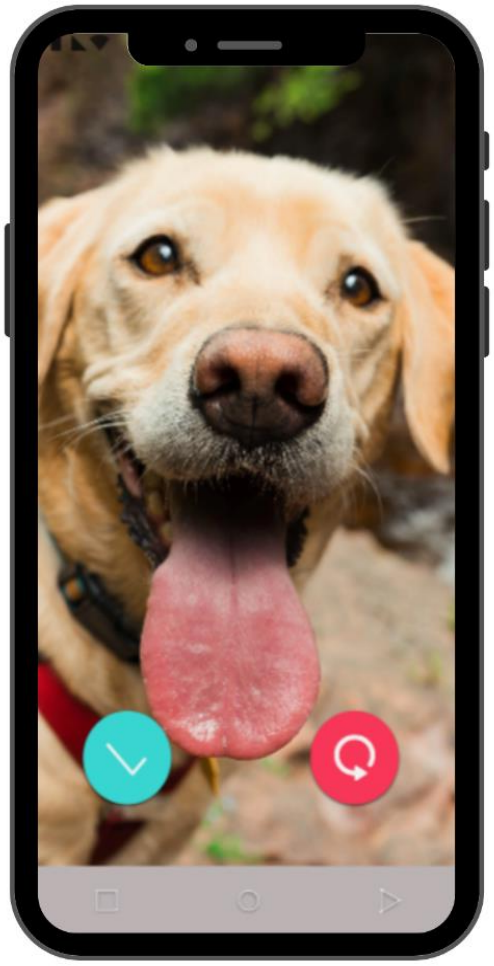
Define pages:

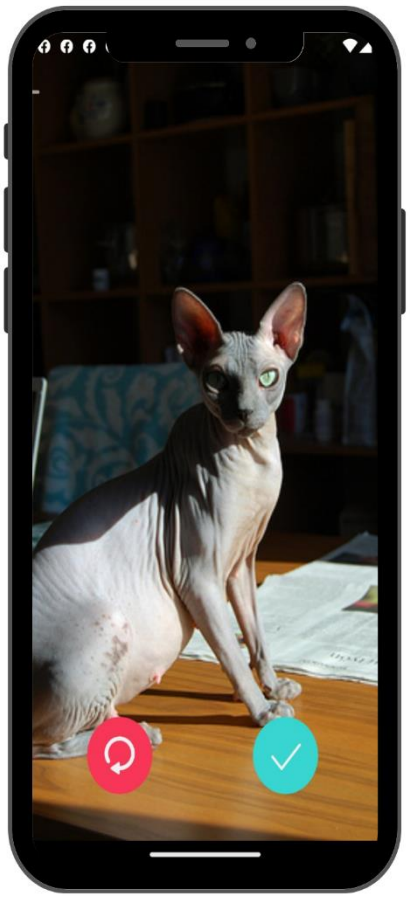




User chooses the way he needs to define the pet with (manually or with jack(ai))

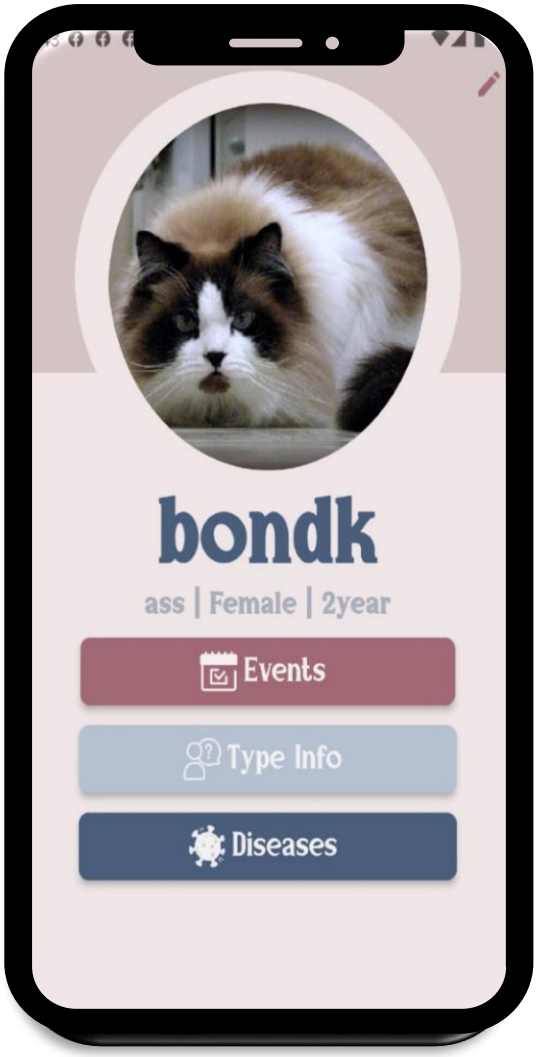
Camera page:





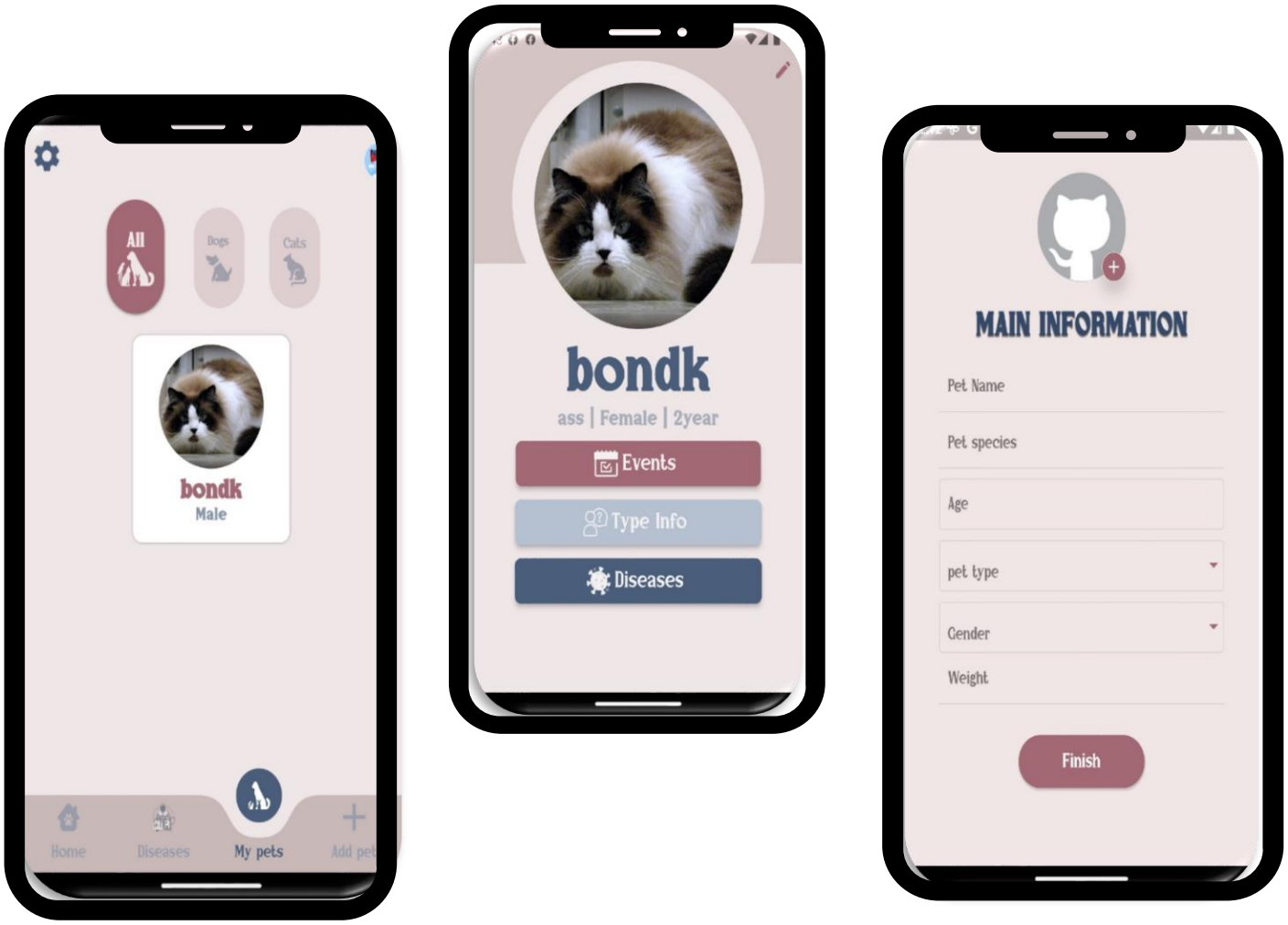
when user chooses add with jack ,he can adds the pet photo from gallery or takes a photo using camera

Result page:



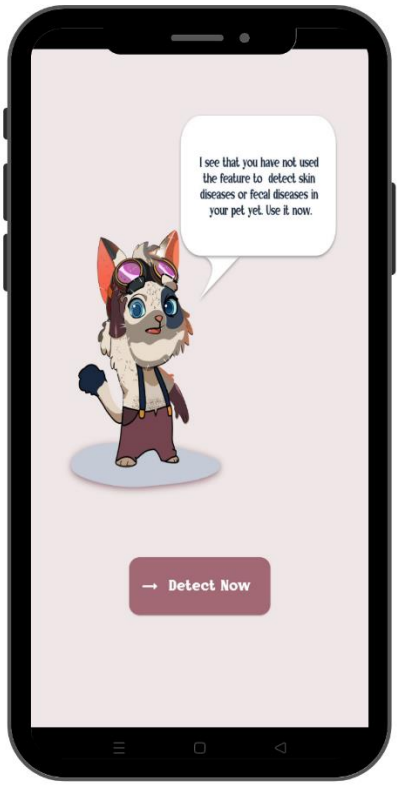
That contains information type of pet

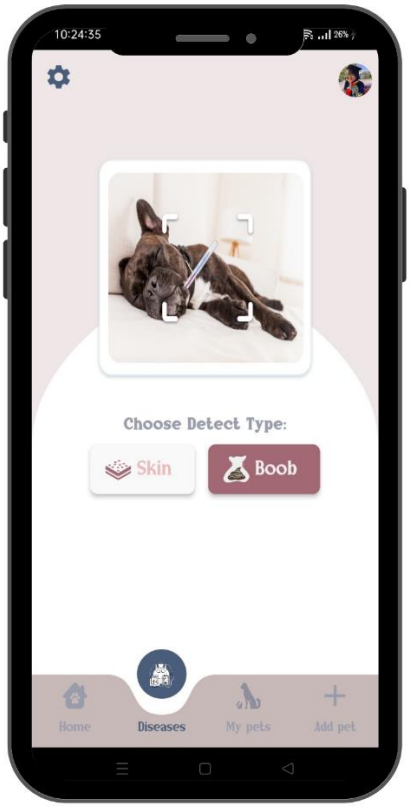
Pet pages



User can add pet manually by entering its information

Detect a disease:

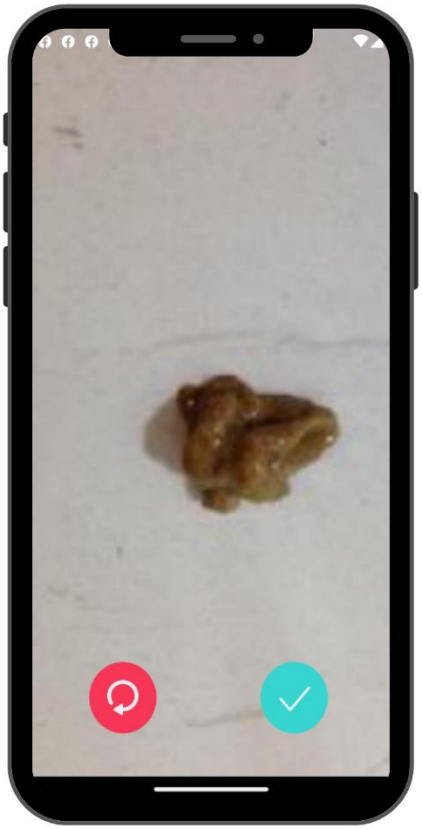


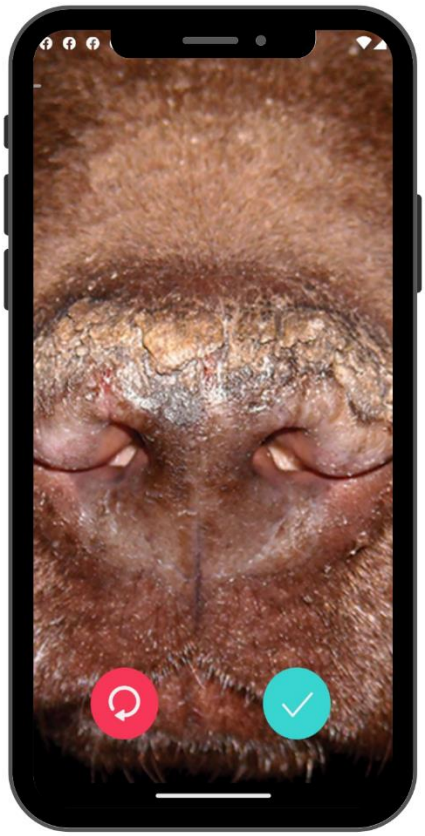


In disease page user click on the button of camera then take a photo

Of the disease

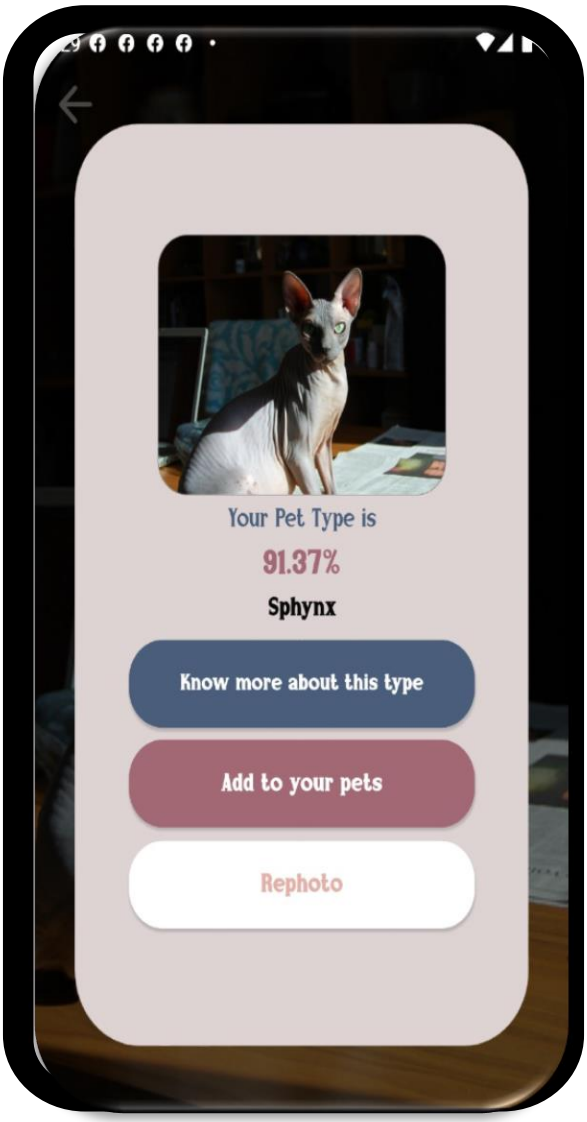
Camera page to detect the diseases:

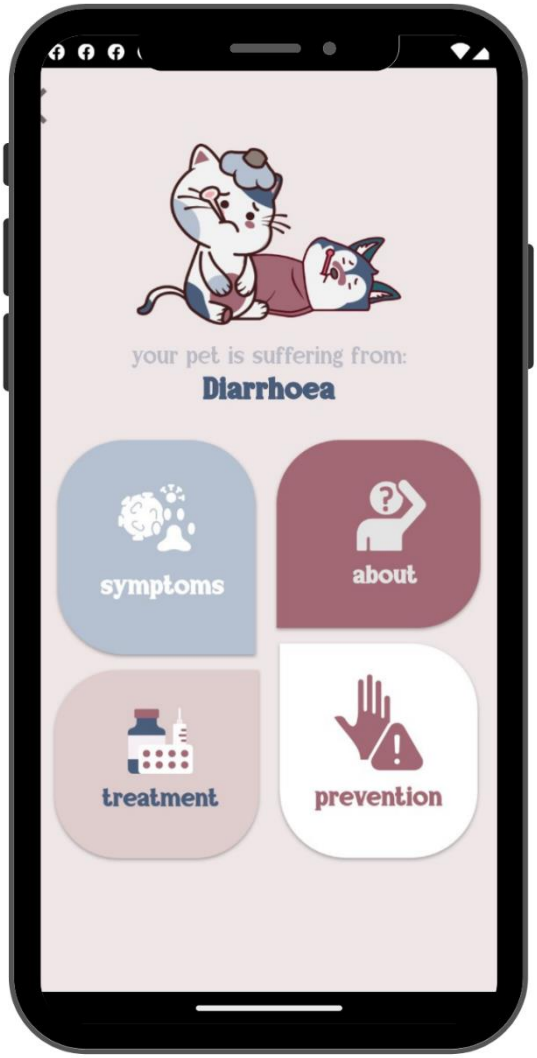




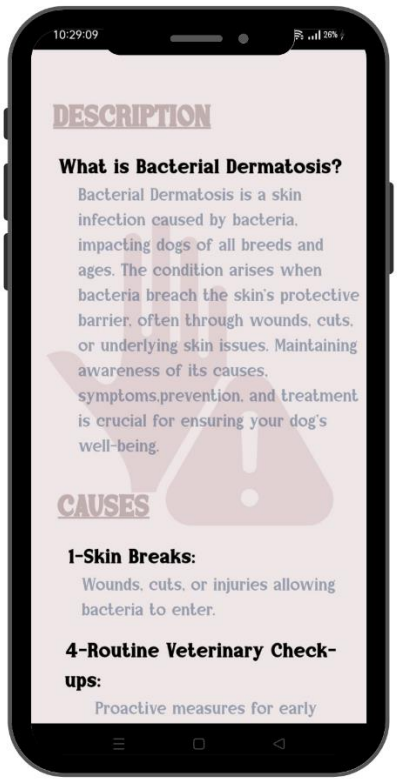
user can take a photo or choose a photo from his gallery

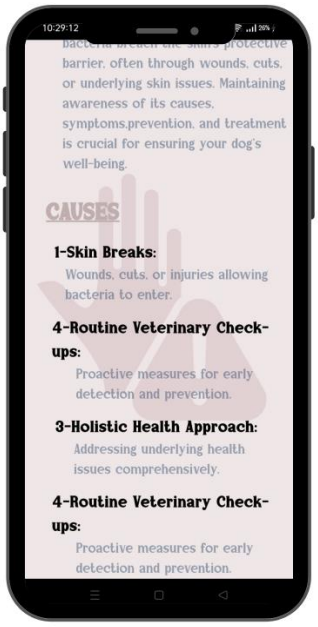
Result of disease detection:





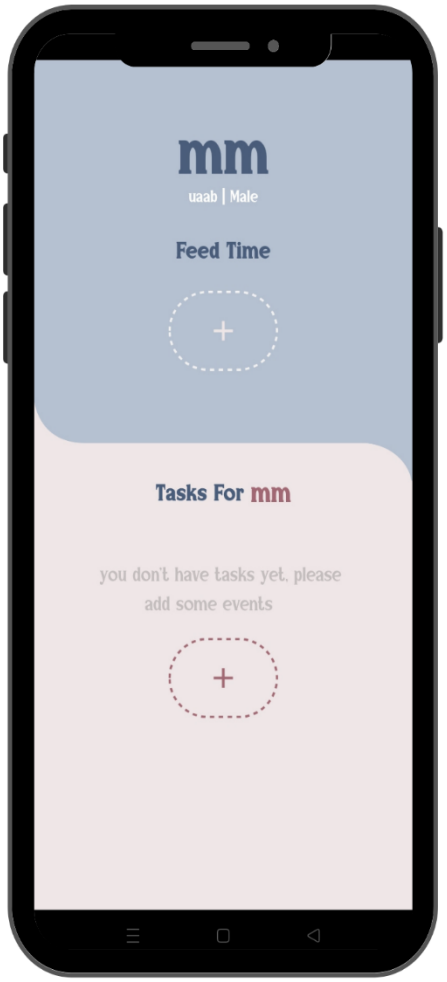
The result contain the name of the disease ,treatment, prevention, description and symptoms of this disease

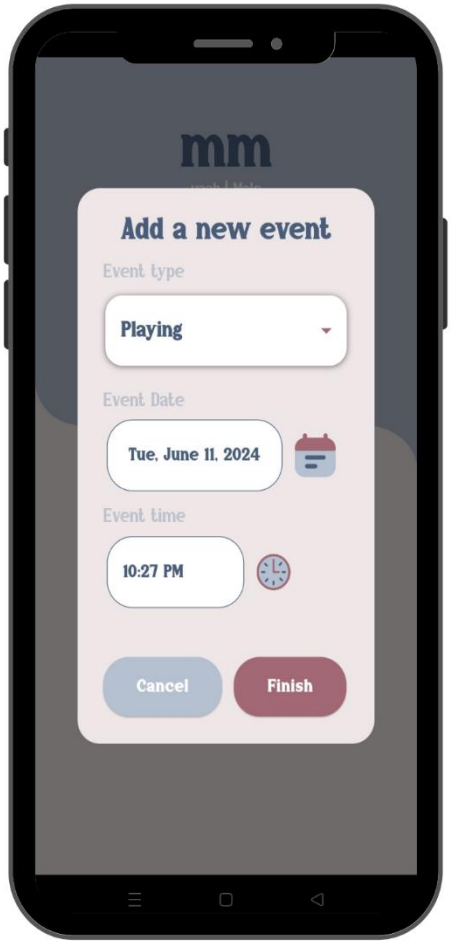




These pages (prevention , symptoms ,treatment, description)help user to know more about pet's disease

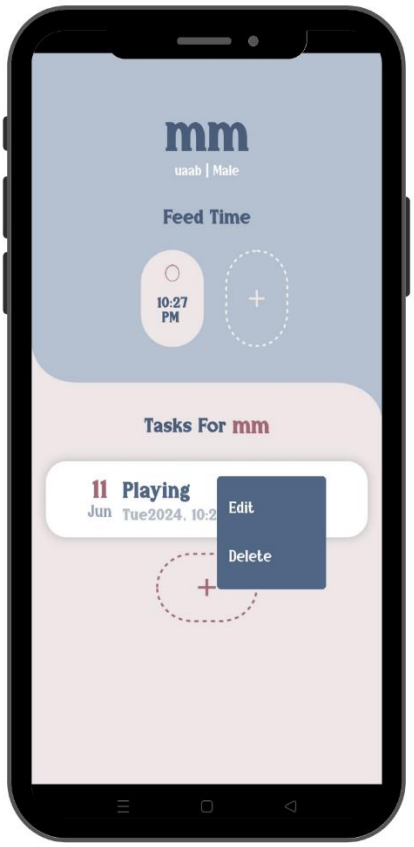
Reminder page:



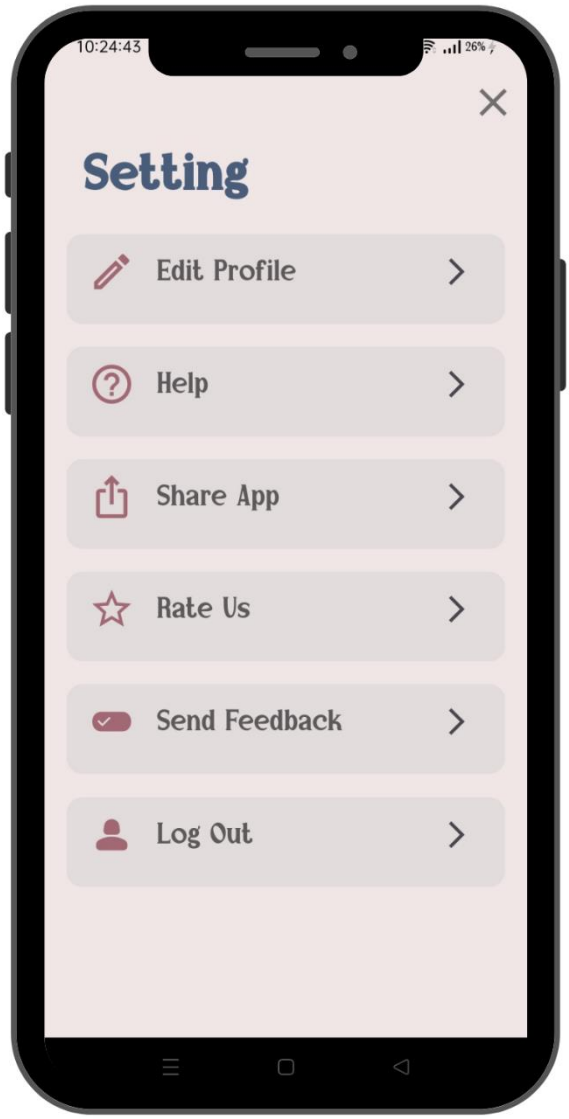


That contains feeds time where user select the time for pet feeding and user can add tasks by adding its name ,its time, and its date from calendar


mm
uaab | Male
Feed Time
Tasks For mm
11 Jun
Playing
Tue2024, 10:27 PM


mm
uaab | Male
Feed Time
10:27 PM
Tasks For mm
11 Jun
Playing
Edit
Delete

Setting page:



The user can edit his profile or edit pet profile also ,there are help page

Coding :

- **Firebase code :**

adding_pet_to_firestore

```dart

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:petapplication/pages/events_system/events_for_pet.dar
t';
class PetsInformation {
  PetsInformation({
    this.petIsDogOrCat,
    required this.imageUrl,
    required this.petName,
    required this.petGender,
    required this.petId,
    required this.petType,
    this.age,
    this.petWeight,
  });

  late String imageUrl;
  late String petName;
  late String petGender;
  late String petId;
  late String petType;
  late String? age;
  late String? petIsDogOrCat;
  bool selected = false;
  String skinDiseaseType = '';
  String poopDiseaseType = '';
  List<CustomTime> feedTimesForPet = [];
  List<ReminderData> remindersData = [];
  double? petWeight;
  factory PetsInformation.fromFirestore(DocumentSnapshot doc) {
    Map data = doc.data() as Map<String, dynamic>;
    return PetsInformation(
      imageUrl: data['imageUrl'] ?? '',
      petName: data['petName'] ?? '',
      petGender: data['petGender'] ?? '',
      petId: data['petId'] ?? '',
      petType: data['petType'] ?? '',
      age: data['age'] ?? '',
      petIsDogOrCat: data['petIsDogOrCat'] ?? '',
    );
  }

  Map<String, dynamic> toFirestore() {
    return {
      'imageUrl': imageUrl,
      'petName': petName,
      'petGender': petGender,
      'petId': petId,
      'petType': petType,
      'age': age,
      'petIsDogOrCat': petIsDogOrCat,
    };
  }
}
```

```dart

Future<void> addPetInFireStore({
  required PetsInformation pet,
}) async {
  try {
    // Get the current user
    User? user = FirebaseAuth.instance.currentUser;

    if (user == null) {
      throw Exception('No user is signed in');
    }
// Upload pet image and get the download URL

    // Reference to the Firestore collection
    CollectionReference petsCollection =
        FirebaseFirestore.instance.collection('pets');

    // Add the pet document with a generated ID
    DocumentReference documentRef = await petsCollection.add({
      'petName': pet.petName,
      'petGender': pet.petGender,
      'petType': pet.petType,
      'age': pet.age,
      'petIsDogOrCat': pet.petIsDogOrCat,
      'petWeight': pet.petWeight,
      'imageUrl': pet.imageUrl,
      'userId': user.uid, // Add current user's ID
    });

    // Update the pet object with the generated ID
    pet.petId = documentRef.id;

    // Optionally, you can update the document with the ID if neede
d   await documentRef.update({'petId': pet.petId});

    print('Pet added with ID: ${pet.petId}');
  } catch (e) {
    print('Error adding pet to Firestore: $e');
 }
}

// fetch data
// fetch data
Future<List<PetsInformation>> fetchUserPets() async {
  try {
    // Get the current user
    User? user = FirebaseAuth.instance.currentUser;

    if (user == null) {
      throw Exception('No user is signed in');
    }

    // Query Firestore collection to fetch pets of the current user
    QuerySnapshot querySnapshot = await FirebaseFirestore.instance
        .collection('pets')
        .where('userId', isEqualTo: user.uid)
        .get();

    // Map each document to a PetsInformation object
    List<PetsInformation> petsList = querySnapshot.docs.map((doc) {
      return PetsInformation.fromFirestore(doc);
    }).toList();

    return petsList;
  } catch (e) {
    print('Error fetching pets: $e');
    return [];
 }
}

```

The PetsInformation class represents information about a pet. The constructor initializes these fields. Fields marked as required must be provided during object instantiation. Optional fields like age and petIsDogOrCat can be null.
The toFirestore method converts a PetsInformation object to a map suitable for Firestore.

The addPetInFireStore function adds a PetsInformation object to Firestore. It first checks if a user is signed in. If not, it throws an exception. Then it adds the pet's data to the pets collection and updates the pet's ID with the generated document ID.

The fetchUserPets function fetches all pets associated with the currently signed-in user from the Firestore pets collection. It maps each document to a PetsInformation object and returns a list of these objects. If there's an error, it prints the error and returns an empty list.

Feeds_ApI.dart :

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

class CustomTime {
  CustomTime({
    required this.hours,
    required this.minutes,
    required this.night,
    required this.checked,
    required this.feedId,
    required this.petId,
    required this.timeOfCreation,
  });
  late String hours;
  late String minutes;
  late String night;
  late String feedId;
  late String petId;
  late Timestamp timeOfCreation;
  bool checked = false;
}

class FeedTimesApi {
  Future<String?> addFeedInFireStore({
    required TimeOfDay timeOfDay,
    required petId,
  }) async {
    try {
      // Get the current user
      User? user = FirebaseAuth.instance.currentUser;

      if (user == null) {
        throw Exception('No user is signed in');
      }

      // Reference to the Firestore collection
      CollectionReference remindersCollection =
          FirebaseFirestore.instance.collection('feedTimes');
      // Get the current time
      Timestamp currentTime = Timestamp.now();

      // Add the pet document with a generated ID
      DocumentReference documentRef = await remindersCollection.add(
        {
          'feed-hour': timeOfDay.hour,
          'feed-minute': timeOfDay.minute,
          'pet-id': petId,
          'user-id': user.uid,
          'checked': "false",
          'feedId': '',
          'timeOfCreation': currentTime,
        },
      );
      //to can edit on this reminder
      final feedId = documentRef.id;
      await documentRef.update({'feedId': feedId});
      return feedId;
    } catch (e) {
      rethrow;
    }
  }

  CustomTime? fromFirestore(DocumentSnapshot doc) {
    try {
      Map data = doc.data() as Map<String, dynamic>;

      int hour = data['feed-hour'];
      int minute = data['feed-minute'];
      TimeOfDay time = TimeOfDay(hour: hour, minute: minute);

      return CustomTime(
        hours: time.hourOfPeriod.toString().padLeft(2, '0'),
        minutes: minute.toString().padLeft(2, '0'),
        night: time.hour < 12 ? 'AM' : 'PM',
        petId: data['pet-id'],
        checked: data['checked'],
        feedId: data['feedId'],
        timeOfCreation: data['timeOfCreation'],
      );
    } catch (error) {
      rethrow;
    }
  }

  CustomTime createFeedTime(
      {required TimeOfDay reminderTime,
      required BuildContext context,
      required String petId}) {
    // Extracting date components

    // Extracting time components
    final String hours = reminderTime.hourOfPeriod.toString().padLeft(2, '0');
    final String period = reminderTime.hour < 12 ? 'AM' : 'PM';
    final String minutes = reminderTime.minute.toString().padLeft(2, '0');

    return CustomTime(
      hours: hours,
      minutes: minutes,
      night: period,
      checked: false,
      feedId: '',
      petId: petId,
      timeOfCreation: Timestamp.now(),
    );
  }

  Future<List<CustomTime>> getFeedTimes({
    required String petId,
  }) async {
    try {
      // Get the current user
      User? user = FirebaseAuth.instance.currentUser;

      if (user == null) {
        throw Exception('No user is signed in');
      }

      // Get the current time
      Timestamp currentTime = Timestamp.now();

      // Query the 'feedTimes' collection for the current user's pet
      QuerySnapshot querySnapshot = await FirebaseFirestore.instance
          .collection('feedTimes')
          .where('user-id', isEqualTo: user.uid)
          .where('pet-id', isEqualTo: petId)
          .get();

      List<CustomTime> feeds = querySnapshot.docs.isNotEmpty
          ? querySnapshot.docs.map((doc) => fromFirestore(doc)).toList()
              as List<CustomTime>
          : [];

      // List to hold feeds created within the last 24 hours
      List<CustomTime> recentFeeds = [];

      for (CustomTime time in feeds) {
        Timestamp timeOfCreation = time.timeOfCreation;
        DateTime creationDateTime = timeOfCreation.toDate();
        DateTime currentDateTime = currentTime.toDate();

        // Calculate the difference between the current time and the creation time
        Duration difference = currentDateTime.difference(creationDateTime);

        if (difference.inHours < 24) {
          // If the feed was created within the last 24 hours, add it to the recentFeeds lis
t         recentFeeds.add(time);
        } else {
          // Otherwise, delete the document from the collection
          await FirebaseFirestore.instance
              .collection('feedTimes')
              .doc(time.feedId) // Assuming `time.id` holds the document ID
              .delete();
        }
      }

      // Return the list of recent feeds
      return recentFeeds;
    } catch (e) {
      rethrow;
    }
  }

  Future<void> deleteFeedTimes(
      {required List<String> selectedItems, required petId}) async {
    try {
      User? user = FirebaseAuth.instance.currentUser;

      if (user == null) {
        throw Exception('No user is signed in');
      }
      for (String feedId in selectedItems) {
        await FirebaseFirestore.instance
            .collection('reminders')
            .doc(feedId)
            .delete();
      }
      print("Deleted feeds successfully");
    } catch (error) {
      print("Error on delete feeds: $error");
      rethrow;
    }
  }
}
```

The CustomTime class represents a custom time structure for a pet's feeding time with the following properties:

- hours: The hour part of the feeding time.
- minutes: The minute part of the feeding time.
- night: AM/PM indicator.
- feedId: ID of the feeding time entry.
- petId: ID of the pet.
- timeOfCreation: Timestamp of when the feeding time was created.
- checked: Boolean indicating if the feeding time is marked as completed (default is false).

This method addFeedInFireStor:

1. Gets the currently signed-in user.
2. References the feedTimes collection in Firestore.
3. Adds a new document to the collection with the feed time details.
4. Updates the document with the generated ID.
5. Returns the generated feed ID.

CustomTime? fromFirestore(DocumentSnapshot doc) : This method converts a Firestore document snapshot into a CustomTime object.

CustomTime createFeedTime: This method creates a new CustomTime object based on the given time of day and pet ID.

Future<List<CustomTime>> getFeedTimes : This method retrieves feed times for a specific pet created in the last 24 hours. It:

1. Gets the currently signed-in user.
2. Queries the feedTimes collection for documents matching the user ID and pet ID.
3. Converts the documents into CustomTime objects.
4. Filters the feed times created within the last 24 hours.
5. Deletes feed times older than 24 hours.
6. Returns the list of recent feed times.

** Future<void> deleteFeedTimes:

This method deletes specific feed times based on their IDs. It:

1. Gets the currently signed-in user.
2. Iterates over the list of selected feed IDs.
3. Deletes each corresponding document from the reminders collection.
4. Logs the deletion success or error.

Reminders_api.dart

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import 'package:petapplication/pages/events_system/events_for_pet.dart';
import 'package:petapplication/some_files_to_data/adding_pet_to_firestore.dar
t';
class MergedReminderData {
  MergedReminderData({
    required this.date,
    required this.remindertime,
  });
  final TimeOfDay remindertime;
  final DateTime date;
  TimeOfDay getReminderTime() {
    return remindertime;
 }

  DateTime getrReminderDate() {
    return date;
 }
}

class ReminderDataApi {
  Future<String?> addReminderInFireStore({
    required DateTime selectedDate,
    required TimeOfDay reminderTime,
    required String reminderType,
    required String petId,
  }) async {
    try {
      User? user = FirebaseAuth.instance.currentUser;

      if (user == null) {
        throw Exception('No user is signed in');
     }

      CollectionReference remindersCollection =
          FirebaseFirestore.instance.collection('reminders');

      DocumentReference documentRef = await remindersCollection.add({
        'reminder-date': selectedDate.toString(),
        'reminder-title': reminderType,
        'reminder-hour': reminderTime.hour,
        'reminder-minute': reminderTime.minute,
        'user-id': user.uid,
        'pet-id': petId,
        'reminder-id': '', // Temporarily set to empty
      });

      final reminderId = documentRef.id;
      await documentRef.update({'reminder-id': reminderId});
      print('Reminder added with ID: $reminderId');
      return reminderId;
    } catch (e) {
      print('Error adding reminder: $e');
      rethrow;
   }
 }

  Future<ReminderData?> createReminderData(
      DateTime selectedDate,
      TimeOfDay reminderTime,
      String currentItemSelected,
      PetsInformation pet) async {
    // Extracting date components
    final String day = selectedDate.day.toString();
    final String month = DateFormat('MMM').format(selectedDate);
    final String year = selectedDate.year.toString();
    final String weekDay = DateFormat('E').format(selectedDate);
    ReminderData returnedReminder;
    // Extracting time components
    final String hours = reminderTime.hourOfPeriod.toString().padLeft(2, '0');
    final String period = reminderTime.hour < 12 ? 'AM' : 'PM';
    final String minutes = reminderTime.minute.toString().padLeft(2, '0');
    returnedReminder = ReminderData(
      day: day,
      month: month,
      reminderType: currentItemSelected,
      hours: hours,
      minutes: minutes,
      night: period,
      weekDay: weekDay,
      year: year,
      petId: pet.petId,
      reminderId: '',
      monthNumber: selectedDate.month,
    );

    return returnedReminder;
 }

```

The MergedReminderData class represents a combined data structure for a reminder, containing:

- remindertime: The time of the reminder (as TimeOfDay).
- date: The date of the reminder (as DateTime).

It also includes methods to retrieve the reminder time and date:

- getReminderTime(): Returns the remindertime.
- getrReminderDate(): Returns the date.

ReminderDataApi Class

The ReminderDataApi class provides methods to interact with Firestore for managing reminders.

This method adds a new reminder to Firestore:

1. Retrieves the currently signed-in user.
2. References the reminders collection in Firestore.
3. Adds a new document with the reminder details (date, time, type, user ID, pet ID).
4. Updates the document with the generated reminder ID.
5. Returns the reminder ID.

- createReminderData :

This method creates a ReminderData object with detailed information about a reminder:

1. Extracts the day, month, year, and weekday from the selected date.
2. Extracts the hour, minute, and AM/PM period from the reminder time.
3. Creates and returns a ReminderData object with the extracted information and the provided reminder type and pet ID.

```dart

  ReminderData? fromFirestore(DocumentSnapshot doc) {
    try {
      Map data = doc.data() as Map<String, dynamic>;

      DateTime date = DateTime.parse(data['reminder-date']);
      int hour = data['reminder-hour'];
      int minute = data['reminder-minute'];
      TimeOfDay time = TimeOfDay(hour: hour, minute: minute);

      return ReminderData(
        day: date.day.toString(),
        month: DateFormat('MMM').format(date),
        reminderType: data['reminder-title'],
        hours: time.hourOfPeriod.toString().padLeft(2, '0'),
        minutes: minute.toString().padLeft(2, '0'),
        night: time.hour < 12 ? 'AM' : 'PM',
        weekDay: DateFormat('EEE').format(date),
        year: date.year.toString(),
        petId: data['pet-id'],
        reminderId: data['reminder-id'],
        monthNumber: date.month,
      );
    } catch (error) {
      rethrow;
    }
  }

  Future<void> updateReminder({
    required reminderId,
    required selectedDate,
    required TimeOfDay reminderTime,
    required reminderType,
    required petId,
  }) async {
    try {
      User? user = FirebaseAuth.instance.currentUser;

      if (user == null) {
        throw Exception('No user is signed in');
      }
      await FirebaseFirestore.instance
          .collection('reminders')
          .where('user-id', isEqualTo: user.uid)
          .where('pet-id', isEqualTo: petId)
          .where('pet-id', isEqualTo: reminderId)
          .get()
          .then((_) {
        FirebaseFirestore.instance
            .collection('reminders')
            .doc(reminderId)
            .update({
          'reminder-date': selectedDate.toString(),
          'reminder-title': reminderType,
          'reminder-hour': reminderTime.hour,
          'reminder-minute': reminderTime.minute,
        });
      }).catchError((error) {
        throw error;
      });
    } catch (error) {
      rethrow;
    }
  }

  Future<List<ReminderData?>> fetchRemindersDatafromFirestore({
    required String petId,
  }) async {
    try {
      User? user = FirebaseAuth.instance.currentUser;

      if (user == null) {
        throw Exception('No user is signed in');
      }
      QuerySnapshot querySnapshot = await FirebaseFirestore.instance
          .collection('reminders')
          .where('user-id', isEqualTo: user.uid)
          .where('pet-id', isEqualTo: petId)
          .get();

      if (querySnapshot.docs.isNotEmpty) {
        List<ReminderData?> remindersList = querySnapshot.docs.map((doc) {
          return fromFirestore(doc);
        }).toList();

        return remindersList;
      } else {
        return [];
      }
    } on FirebaseException {
      rethrow;
    }
  }

  Future<void> deleteReminder(
      {required String reminderId, required String petId}) async {
    if (reminderId.isEmpty) {
      throw Exception('reminderId cannot be empty');
    }

    try {
      User? user = FirebaseAuth.instance.currentUser;

      if (user == null) {
        throw Exception('No user is signed in');
      }

      await FirebaseFirestore.instance
          .collection('reminders')
          .doc(reminderId)
          .delete();

      print("Deleted reminder successfully");
    } catch (error) {
      print("Error deleting reminder: $error");
      rethrow;
    }
  }

  MergedReminderData mergedReminderData(ReminderData reminder) {
    DateTime date = DateTime(
      int.parse(reminder.year),
      reminder.monthNumber,
      int.parse(reminder.day),
    );
    int hour = int.parse(reminder.hours);
    int minute = int.parse(reminder.hours);
    TimeOfDay time = TimeOfDay(hour: hour, minute: minute);
    return MergedReminderData(date: date, remindertime: time);
  }
}
```

The MergedReminderData class combines the reminder time (TimeOfDay) and date (DateTime) into a single object. It provides getter methods to retrieve the reminder time and date.

ReminderDataApi Class

The ReminderDataApi class contains methods to interact with Firestore for managing reminders.

fromFirestore Method

This method converts a Firestore document (DocumentSnapshot) into a ReminderData object. It extracts and formats the necessary data from the document.

updateReminder Method

This method updates an existing reminder in Firestore with the provided details (date, time, type, and pet ID).

fetchRemindersDatafromFirestore Method

This method fetches all reminders for a specific pet from Firestore. It queries the reminders collection and converts each document to a ReminderData object.

deleteReminder Method

This method deletes a specific reminder from Firestore using its ID.

mergedReminderData Method

This method converts a ReminderData object into a MergedReminderData object by extracting and formatting the date and time.

Firebase_options.dart

```dart
// File generated by FlutterFire CLI.
// ignore_for_file: type=lint
import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;


class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      return web;
   }
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;
      case TargetPlatform.iOS:
        return ios;
      case TargetPlatform.macOS:
        return macos;
      case TargetPlatform.windows:
        return windows;
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
        );
    }
  }

  static const FirebaseOptions web = FirebaseOptions(
    apiKey: 'AIzaSyDMhMEmPWB7IlC7oVrCOjMrRRWHY7_JP4M',
    appId: '1:799923700168:web:dae71da0ecf695ac3600fd',
    messagingSenderId: '799923700168',
    projectId: 'petapplication-e28d2',
    authDomain: 'petapplication-e28d2.firebaseapp.com',
    storageBucket: 'petapplication-e28d2.appspot.com',
    measurementId: 'G-J43SZXMW2T',
  );

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyApGEmExepBmFDXOiQ8GPBLtv1Mlzzl5bc',
    appId: '1:799923700168:android:221c24043d4733fe3600fd',
    messagingSenderId: '799923700168',
    projectId: 'petapplication-e28d2',
    storageBucket: 'petapplication-e28d2.appspot.com',
  );

  static const FirebaseOptions ios = FirebaseOptions(
    apiKey: 'AIzaSyDTrhbZiLY-hyPJ2s6HQA4hk4bCSxWu0qU',
    appId: '1:799923700168:ios:75515ff1c8598f303600fd',
    messagingSenderId: '799923700168',
    projectId: 'petapplication-e28d2',
    storageBucket: 'petapplication-e28d2.appspot.com',
    iosClientId: '799923700168-0uh6ur3ivtkhr07vc71csc516b6tg77u.apps.googleusercontent.co
m', iosBundleId: 'com.example.petapplication',
  );

  static const FirebaseOptions macos = FirebaseOptions(
    apiKey: 'AIzaSyDTrhbZiLY-hyPJ2s6HQA4hk4bCSxWu0qU',
    appId: '1:799923700168:ios:75515ff1c8598f303600fd',
    messagingSenderId: '799923700168',
    projectId: 'petapplication-e28d2',
    storageBucket: 'petapplication-e28d2.appspot.com',
    iosClientId: '799923700168-0uh6ur3ivtkhr07vc71csc516b6tg77u.apps.googleusercontent.co
m', iosBundleId: 'com.example.petapplication',
  );

  static const FirebaseOptions windows = FirebaseOptions(
    apiKey: 'AIzaSyDMhMEmPWB7IlC7oVrCOjMrRRWHY7_JP4M',
    appId: '1:799923700168:web:889245190320ab2c3600fd',
    messagingSenderId: '799923700168',
    projectId: 'petapplication-e28d2',
    authDomain: 'petapplication-e28d2.firebaseapp.com',
    storageBucket: 'petapplication-e28d2.appspot.com',
    measurementId: 'G-T3R3T202QT',
  );

}
```

DefaultFirebaseOptions Class

The currentPlatform getter determines the current platform the application is running on and returns the corresponding FirebaseOptions object. If the platform is not supported (like Linux in this example), it throws an UnsupportedError.

Platform-Specific Firebase Options

For each platform, there is a constant FirebaseOptions object containing the necessary configuration details like apiKey, appId, projectId, and other identifiers.

- This configuration file ensures that the correct Firebase configuration is used based on the platform the Flutter application is running on. This setup is essential for initializing Firebase services like authentication, Firestore, messaging, and others within the application. By abstracting platform-specific configurations into a single class, it makes the codebase cleaner and more maintainable.

Uploading image for user

```dart
import 'dart:io';

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

class FirebaseApiForUserImage {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseStorage _storage = FirebaseStorage.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  User? user = FirebaseAuth.instance.currentUser;
  Future<String?> uploadingImageOnFirebase(
      File? uploadedImage, BuildContext context) async {
    if (user == null || uploadedImage == null) {
      return null;
    }

    try {
      final storageRef = _storage
          .ref()
          .child('users')
          .child(user!.uid)
          .child('profile_photos')
          .child('${user!.uid}.png');

      try {
        // Delete the previous image if it exists
        await storageRef.delete();
      } catch (e) {
        // If the image doesn't exist, continue without throwing an error
      }

      // Upload the new image
      await storageRef.putFile(uploadedImage);
      final imageUrl = await storageRef.getDownloadURL();

      return imageUrl;
    } on FirebaseException catch (e) {
      ScaffoldMessenger.of(context).clearSnackBars();
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(e.toString()),
          action: SnackBarAction(
              label: 'Close',
              onPressed: () {
                ScaffoldMessenger.of(context).hideCurrentSnackBar();
              }),
        ),
      );
      return null;
    }
  }

  Future<String?> getProfileImage(BuildContext context) async {
    try {
      final user = _auth.currentUser;
      if (user == null) {
        return null;
      }
      final doc = await _firestore.collection('users').doc(user.uid).get();

      return doc.data()?['profile_image'];
    } on FirebaseException {
      ScaffoldMessenger.of(context).clearSnackBars();
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: const Text(
              'Failed to load your profile photo, please try again later.'),
          action: SnackBarAction(
              label: 'Close',
              onPressed: () {
                ScaffoldMessenger.of(context).hideCurrentSnackBar();
              }),
        ),
      );
      return null;
    }
  }
}

```

The FirebaseApiForUserImage class encapsulates methods to handle user profile images in a Firebase environment:

- uploadingImageOnFirebase: Uploads a user's profile image to Firebase Storage.
- getProfileImage: Retrieves the URL of the user's profile image from Firestore.

These methods also handle error cases by displaying snack bar messages using the provided BuildContext. This class ensures that user profile images are managed securely and efficiently within a Flutter application using Firebase services.

uploading_user_information_to_firestore

```dart
import 'package:firebase_auth/firebase_auth.dart';

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

void uploadingUserInformationTofireStore(
    {String? uploadedImage, required BuildContext context}) async {
  User? user = FirebaseAuth.instance.currentUser;
  if (user == null) {
    return null;
  }

  try {
    await FirebaseFirestore.instance.collection('users').doc(user.uid).set({
      'profile_image': uploadedImage,
      'phone_number': user.phoneNumber,
      'email': user.email,
      'user_id': user.uid,
      'user_name': user.displayName,
    });
  } on FirebaseException {
    ScaffoldMessenger.of(context).clearSnackBars();
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: const Text(
            'Failed to update your profile photo, please try again later....'),
        action: SnackBarAction(
            label: 'Close',
            onPressed: () {
              ScaffoldMessenger.of(context).hideCurrentSnackBar();
            }),
      ),
    );
    return null;
  }
}

void uploadingUserInformationTofireStoreWithManualUploading({
  String? uploadedImage,
  required BuildContext context,
  String? displayName,
  String? phoneNumber,
}) async {
  User? user = FirebaseAuth.instance.currentUser;
  if (user == null) {
    return;
  }

  try {
    await FirebaseFirestore.instance.collection('users').doc(user.uid).set({
      'profile_image': uploadedImage,
      'phone_number': phoneNumber,
      'email': user.email,
      'user_id': user.uid,
      'user_name': displayName,
    });
  } on FirebaseException {
    ScaffoldMessenger.of(context).clearSnackBars();
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: const Text(
            'Failed to update your profile photo, please try again later....'),
        action: SnackBarAction(
            label: 'Close',
            onPressed: () {
              ScaffoldMessenger.of(context).hideCurrentSnackBar();
            }),
      ),
    );
  }
}

```

These functions are responsible for updating user information in Firestore:

- uploadingUserInformationToFirestore: Updates user information with the uploaded profile image, and uses the current user's phone number and display name.
- uploadingUserInformationToFirestoreWithManualUploading: Allows for manual updating of user information with specific values for profile image, phone number, and display name.

Both functions handle errors gracefully by displaying snack bar messages in the provided BuildContext. This ensures that user data is updated accurately and that users are notified of any errors that occur during the update process.

User profile:

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

class UserAcount extends StatefulWidget {
  const UserAcount({
    super.key,
  });

  @override
  State<UserAcount> createState() => _UserAcountState();
}

class _UserAcountState extends State<UserAcount> {
  TextEditingController nameController = TextEditingController();
  TextEditingController emailController = TextEditingController();
  TextEditingController phoneNumberController = TextEditingController();
  User? userInfo = FirebaseAuth.instance.currentUser;
  Future<void> _initializeUserData() async {
    try {
      final doc = await FirebaseFirestore.instance
          .collection('users')
          .doc(userInfo!.uid)
          .get();
      if (userInfo != null) {
        nameController.text =
            userInfo!.displayName ?? doc.data()?['user_name'] ?? '';
        emailController.text = userInfo!.email ?? '';
        phoneNumberController.text =
            userInfo!.phoneNumber ?? doc.data()?['phone_number'] ?? '';
     }
    } on FirebaseException {
      ScaffoldMessenger.of(context).clearSnackBars();
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: const Text(
              'Failed to load your profile informations, please check your connection and try again later.'),
          action: SnackBarAction(
              label: 'Close',
              onPressed: () {
                ScaffoldMessenger.of(context).hideCurrentSnackBar();
              }),
        ),
      );
    }
  }

```

- The UserAcount widget is a StatefulWidget that manages the UI for displaying and updating user account information.
- The _UserAcountState manages the state of the widget.
- The _initializeUserData function fetches user data from Firestore and updates the UI accordingly.
- initState is used to initialize user data when the widget is first created.
- dispose is used to release resources when the widget is destroyed.
- build method constructs the UI using TextFormField widgets for user

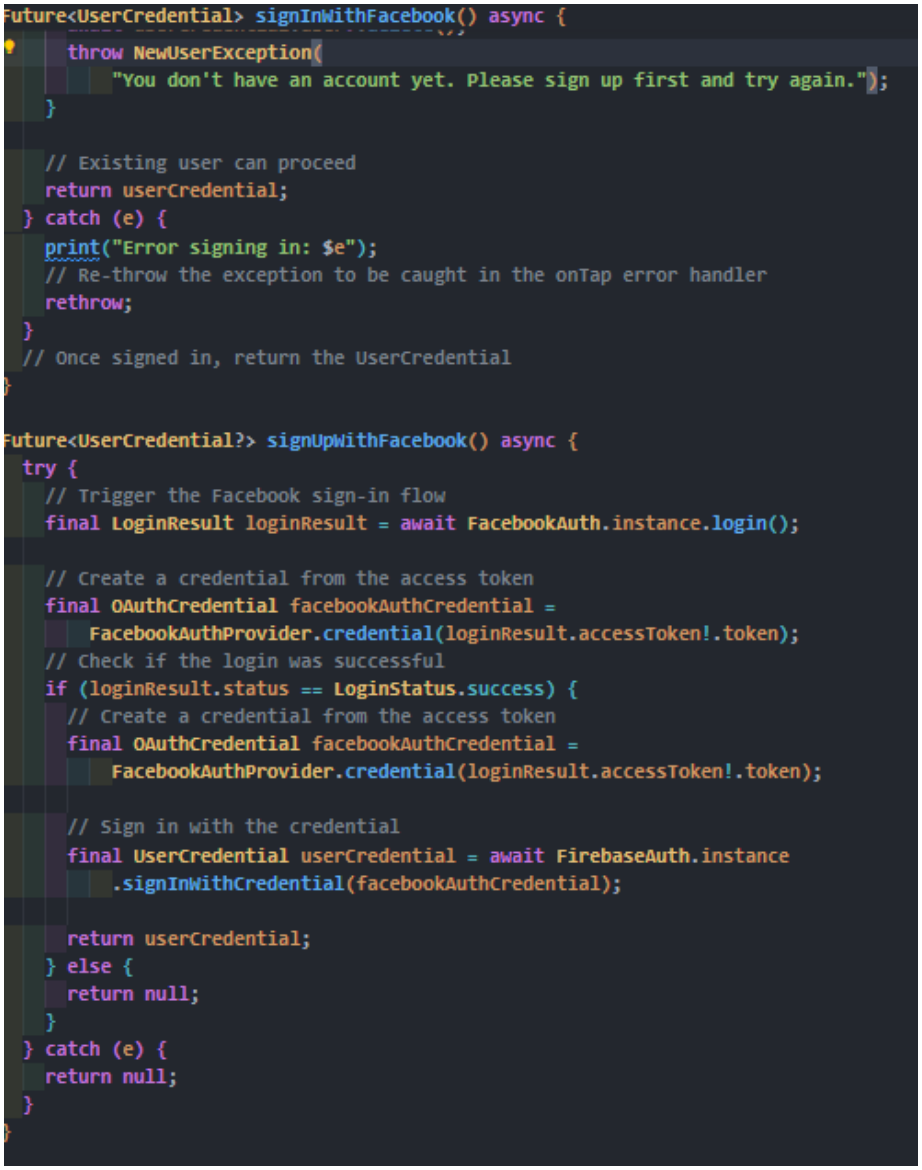
- input and an ElevatedButton for updating user information.

  This widget is designed to manage and update user account information in a Flutter application using Firebase Authentication and Firestore.

Auth_app.dart

```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:flutter_facebook_auth/flutter_facebook_auth.dart';

class NewUserException implements Exception {
  final String message;
  NewUserException(this.message);

  @override
  String toString() => message;
}

final GoogleSignIn googleSignIn = GoogleSignIn();
Future<UserCredential?> signUpWithGoogle() async {
  // Check if the user is already signed in
  GoogleSignInAccount? googleUser = GoogleSignIn.standard().currentUser;

  // If not, trigger the authentication flow
  googleUser ??= await googleSignIn.signIn();

  // If the user cancelled the sign-in flow, return null
  if (googleUser == null) {
    return null;
  }

  // Obtain the auth details from the request
  final GoogleSignInAuthentication googleAuth = await googleUser.authentication;

  // Create a new credential
  final credential = GoogleAuthProvider.credential(
    accessToken: googleAuth.accessToken,
    idToken: googleAuth.idToken,
  );
```

```dart
Future<UserCredential?> signInWithGoogle() async {
  // Try signing in with the selected account
  try {
    final UserCredential userCredential =
        await FirebaseAuth.instance.signInWithCredential(credential);

    // Check if the user is new
    final isNewUser = userCredential.additionalUserInfo?.isNewUser ?? false;

    // If the user is new, delete their account and throw an exception
    if (isNewUser) {
      await userCredential.user?.delete();
      throw NewUserException(
          "You don't have an account yet. Please sign up first and try again.");
    }

    // Existing user can proceed
    return userCredential;
  } catch (e) {
    print("Error signing in: $e");
    // Re-throw the exception to be caught in the onTap error handler
    rethrow;
  }
}

Future<UserCredential> signInWithFacebook() async {
  // Trigger the sign-in flow
  final LoginResult loginResult = await FacebookAuth.instance.login();

  // Create a credential from the access token
  final OAuthCredential facebookAuthCredential =
      FacebookAuthProvider.credential(loginResult.accessToken!.token);
  try {
    final UserCredential userCredential = await FirebaseAuth.instance
        .signInWithCredential(facebookAuthCredential);

    // Check if the user is new
    final isNewUser = userCredential.additionalUserInfo?.isNewUser ?? false;

    // If the user is new, delete their account and throw an exception
    if (isNewUser) {
      await userCredential.user?.delete();
      throw NewUserException(
```



```
Future<UserCredential> signInWithFacebook() async {
      throw NewUserException(
          "You don't have an account yet. Please sign up first and try again.");
    }

    // Existing user can proceed
    return userCredential;
  } catch (e) {
    print("Error signing in: $e");
    // Re-throw the exception to be caught in the onTap error handler
    rethrow;
  }
  // Once signed in, return the UserCredential
}

Future<UserCredential?> signUpWithFacebook() async {
  try {
    // Trigger the Facebook sign-in flow
    final LoginResult loginResult = await FacebookAuth.instance.login();

    // Create a credential from the access token
    final OAuthCredential facebookAuthCredential =
        FacebookAuthProvider.credential(loginResult.accessToken!.token);
    // Check if the login was successful
    if (loginResult.status == LoginStatus.success) {
      // Create a credential from the access token
      final OAuthCredential facebookAuthCredential =
          FacebookAuthProvider.credential(loginResult.accessToken!.token);

      // Sign in with the credential
      final UserCredential userCredential = await FirebaseAuth.instance
          .signInWithCredential(facebookAuthCredential);

      return userCredential;
    } else {
      return null;
    }
  } catch (e) {
    return null;
  }
}
```

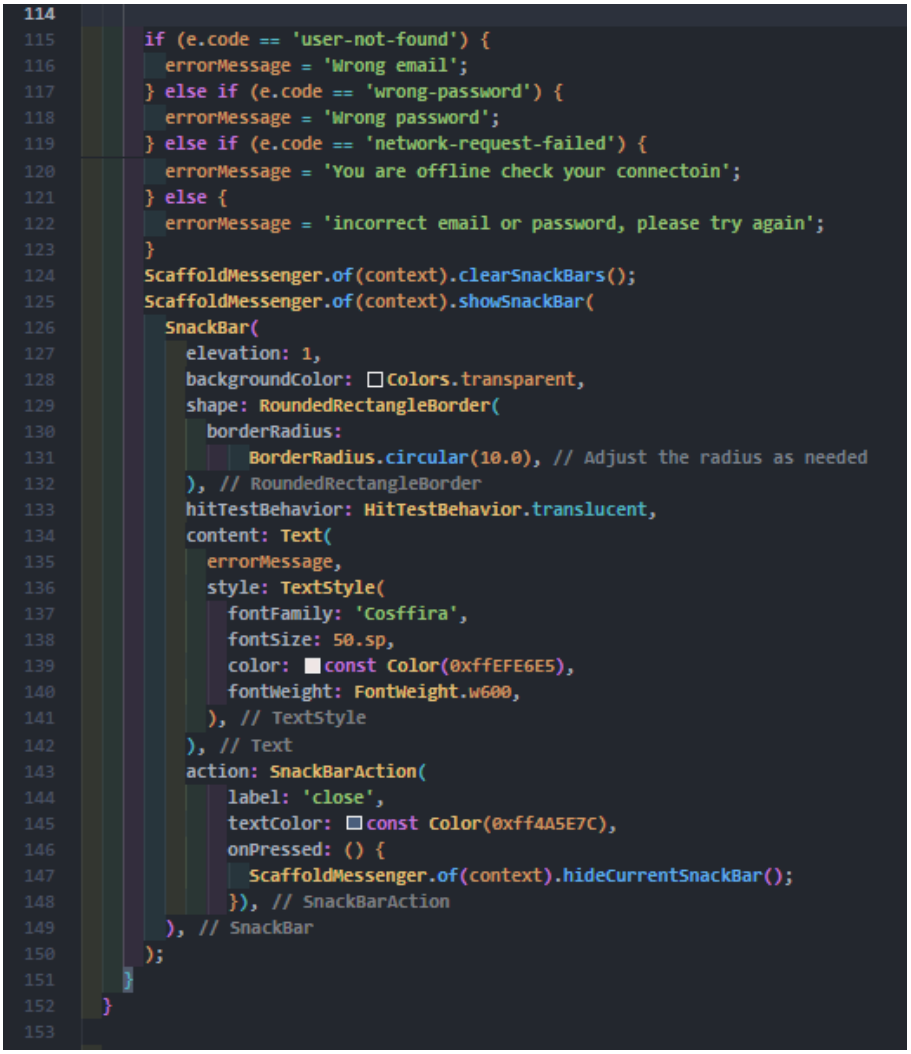
- The code provides functions for signing in and signing up users with Google and Facebook using Firebase Authentication.
- It handles different scenarios, such as existing and new users, and throws a NewUserException when necessary.
- Error handling is implemented using try-catch blocks and rethrows exceptions for further handling.
- The code is organized into functions to make it modular and reusable across a Flutter application.

These functions can be used to integrate Google and Facebook authentication seamlessly into your Flutter app using Firebase. Remember to handle errors and exceptions appropriately in your UI layer based on the needs of your application.

Login_info_email_pass

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

import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:get/get.dart';

import 'package:petapplication/core/utils/widgets/custom_buttom.dart';

import 'package:petapplication/pages/homepage/home_page_with_navigation.dart';
import 'package:petapplication/pages/pageforgetpass/my_verify_pass.dart';

//import 'package:flutter_svg/svg.dart';

class TheMainLoginPage extends StatefulWidget {
  const TheMainLoginPage({super.key});

  @override
  State<TheMainLoginPage> createState() => _TheMainLoginPageState();
}

bool isloading = false;

class _TheMainLoginPageState extends State<TheMainLoginPage> {
  final TextEditingController _email = TextEditingController();
  final TextEditingController _pass = TextEditingController();
  @override
  void dispose() {
    _email.dispose;
    _pass.dispose();
    super.dispose();
  }

  @override
  void initState() {
    isloading = false;
    super.initState();
  }
```

```dart
class _TheMainLoginPageState extends State<TheMainLoginPage> {
  void initState() {
  }

  String? validateEmail(String? value) {
    if (value == null || value.isEmpty) {
      return 'enter your email please';
    } else {
      // Regular expression to validate email format
      const pattern = r"(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'"
          r'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-'
          r'\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*'
          r'[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:(2(5[0-5]|[0-4]'
          r'[0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-9]'
          r'[0-9]|[1-9]?[0-9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\'
          r'x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])';
      final regex = RegExp(pattern);

      if (!regex.hasMatch(value)) {
        return 'Invalid email';
      }
    }
    //Ssoo@121
    return null; // Return null if the email is valid
  }

  // login function
  signInWithEmailAndPassword() async {
    String errorMessage = 'Login Successed';
    try {
      setState(() {
        isloading = true;
      });
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: _email.text,
        password: _pass.text,
      );

      // Navigate to the main homepage upon successful login

      Get.offAll(() => const TheMainHomePage());
      ScaffoldMessenger.of(context).clearSnackBars();

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          backgroundColor: const Color.fromARGB(80, 0, 0, 0),
          elevation: 0,
          content: Text(
            errorMessage,
            style: TextStyle(
              fontFamily: 'Cosffira',
              fontSize: 50.sp,
              color: const Color(0xffEFE6E5),
              fontWeight: FontWeight.w600,
            ), // TextStyle
          ), // Text
          shape: RoundedRectangleBorder(
            borderRadius:
                BorderRadius.circular(10.0), // Adjust the radius as needed
          ), // RoundedRectangleBorder
          action: SnackBarAction(
            label: 'Wellcome',
            textColor:
                const Color(0xff4A5E7C), // Set the color of the close icon
            onPressed: () {
              ScaffoldMessenger.of(context).hideCurrentSnackBar();
            },
          ), // SnackBarAction
          behavior: SnackBarBehavior.floating, // Set the behavior to floating
        ), // SnackBar
      );
    } on FirebaseAuthException catch (e) {
      setState(() {
        isloading = false;
      });

```



```
114
115       if (e.code == 'user-not-found') {
116         errorMessage = 'Wrong email';
117       } else if (e.code == 'wrong-password') {
118         errorMessage = 'Wrong password';
119       } else if (e.code == 'network-request-failed') {
120         errorMessage = 'You are offline check your connectoin';
121       } else {
122         errorMessage = 'incorrect email or password, please try again';
123       }
124       ScaffoldMessenger.of(context).clearSnackBars();
125       ScaffoldMessenger.of(context).showSnackBar(
126         SnackBar(
127           elevation: 1,
128           backgroundColor: Colors.transparent,
129           shape: RoundedRectangleBorder(
130             borderRadius:
131                 BorderRadius.circular(10.0), // Adjust the radius as needed
132           ), // RoundedRectangleBorder
133           hitTestBehavior: HitTestBehavior.translucent,
134           content: Text(
135             errorMessage,
136             style: TextStyle(
137               fontFamily: 'Cosffira',
138               fontSize: 50.sp,
139               color: const Color(0xffEFE6E5),
140               fontWeight: FontWeight.w600,
141             ), // TextStyle
142           ), // Text
143           action: SnackBarAction(
144               label: 'close',
145               textColor: const Color(0xff4A5E7C),
146               onPressed: () {
147                 ScaffoldMessenger.of(context).hideCurrentSnackBar();
148               }), // SnackBarAction
149         ), // SnackBar
150       );
151     }
152   }
153
```

validateEmail**:**

- This method validates the format of the email input using a regular expression.
- It returns a string message if the email is invalid, otherwise, it returns null.

signInWithEmailAndPassword**:**

- This method handles the sign-in process using the provided email and password.
- It updates the isloading flag to true to show loading state.
- It calls signInWithEmailAndPassword method from FirebaseAuth.instance to authenticate user credentials.
- Upon successful authentication, it navigates to the main home page using Get.offAll().
- It shows a snack bar with a success message using ScaffoldMessenger.of(context).showSnackBar() if authentication is successful.

- If authentication fails, it shows a snack bar with an error message indicating the reason for failure.

Page_view_login

```dart

class PageViewLogin extends StatefulWidget {
  final VoidCallback? toggleContainerVisibility;

  const PageViewLogin({super.key, this.toggleContainerVisibility});

  @override
  State<PageViewLogin> createState() => _PageViewLoginState();
}

class _PageViewLoginState extends State<PageViewLogin> {
  String? validateEmail(String? value) {
    if (value == null || value.isEmpty) {
      return 'enter your email please';
    } else {
      // Regular expression to validate email format
      const pattern = r"(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'"
          r'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-'
          r'\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*'
          r'[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:(2(5[0-5]|[0-
4]'       r'[0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-
9]'       r'[0-9]|[1-9]?[0-9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f
\'        r'x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])';
      final regex = RegExp(pattern);

      if (!regex.hasMatch(value)) {
        return 'Invalid email';
      }
    }
    return null; // Return null if the email is valid
  }

  late PageController _pageController;
  TextEditingController myController = TextEditingController();
  final TextEditingController name = TextEditingController();
  final TextEditingController _email = TextEditingController();
  final TextEditingController _password = TextEditingController();
  final TextEditingController _phoneNumber = TextEditingController();
  final _fformKey = GlobalKey<FormState>();
  @override
  void initState() {
    super.initState();
    _pageController = PageController();
  }

  @override
  void dispose() {
    myController.dispose();
    super.dispose();
  }

  bool _obscureText = true;
  registration(BuildContext context, String? image) async {
    try {
      UserCredential userCredential = await FirebaseAuth.instance
          .createUserWithEmailAndPassword(
              email: _email.text, password: _password.text);

      uploadingUserInformationTofireStoreWithManualUploading(
          displayName: name.text,
          phoneNumber: _phoneNumber.text,
          context: context,
          uploadedImage: image);
```

- PageViewLogin is a stateful widget that represents a login page.
- It takes an optional callback function toggleContainerVisibility which can be used to toggle the visibility of a container or any other UI element.
- It overrides createState() to create an instance of _PageViewLoginState.

```dart
class PasswordValidator {
  static String? validate(String? value) {
    if (value == null || value.isEmpty) {
      return 'Password is required';
    }
    if (value.length < 8) {
      return 'Password must be at least 8 characters long';
    }
    if (!value.contains(RegExp(r'[A-Z]'))) {
      return 'Password must contain at least one uppercase letter';
    }
    if (!value.contains(RegExp(r'[a-z]'))) {
      return 'Password must contain at least one lowercase letter';
    }
    if (!value.contains(RegExp(r'[0-9]'))) {
      return 'Password must contain at least one digit';
    }
    return null;

    // Return null if password is valid
  }
}
```

The PasswordValidator class provides a static method validate that encapsulates password validation logic. It checks if the password meets the required criteria: minimum length, contains uppercase and lowercase letters, and contains digits. It returns an error message if any of these conditions are not met, or null if the password is valid. This class can be used to ensure that passwords entered by users meet basic security requirements in an application.

Add_pet

```dart
// ignore_for_file: non_constant_identifier_names

import 'dart:io';

import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

import 'package:flutter_screenutil/flutter_screenutil.dart';

import 'package:get/get_connect/http/src/utils/utils.dart';
import 'package:image_picker/image_picker.dart';

import 'package:petapplication/core/utils/widgets/custom_buttom.dart';

import 'package:petapplication/pages/homepage/home_page_with_navigation.dart';
import 'package:petapplication/pages/my_pets_pages/my_pets.dart';
import 'package:petapplication/some_files_to_data/adding_pet_to_firestore.dar
t';
class AddPets extends StatefulWidget {
  const AddPets({super.key});

  @override
  State<AddPets> createState() => _AddPetsState();
}

class _AddPetsState extends State<AddPets> {
  //String? _imagePath2;
  final _formKey = GlobalKey<FormState>();
  // TextEditingController imageUrlController = TextEditingController();
  TextEditingController petNameController = TextEditingController();
  TextEditingController petGenderController = TextEditingController();
  TextEditingController petIdController = TextEditingController();
  TextEditingController petTypeController = TextEditingController();
  TextEditingController ageController = TextEditingController();
  TextEditingController weightController = TextEditingController();
  String? selectedPetType; // Variable to store the selected pet type
  bool showSecondContainer = false;
  late final PetsInformation petInformation;
  final _gender = ["Male", "Female"];
  final _petTypeGender = ['Cat', 'Dog'];

  String?
      _Selected; // Make _Selected nullable againRemove the nullable operator

  XFile? _selectedImage;
  String? _uploadedImageUrl;
  Future<void> _selectImageFromGallery() async {
    final picker = ImagePicker();
    final pickedImage = await picker.pickImage(source: ImageSource.gallery);
    if (pickedImage != null) {
      setState(() {
        _selectedImage = pickedImage;
      });
      await _uploadImageToFirebase(pickedImage);
    }
  }

  Future<String> _uploadImageToFirebase(XFile image) async {
    try {
      String fileName =
          '${DateTime.now().millisecondsSinceEpoch}_${image.name}';
      Reference storageRef =
          FirebaseStorage.instance.ref().child('pet_images/$fileName');
      await storageRef.putFile(File(image.path));
      String downloadURL = await storageRef.getDownloadURL();
      print('Image uploaded: $downloadURL');
      return downloadURL; // Return the download URL
    } catch (e) {
      print('Error uploading image: $e');
      return ''; // Return an empty string if there's an error
    }
  }

  @override
  Widget build(BuildContext context) {
    DecorationImage? decorationImage;
    if (_selectedImage != null) {
      decorationImage = DecorationImage(
        image: FileImage(File(_selectedImage!.path)),
        fit: BoxFit.cover,
      );
    } else {
      decorationImage = const DecorationImage(
        image: AssetImage(
            'assets/image/profileImage.png'), // Use AssetImage for local asset
        fit: BoxFit.cover,
      );
    }

```

- AddPetsState manages the state for the AddPets widget.
- It includes several TextEditingController instances for handling text input fields for pet details.
- GlobalKey<FormState> is used to manage the form state.
- selectedPetType and _Selected are variables to store the selected pet type and gender, respectively.
- selectedImage stores the currently selected image from the gallery.
- uploadedImageUrl holds the URL of the image uploaded to Firebase Storage.
- formKey is used to validate the form.
- XFile is used from the image_picker package to get the selected image from the gallery.
- selectImageFromGallery() is an asynchronous function that allows the user to pick an image from the gallery and update _selectedImage.
- uploadImageToFirebase(XFile image) is an asynchronous function that uploads the selected image to Firebase Storage and returns the download URL.
- build(BuildContext context) method builds the UI using the current state values.
- decorationImage is a variable that contains the decoration for the background image. If _selectedImage is not null, it sets the selected image from the gallery; otherwise, it uses a default image.

- This code defines a screen (AddPets) where users can add pet details, including a profile picture. It handles form validation, image selection from the gallery, and image uploading to Firebase Storage. The UI is built using Flutter's widgets, and it uses several text editing controllers to manage user input.

Create new password

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

import 'package:flutter_screenutil/flutter_screenutil.dart';

import 'package:get/get.dart';

import 'package:petapplication/core/utils/widgets/custom_buttom.dart';

import 'package:petapplication/pages/sign_login_acount/login_info_email_pass.dart';

class GreateNewPass extends StatefulWidget {
  const GreateNewPass({super.key});

  @override
  State<GreateNewPass> createState() => _GreateNewPassState();
}

class _GreateNewPassState extends State<GreateNewPass> {
  bool _obscureText = true;
  bool _obscureText2 = true;

  @override
  Widget build(BuildContext context) {
    final Size size = MediaQuery.of(context).size;
    var border = OutlineInputBorder(
        borderRadius: BorderRadius.circular(55.r),
        borderSide: const BorderSide(
            width: 0.4, color: Color.fromARGB(70, 112, 112, 112)));
    // double aspectRatio = screenHeight / screenWidth;
    return Scaffold(
      resizeToAvoidBottomInset: false,
      backgroundColor: const Color(0xffDCD3D3),
      appBar: AppBar(
        backgroundColor: const Color(0xffDCD3D3),
        elevation: 2,
        toolbarHeight: 230.h,
        automaticallyImplyLeading: false,
        leading: IconButton(
          icon: const Icon(
            Icons.arrow_back_rounded,
            color: Color(0xff4A5E7C),
          ),
```

```
        onPressed: () {
          Navigator.of(context).pop();
        },
        iconSize: 35.0,
        padding: const EdgeInsets.only(
            left: 6.0), // Set the size of the arrow icon
      ),
      forceMaterialTransparency: true,
      toolbarOpacity: 1,
      foregroundColor: const Color(0xff4A5E7C),
    ),
    extendBodyBehindAppBar: true,
    body: Column(children: [
      Padding(
        padding: EdgeInsets.only(top: size.height * 0.07),
        child: Text(
          'Create New Password',
          style: TextStyle(
            fontFamily: 'Cosffira',
            fontSize: 75.sp,
            color: const Color(0xff4A5E7C),
            fontWeight: FontWeight.w700,
          ),
        ),
      ),
      Padding(
        padding: EdgeInsets.only(
            top: size.height * 0.12, right: size.width * 0.45),
        child: Text(
          'New password',
          style: TextStyle(
            fontFamily: 'Cosffira',
            fontSize: 60.sp,
            color: const Color(0xff4A5E7C),
            fontWeight: FontWeight.w700,
            height: 5.h,
          ),
          textHeightBehavior:
              const TextHeightBehavior(applyHeightToFirstAscent: false),
          softWrap: false,
        ),
```

```
      Container(
        padding: EdgeInsets.only(
          right: size.width * 0.1,
          left: size.width * 0.1,
        ),
        child: TextField(
          obscureText: _obscureText,
          style: TextStyle(
            fontFamily: 'Cosffira',
            fontSize: 50.sp,
            color: const Color(0xff000000),
            fontWeight: FontWeight.w600,
          ),
          decoration: InputDecoration(
            contentPadding: EdgeInsets.only(
                left: size.width * 0.06, top: size.height * 0.05),
            suffixIcon: IconButton(
              icon: Icon(
                size: 20,
                color: const Color.fromARGB(96, 9, 15, 15),
                _obscureText
                    ? Icons.visibility_off_outlined
                    : Icons.visibility_outlined,
                //color: ,
              ),
              onPressed: () {
                setState(() {
                  _obscureText = !_obscureText;
                });
              },
            ),
            fillColor: const Color(0xFFFFFFFF),
            filled: true,
            enabledBorder: border,
            focusedBorder: border,
          ),
        ),
      ),
      Padding(
          padding: EdgeInsets.only(
            top: size.height * 0.01,
            right: size.height * 0.15,
          ),
```

```
),
Padding(
    padding: EdgeInsets.only(
      top: size.height * 0.01,
      right: size.height * 0.15,
    ),
    child: Text(
      ' ! Must be at least 8 characters',
      style: TextStyle(
        fontFamily: 'Cosffira',
        fontSize: 38.sp,
        color: const Color(0xffA26874),
        fontWeight: FontWeight.w800,
        height: 1.9411764705882353.h,
      ),
      textHeightBehavior:
          const TextHeightBehavior(applyHeightToFirstAscent: false),
      softWrap: false,
    )),
Padding(
    padding: EdgeInsets.only(
        top: size.height * 0.06,
        //left: size.width *0.08,
        right: size.width * 0.35),
    child: Text(
      'Confirm Password',
      style: TextStyle(
        fontFamily: 'Cosffira',
        fontSize: 60.sp,
        color: const Color(0xff4A5E7C),
        fontWeight: FontWeight.w700,
        height: 1.9411764705882353.h,
      ),
      textHeightBehavior:
          const TextHeightBehavior(applyHeightToFirstAscent: false),
      softWrap: false,
    )),
Container(
  padding: EdgeInsets.only(
      top: size.height * 0.015,
      left: size.width * 0.1,
      right: size.width * 0.1),
```

```
child: TextField(
  obscureText: _obscureText2,
  style: TextStyle(
    fontFamily: 'Cosffira',
    fontSize: 50.sp,
    color: const Color(0xff000000),
    fontWeight: FontWeight.w600,
  ), // TextStyle
  decoration: InputDecoration(
    contentPadding: EdgeInsets.only(
        left: size.width * 0.06, top: size.height * 0.05), // EdgeInsets.only
    suffixIcon: IconButton(
      icon: Icon(
        size: 20,
        color: const Color.fromARGB(96, 9, 15, 15),
        _obscureText2
            ? Icons.visibility_off_outlined
            : Icons.visibility_outlined,
        //color: ,
      ), // Icon
      onPressed: () {
        setState(() {
          _obscureText2 = !_obscureText2;
        });
      },
    ), // IconButton
    fillColor: const Color(0xFFFFFFFF),
    filled: true,
    hintStyle: const TextStyle(
      fontFamily: 'Cosffira',
      fontSize: 14,
      color: Color.fromARGB(116, 19, 79, 92),
      fontWeight: FontWeight.w600,
    ), // TextStyle
    enabledBorder: border,
    focusedBorder: border,
  ), // InputDecoration
), // TextField
), // Container
```

This screen allows users to create a new password by entering it twice. It provides an option to toggle password visibility. Upon submission, it navigates the user to the login page.

```dart

import 'dart:async';

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

import 'package:petapplication/core/utils/widgets/repeatColorsUse.dart';
import 'package:petapplication/pages/pageForgetpass/create_new_pass.dart';
import 'package:pinput/pinput.dart';

class MyVerify extends StatefulWidget {
  const MyVerify({super.key});

  @override
  State<MyVerify> createState() =>  _MyVerifyState();
}

class _MyVerifyState extends State<MyVerify> {
  int resendTime = 120;
  late Timer countertimer;
  bool isTimerRunning = false;
  @override
  void initState() {
    startTimer();
    super.initState();
  }

  void starttimer() {
    countertimer = Timer.periodic(const Duration(seconds: 1), (timer) {
      setState(() {
        resendTime = resendTime - 1;
      });
      if (resendtime < 1) {
        stopTimer();
      }
    });
  }

  void stopTimer() {
    if (countertimer.isActive) {
      countertimer.cancel();
      setState(() {
        isTimerRunning = false;
      });
    }
  }

  String getFormattedTime() {
    int minutes = resendTime ~/ 60;
    int seconds = resendTime % 60;
    return '$minutes:${seconds.toString().padLeft(1, '0')}';
  }

  @override
  Widget build(BuildContext context) {
     Size size = MediaQuery.of(context).size;
    final defaultPinTheme = PinTheme(
      margin: const EdgeInsets.symmetric(horizontal: 5),
      width: 100.w,
      height: 100.h,
      textStyle: TextStyle(
          fontSize: 42.sp,
          color: const Color(0xff020202),
          fontWeight: FontWeight.w900),
      decoration: BoxDecoration(
        color: const  Color.fromARGB(20, 128, 121, 121),
        border: Border.all(color: const Color.fromARGB(88, 112, 112, 112)),
        borderRadius: BorderRadius.circular(30.r),
      ),
    );

    final focusedPinTheme = defaultPinTheme.copyDecorationWith(
      color: const Color(0xfff7f7f7),
      border: Border.all(
        color: kMainColorSplash,
      ),
      borderRadius: BorderRadius.circular(30.r),
    );

    final submittedPinTheme = defaultPinTheme.copyWith(
      decoration: defaultPinTheme.decoration?.copyWith(
        color: const Color(0xffF1E6E5),
      ),
    );

    return Scaffold(
      backgroundColor: const Color(0xffEFEBE5),
      extendBodyBehindAppBar: true,
      body: Container(
        margin: EdgeInsets.only(
          left: size.width*0.07,
          right: size.width*0.07,
          bottom: size.height*0.05
          ),
        alignment: Alignment.center,
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Image.asset(
                'assets/image/cloud.png',
                width: 500.w,
                height: 200.h,
              ),
              SizedBox(
                height: 100.h,
              ),
              Text(
                "Please Enter 5-Digit Code",
                style: TextStyle(
                  fontFamily: 'Cosffira',
                  fontSize: 70.sp,
                  fontWeight: FontWeight.w700,
                  color: const Color(0xff4A5E7C),
                ),
              ),
              const SizedBox(
                height: 8,
              ),
              RichText(
                text: TextSpan(
                  text: 'We\'ve Sent A Code To ',
                  style: TextStyle(
                    fontFamily: 'Cosffira',
                    fontSize: 45.sp,
                    color: const Color(0xff000000),
                    fontWeight: FontWeight.w300,
                  ),
                  children: <TextSpan>[
                    TextSpan(
                      text: 'malk22@gmail.com\n',
                      style: TextStyle(
                        fontSize: 44.sp,
                        color:  const  Color(0xffA26874),
                        fontWeight: FontWeight.w500,
                      ),
                    ),
                    TextSpan(
                      text: 'Enter a Code In That Message',
                      style: TextStyle(
                        fontSize: 45.sp,
                        color: const Color(0xff000000),
                        fontWeight: FontWeight.w300,
                      ),
                    ),
                  ],
                ),
              ),
              SizedBox(
                height: 60.h,
              ),
              Pinput(
                length: 5,
                defaultPinTheme: defaultPinTheme,
                focusedPinTheme: focusedPinTheme,
                submittedPinTheme: submittedPinTheme,
                showCursor: true,
                onCompleted: (pin) {
                  Navigator.of(context).pushReplacement(
                    MaterialPageRoute(
                        builder: (context) => const CreateNewPass()),
                  );
                },
              ),
              SizedBox(
                height: 40.h,
              ),
              Row(
                mainAxisAlignment: MainAxisAlignment.start,
                children: [
                  SizedBox(width: 660.w),
                  isTimerRunning
                      ? Row(
                          children: [
                            const Icon(
                              Icons.alarm_on,
                              color: Color(0xffA26874),
                              size: 28,
                            ),
                            SizedBox(width: 0.w),
                            Text(
                              ' ${getFormattedTime()} Min',
                              style: TextStyle(
                                fontSize: 40.sp,
                                fontWeight: FontWeight.w700,
                                color: const  Color(0xffA26874),
                              ),

                            ),
                          ],
                        )
                      : InkWell(
                          onTap: () {
                            // resend code
                            setState(() {
                              resendTime = 120;
                              isTimerRunning = true;
                            });
                            starttimer();
                          },
                          child: Text(
                            'Resend',
                            style: TextStyle(
                              fontFamily: 'Cosffira',
                              fontSize: 45.sp,
                              fontWeight: FontWeight.w800,
                              color: const Color(0xff4A5E7C),
                            ),
                          ),
                        ),
                ],
              )
            ],
          ),
        ),
      ),
    );
  }
}

```

verify password page

This screen is used to verify an email or phone number by entering a 5-digit code. It displays instructions, allows the user to input the code, and provides an option to resend the code with a countdown timer. Upon successful verification, it navigates to the screen for creating a new password (GreateNewPass).

Add reminder to firestore

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import 'package:petapplication/pages/events_system/events_for_pet.dart';

import 'package:petapplication/some_files_to_data/adding_pet_to_firestore.dar
t';
Future<ReminderData?> createReminderData(
    DateTime selectedDate,
    TimeOfDay reminderTime,
    String currentItemSelected,
    PetsInformation pet) async {
  // Extracting date components
  final String day = selectedDate.day.toString();
  final String month = DateFormat('MMM').format(selectedDate);
  final String year = selectedDate.year.toString();
  final String weekDay = DateFormat('E').format(selectedDate);

  // Extracting time components
  final String hours = reminderTime.hourOfPeriod.toString().padLeft(2, '0');
  final String period = reminderTime.hour < 12 ? 'AM' : 'PM';
  final String minutes = reminderTime.minute.toString().padLeft(2, '0');
  final fireStore = FirebaseFirestore.instance;
  ReminderData returnedReminder;
  DocumentReference docRef =
      await FirebaseFirestore.instance.collection('reminders').add(
   {
      'day': day,
      'month': month,
      'year': year,
      'weekDay': weekDay,
      'hours': hours,
      'minutes': minutes,
      'reminderType': currentItemSelected,
      'night': period,
      //'user-id': userId == 'null' ? 'not-signed' : userId,
      'pet-id': pet.petId,
    },
  );
  String reminderId = docRef.id;
  await FirebaseFirestore.instance
      .collection('reminders')
      .doc(reminderId)
      .update(
    {'reminder-id': reminderId},
  );

  DocumentSnapshot<Map<String, dynamic>> documentSnapshot =
      await fireStore.collection('reminders').doc(reminderId).get();
  if (documentSnapshot.exists) {
    // Convert document snapshot to ReminderData
    returnedReminder = ReminderData(
      day: documentSnapshot['day'] ?? '',
      month: documentSnapshot['month'] ?? '',
      reminderType: documentSnapshot['reminderType'] ?? '',
      hours: documentSnapshot['hours'] ?? '',
      minutes: documentSnapshot['minutes'] ?? '',
      night: documentSnapshot['night'] ?? '',
      weekDay: documentSnapshot['weekDay'] ?? '',
      year: documentSnapshot['year'] ?? '',
      reminderId: documentSnapshot.id,
    );

    return returnedReminder;
  } else {
    return null;
  }
}

```

1. Function Signature:
   - Future<ReminderData?> createReminderData: Returns a Future that resolves to a ReminderData object or null.
   - Parameters:
     - DateTime selectedDate: The date selected for the reminder.
     - TimeOfDay reminderTime: The time selected for the reminder.
     - String currentItemSelected: The type of reminder (e.g., "Feeding", "Walking").
     - PetsInformation pet: An object representing the pet for which the reminder is being set.
2. Extracting Date and Time Components:
   - Extracts and formats various components of the selectedDate and reminderTime to be stored in Firestore.
   - Uses the intl package to format the month and weekday names.
   - hourOfPeriod and padLeft are used to ensure the hours and minutes are correctly formatted.
3. Firestore Interaction:
   - Gets an instance of Firestore.
   - Adds a new document to the reminders collection with the extracted date and time components, reminder type, and pet ID.
   - Retrieves the document ID (reminderId) of the newly created document.
   - Updates the document to include the reminder-id field with its own ID.
4. Retrieving and Returning the Reminder:
   - Retrieves the document snapshot of the newly created reminder using its reminderId.
   - If the document exists, it constructs a ReminderData object from the document fields and returns it.
   - If the document does not exist, returns null.

ReminderData Class (Assumed)

The code assumes the existence of a ReminderData class which is used to structure the reminder data retrieved from Firestore.

Flutter Code:

today_and_future_reminders_data

```dart
import 'package:petapplication/pages/events_system/events_for_pet.dart';

import 'package:petapplication/some_files_to_data/adding_pet_to_firestore.dar
t';
class Reminders {
  Reminders({
    required this.imageUrl,
    required this.eventTitle,
    required this.time,
    required this.petName,
    required this.checked,
  });
  late String imageUrl;
  late String eventTitle;
  late String time;
  late String petName;
  late int reminderId;
  late bool checked;
}

class FutureEventsInformations {
  FutureEventsInformations({
    required this.imageUrl,
    required this.eventTitle,
    required this.eventDate,
    required this.petName,
    required this.petType,
    required PetsInformation pet,
  }) : pet = pet;
  late String imageUrl;
  late String eventTitle;
  late String eventDate;
  late String petName;
  late String petType;
  late PetsInformation pet;
  PetsInformation get petInfo => pet;
}

List<Reminders> mergeReminders(
    List<PetsInformation> dogsList, List<PetsInformation> catsList) {
  final now = DateTime.now();
  final today = DateTime(now.year, now.month, now.day);
  //final nextThreeDays = today.add(const Duration(days: 3));
  final List<Reminders> returnedList = [];
  // Clear the lists before merging reminders
  Map<String, int> monthsMap = {
    'Jan': 1,
    'Feb': 2,
    'Mar': 3,
    'Apr': 4,
    'May': 5,
    'Jun': 6,
    'Jul': 7,
    'Aug': 8,
    'Sep': 9,
    'Oct': 10,
    'Nov': 11,
    'Dec': 12,
  };
  // Iterate through each pet
  for (PetsInformation pet in dogsList) {
    for (ReminderData reminder in pet.remindersData) {
      final reminderDate = DateTime(
        int.parse(reminder.year),
        monthsMap[reminder.month]!,
        int.parse(reminder.day),
      );
      if (reminderDate.day == today.day) {
        returnedList.add(
          Reminders(
            imageUrl: pet.imageUrl,
            eventTitle: reminder.reminderType,
            time: '${reminder.hours}:${reminder.minutes} ${reminder.night}',
            petName: pet.petName,
            checked: false,
          ),
        );
      }
    }
  }

```

```dart

List<FutureEventsInformations> getFutureEvents(
    List<PetsInformation> dogsList, List<PetsInformation> catsList) {
  final now = DateTime.now();
  final today = DateTime(now.year, now.month, now.day);
  final nextThreeDays = today.add(const Duration(days: 4));

  final List<FutureEventsInformations> returnedList = [];
  // Clear the lists before merging reminders
  Map<String, int> monthsMap = {
    'Jan': 1,
    'Feb': 2,
    'Mar': 3,
    'Apr': 4,
    'May': 5,
    'Jun': 6,
    'Jul': 7,
    'Aug': 8,
    'Sep': 9,
    'Oct': 10,
    'Nov': 11,
    'Dec': 12,
  };
  // Iterate through each pet
  for (PetsInformation pet in dogsList) {
    for (ReminderData reminder in pet.remindersData) {
      final reminderDate = DateTime(
        int.parse(reminder.year),
        monthsMap[reminder.month]!,
        int.parse(reminder.day),
      );

      if (reminderDate.isBefore(nextThreeDays) && reminderDate.isAfter(today)) {
        returnedList.add(FutureEventsInformations(
          imageUrl: pet.imageUrl,
          eventTitle: reminder.reminderType,
          eventDate:
              '${reminder.weekDay} ${reminder.hours}:${reminder.minutes} ${reminder.night}',
          petName: pet.petName,
          petType: pet.petIsDogOrCat!,
          pet: pet,
        ));
      }
    }
  }

  // Merge reminders for cats
  for (PetsInformation pet in catsList) {
    for (ReminderData reminder in pet.remindersData) {
      final reminderDate = DateTime(
        int.parse(reminder.year),
        monthsMap[reminder.month]!,
        int.parse(reminder.day),
      );
      if (reminderDate.isBefore(nextThreeDays) && reminderDate.isAfter(today)) {
        returnedList.add(FutureEventsInformations(
          imageUrl: pet.imageUrl,
          eventTitle: reminder.reminderType,
          eventDate:
              '${reminder.weekDay} ${reminder.hours}:${reminder.minutes} ${reminder.night}',
          petName: pet.petName,
          petType: pet.petIsDogOrCat!,
          pet: pet,
        ));
      }
    }
  }
  return returnedList;
}

```

The Reminders class represents a reminder with properties for:

- imageUrl: URL of the pet's image.
- eventTitle: Title of the event.
- time: Time of the reminder.
- petName: Name of the pet.
- reminderId: ID of the reminder.
- checked: Boolean indicating whether the reminder is checked/completed.The FutureEventsInformations class represents information about future events for a pet with properties for:

  - imageUrl: URL of the pet's image.
  - eventTitle: Title of the event.
  - eventDate: Date of the event.
  - petName: Name of the pet.
  - petType: Type of pet (dog or cat).
  - pet: An instance of the PetsInformation class.

The mergeReminders function merges reminders for dogs and cats for the current day:

1. Gets the current date.
2. Creates an empty list to store reminders.
3. Defines a map to convert month abbreviations to numbers.
4. Iterates over the dogs and cats lists.
5. For each pet, checks if any reminders match the current date.
6. Adds matching reminders to the list.
7. Returns the list of reminders.

The getFutureEvents function gets events for the next three days for dogs and cats:

1. Gets the current date and calculates the date three days from now.
2. Creates an empty list to store future events.
3. Defines a map to convert month abbreviations to numbers.
4. Iterates over the dogs and cats lists.
5. For each pet, checks if any reminders fall within the next three days.
6. Adds matching events to the list.
7. Returns the list of future events.

```dart
import 'dart:io';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/svg.dart';
import 'package:image_picker/image_picker.dart';
import 'package:petapplication/profile_page/updating_user_information.dart';
import 'package:petapplication/profile_page/user_profile.dart';
import 'package:petapplication/profile_page/uploading_image_for_user.dart';

class EditAcount extends StatefulWidget {
  const EditAcount({super.key});

  @override
  State<EditAcount> createState() => _EditAcountState();
}

class _EditAcountState extends State<EditAcount> {
  TextEditingController nameController = TextEditingController();
  TextEditingController emailController = TextEditingController();
  TextEditingController phoneNumberController = TextEditingController();
  User? userInfo = FirebaseAuth.instance.currentUser;
  File? _pickedImageFile;
  bool editPageLoading = false;

  final _formKey = GlobalKey<FormState>();

  String? validateEmail(String? value) {
    if (value == null || value.isEmpty) {
      return 'Enter your email, please';
    } else {
      // Regular expression to validate email format
      const pattern = r"(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'"
          r'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-'
          r'\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*'
          r'[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:(2(5[0-5]|[0-
4]'       r'[0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-
9]'       r'[0-9]|[1-9]?[0-9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f
\'        r'x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])';
      final regex = RegExp(pattern);

      if (!regex.hasMatch(value)) {
        return 'Invalid email';
      }
    }
    return null; // Return null if the email is valid
  }

  Future<void> _pickedImage(BuildContext context) async {
    final pickedImage = await ImagePicker().pickImage(
      source: ImageSource.gallery,
      imageQuality: 100,
      maxWidth: MediaQuery.of(context).size.width,
      maxHeight: MediaQuery.of(context).size.height,
    );
    if (pickedImage != null) {
      setState(() {
        _pickedImageFile = File(pickedImage.path);
      });
    }
  }

  Future<void> _initializeUserData() async {
    try {
      final doc = await FirebaseFirestore.instance
          .collection('users')
          .doc(userInfo!.uid)
          .get();
      if (userInfo != null) {
        nameController.text =
            userInfo!.displayName ?? doc.data()?['user_name'] ?? '';
        emailController.text = userInfo!.email ?? '';
        phoneNumberController.text =
            userInfo!.phoneNumber ?? doc.data()?['phone_number'] ?? '';
      }
    } on FirebaseException {
      print("error in there.....................");
    }
  }

  @override
  void initState() {
    super.initState();
    _initializeUserData();
  }

  @override
  void dispose() {
    nameController.dispose();
    emailController.dispose();
    phoneNumberController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    var size = MediaQuery.of(context).size;
    double height = size.height;
    double width = size.width;
    final imageApi = FirebaseApiForUserImage();
    var border = OutlineInputBorder(
        borderRadius: BorderRadius.circular(40.r),
        borderSide: const BorderSide(
            width: 0.5, color: Color.fromARGB(70, 112, 112, 112)));

```

Edit account

This Dart file defines a Flutter widget, EditAcount, which allows users to edit their account information, including their name, email, phone number, and profile picture.

- EditAcount: A stateful widget representing the account editing screen.
- createState: Creates the mutable state for this widget.
- Controllers: nameController, emailController, and phoneNumberController handle text input for the user's name, email, and phone number.
- userInfo: Stores the currently logged-in user's information.
- pickedImageFile: Stores the picked image file.
- editPageLoading: Indicates if the page is in a loading state.
- ormKey: Key for the form to manage validation and state.
- validateEmail: Validates the email using a regular expression pattern to ensure it follows standard email format.

_pickedImage: Asynchronously picks an image from the gallery and updates _pickedImageFile.

_initializeUserData: Fetches user data from Firestore and initializes text controllers with the user's current information.

- initState: Called when the widget is first created. It initializes user data.
- dispose: Called when the widget is removed from the widget tree. It disposes of the text controllers.

build: Builds the UI of the widget. It includes setting up the form, handling image picking, and providing save functionality.

Updating user information

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

Future<void> updateUserInformation({
  required BuildContext cxt,
  String? imageUrl,
  required String phoneNumber,
  required String name,
  required String email,
}) async {
  User? user = FirebaseAuth.instance.currentUser;
  if (user == null) {
    return;
  }

  try {
    await user.updateDisplayName(name);
    await FirebaseFirestore.instance.collection('users').doc(user.uid).update({
      'profile_image': imageUrl,
      'phone_number': phoneNumber,
      'email': user.email,
      'user_name': user.displayName,
    });
  } on FirebaseException catch (error) {
    print(error);
    ScaffoldMessenger.of(cxt).clearSnackBars();
    ScaffoldMessenger.of(cxt).showSnackBar(
      SnackBar(
        content: Text(
          error.toString(),
        ),
        action: SnackBarAction(
            label: 'Close',
            onPressed: () {
              ScaffoldMessenger.of(cxt).hideCurrentSnackBar();
            }),
      ),
    );
  }
}

```

The updateUserInformation function updates the user's profile information (name, phone number, email, and profile image) in Firebase Authentication and Firestore. If an error occurs, it catches the error and displays a snack bar with the error message. This function is useful for maintaining synchronized user data between Firebase Authentication and Firestore, ensuring that any updates to the user's profile are reflected in both places

- Future<void>: The function is asynchronous and returns a Future that resolves to void.
- updateUserInformation: The name of the function.

 Parameters:

  - cxt: The BuildContext used to show snack bars.
  - imageUrl: The URL of the user's profile image (optional).
  - phoneNumber: The user's phone number.
  - name: The user's display name.
  - email: The user's email address.

- user: Gets the currently logged-in user from Firebase Authentication.
- Check: If no user is logged in, the function returns immediately.
- try block: Attempts to update the user's information.
- user.updateDisplayName(name): Updates the user's display name in Firebase Authentication.
- FirebaseFirestore.instance.collection('users').doc(user.uid).update({...}) Updates the user's document in the Firestore users collection with the new information:

  - profile_image: The URL of the user's profile image.
  - phone_number: The user's phone number.
  - email: The user's email (retrieved from user.email).
  - user_name: The user's display name (retrieved from user.displayName).

Error Handling:

- on FirebaseException catch (error): Catches any FirebaseException that occurs during the update process.
- print(error): Prints the error to the console for debugging purposes.
- ScaffoldMessenger.of(cxt).clearSnackBars(): Clears any existing snack bars.
- ScaffoldMessenger.of(cxt).showSnackBar(...): Shows a new snack bar with the error message and a close button.

Camera screen page

```dart
// ignore_for_file: library_private_types_in_public_api, sized_box_for_whitespace

import 'dart:io';

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

import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/svg.dart';

import 'package:petapplication/pages/detect_page/detect_screen.dart';

class PreviewScreen extends StatefulWidget {
  final String imgPath;
  final void Function(BuildContext) onCapturePressed;

  const PreviewScreen(
      {super.key, required this.imgPath, required this.onCapturePressed});

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

class _PreviewScreenState extends State<PreviewScreen> {
  late CameraController cameraController;
  // wxpwxting type result
  String petType = 'Golden Retrival';
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.transparent,
      appBar: AppBar(
        automaticallyImplyLeading: true,
        backgroundColor: Colors.transparent,
        leading: IconButton(
          icon: const Icon(
            Icons.arrow_back_rounded,
            color: Color(0xff9F9A95),
          ),
          onPressed: () {
            Navigator.of(context).pop();
          },
          iconSize: 39.0,
          padding: const EdgeInsets.only(left: 6.0),
        ),
      ),
      extendBodyBehindAppBar: true,
      body: Stack(
        children: [
          Positioned.fill(
            child: Image.file(
              File(widget.imgPath),
              fit: BoxFit.cover,
            ),
          ),
          Align(
            alignment: Alignment.bottomCenter,
            child: Container(
              width: double.infinity,
              height: 400.h,
              color: Colors.transparent,
              child: Center(
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    Padding(
                      padding: const EdgeInsets.only(left: 5.0),
                      child: IconButton(
                        color: Colors.blue,
                        onPressed: () {
                          Navigator.of(context).pop();
                        },
                        icon: SvgPicture.asset(
                          'assets/icons/rephoto.svg',
                          width: 200.w,
                          height: 200.h,
                        ),
                      ),
                    ),
                    SizedBox(
                      width: 200.w,
                    ),
                    Padding(
                      padding: const EdgeInsets.only(right: 10.0),
                      child: IconButton(
                        color: Colors.blue,
                        onPressed: () {
                          /*  Navigator.push(
                            context,
                            MaterialPageRoute(
                              builder: (context) => DetectScreen(
                                imgPath: widget
                                    .imgPath, // Pass the image path to DetectScreen
                                onCapturePressed: widget
                                    .onCapturePressed, // Pass the function to DetectScree
n                            ),
                            ),
                          ); */

                          showDialog(
                            context: context,
                            builder: (context) => DetectScreen(
                              imgPath: widget.imgPath,
                              onCapturePressed: widget.onCapturePressed,
                              petType: petType,
                            ),
                          );
                        },
                        icon: SvgPicture.asset(
                          'assets/icons/right.svg',
                          width: 200.w,
                          height: 200.h,
                        ),
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ),
        ],
      ),
    );
  }
}
```

PreviewScreen is a stateful widget that takes two required parameters:

- imgPath: A string representing the path to the image file.
- onCapturePressed: A function that takes a BuildContext parameter, usually used to capture or confirm the image.

_PreviewScreenState manages the state for the PreviewScreen.
It includes:

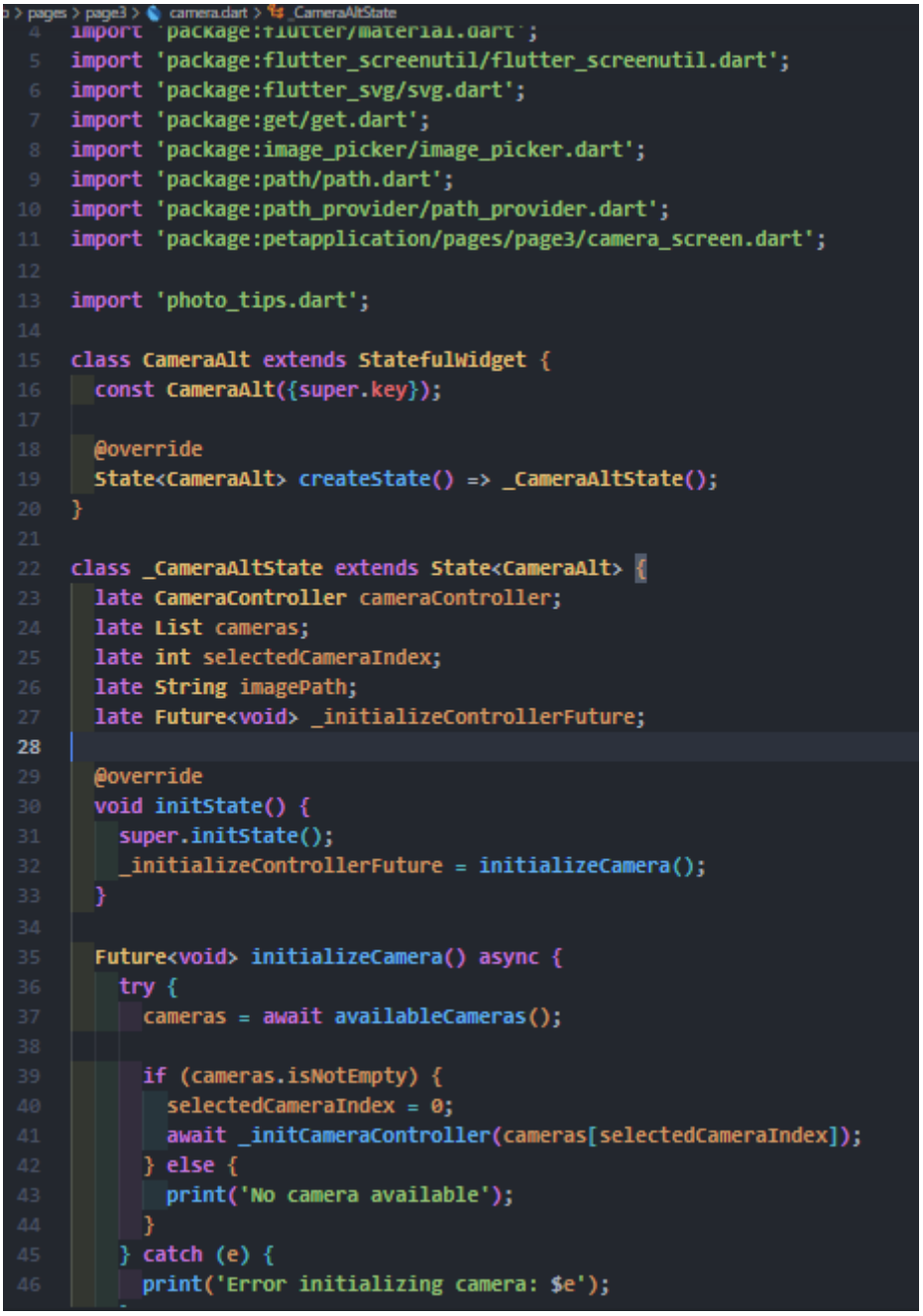
- petType: A string that stores the default pet type ('Golden Retriever').
- cameraController: A CameraController object (though it's declared but not used in the provided code).

- he build(BuildContext context) method constructs the UI for the PreviewScreen.
- Scaffold is used as the root widget with a transparent background.
- AppBar is transparent with a back button.
- Stack is used to overlay the image and action buttons.
- Image.file displays the image using the imgPath.
- Two IconButtons are placed at the bottom center:

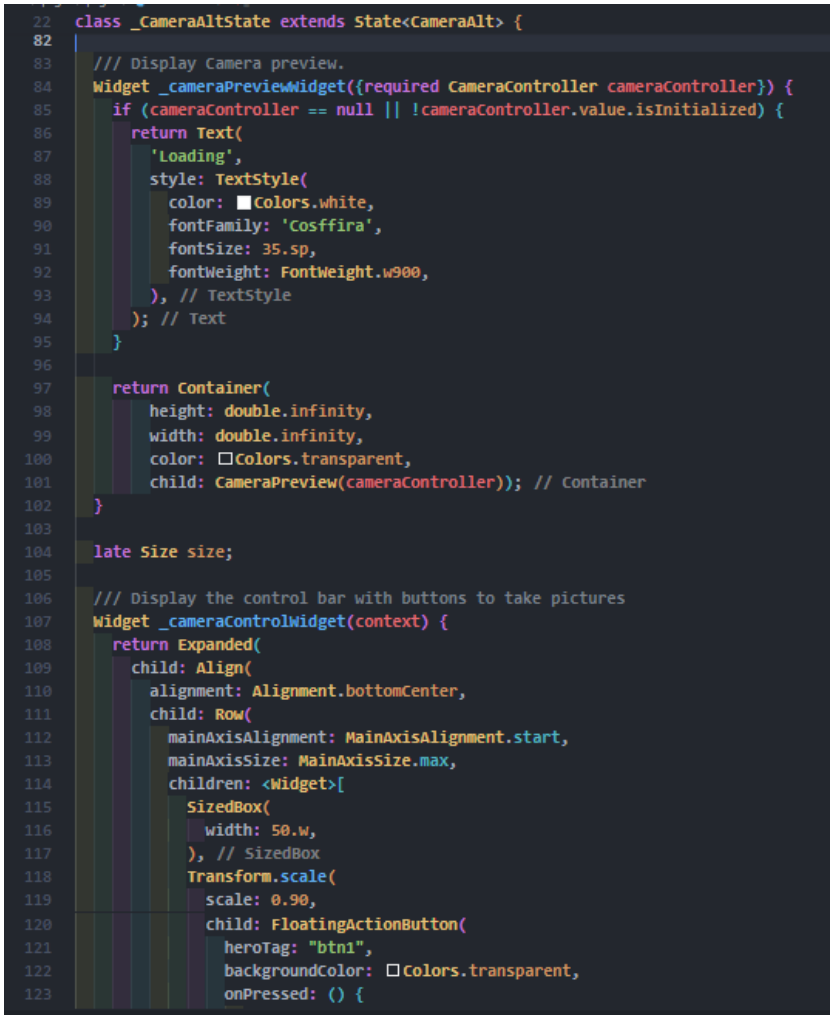
  - The left one (rephoto.svg) navigates back.
  - The right one (right.svg) opens a dialog (DetectScreen) passing the imgPath, onCapturePressed, and petType as parameters.

This screen is designed to preview an image taken with a camera and provides options to re-capture the image or confirm it. The petType is a placeholder for a pet type that can be selected or detected in further functionality. It also integrates with a DetectScreen to perform further actions based on the captured image.

Camera.dart


pages > page3 > camera.dart > _CameraAltState
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/svg.dart';
import 'package:get/get.dart';
import 'package:image_picker/image_picker.dart';
import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart';
import 'package:petapplication/pages/page3/camera_screen.dart';
import 'photo_tips.dart';
class CameraAlt extends StatefulWidget {
const CameraAlt({super.key});
@override
State<CameraAlt> createState() => _CameraAltState();
}
class _CameraAltState extends State<CameraAlt> {
late CameraController cameraController;
late List cameras;
late int selectedCameraIndex;
late String imagePath;
late Future<void> _initializeControllerFuture;
@override
void initState() {
super.initState();
_initializeControllerFuture = initializeCamera();
}
Future<void> initializeCamera() async {
try {
cameras = await availableCameras();
if (cameras.isNotEmpty) {
selectedCameraIndex = 0;
await _initCameraController(cameras[selectedCameraIndex]);
} else {
print('No camera available');
}
} catch (e) {
print('Error initializing camera: $e');

```
    }
  } catch (e) {
    print('Error initializing camera: $e');
  }
}

Future<void> _initCameraController(
    CameraDescription cameraDescription) async {
  cameraController = CameraController(
    cameraDescription,
    ResolutionPreset.max,
    enableAudio: false,
    imageFormatGroup:
        ImageFormatGroup.yuv420, // Specify the image format group
  );

  cameraController.addListener(() {
    if (mounted) {
      setState(() {});
    }

    if (cameraController.value.hasError) {
      print('Camera error: ${cameraController.value.errorDescription}');
    }
  });

  try {
    await cameraController.initialize();
  } catch (e) {
    print('Error initializing camera controller: $e');
  }
}

@override
void dispose() {
  cameraController.dispose();
  super.dispose();
}
```



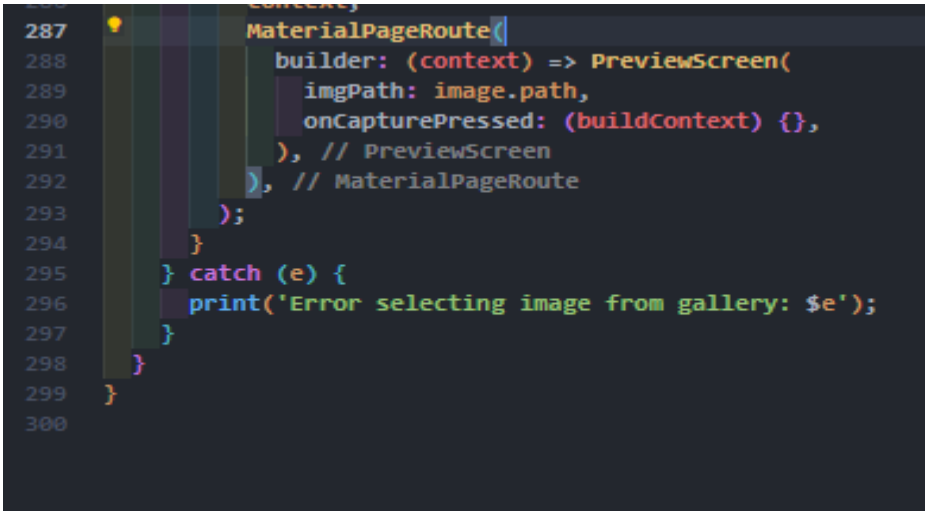
```dart
class _CameraAltState extends State<CameraAlt> {

  /// Display Camera preview.
  Widget _cameraPreviewWidget({required CameraController cameraController}) {
    if (cameraController == null || !cameraController.value.isInitialized) {
      return Text(
        'Loading',
        style: TextStyle(
          color: Colors.white,
          fontFamily: 'Cosffira',
          fontSize: 35.sp,
          fontWeight: FontWeight.w900,
        ), // TextStyle
      ); // Text
    }

    return Container(
        height: double.infinity,
        width: double.infinity,
        color: Colors.transparent,
        child: CameraPreview(cameraController)); // Container
  }

  late Size size;

  /// Display the control bar with buttons to take pictures
  Widget _cameraControlWidget(context) {
    return Expanded(
      child: Align(
        alignment: Alignment.bottomCenter,
        child: Row(
          mainAxisAlignment: MainAxisAlignment.start,
          mainAxisSize: MainAxisSize.max,
          children: <Widget>[
            SizedBox(
              width: 50.w,
            ), // SizedBox
            Transform.scale(
              scale: 0.90,
              child: FloatingActionButton(
                heroTag: "btn1",
                backgroundColor: Colors.transparent,
                onPressed: () {
```

```
Widget _cameraControlWidget(context) {
                onPressed: () {
                  _onGalleryPressed();
                },
                shape: const CircleBorder(),
                child: SvgPicture.asset(
                  'assets/icons/Gallarry.svg',
                  // color: Colors.black,
                  width: 350.w,
                  height: 250.h,
                ), // SvgPicture.asset
              ), // FloatingActionButton
            ), // Transform.scale
            SizedBox(
              width: 230.w,
            ), // SizedBox
            Transform.scale(
              scale: 1.32,
              child: FloatingActionButton(
                heroTag: "btn2",
                backgroundColor: Colors.transparent,
                onPressed: () {
                  _onCapturePressed(context);
                },
                shape: const CircleBorder(),
                child: SvgPicture.asset(
                  'assets/icons/cameraa.svg',
                  // color: Colors.black,
                  width: 300.w,
                  height: 300.h,
                ), // SvgPicture.asset
              ), // FloatingActionButton
            ), // Transform.scale
            SizedBox(
              width: size.width * 0.16,
            ), // SizedBox
            Transform.scale(
              scale: size.width * 0.0017,
              child: FloatingActionButton(
                heroTag: "btn3",
                backgroundColor: Colors.transparent,
                onPressed: () {
```

```
                onPressed: () {
                  Get.to(() => const TipsOfPhoto(),
                      transition: Transition.native);
                  //
                },
                shape: const CircleBorder(),
                child: SvgPicture.asset(
                  'assets/icons/tips.svg',
                  // color: Colors.black,
                  width: 350.w,
                  height: 250.h,
                ), // SvgPicture.asset
              ), // FloatingActionButton
            ), // Transform.scale
          ], // <Widget>[]
        ), // Row
      ), // Align
    ); // Expanded
  }

  @override
  Widget build(BuildContext context) {
    size = MediaQuery.of(context).size;
    return Scaffold(
      backgroundColor: Colors.transparent,
      appBar: AppBar(
        automaticallyImplyLeading: true,
        backgroundColor: Colors.transparent,
        leading: IconButton(
          icon: const Icon(
            Icons.arrow_back_rounded,
            color: Color(0xff9F9A95),
          ), // Icon
          onPressed: () {
            Navigator.of(context).pop();
          },
          iconSize: 39.0,
          padding: const EdgeInsets.only(left: 6.0),
        ), // IconButton
      ), // AppBar
      extendBodyBehindAppBar: true,
```

```
      body: Stack(
        children: [
          Container(
            height: double.infinity,
            width: double.infinity,
            color: Colors.transparent,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.end,
              crossAxisAlignment: CrossAxisAlignment.center,
              mainAxisSize: MainAxisSize.max,
              children: [
                Expanded(
                  child: FutureBuilder<void>(
                    future: _initializeControllerFuture,
                    builder: (context, snapshot) {
                      if (snapshot.connectionState == ConnectionState.done) {
                        return _cameraPreviewWidget(
                            cameraController: cameraController);
                      } else {
                        return const Center(child: CircularProgressIndicator());
                      }
                    },
                  ), // FutureBuilder
                ), // Expanded
                Align(
                  alignment: Alignment.bottomCenter,
                  child: Container(
                    color: Colors.transparent,
                    height: 235.h,
                    width: double.infinity,
                    padding: const EdgeInsets.all(15),
                    //  color: Colors.black12,
                    child: Row(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: <Widget>[
                        _cameraControlWidget(context),
                      ], // <Widget>[]
                    ), // Row
                  ), // Container
                ) // Align
              ]
```

```
void _showCameraException(CameraException e) {
  String errorText = 'Error:${e.code}\nError message : ${e.description}';
  print(errorText);
}

void _onCapturePressed(context) async {
  try {
    final path =
        join((await getTemporaryDirectory()).path, '${DateTime.now()}.png');
    XFile picture = await cameraController
        .takePicture(); // Capture picture without passing path argument

    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => PreviewScreen(
          imgPath: picture.path,
          onCapturePressed: (buildContext) {}, // Use the path from the XFile
        ), // PreviewScreen
      ), // MaterialPageRoute
    );
  } on CameraException catch (e) {
    _showCameraException(e);
  }
}

Future<void> _onGalleryPressed() async {
  if (!mounted) return; // Ensure the widget is still mounted

  BuildContext? context = this.context;
  try {
    XFile? image = await ImagePicker().pickImage(source: ImageSource.gallery);
    if (image != null && context != null) {
      Navigator.push(
        context,
        MaterialPageRoute(
```



```
287             MaterialPageRoute(
288               builder: (context) => PreviewScreen(
289                 imgPath: image.path,
290                 onCapturePressed: (buildContext) {},
291               ), // PreviewScreen
292             ), // MaterialPageRoute
293           );
294         }
295       } catch (e) {
296         print('Error selecting image from gallery: $e');
297       }
298     }
299   }
300
```

This Dart code defines a Flutter screen (CameraAlt) that integrates with the camera functionality and provides options for taking a photo or selecting one from the gallery. It uses the camera package for camera interactions and the image_picker package for gallery access.

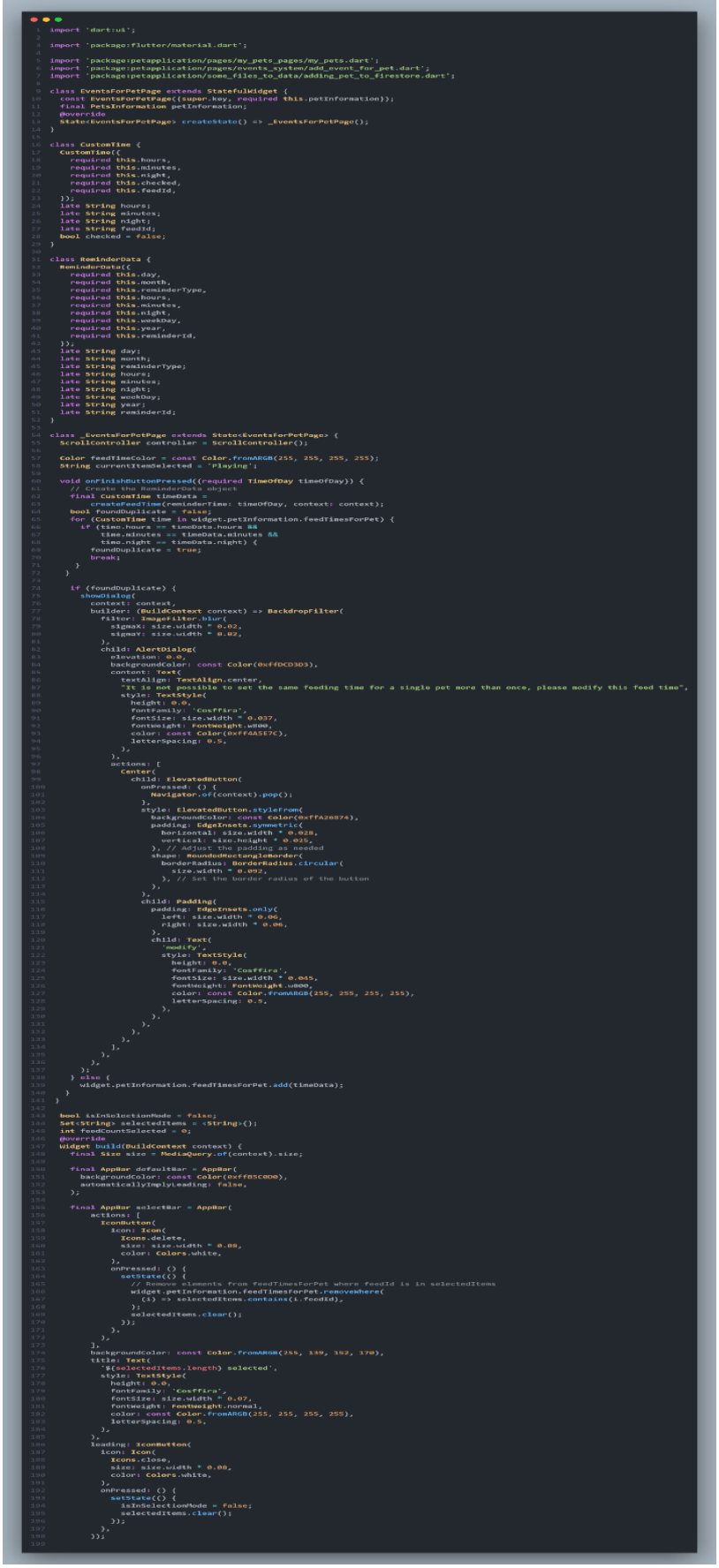
- CameraAlt is a stateful widget without any parameters.
- The state of this widget is managed by _CameraAltState.

Initial Setup and State Management

- CameraController, cameras, selectedCameraIndex, imagePath, and _initializeControllerFuture are declared to manage the camera setup.
- initState initializes the camera by calling initializeCamera.

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

import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:get/get.dart';

class TipsOfPhoto extends StatelessWidget {
  const TipsOfPhoto({super.key});

  @override
  Widget build(BuildContext context) {
    Size size = MediaQuery.of(context).size;
    return Scaffold(
      backgroundColor: const Color(0xffFCFBC5),
      appBar: AppBar(
        elevation: 1,

        automaticallyImplyLeading: true,

        //iconTheme: IconThemeData.fallback(),
        forceMaterialTransparency: true,
        toolbarOpacity: 1,
        toolbarHeight: 70.h,

        leadingWidth: 60.w,

        leading: IconButton(
          icon: const Icon(
            Icons.arrow_back_rounded,
            color: Color.fromARGB(120, 70, 94, 124),
          ),
          onPressed: () {
            Get.back();
          },
          iconSize: 33.0,
          padding: const EdgeInsets.only(
              left: 0.0), // Set the size of the arrow icon
        ),
      ),
      extendBodyBehindAppBar: false,
      body: SingleChildScrollView(
        child: SafeArea(
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Center(
                  child: Padding(
                    padding:  EdgeInsets.only(
                      top: size.height*0.045
                    ),
                    child: Text(
                      overflow: TextOverflow.ellipsis,
                      ' Pet Photo Tips for \n Precision',
                      style: TextStyle(
                          fontFamily: 'Cooffira',
                          fontSize: 45.sp,
                          // Reduce line height (leading)
                          height: 1.1, // Adjust the value as needed
                          // Reduce word spacing
                          wordSpacing:  2, // Adjust the value as needed
                          color: const Color(0xff443E7C),
                          fontWeight: FontWeight.w600,
                          decoration: TextDecoration.underline,
                          decorationThickness: 1,
                          decorationColor: const Color(0xff2A5B6C)),
                      textAlign: TextAlign.center,
                    ),
                  )),
              SizedBox(
                height: 100.h,
              ),
              Padding(
                padding: const EdgeInsets.only(left: 20),
                child: Text.rich(
                  TextSpan(
                    text: 'Welcome to Our Pet Classification \n Feature! ',
                    style: TextStyle(
                      fontFamily: 'Cooffira',
                      fontSize: 55.sp,
                      color: const Color(0xff626874),
                      fontWeight: FontWeight.w600,
                      // Reduce line height (leading)
                      height: 1.2, // Adjust the value as needed
                      // Reduce word spacing
                      wordSpacing:  2, // Adjust the value as needed
                    ), // Optional custom text style
                    children: <TextSpan>[
                      TextSpan(
                          text:
                              ' For accurate results,\n capturing high-quality images is\n essential. Whether identifying your\n pet\'s type, detecting skin issues, or\n assessing stool conditions, these\n considerations enhance the \n precision of our classification process.', // Use narrow no-break space for better underline alignment
                          style: TextStyle(
                            fontFamily: 'Cooffira',
                            fontSize: 52.sp,
                            color: const Color(0xff707070),
                            fontWeight: FontWeight.w600,
                            // Reduce line height (leading)
                            height: 1.2, // Adjust the value as needed
                            // Reduce word spacing
                            wordSpacing:  3, // Adjust the value as needed
                          )),
                    ],
                  ),
                ),
              ),
              SizedBox(
                height: 50.h,
              ),
              Padding(
                padding: const EdgeInsets.only(left: 20),
                child: Text.rich(
                  TextSpan(
                    text: '1. Head Focus \n',
                    style: TextStyle(
                      fontFamily: 'Cooffira',
                      fontSize: 57.sp,
                      color: const Color(0xff626874),
                      fontWeight: FontWeight.w600,
                      // Reduce line height (leading)
                      height: 1.0, // Adjust the value as needed
                      // Reduce word spacing
                      wordSpacing: -3, // Adjust the value as needed
                    ), // Optional custom text style
                    children: <TextSpan>[
                      TextSpan(
                          text:
                              ' Crucial for type identification, \n focus on your pet\'s head. Clear \n shots of facial features and unique\n markings contribute to accurate \n classification. Use treats or toys to \n engage your pet, capturing  \n captivating headshots for better \n insights', // Use narrow no-break space for better underline alignment
                          style: TextStyle(
                            fontFamily: 'Cooffira',
                            fontSize: 52.sp,
                            color: const Color(0xff707070),
                            fontWeight: FontWeight.w600,
                            // Reduce line height (leading)
                            height: 1.2, // Adjust the value as needed
                            // Reduce word spacing
                            wordSpacing: -1, // Adjust the value as needed
                          )),
                    ],
                  ),
                ),
              ),
              SizedBox(
                height: 70.h,
              ),
              Padding(
                padding: const EdgeInsets.only(left: 20),
                child: Text.rich(
                  TextSpan(
                    text: '2. Varied Angles \n',
                    style: TextStyle(
                      fontFamily: 'Cooffira',
                      fontSize: 57.sp,
                      color: const Color(0xff626874),
                      fontWeight: FontWeight.w600,
                      // Reduce line height (leading)
                      height: 1.0, // Adjust the value as needed
                      // Reduce word spacing
                      wordSpacing: -3, // Adjust the value as needed
                    ), // Optional custom text style
                    children: <TextSpan>[
                      TextSpan(
                          text:
                              ' Capture your pet from different \n angles to provide a comprehensive \n view. Showcase their full body, \n distinctive markings, and unique\n features. Multiple perspectives \n enrich the classification process \n for a holistic understanding.', // Use narrow no-break space for better underline alignment
                          style: TextStyle(
                            fontFamily: 'Cooffira',
                            fontSize: 50.sp,
                            color: const Color(0xff707070),
                            fontWeight: FontWeight.w600,
                            // Reduce line height (leading)
                            height: 1.2, // Adjust the value as needed
                            // Reduce word spacing
                            wordSpacing: -1, // Adjust the value as needed
                          )),
                    ],
                  ),
                ),
              ),
              SizedBox(
                height: 50.h,
              ),
              Padding(
                padding: const EdgeInsets.only(left: 20, bottom: 70),
                child: Text.rich(
                  TextSpan(
                    text: '3. Skin Health \n',
                    style: TextStyle(
                      fontFamily: 'Cooffira',
                      fontSize: 57.sp,
                      color: const Color(0xff626874),
                      fontWeight: FontWeight.w600,
                      // Reduce line height (leading)
                      height: 1.0, // Adjust the value as needed
                      // Reduce word spacing
                      wordSpacing:  1, // Adjust the value as needed
                    ), // Optional custom text style
                    children: <TextSpan>[
                      TextSpan(
                          text:
                              ' When addressing skin diseases,\n emphasize the head in clear shots \n of affected areas. Adequate lighting \n and minimal shadows assist in \n accurate detection. Zoom in on \n rashes or lesions for detailed \n insights into your pet\'s skin health.',
                          style: TextStyle(
                            fontFamily: 'Cooffira',
                            fontSize: 50.sp,
                            color: const Color(0xff707070),
                            fontWeight: FontWeight.w600,
                            // Reduce line height (leading)
                            height: 1.2, // Adjust the value as needed
                            // Reduce word spacing
                            wordSpacing:  3, // Adjust the value as needed
                          )),
                    ],
                  ),
                ),
              )
            ],
          ),
        ),
      ),
    );
  }
}
```

Photo tips



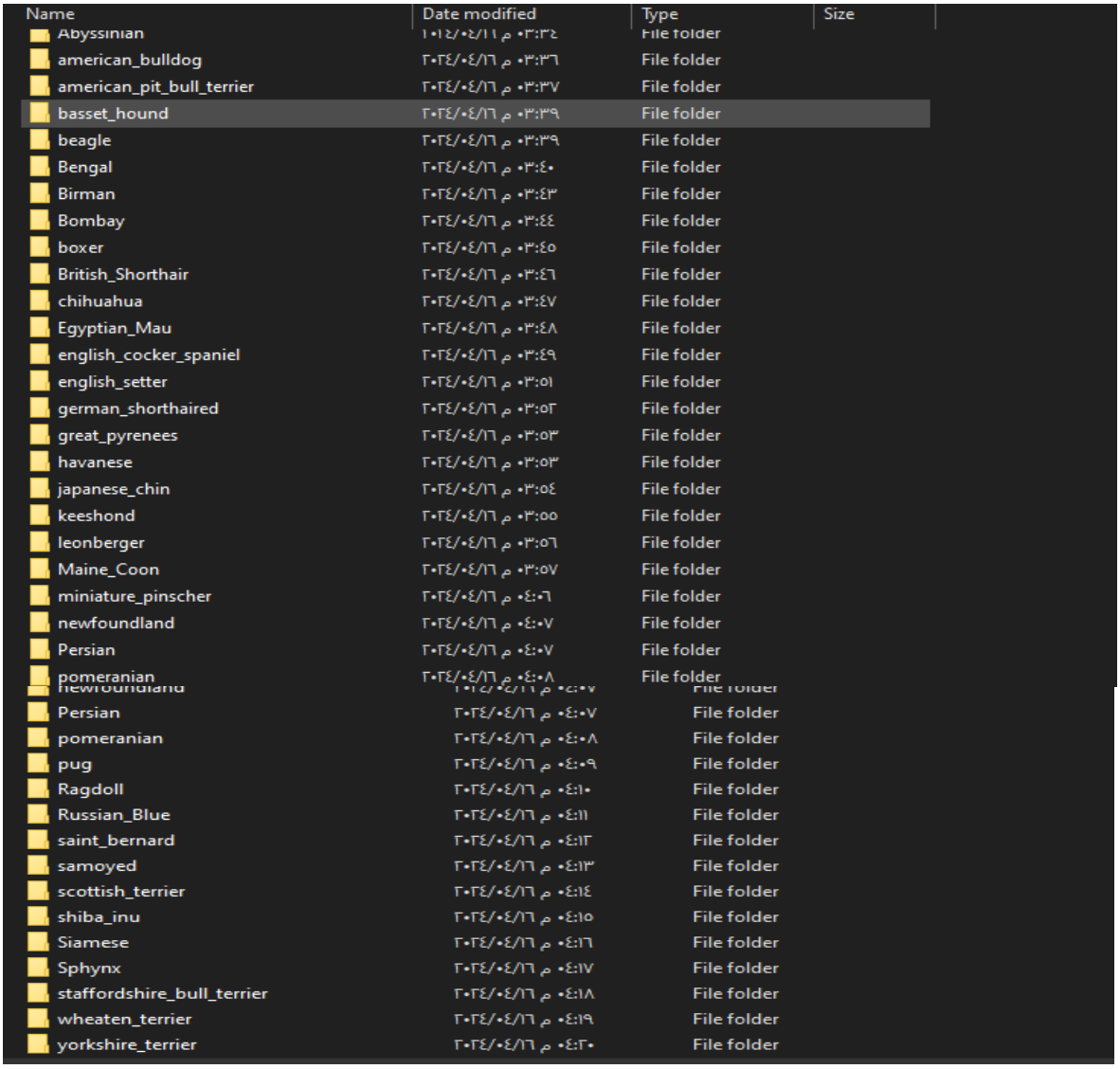
```dart
import 'dart:ui';

import 'package:flutter/material.dart';

import 'package:petapplication/pages/my_pets_pages/my_pets.dart';
import 'package:petapplication/pages/events_system/add_event_for_pet.dart';
import 'package:petapplication/some_files_to_data/adding_pet_to_firestore.dart';

class EventsForPetPage extends StatefulWidget {
  const EventsForPetPage({Super.key, required this.petInformation});
  final PetsInformation petInformation;
  @override
  State<EventsForPetPage> createState() => _EventsForPetPage();
}

class CustomTime {
  CustomTime({
    required this.hours,
    required this.minutes,
    required this.night,
    required this.checked,
    required this.feedId,
  });
  late String hours;
  late String minutes;
  late String night;
  late String feedId;
  bool checked = false;
}

class ReminderData {
  ReminderData({
    required this.day,
    required this.month,
    required this.reminderType,
    required this.hours,
    required this.minutes,
    required this.night,
    required this.weekDay,
    required this.year,
    required this.reminderId,
  });
  late String day;
  late String month;
  late String reminderType;
  late String hours;
  late String minutes;
  late String night;
  late String weekDay;
  late String year;
  late String reminderId;
}

class _EventsForPetPage extends State<EventsForPetPage> {
  ScrollController controller = ScrollController();

  Color feedTimeColor = const Color.fromARGB(255, 255, 255, 255);
  String currentItemSelected = 'Playing';

  void onFinishButtonPressed({required TimeOfDay timeOfDay}) {
    // Create the ReminderData object
    final CustomTime timeData =
        createFeedTime(reminderTime: timeOfDay, context: context);
    bool foundDuplicate = false;
    for (CustomTime time in widget.petInformation.feedTimesForPet) {
      if (time.hours == timeData.hours &&
          time.minutes == timeData.minutes &&
          time.night == timeData.night) {
        foundDuplicate = true;
        break;
      }
    }

    if (foundDuplicate) {
      showDialog(
        context: context,
        builder: (BuildContext context) => BackdropFilter(
          filter: ImageFilter.blur(
            sigmaX: size.width * 0.02,
            sigmaY: size.width * 0.02,
          ),
          child: AlertDialog(
            elevation: 0.0,
            backgroundColor: const Color(0xffDCD3D3),
            content: Text(
              textAlign: TextAlign.center,
              "It is not possible to set the same feeding time for a single pet more than once, please modify this feed time",
              style: TextStyle(
                  height: 0.0,
                  fontFamily: 'Cosffira',
                  fontSize: size.width * 0.037,
                  fontWeight: FontWeight.w800,
                  color: const Color(0xff4A5E7C),
                  letterSpacing: 0.5,
                ),
              ),
              actions: [
                Center(
                  child: ElevatedButton(
                      onPressed: () {
                        Navigator.of(context).pop();
                      },
                      style: ElevatedButton.styleFrom(
                        backgroundColor: const Color(0xffA26874),
                        padding: EdgeInsets.symmetric(
                          horizontal: size.width * 0.028,
                          vertical: size.height * 0.025,
                        ), // Adjust the padding as needed
                        shape: RoundedRectangleBorder(
                          borderRadius: BorderRadius.circular(
                            size.width * 0.092,
                          ), // Set the border radius of the button
                        ),
                      ),
                      child: Padding(
                        padding: EdgeInsets.only(
                          left: size.width * 0.06,
                          right: size.width * 0.06,
                        ),
                        child: Text(
                          'modify',
                          style: TextStyle(
                            height: 0.8,
                            fontFamily: 'Cosffira',
                            fontSize: size.width * 0.045,
                            fontWeight: FontWeight.w800,
                            color: const Color.fromARGB(255, 255, 255, 255),
                            letterSpacing: 0.5,
                          ),
                        ),
                      ),
                    ),
                  ),
                ],
              ),
            ],
          ),
        );
    } else {
      widget.petInformation.feedTimesForPet.add(timeData);
    }
  }

  bool isInSelectionMode = false;
  Set<String> selectedItems = <String>{};
  int feedCountSelected = 0;
  @override
  Widget build(BuildContext context) {
    final Size size = MediaQuery.of(context).size;

    final AppBar defaultBar = AppBar(
      backgroundColor: const Color(0xffD5C0D0),
      automaticallyImplyLeading: false,
    );

    final AppBar selectBar = AppBar(
        actions: [
          IconButton(
            icon: Icon(
              Icons.delete,
              size: size.width * 0.08,
              color: Colors.white,
            ),
            onPressed: () {
              setState(() {
                // Remove elements from feedTimesForPet where feedId is in selectedItems
                widget.petInformation.feedTimesForPet.removeWhere(
                  (i) => selectedItems.contains(i.feedId),
                );
                selectedItems.clear();
              });
            },
          ),
        ],
        backgroundColor: const Color.fromARGB(255, 139, 152, 170),
        title: Text(
          '${selectedItems.length} selected',
          style: TextStyle(
            height: 0.8,
            fontFamily: 'Cosffira',
            fontSize: size.width * 0.07,
            fontWeight: FontWeight.normal,
            color: const Color.fromARGB(255, 255, 255, 255),
            letterSpacing: 0.5,
          ),
        ),
        leading: IconButton(
          icon: Icon(
            Icons.close,
            size: size.width * 0.08,
            color: Colors.white,
          ),
          onPressed: () {
            setState(() {
              isInSelectionMode = false;
              selectedItems.clear();
            });
          },
        ));
```

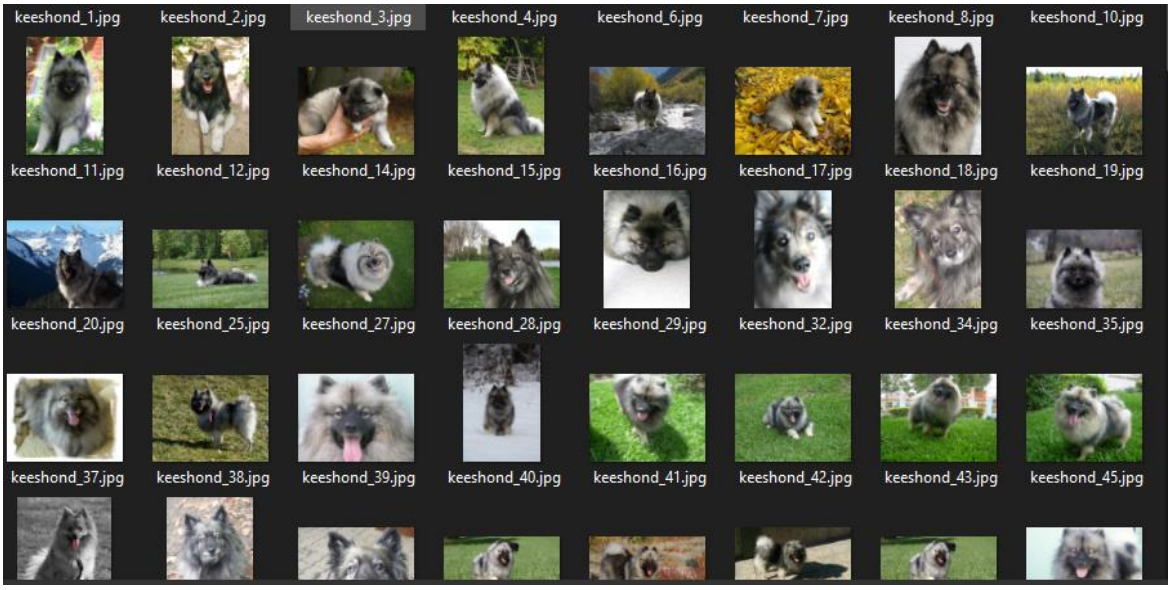
Events for pet

AI implementation:

Dataset1:

The first step is to search for a suitable data set and from a reliable source. Indeed, we found the data that we are working on from Kajal, which contains 37 species of dogs and cats, and we are working on these two species only currently because they are the most widespread through research and studies around the world and each of these species. It contains 200 images so that the model can train on it. The output of all the data was about 7567. Then we worked on the data before entering it into the model. The quality of the data and the sizes of its images were proportional, consistent, and clean from any repetition or errors, which helped us to skip the step of pre-processing the data, which is our first step. The data set is the data partition





Split1:

Dividing the data into training, verification, and testing so that the model can be trained in the training part, which is the largest part, verifying its training through the verification part, and testing the model through the testing part. Let us show you how this is done with the code in detail.

```
In [1]: import os
        import random
        import shutil

        data_path = r"C:\Users\Manga\Desktop\images\images"

        # path to destination folders
        train_folder = os.path.join(data_path, 'train')
        val_folder = os.path.join(data_path, 'validation')
        test_folder = os.path.join(data_path, 'test')

        # Define a list of image extensions
        image_extensions = ['.jpg', '.jpeg', '.png', '.bmp']

        # Create a list of image filenames in 'data_path'
        imgs_list = [filename for filename in os.listdir(data_path) if os.path.splitext(filename)[-1] in image_extensions]
```

“Os”module in Python is used to interact with the operating system. It provides functions for reading or writing to the file system, creating and deleting files or directories, and other operating system dependent functionality.

2- import random

The random module in Python is used to generate random numbers and perform other randomness-related operations.

3-import shutil

The shutil module in Python provides high-level file operations such as copying and moving files and directories, archiving, and

4-data_path = r"C:\Users\Manga\Desktop\images\images"

Here we place the location that contains the data set on our device

train_folder = os.path.join(data_path, 'train')

val_folder = os.path.join(data_path, 'validation')

test_folder = os.path.join(data_path, 'test')

It looks like you're setting up file paths for training, validation, and test data within a larger project or dataset. By using os.path.join(), you're ensuring platform-independent concatenation of directory paths. This is a good practice to ensure your code works across different operating systems without issues.

If data path is the root directory where your data is stored, then train folder, Val folder, and test folder represent the paths to the training, validation, and test data respectively. This organization is common in machine learning projects, where data is often divided into these subsets for model training, evaluation, and testing.

1. Use a set for image_extensions: Since you're checking for membership in **image_extensions**, using a set instead of a list can provide a performance boost, especially if the list of extensions grows.
2. Use the str.casefold() method: To make the extension matching case-insensitive, use the **str.casefold()** method to normalize the extensions and the file names.

3. Consider using pathlib instead of os: **pathlib** provides a more modern and Pythonic way of working with paths and files.

   I replaced os.listdir() with pathlib.Path.iterdir(), which returns an iterator over the files in the directory.

   I used the str.casefold() method to normalize the file extensions and the image extensions, making the matching case-insensitive.

   I used the suffix attribute of the Path object to get the file extension, which is more concise and efficient than using os.path.splitext().

In [2]:
```
random.seed(42)

# Shuffle the list of image filenames
random.shuffle(imgs_list)

# determine the number of images for each set
train_size = int(len(imgs_list) * 0.70)
val_size = int(len(imgs_list) * 0.15)
test_size = int(len(imgs_list) * 0.15)
```

This code snippet shuffles the imgs_list using the random.shuffle() function with a seed of 42. Then, it calculates the sizes for each set (training, validation, and testing) based on the desired percentages. Finally, it determines the indices for each set

Create destination folders if they don't exist

for folder_path in [train_folder, val_folder, test_folder]:

if not os.path.exists(folder_path):

os.makedirs(folder_path)

In [3]:
```
# Create destination folders if they don't exist
for folder_path in [train_folder, val_folder, test_folder]:
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)

# Copy image files to destination folders
for i, f in enumerate(imgs_list):
    if i < train_size:
        dest_folder = train_folder
    elif i < train_size + val_size:
        dest_folder = val_folder
    else:
        dest_folder = test_folder
    shutil.copy(os.path.join(data_path, f), os.path.join(dest_folder, f))
```

In [ ]:

Create destination folders if they don't exist: The code checks if the destination folders (train_folder, val_folder, and test_folder) already

exist. If not, it creates them using the os.makedirs() function.

Copy image files to destination folders: The code iterates through the shuffled list of image filenames (imgs_list) and assigns each image to a destination folder based on its index.

If the index is less than train_size, the image is assigned to the train_folder.

If the index is between train_size and train_size + val_size, the image is assigned to the val_folder.

Otherwise, the image is assigned to the test_folder.

Copy the image files: The shutil.copy() function is used to copy the image files from the source directory (data_path) to the corresponding destination folder.

Model as code

```
In [1]: import numpy as np
        import pandas as pd
        import os
        import matplotlib.pyplot as plt
        import cv2
        import tensorflow as tf
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.applications import MobileNetV2
        from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
        from tensorflow.keras.models import Model
        from tensorflow.keras.optimizers import Adam
        import warnings
        from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, ReLU, MaxPooling2D, Flatten
        from tensorflow.keras.layers import Add
        warnings.filterwarnings("ignore")
```

NumPy(np):

NumPy is a library in Python that provides support for working with multidimensional data.

They are used to create and modify data efficiently and quickly, especially in mathematical operations and numerical conversions.

pandas (pd):

Pandas is a useful library for analyzing data and working with data organized into data structures such as tables.

They can be used to read and write various data files such as CSV and Excel, as well as to clean data and perform statistical analyses.

Matplotlib.pyplot (plt):

Matplotlib is a library for plotting data and creating graphs and charts.

They can be used to display data visually, making the data easier to understand and analyze.

cv2:

OpenCV (cv2) is a library specialized in image processing and computer vision.

They are used to read and write images, convert images between different formats, and apply a variety of operations such as filtering and geometric transformations.

TensorFlow(tf):

TensorFlow is an open source framework for developing artificial intelligence models.

It is used to build and train deep models such as artificial neural networks, computer vision applications, and machine learning.

ImageDataGenerator:

ImageDataGenerator is a tool built into TensorFlow to create synchronized datasets and control data augmentation of images.

It is used to generate diverse datasets from images, which helps improve the model's generalization ability.

MobileNetV2:

MobileNetV2 is a pre-trained neural network model for image classification.

It is used as a basis for building your own classification model, which can be customized to your specific task.

Dense, GlobalAveragePooling2D, Conv2D, BatchNormalization, ReLU, MaxPooling2D, Flatten, Add:

These TensorFlow/Keras layers are used to build the neural network model.

Different layers include different functions such as classification, classification, feature extraction, etc.

Adam:

Adam is an optimization algorithm used to train deep models.

It is used to update the model coefficients based on the individual gaps for each example in the data set.

Warnings:

This property is used to disable notifications or warnings that may appear during code execution.

Usually used when notifications are unnecessary or appear frequently without much importance.

```
2]: train_path = r'J:\together_aug\train'
    test_path = r'J:\together_aug\test'
    valid_path = r'J:\together_aug\validation'
```

The location of training, verification and testing in the data set

```
[3]: from tensorflow import keras
     train_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory(trai
     test_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory( test
     valid_batches =ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory( vali
```

This code generates datasets of images for use in training and testing a neural network model. Let me explain each part of the code:

from tensorflow import keras:

The keras module is imported from the TensorFlow library. This module provides a high-level interface for building and training deep models using TensorFlow.

train_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory(train_path, target_size=(224,224), batch_size=32):

A training dataset is generated using ImageDataGenerator.

preprocessing_function=keras.applications.mobilenet.preprocess_input Defines the preprocessing function to apply to images. In this case, the pre-processing function previously defined in the MobileNet model is used.

flow_from_directory(train_path, target_size=(224,224), batch_size=32):

train_path: Path to the training images folder.

target_size=(224,224): Target size for images after resizing.

batch_size=32: The batch size, i.e. the number of images included in each batch.

test_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory(test_path, target_size=(224,224), batch_size=32, shuffle=False):

A test dataset is created in the same way with the path to the test folder.

valid_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory(valid_path, target_size=(224,224), batch_size=32):

The verification (validation) data set is created in the same way with the validation folder path.

Briefly, ImageDataGenerator is used to generate datasets from images, and a preprocessing function is applied to improve the data quality

before training the model. These groups are used to train and test your model.

```
: base_model = MobileNetV2(weights='imagenet', include_top=False)
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input sh
ape (224, 224) will be loaded as the default.
WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_out
side_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: T
he name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.
```

This line creates a MobileNetV2 model with the pre-loaded weight from ImageNet, using the MobileNetV2() function provided by Keras.

weights='imagenet': Defines the use of preloaded weight from ImageNet. This weight was trained on the ImageNet dataset, and is useful as a good starting point for training the model on different datasets.

include_top=False: Specifies that the top (completed) layer of the form should not be included. Having this option means that the last output layer of the MobileNetV2 model will not be included, and this allows you to add additional layers (such as fully connected layers) on top to suit your particular task, for example, image classification.

In general, this line creates a MobileNetV2 model with the pre-loaded weight from ImageNet, and the top layer of the model is excluded because it can be customized for your task.

```
[5]: # Add custom classification head
     x = base_model.output
     x = GlobalAveragePooling2D()(x)
     output = Dense(7, activation='softmax')(x)
```

This code builds the upper layers of your model after the base layers of the MobileNetV2 model. Let me explain each line:

x = base_model.output:

Here the output of the base_model is used as the starting point for the build.

The output of the base model is stored in a variable x for use in constructing the upper layers.

x = GlobalAveragePooling2D()(x):

The global average pooling layer (GlobalAveragePooling2D) is used to convert hyperdimensional data into 2D data.

This reduces the number of units and reduces the size of the data, while at the same time preserving important information in the image.

output = Dense(3 7, activation='softmax')(x):

The full contact layer (Dense) is used to create the final layer of the model.

The number of units in the layer (37 in this case) and the activation function (softmax) are specified.

37 Here represents the number of different categories into which images should be classified, and which the model is expected to classify.

Softmax activation transforms the results into a probability distribution for different classes, making them more suitable for classification.

```
[6]: # Define model
     model = Model(inputs=base_model.input, outputs=output)
```

This line defines the full model using Model from TensorFlow/Keras. Let me explain each part of the code:

model = Model(inputs=base_model.input, outputs=output):

Model is the class used to build deep models in TensorFlow/Keras.

inputs=base_model.input: Here the inner layer of the model is specified as input for the full model. base_model.input is the previously defined input for the MobileNetV2 model.

outputs=output: Here the final layer of the model is specified as the output of the full model. output is the last layer created after the base layers of the MobileNetV2 model.

In this way, the base layers of the MobileNetV2 model are combined with the upper layers you added (containing the fully connected layer and softmax activation) to create a complete model that can be used to classify images into the different classes.

```
']: # Freeze base layers
    for layer in base_model.layers:
        layer.trainable = False
```

This code freezes (suspends) layers of the base_model, by setting the trainable property to False for each layer in the base model. Let me explain this in more detail:

for layer in base_model.layers::

This loop is used to iterate through each layer in the base_model.

base_model.layers is a list containing all the layers in the base model.

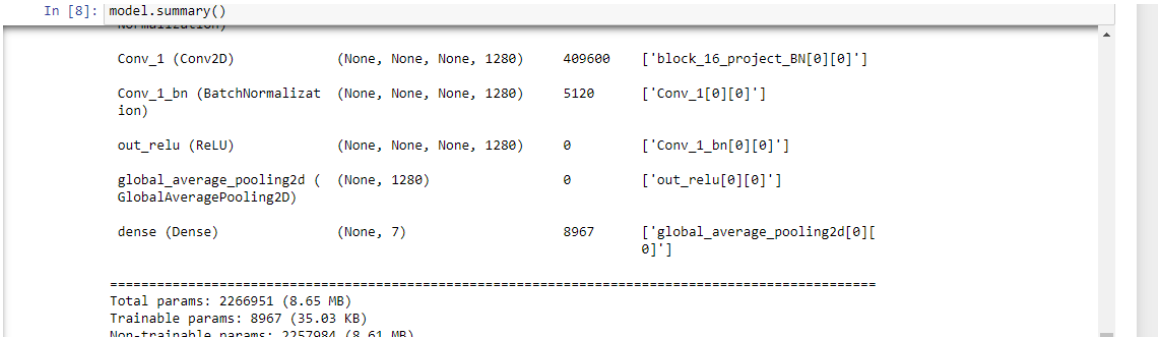
layer.trainable = False:

For each layer in the basic model, the trainable property is set to False.

When the value of trainable is False, it means that the layer will not be updated during the training process, in other words, the weights of the layers will not change.

Freezing the base layers of the base model is usually done in cases

where you have pre-trained base layers and want to use them as an operator to improve performance on your own data without changing it significantly.



```
In [8]: model.summary()
```

Conv_1 (Conv2D)	(None, None, None, 1280)	409600	['block_16_project_BN[0][0]']
Conv_1_bn (BatchNormalization)	(None, None, None, 1280)	5120	['Conv_1[0][0]']
out_relu (ReLU)	(None, None, None, 1280)	0	['Conv_1_bn[0][0]']
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0	['out_relu[0][0]']
dense (Dense)	(None, 7)	8967	['global_average_pooling2d[0][0]']

```
==================================================================================================
Total params: 2266951 (8.65 MB)
Trainable params: 8967 (35.03 KB)
Non-trainable params: 2257984 (8.61 MB)
```

model.summary() is a function in TensorFlow/Keras that is used to display a summary of the model, including the model configuration, the size of the output from each layer, and the number of trainable parameters in the model.

The summary provides you with an overview of the model's structure and can help you verify that the model is built correctly according to your intended design.

The form summary can include information such as:

The total number of layers in the model.

The output format of each layer, including the output size and number of channels (if 3D).

The number of trainable parameters in each layer.

By looking at the model summary, you can verify that the model is built correctly according to your design, and that the layers are connected appropriately.

```
In [9]: model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

This code modifies the training algorithm for your deep model known as model. Let me explain each piece of code:

model.compile():

This function defines how the model should be trained and what should be measured during the training process.

optimizer=Adam(learning_rate=0.0001):

This part sets the optimization algorithm to update the model weights during training.

In this case, the Adam algorithm was used, and the learning rate was set to 0.0001.

loss='categorical_crossentropy':

This part sets the loss function that the model should try to minimize during training.

In a multi-class classification problem, such as classifying images into different categories, the loss function 'categorical_crossentropy' is usually used.

metrics=['accuracy']:

This part specifies the metric that should be measured during the training process.

In this case, the accuracy of the model is measured, i.e. the ratio between the number of correct answers and the total number of samples.

In short, model.compile() defines how to train the model, including the algorithm used to improve the model, the loss function that should be minimized, and the metric that should be measured during training.

```
.0]: history = model.fit(train_batches, validation_data=valid_batches, epochs=100, callbacks=[keras.callbacks.EarlyStopping(monitor='v
```

```
Epoch 1/100
WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTe
nsorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.execut
ing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

162/162 [==============================] - 99s 595ms/step - loss: 2.9368 - accuracy: 0.2776 - val_loss: 2.1433 - val_accuracy:
0.5135
Epoch 2/100
162/162 [==============================] - 95s 587ms/step - loss: 1.6329 - accuracy: 0.6853 - val_loss: 1.2954 - val_accuracy:
0.7527
Epoch 3/100
162/162 [==============================] - 95s 588ms/step - loss: 1.0292 - accuracy: 0.8225 - val_loss: 0.9080 - val_accuracy:
0.8240
Epoch 4/100
162/162 [==============================] - 97s 596ms/step - loss: 0.7477 - accuracy: 0.8668 - val_loss: 0.7146 - val_accuracy:
0.8475
Epoch 5/100
162/162 [==============================] - 99s 612ms/step - loss: 0.5963 - accuracy: 0.8867 - val_loss: 0.6026 - val_accuracy:
0.8583
Epoch 6/100
0.9143
Epoch 20/100
162/162 [==============================] - 95s 585ms/step - loss: 0.1758 - accuracy: 0.9625 - val_loss: 0.2881 - val_accuracy:
0.9134
Epoch 21/100
162/162 [==============================] - 95s 587ms/step - loss: 0.1674 - accuracy: 0.9658 - val_loss: 0.2831 - val_accuracy:
0.9188
Epoch 22/100
162/162 [==============================] - 95s 587ms/step - loss: 0.1593 - accuracy: 0.9675 - val_loss: 0.2796 - val_accuracy:
0.9179
Epoch 23/100
162/162 [==============================] - 95s 588ms/step - loss: 0.1519 - accuracy: 0.9718 - val_loss: 0.2771 - val_accuracy:
0.9170
Epoch 24/100
162/162 [==============================] - 95s 587ms/step - loss: 0.1450 - accuracy: 0.9735 - val_loss: 0.2721 - val_accuracy:
0.9233
Epoch 25/100
162/162 [==============================] - 95s 587ms/step - loss: 0.1382 - accuracy: 0.9754 - val_loss: 0.2677 - val_accuracy:
0.9188
Epoch 26/100
162/162 [==============================] - 94s 579ms/step - loss: 0.1322 - accuracy: 0.9764 - val_loss: 0.2670 - val_accuracy:
0.9188
Epoch 27/100
162/162 [==============================] - 94s 580ms/step - loss: 0.1266 - accuracy: 0.9787 - val_loss: 0.2642 - val_accuracy:
0.9188
```

This code is used to train the model using training and validation data, and allows you to track the progress of training and validation during each training cycle. Let me explain each piece of code:

1. `history = model.fit(train_batches, validation_data=valid_batches, epochs=100, callbacks=[keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=3)])`:

   - `model.fit()`: This function is used to train the model.

   - `train_batches`: The data set that will be used to train the model. They are assumed to be generated using `ImageDataGenerator` and passed to the model.

   - `validation_data=valid_batches`: The dataset that will be used to validate the model during training. This data is used to evaluate the model's performance on data it was not trained on.

- `epochs=100`: The number of complete training cycles on the dataset.

- `callbacks=[keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)]`: List of functions that should be applied during the training process. In this case, `EarlyStopping` was used as a callback function, which stops the training process if there is no improvement in validation accuracy for a specified period of cycles (included in `patience`), which helps in avoiding a training void and increasing Efficiency.

2. `history`: This variable contains the training history, including loss and accuracy during each training session. It can be used to monitor model performance on training and validation data, and can also be used to draw graphs to evaluate training progress.

```
# Save model
model.save("test2MobileNetV2Model.h5")
```

This line is used to save your model in H5 format, which is one of the standard formats for saving models in TensorFlow/Keras. Let me explain how this works:

model.save("test2MobileNetV2Model.h5"): This line is used to save the model in an H5 format file named "test2MobileNetV2Model.h5".

When you run this line, all model structure including layers and weights are saved in the H5 file.

Once you save the model, you can use it whenever you want by loading it again using keras.models.load_model().

Saving the model after training is important because it allows you to reuse the model without having to retrain it again, and allows you to distribute the model to others or use it in production applications.

```
: my_model = tf.keras.models.load_model("test2MobileNetV2Model.h5")
```

```
WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_out
side_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: T
he name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.
```

This line is used to preload a model saved in H5 format using TensorFlow/Keras. Let me explain how this works:

my_model = tf.keras.models.load_model("test2MobileNetV2Model.h5"): This line is used to load the model saved in an H5 file named "test2MobileNetV2Model.h5".

When you run this line, all the model structure including layers and weights are loaded from the H5 file.

After you load the model, you can use it directly to predict data or use it in production applications without having to retrain it.

This allows you to easily interact with the saved model and use it for whatever purpose you need, such as evaluating its performance on new data or using it in production applications.

```
[11]: #Define the function to prepare image
from keras.preprocessing import image
from scipy.special import softmax
def prepare_image(file):
    img = image.load_img(file, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    return tf.keras.applications.mobilenet_v2.preprocess_input(img_array_expanded_dims)
# Path to your image file
file_path = r"C:\Users\Manga\Downloads\dog3.jpg"
# Preprocess the image
preprocessed_image = prepare_image(file_path)
predictions = my_model.predict(preprocessed_image)
# Decode predictions
softmax_probs = softmax(predictions)
predicted_class_index = np.argmax(softmax_probs)
class_labels = train_batches.class_indices
predicted_label = [key for key, value in class_labels.items() if value == predicted_class_index][0]
# Calculate the percentage of the image belonging to the predicted class

#percentage = softmax_probs[0][predicted_class_index] * 100

# Display results
#print("Predicted class:", predicted_label, "-",softmax_probs * 100, ".3f%" )
formatted_probs = ["{:.3f}%".format(prob * 100) for prob in softmax_probs[0]]
print(f'Predicted class: {predicted_label} - {formatted_probs}')
```

```
1/1 [==============================] - 0s 122ms/step
Predicted class: boxer - ['2.597%', '2.597%', '2.597%', '2.597%', '2.597%', '2.597%', '2.597%', '2.597%', '2.597%', '2.597%',
'2.597%', '2.599%', '2.909%', '2.603%', '2.602%', '2.598%', '6.132%', '2.597%', '2.597%', '2.597%', '2.597%', '2.597%', '2.59
7%', '2.597%', '2.597%', '2.597%', '2.597%', '2.597%', '2.597%', '2.599%', '2.628%', '2.597%', '2.597%', '2.597%', '2.614%',
'2.598%', '2.597%']
```

The code I provided loads an image, prepares it to be compatible with the previously trained MobileNetV2 model, and then directs it to the model to predict which class the image belongs to.

But there are some things to consider:

You must ensure that the file path to which the step file_path is passed is correct. In the current case, you must ensure that the path is written correctly and does not contain errors.

After the image is loaded and converted to a MobileNetV2 compatible format, the previously loaded model (my_model) must be ready to receive this image and make predictions on it. Make sure the model is loaded correctly and is ready for forecasting.

After obtaining the predictions from the model, the predictions are decoded to obtain the predicted image class. You are supposed to compare the index returned from predict() with the index of the training classes (train_batches.class_indices) to get the actual label of the class.

Model 2

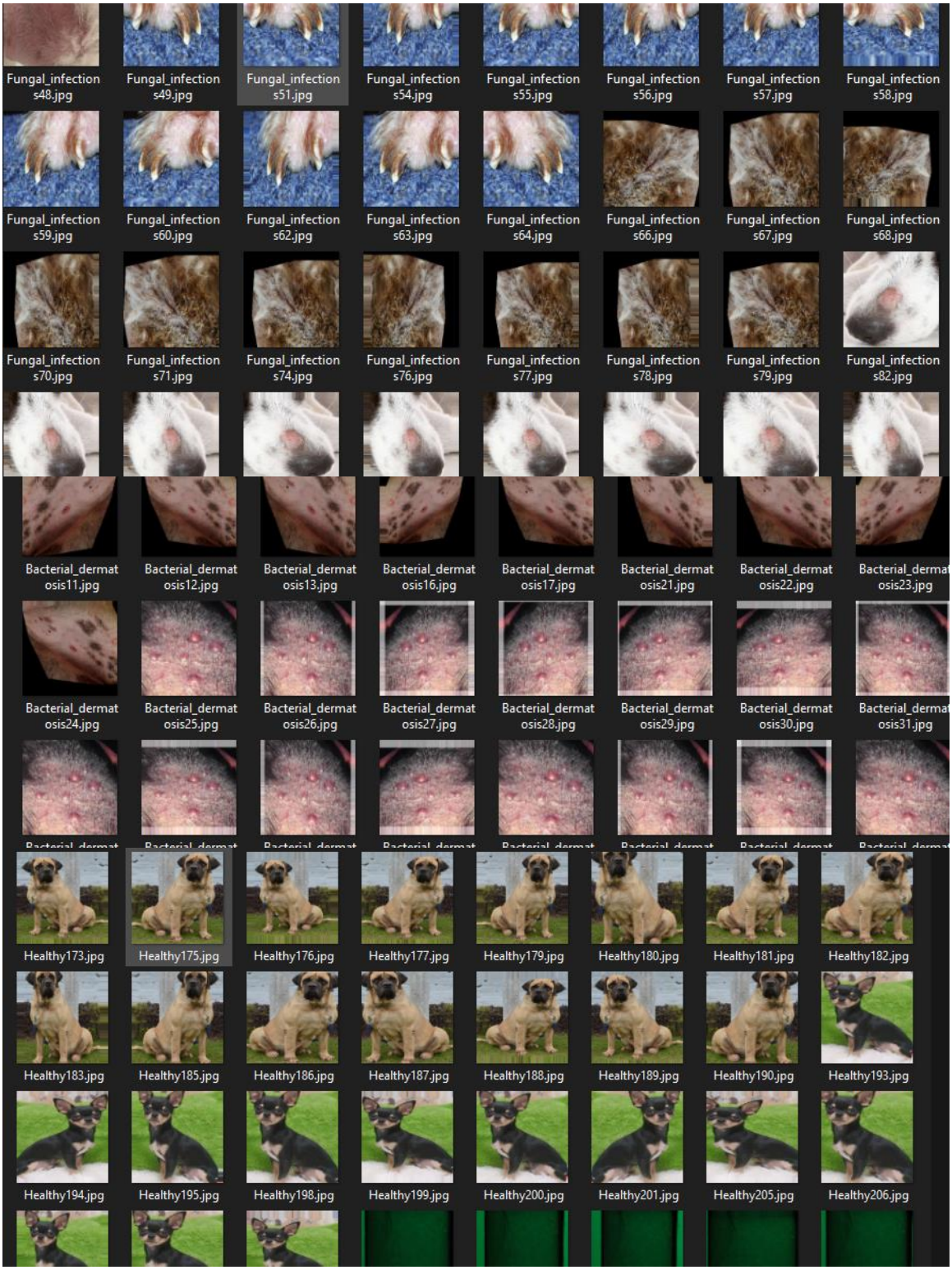
Dog breeders face a major problem, which is the difficulty and inability to recognize whether the animal has a skin disease or not. This may cause the animal's condition to deteriorate, and it is also possible for it to be transmitted to humans as well. Therefore, the second model in this section is to help animal owners be able to Identification at home by photographing the animal's skin, and this model can easily identify whether it is sick or healthy

Dataset2

We searched for a data set that was suitable for our goal and was also reliable. Through the search, we found a data set that contained 3 diseases and another that contained 4

diseases. Therefore, we combined the two so that we could work on a larger number of diseases, especially for dogs currently. These diseases are Bacterial_dermatosis,

Fungal_infections, and Hypersensitivity_allergic_dermatosis. And healthy, keratosis, nasal discharge, and skin lesions, and the result of the merging became the data that we are working on, about 15,532 images.



Argumentation2

Data augmentation :

is a technique that can be used to artificially expand the size of a training set by creating modified data from the existing one. It is a good practice to use DA if you want to prevent overfitting, or the initial dataset is too small to train on, or even if you want to squeeze better performance from your model

data augmentation is not only used to prevent overfitting. In general, having a large dataset is crucial for the performance of both ML and Deep Learning (DL) models. However, we can improve the performance of the model by augmenting the data we already have. It means that data augmentation is also good for enhancing the model's .performance

In general, DA is frequently used when building a DL model. That is why throughout this article, we will mostly talk about performing data augmentation with various DL frameworks. Still, you should keep in .mind that you can augment the data for the ML problems as well

You can augment:

Audio

Text

Images

Any other types of data

We will focus on image augmentations as those are the most popular ones. Nevertheless, augmenting other types of data is as efficient and easy. That is why it's good to remember some common techniques which can be performed to augment the data

Data augmentation techniques

We can apply various changes to the initial data. For example, for images, we can us

Geometric transformations – you can randomly flip, crop, rotate or translate images, and that is just the tip of the iceberg

Color space transformations – change RGB color channels, intensify any color

Kernel filters – sharpen or blur an image

Random Erasing – delete a part of the initial image

Mixing images – basically, mix images with one another. Might be counterintuitive, but it works

For text there are:

Word/sentence shuffling

Word replacement – replace words with synonyms

Syntax-tree manipulation – paraphrase the sentence to be grammatically correct using the same words

For audio augmentation, you can use:

Noise injection

Shifting

Changing the speed of the tape

And many more

Moreover, the greatest advantage of the augmentation techniques is that you may use all of them at once. Thus, you may get plenty of unique samples of data from the initial one

We performed an augmentation for the dogs skin diseases , and there :the code and its results: Importing the required libraries

In [1]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from matplotlib import cm
```

The code imports TensorFlow, the ImageDataGenerator class from Keras for image augmentation, and other necessary libraries such as os, NumPy, matplotlib, and PIL (Python Imaging Library).

2- Defining the dataset path and constants:

```
In [2]: dataset_path = r"C:\Users\doaa\Desktop\Dogs\Bacterial_dermatosis"

In [4]: BATCH_SIZE = 230
        IMG_SHAPE  = 300
```

Here, the dataset path variable is the path to the directory containing the Wheaten Terrier images. BATCH_SIZE defines the number of images to process at each iteration, and IMG_SHAPE specifies the target size of the images for augmentation.

3. Defining a function to plot images:

```
In [5]: # This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
        def plotImages(images_arr):
            fig, axes = plt.subplots(1, 5, figsize=(20,20))
            axes = axes.flatten()
            for img, ax in zip(images_arr, axes):
                ax.imshow(img)
            plt.tight_layout()
            plt.show()
```

This function takes an array of images and plots them in a grid format using matplotlib.

4. Creating an ImageDataGenerator for image augmentation:

```
image_gen_train = ImageDataGenerator(
      rescale=1./255,
      #rotation_range=90,
      #brightness_range=[0.2,1.0],
      #width_shift_range=0.2,
      #height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

An instance of the ImageDataGenerator class is created with various augmentation parameters. In this example, the images are rescaled, sheared, zoomed, and horizontally flipped.

5. Generating augmented images from the dataset:

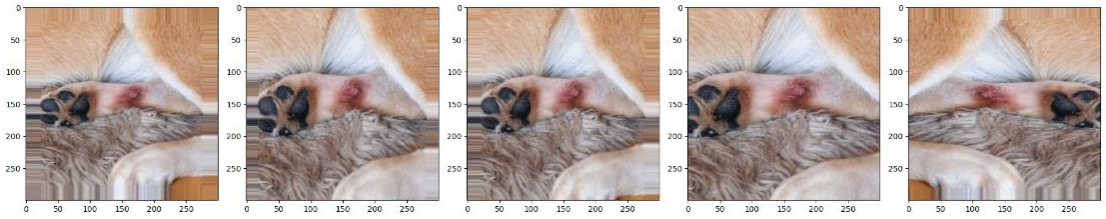
```
dataset_gen = image_gen_train.flow_from_directory(batch_size=BATCH_SIZE,
                                                  directory=dataset_path,
                                                  shuffle=True,
                                                  target_size=(IMG_SHAPE,IMG_SHAPE),
                                                  class_mode='binary')
```

```
Found 94 images belonging to 1 classes.
```

The flow_from_directory function is used to generate augmented images from the dataset directory. It takes parameters such as batch size, directory path, target image size, and class mode. The function returns a generator object that can be used to retrieve augmented images in batches.

6. Plotting augmented images:

```
augmented_images = [dataset_gen[0][0][0] for i in range(5)]
plotImages(augmented_images)
```



Here, we retrieve the first batch of augmented images from the generator and pass them to the plot Images function to visualize the augmentation effects.

7. Saving augmented images:

```
counter = 1
#range(num), Num is the numbers of the images in folder
for imgNum in range(94):
  #range(num), Num is the numbers of augumented images for one image
  augmented_images = [dataset_gen[0][0][imgNum] for i in range(24)]
  for img in augmented_images:
    random_array = (img) * 255
    random_array = random_array.astype(np.uint8)
    random_image = Image.fromarray(random_array)
    random_image.save(r"C:\Users\doaa\Desktop\Dogs\Bacterial_dermatosis\Bacterial_dermatosis"+str(counter)+'.jpg')
    counter+=1
```

In this loop, we iterate through each image in the dataset and generate 15 augmented versions for each image. The augmented images are saved with incremented filenames. The images are converted back to the original pixel range, converted to the np.uint8 data type, and then saved using the PIL library

Split2:

We divided the data in the first model without any difference in the code, so that it turns into a training part with a percentage of 70 percent, a verification part with a percentage of 15 percent, and a test part with a percentage of 15 percent.

Model as code

```
In [1]: import numpy as np
        import pandas as pd
        import os
        import matplotlib.pyplot as plt
        import cv2
        import tensorflow as tf
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.applications import MobileNetV2
        from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
        from tensorflow.keras.models import Model
        from tensorflow.keras.optimizers import Adam
        import warnings
        from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, ReLU, MaxPooling2D, Flatten
        from tensorflow.keras.layers import Add
        warnings.filterwarnings("ignore")
```

Libraries and Tools:

Numpy and Pandas:

numpy is used for high-performance array operations.

pandas is used for data analysis and handling tabular data.

OS:

Used for interacting with the file system.

Matplotlib:

matplotlib.pyplot is used for plotting graphs.

OpenCV (cv2):

Used for image and video processing.

TensorFlow and Keras:

tensorflow is a deep learning framework.

keras is a high-level API built on top of tensorflow.

ImageDataGenerator:

Used for generating batches of tensor image data with real-time data augmentation.

MobileNetV2:

A pre-trained deep learning model used for image classification tasks.

Warnings:

warnings.filterwarnings("ignore") is used to ignore warnings that may appear during code execution.

Explanation of the Functions

1. ImageDataGenerator:

Used for preparing and augmenting data on-the-fly during model training by applying transformations such as rotation, zoom, and shift.

2. MobileNetV2:

A pre-trained deep learning model used as a base for building custom image classification models by adding additional layers.

3. Dense and GlobalAveragePooling2D:

Layers used in model building:

Dense: A fully connected neural network layer.

GlobalAveragePooling2D: A layer that reduces the dimensions by taking the average of each feature across the entire image.

4. Model:

Used to define the entire model by combining different layers.

5. Adam:

An optimization algorithm used for training neural networks.

6. Additional Layers (Input, Conv2D, BatchNormalization, ReLU, MaxPooling2D, Flatten, Add):

Layers used for building custom and deeper models:

Input: Input layer for defining the shape of input data.

Conv2D: Convolutional layer used for feature extraction from images.

BatchNormalization: Layer used to speed up training and stabilize the model.

ReLU: Activation function commonly used in neural networks.

MaxPooling2D: Pooling layer used for dimensionality reduction.

Flatten: Layer used to convert a 2D matrix into a 1D vector.

Add: Layer used for merging the outputs of multiple layers.

```
3]: train_path = r'J:\together_aug\train'
    test_path = r'J:\together_aug\test'
    valid_path = r'J:\together_aug\validation'

4]: from tensorflow import keras
    train_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory(trai
    test_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory( test
    valid_batches =ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory( vali

    Found 10883 images belonging to 7 classes.
    Found 2008 images belonging to 7 classes.
    Found 2641 images belonging to 7 classes.
```

Training Directory (train_path):

Contains images and their corresponding labels used to train the model. This is the data the model learns from.

Testing Directory (test_path):

Contains images and their corresponding labels used to evaluate the model after training. This helps to assess how well the model generalizes to unseen data.

Validation Directory (valid_path):

Contains images and their corresponding labels used to validate the model during training. This helps to tune the model and avoid overfitting by monitoring the model's performance on a separate dataset from the training set.

Training Batches (train_batches):

These batches are generated from the training dataset, and the images are preprocessed using the MobileNet preprocessing function.

The images are resized to 224x224 pixels, and each batch contains 32 images.

Testing Batches (test_batches):

These batches are generated from the test dataset, with the same preprocessing and resizing as the training batches.

shuffle=False ensures the images are not shuffled, which is important for consistent evaluation.

The batch size is also set to 32.

Validation Batches (valid_batches):

These batches are generated from the validation dataset with similar preprocessing and resizing.

The batch size is set to 32, and images are shuffled by default.

```
base_model = MobileNetV2(weights='imagenet', include_top=False)
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input sh
ape (224, 224) will be loaded as the default.
WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_out
side_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: T
he name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.
```

MobileNetV2(weights='imagenet', include_top=False) initializes a MobileNetV2 model without the top classification layers and with weights pre-trained on ImageNet.

This setup is useful for transfer learning, allowing you to add custom layers for your specific classification task while leveraging the powerful feature extraction capabilities of MobileNetV2.

[5]:
```
# Add custom classification head
x = base_model.output
x = GlobalAveragePooling2D()(x)
output = Dense(7, activation='softmax')(x)
```

base_model.output:

This refers to the output of the base MobileNetV2 model, which is a tensor representing the high-level features extracted from the input images. Since include_top=False was used, this output does not include the final dense layers originally designed for ImageNet classification.

GlobalAveragePooling2D():

This layer is added to reduce the spatial dimensions of the feature maps. It calculates the average value of each feature map, effectively converting the 3D tensor (height, width, channels) into a 1D tensor (channels). This helps in reducing the number of parameters and mitigating the risk of overfitting.

The output shape after this layer is (batch_size, channels).

Dense(7, activation='softmax'):

This is a fully connected (dense) layer with 7 units, which corresponds to the number of classes in the new classification task.

activation='softmax': The softmax activation function is used in the output layer of a classification model to convert the raw logits into probabilities that sum to 1, which makes it suitable for multi-class classification.

Each unit in this dense layer represents one of the classes, and the output is a probability distribution over the 7classes.

```
[6]: # Define model
     model = Model(inputs=base_model.input, outputs=output)
```

Model Class:

Model is a class from tensorflow.keras.models used to create a Keras model. It allows you to specify the inputs and outputs of the model, effectively connecting the various layers into a coherent model.

inputs=base_model.input:

This specifies the input tensor(s) for the model. By using base_model.input, you are using the input layer of the MobileNetV2 model as the input layer for your new model. This input layer expects images of shape (224, 224, 3) if you used the default input shape for MobileNetV2.

outputs=predictions:

This specifies the output tensor(s) for the model. The predictions tensor is the output of the custom dense layer that you added for classification. It represents the probabilities for each of the 7 classes in your classification task.

Purpose:

Connecting Layers:

This line of code effectively connects the input layer of the MobileNetV2 (the base model) to your custom layers (GlobalAveragePooling2D and Dense with softmax) to form a complete model. The data flows from the input layer through the base model, then through the added layers, and finally to the output layer.

Defining the Final Model:

By defining model = Model(inputs=base_model.input, outputs=predictions), you create a Keras model that includes both the pre-trained base model (MobileNetV2) and your custom classification head. This new model can be compiled, trained, and evaluated as a single entity.

```
]: # Freeze base layers
for layer in base_model.layers:
    layer.trainable = False
```

for layer in base_model.layers:

This line initiates a loop that iterates over each layer in the base model, which is MobileNetV2 in this context. base_model.layers is a list of all the layers that make up the MobileNetV2 model.

layer.trainable = False:

This line sets the trainable attribute of each layer to False.

The trainable attribute determines whether the weights of the layer will be updated during the training process. By setting trainable = False, you are freezing the layer's weights, meaning they will not be updated during backpropagation.

```
In [8]: model.summary()
 Conv_1 (Conv2D)             (None, None, None, 1280)     409600    ['block_16_project_BN[0][0]']

 Conv_1_bn (BatchNormalizat  (None, None, None, 1280)     5120      ['Conv_1[0][0]']
 ion)

 out_relu (ReLU)             (None, None, None, 1280)     0         ['Conv_1_bn[0][0]']

 global_average_pooling2d (  (None, 1280)                 0         ['out_relu[0][0]']
 GlobalAveragePooling2D)

 dense (Dense)               (None, 7)                    8967      ['global_average_pooling2d[0][[
                                                                    0]']

==================================================================================================
Total params: 2266951 (8.65 MB)
Trainable params: 8967 (35.03 KB)
Non-trainable params: 2257984 (8.61 MB)
```

```
In [9]: model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

model.compile():

This method configures the model for training. It specifies how the model should be trained by defining the optimizer, loss function, and evaluation metrics.

optimizer=Adam(learning_rate=0.0001):

Adam: Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. Adam stands for Adaptive Moment Estimation.

learning_rate=0.0001: This sets the learning rate for the optimizer. The learning rate determines how much to change the model in response to the estimated error each time the model weights are updated. A learning rate of 0.0001 is relatively low, which can lead to more stable training, especially when fine-tuning a pre-trained model.

loss='categorical_crossentropy':

The loss function measures how well the model's predictions match the true labels. categorical_crossentropy is used for multi-class classification problems where the labels are one-hot encoded (each label is represented as a binary vector).

categorical_crossentropy calculates the difference between the predicted probability distribution and the true distribution (one-hot encoded labels), and it penalizes the model more when the predicted probability is low for the true class.

metrics=['accuracy']:

Metrics are used to evaluate the performance of the model. accuracy measures the fraction of correctly predicted instances among the total instances. During training and evaluation, the model will output the accuracy as one of the metrics.

Purpose:

Optimizer (Adam):

Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. It adjusts the learning rate for each parameter dynamically, which can lead to better convergence.

Learning Rate:

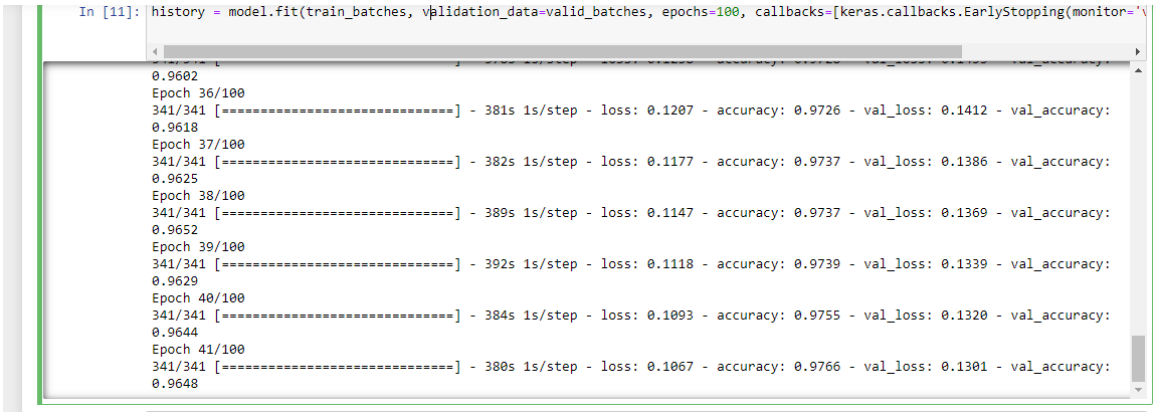
A smaller learning rate (0.0001) is often used when fine-tuning a pre-trained model to prevent drastic updates to the pre-trained weights, which could destabilize the learning process.

Loss Function (categorical_crossentropy):

This loss function is appropriate for multi-class classification problems. It quantifies how well the model's predictions match the actual labels, guiding the optimizer in minimizing the error.

Metrics (accuracy):

Accuracy is a common metric for classification problems, indicating the proportion of correct predictions. It provides an easily interpretable measure of how well the model is performing.



model.fit():

This method is used to train the model on the training dataset (train_batches) and validate it on the validation dataset (valid_batches).

During training, the model learns to minimize the defined loss function

by adjusting its weights using the optimizer.

train_batches:

This parameter is the generator for the training dataset. It provides batches of training data to the model during training.

validation_data=valid_batches:

This parameter specifies the generator for the validation dataset. The model's performance on this dataset is evaluated after each epoch.

epochs=150:

This parameter determines the number of times the entire training dataset is passed forward and backward through the model. In this case, training will run for 150 epochs.

callbacks=[keras.callbacks.EarlyStopping(monitor='accuracy', patience=3)]:

Callbacks are functions that can be applied at various stages of the training process, such as at the end of each epoch.

keras.callbacks.EarlyStopping is a callback function provided by Keras that stops training when a monitored metric (in this case, 'accuracy') has stopped improving.

monitor='accuracy': This specifies the metric to monitor for improvement. In this case, training will stop if the accuracy metric stops improving.

patience=3: This parameter determines the number of epochs to wait before stopping training after the monitored metric has stopped improving. In this case, training will stop after 3 epochs without improvement in accuracy.

```
In [13]: # Save model
         model.save("skin_1MobileNetV2Model.h5")

In [14]: my_model = tf.keras.models.load_model("skin_1MobileNetV2Model.h5")
```

This line of code saves the model (model) to a file named "poop2_MobileNetV2Model.h5".

The model is saved in the HDF5 format, which is a popular file format for storing and exchanging large amounts of data. It's commonly used for saving Keras models due to its simplicity and compatibility.

This line of code loads the saved model from the file "poop2_MobileNetV2Model.h5" and assigns it to the variable my_model.

The load_model function is provided by TensorFlow's Keras API to load a saved model from an HDF5 file.

```
In [16]: # Define function to prepare image
         from keras.preprocessing import image
         def prepare_image(file):
             img = image.load_img(file, target_size=(224, 224))
             img_array = image.img_to_array(img)
             img_array_expanded_dims = np.expand_dims(img_array, axis=0)
             return tf.keras.applications.mobilenet_v2.preprocess_input(img_array_expanded_dims)
         # Path to your image file
         file_path = r"C:\Users\Manga\Desktop\1.jpg"
         # Preprocess the image
         preprocessed_image = prepare_image(file_path)
         predictions = my_model.predict(preprocessed_image)
         # Decode predictions
         predicted_class_index = np.argmax(predictions)
         class_labels = train_batches.class_indices
         predicted_label = [key for key, value in class_labels.items() if value == predicted_class_index][0]

         # Display results
         print("Predicted class:", predicted_label)

         1/1 [==============================] - 10s 10s/step
         Predicted class: Healthy
```

This function, prepare_image, takes an image file path (file) as input.

It loads the image using image.load_img, resizes it to the target size of (224, 224), and converts it to a NumPy array using image.img_to_array.

The array is then expanded to have a batch dimension using np.expand_dims.

Finally, the preprocessed image is returned using tf.keras.applications.mobilenet_v2.preprocess_input, which preprocesses the image according to the MobileNetV2 model's requirements.

file_path is the path to the image file you want to classify.

The image is preprocessed using the prepare_image function defined earlier.

my_model.predict is used to obtain predictions from the loaded model (my_model). The preprocessed image is passed as input to the model, and predictions are obtained.

Explanation:

np.argmax(predictions) finds the index of the class with the highest predicted probability.

train_batches.class_indices contains the mapping of class labels to indices used during training.

The predicted label is obtained by finding the corresponding class label using the index obtained from predictions.

Finally, the predicted class label is printed to the console.

Model 3:

One of the problems that pet owners face is the inability to recognize the disease quickly, and this leads to a delay in treatment and a deterioration in the pet's health condition. This disease may be transmitted to people. Therefore, we have neutered the animal to detect diseases by popping the pet. The pet's owner can photograph him on the application while he is doing this. By giving him the type of disease and information about it and whether the disease is serious

and asking him to go to the doctor or not.

Argumentation3:

We performed an augmentation for the dogs skin diseases , and there the code and its results:

1-Importing the required libraries

```
In [1]: import tensorflow as tf
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        import os
        import numpy as np
        import matplotlib.pyplot as plt
        from PIL import Image
        from matplotlib import cm
```

The code imports TensorFlow, the ImageDataGenerator class from Keras for image augmentation, and other necessary libraries such as os, NumPy, matplotlib, and PIL (Python Imaging Library).

2- Defining the dataset path and constants:

```
In [2]: dataset_path = r"C:\Users\doaa\Desktop\Dogs\Bacterial_dermatosis"

In [4]: BATCH_SIZE = 230
        IMG_SHAPE  = 300
```

Here, the dataset path variable is the path to the directory containing the Wheaten Terrier images. BATCH_SIZE defines the number of images to process at

each iteration, and IMG_SHAPE specifies the target size of the images for augmentation

3-Defining a function to plot images:

```
In [5]: # This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
        def plotImages(images_arr):
            fig, axes = plt.subplots(1, 5, figsize=(20,20))
            axes = axes.flatten()
            for img, ax in zip(images_arr, axes):
                ax.imshow(img)
            plt.tight_layout()
            plt.show()
```

This function takes an array of images and plots them in a grid format using matplotlib.

4-Creating an ImageDataGenerator for image augmentation:

```
image_gen_train = ImageDataGenerator(
      rescale=1./255,
      #rotation_range=90,
      #brightness_range=[0.2,1.0],
      #width_shift_range=0.2,
      #height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

An instance of the ImageDataGenerator class is created with various augmentation parameters. In this example, the images are rescaled, sheared, zoomed, and horizontally flipped.

5-Generating augmented images from the dataset:

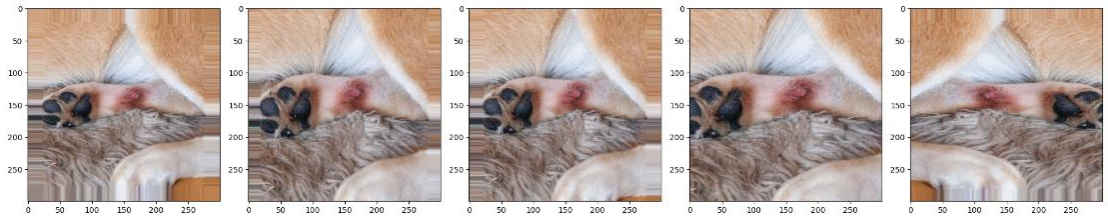
```
dataset_gen = image_gen_train.flow_from_directory(batch_size=BATCH_SIZE,
                                                  directory=dataset_path,
                                                  shuffle=True,
                                                  target_size=(IMG_SHAPE,IMG_SHAPE),
                                                  class_mode='binary')
```

```
Found 94 images belonging to 1 classes.
```

The flow_from_directory function is used to generate augmented images from the dataset directory. It takes parameters such as batch size, directory path, target image size, and class mode. The function returns a generator object that can be used to retrieve augmented images in batches.

6-Plotting augmented images:

```
augmented_images = [dataset_gen[0][0][0] for i in range(5)]
plotImages(augmented_images)
```



Here, we retrieve the first batch of augmented images from the generator and pass them to the plot Images function to visualize the augmentation effects.

7-Saving augmented images:

```
counter = 1
#range(num), Num is the numbers of the images in folder
for imgNum in range(94):
  #range(num), Num is the numbers of augumented images for one image
  augmented_images = [dataset_gen[0][0][imgNum] for i in range(24)]
  for img in augmented_images:
    random_array = (img) * 255
    random_array = random_array.astype(np.uint8)
    random_image = Image.fromarray(random_array)
    random_image.save(r"C:\Users\doaa\Desktop\Dogs\Bacterial_dermatosis\Bacterial_dermatosis"+str(counter)+'.jpg')
    counter+=1
```

In this loop, we iterate through each image in the dataset and generate 15 augmented versions for each image. The augmented images are saved with incremented filenames. The images are converted back to the original pixel range, converted to the np.uint8 data type, and then saved using the PIL library

Split3:

```
In [1]: import os
        import random
        import shutil

        data_path = r"C:\Users\Manga\Desktop\images\images"

        # path to destination folders
        train_folder = os.path.join(data_path, 'train')
        val_folder = os.path.join(data_path, 'validation')
        test_folder = os.path.join(data_path, 'test')

        # Define a list of image extensions
        image_extensions = ['.jpg', '.jpeg', '.png', '.bmp']

        # Create a list of image filenames in 'data_path'
        imgs_list = [filename for filename in os.listdir(data_path) if os.path.splitext(filename)[-1] in image_extensions]
```

“Os”module in Python is used to interact with the operating system. It provides functions for reading or writing to the file system, creating and deleting files or directories, and other operating system dependent functionality.

1- import random

The random module in Python is used to generate random numbers and perform other randomness-related operations.

2-import shutil

The shutil module in Python provides high-level file operations such as copying and moving files and directories, archiving, and more

3-data_path = r"C:\Users\Manga\Desktop\images\images"

Here we place the location that contains the data set on our device

train_folder = os.path.join(data_path, 'train')

val_folder = os.path.join(data_path, 'validation')

test_folder = os.path.join(data_path, 'test')

It looks like you're setting up file paths for training, validation, and test data within a larger project or dataset. By using os.path.join(), you're ensuring platform-independent concatenation of directory paths. This is a good practice to ensure your code works across different operating systems without issues.

If data path is the root directory where your data is stored, then train folder, Val folder, and test folder represent the paths to the training, validation, and test data respectively. This

organization is common in machine learning projects, where data is often divided into these subsets for model training, evaluation, and testing.

1. Use a set for image_extensions: Since you're checking for membership in **image_extensions**, using a set instead of a list can provide a performance boost, especially if the list of extensions grows.
2. Use the str.casefold() method: To make the extension matching case-insensitive, use the **str.casefold()** method to normalize the extensions and the file names.
3. Consider using pathlib instead of os: **pathlib** provides a more modern and Pythonic way of working with paths and files.

   I replaced os.listdir() with pathlib.Path.iterdir(), which returns an iterator over the files in the directory.

   I used the str.casefold() method to normalize the file extensions and the image extensions, making the matching case-insensitive.

I used the suffix attribute of the Path object to get the file extension, which is more concise and efficient than using os.path.splitext().

In [2]:
```
random.seed(42)

# Shuffle the list of image filenames
random.shuffle(imgs_list)

# determine the number of images for each set
train_size = int(len(imgs_list) * 0.70)
val_size = int(len(imgs_list) * 0.15)
test_size = int(len(imgs_list) * 0.15)
```

This code snippet shuffles the imgs_list using the random.shuffle() function with a seed of 42. Then, it calculates the sizes for each set (training, validation, and testing) based on the desired percentages. Finally, it determines the indices for each set

Create destination folders if they don't exist

for folder_path in [train_folder, val_folder, test_folder]:

if not os.path.exists(folder_path):

os.makedirs(folder_path)

In [3]:
```
# Create destination folders if they don't exist
for folder_path in [train_folder, val_folder, test_folder]:
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)

# Copy image files to destination folders
for i, f in enumerate(imgs_list):
    if i < train_size:
        dest_folder = train_folder
    elif i < train_size + val_size:
        dest_folder = val_folder
    else:
        dest_folder = test_folder
    shutil.copy(os.path.join(data_path, f), os.path.join(dest_folder, f))
```

In [ ]:

Create destination folders if they don't exist: The code checks if the destination folders (train_folder, val_folder, and test_folder) already exist. If not, it creates them using the os.makedirs() function.

Copy image files to destination folders: The code iterates through the shuffled list of image filenames (imgs_list) and assigns each image to a destination folder based on its index.

If the index is less than train_size, the image is assigned to the train_folder.

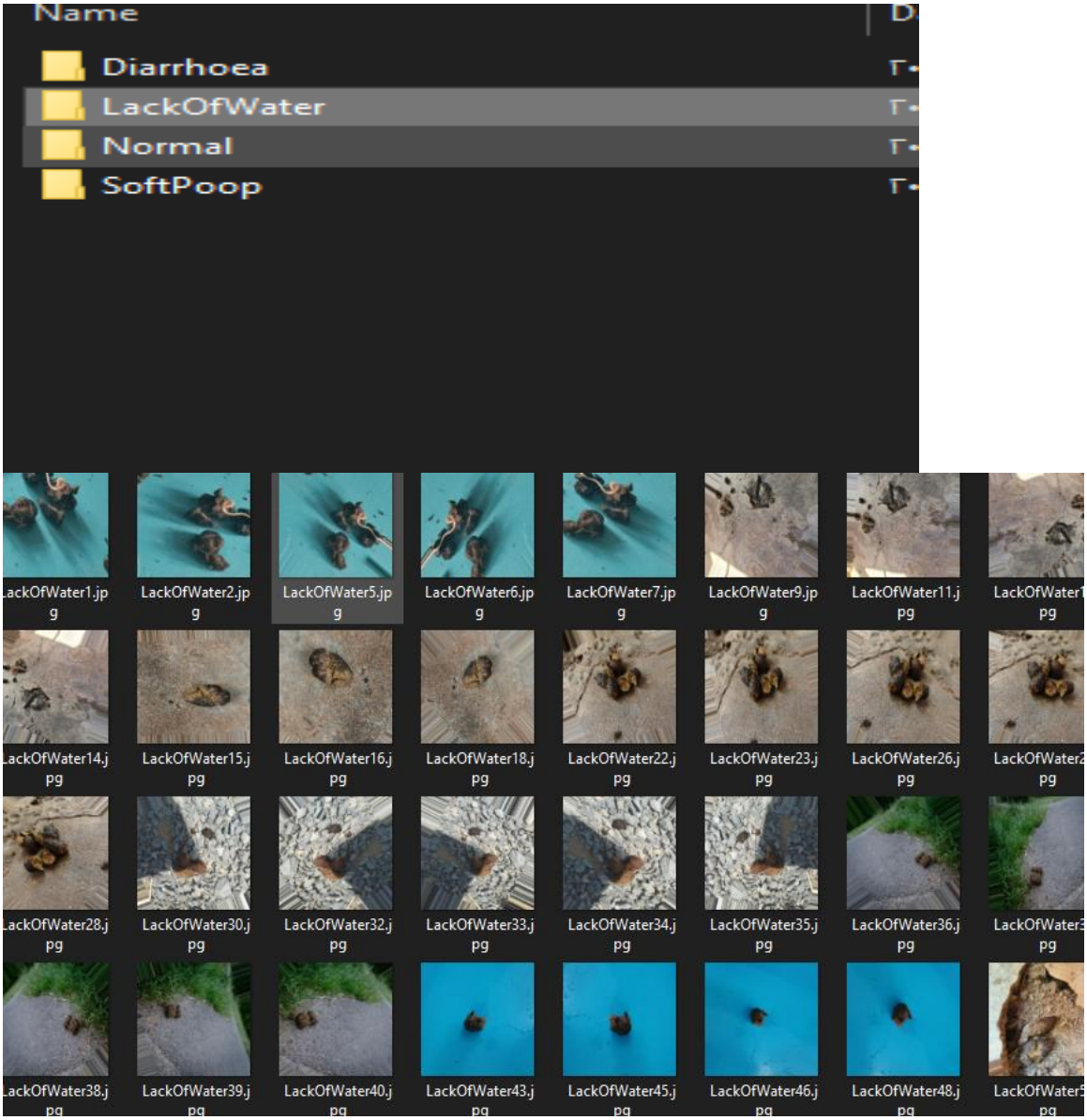
If the index is between train_size and train_size + val_size, the image is assigned to the val_folder.

Otherwise, the image is assigned to the test_folder.

Copy the image files: The shutil.copy() function is used to copy the image files from the source directory (data_path) to the corresponding destination folder.

Dataset3:

One of the most important factors for the good performance of the model that we have is that the data set be large, dedicated, and reliable. Therefore, the data set that we work with has these characteristics and also contains 4 types of diseases. I will show you these types.



Model as code :

In [1]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import warnings
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, ReLU, MaxPooling2D, Flatten
from tensorflow.keras.layers import Add
warnings.filterwarnings("ignore")
```

Libraries and Tools:

Numpy and Pandas:

numpy is used for high-performance array operations.

pandas is used for data analysis and handling tabular data.

OS:

Used for interacting with the file system.

Matplotlib:

matplotlib.pyplot is used for plotting graphs.

OpenCV (cv2):

Used for image and video processing.

TensorFlow and Keras:

tensorflow is a deep learning framework.

keras is a high-level API built on top of tensorflow.

ImageDataGenerator:

Used for generating batches of tensor image data with real-time data augmentation.

MobileNetV2:

A pre-trained deep learning model used for image classification tasks.

Warnings:

warnings.filterwarnings("ignore") is used to ignore warnings that may appear during code execution.

Explanation of the Functions

1. ImageDataGenerator:

Used for preparing and augmenting data on-the-fly during model training by applying transformations such as rotation, zoom, and shift.

2. MobileNetV2:

A pre-trained deep learning model used as a base for building custom image classification models by adding additional layers.

3. Dense and GlobalAveragePooling2D:

Layers used in model building:

Dense: A fully connected neural network
layer.GlobalAveragePooling2D: A layer that reduces the dimensions by taking the average of each feature across the entire image.

4. Model:

Used to define the entire model by combining different layers.

5. Adam:

An optimization algorithm used for training neural networks.

6. Additional Layers (Input, Conv2D, BatchNormalization, ReLU, MaxPooling2D, Flatten, Add):

Layers used for building custom and deeper models:

Input: Input layer for defining the shape of input data.

Conv2D: Convolutional layer used for feature extraction from images.

BatchNormalization: Layer used to speed up training and stabilize the model.

ReLU: Activation function commonly used in neural networks.

MaxPooling2D: Pooling layer used for dimensionality reduction.

Flatten: Layer used to convert a 2D matrix into a 1D vector.

Add: Layer used for merging the outputs of multiple layers.

```
n [2]: train_path = r'J:\stool3\train'
       test_path = r'J:\stool3\test'
       valid_path = r'J:\stool3\validation'
```

Training Directory (train_path):Contains images and their corresponding labels used to train the model. This is the data the model learns from.

Testing Directory (test_path):

Contains images and their corresponding labels used to evaluate the model after training. This helps to assess how well the model generalizes to unseen data.

Validation Directory (valid_path):

Contains images and their corresponding labels used to validate the model during training. This helps to tune the model and avoid overfitting by monitoring the model's performance on a separate dataset from the training set.

```
]: from tensorflow import keras
   train_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory(trai
   test_batches = ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory( test
   valid_batches =ImageDataGenerator(preprocessing_function=keras.applications.mobilenet.preprocess_input).flow_from_directory( vali

   Found 9536 images belonging to 4 classes.
   Found 2048 images belonging to 4 classes.
   Found 2042 images belonging to 4 classes.
```

Training Batches (train_batches):

These batches are generated from the training dataset, and the images are preprocessed using the MobileNet preprocessing function.

The images are resized to 224x224 pixels, and each batch contains 32 images.

Testing Batches (test_batches):

These batches are generated from the test dataset, with the same preprocessing and resizing as the training batches.

shuffle=False ensures the images are not shuffled, which is important for consistent evaluation.

The batch size is also set to 32.

Validation Batches (valid_batches):

These batches are generated from the validation dataset with similar preprocessing and resizing.

The batch size is set to 32, and images are shuffled by default.

```
base_model = MobileNetV2(weights='imagenet', include_top=False)
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input sh
ape (224, 224) will be loaded as the default.
WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_out
side_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: T
he name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.
```

MobileNetV2(weights='imagenet', include_top=False) initializes a MobileNetV2 model without the top classification layers and with weights pre-trained on ImageNet.

This setup is useful for transfer learning, allowing you to add custom layers for your specific classification task while leveraging the powerful feature extraction capabilities of MobileNetV2.

```
he name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.
```

In [6]:
```
# Add custom classification head
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(4, activation='softmax')(x)  # new softmax layer
```

base_model.output:

This refers to the output of the base MobileNetV2 model, which is a tensor representing the high-level features extracted from the input images. Since include_top=False was used, this output does not include the final dense layers originally designed for ImageNet classification.

GlobalAveragePooling2D():

This layer is added to reduce the spatial dimensions of the feature maps. It calculates the average value of each feature map, effectively converting the 3D tensor (height, width, channels) into a 1D tensor (channels). This helps in reducing the number of parameters and mitigating the risk of overfitting.

The output shape after this layer is (batch_size, channels).

Dense(4, activation='softmax'):

This is a fully connected (dense) layer with 4 units, which corresponds to the number of classes in the new classification task.

activation='softmax': The softmax activation function is used in the output layer of a classification model to convert the raw logits into probabilities that sum to 1, which makes it suitable for multi-class classification.

Each unit in this dense layer represents one of the classes, and the output is a probability distribution over the 4 classes.

[6]:
```
# Define model
model = Model(inputs=base_model.input, outputs=output)
```

Model Class:

Model is a class from tensorflow.keras.models used to create a Keras model. It allows you to specify the inputs and outputs of the model, effectively connecting the various layers into a coherent model.

inputs=base_model.input:

This specifies the input tensor(s) for the model. By using base_model.input, you are using the input layer of the MobileNetV2 model as the input layer for your new model. This input layer expects images of shape (224, 224, 3) if you used the default input shape for MobileNetV2.

outputs=predictions:

This specifies the output tensor(s) for the model. The predictions tensor is the output of the custom dense layer that you added for classification. It represents the probabilities for each of the 4 classes in your classification task.

Purpose:

Connecting Layers:

This line of code effectively connects the input layer of the MobileNetV2 (the base model) to your custom layers (GlobalAveragePooling2D and Dense with softmax) to form a

complete model. The data flows from the input layer through the base model, then through the added layers, and finally to the output layer.

Defining the Final Model:

By defining model = Model(inputs=base_model.input, outputs=predictions), you create a Keras model that includes both the pre-trained base model (MobileNetV2) and your custom classification head. This new model can be compiled, trained, and evaluated as a single entity.

```
']: # Freeze base layers
    for layer in base_model.layers:
        layer.trainable = False
```

for layer in base_model.layers:

This line initiates a loop that iterates over each layer in the base model, which is MobileNetV2 in this context. base_model.layers is a list of all the layers that make up the MobileNetV2 model.

layer.trainable = False:

This line sets the trainable attribute of each layer to False.

The trainable attribute determines whether the weights of the layer will be updated during the training process. By setting trainable = False, you are freezing the layer's weights, meaning they will not be updated during backpropagation.

```
In [9]: model.summary()
```

```
Conv_1 (Conv2D)             (None, None, None, 1280)     409600    ['block_16_project_BN[0][0]']

Conv_1_bn (BatchNormalizat  (None, None, None, 1280)     5120      ['Conv_1[0][0]']
ion)

out_relu (ReLU)             (None, None, None, 1280)     0         ['Conv_1_bn[0][0]']

global_average_pooling2d (  (None, 1280)                 0         ['out_relu[0][0]']
GlobalAveragePooling2D)

dense (Dense)               (None, 4)                    5124      ['global_average_pooling2d[0][
                                                                   0]']

==================================================================================================
Total params: 2263108 (8.63 MB)
Trainable params: 5124 (20.02 KB)
Non-trainable params: 2257984 (8.61 MB)
__________________________________________________________________________________________________
```

```
In [9]: model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

model.compile():

This method configures the model for training. It specifies how the model should be trained by defining the optimizer, loss function, and evaluation metrics.

optimizer=Adam(learning_rate=0.0001):

Adam: Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. Adam stands for Adaptive Moment Estimation.

learning_rate=0.0001: This sets the learning rate for the optimizer. The learning rate determines how much to change the model in response to the estimated error each time the model weights are updated. A learning rate of 0.0001 is relatively low, which can lead to more stable training, especially when fine-tuning a pre-trained model.

loss='categorical_crossentropy':

The loss function measures how well the model's predictions match the true labels. categorical_crossentropy is used for multi-class classification problems where the labels are one-hot encoded (each label is represented as a binary vector).

categorical_crossentropy calculates the difference between the predicted probability distribution and the true distribution (one-hot encoded labels), and it penalizes the model more when the predicted probability is low for the true class.

metrics=['accuracy']:

Metrics are used to evaluate the performance of the model. accuracy measures the fraction of correctly predicted instances among the total instances. During training and evaluation, the model will output the accuracy as one of the metrics.

Purpose:

Optimizer (Adam):

Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. It adjusts the learning rate for each parameter dynamically, which can lead to better convergence.

Learning Rate:

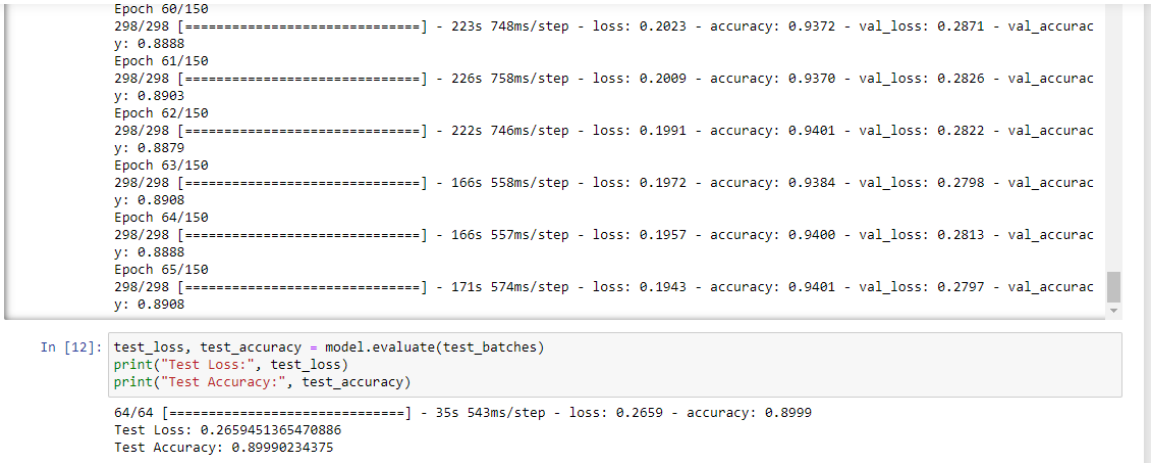
A smaller learning rate (0.0001) is often used when fine-tuning a pre-trained model to prevent drastic updates to the pre-trained weights, which could destabilize the learning process.

Loss Function (categorical_crossentropy):

This loss function is appropriate for multi-class classification problems. It quantifies how well the model's predictions match the actual labels, guiding the optimizer in minimizing the error.

Metrics (accuracy):

Accuracy is a common metric for classification problems, indicating the proportion of correct predictions. It provides an easily interpretable measure of how well the model is performing.



model.fit():

This method is used to train the model on the training dataset (train_batches) and validate it on the validation dataset (valid_batches).

During training, the model learns to minimize the defined loss function by adjusting its weights using the optimizer.

train_batches:

This parameter is the generator for the training dataset. It provides batches of training data to the model during training.

validation_data=valid_batches:

This parameter specifies the generator for the validation dataset. The model's performance on this dataset is evaluated after each epoch.

epochs=150:

This parameter determines the number of times the entire training dataset is passed forward and backward through the model. In this case, training will run for 150 epochs.

callbacks=[keras.callbacks.EarlyStopping(monitor='accuracy', patience=3)]:

Callbacks are functions that can be applied at various stages of the training process, such as at the end of each epoch.

keras.callbacks.EarlyStopping is a callback function provided by Keras that stops training when a monitored metric (in this case, 'accuracy') has stopped improving.

monitor='accuracy': This specifies the metric to monitor for improvement. In this case, training will stop if the accuracy metric stops improving.

patience=3: This parameter determines the number of epochs to wait before stopping training after the monitored metric has stopped improving. In this case, training will stop after 3 epochs without improvement in accuracy.

```
In [13]: # Save model
         model.save("poop2_MobileNetV2Model.h5")

In [4]:  my_model = tf.keras.models.load_model("poop2_MobileNetV2Model.h5")

         WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_out
         side_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

         WARNING:tensorflow:From C:\Users\Manga\anaconda3\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: T
         he name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.
```

This line of code saves the model (model) to a file named "skin_1MobileNetV2Model.h5".

The model is saved in the HDF5 format, which is a popular file format for storing and exchanging large amounts of data. It's commonly used for saving Keras models due to its simplicity and compatibility.

This line of code loads the saved model from the file "skin_1MobileNetV2Model.h5" and assigns it to the variable my_model.

The load_model function is provided by TensorFlow's Keras API to load a saved model from an HDF5 file.

In [5]:
```
# Define function to prepare image
from keras.preprocessing import image
def prepare_image(file):
    img = image.load_img(file, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    return tf.keras.applications.mobilenet_v2.preprocess_input(img_array_expanded_dims)
# Path to your image file
file_path =r"C:\Users\Manga\Desktop\4.jpg"
# Preprocess the image
preprocessed_image = prepare_image(file_path)
predictions = my_model.predict(preprocessed_image)
# Decode predictions
predicted_class_index = np.argmax(predictions)
class_labels = train_batches.class_indices
predicted_label = [key for key, value in class_labels.items() if value == predicted_class_index][0]

# Display results
print("Predicted class:", predicted_label)
```

```
1/1 [==============================] - 1s 1s/step
Predicted class: Normal
```

This function, prepare_image, takes an image file path (file) as input.

It loads the image using image.load_img, resizes it to the target size of (224, 224), and converts it to a NumPy array using image.img_to_array.

The array is then expanded to have a batch dimension using np.expand_dims.

Finally, the preprocessed image is returned using tf.keras.applications.mobilenet_v2.preprocess_input, which preprocesses the image according to the MobileNetV2 model's requirements.

file_path is the path to the image file you want to classify.

The image is preprocessed using the prepare_image function defined earlier.

my_model.predict is used to obtain predictions from the loaded model (my_model). The preprocessed image is passed as input to the model, and predictions are obtained.

Explanation:

np.argmax(predictions) finds the index of the class with the highest predicted probability.

train_batches.class_indices contains the mapping of class labels to indices used during training.

The predicted label is obtained by finding the corresponding class label using the index obtained from predictions.

Finally, the predicted class label is printed to the console.

Chapter 4: Summary, Conclusions and Future Work

Summary: -

1. Determine the type of pet and its species using ai.
2. User can add the picture of its pet using camera of his phone or Choose it from the gallery of the phone.
3. The application displays information about the pet for which a photo was taken or this photo was selected from the gallery, including information about its appearance or species, the diseases it is susceptible to and the type of food it eats.
4. Take a picture of the disease or symptoms appearing on the pet's skin or feces.
5. Determine the type and name of the disease the pet is afflicted with, its symptoms, and how to treat or prevent it.
6. Reminder System: The user adds tasks related to the pet, Specify a time and date for each task so that the application can remind the user of it via notification

Conclusion: -

In conclusion, our innovative application leverages advanced AI technology to enhance pet care by providing comprehensive insights and support to pet owners. By allowing users to identify their pet's species and appearance through a simple photo upload, the app delivers detailed information about the pet's characteristics, potential diseases, and dietary needs. Furthermore, the application extends its functionality to diagnosing health issues by analysing photos of symptoms, offering critical information on the type of disease, its symptoms, and treatment options. The user-friendly design includes a robust reminder system,

ensuring pet owners never miss a critical task related to their pet's well-being. By specifying times and dates, users receive timely notifications to manage their pet's needs effectively. This all-encompassing tool is a must-have for any pet owner, combining cutting-edge AI with practical features to ensure pets receive the best possible care. With this application, managing pet health and maintenance becomes more efficient, accurate, and convenient, empowering pet owners to keep their furry friends healthy and happy.

Challenges: -

One of the biggest challenges is how to analyse the image of the pet and compare it with the learner model.

Also, the method of analysing the image in the app, sending it to the model and identifying it and giving us the correct information about the pet.

Another one is also making the model work with high accuracy. Also, of these challenges was linking the model to the application.

Future Work: -

● There are some features we plan to add it in our app in the future like: -

1. Adding Arabic language to the application.
2. Adding a community for a pet lovers like a Facebook but for users of this app
3. Add a store that sells pet care supplies.

Chapter 5: References

Resourses :-

datasets resources:

- https://data.mendeley.com/datasets/5dbht54kw7/1
- https://www.healthforanimals.org/reports/pet-care-report/global-trends-in-the-pet-population/#ownership

sources of statistics :

- https://www.healthforanimals.org/reports/pet-care-report/global-trends-in-the-pet-population/#ownership
- https://www.forbes.com/advisor/pet-insurance/pet-ownership-statistics/
- https://www.gfk.com/insights/mans-best-friend-global-pet-ownership-and-feeding-trends

Articles:

- https://www.healthforanimals.org/reports/pet-care-report/global-trends-in-the-pet-population/#ownership
- https://www.hindawi.com/journals/js/2023/5602595/

Design:-

- https://xd.adobe.com/view/6000af1a-a0f9-4450-b1aa-8f244208c7d3-3553/screen/703bd864-1d57-402b-b4dd-eaa8b0b20414

- https://www.figma.com/board/xfMuqQ1xxPSpVDy7CEhbEc/the-final-Yuna-userFlow?node-id=0-1&t=Fvk506vg0Sz6oMJj-0
- https://www.canva.com

Flutter &firebase:-

- https://kymoraa.medium.com/to-do-list-app-with-flutter-firebase-7910bc42cf14

Connect Ai model with flutter:

- https://youtu.be/djgMDd8emFg?si=WhB1BqkKh0nVXXv5
- https://youtube.com/playlist?list=PLZoTAELRMXVPBaLN3e-uoVRR9hlRFRfUc&si=7ImM75qLk2WnLPjP

Flutter course:

- https://www.udemy.com/course/learn-flutter-dart-to-build-ios-android-apps/?couponCode=ST21MT61124

GitHub project link:

- https://github.com/soo20/petapplication.git

Scraping resourses:

- https://www.akc.org/
- https://www.purina.co.uk/
- https://www.petmd.com/
- https://en.wikipedia.org/wiki/Main_Page
- https://dogtime.com/
- https://www.thesprucepets.com/
- https://www.trupanion.com/pet-blog/article/american-pit-bull-terrier

- https://www.purina.com/cats/cat-breeds/abyssinian-cat
- https://www.akc.org/