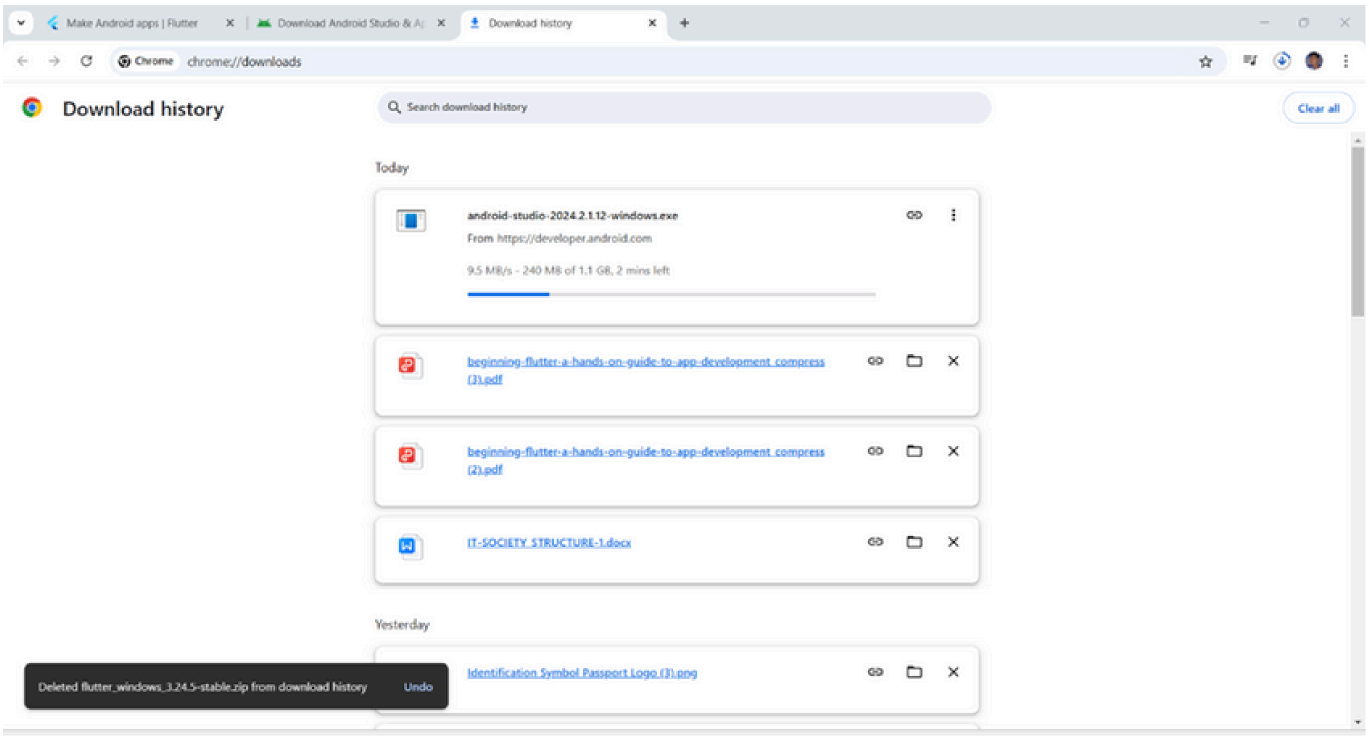
Chapter 1: Installation and Set Up of VS Code

I started by installing and setting up Visual Studio Code, which would be my main development tool for the Pahimakas application. I made sure to install the necessary extensions for Flutter and Dart, ensuring that everything was ready to start coding efficiently and without any setup issues.



Download history

Today

android-studio-2024.2.1.12-windows.exe
From https://developer.android.com
9.5 MB/s - 240 MB of 1.1 GB, 2 mins left

beginning-flutter-a-hands-on-guide-to-app-development compress (3).pdf

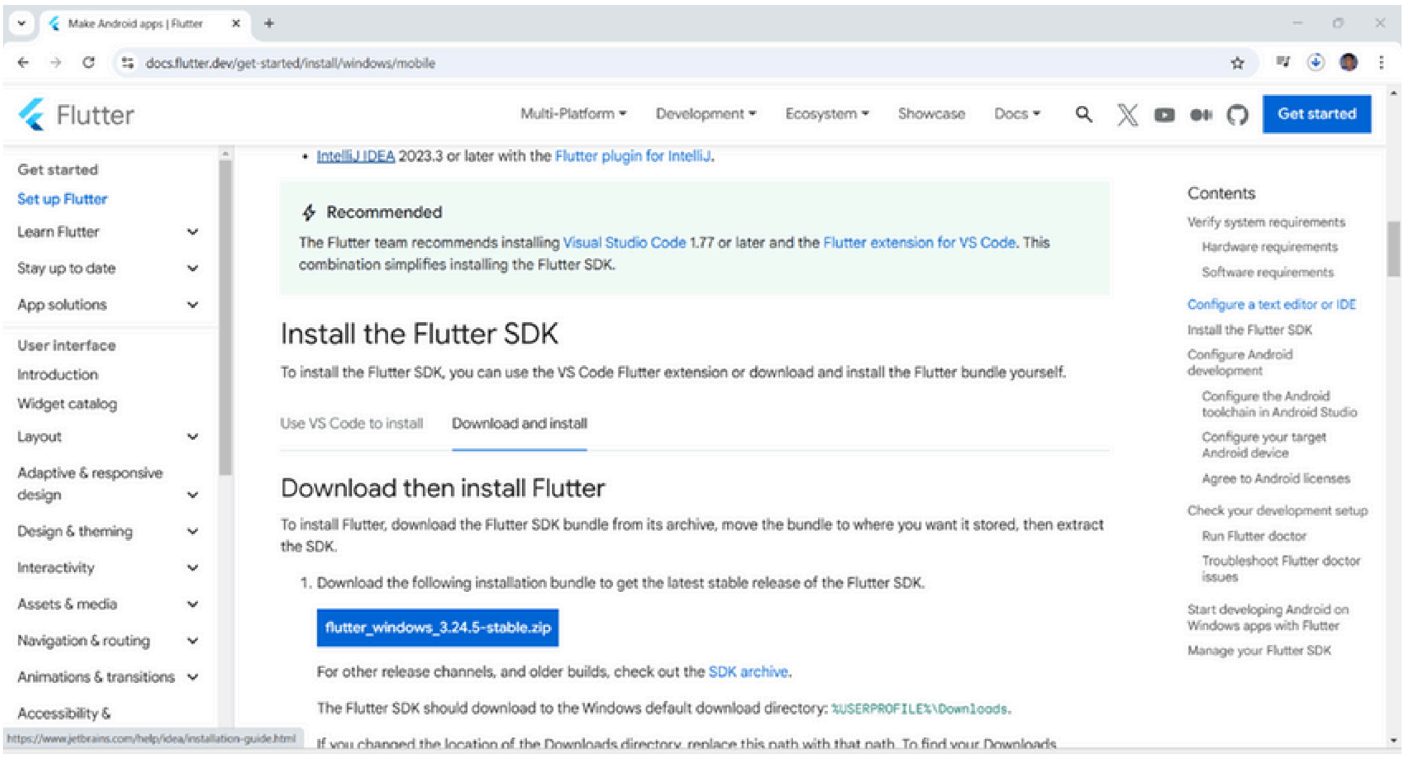
beginning-flutter-a-hands-on-guide-to-app-development compress (2).pdf

IT-SOCIETY STRUCTURE-1.docx

Yesterday

Identification Symbol Passport Logo (3).png

Deleted flutter_windows_3.24.5-stable.zip from download history Undo


Make Android apps | Flutter
docs.flutter.dev/get-started/install/windows/mobile
Flutter
Multi-Platform
Development
Ecosystem
Showcase
Docs
Get started
Get started
Set up Flutter
Learn Flutter
Stay up to date
App solutions
User interface
Introduction
Widget catalog
Layout
Adaptive & responsive design
Design & theming
Interactivity
Assets & media
Navigation & routing
Animations & transitions
Accessibility &
Contents
Verify system requirements
Hardware requirements
Software requirements
Configure a text editor or IDE
Install the Flutter SDK
Configure Android development
Configure the Android toolchain in Android Studio
Configure your target Android device
Agree to Android licenses
Check your development setup
Run Flutter doctor
Troubleshoot Flutter doctor issues
Start developing Android on Windows apps with Flutter
Manage your Flutter SDK
https://www.jetbrains.com/help/idea/installation-guide.html

- IntelliJ IDEA 2023.3 or later with the Flutter plugin for IntelliJ.

Recommended

The Flutter team recommends installing Visual Studio Code 1.77 or later and the Flutter extension for VS Code. This combination simplifies installing the Flutter SDK.

Install the Flutter SDK

To install the Flutter SDK, you can use the VS Code Flutter extension or download and install the Flutter bundle yourself.

Use VS Code to install | Download and install

Download then install Flutter

To install Flutter, download the Flutter SDK bundle from its archive, move the bundle to where you want it stored, then extract the SDK.

1. Download the following installation bundle to get the latest stable release of the Flutter SDK.

flutter_windows_3.24.5-stable.zip

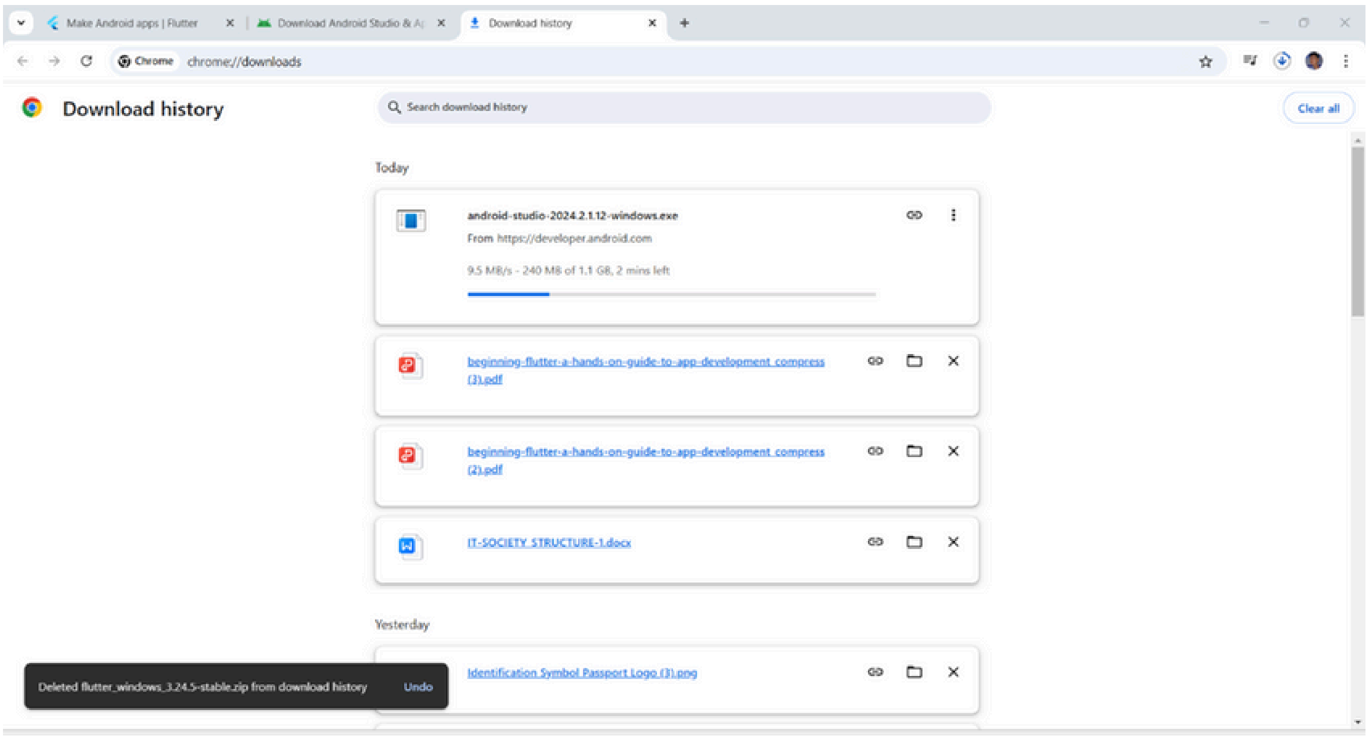
For other release channels, and older builds, check out the SDK archive.

The Flutter SDK should download to the Windows default download directory: `%USERPROFILE%\Downloads`.

If you changed the location of the Downloads directory, replace this path with that path. To find your Downloads

Chapter 1: Installation and Set Up of VS Code

I started by installing and setting up Visual Studio Code, which would be my main development tool for the Pahimakas application. I made sure to install the necessary extensions for Flutter and Dart, ensuring that everything was ready to start coding efficiently and without any setup issues.



Download history

Today

android-studio-2024.2.1.12-windows.exe
From https://developer.android.com
9.5 MB/s - 240 MB of 1.1 GB, 2 mins left

beginning-flutter-a-hands-on-guide-to-app-development_compress (3).pdf

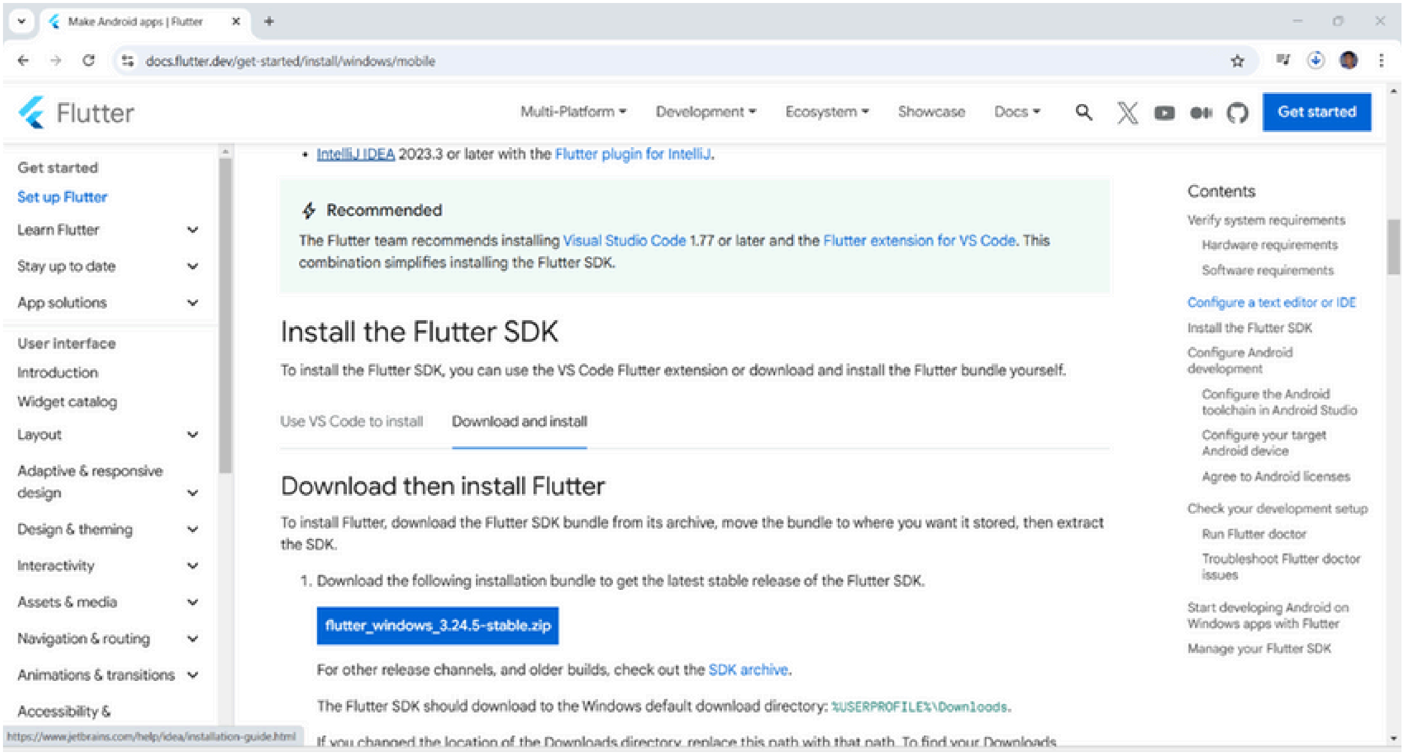
beginning-flutter-a-hands-on-guide-to-app-development_compress (2).pdf

IT-SOCIETY STRUCTURE-1.docx

Yesterday

Identification Symbol Passport Logo (3).png

Deleted flutter_windows_3.24.5-stable.zip from download history Undo


Make Android apps | Flutter
docs.flutter.dev/get-started/install/windows/mobile
Flutter
Multi-Platform
Development
Ecosystem
Showcase
Docs
Get started
Get started
Set up Flutter
Learn Flutter
Stay up to date
App solutions
User interface
Introduction
Widget catalog
Layout
Adaptive & responsive design
Design & theming
Interactivity
Assets & media
Navigation & routing
Animations & transitions
Accessibility &
https://www.jetbrains.com/help/idea/installation-guide.html
Contents
Verify system requirements
Hardware requirements
Software requirements
Configure a text editor or IDE
Install the Flutter SDK
Configure Android development
Configure the Android toolchain in Android Studio
Configure your target Android device
Agree to Android licenses
Check your development setup
Run Flutter doctor
Troubleshoot Flutter doctor issues
Start developing Android on Windows apps with Flutter
Manage your Flutter SDK

- IntelliJ IDEA 2023.3 or later with the Flutter plugin for IntelliJ.

Recommended

The Flutter team recommends installing Visual Studio Code 1.77 or later and the Flutter extension for VS Code. This combination simplifies installing the Flutter SDK.

Install the Flutter SDK

To install the Flutter SDK, you can use the VS Code Flutter extension or download and install the Flutter bundle yourself.

Use VS Code to install   Download and install

Download then install Flutter

To install Flutter, download the Flutter SDK bundle from its archive, move the bundle to where you want it stored, then extract the SDK.

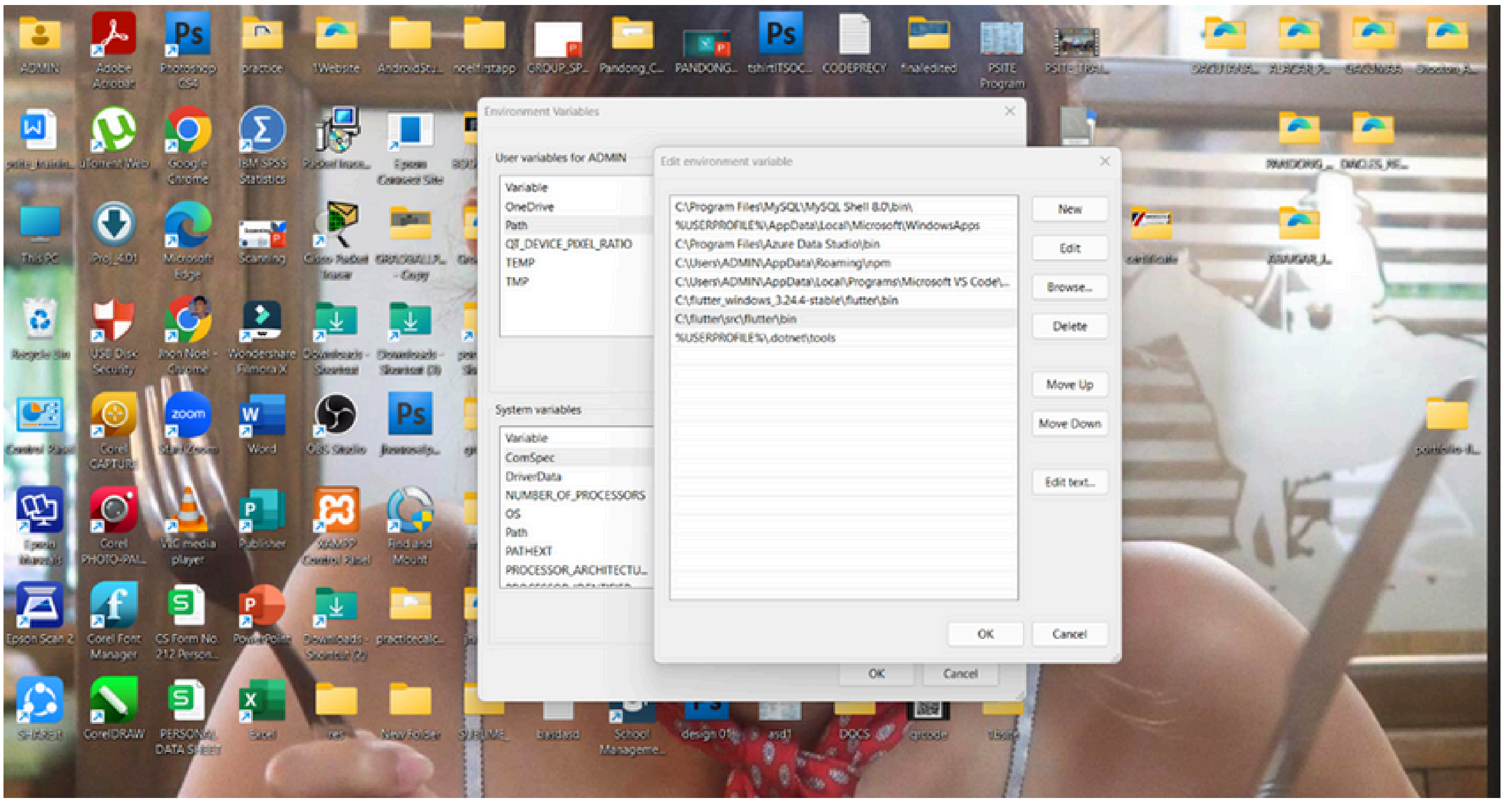
1. Download the following installation bundle to get the latest stable release of the Flutter SDK.

flutter_windows_3.24.5-stable.zip

For other release channels, and older builds, check out the SDK archive.

The Flutter SDK should download to the Windows default download directory: `%USERPROFILE%\Downloads`.

If you changed the location of the Downloads directory, replace this path with that path. To find your Downloads



Environment Variables

User variables for ADMIN

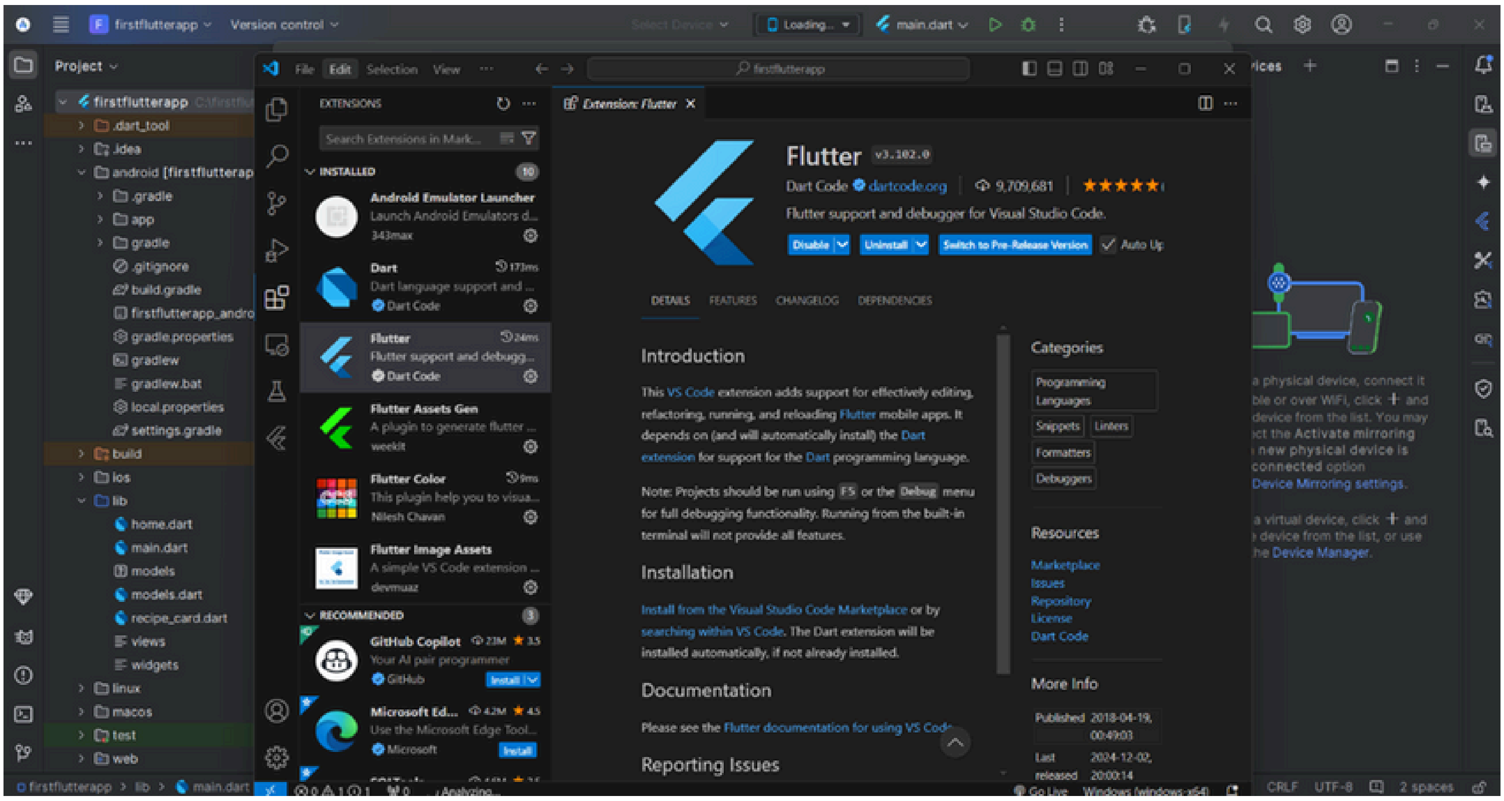
Variable
OneDrive
Path
QT_DEVICE_PIXEL_RATIO
TEMP
TMP

System variables

Variable
ComSpec
DriverData
NUMBER_OF_PROCESSORS
OS
Path
PATHEXT
PROCESSOR_ARCHITECTU...

Edit environment variable

```
C:\Program Files\MySQL\MySQL Shell 8.0\bin\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Program Files\Azure Data Studio\bin
C:\Users\ADMIN\AppData\Roaming\npm
C:\Users\ADMIN\AppData\Local\Programs\Microsoft VS Code\...
C:\flutter_windows_3.24.4-stable\flutter\bin
C:\flutter\src\flutter\bin
%USERPROFILE%\.dotnet\tools
```



Project tree:

```
firstflutterapp
  .dart_tool
  .idea
  android [firstflutterapp]
    .gradle
    app
    gradle
    .gitignore
    build.gradle
    firstflutterapp_andro...
    gradle.properties
    gradlew
    gradlew.bat
    local.properties
    settings.gradle
  build
  ios
  lib
    home.dart
    main.dart
    models
    models.dart
    recipe_card.dart
    views
    widgets
  linux
  macos
  test
  web
```

Installed extensions:

- Android Emulator Launcher — Launch Android Emulators d... — 343max
- Dart — Dart language support and ... — Dart Code
- Flutter — Flutter support and debugg... — Dart Code
- Flutter Assets Gen — A plugin to generate flutter ... — weekit
- Flutter Color — This plugin help you to visua... — Nilesh Chavan
- Flutter Image Assets — A simple VS Code extension ... — devmuaz

Recommended:

- GitHub Copilot — 23M ★ 3.5 — Your AI pair programmer — GitHub — Install
- Microsoft Ed... — 4.2M ★ 4.5 — Use the Microsoft Edge Tool... — Microsoft — Install

Extension: Flutter

Flutter v3.102.0

Dart Code dartcode.org | 9,709,681 | ★★★★★

Flutter support and debugger for Visual Studio Code.

Disable | Uninstall | Switch to Pre-Release Version | ✓ Auto Up

DETAILS FEATURES CHANGELOG DEPENDENCIES

Introduction

This VS Code extension adds support for effectively editing, refactoring, running, and reloading Flutter mobile apps. It depends on (and will automatically install) the Dart extension for support for the Dart programming language.

Note: Projects should be run using F5 or the Debug menu for full debugging functionality. Running from the built-in terminal will not provide all features.

Installation

Install from the Visual Studio Code Marketplace or by searching within VS Code. The Dart extension will be installed automatically, if not already installed.

Documentation

Please see the Flutter documentation for using VS Code

Reporting Issues

Categories: Programming Languages, Snippets, Linters, Formatters, Debuggers

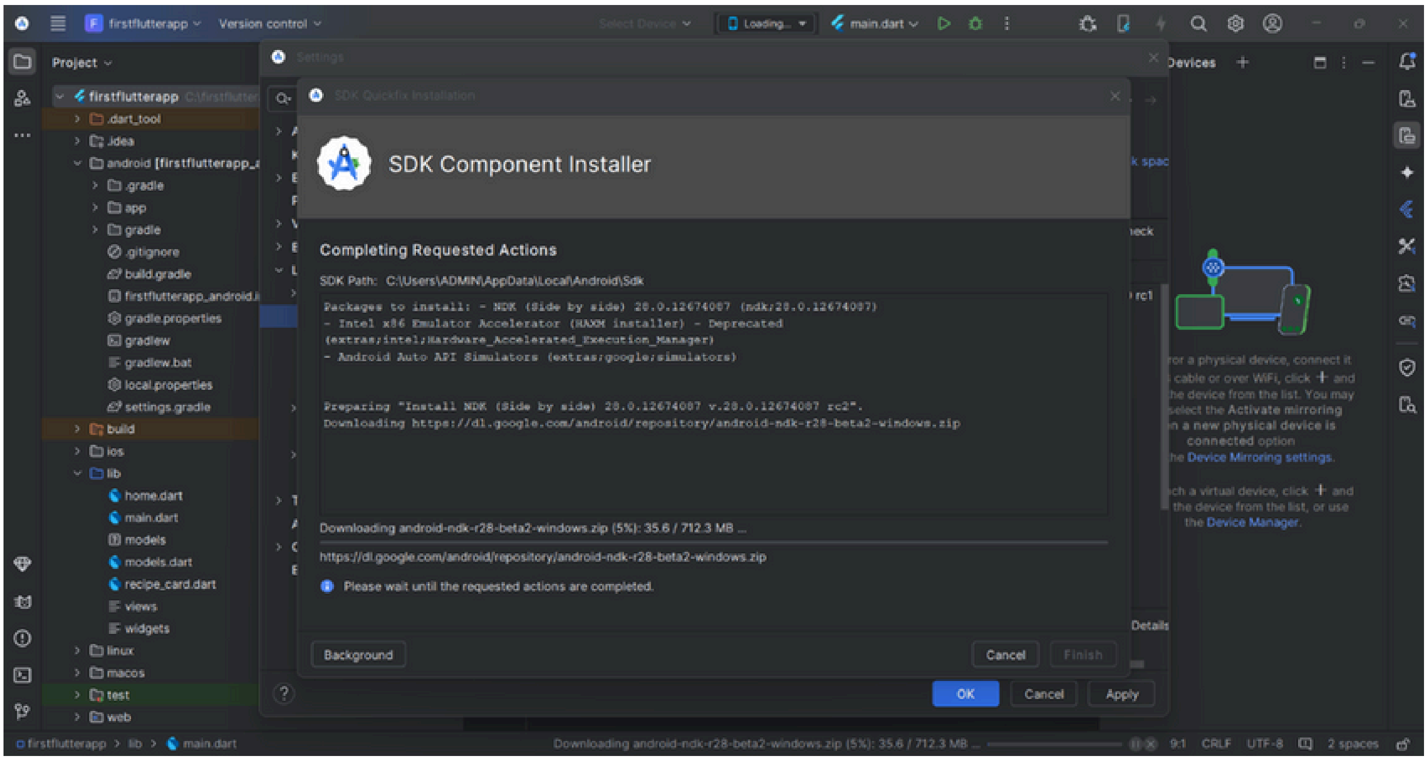
Resources: Marketplace, Issues, Repository, License, Dart Code

More Info

Published	2018-04-19, 00:49:03
Last released	2024-12-02, 20:00:14

a physical device, connect it ble or over WiFi, click + and device from the list. You may ct the Activate mirroring new physical device is connected option Device Mirroring settings.

a virtual device, click + and device from the list, or use he Device Manager.



SDK Component Installer

Completing Requested Actions

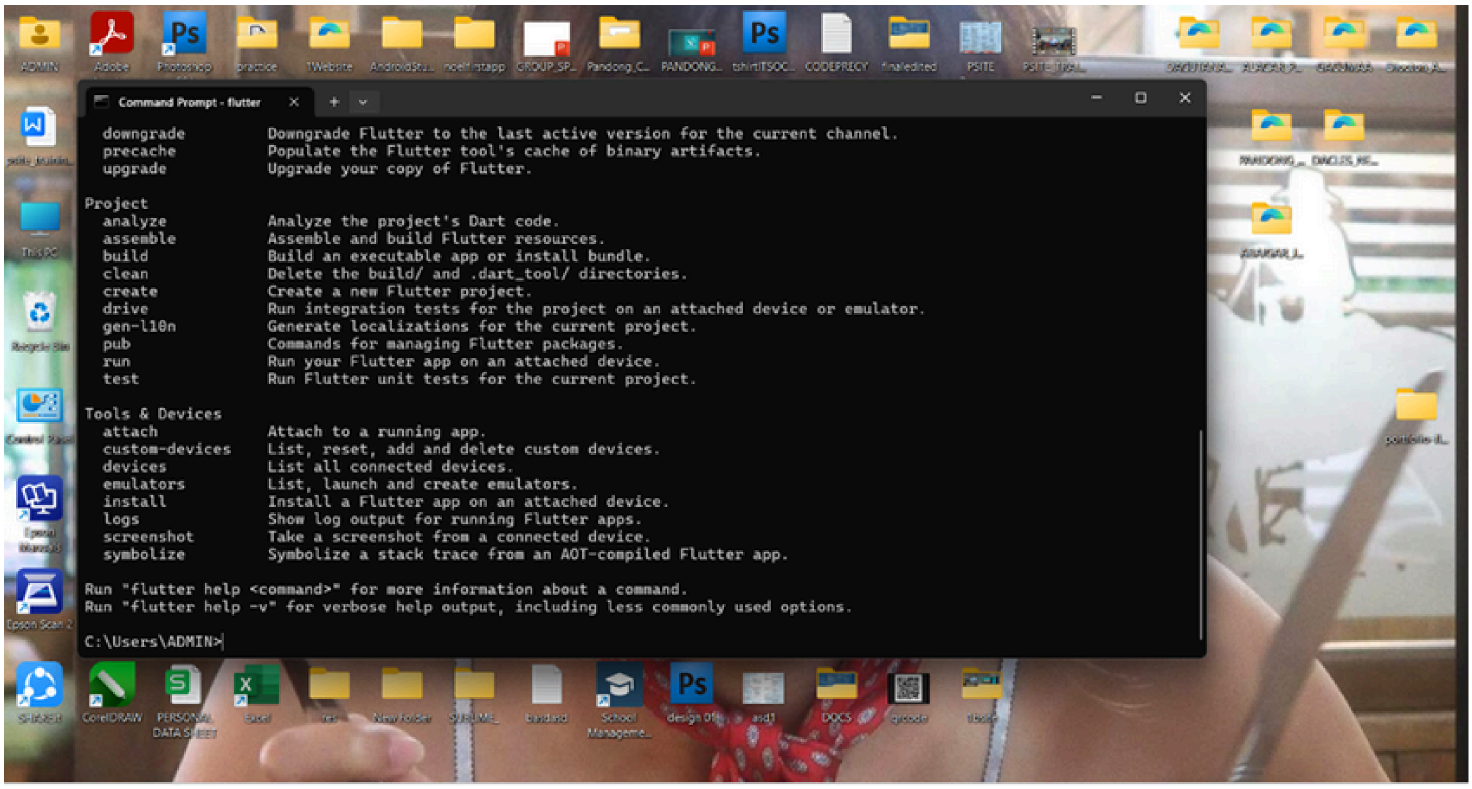
SDK Path: C:\Users\ADMIN\AppData\Local\Android\Sdk

```
Packages to install: - NDK (Side by side) 28.0.12674087 (ndk;28.0.12674087)
- Intel x86 Emulator Accelerator (HAXM installer) - Deprecated
(extras;intel;Hardware_Accelerated_Execution_Manager)
- Android Auto API Simulators (extras;google;simulators)


Preparing "Install NDK (Side by side) 28.0.12674087 v.28.0.12674087 rc2".
Downloading https://dl.google.com/android/repository/android-ndk-r28-beta2-windows.zip
```

Downloading android-ndk-r28-beta2-windows.zip (5%): 35.6 / 712.3 MB ...

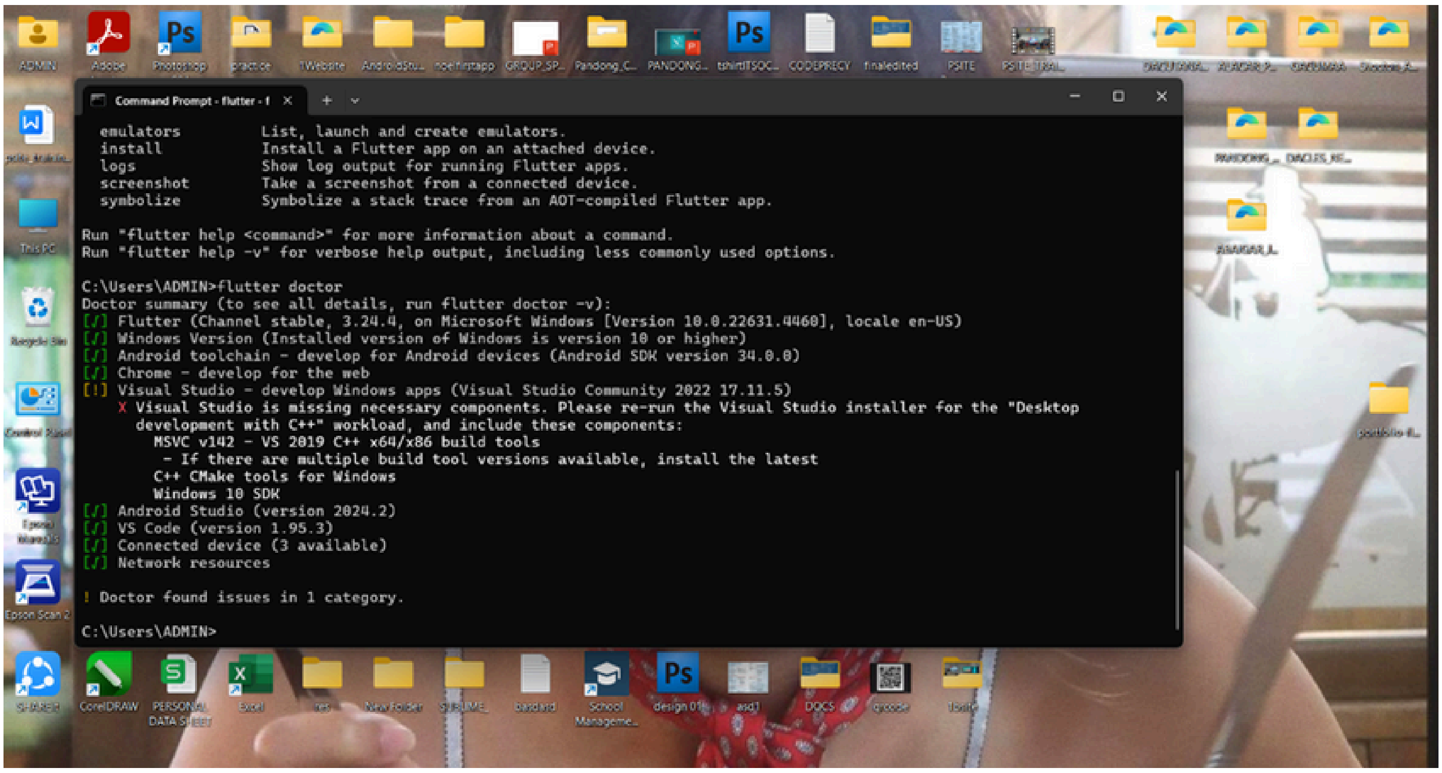
https://dl.google.com/android/repository/android-ndk-r28-beta2-windows.zip

Please wait until the requested actions are completed.



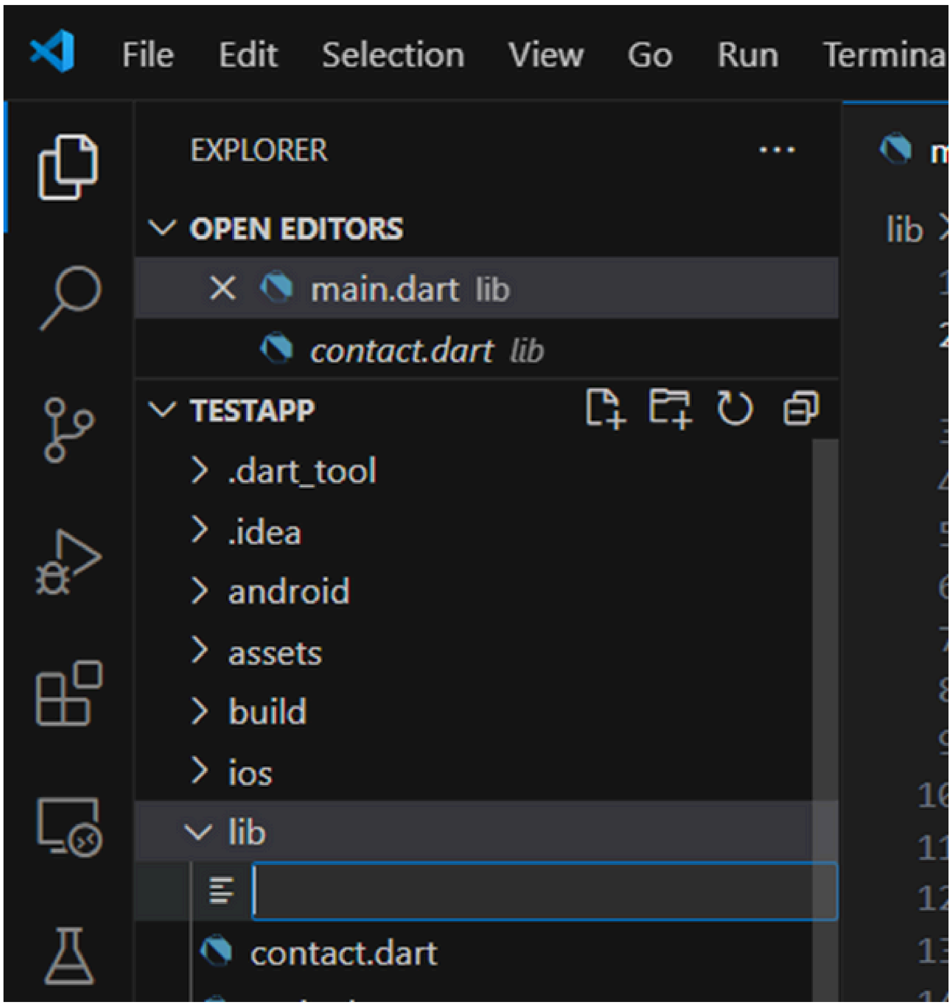
Command Prompt - flutter

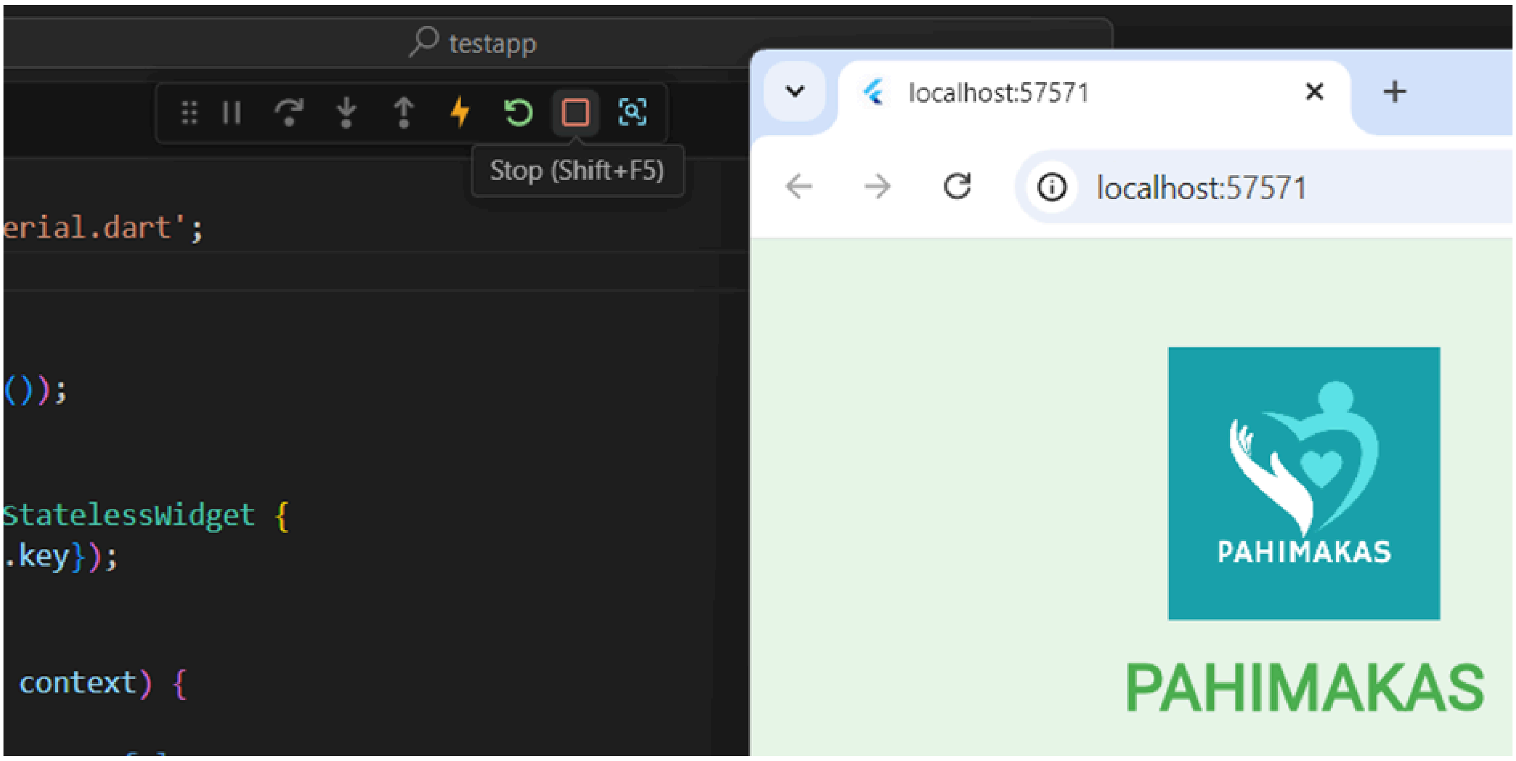
```
  downgrade         Downgrade Flutter to the last active version for the current channel.
  precache          Populate the Flutter tool's cache of binary artifacts.
  upgrade           Upgrade your copy of Flutter.

Project
  analyze           Analyze the project's Dart code.
  assemble          Assemble and build Flutter resources.
  build             Build an executable app or install bundle.
  clean             Delete the build/ and .dart_tool/ directories.
  create            Create a new Flutter project.
  drive             Run integration tests for the project on an attached device or emulator.
  gen-l10n          Generate localizations for the current project.
  pub               Commands for managing Flutter packages.
  run               Run your Flutter app on an attached device.
  test              Run Flutter unit tests for the current project.

Tools & Devices
  attach            Attach to a running app.
  custom-devices    List, reset, add and delete custom devices.
  devices           List all connected devices.
  emulators         List, launch and create emulators.
  install           Install a Flutter app on an attached device.
  logs              Show log output for running Flutter apps.
  screenshot        Take a screenshot from a connected device.
  symbolize         Symbolize a stack trace from an AOT-compiled Flutter app.

Run "flutter help <command>" for more information about a command.
Run "flutter help -v" for verbose help output, including less commonly used options.

C:\Users\ADMIN>
```



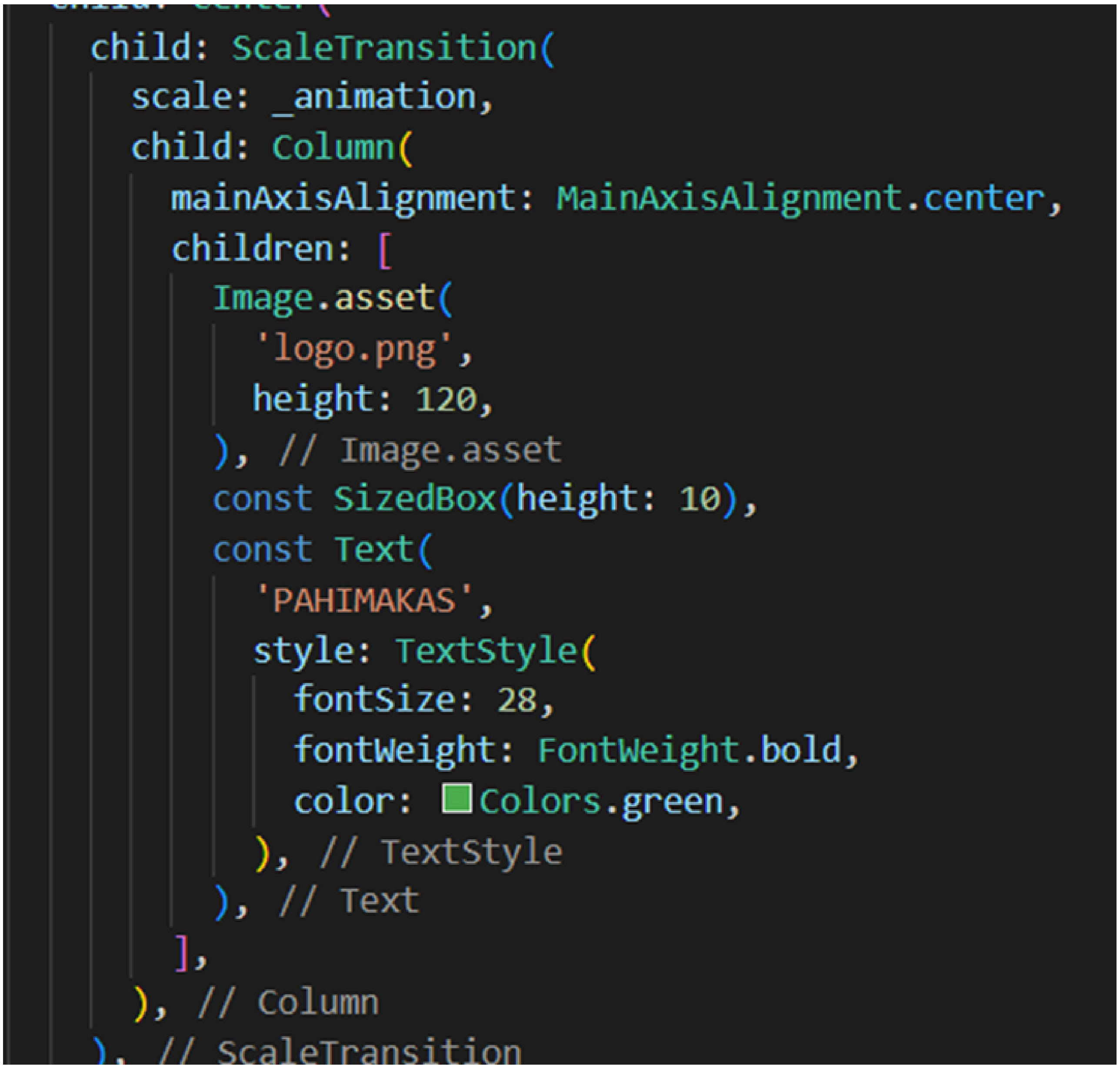
```
  emulators         List, launch and create emulators.
  install           Install a Flutter app on an attached device.
  logs              Show log output for running Flutter apps.
  screenshot        Take a screenshot from a connected device.
  symbolize         Symbolize a stack trace from an AOT-compiled Flutter app.

Run "flutter help <command>" for more information about a command.
Run "flutter help -v" for verbose help output, including less commonly used options.

C:\Users\ADMIN>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.24.4, on Microsoft Windows [Version 10.0.22631.4460], locale en-US)
[√] Windows Version (Installed version of Windows is version 10 or higher)
[√] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[√] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Community 2022 17.11.5)
    X Visual Studio is missing necessary components. Please re-run the Visual Studio installer for the "Desktop
      development with C++" workload, and include these components:
        MSVC v142 - VS 2019 C++ x64/x86 build tools
         - If there are multiple build tool versions available, install the latest
        C++ CMake tools for Windows
        Windows 10 SDK
[√] Android Studio (version 2024.2)
[√] VS Code (version 1.95.3)
[√] Connected device (3 available)
[√] Network resources

! Doctor found issues in 1 category.

C:\Users\ADMIN>
```

Chapter 2: Testing VS Code Using Browser

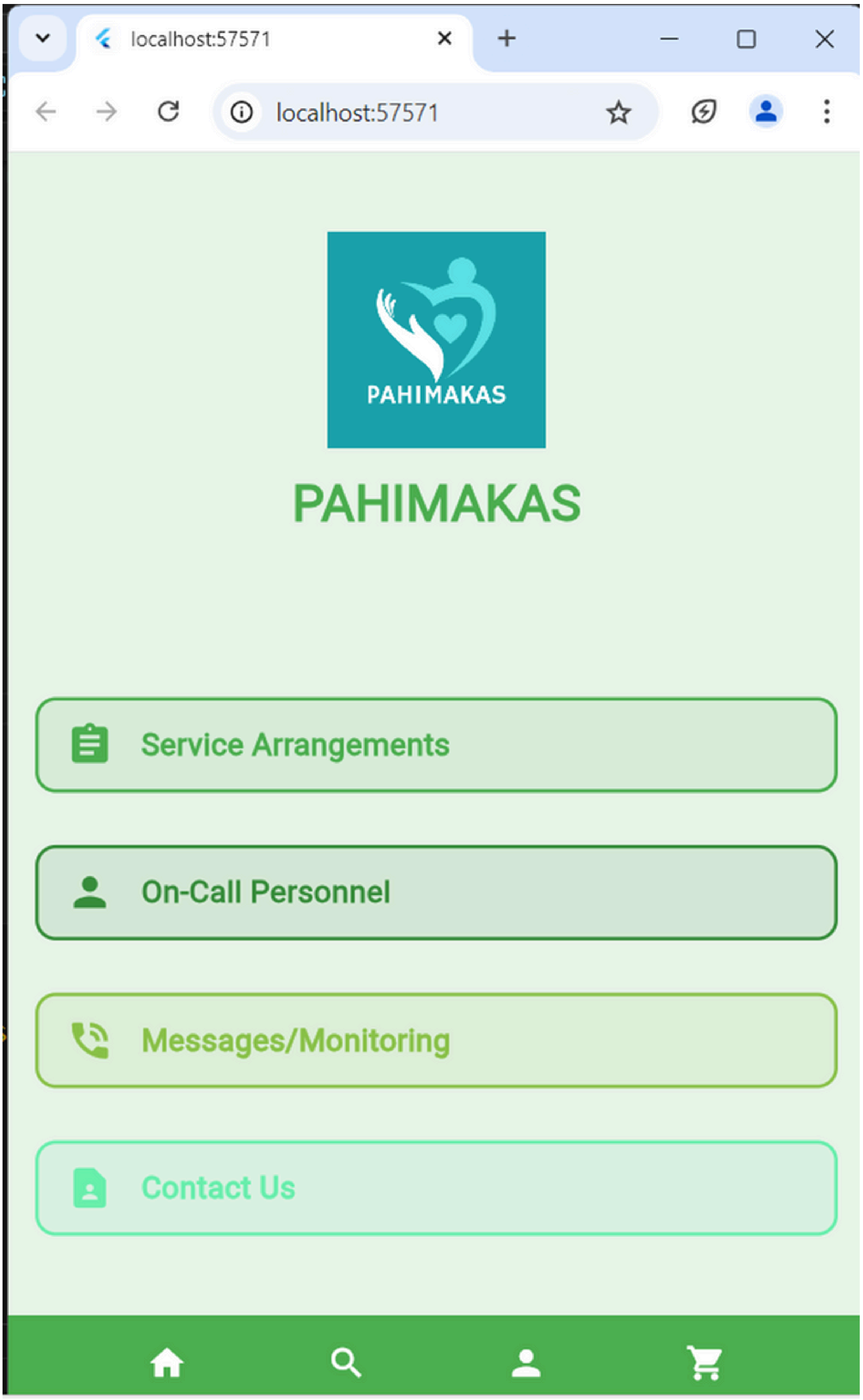




```
erial.dart';

());

StatelessWidget {
.key});

 context) {
```



```
  child: ScaleTransition(
    scale: _animation,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Image.asset(
          'logo.png',
          height: 120,
        ), // Image.asset
        const SizedBox(height: 10),
        const Text(
          'PAHIMAKAS',
          style: TextStyle(
            fontSize: 28,
            fontWeight: FontWeight.bold,
            color: 🟩Colors.green,
          ), // TextStyle
        ), // Text
      ],
    ), // Column
  ), // ScaleTransition
```


localhost:57571
localhost:57571
PAHIMAKAS
PAHIMAKAS
Service Arrangements
On-Call Personnel
Messages/Monitoring
Contact Us

I started by setting up the mobile application project in Flutter using Dart, ensuring the development environment was fully configured and organized for efficient navigation. Leveraging Flutter's hot reload feature has been invaluable, enabling me to instantly view changes during development without restarting the app. To maintain a cohesive design, I implemented a global app theme that defines primary and secondary colors, text styles, and button designs, ensuring consistency across the app. Additionally, I customized themes for specific screens to enhance their functionality and visual appeal, such as creating a unique design for the homepage while keeping settings pages more neutral.

As part of the app structure, I focused on mastering stateless and stateful widgets, using stateless widgets for static content and stateful widgets for dynamic elements like user interactions. This foundational understanding has allowed me to manage app states efficiently. To extend functionality, I've integrated external packages, such as those for state management, image loading, and local storage. This has significantly improved performance and feature richness. I'm continuously exploring new packages from Flutter's repository to enhance features like notifications, API integrations, and UI components, ensuring the app remains modern and user-friendly.

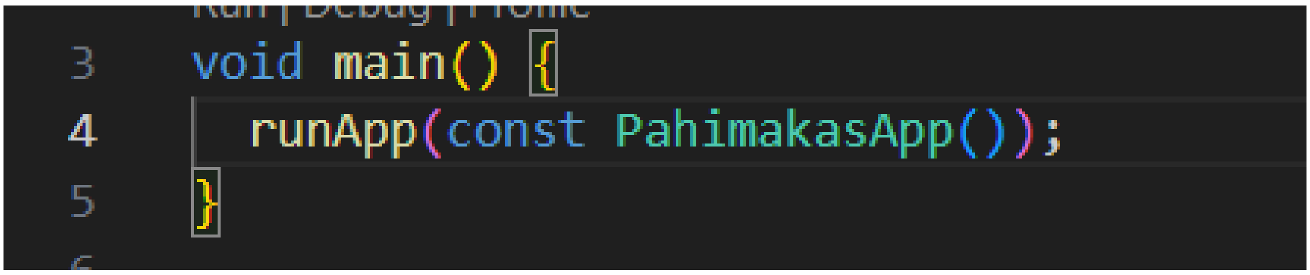
After setting up the development environment, I tested it by running some basic Flutter code in the browser. This step was crucial to ensure that everything was functioning correctly, and it allowed me to confirm that I was ready to begin building the app without encountering any major issues

Chapter 3: Learning the Basics of Dart with My Codes

1. main() Function (Entry Point)

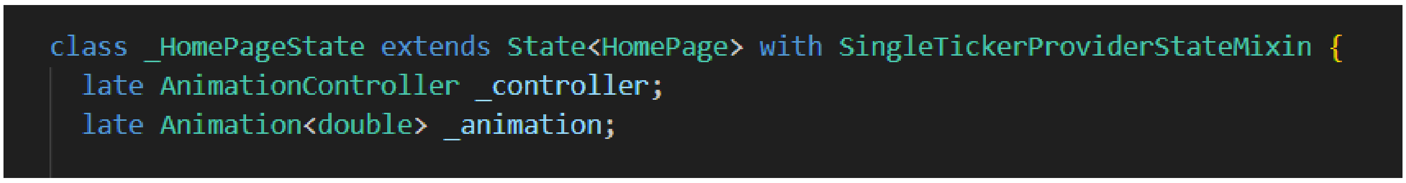
The main() function is the entry point for Dart programs, including Flutter apps.

```
void main() {
  runApp(const PahimakasApp());
}
```



```
void main() {
  runApp(const PahimakasApp());
}
```

2. Variables



```
class _HomePageState extends State<HomePage> with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _animation;
```

3. Classes and Object-Oriented Programming (OOP)

Dart's object-oriented nature is evident through the use of classes like PahimakasApp, HomePage, and _HomePageState.
The HomePage class demonstrates state management with StatefulWidget and its associated _HomePageState.

4. Functions and Methods

Functions modularize your code. Examples include:

- build() to define the widget tree.
- _buildAnimatedButton() to generate reusable UI elements with animations.

```
Widget _buildAnimatedButton(BuildContext context, IconData icon, String label, Color color) {
  // Button logic here
}
```

5. Flow Control Statements
Basic flow controls like the initializer (initState) and cleanup (dispose) methods manage the animation lifecycle.

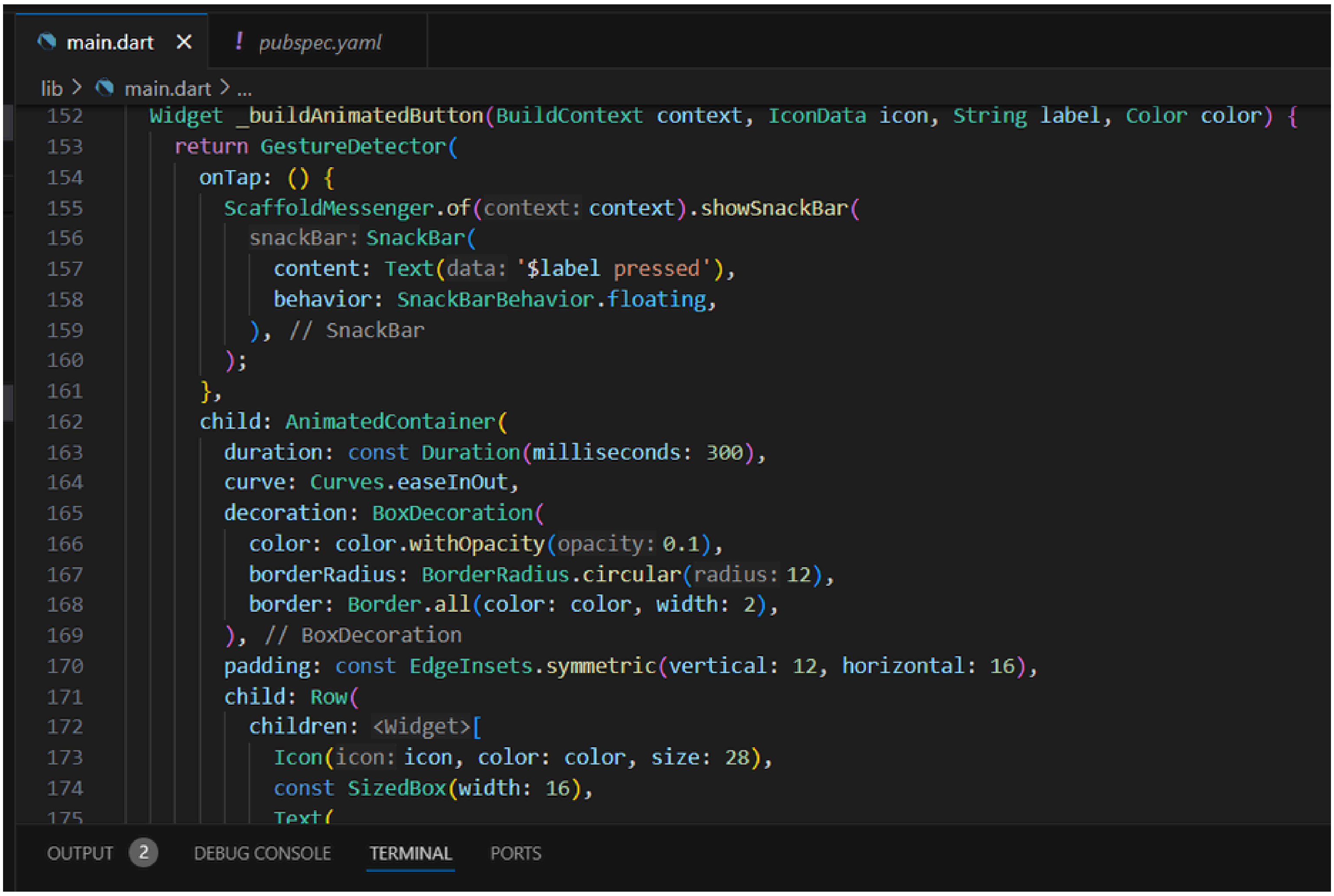
```
void initState() {
  super.initState();
  _controller = AnimationController(
    duration: const Duration(seconds: 2),
    vsync: this,
  )..repeat(reverse: true);
}

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

6. Asynchronous Programming
Although not explicitly used here, animations utilize asynchronous principles, ensuring smooth transitions without blocking the UI.

7. Collections (Lists)
While not directly in use, Dart basics like lists could be used to store buttons or other UI elements dynamically.

8. Widgets and State Management (Flutter Specific)
StatelessWidget and *StatefulWidget* are foundational Flutter widgets that align with Dart’s class-based structure.

Chapter 4-8



```dart
Widget _buildAnimatedButton(BuildContext context, IconData icon, String label, Color color) {
  return GestureDetector(
    onTap: () {
      ScaffoldMessenger.of(context: context).showSnackBar(
        snackBar: SnackBar(
          content: Text(data: '$label pressed'),
          behavior: SnackBarBehavior.floating,
        ), // SnackBar
      );
    },
    child: AnimatedContainer(
      duration: const Duration(milliseconds: 300),
      curve: Curves.easeInOut,
      decoration: BoxDecoration(
        color: color.withOpacity(opacity: 0.1),
        borderRadius: BorderRadius.circular(radius: 12),
        border: Border.all(color: color, width: 2),
      ), // BoxDecoration
      padding: const EdgeInsets.symmetric(vertical: 12, horizontal: 16),
      child: Row(
        children: <Widget>[
          Icon(icon: icon, color: color, size: 28),
          const SizedBox(width: 16),
          Text(
```

This chapter discusses the development and functionality of my mobile application, "Pahimakas," designed using Flutter and Dart. The application simplifies funeral service management, ensuring an intuitive experience for users.

In Flutter, the widget tree is a structure that organizes all UI elements and their properties. Since everything in Flutter is a widget, the widget tree becomes crucial for managing how the app's UI is built and displayed. In this chapter, I will show how the widget tree is used in my mobile application, "Pahimakas," designed to simplify funeral service management.

The app's UI is built by nesting various widgets, such as containers, rows, columns, and buttons. This approach makes the layout flexible and responsive. However, as we nest more widgets, the code can become difficult to maintain, and this is where refactoring becomes essential.

Understanding the Full Widget Tree

The full widget tree for the "Pahimakas" app can become quite deep, which may affect code readability and performance. Here's an example of how the widget tree might look when structured deeply.

```
MaterialApp
 ├── Scaffold
 │   ├── AppBar
 │   ├── Body
 │   │   ├── Column
 │   │   │   ├── TextWidget
 │   │   │   ├── ImageWidget
 │   │   │   ├── AnimatedButton (Custom Widget)
 │   │   │   ├── CustomCardWidget (Custom Widget)
 │   │   │   └── FooterWidget (Custom Widget)
 │   └── BottomNavigationBar
```

Refactoring the Widget Tree

To keep the widget tree shallow and more manageable, we refactor the code by breaking down complex widget structures into smaller, reusable widgets. This not only makes the code cleaner but also improves performance by reducing unnecessary nesting.

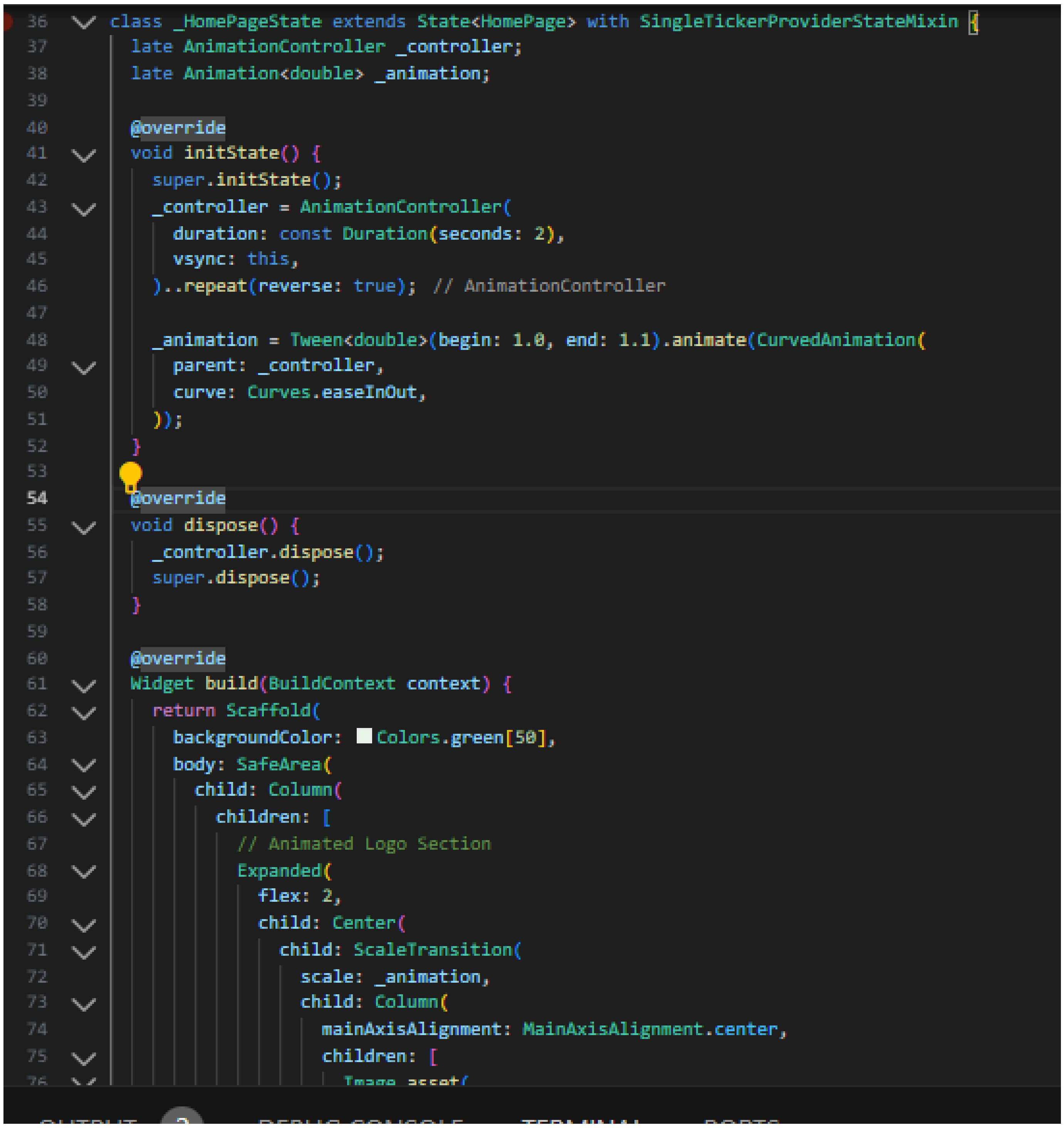
Refactored Widget Tree

```
MaterialApp
 ├── HomePage (StatelessWidget)
 │   ├── AppBar
 │   ├── Body
 │   │   ├── HeaderWidget (Custom Widget)
 │   │   ├── ContentWidget (Custom Widget)
 │   │   ├── FooterWidget (Custom Widget)
 │   └── BottomNavigationBarWidget (Custom Widget)
```

Example: Animated Button Widget

One of the key features in the app is the use of animated buttons. Instead of defining the button in every place it's used, we can create a custom widget that encapsulates its behavior. Here's an example of how the animated button widget is defined:

Flutter Animation

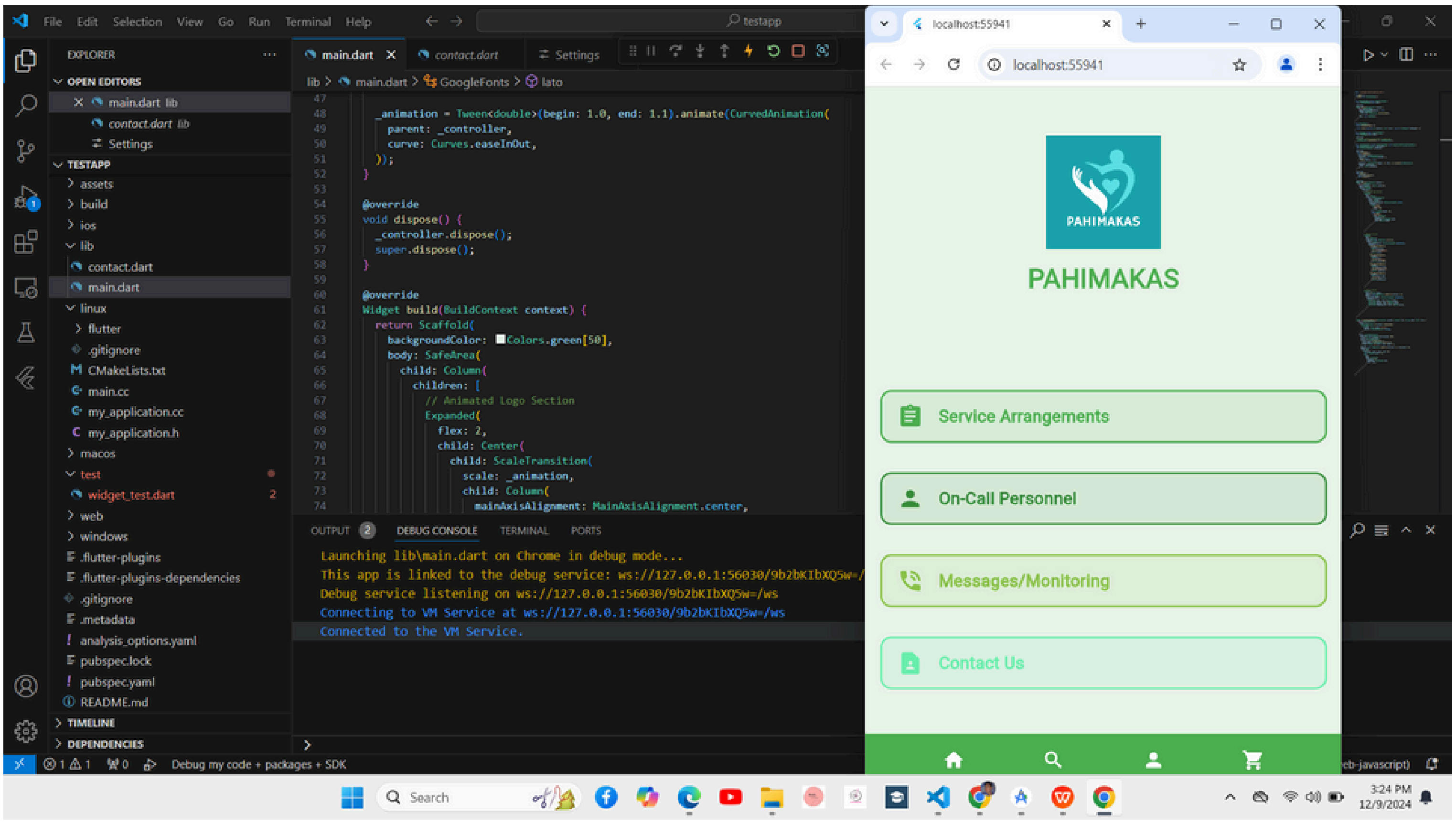


```dart
class _HomePageState extends State<HomePage> with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _animation;

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(
      duration: const Duration(seconds: 2),
      vsync: this,
    )..repeat(reverse: true); // AnimationController

    _animation = Tween<double>(begin: 1.0, end: 1.1).animate(CurvedAnimation(
      parent: _controller,
      curve: Curves.easeInOut,
    ));
  }

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.green[50],
      body: SafeArea(
        child: Column(
          children: [
            // Animated Logo Section
            Expanded(
              flex: 2,
              child: Center(
                child: ScaleTransition(
                  scale: _animation,
                  child: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      Image.asset(
```

Adding Animation to an App

This chapter delves into how animations can enhance the user experience in the "Pahimakas" app. Using Flutter's animation tools, the app includes smooth transitions and dynamic visual effects to engage users.

Creating an App's Navigation



```dart

    _animation = Tween<double>(begin: 1.0, end: 1.1).animate(CurvedAnimation(
      parent: _controller,
      curve: Curves.easeInOut,
    ));
  }

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.green[50],
      body: SafeArea(
        child: Column(
          children: [
            // Animated Logo Section
            Expanded(
              flex: 2,
              child: Center(
                child: ScaleTransition(
                  scale: _animation,
                  child: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
```

```
Launching lib\main.dart on Chrome in debug mode...
This app is linked to the debug service: ws://127.0.0.1:56030/9b2bKIbXQ5w=/
Debug service listening on ws://127.0.0.1:56030/9b2bKIbXQ5w=/ws
Connecting to VM Service at ws://127.0.0.1:56030/9b2bKIbXQ5w=/ws
Connected to the VM Service.
```

Navigation is a crucial component of the "Pahimakas" app, ensuring users can move seamlessly between pages. This chapter explores how navigation has been implemented to enhance user experience, connecting different sections of the app.