Application Development

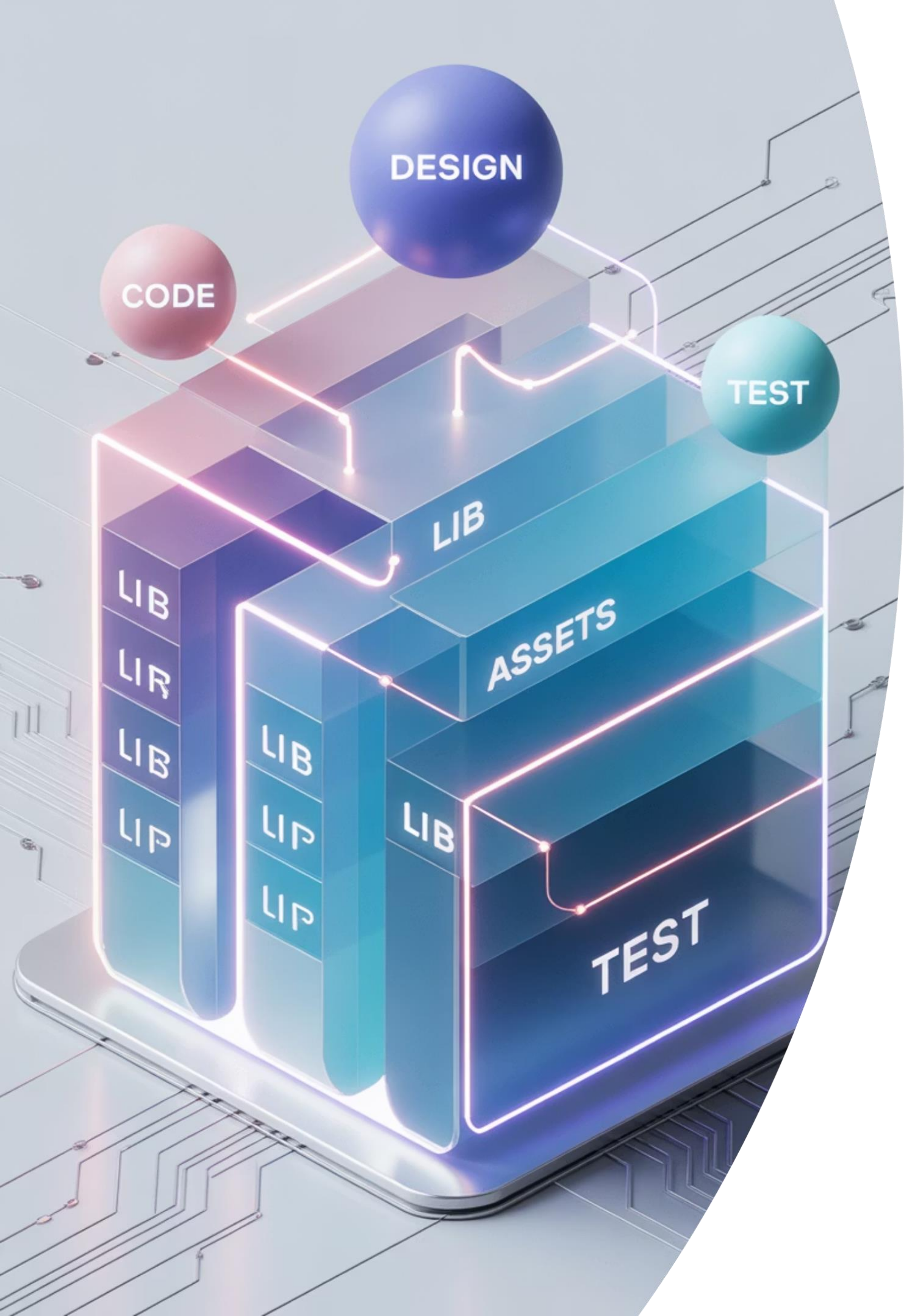
Lecture 2
Project Structure and Application Lifecycle

Asst. Lect. Ali Al-khawaja

Class Room

Welcome & Connection to Week 1

- Welcome back to our second session.
- Recap: Last week we explored Flutter fundamentals, environment setup, and cross-platform concepts.
- Focus Today: Learn about Flutter's project folder architecture, the main.dart entry point, and the application lifecycle including Hot Reload and Hot Restart.



General Lecture Goal

Provide students with an in-depth understanding of how a Flutter project is organized

Learn how the app starts and runs, and how Hot Reload and Hot Restart streamline the developer workflow.

Behavioral Objectives

By the end of this lecture you will be able to:

01

Describe every folder and file generated by flutter create.

02

Explain the role of main.dart as the starting point of the app.

03

Illustrate the Flutter application lifecycle from initialization to termination.

04

Differentiate Hot Reload from Hot Restart and know when to use each.

05

Apply best practices for organizing large Flutter projects.

Lecture contents

- Overview of Flutter project creation
- Folder architecture explained
- Deep dive into main.dart
- Application lifecycle and key methods
- Hot Reload vs. Hot Restart
- Best practices and common pitfalls
- Interactive activities & wrap-up

Creating a Project

Command: flutter create my_app

Generates a complete starter application with a standard directory tree and a default counter app.

High-Level Folder Overview

android/

native Android code & Gradle build files

ios/

native iOS project & Xcode settings

lib/

all Dart source code, main application logic

test/

unit & widget tests

web/

present if web support enabled

pubspec.yaml

app metadata, dependencies, and assets

Hidden: .dart_tool/, .idea/ for IDE configuration

The lib/ Directory

Heart of the App: contains all Dart code.

Starts with main.dart but should be organized into subfolders (e.g., models/, views/, controllers/, services/, widgets/) for scalability.

Key Files in Detail

pubspec.yaml:

Declares app name, version, dependencies, assets, fonts.

pubspec.lock:

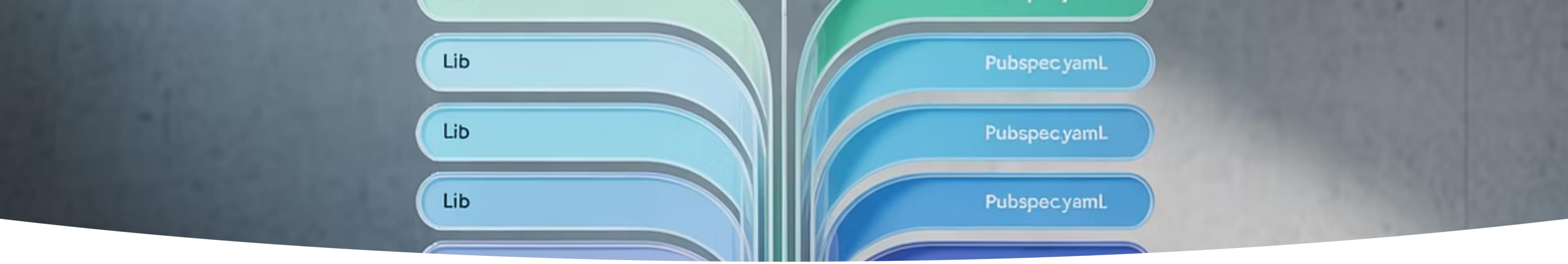
Locks specific dependency versions.

analysis_options.yaml:

Optional linter rules for consistent code style.

README.md:

Project documentation for collaborators.



Visual Directory Tree

```
my_app/  android/  ios/  lib/    main.dart  test/  pubspec.yaml
```

Explain how this structure simplifies collaboration and continuous integration.

Activity 1 – Brainstorming

> *"Why is a standardised folder structure essential for teamwork and scalability in large apps?"*

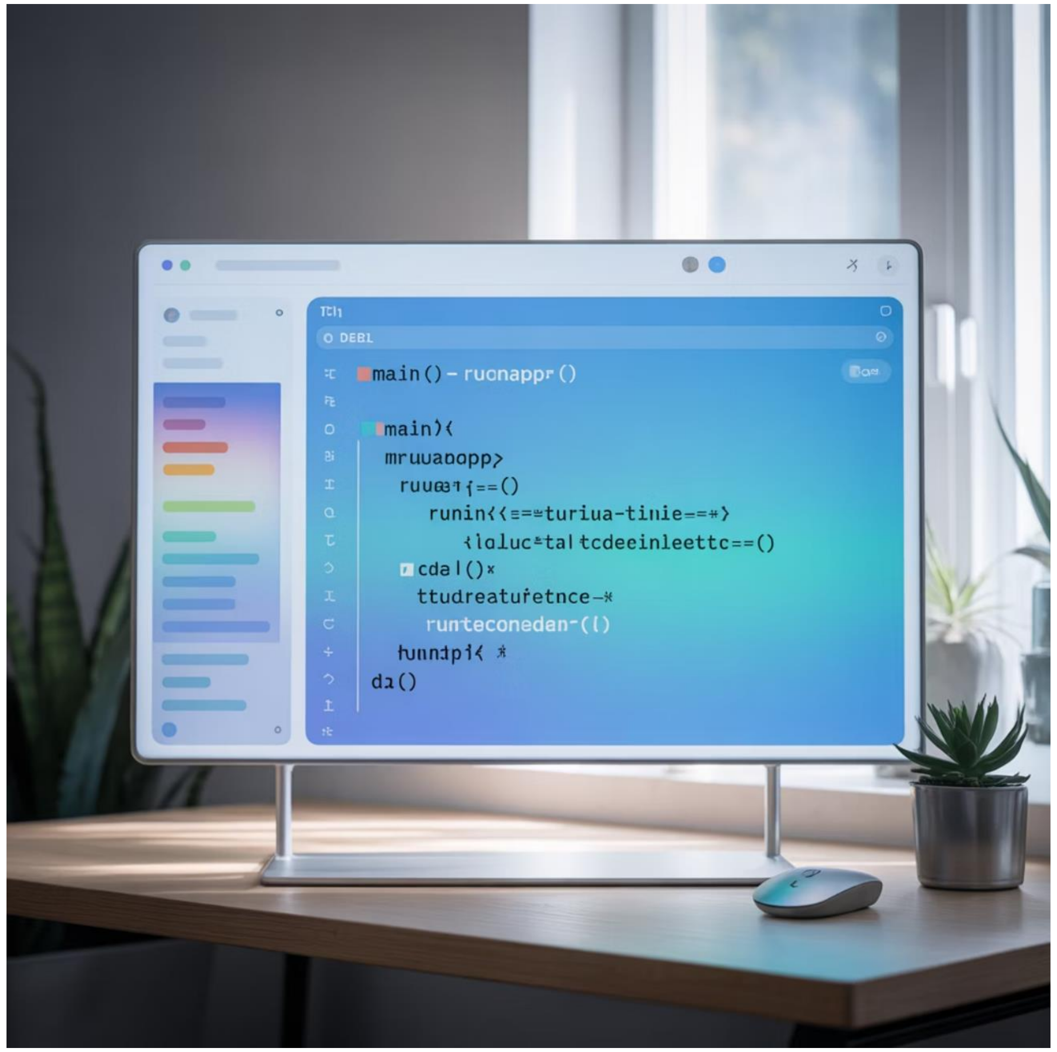
Purpose of main.dart

Contains the main() function, the starting point of every Flutter application.

Example:

```
void main()
{  runApp(MyApp());
}
```

runApp() inflates the widget tree and attaches it to the screen.



Root Widget Structure

1

MyApp usually extends StatelessWidget or StatefulWidget.

2

Returns a MaterialApp or CupertinoApp.

3

Defines theme, routes, and home screen.

Activity 2 – Code Exploration

Display a sample main.dart file.

Students identify:

01

The main() function

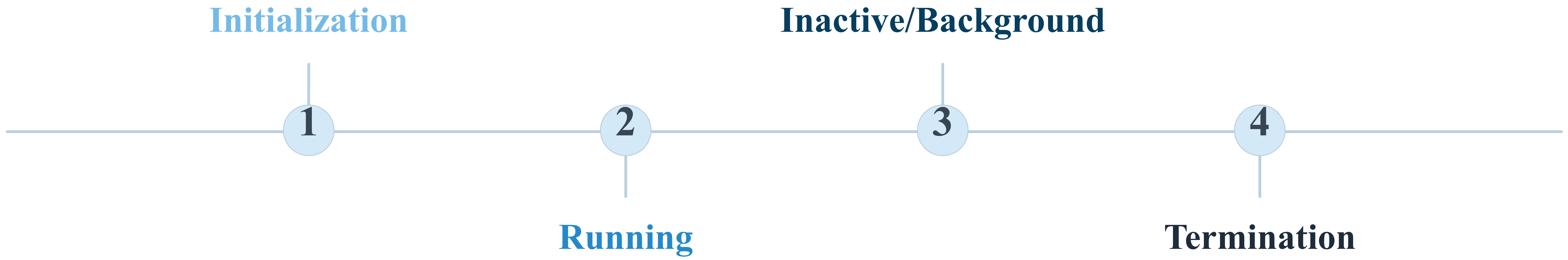
02

The root widget

03

Where the first screen is set.

Application Lifecycle Overview



Crucial for managing memory, state, and background services.

Key Lifecycle Methods (StatefulWidget)

initState()

called once when inserted into widget tree.

didChangeDependencies()

when dependencies change.

build()

renders UI; may run multiple times.

dispose()

clean up resources (streams, controllers) before removal.

Activity 3 – Lifecycle Demo

Instructor shows code printing messages inside lifecycle methods.

Students predict the call order, then watch the demo run to confirm.

Hot Reload

- Injects updated source code into the running Dart VM.
- Preserves current app state.
- Ideal for UI changes, adding methods, or adjusting layouts.

Hot Restart

- Restarts the app from main().
- Resets all state and rebuilds the widget tree.
- Needed for changes to global variables or app initialisation code.

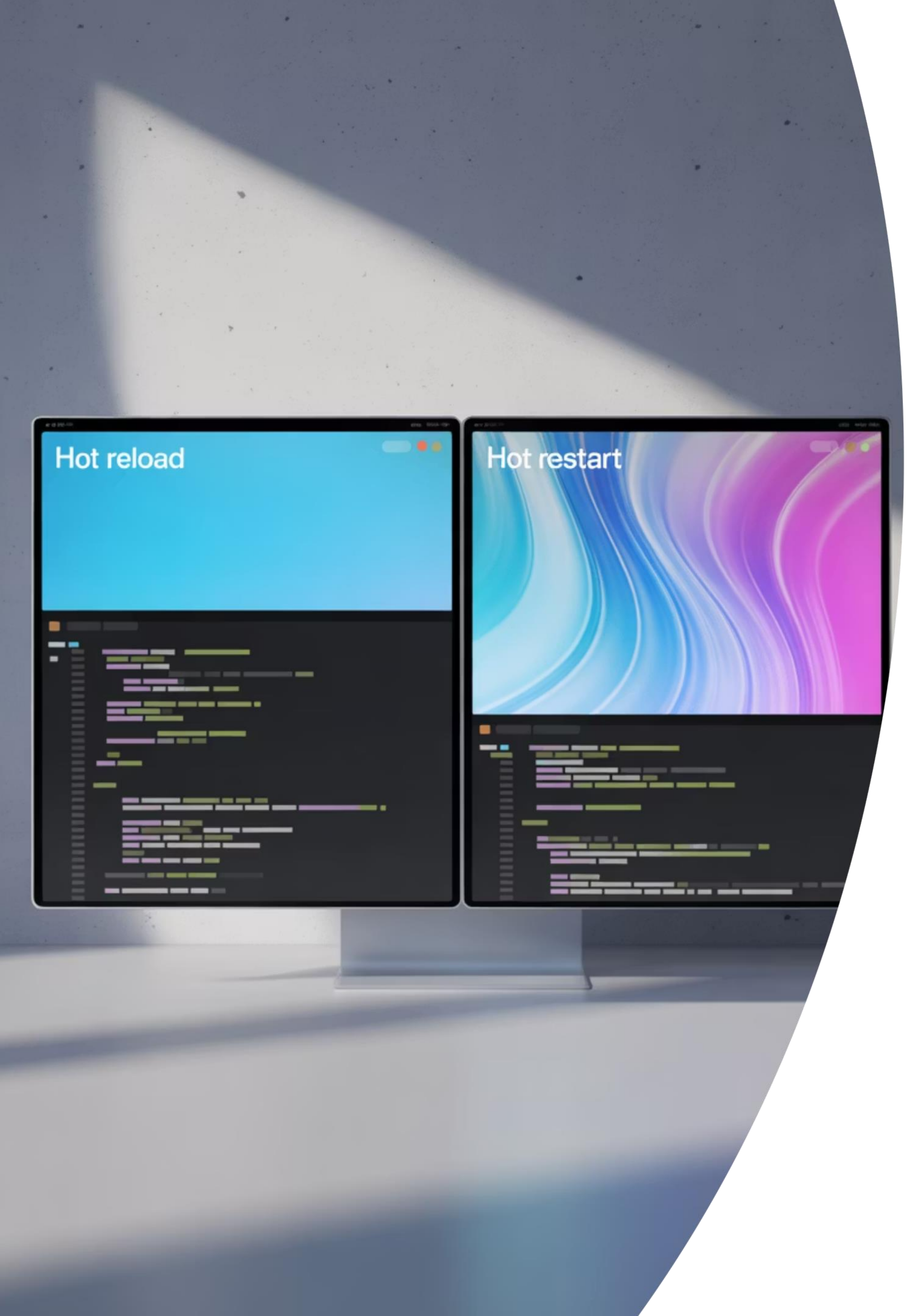
Performance & Use Cases

Hot Reload

faster iteration for UI tweaks.

Hot Restart

necessary for structural or state-critical changes.



Activity 4 – Live Comparison

1. Instructor changes a colour constant → Hot Reload → instant UI change.
2. Adds a new global variable → requires Hot Restart → observe state reset.

Recommended Folder Strategy

Feature-based structure for scalability:

```
lib/  main.dart  src/    models/    views/    controllers/    services/    widgets/
```

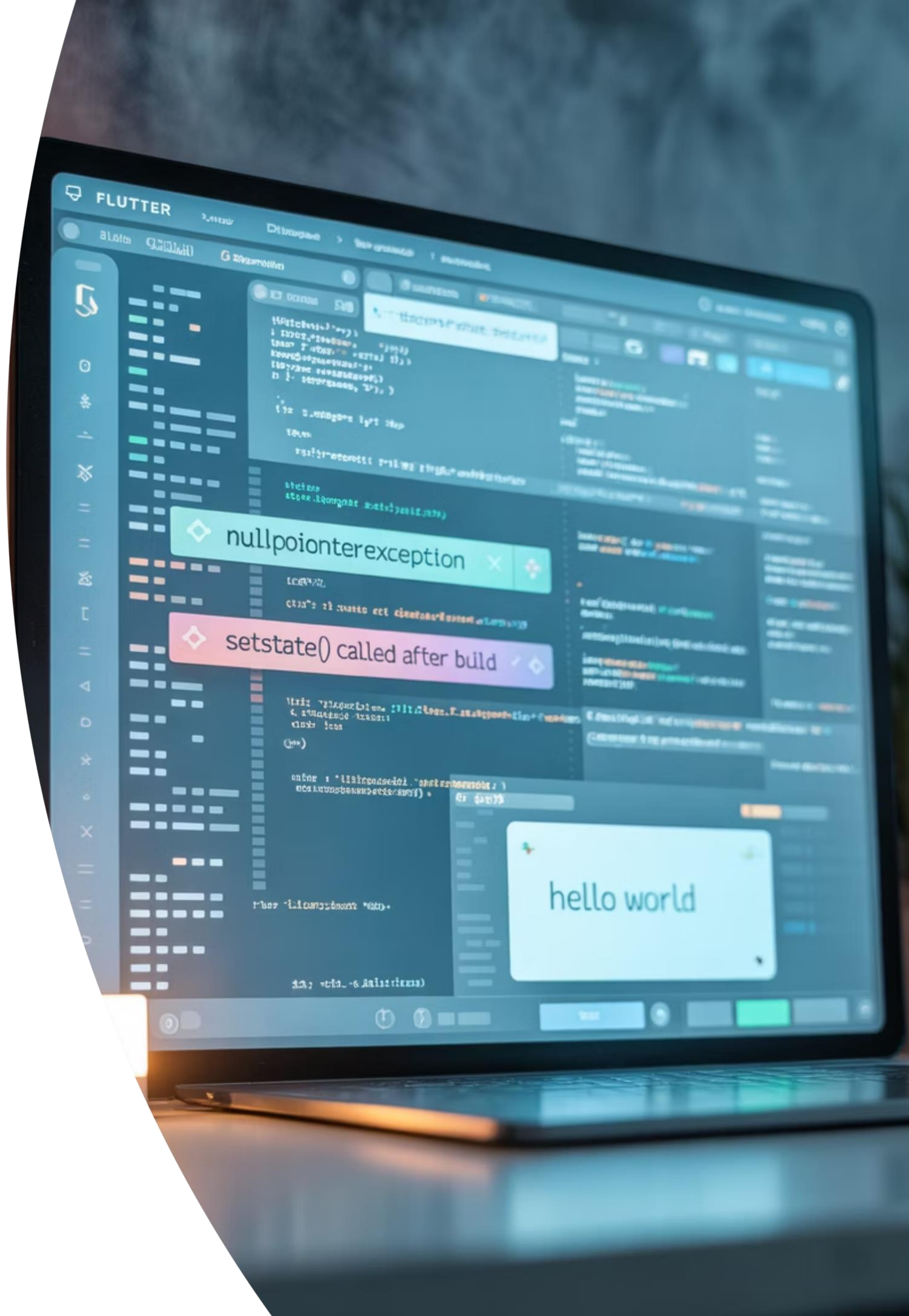
Improves clarity and collaboration in large projects.

Common Pitfalls

Placing all logic inside main.dart.

Forgetting to declare assets in pubspec.yaml.

Neglecting the test/ directory for automated tests.



Activity 5 – Group Discussion

Groups of 4–5 students: *Design a folder layout for a Flutter e-commerce app (home, cart, profile). Explain your reasoning.*

10 minutes group work + 5 minutes presentations.

Key Resources

Flutter official docs:

https://docs.flutter.dev

Dart language:

https://dart.dev

Textbook:

Flutter Complete Reference (Alberto Miola, 2023).

Key Takeaways

Flutter provides a predictable project structure for maintainability.

main.dart is the entry point and starting node of the widget tree.

Mastering lifecycle methods ensures efficient resource management.

Hot Reload and Hot Restart are crucial for rapid development.

Assignment

Create a new Flutter project.

Write a 200-word reflection explaining:

01

Each folder's purpose.

02

The role of main.dart.

03

Your observations using Hot Reload vs Hot Restart.

Submit via Google Classroom within 48 hours.

Thank you…

Any questions??

My google site

يرجى مسح رمز الاستجابة السريعة QR Code لتعبئة نموذج التغذية الراجعة حول المحاضرة. ملاحظاتكم مهمة لتحسين المحاضرات القادمة.