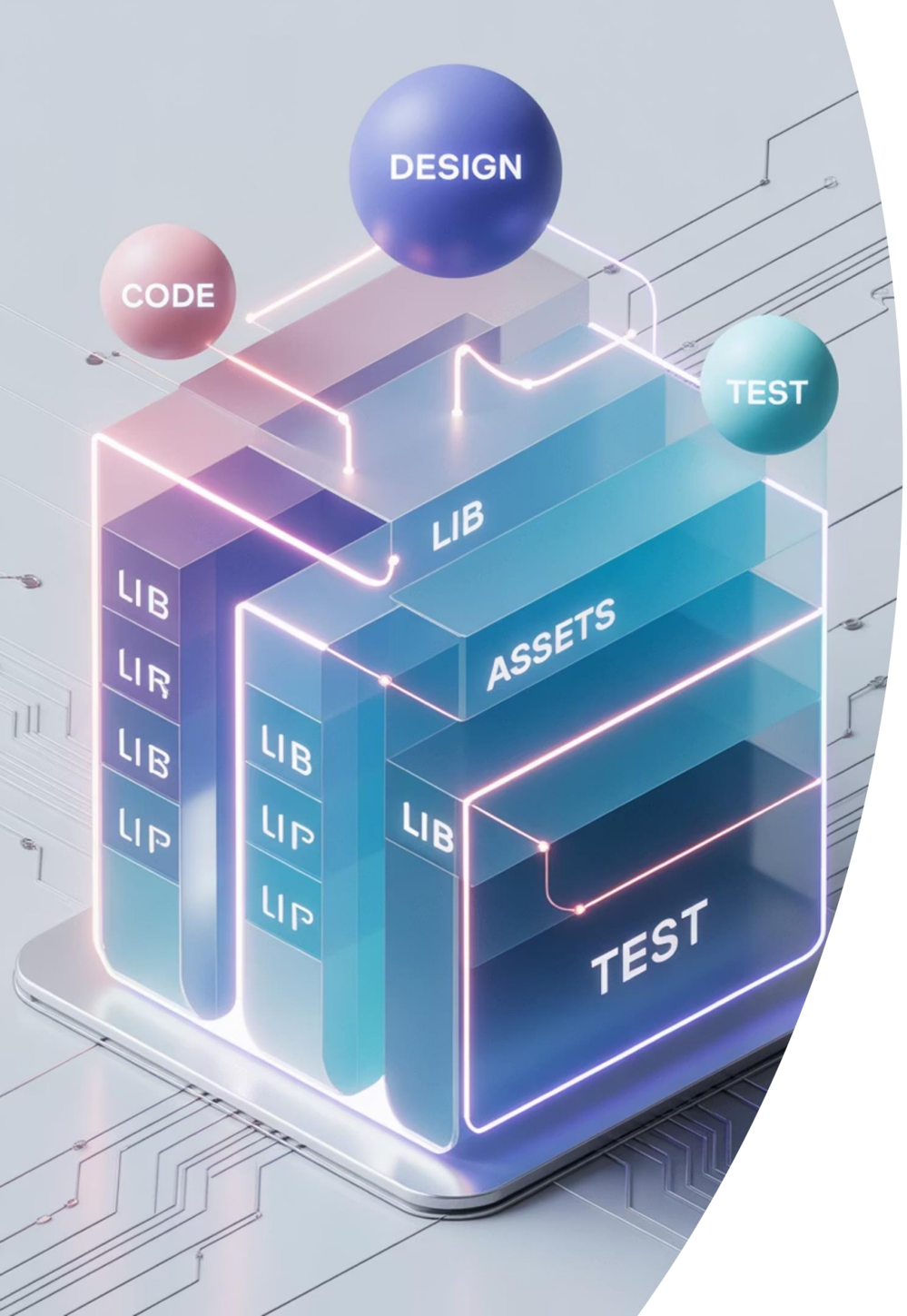
Application Development

Lecture 2
Project Structure and Application Lifecycle

Asst. Lect. Ali Al-khawaja

Class Room



General Lecture Goal

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

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

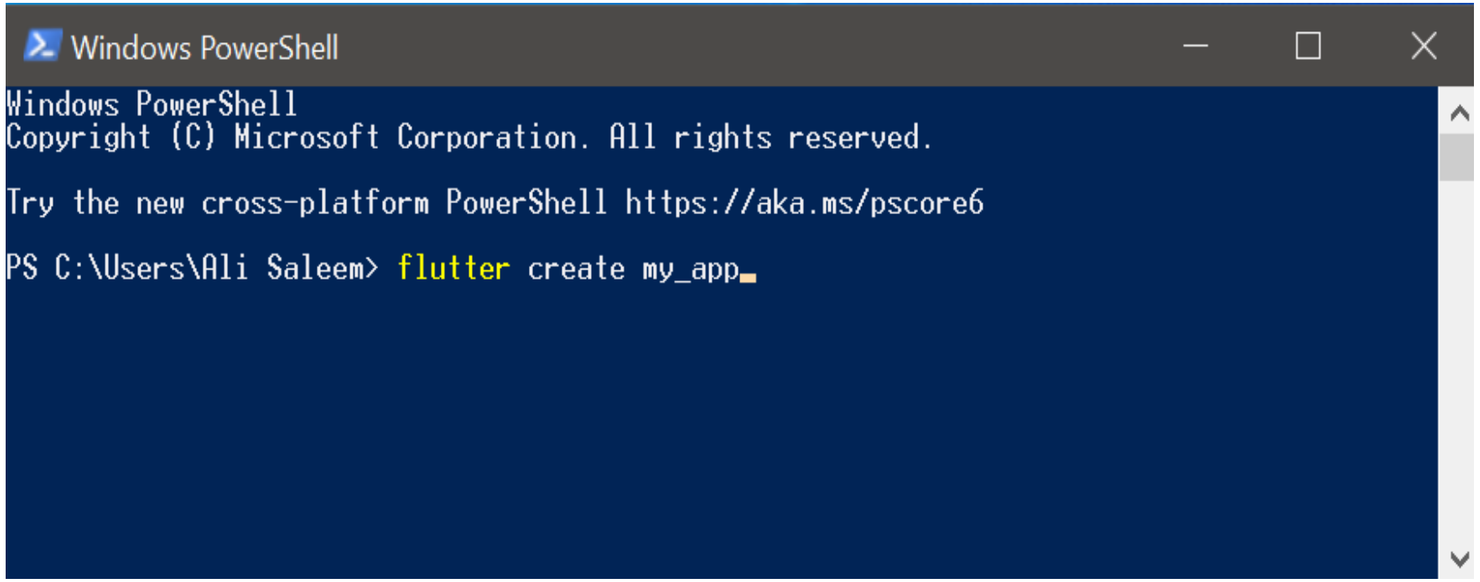
Apply best practices for organizing large Flutter projects.

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

Creating a Project

On PowerShell write the 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

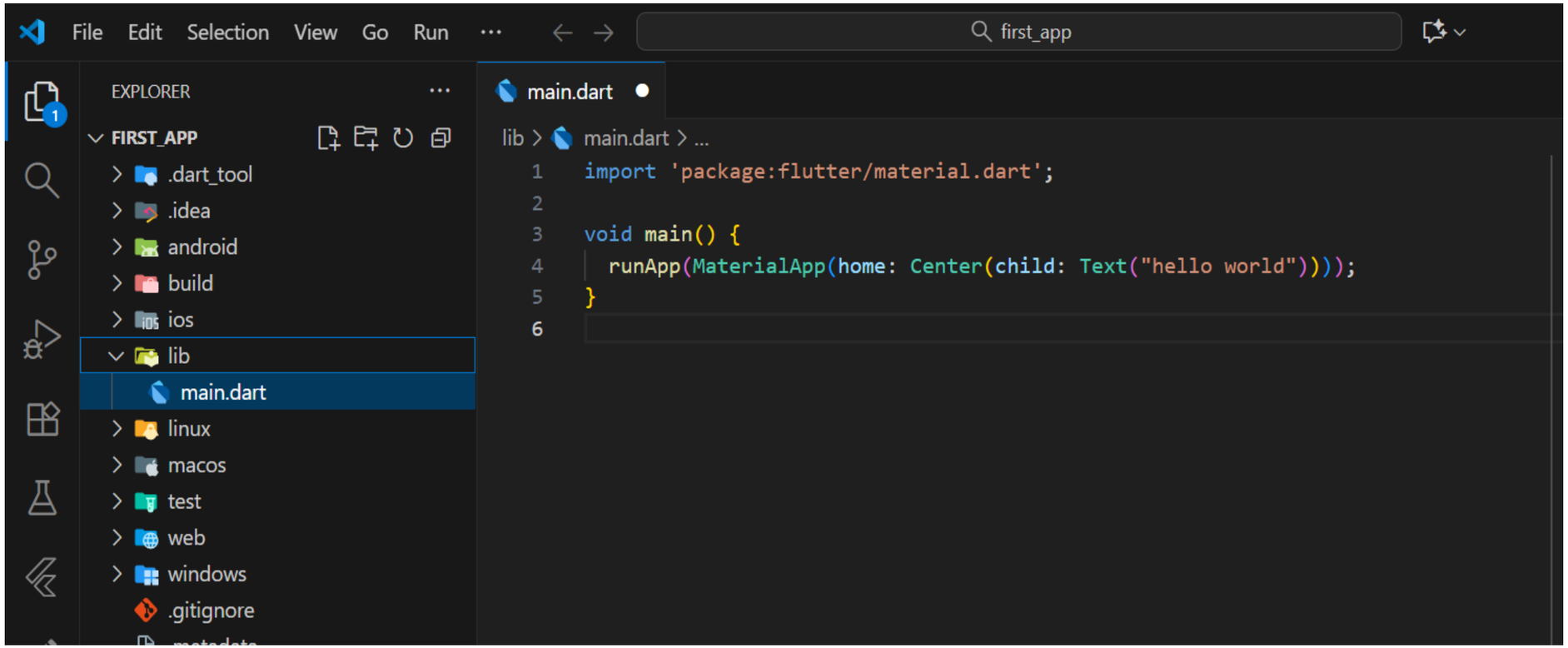
app metadata, dependencies, and assets

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

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.

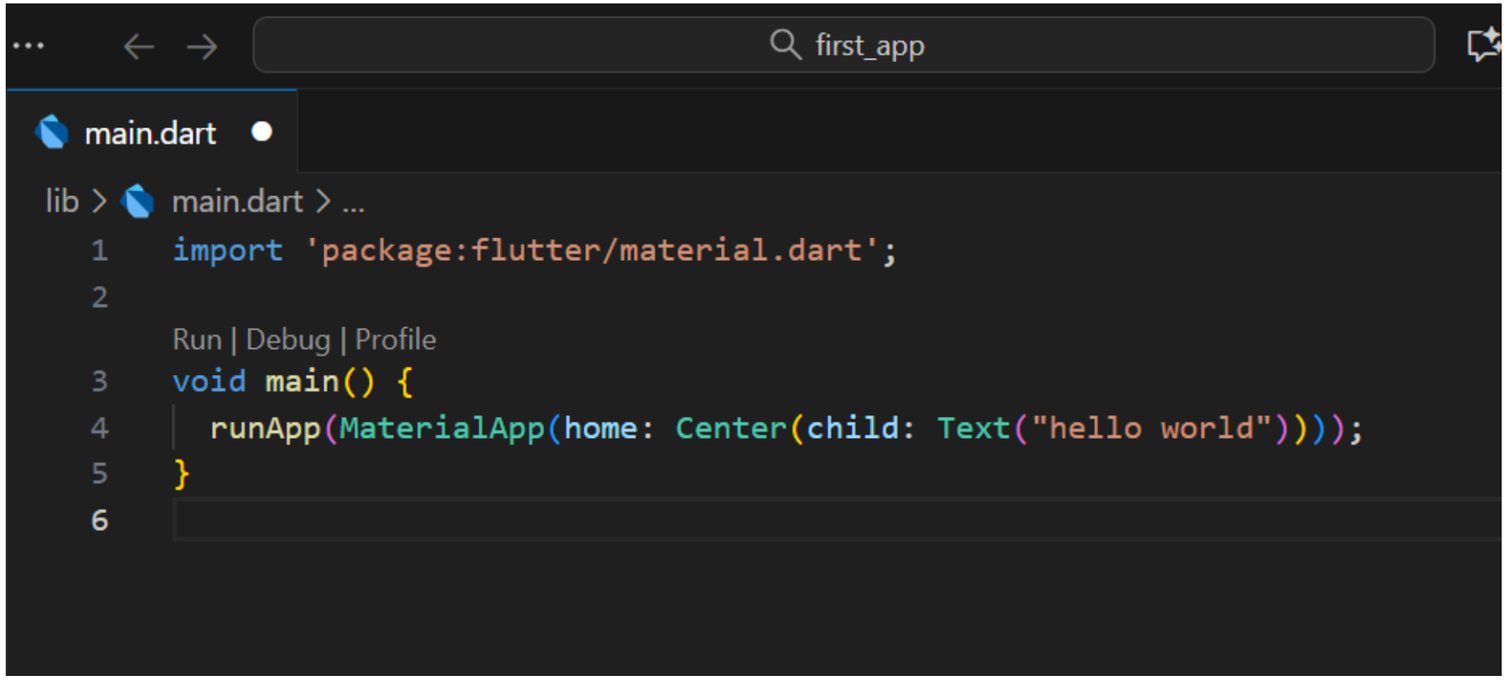
Purpose of main.dart

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.

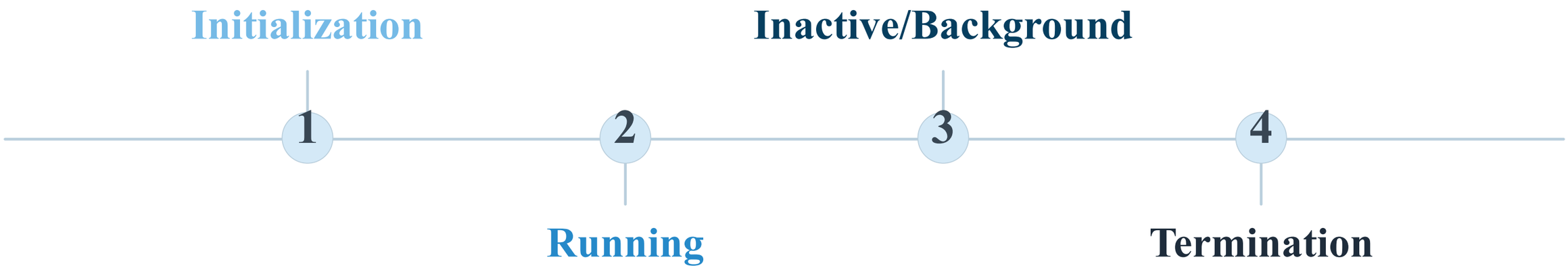
2

Returns a MaterialApp or CupertinoApp.

3

Defines theme, routes, and home screen.

Application Lifecycle Overview



Crucial for managing memory, state, and background services.

Initialization Phase

- This is the **starting phase** of the Flutter application lifecycle.
- It begins when the runApp() function is executed and the app is loaded into memory.
- During this phase, all essential components are **initialized**, and the root widget (usually MyApp) is built for the first time.

```
void initState() {
  super.initState();
  initializeData(); // Prepare resources before UI rendering
}
```

Running Phase

- In this phase, the app is **fully active and interactive**.
- The user can view screens, press buttons, and navigate through pages.
- The Flutter framework continuously rebuilds widgets whenever the state changes

```
setState(() {
  counter++; // Update UI dynamically
});
```

Inactive / Background Phase

- This occurs when the app is temporarily not visible but still in memory — for example, when the user receives a call, opens another app, or locks the screen.
- Flutter provides the ***AppLifecycleState*** to handle these transitions.

```
@override
void didChangeAppLifecycleState(AppLifecycleState state) {
  if (state == AppLifecycleState.paused) {
    saveUserProgress(); // Save data before going to background
  }
}
```

Termination Phase

- This is the final phase when the app is completely closed by the user or the operating system. All resources are released, and background services stop running.

```
@override
void dispose() {
  controller.dispose(); // Release memory
  super.dispose();
}
```

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.

Hot Reload

- Hot Reload is a Flutter feature that allows developers to **inject updated source code directly into the running Dart Virtual Machine (VM)** without restarting the entire application.

- When you save your file, Flutter recompiles only the modified code and updates the app's UI **instantly**, while **preserving the current state** of widgets.

Hot Restart

- Hot Restart completely **restarts the app from the main() function**, clearing all stored variables and states in memory.

- Unlike Hot Reload, it does not preserve the app’s current state. The entire widget tree is rebuilt, reinitializing global variables and running the app as if it were just launched.

Performance & Use Cases

Hot Reload
faster iteration for UI tweaks.

Hot Restart
necessary for structural or state-critical changes.

Key Resources

Flutter official docs:

https://docs.flutter.dev

Dart language:

https://dart.dev

Textbook:

Flutter Complete Reference (Alberto Miola, 2023).

Thank you…

Any questions??

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

My google site