Flutter

Short notes for beginners · Dart · widgets · Android · iOS · app developers

Uncodemy Handwritten Notes · free for students

I. Flutter Basics

- **Flutter** is Google's open source UI toolkit. One codebase builds apps for Android, iOS, web, Windows, macOS and Linux. It uses the **Dart** language and draws its own UI with its own rendering engine, so the look is the same on every device.
- Architecture: **Framework** (Dart: widgets, rendering, gestures) → **Engine** (C++: Skia / Impeller graphics, text, Dart runtime) → **Embedder** (platform specific code). Everything is a widget.
- **Hot reload** injects changed code in under a second and keeps the app state. **Hot restart** restarts the app and resets state. Use r for reload and R for restart in the terminal.

➤ Setup and project structure

- Install the Flutter SDK, Android Studio (emulator) or VS Code with the Flutter plugin. Run flutter doctor to check the setup. Create: flutter create my_app. Run: flutter run.

File / folder	Purpose
lib/main.dart	Entry point, main() calls runApp()
pubspec.yaml	Dependencies, assets, fonts, app version
android/, ios/	Native platform projects
test/	Unit and widget tests
assets/	Images, fonts, JSON files (declare in pubspec)

2. Dart Language Basics

- Variables: var (type inferred), int, double, String, bool. **final** = set once at runtime. **const** = compile time constant. **dynamic** can hold any type (avoid when possible).
- Collections: List [1, 2, 3], Set {1, 2}, Map {"a": 1}. Useful methods: add, remove, map, where, forEach, reduce, contains, length. String interpolation: "Hello $name, total ${a + b}".
- **Null safety**: a type is non-null by default. String? can be null. Operators: ?. (safe call), ?? (default value), ! (assert not null), late (initialise later).

EXAMPLE String? name; print(name ?? "Guest"); // prints Guest int total(int a, [int b = 0]) => a + b; void greet({required String user}) { }

- Functions: return type, optional positional [ ], named parameters { } with required, arrow function =>. Functions are first class values (can be passed around). main() is the entry point.
- Control flow: if / else, switch, for, while, for-in, ternary (a ? b : c), collection if and for inside lists.

3. Dart OOP and Async

- **Class** with fields, constructor and methods. Short constructor: Person(this.name, this.age);. Named constructor: Person.guest(). Inheritance: extends, super. **abstract** class cannot be instantiated. **implements** copies the interface. **mixin** / with reuses code across classes.

Keyword	Use
extends	Single inheritance
implements	Promise to provide all members of an interface
with	Add mixin behaviour
enum	Fixed set of values
static	Belongs to the class, not objects

Keyword	Use
factory	Constructor that may return an existing object

- Private members start with an underscore: _count (private to the library / file). Getters and setters: get area => w * h;
- **Async**: a **Future** is a value available later (one result). Use async / await. A **Stream** gives many values over time (await for or listen). Dart is single threaded with an event loop; use **Isolates** for heavy work.

EXAMPLE Future<String> getName() async { await Future.delayed(Duration(seconds: 2)); return "Asha"; } var n = await getName();

4. Widgets: Stateless and Stateful

- A **widget** is a description of part of the UI. Widgets form a tree; Flutter rebuilds only what changed. **StatelessWidget**: no changing data (logo, label). **StatefulWidget**: has a State object that changes (counter, form, toggle).

Point	StatelessWidget	StatefulWidget
State	Immutable	Mutable via setState()
Classes	One	Two: widget + State
Rebuild	Only when parent changes	On setState or parent change
Example	Text, Icon, simple card	Checkbox, TextField, counter

EXAMPLE class Counter extends StatefulWidget { ... } in State: int _count = 0; onPressed: () { setState(() { _count++; }); } Text("$_count")

- **State lifecycle**: createState → initState (once) → didChangeDependencies → build (many times) → didUpdateWidget → setState → deactivate → dispose (free controllers and streams).
- App skeleton: runApp(const MyApp()) → MaterialApp (theme, routes, home) → **Scaffold** with appBar, body, floatingActionButton, drawer, bottomNavigationBar. **BuildContext** tells a widget where it is in the tree.

- Use **const** constructors where possible (const Text("Hi")). They avoid needless rebuilds and improve performance. Keys (ValueKey, GlobalKey) help Flutter identify widgets in lists.

5. Layout Widgets

Widget	Use
Container	Box with size, colour, padding, margin, decoration
Row / Column	Children horizontally / vertically
Stack + Positioned	Overlap widgets (badge on an image)
Expanded / Flexible	Share remaining space inside Row, Column
Padding, Center, Align, SizedBox	Spacing, alignment, fixed gaps
ListView / GridView	Scrollable list / grid (use .builder for long data)
Wrap	Row that wraps to next line
Card, ListTile	Material card, standard list row
SingleChildScrollView	Makes a Column scrollable

- **Row / Column**: mainAxisAlignment (along the direction: start, center, end, spaceBetween, spaceAround, spaceEvenly) and crossAxisAlignment (across: start, center, end, stretch). mainAxisSize: min or max.
- **Layout rule**: constraints go down, sizes go up, parent sets the position. A child picks its own size inside the limits given by its parent.

REMEMBER Yellow and black stripes = RenderFlex overflow. Fix with Expanded, Flexible, scroll views or smaller content. ListView inside Column needs Expanded or shrinkWrap.

- Responsive UI: MediaQuery.of(context).size for screen size, LayoutBuilder for parent constraints, OrientationBuilder, AspectRatio, FractionallySizedBox.

6. Common Widgets and Theming

- **Text** (style: TextStyle), **Image** (Image.asset, Image.network, fit: BoxFit.cover), **Icon**, buttons: ElevatedButton, TextButton, OutlinedButton, IconButton, FloatingActionButton.
- Input: **TextField** with TextEditingController (dispose it!), **Form** + GlobalKey<FormState> + TextFormField validator, Checkbox, Radio, Switch, Slider, DropdownButton, DatePicker.
- Feedback: AlertDialog (showDialog), SnackBar (ScaffoldMessenger), BottomSheet, CircularProgressIndicator, RefreshIndicator.

EXAMPLE TextFormField(validator: (v) => v!.isEmpty ? "Required" : null) if (_formKey.currentState!.validate()) { save(); }

- **Theming**: ThemeData(colorSchemeSeed: Colors.indigo, useMaterial3: true), light and dark theme, Theme.of(context).textTheme. Custom fonts and images are declared in pubspec.yaml under fonts: and assets:.
- Material widgets follow Google design, Cupertino widgets look like iOS. Gestures: GestureDetector, InkWell (ripple), Dismissible (swipe), Draggable.

7. Navigation and State Management

- **Navigator**: Navigator.push(context, MaterialPageRoute(builder: (_) => Detail())) opens a screen, Navigator.pop(context) goes back. Named routes: routes: { "/detail": ... } and Navigator.pushNamed. For deep links and web use the go_router package.
- Pass data using constructor parameters, and return data with pop(result). Bottom navigation: BottomNavigationBar or NavigationBar with a selected index kept in state.

Approach	Use when
setState	State is local to one widget

Approach	Use when
InheritedWidget	Share data down the tree (base of Provider)
Provider	Simple and popular, ChangeNotifier + Consumer
Riverpod	Safer, testable, no BuildContext needed
Bloc / Cubit	Large apps, strict event and state flow
GetX	Quick apps, all in one (navigation, state, DI)

- Provider flow: class CartModel extends ChangeNotifier → call notifyListeners() after change → wrap app with ChangeNotifierProvider → read with context.watch or Consumer, call methods with context.read.

8. Networking, Storage and Firebase

- **http** package: final res = await http.get(Uri.parse(url)); if (res.statusCode == 200) { var data = jsonDecode(res.body); }. **Dio** adds interceptors, timeouts and uploads.
- Show async data with **FutureBuilder** (one future) or **StreamBuilder** (stream), and handle the states loading, error, data. Convert JSON with model classes: Model.fromJson(map) and toJson().

Storage	Use
shared_preferences	Small key-value data (settings, login flag)
sqflite / drift	Local SQLite database
Hive / Isar	Fast local NoSQL
path_provider	Find device folders for files
flutter_secure_storage	Tokens and secrets (keystore / keychain)

- **Firebase**: Authentication (email, Google, phone), Cloud Firestore (realtime database), Storage, Cloud Messaging (push), Analytics, Crashlytics. Add packages with flutter pub add package_name, then flutter pub get.
- Native features come from plugins: camera, geolocator, image_picker, url_launcher, permission_handler. Platform channels call native Android / iOS code when no plugin exists.

9. Testing, Build and Release

- Modes: **Debug** (hot reload, slow), **Profile** (performance check), **Release** (optimised, final). Test types: **unit** (logic), **widget** (single UI part), **integration** (full app). Run: flutter test.
- Tools: Flutter DevTools (widget inspector, performance, memory), flutter analyze, dart format. Performance tips: const widgets, ListView.builder for long lists, split big widgets, cache images, dispose controllers.

Command	Output
flutter build apk	APK file for Android install
flutter build appbundle	AAB file for Google Play
flutter build ios	iOS build (needs a Mac and Xcode)
flutter build web	Web files for hosting

- Android release: create a keystore, set signing in build.gradle, update app icon, name, version in pubspec.yaml (1.0.0+1). Google Play has a one time developer fee of 25 USD. Apple Developer Program costs 99 USD per year.
- Good habits: clean folder structure (models, services, screens, widgets), Git, follow Dart style, handle errors and loading states, support dark mode, test on real devices.

10. Most Asked Questions (practice)

1. What is Flutter? Explain its architecture and why Dart is used.
2. Hot reload vs hot restart. Stateless vs Stateful widget.
3. Explain the StatefulWidget lifecycle.
4. Difference between final and const. What is null safety?
5. Explain Row, Column, Stack, Expanded and ListView.builder.
6. What is the widget, element and render tree?
7. Future vs Stream. Explain async / await with FutureBuilder.
8. State management options: setState, Provider, Bloc. When to use which?

9. How do you call an API and show the data in a list?

10. How do you build and publish an app to the Play Store?

TIP Build 3 apps for your portfolio: a to-do app with local storage, a weather or news app with an API, and an app with Firebase login. Upload the code to GitHub with screenshots.