Welcome!

The workshop will include **interactive** segments. In order to participate*, make sure that you do the following **before we begin**:

- Complete the installations (instructions were sent to you).
- `git clone https://github.com/ofek-bs/cs_hackathon_flutter_2024.git`
- Open the cloned folder in your IDE, make sure that you're working on the `master` branch, run your emulator and run the given code as you've learned.

* The interactive parts can be done in groups ☺

Android Workshop

CS Hackathon 2024 | Ofek Bengal Shmueli

Plans For Today

- Understand **what Flutter's all about** and why is it recommended to use.
- Learn the **basic UI concepts** in depth.
- See some **actual code** and play with it.
- Overview interesting **features and packages** that might be useful for your projects.
- Get to know **great self-learning sources**, mainly for documentation, packages, etc.
- Questions?

Plans For Today

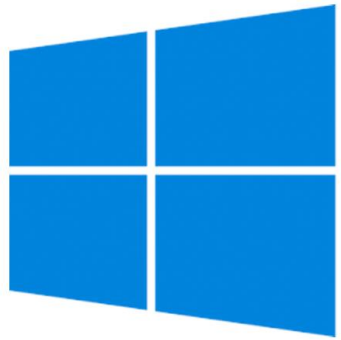
- Understand **what Flutter's all about** and why is it recommended to use.
- Learn the **basic UI concepts** in depth.
- See some **actual code** and play with it.
- Overview interesting **features and packages** that might be useful for your projects.
- Get to know **great self-learning sources**, mainly for documentation, packages, etc.
- Questions?

Motivation

Java, Kotlin, XML
(Android Native)

Swift, Obj-C

JS, HTML, CSS



C#

- We need to specialize in different languages for different devices.
- An additional platform == An additional codebase.
- Logic and UI are often separated and hard to maintain.

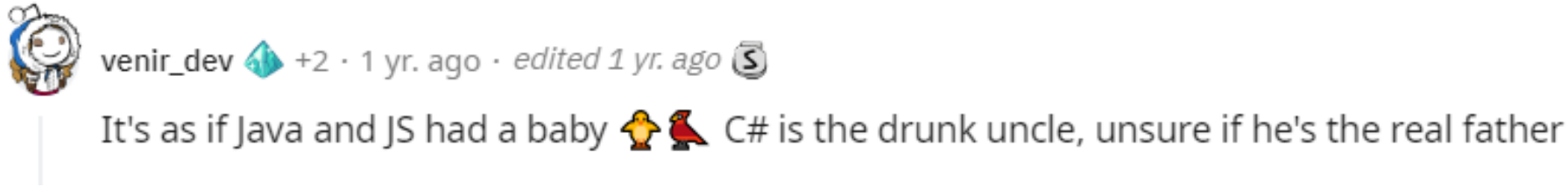
Motivation

Context

- **Flutter** is Google's toolkit for building cross-platform apps, using a single programming language.
  - Tools for compiling, debugging, testing
  - Foundation library (primitives, utility classes and functions)
  - UI library (widgets)
- That programming language is called **Dart**.

Dart in a nutshell

- Mainly used for UI & consumer applications.
- Similar to other languages you (possibly) know and (certainly) love.



- Interesting features:
  - OOP
  - Support for asynchronous development (future, async, await)

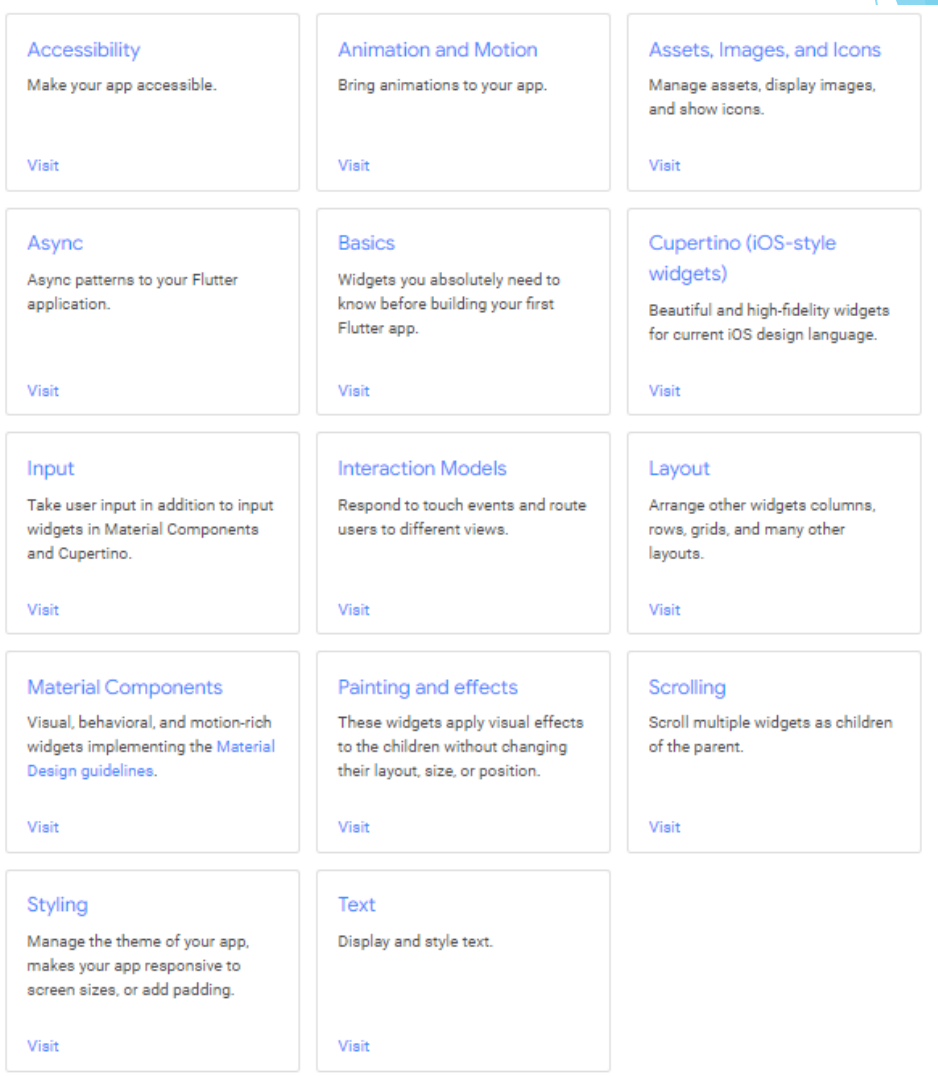
Dart

Plans For Today

- Understand **what Flutter's all about** and why is it recommended to use.
- Learn the **basic UI concepts** in depth.
- See some **actual code** and play with it.
- Overview interesting **features and packages** that might be useful for your projects.
- Get to know **great self-learning sources**, mainly for documentation, packages, etc.
- Questions?

Widgets

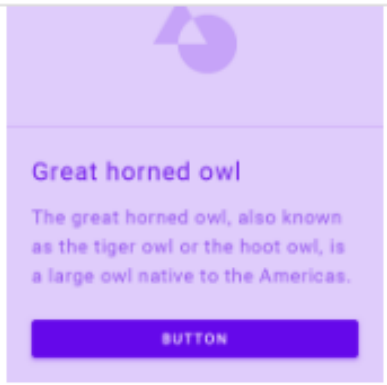
- (Almost) **everything is a widget.**
  - Lists, buttons, images, text fields…
- **Flutter UI** is built by composing widgets.
- Flutter code defines a giant hierarchy of widgets
  - Also known as **The Widget Tree.**
- Follows **Material Design.**



Basic Widgets - Examples

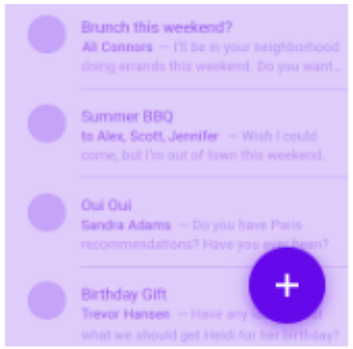
Text

A run of text with a single style.



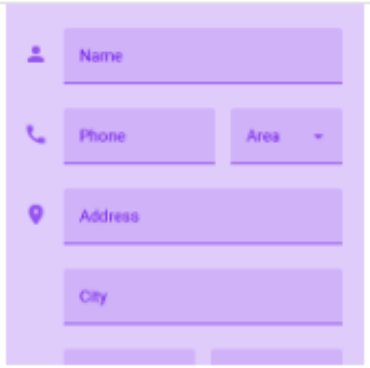
ElevatedButton

A Material Design elevated button. A filled button whose material elevates when pressed.



FloatingActionButton

A floating action button is a circular icon button that hovers over content to promote a primary action in the application. Floating action buttons are...



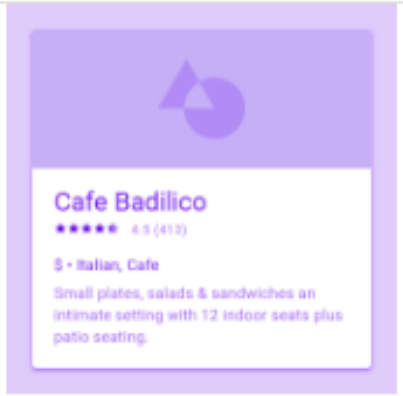
TextField

Touching a text field places the cursor and displays the keyboard. The TextField widget implements this component.

Date & Time Pickers

Date pickers use a dialog window to select a single date on mobile. Time pickers use a dialog to select a single time (in the...

Basic Widgets - Examples

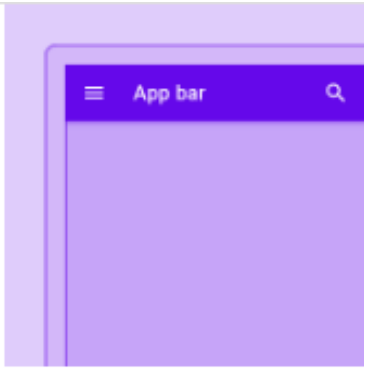


Card

A Material Design card. A card has slightly rounded corners and a shadow.

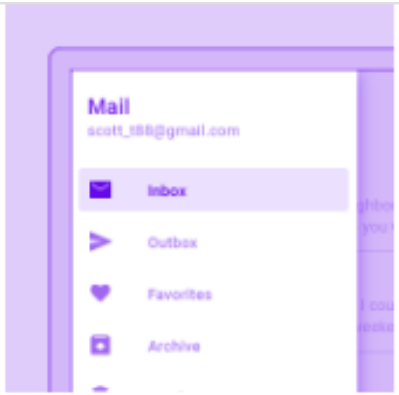
Form

An optional container for grouping together multiple form field widgets (e.g. TextField widgets).



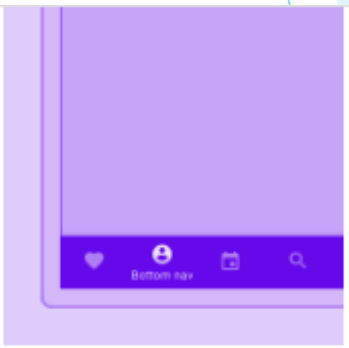
AppBar

A Material Design app bar. An app bar consists of a toolbar and potentially other widgets, such as a TabBar and a FlexibleSpaceBar.



Drawer

A Material Design panel that slides in horizontally from the edge of a Scaffold to show navigation links in an application.



BottomNavigationBar

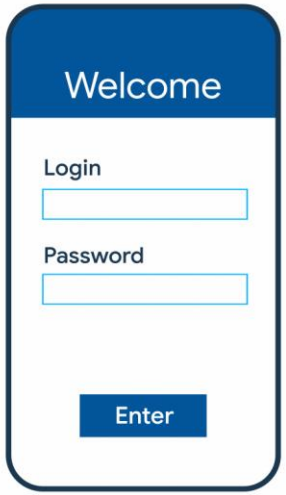
Bottom navigation bars make it easy to explore and switch between top-level views in a single tap. The BottomNavigationBar widget implements this component.

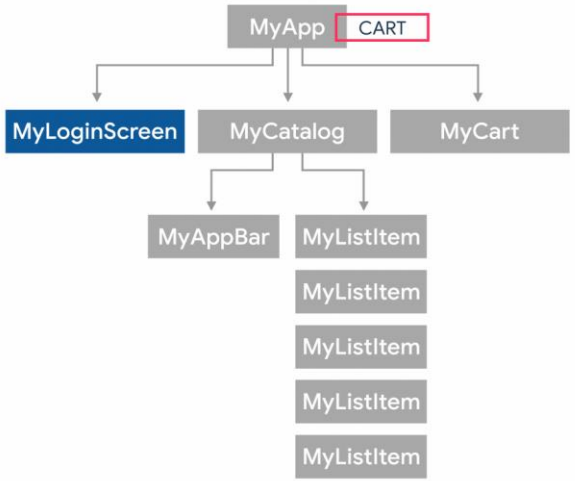
Reactive Programming

- State = all the information that is part of the widget / app.
  - Variables (their values)
  - User input
  - Cache, Etc.
- Declarative UI = **User interface** is defined declaratively, as the result of a **constant function (build)** given the **current state**.

Reactive Programming (cont.)

- Flutter is a **declarative and reactive** framework:
  - The developer **declares** how the program should behave (By managing the state and hierarchy of the UI), and the framework handles (most of) everything else.
  - The program changes its state and UI by **reacting** to changes in data.





Reactive Programming (cont.)

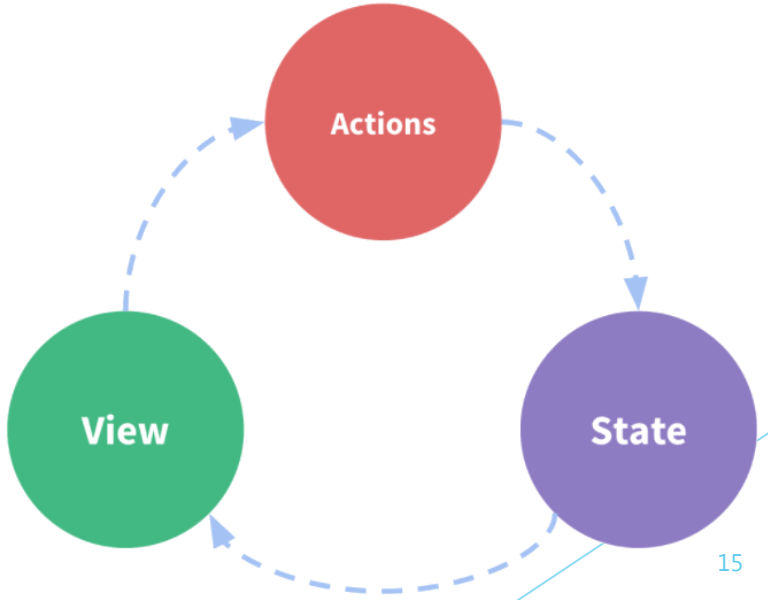
- In **Reactive programming**, we can divide any application into 3 parts:
  - **The state** – source of truth that drives the app.
  - **The view** – a declarative mapping of the state.
  - **The actions** – the possible ways the state could change in reaction to user inputs from the view.

UI = f(state)

The layout on the screen

Your build methods

The application state



Stateless Widgets

- **Stateless Widgets** are the simplest form of widgets, and **don't maintain a state**.
- In other words, **UI** = **f(initial configuration)**, and therefore UI is immutable.
- Their **build** method is called less often. Therefore, they offer better performance.
- **Studio cheat:** stless + Enter = StatelessWidget.

DEMO

Layout Widgets

- In other platforms, arrangements of UI components is done with XML, CSS…
- Here, the arrangement is also done with **widgets**.

Examples:

- Arrange widgets horizontally / vertically by wrapping them with **Row / Column**.
- Center a widget by wrapping it with a **Center** widget.
- Add padding, background color and much more with **Container**.

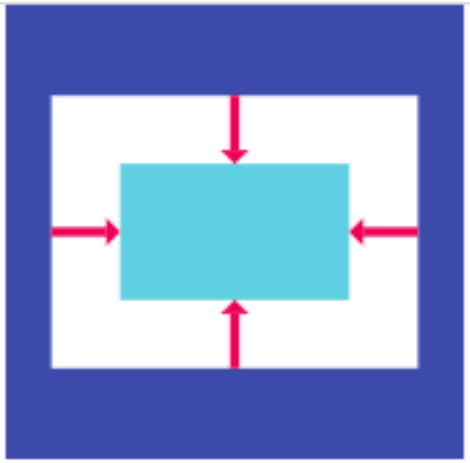
Layout Widgets - Examples

Column

Layout a list of child widgets in the vertical direction.

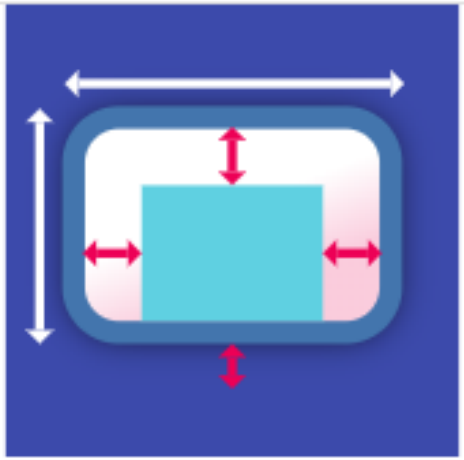
Row

Layout a list of child widgets in the horizontal direction.



Center

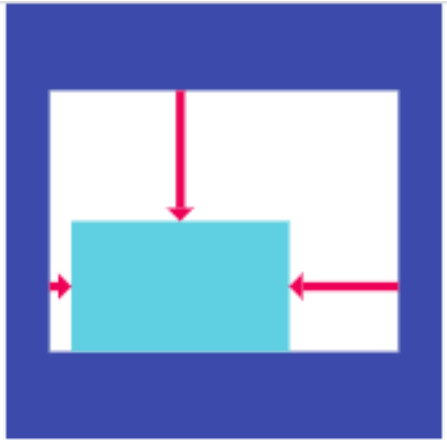
A widget that centers its child within itself.



Container

A convenience widget that combines common painting, positioning, and sizing widgets.

Layout Widgets - Examples

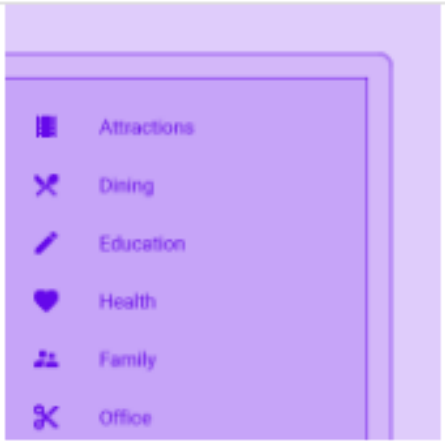


Align

A widget that aligns its child within itself and optionally sizes itself based on the child's size.

Stack

This class is useful if you want to overlap several children in a simple way, for example having some text and an image, overlaid with...



ListView

A scrollable, linear list of widgets. ListView is the most commonly used scrolling widget. It displays its children one after another in the scroll direction....

GridView

A grid list consists of a repeated pattern of cells arrayed in a vertical and horizontal layout. The GridView widget implements this component.

Exercise 1

- In the cloned git repo, switch to the **1_stateless** branch.
- Run the app on your emulator.
- In `main_screen.dart`, add the necessary code such that:
  - A rectangle button is added below the text.
  - The button should contain the text “My Button”.
  - Both the text and button are centered (middle of the screen).

- **It is recommended to use online sources!**

Stateful Widgets

- **Stateful Widgets** hold a **state**.
- In other words, **UI** = **f(current state)**, and therefore UI is mutable.
- To update the state of the widget, wrap mutable changes with a call to `setState()`.
- Such a call marks the wrapping widget as “dirty”, thus causing a rebuild for the widget (and the subtree below) = calling **build** = redrawing related UI according to the updated state.
- Composed of a state class (where all the logic is added) and a regular widget class (which usually contains boilerplate code).
- **Studio cheat:** stful + Enter = StatefulWidget + State class.

DEMO

Exercise 2

- In the cloned git repo, switch to the **2_tamago** branch.
- Run the app on your emulator.
- In `egg_screen.dart`, add the necessary code such that:
  - **Our game is fixed**. That is: Whenever the egg is tapped, the text which displays the current count and the image should reflect the actual _count value.
  - Increase the font size for the "_count" label.
  - Make the "_count" label **bold**.
- **It is recommended to use online sources!**

Plans For Today

- Understand **what Flutter's all about** and why is it recommended to use.
- Learn the **basic UI concepts** in depth.
- See some **actual code** and play with it.
- Overview interesting **features and packages** that might be useful for your projects.
- Get to know **great self-learning sources**, mainly for documentation, packages, etc.
- Questions?

Flutter Packages

- Flutter supports using shared packages, contributed by other developers, to allow quickly building an app without having to develop everything from scratch.
- There are packages for almost everything!
- Packages are added are updated on a daily basis.
  - You can create packages too ☺
- You can find them in pub.dev, and install them by running:
  - `flutter pub add <package_name>`

Camera Interaction

- Sometimes we need access to the user's camera, to take pictures or capture video.
  - For example: Uploading an avatar picture.
- We can use the **camera** package.
- This package provides us with 2 main interfaces:
  - **`CameraPreview`** widget, for getting a feed from the camera.
  - **`CameraController`** for controlling the camera
    - Taking pictures
    - Switching camera
    - Recording video, etc.

Camera Interaction – Example (1/2)

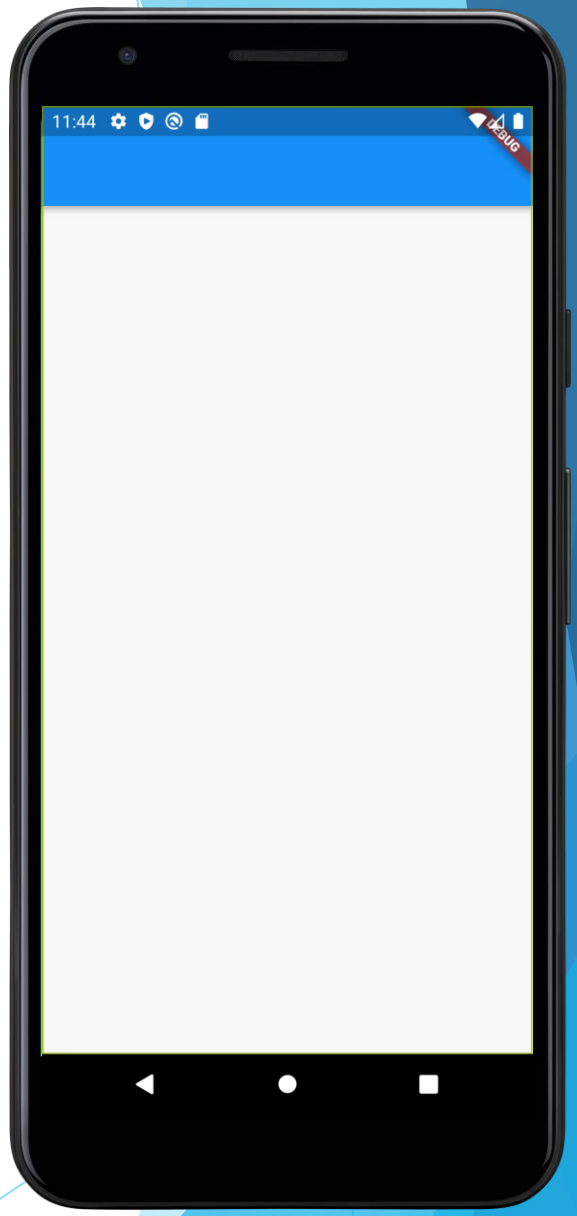
```
class _MyHomePageState extends State<MyHomePage> {
  late CameraController cameraController;
  bool cameraReady = false;

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

  void startCamera() async {
    List<CameraDescription> cameras = await availableCameras();
    cameraController = CameraController(cameras[0], ResolutionPreset.high);
    await cameraController.initialize();
    setState(() { cameraReady = true; });
  }

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

Camera Interaction – Example (2/2)

```
class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
     return Scaffold(
       appBar: AppBar(),
       body: cameraReady ? CameraPreview(cameraController) : Container();
     );
  }
}
```

- After initializing the controller, we can use it for many camera features, such as taking pictures, changing zoom level, etc.

```
cameraController.takePicture();
```



Location awareness

- Sometimes we want our application to be location-aware.
  - For example: Navigation apps.
- We can use the **location** package.
- Allows us to get **live location** of our device and be notified whenever it changes.
- We access the package's functions via an instance:

```
var location = Location();
```

Location awareness (cont.)

- In order to access the device's location, we need to:
  - Make sure that the **device's location service** is enabled.
  - Make sure that the **app's location permission** is granted.

```
var serviceEnabled = await location.serviceEnabled();
if (!serviceEnabled) {
  serviceEnabled = await location.requestService();
  if (!serviceEnabled) {
    return;
  }
}

var permissionGranted = await location.hasPermission();
if (permissionGranted == PermissionStatus.denied) {
  permissionGranted = await location.requestPermission();
  if (permissionGranted != PermissionStatus.granted) {
    return;
  }
}

setState(() { _locationAvailable = true; });
```

Location awareness (cont.)

- Now, we can access our device's latest known location with:

```
var _locationData = await location.getLocation();
```

```
LocationData<lat: 32.7777353, long: 35.0216254>
```

- Or request location updates in a **Stream**:

```
location.onLocationChanged.listen((newLoc) {
  print(newLoc);
});
```

- We can also change the update interval and accuracy:

```
location.changeSettings(
  accuracy: LocationAccuracy.powerSave,
  interval: 10000, // Every 10 seconds
);
```

Location awareness – Maps

- Sometimes we want to display location-based information for our users on a map.
- Flutter has 2 recommended APIs for that: **Google Maps** and **Leaflet**

API	Package	Pros	Cons
Google Maps	google_maps_flutter	• Supported by Google • Easier to setup	• Not very customizable • Limited quota
Leaflet	flutter_map	• Highly customizable • Free for life	• Relatively new • No official Google support

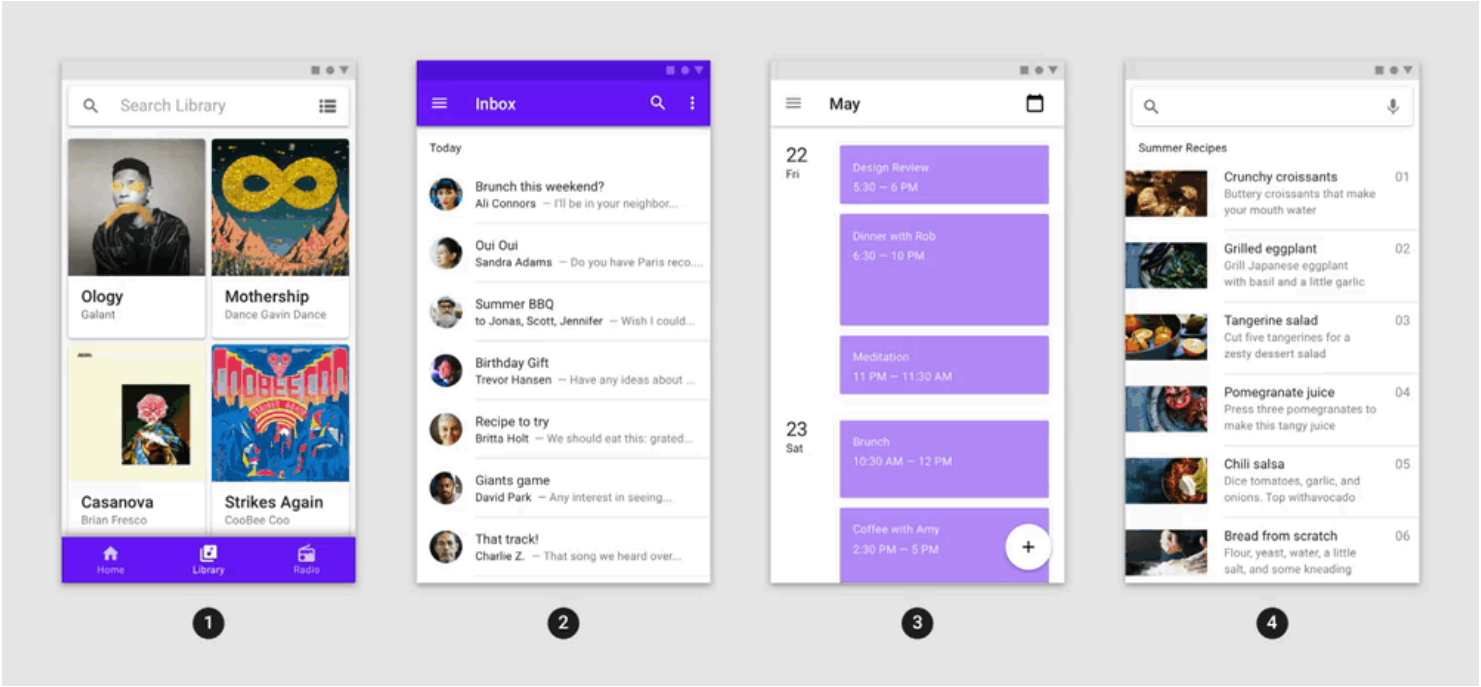
Firebase Services

- A Google platform which provides a variety of cloud services, along with an intuitive SDK. Suitable for small to medium-scale projects.

- In Flutter, we use those services via the official packages:
  - **firebase_auth** allows users to sign up and log into the app, with a classic email/password credentials or with Facebook/Google/Phone…
  - **cloud_firestore** allows users to maintain an easy-to-use NoSQL database, listen to changes and respond to them, queries…
  - **firebase_storage** allows users to save files in the cloud: images, profile pictures, PDFs, chat logs…
  - **firebase_messaging** allows users to send and receive push notifications.
    - A bit harder to implement, but still not too bad!

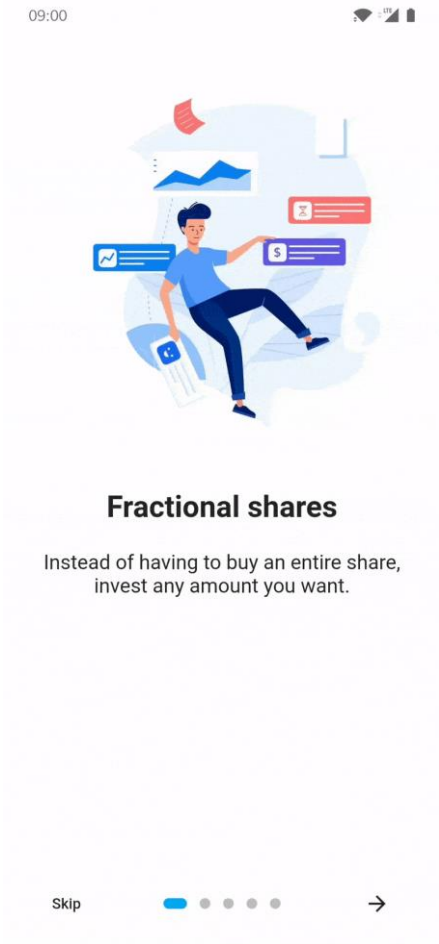
UI/UX

- Flutter's **animations** package includes pre-made animations for commonly-desired effects.



UI/UX

- You can easily change your app's launcher icon with **flutter_launcher_icons**.
- You can add a nice splash screen with no effort by using **flutter_native_splash**.
- Welcome your first-time users with an introduction screen, created with the help of **introduction_screen**.



Gaming

- **flame** is an engine which provides utility functions and APIs useful for game development, including:
  - Game loop
  - Characters and objects movement.
  - Collision detection.
  - Images, animation and sprites.

FLAME

The List Goes On…

- image_picker
- share_plus
- flutter_login
- google_sign_in
- mobile_scanner
- sensors_plus
- story_view
- health
- intl

Plans For Today

- Understand **what Flutter's all about** and why is it recommended to use.
- Learn the **basic UI concepts** in depth.
- See some **actual code** and play with it.
- Overview interesting **features and packages** that might be useful for your projects.
- Get to know **great self-learning sources**, mainly for documentation, packages, etc.
- Questions?

Useful Links

- For packages, check out pub.dev.
- For widgets & APIs, check out Flutter's catalog.
- For learning about new widgets, check out the "Widget of the Week" YouTube playlist.
- For advanced articles about Flutter, check out Medium's Flutter community.

- If you have questions, try to contact me ☺

Plans For Today

- Understand **what Flutter's all about** and why is it recommended to use.
- Learn the **basic UI concepts** in depth.
- See some **actual code** and play with it.
- Overview interesting **features and packages** that might be useful for your projects.
- Get to know **great self-learning sources**, mainly for documentation, packages, etc.
- Questions?

THANK YOU!