

# ESCRIU LA TEVA PRIMERA APP AMB FLUTTER



**Eduard Carreras**

Desenvolupament d'Aplicacions mòbils per iOS i Android amb Flutter  
Activitat d'Aprenentatge 1.2

## ÍNDEX

<b>INTRODUCCIÓ</b>	<b>2</b>
<b>QUÈ APRENDRÀS?</b>	<b>2</b>
<b>QUÈ CONSTRUIRÀS?</b>	<b>2</b>
<b>ENTORN</b>	<b>2</b>
<b>CREACIÓ DE L'APLICACIÓ FLUTTER INICIAL</b>	<b>3</b>
<b>UTILITZACIÓ DE PAQUETS EXTERNS</b>	<b>5</b>
<b>AFEGIR UN WIDGET AMB ESTAT</b>	<b>7</b>
<b>CREAR UN LISTVIEW DE DESPLAÇAMENT INFINIT</b>	<b>11</b>
<b>TASQUES ADDICIONALS (OPCIONAL)</b>	<b>15</b>
<b>REFERÈNCIES</b>	<b>15</b>

## INTRODUCCIÓ

En aquesta activitat d'aprenentatge, crearàs una aplicació mòbil senzilla en Flutter. Si esteu familiaritzats amb el codi orientat a objectes i conceptes bàsics de programació, com ara variables, bucles i condicionals, podreu completar l'Activitat d'Aprenentatge. No cal experiència prèvia amb programació de Dart, mòbil, d'escriptori o web. L'activitat us guiarà pas a pas.

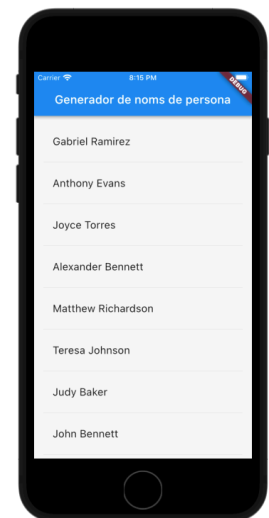
## QUÈ APRENDRÀS?

- Com escriure una aplicació Flutter que funcioni a iOS i Android
- Estructura bàsica d'una aplicació Flutter
- Trobar i utilitzar paquets per ampliar la funcionalitat
- A utilitzar el hot reload per a un cicle de desenvolupament més ràpid
- Com implementar un widget amb estat
- Com crear una llista infinita que s'omple de manera dinàmica

## QUÈ CONSTRUIRÀS?

Implementarem una aplicació senzilla que generi noms de persones. El codi genera els noms que apareixen a la pantalla. A mesura que l'usuari es desplaça, es generen més noms. No hi ha límit a la distància que un usuari pot desplaçar.

Aquesta imatge mostra com l'app quedaria en finalitzar l'activitat.



## ENTORN

Per completar l'activitat necessiteu:

1. Tenir instal·lat el Flutter SDK
2. Tenir instal·lat Visual Studio Code o Android Studio
3. Una d'aquestes tres opcions:
  - a. Un dispositiu físic Android o iOS connectat a l'ordinador i configurat en mode de desenvolupador
  - b. El simulador d'iOS (requereix la instal·lació d'eines Xcode)

- c. L'emulador d'Android (requereix configuració a Android Studio)

## CREACIÓ DE L'APLICACIÓ FLUTTER INICIAL

Crea un projecte Flutter:

- A Visual Studio Code, obre la paleta d'ordres (amb **F1**, **Ctrl+Maj+P** o **Maj+Cmd+P**) i escriu i selecciona **"Flutter: nou projecte"**.
- A Android Studio, pots utilitzar el **Menú "Fitxer -> Nou -> Nou projecte Flutter"**
- També podeu crear-lo per línia de comandes amb aquesta instrucció:

```
$ flutter create actividad_1
$ cd actividad_1
```

El primer que farem és editar l'exemple que es crea per defecte.

Edita el contingut de `lib/main.dart`. Suprimeix **tot el codi** de `lib/main.dart` i **substitueix-lo** pel codi següent, que mostra "Hola món" al centre de la pantalla:

```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

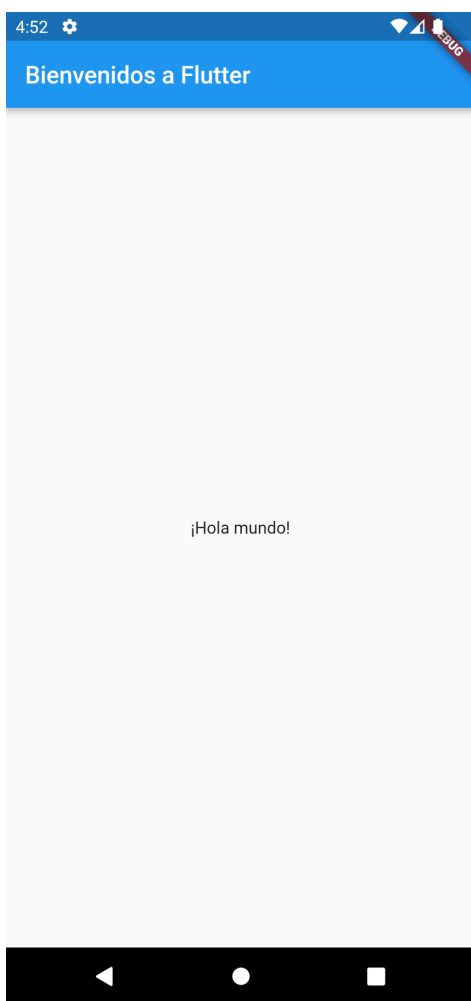
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blue,
          foregroundColor: Colors.white,
          title: const Text('Benvinguts a Flutter'),
        ),
        body: const Center(child: Text('Hola món!')),
      ),
    );
  }
}
```



En enganxar el codi, la indentació podria no quedar alineada.

Per formatar el codi a Visual Studio Code, feu servir Maj+Alt+F (Windows i Linux) o Maj+Option+F (Mac). Per formatar automàticament el codi a Android Studio, podeu prémer Cmd+Alt+L (a Mac) o Ctrl+Alt+L (a Windows i Linux).

Executa l'app. S'hauria de veure una cosa així:





La primera vegada que utilitzeu un dispositiu físic, pot trigar una estona a carregar-vos. Després, pots utilitzar *Hot Reload* per a actualitzacions ràpides. Guardant el nostre codi també es realitza un *hot reload* si l'aplicació s'està executant.

### Observacions

Aquest exemple crea una aplicació utilitza la llibreria `material`, que conté widgets que utilitzen [Material Design](#). Material Design és un llenguatge de disseny visual de Google que és estàndard a mòbil i web. Flutter ofereix un ampli conjunt de widgets de Material Design.

La classe de l'aplicació (`MyApp`) estén de la classe `StatelessWidget` cosa que fa que l'aplicació sigui un widget. A Flutter, gairebé tot és un widget, fins i tot `Alignment`, `Padding` o `Layout`.

El widget `Scaffold`, de la llibreria `material`, proporciona una barra d'aplicacions predeterminada, un títol i una propietat `body` que conté l'arbre del widget per a la pantalla d'inici. A l'exemple només tenim un text, però aquest arbre pot arribar a ser molt complex.

El treball principal d'un widget és proporcionar un mètode `build` que defineix com es mostra el widget en termes d'altres widgets de nivell inferior.

El `body` d'aquest exemple consisteix en un widget `Center` que conté un widget `Text` com `child`. El widget `Center` alinea el seu `child` al centre de la pantalla.

## UTILITZACIÓ DE PAQUETS EXTERNS

En aquest pas, utilitzaràs un paquet de codi obert anomenat `random_name_generator`, que serveix per generar a l'atzar noms de persones.

Pots trobar el paquet `random_name_generator`, així com molts altres paquets de codi obert, a <https://pub.dev/>.

Afegirem el paquet `random_name_generator` com a dependència d'aquesta aplicació. Per fer-ho, editarem l'arxiu `pubspec.yaml`, que és l'arxiu de configuració de l'aplicació, afegint aquesta línia a l'apartat `dependencies`:

```
random_name_generator: ^1.4.0
```

Per comprovar quina és la darrera versió d'una llibreria, podem anar a la pàgina de la llibreria al repositori *pub.dev*, en aquest cas

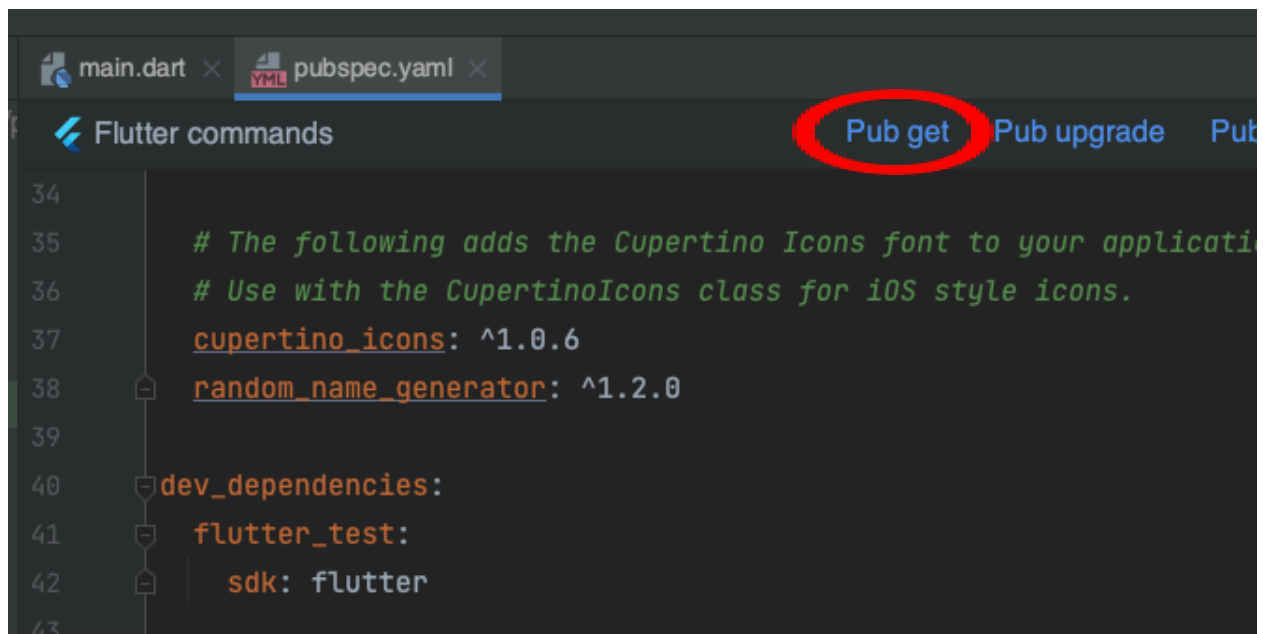
[https://pub.dev/packages/random\\_name\\_generator](https://pub.dev/packages/random_name_generator) i allà podrem clicar a la icona de copiar per copiar el text a inserir a l'arxiu *pubspec.yaml*.

# random\_name\_generator 1.4.0

Published 10 days ago •  [poquesoft.net](https://poquesoft.net) Dart 3 compatible

SDK | DART | FLUTTER | PLATFORM | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

Per descarregar la llibreria per poder utilitzar-la, cliquem a l'enllaç *pub get* que apareix a Android Studio.



Ara, a *lib/main.dart* afegim una línia als imports:

```
import 'package:flutter/material.dart';
import 'package:random_name_generator/random_name_generator.dart';
```

Observa com a mesura que escrius, se t'ofereixen suggeriments per importar llibreries. A continuació, mostrarà la línia de codi d'importació en gris i, passant el

cursor per sobre, us informarà que la llibreria importada no s'utilitza (de moment).


Ara utilitzarem el paquet `random_name_generator` per generar un nom aleatori on ara diu "Hola món".

Farem els següents canvis a la nostra app:

```
@override
Widget build(BuildContext context) {
  final randomNames = RandomNames(Zone.us); // Afegeix aquesta línia
  return MaterialApp(
    title: 'Flutter Demo',
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Benvinguts a Flutter'),
      ),
      body: Center(child: Text(randomNames.fullName())), // Modifica el body
    ),
  );
}
```



Si veieu que l'avaluació del widget `Text` dona un error, probablement us heu oblidat d'eliminar `const` del widget `Center`.

Si l'aplicació s'està executant, podeu forçar un *hot reload*  per actualitzar-la. Cada vegada que feu clic a *hot reload* o guardis el projecte, hauries de veure un nom diferent, escollit a l'atzar, a l'aplicació en execució. Això és degut a que la generació de noms es genera dins del mètode `build`, que s'executa cada vegada que es construeix el widget.

## AFEGIR UN WIDGET AMB ESTAT

Els `StatelessWidget` són immutables, cosa que significa que les seves propietats no poden canviar; tots els valors són finals.

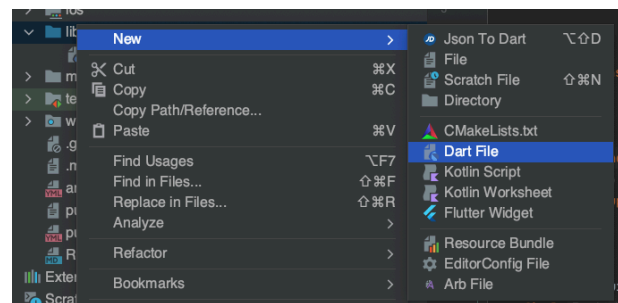
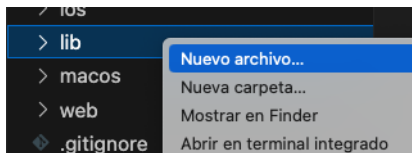
En canvi, els `StatefulWidget` mantenen un estat que pot canviar durant la vida útil del widget. La implementació d'un widget amb estat requereix almenys dues classes, una classe `StatefulWidget` que conté una instància d'una classe `State`.



L'objecte `StatefulWidget` és, per si mateix, immutable, però l'objecte `State` evoluciona durant la vida útil del widget.

En aquest pas, afegiràs un `StatefulWidget`, que anomenarem `PeopleList`, que crea la seva classe `State`, que es dirà `_PeopleListState`. A continuació, utilitzaràs `PeopleList` com a `child` dins del widget `MyApp`.

Per mantenir un codi net és convenient tenir cada classe en un fitxer separat. Crearem un nou arxiu `people_list.dart` al directori `lib` on guardarem el nostre widget. Per això cliquem al directori `lib` amb el botó dret i seleccionem `New File` (VSCode) o `New Dart File` (Android Studio):



Observem les *naming conventions* de Dart: mentre que els noms de les classes comencen sempre amb majúscules (`PeopleList`), els noms de fitxer van sempre en minúscula i utilitzant el `_` com a separador de paraules (`people_list.dart`)

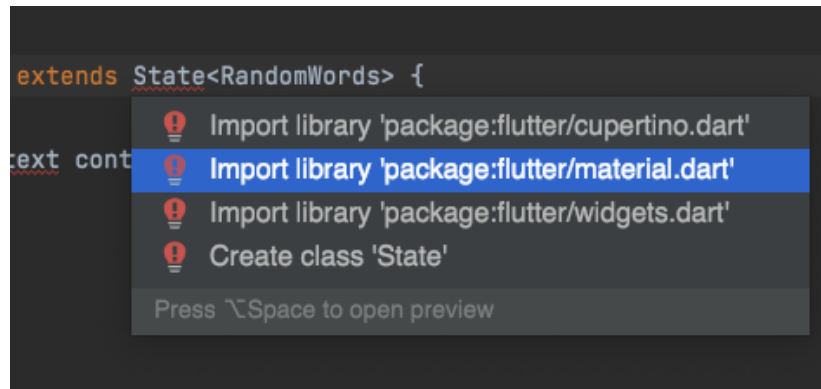
Situem-nos ara al nou arxiu creat, escrivim `stful`. A mesura que anem escrivint, s'ens suggerirà crear un nou `StatefulWidget`. Prement la tecla Return crearà tota l'estructura d'un `StatefulWidget`. Introduïrem ara el nom del widget: `PeopleList`.

Quan hagueu introduït `PeopleList` com a nom del `StatefulWidget`, el nostre IDE actualitza automàticament la classe `State` que l'acompanya, anomenant-la `_PeopleListState`. Per defecte, el nom de la classe `State` té el prefix d'un guió baix. Prefixar un identificador amb un guió baix fa que l'objecte sigui privat (és a dir que no es vegi fora d'aquest fitxer) i és una pràctica recomanada per als objectes `State`.

També s'actualitza automàticament la classe d'estat com `State<PeopleList>`, indicant que utilitzeu una classe `State` genèrica especialitzada per utilitzar-la amb un objecte del tipus `PeopleList`. La major part de la lògica de l'aplicació resideix aquí mantenint l'estat del widget `PeopleList`. Aquesta classe mostrarà una llista de noms

generats, que creixerà infinitament a mesura que l'usuari es desplaça.

És possible que ens marqui alguns errors. Això és perquè falten alguns *imports*. Passant el cursor pels errors i clicant a *More actions...* us apareixeran aquestes opcions, i escollirem importar la llibreria `material.dart`.



Ara les dues classes tenen el següent aspecte:

```
import 'package:flutter/material.dart';

class PeopleList extends StatefulWidget {
  const PeopleList({super.key});

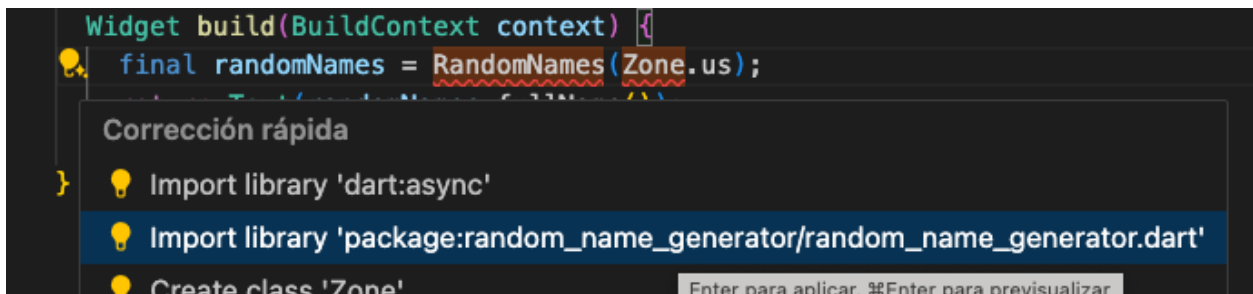
  @override
  State<PeopleList> createState() => _PeopleListState();
}

class _PeopleListState extends State<PeopleList> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

En aquest punt afegirem tot el que volem que aparegui en el mètode `build` de la classe `_PeopleListState`.

```
class _PeopleListState extends State<PeopleList> {
  @override
  Widget build(BuildContext context) {
    final randomNames = RandomNames(Zone.us);
    return Text(randomNames.fullName());
  }
}
```

Fixa't que també hauràs d'afegir un import per la classe `RandomNames`. Això és molt senzill clicant en la bombeta que apareixerà a la dreta del codi si ens posicionem sobre l'error. Ens donarà alternatives de la llibreria que hem d'importar i ho afegirà al codi de forma automàtica:



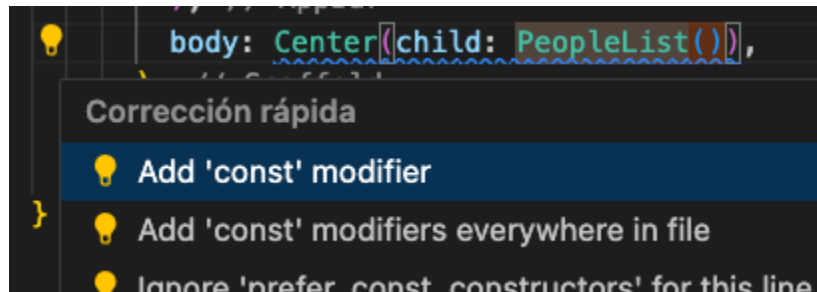
Ara, a `main.dart`, eliminarem el codi de generació de noms de la classe `MyApp` i apuntarem al nou widget que hem creat.

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    final randomNames = RandomNames(Zone.us); // Eliminem aquesta línia
    return MaterialApp(
      title: 'Flutter Demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text(Benvinguts a Flutter'),
        ),
        body: Center(child: PeopleList()), // Modifica el body
      ),
    );
  }
}
```

Revisa els *import*, que s'hauran de modificar (afegir-ne un i eliminar-ne un altre, però és possible que l'IDE l'afegeixi automàticament).

Fixa't que ara s'ens mostra un warning. Posicionant el cursor sobre l'error us suggereix afegir **const** abans del widget **Center**. Això és perquè ara, el contingut del widget **Center** no depèn de cap variable. Accepta el suggeriment fent clic on diu **Add 'const' modifier**.



Torna a fer un *hot reload*. L'aplicació ha de comportar-se com abans, mostrant un nom aleatori de persona cada cop que recarregueu o deseu l'aplicació.

## CREAR UN LISTVIEW DE DESPLAÇAMENT INFINIT

En aquest pas, s'ampliarà **\_PeopleListState** per generar i mostrar una llista de noms de persones. A mesura que l'usuari es desplaça, la llista (que es mostra en un widget **ListView**) creix infinitament. El constructor **builder** de **ListView** us permetrà crear una vista de llista a demanda.

Afegirem algunes variables d'estat a la classe **\_PeopleListState**.

Afegeix una llista **\_suggestions** per desar les combinacions de paraules suggerides. A més, afegeix una variable **\_biggerFont** per augmentar la mida del tipus de lletra. També creem una variable per al generador de noms **\_randomNames**.

```
class _PeopleListState extends State<PeopleList> {
  final _suggestions = <String>[]; // Nou
  final _biggerFont = const TextStyle(fontSize: 18); // Nou
  final _randomNames = RandomNames(Zone.us); // Nou
  ...
}
```

A continuació, actualitzarem el mètode `build` de la classe `_PeopleListState`.

El widget `ListView` ens mostrarà una llista. Aquest widget té un constructor `ListView.builder` que ens facilita molt les coses a l'hora d'implementar els elements de la llista.

Al constructor `ListView.builder` li passem el paràmetre `itemBuilder`, que és una funció *callback* que definim a partir de dos paràmetres: el `BuildContext` actual (que es refereix a la pantalla on estem) i un valor `i` que és l'índex de l'element que estem pintant. L'índex comença a zero i augmenta cada cop que es crida la funció, una vegada per nom de persona generat. Aquest model permet que la llista de suggeriments segueixi creixent a mesura que l'usuari es desplaça.

```
@override
Widget build(BuildContext context) {
  return ListView.builder(
    itemBuilder: (context, i) {
      if (i >= _suggestions.length) {
        //Afegim 10 noms més
        for (var i = 0; i < 10; i++) {
          _suggestions.add(_randomNames.fullName());
        }
      }
      return ListTile(
        title: Text(
          _suggestions[i],
          style: _biggerFont,
        ),
      );
    },
  );
}
```

La funció `itemBuilder` es crida una vegada per cada element del `ListView` i col·loca per cada nom un widget `ListTile` que conté el nom generat.



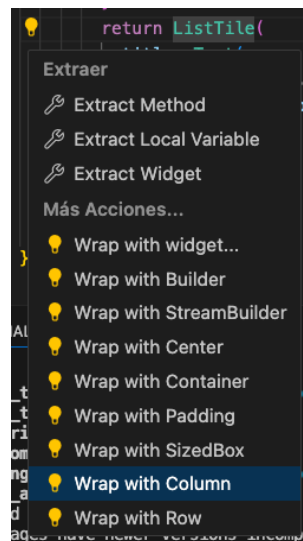
El codi a l'inici de la funció `itemBuilder` serveix per generar dinàmicament més noms aleatoris si l'índex de l'element que es mostra a la pantalla és més gran que la llista d'elements generats, és a

dir, si encara no s'ha generat aquest element.

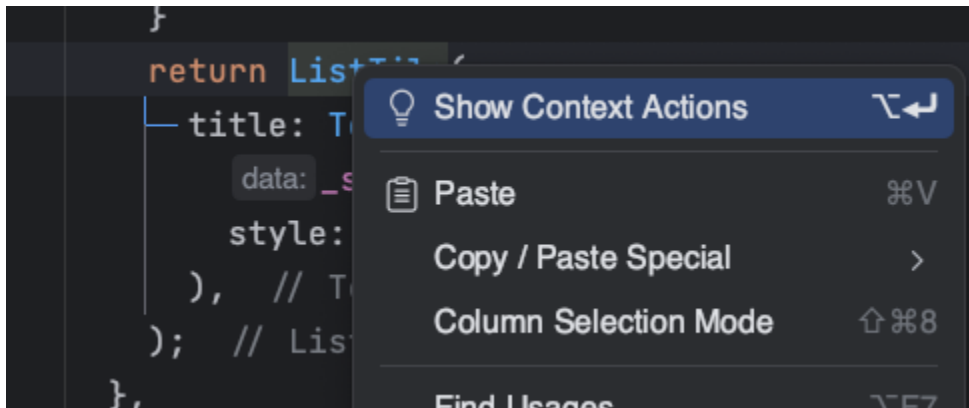
Ara afegirem un separador entre els elements. Per fer això, embolicarem el widget `ListTile` en una columna (un widget `Column`) d'aquesta manera:

```
return Column(
  children: [
    ListTile(
      title: Text(
        _suggestions[i],
        style: _biggerFont,
      ),
    ),
  ],
);
```

Aquest pas (embolicar en un altre widget) el podem fer de forma ràpida fent servir altre cop la bombeta:



En Android Studio, també podem utilitzar el menú contextual (botó dret del ratolí). Si el polsem ens apareixerà això:



Seleccionant **Show Context Actions**, triarem l'opció **Wrap with Column**:



És important dominar aquesta manera de treballar ja que ens agilitzarà molt el procés d'afegir i eliminar widgets dins de l'arbre.

Ara tenim una columna amb només un element per a cada nom. Afegirem sota el widget `ListTile` un widget `Divider`:

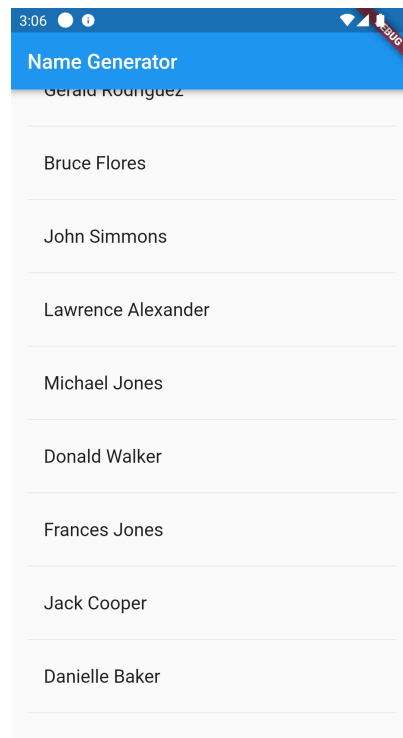
```
return Column(
  children: [
    ListTile(
      title: Text(
        data: _suggestions[i],
        style: _biggerFont,
      ), // Text
    ), // ListTile
    const Divider(),
  ],
); // Column
```

Ja només ens queda actualitzar `MyApp` per canviar el títol en dos llocs:

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Generador de noms',
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blue,
          foregroundColor: Colors.white,
          title: const Text('Generador de noms'),
        ),
        body: const Center(child: PeopleList()),
      ),
    );
  }
}
```

Reinicieu l'aplicació. Hauries de veure una llista de noms generats a l'atzar, independentment de la distància que es desplaci.





## TASQUES ADDICIONALS (OPCIONAL)

A partir de l'app que has realitzat, et proposo fer modificacions com mirar la documentació de la llibreria per generar noms d'unes altres zones, personalitzar l'aspecte de l'app canviant colors o el disseny de pantalla, ampliar l'app afegint més widgets a la pantalla o afegir alguna llibreria addicional a la tasca.

## REFERÈNCIES

1. Google Developer CodeLabs:  
<https://codelabs.developers.google.com/codelabs/first-flutter-app-pt1>
2. Paquet `random_name_generator`:  
[https://pub.dev/packages/random\\_name\\_generator](https://pub.dev/packages/random_name_generator)