Flutter Fundamentals
Short recap day #1

Peter Kassenaar –
info@kassenaar.com

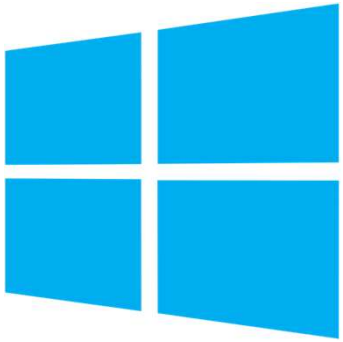
Agenda - details

- Introduction – overview of the Flutter landscape
- Flutter tooling – installation
- Hello World –the structure and architecture of Flutter apps.
- Zooming in on Flutter:
  - Components – Stateless vs. Stateful

Agenda – cont'd

- Widgets - introduction
- The Flutter layout system – `Scaffold()` and more
- Using Images and assets
- More layout widgets
  - `Button()`, `Icon()`, `Container()`, `Padding()`
  - `Row()`, `Column()`, properties, children and more
- Working with data
  - `ListView()`, `Card()`, Designing layouts

"Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase."


Installation – recommended order

1. Install Visual Studio
2. Install IntelliJ
3. Install Flutter plug-in for IntelliJ
4. Install Flutter SDK
5. Update Windows PATH variable
6. Run `flutter doctor`, fix any possible problems

Default code – recognize the structure

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
…
```

(Yours might be slightly different, due to updates)

Study the default code. It has useful comments.

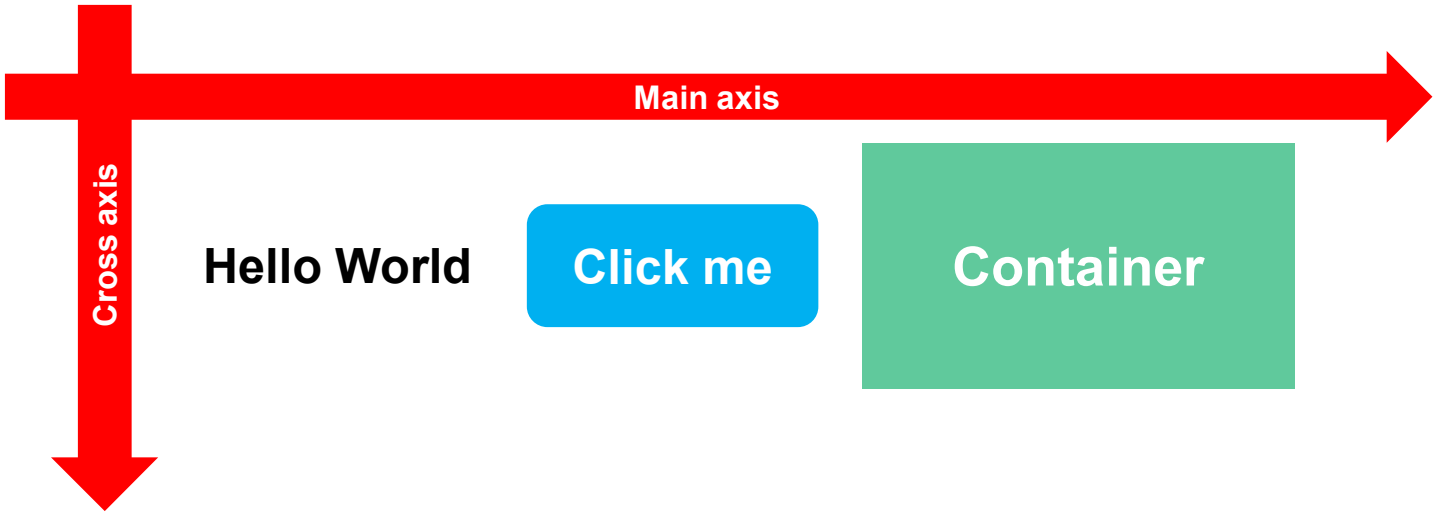
Flutter == Dart in action

- Important widgets
  - Scaffold()
  - AppBar()
  - Themes, fonts & colors
  - Image()
  - Icon()
  - Row(), Column()
  - ListView()
  - Container()
  - Expanded()

"Every Flutter App is composed as a tree of widgets"

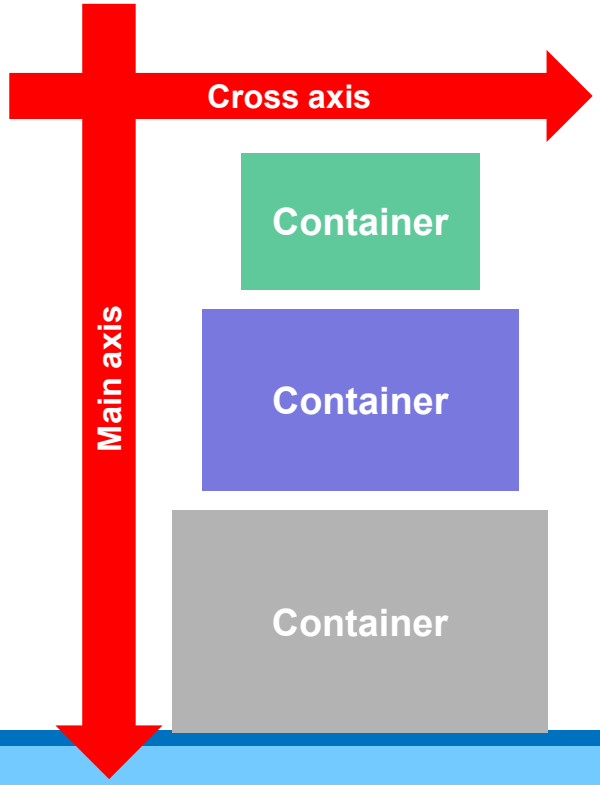
Alignment in Rows/Columns

- In Rows:
  - Use `MainAxisAlignment` for horizontal layout
  - Use `CrossAxisAlignment` for vertical layout

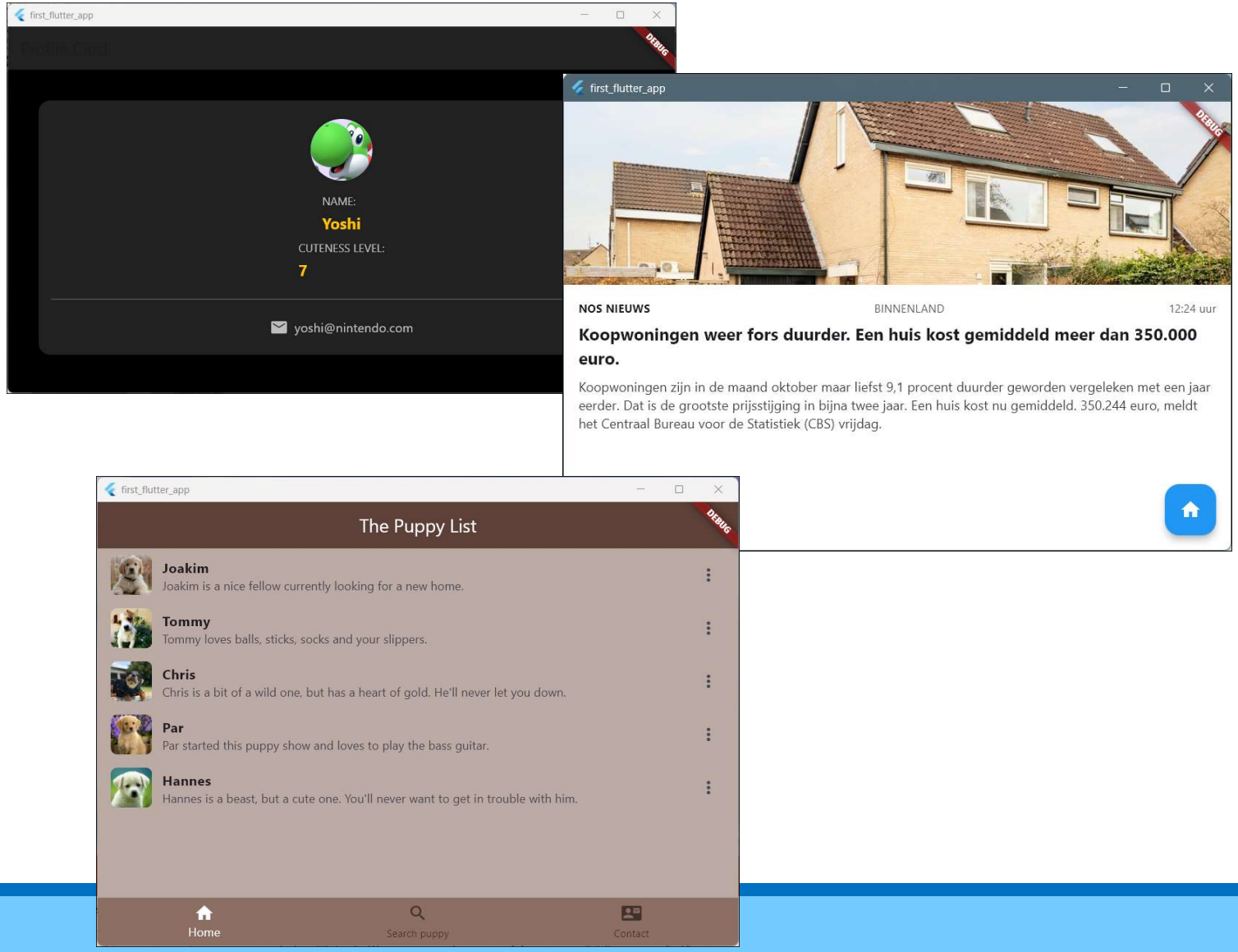


Alignment

- In Columns – opposite to Rows:
  - Use `MainAxisAlignment` for vertical layout
  - Use `CrossAxisAlignment` for horizontal layout

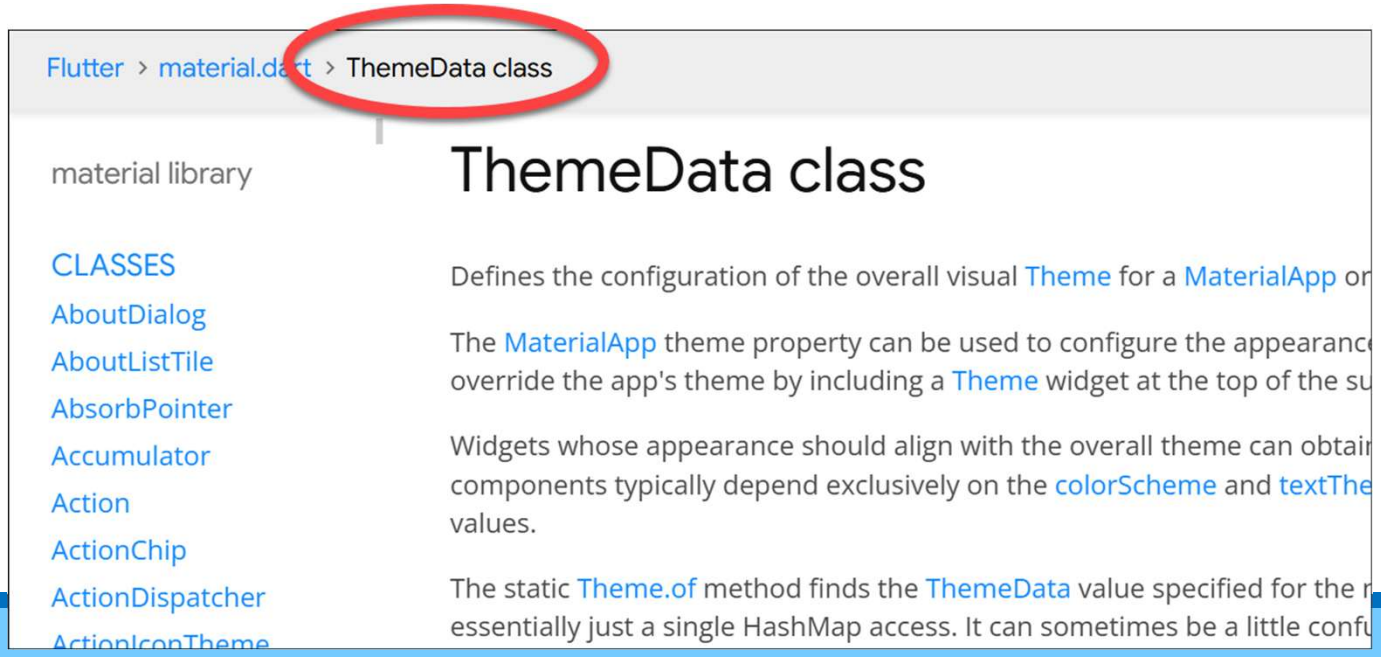


Creating various layouts by combining Widgets



Colors in Theme?

- Instead of defining separate colors and `TextStyle()` properties in a file, we can leverage the Flutter `ThemeData()` class
- https://api.flutter.dev/flutter/material/ThemeData-class.html



Create `ThemeData()` in a separate file

- Create a file (for instance: `theme.dart`) with the following content:

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

// Create a theme in a separate file.
// NOTE: This file DOES NOT have
// a user interface. It just returns a theme.
ThemeData buildThemeData() {
  return ThemeData(
    // 1. Define the main color scheme
    primaryColor: Colors.green[800],
    hintColor: Colors.green[800],

    // 2. Set the default font family
    fontFamily: 'BungeeOutline',

    // …more Theme Styles, as needed.
  );
}
```

Consuming your theme

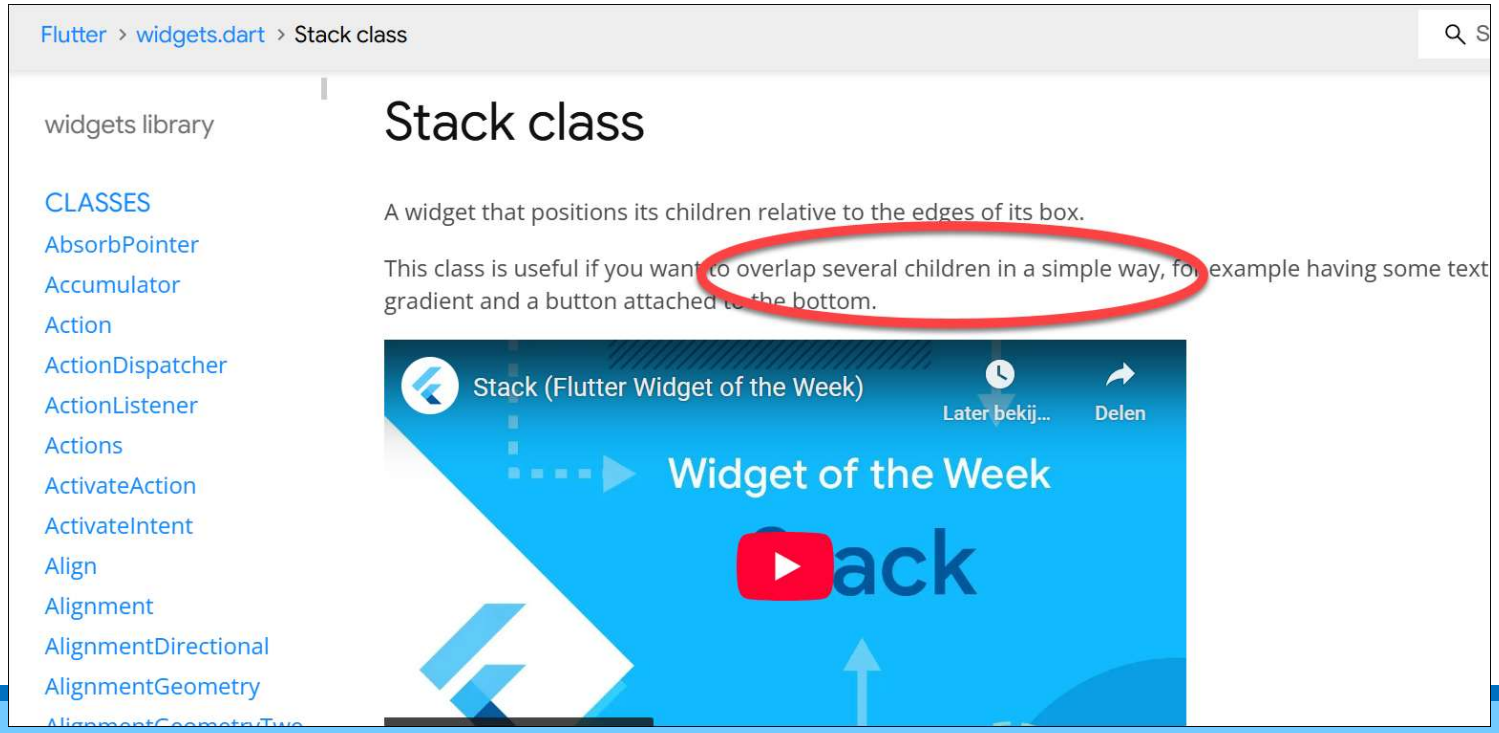
```
import 'package:flutter/material.dart';

// 1. Import the (external) theme.
import 'theme.dart';

void main() {
  // 2. Build custom theme data from theme.dart
  final theme = buildThemeData();

  runApp(
    MaterialApp(
      theme: theme, // 3. Use the theme here!
      home: Scaffold(
        appBar: AppBar(
          title: new Text('Hello Flutter'), // 4. rest of formatting comes from theme.
        ),
        body: Center(
          child: Text(
            'This is my Flutter app',
            style: theme.textTheme.bodyMedium // 5. Access textTheme directly.
          ),
    // …
  );
}
```

Benefits of using `ThemeData()`

- Reusability
  - A centralized theme ensures consistent usage across the app.
- Scalability
  - If your app grows, the design system already supports easy adjustments through the theme.
- Readability
  - Removes clutter from the UI definitions by eliminating repetitive styles.

Background images

- Use a `Stack()` widget to literally stack widgets on top of each other.
- https://api.flutter.dev/flutter/widgets/Stack-class.html

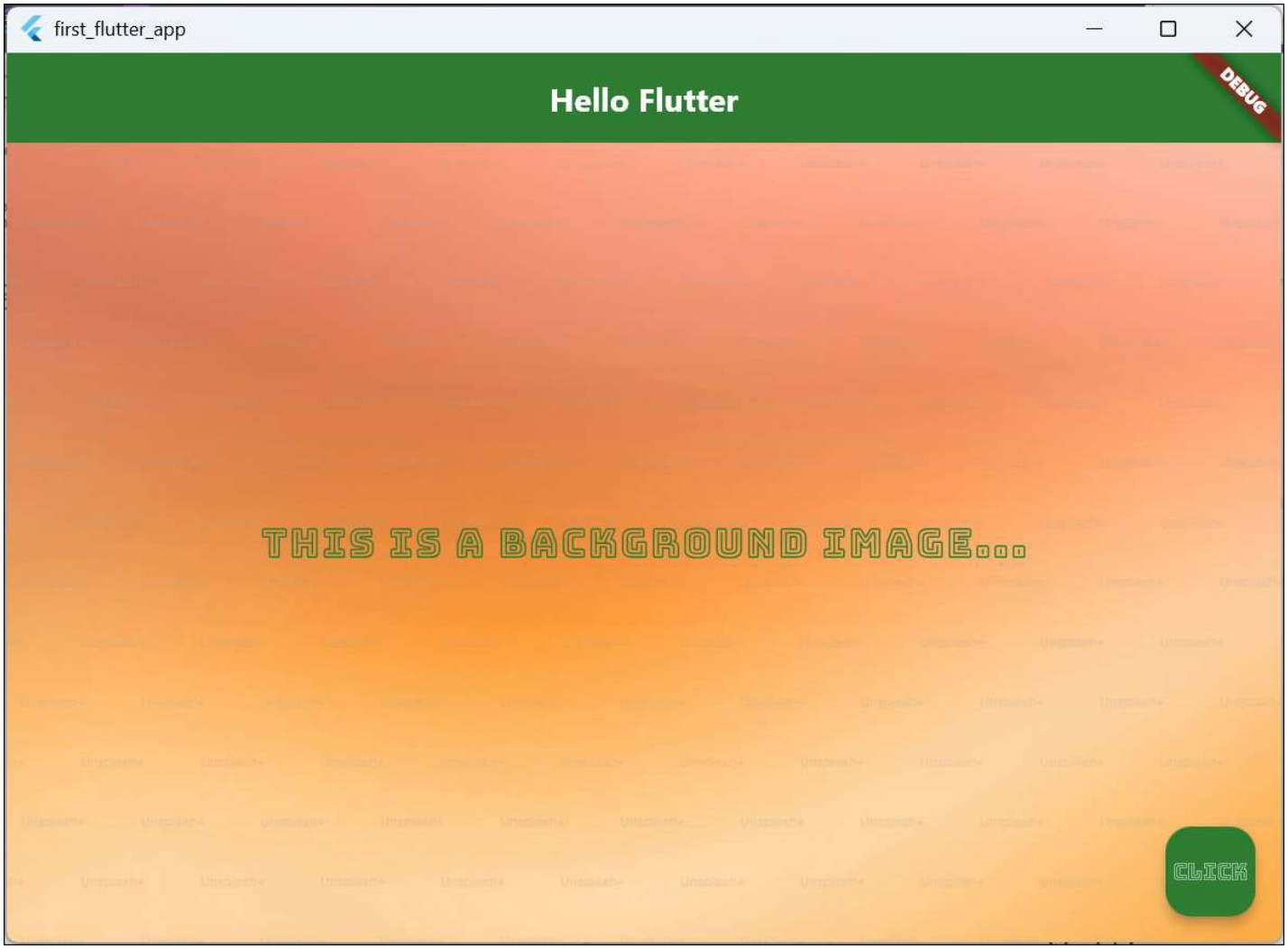


For instance:

- A `Container()` with a background image
  - Remember, if a `Container()` has no children, it fills the complete space.
  - In this case, there is only a decoration: `BoxDecoration()` property, no children!

```
body: Stack(
  children: [
    Container(
      // 2. Set the background image here
      decoration: BoxDecoration(
        image: DecorationImage(
          image: AssetImage('assets/background.jpg'),
          fit: BoxFit.cover, // Covers the entire screen area
        ),
      ),
    ),
    // …more children….
  ]
…
```

Result:


first_flutter_app
Hello Flutter
DEBUG
THIS IS A BACKGROUND IMAGE...
CLICK

other

Questions?

Today:

- State management in various ways
  - Stateful widgets – using models/custom classes
  - passing parameters, passing functions
- Communicating with external API's
  - http / other methods
- More state management
  - bloc pattern, bloc + cubit implementation
  - payload – emit multiple events

Tomorrow

- TextFields
- Routing / Navigation
- Complete applications
- gRPC
- Gestures
- Publication (executables/packages)
- Evals & goodbye
- …