ChatGPT

1.
Differentiate “runApp( )” and “main( )” functions. Also explain, when to
use material.dart package.
2.

List any four category of widgets and explain any two out of it. 3.
With appropriate example and required code snippet, explain input and
layout widgets.
4.
List at least four material widgets and explain any one with necessary
code snippet.
5.
List at least four layout widgets and explain any one with necessary code
snippet.
6.
Develop a Dart class including “Padding” and “BoxDecoration” of

“Container” widget. Discuss your code snippet.

ANSWER DESCRIPTIVELY WITH FULL FLUTTER CODE

1. Differentiate between `runApp()` and `main()` functions. Also, explain when to use the `material.dart` package.

`main()` Function:

- The `main()` function is the entry point of any Dart or Flutter program. It is where execution begins when a Flutter app is launched.
- In Flutter, the `main()` function calls the `runApp()` function to start the app's execution.
- Example:

```dart
void main() {
  runApp(MyApp());  // Runs the application
}
```

`runApp()` Function:

- `runApp()` takes a `Widget` (usually the root widget of your app) as a parameter and makes it the root of the widget tree.
- It is responsible for initializing the Flutter app and starting the rendering of the widget tree.

- Example:

```dart
void runApp(Widget app) {
  // Initializes the widget tree
}
```

`material.dart` Package:

- The `material.dart` package is used in Flutter to build apps that follow the Material Design principles (a design system created by Google).
- You should use this package when you need to build apps that look and feel like modern Android apps with a rich set of UI components such as buttons, text fields, icons, and more.
- This package provides Material-themed widgets like `Scaffold`, `AppBar`, `FloatingActionButton`, and others.
- Example:

```dart
import 'package:flutter/material.dart';  // Import material.dart for Material widgets

void main() {
  runApp(MaterialApp(
    home: Scaffold(
      appBar: AppBar(title: Text('Material Design App')),
      body: Center(child: Text('Hello, Flutter!')),
    ),
  ));
}
```

2. List any four categories of widgets and explain any two out of them.

Flutter widgets are categorized into several groups. Here are four common categories:

1. **Structural Widgets**: These widgets deal with layout and structure.
2. **Functional Widgets**: These widgets are used to add functionality like buttons and user input.
3. **Material Design Widgets**: These widgets implement Material Design components, like `Scaffold`, `AppBar`, etc.
4. **Cupertino Widgets**: These are used to create iOS-style widgets.

Let's explain **Structural Widgets** and **Functional Widgets**.

Structural Widgets:

- Structural widgets are responsible for the overall layout of the UI. They don't necessarily look like traditional UI components but instead organize other widgets.
- Example: `Column`, `Row`, `Stack`, and `Container`.
  - `Column`: Arranges child widgets vertically.
  - `Row`: Arranges child widgets horizontally.

Functional Widgets:

- Functional widgets add specific functionality like handling user input, gestures, and interactions.

- Example: `TextField`, `Button`, `IconButton`, and `GestureDetector`.
  - `TextField`: A widget for taking user input in the form of text.
  - `Button`: A clickable button widget.

3. Input and Layout Widgets

Input Widgets:

- Input widgets allow users to enter data into the application, such as text or selection choices.

Example of `TextField` (input widget):

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Input Widget Example')),
        body: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            children: [
              Text('Enter your name:'),
              TextField(  // Input widget for text entry
                decoration: InputDecoration(
                  border: OutlineInputBorder(),
                  labelText: 'Name',
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Layout Widgets:

- Layout widgets help arrange and position other widgets within a given space.

Example of `Column` (layout widget):

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Layout Widget Example')),
        body: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(  // Layout widget to arrange widgets vertically
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text('First Name:'),
              TextField(),
              SizedBox(height: 20),  // Spacer between fields
              Text('Last Name:'),
              TextField(),
            ],
          ),
        ),
      ),
    );
  }
}
```

4. List at least four material widgets and explain any one with necessary code snippet.

Four Material Widgets:

1. `Scaffold`
2. `AppBar`
3. `FloatingActionButton`
4. `RaisedButton`

Explanation of `Scaffold`:

- `Scaffold` provides a basic structure for visual layout like AppBar, body, floating action button, and more.
- It's typically the root widget of an app or page.

Code Example:

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

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

  class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
      return MaterialApp(
        home: Scaffold(  // Scaffold widget to hold layout
          appBar: AppBar(
            title: Text('Scaffold Example'),
          ),
          body: Center(
            child: Text('Hello, Flutter!'),
          ),
          floatingActionButton: FloatingActionButton(
            onPressed: () {
              print('FAB pressed!');
            },
            child: Icon(Icons.add),
          ),
        ),
      );
    }
  }
```

5. List at least four layout widgets and explain any one with necessary code snippet.

Four Layout Widgets:

1. `Container`
2. `Column`
3. `Row`
4. `Stack`

Explanation of `Container`:

- `Container` is a versatile widget used for creating a box with decoration (padding, margins, background color, etc.).
- It can also be used to control dimensions like width, height, and alignment.

Code Example:

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

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

  class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
      return MaterialApp(
        home: Scaffold(
          appBar: AppBar(title: Text('Container Example')),
          body: Center(
            child: Container(  // Container used for decoration and layout
              padding: EdgeInsets.all(20),
              decoration: BoxDecoration(
                color: Colors.blueAccent,
                borderRadius: BorderRadius.circular(10),
                boxShadow: [
                  BoxShadow(color: Colors.black, blurRadius: 5)
                ],
              ),
              child: Text(
                'Hello, Flutter!',
                style: TextStyle(fontSize: 24, color: Colors.white),
              ),
            ),
          ),
        ),
      );
    }
  }
```

6. Develop a Dart class including "Padding" and "BoxDecoration" of "Container" widget. Discuss your code snippet.

Explanation: In this code, we create a `Container` widget that includes padding and uses `BoxDecoration` for adding decoration features like background color, border radius, and a shadow effect.

Code Example:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Padding and BoxDecoration Example')),
        body: Center(
          child: Container(  // Container widget with decoration and padding
            padding: EdgeInsets.all(16.0),  // Adds padding around the container
            decoration: BoxDecoration(
              color: Colors.orangeAccent,  // Background color
              borderRadius: BorderRadius.circular(12),  // Rounded corners
              boxShadow: [
                BoxShadow(
                  color: Colors.black26,  // Shadow color
                  blurRadius: 10,  // Shadow blur radius
                  offset: Offset(0, 4),  // Shadow offset
                ),
              ],
            ),
            child: Text(
              'Decorated Container!',
              style: TextStyle(fontSize: 20, color: Colors.white),
            ),
          ),
        ),
      ),
    );
  }
}
```

Explanation of Code:

- The `Container` has `padding` of 16.0 units to give space inside the container.
- The `BoxDecoration` adds a background color (`orangeAccent`), a border radius for rounded corners, and a box shadow for a 3D effect.
- The `Text` widget inside the container is styled with a white color and larger font size.

7.

Create a flutter app demonstrating use of Row and Column widgets.
Explain your solution.
8.
How to create flutter project from terminal? Write command to add dependency in a flutter project. Explain using example of google_fonts.

9.
Explain AppBar and SnackBar of Scaffold with appropriate example and code snippet.
10. Explain BottomNavigationBar of Scaffold with appropriate example and code snippet.
11. Create a flutter app demonstrating use of Stack and Positioned widgets. Explain your solution.

7. Flutter App Demonstrating Use of Row and Column Widgets

Explanation: The `Row` and `Column` widgets are used to arrange child widgets horizontally and vertically, respectively. In this example, we'll use both widgets in a single app, with `Row` arranging widgets horizontally and `Column` stacking widgets vertically inside the body of the `Scaffold`.

Code Example:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Row and Column Example'),
        ),
        body: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Column(  // Column arranges widgets vertically
            children: [
              Row(  // Row arranges widgets horizontally
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Icon(Icons.access_alarm, size: 50, color: Colors.blue),
                  SizedBox(width: 20),  // Adds space between the icons
                  Icon(Icons.account_box, size: 50, color: Colors.green),
                ],
              ),
              SizedBox(height: 20),  // Adds space between Row and Column
              Text('This is a Row inside a Column!', style: TextStyle(fontSize: 20)),
              SizedBox(height: 20),
              Container(
                color: Colors.orange,
                width: double.infinity,
                height: 100,
                child: Center(child: Text('Column Widget Below', style: TextStyle(color: Col
ors.white, fontSize: 18))),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Explanation of Solution:

- `` `Column` `` is used to arrange the `` `Row` `` and the `` `Text` `` widget vertically.
- `` `Row` `` is used to display two icons (`` `access_alarm` `` and `` `account_box` ``) horizontally. The `` `SizedBox` `` widget is used to add some spacing between the icons.
- `` `SizedBox` `` is used for spacing between different sections (like between the `` `Row` `` and `` `Column` ``).
- Inside the `` `Column` ``, a `` `Container` `` is used with some padding and background color to add a visually distinct section.

---

8. How to Create a Flutter Project from Terminal and Add Dependencies (Using `` `google_fonts` `` as Example)

Steps to Create a Flutter Project:

1. Open your terminal or command prompt.
2. Run the following command to create a new Flutter project:

```bash
flutter create my_app
```

   This command will create a directory `` `my_app` `` with the default Flutter project structure.

3. After the project is created, navigate into the project directory:

```bash
cd my_app
```

Steps to Add Dependency (google_fonts Example):

1. Open the `` `pubspec.yaml` `` file in your project directory.
2. Under the `` `dependencies` `` section, add the `` `google_fonts` `` package:

```yaml
dependencies:
  flutter:
    sdk: flutter
  google_fonts: ^4.0.0  # Example version
```

3. Save the file and run the following command in the terminal to install the dependency:

```bash
flutter pub get
```

Example of Using `google_fonts`: Here is how to use the `google_fonts` package to apply custom fonts in your Flutter app.

```dart
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';  // Import the google_fonts package

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Google Fonts Example')),
        body: Center(
          child: Text(
            'Hello, Flutter!',
            style: GoogleFonts.lato(fontSize: 30),  // Applying Google font 'Lato'
          ),
        ),
      ),
    );
  }
}
```

---

9. Explanation of AppBar and SnackBar of Scaffold with Code Snippet

Explanation of `AppBar`:

- The `AppBar` widget is placed inside the `Scaffold` to provide a top bar with a title and other actions like buttons or menus.

Explanation of `SnackBar`:

- The `SnackBar` widget provides temporary messages that appear at the bottom of the screen. It's often used for showing success, error, or informational messages.

Code Example:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('AppBar and SnackBar Example'),
        ),
        body: Center(
          child: ElevatedButton(
            onPressed: () {
              ScaffoldMessenger.of(context).showSnackBar(  // Showing SnackBar
                SnackBar(
                  content: Text('This is a SnackBar!'),
                  action: SnackBarAction(
                    label: 'Undo',
                    onPressed: () {
                      print('Undo pressed!');
                    },
                  ),
                ),
              );
            },
            child: Text('Show SnackBar'),
          ),
        ),
      ),
    );
  }
}
```

Explanation:

- The `AppBar` is used to display the title of the app at the top of the screen.
- When the button is pressed, a `SnackBar` is shown at the bottom of the screen using `ScaffoldMessenger.of(context).showSnackBar()`.
- The `SnackBar` can have an action, in this case, an "Undo" action.

---

10. Explanation of BottomNavigationBar of Scaffold with Code Snippet

Explanation of `BottomNavigationBar`:

- The `BottomNavigationBar` is typically used for navigation purposes at the bottom of the screen. It displays icons and allows users to switch between different views or sections of the app.

Code Example:

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

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

  class MyApp extends StatefulWidget {
    @override
    _MyAppState createState() => _MyAppState();
  }

  class _MyAppState extends State<MyApp> {
    int _selectedIndex = 0;  // To track the selected tab

    void _onItemTapped(int index) {
      setState(() {
        _selectedIndex = index;  // Update the selected index
      });
    }

    @override
    Widget build(BuildContext context) {
      return MaterialApp(
        home: Scaffold(
          appBar: AppBar(title: Text('Bottom Navigation Example')),
          body: Center(
            child: Text('Selected index: $_selectedIndex', style: TextStyle(fontSize: 24)),
          ),
          bottomNavigationBar: BottomNavigationBar(
            currentIndex: _selectedIndex,  // Set the currently selected tab
            onTap: _onItemTapped,  // Handle tab selection
            items: [
              BottomNavigationBarItem(
                icon: Icon(Icons.home),
                label: 'Home',
              ),
              BottomNavigationBarItem(
                icon: Icon(Icons.search),
                label: 'Search',
              ),
              BottomNavigationBarItem(
                icon: Icon(Icons.notifications),
                label: 'Notifications',
              ),
            ],
          ),
        ),
      );
    }
  }
```

Explanation:

- The `BottomNavigationBar` displays three items: "Home", "Search", and "Notifications".
- When a user taps on any of these items, the selected index is updated, and the corresponding index is displayed on the screen.

---

11. Flutter App Demonstrating Use of Stack and Positioned Widgets

Explanation: The `Stack` widget allows you to overlay multiple widgets on top of each other. The `Positioned` widget is used inside the `Stack` to control the position of its child relative to the stack's edges.

Code Example:

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

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

  class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
      return MaterialApp(
        home: Scaffold(
          appBar: AppBar(title: Text('Stack and Positioned Example')),
          body: Center(
            child: Stack(  // Stack widget for overlaying widgets
              children: [
                Container(
                  width: 300,
                  height: 300,
                  color: Colors.blue[200],
                ),
                Positioned(
                  top: 50,  // Position the widget 50 units from the top
                  left: 50,  // Position the widget 50 units from the left
                  child: Container(
                    width: 100,
                    height: 100,
                    color: Colors.red,
                  ),
                ),
                Positioned(
                  bottom: 50,  // Position widget 50 units from the bottom
                  right: 50,   // Position widget 50 units from the right
                  child: Container(
                    width: 100,
                    height: 100,
                    color: Colors.green,
                  ),
                ),
              ],
            ),
          ),
        ),
      );
    }
  }
```

Explanation of Solution:

- The `Stack` widget allows multiple child widgets to be layered on top of each other.
- Inside the `Stack`, two `Positioned` widgets are used to place `Container` widgets at specific positions relative to the stack's edges.
- The red `Container` is placed 50 units from the top and left, and the green `Container` is placed 50 units from the bottom and right.

12.Design a flutter app to display author avatar like given below:
(It is the square box with circular image of author)

Write necessary code snippet and explain your solution.
13. With respect to Row widget, write at least four different values of MainAxisAlignment. Draw the diagrams and explain meaning of any two MainAxisAlignment values.
14. Design a flutter app demonstrating use of Wrap and Chip widgets. Explain your solution.
16. What is the use of ListView widget? Explain any two constructors of ListView.
18. List any five key parameters of GridView and explain any two parameters.
19. How to add google_fonts package in flutter project from terminal? Write the code it will add in pubspec.yaml.

12. Flutter App to Display Author Avatar (Square Box with Circular Image)

Explanation: To create a square box with a circular image of the author, you can use the `Container` widget to create the square shape and the `ClipOval` widget to create a circular image. The `ClipOval` widget clips the image into a circle shape, and the `Container` provides the outer square box for the avatar.

Code Example:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Author Avatar Example')),
        body: Center(
          child: Container(
            width: 150,  // Width of the square box
            height: 150, // Height of the square box
            decoration: BoxDecoration(
              color: Colors.blueGrey,  // Color of the square box
              borderRadius: BorderRadius.circular(20), // Rounded corners for the square
            ),
            child: ClipOval(  // ClipOval to create a circular image
              child: Image.network(
                'https://www.example.com/author_avatar.jpg',  // URL of the author's avatar
                fit: BoxFit.cover,  // Fill the circle with the image
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

Explanation:

- The `Container` widget is used to define the size of the square box (150x150).
- The `ClipOval` widget is used to clip the image into a circle shape.
- The `Image.network` widget loads the image from a given URL and the `BoxFit.cover` ensures the image fills the circular area.

---

13. Row Widget: MainAxisAlignment Values

Explanation of `MainAxisAlignment`: `MainAxisAlignment` controls the alignment of children along the main axis of the `Row` widget (horizontally). Here are four different values of `MainAxisAlignment`:

1. **MainAxisAlignment.start**: Aligns children to the start (left) of the main axis.
2. **MainAxisAlignment.end**: Aligns children to the end (right) of the main axis.
3. **MainAxisAlignment.center**: Centers the children along the main axis.
4. **MainAxisAlignment.spaceBetween**: Places free space between the children, leaving no space at the start or end.

Example Code with `MainAxisAlignment` Values:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('MainAxisAlignment Example')),
        body: Column(
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.start,  // Align items to the start
              children: [
                Icon(Icons.home, size: 50),
                Icon(Icons.search, size: 50),
              ],
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.end,  // Align items to the end
              children: [
                Icon(Icons.home, size: 50),
                Icon(Icons.search, size: 50),
              ],
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,  // Center items
              children: [
                Icon(Icons.home, size: 50),
                Icon(Icons.search, size: 50),
              ],
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,  // Space between items
              children: [
                Icon(Icons.home, size: 50),
                Icon(Icons.search, size: 50),
              ],
            ),
          ],
        ),
      ),
    );
  }
}
```

Explanation of `MainAxisAlignment`:

- `MainAxisAlignment.start`: Aligns all the items to the left of the Row.
- `MainAxisAlignment.end`: Aligns all the items to the right of the Row.
- `MainAxisAlignment.center`: Centers the items in the Row.
- `MainAxisAlignment.spaceBetween`: Places equal space between the items, with no space at the start or end.

---

14. Flutter App Demonstrating the Use of Wrap and Chip Widgets

Explanation: The `Wrap` widget is used to display a series of widgets in a horizontal or vertical direction, wrapping them to the next line when space runs out. The `Chip` widget is a small, interactive element that can represent information like tags or labels.

Code Example:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Wrap and Chip Example')),
        body: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Wrap(  // Wrap widget to arrange chips in multiple lines
            spacing: 8.0,  // Horizontal space between chips
            runSpacing: 4.0,  // Vertical space between lines of chips
            children: [
              Chip(
                label: Text('Flutter'),
                avatar: Icon(Icons.favorite, size: 18),
              ),
              Chip(
                label: Text('Dart'),
                avatar: Icon(Icons.code, size: 18),
              ),
              Chip(
                label: Text('Mobile'),
                avatar: Icon(Icons.phone_android, size: 18),
              ),
              Chip(
                label: Text('Development'),
                avatar: Icon(Icons.build, size: 18),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Explanation:

- The `Wrap` widget is used to arrange `Chip` widgets in a responsive manner, automatically wrapping them to the next line when they overflow.
- `spacing` controls horizontal space between chips, and `runSpacing` controls vertical space between lines of chips.
- Each `Chip` has a `label` and an `avatar` (an icon).

---

16. Use of ListView Widget and Its Constructors

Explanation: The `ListView` widget is used to display a scrollable list of widgets. It is one of the most common widgets for displaying lists in Flutter.

Two Constructors of `ListView`:

1. `ListView.builder`: Builds items on demand as they scroll into view.

- Ideal for displaying large or infinite lists because it only builds the widgets that are currently visible.
- Example:

```dart
ListView.builder(
  itemCount: 100,  // Number of items in the list
  itemBuilder: (context, index) {
    return ListTile(
      title: Text('Item #$index'),
    );
  },
);
```

2. **`ListView.separated`**: Creates a list with separators between the items.
    - It allows you to define the separator widget between each list item.
    - Example:

```dart
ListView.separated(
  itemCount: 100,
  itemBuilder: (context, index) {
    return ListTile(
      title: Text('Item #$index'),
    );
  },
  separatorBuilder: (context, index) {
    return Divider();  // Divider between list items
  },
);
```

---

18. Key Parameters of GridView and Explanation of Two Parameters

Five Key Parameters of `GridView`:

1. **`crossAxisCount`**: The number of columns in the grid.
2. **`childAspectRatio`**: The aspect ratio of the child widgets.
3. **`mainAxisSpacing`**: The space between items along the main axis (vertical for **`GridView.count`**).
4. **`crossAxisSpacing`**: The space between items along the cross axis (horizontal for **`GridView.count`**).
5. **`padding`**: The padding around the entire grid.

Explanation of Two Parameters:

1. **`crossAxisCount`**: Defines the number of columns in the grid layout.
    - Example:

```dart
GridView.count(
  crossAxisCount: 2,  // 2 columns in the grid
);
```

2. `childAspectRatio`: Controls the aspect ratio (width to height) of the child widgets.
   - Example:

     ```dart
     GridView.count(
       crossAxisCount: 2,
       childAspectRatio: 1.0,  // Square grid items
     );
     ```

---

19. How to Add `google_fonts` Package in Flutter Project from Terminal

Steps:

1. Open the terminal and navigate to your Flutter project directory.
2. Run the following command to add the `google_fonts` package:

   ```bash
   flutter pub add google_fonts
   ```

Code to be added in `pubspec.yaml`:

```yaml
dependencies:
  flutter:
    sdk: flutter
  google_fonts: ^4.0.0  # Add this line under dependencies
```

After adding the dependency, run the following command to install the package:

```bash
flutter pub get
```

15.
Analyze the following code snippet and explain each line of it:
// 1
return FutureBuilder(
// 2
future: mockService.getExploreData(), // 3
builder: (context, snapshot) {
// TODO: Add Nested List Views
// 4

if (snapshot.connectionState == ConnectionState.done) {
// 5
final recipes
snapshot.data?.todayRecipes;
// TODO: Replace this with TodayRecipeListView
return Center(
child:
} else {
}
// 6
Container (child: const Text('Show TodayRecipeListView')));
return const Center (child: Circular Progress Indicator());
}); // FutureBuilder
20.
What is Scaffold? Analyze and explain following code:
import 'package: flutter/material.dart';
// 1
class Home extends StatefulWidget {
const Home({Key? key}): super(key: key); @override
HomeState createState() => _HomeState();
class _HomeState extends State<Home> { @override
Widget build (BuildContext context) {
return Scaffold (
appBar: AppBar(
title: Text('Fooderlich',
1/2
style: Theme.of (context).textTheme.titleLarge)), /
body: Center(
child: Text('Let\'s get cooking ',
// 3
style: Theme.of (context).textTheme.displayLarge)),
); // Scaffold

15. Analyze the Code Snippet with `FutureBuilder`

Let's break down the code snippet line by line:

```dart
return FutureBuilder(
```

- **Line 1**: This creates a `FutureBuilder` widget, which is used to asynchronously load data and rebuild the widget based on the state of the `Future`. It takes two important properties: `future` and `builder`.

```dart
future: mockService.getExploreData(),
```

- **Line 2**: The `future` property is assigned a `Future` that fetches data from `mockService.getExploreData()`. This `Future` represents an asynchronous operation that will eventually return some data. In this case, it seems like the `mockService` is responsible for fetching some kind of data (possibly a list of recipes or other exploration data).

```dart
builder: (context, snapshot) {
```

- **Line 3**: The `builder` property is a function that takes in two arguments: `context` and `snapshot`. `context` refers to the BuildContext of the widget, while `snapshot` holds the current state of the `Future` (whether it's waiting, has completed, or has an error).

```dart
if (snapshot.connectionState == ConnectionState.done) {
```

- **Line 4**: This checks if the `Future` has completed (i.e., the data has been successfully retrieved). `ConnectionState.done` means that the `Future` has completed, and now the `snapshot` contains the data.

```dart
final recipes = snapshot.data?.todayRecipes;
```

- **Line 5**: Once the `Future` is complete, this line extracts the `todayRecipes` data from the `snapshot.data`. This assumes that the `snapshot.data` contains a structured object with a property `todayRecipes` (perhaps a list of recipes for the day).

```dart
return Center(
  child:
}
```

- **Line 6**: This part of the code seems incomplete. It looks like the `Center` widget is intended to center its child widget on the screen, but the child widget (which should display the list of recipes or data) is missing.

```dart
  } else {
```

- **Line 7**: This `else` block handles cases when the `Future` hasn't finished yet or has an error. We expect to see some code here to handle those situations (like showing a loading spinner or an error message).

```dart
  Container(child: const Text('Show TodayRecipeListView'));
```

- **Line 8**: This line seems to display a placeholder `Container` with a simple `Text` widget saying 'Show TodayRecipeListView'. In practice, this would be replaced by a list view that shows today's recipes, which will need to be implemented in the `TODO` comment above.

```dart
  return const Center(child: CircularProgressIndicator());
```

- **Line 9**: If the `Future` is still in progress (i.e., if `snapshot.connectionState` is not `ConnectionState.done`), this line returns a `Center` widget with a `CircularProgressIndicator` that shows a loading spinner. This indicates to the user that data is being fetched.

```dart
  }); // FutureBuilder
```

- **Line 10**: This closes the `FutureBuilder` widget.

Explanation of Code Behavior:

1. The `FutureBuilder` widget waits for a `Future` to complete (in this case, the `getExploreData()` call).
2. Once the `Future` completes (in `ConnectionState.done`), it extracts the `todayRecipes` data from the `snapshot` and renders the content (possibly a list view).
3. If the `Future` is still in progress, a loading spinner (`CircularProgressIndicator`) is shown.
4. In case of an error or when the `Future` is not done, the code would need to handle that scenario, potentially by displaying an error message or placeholder content.

20. What is Scaffold? Analyze and Explain the Following Code:

Explanation of Scaffold: `Scaffold` is a fundamental layout structure in Flutter that provides several useful widgets to build a basic app interface. It typically includes an `AppBar`, a `body`, a `floatingActionButton`, a `Drawer`, and more.

Now let's analyze the provided code:

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

- This imports the Flutter Material package, which is necessary for using Material Design widgets like `` `Scaffold` ``, `` `AppBar` ``, and `` `Text` ``.

```dart
class Home extends StatefulWidget {
  const Home({Key? key}) : super(key: key);
  @override
  HomeState createState() => _HomeState();
}
```

- This creates a `` `Home` `` class, which is a `` `StatefulWidget` ``. `` `StatefulWidget` `` is used when the widget has mutable state, meaning it can change over time. In this case, it creates an instance of the `` `HomeState` `` class which manages the state of the `` `Home` `` widget.

```dart
class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Fooderlich',
          style: Theme.of(context).textTheme.titleLarge),
      ),
```

- The `` `HomeState` `` class overrides the `` `build` `` method, which describes the part of the user interface represented by this widget.
- A `` `Scaffold` `` is returned here. Inside the `` `Scaffold` ``, the `` `appBar` `` property is set to an `` `AppBar` `` widget, which contains the title "Fooderlich". The title's style is set using the `` `Theme.of(context).textTheme.titleLarge` ``, which applies the app's theme to the text.

```dart
      body: Center(
        child: Text('Let\'s get cooking ',
          style: Theme.of(context).textTheme.displayLarge),
      ),
    );
  }
}
```

- The `` `body` `` property of the `` `Scaffold` `` contains a `` `Center` `` widget, which centers its child widget. The child is a `` `Text` `` widget with the message "Let's get cooking". The text style is set to the app's theme (`` `Theme.of(context).textTheme.displayLarge` ``), which likely corresponds to a larger font size.
- This simple UI displays a centered message under the app bar.

Code Summary:

- `Scaffold` is the main layout structure for the screen.
- `AppBar` displays the app's title at the top.
- The `body` is centered, containing a `Text` widget with the message "Let's get cooking". The text is styled using the app's theme.

The code represents a basic Flutter app with a title bar and centered text, suitable for a simple landing screen or introductory page in a food-related app.