Name - Tejas Shailendra Rokade Div - D15A Roll no - 50 Batch - C

Experiment - 3

Aim - To include fonts and image in the flutter app.

Theory -

In essence, Flutter widgets serve as fundamental building blocks for crafting the user interface of a Flutter application. They can be broadly classified into two categories: `StatelessWidget`, representing immutable aspects of the UI, and `StatefulWidget`, representing mutable components that can undergo changes over time.

Some noteworthy Flutter widgets include:

- 1. **Scaffold**: Provides the basic structure for a Flutter app, incorporating layout elements like AppBar, BottomNavigationBar, and a body for primary content.
- 2. **Container**: A versatile box model utilized for layout, padding, margin, decoration, and constraints, capable of containing other widgets.
- 3. **Row & Column**: Widgets designed for arranging child widgets horizontally (Row) or vertically (Column), pivotal for creating flexible and responsive layouts.
- 4. **Text**: Used for displaying text on the screen, with support for various styling options such as font size, color, and alignment.
- 5. **TextField**: Captures user input, such as text, numbers, or passwords, with the `onChanged` property enabling dynamic updates based on user input.
- 6. **Buttons**: Various button widgets like `ElevatedButton` or `TextButton` trigger actions when pressed, providing a means for user interaction.
- 7. **Forms**: The `Form` widget manages a group of `TextFormField` widgets, facilitating input validation and submission.
- 8. **Icons**: The `Icon` widget displays icons from libraries, enhancing visual elements and conveying meaning through symbols.

Key Design Principles emphasized include:

- **Consistency**: Utilizing common widgets fosters a consistent design language throughout the app.
- **Responsive Layouts**: Widgets like `Row` and `Column` assist in creating responsive and flexible layouts, adapting to different screen sizes.
- **User Input Handling**: `TextField` and `Form` widgets facilitate proper handling, ensuring data integrity and validation.
- **Interactive Elements**: Buttons and icons contribute to interactivity and user engagement within the app.

- **Visual Styling**: The `Container` widget and styling properties of other widgets allow for visual customization and theming.

Code -

```
For Image and card creation -
class CategoryListState extends State<CategoryList> {
final List<String> categories = ['Category 1', 'Category 2', 'Category 3'];
String selectedCategory = 'Category 1'; // Default category
@override
Widget build(BuildContext context) {
  return Container(
   padding: EdgeInsets.symmetric(horizontal: 16.0),
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     Text(
      'Category List',
      style: TextStyle(
        fontSize: 20.0,
        fontWeight: FontWeight.bold,
      ),
     ),
     SizedBox(height: 8.0),
     Row(
      children: [
        Text('Select a category: '),
        DropdownButton<String>(
         value: selectedCategory,
         onChanged: (String? newValue) {
          // Update the selected category when the dropdown changes
          if (newValue != null) {
            setState(() {
             selectedCategory = newValue;
           });
          }
         items: categories.map((String category) {
          return DropdownMenuItem<String>(
           value: category,
            child: Text(category),
          );
         }).toList(),
```

```
),
     ],
    SizedBox(height: 50.0),
   // Updated section with 4 cards and images
   Container(
     height: 200.0,
     child: ListView.builder(
      scrollDirection: Axis.horizontal,
      itemCount: 4, // Number of cards
      itemBuilder: (context, index) {
       return Card(
         margin: EdgeInsets.all(8.0),
         child: Container(
          width: 150.0, // Customize the card width as needed
          child: Column(
           children: [
             // Add an Image widget with the image path
             Image.asset(
              'assets/card_image_${index + 1}.jpeg',
              height: 120.0,
              width: 150.0,
              fit: BoxFit.cover,
             ),
             ListTile(
              title: Text('Card ${index + 1}'),
              subtitle: Text('Description of card ${index + 1}'),
              // Add onTap callback if needed
             ),
  ],
);
```

```
# Flutter commands

# The following section is specific to Flutter packages.

| The following line ensures that the Material Icons font is
| # The following line ensures that the Material Icons font is
| # Included with your application, so that you can use the icons in
| # the material Icons class.
| Uses-material-design: true

| # To add assets to your application, add an assets section, like this:
| assets:
| assets/card_image_4.jpeg |
| assets/card_image_1.jpeg |
| assets/card_image_2.jpeg |
| assets/card_image_2.jpeg |
| assets/card_image_2.jpeg |
| # - images/a_dot_burr.jpeg |
| # - images/a_dot_burr.jpeg |
| # An image asset can refer to one or more resolution-specific "variants", see |
| # https://flutter.dev/assets-and-images/#resolution-aware |
| # For details regarding adding assets from package dependencies, see |
| # https://flutter.dev/assets-and-images/#from-packages |
| # To add custom fonts to your application, add a fonts section here,
```



Category List

restaurants in town.

Select a category: Hot food



Card 1
Description of card 1



Card 2
Description of card 2

([c

Special Offers On

Select a category: Hot food

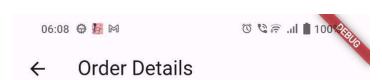












Your Order Details

Add your order details here

Go Back

Conclusion -

Through this experiment we understood how to insert images and fonts into the flutter app and how to use them and call them from the assets folder .