RIPHAH INTERNATIONAL UNIVERSITY

MAD LAB REPORT

NAME:

Muhammad Shameel

Sap Id:

55579

Section:

SE (5-2)

Abstract

This lab demonstrates the implementation of various scrolling widgets and layout components in Flutter, including **Card**, **ListView**, **GridView**, **Stack**, and **CustomScrollView** with slivers.

1. Introduction

1.1 Objectives

- Learn to use **Card** widget for grouping information with rounded corners and shadows.
- Implement linear scrollable lists using **ListView** and **ListTile**.
- Create grid layouts using **GridView**.
- Overlap and position widgets using **Stack**.
- Create custom scrolling effects using **CustomScrollView** and **slivers**.

1.2 Tools and Technologies

- **Flutter SDK**
- **Dart Programming Language**
- **Android Studio**
- **Mobile Emulator**

2. Implementation

2.1 Project Structure

```
lib/
├── main.dart
├── screens/
│   └── home_screen.dart
└── widgets/
    ├── card_demo.dart
    ├── listview_demo.dart
    ├── gridview_demo.dart
    ├── stack_demo.dart
    └── custom_scrollview_demo.dart
```

2.2 Main Application Setup

main.dart

```
MaterialApp(
  title: 'Scrolling Lists & Effects Demo',
  theme: ThemeData(primarySwatch: Colors.blue),
  home: HomeScreen(),
  debugShowCheckedModeBanner: false,
)
```

3. Widget Implementations and Results

3.1 Card Widget

Purpose: Group information with rounded corners and drop shadow.

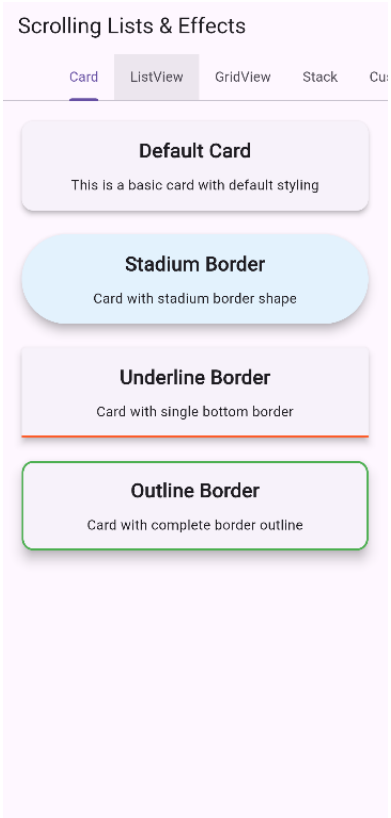
Key Features:

- Custom elevation for shadow effects
- Various border shapes (StadiumBorder, UnderlineInputBorder, OutlineInputBorder)
- Customizable colors and margins

Code Implementation:

```
Card(
  elevation: 8.0,
  shape: StadiumBorder(),
  color: Colors.blue[50],
  child: Padding(
    padding: EdgeInsets.all(16),
    child: Column(children: [/* Content */]),
  ),
)
```

Screenshot:



Observations:

- Cards provide excellent visual hierarchy.
- Different shapes create varied visual effects.
- Elevation property controls shadow intensity.

3.2 ListView with ListTile

Purpose: Create linear scrollable lists.

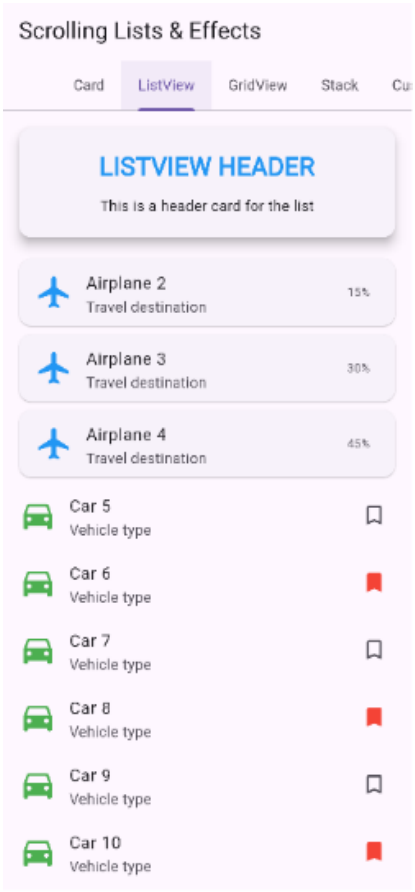
Key Features:

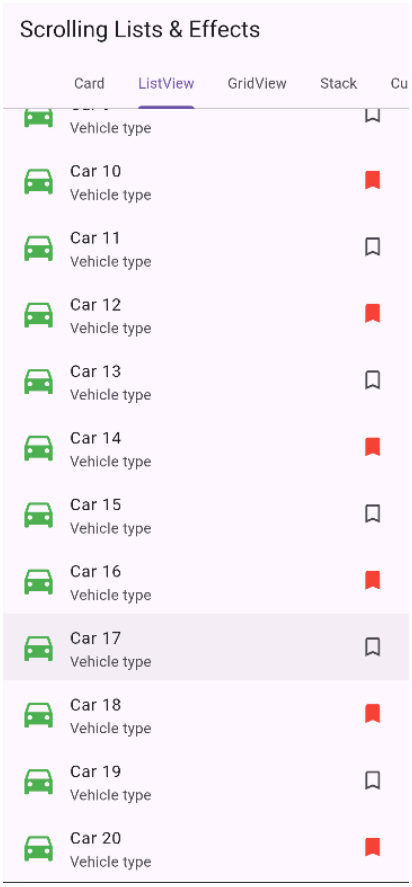
- `ListView.builder` for efficient rendering.
- `ListTile` for consistent list item layout.
- Leading and trailing widgets.
- OnTap functionality.

Code Implementation:

```
ListView.builder(
  itemCount: items.length,
  itemBuilder: (context, index) {
    return ListTile(
      leading: Icon(Icons.flight),
      title: Text('Item $index'),
      trailing: Icon(Icons.bookmark),
      onTap: () {/* Handle tap */},
    );
  },
)
```

Screenshot:





Observations:

- `ListView.builder` only renders visible items (performance optimization).
- `ListTile` provides consistent spacing and alignment.
- Different item types can be mixed in the same list.

3.3 GridView

Purpose: Display items in grid format.

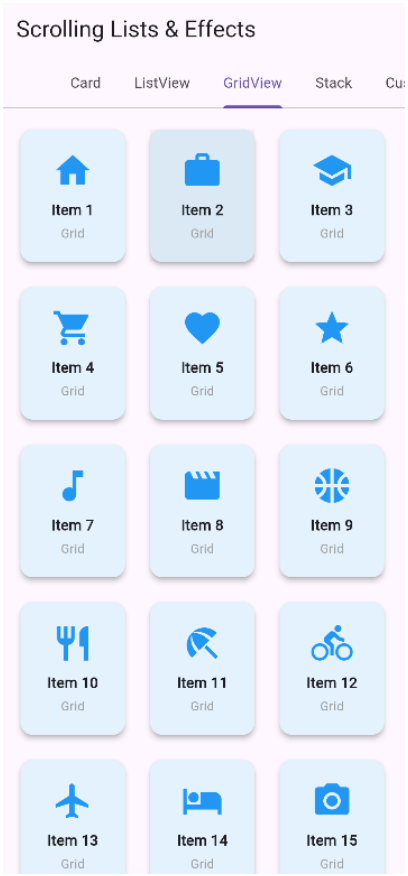
Key Features:

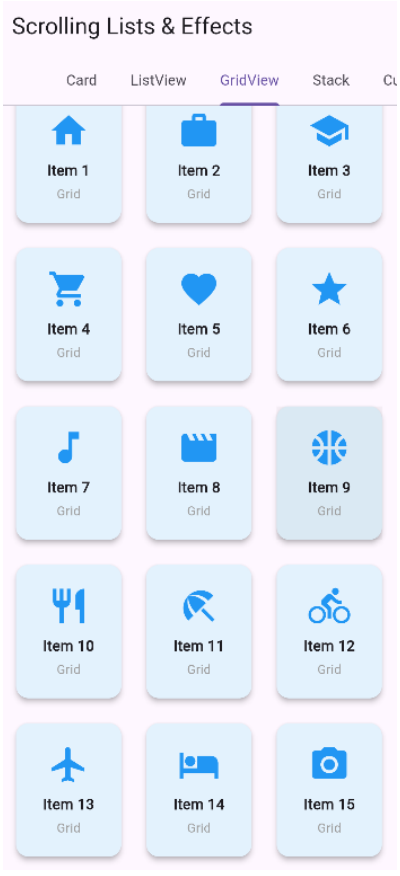
- `GridView.builder` for efficient rendering.
- Customizable grid dimensions.
- Responsive layout.
- Interactive grid items.

Code Implementation:

```
GridView.builder(
  gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(
    maxCrossAxisExtent: 150,
    mainAxisSpacing: 16,
    crossAxisSpacing: 16,
  ),
  itemBuilder: (context, index) {
    return Card(child: /* Grid item content */);
  },
)
```

Screenshot:





Observations:

- Grid adapts to different screen sizes.
- `maxCrossAxisExtent` controls item width.
- Spacing properties control visual separation.

3.4 Stack Widget

Purpose: Overlap and position widgets.

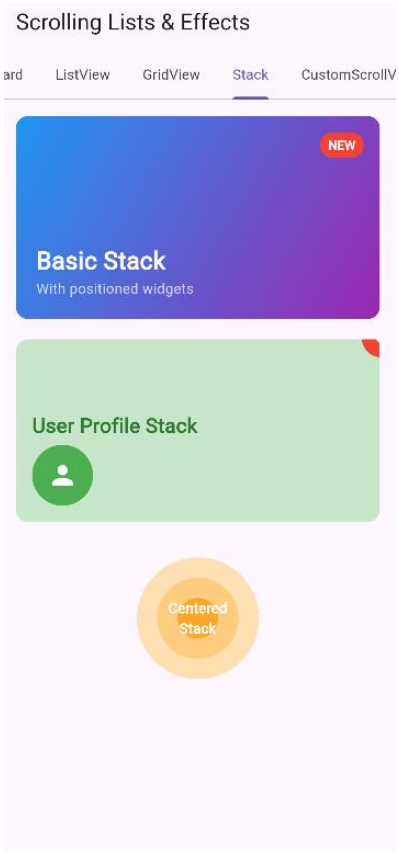
Key Features:

- Widget overlapping.
- `Positioned` widget for precise placement.
- `FractionalTranslation` for relative positioning.
- Z-index ordering.

Code Implementation:

```
Stack(
  children: [
    Container(/* Background */),
    Positioned(
      bottom: 20,
      left: 20,
      child: Text('Overlay Text'),
    ),
    Positioned(
      top: -10,
      right: -10,
      child: FractionalTranslation(
        translation: Offset(0.3, -0.3),
        child: CircleAvatar(/* Icon */),
      ),
    ),
  ],
)
```

Screenshot:



Observations:

- Children are drawn in order (first = bottom).
- `Positioned` allows pixel-perfect placement.
- `FractionalTranslation` enables relative positioning.

3.5 CustomScrollView with Slivers

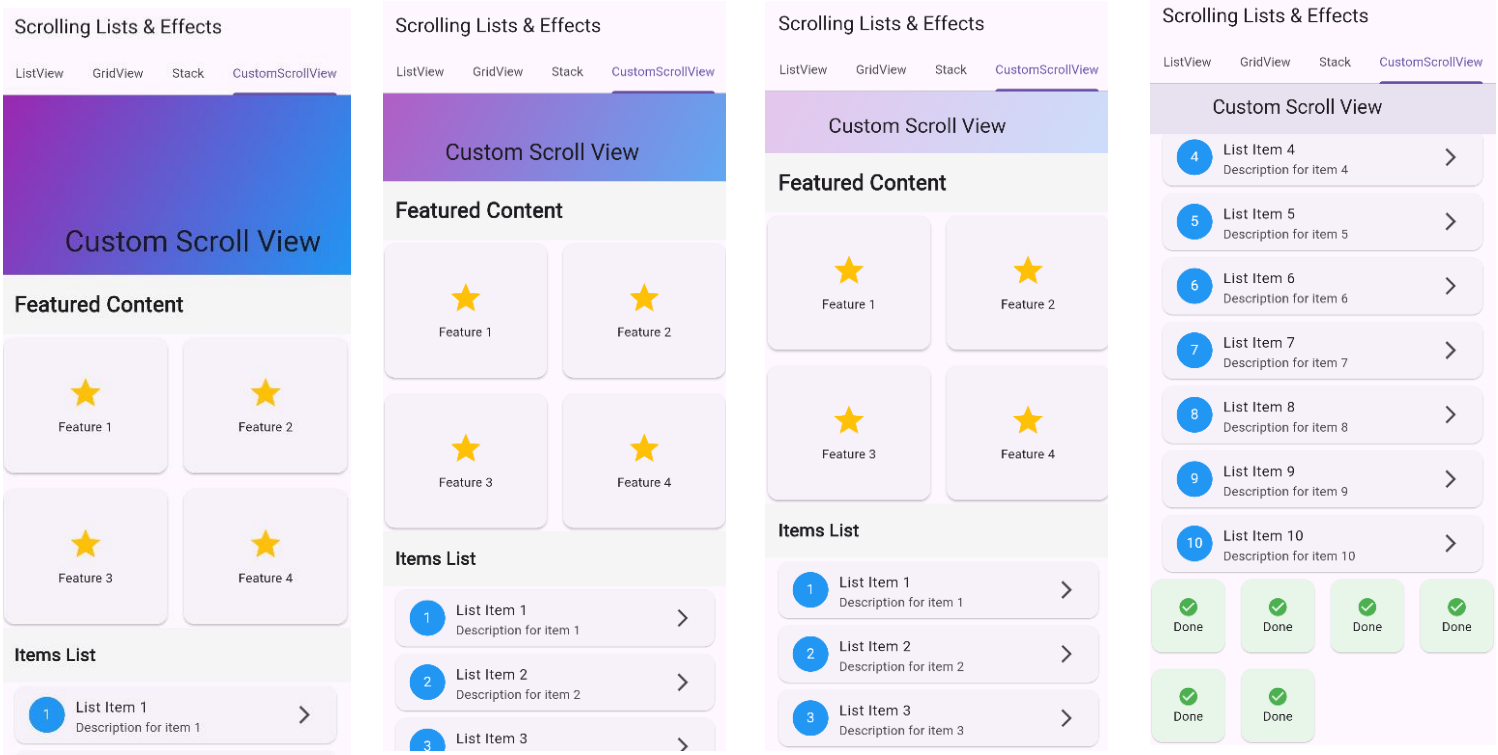
Purpose: Create custom scrolling effects.

Key Features:

- Multiple sliver widgets in a single scroll view.
- Parallax effects.
- Mixed content types (list + grid).
- `SliverAppBar` with flexible space.

Code Implementation:

```
CustomScrollView(
  slivers: [
    SliverAppBar(expandedHeight: 200, flexibleSpace: /* Parallax */),
    SliverGrid(delegate: /* Grid items */, gridDelegate: /* Layout */),
    SliverList(delegate: /* List items */),
  ],
```



Observations:

- Slivers enable complex scrolling behaviors.
- Parallax effects enhance visual appeal.
- Multiple content types can coexist in a single scroll view.

4. Performance Considerations

4.1 Efficient List Rendering

- Used `ListView.builder` and `GridView.builder` for large datasets.
- Lazy loading improves performance.
- Only visible items are rendered.

4.2 Memory Management

- Used **Stateless** widgets where possible.
- Used **const constructors** for better performance.
- Implemented efficient rebuild strategies.

5. Conclusion

This lab successfully demonstrated the implementation of various scrolling and layout widgets in Flutter. Each widget serves specific purposes:

- **Card:** Perfect for grouping related information with visual appeal.
- **ListView:** Ideal for linear data presentation.
- **GridView:** Excellent for two-dimensional data display.
- **Stack:** Powerful for overlapping and custom layouts.
- **CustomScrollView:** Provides ultimate flexibility for complex scrolling needs.

The implementation followed Flutter best practices and focused on **performance optimization** through efficient rendering techniques.

6. GitHub Repository

You can access the full Flutter project source code and files at the following link:

🔗 **GitHub Link:** https://github.com/M-Shameel-375/Mobile-App-Development/tree/main/lab_10