Component trong Jetpack Compose (tiếp theo)

- List

Nội dung

- Lazy Lists (LazyColumn, LazyRow)
- Lazy grids
- Lazy Staggered Grid
- AsyncImage

Lazy composables

Nếu bạn cần hiển thị một số lượng lớn các mục (hoặc một danh sách có độ dài không xác định), việc sử dụng một bố cục như Column có thể gây ra vấn đề về hiệu suất, vì tất cả các mục sẽ được sắp xếp và bố trí dù chúng có hiển thị hay không.

Trong Jetpack Compose, “Lazy composables” là những thành phần được thiết kế để hiển thị danh sách hoặc lưới dữ liệu một cách hiệu quả. Chúng chỉ tạo ra và hiển thị những mục đang được nhìn thấy trên màn hình, thay vì tải tất cả mục ngay từ đầu. Điều này giúp tiết kiệm bộ nhớ và tăng hiệu suất, đặc biệt là khi làm việc với danh sách lớn

Lazy composables

LazyColumn – Tạo ra một danh sách cuộn dọc

LazyRow – Tạo ra một danh sách cuộn ngang

For Android developers,

LazyColumn - Vertical RecyclerView

LazyRow - Horizontal RecyclerView

LazyColumn

Trong LazyColumn bạn có thể thêm một **item()** hoặc **items()**

Đối với single composable có thể sử dụng item()

Đổi với danh sách (list of composable) có thể sử dụng **items(count: Int)** hoặc **items(items: List<T>)**

LazyColumn

***Ví dụ:** **Single item**

```
@Composable
fun ListListScopeSample(){
    LazyColumn {
        // Add a single item
        item {
            Text(text = "Header")
        }
    }
}
```

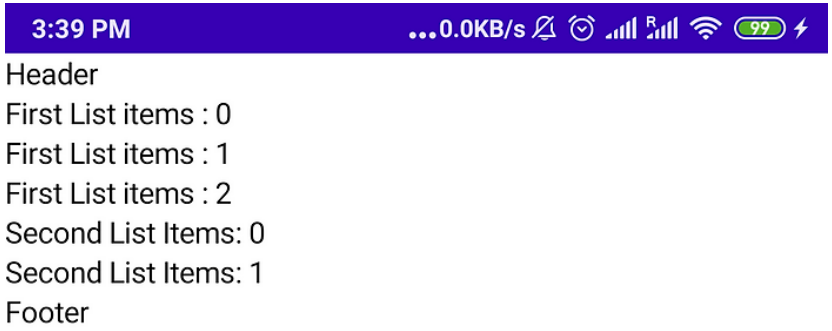
LazyColumn

***Ví dụ: Danh sách**

```
@Composable
fun ListListScopeSample(){
    LazyColumn {
        // Add 3 items
        items(3) { index ->
            Text(text = "First List items : $index")
        }
    }
}
```

LazyColumn

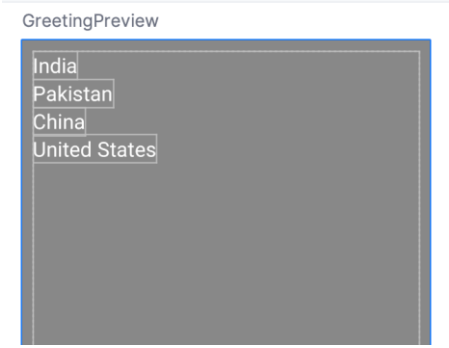
***Ví dụ: Single item kết hợp với Danh sách**




```
@Composable
fun ListListScopeSample(){
    LazyColumn {
        // Add a single item
        item {
            Text(text = "Header")
        }

        // Add 3 items
        items(3) { index ->
            Text(text = "First List items : $index")
        }

        // Add 2 items
        items(2) { index ->
            Text(text = "Second List Items: $index")
        }

        // Add another single item
        item {
            Text(text = "Footer")
        }
    }
}
```

Simple List

***Ví dụ: Hiển thị danh sách các quốc gia – sử dụng items(items: List<T>)**




```
private val countryList =
    mutableListOf("India", "Pakistan", "China", "United States")

private val listModifier = Modifier
    .fillMaxSize()
    .background(Color.Gray)
    .padding(10.dp)

private val textStyle = TextStyle(fontSize = 20.sp, color = Color.White)

@Composable
fun SimpleListView() {
    LazyColumn(modifier = listModifier) {
        items(countryList) { country ->
            Text(text = country, style = textStyle)
        }
    }
}
```

Custom List

***Ví dụ:** Hiển thị danh sách các loại trái cây với hình ảnh của từng loại*

Bước 1: Tạo model Fruit gồm 2 thuộc tính: tên và hình ảnh

```
data class FruitModel(val name:String, val image : Int)
```

Custom List

***Ví dụ:** Hiển thị danh sách các loại trái cây với hình ảnh của từng loại*

Bước 2: Custom giao diện của từng item hiển thị trên danh sách

- **Để hiển thị hình ảnh ta sử dụng Image()**
- **Để hiển thị tên ta sử dụng Text()**
- **Sử dụng Row để sắp xếp Image() và Text() nằm ngang với nhau (horizontal)**

Custom List

```
@Composable
fun ListRow(model: FruitModel) {
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = Modifier
            .wrapContentHeight()
            .fillMaxWidth()
            .background(color = Color.Blue)
    ) {
        Image(
            painter = painterResource(id = model.image),
            contentDescription = "",
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .size(100.dp)
                .padding(5.dp)
        )
        Text(
            text = model.name,
            fontSize = 24.sp,
            fontWeight = FontWeight.SemiBold,
            color = Color.White
        )
    }
}
```

Custom List

***Ví dụ: Hiển thị danh sách các loại trái cây với hình ảnh của từng loại**

Bước 3: Tạo mutableListOf để add danh sách trái cây

```
private val fruitsList = mutableListOf<FruitModel>()
```

Custom List

***Ví dụ: Hiển thị danh sách các loại trái cây với hình ảnh của từng loại**

Bước 4: Thêm trái cây vào danh sách

```
fruitsList.add(FruitModel("Apple", R.drawable.apple))
fruitsList.add(FruitModel("Orange", R.drawable.orange))
fruitsList.add(FruitModel("Banana", R.drawable.banana))
fruitsList.add(FruitModel("Strawberry", R.drawable.strawberry))
fruitsList.add(FruitModel("Mango", R.drawable.mango))
```

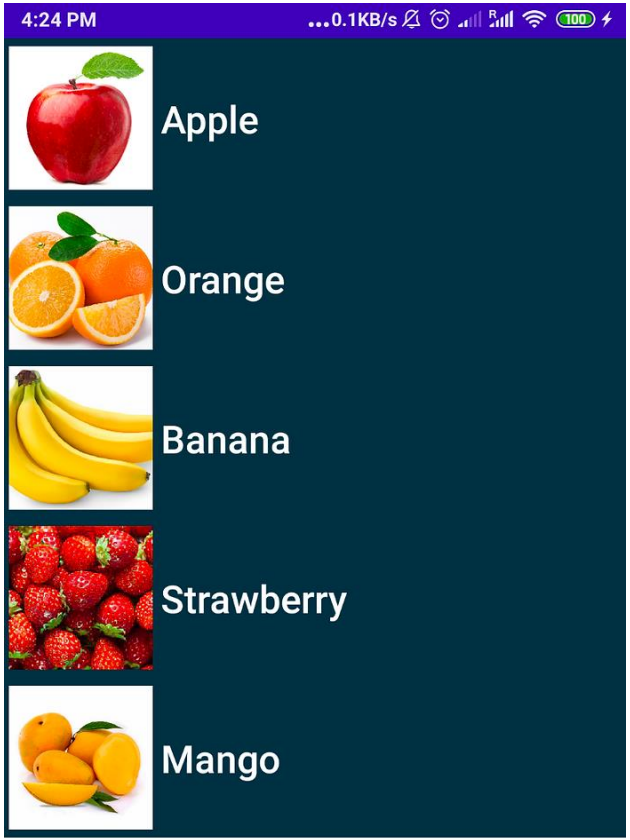
Custom List

***Ví dụ: Hiển thị danh sách các loại trái cây với hình ảnh của từng loại**

Bước 5: Sử dụng LazyColumn để hiển thị danh sách

```
LazyColumn(
    modifier = Modifier
        .fillMaxSize()
        .background(Color.White)
) {
    items(fruitsList) { model ->
        ListRow(model = model)
    }
}
```

Custom List



Content Padding

Để thêm khoảng cách cho các nội dung (đối tượng xung quanh) ta có thể sử dụng **PaddingValues**

Ví dụ:

```
LazyColumn(
    contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),
) {
        //…
}
```

Khoảng cách nội dung (Content Spacing)

Để thêm khoảng cách giữa các item, bạn có thể sử dụng **Arrangement.spacedBy()**.

Ví dụ: thêm 4.dp khoảng trắng ở giữa items:

```
LazyColumn(
    verticalArrangement = Arrangement.spacedBy(4.dp),
) {
        //…
}
```

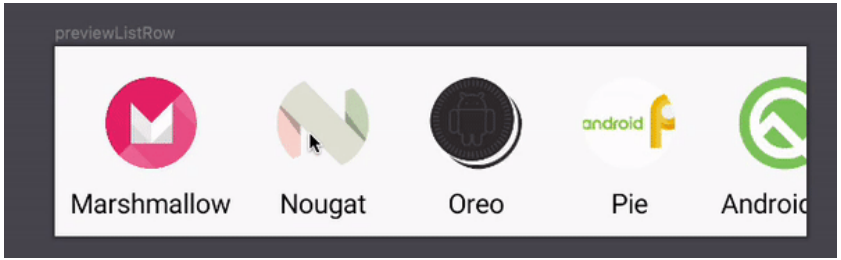
LazyRow

Một số thuộc tính chính trong LazyRow:

- **state**: Để kiểm soát và truy xuất vị trí cuộn.
- **contentPadding**: Padding xung quanh nội dung bên trong.
- **horizontalArrangement**: Sắp xếp các item theo chiều ngang.
- **reverseLayout**: Hiển thị item theo thứ tự ngược lại.
- **flingBehavior**: Tùy chỉnh cách cuộn nhanh phản ứng.
- **content**: Định nghĩa nội dung bên trong, bao gồm các item.
- **verticalAlignment**: Căn chỉnh các item theo chiều dọc.

LazyRow

```
LazyRow {
    items(fruitsList) { item ->
        Column(
            Modifier.padding(8.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Image(
                modifier = Modifier
                    .size(64.dp)
                    .padding(8.dp)
                    .clip(RoundedCornerShape(50)),
                contentScale = ContentScale.Crop,
                painter = painterResource(id = item.image),
                contentDescription = null
            )
            Text(text = item.name)
        }
    }
}
```



LazyGrids

Trong Jetpack Compose, “Lazy grids” là một cách để hiển thị các item trong một lưới có thể cuộn mà chỉ soạn và bố trí các mục hiển thị trong phạm vi nhìn thấy. Điều này giúp cải thiện hiệu suất và giảm tiêu thụ bộ nhớ khi bạn có một số lượng lớn mục cần hiển thị. Có hai loại lưới lười biếng chính:

1.LazyVerticalGrid: Hiển thị các mục trong một lưới dọc có thể cuộn qua nhiều cột.

2.LazyHorizontalGrid: Tương tự như LazyVerticalGrid nhưng cho phép cuộn theo chiều ngang.

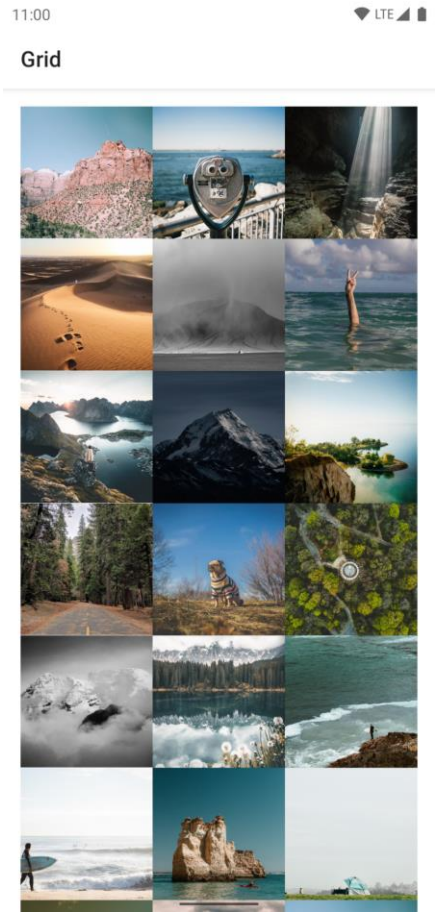
Cả hai đều sử dụng một DSL tương tự nhau - **LazyGridScope.()** để mô tả nội dung.

LazyGrids

```
@Composable
fun MyGrid() {
    LazyVerticalGrid(
        columns = GridCells.Fixed(3),
        contentPadding = PaddingValues(8.dp)
    ) {
        items(50) { index ->
            Card(
                modifier = Modifier.padding(8.dp),
                elevation = CardDefaults.cardElevation(2.dp)
            ) {
                Text("Item $index")
            }
        }
    }
}
```

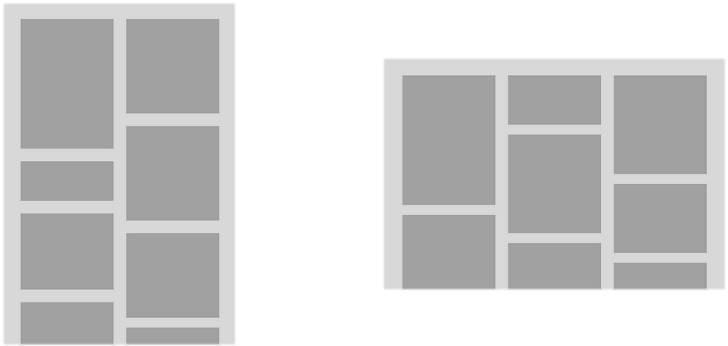
LazyGrids

Ta được kết quả:



Lazy Staggered Grid

LazyVerticalStaggeredGrid và **LazyHorizontalStaggeredGrid** là những thành phần mới được thêm vào trong Jetpack Compose từ phiên bản 1.3.0, cho phép bạn tạo ra các danh sách nội dung theo kiểu lưới chồng chéo (staggered grid), hỗ trợ việc sắp xếp các mục có kích thước đa dạng một cách hiệu quả và hỗ trợ lazy composition.



LazyVerticalStaggered (Grid Lazy Staggered Grid)

Một số thuộc tính chính thường gặp:

- **columns**: Định nghĩa cách phân chia các cột trong lưới.
  StaggeredGridCells.Fixed(count) xác định số cột cố định, trong khi **StaggeredGridCells.Adaptive(minSize)** cho phép điều chỉnh số cột dựa trên kích thước tối thiểu của item.
- **state**: Một instance của **LazyStaggeredGridState** giúp quản lý và truy cập trạng thái cuộn của lưới.

LazyVerticalStaggered (Grid Lazy Staggered Grid)

Một số thuộc tính chính thường gặp:

- **contentPadding**: Định nghĩa khoảng cách padding xung quanh nội dung của lưới.
- **horizontalArrangement**: Xác định cách các item được sắp xếp và khoảng cách giữa chúng theo chiều ngang.
- **verticalItemSpacing**: Khoảng cách giữa các item theo chiều dọc.

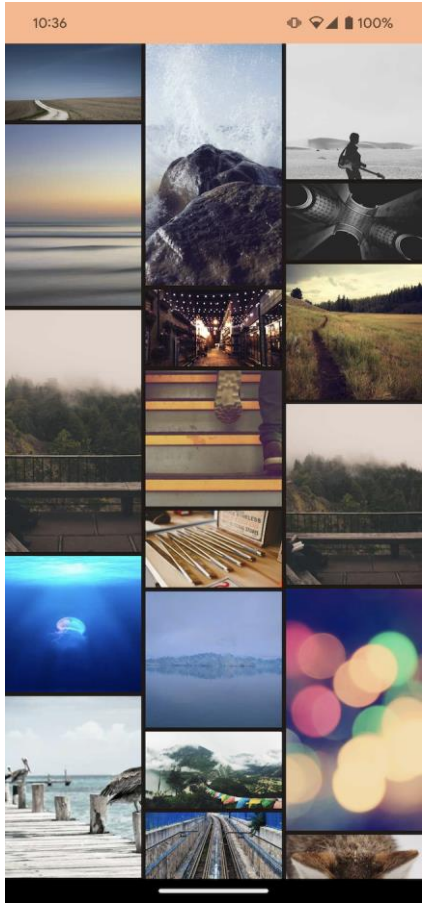
- **content**: Định nghĩa nội dung bên trong của **LazyVerticalStaggeredGrid**, thường là một danh sách các composable item được hiển thị.

LazyVerticalStaggered (Grid Lazy Staggered Grid)

```
@Composable
fun SampleStaggeredGrid() {
    LazyVerticalStaggeredGrid(
        columns = StaggeredGridCells.Fixed(3),
        verticalItemSpacing = 4.dp,
        horizontalArrangement = Arrangement.spacedBy(4.dp),
        content = {
            items(randomSizedPhotos) { photo ->
                AsyncImage(
                    model = photo,
                    contentScale = ContentScale.Crop,
                    contentDescription = null,
                    modifier = Modifier.fillMaxWidth().wrapContentHeight()
                )
            }
        },
        modifier = Modifier.fillMaxSize()
    )
}
```

LazyVerticalStaggered (Grid Lazy Staggered Grid)

Ta được kết quả:



AsyncImage

AsyncImage trong Jetpack Compose giúp hiển thị hình ảnh từ nguồn như URL hay tài nguyên. Dưới đây là các thuộc tính chính:

- **model**: Nguồn hình ảnh cần hiển thị.
- **contentDescription**: Mô tả về hình ảnh, hỗ trợ, trợ năng.
- **contentScale**: Cách hình ảnh được điều chỉnh để khớp với kích thước **composable**.
- **modifier**: Hiệu chỉnh **layout** và vẽ, như cắt hoặc thiết lập kích thước.

AsyncImage

AsyncImage trong Jetpack Compose giúp hiển thị hình ảnh từ nguồn như URL hay tài nguyên. Dưới đây là các thuộc tính chính:

- **placeholder**: Hình ảnh hiển thị khi hình chính đang tải.
- **error**: Hình ảnh hiển thị nếu có lỗi tải hình chính.
- **onLoading**: **Callback** khi bắt đầu tải hình ảnh.
- **onSuccess**: **Callback** khi hình ảnh tải và hiển thị thành công.
- **onError**: **Callback** khi có lỗi trong quá trình tải hoặc hiển thị hình ảnh.

AsyncImage

```
AsyncImage(
    model = ImageRequest.Builder(LocalContext.current)
        .data("https://example.com/image.jpg")
        .crossfade(true)
        .build(),
    placeholder = painterResource(R.drawable.placeholder),
    contentDescription = stringResource(R.string.description),
    contentScale = ContentScale.Crop,
    modifier = Modifier.clip(CircleShape)
)
```

Thanks!