Code Report – Homework1

In this project, I created a simple Jetpack Compose screen to practice building a user interface. I started by setting up a MainActivity and used setContent to display my composable function. Inside my MyScreen() composable, I added different UI elements step by step to understand how Compose layouts work.

First, I created a state variable using remember { mutableStateOf(0) } so I could update the screen whenever the user clicked a button. This helped me learn how state and recomposition work in Compose.

Then I used a LazyColumn because I wanted the whole page to be scrollable and also include a long list of items. I learned that LazyColumn is better than a normal Column for long lists because it only loads the items that are visible on the screen.

Inside the LazyColumn, I added an image at the top using Image() and loaded it from the drawable folder. After that, I added two Text elements: one as a title and one as a smaller description. I used Spacer to create space between the elements so the layout looks cleaner.

Next, I added a Button that increases the counter every time it is clicked. Below the button, I displayed the updated counter value using another Text. This part helped me understand how user interaction updates the UI automatically.

Finally, I added a list of 40 scrollable items using items() inside the LazyColumn. Each item shows its index number, and I added a small space between them to make the list easier to read.

Overall, this project helped me understand how composables, state, layout, and scrolling work together in Jetpack Compose. I built the screen step by step and tested each part to make sure I understood how it behaves

1. Overall Structure of the App

The MainActivity class extends ComponentActivity and uses setContent to display a composable function instead All UI elements are created inside the MyScreen() composable.

2. State Management

Inside MyScreen, the following line creates a state variable:

kotlin

var counter by remember { mutableStateOf(0) }

- counter stores how many times the button has been clicked.
- remember keeps the value during recomposition.
- mutableStateOf triggers UI updates when the value changes.

Purpose: This demonstrates reactive UI behavior in Compose.

3. Using LazyColumn

kotlin

```
LazyColumn(
    modifier = Modifier.fillMaxSize().padding(16.dp),
    horizontalAlignment = Alignment.CenterHorizontally
)
```

- LazyColumn creates a vertically scrollable list.
- It loads only visible items, making it efficient for long lists.
- Items are added using item { ... } and items { ... }.

Why LazyColumn? It is optimized for performance and ideal for lists with many elements.

4. Displaying an Image

kotlin

```
Image(
    painter = painterResource(id = R.drawable.pic),
    contentDescription = null,
    modifier = Modifier.size(200.dp)
)
```

- Loads an image from the drawable folder.
- The size is fixed at 200dp.
- contentDescription is null because the image is decorative.

Purpose: Shows how to load and display images in Compose.

5. Text Elements

Two text components are used:

kotlin

```
Text("Welcome to my Compose App", fontSize = 24.sp)
Text("This is a smaller description text.", fontSize = 16.sp)
```

- The first is a title with a larger font.
- The second is a smaller descriptive text.

Purpose: Demonstrates typography and text styling.

6. Button and Counter

kotlin

```
Button(onClick = { counter++ }) {
    Text("Click me")
}
```

- Clicking the button increases the counter.
- The updated value is shown below:

kotlin

```
Text("Button clicked $counter times", fontSize = 20.sp)
```

Purpose: Shows how user interaction updates UI using state.

7. Spacing with Spacer

kotlin

```
Spacer(modifier = Modifier.height(20.dp))
```

- Adds vertical space between UI elements.
- Improves readability and layout structure.

8. Scrollable List of Items

kotlin

```
items((0 until 40).toList()) { index ->
    Text("Scrollable item #$index", fontSize = 18.sp)
    Spacer(modifier = Modifier.height(10.dp))
}
```

- Generates 40 scrollable items.
- Each item displays its index.
- A small space is added between items.

Purpose: Demonstrates dynamic list creation and scrolling behavior.