Task 1 — Composables Practice Reflection & Planning

Submitted by Vedaa.V

Contents:

Problem 1: Compose Article

1 .Short Summary of the Problem

This problem required building an article screen for a learning app. The UI consists of an image at the top and three text sections below it: a title, a short description, and a long description. The layout must follow specific padding, font size, and text alignment rules.

---

2. Planned Approach (Before Coding)

Before coding, I would:

- Use a Column composable because the elements are arranged vertically.
- Place the Image at the top and set it to fill the screen width.
- Use three Text composables below the image.
- Apply:
    - fontSize = 24.sp for the title.
    - Modifier.padding(16.dp) where required.
    - TextAlign.Justify for the description texts.
- Use stringResource() and painterResource() to properly load resources instead of hardcoding.

Layout structure plan:

```
Column
 ├── Image
 ├── Title Text
 ├── Short Description Text
 └── Long Description Text
```

---

3. Actual Result & Reflection (After Implementation)

After implementing:

- I successfully used a Column to vertically arrange elements.
- I applied padding correctly using Modifier.padding().
- I used TextAlign.Justify for paragraph alignment.
- I loaded strings and images using stringResource() and painterResource().

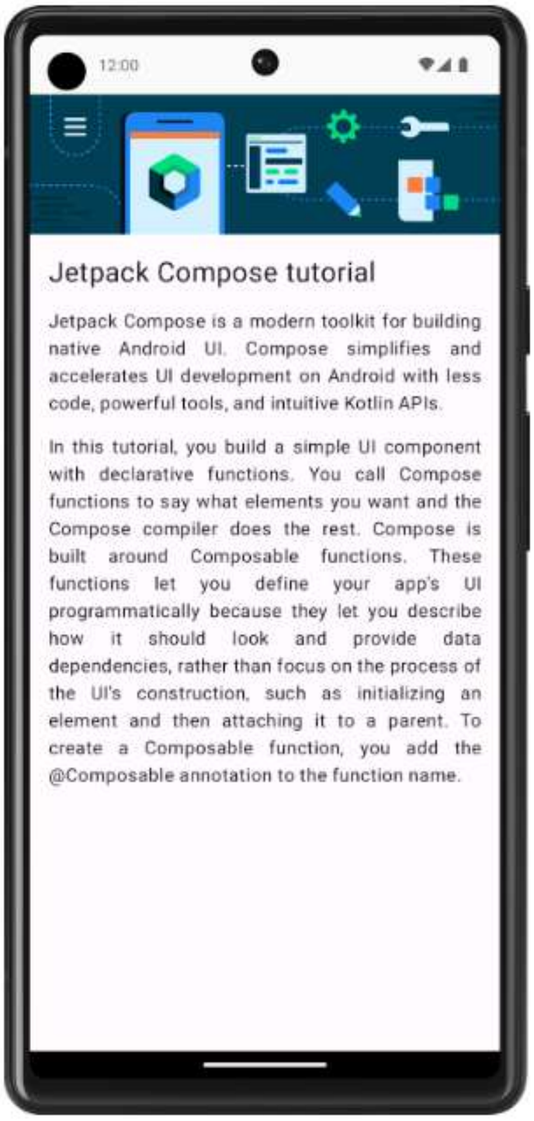
What was tricky?

- Remembering exact padding values (start/end vs full padding).
- Understanding how TextAlign.Justify affects long text.

- Making sure the image fills properly across the width.

What I would do differently next time

- I would explicitly add Modifier.fillMaxWidth() to the image to ensure full-width scaling.
- I would sketch the layout on paper first to visualize spacing.
- I would test on different screen sizes to check text wrapping.



Jetpack Compose tutorial

Jetpack Compose is a modern toolkit for building native Android UI. Compose simplifies and accelerates UI development on Android with less code, powerful tools, and intuitive Kotlin APIs.

In this tutorial, you build a simple UI component with declarative functions. You call Compose functions to say what elements you want and the Compose compiler does the rest. Compose is built around Composable functions. These functions let you define your app's UI programmatically because they let you describe how it should look and provide data dependencies, rather than focus on the process of the UI's construction, such as initializing an element and then attaching it to a parent. To create a Composable function, you add the @Composable annotation to the function name.

Problem 2 : Task Manager

1. Short Summary of the Problem

This problem required building a completion screen that displays an image and two text elements centered both vertically and horizontally. The first text must be bold with specific padding, and the second text must use a 16sp font size.

2.Planned Approach (Before Coding)

Before coding, I would:

- Use a Column composable since elements are arranged vertically.
- Set:
    - verticalArrangement = Arrangement.Center
    - horizontalAlignment = Alignment.CenterHorizontally
- Add an Image at the top.
- Add two Text composables:
    - First: Bold font weight + padding.
    - Second: 16sp font size.

Layout structure plan:

```
Column (centered)
 ├── Image
 ├── Bold Title Text
 └── Subtitle Text
```

2.3. Actual Result & Reflection (After Implementation)

After implementing:

- I used Column correctly with vertical and horizontal centering.
- I applied FontWeight.Bold properly.
- I added correct padding using Modifier.padding(top = 24.dp, bottom = 8.dp).
- I set font size using fontSize = 16.sp.

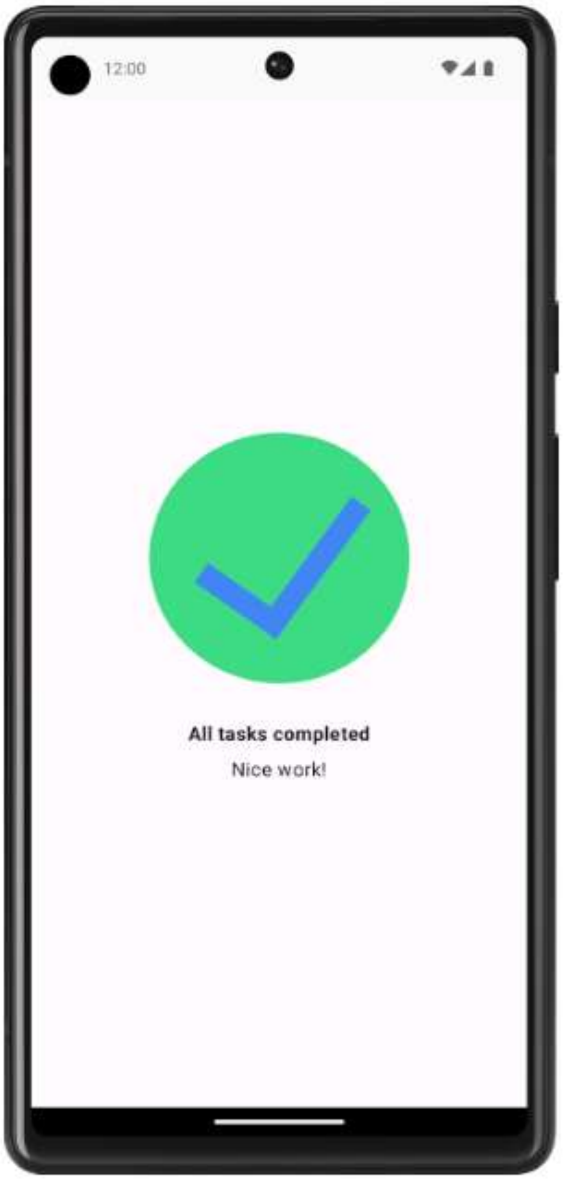
What was tricky?

- Understanding the difference between fillMaxWidth() and fillMaxSize().
- Making sure the content was truly centered vertically.
- Remembering that alignment is controlled by Column parameters, not by Modifier.padding().

What I would do differently next time

- I would simplify the modifier using fillMaxSize() instead of both width and height.

- I would preview early to check alignment visually.
- I would test how layout behaves if more text is added.



3 .Planning Checklist Template for New Composable UI Problems

When solving a new UI problem, I will follow this checklist:

- Carefully read the UI specification
- Identify layout direction (Vertical → Column, Horizontal → Row, Overlapping → Box)
- Sketch the layout structure
- Decide:
    - Which composables are needed?
    - What alignment is required?
- Add modifiers:
    - Padding
    - Size
    - Alignment
    - Font styling
      Use string and image resources (avoid hardcoding)
      Run Preview
      Test on different screen sizes
      Refactor and clean code

---

4. Short Reflection (3–4 lines)

Planning before coding improves my ability to build Compose UIs efficiently. It helps me clearly understand layout structure, alignment, and modifier usage. Reflection after implementation allows me to identify mistakes and improve layout decisions. Over time, this strengthens my conceptual understanding of composables instead of just memorizing syntax.