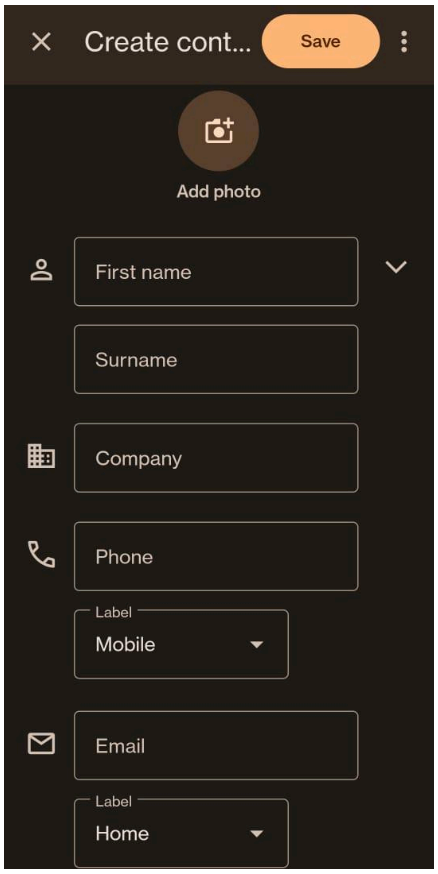
Android - Jetpack Compose Study Jam

Week 2

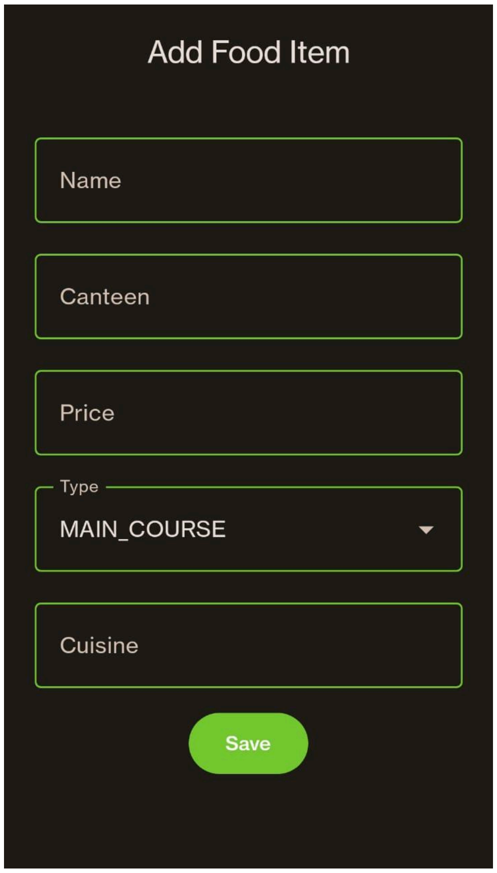
Welcome to week 2 of the Jetpack Compose Study Jam. This week we'll be learning how to create an AddItem screen and file to store our data for our project.

AddItem Screen



Ever notice those screens in your favorite apps where you can just add new stuff? Yep, those are called "Add Item Screens." They're basically like little creation stations, letting you add similar things to the main app area for all sorts of reasons

So,how do we create one?
Most of us know what a data class is.[Click here if you don't](). The basic activity for creating an AddItem screen is creating a data class.Its members are usually the details of the item that is displayed.Every item is an instance of the general data class created.It is obtained various UI components.



Some of the UI components and their uses could be as follows:

- Scaffold: Provides a basic structure with a top bar and content area.
- CenterAlignedTopAppBar: Displays the title at the top.
- Column: Arranges UI elements vertically with spacing.
- OutlinedTextField: Text input fields
- ExposedDropdownMenuBox: A dropdown menu to select the itemtype.
- Button: Saves the entered information upon clicking.

FILE STORAGE:

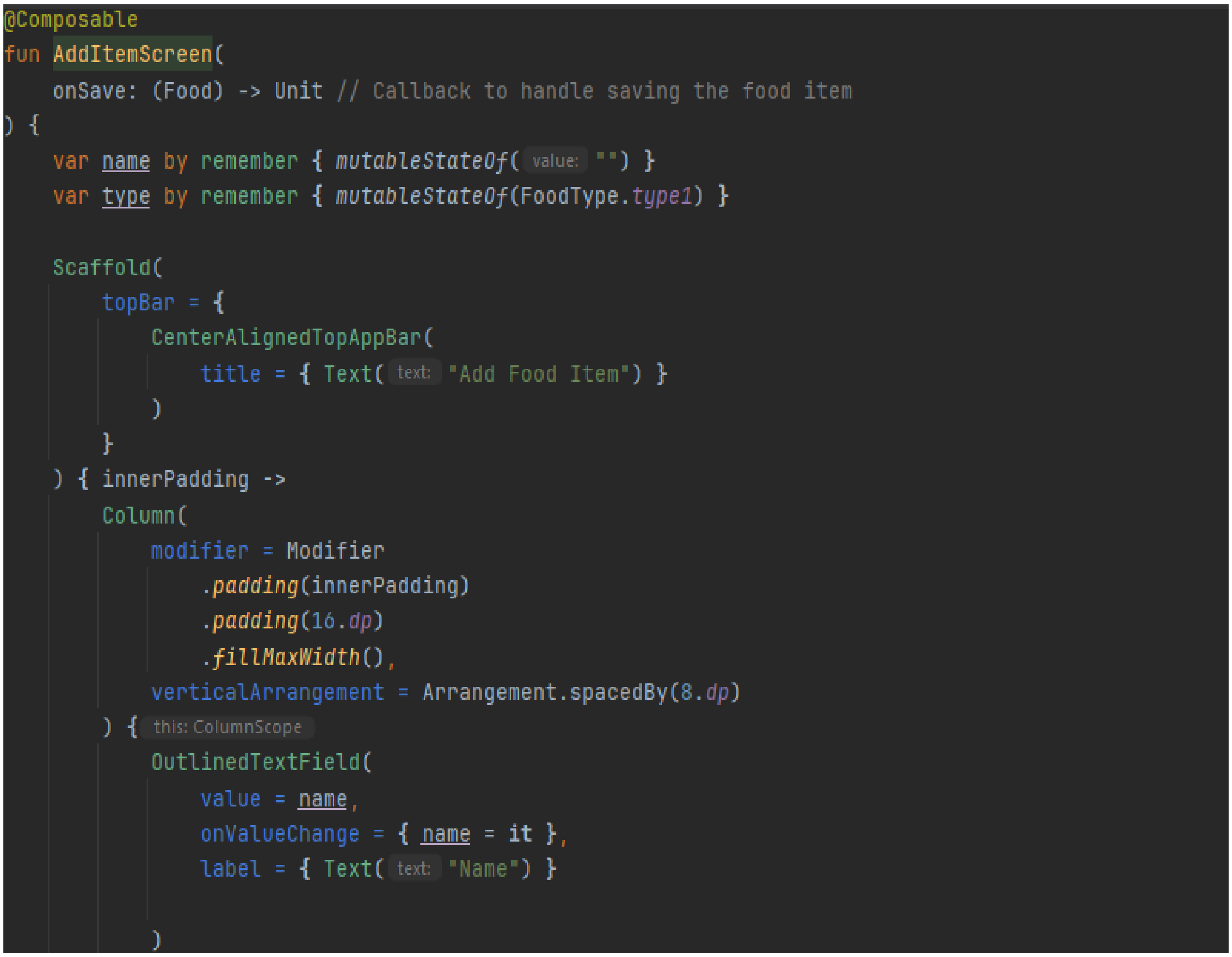
It is essential to have a storage to store and retrieve all the data used in the our app.While there are several ways for storing the data,the use of a file is one basic and easy way to do so.

Functions can be used to store and retrieve the data from these files.Such function is expected to perform the following:

- This function takes the context (the activity state), a object, and a filename as arguments.

- It creates a file in the application's internal storage directory with the provided filename.
- It formats the food information into a comma-separated string (CSV format).
- It uses FileOutputStream to write the formatted string to the file in append mode (meaning it adds new data without erasing existing content).

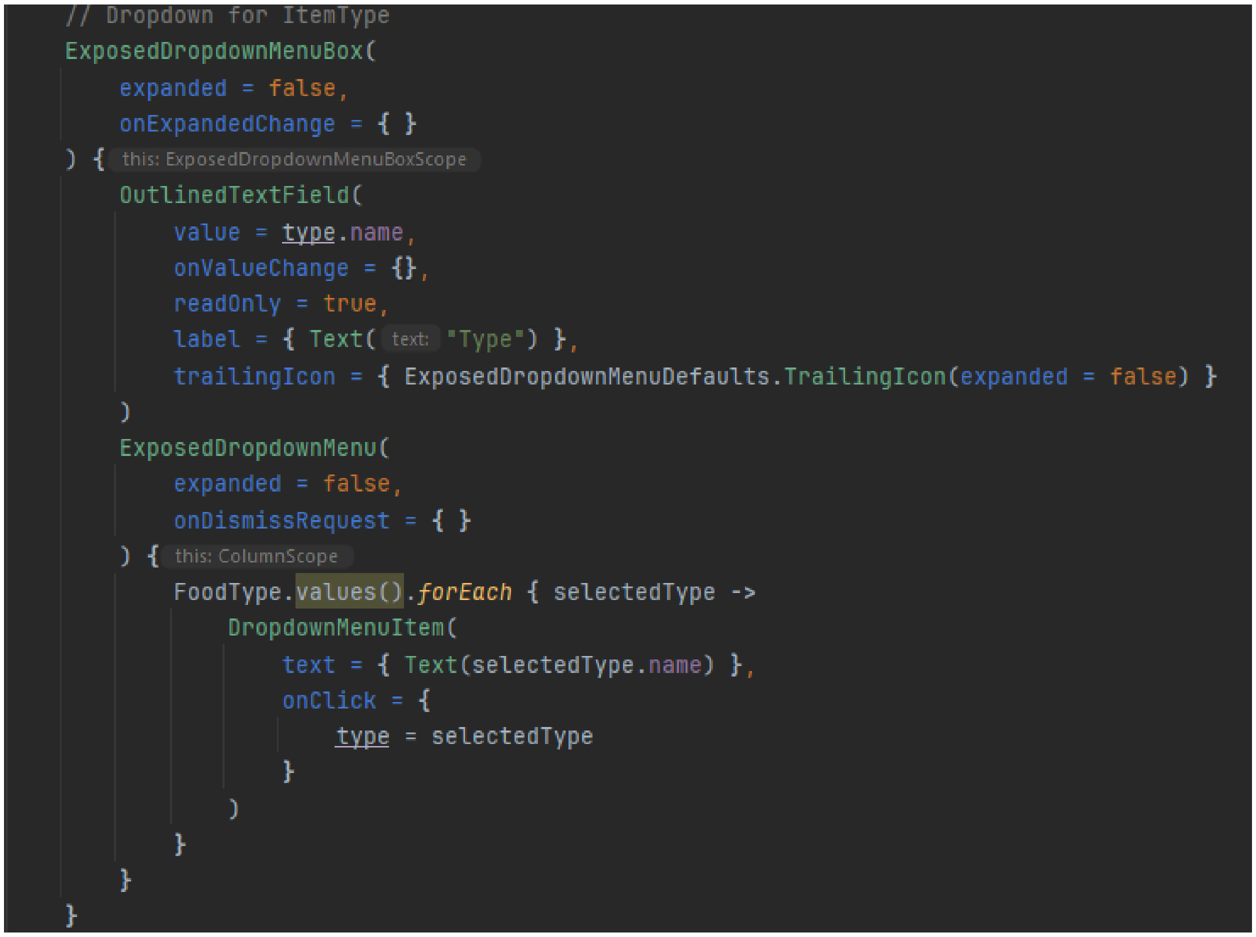
AddItem Screen



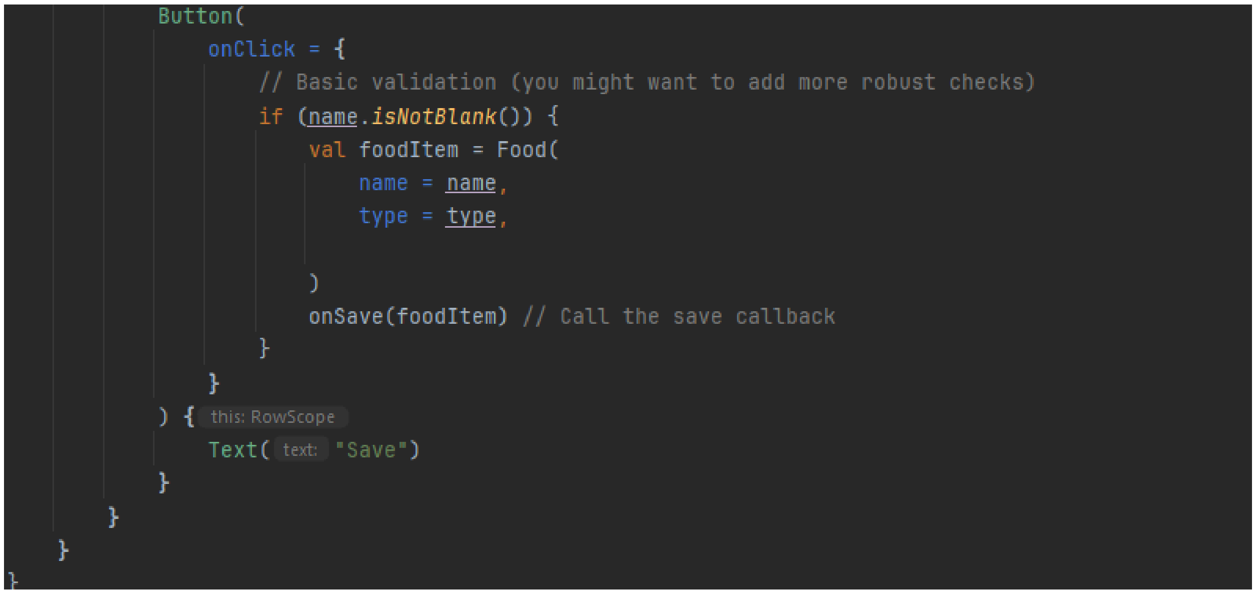
```kotlin
@Composable
fun AddItemScreen(
    onSave: (Food) -> Unit // Callback to handle saving the food item
) {
    var name by remember { mutableStateOf( value: "") }
    var type by remember { mutableStateOf(FoodType.type1) }

    Scaffold(
        topBar = {
            CenterAlignedTopAppBar(
                title = { Text( text: "Add Food Item") }
            )
        }
    ) { innerPadding ->
        Column(
            modifier = Modifier
                .padding(innerPadding)
                .padding(16.dp)
                .fillMaxWidth(),
            verticalArrangement = Arrangement.spacedBy(8.dp)
        ) { this: ColumnScope
            OutlinedTextField(
                value = name,
                onValueChange = { name = it },
                label = { Text( text: "Name") }

            )
```



```
    // Dropdown for ItemType
    ExposedDropdownMenuBox(
        expanded = false,
        onExpandedChange = { }
    ) { this: ExposedDropdownMenuBoxScope
        OutlinedTextField(
            value = type.name,
            onValueChange = {},
            readOnly = true,
            label = { Text( text: "Type") },
            trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = false) }
        )
        ExposedDropdownMenu(
            expanded = false,
            onDismissRequest = { }
        ) { this: ColumnScope
            FoodType.values().forEach { selectedType ->
                DropdownMenuItem(
                    text = { Text(selectedType.name) },
                    onClick = {
                        type = selectedType
                    }
                )
            }
        }
    }
```

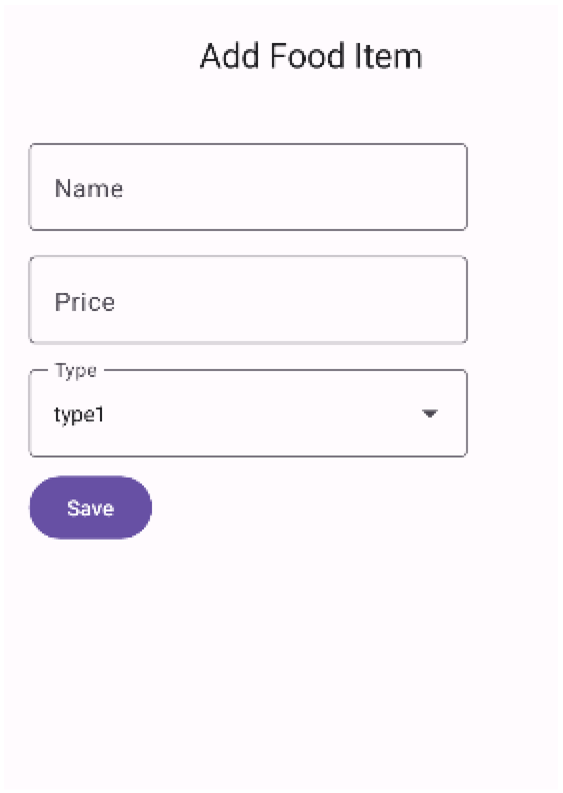


```
            Button(
                onClick = {
                    // Basic validation (you might want to add more robust checks)
                    if (name.isNotBlank()) {
                        val foodItem = Food(
                            name = name,
                            type = type,

                        )
                        onSave(foodItem) // Call the save callback
                    }
                }
            ) { this: RowScope
                Text( text: "Save")
            }
        }
    }
}
```

Do note that the above code snippet is not the solution to the task this week.This is just how the screen components work.The resources required to build the expected screen and given at the end of the document.Also remember to use the reference for creating the functions to store and retrieve the data of the application.

The above snippet will result in such a screen:



You can be as creative as you want in terms with the design of every screen in the app.

Links for Reference:

1. **Click here for reference for handling user inputs**
2. **Click here for reference for file storage**
3. **Articles for AddItem Screen**