Mobile Apps 2

Assignment Four

Submission Date 01/10/2024

Percent 10%

Include all workings and references

Section A.

"Data Validation can be provided by both in-built Compose functionality and program-specific code"

1. Discuss, giving examples of how each can be applied (*3 marks*)

Built-in Compose Functionality: Jetpack Compose provides TextField composables with built-in validation capabilities. For example, TextField offers properties like isError and visualTransformation to provide basic validation feedback directly within the UI.

Example: You can use the isError parameter in a TextField to display an error state when the input doesn't meet certain criteria. For example, if a user enters an invalid email, you can set isError = true, which will display the TextField with a red underline.

```
var email by remember { mutableStateOf("") }
val isValidEmail = android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()

TextField (
        value = email,
        onValueChange = { email = it },
        label = { Text ("Email") },
        isError = !isValidEmail
)
```

Program-specific Code: For more complex validation, custom logic can be implemented in the ViewModel or a validation function. This allows for more flexible checks and can support multi-field validation.

Example: If the form requires checking whether multiple fields (like username and password) are non-empty before submission, a custom validation function can be used in the ViewModel to check each field's state and return the final validation result.

```
fun validateForm(username: String, password: String): Boolean {
        return username.isNotBlank() && password.length >= 8
}
```

2. The state of a Composable can be tracked by various means. Give two examples of this. (*2 marks*)

In Jetpack Compose, tracking the state of a Composable can be achieved through different mechanisms:

- Using remember and mutableStateOf: This approach allows you to track simple UI state within a Composable. The remember function retains the state across recompositions, while mutableStateOf enables the state to be observable, so the UI re-renders whenever the state changes.

Example: A counter value can be tracked and updated using remember and mutableStateOf within a Composable.

```
var count by remember { mutableStateOf(0) }

Button(onClick = { count++ }) {

        Text("Count: $count")

}
```

- Using a ViewModel: For more complex or shared state management across multiple Composables, a ViewModel can be used. The state is kept in the ViewModel, allowing it to survive configuration changes and be shared across screens.

Example: A login form with multiple input fields can have its state managed by a ViewModel, which holds properties for each field. The Composable then observes these properties using LiveData or StateFlow to update UI as the ViewModel state changes.

```
class LoginViewModel : ViewModel() {

        var username by mutableStateOf("")

        var password by mutableStateOf("")

}
```

(5 marks)

Section B.

You are now going to complete the front end of the app that you started. The database will be populated next week.

The first page of the app will allow the user to enter their location and dates to and from the job in hand. If a decorator is available for those dates, they will be added to a list under the search window. If the user wishes to book the decorator (based on a criteria of your choice), a new screen will then be shown with the contact details of the relevant decorator.

Based on mock data, build the relevant UIs, with some notion of navigation (2 marks)

Include some level of validation (1 marks)

Distinctly mark where State has been used to reflect the change of a composable, using two different techniques (2 marks)

```
@Composable
// JobApp is the main entry point for the application
fun JobApp(viewModel: DecoratorViewModel) {
    // NavController to handle navigation
    val navController = rememberNavController()
    // NavHost for the application
    NavHost(navController, startDestination = "search") { this: NavGraphBuilder
        // Composable for the search screen
        composable( route: "search") { this: AnimatedContentScope  it: NavBackStackEntry
            SearchScreen(viewModel = viewModel, onSelectDecorator = { decorator ->
                navController.currentBackStackEntry?.savedStateHandle?.set("selectedDecorator", decorator)
                navController.navigate( route: "booking")
            })
        }
        // Composable for the decorator details screen
        composable( route: "booking") { this: AnimatedContentScope  it: NavBackStackEntry
            val decorator = navController.previousBackStackEntry?.savedStateHandle?.get<Decorator>("selectedDecorator")
            if (decorator != null) {
                DecoratorScreen(decorator = decorator, onBack = { navController.popBackStack() })
            } else {
                // Handle case when decorator is null
                navController.currentBackStackEntry
            }
        }
    }
}
```

```
// Initialize the ViewModel
viewModel = ViewModelProvider(
    owner: this,
    DecoratorViewModelFactory(application)
)[DecoratorViewModel::class.java]

// Add mock data to the ViewModel
val mockDecorators = listOf(
    Decorator(name = "John Smith", location = "Dublin", contactInfo = "johnsmith@example.com", availableFrom = "25-10-2024", availableTo = "10-11-2024"),
    Decorator(name = "Dan Morley", location = "Cork", contactInfo = "danmorley@example.com", availableFrom = "01-11-2024", availableTo = "20-11-2024"),
    Decorator(name = "Filipe Lutz", location = "Torres - RS, Brazil", contactInfo = "filipelutz@example.com", availableFrom = "15-11-2024", availableTo = "05-12-2024"),
    Decorator(name = "Rebecca", location = "Dublin", contactInfo = "rebecca@example.com", availableFrom = "25-11-2024", availableTo = "15-12-2024"),
    Decorator(name = "Alice Johnson", location = "Galway", contactInfo = "alice.johnson@example.com", availableFrom = "30-10-2024", availableTo = "15-11-2024"),
    Decorator(name = "Mark Thompson", location = "Limerick", contactInfo = "mark.thompson@example.com", availableFrom = "05-11-2024", availableTo = "25-11-2024"),
    Decorator(name = "Sophie Turner", location = "Belfast", contactInfo = "sophie.turner@example.com", availableFrom = "10-11-2024", availableTo = "01-12-2024"),
    Decorator(name = "Chris Evans", location = "Dublin", contactInfo = "chris.evans@example.com", availableFrom = "28-10-2024", availableTo = "18-11-2024"),
    Decorator(name = "Liam Neeson", location = "Waterford", contactInfo = "liam.neeson@example.com", availableFrom = "12-11-2024", availableTo = "30-11-2024"),
    Decorator(name = "Emma Watson", location = "Dublin", contactInfo = "emma.watson@example.com", availableFrom = "20-11-2024", availableTo = "10-12-2024"),
    Decorator(name = "David Beckham", location = "Cork", contactInfo = "david.beckham@example.com", availableFrom = "01-11-2024", availableTo = "15-11-2024"),
    Decorator(name = "Olivia Brown", location = "Derry", contactInfo = "olivia.brown@example.com", availableFrom = "25-11-2024", availableTo = "20-12-2024"),
    Decorator(name = "Tom Cruise", location = "Dublin", contactInfo = "tomcruise@example.com", availableFrom = "05-11-2024", availableTo = "25-11-2024")
)
// Add mock data to the ViewModel
viewModel.addMockData(mockDecorators)
```

```
class MainActivity : ComponentActivity() {
    private lateinit var viewModel: DecoratorViewModel
    // onCreate method
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Initialize the ViewModel
        viewModel = ViewModelProvider(
            owner: this,
            DecoratorViewModelFactory(application)
        )[DecoratorViewModel::class.java]
```

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SearchScreen(viewModel: DecoratorViewModel, onSelectDecorator: (Decorator) -> Unit) {
    // State variables to hold user input and button click states
    var location by rememberSaveable { mutableStateOf( value: "") }
    var dateFrom by rememberSaveable { mutableStateOf( value: "") }
    var dateTo by rememberSaveable { mutableStateOf( value: "") }
    var isSearchClicked by rememberSaveable { mutableStateOf( value: false) }
    var isAllDecoratorsClicked by rememberSaveable { mutableStateOf( value: false) }

    // Collecting available decorators from the ViewModel
    val availableDecorators by viewModel.availableDecorators.collectAsState()
```

(5 marks)

References

- **Save UI state in Compose**
- **Validate input as the user types**
- **Validate Forms with Clean Architecture**
- **ViewModels & Configuration Changes**
- **StateFlow and SharedFlow**
- **GitHub Repository**