NANYANG POLYTECHNIC

School of Information Technology

Course:	Diploma in Information Technology
Module:	IT2161 Mobile Applications Development

View Model

Preparing the environment

1. Open up the previous MessageMe project.

Register View Model

2. Create a new package called “data” under “com.it2161.messageme”.
3. In the data package, create a new class RegisterUIState

```
data class RegisterUIState(
    val username: String = "",
    val password: String = "",
    val gender: String = ""
)
```

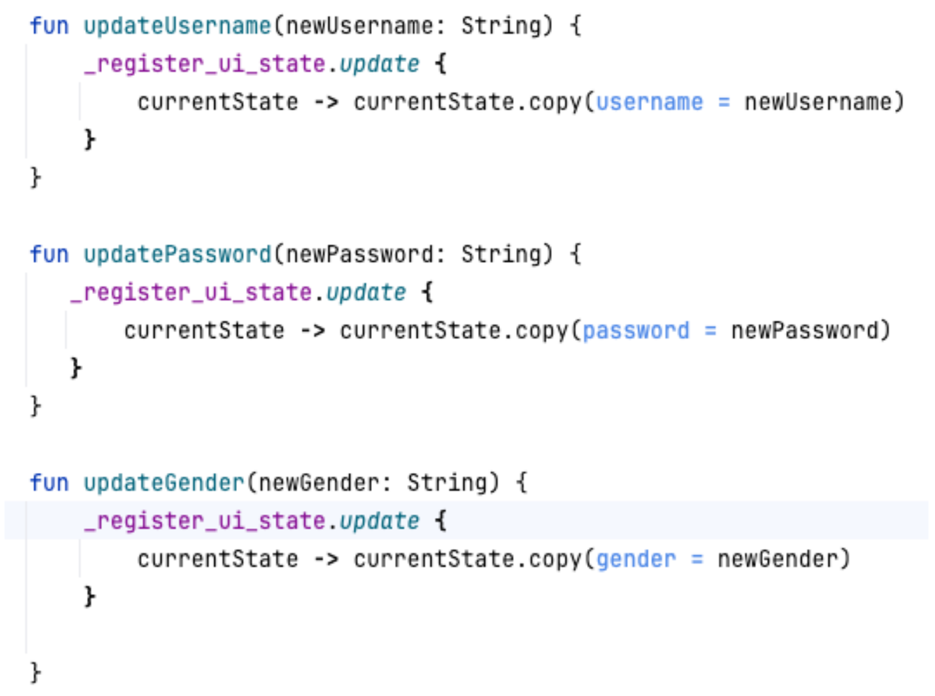
4. Inside components, create a new class RegisterViewModel. Modify the class to extend ViewModel().
5. Create the follow variables:
   - a private a mutable state of RegisterUIState.
   - a stateflow variable using RegisterUIState

```
class RegisterViewModel : ViewModel(){

    private val _register_ui_state = MutableStateFlow(RegisterUIState())
    val register_ui_state: StateFlow<RegisterUIState> = _register_ui_state.asStateFlow()

}
```

6. Add in the following functions to init and reset the view model.
   - Init
   - Reset()

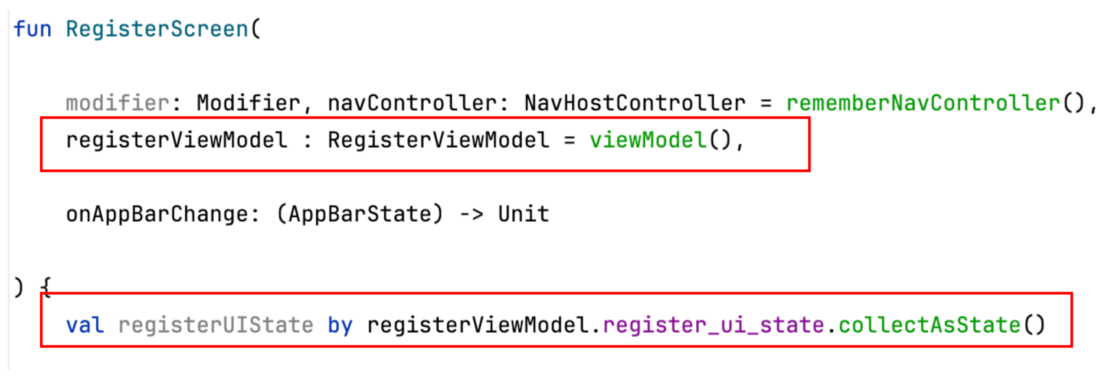
```
init {
    reset()
}

fun reset() {
    _register_ui_state.value = RegisterUIState()
}
```

7. Add in the following functions to update the variables binded to the view model.
    - updateUsername
    - updatePassword
    - updateGender



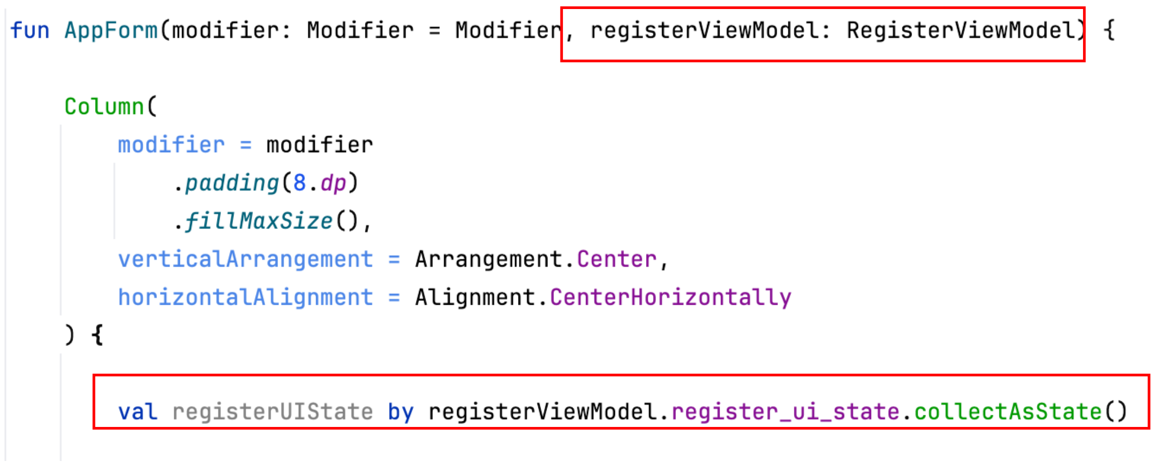
```kotlin
fun updateUsername(newUsername: String) {
    _register_ui_state.update {
        currentState -> currentState.copy(username = newUsername)
    }
}

fun updatePassword(newPassword: String) {
   _register_ui_state.update {
       currentState -> currentState.copy(password = newPassword)
   }
}

fun updateGender(newGender: String) {
    _register_ui_state.update {
        currentState -> currentState.copy(gender = newGender)
    }

}
```

8. Update RegisterScreen function to have an additional “RegisterViewModel” parameter.
9. In RegisterScreen function, add a new variable “registerUIState”. Use the by delegate and call collectAsState on uiState



```kotlin
fun RegisterScreen(

    modifier: Modifier, navController: NavHostController = rememberNavController(),
    registerViewModel : RegisterViewModel = viewModel(),

    onAppBarChange: (AppBarState) -> Unit

) {
    val registerUIState by registerViewModel.register_ui_state.collectAsState()
```

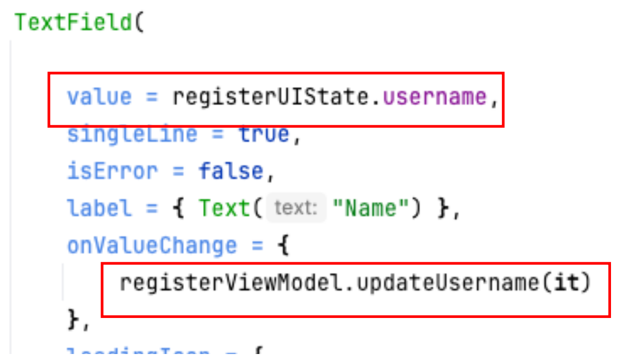
10. Do the same for AppForm function.



```
fun AppForm(modifier: Modifier = Modifier, registerViewModel: RegisterViewModel) {

    Column(
        modifier = modifier
            .padding(8.dp)
            .fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

        val registerUIState by registerViewModel.register_ui_state.collectAsState()
```

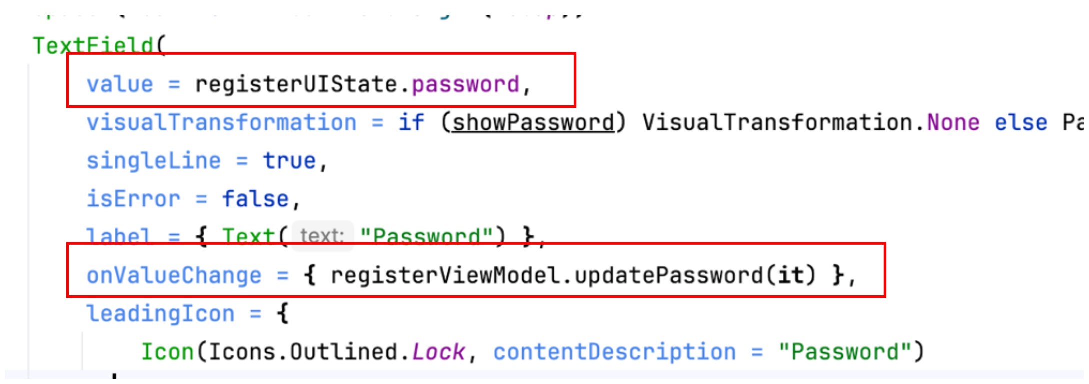
11. Update AppForm function to make use of registerUIState to reflect the values of the variable.

User name TextField



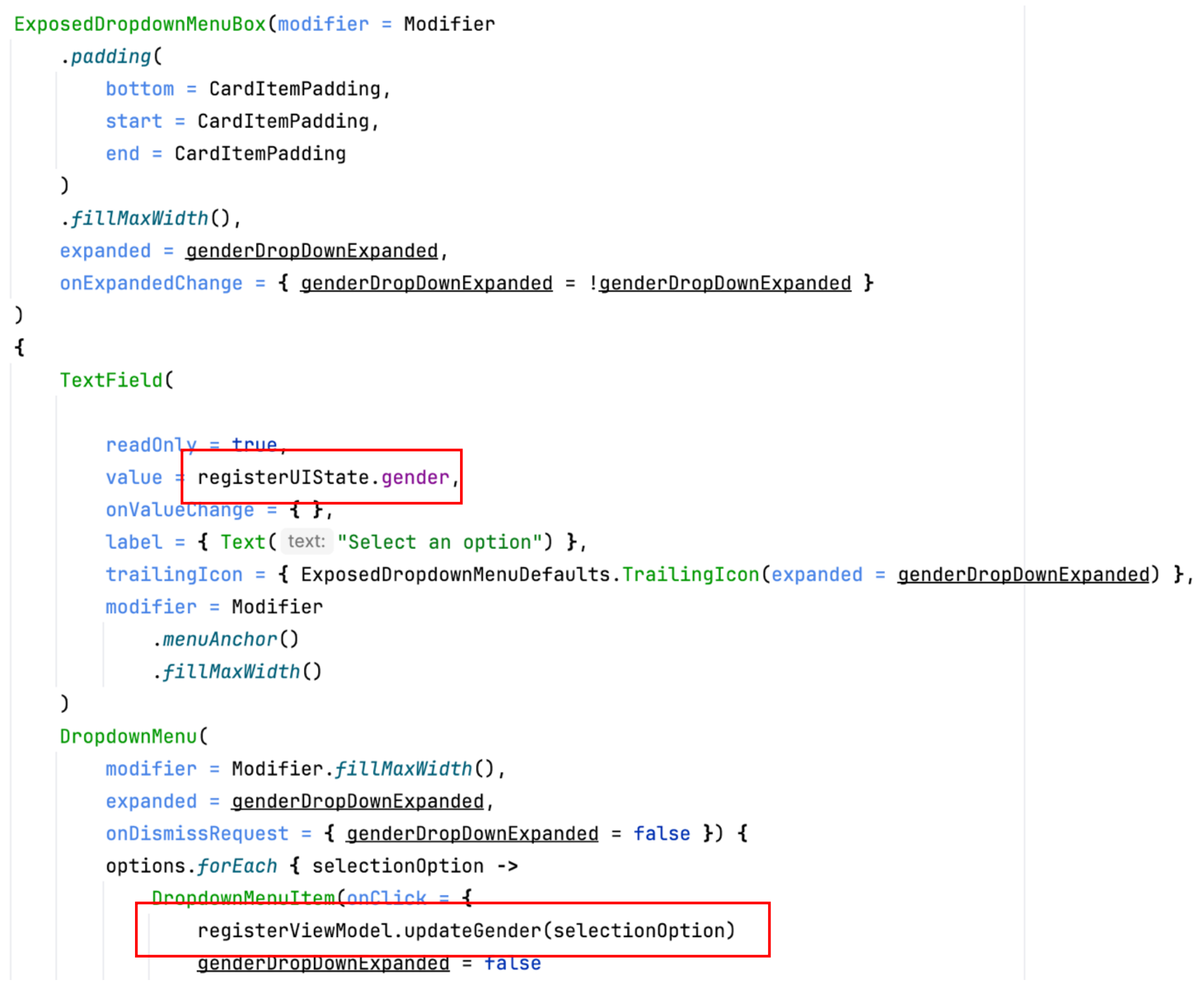
```
TextField(

    value = registerUIState.username,
    singleLine = true,
    isError = false,
    label = { Text( text: "Name") },
    onValueChange = {
        registerViewModel.updateUsername(it)
    },
    leadingIcon = {
```

Password TextField



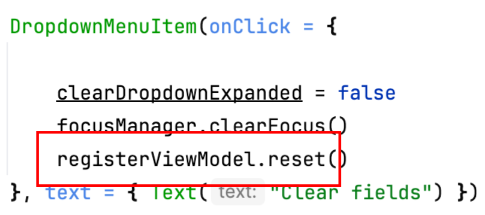
```
TextField(
    value = registerUIState.password,
    visualTransformation = if (showPassword) VisualTransformation.None else Pa
    singleLine = true,
    isError = false,
    label = { Text( text: "Password") },
    onValueChange = { registerViewModel.updatePassword(it) },
    leadingIcon = {
        Icon(Icons.Outlined.Lock, contentDescription = "Password")
```

Gender Dropdown menu



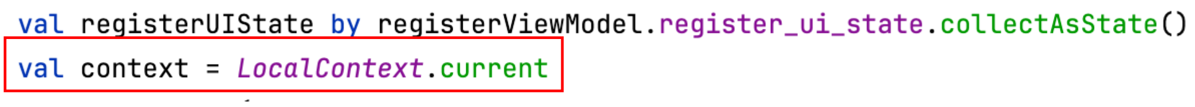
```
ExposedDropdownMenuBox(modifier = Modifier
    .padding(
        bottom = CardItemPadding,
        start = CardItemPadding,
        end = CardItemPadding
    )
    .fillMaxWidth(),
    expanded = genderDropDownExpanded,
    onExpandedChange = { genderDropDownExpanded = !genderDropDownExpanded }
)
{
    TextField(

        readOnly = true,
        value = registerUIState.gender,
        onValueChange = { },
        label = { Text( text: "Select an option") },
        trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = genderDropDownExpanded) },
        modifier = Modifier
            .menuAnchor()
            .fillMaxWidth()
    )
    DropdownMenu(
        modifier = Modifier.fillMaxWidth(),
        expanded = genderDropDownExpanded,
        onDismissRequest = { genderDropDownExpanded = false }) {
        options.forEach { selectionOption ->
            DropdownMenuItem(onClick = {
                registerViewModel.updateGender(selectionOption)
                genderDropDownExpanded = false
```

Reset Text



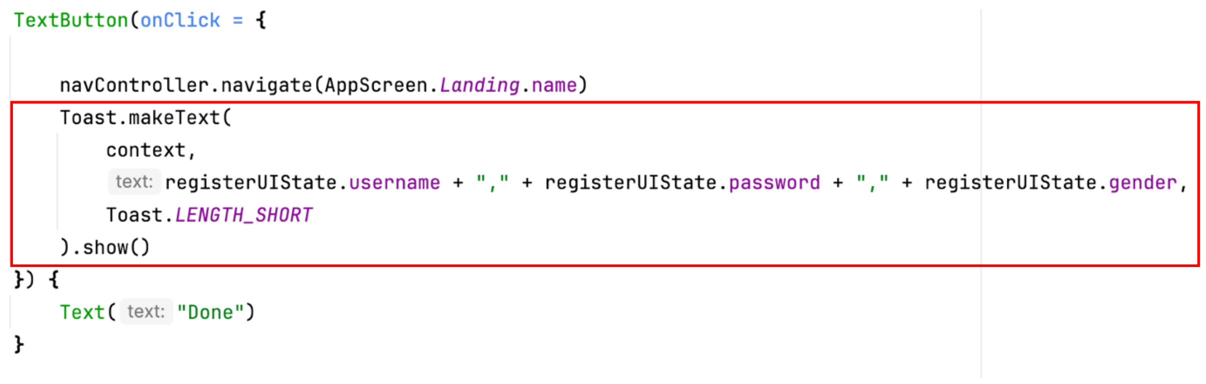
```
DropdownMenuItem(onClick = {

    clearDropdownExpanded = false
    focusManager.clearFocus()
    registerViewModel.reset()
}, text = { Text( text: "Clear fields") })
```

12. Back to RegisterScreen function, add in the context variable



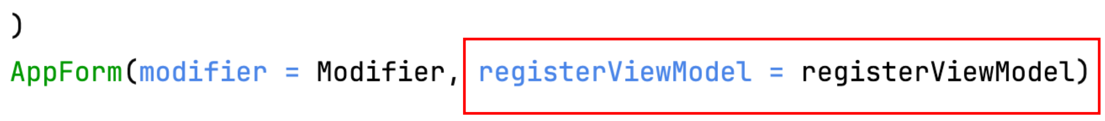
```
val registerUIState by registerViewModel.register_ui_state.collectAsState()
val context = LocalContext.current
```

13. Add in the Toast when user tap on the tap action button



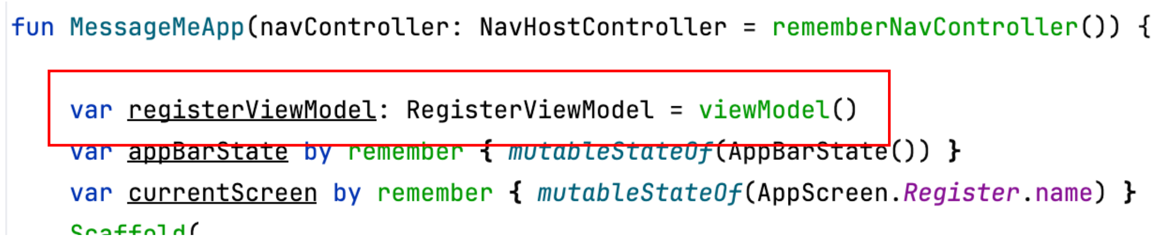
```
TextButton(onClick = {

    navController.navigate(AppScreen.Landing.name)
    Toast.makeText(
        context,
        text: registerUIState.username + "," + registerUIState.password + "," + registerUIState.gender,
        Toast.LENGTH_SHORT
    ).show()
}) {
    Text( text: "Done")
}
```

14. At the end of RegisterScreen function, pass registerViewModel to AppForm.



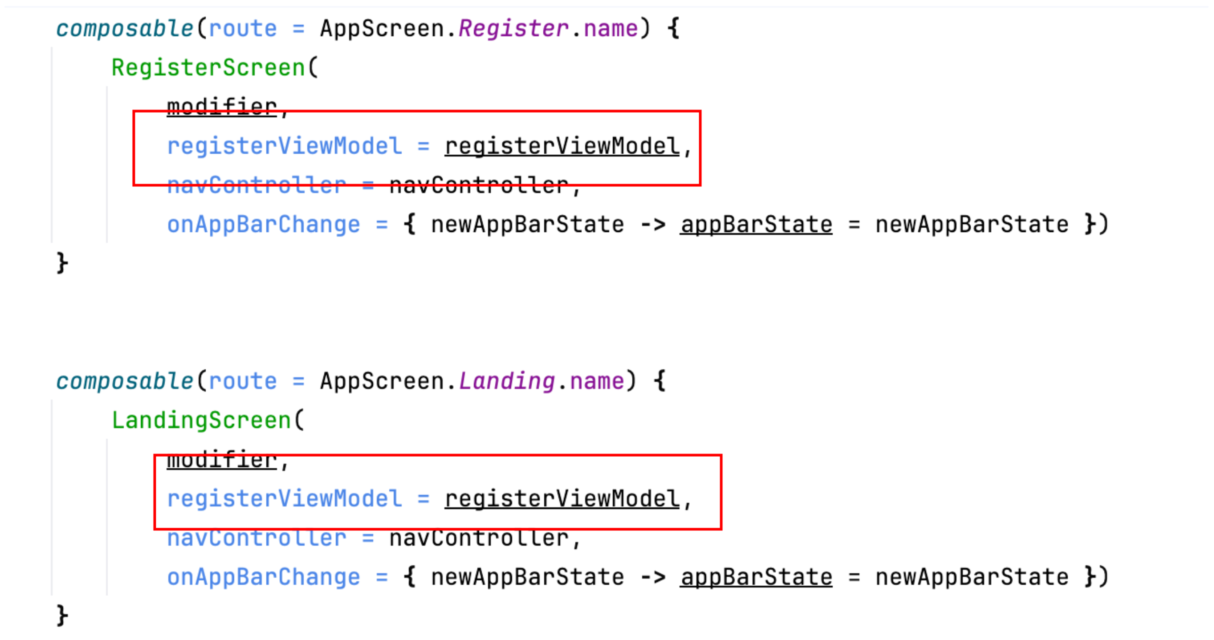
```
)
AppForm(modifier = Modifier, registerViewModel = registerViewModel)
```

15. In MessageMeApp.kt, update MessageMeApp function to initialize the viewModel.



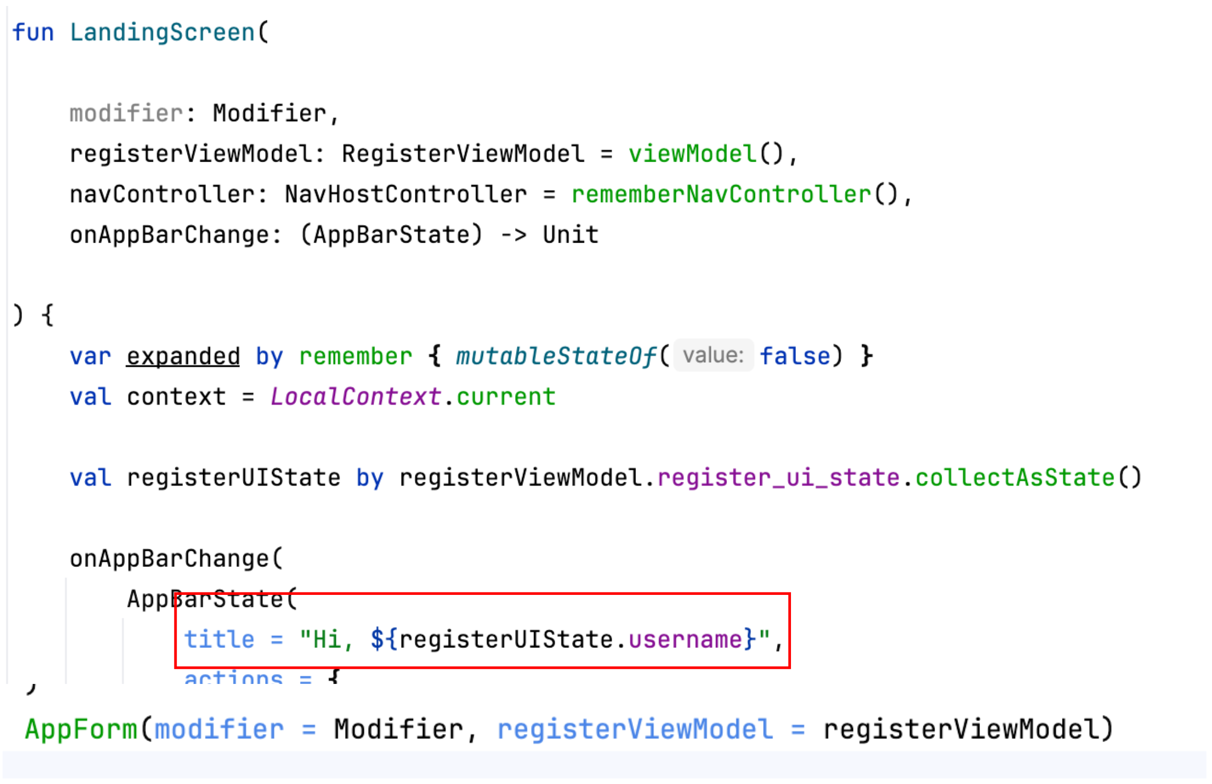
```
fun MessageMeApp(navController: NavHostController = rememberNavController()) {

    var registerViewModel: RegisterViewModel = viewModel()
    var appBarState by remember { mutableStateOf(AppBarState()) }
    var currentScreen by remember { mutableStateOf(AppScreen.Register.name) }
    Scaffold(
```

16. In MessageMeApp function to update NavHost to call LandingScreen and RegisterScreen with the viewModel as an argument.



```
composable(route = AppScreen.Register.name) {
    RegisterScreen(
        modifier,
        registerViewModel = registerViewModel,
        navController = navController,
        onAppBarChange = { newAppBarState -> appBarState = newAppBarState })
}


composable(route = AppScreen.Landing.name) {
    LandingScreen(
        modifier,
        registerViewModel = registerViewModel,
        navController = navController,
        onAppBarChange = { newAppBarState -> appBarState = newAppBarState })
}
```

17. In LandingScreen function to update NavHost to call LandingScreen and RegisterScreen with the viewModel as an argument.
18. Make use of the viewModel to change the title of the AppBar



```
fun LandingScreen(

    modifier: Modifier,
    registerViewModel: RegisterViewModel = viewModel(),
    navController: NavHostController = rememberNavController(),
    onAppBarChange: (AppBarState) -> Unit

) {
    var expanded by remember { mutableStateOf( value: false) }
    val context = LocalContext.current

    val registerUIState by registerViewModel.register_ui_state.collectAsState()

    onAppBarChange(
        AppBarState(
            title = "Hi, ${registerUIState.username}",
            actions = {
```

```
AppForm(modifier = Modifier, registerViewModel = registerViewModel)
```