NANYANG POLYTECHNIC

School of Information Technology

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

List & Dropdowns

Preparing the environment

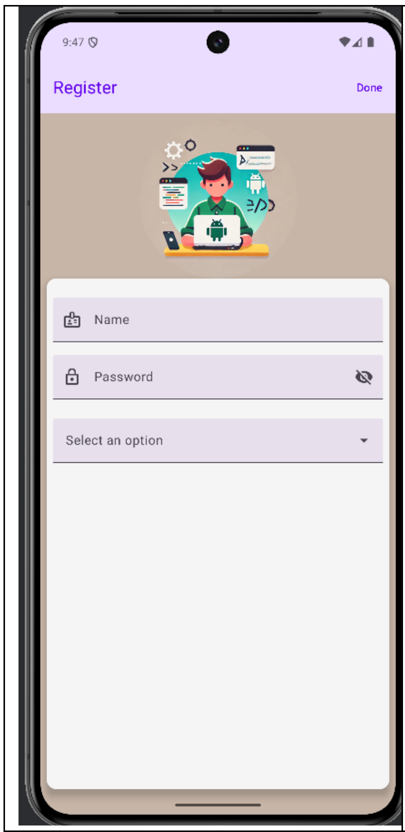
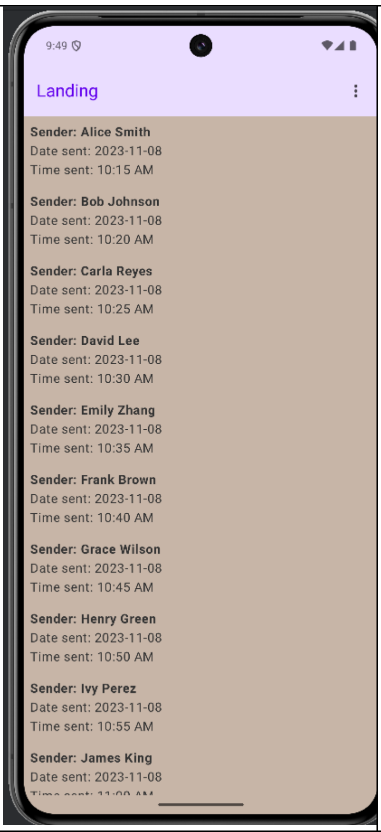
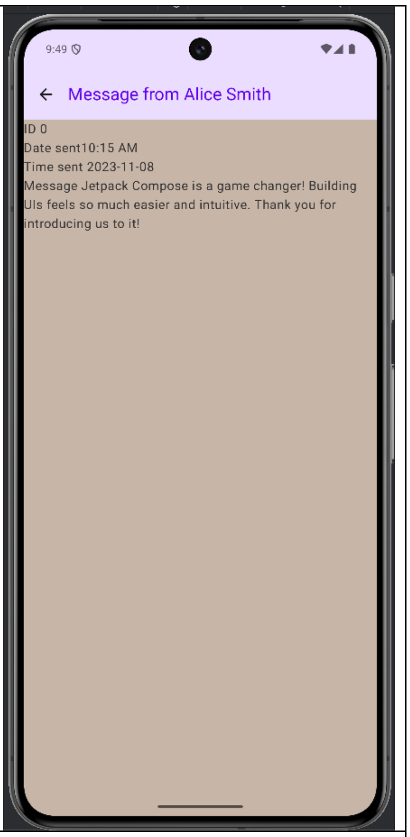
Open up the starter project.
You should see the following
2 files under the package com.it2161.message

- MainActivity
- MessageMeApp.kt

3 Kotlin class files under the package com.it2161.messageme.components.

- LandingScreen.kt
- MessageDetailScreen.kt
- RegisterScreen.kt

Screens

		
RegisterScreen.kt	LandingScreen.kt	MessageDetailScreen.kit

Sender: Alice Smith
Date sent: 2023-11-08
Time sent: 10:15 AM

Sender: Bob Johnson
Date sent: 2023-11-08
Time sent: 10:20 AM

Sender: Carla Reyes
Date sent: 2023-11-08
Time sent: 10:25 AM

Sender: David Lee
Date sent: 2023-11-08
Time sent: 10:30 AM

Sender: Emily Zhang
Date sent: 2023-11-08
Time sent: 10:35 AM

Sender: Frank Brown
Date sent: 2023-11-08
Time sent: 10:40 AM

Sender: Grace Wilson
Date sent: 2023-11-08
Time sent: 10:45 AM

Sender: Henry Green
Date sent: 2023-11-08
Time sent: 10:50 AM

Sender: Ivy Perez
Date sent: 2023-11-08
Time sent: 10:55 AM

Sender: James King
Date sent: 2023-11-08

ID 0
Date sent10:15 AM
Time sent 2023-11-08
Message Jetpack Compose is a game changer! Building UIs feels so much easier and intuitive. Thank you for introducing us to it!

Register Screen

1. Open up RegisterScreen.kt.
2. Create the composable function as show below. It will create the form structure for the screen.

```
@Composable
fun AppForm(modifier: Modifier = Modifier){


}
```

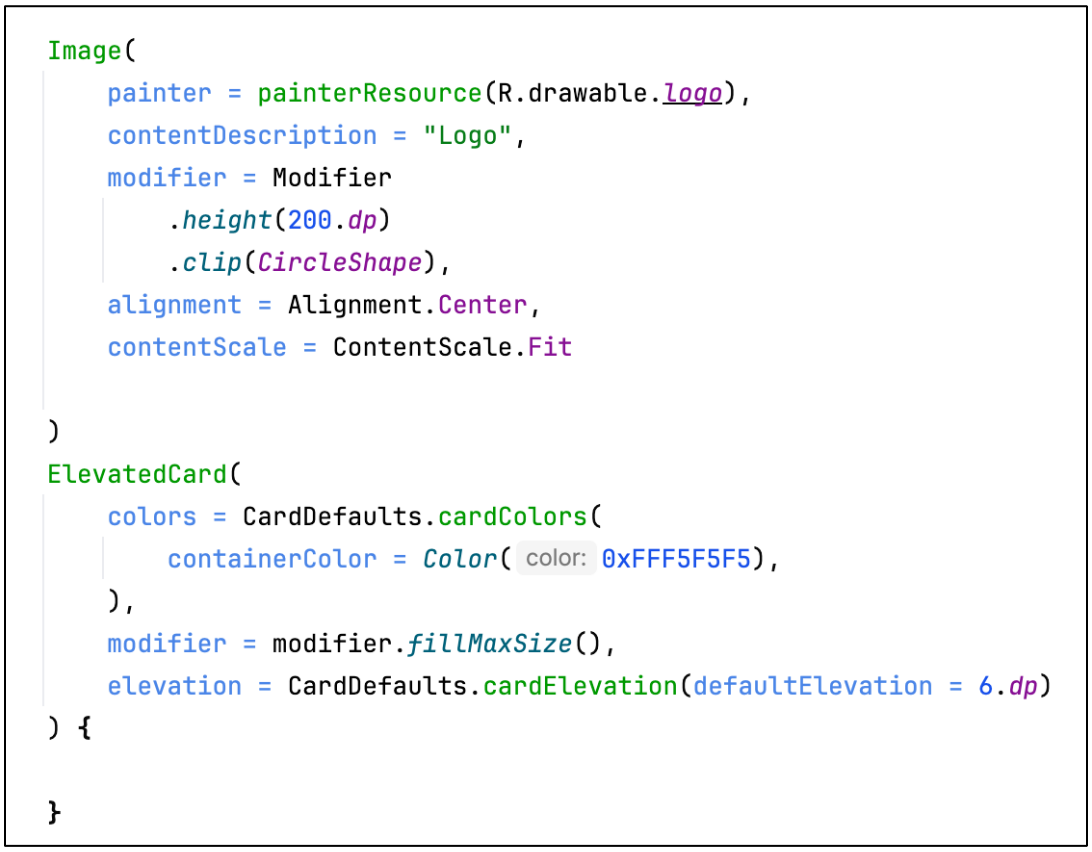
3. Within RegisterScreen function, call the AppForm function.

```
@Composable
fun RegisterScreen() {

    AppForm()
}
```

4. Inside AppForm, add in a column with the modifier as shown below.

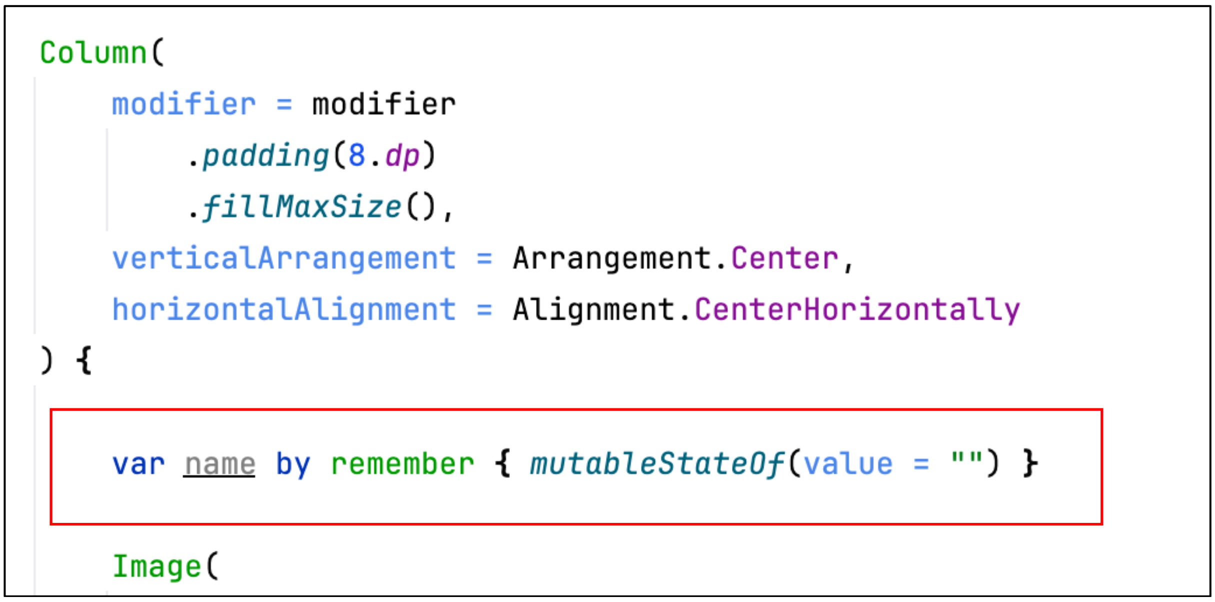
```
@Composable
fun AppForm(modifier: Modifier = Modifier){

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

    }
}
```

5. Add in an Image and ElevatedCard inside the column. The image will display the logo in the resource folder. Make use of the modifier to clip the logo using a circle shape. The Elevated card will hold the controls of the form.



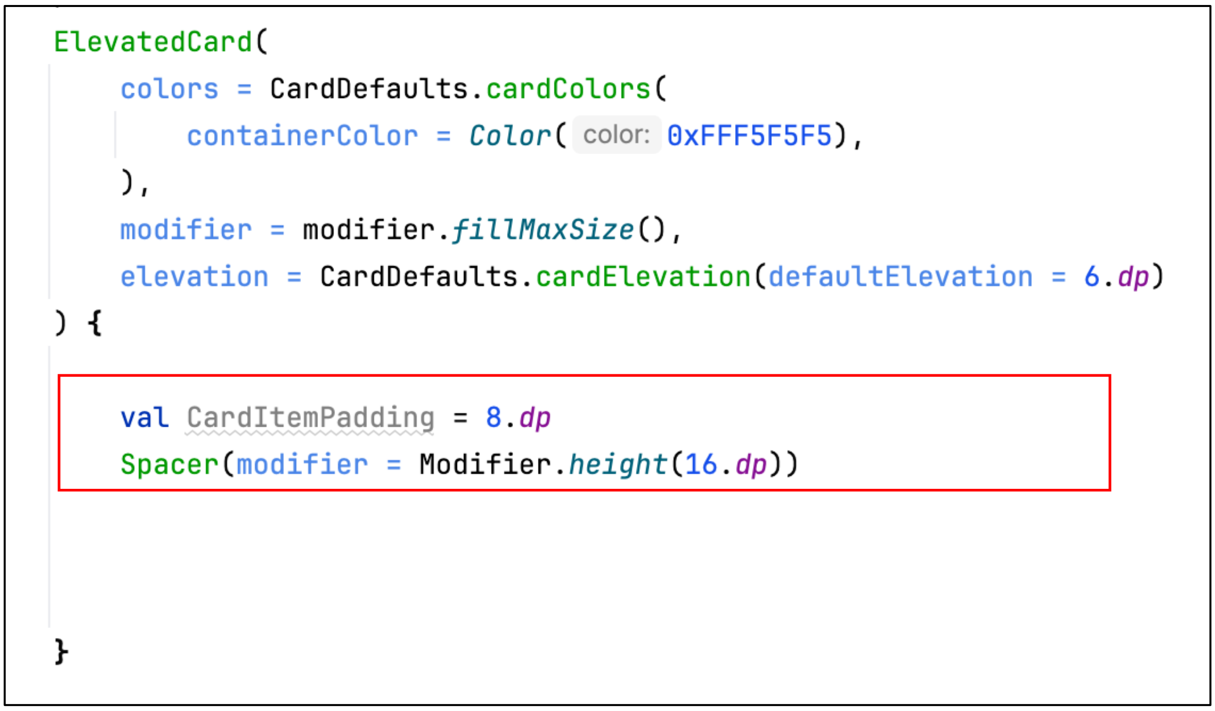
```
Image(
    painter = painterResource(R.drawable.logo),
    contentDescription = "Logo",
    modifier = Modifier
        .height(200.dp)
        .clip(CircleShape),
    alignment = Alignment.Center,
    contentScale = ContentScale.Fit

)
ElevatedCard(
    colors = CardDefaults.cardColors(
        containerColor = Color( color: 0xFFF5F5F5),
    ),
    modifier = modifier.fillMaxSize(),
    elevation = CardDefaults.cardElevation(defaultElevation = 6.dp)
) {

}
```

6. Add in the name variable to hold the user name.



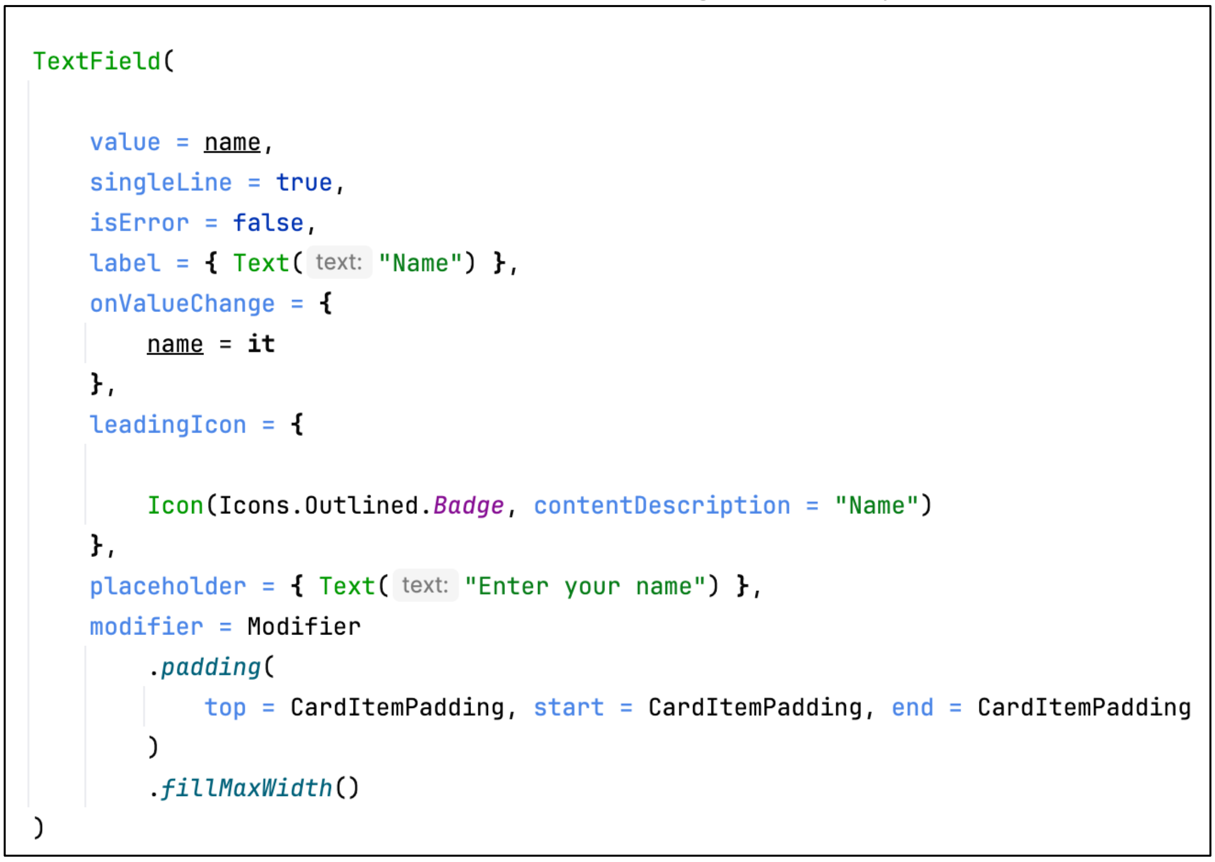
```
Column(
    modifier = modifier
        .padding(8.dp)
        .fillMaxSize(),
    verticalArrangement = Arrangement.Center,
    horizontalAlignment = Alignment.CenterHorizontally
) {

    var name by remember { mutableStateOf(value = "") }

    Image(
```

7. Inside the ElevatedCard, add in a variable to hold a padding for each card item and a Spacer.



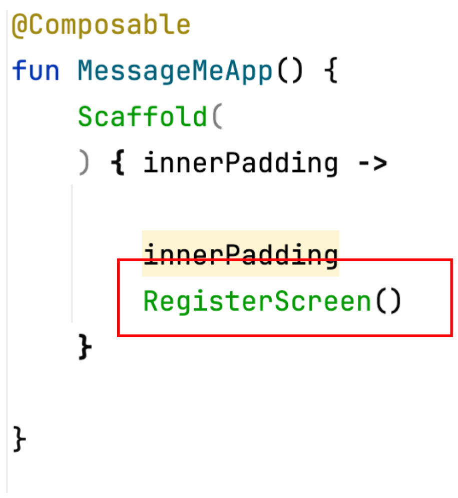
```
ElevatedCard(
    colors = CardDefaults.cardColors(
        containerColor = Color( color: 0xFFF5F5F5),
    ),
    modifier = modifier.fillMaxSize(),
    elevation = CardDefaults.cardElevation(defaultElevation = 6.dp)
) {

    val CardItemPadding = 8.dp
    Spacer(modifier = Modifier.height(16.dp))



}
```

8. Add the TextField using the codes below. It sets the properties of the TextField, which includes a leading icon and placerholder.



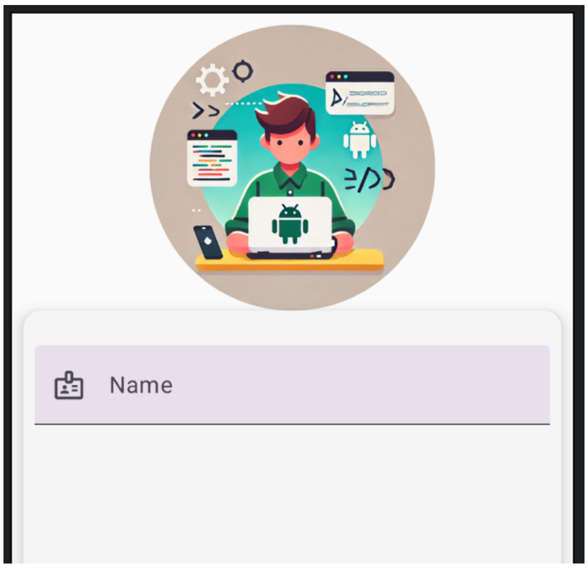
```
TextField(

    value = name,
    singleLine = true,
    isError = false,
    label = { Text( text: "Name") },
    onValueChange = {
        name = it
    },
    leadingIcon = {

        Icon(Icons.Outlined.Badge, contentDescription = "Name")
    },
    placeholder = { Text( text: "Enter your name") },
    modifier = Modifier
        .padding(
            top = CardItemPadding, start = CardItemPadding, end = CardItemPadding
        )
        .fillMaxWidth()
)
```

9. Call RegisterScreen() in MessageMeApp()

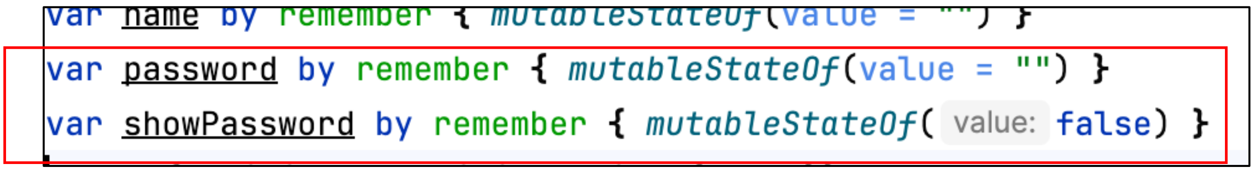


```
@Composable
fun MessageMeApp() {
    Scaffold(
    ) { innerPadding ->

        innerPadding
        RegisterScreen()
    }

}
```

10. Run the application. You should see the text field as shown below.

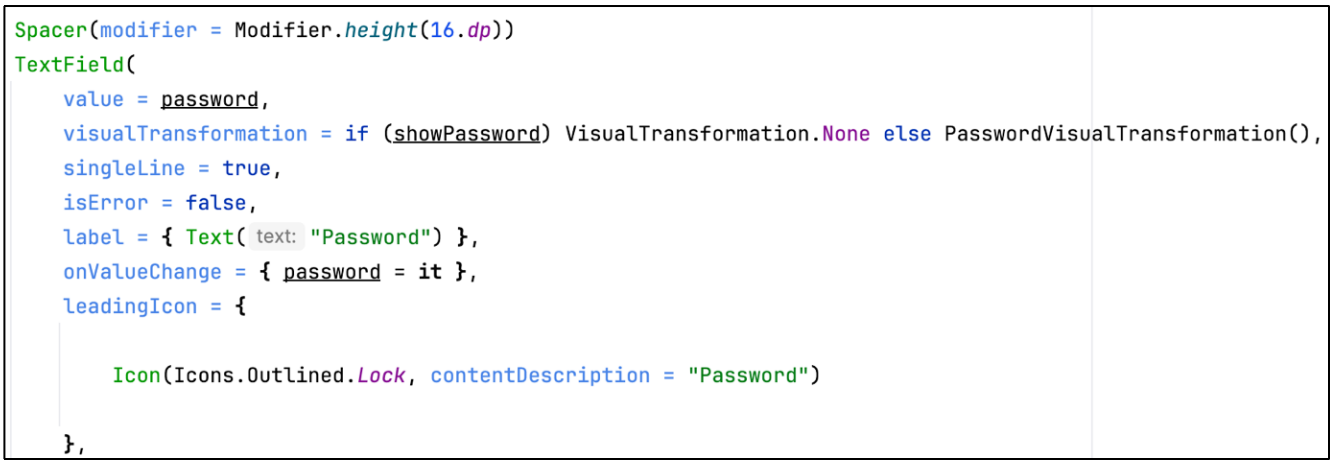


11. Inside the column, add in the state for password.



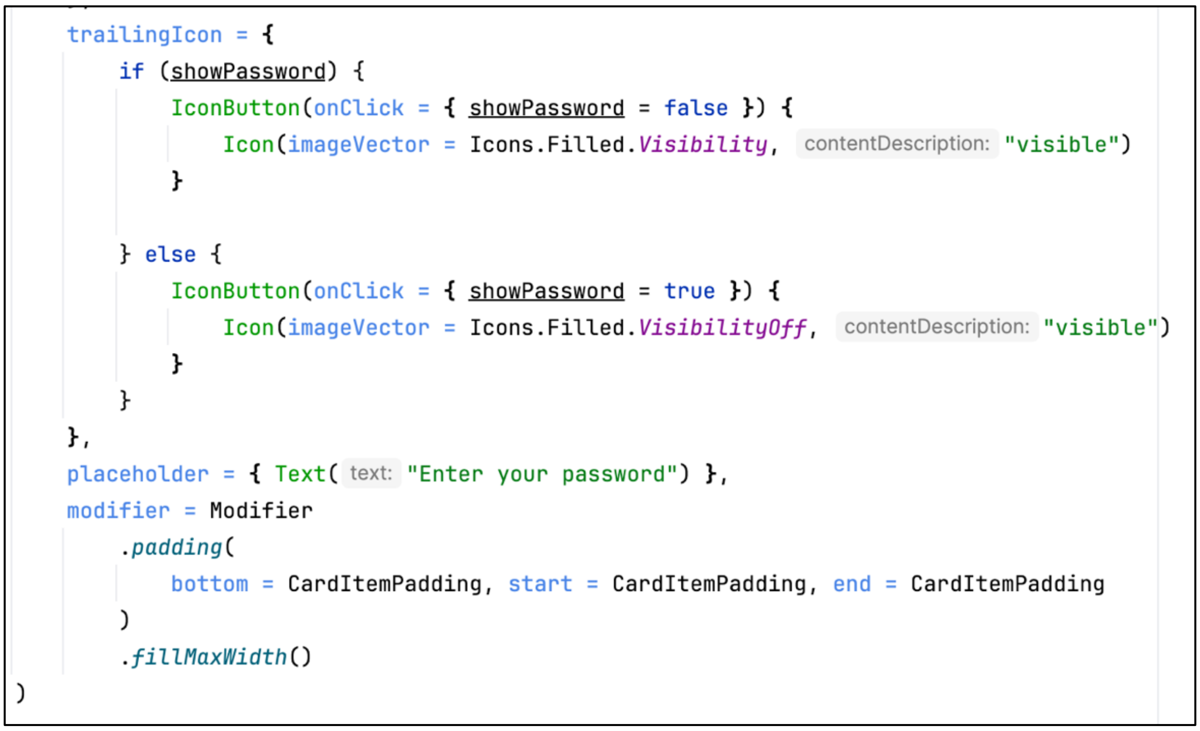
```
var name by remember { mutableStateOf(value = "") }
var password by remember { mutableStateOf(value = "") }
var showPassword by remember { mutableStateOf( value: false) }
```

12. Add in a Spacer and a TextField to allow user to enter a password. The TextField changes the visual of each character entered by the user.



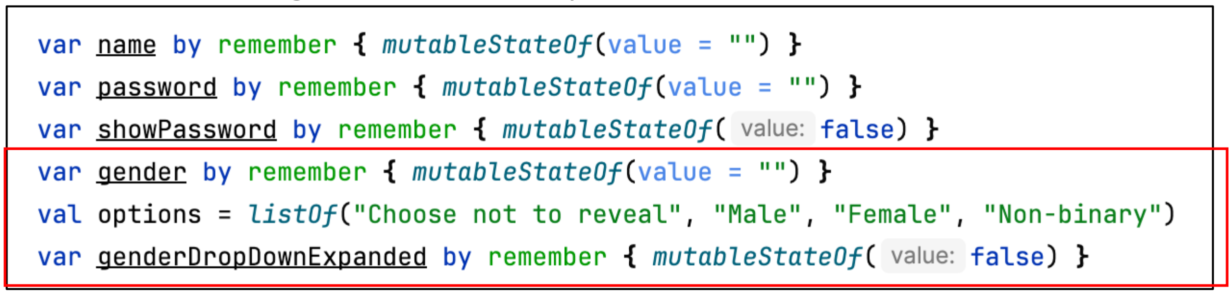
```
Spacer(modifier = Modifier.height(16.dp))
TextField(
    value = password,
    visualTransformation = if (showPassword) VisualTransformation.None else PasswordVisualTransformation(),
    singleLine = true,
    isError = false,
    label = { Text( text: "Password") },
    onValueChange = { password = it },
    leadingIcon = {

        Icon(Icons.Outlined.Lock, contentDescription = "Password")

    },
```

13. Add in the trailing icon for the password TextField. It toggles between the different icons as shown below.



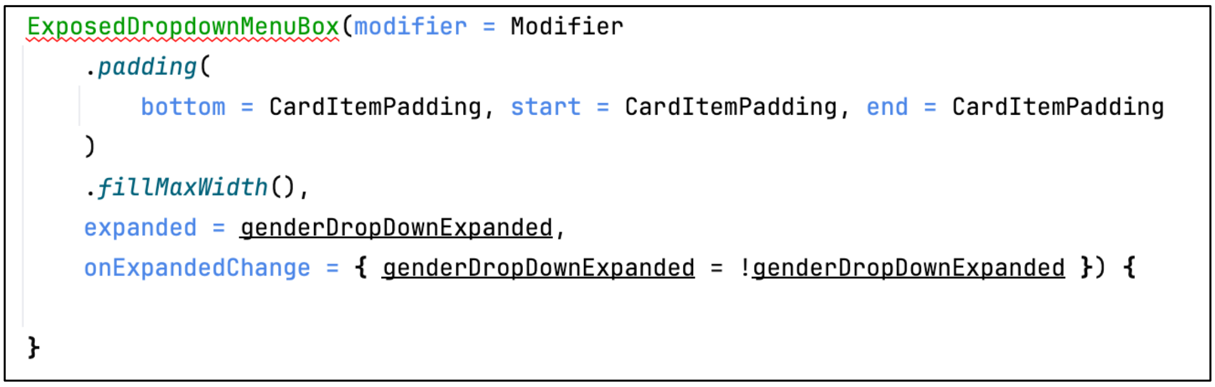
```
    trailingIcon = {
        if (showPassword) {
            IconButton(onClick = { showPassword = false }) {
                Icon(imageVector = Icons.Filled.Visibility,  contentDescription: "visible")
            }

        } else {
            IconButton(onClick = { showPassword = true }) {
                Icon(imageVector = Icons.Filled.VisibilityOff,  contentDescription: "visible")
            }
        }
    },
    placeholder = { Text( text: "Enter your password") },
    modifier = Modifier
        .padding(
            bottom = CardItemPadding, start = CardItemPadding, end = CardItemPadding
        )
        .fillMaxWidth()
)
```

14. Inside the column, add in the states which will allow the user to choose their gender via a dropdown.



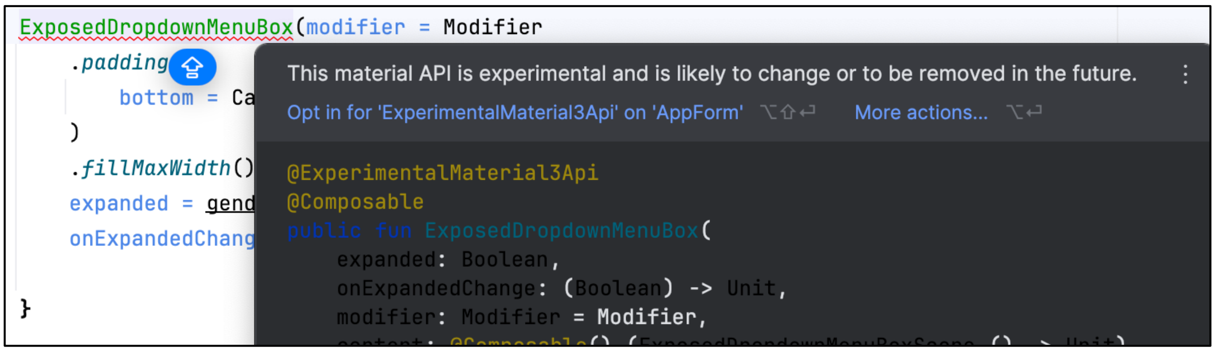
```
var name by remember { mutableStateOf(value = "") }
var password by remember { mutableStateOf(value = "") }
var showPassword by remember { mutableStateOf( value: false) }
var gender by remember { mutableStateOf(value = "") }
val options = listOf("Choose not to reveal", "Male", "Female", "Non-binary")
var genderDropDownExpanded by remember { mutableStateOf( value: false) }
```

15. After the password TextField, add in a Spacer and an ExposedDropdownMenuBox. Notice that an error appears. Android studio requires an Opt in for "ExperimentalMaterial3Api".



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

}
```

16. Opt in and the error will resolve by itself.



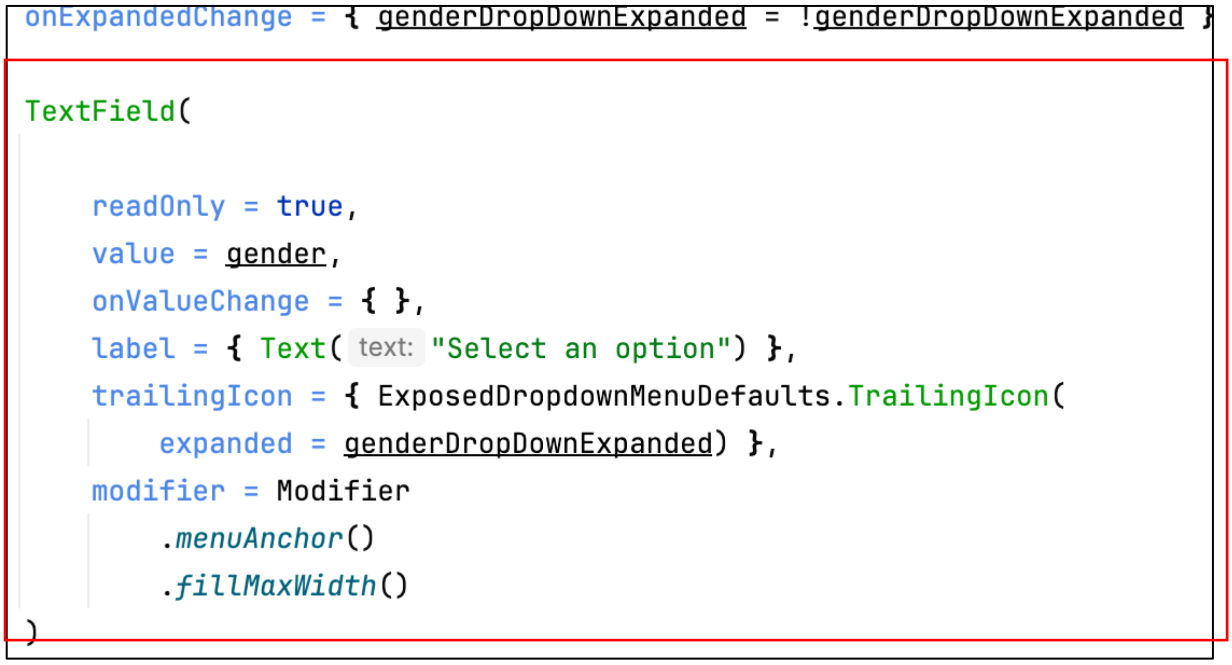
```
ExposedDropdownMenuBox(modifier = Modifier
    .padding
        bottom = Ca
    )
    .fillMaxWidth()
    expanded = gend
    onExpandedChang

}
```

This material API is experimental and is likely to change or to be removed in the future.

Opt in for 'ExperimentalMaterial3Api' on 'AppForm' ⌥⇧↵ More actions... ⌥↵

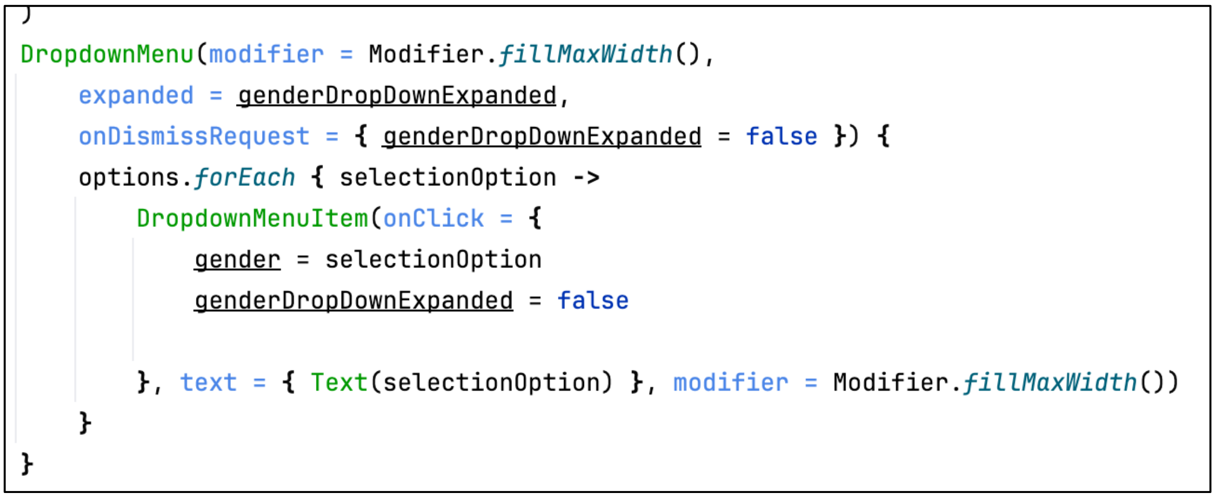
```
@ExperimentalMaterial3Api
@Composable
public fun ExposedDropdownMenuBox(
    expanded: Boolean,
    onExpandedChange: (Boolean) -> Unit,
    modifier: Modifier = Modifier,
    content: @Composable() (ExposedDropdownMenuBoxScope.() -> Unit)
```

17. Inside ExposedDropdownMenuBox, add in a TextField which will display the chosen option and store it in the gender variable.



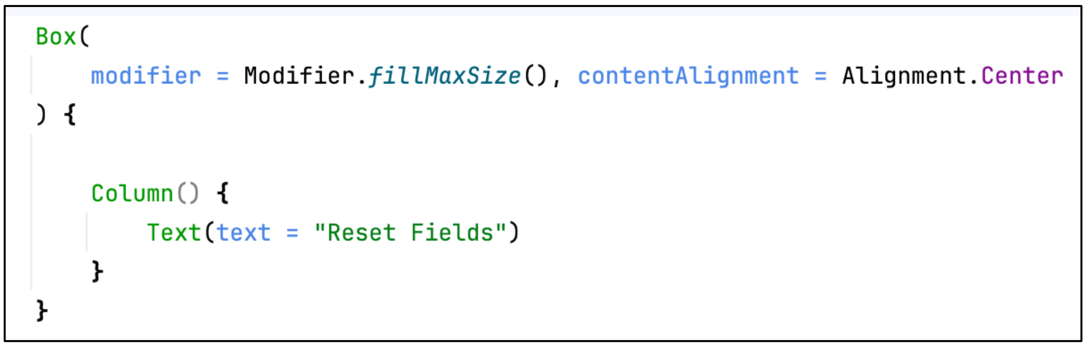
```
onExpandedChange = { genderDropDownExpanded = !genderDropDownExpanded }

TextField(

    readOnly = true,
    value = gender,
    onValueChange = { },
    label = { Text( text: "Select an option") },
    trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(
        expanded = genderDropDownExpanded) },
    modifier = Modifier
        .menuAnchor()
        .fillMaxWidth()
)
```

18. Inside ExposedDropdownMenuBox, add in an DropdownMenu which will display the drop down options. Inside the menu, add in a DropdownMenuItem.



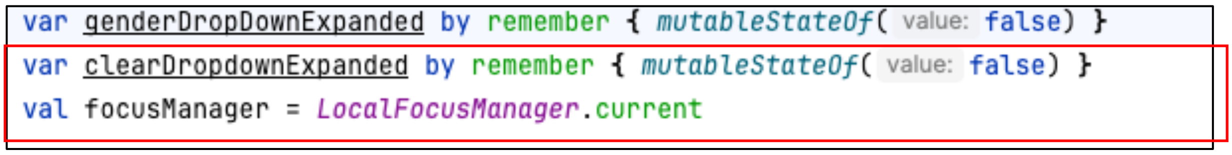
```
)
DropdownMenu(modifier = Modifier.fillMaxWidth(),
    expanded = genderDropDownExpanded,
    onDismissRequest = { genderDropDownExpanded = false }) {
    options.forEach { selectionOption ->
        DropdownMenuItem(onClick = {
            gender = selectionOption
            genderDropDownExpanded = false

        }, text = { Text(selectionOption) }, modifier = Modifier.fillMaxWidth())
    }
}
```

19. After ExposedDropdownMenuBox, add in a Box which holds the rest of the space. Add in a column inside the box and align it in the center.



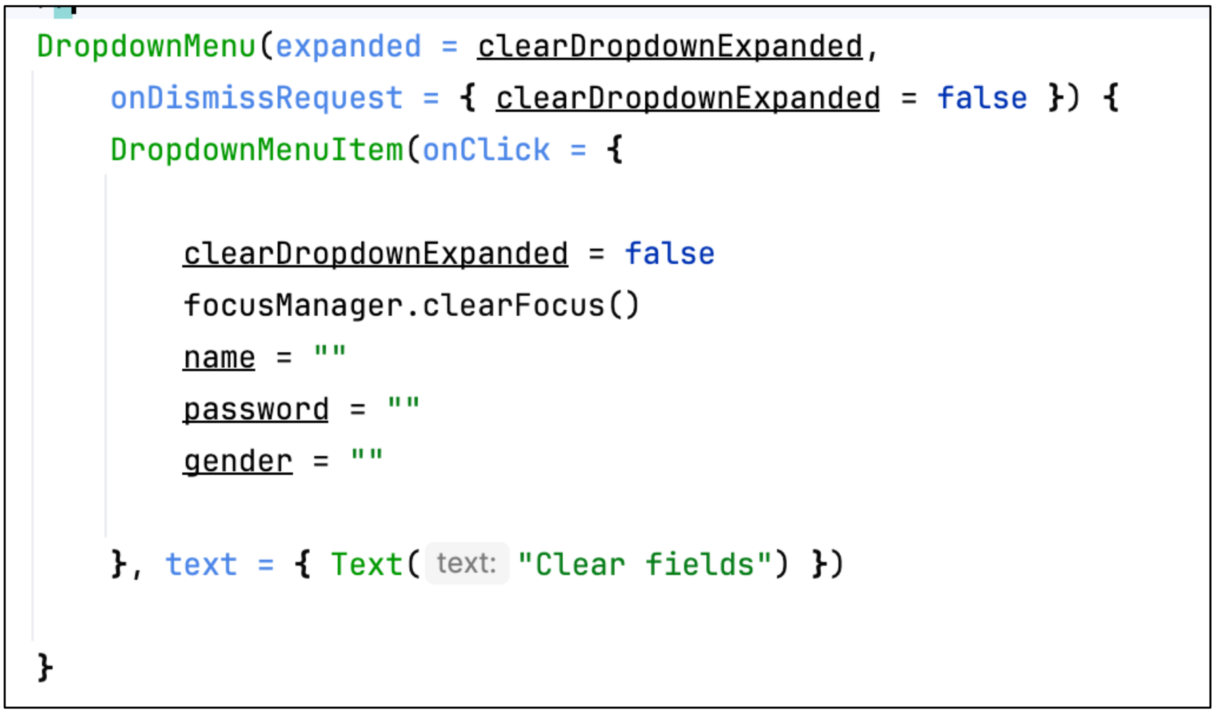
```
Box(
    modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center
) {

    Column() {
        Text(text = "Reset Fields")
    }
}
```

20. Add in the below variables to keep track of the dropdown state. Also to ensure the dropdown does not maintain focus after the user trigger the reset. Make use of the focus manager to clear the focus.



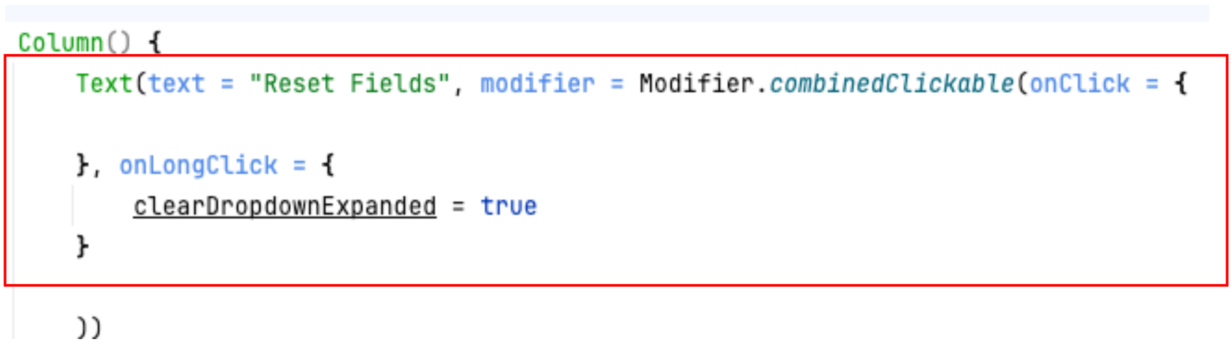
```
var genderDropDownExpanded by remember { mutableStateOf( value: false) }
var clearDropdownExpanded by remember { mutableStateOf( value: false) }
val focusManager = LocalFocusManager.current
```

21. Inside the column, add in a DropdownMenu that displays “Clear field” menu item.



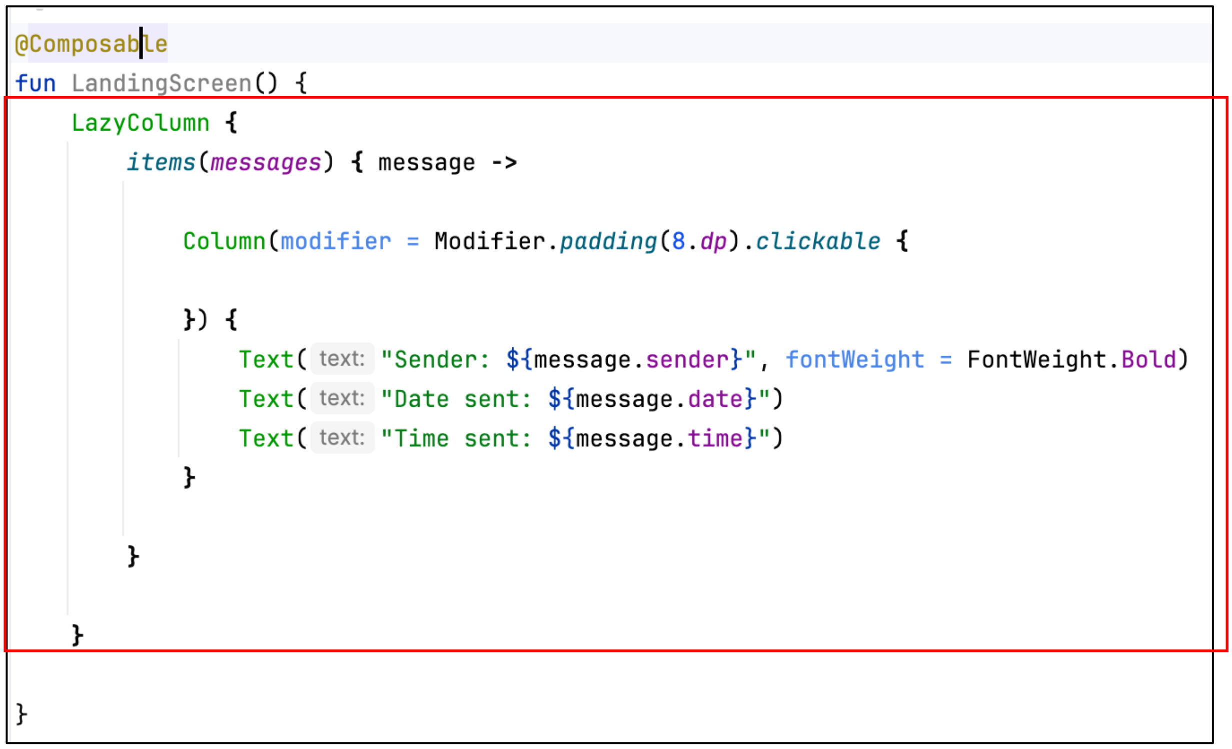
```
DropdownMenu(expanded = clearDropdownExpanded,
    onDismissRequest = { clearDropdownExpanded = false }) {
    DropdownMenuItem(onClick = {

        clearDropdownExpanded = false
        focusManager.clearFocus()
        name = ""
        password = ""
        gender = ""

    }, text = { Text( text: "Clear fields") })

}
```

22. Update the column with a modifier that listens for a “Long click”. When a “LongClick” is triggered, it will display the Dropdownmenu that was created above.



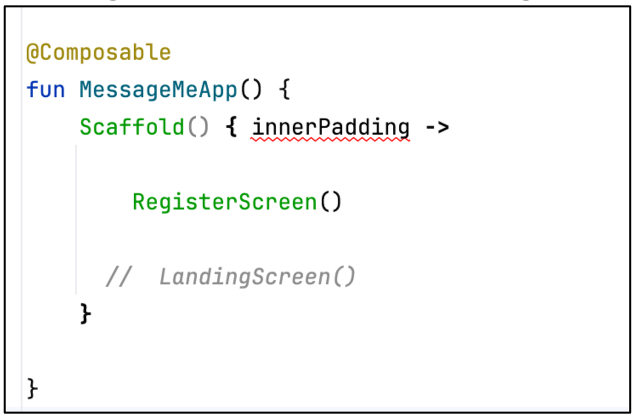
```
Column() {
    Text(text = "Reset Fields", modifier = Modifier.combinedClickable(onClick = {

    }, onLongClick = {
        clearDropdownExpanded = true
    }

    ))
```

23. Open up LandingScreen.kt and add in a LazyColumn that displays a list of messages that was pre-defined in the file. Inside each item, make use of a column to display each message.



```
@Composable
fun LandingScreen() {
    LazyColumn {
        items(messages) { message ->

            Column(modifier = Modifier.padding(8.dp).clickable {

            }) {
                Text( text: "Sender: ${message.sender}", fontWeight = FontWeight.Bold)
                Text( text: "Date sent: ${message.date}")
                Text( text: "Time sent: ${message.time}")
            }

        }

    }

}
```

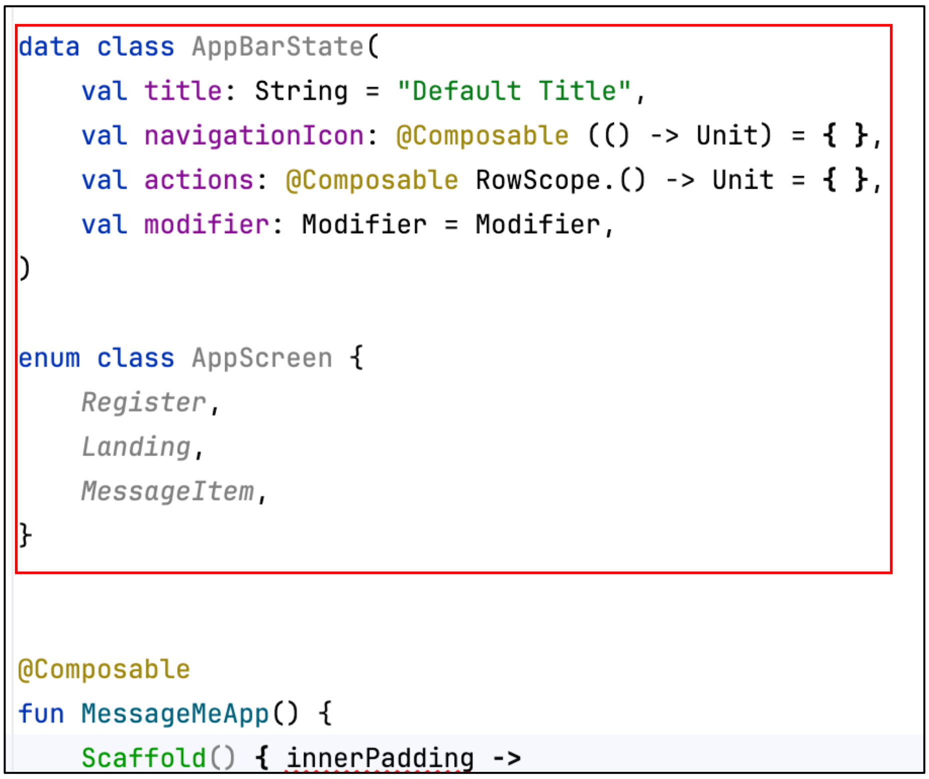
24. Open up MessageMeApp, make use of the function to test RegisterScreen and Landing Screen function.



```
@Composable
fun MessageMeApp() {
    Scaffold() { innerPadding ->

        RegisterScreen()

    //  LandingScreen()
    }

}
```

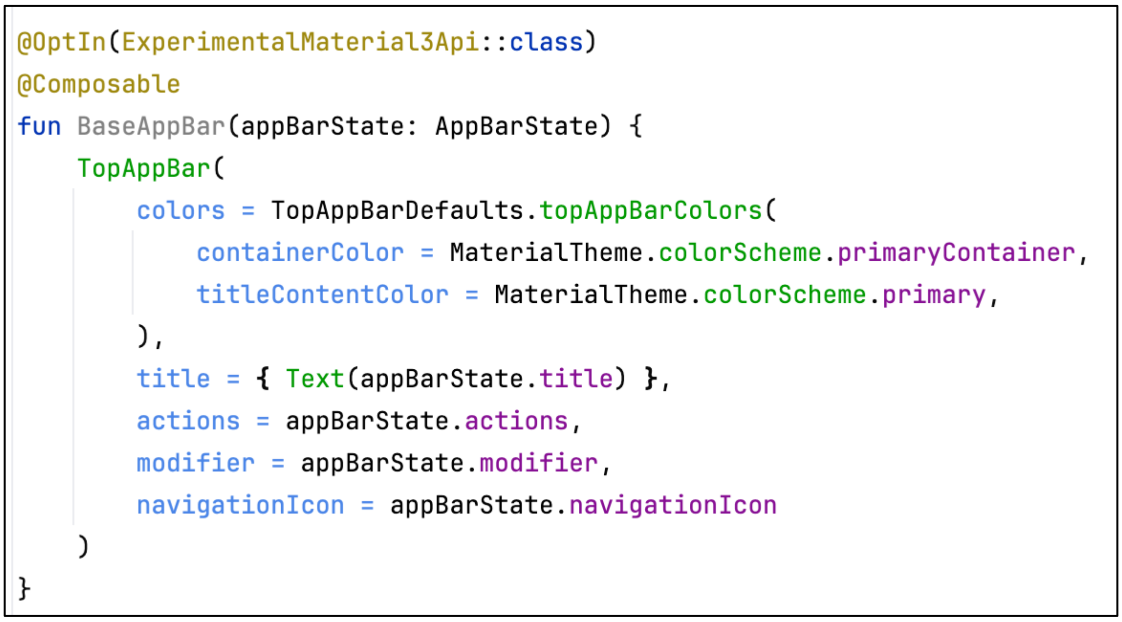
App Bar

25. Inside MessageMeApp.kt, add in the class below for AppBar and Navigation. AppBarState is a data class which holds the properties of the AppBar. AppScreen holds the three screens the app will have.



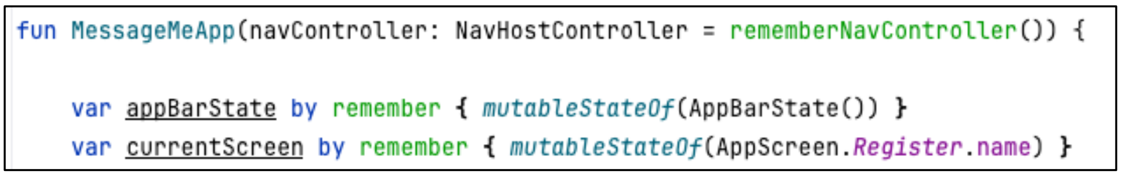
```
data class AppBarState(
    val title: String = "Default Title",
    val navigationIcon: @Composable (() -> Unit) = { },
    val actions: @Composable RowScope.() -> Unit = { },
    val modifier: Modifier = Modifier,
)

enum class AppScreen {
    Register,
    Landing,
    MessageItem,
}


@Composable
fun MessageMeApp() {
    Scaffold() { innerPadding ->
```

26. Inside MessageMeApp.kt, add in a function BaseAppBar. It will make use of the appBarState variable to update the Top AppBar.



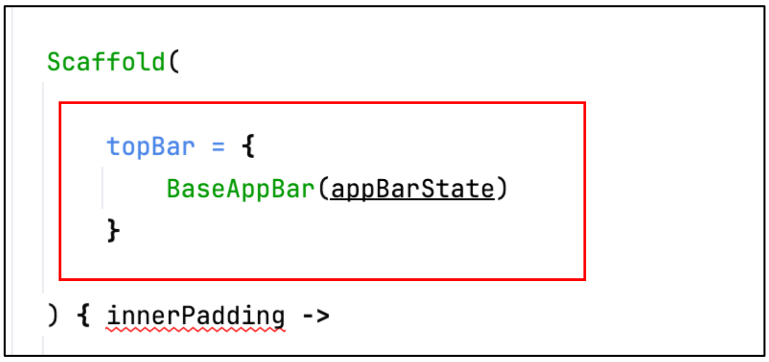
```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun BaseAppBar(appBarState: AppBarState) {
    TopAppBar(
        colors = TopAppBarDefaults.topAppBarColors(
            containerColor = MaterialTheme.colorScheme.primaryContainer,
            titleContentColor = MaterialTheme.colorScheme.primary,
        ),
        title = { Text(appBarState.title) },
        actions = appBarState.actions,
        modifier = appBarState.modifier,
        navigationIcon = appBarState.navigationIcon
    )
}
```

27. Inside MessageMeApp.kt, update the MessageMeApp function. Add in the NavHostController argument. Add in the two variables that will track the state of the appbar and the current screen being displayed to the user.



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

    var appBarState by remember { mutableStateOf(AppBarState()) }
    var currentScreen by remember { mutableStateOf(AppScreen.Register.name) }
```

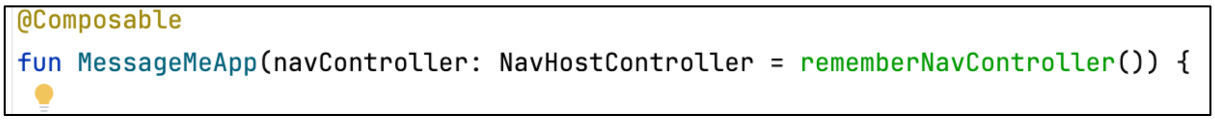
28. Inside the MessageMeApp function. Add in the topBar by calling the BaseAppBar function.



```
Scaffold(

    topBar = {
        BaseAppBar(appBarState)
    }

) { innerPadding ->
```

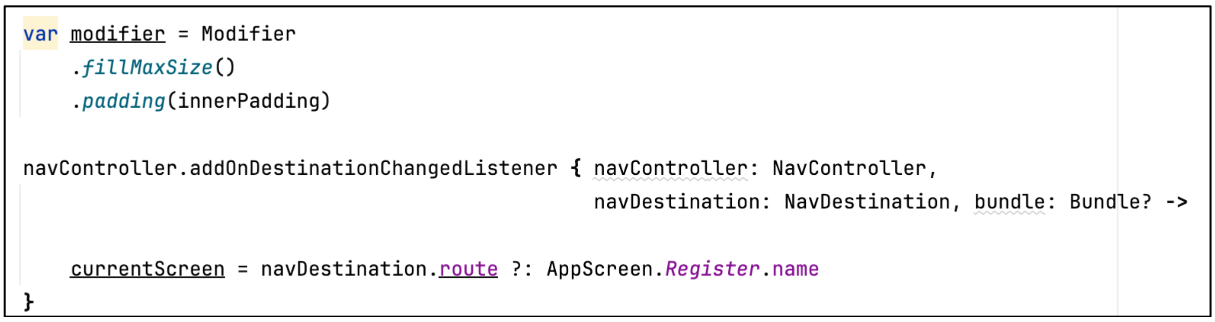
Navigation

29. Inside MessageMeApp.kt, update the MessageMeApp function. Add in the NavHostController argument.



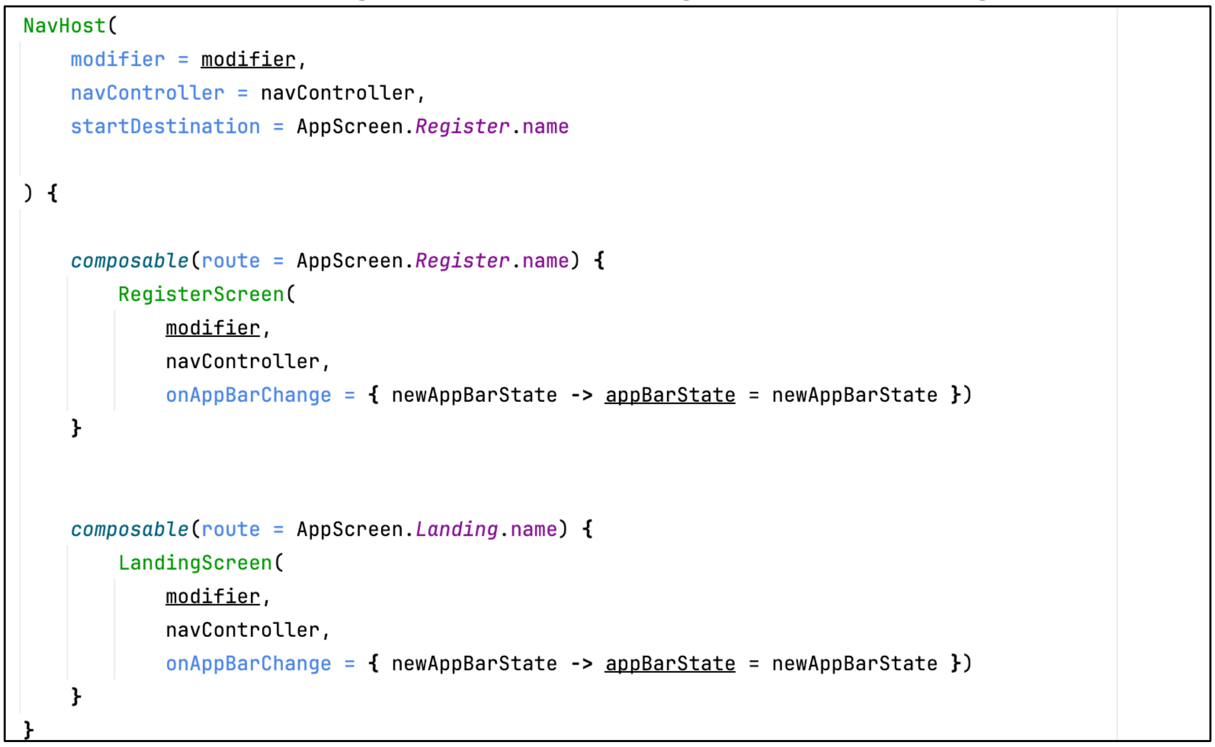
```kotlin
@Composable
fun MessageMeApp(navController: NavHostController = rememberNavController()) {
```

30. Inside the MessageMeApp function. Make use of the navController variable to add a destination change listener. Each time the navigation change, the current screen state will be updated.



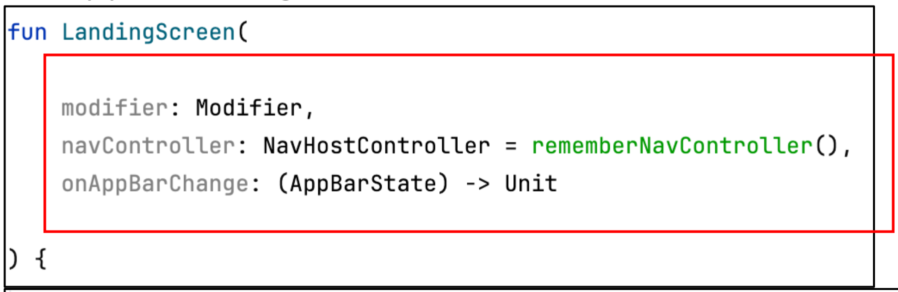
```kotlin
var modifier = Modifier
    .fillMaxSize()
    .padding(innerPadding)

navController.addOnDestinationChangedListener { navController: NavController,
                                                navDestination: NavDestination, bundle: Bundle? ->

    currentScreen = navDestination.route ?: AppScreen.Register.name
}
```

31. Inside the MessageMeApp function. Add in a NavHost that declares the navigation route for Register and Landing screen.

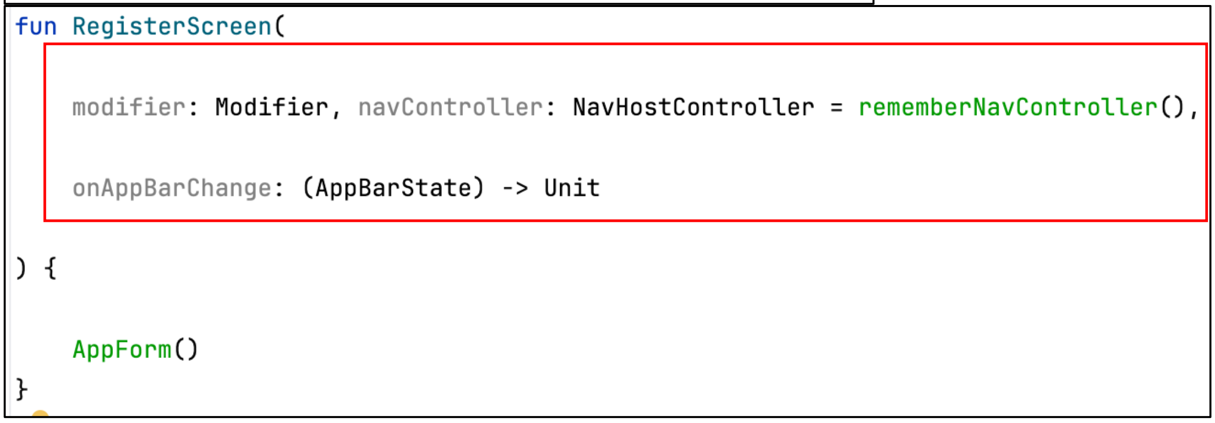


```
NavHost(
    modifier = modifier,
    navController = navController,
    startDestination = AppScreen.Register.name

) {

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


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

32. Update the different screen functions to add in navController and AppBarChange .

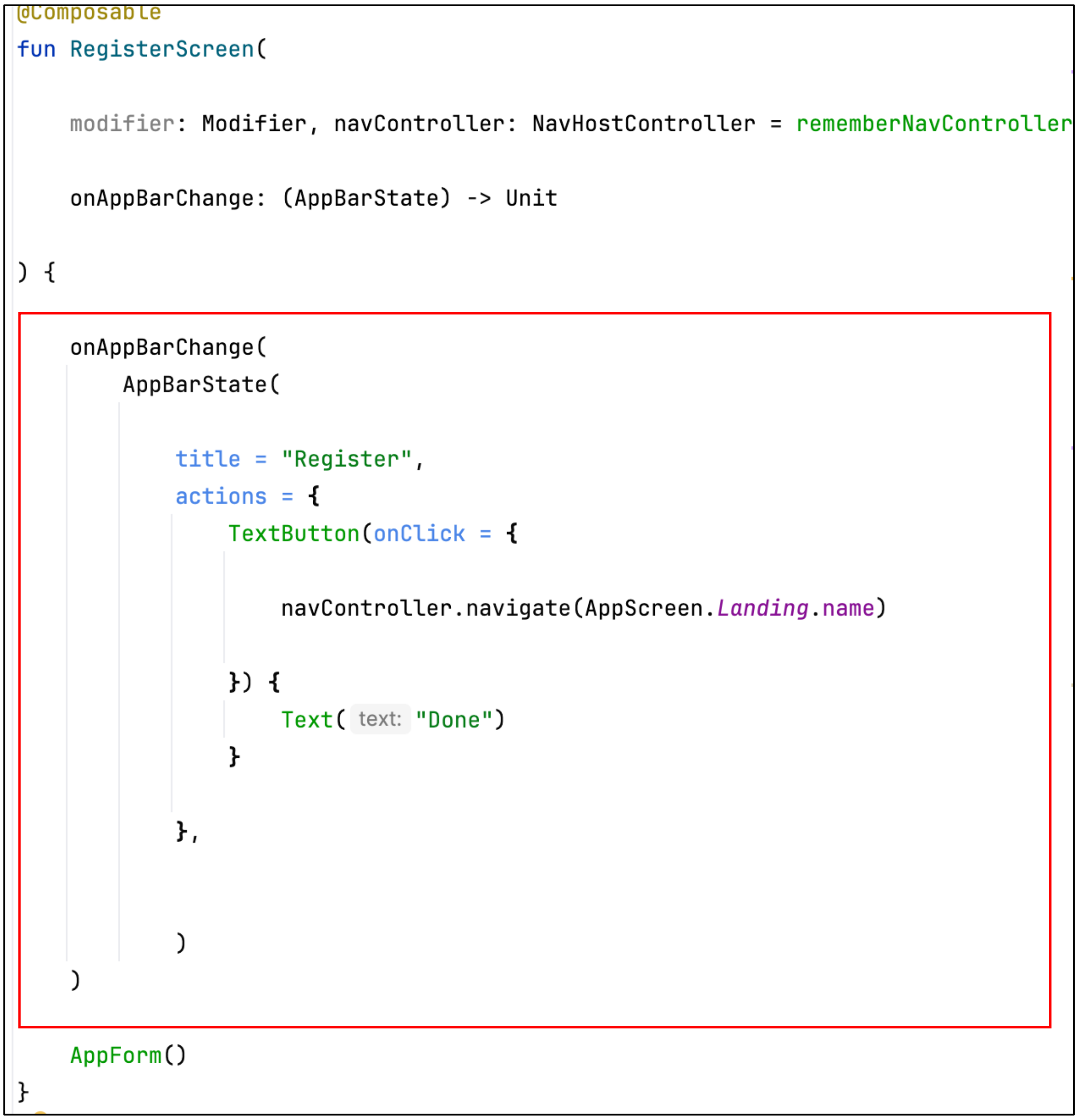


```
fun LandingScreen(

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

) {
```



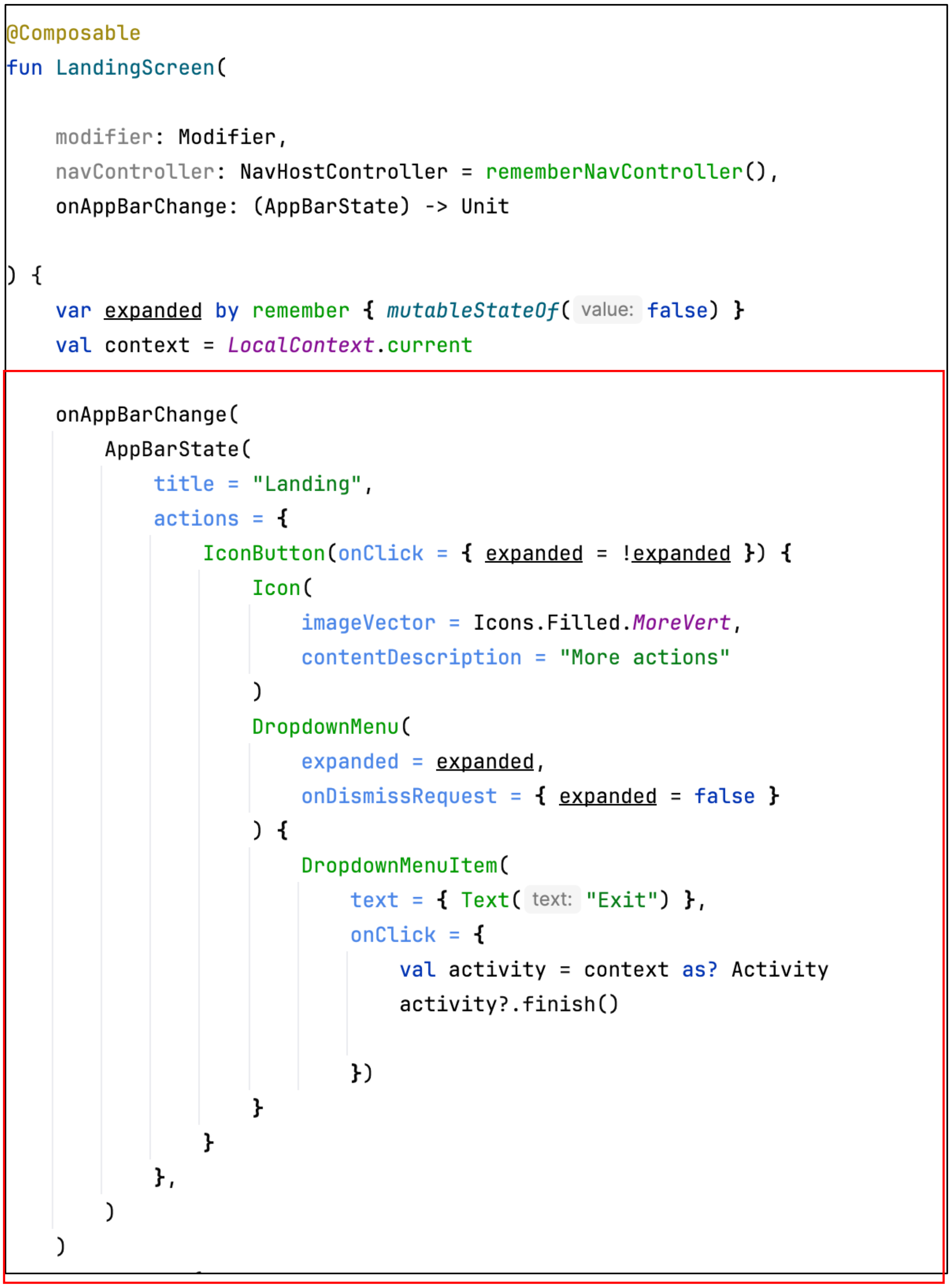
```
fun RegisterScreen(

    modifier: Modifier, navController: NavHostController = rememberNavController(),

    onAppBarChange: (AppBarState) -> Unit

) {

    AppForm()
}
```

33. Inside Register Screen function, call the onAppBarChange function that states the a new AppBarState.



```
@Composable
fun RegisterScreen(

    modifier: Modifier, navController: NavHostController = rememberNavController

    onAppBarChange: (AppBarState) -> Unit

) {

    onAppBarChange(
        AppBarState(

            title = "Register",
            actions = {
                TextButton(onClick = {

                    navController.navigate(AppScreen.Landing.name)

                }) {
                    Text( text: "Done")
                }

            },


        )
    )

    AppForm()
}
```

34. Inside Landing Screen function, call the onAppBarChange function that states the a new AppBarState.

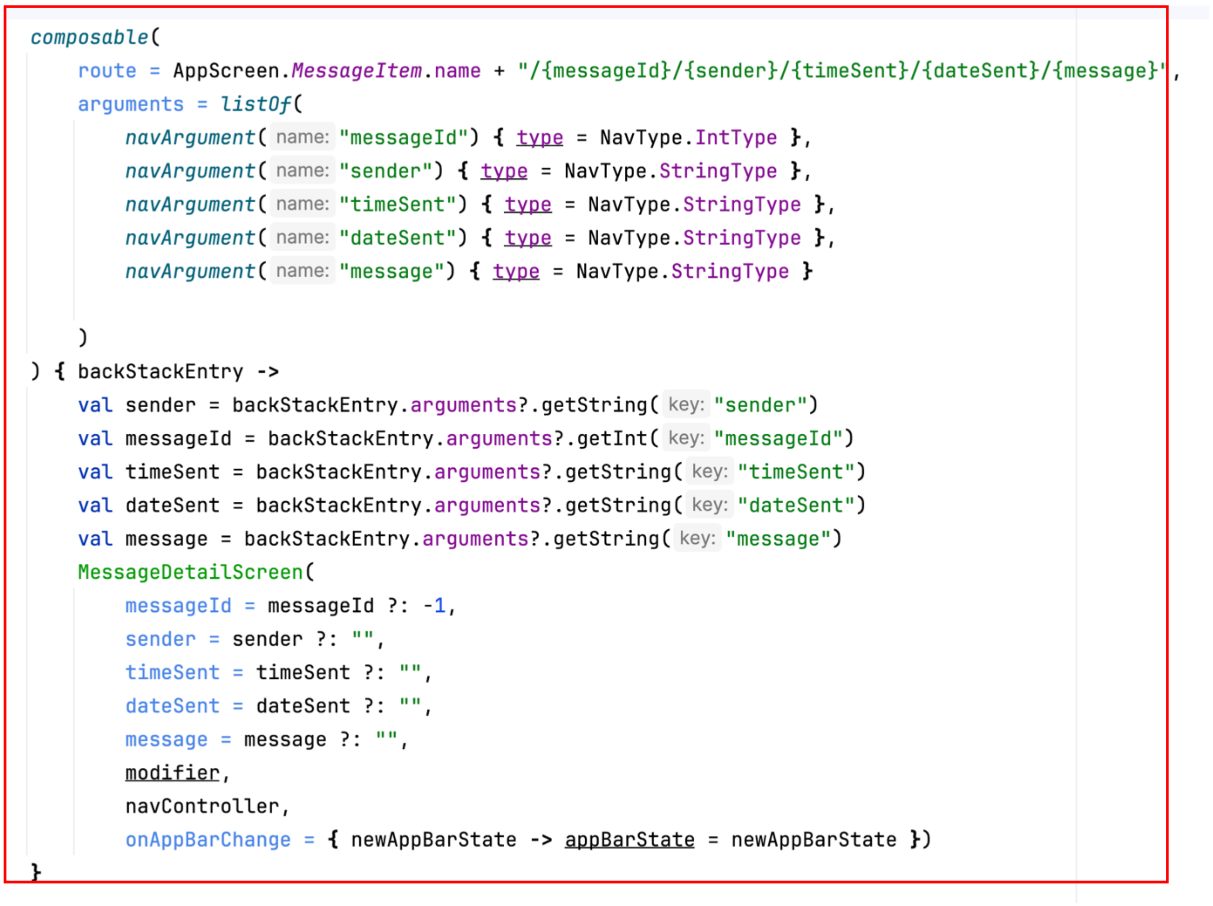


```
@Composable
fun LandingScreen(

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

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

    onAppBarChange(
        AppBarState(
            title = "Landing",
            actions = {
                IconButton(onClick = { expanded = !expanded }) {
                    Icon(
                        imageVector = Icons.Filled.MoreVert,
                        contentDescription = "More actions"
                    )
                    DropdownMenu(
                        expanded = expanded,
                        onDismissRequest = { expanded = false }
                    ) {
                        DropdownMenuItem(
                            text = { Text( text: "Exit") },
                            onClick = {
                                val activity = context as? Activity
                                activity?.finish()

                            })
                    }
                }
            },
        )
    )
```

35. Run the application and attempt to navigate between the two screens.

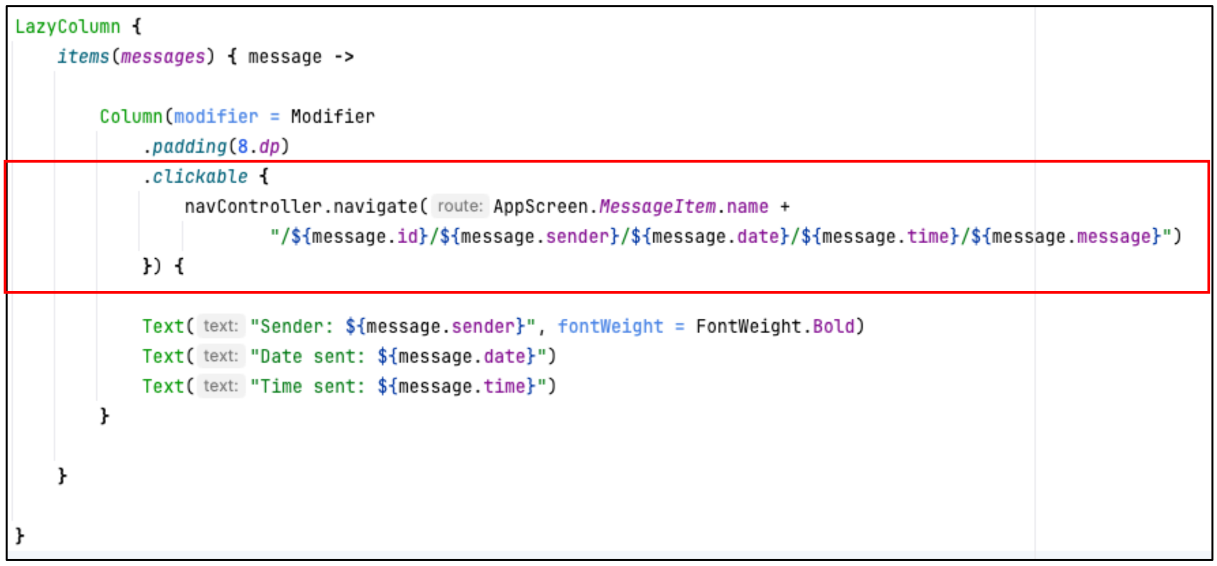
Navigation with arguments

36. Inside the MessageMeApp function, add in the next navigation route. This route will display message in detailed to the user. Inside the composable route declaration, parse the selected message property to MessageDetailScreen.



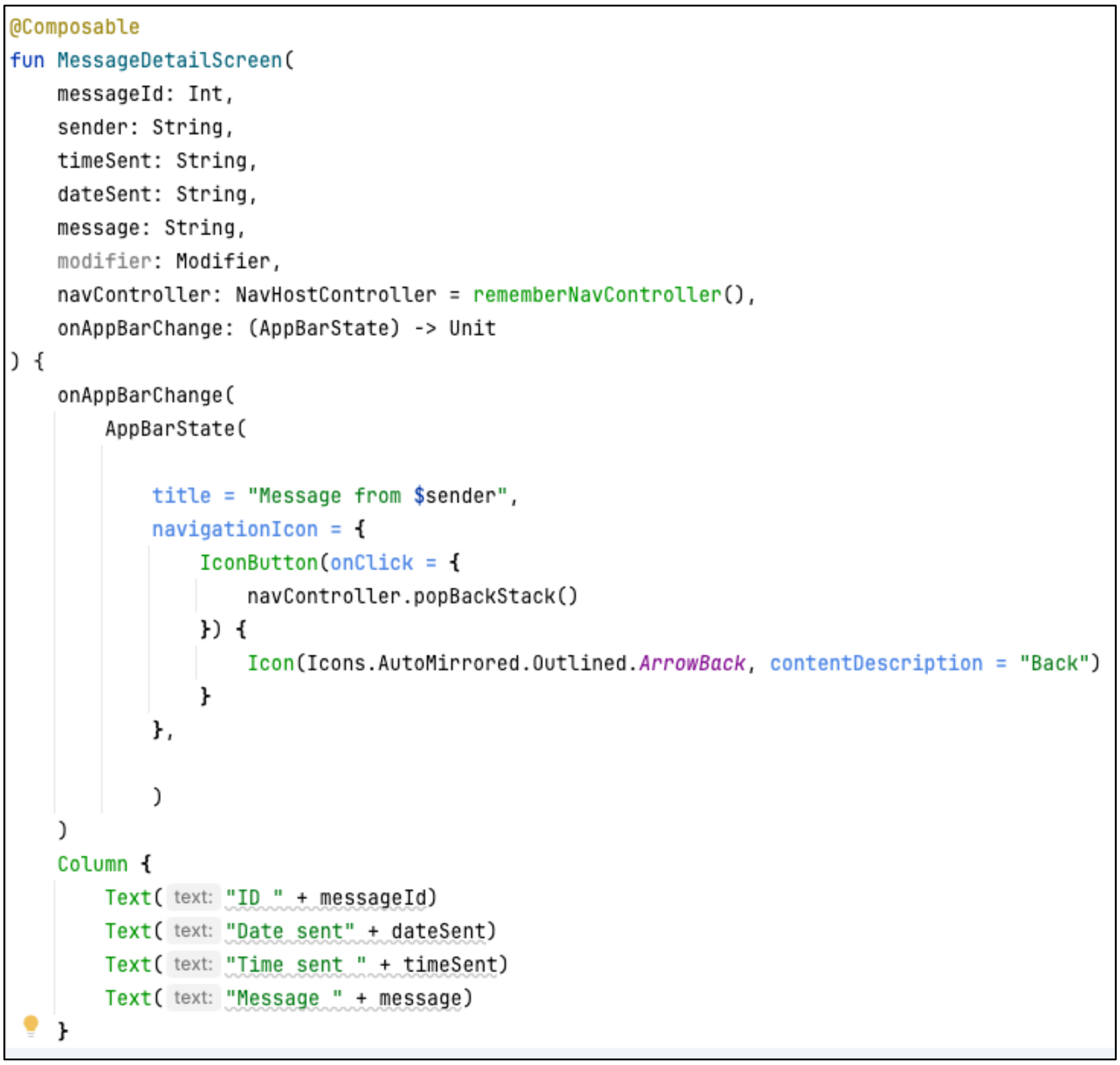
```
composable(
    route = AppScreen.MessageItem.name + "/{messageId}/{sender}/{timeSent}/{dateSent}/{message}",
    arguments = listOf(
        navArgument( name: "messageId") { type = NavType.IntType },
        navArgument( name: "sender") { type = NavType.StringType },
        navArgument( name: "timeSent") { type = NavType.StringType },
        navArgument( name: "dateSent") { type = NavType.StringType },
        navArgument( name: "message") { type = NavType.StringType }

    )
) { backStackEntry ->
    val sender = backStackEntry.arguments?.getString( key: "sender")
    val messageId = backStackEntry.arguments?.getInt( key: "messageId")
    val timeSent = backStackEntry.arguments?.getString( key: "timeSent")
    val dateSent = backStackEntry.arguments?.getString( key: "dateSent")
    val message = backStackEntry.arguments?.getString( key: "message")
    MessageDetailScreen(
        messageId = messageId ?: -1,
        sender = sender ?: "",
        timeSent = timeSent ?: "",
        dateSent = dateSent ?: "",
        message = message ?: "",
        modifier,
        navController,
        onAppBarChange = { newAppBarState -> appBarState = newAppBarState })
}
```

37. Inside the LandingScreen.kt, update LandingScreen function. Add in the clickable lambda which will make use of navController to navigate to MessageDetailScreen by calling the MessageItem route with the message details as arguments.



```
LazyColumn {
    items(messages) { message ->

        Column(modifier = Modifier
            .padding(8.dp)
            .clickable {
                navController.navigate( route: AppScreen.MessageItem.name +
                        "/${message.id}/${message.sender}/${message.date}/${message.time}/${message.message}")
            }) {

            Text( text: "Sender: ${message.sender}", fontWeight = FontWeight.Bold)
            Text( text: "Date sent: ${message.date}")
            Text( text: "Time sent: ${message.time}")
        }

    }

}
```

38. Inside the MessageDetailScreen.kt, update MessageDetailScreen function. It adds in the new AppBarState and display the message details in the screen.



```
@Composable
fun MessageDetailScreen(
    messageId: Int,
    sender: String,
    timeSent: String,
    dateSent: String,
    message: String,
    modifier: Modifier,
    navController: NavHostController = rememberNavController(),
    onAppBarChange: (AppBarState) -> Unit
) {
    onAppBarChange(
        AppBarState(

            title = "Message from $sender",
            navigationIcon = {
                IconButton(onClick = {
                    navController.popBackStack()
                }) {
                    Icon(Icons.AutoMirrored.Outlined.ArrowBack, contentDescription = "Back")
                }
            },

            )
    )
    Column {
        Text( text: "ID " + messageId)
        Text( text: "Date sent" + dateSent)
        Text( text: "Time sent " + timeSent)
        Text( text: "Message " + message)
    }
```