

Kazakh British Technical University

Mobile Programming

Assignment II: Create Basic Pages of Instagram in Android Studio

by Moldir Polat

October 18th 2024

Table of contents

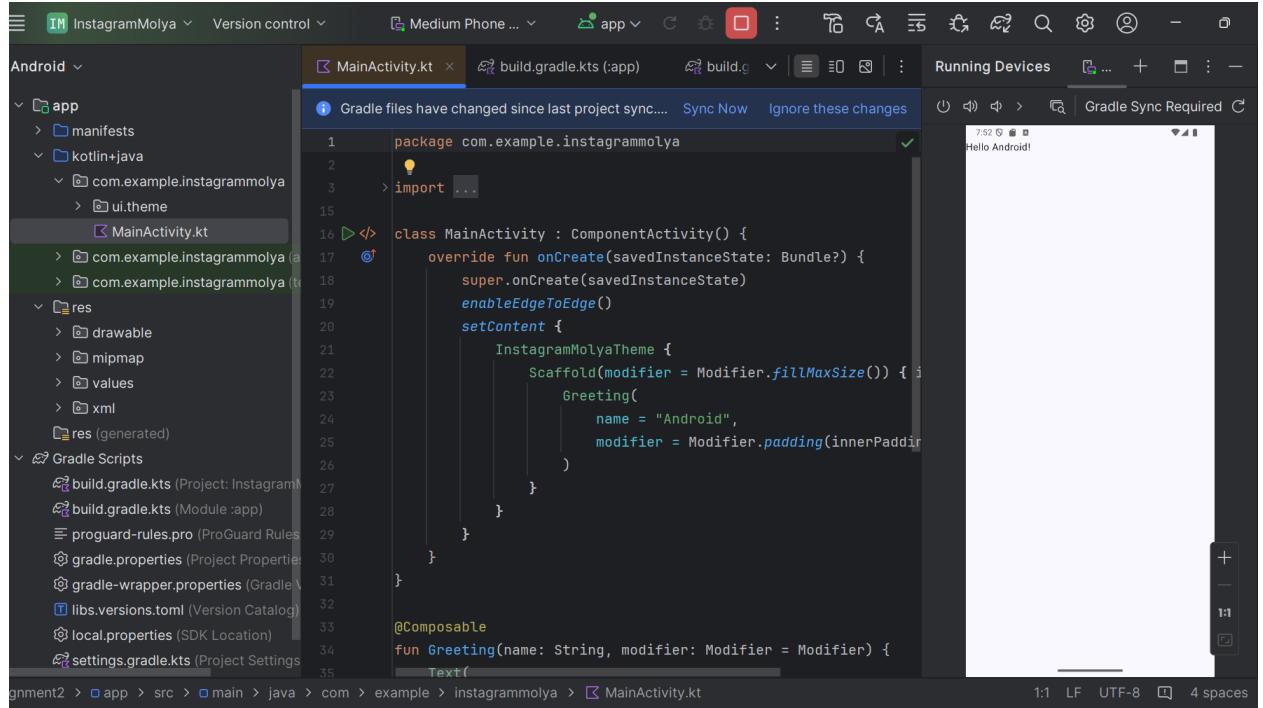
• Introduction -----	3
• Project Setup -----	4
• Page Design -----	5
• Navigation -----	17
• User Interaction -----	18
• Challenges and Solutions -----	21
• Conclusion -----	24
• References -----	25

Introduction

The goal of this assignment is to develop a simplified version of Instagram's core pages using Android Studio. We will learn to implement layouts, navigation, and user interface elements similar to Instagram.

Project Setup

We created project from Empty Activity template.



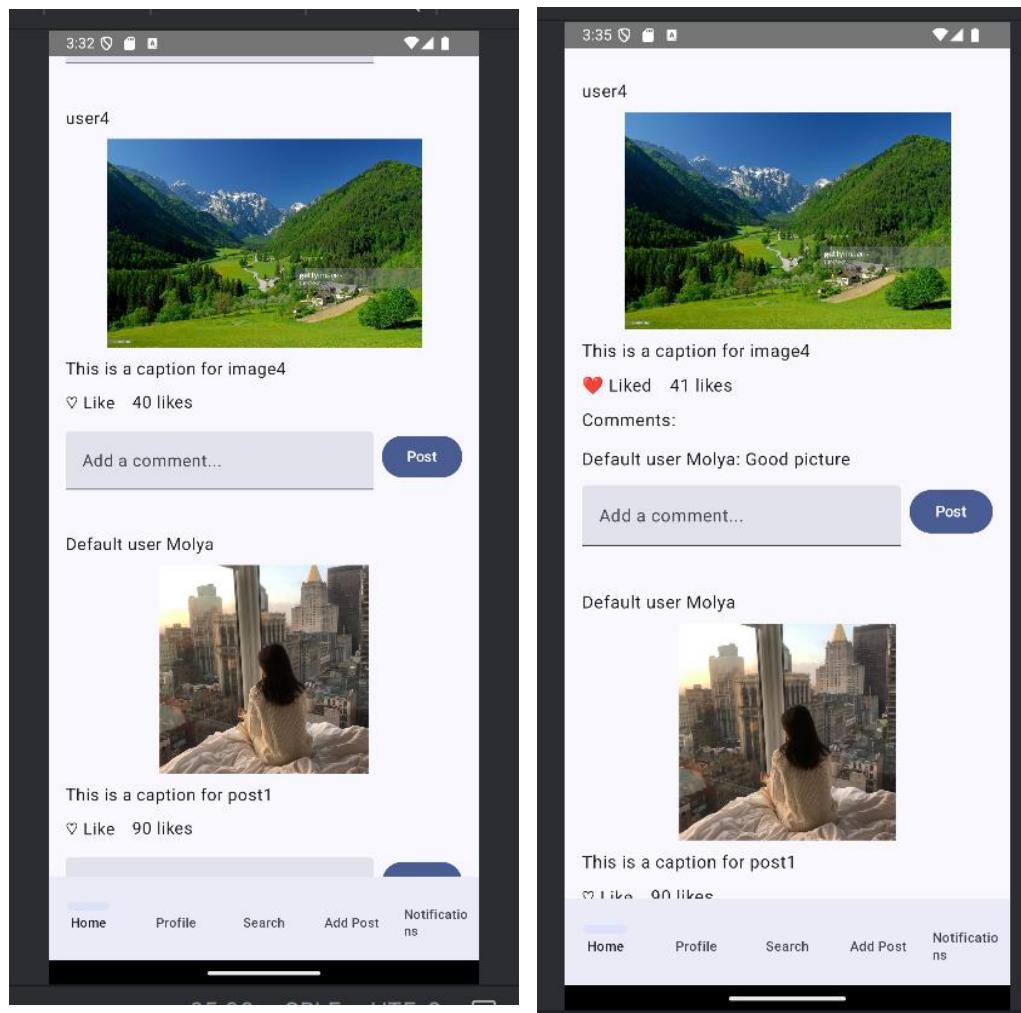
The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "Android". It includes the "app" module with subfolders like "manifests", "kotlin+java", "res", and "Gradle Scripts".
- MainActivity.kt:** The main code editor window displays the Kotlin code for the MainActivity. The code initializes a Scaffold with a Greeting component and a Text component.
- Build Scripts:** The "Gradle Scripts" section shows the build.gradle.kts file for the app module.
- Running Device:** On the right, a preview window shows the Android emulator with the text "Hello Android!" displayed.
- Status Bar:** At the bottom, the status bar shows "1:1 LF UTF-8 4 spaces".

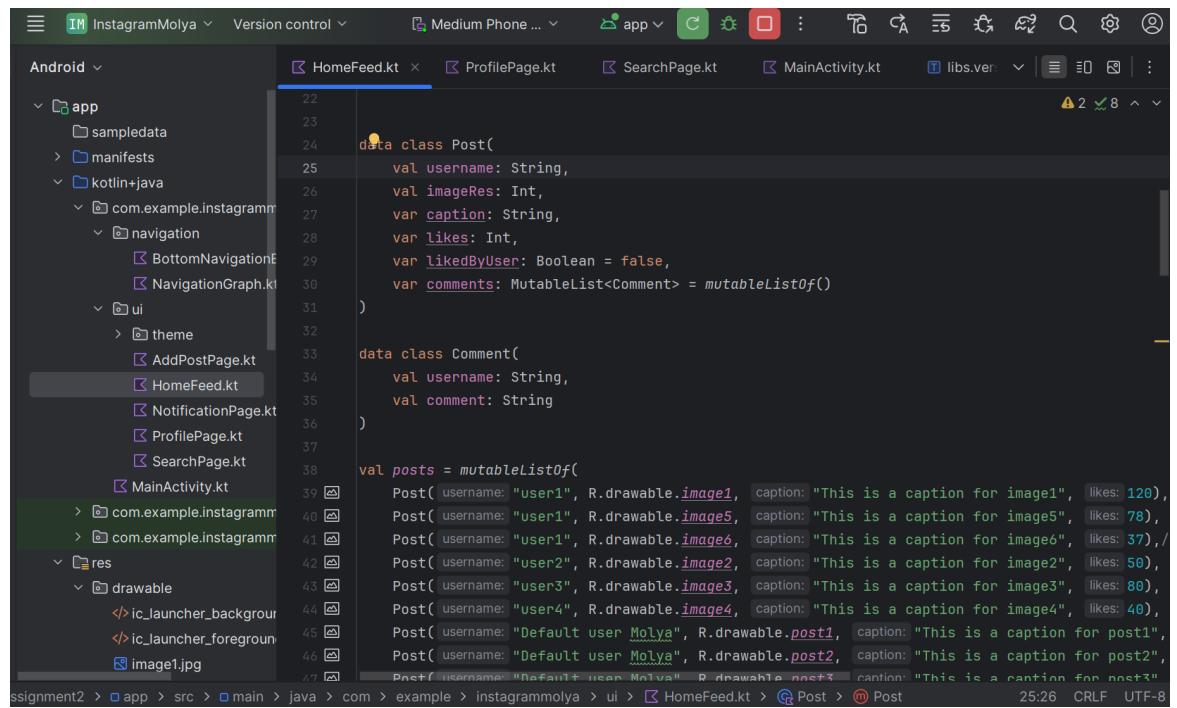
Page Design

1. HomeFeed*

This page is designed to display a feed of posts, similar to Instagram. It uses a vertical scrolling list where each post contains the following elements: a username, an image, a caption, a like button, a like count, and a comments section.



Data structure: there are two main classes – Post and Comment.



```
data class Post(
    val username: String,
    val imageRes: Int,
    var caption: String,
    var likes: Int,
    var likedByUser: Boolean = false,
    var comments: MutableList<Comment> = mutableListOf()
)

data class Comment(
    val username: String,
    val comment: String
)

val posts = mutableListOf(
    Post(username: "user1", R.drawable.image1, caption: "This is a caption for image1", likes: 120),
    Post(username: "user1", R.drawable.image5, caption: "This is a caption for image5", likes: 78),
    Post(username: "user1", R.drawable.image6, caption: "This is a caption for image6", likes: 37),
    Post(username: "user2", R.drawable.image2, caption: "This is a caption for image2", likes: 50),
    Post(username: "user3", R.drawable.image3, caption: "This is a caption for image3", likes: 80),
    Post(username: "user4", R.drawable.image4, caption: "This is a caption for image4", likes: 40),
    Post(username: "Default user Molya", R.drawable.post1, caption: "This is a caption for post1"),
    Post(username: "Default user Molya", R.drawable.post2, caption: "This is a caption for post2"),
    Post(username: "Default user Molya", R.drawable.post3, caption: "This is a caption for post3")
)
```

Layout: LazyColumn was used to display the lists of posts. Then **PostItem** represents objects in LazyColumn.

- Username
- Image
- Caption
- Like button & like count:
- Comments Section (**CommentItem** was used)
- Add new comment

Users can click on the "♥ Like" text to like the post, which increments the likes count, changes the icon to "♥ Liked", and adds a notification that the post was liked by the current user.

```

Android
app
└ sampledata
manifests
kotlin+java
└ com.example.instagramm
    └ navigation
        └ BottomNavigationE
        └ NavigationGraph.Kt
ui
└ theme
    └ AddPostPage.kt
    └ HomeFeed.kt
    └ NotificationPage.kt
    └ ProfilePage.kt
    └ SearchPage.kt
    └ MainActivity.kt
com.example.instagramm
com.example.instagramm
res
└ drawable
    </> ic_launcher_background
    </> ic_launcher_foreground
    image1.jpg

```

HomeFeed.kt

```

65 fun PostItem(post: Post, navController: NavController?) {
68     Column(modifier = Modifier.padding(16.dp)) {
89         Text(text = post.caption, modifier = Modifier.padding(top = 8.dp))
90         Row(modifier = Modifier.padding(vertical = 8.dp)) {
91             // Like button
92             Text(
93                 text = if (post.likedByUser) "❤️ Liked" else "♡ Like",
94                 modifier = Modifier.clickable {
95                     if (!post.likedByUser) {
96                         post.likedByUser = true
97                         post.likes += 1
98                         // Add a like notification
99                         notifications.add(
100                             Notification(
101                                 type = "like",
102                                 username = currentUser,
103                                 message = "liked ${post.username}'s post"
104                             )
105                         )
106                     } else {
107                         post.likedByUser = false
108                         post.likes -= 1
109                     }
110                 )
111             )
112         }
113     }
114 }
115 
```

Assignment2 > app > src > main > java > com > example > instagrammolya > ui > HomeFeed.kt > PostItem

Users can add comment by typing into a TextField and clicking the "Post" button. The new comment is appended to the list of comments for that post, and a notification is generated for the comment action.

```

Android
app
└ sampledata
manifests
kotlin+java
└ com.example.instagramm
    └ navigation
        └ BottomNavigationE
        └ NavigationGraph.Kt
    └ ui
        └ theme
            └ AddPostPage.kt
            └ HomeFeed.kt
            └ NotificationPage.kt
            └ ProfilePage.kt
            └ SearchPage.kt
            └ MainActivity.kt
com.example.instagramm
com.example.instagramm
res
└ drawable
    </> ic_launcher_background
    </> ic_launcher_foreground
    image1.jpg

```

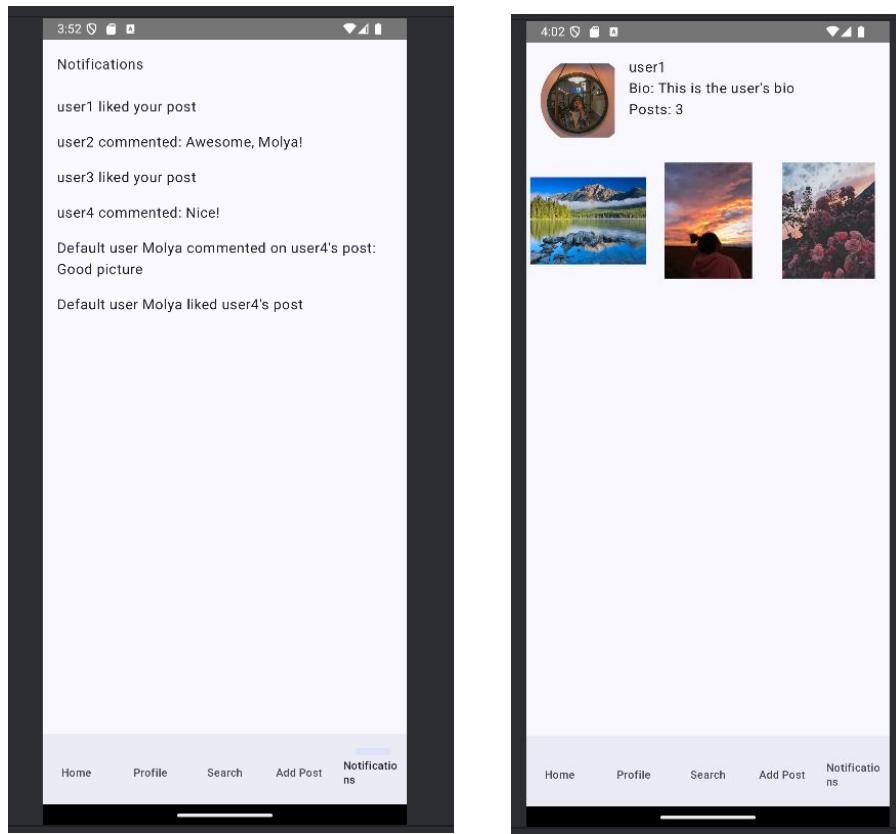
HomeFeed.kt

```

65 fun PostItem(post: Post, navController: NavController?) {
68     Column(modifier = Modifier.padding(16.dp)) {
129     ) {
130         TextField(
131             value = newComment,
132             onValueChange = { newComment = it },
133             placeholder = { Text(text: "Add a comment...") },
134             modifier = Modifier.weight(1f)
135         )
136         Button(
137             onClick = {
138                 if (newComment.isNotBlank()) {
139                     post.comments.add(Comment(username = currentUser, comment = newComment))
140                     notifications.add(
141                         Notification(
142                             type = "comment",
143                             username = currentUser,
144                             message = "commented on ${post.username}'s post: $newComment"
145                         )
146                     )
147                     newComment = ""
148                 }
149             },
150             modifier = Modifier.padding(start = 8.dp)
151         ) {
152             Text(text: "Post")
153         }
154     }
155 }
156 
```

Assignment2 > app > src > main > java > com > example > instagrammolya > ui > HomeFeed.kt > PostItem

When a user likes or comments on a post, a corresponding notification is created and added to the global list of notifications (navigates to Notificaion page).

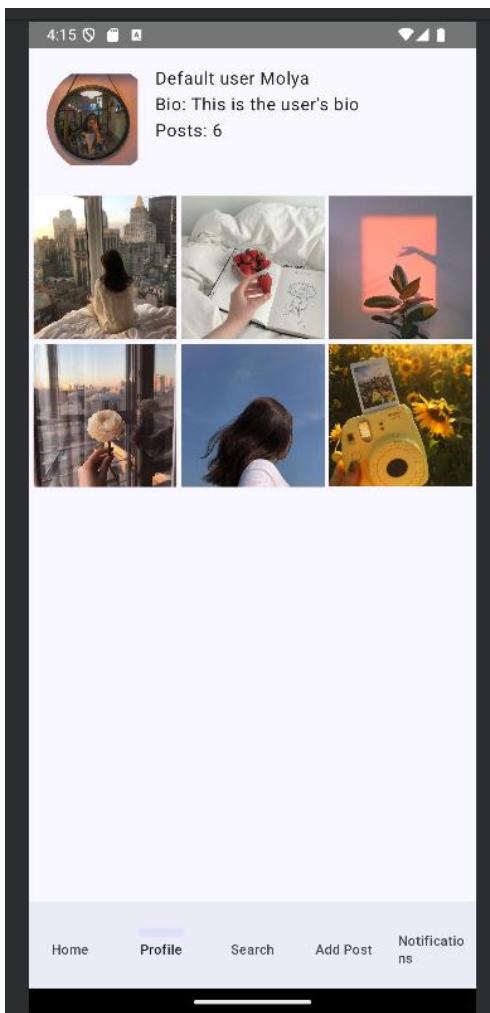


User can click on user's name and open his Profile page (navigates to Profile page)

*See file *HomeFeed.kt* in ui folder.

1. Profile page*

This page is designed to display certain user's profile. It displays the user's profile picture(just static photo in current assignment), basic information, posts' count and their posts in a grid layout, with three posts per row. By default it opens Default user's profile.



Layout consists of two parts – *ProfileHeader* and *UserPostsGrid*. First part is on top, includes profile image, username, bio, posts' count. Second part is user's posts in grid format. Posts are filtered by username from global list posts (in *HomeFeed.kt*). *LazyVerticalGrid* was used to adjust the format.

```

@Composable
fun UserPostsGrid(username: String) {
    val userPosts = posts.filter { it.username == username } // Filter posts by the username

    if (userPosts.isNotEmpty()) {
        LazyVerticalGrid(
            columns = GridCells.Fixed(count: 3), // 3 columns
            modifier = Modifier.fillMaxSize(),
            contentPadding = PaddingValues(4.dp),
            verticalArrangement = Arrangement.spacedBy(4.dp),
            horizontalArrangement = Arrangement.spacedBy(4.dp)
        ) {
            items(userPosts) { post ->
                Image(
                    painter = painterResource(id = post.imageRes),
                    contentDescription = null,
                    modifier = Modifier
                        .fillMaxWidth()
                        .aspectRatio(ratio: 1f) // Make images square
                )
            }
        }
    } else {
        Text(text = "No posts yet.", modifier = Modifier.padding(16.dp))
    }
}

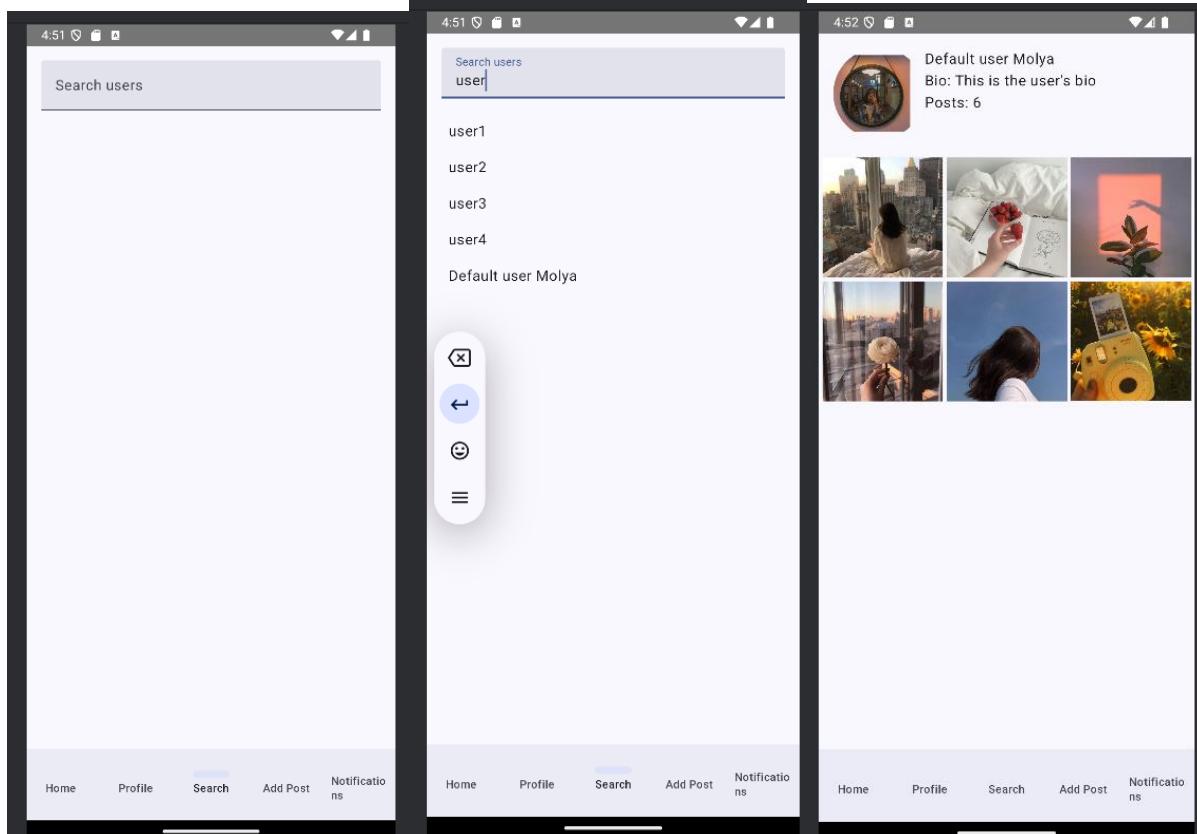
```

```
61  @Composable
62  fun ProfileHeader(username: String) {
63      Row(
64          modifier = Modifier
65              .fillMaxWidth()
66              .padding(16.dp),
67          horizontalArrangement = Arrangement.Start
68      ) {
69          Image(
70              painter = painterResource(id = R.drawable.profile_picture),
71              contentDescription = "Profile Picture",
72              modifier = Modifier
73                  .size(100.dp)
74                  .clip(CircleShape) // Clip the image to be circular
75                  .padding(end = 16.dp)
76          )
77          Column {
78              Text(text = username)
79              Text(text = "Bio: This is the user's bio")
80              Text(text = "Posts: ${posts.count { it.username == username }}")
81          }
82      }
83  }
```

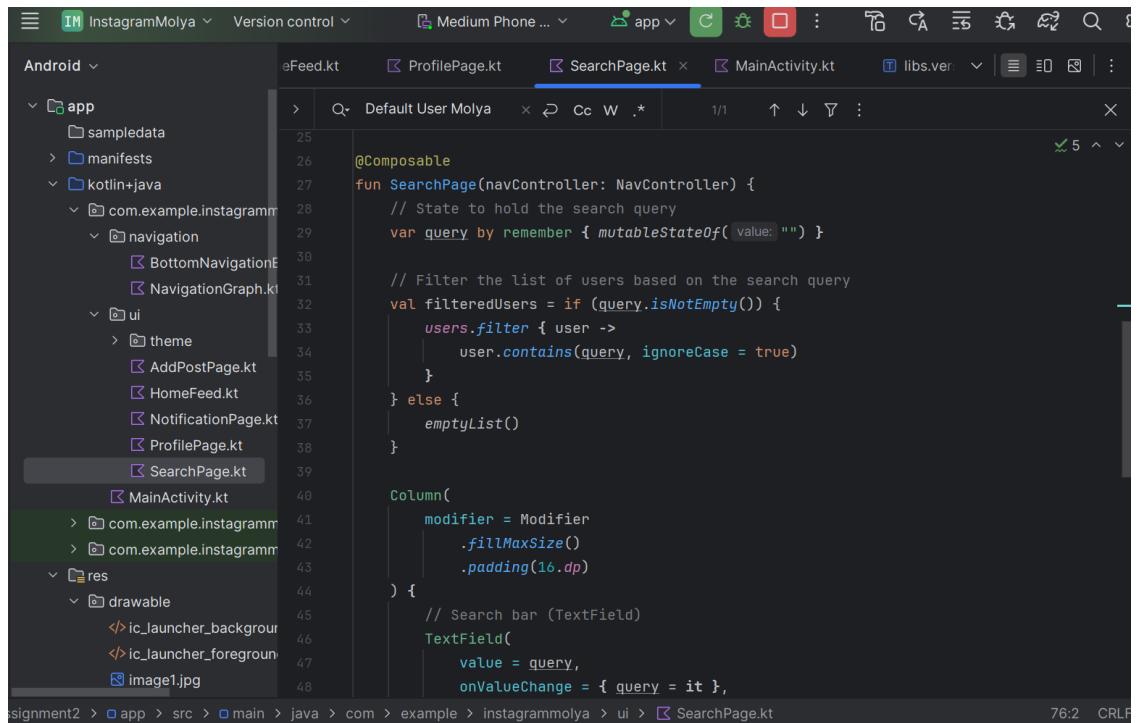
*See file *ProfilePage.kt* in ui folder.

3. Search page*

This page allows users to search for other user profiles. Clicking on a username navigates to certain user's profile page.



Layout consists of 2 parts: Search bar, filtered user list. When query is not empty, it filters users' list based on search query.



```

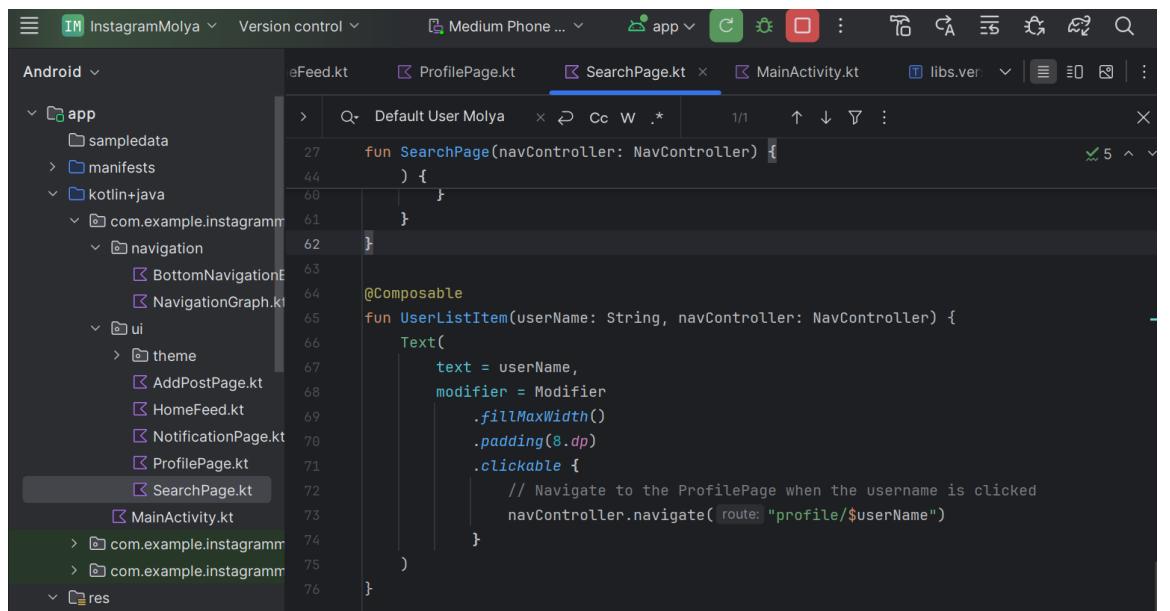
@Composable
fun SearchPage(navController: NavController) {
    // State to hold the search query
    var query by remember { mutableStateOf("") }

    // Filter the list of users based on the search query
    val filteredUsers = if (query.isNotEmpty()) {
        users.filter { user ->
            user.contains(query, ignoreCase = true)
        }
    } else {
        emptyList()
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        // Search bar (TextField)
        TextField(
            value = query,
            onValueChange = { query = it },
        )
        ...
    }
}

```

UserListItem displays each individual user in the search result list and makes their name clickable. Clicking on the username navigates to that user's profile page.



```

@Composable
fun UserListItem(userName: String, navController: NavController) {
    Text(
        text = userName,
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp)
            .clickable {
                // Navigate to the ProfilePage when the username is clicked
                navController.navigate(route: "profile/$userName")
            }
    )
}

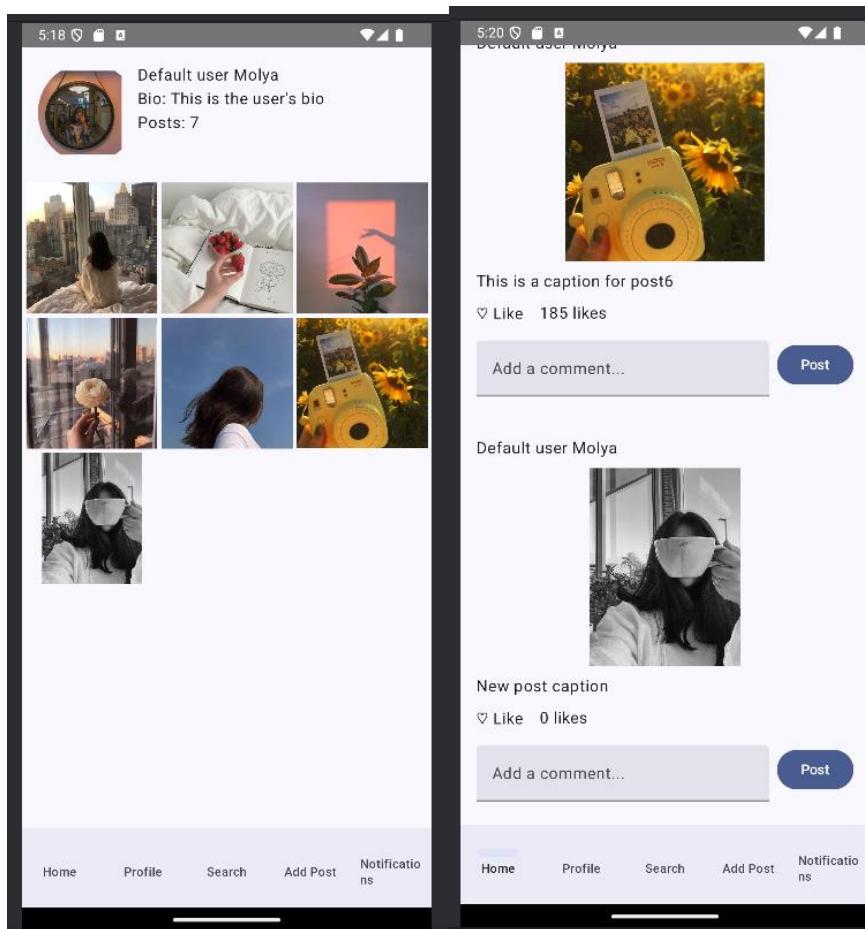
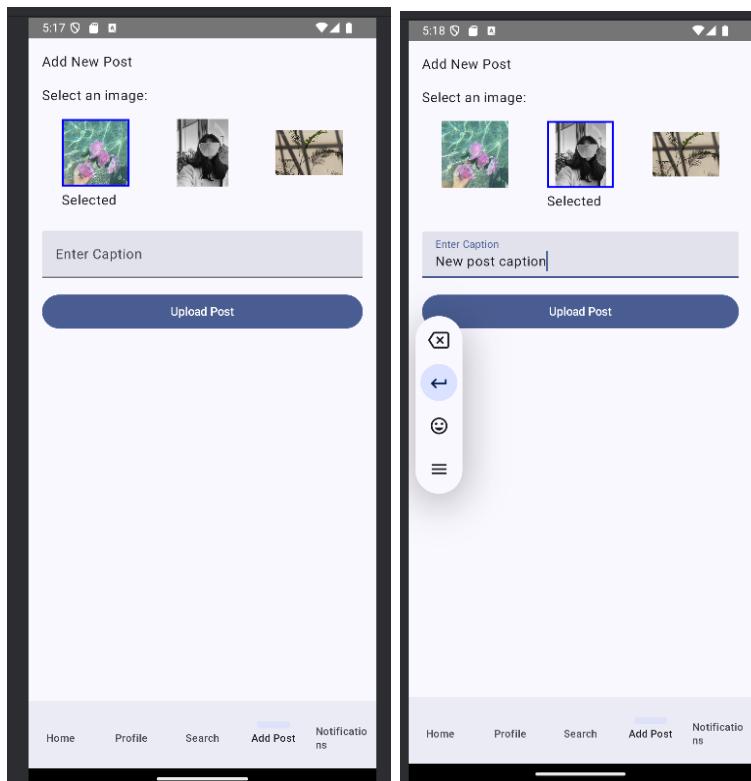
```

*See file *SearchPage.kt* in ui folder.

4. Add Post Page*

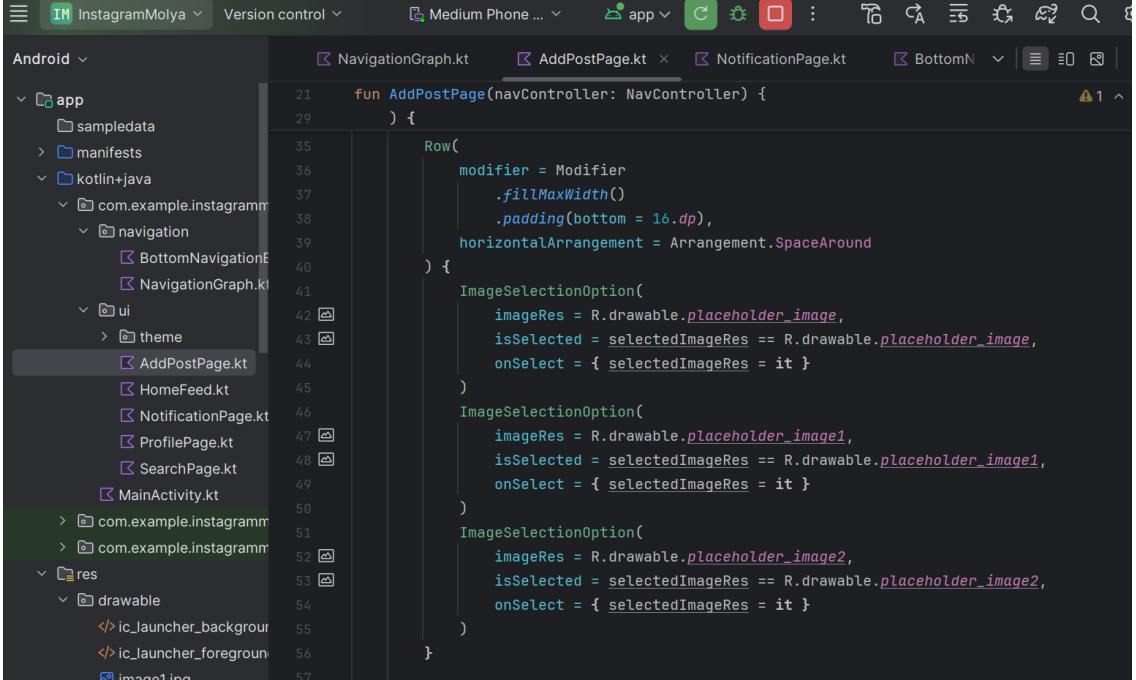
This page allows users to create new posts by selecting an image, entering a caption, and submitting the post. After uploading the post, it navigates to Profile

page of default user and the new post is there. We can view it at Home Feed page as well.



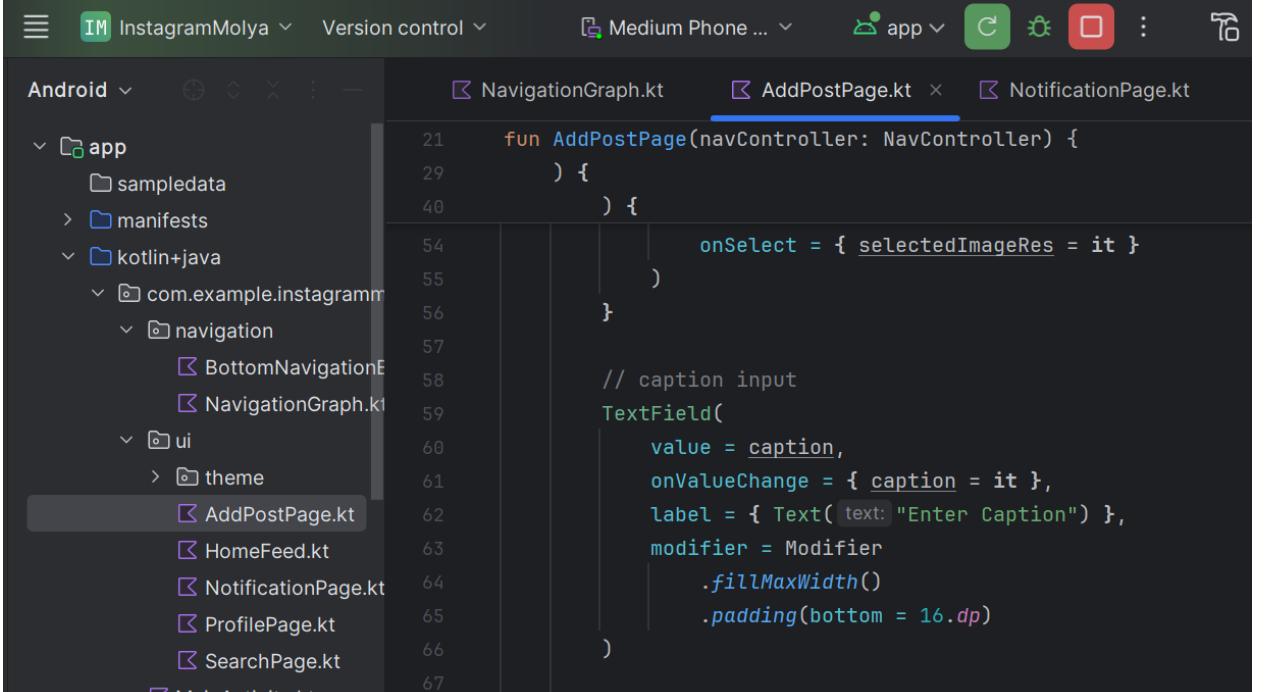
Users can click on any of the available image options, and the selected image resource is stored in the state variable `selectedImageRes`.

`ImageSelectionOption` composable is responsible for displaying each image option. Each image is clickable, and clicking on an image updates the selected image resource.



```
fun AddPostPage(navController: NavController) {  
    Row(  
        modifier = Modifier  
            .fillMaxWidth()  
            .padding(bottom = 16.dp),  
        horizontalArrangement = Arrangement.SpaceAround  
) {  
    ImageSelectionOption(  
        imageRes = R.drawable.placeholder_image,  
        isSelected = selectedImageRes == R.drawable.placeholder_image,  
        onSelect = { selectedImageRes = it }  
    )  
    ImageSelectionOption(  
        imageRes = R.drawable.placeholder_image1,  
        isSelected = selectedImageRes == R.drawable.placeholder_image1,  
        onSelect = { selectedImageRes = it }  
    )  
    ImageSelectionOption(  
        imageRes = R.drawable.placeholder_image2,  
        isSelected = selectedImageRes == R.drawable.placeholder_image2,  
        onSelect = { selectedImageRes = it }  
    )  
}
```

There is text input field (`TextField`) for users to enter a caption for their post. The entered caption is stored in the `caption` state variable.



```
fun AddPostPage(navController: NavController) {  
    ) {  
    ) {  
        onSelect = { selectedImageRes = it }  
    )  
    // caption input  
    TextField(  
        value = caption,  
        onValueChange = { caption = it },  
        label = { Text(text: "Enter Caption") },  
        modifier = Modifier  
            .fillMaxWidth()  
            .padding(bottom = 16.dp)  
    )  
}
```

The newly created post is added to global list of posts (`posts`), so it will be displayed in both the Home feed and the user's Profile page(default user's).

```

21     fun AddPostPage(navController: NavController) {
22         ...
23     }
24
25         // upload button
26         Button(
27             onClick = {
28                 val newPost = Post(
29                     username = currentUser,
30                     imageRes = selectedImageRes,
31                     caption = caption,
32                     likes = 0
33                 )
34
35                 posts.add(newPost) // Add to global list
36
37                 // Navigate to Default user profile after posting
38                 navController.navigate(route: "profile/${currentUser}") {
39                     popUpTo(navController.graph.startDestinationId) {
40                         inclusive = true
41                     }
42                 },
43                 modifier = Modifier.fillMaxWidth()
44             }
45         )
46
47         Text(text = "Upload Post")
48     }

```

ImageSelectionOption displays available images with a clickable option and highlights the currently selected image.

```

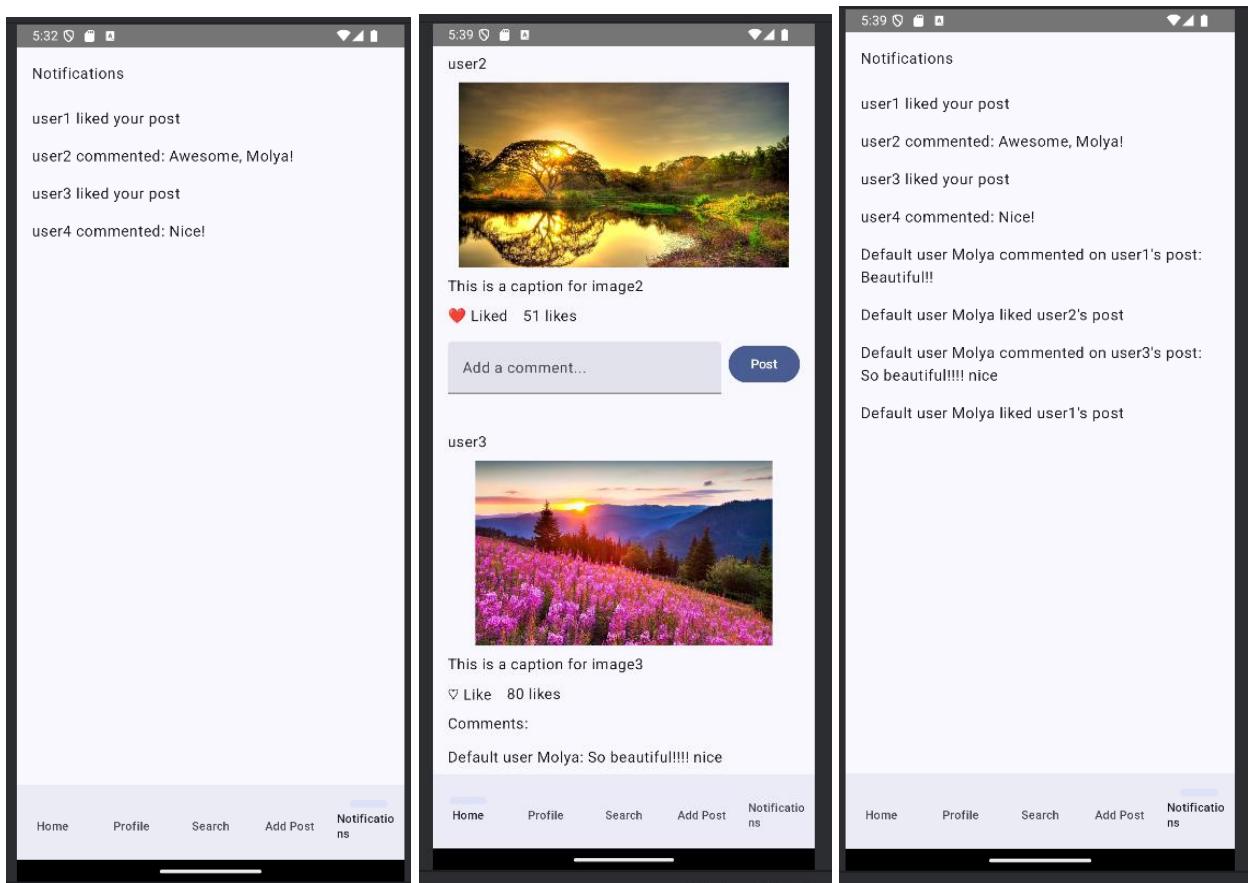
92
93     @Composable
94     fun ImageSelectionOption(imageRes: Int, isSelected: Boolean, onSelect: (Int) -> Unit) {
95
96         Column(
97             modifier = Modifier
98                 .clickable { onSelect(imageRes) }
99                 .padding(8.dp)
100        ) {
101
102             Image(
103                 painter = painterResource(id = imageRes),
104                 contentDescription = null,
105                 modifier = Modifier
106                     .size(80.dp)
107                     .border(2.dp, if (isSelected) Color.Blue else Color.Transparent) // Highlight
108             )
109             if (isSelected) {
110                 Text(text = "Selected", modifier = Modifier.padding(top = 4.dp))
111             }
112         }
113     }

```

*See file *AddPostPage.kt* in ui folder.

5. Notifications Page

This page displays all notifications. Notifications inform users of actions such as likes and comments on their posts. The list of notifications (*LazyColumn*) is displayed in a vertically scrollable layout. Each notification shows the username of the user who liked or commented and a corresponding message. All actions that are made at Home Feed page such like or comment are reflected in this page.



NotificationItem shows the username of the user who triggered the notification and the corresponding message.

```

Android
app
navigation
ui
com.example.instagramm
AddPostPage.kt
BottomNavigationBar.kt
BottomNavigationE
NavigationGraph.kt
NotificationPage.kt
ProfilePage.kt
SearchPage.kt
MainActivity.kt
Main.kt
com.example.instagramm
com.example.instagramm
res

fun NotificationPage(navController: NavController) {
    LazyColumn(modifier = Modifier.fillMaxSize()) {
        NotificationItem(notification = notification)
    }
}

@Composable
fun NotificationItem(notification: Notification) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 8.dp)
    ) {
        Text(text = "${notification.username} ${notification.message}")
    }
}

```

Class *Notification* stores all information about notification. The global *notifications* list stores all notifications. This list is used to populate the LazyColumn on the Notification Page. It can be updated when user executes certain actions.

The screenshot shows the Android Studio interface with the project navigation graph on the left and the code editor on the right. The code editor displays the `NotificationPage.kt` file, which contains the following code:

```
data class Notification(
    val type: String,
    val username: String,
    val message: String
)

val notifications = mutableListOf(
    Notification(type = "like", username = "user1", message = "liked your post"),
    Notification(type = "comment", username = "user2", message = "commented: Awesome, Molya!"),
    Notification(type = "like", username = "user3", message = "liked your post"),
    Notification(type = "comment", username = "user4", message = "commented: Nice!")
)

@Composable
fun NotificationPage(navController: NavController) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Text(text = "Notifications", modifier = Modifier.padding(bottom = 16.dp))

        // LazyColumn to display notifications
        LazyColumn(modifier = Modifier.fillMaxSize()) {
            items(notifications) { notification ->

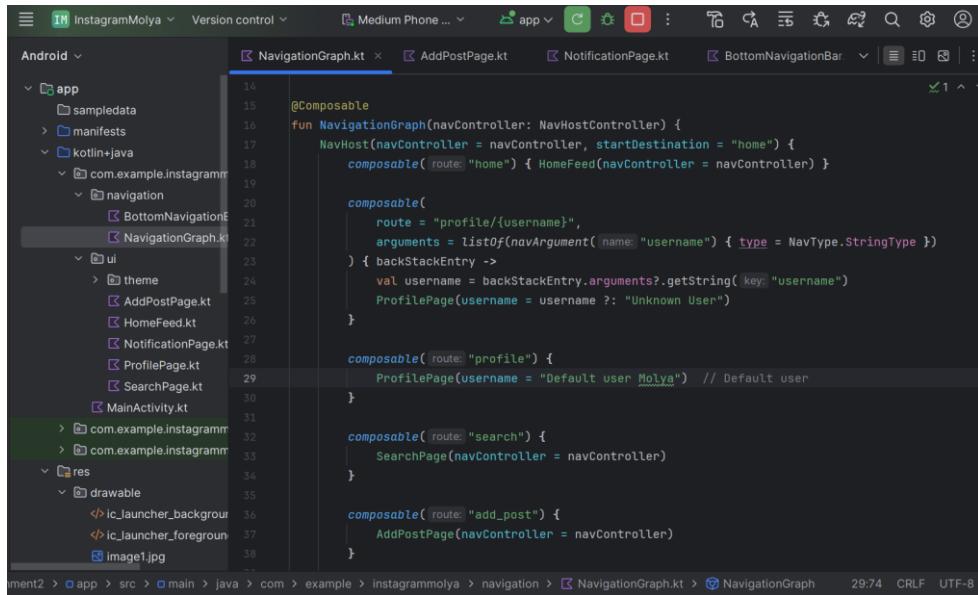
```

*See file `NotificationPage.kt` in ui folder.

Navigation

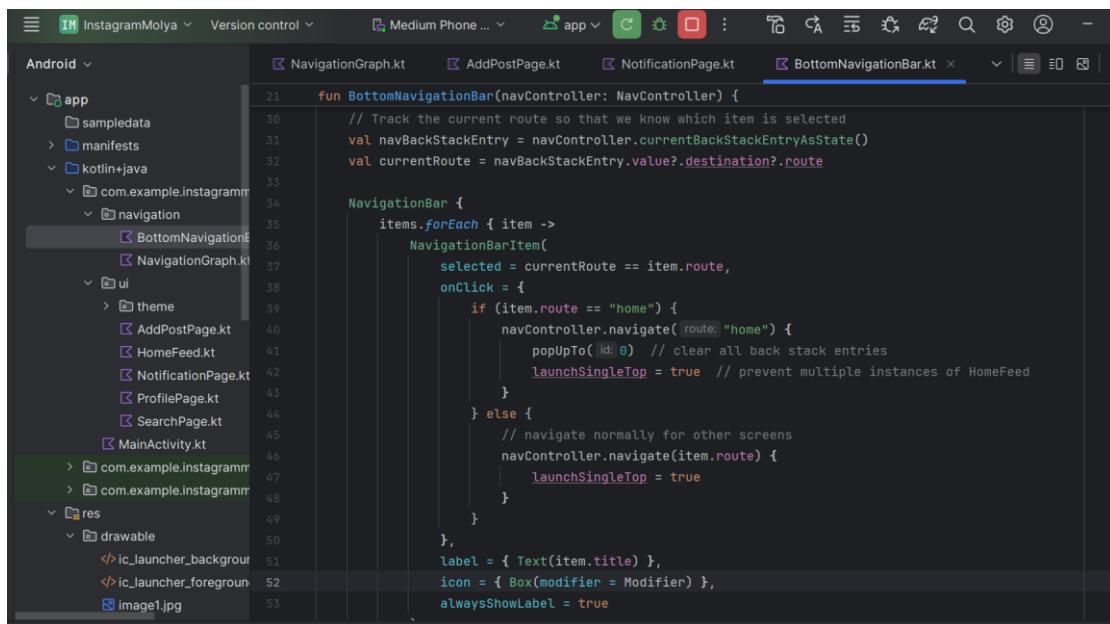
Jetpack Compose's Navigation Component handles the navigation in this project. *NavigationGraph* function is responsible for defining the different screens. Each route corresponds to a specific composable screen, and navigation between these routes is handled by the *NavController*. Start screen is Home.

Each screen is defined by a *composable* function, which associates a route with a composable function. Profile page has dynamic routes.



```
14
15 @Composable
16 fun NavigationGraph(navController: NavController) {
17     NavHost(navController = navController, startDestination = "home") {
18         composable(route: "home") { HomeFeed(navController = navController) }
19
20         composable(
21             route = "profile/{username}",
22             arguments = listOf(navArgument(name = "username") { type = NavType.StringType })
23         ) { backStackEntry ->
24             val username = backStackEntry.arguments?.getString(key: "username")
25             ProfilePage(username = username ?: "Unknown User")
26         }
27
28         composable(route: "profile") {
29             ProfilePage(username = "Default user Molya") // Default user
30         }
31
32         composable(route: "search") {
33             SearchPage(navController = navController)
34         }
35
36         composable(route: "add_post") {
37             AddPostPage(navController = navController)
38     }
39
40     
```

BottomNavigationBar provides way to switch between the main screens easily. Each item in the bottom navigation corresponds to specific route in the *NavigationGraph*. *NavController* handling the *Home Page* ensures the back stack is cleared when returning to the home feed.



```
21 fun BottomNavigationBar(navController: NavController) {
22     // Track the current route so that we know which item is selected
23     val navBackStackEntry = navController.currentBackStackEntryAsState()
24     val currentRoute = navBackStackEntry.value?.destination?.route
25
26     NavigationBar {
27         items.forEach { item ->
28             NavigationBarItem(
29                 selected = currentRoute == item.route,
30                 onClick = {
31                     if (item.route == "home") {
32                         navController.navigate(route: "home") {
33                             popUpTo(id: 0) // clear all back stack entries
34                             launchSingleTop = true // prevent multiple instances of HomeFeed
35                         }
36                     } else {
37                         // navigate normally for other screens
38                         navController.navigate(item.route) {
39                             launchSingleTop = true
40                         }
41                     }
42                 },
43                 label = { Text(item.title) },
44                 icon = { Box(modifier = Modifier) },
45                 alwaysShowLabel = true
46             )
47         }
48     }
49 }
```

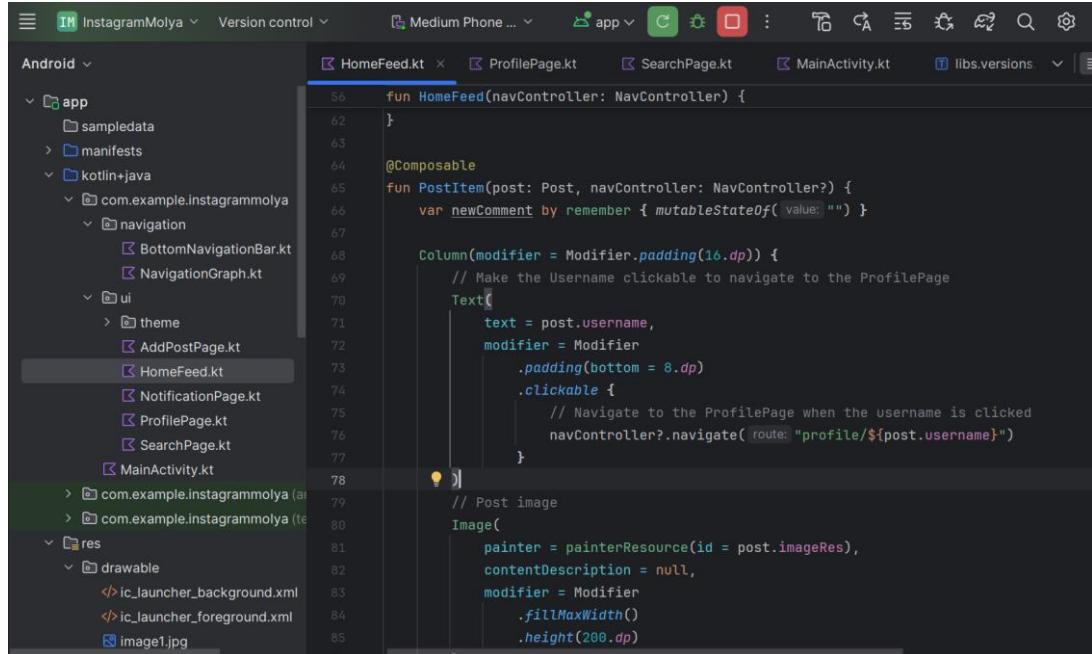
*See *NavigationGraph.kt* and *BottomNavigationBar.kt* files in ui/navigation folder.

User Interaction

Each interaction is handled using composable functions and `Modifier.clickable` to make UI elements interactive.

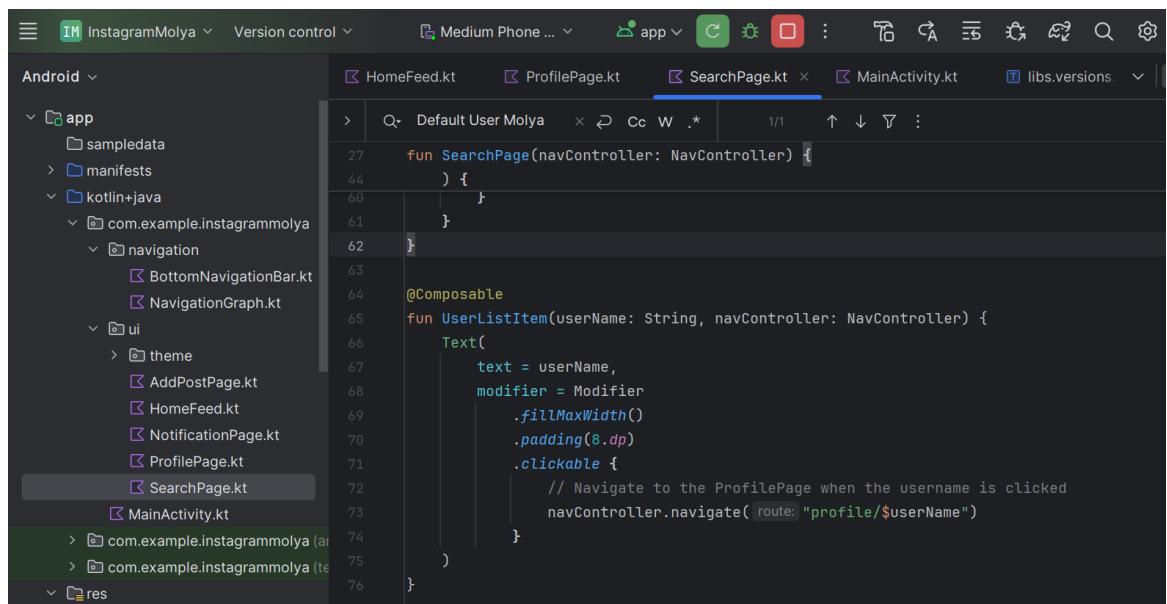
Main user interaction points:

1. Clicking on a username to view a profile (Home Feed and Search).



```
fun PostItem(post: Post, navController: NavController?) {
    var newComment by remember { mutableStateOf("") }

    Column(modifier = Modifier.padding(16.dp)) {
        // Make the Username clickable to navigate to the ProfilePage
        Text(
            text = post.username,
            modifier = Modifier
                .padding(bottom = 8.dp)
                .clickable {
                    // Navigate to the ProfilePage when the username is clicked
                    navController?.navigate("profile/${post.username}")
                }
        )
        // Post image
        Image(
            painter = painterResource(id = post.imageRes),
            contentDescription = null,
            modifier = Modifier
                .fillMaxWidth()
                .height(200.dp)
        )
    }
}
```



```
fun UserListItem(userName: String, navController: NavController) {
    Text(
        text = userName,
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp)
            .clickable {
                // Navigate to the ProfilePage when the username is clicked
                navController.navigate("profile/$userName")
            }
    )
}
```

2. Liking a post (Home Feed)

```
fun PostItem(post: Post, navController: NavController?) {
    Column(modifier = Modifier.padding(16.dp)) {
        // Caption and Likes
        Text(text = post.caption, modifier = Modifier.padding(top = 8.dp))
        Row(modifier = Modifier.padding(vertical = 8.dp)) {
            // Like button
            Text(
                text = if (post.likedByUser) "Heart Liked" else "Heart Like",
                modifier = Modifier.clickable {
                    if (!post.likedByUser) {
                        post.likedByUser = true
                        post.likes += 1
                        // Add a like notification
                        notifications.add(
                            Notification(
                                type = "like",
                                username = currentUser,
                                message = "liked ${post.username}'s post"
                            )
                        )
                    } else {
                        post.likedByUser = false
                        post.likes -= 1
                    }
                }
            )
        }
    }
}
```

3. Adding a comment (Home Feed)

```
fun PostItem(post: Post, navController: NavController?) {
    Column(modifier = Modifier.padding(16.dp)) {
        // ...
        TextField(
            value = newComment,
            onValueChange = { newComment = it },
            placeholder = { Text("Add a comment...") },
            modifier = Modifier.weight(if)
        )
        Button(
            onClick = {
                if (newComment.isNotBlank()) {
                    post.comments.add(Comment(username = currentUser, comment = newComment))
                    notifications.add(
                        Notification(
                            type = "comment",
                            username = currentUser,
                            message = "commented on ${post.username}'s post: $newComment"
                        )
                    )
                }
                newComment = ""
            },
            modifier = Modifier.padding(start = 8.dp)
        )
    }
}
```

4. Uploading a new post (Add Post)

```
fun AddPostPage(navController: NavController) {
    // ...
    // Upload button
    Button(
        onClick = {
            val newPost = Post(
                username = currentUser,
                imageRes = selectedImageRes,
                caption = caption,
                likes = 0
            )
            posts.add(newPost) // Add to global list

            // Navigate to Default user profile after posting
            navController.navigate(route = "profile/${currentUser}") {
                popUpTo(navController.graph.startDestinationId) {
                    inclusive = true
                }
            }
        },
        modifier = Modifier.fillMaxWidth()
    ) {
        Text(text = "Upload Post")
    }
}
```

5. Bottom navigation bar clicks

```
fun BottomNavigationBar(navController: NavController) {  
    NavigationBar {  
        items.forEach { item ->  
            NavigationBarItem(  
                selected = currentRoute == item.route,  
                onClick = {  
                    if (item.route == "home") {  
                        navController.navigate(route: "home") {  
                            popUpTo(id: 0) // clear all back stack entries  
                            launchSingleTop = true // prevent multiple instances of HomeFeed  
                        }  
                    } else {  
                        // navigate normally for other screens  
                        navController.navigate(item.route) {  
                            launchSingleTop = true  
                        }  
                    }  
                },  
                label = { Text(item.title) },  
                icon = { Box(modifier = Modifier) },  
                alwaysShowLabel = true  
            )  
        }  
    }  
}
```

6. Image selection for the new post (Add Post)

```
fun AddPostPage(navController: NavController) {  
    ...  
}  
  
@Composable  
fun ImageSelectionOption(imageRes: Int, isSelected: Boolean, onSelect: (Int) -> Unit) {  
    Column(  
        modifier = Modifier  
            .clickable { onSelect(imageRes) }  
            .padding(8.dp)  
    ) {  
        Image(  
            painter = painterResource(id = imageRes),  
            contentDescription = null,  
            modifier = Modifier  
                .size(80.dp)  
                .border(2.dp, if (isSelected) Color.Blue else Color.Transparent) // Highlight selected  
        )  
        if (isSelected) {  
            Text(text = "Selected", modifier = Modifier.padding(top = 4.dp))  
        }  
    }  
}
```

Challenges and Solutions

During the development process faced some challenges.

One of the initial challenges was implementing dynamic navigation between user profiles, particularly when clicking on usernames in various parts of the app, such as in posts on the *Home Feed* or the *Search Page*. The complexity arose from the need to dynamically pass the username through the navigation system and render the correct profile based on that input. The solution was dynamic routing in Jetpack Compose's NavController. Username could be passed as an argument when navigating between screens.

```
@Composable
fun NavigationGraph(navController: NavHostController) {
    NavHost(navController = navController, startDestination = "home") {
        composable(route: "home") { HomeFeed(navController = navController) }

        composable(
            route = "profile/{username}",
            arguments = listOf(navArgument(name: "username") { type = NavType.StringType })
        ) { backStackEntry ->
            val username = backStackEntry.arguments?.getString(key: "username")
            ProfilePage(username = username ?: "Unknown User")
        }

        composable(route: "profile") {
            ProfilePage(username = "Default user Molya") // Default user
        }

        composable(route: "search") {
            SearchPage(navController = navController)
        }
    }
}
```

Another challenge was drawing user's post in Instagram like format. The standard vertical list (LazyColumn) was not suitable for this. The solution was to use *LazyVerticalGrid* from the Accompanist library, which let us display posts in a 3-column grid.

```

@Composable
fun UserPostsGrid(username: String) {
    val userPosts = posts.filter { it.username == username } // Filter posts by the username

    if (userPosts.isNotEmpty()) {
        LazyVerticalGrid(
            columns = GridCells.Fixed( count: 3), // 3 columns
            modifier = Modifier.fillMaxSize(),
            contentPadding = PaddingValues(4.dp),
            verticalArrangement = Arrangement.spacedBy(4.dp),
            horizontalArrangement = Arrangement.spacedBy(4.dp)
        ) {
            items(userPosts) { post ->
                Image(
                    painter = painterResource(id = post.imageRes),
                    contentDescription = null,
                    modifier = Modifier
                        .fillMaxWidth()
                        .aspectRatio( ratio: 1f) // Make images square
                )
            }
        }
    }
}

```

Managing state for user interactions such as likes and comments was problematic. Each post needed to track whether it was liked by the current user, dynamically update the like count, and display comments added by users. To manage the state efficiently, *remember* and *mutableStateOf* were used to keep track of likes, comments, and other post-specific states. Additionally, *Post* data class data structure was essential there.

```

@Composable
fun PostItem(post: Post, navController: NavController?) {
    var newComment by remember { mutableStateOf( value: "") }

    Column(modifier = Modifier.padding(16.dp)) {
        // Make the Username clickable to navigate to the ProfilePage
        Text(
            text = post.username,
            modifier = Modifier
                .padding(bottom = 8.dp)
                .clickable {
                    // Navigate to the ProfilePage when the username is clicked
                    navController?.navigate( route: "profile/${post.username}")
                }
        )
        // Post image
        Image(

```

Managing the navigation stack when using the *Bottom nav bar* was another challenge. The goal was to ensure that users could navigate back to the Home feed without creating multiple instances of the home screen in the back stack. The solution was to use the *popUpTo* and *launchSingleTop* options in the navigation logic.

```
NavigationBar {
    items.forEach { item ->
        NavigationBarItem(
            selected = currentRoute == item.route,
            onClick = {
                if (item.route == "home") {
                    navController.navigate(route: "home") {
                        popUpTo(id: 0) // clear all back stack entries
                        launchSingleTop = true // prevent multiple instances of HomeFeed
                    }
                } else {
                    // navigate normally for other screens
                    navController.navigate(item.route) {
                        launchSingleTop = true
                    }
                }
            },
            label = { Text(item.title) },
            icon = { Box(modifier = Modifier) },
            alwaysShowLabel = true
        )
    }
}
```

Conclusion

This assignment provided valuable hands-on experience in development of an Instagram-like mobile application using Jetpack Compose. We created from scratch very simple application, however the gained knowledge and insights are big. I've learned how to do page design, navigation between screens and interaction with different data, state management, grid layout.

UI/UX part of any application is one of the main parts, because it decides the overall experience. Interactive elements, navigation consistency, clean design, spacing these can affect the usability of the application.

References

<https://developer.android.com/jetpack/compose/documentation>

<https://developer.android.com/jetpack/compose/navigation>

<https://google.github.io/accompanist/>

<https://developer.android.com/jetpack/compose/state>

<https://developer.android.com/guide/navigation>

<https://developer.android.com/jetpack/compose/lists>

<https://chatgpt.com/> (data class creation, for issues resolution)