Kazakh British Technical University

Mobile Programming

Assignment II: Create Basic Pages of Instagram in Android Studio

by Moldir Polat

October 18th 2024

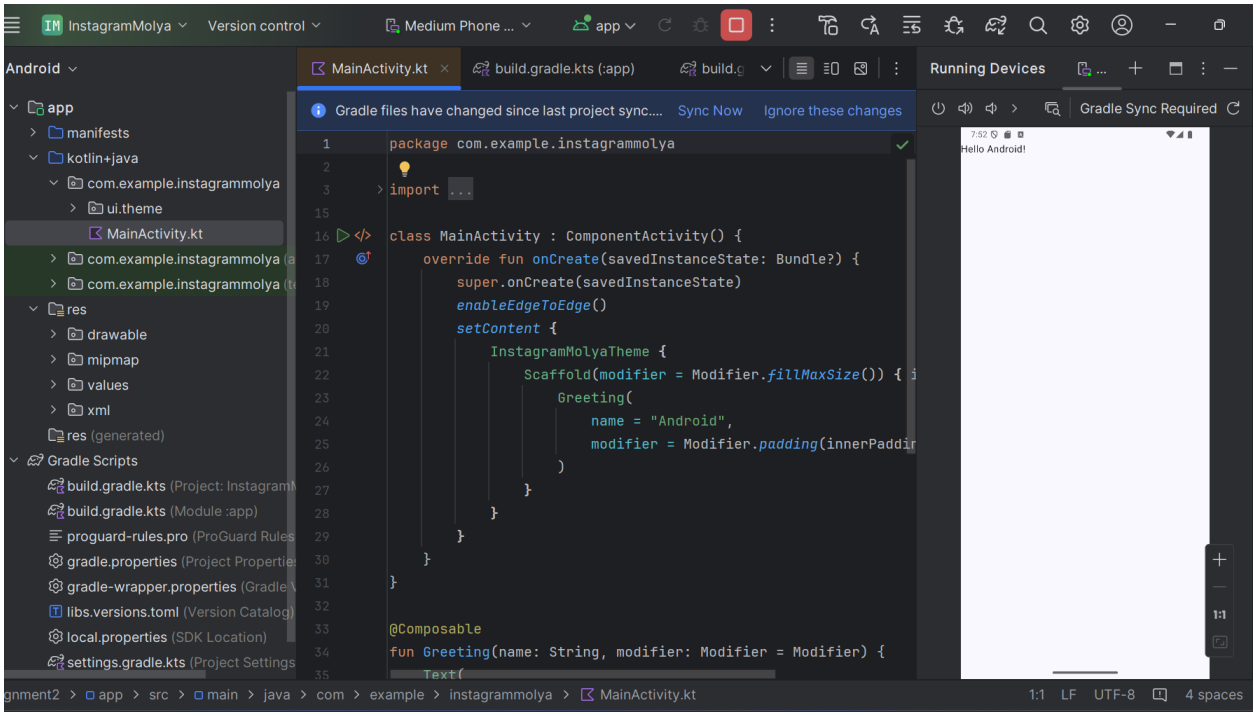
Table of contents

Introduction

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.

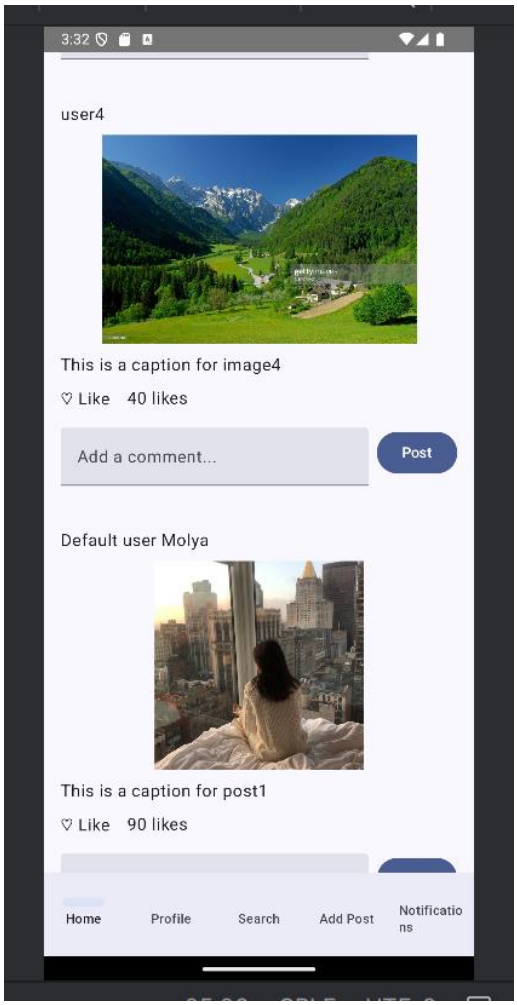
Project Setup

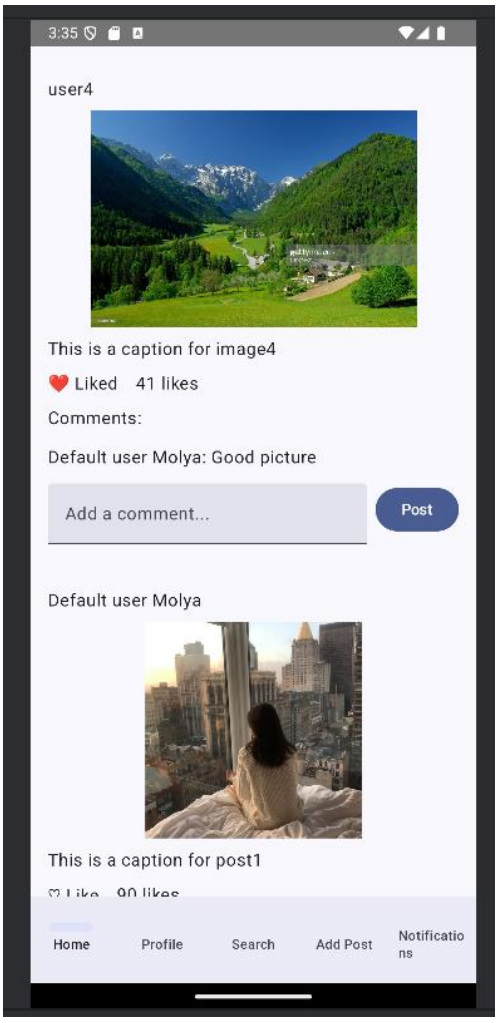
We created project from Empty Activity template.



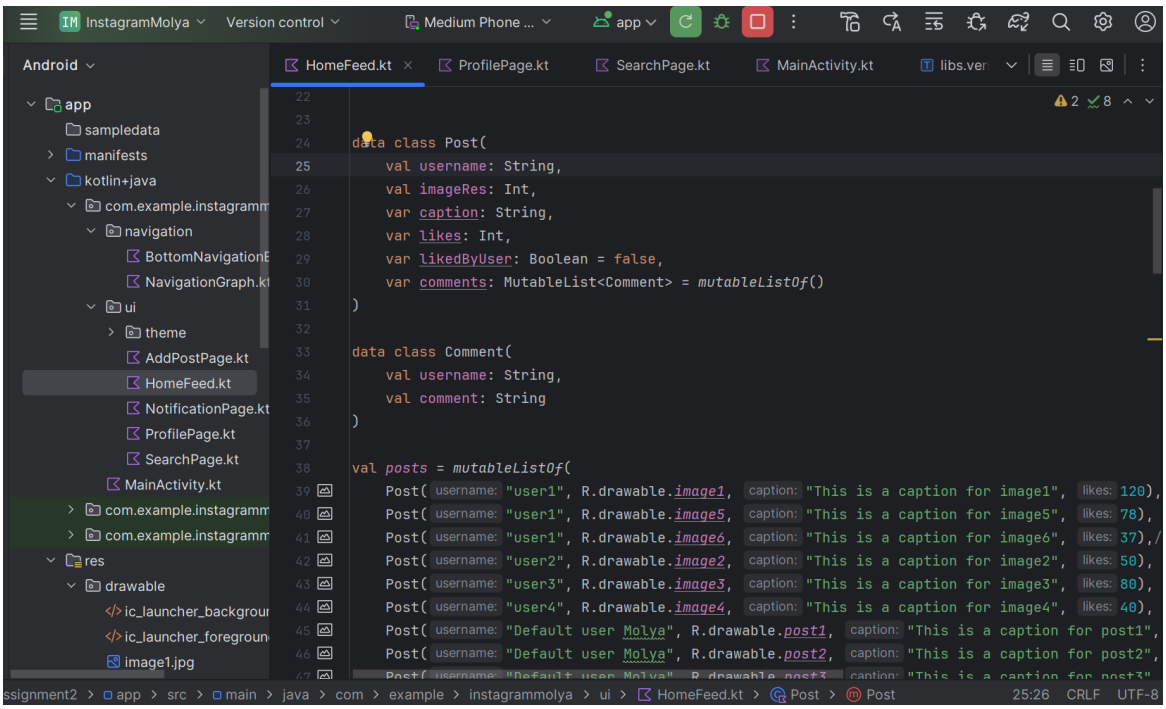
Page Design

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.





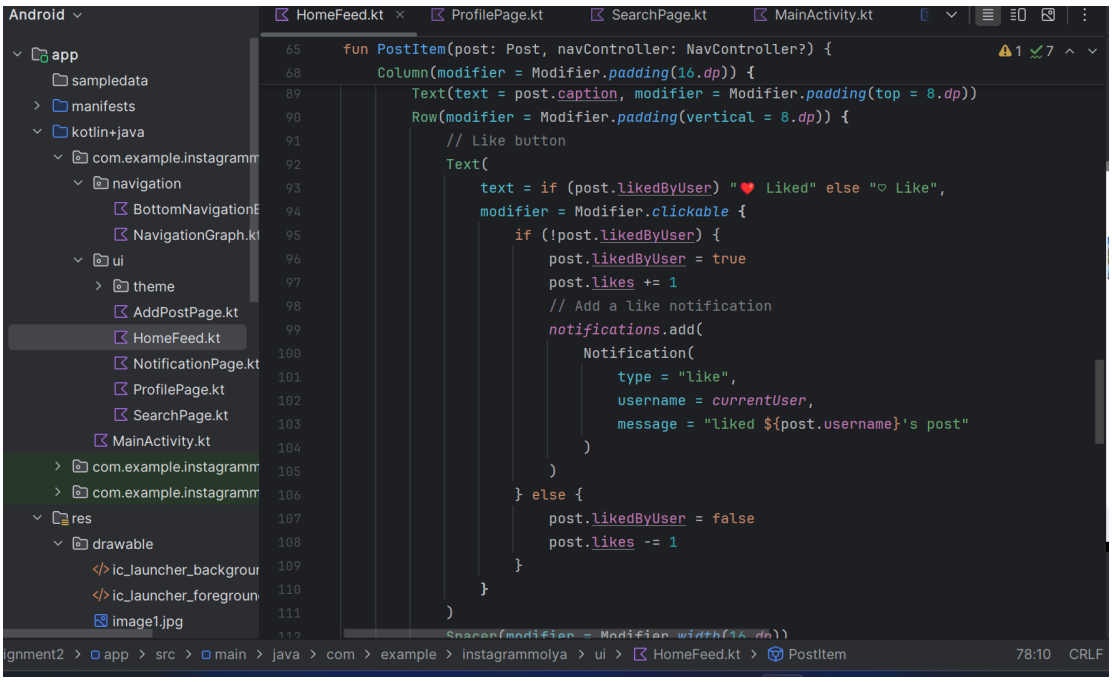
Data structure: there are two main classes – Post and Comment.



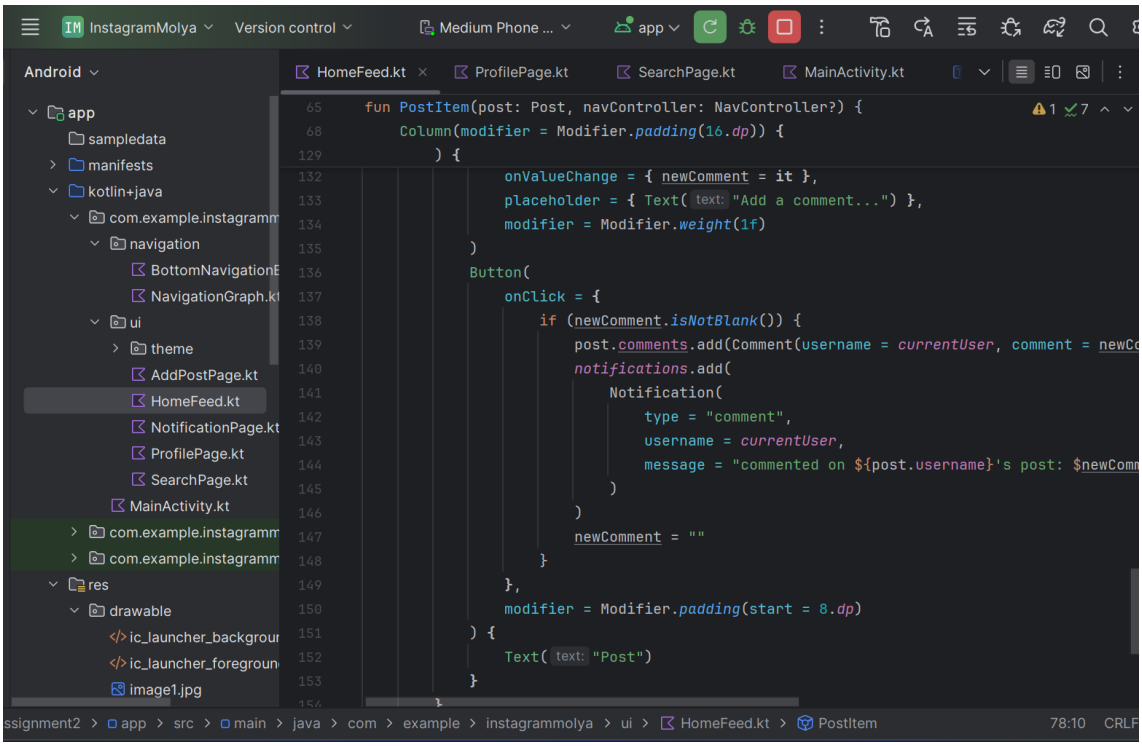
Layout: LazyColumn was used to display the lists of posts. Then **PostItem** represents objects in LazyColumn.

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

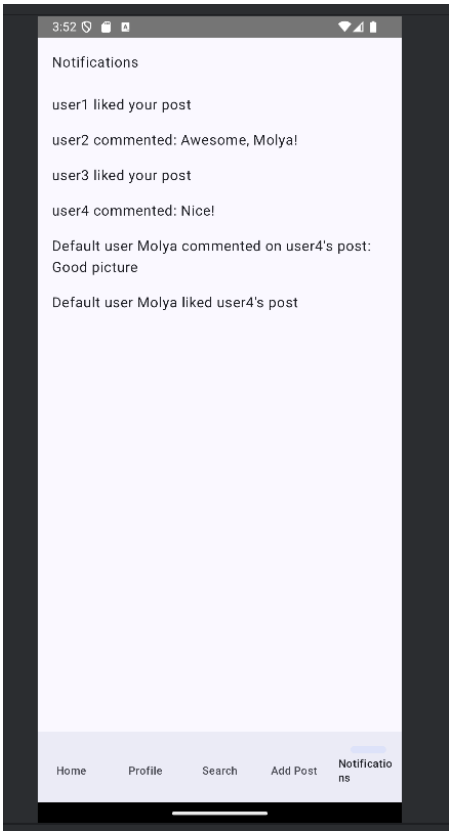
Users can click on the "♡ Like" text to like the post, which increments the likes count, changes the icon to "♡ Liked", and adds a notification that the post was liked by the current user.

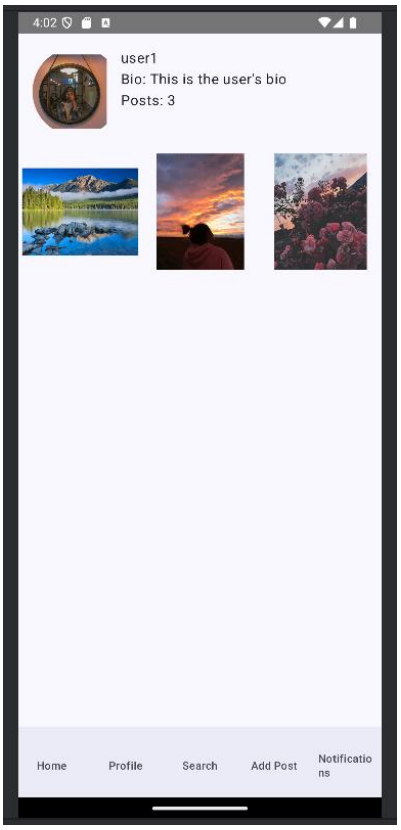


Users can add comment by typing into a TextField and clicking the "Post" button. The new comment is appended to the list of comments for that post, and a notification is generated for the comment action.



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).



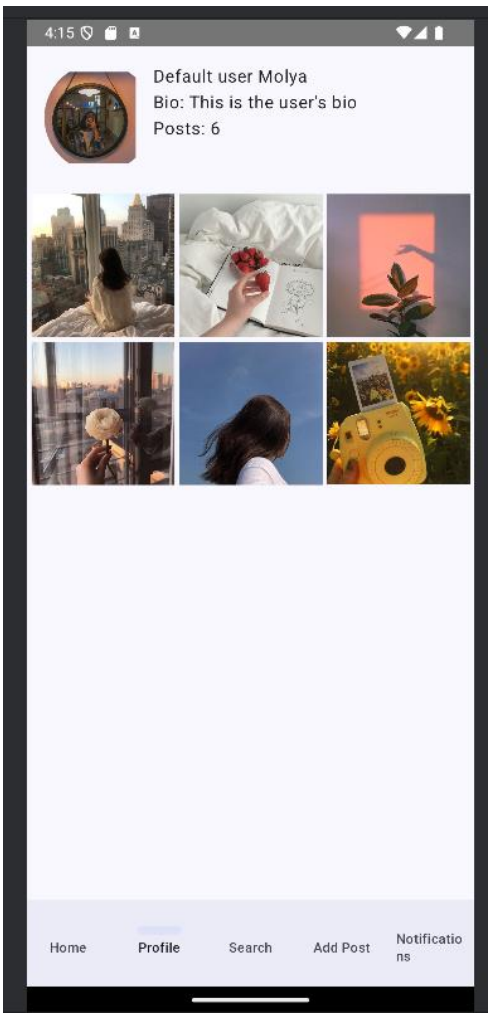


User can click on user's name and open his Profile page (navigates to Profile page)

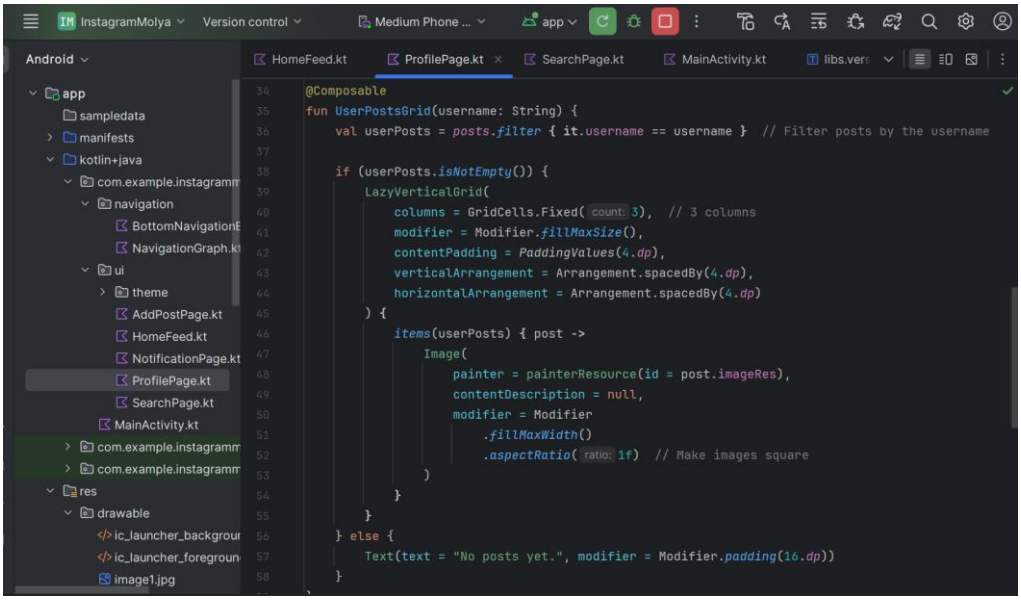
*See file *HomeFeed.kt* in ui folder.

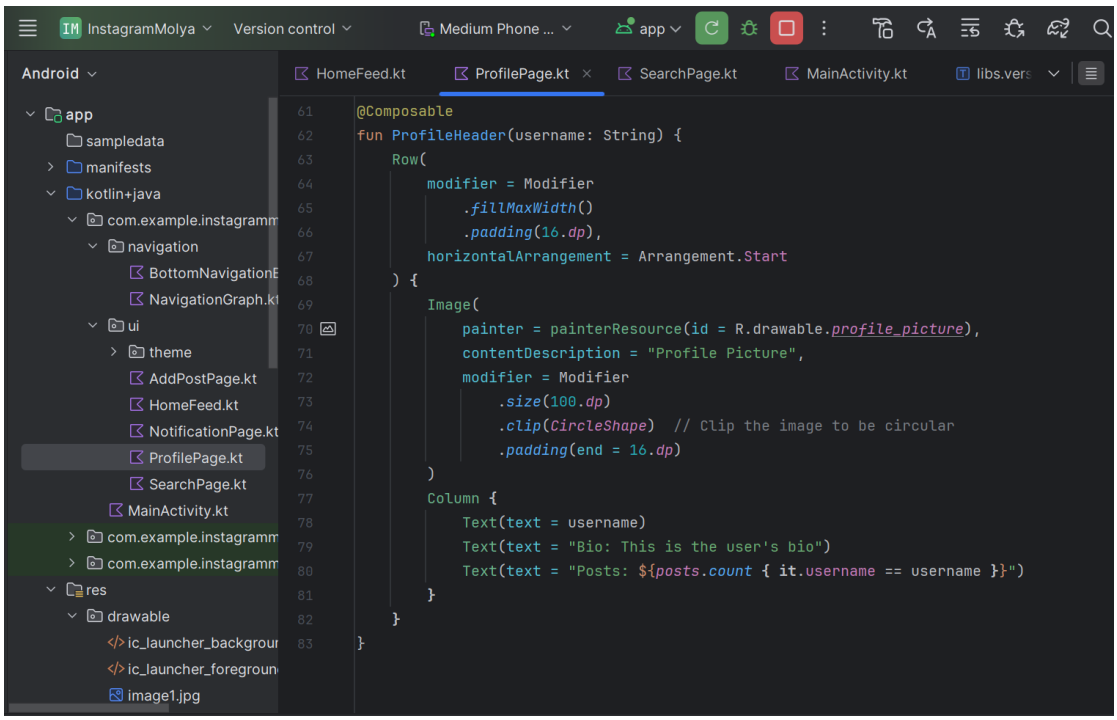
1. Profile page*

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.

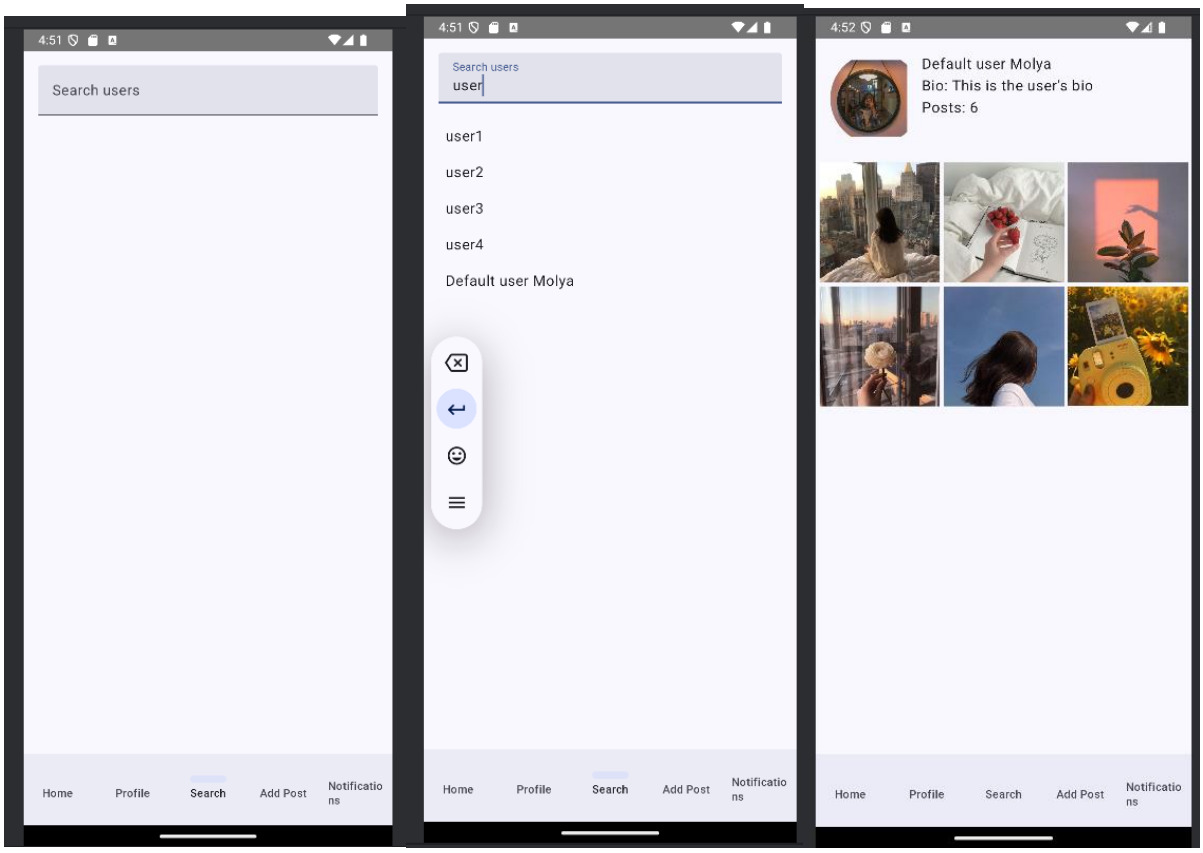




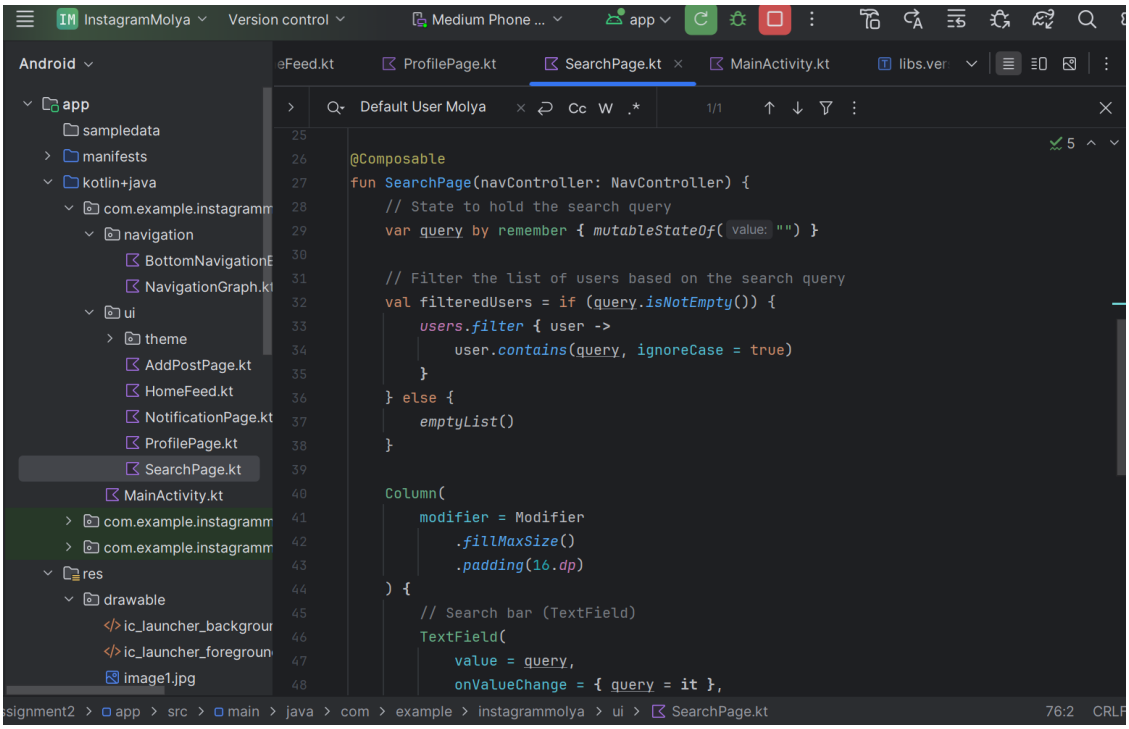
*See file *ProfilePage.kt* in ui folder.

3. Search page*

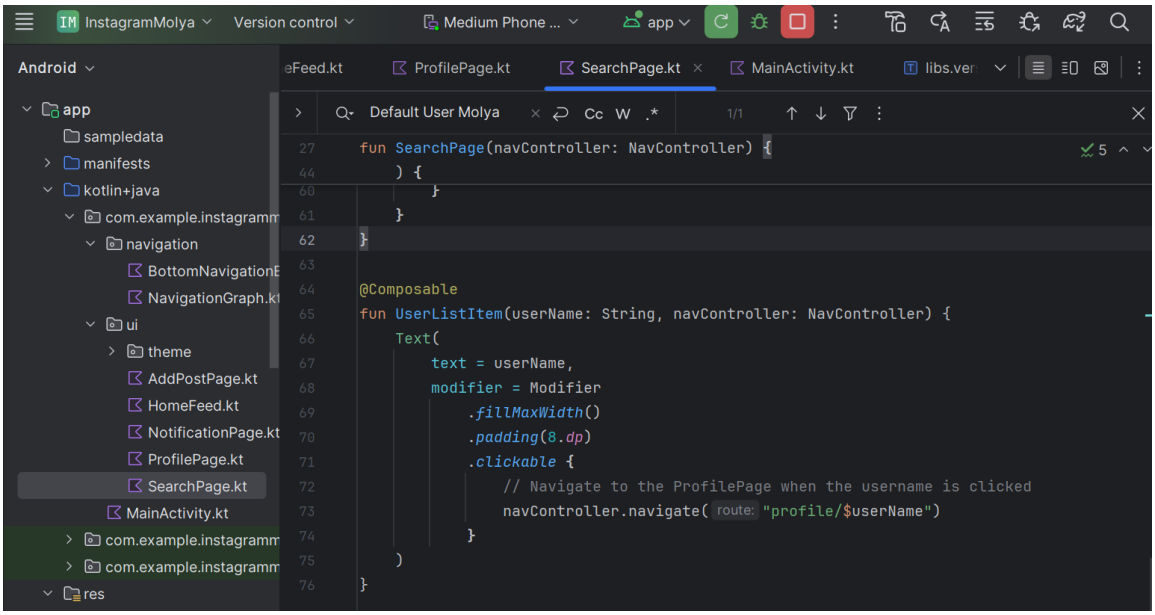
This page allows users to search for other user profiles. Clicking on a username navigates to certain user's profile page.



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



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.

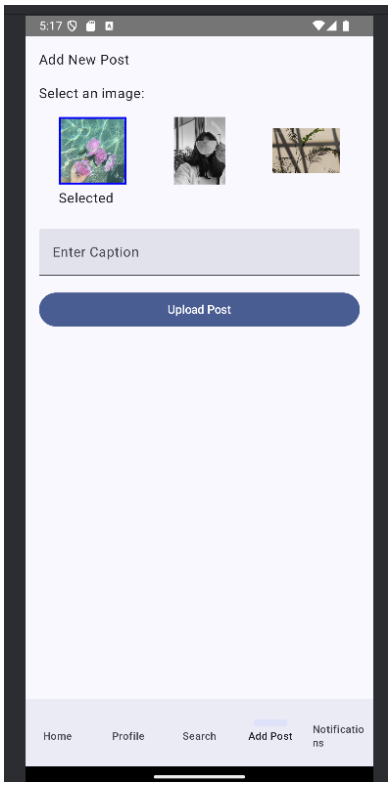


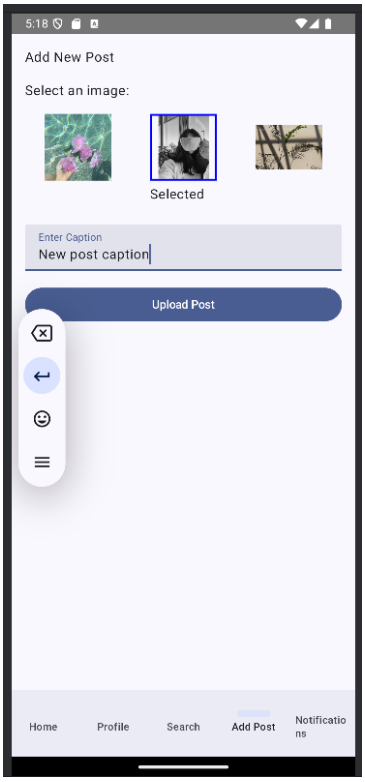
*See file *SearchPage.kt* in ui folder.

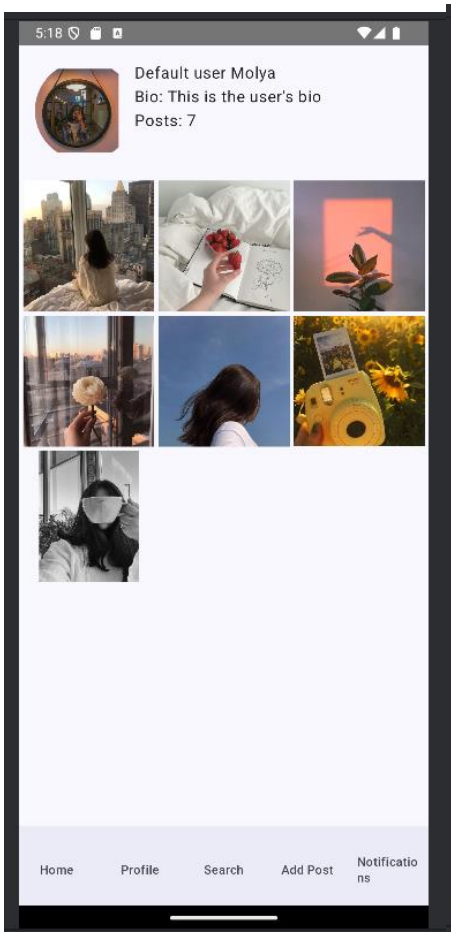
4. Add Post Page*

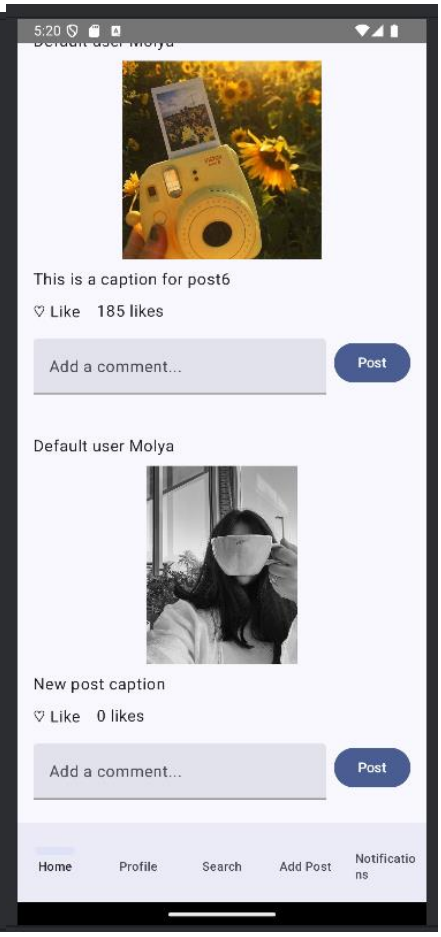
This page allows users to create new posts by selecting an image, entering a caption, and submitting the post. After uploading the post, it navigates to Profile

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

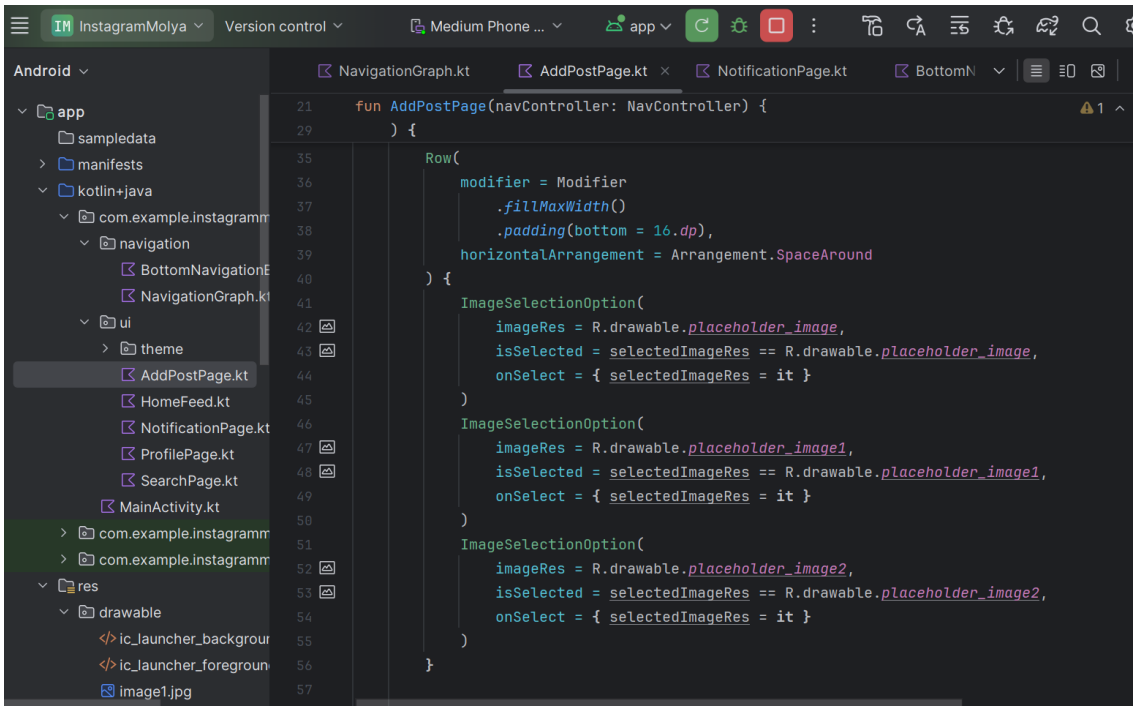




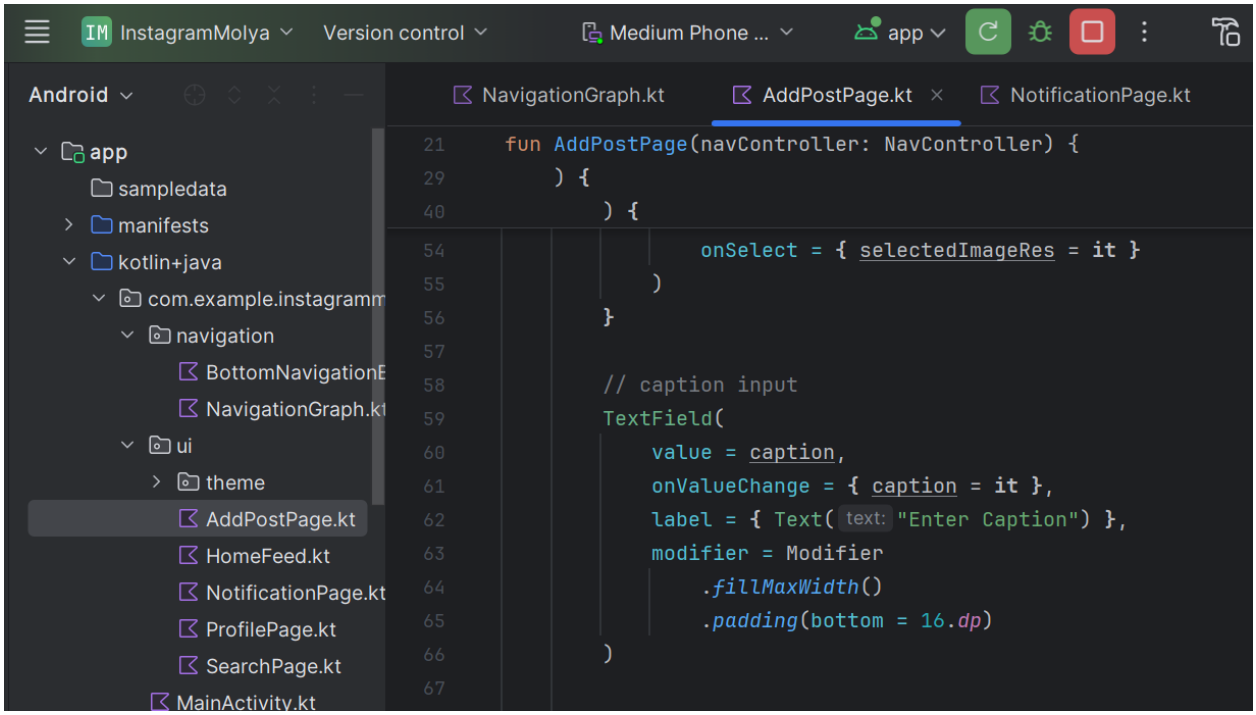




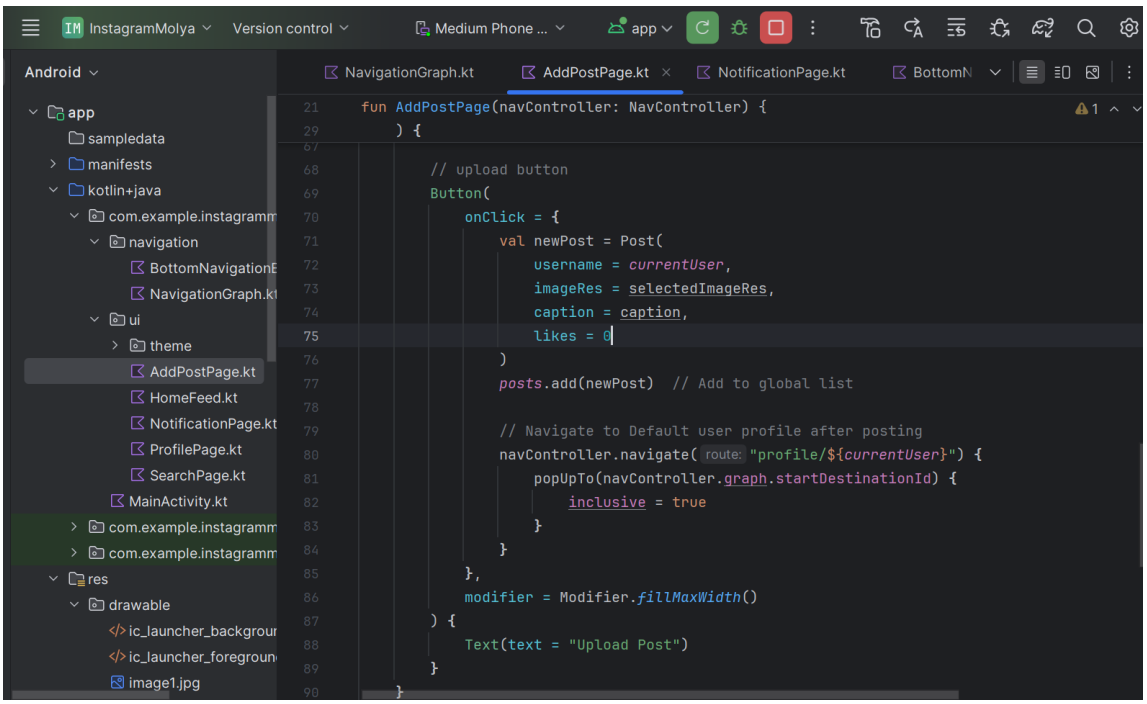
Users can click on any of the available image options, and the selected image resource is stored in the state variable *selectedImageRes*. *ImageSelectionOption* composable is responsible for displaying each image option. Each image is clickable, and clicking on an image updates the selected image resource.



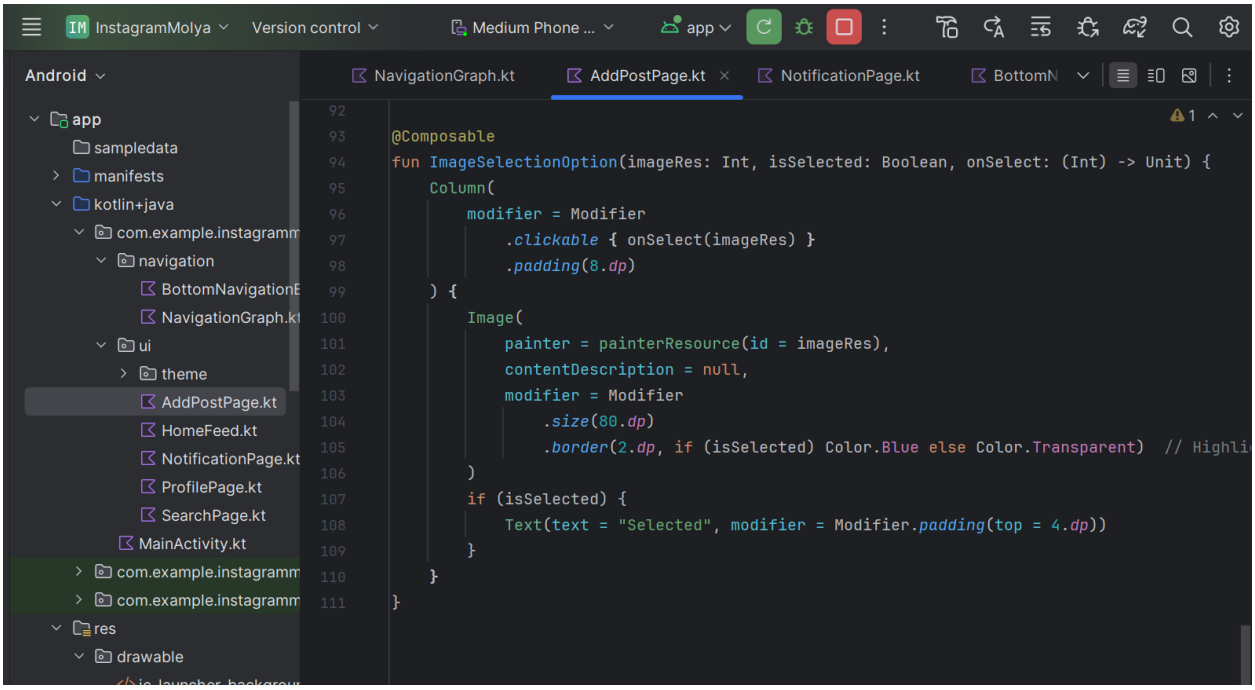
There is text input field (*TextField*) for users to enter a caption for their post. The entered caption is stored in the caption state variable.



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).



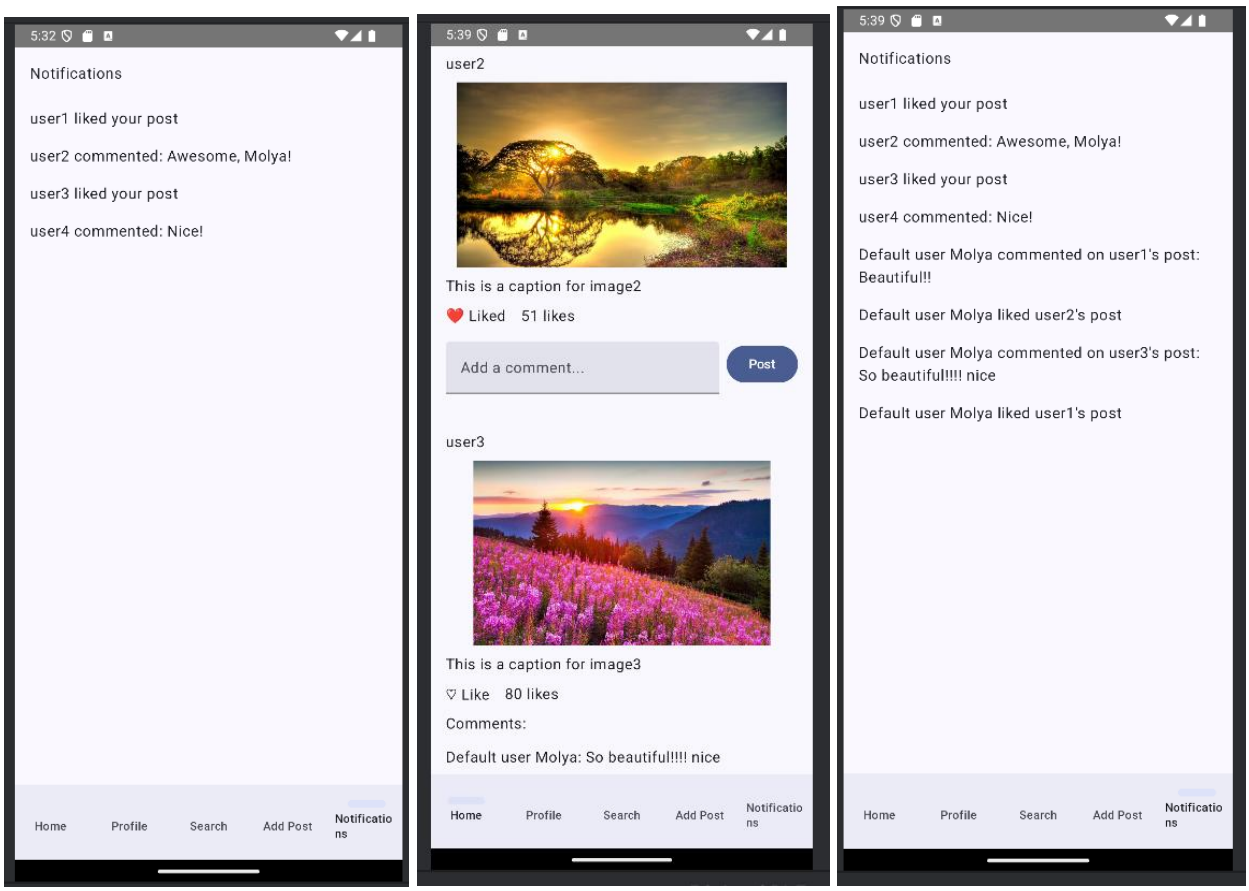
ImageSelectionOption displays available images with a clickable option and highlights the currently selected image.



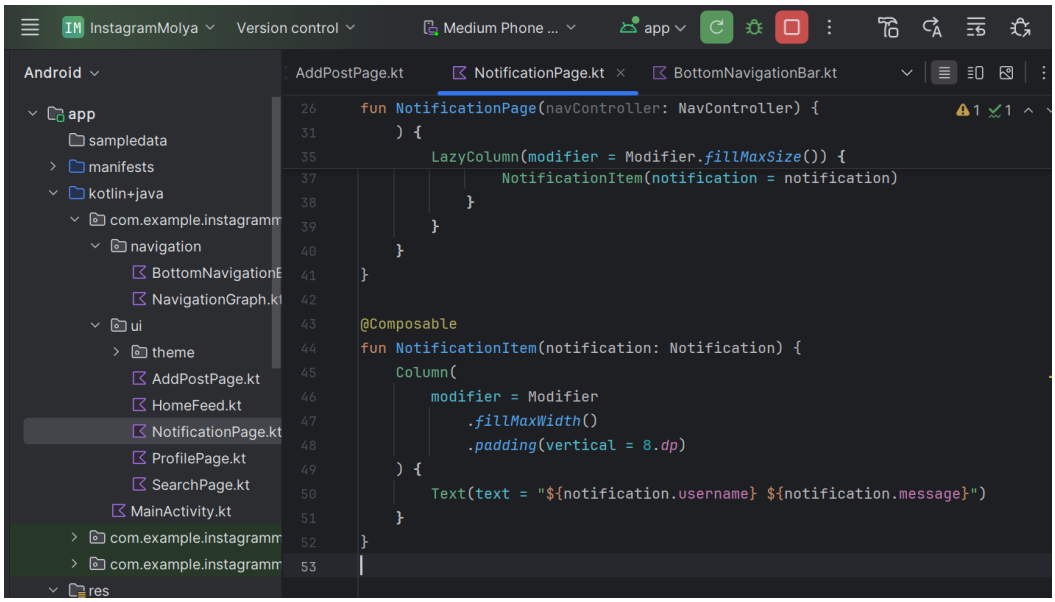
*See file *AddPostPage.kt* in ui folder.

5. Notifications Page

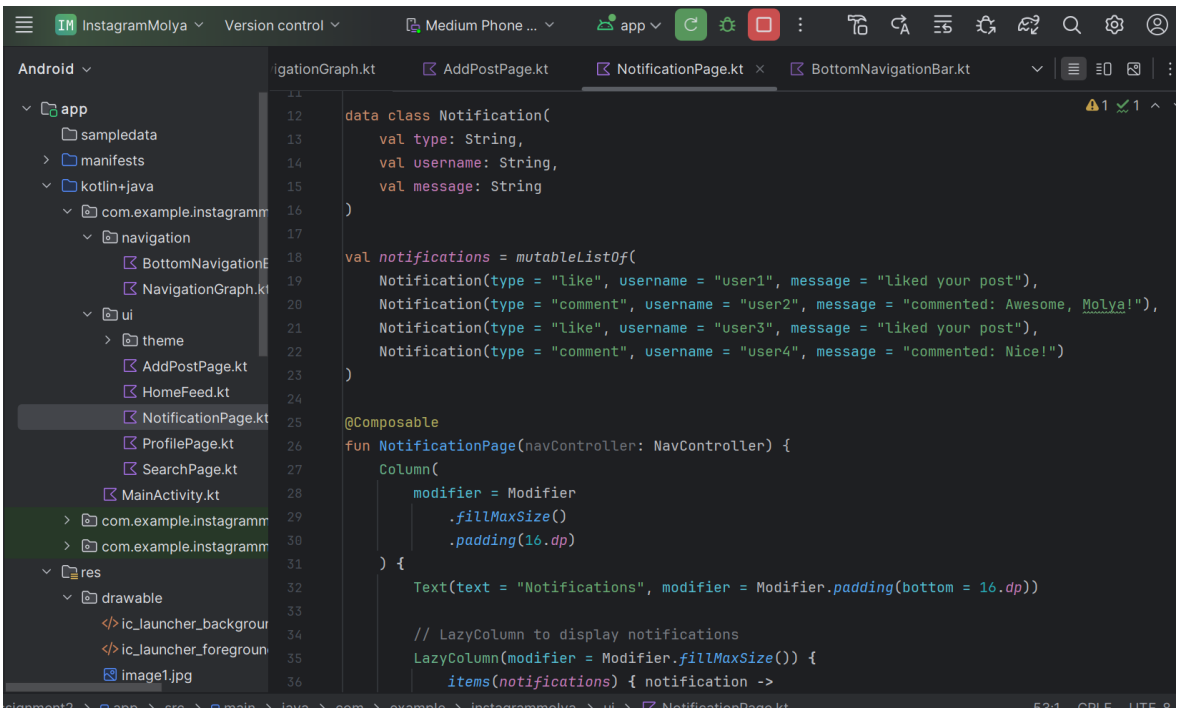
This page displays all notifications. Notifications inform users of actions such as likes and comments on their posts. The list of notifications (*LazyColumn*) is displayed in a vertically scrollable layout. Each notification shows the username of the user who liked or commented and a corresponding message.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



Class *Notification* stores all information abpout 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.

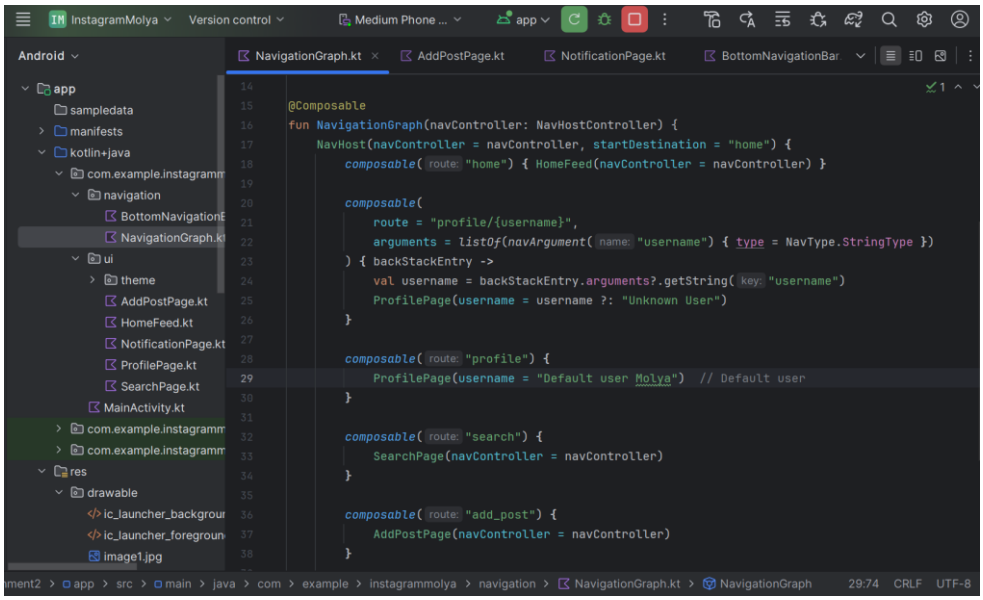


*See file *NotificationPage.kt* in ui folder.

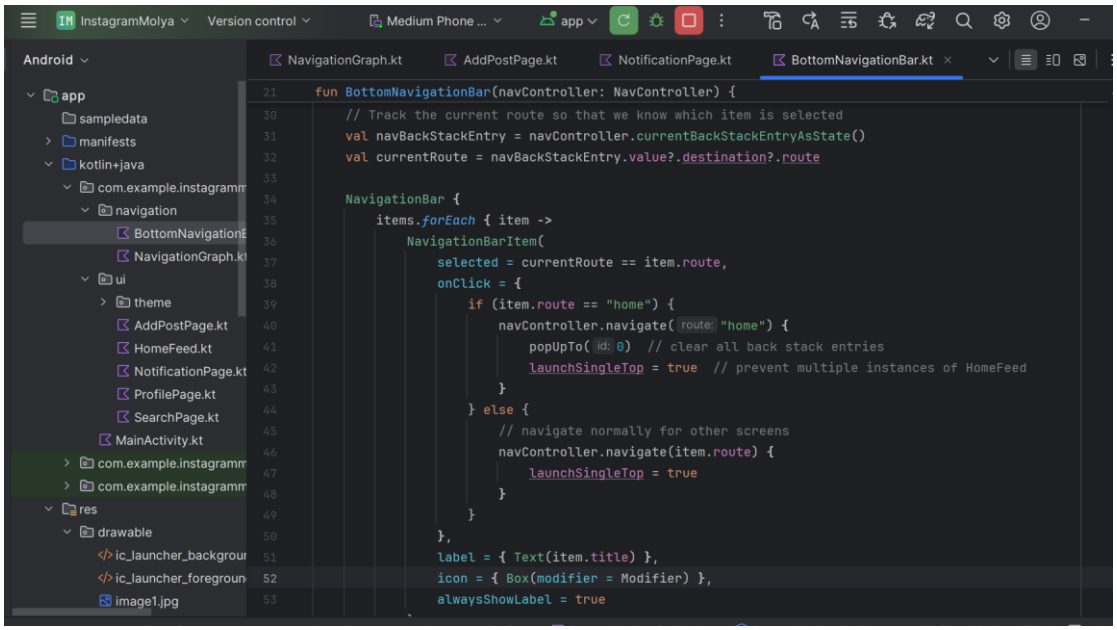
Navigation

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.



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.



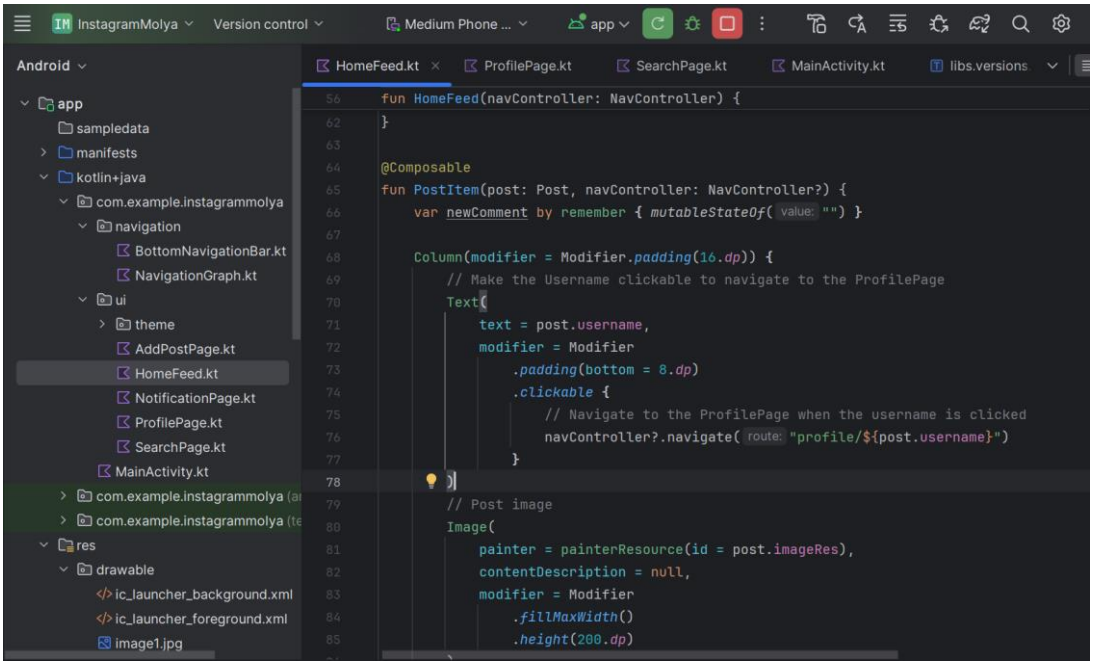
*See *NavigationGraph.kt* and *BottomNavigationBar.kt* files in ui/navigation folder.

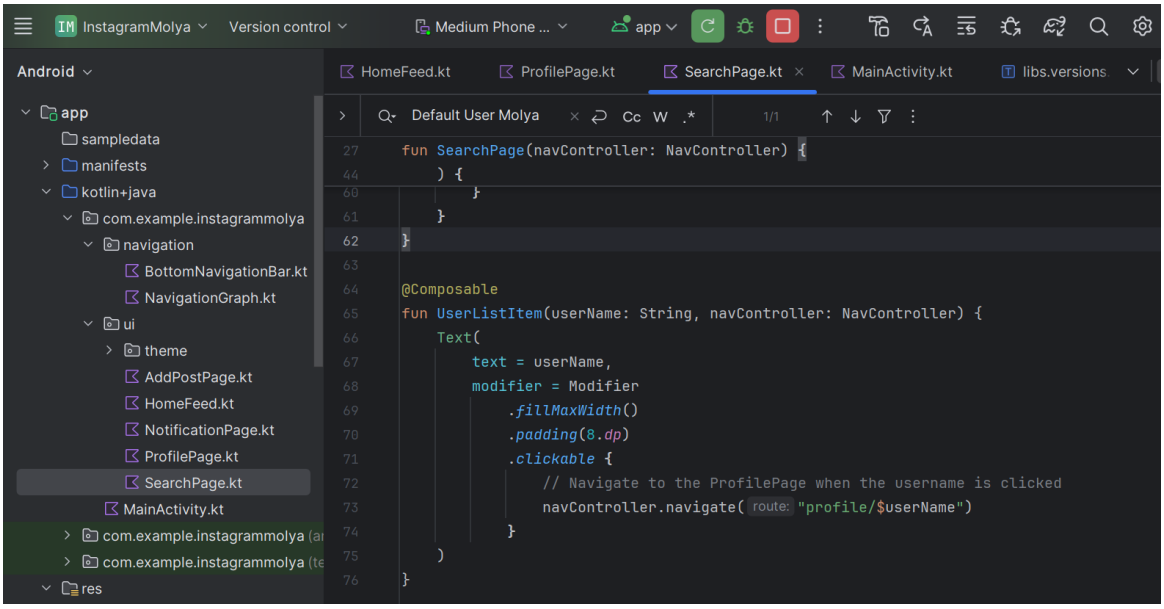
User Interaction

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

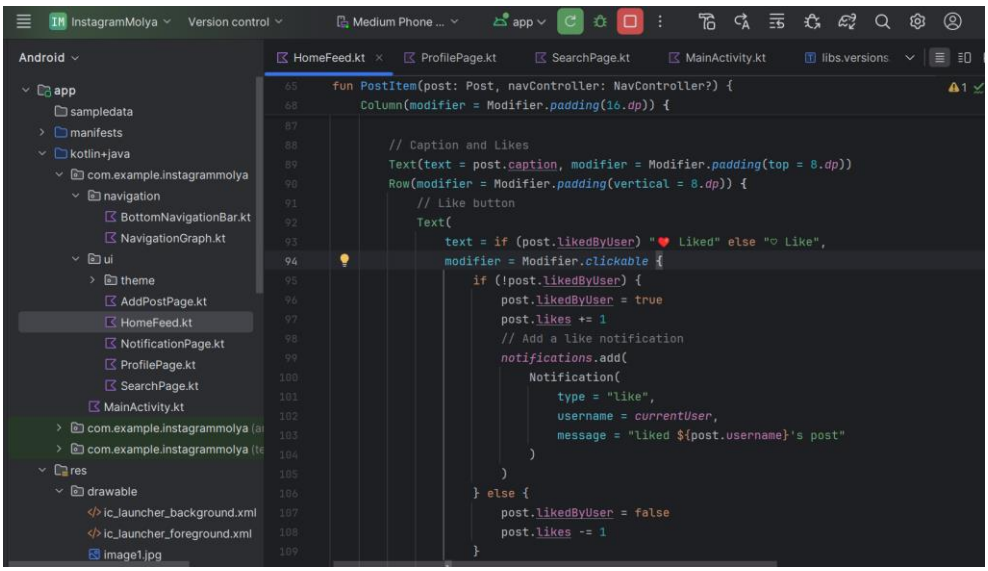
Main user interaction points:

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



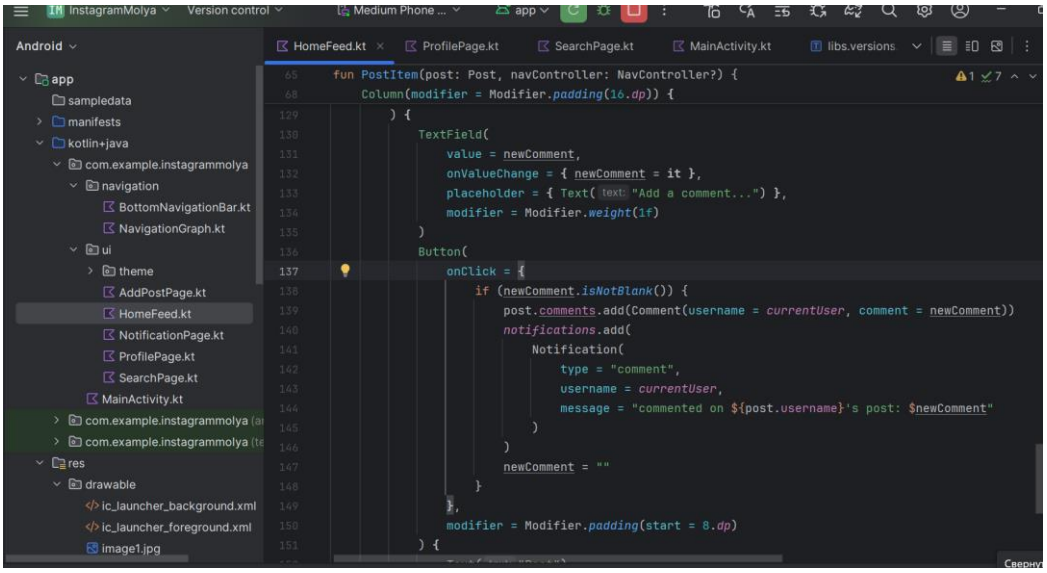


2. Liking a post (Home Feed)



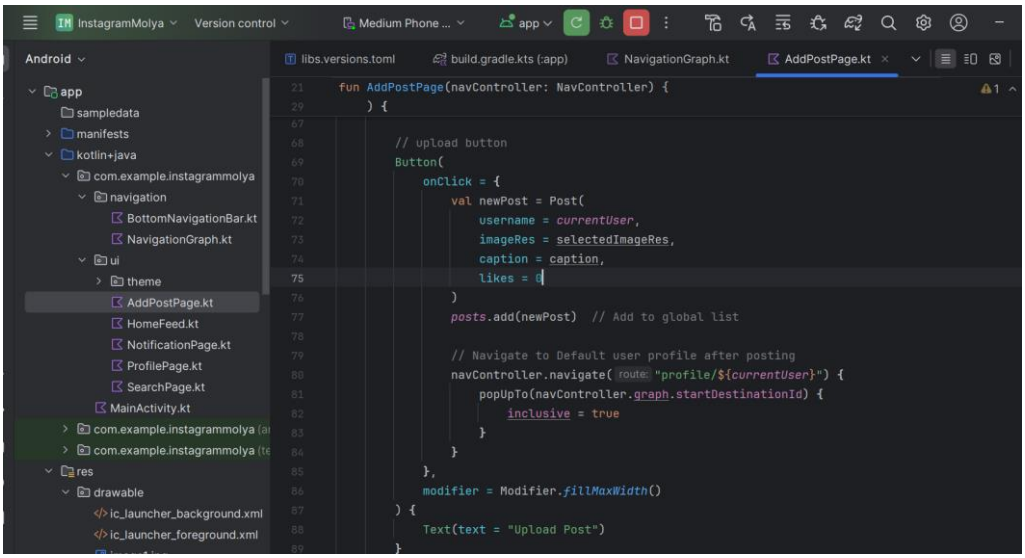
```
fun PostItem(post: Post, navController: NavController?) {
    Column(modifier = Modifier.padding(16.dp)) {

        // 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) "❤️ Liked" else "🤍 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)



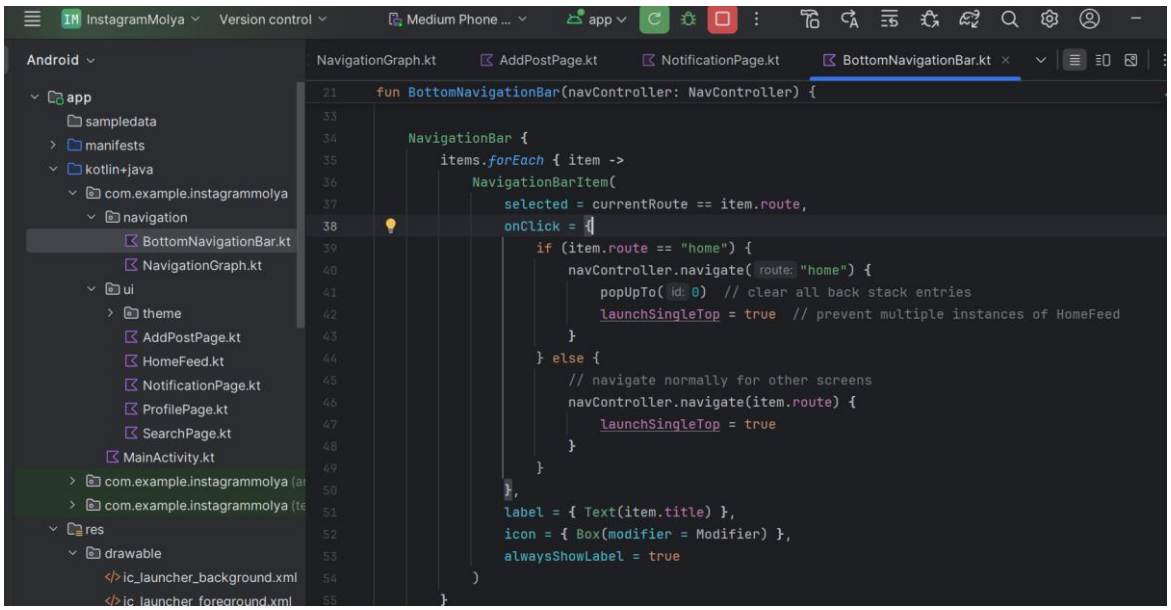
```
fun PostItem(post: Post, navController: NavController?) {
    Column(modifier = Modifier.padding(16.dp)) {
        ) {
            TextField(
                value = newComment,
                onValueChange = { newComment = it },
                placeholder = { Text( text: "Add a comment...") },
                modifier = Modifier.weight(1f)
            )
            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)

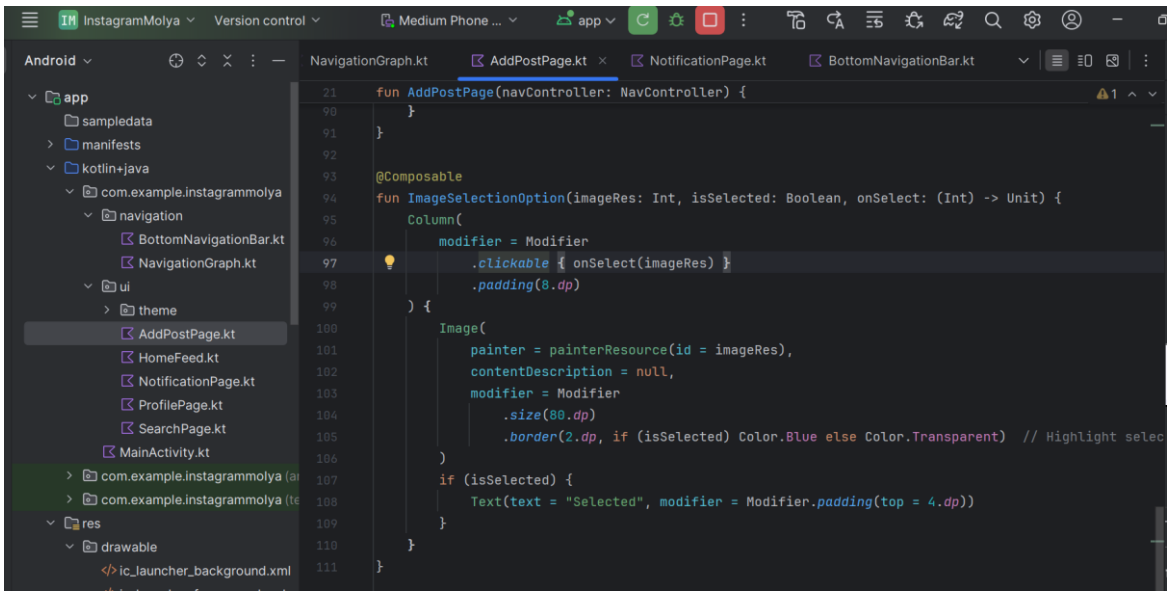


```
fun AddPostPage(navController: NavController) {
    ) {

        // 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



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



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)