KAZAKH-BRITISH TECHNICAL UNIVERSITY

Assignment #2

Mobile Programing
Create Basic Pages of Instagram
in Android Studio

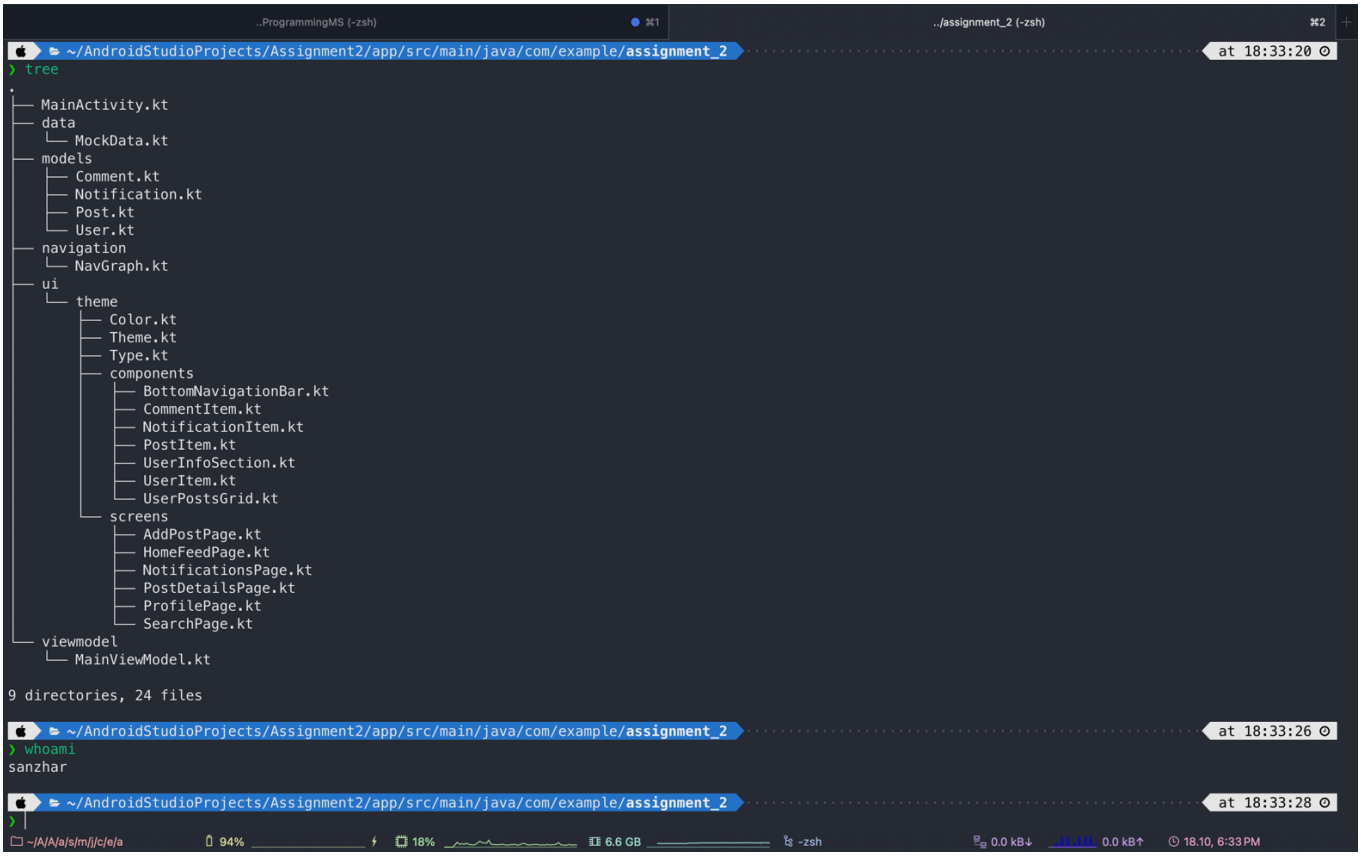
Prepared by:
Seitbekov S.
Checked by:
Serek A.

Almaty, 18.10.2024

Table of Contents

Introduction

This project will be an attempt at re-implementing Instagram's most basic pages in a simplified manner using Android Studio. For constructing the UI components, Jetpack Compose was adopted instead of the imperative RecyclerView, which traditionally deals with list-based layouts. I chose Compose because, with its declarative approach to UI, one has more flexibility, simpler state management, and an intuitive flow of designing your UI compared to the often-brutish implementations Carbide Divide produces when working with RecyclerViews. By making the most of Jetpack Compose, the development is much denser and follows modern Android development practices. During the laboratory work, I used the `whoami` command to verify my identity (Sanzhar). Also, I used my username in github (diable201) for username in Application. The source code for this project is available on my GitHub repository - https://github.com/diable201/MobileProgrammingMS

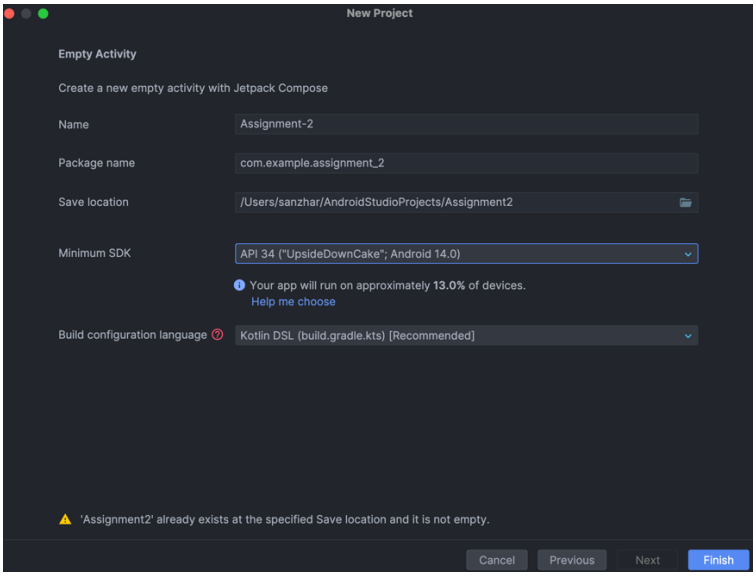


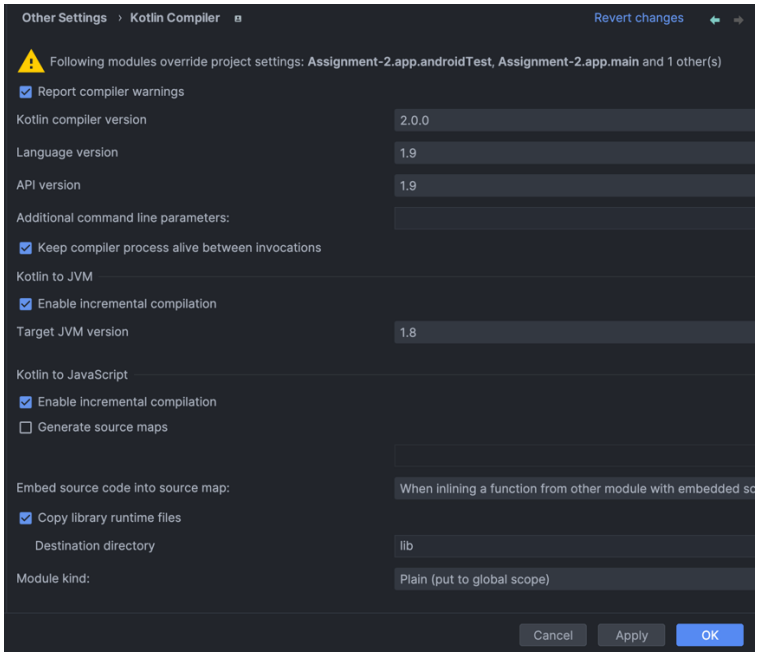
Project Setup

Initial Setup

The project was initiated in Android Studio with the following configurations:

- Language: Kotlin
- Minimum SDK Version: 34 (Android 14.0 UpsideDownCake)
- Compiler: Kotlin 2.0.0
- Build System: Gradle





Libraries

I used the following libraries, integrated into development for enhanced functionality:

- Jetpack Compose: Selected for declarative construction of the UI, making UI development much easier and more straightforward.
- Material3: Concretely implements Material Design 3 at Compose. Material3 is designed to offer modern UI components and theming.
- Material3Icons: A more extensive set of Material Design icons that work with Compose UI elements.
- Navigation Component: Jetpack Compose and the Navigation Component offer the app's navigation.
- Coil: Integrated as a library for loading images, it effectively handles the rendering of images from local storage and URLs.

```
dependencies {
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    implementation(libs.androidx.navigation.runtime.ktx)
    implementation(libs.androidx.navigation.compose)
    implementation(libs.androidx.runtime.livedata)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)
    implementation(libs.coil.compose)
    implementation(libs.androidx.material.icons.extended)
    implementation(libs.androidx.navigation.compose)
    implementation(libs.androidx.lifecycle.viewmodel.compose)
    implementation(libs.androidx.lifecycle.livedata.ktx)
    implementation(libs.lifecycle.livedata.ktx)
}
```

Page Design

Home Feed

The home feed gives the screen from which the user views posts of different users. In doing so, it reuses Jetpack Compose's LazyColumn to efficiently render the scrollable list of posts. It

represents each post by a composable, known as PostItem, which shows the avatar and username of the user, post image, caption, like count, and a comment button with a like button.

```
@Composable
fun HomeFeedPage(navController: NavController, mainViewModel: MainViewModel) {
    val posts by mainViewModel.posts.observeAsState(emptyList())

    for (post in posts) {
        Log.d( tag: "HomeFeedPage",  msg: "Post ID: ${post.id}, Title: ${post.caption}")
    }

    LazyColumn(modifier = Modifier.fillMaxSize()) {
        items(posts) { post →
            PostItem(
                post = post,
                navController = navController,
                onLikeClicked = {
                    mainViewModel.toggleLike(post.id)
                }
            )
        }
    }
}
```

Fig 1. HomeFeedPage snippet

```
@Composable
fun PostItem(
    post: Post,
    navController: NavController,
    onLikeClicked: () → Unit
) {
    Column(modifier = Modifier.fillMaxWidth()) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(8.dp)
                .clickable {
                    navController.navigate(Screen.Profile.createRoute(post.user.id)) {
                        popUpTo(navController.graph.startDestinationId) {
                            saveState = true
                        }
                        launchSingleTop = true
                        restoreState = true
                    }
                }, verticalAlignment = Alignment.CenterVertically
        ) {
            Image(
                painter = rememberAsyncImagePainter(model = post.user.profileImageUrl),
                contentDescription = "Profile Image",
                modifier = Modifier.size(40.dp)
            )
            Text(
                text = post.user.username,
                fontWeight = FontWeight.Bold,
                modifier = Modifier
                    .padding(start = 8.dp)
                    .clickable {
                        navController.navigate(Screen.Profile.createRoute(post.user.id)) {
                            popUpTo(navController.graph.startDestinationId) {
                                saveState = true
                            }
                            launchSingleTop = true
                            restoreState = true
                        }
```

Fig 2. PostItem snippet

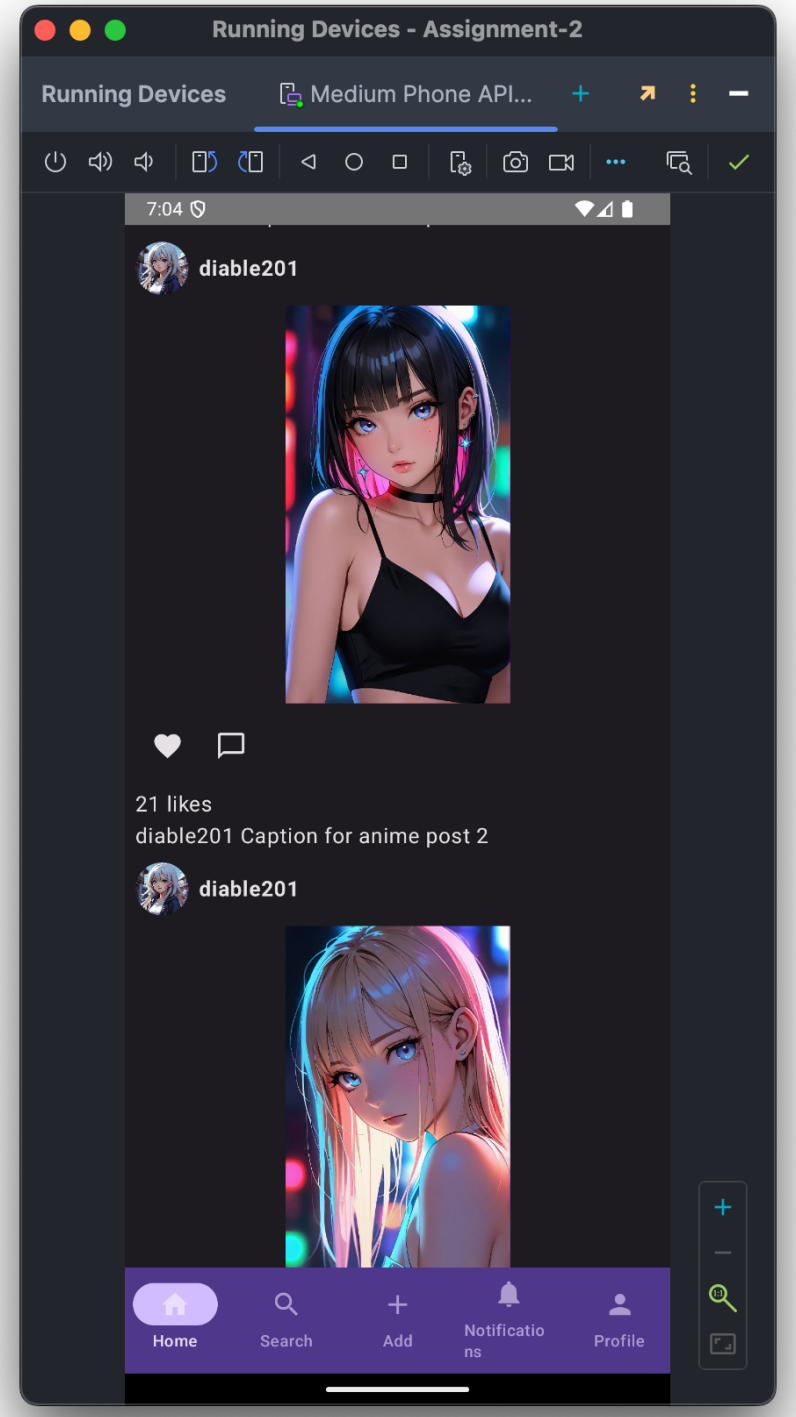


Fig 3. Application Home Page

Profile Page

The details of the users will be displayed in a Profile Page along with the user's avatar, a username, and a grid of their posts. Wrapping up with Compose's Column and LazyColumn on the page demonstrates user details followed by their posts. The posts are displayed in a grid format using a combination of Column and LazyColumn, ensuring seamless display of user-generated content.

```
@Composable
fun ProfilePage(navController: NavController, mainViewModel: MainViewModel, userId: Int) {

    val user by mainViewModel.getUserById(userId)
        .observeAsState(initial = mainViewModel.currentUser)

    Log.d( tag: "ProfilePage",  msg: "User ID: $userId")

    val posts by mainViewModel.getPostsForUser(userId).observeAsState(initial = emptyList())
    val postCount by mainViewModel.getPostCountForUser(userId).observeAsState(initial = 0)
    user.postsCount = postCount

    Column(modifier = Modifier.fillMaxSize()) {
        UserInfoSection(user)
        UserPostsGrid(posts = posts, navController = navController)
    }
}
```

Fig 4. ProfilePage Snippet

```
@Composable
fun UserInfoSection(user: User) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Image(
            painter = rememberAsyncImagePainter(model = user.profileImageUrl),
            contentDescription = "Profile Image",
            modifier = Modifier
                .size(80.dp)
                .clip(CircleShape)
        )
        Text(text = user.username, fontWeight = FontWeight.Bold, fontSize = 20.sp)
        user.bio?.let { Text(text = it) }
        Text(text = "${user.postsCount} Posts")
    }
}
```

Fig 5. UserInfoSection Snippet

```kotlin
@Composable
fun UserPostsGrid(posts: List<Post>, navController: NavController) {
    val gridState = rememberLazyGridState()
    LazyVerticalGrid(
        columns = GridCells.Fixed( count: 3),
        state = gridState,
        contentPadding = PaddingValues(1.dp),
        modifier = Modifier.fillMaxSize()
    ) {
        items(posts) { post ->
            Image(
                painter = rememberAsyncImagePainter(model = post.imageUrl ?: post.imageData),
                contentDescription = "Post Image",
                modifier = Modifier
                    .aspectRatio( ratio: 1f)
                    .clickable {
                        navController.navigate(Screen.PostDetails.createRoute(post.id))
                    }
            )
        }
    }
}
```

Fig 6. UserPostsGrid Snippet

Fig 7. Application ProfilePage

Search Page

The Search Page enables users to find other users or posts. It features a search bar (TextField) and displays results dynamically as the user types, utilizing a LazyColumn for listing the results. Users can filter search results for improved discoverability.

```
@Composable
fun SearchPage(navController: NavController, mainViewModel: MainViewModel) {
    var query by remember { mutableStateOf( value: "") }
    val filteredUsers by mainViewModel.searchUsers(query).observeAsState(emptyList())

    Column(modifier = Modifier.fillMaxSize()) {
        TextField(
            value = query,
            onValueChange = { query = it },
            placeholder = { Text( text: "Search Users") },
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Search,
                    contentDescription = "Search"
                )
            },
            modifier = Modifier
                .fillMaxWidth()
                .padding(8.dp)
        )
        Spacer(modifier = Modifier.height(8.dp))
        LazyColumn(
            modifier = Modifier.fillMaxSize()
        ) {
            if (filteredUsers.isEmpty() && query.isNotBlank()) {
                item {
                    Text(
                        text = "No users found.",
                        modifier = Modifier.padding(16.dp),
                        style = MaterialTheme.typography.bodyMedium
                    )
                }
            } else {
                items(filteredUsers) { user ->
                    UserItem(user = user, navController = navController)
                }
            }
        }
    }
}
```

Fig 8. SearchPage Snippet

```
@Composable
fun UserItem(user: User, navController: NavController) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .clickable {
                navController.navigate(Screen.Profile.createRoute(user.id)) {
                    popUpTo(navController.graph.startDestinationId) {
                        saveState = true
                    }
                    launchSingleTop = true
                    restoreState = true
                }
            }
            .padding(8.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Image(
            painter = rememberAsyncImagePainter(model = user.profileImageUrl),
            contentDescription = "Profile Image",
            modifier = Modifier
                .size(40.dp)
                .clip(CircleShape)
        )
        Text(text = user.username, modifier = Modifier.padding(start = 8.dp))
    }
}
```

Fig 9. UserItem Snippet

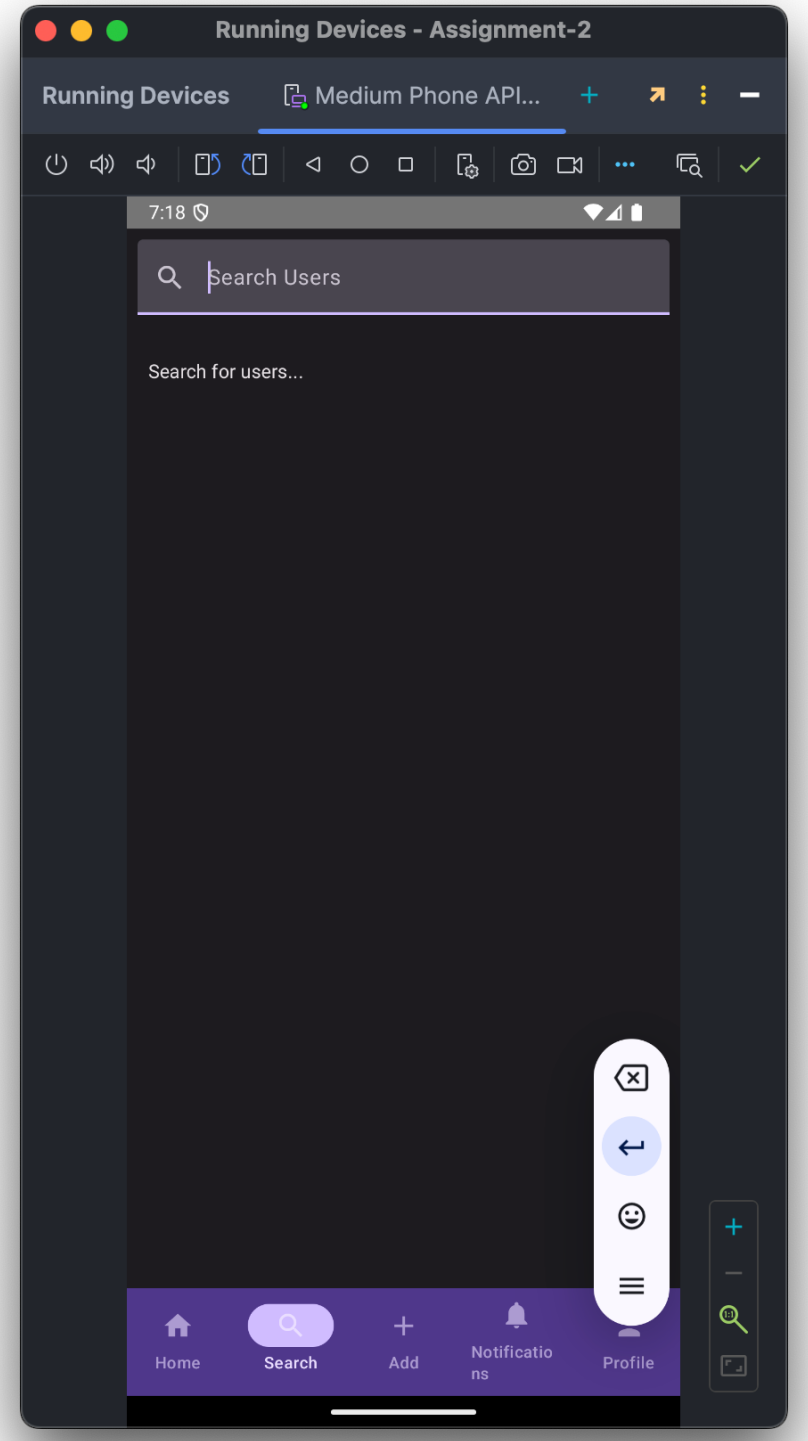


Fig 10. Empty Search

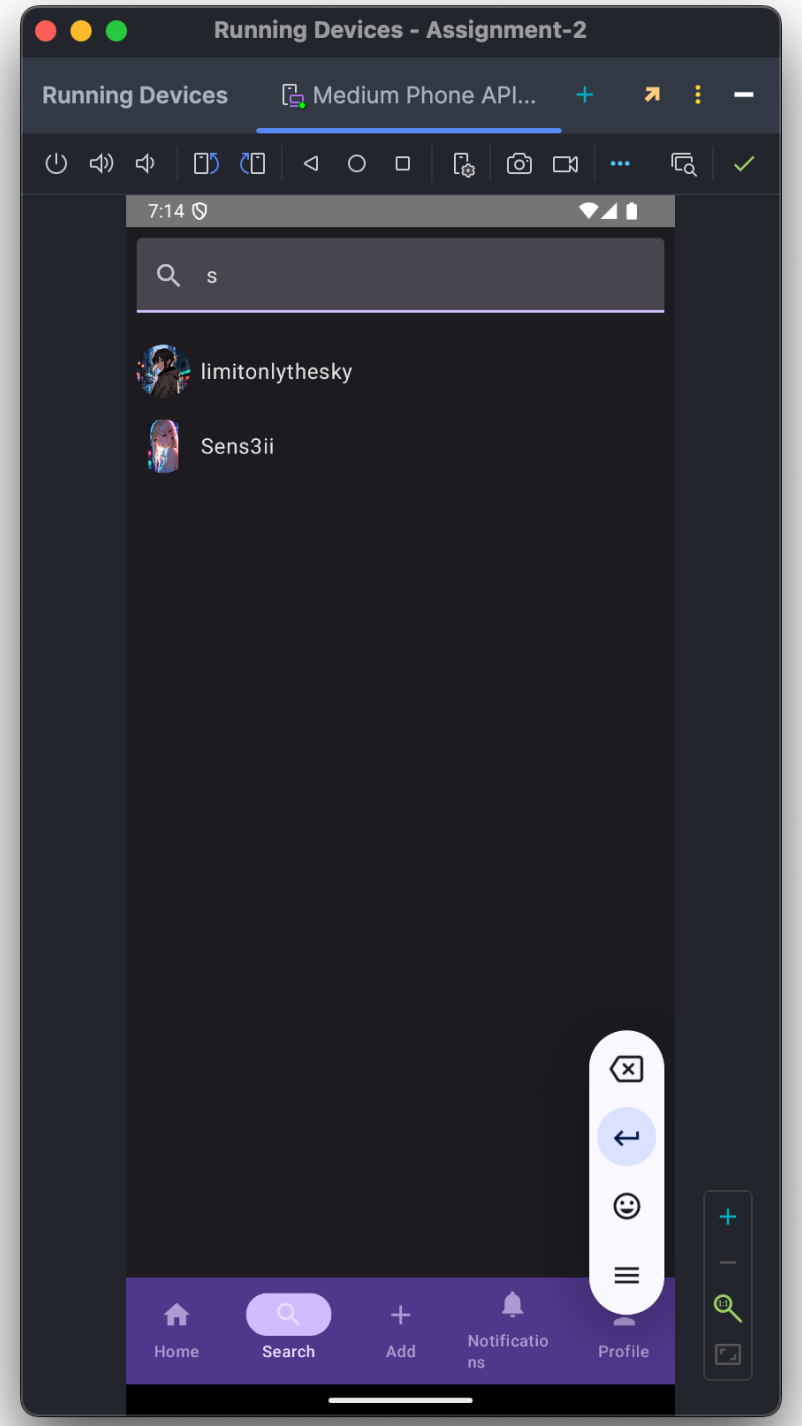


Fig 11. Serch Filter

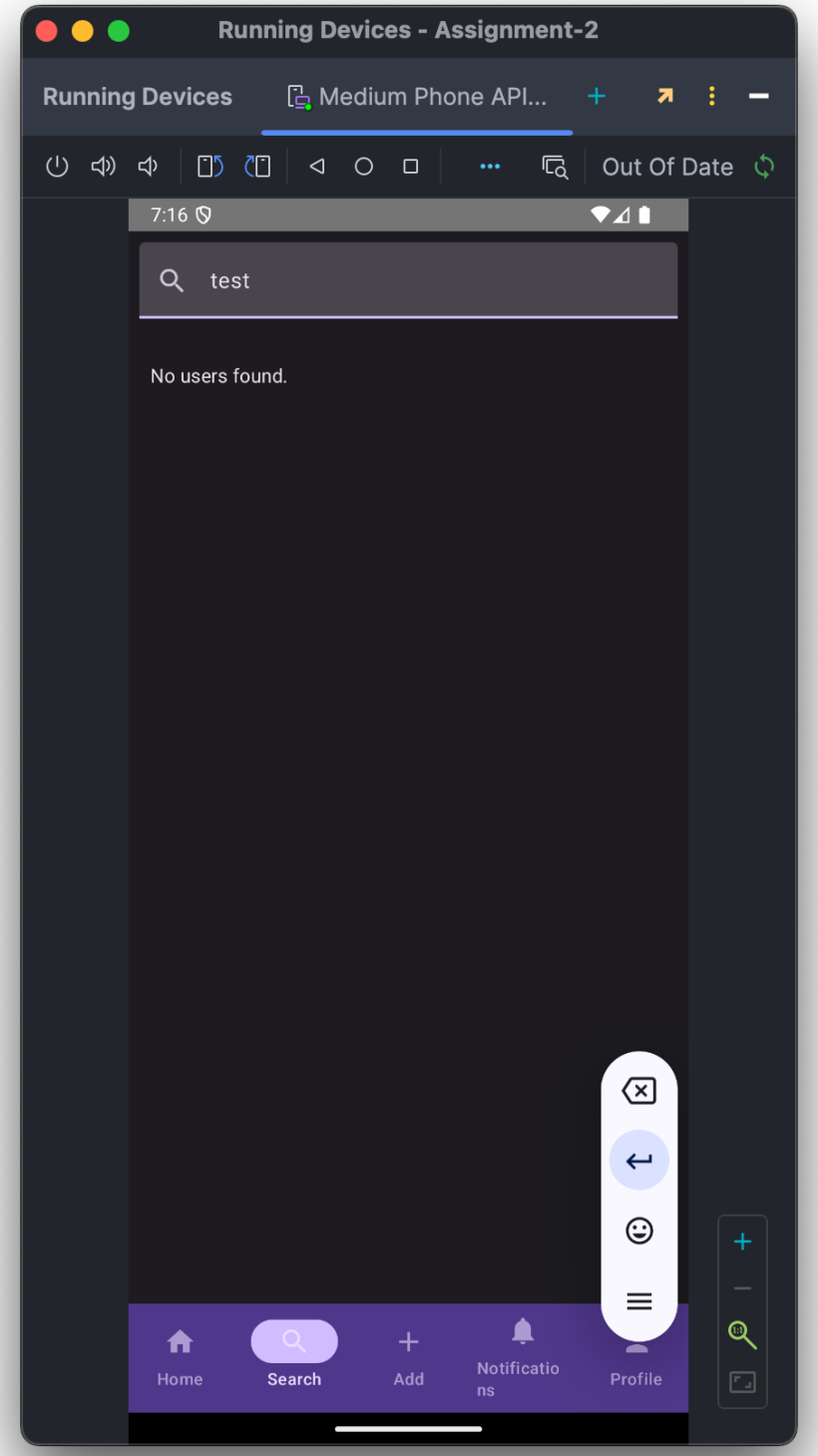


Fig 12. Search with no results

Add Post Page

The Add Post Page allows users to create new posts by selecting an image and adding a caption. It features an image picker and a submission button that uploads the image and adds the post to the Home Feed. It integrates an image picker using ActivityResultContracts.PickVisualMedia for smooth media selection.

```
@Composable
fun AddPostPage(navController: NavController, mainViewModel: MainViewModel) {
    val context = LocalContext.current
    var caption by remember { mutableStateOf( value: "") }
    var imageBitmap by remember { mutableStateOf<Bitmap?>( value: null) }

    // Image Picker Launcher
    val pickImageLauncher =
        rememberLauncherForActivityResult(ActivityResultContracts.PickVisualMedia()) { uri →
            uri?.let {
                Log.d( tag: "AddPostPage",  msg: "Image URI: $uri")
                val bitmap = loadBitmapFromUri(context, it)
                if (bitmap ≠ null) {
                    imageBitmap = resizeBitmap(bitmap, maxSize = 1024)
                    Log.d( tag: "AddPostPage",  msg: "Bitmap loaded successfully")
                } else {
                    Log.d( tag: "AddPostPage",  msg: "Failed to load bitmap")
                }
            } ?: Log.d( tag: "AddPostPage",  msg: "No image selected")
        }

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = "Posting as ${currentUser.username}",
            style = MaterialTheme.typography.titleMedium,
            modifier = Modifier.padding(bottom = 16.dp)
        )

        Image(
            painter = rememberAsyncImagePainter(currentUser.profileImageUrl),
            contentDescription = "User Avatar",
            modifier = Modifier
                .size(80.dp)
                .clip(CircleShape)
                .background(Color.Gray),
            contentScale = ContentScale.Crop
```

Fig 13. AddPostPage snippet

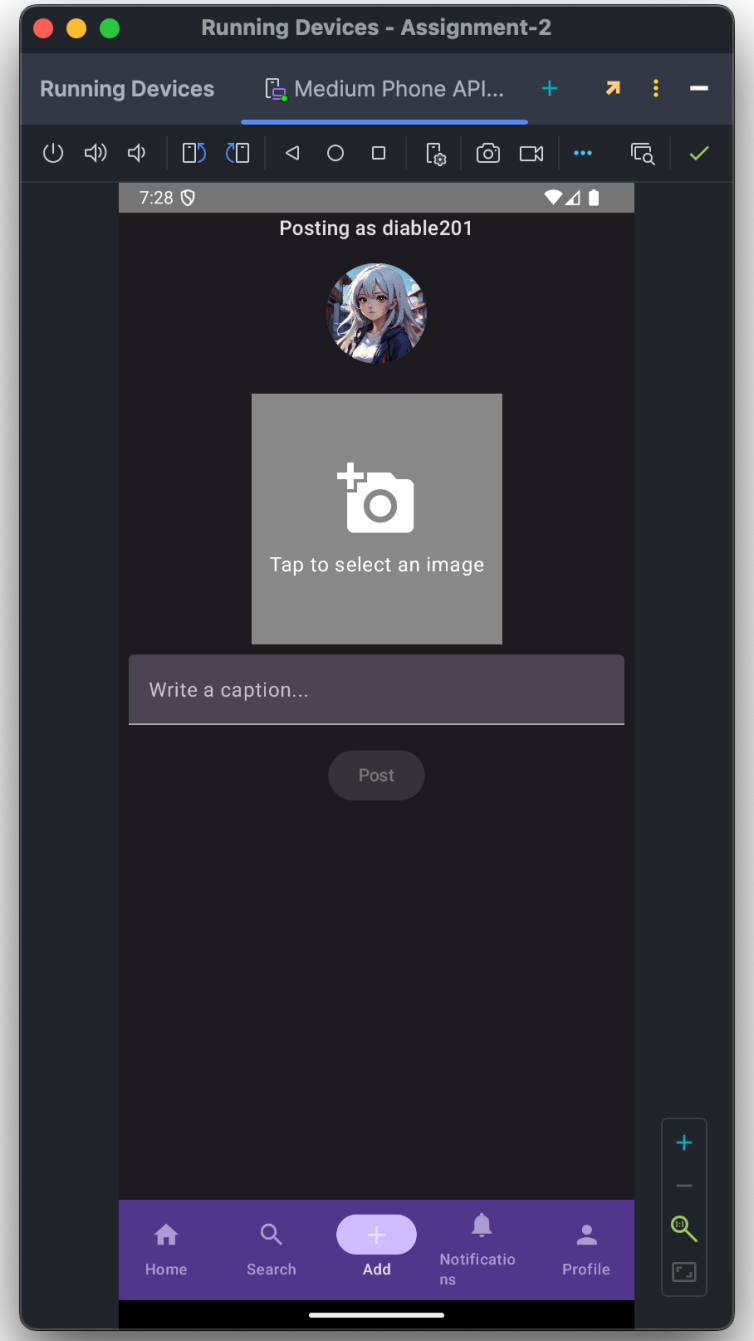


Fig 14. Selecting an image box

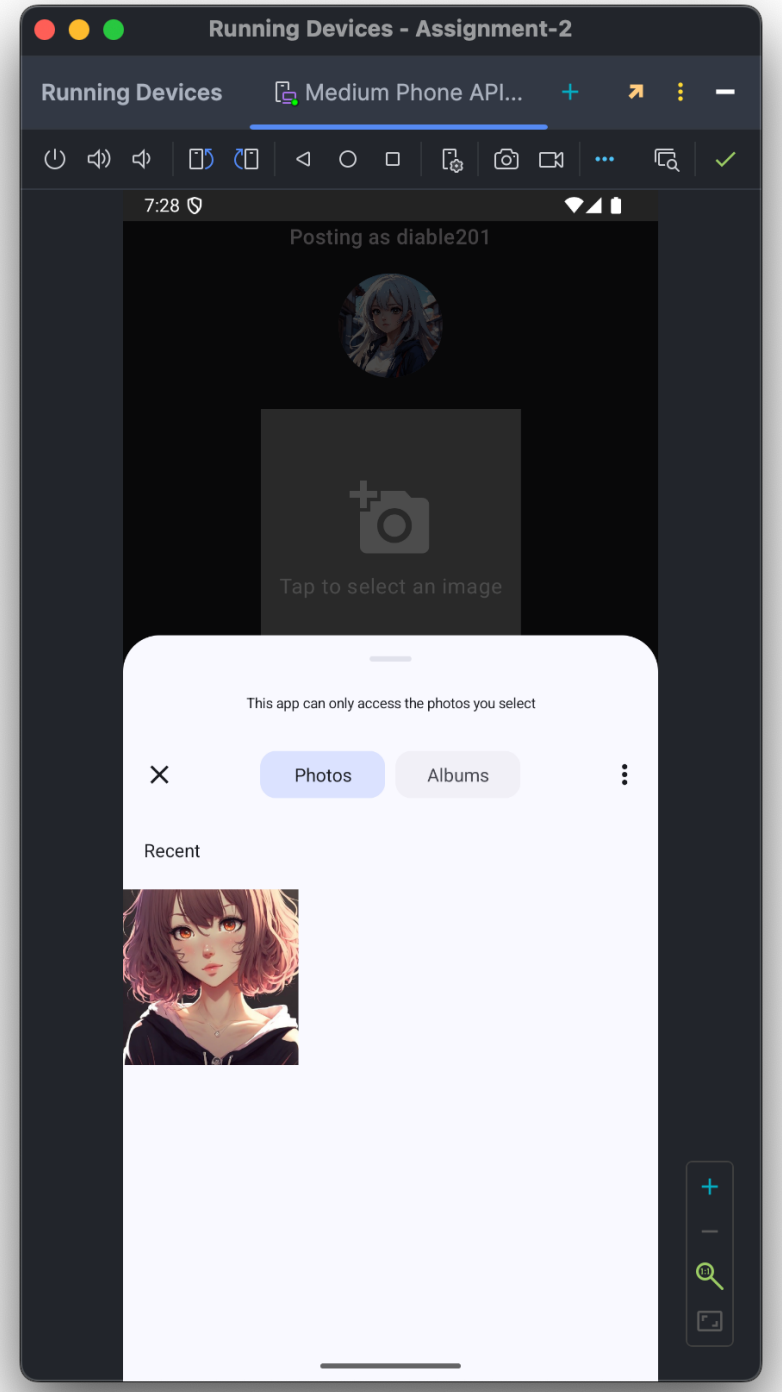


Fig 15. Selecting an image from local storage

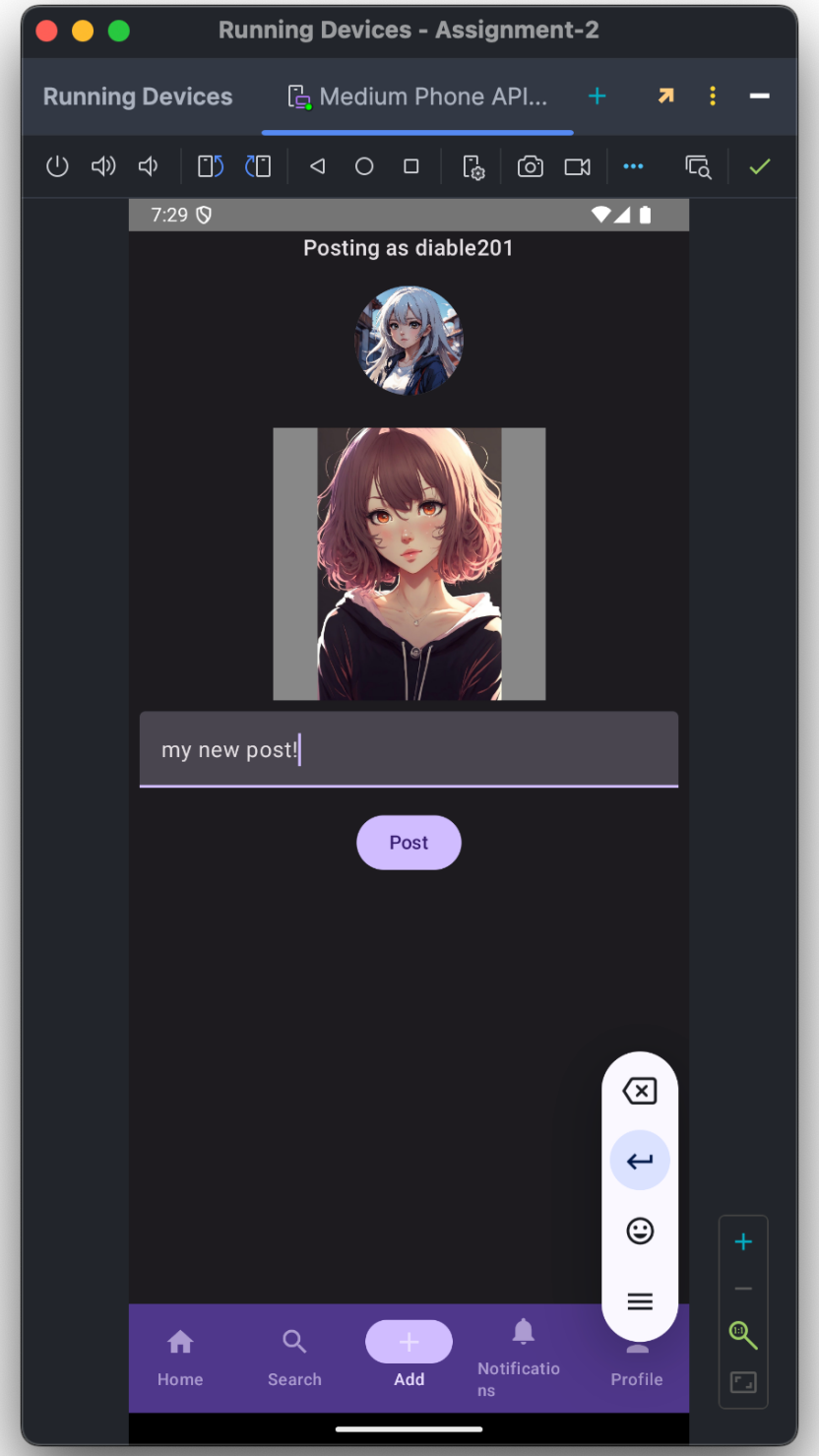


Fig 15. Writing a caption for post

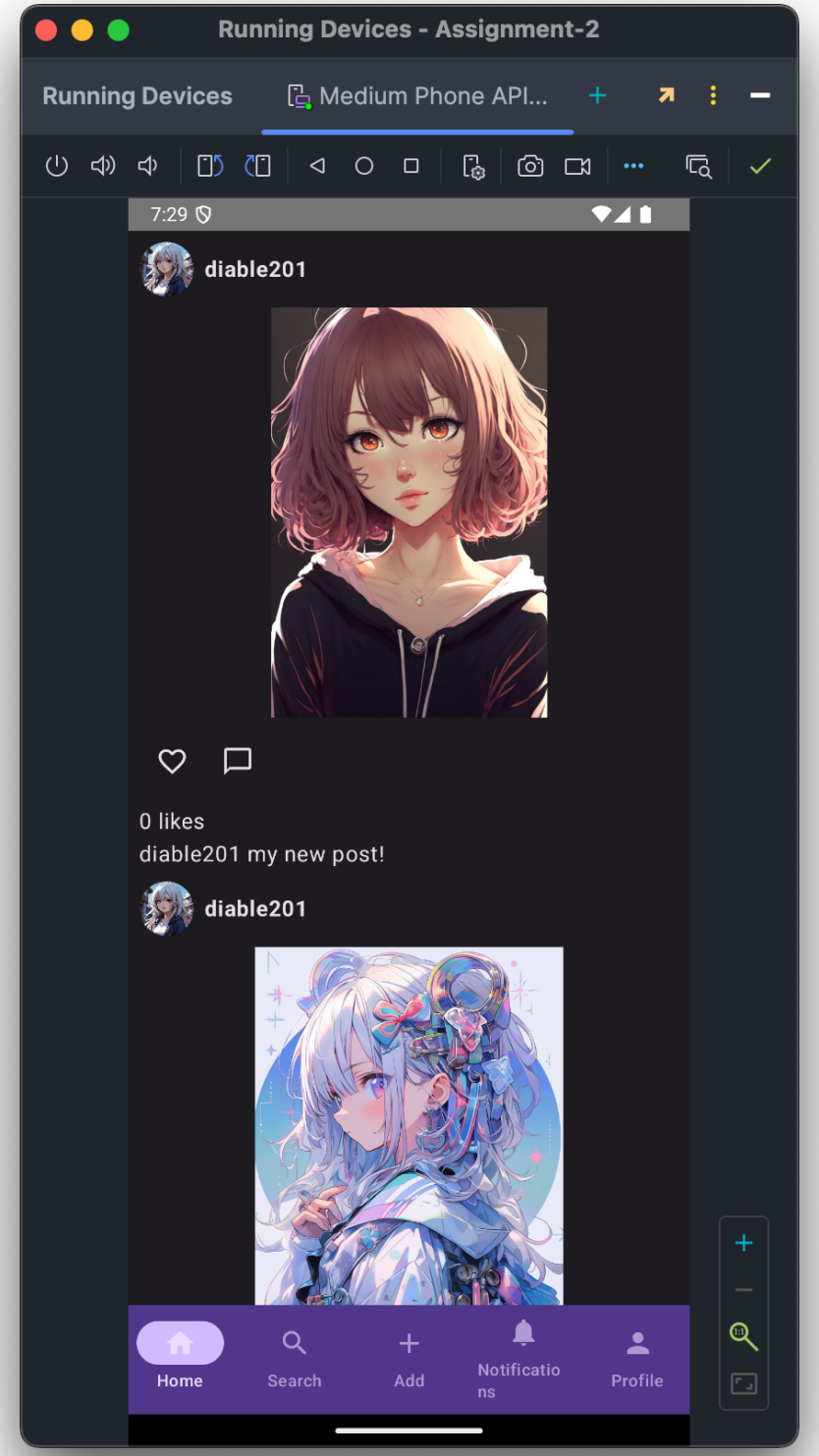


Fig 16. New post in home page

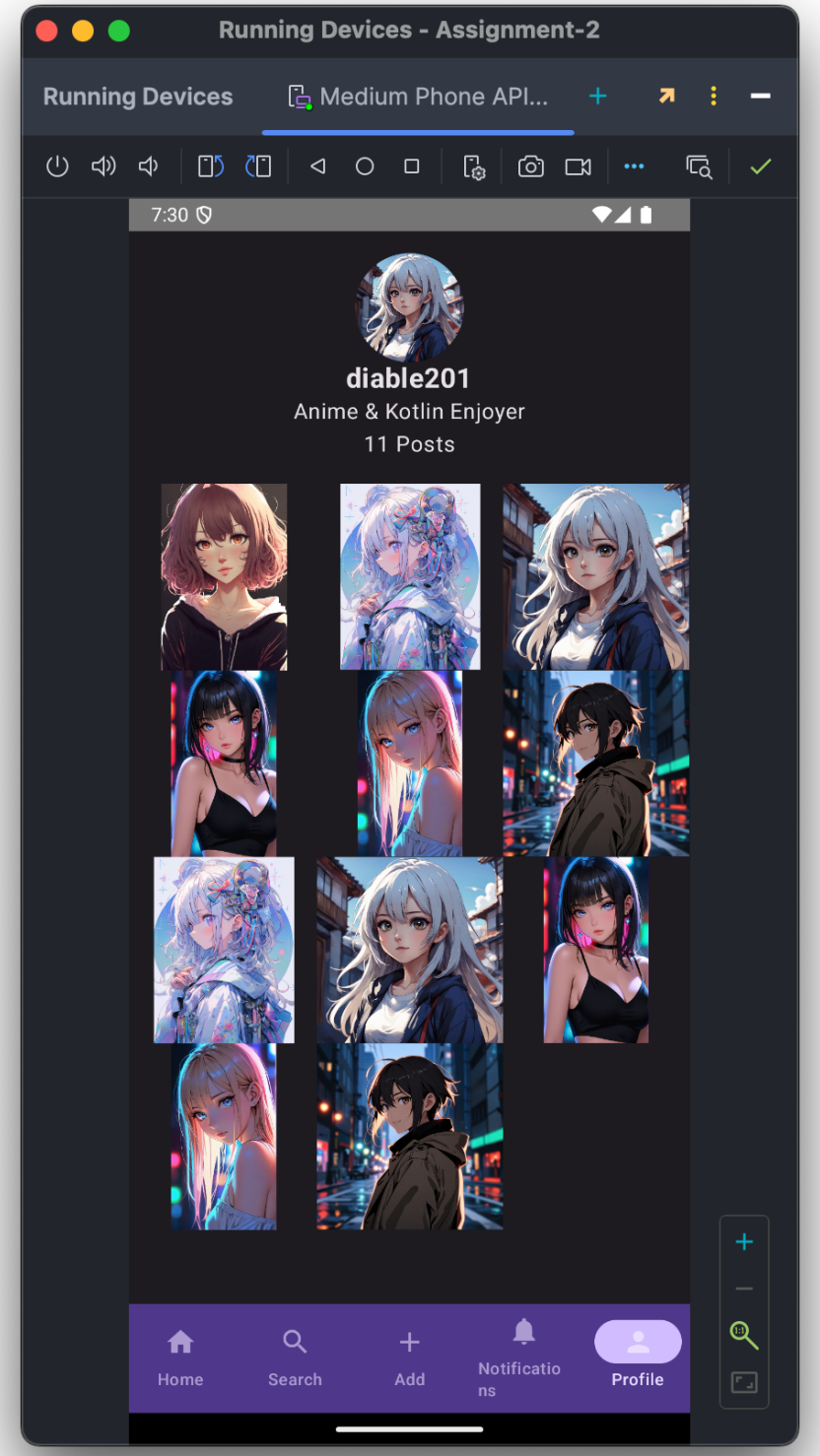


Fig 17. New post in proflie page. Count of posts also updated

Notifications Page

The Notifications Page lists all user notifications, such as likes and comments. Each notification is presented using the Notification composable, which includes the notifier's avatar, message, and timestamp wuth a NotificationItem composable.

```
@Composable
fun NotificationsPage(navController: NavController, mainViewModel: MainViewModel) {
    // Observe notifications from the ViewModel
    val notifications by mainViewModel.notifications.observeAsState(initial = emptyList())

    LazyColumn(modifier = Modifier.fillMaxSize()) {
        items(notifications, key = { it.id }) { notification ->
            NotificationItem(notification = notification, navController = navController)
        }
    }
}
```

Fig 18. Notifications Page snippet

```
@Composable
fun NotificationItem(notification: NotificationItem, navController: NavController) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .clickable {
                notification.targetId?.let { postId ->
                    navController.navigate(Screen.PostDetails.createRoute(postId)) {
                        popUpTo(navController.graph.startDestinationId) {
                            saveState = true
                        }
                        launchSingleTop = true
                        restoreState = true
                    }
                }
            }
            .padding(8.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Image(
            painter = rememberAsyncImagePainter(model = notification.user.profileImageUrl),
            contentDescription = "Profile Image",
            modifier = Modifier
                .size(40.dp)
                .clip(CircleShape)
        )
        Column(modifier = Modifier.padding(start = 8.dp)) {
            Text(text = notification.message)
            Text(text = notification.timestamp, color = Color.Gray, fontSize = 12.sp)
        }
    }
}
```

Fig 19. NotificationItem snippet

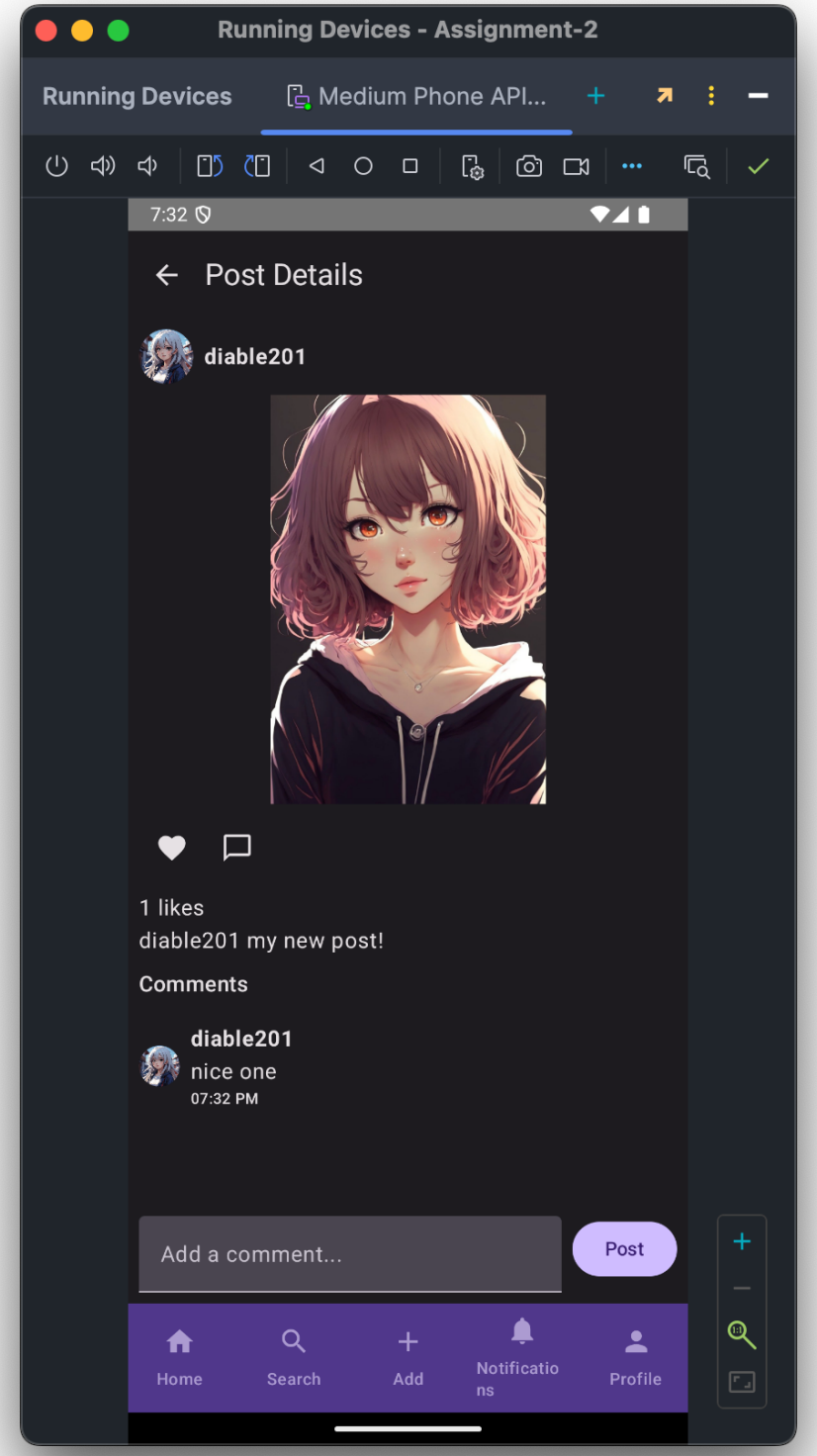


Fig 20. Adding comment and like to post

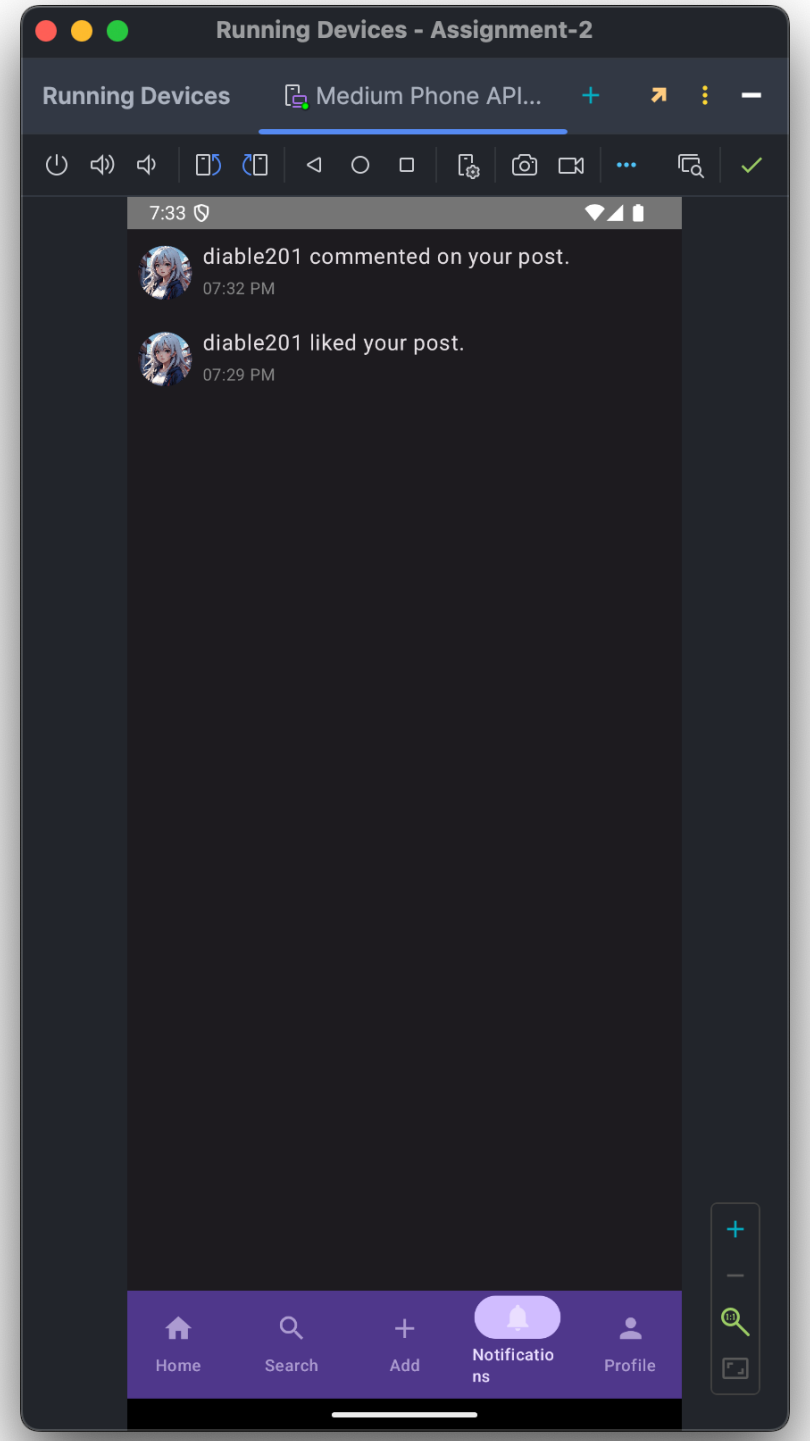


Fig 21. New notifications about like and comment

Navigation

Navigation within the app is managed using Android's Navigation Component integrated with Jetpack Compose. A sealed class Screen defines all possible navigation routes, ensuring type safety and ease of maintenance.

```kotlin
sealed class Screen(val route: String) {
    data object Home : Screen( route: "home")
    data object Search : Screen( route: "search")
    data object AddPost : Screen( route: "add_post")
    data object Notifications : Screen( route: "notifications")
    data object Profile : Screen( route: "profile/{userId}") {
        fun createRoute(userId: Int) = "profile/$userId"
    }
    data object PostDetails : Screen( route: "post_details/{postId}") {
        fun createRoute(postId: Int) = "post_details/$postId"
    }
}
```

Fig 22. Screen snippet

```kotlin
@Composable
fun AppNavGraph(navController: NavHostController, mainViewModel: MainViewModel) {
    NavHost(navController, startDestination = Screen.Home.route) {
        composable(Screen.Home.route) {
            HomeFeedPage(navController, mainViewModel)
        }
        composable(Screen.Search.route) {
            SearchPage(navController, mainViewModel)
        }
        composable(Screen.AddPost.route) {
            AddPostPage(navController, mainViewModel)
        }
        composable(Screen.Notifications.route) {
            NotificationsPage(navController, mainViewModel)
        }
        composable(
            route = Screen.Profile.route,
            arguments = listOf(navArgument( name: "userId") { type = NavType.IntType })
        ) { backStackEntry →
            val userId = backStackEntry.arguments?.getInt( key: "userId") ?: mainViewModel.currentUser.id
            ProfilePage(navController, mainViewModel, userId)
        }
        composable(
            route = Screen.PostDetails.route,
            arguments = listOf(navArgument( name: "postId") { type = NavType.IntType })
        ) { backStackEntry →
            val postId = backStackEntry.arguments?.getInt( key: "postId") ?: 0
            PostDetailsPage(navController, postId, mainViewModel)
        }
    }
}

```

Fig 23. Navigation snippet

```kotlin
@Composable
fun BottomNavigationBar(navController: NavController, mainViewModel: MainViewModel) {
    val items = listOf(
        Screen.Home,
        Screen.Search,
        Screen.AddPost,
        Screen.Notifications,
        Screen.Profile,
    )

    NavigationBar(
        containerColor = MaterialTheme.colorScheme.primaryContainer
    ) {
        val navBackStackEntry = navController.currentBackStackEntryAsState()
        val currentRoute = navBackStackEntry.value?.destination?.route

        items.forEach { screen ->
            val isSelected = currentRoute?.startsWith(screen.route) == true
            NavigationBarItem(
                icon = {
                    Icon(
                        imageVector = when (screen) {
                            Screen.Home -> Icons.Default.Home
                            Screen.Search -> Icons.Default.Search
                            Screen.AddPost -> Icons.Default.Add
                            Screen.Notifications -> Icons.Default.Notifications
                            Screen.Profile -> Icons.Filled.Person
                            else -> {
                                throw IllegalArgumentException("Invalid screen route: $screen")
                            }
                        },
                        contentDescription = screen.route
                    )
                },
                label = {
                    Text(
                        text = when (screen) {
                            Screen.Home -> "Home"
                            Screen.Search -> "Search"
```

Fig 24. Bottom Navigation snippet

User Interaction

User interactions are central to creating an engaging and responsive application. This project handles various interactions such as image selection, liking posts, adding comments, and navigating between different screens.

Image Selection

Users can select images from their device's gallery to create new posts. This is implemented using rememberLauncherForActivityResult combined with

ActivityResultContracts.PickVisualMedia.

```
56      val pickImageLauncher =
57          rememberLauncherForActivityResult(ActivityResultContracts.PickVisualMedia()) { uri →
58              uri?.let {
59                  Log.d( tag: "AddPostPage",  msg: "Image URI: $uri")
60                  val bitmap = loadBitmapFromUri(context, it)
61                  if (bitmap ≠ null) {
62                      imageBitmap = resizeBitmap(bitmap, maxSize = 1024)
63                      Log.d( tag: "AddPostPage",  msg: "Bitmap loaded successfully")
64                  } else {
65                      Log.d( tag: "AddPostPage",  msg: "Failed to load bitmap")
66                  }
67              } ?: Log.d( tag: "AddPostPage",  msg: "No image selected")
68          }
```

Fig 25. Image Launcher snippet

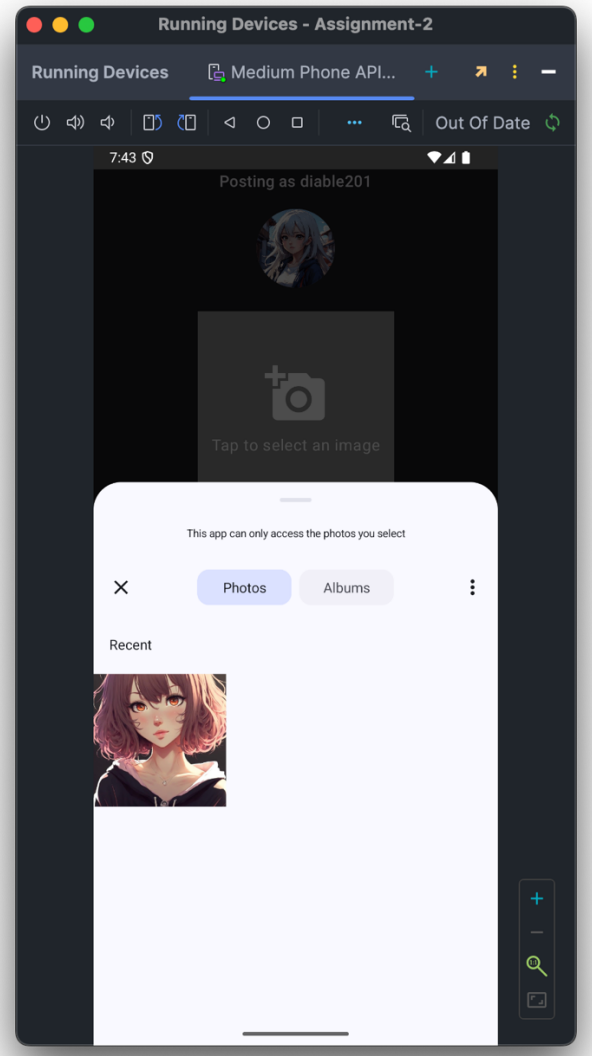


Fig 26. Image Selection from local storage

Liking a Post

Users can like or unlike posts by tapping the like button. The like count updates in real-time, and a corresponding notification is generated.

```
109         Row(modifier = Modifier.padding(8.dp)) {
110             IconButton(onClick = onLikeClicked) {
111                 Icon(
112                     imageVector = if (post.isLiked) Icons.Filled.Favorite else Icons.Outlined.FavoriteBorder,
113                     contentDescription = "Like"
114                 )
115             }
116             IconButton(onClick = {
117                 navController.navigate(Screen.PostDetails.createRoute(post.id))
118             }) {
119                 Icon(imageVector = Icons.Outlined.ChatBubbleOutline, contentDescription = "Comment")
120             }
121         }
122         Text(
123             text = "${post.likes} likes", modifier = Modifier.padding(horizontal = 8.dp)
124         )
```

Fig 27. Liking post logic snippet

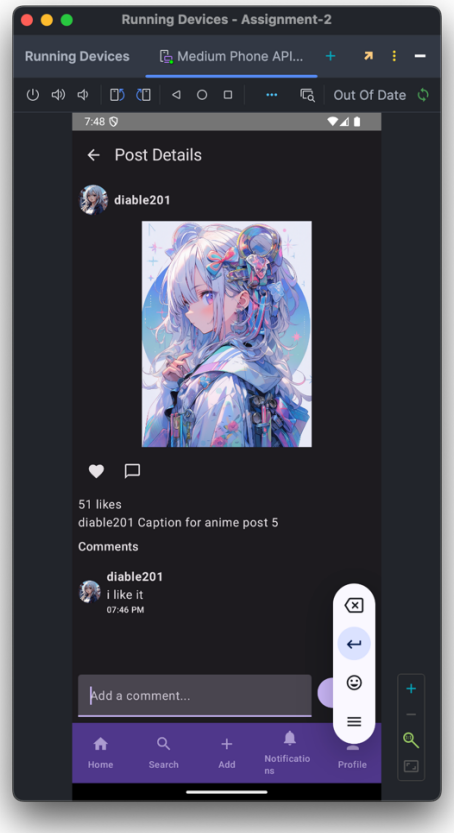


Fig 28. Liking post in application

Navigating via Notifications

Notifications are interactive, allowing users to navigate directly to related content such as post details or user profiles.

```
@Composable
fun NotificationItem(notification: NotificationItem, navController: NavController) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .clickable {
                notification.targetId?.let { postId →
                    navController.navigate(Screen.PostDetails.createRoute(postId)) {
                        popUpTo(navController.graph.startDestinationId) {
                            saveState = true
                        }
                        launchSingleTop = true
                        restoreState = true
                    }
                }
            }
            .padding(8.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Image(
            painter = rememberAsyncImagePainter(model = notification.user.profileImageUrl),
            contentDescription = "Profile Image",
            modifier = Modifier
                .size(40.dp)
                .clip(CircleShape)
        )
        Column(modifier = Modifier.padding(start = 8.dp)) {
            Text(text = notification.message)
            Text(text = notification.timestamp, color = Color.Gray, fontSize = 12.sp)
        }
    }
}
```

Fig 29. Notification logic snippet

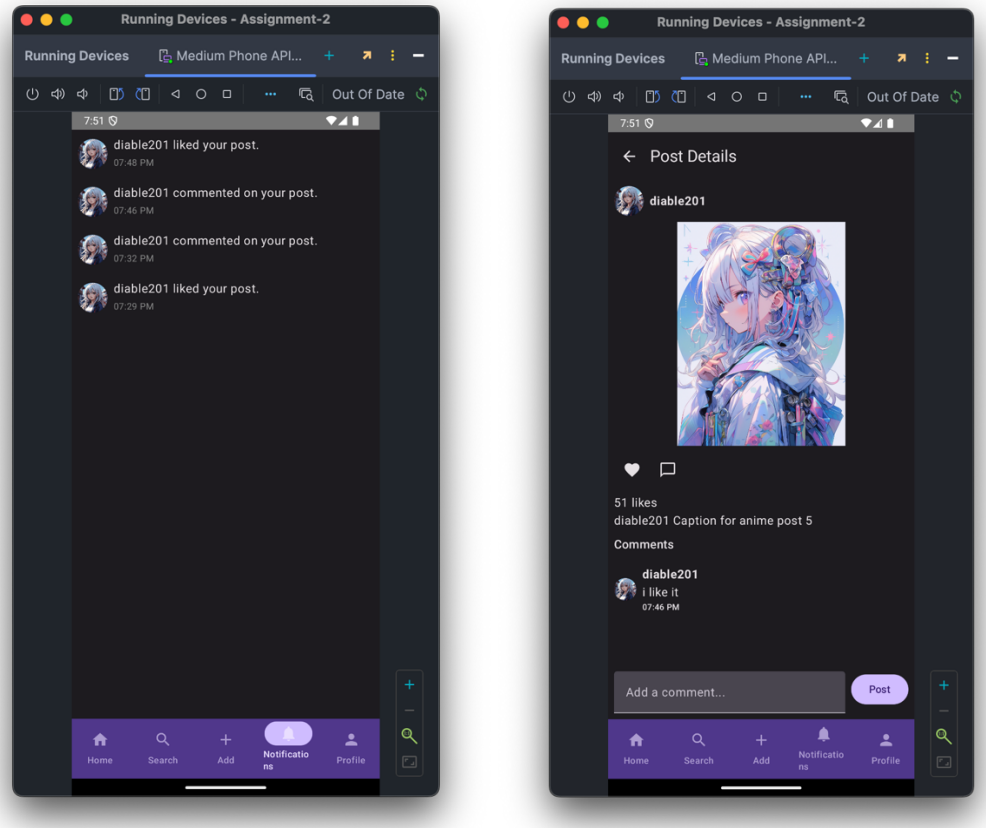


Fig 30. Notification page in application

Submitting a Post

Users can create new posts by selecting an image and adding a caption. The submission process involves uploading the image and updating the Home Feed.

```
128         TextField(
129             value = caption,
130             onValueChange = { caption = it },
131             placeholder = { Text( text: "Write a caption ... ") },
132             modifier = Modifier
133                 .fillMaxWidth()
134                 .padding(8.dp)
135         )
136
137         Button(
138             onClick = {
139                 imageBitmap?.let { bitmap →
140                     mainViewModel.addPost(bitmap, caption)
141                     imageBitmap = null
142                     caption = ""
143                     navController.navigate(Screen.Home.route) {
144                         popUpTo(Screen.Home.route) { inclusive = true }
145                     }
146                 }
147             },
148             enabled = imageBitmap ≠ null && caption.isNotBlank(),
149             modifier = Modifier.padding(8.dp)
150         ) {
151             Text( text: "Post")
152         }
153     }
```

Fig 31. Adding post snippet

```
69      fun addPost(imageData: Bitmap, caption: String) {
70          val newPost = Post(id = (_posts.value?.maxOfOrNull { it.id } ?: 0) + 1,
71              user = MockData.currentUser,
72              imageData = imageData,
73              imageUrl = null,
74              caption = caption,
75              likes = 0,
76              isLiked = false)
77          Log.d( tag: "MainViewModel",  msg: "New Post: $newPost")
78
79          val updatedPosts = listOf(newPost) + (_posts.value ?: emptyList())
80
81          _posts.value = updatedPosts
82
83          Log.d( tag: "MainViewModel",  msg: "Post added. Total posts: ${updatedPosts.size}")
84      }
85
```

Fig 32. addPost function

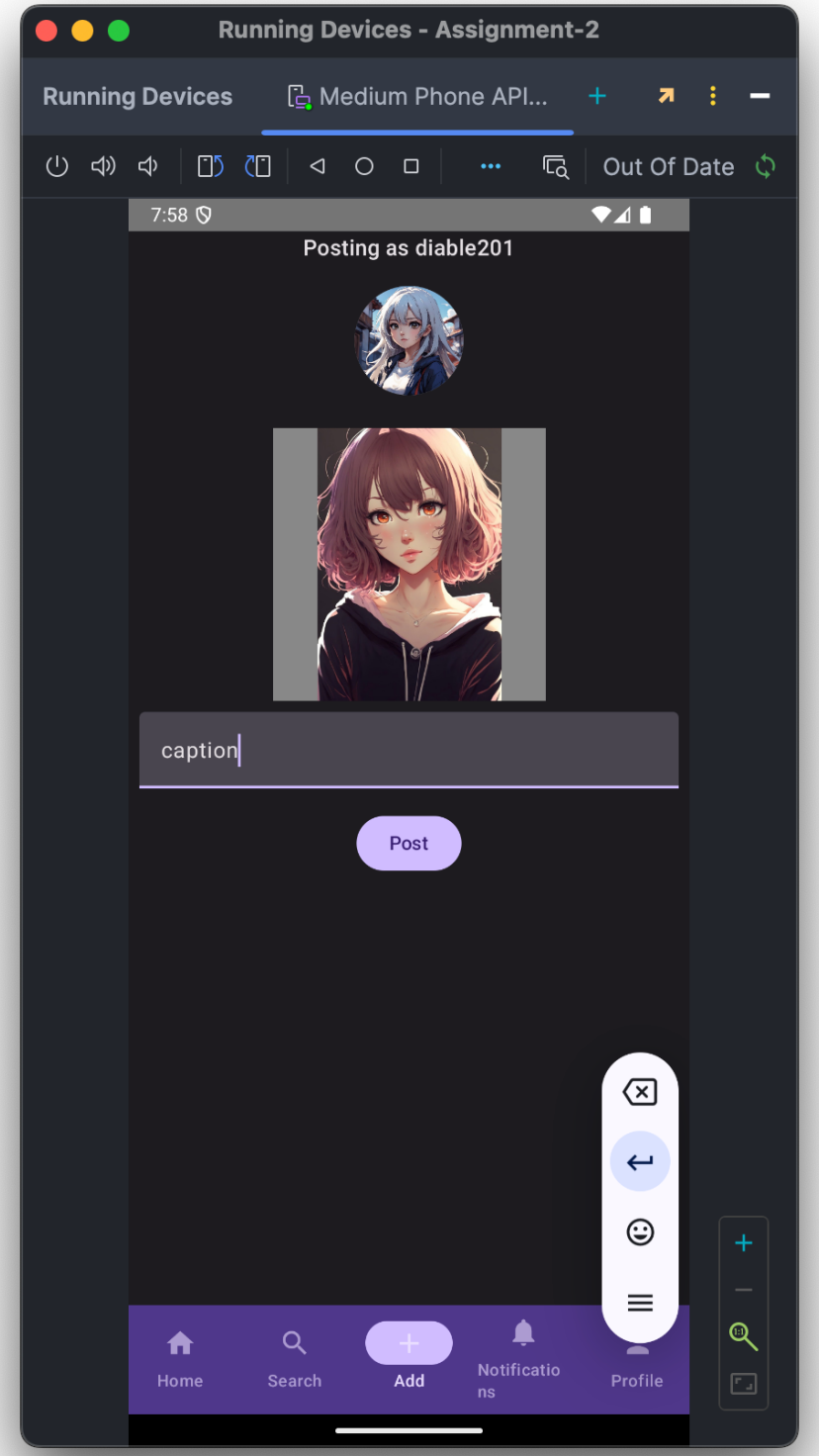
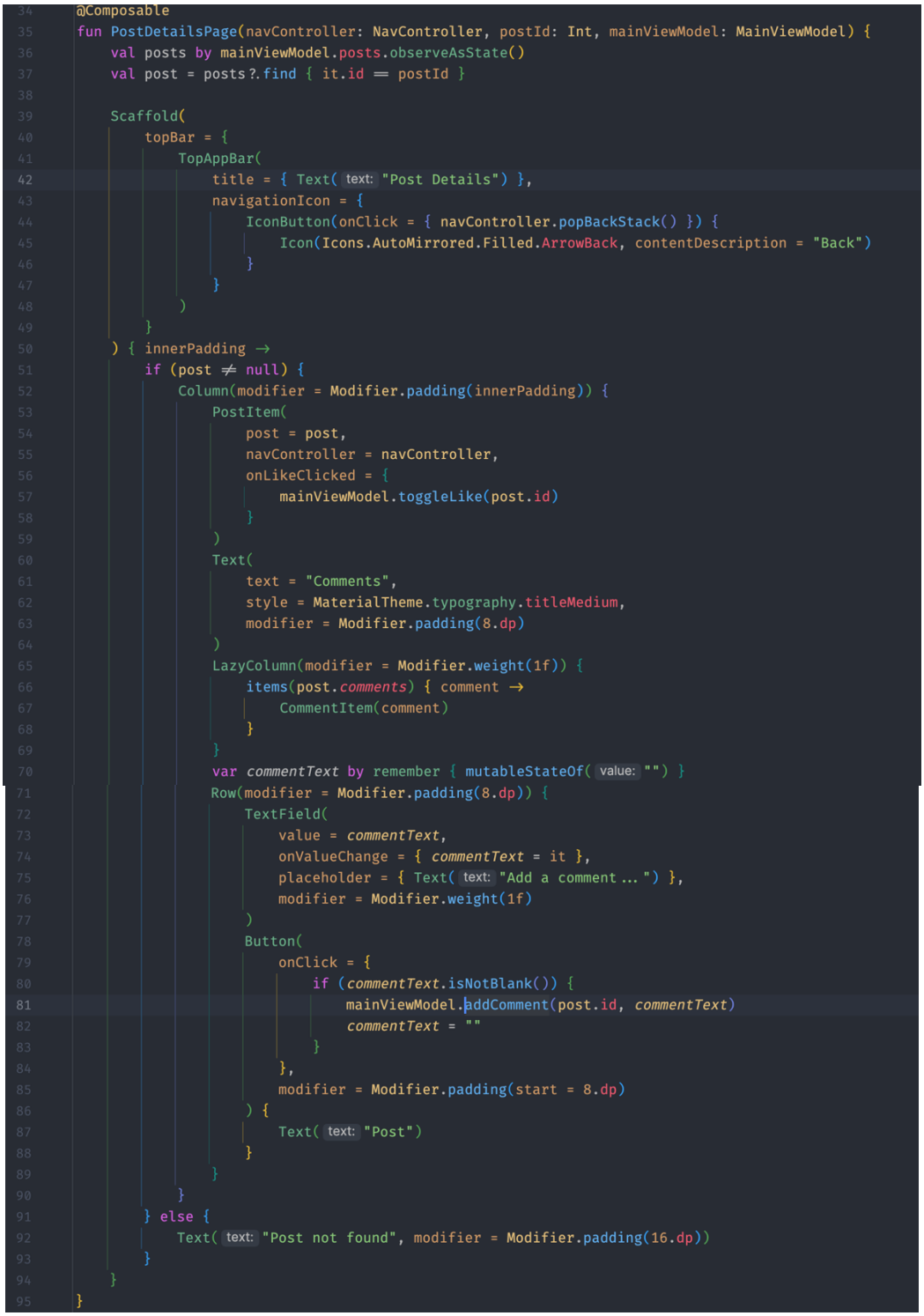


Fig 33. Adding post in application

Adding Comments

Users can add comments to posts, enhancing interaction and engagement within the app. The commenting feature allows users to view existing comments and submit new ones seamlessly.



```
@Composable
fun PostDetailsPage(navController: NavController, postId: Int, mainViewModel: MainViewModel) {
    val posts by mainViewModel.posts.observeAsState()
    val post = posts?.find { it.id == postId }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text( text: "Post Details") },
                navigationIcon = {
                    IconButton(onClick = { navController.popBackStack() }) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Back")
                    }
                }
            )
        }
    ) { innerPadding ->
        if (post != null) {
            Column(modifier = Modifier.padding(innerPadding)) {
                PostItem(
                    post = post,
                    navController = navController,
                    onLikeClicked = {
                        mainViewModel.toggleLike(post.id)
                    }
                )
                Text(
                    text = "Comments",
                    style = MaterialTheme.typography.titleMedium,
                    modifier = Modifier.padding(8.dp)
                )
                LazyColumn(modifier = Modifier.weight(1f)) {
                    items(post.comments) { comment ->
                        CommentItem(comment)
                    }
                }
                var commentText by remember { mutableStateOf( value: "") }
                Row(modifier = Modifier.padding(8.dp)) {
                    TextField(
                        value = commentText,
                        onValueChange = { commentText = it },
                        placeholder = { Text( text: "Add a comment ...") },
                        modifier = Modifier.weight(1f)
                    )
                    Button(
                        onClick = {
                            if (commentText.isNotBlank()) {
                                mainViewModel.addComment(post.id, commentText)
                                commentText = ""
                            }
                        },
                        modifier = Modifier.padding(start = 8.dp)
                    ) {
                        Text( text: "Post")
                    }
                }
            }
        } else {
            Text( text: "Post not found", modifier = Modifier.padding(16.dp))
        }
    }
}
```

Fig 34. PostDetails page snippet

```
86      fun addComment(postId: Int, commentText: String) {
87          val currentUser = MockData.currentUser
88          val timestamp = SimpleDateFormat( pattern: "hh:mm a", Locale.getDefault()).format(Date())
89          val newComment = Comment(
90              id = Random.nextInt(), user = currentUser, text = commentText, timestamp = timestamp
91          )
92
93          _posts.value = _posts.value?.map { post ->
94              if (post.id == postId) {
95                  addNotification(
96                      user = currentUser,
97                      message = "${currentUser.username} commented on your post.",
98                      targetId = post.id
99                  )
100                 post.copy(comments = post.comments + newComment)
101             } else {
102                 post
103             }
104         }
105     }
```

Fig 35. addComment function snippet

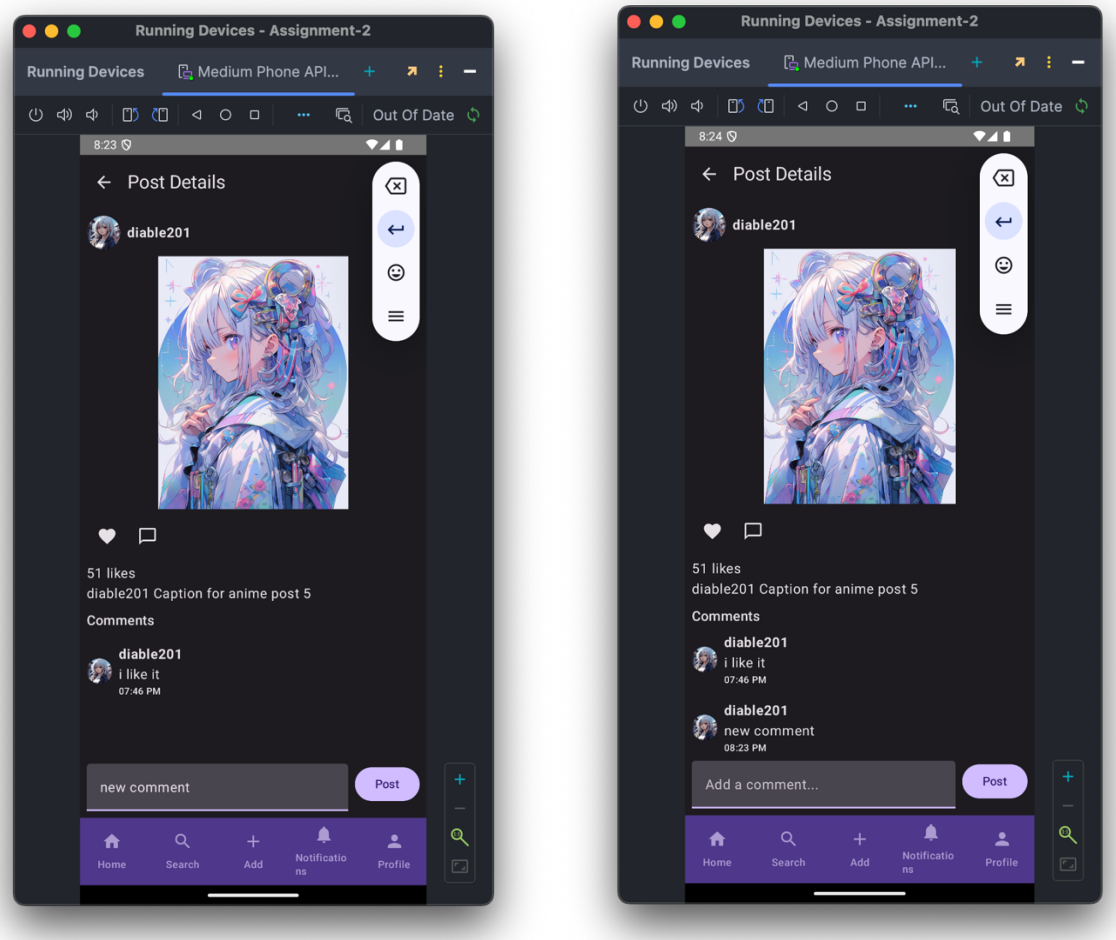


Fig 36. Comments in application

Challenges and Solutions

Problem: Every time switching to another screen was performed, it required certain arguments like userId or postId. In case the application crashed because of a missing or wrong parameter, it always raised a runtime exception on the first attempt.

Solution: Creating a sealed class screen that possessed all the necessary parameters and workable navigation paths was done with the use of helper functions; for example, createRoute, to ensure navigation received necessary input. Thus, better type safety and complete removal of missing argument problems arose from here.

```
sealed class Screen(val route: String) {
    data object Home : Screen( route: "home")
    data object Search : Screen( route: "search")
    data object AddPost : Screen( route: "add_post")
    data object Notifications : Screen( route: "notifications")
    data object Profile : Screen( route: "profile/{userId}") {
        fun createRoute(userId: Int) = "profile/$userId"
    }
    data object PostDetails : Screen( route: "post_details/{postId}") {
        fun createRoute(postId: Int) = "post_details/$postId"
    }
}
```

Fig 37. Screen snippet

Problem: To develop some kind of dynamic notification system that would be changed dynamically in real time, and by the type of notification, the user would be forwarded to material relevant to them. Initially, the difficult part was to ensure each and every notification was being sent to their correct place, say a particular post or user profile.

The Solution: Addition of notificationType and targetId to the data class of NotificationItem will provide both the type of a notification and its target, respectively. It has used ViewModel's LiveData in order to watch and update notifications in real time. Clickable NotificationCard composables were implemented and drove users according to the related screens when target ID and notification type were tapped.

```
    private fun addNotification(
        user: User, message: String, targetId: Int? = null
    ) {
        val timestamp = SimpleDateFormat( pattern: "hh:mm a", Locale.getDefault()).format(Date())
        val newNotification = NotificationItem(
            id = Random.nextInt(),
            user = user,
            message = message,
            timestamp = timestamp,
            targetId = targetId
        )
        _notifications.value = listOf(newNotification) + (_notifications.value ?: emptyList())
    }
}
```

Fig 37. addNotification snippet

Conclusion

Creating an Instagram clone was simplified with Jetpack Compose. The main features implemented included the Home Feed, Profile Page, Search feature, Add Post process, and Notifications.

This involved several challenges during development that had to be surmounted, which included navigation argument handling, dynamic notifications management, and efficient image loading. These solutions will enhance not just the functionality of the app but also best practices in building scalable and maintainable Android applications.

This project identified how important UI/UX is in developing a mobile application: how well thought out and put design contributes to ease and engagement. Besides this, some features that could be added or improved are real-time data synchronization, advanced image editing, and push notifications to enhance the capability and user engagement of the application.

References

- Jetpack Compose https://developer.android.com/jetpack/compose/documentation
- Android Navigation Component: https://developer.android.com/guide/navigation
- Coil - An Image Loading Library: https://coil-kt.github.io/coil/compose/
- Material Design Guidelines: https://m3.material.io/