Android Programming

Meruyert Oshayeva

Hi, Teacher!

Here is my practice task 6 (+ practice task 5 since I was sick, I know it's late but I still added those elements)

In this task I added a ProfileViewModel with a ViewModelFactory to manage the app's state, separating it from the composables for a cleaner architecture

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MaterialTheme {
                val application = application as ProfileApplication
                val viewModel: ProfileViewModel = viewModel(
                    factory = ProfileViewModelFactory( userRepository = application.userRepository)
                )
                ProfileApp(viewModel = viewModel)
            }
        }
    }
}
```

Then I connected the follower list to the ViewModel, so adding or removing a follower updates the state

```
val followers by viewModel.followers.collectAsState()
val userProfile by viewModel.userProfile.collectAsState()
val removedFollowerIds = remember { mutableStateOf( value = setOf<Int>()) }
```

```
val onRemoveFollower: () -> Unit = {
    scope.launch {
        val removedId = follower.id
        removedFollowerIds.value = removedFollowerIds.value + removedId

        delay( timeMillis = 300)

        viewModel.removeFollower( followerId = removedId)
        removedFollowerIds.value = removedFollowerIds.value - removedId
```

```
Button(
    onClick = {
        val randomName = creativeFollowerNames.random()
        val randomAvatar = availableAvatars.random()
        val newFollower = Follower(
            id = (followers.maxOfOrNull { it.id } ?: 0) + 1,
            name = randomName,
            avatarResId = randomAvatar
        )
        viewModel.addFollower(newFollower)
        scope.launch {
            snackbarHostState.showSnackbar( message = "🌟 $randomName started following you!")
        }
```

So I used collectAsState() on the ViewModel's snackbarMessage to show a Snackbar whenever the data is updated, providing user feedback

```
val snackbarMessage by viewModel.snackbarMessage.collectAsState()
val showSnackbar by viewModel.showSnackbar.collectAsState()
```

```
LaunchedEffect( key1 = showSnackbar) {
    if (showSnackbar && snackbarMessage.isNotEmpty()) {
        snackbarHostState.showSnackbar(snackbarMessage)
        viewModel.resetSnackbar()
    }
}
```

I implemented two-way binding for the profile data by having the screen observe a userProfile state from the ViewModel, which can be updated from a central source

```
val followers by viewModel.followers.collectAsState()
val userProfile by viewModel.userProfile.collectAsState()
val removedFollowerIds = remember { mutableStateOf( value = setOf<Int>()) }
val scope = rememberCoroutineScope()
```

And I added a "Refresh Data" option in the navigation drawer that calls a method in the ViewModel

```
HorizontalDivider(color = Color.Gray.copy(alpha = 0.5f), modifier = Modifi
DrawerItem(title = "🔄 Refresh Data", onClick = {
    scope.launch {
        viewModel.refreshUserData()
        drawerState.close()
    }
})
```

And here are some highlights from ApiService:

```
3 Usages
interface UserApiService {
    1 Usage
    @GET( value = "users")
    suspend fun getUsers(): List<ApiUser>
}

1 Usage
object RetrofitClient {
    1 Usage
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"

    val instance: UserApiService by lazy {
        Retrofit.Builder()
            .baseUrl( baseUrl = BASE_URL)
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
            .create(UserApiService::class.java)
    }
}
```

Application:

```kotlin
package com.example.practice1

import android.app.Application

2 Usages
class ProfileApplication : Application() {
    1 Usage
    val database: AppDatabase by lazy { AppDatabase.getInstance( context = this) }
    1 Usage
    val userRepository: UserRepository by lazy {
        UserRepository(
            userProfileDao = database.userProfileDao(),
            userApiService = RetrofitClient.instance
        )
    }
}
```

Database:

```kotlin
33 Usages
@Entity(tableName = "user_profile")
data class UserProfile(
    @PrimaryKey val id: Int,
    val name: String,
    val username: String,
    val email: String,
    val phone: String,
    val website: String,
    val companyName: String,
    val catchPhrase: String,
    val bio: String = "",
    val avatarResId: Int = R.drawable.ic_avatar
)

6 Usages  1 Implementation
@Dao
interface UserProfileDao {
    3 Usages  1 Implementation
    @Query( value = "SELECT * FROM user_profile WHERE id = :userId")
    suspend fun getUserProfile(userId: Int): UserProfile?

    2 Usages  1 Implementation
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUserProfile(userProfile: UserProfile)
```

```kotlin
6 Usages  1 Implementation
@Database(
    entities = [UserProfile::class],
    version = 1,
    exportSchema = false
)
abstract class AppDatabase : RoomDatabase() {
    1 Usage  1 Implementation
    abstract fun userProfileDao(): UserProfileDao

    4 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null

        1 Usage
        fun getInstance(context: android.content.Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    klass = AppDatabase::class.java,
                    name = "app_database"
                ).build()
                INSTANCE = instance
                instance
            }
```

Repisotory:

```kotlin
class UserRepository(
    private val userProfileDao: UserProfileDao,
    private val userApiService: UserApiService
) {
    2 Usages
    suspend fun refreshUserData(userId: Int = 1): Result<UserProfile> {
        return try {
            val apiUsers = userApiService.getUsers()
            val apiUser = apiUsers.firstOrNull { it.id == userId } ?: apiUsers.first()

            val userProfile = UserProfile(
                id = apiUser.id,
                name = apiUser.name,
                username = apiUser.username,
                email = apiUser.email,
                phone = apiUser.phone,
                website = apiUser.website,
                companyName = apiUser.company?.name ?: "Unknown Company",
                catchPhrase = apiUser.company?.catchPhrase ?: "No catchphrase",
                bio = "${apiUser.address?.city ?: "Unknown City"} • ${apiUser.company?.bs ?: "Professional"}"
            )

            userProfileDao.insertUserProfile(userProfile)

            Result.success( value = userProfile)
        } catch (e: Exception) {
```

ViewModel:

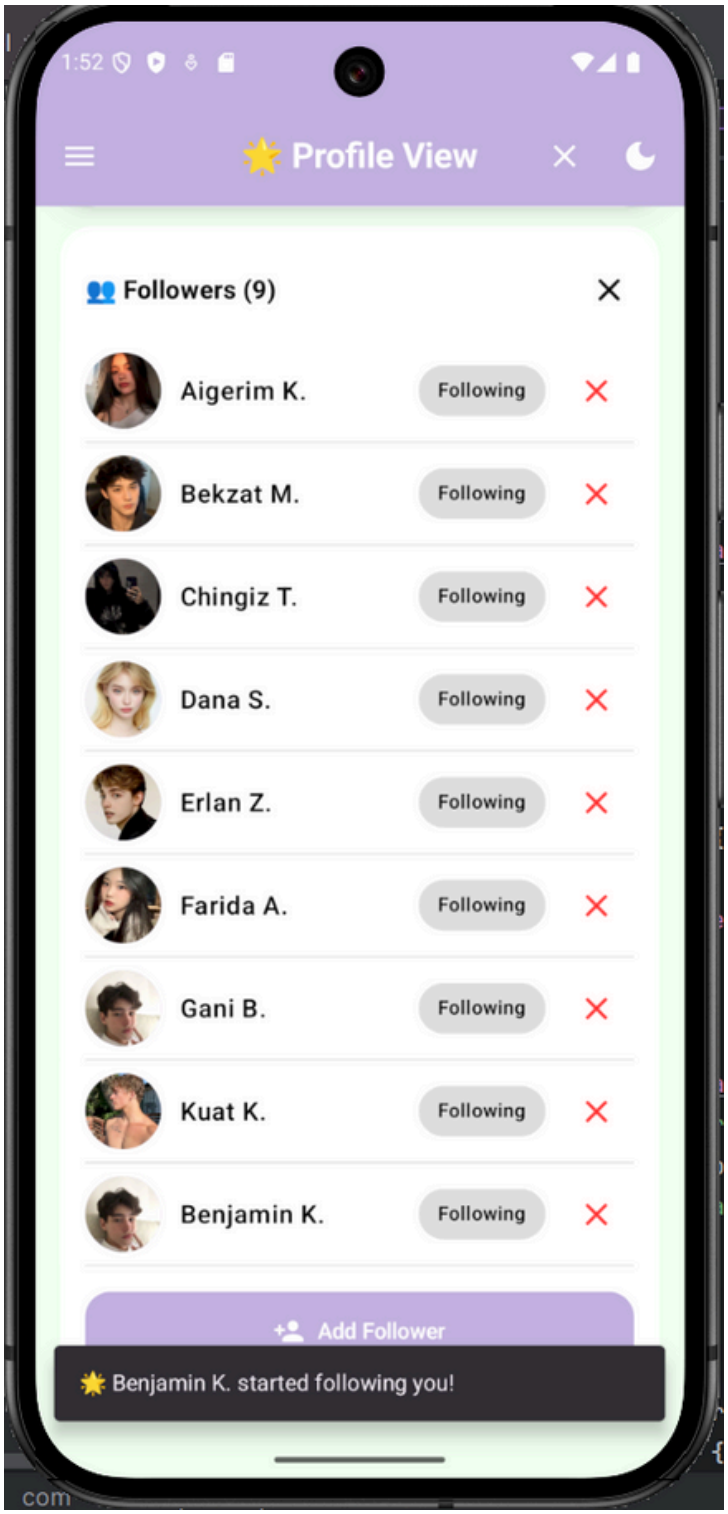
```kotlin
ass ProfileViewModel(
  private val userRepository: UserRepository
: ViewModel() {
  7 Usages
  private val _userProfile = MutableStateFlow<UserProfile?>( value = null)
  2 Usages
  val userProfile: StateFlow<UserProfile?> = _userProfile.asStateFlow()

  3 Usages
  private val _isLoading = MutableStateFlow( value = false)
  val isLoading: StateFlow<Boolean> = _isLoading.asStateFlow()

  5 Usages
  private val _followers = MutableStateFlow( value = initialFollowers)
  1 Usage
  val followers: StateFlow<List<Follower>> = _followers.asStateFlow()

  2 Usages
  private val _snackbarMessage = MutableStateFlow( value = "")
  1 Usage
  val snackbarMessage: StateFlow<String> = _snackbarMessage.asStateFlow()
```

```kotlin
fun loadUserProfile(userId: Int = 1) {
    viewModelScope.launch {
        userRepository.getUserProfile(userId).collect { profile ->
            _userProfile.value = profile
        }
    }
}

1 Usage
fun refreshUserData(userId: Int = 1) {
    viewModelScope.launch {
        _isLoading.value = true
        val result = userRepository.refreshUserData(userId)
        _isLoading.value = false

        result.onSuccess { userProfile ->
            _userProfile.value = userProfile
            showMessage("Profile updated from API!")
        }.onFailure { exception ->
            showMessage("Failed to update: ${exception.message}")
```

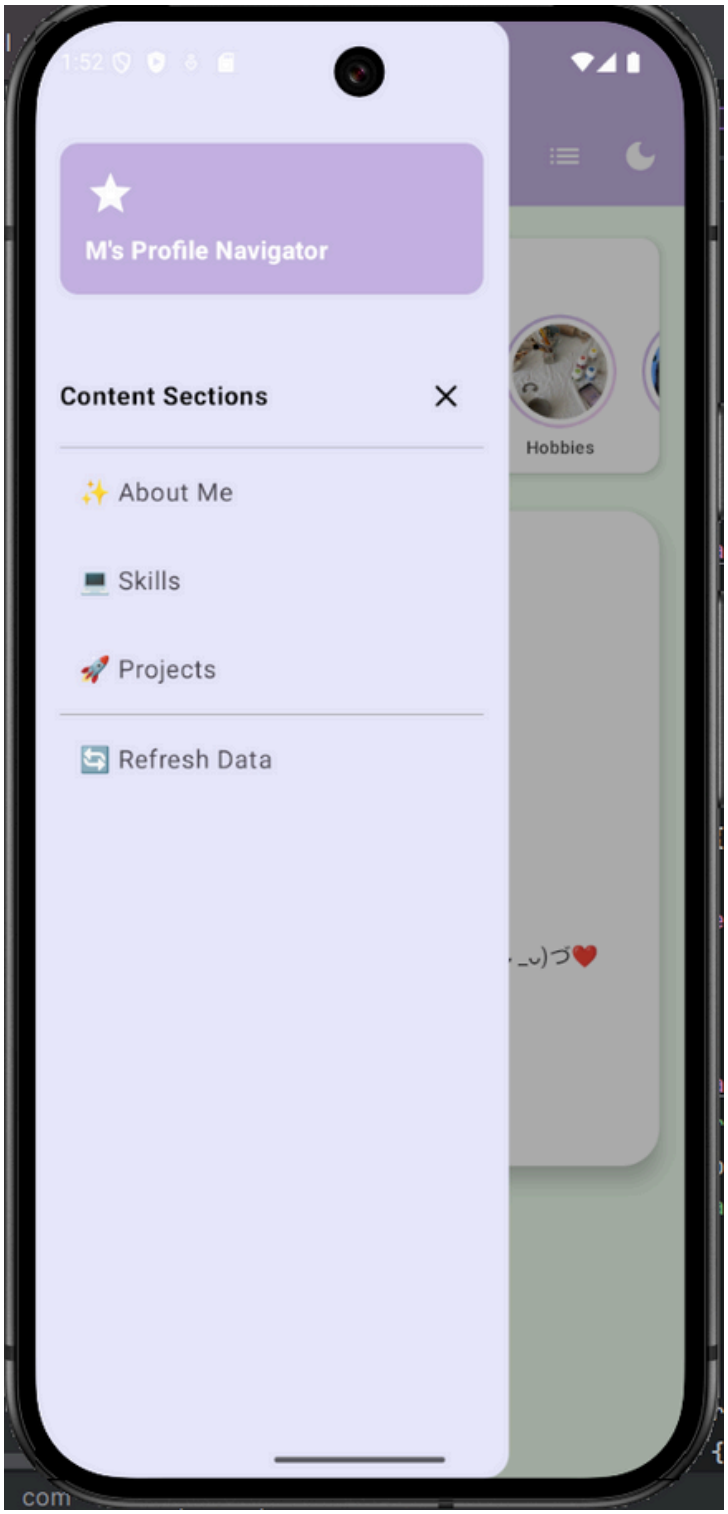
And here is how it looks like, as you can see I did some redesign

Now the Follow button changes it’s color to grey

You can also add new followers now



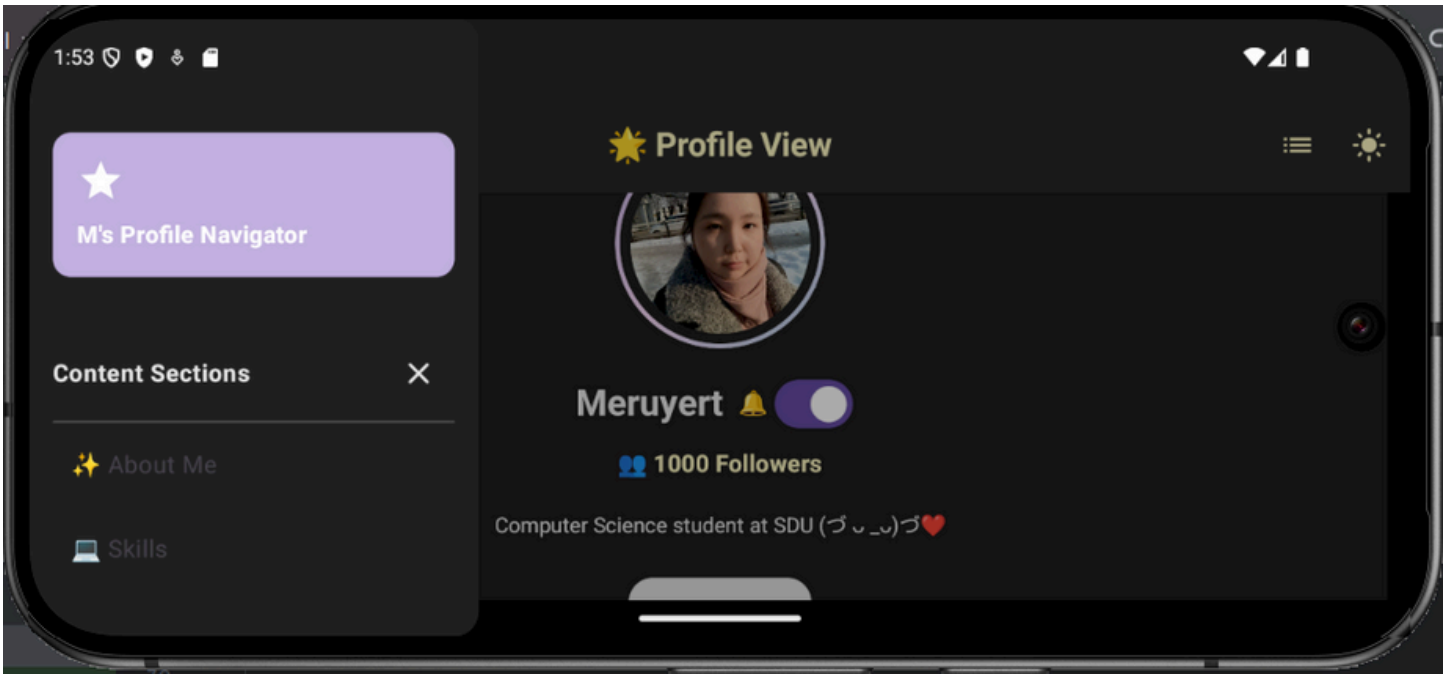
Here is the Refresh Page button, I wanted to add it to the sidebar because it fits more there rather than at the top bar



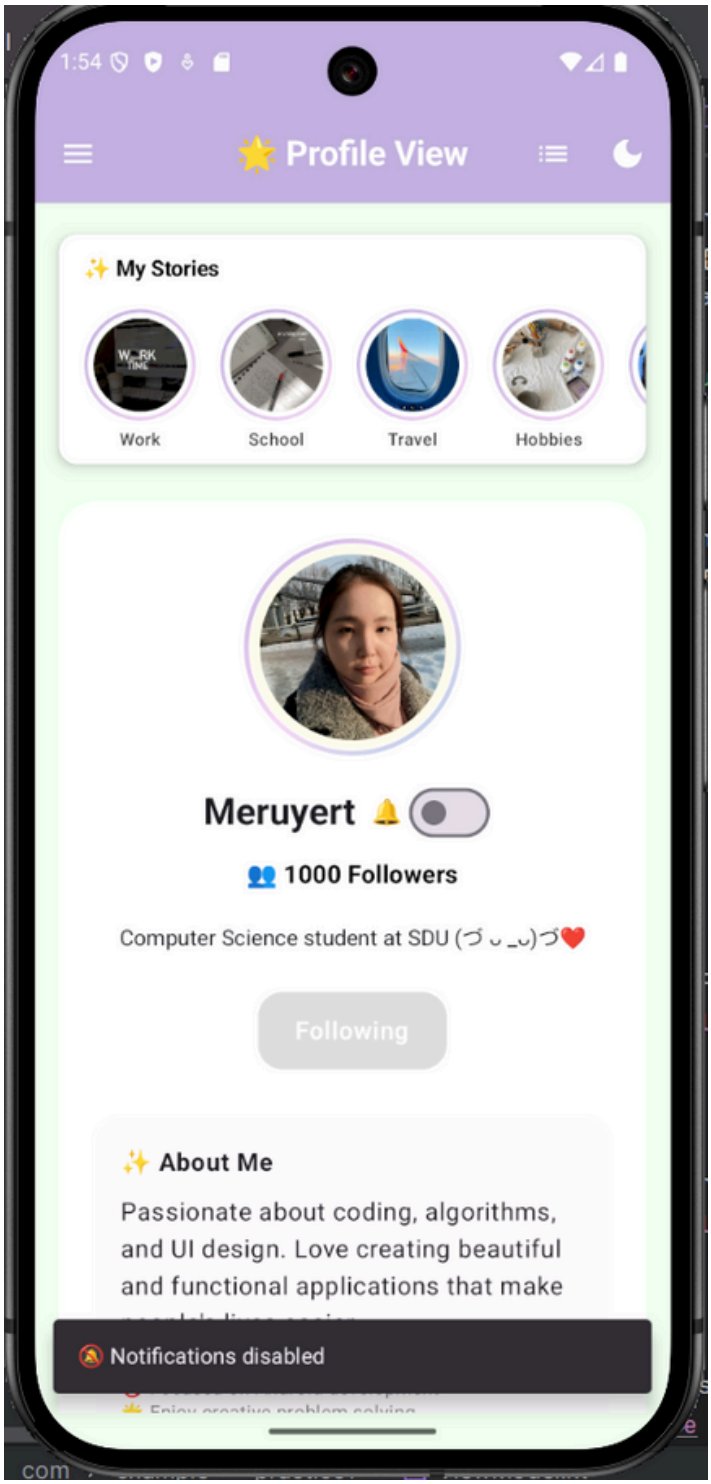
I also updated the informations on the sections

And I also wanted to show that it also works on a Dark Mode and even when you rotate it


1:53
Profile View
Work
School
Travel
Hobbies
Meruyert
1000 Followers
Computer Science student at SDU (づ｡◕‿‿◕｡)づ❤️
Following
About Me
Passionate about coding, algorithms, and UI design. Love creating beautiful and functional applications that make people's lives easier.
Always learning new technologies
Focused on Android development
Enjoy creative problem solving



Here I am saving the data



And when you restart the app, the toggles remain as we left


1:55
Profile View
My Stories
Work
School
Travel
Hobbies
Meruyert
1000 Followers
Computer Science student at SDU (づ｡◕‿‿◕｡)づ❤️
Following

Thank You!