Android Programming

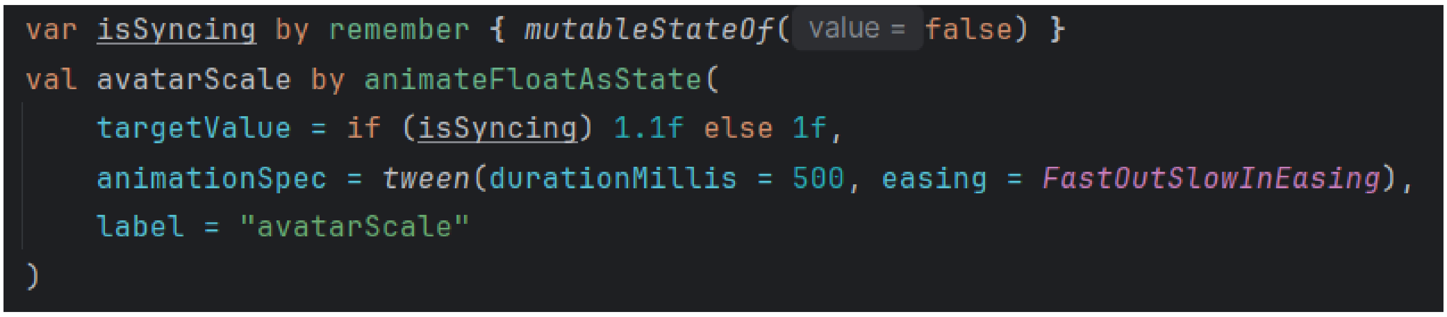
Meruyert Oshayeva

Hi, Teacher!

Here is my practice task 8 about animations

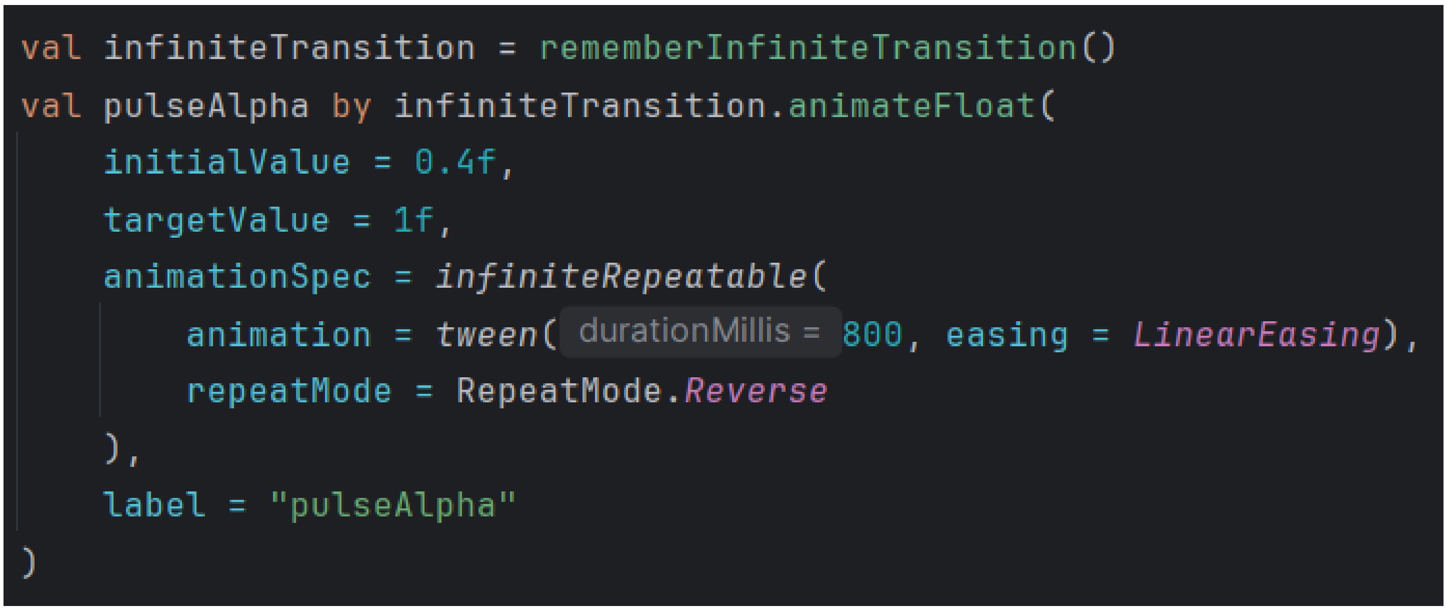
I added several animations to make the app more interactive and fun. However, struggled with the app crashing multiple times while implementing these features

First, I added a scale animation to the profile picture that activates during data sync. The avatar gently scales up and down when you refresh your profile data



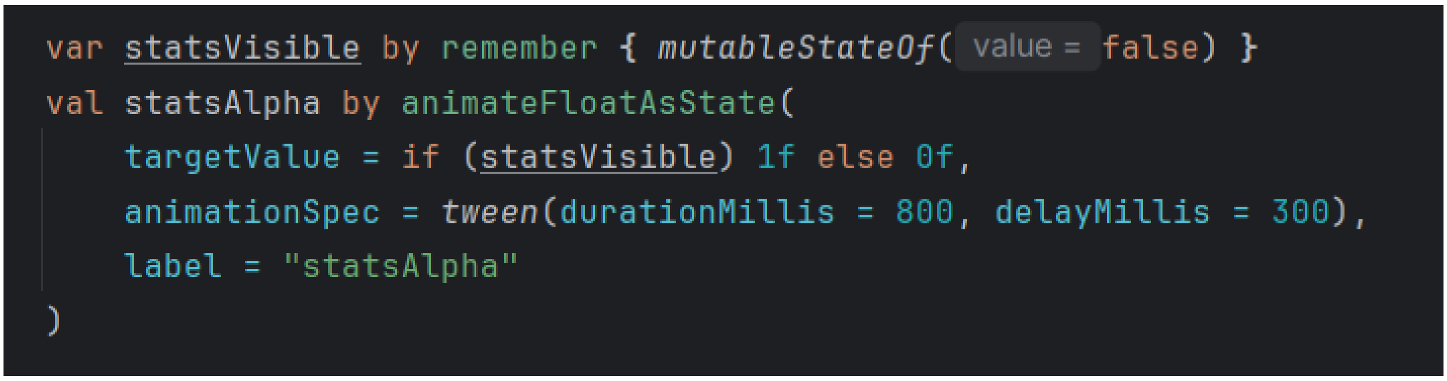
```kotlin
var isSyncing by remember { mutableStateOf( value = false) }
val avatarScale by animateFloatAsState(
    targetValue = if (isSyncing) 1.1f else 1f,
    animationSpec = tween(durationMillis = 500, easing = FastOutSlowInEasing),
    label = "avatarScale"
)
```

Next, I created a pulsing online status indicator that continuously fades in and out. This green dot shows when the user is active on the profile



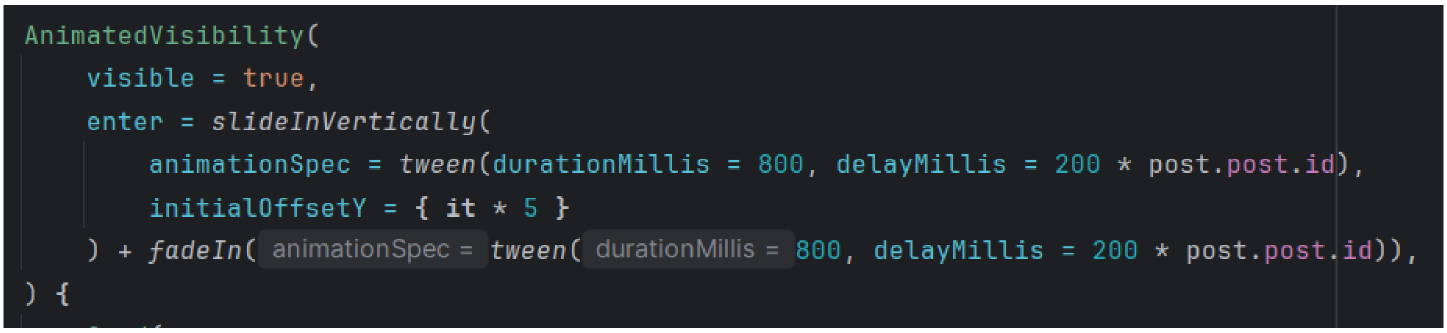
```kotlin
val infiniteTransition = rememberInfiniteTransition()
val pulseAlpha by infiniteTransition.animateFloat(
    initialValue = 0.4f,
    targetValue = 1f,
    animationSpec = infiniteRepeatable(
        animation = tween( durationMillis = 800, easing = LinearEasing),
        repeatMode = RepeatMode.Reverse
    ),
    label = "pulseAlpha"
)
```

I also implemented a fade-in animation for the follower count statistics. The follower number smoothly appears when the profile loads instead of just popping up



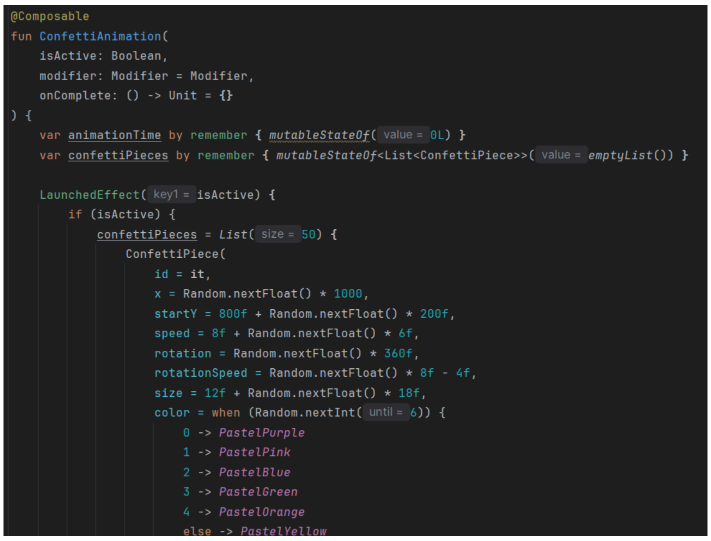
```
var statsVisible by remember { mutableStateOf( value = false) }
val statsAlpha by animateFloatAsState(
    targetValue = if (statsVisible) 1f else 0f,
    animationSpec = tween(durationMillis = 800, delayMillis = 300),
    label = "statsAlpha"
)
```

For the social feed, I added slide animations to the timeline cards. Each post now slides up from the bottom with a staggered delay between them

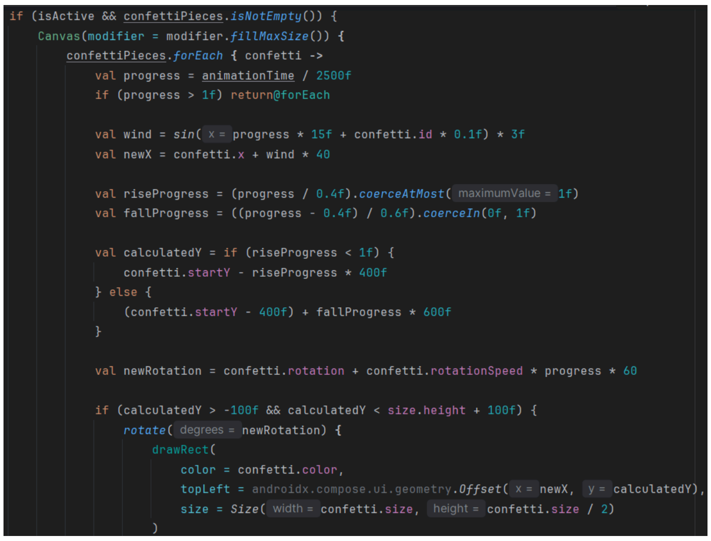


```
AnimatedVisibility(
    visible = true,
    enter = slideInVertically(
        animationSpec = tween(durationMillis = 800, delayMillis = 200 * post.post.id),
        initialOffsetY = { it * 5 }
    ) + fadeIn( animationSpec = tween( durationMillis = 800, delayMillis = 200 * post.post.id)),
) {
```

Instead of a shimmer loading effect, I implemented a celebratory confetti animation. When someone follows you, colorful confetti bursts up and falls across the screen

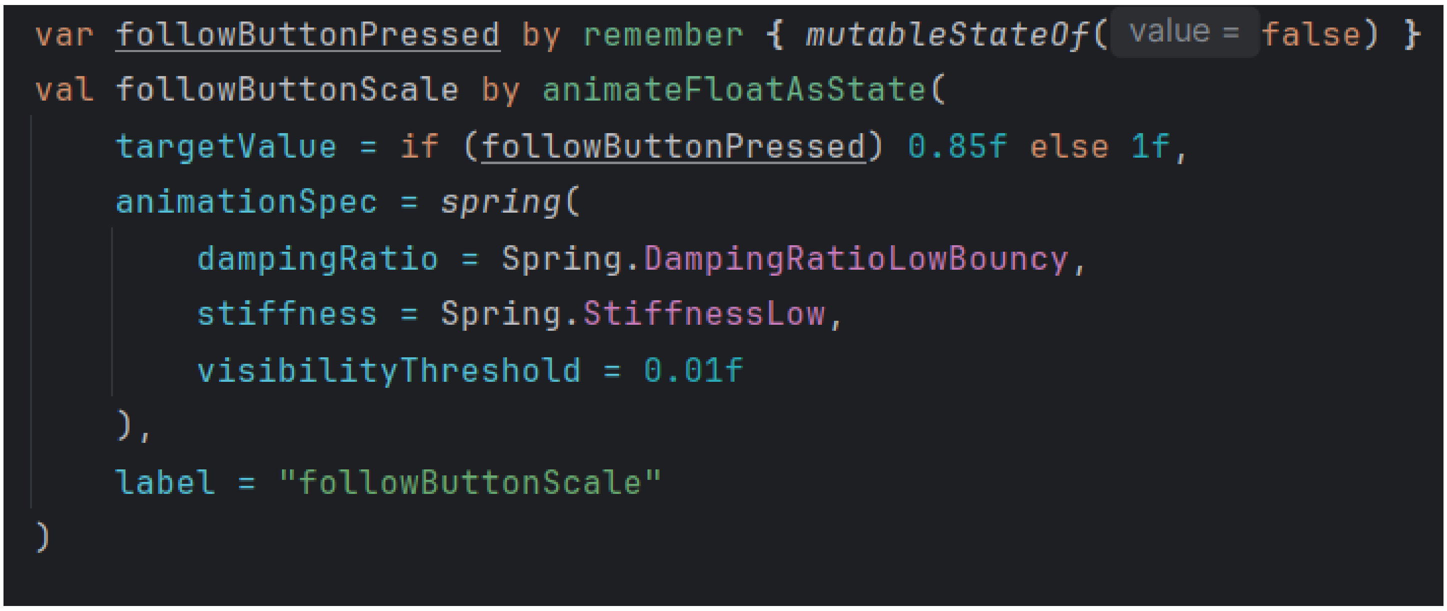


```
@Composable
fun ConfettiAnimation(
    isActive: Boolean,
    modifier: Modifier = Modifier,
    onComplete: () -> Unit = {}
) {
    var animationTime by remember { mutableStateOf( value = 0L) }
    var confettiPieces by remember { mutableStateOf<List<ConfettiPiece>>( value = emptyList()) }

    LaunchedEffect( key1 = isActive) {
        if (isActive) {
            confettiPieces = List( size = 50) {
                ConfettiPiece(
                    id = it,
                    x = Random.nextFloat() * 1000,
                    startY = 800f + Random.nextFloat() * 200f,
                    speed = 8f + Random.nextFloat() * 6f,
                    rotation = Random.nextFloat() * 360f,
                    rotationSpeed = Random.nextFloat() * 8f - 4f,
                    size = 12f + Random.nextFloat() * 18f,
                    color = when (Random.nextInt( until = 6)) {
                        0 -> PastelPurple
                        1 -> PastelPink
                        2 -> PastelBlue
                        3 -> PastelGreen
                        4 -> PastelOrange
                        else -> PastelYellow
```



```
if (isActive && confettiPieces.isNotEmpty()) {
    Canvas(modifier = modifier.fillMaxSize()) {
        confettiPieces.forEach { confetti ->
            val progress = animationTime / 2500f
            if (progress > 1f) return@forEach

            val wind = sin( x = progress * 15f + confetti.id * 0.1f) * 3f
            val newX = confetti.x + wind * 40

            val riseProgress = (progress / 0.4f).coerceAtMost( maximumValue = 1f)
            val fallProgress = ((progress - 0.4f) / 0.6f).coerceIn(0f, 1f)

            val calculatedY = if (riseProgress < 1f) {
                confetti.startY - riseProgress * 400f
            } else {
                (confetti.startY - 400f) + fallProgress * 600f
            }

            val newRotation = confetti.rotation + confetti.rotationSpeed * progress * 60

            if (calculatedY > -100f && calculatedY < size.height + 100f) {
                rotate( degrees = newRotation) {
                    drawRect(
                        color = confetti.color,
                        topLeft = androidx.compose.ui.geometry.Offset( x = newX,  y = calculatedY),
                        size = Size( width = confetti.size,  height = confetti.size / 2)
                    )
```

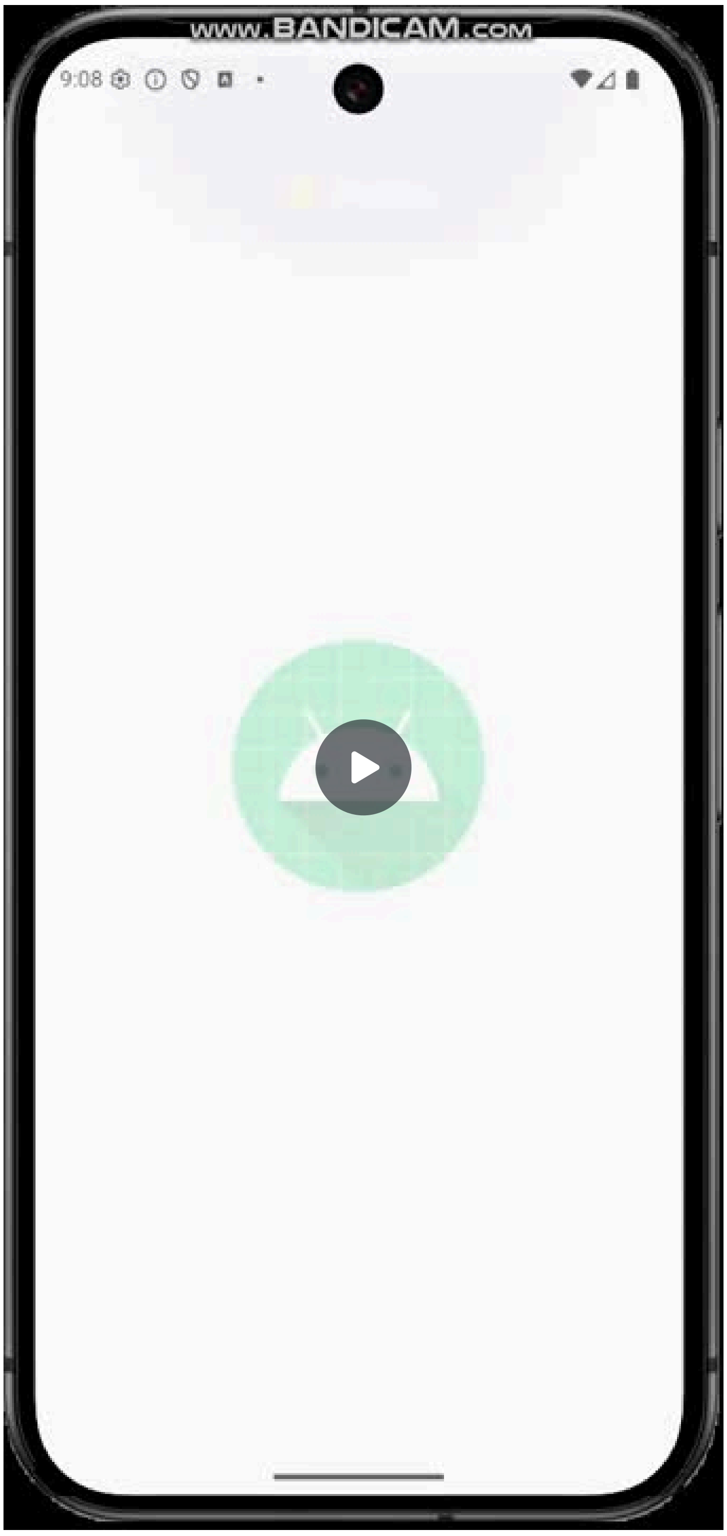
I also added spring bounce effects to the buttons for better interaction feedback. The follow and add follower buttons now scale down with a bouncy spring animation when pressed



```kotlin
var followButtonPressed by remember { mutableStateOf( value = false) }
val followButtonScale by animateFloatAsState(
    targetValue = if (followButtonPressed) 0.85f else 1f,
    animationSpec = spring(
        dampingRatio = Spring.DampingRatioLowBouncy,
        stiffness = Spring.StiffnessLow,
        visibilityThreshold = 0.01f
    ),
    label = "followButtonScale"
)
```

Despite many crashes and import issues, we successfully integrated these animations

And here is the video of my application for you to better see(I'll add this video to my Github, in case if it's not loading):


www.BANDICAM.com
9:08