Dream Catcher

Qinfeng Li, Dayu Li

Context

DreamCatcher is an app that helps users record, analyze, and explore their dreams.

Dream Log: Users can detail their dreams upon waking, adding keywords and emotional descriptions to facilitate review and analysis. The log supports voice input for quickly capturing early morning memories of dreams.

Emotion Analysis: Using AI, the app analyzes keywords and descriptions from the user's dreams, generating an emotional map (e.g., "joyful," "fear") to help users understand the underlying emotions of their dreams.

Dream Visualization: Based on the keywords and emotions in the dream, the app creates unique AI-generated artwork, producing a personalized "dream painting" for each dream. Users can compile these into a "Dream Gallery" to revisit and share their dreams.

Dream Pattern Tracking: The app identifies recurring themes and emotional patterns in users' dreams, displaying changes and patterns over time on a timeline.

Therapist Finder: Using in app map, user can find therapist near their location. The App can track their emotion history, and recommend therapist when user mood is low.

Live Demo

Main Feature

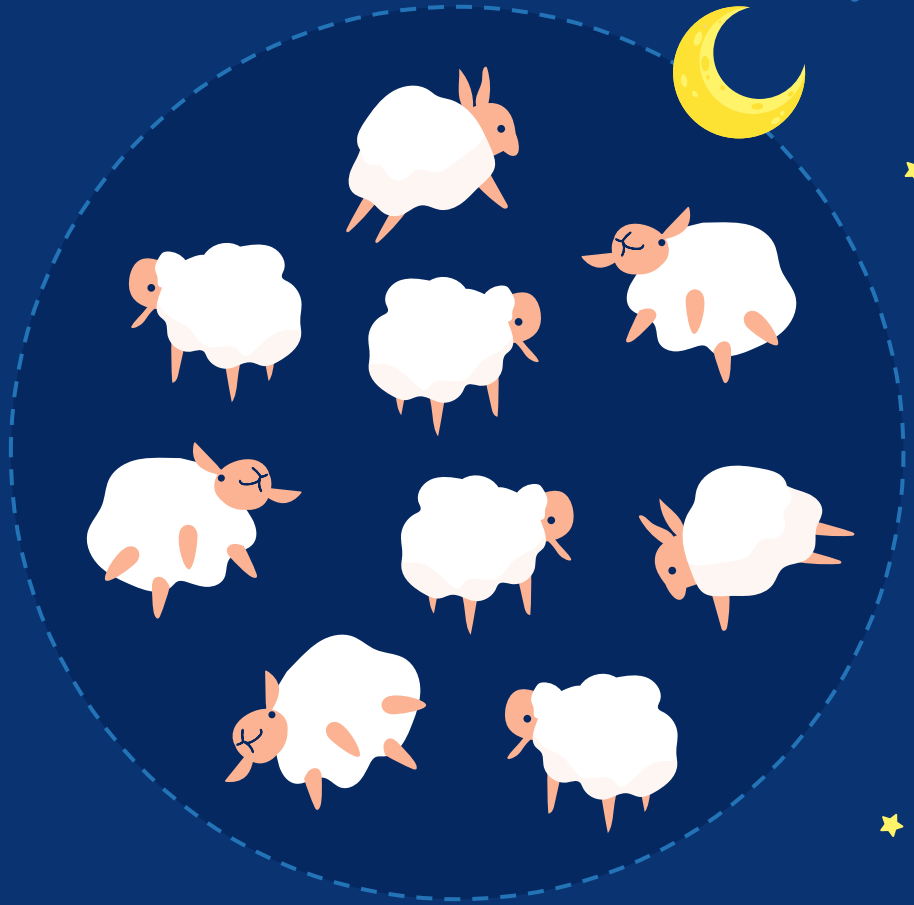
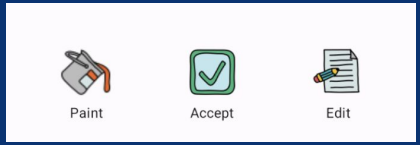


Image Generation

```
@Composable
fun ImageGeneration(prompt: String, onImageGenerated: (String) -> Unit) {
    val context = LocalContext.current
    val isLoading = remember { mutableStateOf( value: false) }
    val scope = rememberCoroutineScope()

    LaunchedEffect(prompt) {
        if (prompt.isNotEmpty()) {
            isLoading.value = true
            scope.launch {
                try {
                    // Call the OpenAI API to generate an image based on the prompt
                    val response = RetrofitInstance.openAIImageAPI.generateImage(
                        ImageRequest(prompt = prompt, n = 1, size = "512x512")
                    )
                    if (response.data.isNotEmpty()) {
                        val remoteImageUrl = response.data[0].url
                        val localImagePath = downloadImage(
                            context = context,
                            imageUrl = remoteImageUrl,
                            fileName = "dream_${System.currentTimeMillis()}.png"
                        )

                        if (localImagePath != null) {
                            onImageGenerated(localImagePath)
                        } else {
                            Toast.makeText(context, text: "Failed to save image", Toast.LENGTH_SHORT)
                                .show()
                        }
                    } else {
                        Toast.makeText(context, text: "No image generated", Toast.LENGTH_SHORT).show()
                    }
                } catch (e: Exception) {
                    Toast.makeText(context, text: "Error: ${e.message}", Toast.LENGTH_SHORT).show()
                } finally {
                    isLoading.value = false
                }
            }
        }
    }
}
```



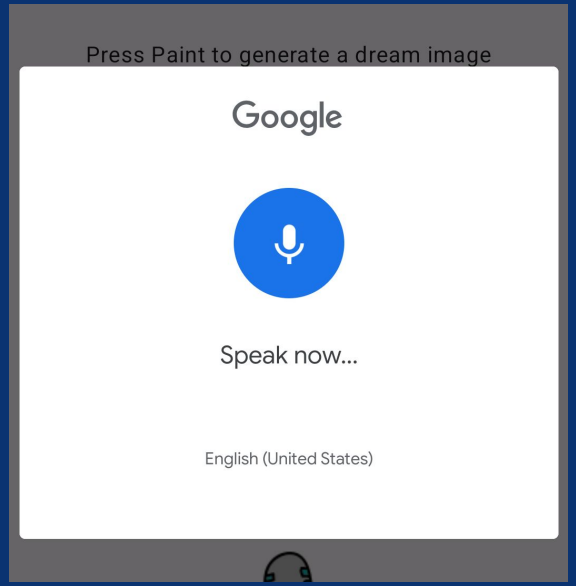

```
suspend fun downloadImage(context: Context, imageUrl: String, fileName: String): String? {
    return withContext(Dispatchers.IO) {
        try {
            val url = URL(imageUrl)
            // Open connection to the URL and get the input stream for image data
            val bitmap = BitmapFactory.decodeStream(url.openConnection().getInputStream())
            // Define the file in local storage to save the image
            val file = File(context.filesDir, fileName)
            val outputStream = FileOutputStream(file)
            //Compose into PNG
            bitmap.compress(Bitmap.CompressFormat.PNG, quality: 100, outputStream)
            outputStream.close()

            // Return the absolute path of the image file
            file.absolutePath
        } catch (e: Exception) {
            null
        }
    }
}
```

RecognizerIntent

```
if (spokenTextState.value == "Press the microphone to speak"
    || spokenTextState.value.isEmpty()
    || spokenTextState.value == "Didn't catch that. Please try again.") {
    // Show button with microphone icon and placeholder text
    Button(
        onClick = {
            isRecording.value = true
            if (ContextCompat.checkSelfPermission(context, Manifest.permission.RECORD_AUDIO)
                == PackageManager.PERMISSION_GRANTED
            ) {
                try {
                    // Prompt the user for speech and send it through a speech recognizer.
                    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH).apply {
                        putExtra(
                            RecognizerIntent.EXTRA_LANGUAGE_MODEL, // conversational speech
                            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM // casual phrases
                        )
                        // Add specific information
                        putExtra(RecognizerIntent.EXTRA_LANGUAGE, value: "en-US")
                        putExtra(RecognizerIntent.EXTRA_PROMPT, value: "Speak now...")
                    }
                    launcher.launch(intent)
                } catch (e: ActivityNotFoundException) {
                    Toast.makeText(
                        context,
                        text: "Speech recognition not supported",
                        Toast.LENGTH_SHORT
                    ).show()
                }
            } else {
                ActivityCompat.requestPermissions(
                    context as ComponentActivity,
                    arrayOf(Manifest.permission.RECORD_AUDIO),
                    requestCode: 100
                )
            }
        },
        modifier = Modifier.size(258.dp),
        colors = ButtonDefaults.buttonColors(containerColor = Color.Transparent)
    ) {
```

```
val launche
    contrac
) { result
    if (res
        val
        val
        spo
        isR
    } else
        spo
    }
}
```

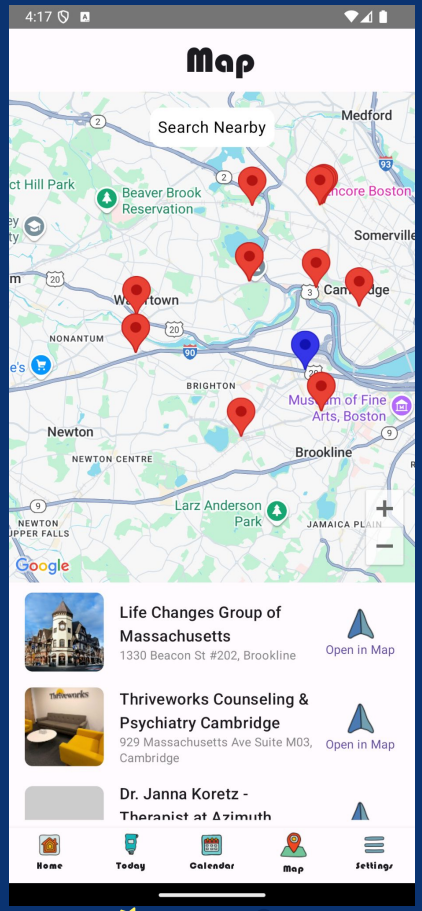
```
ıull) {
Intent.EXTRA_RESULTS)
```



```kotlin
LaunchedEffect(email) {
    email?.let {
        val user = viewModel.getUserByEmailSync(it)
        userAddress.value = user?.address
        Log.e( tag: "MapScreen",  msg: "User: ${user?.userId} User Address: ${user?.address}")

        user?.address?.let { address ->
            viewModel.fetchUserLocation(address, apiKey) { location ->
                location?.let {
                    userLatLng.value = LatLng(it.latitude, it.longitude)
                    cameraPositionState.position = CameraPosition.fromLatLngZoom(
                        LatLng(it.latitude, it.longitude),
                         zoom: 12f
                    )
                } ?: Log.e( tag: "MapScreen",  msg: "Failed to fetch user location")
            }
        }

        viewModel.fetchTherapyCenters(email, apiKey)
    }
}
```

```kotlin
private suspend fun findNearbyTherapies(lat: Double, lng: Double, apiKey: String): List<TherapyCenter> {
    return try {
        val location = "$lat,$lng"
        val response = RetrofitInstance.placesAPI.findPlaces(
            location,
            radius = 5000,
            type = "health",
            keyword = "therapist|psychiatrist",
            apiKey
        )
        if (response.status == "OK" && response.results.isNotEmpty()) {
            response.results.map { place ->
                TherapyCenter(
                    name = place.name,
                    address = place.vicinity,
                    latitude = place.geometry.location.lat,
                    longitude = place.geometry.location.lng,
                    photoReference = place.photos?.firstOrNull()?.photoReference
                )
            }
        } else {
            Log.e( tag: "Places",  msg: "Failed to find places: ${response.status}")
            emptyList()
        }
    } catch (e: Exception) {
        Log.e( tag: "Places",  msg: "Error: ${e.message}")
        emptyList()
    }
}
```

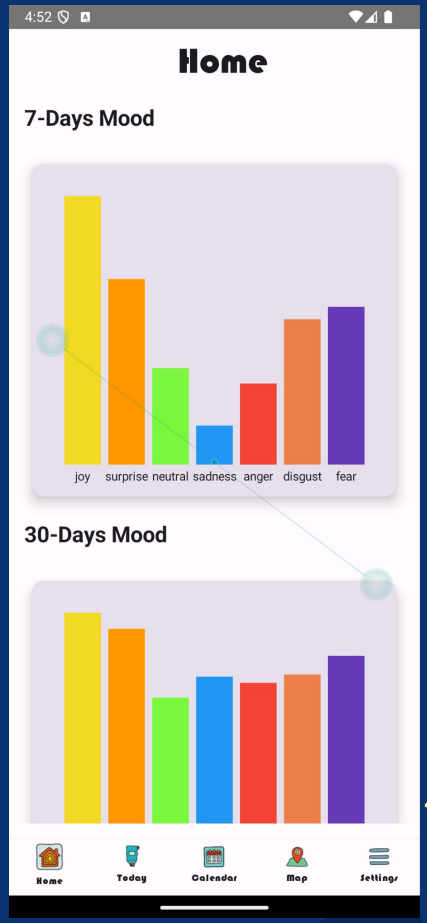
```kotlin
@Composable
fun OpenMapButton(center: TherapyCenter) {
    val context = LocalContext.current
    Column(
        modifier = Modifier
            .padding(4.dp)
            .clickable {
                //geo:latitude,longitude?q=location_name for maps
                val uri = "geo:${center.latitude},${center.longitude}?q=${center.name}"
                val mapIntent = Intent(Intent.ACTION_VIEW, Uri.parse(uri))
                mapIntent.resolveActivity(context.packageManager)
                    ?.let {
                        context.startActivity(mapIntent)
                    } ?: Log.e( tag: "TherapyCenterRow",  msg: "No map application found")
            },
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Icon(
            painter = painterResource(id = R.drawable.navigation),
            contentDescription = "Open in Map",
            tint = Color.Unspecified,
            modifier = Modifier.size(36.dp)
        )
        Text(
            text = "Open in Map",
            style = MaterialTheme.typography.bodySmall,
            color = MaterialTheme.colorScheme.primary
        )
    }
}
```



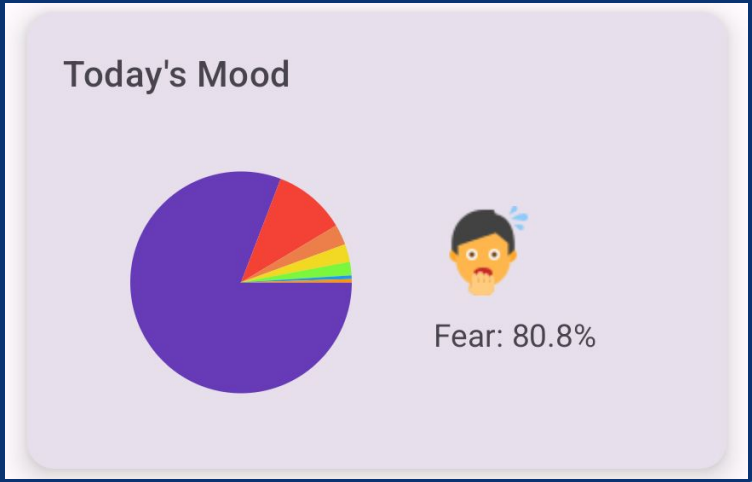
```kotlin
@Composable
fun BarChart(
    moodData: Map<String, Float>,
    textColor: Color = MaterialTheme.colorScheme.onBackground,
    modifier: Modifier = Modifier
        .fillMaxWidth()
        .height(300.dp)
        .padding(16.dp)
) {
    Log.d( tag: "BarChart",  msg: "Mood Data: $moodData")
    val moodOrder = listOf(
        "joy", "surprise", "neutral", "sadness", "anger", "disgust", "fear"
    )
    val sortedMood = moodOrder.mapNotNull { mood ->
        moodData[mood]?.let { mood to it }
    }
    val maxScore = moodData.values.maxOrNull() ?: 1f
```

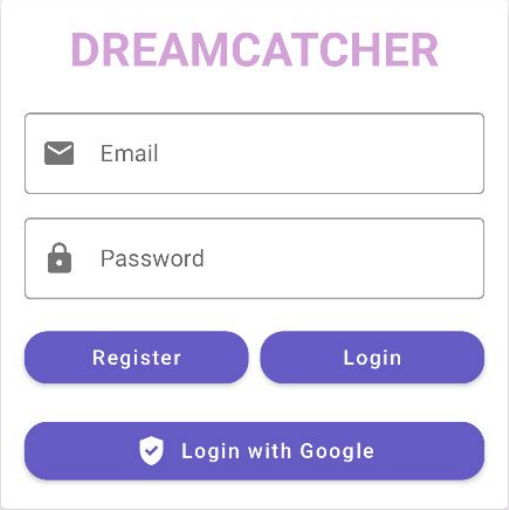
```kotlin
val moodColors = mapOf(
    "joy" to Color( color: 0xFFF2D923), // Yellow
    "sadness" to Color( color: 0xFF2196F3), // Blue
    "anger" to Color( color: 0xFFF44336), // Red
    "neutral" to Color( color: 0xFF7BF73E), // Green
    "surprise" to Color( color: 0xFFFF9800), // Orange
    "disgust" to Color( color: 0xFFED7F4A), // Dark red
    "fear" to Color( color: 0xFF673AB7) // Purple
)
```

```kotlin
            drawContext.canvas.nativeCanvas.apply {
                drawText(
                    label,
                    x: xOffset + barWidth / 2,
                    y: size.height + 16.dp.toPx(),
                    Paint().apply {
                        textAlign = android.graphics.Paint.Align.CENTER
                        textSize = 12.dp.toPx()
                        color = textColor.toArgb()
                    }
                )
            }
        }
    }
}
```



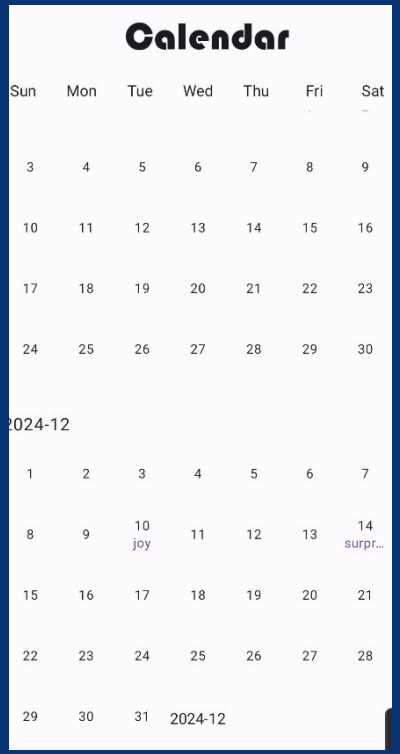
```
@Composable
fun PieChart(
    moodData: Map<String, Float>,
    modifier: Modifier = Modifier,
) {
    Canvas(modifier = modifier) {
        if (moodData.isEmpty()) return@Canvas
        val total = moodData.values.sum()

        val diameter = minOf(size.width, size.height)
        val chartSize = Size(diameter, diameter)
        var startAngle = 0f

        moodData.forEach { (label, value) ->
            val sweepAngle = (value / total) * 360f
            val color = moodColors[label] ?: Color.Gray

            drawArc(
                color = color,
                startAngle = startAngle,
                sweepAngle = sweepAngle,
                useCenter = true, // auto determined center of the pie
                topLeft = Offset(
                    x: (size.width - diameter) / 2f,
                    y: (size.height - diameter) / 2f
                ),
                size = chartSize
            )
            startAngle += sweepAngle
        }
    }
}
```

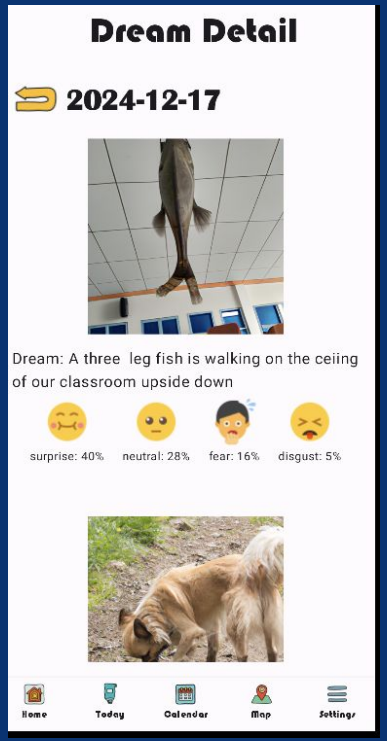




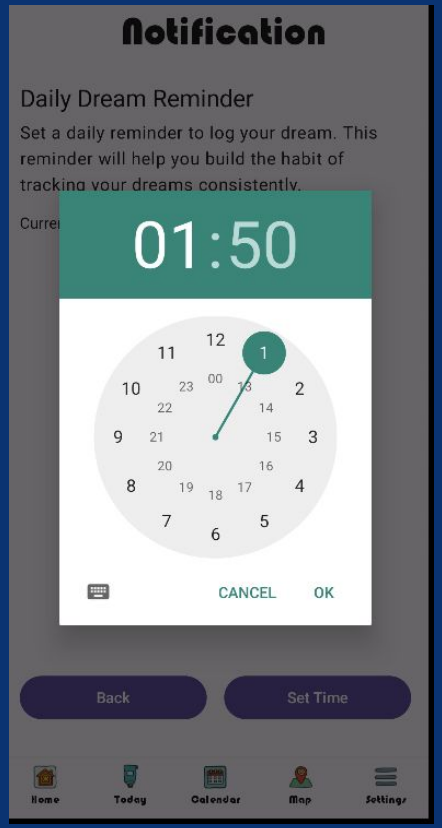
Firebase Authentication Google Login

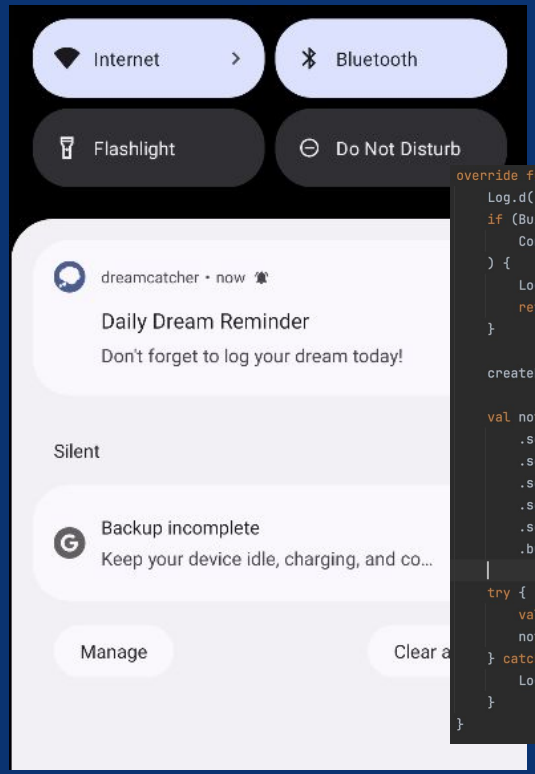
Calendar & Dream Gallery





Notification: daily reminder






```
override fun onReceive(context: Context, intent: Intent) {
    Log.d( tag: "ReminderReceiver",  msg: "Alarm triggered at ${System.currentTimeMillis()}")
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU &&
        ContextCompat.checkSelfPermission(context, Manifest.permission.POST_NOTIFICATIONS) !
    ) {
        Log.e( tag: "ReminderReceiver",  msg: "Notification permission not granted")
        return
    }

    createNotificationChannel(context)

    val notification = NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.dream)
        .setContentTitle("Daily Dream Reminder")
        .setContentText("Don't forget to log your dream today!")
        .setPriority(NotificationCompat.PRIORITY_HIGH)
        .setAutoCancel(true)
        .build()

    try {
        val notificationManager = NotificationManagerCompat.from(context)
        notificationManager.notify(NOTIFICATION_ID, notification)
    } catch (e: SecurityException) {
        Log.e( tag: "ReminderReceiver",  msg: "Failed to send notification: ${e.message}")
    }
}
```