

The background is a dark blue night sky. It features several yellow five-pointed stars of varying sizes. There are dark blue, stylized cloud shapes scattered across the scene. Three white sheep with orange faces and legs are depicted in a simple, cartoonish style. One sheep is on the left, one is on the right, and one is at the bottom center. The title "Dream Catcher" is written in a large, white, sans-serif font in the center of the image.

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.

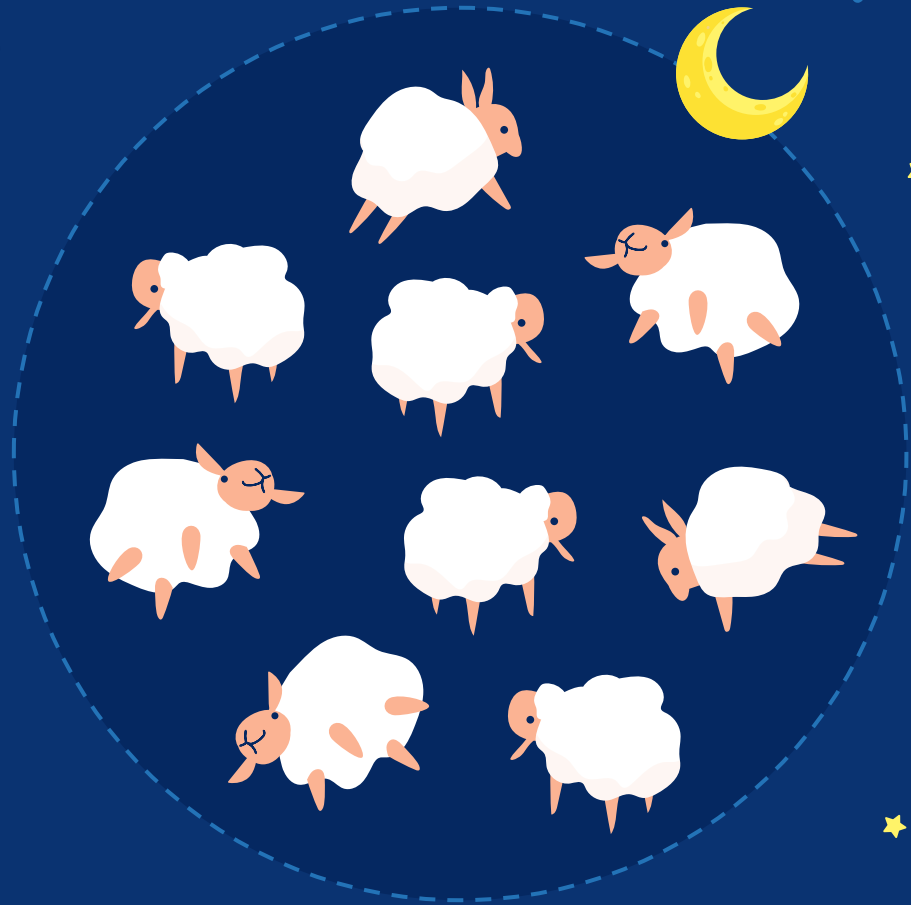
Dream Visualization: Based on the keywords and emotions in the dream, the app creates unique AI-generated artwork, producing a personalized "dream painting" for each dream. Users can compile these into a "Dream Gallery" to revisit and share their dreams.

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.



Main Feature



Speech Recognition API

```
val launcher
    contrainst
) { result
    if (res
        val
        val
        spo
        isR
    } else
        spo
    }
}

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,
                        "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)
    ) {
        will) {
            [intent.EXTRA_RESULTS)
```

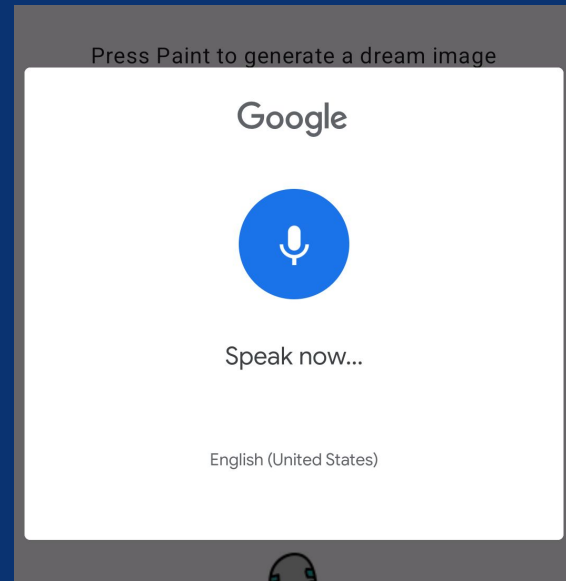


Image Generation

```
@Composable
fun ImageGeneration(prompt: String, onImageGenerated: (String) -> Unit) {
    val context = LocalContext.current
    val isLoading = remember { mutableStateOf<Boolean>(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.openAIImpAPI.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
                }
            }
        }
    }
}
```



Paint



Accept



Edit

```
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
        }
    }
}
```

```

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)
    }
}

```

```

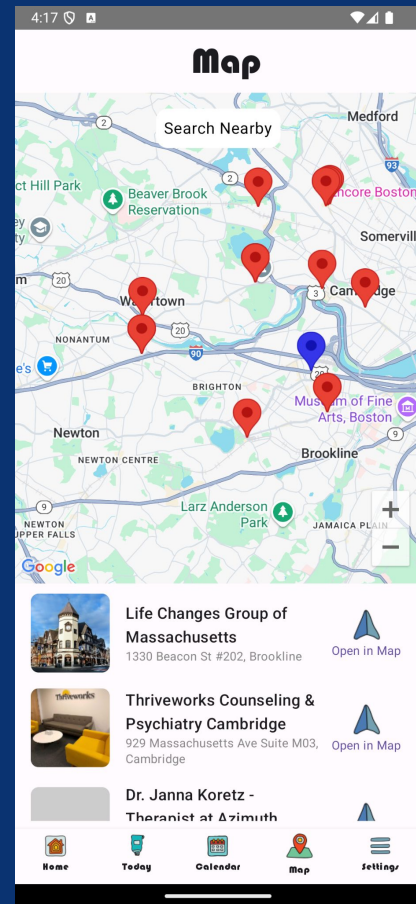
private suspend fun findNearbyTherapies(lat: Double, lng: Double, apiKey: String): List<TherapyCenter> {
    return try {
        val location = "$lat,$lng"
        val response = RetrofitInstance.placesAPI.findPlaces(
            location,
            radius = 5000,
            type = "health",
            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()
    }
}

```

```

@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
        )
    }
}

```



```

@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
    DrawingCacheEnabled = true
}

```

```

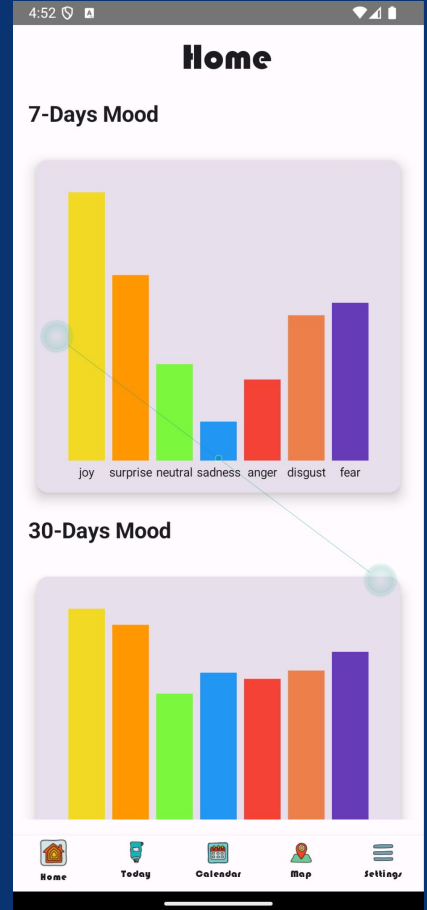
19 val moodColors = mapOf(
20     "joy" to Color( color: 0xFFFF2D923), // Yellow
21     "sadness" to Color( color: 0xFF2196F3), // Blue
22     "anger" to Color( color: 0xFFFF44336), // Red
23     "neutral" to Color( color: 0xFF7BF73E), // Green
24     "surprise" to Color( color: 0xFFFF9800), // Orange
25     "disgust" to Color( color: 0xFFED7F4A), // Dark red
26     "fear" to Color( color: 0xFF673AB7) // Purple
27 )
28

```

```

        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()

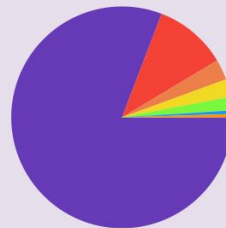
        val diameter = minOf(size.width, size.height)
        val chartSize = Size(diameter, diameter)
        var startAngle = 0f

        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
        }
    }
}

```

Today's Mood



Fear: 80.8%

DREAMCATCHER

Email



Password

Register

Login



Login with Google


Firebase Authentication Google Login


Firebase				
DreamCatcher Authentication				
项目概览				
生成式 AI				
Build with Gemini				
项目快捷方式				
Authentication				
产品类别				
按电子邮件地址、电话号码或用户 UID 搜索				
ID	提供方	创建日期 ↓	上次登录日期	用户的 UID
tingliu@bu.edu	📧	2024年12月9日	2024年12月9日	FDOOhVfX6Mfa3N09VU2rhxt...
idd1@bu.edd	📧	2024年12月8日	2024年12月8日	diMmiJtFeLa7IICOLLfGh7D6C...
ldy1@aa.bb	📧	2024年12月8日	2024年12月8日	Q6GTHIDDISe37CGYuyj7zVZt...

Calendar & Dream Gallery





Calendar						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
2024-12						
1	2	3	4	5	6	7
8	9	10 joy	11	12	13	14 surpr...
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	2024-12			

Dream Detail


 **2024-12-17**



Dream: A three leg fish is walking on the ceiling of our classroom upside down



surprise: 40% neutral: 28% fear: 16% disgust: 5%



Home

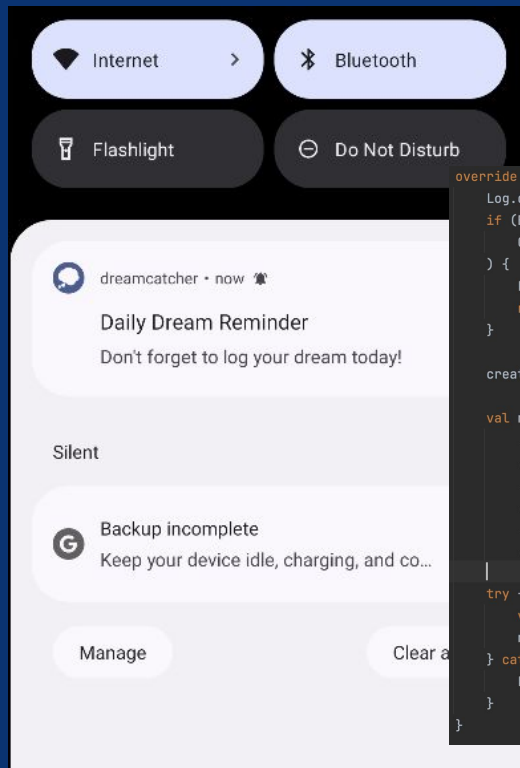
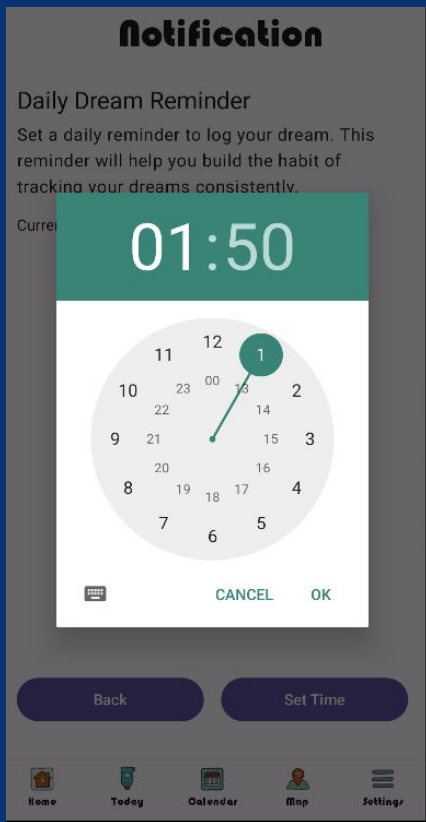
Today

Calendar

Map

Settings

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) !=
            PackageManager.PERMISSION_GRANTED) {
        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}")
    }
}
```

Live Demo