Sleep Tracking App For A Better Night Rest

1.Introduction:

1.1.Overview:

Sleep tracking is the process of monitoring a person's sleep, most commonly through measuring inactivity and movement. A device that tracks a person's sleep is called a sleep tracker. Devices capable of tracking a person's sleep include dedicated sleep trackers, trackers that clip onto person's pillow, smartphones, fitness trackers, smartwatches, and other wearable devices.

1.2.Purpose:

Sleep tracking apps use smartphones'built-in accelerometers to record and interpret sleep data each night.

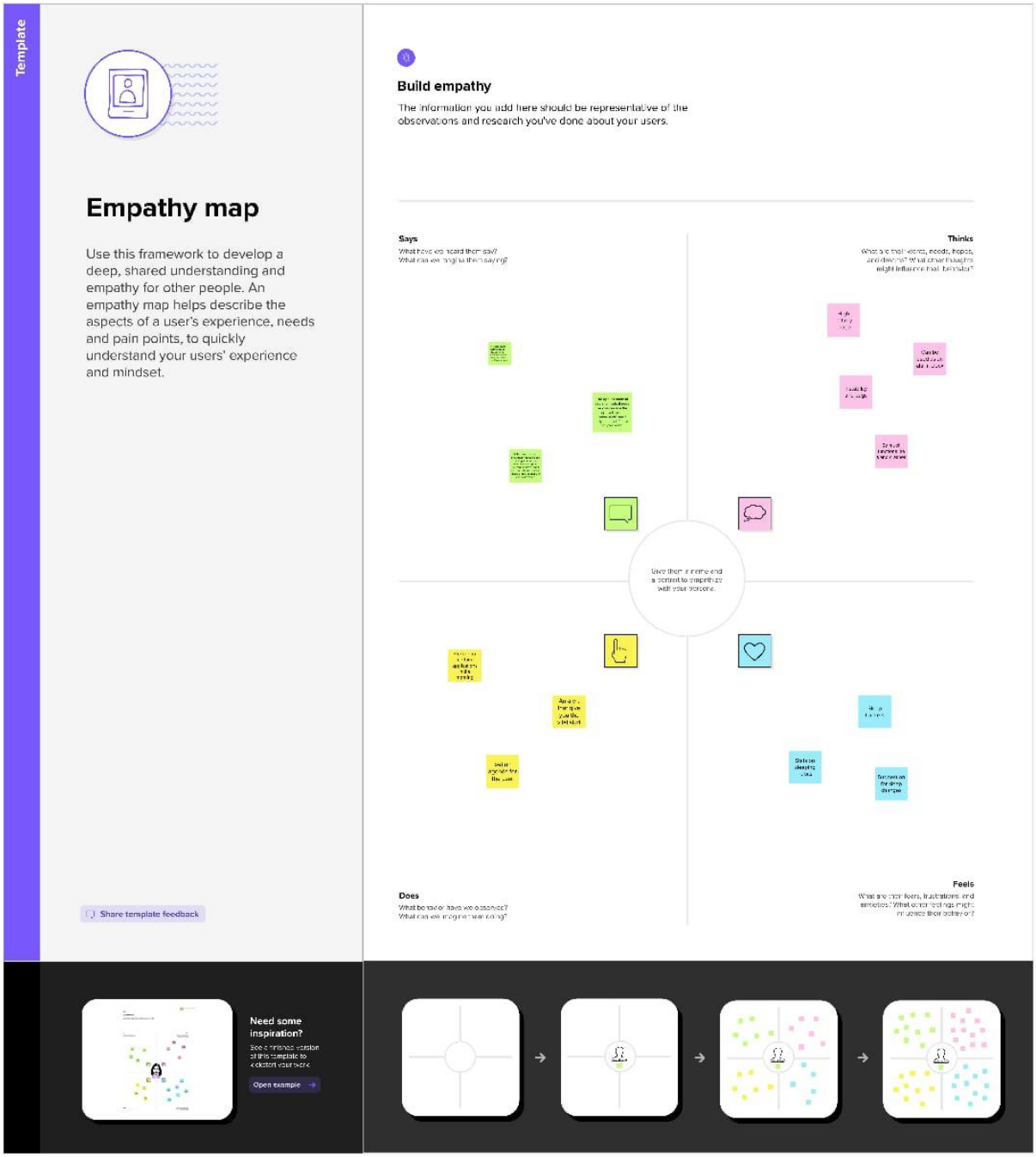
These apps commonly track movements during sleep,record sound,wake sleepers up during light stages of their sleep cycle,and provide insights to help you interpret the data.

Sleep apps monitor your sleep and provide you with an easy-to-read analysis of how well you slept.

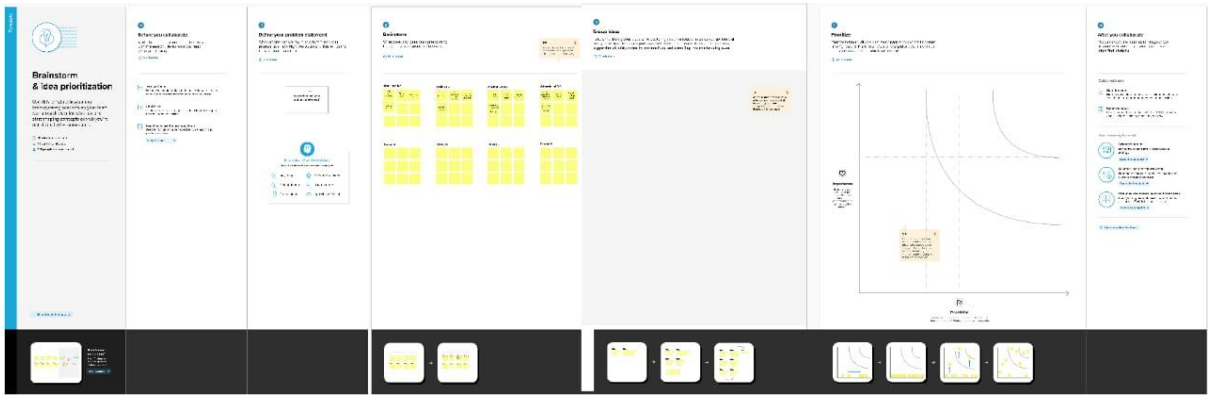
They are useful for people who want to identify how many times they wake up during the night and why, and want to set goals to improve their sleep.

2.Problem Definition & Design Thinking

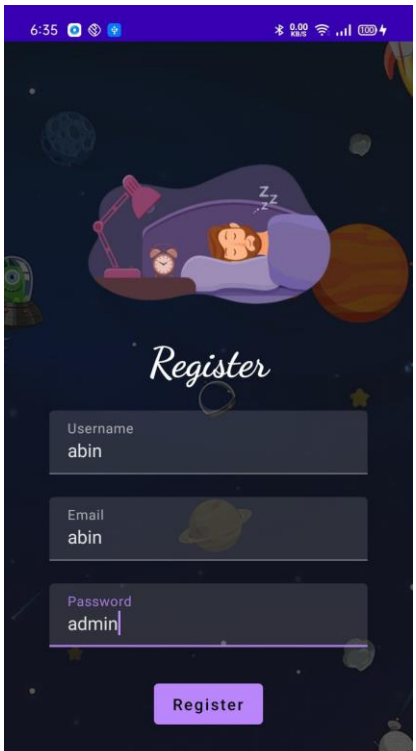
2.1.Empathy map:

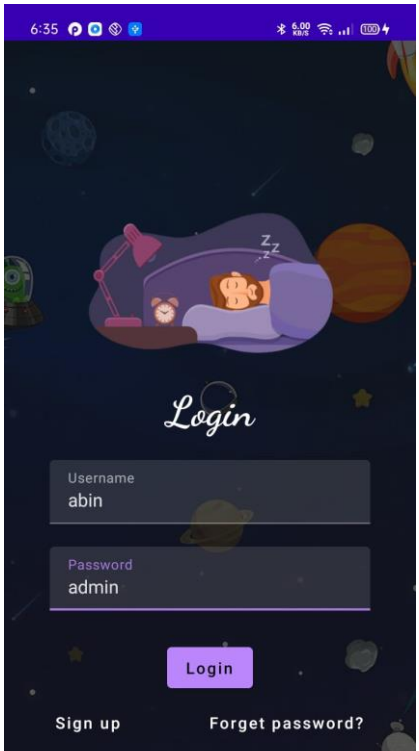


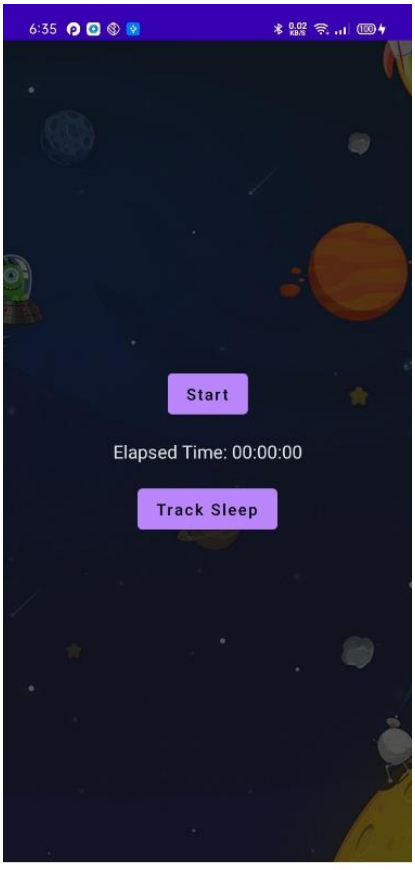
2.2.Ideation & Brainstroming map:

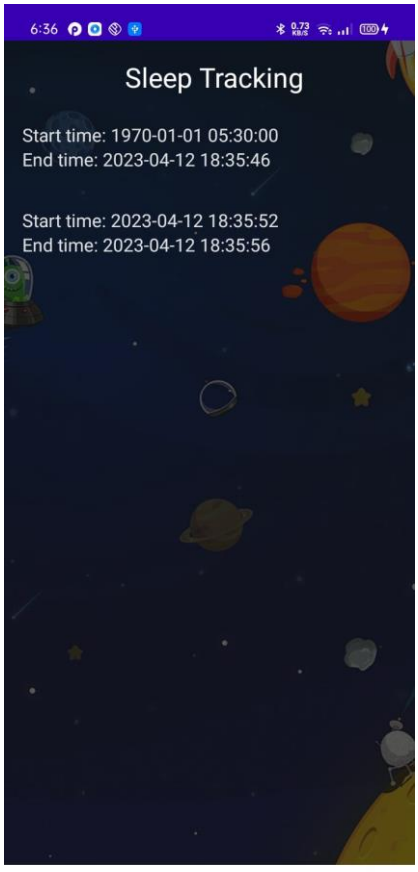


3.Result:






6:35
Start
Elapsed Time: 00:00:00
Track Sleep


6:36
Sleep Tracking
Start time: 1970-01-01 05:30:00
End time: 2023-04-12 18:35:46
Start time: 2023-04-12 18:35:52
End time: 2023-04-12 18:35:56

4.Advantage and Disadvantages:

Advantages:

Sleep-tracking applications on your phone or other smart devices can help you keep track of your sleeping cycles and patterns.

Understanding these sleep patterns will empower you to prioritize your sleep when you need it most.

Trackers can detect interrupted sleep, letting you know when you're tossing and turning or waking during the night.

Disadvantages:

Sleep-tracking apps could contribute to anxiety and perfectionism.

They can't monitor your body as accurately as a regular sleep study can. If you're having trouble sleeping, talk to your doctor.

Turns out I wasn't alone in my sleep-data fixation. According to a recent scientific study, tracking sleep with apps, wearables or in-bed sleep monitors can be directly linked to causing bedtime anxiety and stress.... which are hardly the ideal recipe for achieving a sound night's sleep.

Applications:

"Consumer sleep trackers are pretty accurate at tracking total sleep time," says Rieck. So you can feel fairly confident about using a sleep tracker to determine whether you're getting the recommended amount of sleep at night.

Plus, a sleep tracker may help improve your sleep habits. Many apps include features that address sleep hygiene — like bedtime alarms or screen time limits. And some make recommendations to improve your sleep environment — like changing the room temperature or using a white noise machine. Sleep trackers that also track daytime activities might help you notice if things like caffeine and exercise affect your nighttime sleep.

Typically, you view the collected data about your sleep in an app. The report includes total sleep time, how often you woke in the night and what times you woke up. Some devices also include a summary of your sleep stages — light sleep versus deep sleep.

Since sleep trackers are a developing technology, Rieck cautions about jumping to conclusions based on your data. He offers a few dos and don'ts to keep in mind:

CONCLUSION:

Most sleep trackers measure sleep quantity and quality by using accelerometers, small motion detectors. Accelerometers measure how much movement you're making while you sleep. This data is then analyzed using an

algorithm to estimate sleep time and quality. If you want to get more details about your sleep stages, a sleep tracker that only offers an accelerometer isn't the best fit, they can't accurately measure sleep stages because there is little difference in movement between the sleep stages, according to the Sleep Foundation

Enter sleep trackers. They come in several forms, from wearable smart watches, to headbands that provide biofeedback, rings you slip on a finger, a device you slip under your sheet, or apps that use motion detection and microphones to detect when you're in the different stages of sleep.

FUTURE SCOPE:

The “commercial-grade” devices performed about as well (and sometimes, better) when compared to the commonly used, “scientific-grade” actigraphy device. For this reason, the delineation between devices approved for research, and the devices used by the general public should not be assumed to reflect the accuracy and validity of the devices. Rather, any device that purports to measure sleep should do so in accordance with published technical standards , should demonstrate validity according to published guidelines ,and should be implemented in the context of published recommendations . Any device that meets these criteria, irrespective of how the device is marketed, should be considered appropriate for scientific use. Once a device and protocol have met these minimum standards, the choice to use that device should be made based on the quality of the data, the demonstrated utility in the

population of interest, and other factors that are adjacent to the device accuracy such as access to raw data, ability to modify assessment windows, compatibility with statistical software packages, presence of other useful sensors, and Bluetooth capabilities.

APPENDIX:

AppDatabase.kt

```
package com.example.projectone

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [TimeLog::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {

    abstract fun timeLogDao(): TimeLogDao
```

```
companion object {
    private var INSTANCE: AppDatabase? = null

    fun getDatabase(context: Context): AppDatabase {
        val tempInstance = INSTANCE
        if (tempInstance != null) {
            return tempInstance
        }
        synchronized(this) {
            val instance = Room.databaseBuilder(
                context.applicationContext,
                AppDatabase::class.java,
                "app_database"
            ).build()
            INSTANCE = instance
            return instance
        }
    }
```

```
    }
}
```

LoginActivity.kt

```
package com.example.projectone

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.alpha
```

```
import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontFamily

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import com.example.projectone.ui.theme.ProjectOneTheme



class LoginActivity : ComponentActivity() {

    private lateinit var databaseHelper: UserDatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        databaseHelper = UserDatabaseHelper(this)

        setContent {

            ProjectOneTheme {

                // A surface container using the 'background' color from the theme
```

```
            Surface(
                modifier = Modifier.fillMaxSize(),
                color = MaterialTheme.colors.background
            ) {
                LoginScreen(this, databaseHelper)
            }
        }
    }
}
@Composable
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {
    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }
    val imageModifier = Modifier
    Image(
        painterResource(id = R.drawable.sleeptracking),
        contentScale = ContentScale.FillHeight,
```

```
    contentDescription = "",
    modifier = imageModifier
        .alpha(0.3F),
)
Column(
    modifier = Modifier.fillMaxSize(),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
) {

    Image(
        painter = painterResource(id = R.drawable.sleep),
        contentDescription = "",

        modifier = imageModifier
            .width(260.dp)
            .height(200.dp)
    )
    Text(
```

```
    fontSize = 36.sp,
    fontWeight = FontWeight.ExtraBold,
    fontFamily = FontFamily.Cursive,
    color = Color.White,
    text = "Login"
)
Spacer(modifier = Modifier.height(10.dp))

TextField(
    value = username,
    onValueChange = { username = it },
    label = { Text("Username") },
    modifier = Modifier.padding(10.dp)
        .width(280.dp)
)

TextField(
    value = password,
    onValueChange = { password = it },
```

```
    label = { Text("Password") },
    modifier = Modifier.padding(10.dp)
        .width(280.dp)
)

if (error.isNotEmpty()) {
    Text(
        text = error,
        color = MaterialTheme.colors.error,
        modifier = Modifier.padding(vertical = 16.dp)
    )
}

Button(
    onClick = {
        if (username.isNotEmpty() && password.isNotEmpty()) {
            val user = databaseHelper.getUserByUsername(username)
            if (user != null && user.password == password) {
                error = "Successfully log in"
```

```
                context.startActivity(
                    Intent(
                        context,
                        MainActivity::class.java
                    )
                )

                //onLoginSuccess()
            } else {
                error = "Invalid username or password"
            }
        } else {
            error = "Please fill all fields"
        }
    },
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(text = "Login")
}
```

```
Row {
    TextButton(onClick = {context.startActivity(
        Intent(
            context,
            MainActivity2::class.java
        )
    )}
    )
    { Text(color = Color.White,text = "Sign up") }
    TextButton(onClick = {
        /*startActivity(
        Intent(
            applicationContext,
            MainActivity2::class.java
        )
    )*/
    })


    {
```

```
            Spacer(modifier = Modifier.width(60.dp))
            Text(color = Color.White,text = "Forget password?")
        }
    }
  }
}
private fun startMainPage(context: Context) {
  val intent = Intent(context, MainActivity2::class.java)
  ContextCompat.startActivity(context, intent, null)
}
```

MainActivity.kt

```
package com.example.projectone

import android.content.Context
import android.content.Intent
import android.icu.text.SimpleDateFormat
import android.os.Bundle
```

```
import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.Image

import androidx.compose.foundation.layout.*

import androidx.compose.material.Button

import androidx.compose.material.MaterialTheme

import androidx.compose.material.Surface

import androidx.compose.material.Text

import androidx.compose.runtime.*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.draw.alpha

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.unit.dp

import androidx.core.content.ContextCompat

import com.example.projectone.ui.theme.ProjectOneTheme

import java.util.*
```

```
class MainActivity : ComponentActivity() {


    private lateinit var databaseHelper: TimeLogDatabaseHelper


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = TimeLogDatabaseHelper(this)
        databaseHelper.deleteAllData()
        setContent {
            ProjectOneTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    MyScreen(this,databaseHelper)
                }
            }
        }
```

```
    }
}
@Composable
fun MyScreen(context: Context, databaseHelper: TimeLogDatabaseHelper) {
    var startTime by remember { mutableStateOf(0L) }
    var elapsedTime by remember { mutableStateOf(0L) }
    var isRunning by remember { mutableStateOf(false) }
    val imageModifier = Modifier
    Image(
        painterResource(id = R.drawable.sleeptracking),
        contentScale = ContentScale.FillHeight,
        contentDescription = "",
        modifier = imageModifier
            .alpha(0.3F),
    )

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
```

```
    verticalArrangement = Arrangement.Center
) {
    if (!isRunning) {
        Button(onClick = {
            startTime = System.currentTimeMillis()
            isRunning = true
        }) {
            Text("Start")
            //databaseHelper.addTimeLog(startTime)
        }
    } else {
        Button(onClick = {
            elapsedTime = System.currentTimeMillis()
            isRunning = false
        }) {
            Text("Stop")
            databaseHelper.addTimeLog(elapsedTime,startTime)
        }
    }
```

```
        Spacer(modifier = Modifier.height(16.dp))
        Text(text = "Elapsed Time: ${formatTime(elapsedTime- startTime)}")



        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = { context.startActivity(
            Intent(
                context,
                TrackActivity::class.java
            )
        ) }) {
            Text(text = "Track Sleep")
        }


    }


}

private fun startTrackActivity(context: Context) {
```

```
    val intent = Intent(context, TrackActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
fun getCurrentDateTime(): String {
    val dateFormat = SimpleDateFormat("yyyy-MM-dd HH:mm:ss",
Locale.getDefault())
    val currentTime = System.currentTimeMillis()
    return dateFormat.format(Date(currentTime))
}

fun formatTime(timeInMillis: Long): String {
    val hours = (timeInMillis / (1000 * 60 * 60)) % 24
    val minutes = (timeInMillis / (1000 * 60)) % 60
    val seconds = (timeInMillis / 1000) % 60
    return String.format("%02d:%02d:%02d", hours, minutes, seconds)
}
```

RegisterActivity.kt

```
package com.example.projectone
```

```
import android.content.Context

import android.content.Intent

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.Image

import androidx.compose.foundation.layout.*

import androidx.compose.material.*

import androidx.compose.runtime.*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.draw.alpha

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontFamily

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.unit.dp
```

```
import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import com.example.projectone.ui.theme.ProjectOneTheme



class MainActivity2 : ComponentActivity() {

    private lateinit var databaseHelper: UserDatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        databaseHelper = UserDatabaseHelper(this)

        setContent {

            ProjectOneTheme {

                // A surface container using the 'background' color from the theme

                Surface(

                    modifier = Modifier.fillMaxSize(),

                    color = MaterialTheme.colors.background

                ) {



                    RegistrationScreen(this,databaseHelper)
```

```
            }
        }
    }
}
```

```
@Composable
fun RegistrationScreen(context: Context, databaseHelper: UserDatabaseHelper)
{
    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    val imageModifier = Modifier
    Image(
        painterResource(id = R.drawable.sleeptracking),
        contentScale = ContentScale.FillHeight,
```

```
    contentDescription = "",
    modifier = imageModifier
        .alpha(0.3F),
)
Column(
    modifier = Modifier.fillMaxSize(),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
) {

    Image(
        painter = painterResource(id = R.drawable.sleep),
        contentDescription = "",

        modifier = imageModifier
            .width(260.dp)
            .height(200.dp)
    )
    Text(
```

```
    fontSize = 36.sp,
    fontWeight = FontWeight.ExtraBold,
    fontFamily = FontFamily.Cursive,
    color = Color.White,
    text = "Register"
)

Spacer(modifier = Modifier.height(10.dp))
TextField(
    value = username,
    onValueChange = { username = it },
    label = { Text("Username") },
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)

)

TextField(
```

```
    value = email,
    onValueChange = { email = it },
    label = { Text("Email") },
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)
)

TextField(
    value = password,
    onValueChange = { password = it },
    label = { Text("Password") },
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)
)

if (error.isNotEmpty()) {
```

```
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }

        Button(
            onClick = {
                if (username.isNotEmpty() && password.isNotEmpty() &&
email.isNotEmpty()) {
                    val user = User(
                        id = null,
                        firstName = username,
                        lastName = null,
                        email = email,
                        password = password
                    )
                    databaseHelper.insertUser(user)
```

```
            error = "User registered successfully"
            // Start LoginActivity using the current context
            context.startActivity(
                Intent(
                    context,
                    LoginActivity::class.java
                )
            )

        } else {
            error = "Please fill all fields"
        }
    },
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(text = "Register")
}
Spacer(modifier = Modifier.width(10.dp))
Spacer(modifier = Modifier.height(10.dp))
```

```
        Row() {
            Text(
                modifier = Modifier.padding(top = 14.dp), text = "Have an account?"
            )
            TextButton(onClick = {


            })


            {
                Spacer(modifier = Modifier.width(10.dp))
                Text(text = "Log in")
            }
        }
    }
}
private fun startLoginActivity(context: Context) {
    val intent = Intent(context, LoginActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
```

}

TimeDatabaseHelper.kt

```
package com.example.projectone

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import java.util.*

class TimeLogDatabaseHelper(context: Context) : SQLiteOpenHelper(context,
DATABASE_NAME, null, DATABASE_VERSION) {
    companion object {
        private const val DATABASE_NAME = "timelog.db"
        private const val DATABASE_VERSION = 1
```

```
        const val TABLE_NAME = "time_logs"
        private const val COLUMN_ID = "id"
        const val COLUMN_START_TIME = "start_time"
        const val COLUMN_END_TIME = "end_time"

        // Database creation SQL statement
        private const val DATABASE_CREATE =
            "create table $TABLE_NAME ($COLUMN_ID integer primary key
autoincrement, " +
                    "$COLUMN_START_TIME integer not null, $COLUMN_END_TIME
integer);"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        db?.execSQL(DATABASE_CREATE)
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion:
Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
```

```
        onCreate(db)
    }

    // function to add a new time log to the database
    fun addTimeLog(startTime: Long, endTime: Long) {
        val values = ContentValues()
        values.put(COLUMN_START_TIME, startTime)
        values.put(COLUMN_END_TIME, endTime)
        writableDatabase.insert(TABLE_NAME, null, values)
    }

    // function to get all time logs from the database
    @SuppressLint("Range")
    fun getTimeLogs(): List<TimeLog> {
        val timeLogs = mutableListOf<TimeLog>()
        val cursor = readableDatabase.rawQuery("select * from $TABLE_NAME",
null)
        cursor.moveToFirst()
        while (!cursor.isAfterLast) {
```

```
            val id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID))
            val startTime =
cursor.getLong(cursor.getColumnIndex(COLUMN_START_TIME))
            val endTime =
cursor.getLong(cursor.getColumnIndex(COLUMN_END_TIME))
            timeLogs.add(TimeLog(id, startTime, endTime))
            cursor.moveToNext()
        }
        cursor.close()
        return timeLogs
    }

    fun deleteAllData() {
        writableDatabase.execSQL("DELETE FROM $TABLE_NAME")
    }

    fun getAllData(): Cursor? {
        val db = this.writableDatabase
        return db.rawQuery("select * from $TABLE_NAME", null)
    }
```

```
    data class TimeLog(val id: Int, val startTime: Long, val endTime: Long?) {
        fun getFormattedStartTime(): String {
            return Date(startTime).toString()
        }

        fun getFormattedEndTime(): String {
            return endTime?.let { Date(it).toString() } ?: "not ended"
        }
    }
}
```

TimeLog.kt

```
package com.example.projectone

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.sql.Date
```

```
@Entity(tableName = "TimeLog")
data class TimeLog(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val startTime: Date,
    val stopTime: Date
)
```

TimeLogDao.kt

```
package com.example.projectone

import androidx.room.Dao
import androidx.room.Insert

@Dao
interface TimeLogDao {
    @Insert
```

```
    suspend fun insert(timeLog: TimeLog)


}
```

TrackActivity.kt

```
package com.example.projectone


import android.icu.text.SimpleDateFormat

import android.os.Bundle

import android.util.Log

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.Image

import androidx.compose.foundation.layout.*

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.foundation.lazy.LazyRow

import androidx.compose.foundation.lazy.items

import androidx.compose.material.MaterialTheme
```

```
import androidx.compose.material.Surface

import androidx.compose.material.Text

import androidx.compose.runtime.Composable

import androidx.compose.ui.Modifier

import androidx.compose.ui.draw.alpha

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import com.example.projectone.ui.theme.ProjectOneTheme

import java.util.*


class TrackActivity : ComponentActivity() {


    private lateinit var databaseHelper: TimeLogDatabaseHelper


    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
```

```
    databaseHelper = TimeLogDatabaseHelper(this)
    setContent {
        ProjectOneTheme {
            // A surface container using the 'background' color from the theme
            Surface(
                modifier = Modifier.fillMaxSize(),
                color = MaterialTheme.colors.background
            ) {
                //ListListScopeSample(timeLogs)

                val data=databaseHelper.getTimeLogs();
                Log.d("Sandeep" ,data.toString())
                val timeLogs = databaseHelper.getTimeLogs()
                ListListScopeSample(timeLogs)
            }
        }
    }
}
```

```
}



@Composable
fun ListListScopeSample(timeLogs: List<TimeLogDatabaseHelper.TimeLog>) {
    val imageModifier = Modifier
    Image(
        painterResource(id = R.drawable.sleeptracking),
        contentScale = ContentScale.FillHeight,
        contentDescription = "",
        modifier = imageModifier
            .alpha(0.3F),
    )


    Text(text = "Sleep Tracking", modifier = Modifier.padding(top = 16.dp, start =
106.dp ), color = Color.White, fontSize = 24.sp)
    Spacer(modifier = Modifier.height(30.dp))
    LazyRow(
        modifier = Modifier
```

```
        .fillMaxSize()
        .padding(top = 56.dp),

    horizontalArrangement = Arrangement.SpaceBetween
){
    item {

        LazyColumn {
            items(timeLogs) { timeLog->
                Column(modifier = Modifier.padding(16.dp)) {
                    //Text("ID: ${timeLog.id}")
                    Text("Start time: ${formatDateTime(timeLog.startTime)}")
                    Text("End time: ${timeLog.endTime?.let { formatDateTime(it) }}")
                }
            }
        }
    }

    }
```

```
    }

private fun formatDateTime(timestamp: Long): String {
    val dateFormat = SimpleDateFormat("yyyy-MM-dd HH:mm:ss",
Locale.getDefault())
    return dateFormat.format(Date(timestamp))
}
```

User.kt

```
package com.example.projectone

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "user_table")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int?,
```

```
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?,
    @ColumnInfo(name = "email") val email: String?,
    @ColumnInfo(name = "password") val password: String?,

    )
```

UserDao.kt

```
package com.example.projectone

import androidx.room.*

@Dao
interface UserDao {

    @Query("SELECT * FROM user_table WHERE email = :email")
    suspend fun getUserByEmail(email: String): User?
```

```
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUser(user: User)

    @Update
    suspend fun updateUser(user: User)

    @Delete
    suspend fun deleteUser(user: User)
}
```

UserDatabase.kt

```
package com.example.projectone

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
```

```
@Database(entities = [User::class], version = 1)
abstract class UserDatabase : RoomDatabase() {

    abstract fun userDao(): UserDao

    companion object {

        @Volatile
        private var instance: UserDatabase? = null

        fun getDatabase(context: Context): UserDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    UserDatabase::class.java,
                    "user_database"
                ).build()
                instance = newInstance
                newInstance
```

```
        }
    }
  }
}
```

userDatabaseHelper.kt

```
package com.example.projectone

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class UserDatabaseHelper(context: Context) :
  SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
```

```
companion object {
    private const val DATABASE_VERSION = 1
    private const val DATABASE_NAME = "UserDatabase.db"

    private const val TABLE_NAME = "user_table"
    private const val COLUMN_ID = "id"
    private const val COLUMN_FIRST_NAME = "first_name"
    private const val COLUMN_LAST_NAME = "last_name"
    private const val COLUMN_EMAIL = "email"
    private const val COLUMN_PASSWORD = "password"
}

override fun onCreate(db: SQLiteDatabase?) {
    val createTable = "CREATE TABLE $TABLE_NAME (" +
            "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "$COLUMN_FIRST_NAME TEXT, " +
            "$COLUMN_LAST_NAME TEXT, " +
            "$COLUMN_EMAIL TEXT, " +
            "$COLUMN_PASSWORD TEXT" +
```

```
            ")"

        db?.execSQL(createTable)
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion:
Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

    fun insertUser(user: User) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_FIRST_NAME, user.firstName)
        values.put(COLUMN_LAST_NAME, user.lastName)
        values.put(COLUMN_EMAIL, user.email)
        values.put(COLUMN_PASSWORD, user.password)
        db.insert(TABLE_NAME, null, values)
```

```
        db.close()
    }

    @SuppressLint("Range")
    fun getUserByUsername(username: String): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_FIRST_NAME = ?", arrayOf(username))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
```

```
        }
        cursor.close()
        db.close()
        return user
    }
    @SuppressLint("Range")
    fun getUserById(id: Int): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_ID = ?", arrayOf(id.toString()))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
```

```
            password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
        )
    }
    cursor.close()
    db.close()
    return user
}

@SuppressLint("Range")
fun getAllUsers(): List<User> {
    val users = mutableListOf<User>()
    val db = readableDatabase
    val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
    if (cursor.moveToFirst()) {
        do {
            val user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
```

```
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
            users.add(user)
        } while (cursor.moveToNext())
    }
    cursor.close()
    db.close()
    return users
  }


}
```