SLEEP TRACKER

CONTENTS

INTRODUCTION

1.1 Overview

Sleep is a multifaceted and dynamic phenomenon that indicates individuals' overall health and well-being and is affected by a variety of factors such as behavioral habits, stress, and disorders. Sleep disturbances are common across different population groups (e.g., older people and pregnant women) and negatively impact body functions, including the cardiovascular and immune system and hormonal release. Such sleep problems need to be investigated thoroughly to reduce the associated heal thrisks and complications. Monitoring sleep quality is a vital step in this regard when the individuals' sleep parameters are tracked.

Sleep quality assessment methods have been conventionally performed in clinical settings by monitoring users' biological signals and body movements. Polysomnography (PSG), the gold standard method used for sleep analysis, is enabled by the continuous monitoring of different cardiorespiratory and neurophysiological indicators. Owing to PSG's complex and multichannel data collection, this method is limited to short-term hospital or laboratory-based monitoring. Actigraphy is another well-established method enabled by a 3D accelerometer that captures the movements of a limb to monitor sleep.

1.2 Purpose

Sleep trackers are devices used to measure sleep quality. Sleep trackers measure how long you have slept, the depth of your sleep, and other sleeping habits. Sleep trackers can be valuable in gaining insight into how much sleep you are getting and, as a result, making lifestyle changes to optimise your sleeping experience.

A good night's sleep is essential for your overall health and well-being. Conserving energy, healing the body, consolidating memories, and regulating emotions are some of the key reasons why we sleep. Normally, human adults require anywhere between seven and nine hours of undisturbed sleep during the night, so that the body can get the optimum rest it requires. However, professional commitments and changing lifestyles are some of the key factors that negatively impact our sleep health.

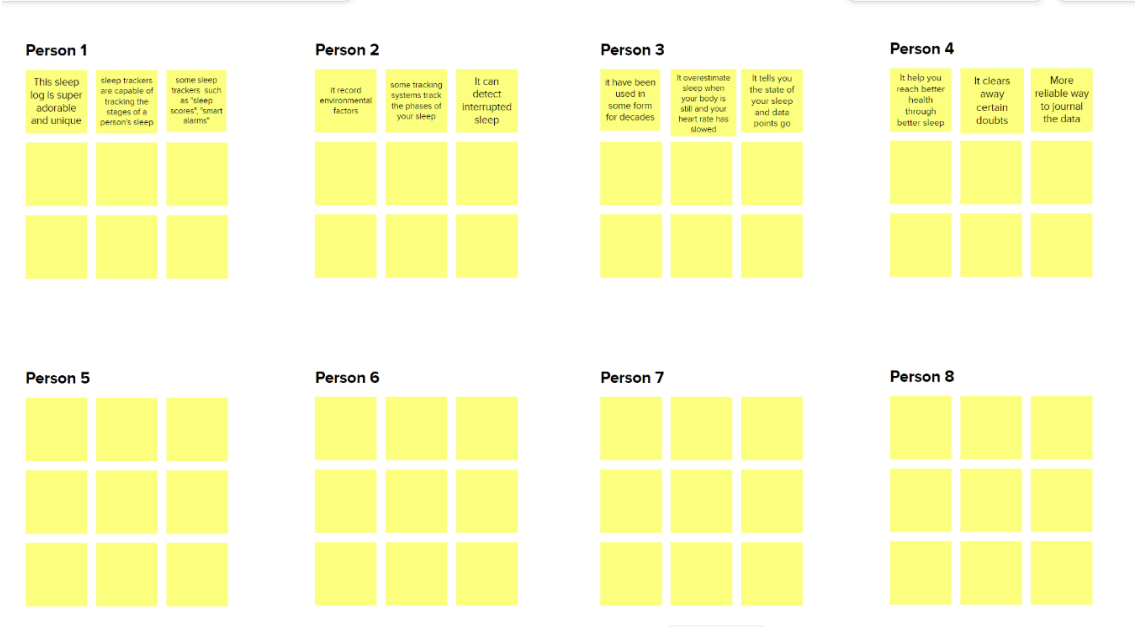
PROBLEM DEFINITION AND

DESIGN THINKING

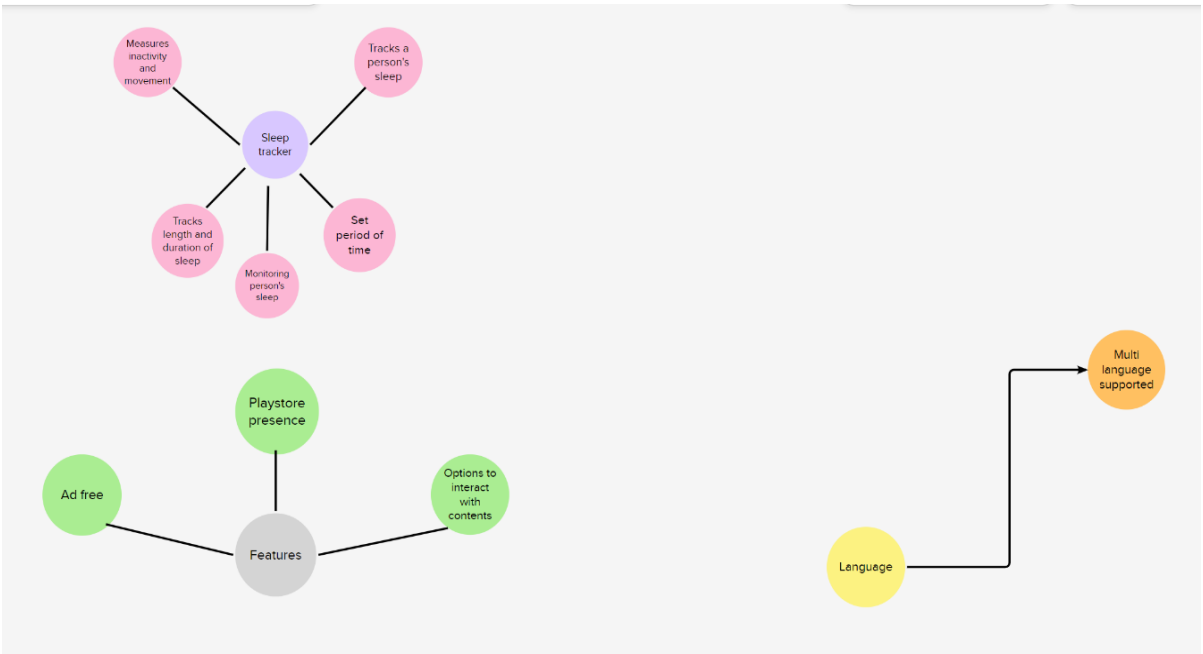
2.1 EMPATHY MAP

2.2 IDEATION AND BRAINSTORMING MAP

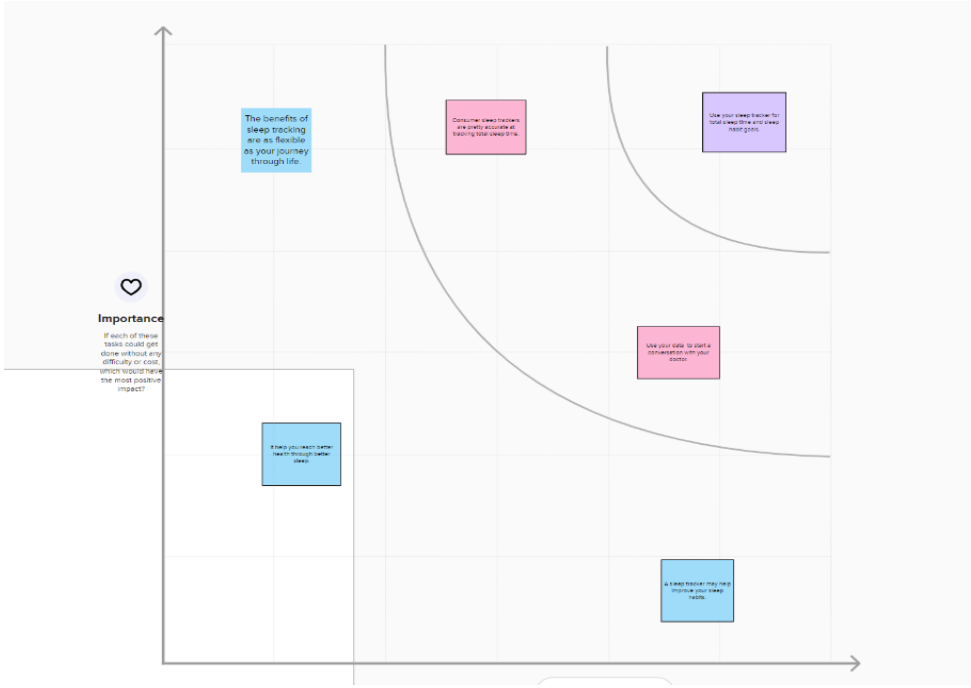
BRAINSTORM:



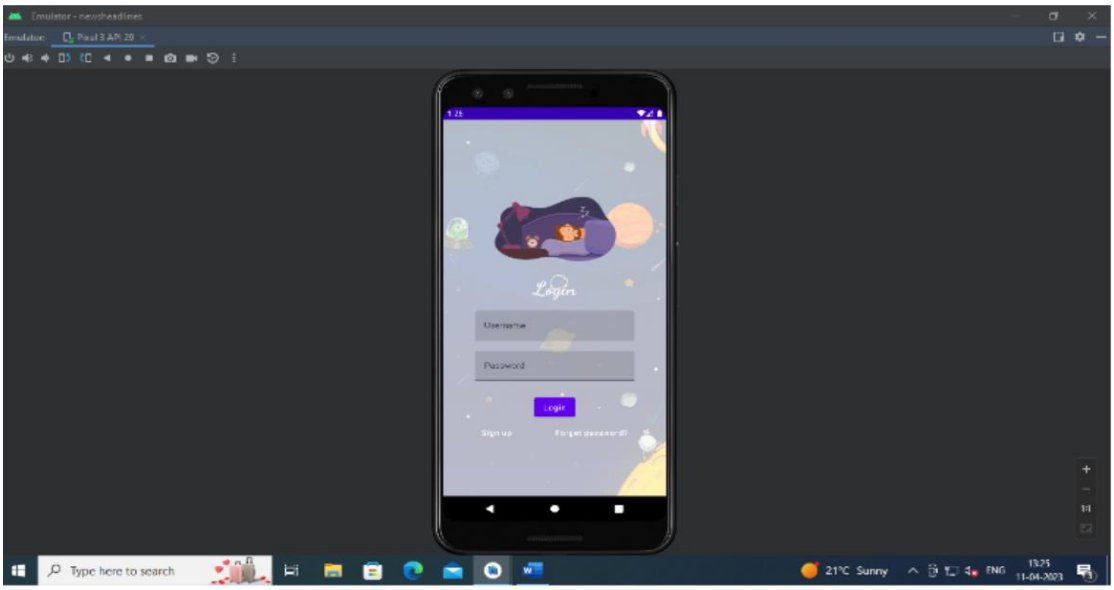
GROUP IDEAS:

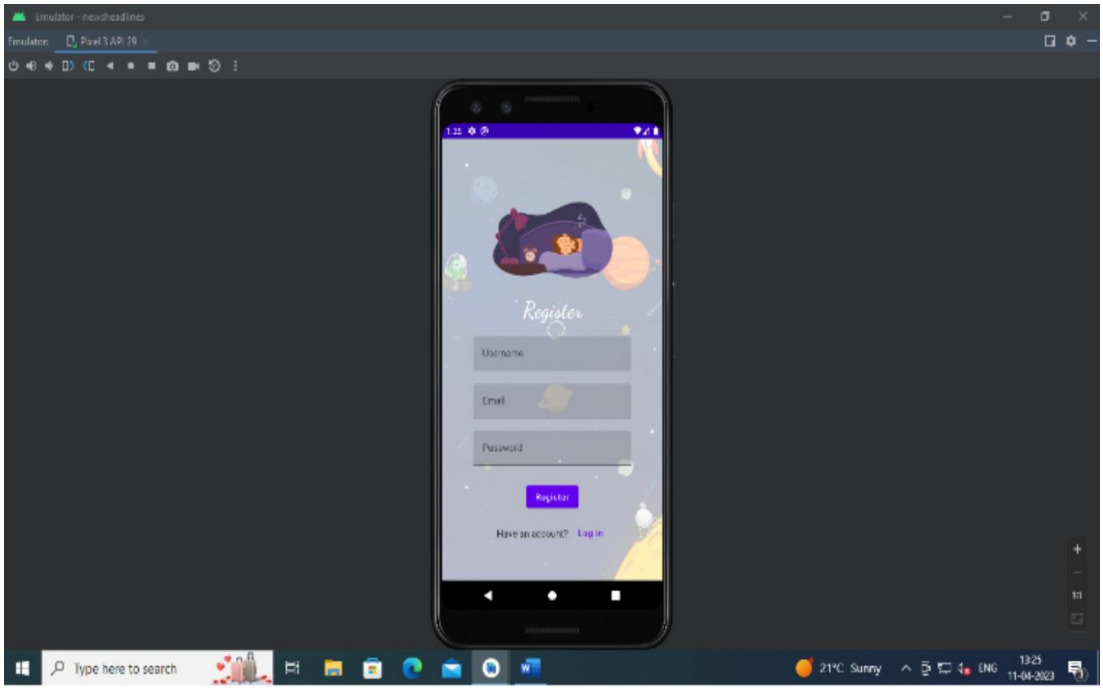


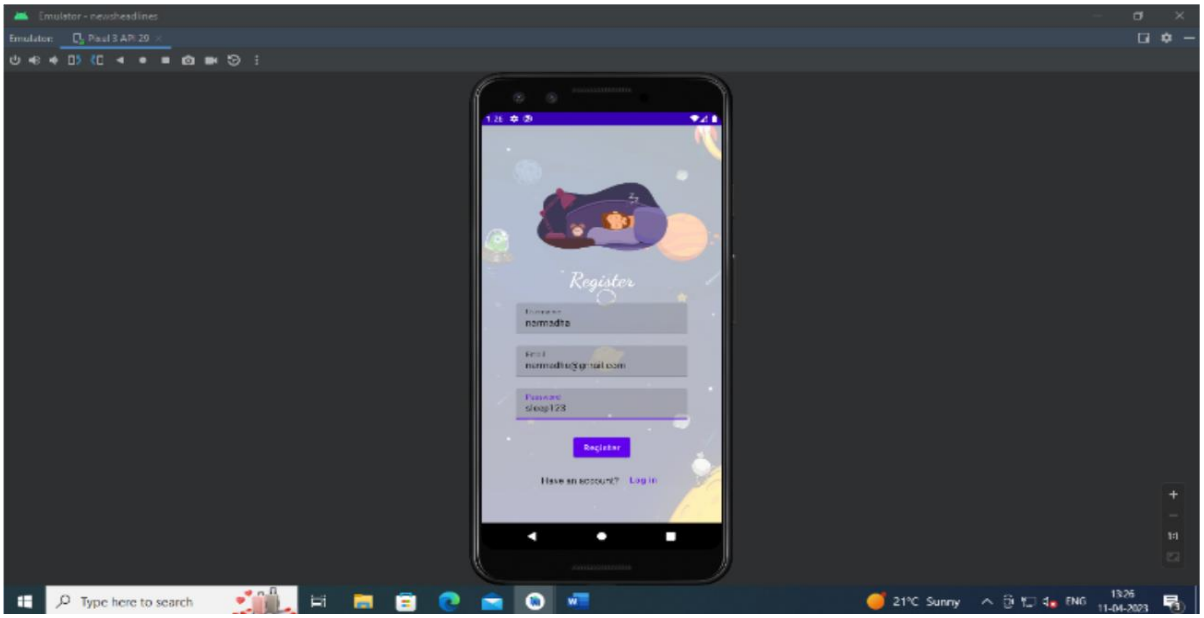
PRIORITIZE:

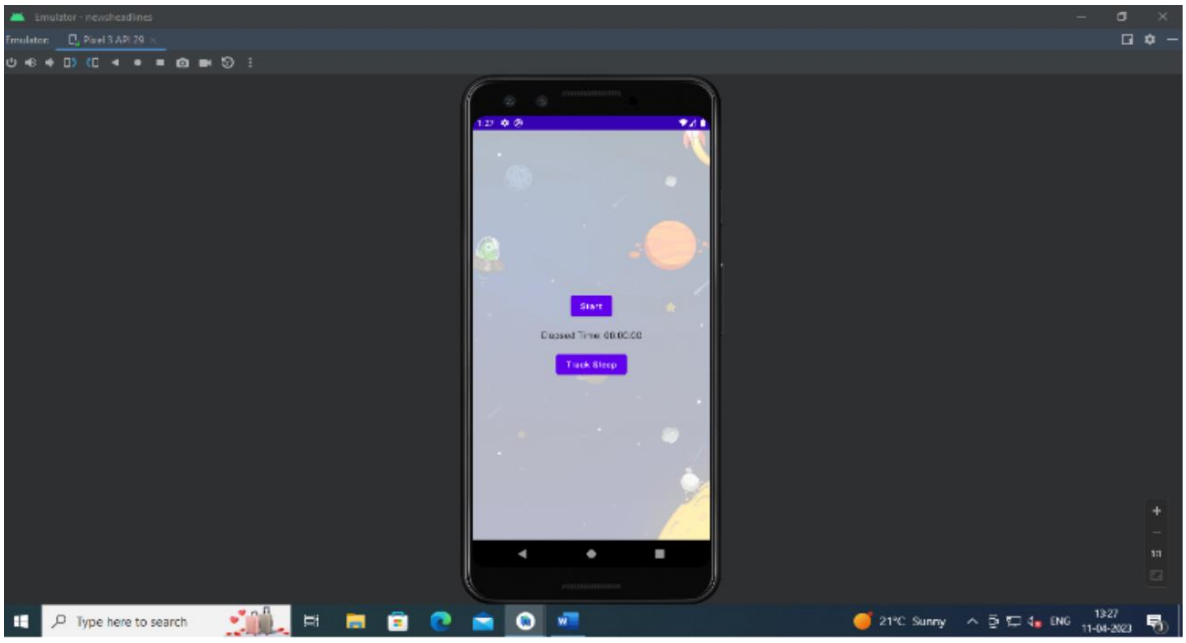

The benefits of sleep tracking are as flexible as your journey through life.
Use your sleep tracker for total sleep time and sleep reading goes.
Importance
Use your data to start a conversation with your doctor.

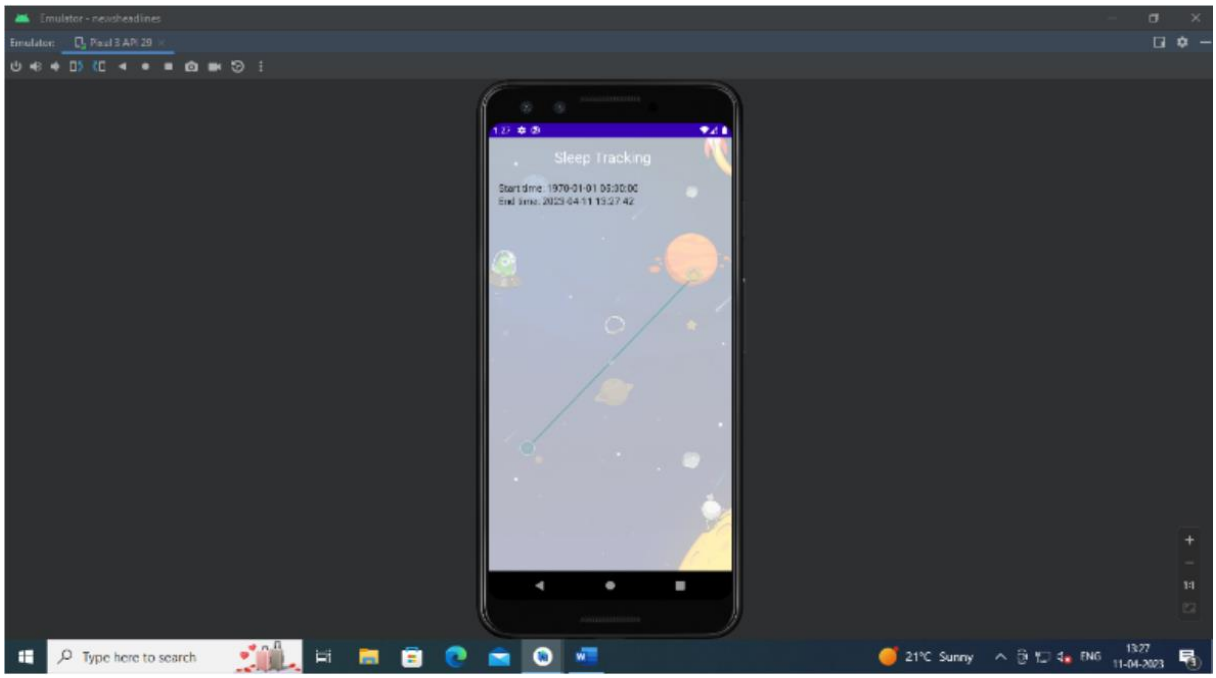
RESULT


Login
Username
Password
Login
Sign up
Forget password?
Type here to search
21°C Sunny
ENG
13:25
11-04-2023


Register
Username
Email
Password
Register
Have an account? Log in
Type here to search
21°C Sunny
ENG
13:25
11-04-2023


Register
Register
Have an account? Log in
Type here to search
21°C Sunny
ENG
13:26
11-04-2023


Start
Track Sleep
Type here to search
21°C Sunny
ENG
13:27
11-04-2023


Emulator - newsheadlines
Emulator:
Pixel 3 API 29
Sleep Tracking
Start time: 1970-01-01 05:30:00
End time: 2023-04-11 13:27:42
Type here to search
21°C Sunny
ENG
13:27
11-04-2023

ADVANTAGES & DISADVANTAGES

ADVANTAGES:

You can manage what you measure:

Dr. Kelly Baron, director of the University of Utah's behavioral sleep medicine program, claims this is the biggest benefit to monitoring your sleep. The idea is simple: If you can detect patterns in how your night time behaviors impact your sleep quality and duration, you can make changes to help you sleep better.

Sleep trackers may help with bedtime routines:

Dr. Conor Heneghan, lead research scientist at Fitbit, argues that tracking your sleep makes you more conscious of when you go to bed and wake up each day. As a result, it may help you get the seven to nine hours of sleep you need.

Sleep trackers may help detect sleep disorders:

Advanced sensors that measure your blood oxygen can flag the possibility that you have sleep apnea. However, a sleep study prescribed by your doctor will be required for an official diagnosis.

Sleep trackers may help you wake up:

Some sleep trackers, which claim to differentiate between light and deep sleep stages, have special alarms that wake you up when you're closest to being awake. The theory is that you'll find it easier to wake up and be in a better mood as a result.

DISADVANTAGES:

Sleep trackers introduce poor sleep hygiene:

When it comes to proper sleep hygiene, viewing devices immediately before or after sleep is a no-no. A sleep tracker may incentivize people to break this rule, argues Dr. Baron and colleagues.

Sleep trackers may be inaccurate:

Sleep trackers may suggest you get more sleep than you do in reality. This is especially true for trackers that rely on movement sensors, as you could simply be lying awake at night. They're also unable to accurately distinguish between light and deep sleep. “Even medically validated diagnostic technologies are at risk for false positive and false negative results," points out Massachusetts General Hospital's Kathryn Russo and colleagues.

Sleep trackers can worsen insomnia:

Across three case studies, patients spent an excessive amount of time in bed trying to maximize their sleep duration. Unfortunately, that's known to exacerbate insomnia.

Sleep trackers make some people resistant to treatment:

In the same study, patients trusted data from their wearable devices over results from official sleep studies. They also neglected evidence-based treatments for insomnia, instead preferring to go it alone.

Sleep trackers are tied to a sleep disorder:

Some people who wear sleep trackers become preoccupied with optimizing their sleep data. This condition, known as orthosomnia, enhances your night time anxiety, which can make it harder for you to sleep.

APPLICATIONS

- **Sleep duration**: Apps can record and keep track of when we fall asleep and when we wake up.

- **Sleep quality**: Apps can detect when your sleep is interrupted or when you wake up in the night.

- **Sleep stages**: Some apps also have the ability to track the stages of sleep and even allow you to set an alarm to go off at a specific stage when you're not in a deep sleep state, making it easier to get up and feel rested.

- **Outside factors**: Sleep apps can record outside factors like the temperature or the amount of light in your bedroom.

- **Lifestyle factors**: Apps allow users to enter information about themselves which can affect their sleep, like how much caffeine they've consumed, stress levels, when they last ate, and so on.

CONCLUSION

The impact that sleep has on human health is undeniable. Recent advances in sensing technology, big data analytics and AI allow for truly ubiquitous and unobtrusive monitoring of sleep and circadian rhythms. However, challenges remain to realisation of the benefits of this monitoring for individuals, research and clinican. Here, we introduced the Digital Sleep Framework, a framework outlining the steps required from the multi-modal acquisition of sleep-related data through to its clinical and commercial application and exploring all aspects of this chain. As the number and scope of sleep monitoring technologies continues to grow and the diversity of digital sleep solutions and applications continues to multiply, the need for careful, risk-based product validation has become increasingly important. The heterogeneity of sensors used for the monitoring of sleep–wake cycles and circadian rhythms poses a unique set of challenges for modelling and interpretability. Hence, the identification and standardisation of robust, reproducible digital sleep biomarkers is of paramount importance. Modelling based on these signals must be as free as possible from conscious and unconscious bias and the development of algorithms must be transparent and readily available for all stakeholders.

FUTURE SCOPE

In recent years, there has been a significant expansion in the development and use of multi-modal sensors and technologies to monitor physical activity, sleep and circadian rhythms. These developments make accurate sleep monitoring at scale a possibility for the first time. Vast amounts of multi-sensor data are being generated with potential applications ranging from large-scale epidemiological research linking sleep patterns to disease, to wellness applications, including the sleep coaching of individuals with chronic conditions. However, in order to realise the full potential of these technologies for individuals, medicine and research, several significant challenges must be overcome. There are important outstanding questions regarding performance evaluation, as well as data storage, curation, processing, integration, modelling and interpretation. Here, we leverage expertise across neuroscience, clinical medicine, bioengineering, electrical engineering, epidemiology, computer science, mHealth and human–computer interaction to discuss the digitisation of sleep from a inter-disciplinary perspective. We introduce the state-of- the-art in sleep-monitoring technologies, and discuss the opportunities and challenges from data acquisition to the eventual application of insights in clinical and consumer settings. Further, we explore the strengths and limitations of current and emerging sensing methods with a particular focus on novel data-driven technologies, such as Artificial Intelligence.

GRADLE SCRIPTS>BUILD.GRADLE(MODULE:APP)

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    namespace 'com.example.projectone'
    compileSdk 33

    defaultConfig {
        applicationId "com.example.projectone"
        minSdk 21
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary true
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion '1.2.0'
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.8.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    implementation "androidx.compose.ui:ui:$compose_ui_version"
    implementation "androidx.compose.ui:ui-tooling-
preview:$compose_ui_version"
    implementation 'androidx.compose.material:material:1.2.0'
```

```
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    implementation 'androidx.room:room-common:2.5.0'

    implementation 'androidx.room:room-ktx:2.5.0'
    androidTestImplementation "androidx.compose.ui:ui-test-
junit4:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-
tooling:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-test-
manifest:$compose_ui_version"
}
```

CREATING THE DATABASE CLASSES:

DATABASE1(CREATE USER DATA CLASS)

```
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?,)
```

CREATE AN USERDAO INTERFACE

```
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)
}
```

CREATE AN USERDATABASE CLASS

```
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
            }
        }
    }
}
```

CREATE AN USERDATABASEHELPER CLASS

```kotlin
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 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
    }

}
```

DATABASE2:

CREATE TIMELOG DATA CLASS

```
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)
```

CREATE AN TIMELOGDAO INTERFACE

```
package com.example.projectone
import androidx.room.Dao
import androidx.room.Insert

@Dao
interface TimeLogDao {
    @Insert
    suspend fun insert(timeLog: TimeLog)
}
```

CREATE AN APPDATABASE CLASS

```
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
            }
        }
    }
}
```

CREATE AN TIMEDATABASEHELPER CLASS

```
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"
        }
    }
}
```

BUILDING APPLICATION UI AND CONNECTING TO DATABASE:

CREATING LOGINACTIVITY.KT WITH DATABASE

```
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.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)
}
```

CREATING REGISTRATIONACTIVITY.KT WITH DATABASE

```
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)
}
```

CREATING MAINACTIVITY.KT FILE

```
package com.example.projectone

import android.content.Context
import android.content.Intent
import android.icu.text.SimpleDateFormat
import android.os.Build
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.annotation.RequiresApi
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.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)
}
@RequiresApi(Build.VERSION_CODES.N)
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)
}
```

CREATING TRACKACTIVITY.KT FILE

```
package com.example.projectone

import android.icu.text.SimpleDateFormat
import android.os.Build
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.annotation.RequiresApi
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.Theme
import java.util.*

class TrackActivity : ComponentActivity() {

    private lateinit var databaseHelper: TimeLogDatabaseHelper

    @RequiresApi(Build.VERSION_CODES.N)
    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)
                }
            }
        }
    }
}


@RequiresApi(Build.VERSION_CODES.N)
@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) }}")
                    }
                }
            }
        }

    }
}

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

MODIFYING ANDROIDMANIFEST.XML:

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.ProjectOne"
        tools:targetApi="31">
        <activity
            android:name=".Type"
            android:exported="false"
            android:label="@string/title_activity_type"
            android:theme="@style/Theme.ProjectOne" />
        <activity
            android:name=".Shape"
            android:exported="false"
            android:label="@string/title_activity_shape"
            android:theme="@style/Theme.ProjectOne" />
        <activity
            android:name=".Color"
            android:exported="false"
            android:label="@string/title_activity_color"
            android:theme="@style/Theme.ProjectOne" />
        <activity
            android:name=".Theme"
            android:exported="false"
            android:label="@string/title_activity_theme"
            android:theme="@style/Theme.ProjectOne" />
        <activity
            android:name=".TrackActivity"
            android:exported="false"
            android:label="@string/title_activity_track"
            android:theme="@style/Theme.ProjectOne" />
        <activity
            android:name=".MainActivity"
            android:exported="false"
            android:label="@string/app_name"
            android:theme="@style/Theme.ProjectOne" />
        <activity
            android:name=".MainActivity2"
            android:exported="false"
            android:label="RegisterActivity"
            android:theme="@style/Theme.ProjectOne" />
        <activity
            android:name=".LoginActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.ProjectOne">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
```

```
        </activity>
    </application>

</manifest>
```