A Sleep Tracking App for a Better Night's Rest

1.Introduction

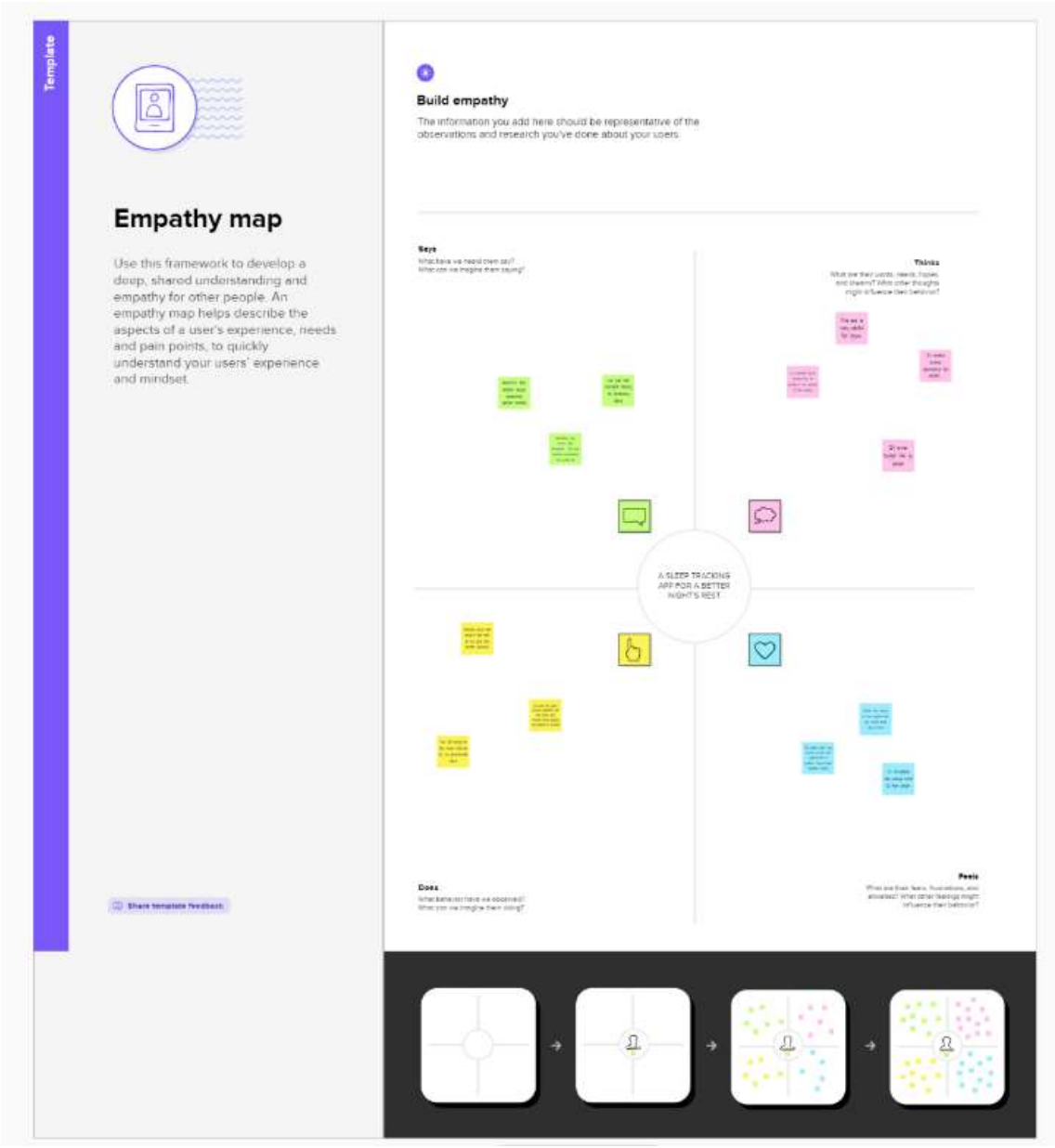
1.1 Overview

This application will be track the peoples sleeping time with using timer on this application. And showing their sleeping activities like histories in sleep tracking page. This application will identify the user authentication details with help of backend database.

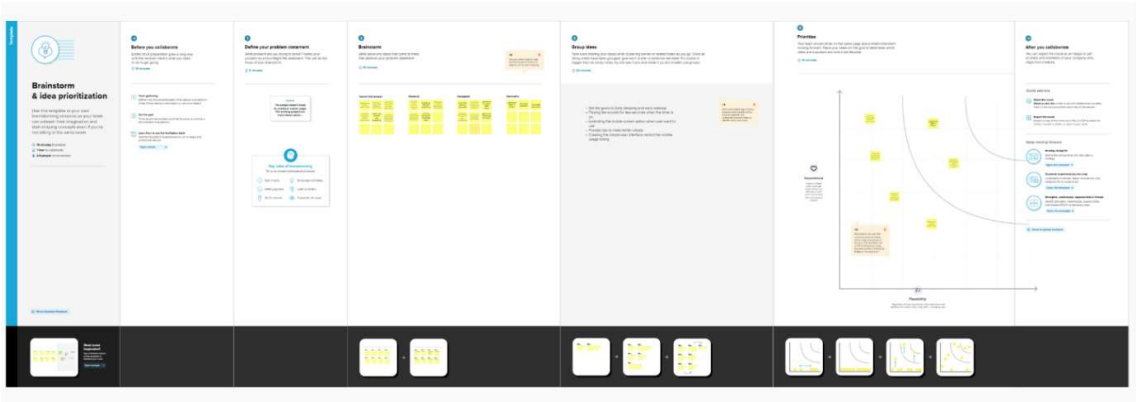
1.2 Purpose

The peoples are does not sleep in correct timing its affecting their health. The people doesn’t care their health. The human body will recover at sleeping time. But the peoples avoiding this, they don’t sleep properly. So this application will calculating the peoples sleeping time with help of timer. The application will be calculating the sleeping time and its stored been like history. This is helps to seen over all sleep time to the people. It was more helpful to they know about their health. And its helps to improve their physical health.

2. Problem Definition & Design Thinking : 2.1 EMPATHY MAP

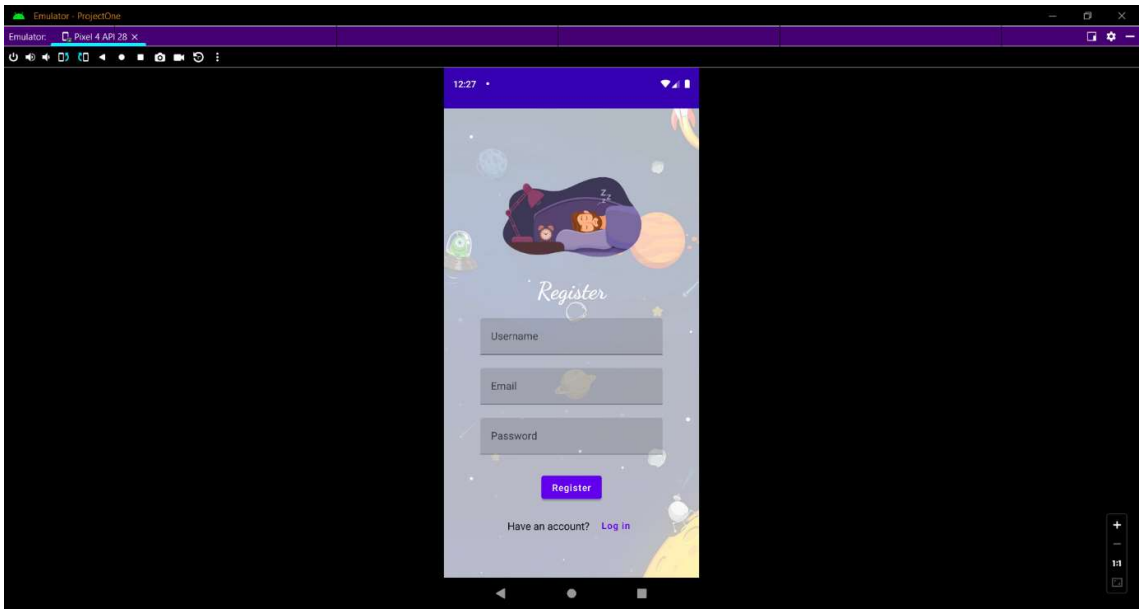


2.2 Ideation & Brainstorming Map :

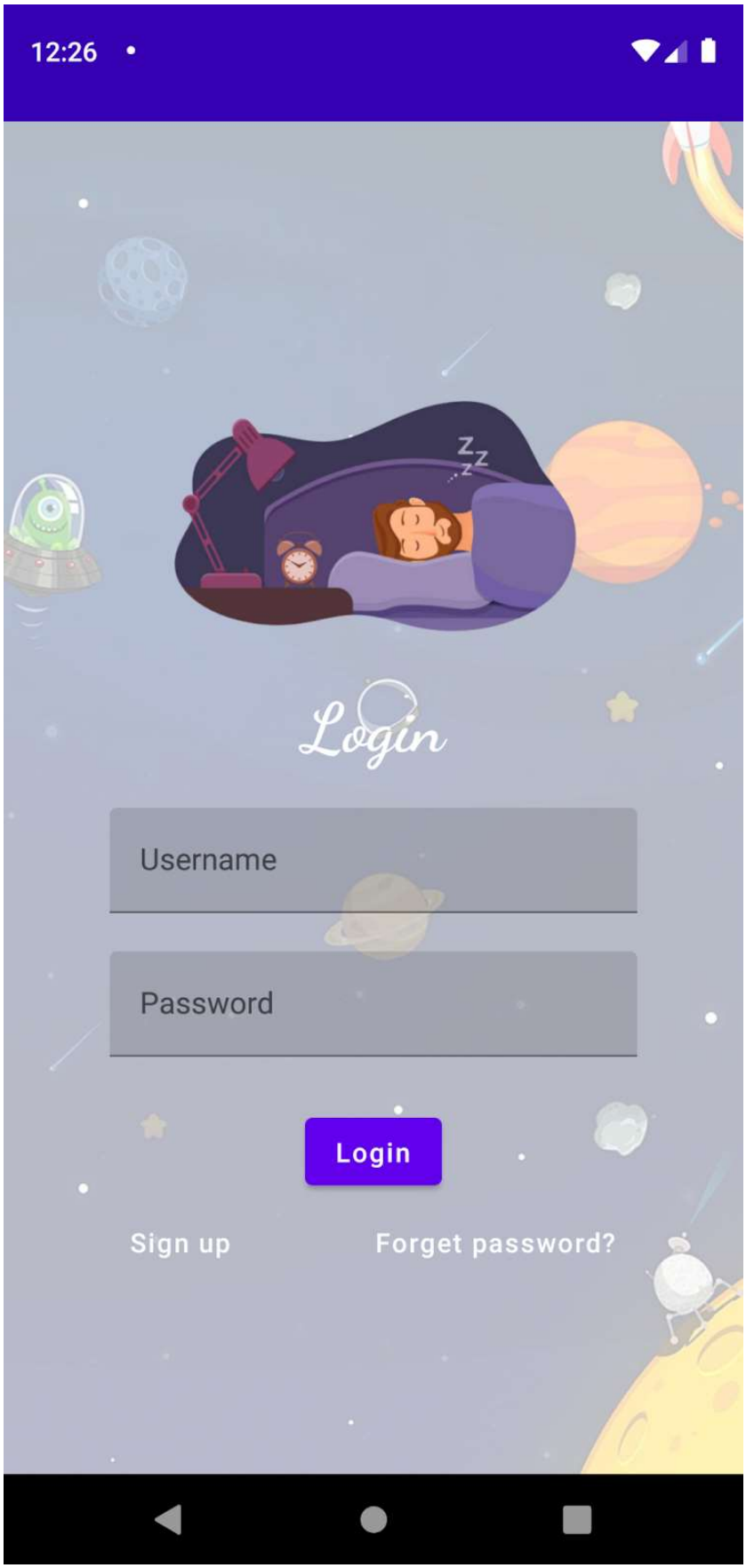


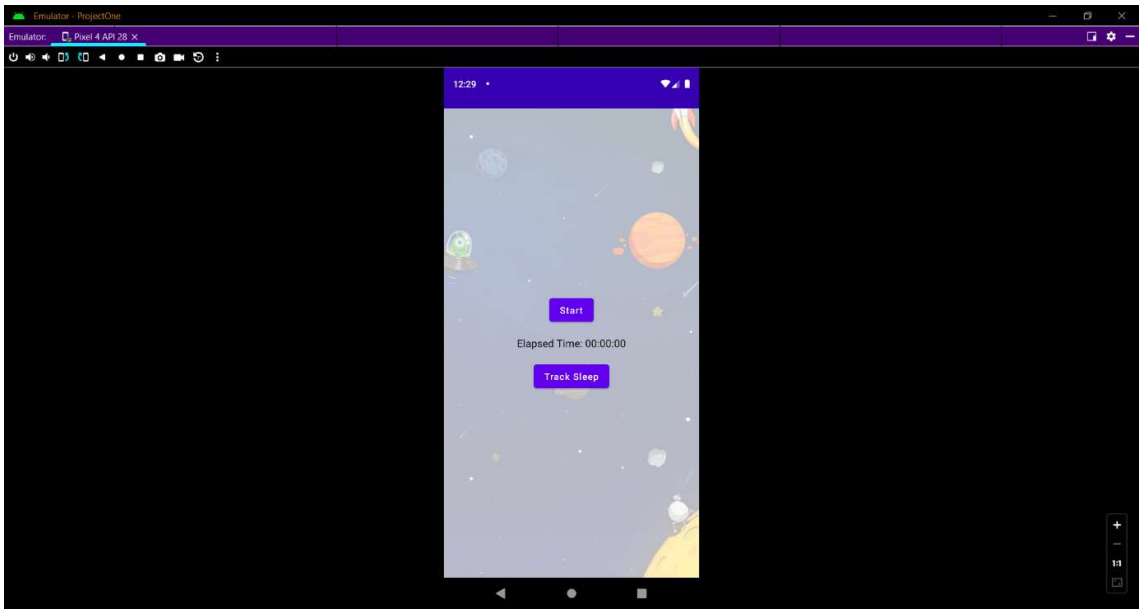
3. RESULT

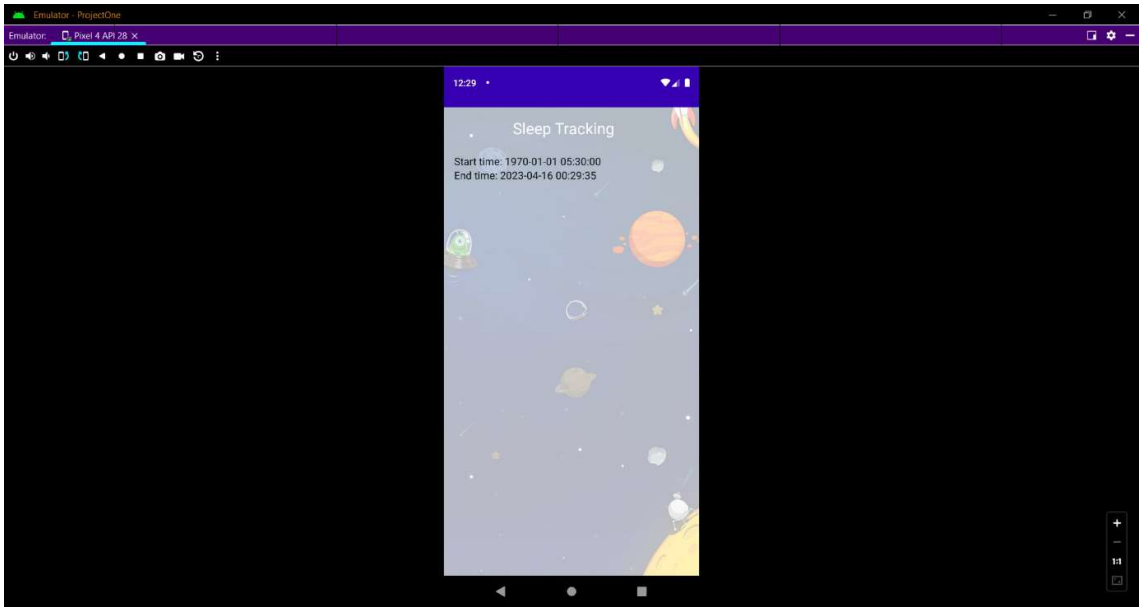
Register Page:



Login page


12:26
Login
Username
Password
Login
Sign up
Forget password?


Emulator - ProjectOne
Emulator: Pixel 4 API 28
12:29
Start
Elapsed Time: 00:00:00
Track Sleep


Emulator - ProjectOne
Emulator: Pixel 4 API 28
12:29
Sleep Tracking
Start time: 1970-01-01 05:30:00
End time: 2023-04-16 00:29:35

4. ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

1. Provides insights: Sleep tracking apps can provide detailed information about your sleep patterns, including the time you fall asleep, the duration and quality of your sleep, and any disturbances that occur during the night. This information can help you identify any patterns or behaviors that may be affecting your sleep, so you can make adjustments accordingly.
2. Promotes healthy sleep habits: Many sleep tracking apps come with features such as guided meditation, white noise, and other relaxation techniques that can help you fall asleep faster and stay asleep longer.
3. Increases accountability: Using a sleep tracking app can help you stay accountable for your sleep habits. By tracking your progress and seeing your results over time, you may be more motivated to stick to healthy sleep habits.

DISADVANTAGES:

1. Inaccuracy: Sleep tracking apps may not always be accurate in tracking sleep stages, especially if they rely on motion sensors or algorithms to determine sleep patterns. This can lead to false or misleading data.
2. Distraction: Using a sleep tracking app can be a distraction, especially if you are constantly checking your phone or device to see your sleep data. This can interfere with your ability to relax and fall asleep.
3. Privacy concerns: Sleep tracking apps may collect sensitive data about your sleep patterns and habits, which could be used for marketing or other purposes. It is important to read the privacy policy of any sleep tracking app before using it, to ensure that your data is protected.

5.APPLICATIONS:

1. Personal Use: Individuals can use sleep tracking apps on their smartphones or other devices to monitor their sleep patterns and improve their sleep quality.
2. Clinical Settings: Sleep tracking apps can also be used in clinical settings to monitor patients with sleep disorders such as insomnia or sleep apnea. By collecting data on patients' sleep patterns, doctors and sleep specialists can diagnose and treat these disorders more effectively.

3. Workplace Wellness Programs: Employers can also incorporate sleep tracking apps into workplace wellness programs to encourage employees to prioritize their sleep health. By tracking their sleep patterns, employees can identify any issues that may be affecting their sleep and make changes to improve their overall health and productivity.
4. Fitness and Wellness Apps: Sleep tracking apps can also be integrated into fitness and wellness apps to provide a more comprehensive view of overall health and wellbeing. By tracking sleep patterns along with exercise, diet, and other lifestyle factors, individuals can get a better understanding of how these factors interact and affect their health.

6.CONCLUSION:

This work discussed the advantages and disadvantages of using a sleep tracking app for a better night's rest. The advantages of using a sleep tracking app include providing insights into sleep patterns, promoting healthy sleep habits, and increasing accountability for sleep habits. However, the disadvantages of using a sleep tracking app include inaccuracy in tracking sleep stages, potential distractions, and privacy concerns with the data collected.

Additionally, this work highlighted the various contexts where a sleep tracking app can be applied, such as personal use, clinical settings, workplace wellness programs, and fitness and wellness apps. Overall, a sleep tracking app can be a useful tool for individuals to monitor and improve their sleep quality, but it is important to consider the potential drawbacks and ensure that any app used prioritizes user privacy and accuracy.

7.FUTURE SCOPE

1. Improved accuracy: Advancements in sensor technology and machine learning algorithms could lead to more accurate tracking of sleep stages and patterns, reducing the potential for false or misleading data.
2. Personalized recommendations: As sleep tracking apps collect more data on individual sleep patterns and habits, they could provide personalized recommendations for improving sleep quality, such as adjusting bedtime routines or changing environmental factors like temperature and noise.
3. Integration with other health data: Sleep tracking apps could be integrated with other health data, such as heart rate and activity level, to provide a more comprehensive view of overall health and wellness.
4. Sleep coaching: Sleep tracking apps could also incorporate coaching features, such as guided meditations or sleep hygiene tips, to help individuals improve their sleep habits and overall health.
5. Wearable technology: Sleep tracking apps could be integrated with wearable technology such as smartwatches or fitness trackers, allowing for continuous monitoring of sleep patterns and real-time feedback on sleep quality.

APPENDIX

```
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)
}


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)
}



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)
}


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"
    }
```

```
    }
}

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"
        }
    }
}



package com.example.projectone

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

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

}
```

```
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))
}
```

```
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?,

    )


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)
}


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
            }
        }
    }
}


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
    }

}
```