ДОДАТОК А

Лістинг класу Database.cs

```
using Microsoft.Data.Sqlite;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Documents;
using TrainingTrackingActiv.Models;

namespace TrainingTrackingActiv
{
   public static class Database
   {
      private const string DbFile = "database.db";
      private static string ConnectionString;

      static Database()
      {
         //створення строки підключення до БД
         var builder = new SqliteConnectionStringBuilder
         {
            DataSource = DbFile,
            Mode = SqliteOpenMode.ReadWriteCreate,
            Cache = SqliteCacheMode.Shared
         };
         ConnectionString = builder.ToString();
      }
      //створення таблиць
      public static void Inizialize()
      {
         using var connection = new SqliteConnection(ConnectionString);
         connection.Open();
         using (var pragma = connection.CreateCommand())
         {
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
         }
         var command = connection.CreateCommand();
         //Таблиця для тренування МН
         command.CommandText = @"
            CREATE TABLE IF NOT EXISTS TrainingData (
               Id INTEGER PRIMARY KEY AUTOINCREMENT,
               TimeStamp TEXT NOT NULL,
               ProcessName TEXT NOT NULL,
```

```
        WindowsTitle TEXTNOT NULL,
        UNIQUE(ProcessName, WindowsTitle)
    )";
command.ExecuteNonQuery();

// 1) Сесії активності
command.CommandText = @"
    CREATE TABLE IF NOT EXISTS ActivitySessions (
        Id        INTEGER PRIMARY KEY AUTOINCREMENT,
        StartTime TEXT    NOT NULL,
        EndTime   TEXT    NOT NULL,
        TotalTime INTEGER NOT NULL,
        IsActive  INTEGER NOT NULL
    );";
command.ExecuteNonQuery();

// 2) Детальні логи активності
command.CommandText = @"
    CREATE TABLE IF NOT EXISTS ActivityLogs (
        Id          INTEGER PRIMARY KEY AUTOINCREMENT,
        SessionId   INTEGER NOT NULL,
        TimeStamp   TEXT    NOT NULL,
        AppName     TEXT,
        WindowTitle TEXT,
        Duration    INTEGER,
        InputEvents INTEGER,
        ActivityType INTEGER,
        FOREIGN KEY(SessionId) REFERENCES ActivitySessions(Id)
    );";
command.ExecuteNonQuery();

// 3) Таблиця задач
command.CommandText = @"
    CREATE TABLE IF NOT EXISTS Tasks (
        Id        INTEGER PRIMARY KEY AUTOINCREMENT,
        Title     TEXT    NOT NULL,
        Priority  INTEGER NOT NULL,
        CreatedAt TEXT    NOT NULL
    );";
command.ExecuteNonQuery();
// 4) Згруповані щоденні статистики для графіків
command.CommandText = @"
    CREATE TABLE IF NOT EXISTS AggregatedStats (
        Id             INTEGER PRIMARY KEY AUTOINCREMENT,
        Date           TEXT    NOT NULL,  -- дата у форматі YYYY-MM-DD
        ActiveTime     INTEGER NOT NULL,  -- сума всіх активних сесій (сек)
        BreakTime      INTEGER NOT NULL,  -- сума неактивних періодів (сек)
        EffectiveWork  INTEGER NOT NULL,  -- сума продуктивних подій (сек)
        TaskCount      INTEGER NOT NULL,  -- кількість виконаних задач
        ProductivityScore REAL  NOT NULL  -- EffectiveWork / ActiveTime
    );";
command.ExecuteNonQuery();
```

```
    }
    //додання даних у таблицю тренування моделі
    public static void InsertTrainingData(IEnumerable<(string processName, string windowTitle)>
batch)
    {
        using var connection = new SqliteConnection(ConnectionString);
        connection.Open(); // Open the connection
        using (var pragma = connection.CreateCommand())
        {
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
        }
        using var tx = connection.BeginTransaction();
        var command = connection.CreateCommand();
        command.CommandText = @"
          INSERT OR IGNORE INTO TrainingData (TimeStamp, ProcessName, WindowsTitle)
          VALUES ($timeStamp, $processName, $windowTitle)";

        var pTs = command.CreateParameter();
        pTs.ParameterName = "$timeStamp";
        command.Parameters.Add(pTs);

        var pProcessName = command.CreateParameter();
        pProcessName.ParameterName = "$processName";
        command.Parameters.Add(pProcessName);

        var pWindowTitle = command.CreateParameter();
        pWindowTitle.ParameterName = "$windowTitle";
        command.Parameters.Add(pWindowTitle);

        foreach (var (processName, windowTitle) in batch)
        {
            pTs.Value = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
            pProcessName.Value = processName ?? "";
            pWindowTitle.Value = windowTitle ?? "";
            command.ExecuteNonQuery();
        }

        tx.Commit();
    }
    //додавання даних у розширену таблицю активних сесій
    public static int StartActivitySession(DateTime startTime, bool isActive)
    {
        if (string.IsNullOrEmpty(ConnectionString))
            throw new InvalidOperationException("ConnectionString не заданий.");

        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();

        using (var pragma = conn.CreateCommand())
        {
```

```
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
        }

        using var command = conn.CreateCommand();
        command.CommandText = @"
          INSERT INTO ActivitySessions (StartTime, EndTime, TotalTime, IsActive)
          VALUES ($s, $e, $t, $a);";
        command.Parameters.AddWithValue("$s",                startTime.ToString("yyyy-MM-dd
HH:mm:ss"));
        command.Parameters.AddWithValue("$e",                startTime.ToString("yyyy-MM-dd
HH:mm:ss"));
        command.Parameters.AddWithValue("$t", 0);
        command.Parameters.AddWithValue("$a", isActive ? 1 : 0);
        command.ExecuteNonQuery();

        command.CommandText = "SELECT last_insert_rowid();";
        return Convert.ToInt32(command.ExecuteScalar());
    }
    public static void CloseActivitySession(int sessionId, DateTime endTime)
    {
        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();

        using var command = conn.CreateCommand();
        command.CommandText = @"
            UPDATE ActivitySessions
            SET EndTime = $e, TotalTime = CAST((julianday($e) - julianday(StartTime))*86400 AS
INTEGER)
            WHERE Id = $id;";
        command.Parameters.AddWithValue("$e", endTime.ToString("yyyy-MM-dd HH:mm:ss"));
        command.Parameters.AddWithValue("$t", (int)(endTime - DateTime.Now).TotalSeconds);
        command.Parameters.AddWithValue("$id", sessionId);
        command.ExecuteNonQuery();
    }
    public static void InsertActivityLogsBatch(IEnumerable<ActivityLogs> logs)
    {
        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();
        using (var pragma = conn.CreateCommand())
        {
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
        }

        using var tx = conn.BeginTransaction();
        var command = conn.CreateCommand();

        command.CommandText = @"
        INSERT INTO ActivityLogs
            (SessionId, TimeStamp, AppName, WindowTitle, Duration, InputEvents, ActivityType)
        VALUES
```

```
        ($sid, $ts, $app, $win, $dur, $evt, $type);";

    var pSid = command.CreateParameter();
    pSid.ParameterName = "$sid";
    command.Parameters.Add(pSid);

    var pTs = command.CreateParameter();
    pTs.ParameterName = "$ts";
    command.Parameters.Add(pTs);

    var pApp = command.CreateParameter();
    pApp.ParameterName = "$app";
    command.Parameters.Add(pApp);

    var pWin = command.CreateParameter();
    pWin.ParameterName = "$win";
    command.Parameters.Add(pWin);

    var pDur = command.CreateParameter();
    pDur.ParameterName = "$dur";
    command.Parameters.Add(pDur);

    var pEvt = command.CreateParameter();
    pEvt.ParameterName = "$evt";
    command.Parameters.Add(pEvt);

    var pType = command.CreateParameter();
    pType.ParameterName = "$type";
    command.Parameters.Add(pType);

    foreach (var log in logs)
    {
        pSid.Value = log.SessionId;
        pTs.Value = log.TimeStamp.ToString("yyyy-MM-dd HH:mm:ss");
        pApp.Value = log.AppName ?? "";
        pWin.Value = log.WindowsTitle ?? "";
        pDur.Value = log.Duration;
        pEvt.Value = log.IntputEvent;
        pType.Value = log.ActivityType;
        command.ExecuteNonQuery();
    }
    tx.Commit();
}
public static int InsertTask(string title, int priority, string description, DateTime createdAt)
{
    using var conn = new SqliteConnection(ConnectionString);
    conn.Open();
    using (var pragma = conn.CreateCommand())
    {
        pragma.CommandText = "PRAGMA foreign_keys = ON;";
        pragma.ExecuteNonQuery();
    }
```

```
        using var command = conn.CreateCommand();
        command.CommandText = @"
            INSERT INTO Tasks (Title, Priority, Description, CreatedAt)
            VALUES ($title, $priority, $description, $createdAt);";
        command.Parameters.AddWithValue("$title", title);
        command.Parameters.AddWithValue("$priority", priority);
        command.Parameters.AddWithValue("$description", description ?? string.Empty);
        command.Parameters.AddWithValue("$createdAt",        createdAt.ToString("yyyy-MM-dd
HH:mm:ss"));
        command.ExecuteNonQuery();

        command.CommandText = "SELECT last_insert_rowid();";
        return Convert.ToInt32(command.ExecuteScalar());
    }
    public static void UpdateTaskEvaluation(int taskId, int status, string comment, DateTime
evaluatedAt)
    {
        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();
        using (var pragma = conn.CreateCommand())
        {
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
        }
        using var command = conn.CreateCommand();
        command.CommandText = @"
            UPDATE Tasks
     SET Status      = @status,
         LastComment = @comment,
         EvaluatedAt = @evalAt
    WHERE Id = @id;
    ";
        command.Parameters.AddWithValue("@id", taskId);
        command.Parameters.AddWithValue("@status", status);
        command.Parameters.AddWithValue("@comment", comment ?? string.Empty);
        command.Parameters.AddWithValue("@evalAt",        evaluatedAt.ToString("yyyy-MM-dd
HH:mm:ss"));
        command.ExecuteNonQuery();
    }
    public static List<TaskRecord> GetAllTasks()
    {
        var list = new List<TaskRecord>();
        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();

        using (var pragma = conn.CreateCommand())
        {
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
        }
```

```
        using var command = conn.CreateCommand();
        command.CommandText = @"
       SELECT Id, Title, Priority, CreatedAt, Status, LastComment, EvaluatedAt, Description
        FROM Tasks
       ORDER BY CreatedAt DESC;
    ";
        using var reader = command.ExecuteReader();
        while (reader.Read())
        {
            list.Add(new TaskRecord
            {
                Id = reader.GetInt32(0),
                Title = reader.GetString(1),
                Priority = reader.GetInt32(2),
                CreatedAt = DateTime.Parse(reader.GetString(3)),
                Status = reader.GetInt32(4),
                LastComment = reader.IsDBNull(5) ? null : reader.GetString(5),
                EvaluatedAt = reader.IsDBNull(6) ? (DateTime?)null :
DateTime.Parse(reader.GetString(6)),
                Description = reader.IsDBNull(7) ? string.Empty : reader.GetString(7)
            });

        }
        return list;
    }
    public static void UpsertAggregatedStats(AggregatedStatsRecord s)
    {
        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();
        using (var pragma = conn.CreateCommand())
        {
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
        }

        int? existingID = null;
        using (var find = conn.CreateCommand())
        {
            find.CommandText = @"
                SELECT Id
                   FROM AggregatedStats
                WHERE Date = $d
                LIMIT 1;";
            find.Parameters.AddWithValue("$d", s.Date.ToString("yyyy-MM-dd"));
            var reader = find.ExecuteScalar();
            if (reader != null && reader != DBNull.Value)
            {
                existingID = Convert.ToInt32(reader);
            }
        }
        if (existingID.HasValue)
        {
```

```
        using var update = conn.CreateCommand();
        update.CommandText = @"
  UPDATE AggregatedStats
     SET ActiveTime       = $a,
         BreakTime        = $b,
         EffectiveWork    = $e,
         TaskCount        = $t,
         ProductivityScore = $p
   WHERE Id = $id;";

        update.Parameters.AddWithValue("$a", s.ActiveTime);
        update.Parameters.AddWithValue("$b", s.BreakTime);
        update.Parameters.AddWithValue("$e", s.EffectiveWork);
        update.Parameters.AddWithValue("$t", s.TaskCount);
        update.Parameters.AddWithValue("$p", s.ProductivityScore);
        update.Parameters.AddWithValue("$id", existingID.Value);
        update.ExecuteNonQuery();
    }
    else
    {
        using var insert = conn.CreateCommand();
        insert.CommandText = @"INSERT INTO AggregatedStats
    (Date, ActiveTime, BreakTime, EffectiveWork, TaskCount, ProductivityScore)
  VALUES
    ($d, $a, $b, $e, $t, $p);";
        insert.Parameters.AddWithValue("$d", s.Date.ToString("yyyy-MM-dd"));
        insert.Parameters.AddWithValue("$a", s.ActiveTime);
        insert.Parameters.AddWithValue("$b", s.BreakTime);
        insert.Parameters.AddWithValue("$e", s.EffectiveWork);
        insert.Parameters.AddWithValue("$t", s.TaskCount);
        insert.Parameters.AddWithValue("$p", s.ProductivityScore);
        insert.ExecuteNonQuery();
    }
}
public static AggregatedStatsRecord ComputeAggregatedStats(DateTime date)
{
    var d = date.ToString("yyyy-MM-dd");
    Debug.WriteLine($"[ComputeAggregatedStats] date = {d}");
    const string sql = @"SELECT
COALESCE((SELECT SUM(TotalTime)
            FROM ActivitySessions
           WHERE substr(StartTime,1,10) = $d
             AND IsActive = 1), 0),
COALESCE((SELECT SUM(TotalTime)
            FROM ActivitySessions
           WHERE substr(StartTime,1,10) = $d
             AND IsActive = 0), 0),
COALESCE((SELECT SUM(Duration)
            FROM ActivityLogs
           WHERE substr(TimeStamp,1,10) = $d
             AND ActivityType = 2), 0),
COALESCE((SELECT COUNT(*)
```

```
                FROM TaskEvaluations
               WHERE substr(EvaluatedAt,1,10) = $d
                 AND Status = 2), 0);
    ";
        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();
        using (var pragma = conn.CreateCommand())
        {
           pragma.CommandText = "PRAGMA foreign_keys = ON;";
           pragma.ExecuteNonQuery();
        }
        using var command = conn.CreateCommand();
        command.CommandText = sql;
        command.Parameters.AddWithValue("$d", d);

        Debug.WriteLine("[ComputeAggregatedStats] executing SQL:");
        Debug.WriteLine(sql.Replace("$d", d));

        using var reader = command.ExecuteReader();
        var stats = new AggregatedStatsRecord { Date = date.Date };

        if (!reader.Read())
        {
           Debug.WriteLine("[ComputeAggregatedStats] no rows returned at all.");
        }
        else
        {
           var a = reader.GetInt32(0);
           var b = reader.GetInt32(1);
           var e = reader.GetInt32(2);
           var t = reader.GetInt32(3);
           Debug.WriteLine($"[ComputeAggregatedStats] a = {a}, b = {b}, e = {e}, t = {t}");

           stats.ActiveTime = a;
           stats.BreakTime = b;
           stats.EffectiveWork = e;
           stats.TaskCount = t;
        }

        stats.ProductivityScore = stats.ActiveTime > 0
           ? (double)stats.EffectiveWork / stats.ActiveTime
           : 0.0;
        Debug.WriteLine($"[ComputeAggregatedStats]  ProductivityScore  =
{stats.ProductivityScore}");
        return stats;
     }
     public static List<AggregatedStatsRecord> GetAggregatedStatsRange(DateTime startDate,
DateTime endDate)
     {
        var days = new List<DateTime>();
        for (var d = startDate; d <= endDate; d = d.AddDays(1))
        {
```

```
            days.Add(d);
        }

        var existing = new Dictionary<string, AggregatedStatsRecord>();
        using (var conn = new SqliteConnection(ConnectionString))
        {
            conn.Open();
            using (var pragma = conn.CreateCommand())
            {
                pragma.CommandText = "PRAGMA foreign_keys = ON;";
                pragma.ExecuteNonQuery();
            }
            using var command = conn.CreateCommand();
            command.CommandText = @"
            SELECT Date, ActiveTime, BreakTime, EffectiveWork, TaskCount, ProductivityScore
              FROM AggregatedStats
             WHERE Date >= $start
          AND Date <= $end
        ORDER BY Date;";
            command.Parameters.AddWithValue("$start", startDate.ToString("yyyy-MM-dd"));
            command.Parameters.AddWithValue("$end", endDate.ToString("yyyy-MM-dd"));
            using var reader = command.ExecuteReader();
            while (reader.Read())
            {
                var dateText = reader.GetString(0);
                existing[dateText] = new AggregatedStatsRecord
                {
                    Date = DateTime.ParseExact(dateText, "yyyy-MM-dd",
CultureInfo.InvariantCulture),
                    ActiveTime = reader.GetInt32(1),
                    BreakTime = reader.GetInt32(2),
                    EffectiveWork = reader.GetInt32(3),
                    TaskCount = reader.GetInt32(4),
                    ProductivityScore = reader.GetDouble(5)
                };
            }
        }
        var result = new List<AggregatedStatsRecord>();
        foreach (var d in days)
        {
            var dateText = d.ToString("yyyy-MM-dd");
            if (existing.TryGetValue(dateText, out var stats))
            {
                result.Add(stats);
            }
            else
            {
                result.Add(new AggregatedStatsRecord
                {
                    Date = d,
                    ActiveTime = 0,
                    BreakTime = 0,
```

```
                EffectiveWork = 0,
                TaskCount = 0,
                ProductivityScore = 0.0
            });
        }
    }
    return result;
}
public class ActivityDistribution
{
    public double Productive { get; set; }
    public double Neutral { get; set; }
    public double Unproductive { get; set; }
}
public static ActivityDistribution GetActivityDistribution(DateTime date)
{
    var d = date.ToString("yyyy-MM-dd");
    using var conn = new SqliteConnection(ConnectionString);
    conn.Open();
    using (var pragma = conn.CreateCommand())
    {
        pragma.CommandText = "PRAGMA foreign_keys = ON;";
        pragma.ExecuteNonQuery();
    }
    using var command = conn.CreateCommand();
    command.CommandText = @"
  SELECT
   COALESCE(SUM(CASE WHEN ActivityType = 2 THEN Duration ELSE 0 END), 0),
   COALESCE(SUM(CASE WHEN ActivityType = 1 THEN Duration ELSE 0 END), 0),
   COALESCE(SUM(CASE WHEN ActivityType = 0 THEN Duration ELSE 0 END), 0)
  FROM ActivityLogs
  WHERE substr(TimeStamp,1,10) = $d;
";
    command.Parameters.AddWithValue("$d", d);
    using var reader = command.ExecuteReader();
    if (reader.Read())
    {
        return new ActivityDistribution
        {
            Productive = reader.GetInt32(0),
            Neutral = reader.GetInt32(1),
            Unproductive = reader.GetInt32(2)
        };
    }
    return new ActivityDistribution();
}
public static List<int> GetSessionDuration(DateTime date)
{
    var list = new List<int>();
    var d = date.ToString("yyyy-MM-dd");
    using var conn = new SqliteConnection(ConnectionString);
    conn.Open();
```

```
    using (var pragma = conn.CreateCommand())
    {
        pragma.CommandText = "PRAGMA foreign_keys = ON;";
        pragma.ExecuteNonQuery();
    }
    using var command = conn.CreateCommand();
    command.CommandText = @"
SELECT TotalTime
    FROM ActivitySessions
     WHERE substr(StartTime,1,10) = $d
     ";
    command.Parameters.AddWithValue("$d", d);
    using var reader = command.ExecuteReader();
    while (reader.Read())
    {
        list.Add(reader.GetInt32(0));
    }
    return list;
}
public static List<TopApplicationUsage> GetTopApplicationUsagesTime(int topN = 3)
{
    var result = new List<TopApplicationUsage>();
    using var conn = new SqliteConnection(ConnectionString);
    conn.Open();
    using (var pragma = conn.CreateCommand())
    {
        pragma.CommandText = "PRAGMA foreign_keys = ON;";
        pragma.ExecuteNonQuery();
    }
    using var command = conn.CreateCommand();
    command.CommandText = @"
        SELECT AppName, SUM(Duration) AS TotalSeconds
        FROM ActivityLogs
        GROUP BY AppName
        ORDER BY TotalSeconds DESC
        LIMIT $topN;";
    command.Parameters.AddWithValue("$topN", topN);

    using var reader = command.ExecuteReader();
    while (reader.Read())
    {
        var appName = reader.GetString(0);
        var totalSeconds = reader.GetInt32(1);
        result.Add(new TopApplicationUsage
        {
            ApplicationName = appName,
            TotalSeconds = totalSeconds
        });
    }
    return result;
}
public static TaskSummary GetTaskSummary()
```

```
    {
        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();
        using (var pragma = conn.CreateCommand())
        {
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
        }
        using var command = conn.CreateCommand();
        command.CommandText = @"
            SELECT COUNT(*) AS TotalTasks,
                SUM(CASE WHEN Status = 2 THEN 1 ELSE 0 END) AS CompletedTasks,
                SUM(CASE WHEN Status = 1 THEN 1 ELSE 0 END) AS PartiallyCompletedTasks,
                SUM(CASE WHEN Status = 0 THEN 1 ELSE 0 END) AS RemainingTasks,
                SUM(CASE WHEN date(CreatedAt)  = date('now') THEN 1 ELSE 0 END) AS
AssignedToday,
                SUM(CASE WHEN Status = 2 AND date(EvaluatedAt) = date('now') THEN 1 ELSE
0 END) AS CompletedToday
            FROM Tasks;";
        using var reader = command.ExecuteReader();
        if (!reader.Read()) throw new InvalidOperationException("Не вдалося отримати
статистику.");

        return new TaskSummary
        {
            TotalTasks = reader.GetInt32(0),
            CompletedTasks = reader.GetInt32(1),
            PartiallyCompletedTasks = reader.GetInt32(2),
            RemainingTasks = reader.GetInt32(3),
            AssignedToday = reader.GetInt32(4),
            CompletedToday = reader.GetInt32(5)
        };
    }
    public static double? GetAggregatedProductivity(DateTime date)
    {
        using var conn = new SqliteConnection(ConnectionString);
        conn.Open();
        using (var pragma = conn.CreateCommand())
        {
            pragma.CommandText = "PRAGMA foreign_keys = ON;";
            pragma.ExecuteNonQuery();
        }
        using var command = conn.CreateCommand();
        command.CommandText = @"
            SELECT ProductivityScore
            FROM AggregatedStats
            WHERE Date = $d;";
        command.Parameters.AddWithValue("$d", date.ToString("yyyy-MM-dd"));
        var reader = command.ExecuteReader();
        if (!reader.Read() || reader.IsDBNull(0))
        {
            return null;
```

```
            }
            return reader.GetDouble(0);
        }
        public static Dictionary<int,double> GetUsagePerHour(DateTime date)
        {
            var result = new Dictionary<int, double>();
            using var conn = new SqliteConnection(ConnectionString);
            conn.Open();
            using (var pragma = conn.CreateCommand())
            {
                pragma.CommandText = "PRAGMA foreign_keys = ON;";
                pragma.ExecuteNonQuery();
            }
            using var command = conn.CreateCommand();
            command.CommandText = @"
                SELECT strftime('%H', TimeStamp) AS Hour, SUM(Duration) AS TotalSeconds
                FROM ActivityLogs
                WHERE date(TimeStamp) = $d
                GROUP BY Hour;";
            command.Parameters.AddWithValue("$d", date.ToString("yyyy-MM-dd"));
            using var reader = command.ExecuteReader();
            while (reader.Read())
            {
                int hour = int.Parse(reader.GetString(0));
                double totalSeconds = reader.GetInt32(1);
                result[hour] = totalSeconds;
            }
            return result;
        }
    }
}
```

ДОДАТОК Б

Лістинг класу Tracking.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Runtime.InteropServices;
using System.Diagnostics;

public class Tracking
{
   [DllImport("user32.dll")]
   public static extern IntPtr GetForegroundWindow();

   [DllImport("user32.dll")]
   public static extern int GetWindowText(IntPtr hWnd, StringBuilder lpString, int nMaxCount);

   [DllImport("user32.dll")]
   public static extern int GetWindowTextLength(IntPtr hWnd);

   [DllImport("user32.dll")]
   private static extern uint GetWindowThreadProcessId(IntPtr hWnd, out uint lpdwProcessId);

   public static string GetActiveWindowTitle()
   {
      IntPtr handle = GetForegroundWindow();
      if (handle == IntPtr.Zero)
         return string.Empty;

      int length = GetWindowTextLength(handle);
      if (length <= 0)
         return string.Empty;

      StringBuilder builder = new StringBuilder(length + 1);
      GetWindowText(handle, builder, builder.Capacity);
      return builder.ToString();
   }

   public static string GetActiveWindowProcessName()
   {
      IntPtr handle = GetForegroundWindow();
      if (handle == IntPtr.Zero)
```

```
            return string.Empty;

        GetWindowThreadProcessId(handle, out uint processId);
        try
        {
            Process process = Process.GetProcessById((int)processId);
            return process.ProcessName + ".exe";
        }
        catch
        {
            return null; // Якщо процес вже закритий
        }
    }
}
```

ДОДАТОК В

Демонстраційний матеріал

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 502100.015 ПЗ	Пояснювальна записка	А4, 62 с.
		Додаткові матеріали	
		Лістинг класу Database.cs	А4, 14 с.
		Лістинг класу Tracking.cs	А4, 2 с.
2		Демонстраційний матеріал	15 с.

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК. 502100.015 ВД			
Розроб.		Пристенський О.Р.			Розроблення програмного забезпечення оцінювання виконання робочих задач для раціонального розподілу часу	Літера	Аркуш	Аркушів
Перевір.		Теслюк С.І.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Демська Н.П.						
Затв.		Невлюдов І.Ш.						