

Bahria University,

Karachi Campus



COURSE: SOFTWARE DESIGN AND ARCHITECTURE
TERM: SPRING 2025, CLASS: BSE- 4 (A)

PROJECT NAME

Engr. Ramsha Mashood

Signed

Remarks:

Score:

Semester Project

1. Title Page

- Project Title: CIES (Customer Interaction Evaluation System)
- Student Name(s):
 - Zohaib Arshad Noor (02-131232-066)(**Leader**)
 - Keyan Majid (02-131232-021)
 - Ifrahim Yousuf (02-131232-018)
 - Jawad Ul Hassan (02-131232-069)
- Course Name & Code: Software Design Architecture (SEN-221)
- Instructor's Name: Engr. Ramsha Mashood
- Date: 02-06-2025

2. Introduction

- **Project Background:**

The purpose of this project is to improve customer interaction which in turn brings good business. The employees are evaluated based on their interaction with their customers and those not behaving will get fired immediately.

- **Project Objectives:** The aim is to develop a robust system that evaluates employees based on their interaction with the customers and allows manager to overlook it's all over department's performance based on each employee's customer interaction. The objective is to improve the customer support of the organization specifically by improving conversations between employees and customers.

3. System Overview

- **Key Features:** The Key features are as follows:
 - Keeping track of use of words of each employee of a department
 - Registration of a new employee with the DeviceID as the Employee's identifier indicating the system allotted to the employee.
 - Checking the conversation and comparing it with a predefined list of words named BadWordsList
 - Scoring based on the use of bad words
 - Real-Time update in database of key-logs captured
 - As the keys reach database, automated update in score based on the misuse of words along with the storing of bad words used in the conversation
 - User-Friendly Dashboard for Manager of each department to check for the Employees performance working in that department.
 - Firing of the employees with lowest score and worst conversations impacting business in the end

- Keeping track of Manager's login/logout activity in database
- **User Roles:** There are basically two types of users in the system:
 - User/Manager: using the system to evaluate it's employees
 - Employee: the one whose keys are captured by the keylogger and sent to the database where the logic is applied and is sent to the dashboard, consequently, displaying the Employee's performance.

4. Architectural Style

-> layered, client-server, oneDrive as an API

- **Selected Style:** The selected architectural style for the system are as follows:
 - MVC
 - Microservice
- **Justification:**
 - **MVC:** This one is mainly used for front-end. This gives a layered and easy way to connect interfaces(views) with database(through Model Class) using the Controller Classes
- **Diagram:** (Optional) Insert a simple architecture diagram showing components and interactions



5. Database Design

- **Database Type: SQL.**
- **Schema Overview:** We used around 11 tables. These are as follows:
 - **Employess:** record about employess.
 - DeviceID(Primary Key)
 - ManagerID(Foreign Key)
 - DepartmentID(Foreign Key)
 - EmpiD
 - Address
 - ContactNo
 - **Departments:** record about departments
 - DepartmentID(Foreign Key)
 - DepartmentName
 - **Users:** record about Managers
 - UserID(Primary Key)
 - Name
 - Password
 - Email
 - DepartmentID(ForeignKey)

- **DeviceInfo:** record about devices allotted to the employee
 - DeviceID(Primary Key)
 - DesktopName
 - OSVersion
 - RamSize
 - MacAddress
- **Keyss:** record about keys pressed
 - KeyID (Primary Key)
 - DeviceID (Foreign Key)
 - TypedText
 - PressedAt
- **Scoring:** updated scores at very black listed words used
 - ScoreID (Primary Key)
 - DeviceID(ForeignID)
 - Score
 - LastUpdatedAt
- **KeyStrokeViolation:** stores the violated words used by the employee
 - KeyID(PrimaryKey, ForeignKey)
 - WordID(PrimaryKey, ForeignKey)
- **EmpDataTrail:** keeps track of data inserted, updated and deleted data of Employee Table
 - ID (PrimaryKey)
 - DeviceID (ForeignKey)
 - Action
 - OldValue
 - NewValue
 - Time

- **Sample Table Structure:** Let's take an example of Employess Table:

➔ Employee Table's Field along with it's DataTypes:

LAPTOP-U82D888G....s - dbo.employess			
	Column Name	Data Type	Allow Nulls
	DeviceID	uniqueidentifier	<input type="checkbox"/>
	EmpiName	varchar(255)	<input type="checkbox"/>
	Address	varchar(255)	<input type="checkbox"/>
	ContactNo	varchar(20)	<input type="checkbox"/>
	DepartmentID	int	<input type="checkbox"/>
	ManagerID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Employee Table:

Results		Messages				
	DeviceID	EmpiName	Address	ContactNo	DepartmentID	ManagerID
1	A1111111-AA44-1111-AA44-111111111111	John HR	123 HR Street	123-456-7890	1	1

6. Design Pattern

- **Pattern Used:** The design pattern used for the system are as follows:
 - Singleton
 - Repository (Data-Centered)
 - Facade
- **Purpose:** It suites the logic of the system as follows:
 - Singleton: For creating a single session for managers logging in
 - Repository: Whole data is being stored at a single place from where it is being accessed through queries. Here Database is passive whereas Keylogger and the Dashboard are the client. Given that, a console based app and oneDrive is acting as a mediator (repository).
 - Façade: This was used in the middle layer between database and OneDrive that reads data from the text file and write it in the database. We reduced it's complexity by implementing it in Façade design pattern.
- **Implementation:** Here, the client (Dashboard and Keylogger) is sending and retrieving data from database based on their needs through queries. Additionally, between keylogger and Database. Keylogger sends data to a text file stored in oneDrive which is retrieved at admin-side where the console app enters the data from text file to the database from where the dashboards gets data and displays it to the manager.

Code Snippets:

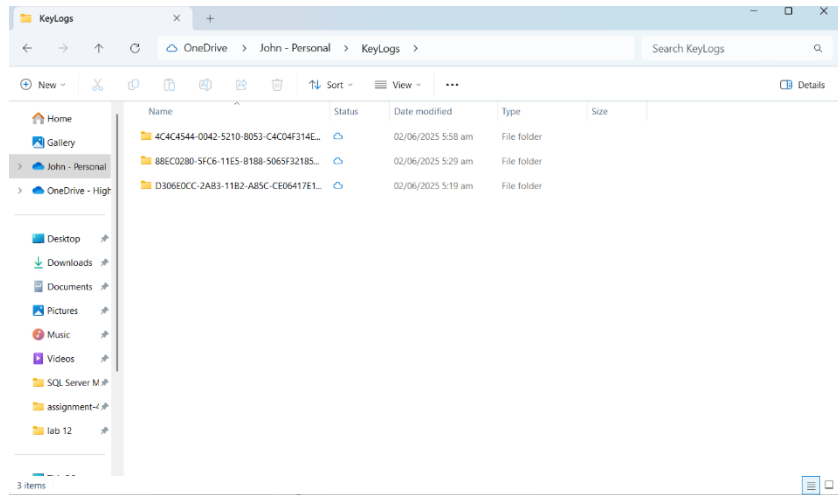
This is the console-based application reading data from text file and entering it into the database. This will not appear anywhere. This will enter data when PC shuts down or KeyLogger stops. It enters data into the database and erases the data from text file. So every time it gets new data from text file and enters it into the database.

```

FileReaderService.cs  DeviceInfoService.cs  KeylogData.cs  KeylogProcessorFacade.cs  Program.cs  ScoringService.cs  DatabaseService.cs
OneDriveKeylogReader
5  namespace OneDriveKeylogReader
6  {
7      2 references
8      public class FileReaderService
9      {
10         1 reference
11         public KeylogData ReadKeylogData(string folder)
12         {
13             string uuidStr = Path.GetFileName(folder);
14             if (!Guid.TryParse(uuidStr, out Guid deviceId))
15                 return null;
16             string keylogPath = Path.Combine(folder, "keylogs");
17             if (!File.Exists(keylogPath)) return null;
18             string[] keylogLines = File.ReadAllLines(keylogPath);
19             if (keylogLines.Length == 0 || string.IsNullOrWhiteSpace(string.Join("", keylogLines)))
20                 return null;
21             string deviceInfoPath = Path.Combine(folder, "deviceinfo");
22             var data = new KeylogData
23             {
24                 DeviceId = deviceId,
25                 KeyStrokes = string.Join(Environment.NewLine, keylogLines.Skip(5).Where(line => !string.IsNullOrWhiteSpace(line)));
26             };
27             if (File.Exists(deviceInfoPath))
28             {
29                 string[] lines = File.ReadAllLines(deviceInfoPath);
30                 if (lines.Length >= 5)
31                 {
32                     data.DeviceName = lines[1].Trim();
33                 }
34             }
35         }
36     }
37 }

```

This is the oneDrive that is being used to take data from keyLogger (Client-side) to Database.



7. Implementation

- **Tools & Technologies:** The technology used for the implementation of this idea are as follows:
 - **C# :** For backend and Front-end implementation
 - **Database:**
 - **MSSQL:** to Develop Database
 - **SQL:** Query language used to insert and retrieve data from database
 - **MVC .Net:** for front-Ent, MVC .Net Framework is used.
- **Functionalities Developed:** Modules Implemented for Different Functionalities:
 - **Keylogger:** Capturing Key strokes at the employee PC. Additionally, it is connected to OneDrive folder where it creates a separate folder for each employee. Within this folder, two text-files are created. One includes the PC information allotted to the employee and the other text-file includes the key strokes captured. This work was done **Ibrahim Yousuf and Keyan Majid**.

```

string uuid = GetUUID();
string mac = GetMacAddress();
string ram = GetRAMSize();
string os = Environment.OSVersion.ToString();
string desktopName = Environment.MachineName;

// Prepare folder path
string oneDrivePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.UserProfile), "OneDrive");
string folderPath = Path.Combine(oneDrivePath, "KeyLogs", uuid);
Directory.CreateDirectory(folderPath);

// Write device info to separate file
string deviceInfoPath = Path.Combine(folderPath, "deviceinfo.txt");
if (!File.Exists(deviceInfoPath))
{
    StringBuilder deviceInfo = new StringBuilder();
    deviceInfo.AppendLine($"{uuid}");
    deviceInfo.AppendLine($"{mac}");
    deviceInfo.AppendLine($"{ram} MB");
    deviceInfo.AppendLine($"{os}");
    deviceInfo.AppendLine($"{desktopName}");

    File.WriteAllText(deviceInfoPath, deviceInfo.ToString());
}

// Set keylog file path
keyLogFilePath = Path.Combine(folderPath, "keylogs.txt");
if (!File.Exists(keyLogFilePath))
{
}

// Keylogging loop
while (true)
{
    for (int i = 8; i <= 255; i++)
    {
        if (GetAsyncKeyState(i) == -32767)
        {
            string key = ConvertKeyCode(i);

```

```

while (true)
{
    for (int i = 8; i <= 255; i++)
    {
        if (GetAsyncKeyState(i) == -32767)
        {
            string key = ConvertKeyCode(i);

            if (!string.IsNullOrEmpty(key))
            {
                if (key == "[Backspace]")
                {
                    if (typedBuffer.Length > 0)
                        typedBuffer = typedBuffer.Substring(0, typedBuffer.Length - 1);
                    if (logBuffer.Length > 0)
                        logBuffer.Remove(logBuffer.Length - 1, 1);
                }
                else
                {
                    logBuffer.Append(key);
                    typedBuffer += key;
                }

                if (typedBuffer.EndsWith("xyz"))
                {
                    FlushBufferToFile();
                    return;
                }
            }

            if (((DateTime.Now - lastFlushTime).TotalSeconds > 5 || logBuffer.Length > 100))
            {
                FlushBufferToFile();
            }
        }
    }
    Thread.Sleep(s);
}
}

```

- **Remote Connectivity With Database:** This included the joint effort of the whole team.
 - **KeyLogger Side Logic: OneDrive Folder Creation.** This included the creation of folder in the drive within this folder, two text files are created i.e DeviceInfo and Keylogs. The folder is created with the name of **DeviceID** which is basically a **UUID** which is the primary key for Employee Table and is unique for each PC given to the employees. This work was done by **Zohaib Arshad Noor and Keyan Majid**.

```

3 reference
public class DeviceInfoService
{
    private readonly string _connectionString;

    1 reference
    public DeviceInfoService(string connectionString) => _connectionString = connectionString;

    1 reference
    public bool Exists(Guid uuid)
    {
        using (var conn = new SqlConnection(_connectionString))
        {
            conn.Open();
            var cmd = new SqlCommand("SELECT COUNT(*) FROM DeviceInfo WHERE DeviceID = @uuid", conn);
            cmd.Parameters.AddWithValue("@uuid", uuid);
            return (int)cmd.ExecuteScalar() > 0;
        }
    }

    1 reference
    public void Insert(Guid uuid, string mac, string ram, string os, string desktop)
    {
        using (var conn = new SqlConnection(_connectionString))
        {
            conn.Open();
            var cmd = new SqlCommand(@"INSERT INTO DeviceInfo (DeviceID, MacAddress, RamSize, OSVersion, DesktopName)
            VALUES (@uuid, @mac, @ram, @os, @desktop)", conn);
            cmd.Parameters.AddWithValue("@uuid", uuid);
            cmd.Parameters.AddWithValue("@mac", mac);
            cmd.Parameters.AddWithValue("@ram", ram);
            cmd.Parameters.AddWithValue("@os", os);
            cmd.Parameters.AddWithValue("@desktop", desktop);
            cmd.ExecuteNonQuery();
        }
    }
}

```

- **Admin-Side Logic:** This included the retrieval of data from folder of a single employee and inserting it into the relevant tables in database and in the relevant employeeID, not mixing up the employees information. The logic of this side was developed by **Ibrahim Yousuf and Jawad Ul Hassan**. This concluded the remote connectivity of client-side (KeyLogger) to our Database server

➔ Some part of code in keylogger, creates folder in OneDrive and logs data in text file created in the folder. Later, at admin side, the above written code retrieves data from text file present in OneDrive and inserts in the Data Base in relevant tables with relevant DeviceID of the Employee.

```
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace facadefinal
{
    3 references
    public class KeyLogService
    {
        private readonly string _connectionString;

        1 reference
        public KeyLogService(string connectionString) => _connectionString = connectionString;

        1 reference
        public void InsertKeyLogs(Guid uuid, string keystrokes)
        {
            using (var conn = new SqlConnection(_connectionString))
            {
                conn.Open();
                var cmd = new SqlCommand("INSERT INTO Keyss (DeviceID, TypedText) VALUES (@uuid, @keystrokes)", conn);
                cmd.Parameters.AddWithValue("@uuid", uuid);
                cmd.Parameters.AddWithValue("@keystrokes", keystrokes);
                cmd.ExecuteNonQuery();
            }
        }
    }
}
```

- **DashBoard:** This module included the logic of retrieval of data from database and showing it to the user(Manager in our case). This includes three main phases: Authentication, Employees Directory, Employee Details.
 - **Authentication:** This is the login Module which is authenticated by the database. This only visibles that manager's employees' data working under that manager.

```
1 using System.ComponentModel.DataAnnotations;
2
3 namespace SDAPProject.Models
4 {
5     4 references
6     public class LoginModel
7     {
8         [Required]
9         2 references
10        public string Username { get; set; }
11
12        [Required]
13        2 references
14        public string Password { get; set; }
15    }
16 }
```

- **Employees' Directory:** This phase display compact information of all the employees working under the manager who has logged in. The data included for each employee in this interface is as follows:
 - Device ID

- Employee Name
- Department ID
- Manager ID

```

}

@if (TempData["ErrorMessage"] != null)
{
    <div class="alert alert-danger alert-dismissible fade show" role="alert" style="margin-bottom: 20px;">
        @TempData["ErrorMessage"]
        <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
    </div>
}

<h1>Employee Directory</h1>

<div class="card-container">
    @foreach (var emp in Model)
    {
        <a style="text-decoration:none" href="@Url.Action("EmployeeDetails", "Dashboard", new { id = emp.DeviceID })">
            <div class="card">
                <h3>@emp.EmpName</h3>
                <p><span>Department ID:</span> @emp.DepartmentID</p>
                <p><span>Manager ID:</span> @emp.ManagerID</p>
                <p><span>Device ID:</span> @emp.DeviceID.ToString().ToUpper</p>
            </div>
        </a>
    }
</div>
</div>
</body>
</html>

```

- **Employee Details:** This last phase shows all the information of the employee upon whom it is clicked in Employees' Directory interface. This shows all the information of that employee which included:
 - Device ID
 - Employee Name
 - PC Information: OS Version, Mac Address e.t.c
 - Key Stroke Violation
 - Score
 - Key Strokes: Displaying all the keystrokes pressed by the employee
- This module was developed by **Ibrahim Yousuf and Keyan Majid**.

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace EmployeeDirectory
{
    public class EmployeeDetailsController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly ILogger<EmployeeDetailsController> _logger;

        public EmployeeDetailsController(ApplicationDbContext context, ILogger<EmployeeDetailsController> logger)
        {
            _context = context;
            _logger = logger;
        }

        public async Task<ActionResult> EmployeeDetails(Guid id)
        {
            var employee = new EmployeeDetailModel();

            using (var connection = new SqlConnection(_configuration.GetConnectionString("DefaultConnection")))
            {
                await connection.OpenAsync();

                using (var command = new SqlCommand("GetEmployeeDetailsByDeviceID", connection))
                {
                    command.CommandType = CommandType.StoredProcedure;
                    command.Parameters.AddWithValue("@DeviceID", id);

                    using (var reader = await command.ExecuteReaderAsync())
                    {
                        if (await reader.ReadAsync())
                        {
                            employee.DeviceID = reader.GetGuid(reader.GetOrdinal("DeviceID"));
                            employee.EmpName = reader["EmpName"].ToString();
                            employee.Address = reader["Address"].ToString();
                            employee.ContactNo = reader["ContactNo"].ToString();
                            employee.DepartmentName = reader["DepartmentName"].ToString();
                            employee.ManagerName = reader["ManagerName"].ToString();
                            employee.DesktopName = reader["DesktopName"].ToString();
                            employee.OSVersion = reader["OSVersion"].ToString();
                            employee.RamSize = reader["RamSize"].ToString();
                            employee.MacAddress = reader["MacAddress"].ToString();
                            employee.ViolationCount = Convert.ToInt32(reader["ViolationCount"]);
                            employee.Score = Convert.ToInt32(reader["Score"]);
                            employee.KeylogData = reader["KeylogData"].ToString();
                        }
                    }
                }
            }

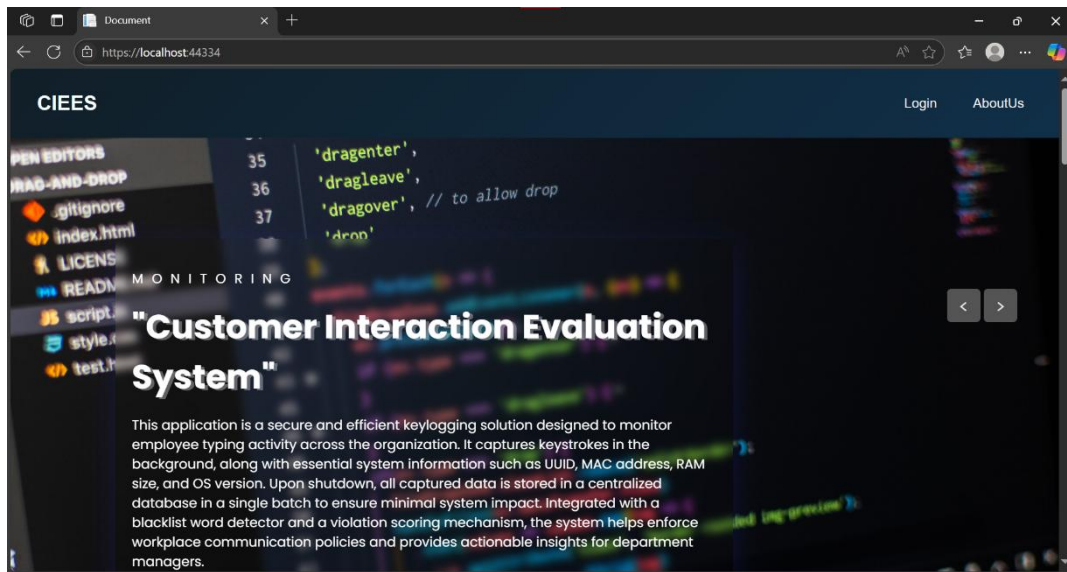
            return View(employee);
        }
    }
}

```

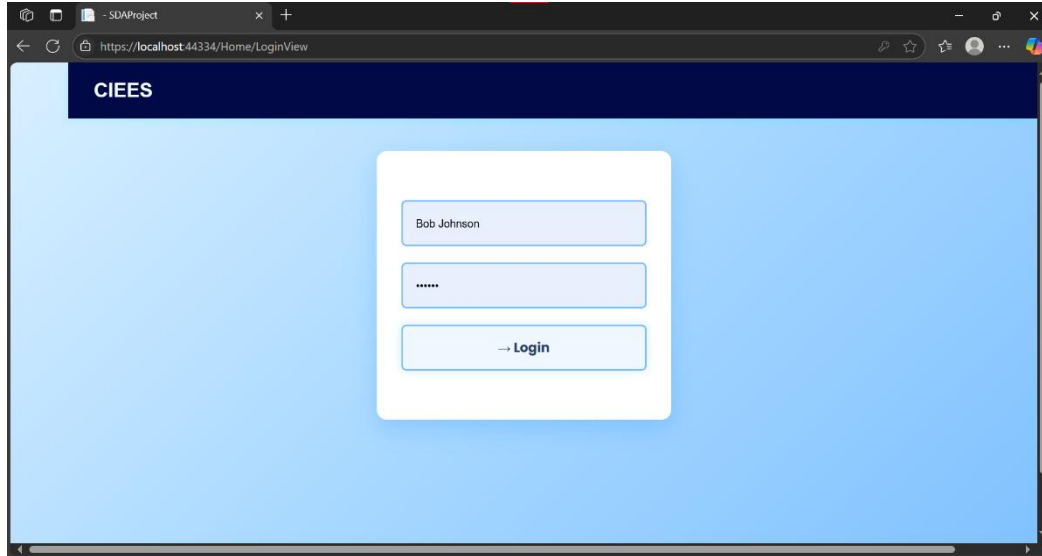
- The integration of all modules together to work in coordination in order to generate the desired output and attain the objective defined is the joint effort of the whole team.

- **Screenshots:** GUI of the System

➔ Landing Page: starting page



➔ Login Page:



➔ Registration:

Bright Gradient Registration x +

https://localhost:44334/Home/RegistrationView

Keylogger Insight Dashboard

Register

Enter Device ID

Enter Name

Enter Address

Enter Mobile Number

Select department

☐ I accept all terms & conditions

Windows Taskbar: Type here to search, 16°C, 10:28 PM 6/1/2025

➔ Employee Directory:

Employee Cards - Bright Theme x +

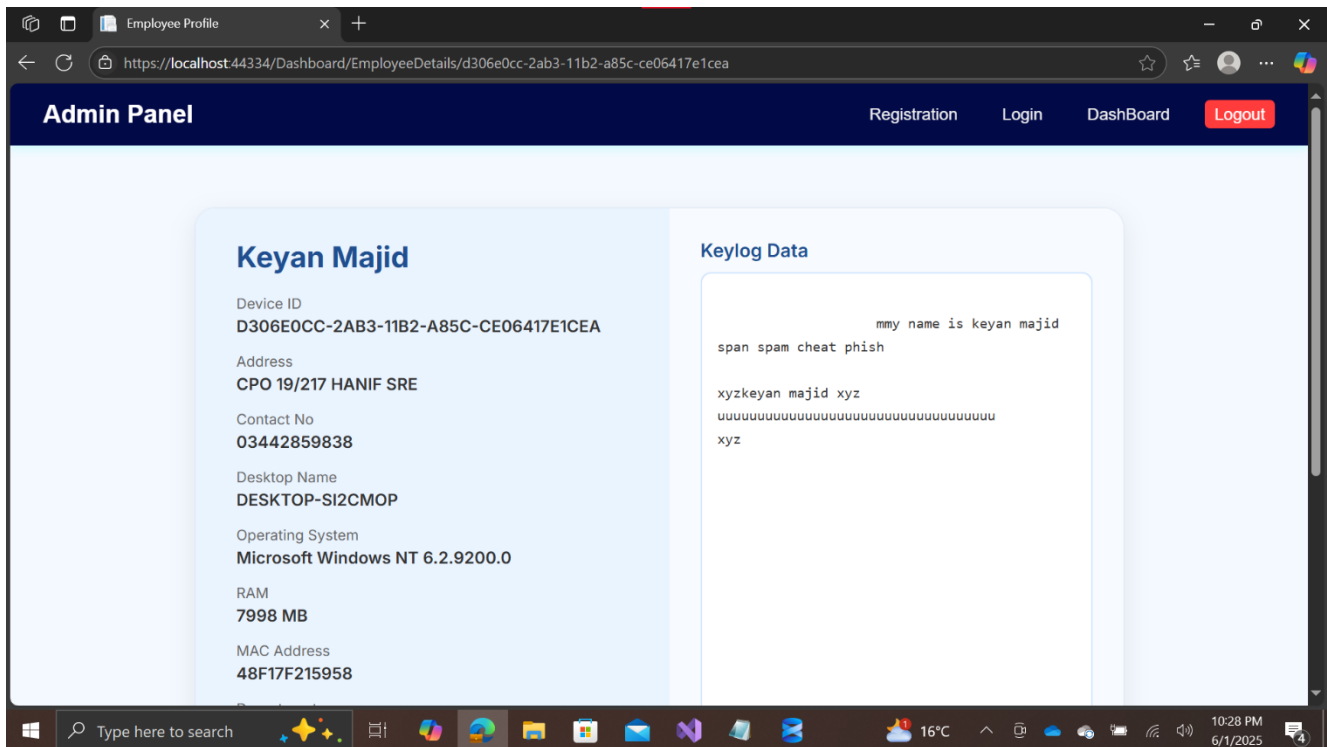
https://localhost:44334/Dashboard/DashBoardView

Admin Panel Registration Logout

Employee Directory

Zohaib Arshad Noor Department ID: 2 Manager ID: 2 Device ID: 88EC0280-5FC6-11E5-B188-5065F32185D2	Ibrahim Yousuf Department ID: 1 Manager ID: 1 Device ID: 4C4C4544-0042-5210-8053-C4C04F314E32	Keyan Majid Department ID: 1 Manager ID: 1 Device ID: D306E0CC-2AB3-11B2-A85C-CE06417E1CEA
---	---	--

➔ Individual Employee Details:



8.

Testing

- **Testing Approach:**

- At very first, authentication is checked along with rerouting of front-end without implementing any functionality in it.
- Manual Testing is done. One person typed the on whose PC, keyLogger was running. Later on, the database is checked whether the data is logged on it or not.
- Second stage is typing the blacklisted words to check whether the scoring is done or not and whether it is stored in database in their relevant tables or not.
- Third stage is to check the remote connectivity. This is done in three stages. For this, a person manually types keys from another PC which are recorded in the remote database.
- In the last stage, we check whether the data stored in database is being shown at front-end or not in the required format.

- **Test Cases:** The basic idea of our application is to monitor the employees' keypresses and evaluate performance of each employee.

- The key test cases for this are follows:
 - In first stage, it is checked whether the data is being logged from keyLogger to the text file present in the OneDrive folder or not.
 - In second stage, it is checked whether the log sent to text file is being read at the admin side OneDrive folder or not
 - In the third stage, it is checked whether the data read at admin-side is put into the relevant table or not.
 - In the last stage, it is checked whether this whole process when repeated with black listed words, whether this will be executed with the scoring logic or not.

-

- **Results:** The main idea of keypresses was to keep the employer informed about the employee's interaction with the customers based on which the evaluation of the individual is done.
 - The most bug fixes were done during the sending of data from keylogger to remote database.
 - First the idea of TCP/IP of Sql was put in practice to directly send your data from keylogger to Sql Database which failed miserably
 - **Cause:** Although all settings were done, but yet Sql was not listening to 1433 port whatsoever
 - Then there came a point when the idea of remote connectivity was about to be dropped but later on this changed to Remote connectivity through OneDrive. We efficiently tried to use OneDrive to log data.
 - This remote connectivity took most of our time as we needed it to input data in the required format so that it could input it correctly in database. Then we needed it to read data and insert it correctly into the database.
 - From here onwards, sending data from database to Admin Dashboard was a piece of cake.

9. Conclusion

- **Summary:**

Monitor employee productivity and compliance with our advanced keylogger tool. This solution tracks all keystrokes, flagging blacklisted words or phrases in real-time. Ensuring a secure, professional work environment while maintaining transparency.

- **Challenges:**

The main challenge was capturing keypresses and remote connectivity. For this, we used techniques to implement the whole business logic

- **Future Work:**

AI integration for reading data trend of Employees performance. Online deployment through AWS is the future goal for better online connectivity