EMPLOYEE MANAGEMENT SYSTEM REPORT

By Memri

Table of Contents

Introduction

This report presents the design and development of an Employee Management Database created by Memri. The system was built to streamline the management of employee records and track workplace infractions in a centralised, secure environment. Designed for use by HR teams and organizational leadership, this solution helps maintain transparency, accountability, and administrative efficiency.

The report is organised into four main sections. The Design section outlines the architectural decisions, data structures, and core algorithmic logic using pseudocode. The Testing section explains the approach taken to validate functionality and includes test cases in tabular form. The Conclusion summarizes the work, reflects on limitations, and discusses lessons learned.

Design

Justification of Data Structures and Algorithms

Relational Data Model:
SQL Server was used to store employee data (Employees), departmental information (Departments), and infractions (ManagementInfo). This normalised structure ensures minimal redundancy and clear data relationships.

Data Binding:
A BindingSource was used in C# to dynamically link the UI (DataGridView) with query results, improving responsiveness and ease of display updates.

Modular Methods:
Methods such as commentSave(), workerData(), and Skip() separate logic concerns (data entry, data navigation, and data viewing), improving maintainability.

Stored Procedures:
GetWorkerByID, ComWorkAndManage, and ViewID encapsulate database logic, which enhances security, efficiency, and abstraction.

Key Algorithm Pseudocode

Saving an Infraction Comment

```
FUNCTION SaveComment(employeeID, comment, date):
    IF employeeID is invalid THEN
        RETURN Error
    ENDIF
    EXECUTE SQL:
        INSERT INTO ManagementInfo (Id, Date, Comment)
        VALUES (employeeID, date, comment)
    RETURN Success
```

Load Employee Profile

```
FUNCTION LoadEmployee(index):
    employeeRow = dataset.Table[0].Row[index]
    DISPLAY employeeRow data in UI fields
```

Retrieve Infraction for Employees

```
FUNCTION LoadInfractions(employeeID):
    EXECUTE stored_procedure ComWorkAndManage WITH @Combine = employeeID
    BIND result to DataGridView
```

These core functionalities enable recordkeeping, reviewing, and updating workflows related to employee behaviour.

Testing

Testing Method

We used black-box testing, focusing on verifying each functionality through its inputs and expected outputs without viewing source code (GeeksForGeeks, 2025). Scenarios included adding comments, navigating profiles, and searching by employee ID.

Test Cases

Description	Input	Expected Output	Result
Save valid infraction comment	Comment="Late", Date= 17/04/2025	"Updated" message box	Pass
View employee profile	Form Load	Employee data populated in form	Pass
Navigate to next profile	Click 'Next'	Next employee data displayed	Pass
Search employee by valid ID	ID=2	Employee and comments shown	Pass
Search by non-numeric input	ID="John"	Warning about numeric input	Pass
Navigate beyond last profile	Click 'Next' at end	"No more profiles" message shown	Pass

Conclusion

Group Members

Team Leader (SCRUM Master): Egypt Thomas || M001009527 || ET588@live.mdx.ac.uk

Secretary: Imaan M || M00983697 || IM671@live.mdx.ac.uk

2x Developers:

Rabia Cheema || M00946446 || RC1114@live.mdx.ac.uk

Mihai Popescu || M00946446 || MP1643@live.mdx.ac.uk

Tester: Maha Fareed || M00949933 || MF955@live.mdx.ac.uk

Summary of Work Done

The application provides a reliable interface for employee management, integrating:

- Employee viewing and editing
- Logging infractions and comments
- Navigating employee records
- Searching by ID
- SQL-backed storage and querying via stored procedures

SCRUM

As a team, we actively participated in weekly remote scrum meetings (Sutherland & Ken, 2020)on Microsoft Teams, held every Monday for 30 minutes from 6th February to 7th April. These meetings provided a structured platform for discussing key tasks, tracking progress, and ensuring alignment across the team.

Limitations & Critical Reflection

- Lack of Role-Based Access Control (RBAC): Currently, there is one user which all permissions, which could be problematic in a real-world deployment.
- No real-time sync or API integration: Data updates occur on user action; no background sync or notifications are implemented.
- Code Duplication in Navigation Logic: Repeated SQL and binding logic in btnNext, button1, and btnInfo methods could be abstracted.

Causes of Limitations

Team Performance: This project was significantly impacted by poor group dynamics. Out of five group members — a group leader, two developers, a secretary, and a tester — only the group leader and developers maintained consistent commitment and communication.

- The **group leader** organized weekly meetings, managed documentation, and led development when others were unable to contribute effectively.
- The **developers** made some effort to participate, but both struggled:
    - One lacked access to a computer for a large part of the project.
    - The other was frequently unavailable due to work or prioritising other modules.
    - Both showed a limited understanding of C# and the project requirements, which made collaboration challenging and increased the workload on the group leader.
- The **secretary and tester** did not attend any meetings, never contributed to group chats, and made no visible effort to participate in the project at all.

Attempts to Resolve Issues: These challenges were communicated to lecturers through messages. However, despite efforts to raise concerns early, little action was taken, and the group structure remained the same. This meant that the majority of the project responsibility fell to one individual.

Impact on the Project: Because of the imbalance in workload and limited technical contribution from team members, some advanced features were not implemented. Testing was constrained, and the code structure could not be refactored to a more optimal standard due to time limitations. While the core functionality was completed, the experience highlighted the difficulty of working in a group where participation is unequal.

Future Approach: If a similar task arises in the future, stronger boundaries would be set early — including setting expectations clearly, reporting issues sooner, and pushing harder for adjustments in group assignments if needed. Having reliable teammates is important but so is advocating for your own workload and wellbeing when things start going wrong.

Future Improvements

- Introduce RBAC using a login system that defines roles (e.g., Admin, Manager, Viewer).
- Create a LoadData() method to consolidate repeated SQL fetch logic.
- Refactor navigation and search methods for better scalability.
- Expand the UI to include export/reporting features (PDF, Excel).

Bibliography

GeeksForGeeks. (2025, April 9). *Black Box Testing – Software Engineering*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/software-engineering-black-box-testing/

Sutherland, J., & Ken, S. (2020). *The Scrum Guide*.