

**Project Report**

**Project: Employee Management System**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Only for Course Teacher** | | | | | | |
|  | | **Needs Improvement** | **Developing** | **Sufficient** | **Above Average** | **Total Mark** |
| **Allocate Mark & Percentage** | | **25%** | **50%** | **75%** | **100%** | **5** |
| **Clarity** | **1** |  |  |  |  |  |
| **Content Quality** | **2** |  |  |  |  |  |
| **Spelling & Grammar** | **1** |  |  |  |  |  |
| **Organization and Formatting** | **1** |  |  |  |  |  |
| **Total Obtained Mark** | | | | | |  |
| **Comments** |  | | | | | |

**Semester: Summer 2025** **Presented by: Group-E**

|  |  |
| --- | --- |
| **Rusafi Bin Rowshan** | **242-35-520** |
| **Jannatul Fardous** | **242-35-156** |
| **Prioty Hossain** | **242-35-276** |

**Batch: 43 Section: K1**

**Course Code: SE133 Course Name: Software development capstone Project**

**Course Teacher Name: MD. Sakib Ali Mazumder**

**Designation: Lecturer Submission Date:** **20/08/2025**

A blue sign with text

AI-generated content may be incorrect.

Project Report

Project Title: Employee Management System

**Presented by**:  
Prioty Hossain (Leader) 242-35-276

Jannatul Fardous 242-35-156

Rusafi Bin Rowshan 242-35-520

# **Acknowledgement**

We, the members of this project group, would like to acknowledge the support of everyone who helped us complete this project.

First, we are thankful to our project teacher for their guidance, valuable suggestions, and continuous encouragement, which helped us understand the concepts clearly and implement them successfully. We are also grateful to our friends for their help, ideas, and support during testing and refining the project.

Finally, we would like to thank our families for their patience and motivation, which kept us focused and confident throughout the project. This project would not have been possible without the guidance, support, and encouragement of all these people.

# **Abstract**

Every organization, whether big or small, has to manage employees.  
If we try to do this work manually, it becomes slow, messy, and full of mistakes.  
That is why we thought of creating a simple digital solution – the Employee Management System (EMS) using the C programming language.

In this system, employees can register themselves by giving all their information, like personal details, education history, and emergency contacts.  
After registration, they get a unique ID, which they can use to log in whenever they want.  
Once logged in, employees can see their profile, apply for leave, and also check the status of their applications.

On the other hand, the Admin has more control.  
The admin can see a list of all registered employees, check their details, and review leave applications.  
If needed, the admin can even delete an employee’s record from the system.

The biggest strength of this project is that it is easy, organized, and employee-friendly.  
Employees don’t have to depend on the admin for registration, and at the same time, the admin has enough control to keep the system clean and updated.  
All information is saved permanently in files using C file handling, which makes the system reliable and transparent.

Overall, this project makes employee management faster and more systematic.  
It also shows how simple C programming concepts like structures, file handling, and authentication can be used to build something that works like a real office tool.  
Even though this is a console-based project, in the future it can grow into a full Human Resource Management System (HRMS) with more advanced features.

Table of Contents

[Presented by: 1](#_Toc206612773)

[**1.** **Executive Summary:** 4](#_Toc206612774)

[**2.** **Introduction:** 4](#_Toc206612775)

[**3.** **Problem Statement**: 4](#_Toc206612776)

[**4.** **Project Objectives**: 5](#_Toc206612777)

[Scope of the Project: 6](#_Toc206612778)

[07. Requirements Elicitation: 6](#_Toc206612779)

[Techniques Used 6](#_Toc206612780)

[Application 7](#_Toc206612781)

[08. Functional Requirements: 8](#_Toc206612782)

[Non-Functional Requirements: 9](#_Toc206612783)

[09. Requirement 9](#_Toc206612784)

[Technique 9](#_Toc206612785)

[10. Priority List 9](#_Toc206612786)

[11. Tools & Technologies (What We Will Use): 10](#_Toc206612787)

[12. Gantt chart Graph: 11](#_Toc206612788)

[13. Use Case diagram: 12](#_Toc206612789)

[14. Use Case Description: 13](#_Toc206612790)

[Use Case: Employee Registration 13](#_Toc206612791)

[Use Case: View all employees 13](#_Toc206612792)

[Use Case Description: Application requests 14](#_Toc206612793)

[Use Case Description: Exit 14](#_Toc206612794)

[15. Activity Diagram: 15](#_Toc206612795)

[16. Sequence diagram: 16](#_Toc206612796)

[18. Limitations: 21](#_Toc206612797)

[19. Future Improvements: 21](#_Toc206612798)

[20. Conclusion: 22](#_Toc206612799)

[User Manual 22](#_Toc206612800)

# **Executive Summary:**

This project focuses on building a simple and user-friendly Employee Management System using the C programming language. The goal is to create a basic software that allows storing and managing employee records such as name, ID, department, salary, and position.

The system will be menu-driven, run in the console, and use file handling to store employee data. This project will help students or beginners understand how real-life HR systems work in a simple way and how to implement file-based databases in C.

# **Introduction:**

In real companies, employee data is managed using large and complex HR software. But for learning purposes, we can build a mini version of such a system that includes important features like adding, updating, searching, and deleting employee data.

This system is useful for small businesses, training centers, or class projects where we want to manage employee information without using advanced databases or online systems.

# **Problem Statement**:

In many small offices or training centers, employee information is written on paper or Excel. This takes a lot of time, it can be confusing and may have mistakes.

Our project will solve this problem by creating a simple computer program to manage employee details. It will help users store and find employee information easily. At the same time it will help students understand how to build real applications using C language and file handling.

# **Project Objectives**:

The main objective of the Employee Management System is to create a simple, efficient, and secure platform for managing employee information. Specifically, this project aims to:

1. Allow employees to register themselves with full personal and professional details.
2. Provide a secure login system for employees to access their own data.
3. Enable the admin to view all employee records easily.
4. Allow the admin to delete employee records when necessary.
5. Reduce manual paperwork and increase accuracy in employee management.
6. Demonstrate practical file handling and programming skills using C

The main goal is to keep it simple, educational and effective.

1. **Tools Used**
2. Programming Language: C
3. Compiler: CodeBlocks / GCC
4. Operating System: Windows / Linux
5. File Handling: Text files to store employee data

# Scope of the Project:

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Employee Registration | Add new employee data like name, ID, etc. |
| View Employees | Display a list of all employees |
| Search Employee | Find employee by ID or name |
| Make/ view Application | Querry management within the system |
| Delete Employee | Remove an employee from the system |
| File Handling | Save employee data using text or binary files |

# 07. Requirements Elicitation:

Requirements elicitation means finding out what users and people involved really need from the software. It helps to understand what the system should do to solve their problems.

For the Employee Management System project, we used different ways to gather this information so that the system would actually help with managing employee details in real life.

## Techniques Used

| Technique | Description |
| --- | --- |
| Observation | Watching how employee records are maintained manually in small offices. |
| Interviews | Asking questions to potential users (e.g., office admins, teachers) about what they need. |
| Brainstorming | Team discussion to list out essential features. |
| Document Analysis | Reviewing existing EMS or HR software systems, textbooks, and sample projects. |

Application of Techniques

**Observation:-**

01. Visited 2 small businesses where employee records were stored in paper files or Excel sheets.

02. Noted challenges:

- Data duplication

- Time-consuming to search manually

- No backup

03. Concluded that digital record-keeping with search/update/delete was essential.

**Interviews:-**

01. Interviewed 2 university staff and 1 private tutor managing assistant records.

02. Common requirements:

- Add employee details with unique ID

- View employee list

- Search/filter by name or ID

- Update mistakes (e.g., wrong salary or department)

- Print or export reports

**Brainstorming**

01. Brainstormed with teammates and supervisor.

02. Derived additional ideas like:

- Marking employees as active/inactive

- Exporting data to a file

- Sorting records by salary or name

- Showing average salary or total employees

**Document Analysis**

01. Studied:

- Sample EMS projects on GitHub (in C and Python)

- HRMS case studies in IT textbooks

- Tutorials on CRUD and file management in C

# 08. Functional Requirements:

* Login system
* Accept employee data from user
* Validate input (no duplicate inputs)
* Perform file operations (save, read, update)
* Verify input
* Add employee
* Display all employees
* Search employee by ID
* Search employee by Name
* View employee information
* Delete employee by ID
* Count total employees
* Filter employees by department
* Sort employees by name
* Sort employees by salary

# Non-Functional Requirements:

* Easy to use with clear menu instructions
* Responds quickly to user actions
* Works on any computer with a C compiler
* Does not crash on invalid input
* Stores data in files even after closing program

09. RequirementPrioritization

Requirement prioritization is the process of ranking system requirements based on their importance, urgency, and impact. This helps in focusing on the most critical features first, ensuring an efficient use of development time and resources.

TechniqueUsed:MoSCoWMethod

The MoSCoW method classifies requirements into four categories:

- **Must Have :** Essential features without which the system will fail to meet its purpose.

- **Should Have :** Important features that add significant value but are not vital.

- **Could Have :** Nice-to-have features that enhance user experience but can be delayed.

- **Won’t Have :** Features agreed to be out of scope for the current project phase.

This method was chosen for its simplicity and effectiveness in balancing scope and delivery.

# 10. Priority List

**Must Have**

* Accept employee data from user
* Validate input (no duplicate inputs)
* Show menu with options (Add, View, Search, Update, Delete)
* Add a new employee
* Display all employees
* Search employee by ID
* Delete employee by ID
* Perform file operations (save, read, update)
* Verify input

**Should Have**

* Search employee by Name
* Login system for admin

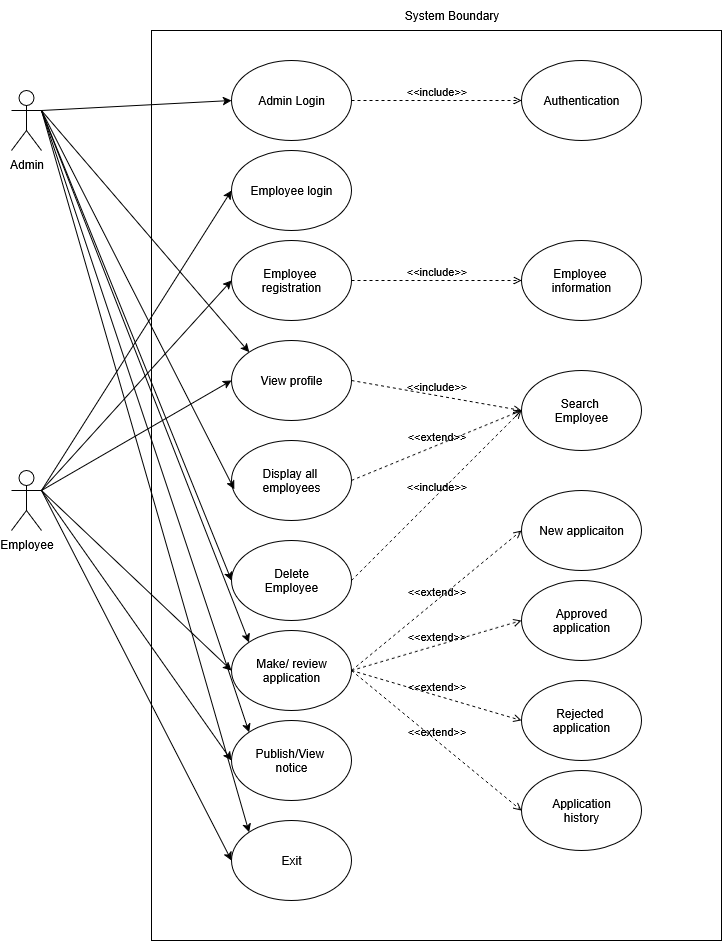
**Would/Won’t Have**

* Filter employees by department
* Sort employees by name
* Sort employees by salary

# 11. Tools & Technologies (What We Will Use):

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| C Programming | Main language for development |
| Code::Blocks / VS Code | For writing and compiling code |
| File Handling (C) | For storing data like accounts, logs |
| Windows/Linux OS/Android/IOS | Operating system where project will run |

# 13. Use Case diagram:



# 14. Use Case Description:

# Use Case: Employee Registration

|  |  |
| --- | --- |
| Field | Description |
| Use Case Name | Employee Registration |
| Actor(s) | Employee, |
| Description | Allows an employee to input their own information for entry into the system. |
| Precondition(s) | System is running. Employee has access to the ‘Employee Registration' form. |
| Postcondition(s) | Employee data is successfully saved to the system. |
| Normal Flow | 1. Actor selects ‘Employee Registration’ from the menu. 2. System displays an input form. 3. Actor enters employee details (Name, ID, Department, etc.). 4. Actor submits the form. 5. System validates and stores the data. |
| Alternate Flow | 3a. If any field is left blank, system prompts for completion. 3b. If ID already exists, system requests a new unique ID. |
| Exception(s) | E1. System fails to save due to file error. E2. Validation fails due to invalid input format. |

# Use Case: View all employees

|  |  |
| --- | --- |
| Field | Description |
| Use Case Name | View All Employees |
| Actor(s) | Admin |
| Description | Allows viewing a list of all employees. Admins can search employees by ID or Name. |
| Precondition(s) | User or Admin must be logged into the system. |
| Postcondition(s) | List of employees is displayed based on search/filter criteria. |
| Normal Flow | 1. Actor selects 'View All Employees' from the menu. 2. System retrieves all employee records. 3. System displays the list of employees. 4. Actor may enter search criteria (e.g., ID, Name). 5. System filters and displays matching employees. |
| Alternate Flow | 4a. If no search input is provided, system shows the full list. 4b. If partial name or ID is provided, system performs a partial match. |
| Exception(s) | E1. File containing employee data not found. E2. No employees match the search criteria. |

# 

# Use Case Description: Application requests

|  |  |
| --- | --- |
| Use Case Name | Application Requests |
| Actor(s) | Admin |
| Description | Allows reviewing of submitted application from employee |
| Precondition | User must be logged in. Application record must exist. |
| Postcondition | The Application data is updated and saved to the database. |
| Main Flow | 1. Actor selects ‘Application requests'. 2. System displays existing Application data. 3. Actor inputs the required fields. 4. System validates input. 5. System updates the data in the file/database. 6. Confirmation is shown. |
| Alternative Flow | 2a. If Application does not exist, an error message is shown. 4a. If input is invalid, system prompts for correction. |
| Exception Flow | • File/database error • Unauthorized access attempt |
| Priority | High |

# Use Case: View Profile

|  |  |
| --- | --- |
| Field | Description |
| Use Case Name | View Profile |
| Actor(s) | Employee, admin |
| Description | Allows an employee to view their personal profile after logging in. |
| Precondition(s) | Employee is logged into the system. |
| Postcondition(s) | Employee profile information (ID, Name, Department, etc.) is displayed. |
| Normal Flow | 1. Actor selects "View Profile" from the employee menu.  2. System retrieves the logged-in employee’s data.  3. System displays profile details (Name, ID, Department, etc.). |
| Alternate Flow | 2a. If no profile data is found, system shows "No profile available." |
| Exception(s) | E1. File error while retrieving data. |

# Use Case: Application Requests (Admin)

|  |  |
| --- | --- |
| Field | Description |
| Use Case Name | Application Requests (Admin) |
| Actor(s) | Admin |
| Description | Allows the admin to review, approve, or reject applications submitted by employees. |
| Precondition(s) | Admin is logged into the system. At least one application exists. |
| Postcondition(s) | Application status is updated (Approved / Rejected) and stored in the system. |
| Normal Flow | 1. Actor selects "Application Requests" from the admin menu.  2. System displays the list of pending applications.  3. Actor selects an application to review.  4. System shows application details.  5. Actor chooses to approve or reject.  6. System updates and saves the decision. |
| Alternate Flow | 2a. If no new applications exist, system shows "No applications available." |
| Exception(s) | E1. Application not found.  E2. Invalid input during approval/rejection.  E3. File/database error. |

# **Use Case:** **View Application Status**

|  |  |
| --- | --- |
| Field | Description |
| Use Case Name | View Application Status |
| Actor(s) | Employee |
| Description | Allows an employee to check the approval status of their submitted applications (e.g., leave request). |
| Precondition(s) | Employee is logged into the system and has submitted an application. |
| Postcondition(s) | Application status (Approved / Rejected / Pending) is displayed. |
| Normal Flow | 1. Actor selects "View Application Status" from the employee menu.  2. System retrieves the employee’s application record.  3. System displays the current status of the application. |
| Alternate Flow | 2a. If no application exists, system shows "No application found." |
| Exception(s) | E1. File/database error. |

# Use Case: Search Employee

|  |  |
| --- | --- |
| Field | Description |
| Use Case Name | Search Employee |
| Actor(s) | Admin |
| Description | Allows the admin to search for an employee by ID or Name. |
| Precondition(s) | Admin is logged into the system. |
| Postcondition(s) | Employee profile matching the search criteria is displayed. |
| Normal Flow | 1. Actor selects "Search Employee" from the admin menu.  2. System prompts for search input (ID or Name).  3. Actor enters search criteria.  4. System retrieves and displays the matching employee details. |
| Alternate Flow | 4a. If partial input is given, system shows a list of possible matches. |
| Exception(s) | E1. No employee matches the search criteria.  E2. File containing employee data not found. |

# Use Case: Delete Employee

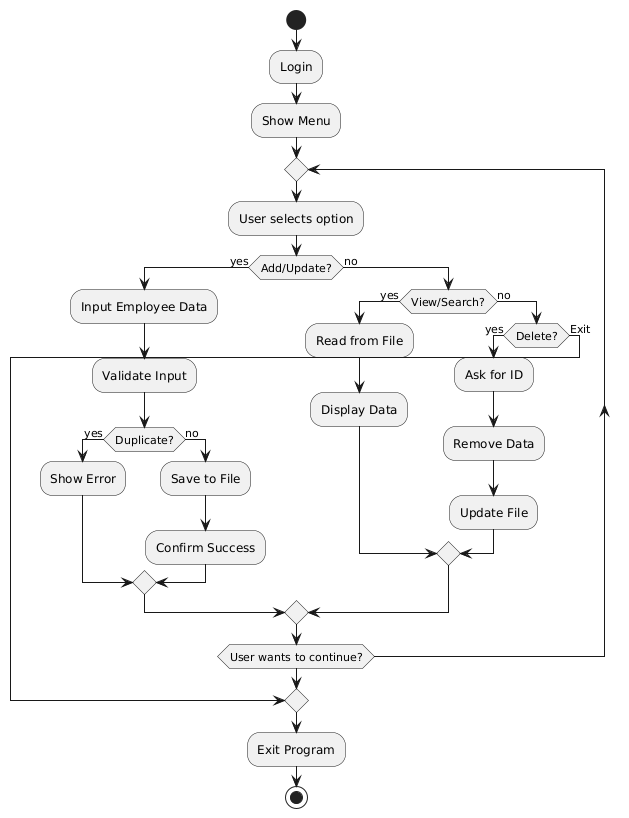
|  |  |
| --- | --- |
| Field | Description |
| Use Case Name | Delete Employee |
| Actor(s) | Admin |
| Description | Allows the admin to delete an employee record by ID or Name. |
| Precondition(s) | Admin is logged into the system. Employee record exists. |
| Postcondition(s) | The selected employee record is permanently removed from the system. |
| Normal Flow | 1. Actor selects "Delete Employee" from the admin menu.  2. System prompts for input (ID or Name).  3. Actor enters the employee ID or Name.  4. System locates the record and deletes it.  5. Confirmation message is shown. |
| Alternate Flow | 3a. If input is invalid, system requests correction.  4a. If multiple employees match (by name), system asks for clarification |
| Exception(s) | E1. Employee record not found.  E2. File/database error prevents deletion. |

# Use Case Description: Exit

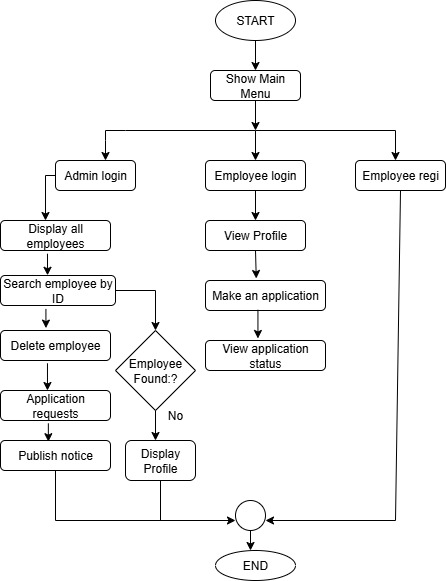
|  |  |
| --- | --- |
| Use Case Name | Exit |
| Actor(s) | Admin, HR, Employee |
| Description | Allows the user to safely exit the application. |
| Precondition | User is logged in and interacting with the system. |
| Postcondition | System is closed; all unsaved data is discarded or saved based on system configuration. |
| Main Flow | 1. Actor selects 'Exit' from the menu. 2. System prompts to save any unsaved changes. 3. Actor confirms exit. 4. System closes all active sessions and exits. |
| Alternative Flow | 2a. If there are no unsaved changes, system directly proceeds to exit. |
| Exception Flow | • System fails to close due to an internal error |
| Priority | Medium |

# 

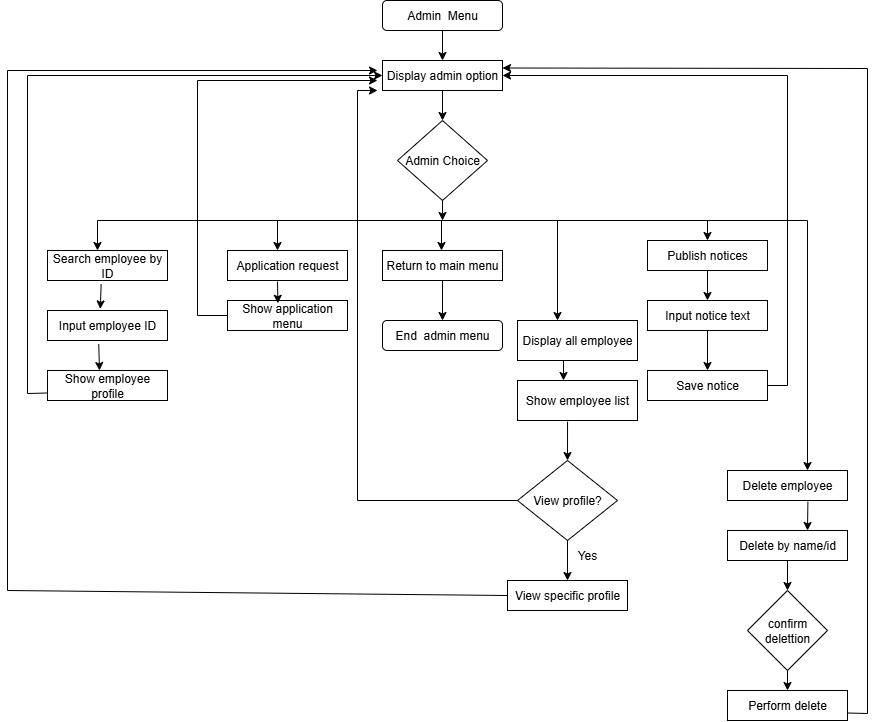
# 15. Activity Diagram:



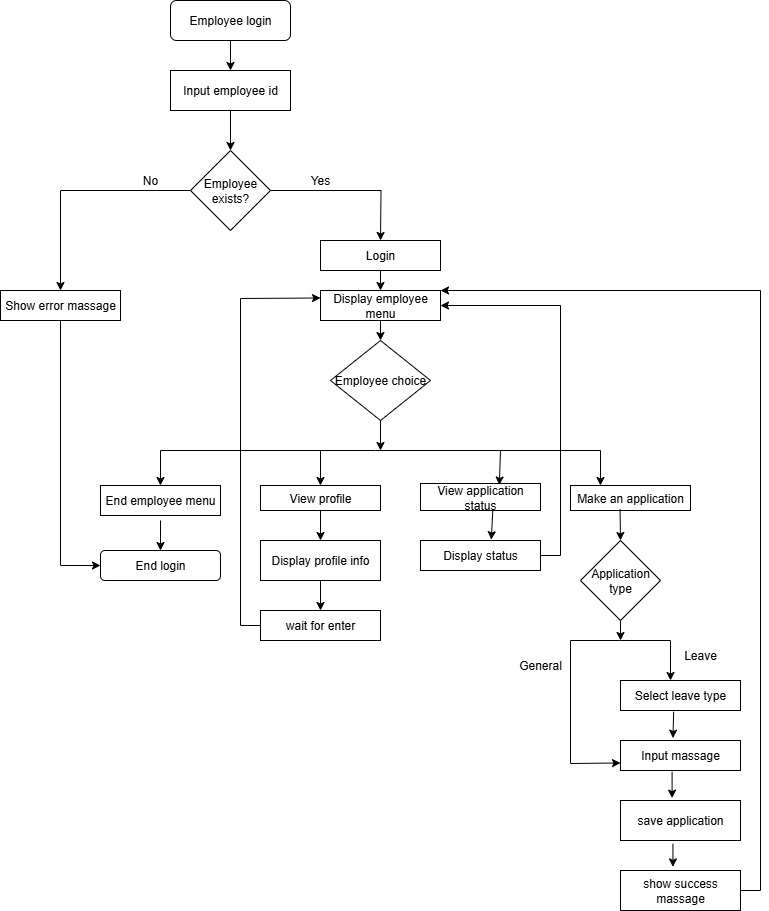
**Activity Diagram: Main menu**

****

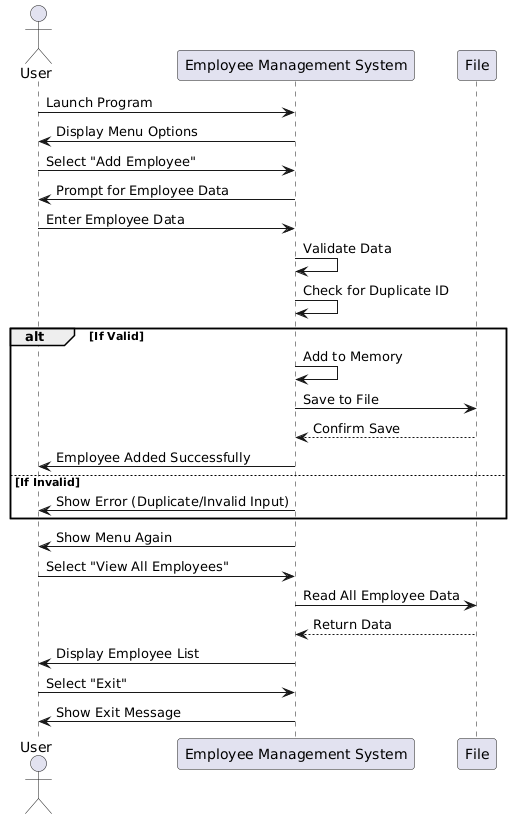
**Activity Diagram: Admin Menu**

****

**Activity: Employee menu**

****

# 16. Sequence diagram:



**C Concepts We Used Our Project**

This project uses core C concepts like structures, arrays, file handling, functions, loops, conditional statements, and string manipulation to manage employee data efficiently.

**In figure 1**, we have used,

**A screenshot of a computer code

AI-generated content may be incorrect.**

Fig 1: Uses of Header File

Here,

**#include** in C is used to include libraries or header files so that their functions can be used in the program.

**stdio.h** – Handles input/output, like displaying messages with printf and taking user input with scanf.

**stdlib**.**h** – Provides system-related functions, like memory allocation (malloc), program exit (exit), and random number generation.

**string.h** – Used for handling strings, like copying (strcpy), comparing (strcmp), and finding length (strlen).

**windows**.**h** – Helps format the Windows console, like changing colors or setting cursor positions.

**time.h** – Manages date and time, like recording registration dates or getting the current system time.

**conio.h** – Used for console input, mainly to hide password input using getch().

**In Figure 2**, We have used structures to organize employee data efficiently.

A screen shot of a computer

AI-generated content may be incorrect.

Fig 2: Structures

**In Figure 3**, Arrays are used to store multiple educational details for each employee.A computer code with text

AI-generated content may be incorrect.

Fig 3: Arrays

**In Figure 4**, shows all functions used for modularity and reusability with prototypes declared at the program’s start.A screenshot of a computer

AI-generated content may be incorrect.

Fig 4: Uses of Function

**In Figure 5**, Loops and conditions are used to control program flow and handle user choices efficiently.A screenshot of a computer code

AI-generated content may be incorrect.

Fig 5: Conditions and Loops

**In Figure 6**, We have used file handling to store and manage employee details permanently.

A close up of text

AI-generated content may be incorrect.

Fig 6: File Handing

**In Figure 7**, Login system ensures secure access for registered employees.

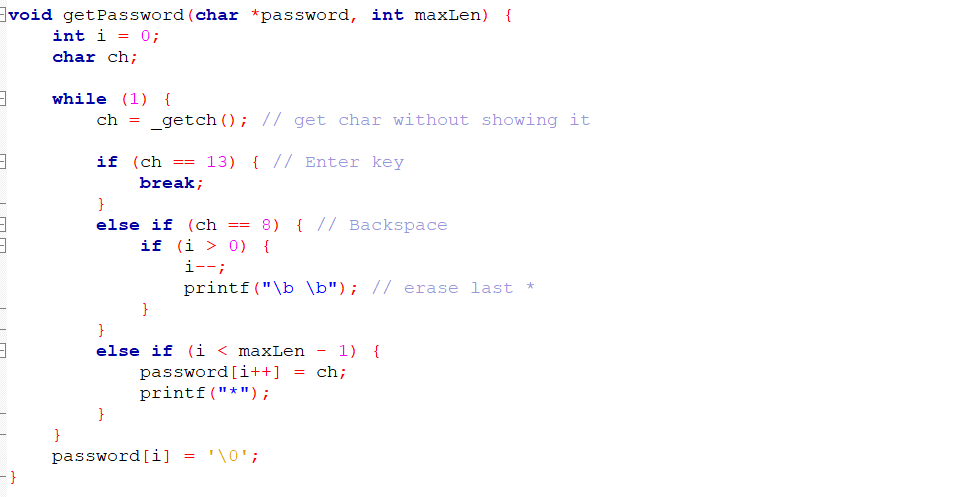


Fig 7: User Authentication

# 18. Limitations:

* The program works only in the text screen (no graphical interface).
* It only works on one computer, data is not shared online.
* There is no sorting of employee data (like sorting by name or salary).
* It cannot connect to a real database like MySQL.

# 19. Future Improvements:

* Add data sorting and filtering features
* Add GUI using C++, Python, or Java
* Use databases like MySQL for large-scale data
* Export reports to Excel or PDF

# 20. Conclusion:

This capstone project is an excellent opportunity for us to apply our programming knowledge to a real-life problem. It teaches us:

• File handling in C

• Project planning and teamwork

• Menu-based system design

• Basic security and validation logic

By completing this project, we will understand the structure of employee management system and be ready for more complex software development in the future.

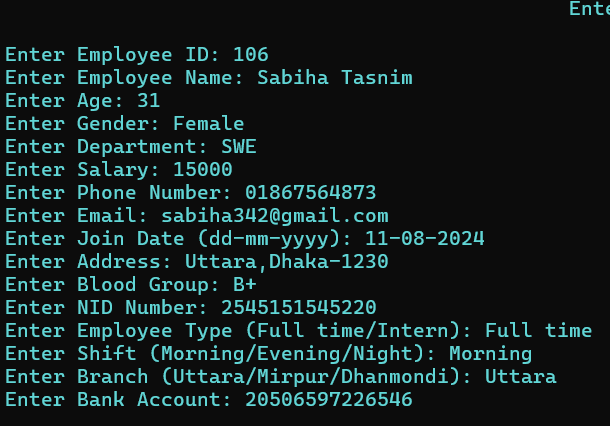
User Manual**:**

1. Start the Program
2. Compile and run the program.
3. You will see the main menu with options:

****

**Function:** Employee Registration

1. If you are a new employee, choose “Employee Registration”.
2. Enter your details as prompted:

****

A screenshot of a computer

AI-generated content may be incorrect.

1. Once completed, your details are saved in the system.
2. After registration, you can use these credentials to log in.

**About Employee Login**

1. Select “Employee Login” from the main menu.
2. Enter your Employee ID and Name.



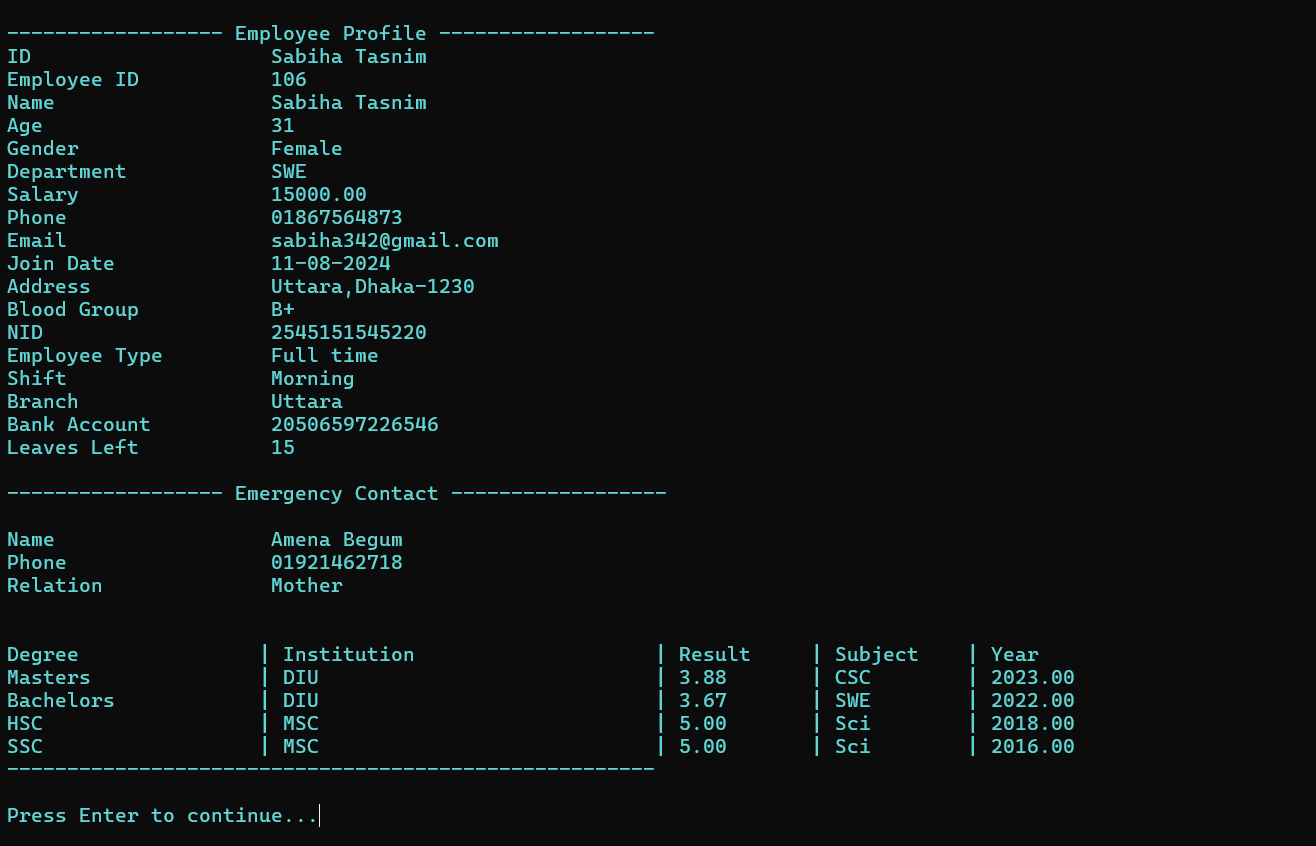
1. If credentials match, you will be logged in successfully.

A screen shot of a black background

AI-generated content may be incorrect.

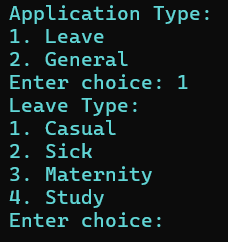
1. Once logged in, you can view your personal details.
2. Make an application and view applications status, then exit.

**1. View Profile:** choose 1**.**

****

Choosing option 1 from the employee menu shows the profile of the user.

**Fuction: Make an application**

****

The user can make an application to the admin using option 2 from the menue

**If user choose 2 for leaving because of sickness.**

**A screen shot of a black background

AI-generated content may be incorrect.**

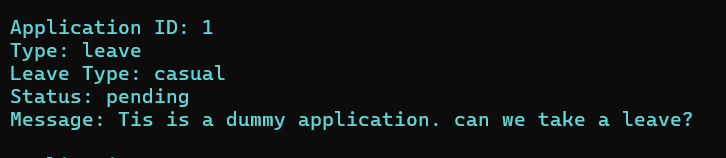
**Step 1:** Choose application type.

**Step 2:** Choose leave type.

**Step 3:** Write the desired application.

**Step 4:** Press enter to submit the application for the admin to review.

**Function: View application status**

****

Selecting option 3 from employee menu shows application status weather it is approved or not by the admin.

That’s all from the employee menu.

**Admin Menu**From the main menu, if the user chooses option 1, it wll navigate them to the admin menu after asking password

**A black screen with a black background

AI-generated content may be incorrect.**

If the password is correct this menu is shown as admin menue.

**A screen shot of a computer

AI-generated content may be incorrect.**

**1. Display all employees**

**A screenshot of a computer

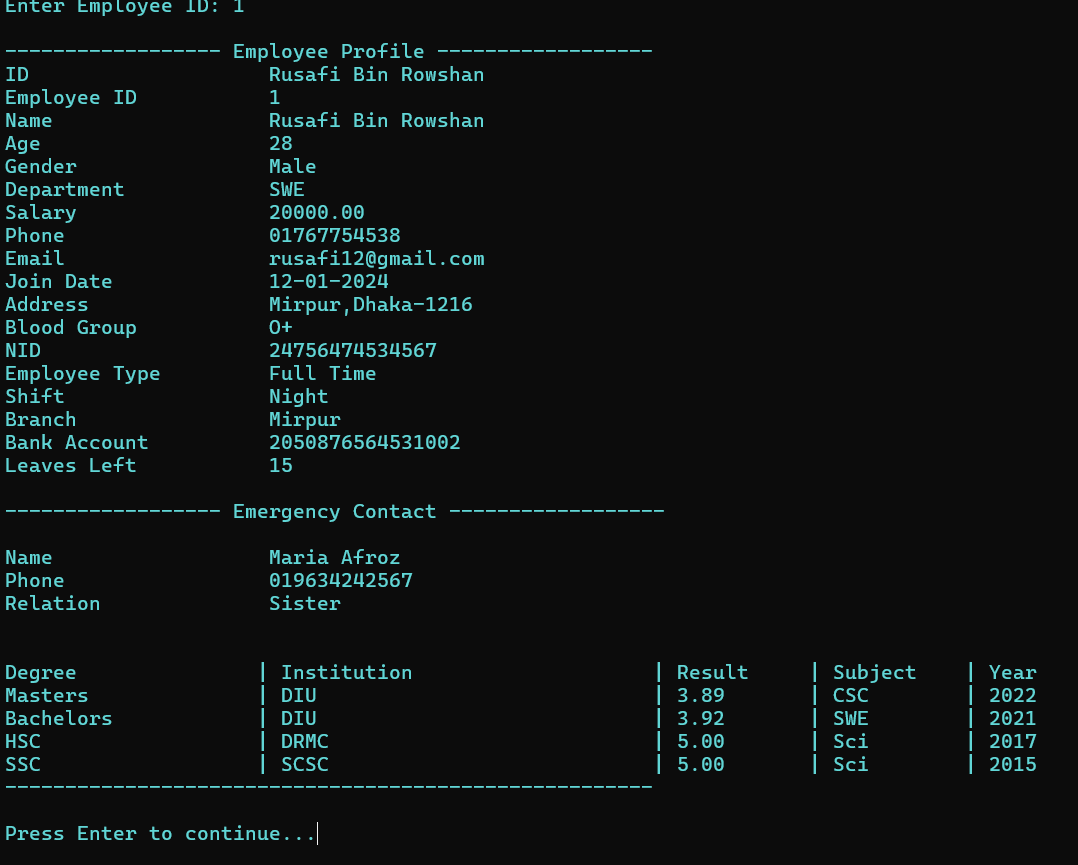
AI-generated content may be incorrect.**

**Step 1:** The user chooses 1. Display all employees.

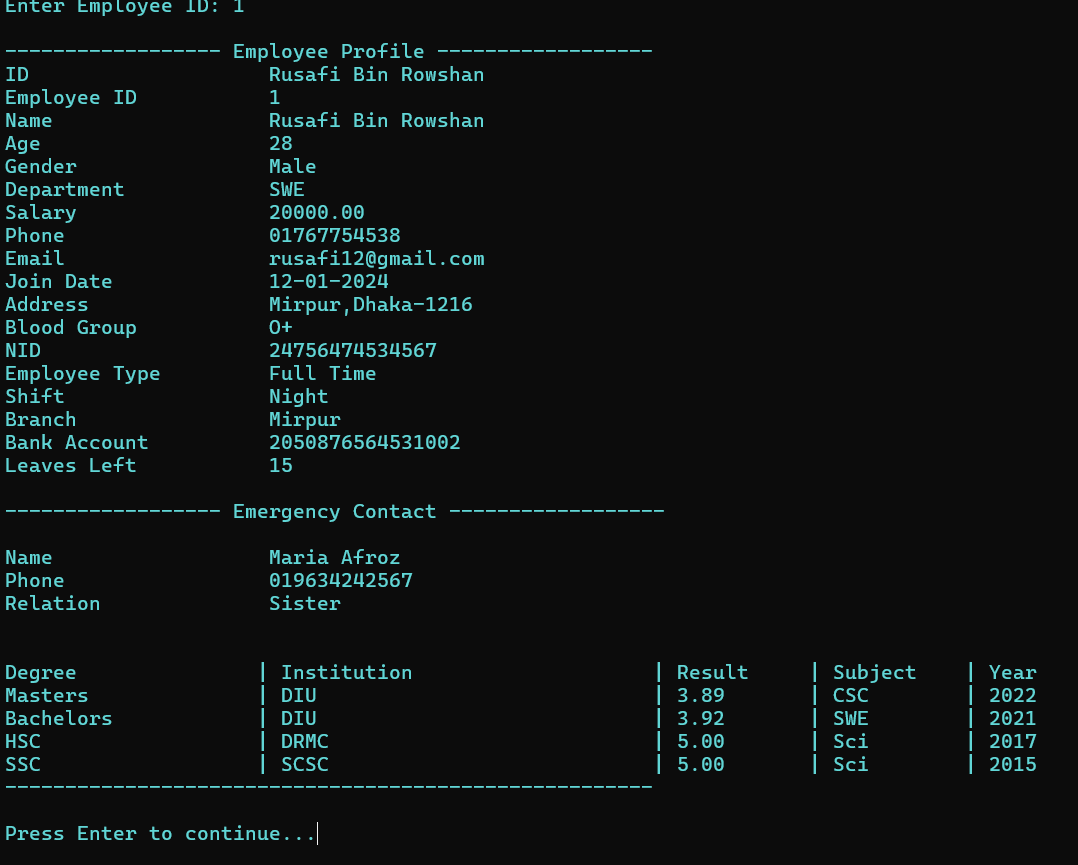
**Step 2:** The menu shows all employees and their id.

**Step 3:** If the user wants a specific employees profile, he inputs ‘y’.

**Step 4:** Then he is asked to input the ID

**Step 5:** The input ID profile is shown

**2. Search employee by ID**

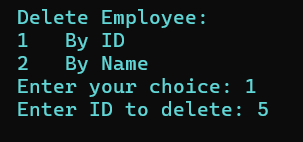
****

**Step 1:** Input 2

**Step 2:** the user is asked to input employee ID

**Step 3:** The employee profile is displayed.

**3. Delete employee**

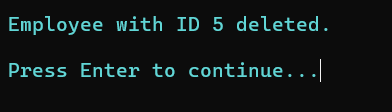
****

**Step 1:** Select the option 3 from admin meu

**Step 2:** Select by ID or By name

**Step 3:** Enter the ID or name

**Step 4:** The employee profile is deleted

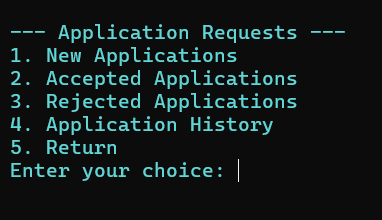


1. **View application requests**

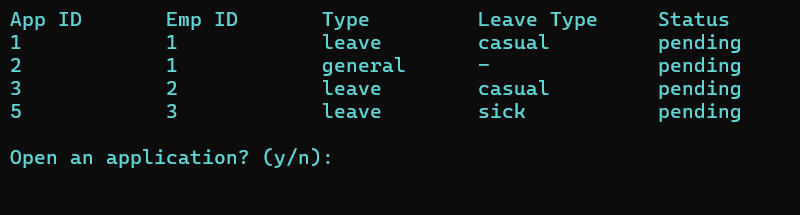
****

**Step 1:** to view application requests, select 4.

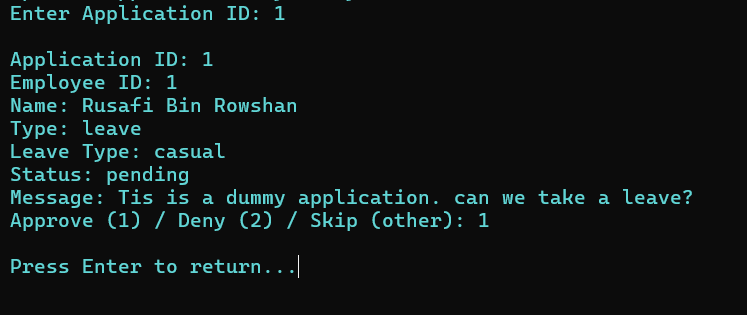
There are several options here.



To view new applications: select 1.

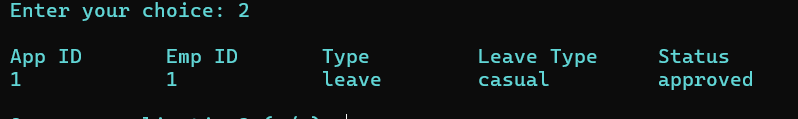


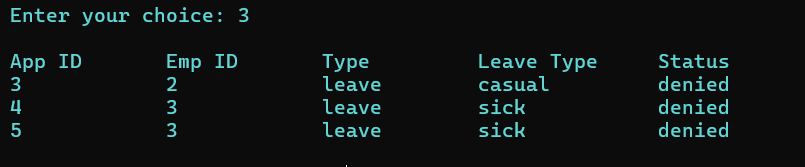
Applications can be approved or denied from here.  
Input ‘y’ to open an application and then enter application ID,



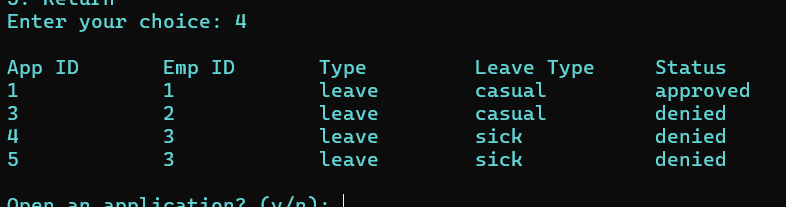
The console outputs the employee application information and the application. It can be approved or denied from here

The accepted and rejected applications are recorded in the accepted and rejected applications respectively.



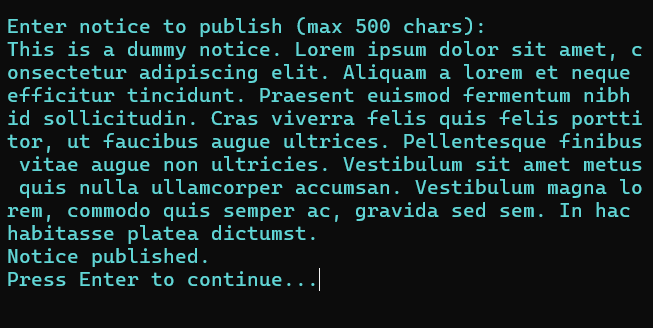


All the applications can be seen in the application history which requires the input 4 in the application menu to open

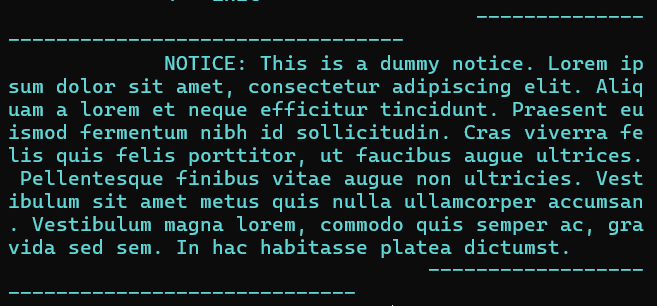


The user can return to the admin menu selecting 5

1. **Publish notice**

****

Select 5 from admin menu to enter the publish notice section. Enter desired notice and it will published which the employees will be able to view them from their profile.



1. **Logout:** Input 6 to logout to the main menu.
2. **Exit:** Select 4 to exit the system.