PAYROLL AND TIME-TRACKING SYSTEM FOR EMPLOYEES (PATTS)

"*PATTS ensures every minute counts, every salary matters*."

Cahongcoy, Rain B.
Dela Providencia, Dane Matthew F.
Duenas, Ralph Florenz F.
Galvez, Nichollo Dave A.
Mendoza, Ryo Shin P.

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

Based on recent findings from Lift HCM (2024), many companies struggle to maintain accurate time-tracking and payroll systems, leading to payroll errors, compliance issues, and decreased employee satisfaction. Connecteam (2024) reported that inaccurate information often results in underpayment or overpayment when employees fail to record their work hours or when systems are not properly established. According to Workforce (2024), manual data processing increases administrative workload and the likelihood of human error. Studies suggest that excessive or surveillance-based tracking can negatively affect employee confidence and overall performance (Calamari, 2024). The research indicates that outdated systems are often unable to adapt to flexible work schedules and changing workplace environments (Lift HCM, 2024). Findings from Connecteam (2024) also reveal that inconsistent data entry and lack of automated verification contribute to unreliable payroll processing. As highlighted by Workforce (2024), these problems place businesses at risk of financial losses, employee frustration, and potential wage and hour penalties.

According to Lift HCM (2024) a bunch of companies still use manual methods to record employee attendance and payroll, which often results in errors in time tracking, payroll, and employee monitoring. Connecteam (2024) also found that if employees forget to log their time or if systems are not well organized, it can lead to overpayment or underpayment. Additionally, Workforce (2024) stated that manual data entry increases the workload of HR or accountant and can lead to delays and human error. This makes it difficult for Admin to track who came in late, who went home early, or how a bunch of hours each employee actually worked. Because of these problems, salaries are often calculated unfairly and attendance records are inaccurate.

To solve these problems, the researchers provide a solution about inaccurate and time-consuming payroll processes.

Through C++ Language / Program. The program is designed to simplify payroll tasks through the automatic recording of employee attendance, monitoring total working hours, and calculating salaries within a single program. It begins by allowing employees to input their name, code, and work schedule. Their in and out are then marked and computed automatically to arrive at the total hours worked. The system determines if the employee was late coming in, or early leaving. When lateness or undertime is indicated, the system automatically calculates the correct deduction. The total pay or gross wage is then calculated by multiplying the number of total hours worked with the hourly wage, then subtracting deductions to arrive at net pay. The organized process, indicated in the system's flowchart, ensures that all calculations are accurate, equal, and uniform.

This program process might change the necessity of constant manual effort and paper work. It accelerates the process of payroll processing and makes it more transparent, providing a clear record to both employers and employees regarding attendance, overtime, and salary payment.

The Project

The Payroll and Time-Tracking System prompts users to select a role (Admin, Manager, Employee); Admins authenticate with a username and password to access the Admin Dashboard, which allows viewing and editing employee records, inspecting attendance logs, generating weekly payroll reports that show gross pay, deductions, and net pay and exiting to the dashboard that will ask the user again to enter their role. Managers and employees confirm whether they are logging IN or OUT, after which the system automatically records the current date and time and links the entry to the employee’s name and code; IN entries record arrival and OUT entries complete the day so

the system can compare times to the schedule, compute hours worked, and calculate deductions for lateness or undertime. All daily computations and raw attendance entries are stored in a central log and can be exported to a .txt file for administrative review and reporting.

Objectives

The purpose of this project is to design and implement an automated payroll and time-tracking system that accurately records the employees attendance, calculate the salaries based on working hours of a worker, and to minimize the human errors in manual computation/tracking to ensure the fairness and efficiency of the salary management.

1. To ensure fair and accurate compensation for the actual hours worked, it is essential that each employee's daily salary is accurately calculated based on their role, determining between managerial and regular staff positions.
2. To develop an accurate and transparent wage calculation while providing fair compensation for the employee's salary.
3. The system will automatically track work hours, and generate clear weekly reports showing gross pay, deductions, and net salary.
4. To develop an admin system that allows authorized users to manage employee records, attendance logs, and other file paths.

Flowchart of the System

This section will introduce the flowchart of Payroll and Time-Tracking System

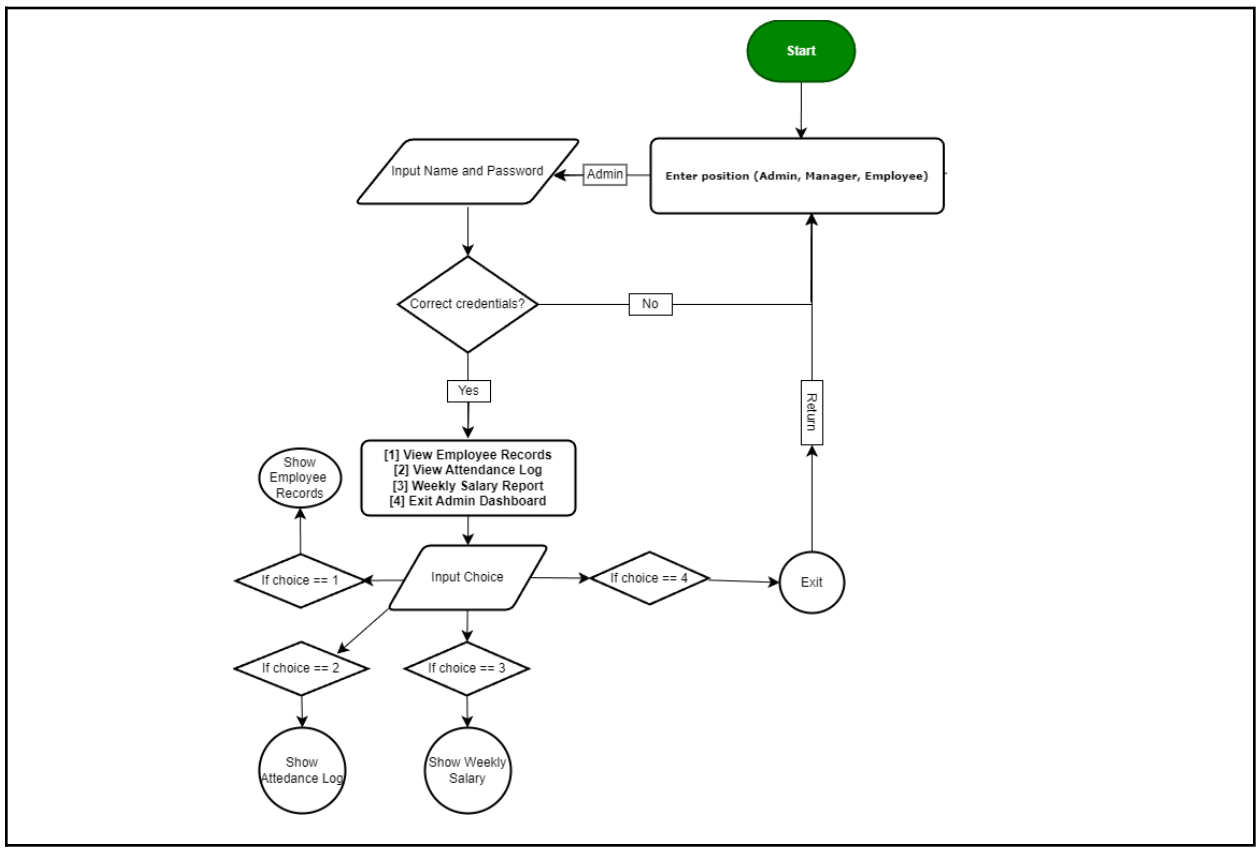

Figure 1. The Admin System

The flowchart illustrates the logical flow of the Admin System within the employee attendance and payroll program. The process begins with a Start symbol, indicating the initialization of the program. The user is first prompted to enter their position, choosing between *Admin, Manager,* or *Employee*. If the user selects *Admin*, the system proceeds to request the admin name and password for authentication. A decision symbol follows, checking whether the credentials are correct. If the credentials are invalid, the system loops back and prompts the user to re-enter the login information. If valid, the admin gains access to the Admin Dashboard, where four main options are displayed:

1. View Employee Records
2. View Attendance Log
3. Weekly Salary Report
4. Exit the Admin Dashboard

The admin inputs their desired choice, and the system processes it accordingly. If the admin chooses Option 1, the system retrieves and displays all employee records, including their names, positions, and lateness status. If Option 2 is chosen, the program shows the attendance log recorded in the text file. Selecting Option 3 displays the weekly salary report, which includes computed salaries and deductions for employees who were late. Finally, choosing Option 4 exits the dashboard, returning the flow to the initial position selection phase.

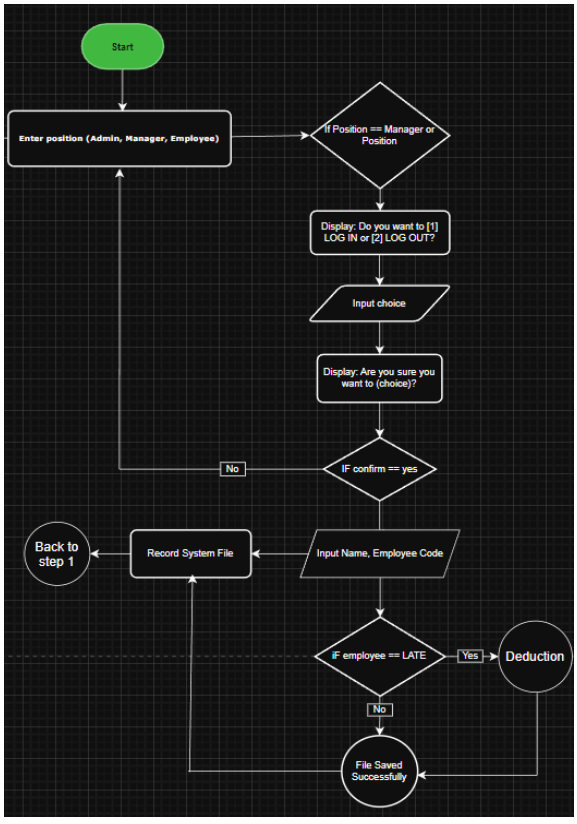

Figure 2. The Employee Log

The Employee Log Flowchart illustrates the process followed by the system when a manager or employee records their attendance. The flow starts with the Start symbol, followed by a prompt for the user to enter their position—either *Admin, Manager,* or *Employee*. If the user selects *Manager* or *Employee*, the program proceeds to the attendance process. The system then displays a prompt asking whether the user wants to log in or log out, allowing them to select between options [1] and [2]. After choosing, the user is asked for confirmation with the question, *"Are you sure you want to (choice)?"* If the user confirms with "yes," the program continues; otherwise, it loops back to the beginning. Once confirmed, the user is prompted to input their name and employee code, which ensures the system can properly identify the person logging their attendance. Afterward, the system checks whether the employee is late, comparing the current time against the scheduled start time (8:00 AM). If the employee is late, a deduction is automatically applied to their salary as a penalty. If the employee is on time, the system proceeds without deduction. Finally, the log entry is recorded in the system file, saving the details such as name, code, position, and time of login or logout. The process concludes with a confirmation message stating that the file has been saved successfully, ensuring that all attendance data is properly stored for the admin to review later.

Pseudocode

This section introduces the pseudo code of the Payroll and Time-Tracking System (PATTS), outlining the program's main processes such as recording attendance, tracking time, and computing employee salaries.

```
STRUCT Employee
   STRING name
   STRING position
   STRING empCode
   FLOAT hourlyRate
   FLOAT hoursWorled[6]
   BOOLEAN waslate
END STRUCT
```

Figure 3: Pseudo code of Structure Declaration

Figure 2 presents the definition of a structure named Employee. A structure is a data type that groups related information together. In this program, it stores details about each employee such as their name, position, employee code, hourly rate, hours worked, and a late indicator. The structure helps organize employee information efficiently, allowing the program to manage multiple employees using a single, consistent format.

```
STRUCTURE EMPLOYEE  employees[50]
INT empCount  = 0
```

Figure 4: Pseudo code of Arrays

In this figure, the arrays are declared; employees and empCount. The employees array can hold up to 50 employee records, while empCount keeps track of how many employees are currently recorded in the system. These variables are accessible by all functions, ensuring that any updates made to employee data (like attendance or salary details) are shared across the whole program.

```
FUNCTION getSystemDateTime
FUNCTION isLate(STRING currentTime)
FUNCTION adminDashboard()
FUNCTION employeeLog(STRING role)
```

Figure 5: Pseudo Code of Function Declaration

This figure lists all the functions used in the program. Each function performs a specific task:

- getSystemDateTime() retrieves the current date and time.
- isLate() checks if an employee logged in after 8:00 AM.
- adminDashboard() handles the admin login and displays management options.
- employeeLog() allows managers and employees to log in or out.

By using functions, the program becomes more organized, easier to maintain, and less repetitive.

```
MAIN
   DISPLAY "Enter position (Admin, Manager, Employee): "
   INPUT position
   IF position IS "Admin" THEN
      CALL adminDashboard()
   ELSE IF position IS "Manager" OR "Employee" THEN
      CALL employeeLog(position)
   ELSE
      DISPLAY "Invalid position entered."
      RETURN TO MAIN
   END IF
END
```

Figure 6: Pseudo code of the Main Program

In this figure, it will introduce the main program that serves as the starting point of the system. It asks the user to enter their position—whether they are an Admin, Manager, or Employee.

Based on their input, the program calls the appropriate function:

- adminDashboard() for admins, and
- employeeLog() for managers or regular employees.

If the input does not match any valid position, an error message appears. This section demonstrates how conditional statements (IF–ELSE) direct the flow of the program.

```
FUNCTION getSystemDateTime RETURNS STRING
   SET currentTime = SYSTEM CURRENT DATE AND TIME
   RETURN currentTime
END FUNCTION
```

Figure 7: Pseudo code of the system date and time

This figure assigned the getSystemDateTime function simply retrieves the current date and time from the computer's system clock. This is useful for logging when an employee signs in or out. By storing this time, the program can later determine if the employee was late.

```
FUNCTION isLate(STRING currentTime) RETURNS BOOLEAN
   EXTRACT hour = INTEGER VALUE OF currentTime[11 TO 12]
   EXTRACT minute = INTEGER VALUE OF currentTime[14 TO 15]
   IF hour > 8 OR (hour = 8 AND minute > 0) THEN
      RETURN TRUE
   ELSE
      RETURN FALSE
   END IF
END FUNCTION
```

Figure 8: Pseudo code of the late function

Figure # presents the isLate function where it checks whether an employee logged in after 8:00 AM. It takes the current time as input and extracts the hour and minute. If the hour is greater than 8 or if it is exactly 8 but the minutes are more than 0, the function returns TRUE, meaning the employee is late. Otherwise, it returns FALSE. This logical condition helps apply salary deductions for lateness.

```
FUNCTION adminDashboard
  STRING username, password, loginChoice
   INT choice

   DISPLAY "Username: "
      INPUT username
      DISPLAY "Password: "
      INPUT password

   // --- Admin Login Section ---
```

```
WHILE TRUE
    IF username = adminUsername AND password = adminPassword THEN
     DISPLAY "Welcome to Admin Dashboard!"

    DO
        DISPLAY "===== ADMIN MENU ====="
        DISPLAY "[1] View Employee Records"
        DISPLAY "[2] View Attendance Logs"
        DISPLAY "[3] Generate Weekly Salary Report"
        DISPLAY "[4] Exit to Position Selection"
        INPUT choice

       IF choice = 1 THEN
            DISPLAY "--- Employee Records ---"
            IF employeeList is EMPTY THEN
               DISPLAY "No employee records found."
            ELSE
               FOR EACH employee IN employeeList
                  DISPLAY employee.name, employee.position, employee.empCode, employee.totalHours
               ENDFOR
            ENDIF

       ELSE IF choice = 2 THEN
            DISPLAY "--- Attendance Logs ---"
            OPEN "attendance.txt"
            READ AND DISPLAY all lines
            CLOSE FILE

       ELSE IF choice = 3 THEN
            DISPLAY "--- Weekly Salary Report ---"
            FOR EACH employee IN employeeList
               IF employee.position = "Manager" THEN
                  rate ← 250
               ELSE
                  rate ← 150
               ENDIF

              salary ← employee.totalHours × rate

              IF employee.isLate = TRUE THEN
                  deduction ← salary × 0.05
                  salary ← salary − deduction
               ELSE
```

```
                deduction ← 0
            ENDIF

            DISPLAY employee.name, employee.position, rate, deduction, salary
        ENDFOR

    ELSE IF choice = 4 THEN
        DISPLAY "Returning to position selection..."
        BREAK LOOP

    ELSE
        DISPLAY "Invalid choice."
    ENDIF
  UNTIL choice = 4

ELSE
    DISPLAY "Invalid admin credentials."
ENDIF
```

Figure 10: Pseudo code of the Function Admin Dashboard

This figure presents the adminDashboard function that manages all administrative operations. It begins with a while loop that repeatedly asks for the admin's username and password until the correct credentials are entered. If the login is incorrect, the admin is given the option to either try again or exit the system.

Once logged in, the admin is shown a menu with four options:

1. View Employee Records
2. View Attendance Logs
3. View Weekly Salary Report
4. Exit Dashboard

Each option uses conditional statements and for loops to process and display the appropriate information. This function highlights both repetition structures (loops) and decision-making logic (switch cases).

```
FUNCTION employeeLog(role : STRING)
  STRING name, empCode, confirm, again
  INTEGER action
  STRING dateTime
  BOOLEAN late

  SET dateTime = getSystemDateTime()

  DISPLAY "Do you want to [1] LOG IN or [2] LOG OUT? "
  INPUT action
```

```
    DISPLAY "Are you sure? (Yes/No): "
    INPUT confirm

    IF confirm = "Yes" OR confirm = "yes" OR confirm = "Y" OR confirm = "y" THEN
        DISPLAY "Enter your name: "
        INPUT name
        DISPLAY "Enter your employee code: "
        INPUT empCode

        SET late = isLate(dateTime)

        OPEN "attendance.txt" FOR APPEND AS FILE
        IF FILE IS OPEN THEN
            WRITE "[" + dateTime + "] " + role + " " + name + " (" + empCode + ") " TO FILE
            IF action = 1 THEN
                IF late = TRUE THEN
                    WRITE "LOGGED IN - LATE (5% deduction applied)" TO FILE
                ELSE
                    WRITE "LOGGED IN on time" TO FILE
                END IF
            ELSE IF action = 2 THEN
                WRITE "LOGGED OUT" TO FILE
            ELSE
                DISPLAY "Invalid option."
                CLOSE FILE
                RETURN
            END IF

            CLOSE FILE
            DISPLAY "Attendance logged successfully."

            employees[empCount].name = name
            employees[empCount].position = role
            employees[empCount].empCode = empCode
            employees[empCount].wasLate = late
            empCount = empCount + 1

        ELSE
            DISPLAY "Error writing to attendance file."
        END IF
    ELSE
        DISPLAY "Would you like to try again? (Yes/No): "
        INPUT again
        IF again = "Yes" OR again = "yes" THEN
            CALL employeeLog(role)
        ELSE
            DISPLAY "Goodbye."
        END IF
    END IF
END FUNCTION
```

Figure 11: Pseudo-code for Employee Log

This figure allows employees and managers to log in or log out of the system.First, the program asks whether the user wants to log in or log out. After confirming the choice, it records the employee's name and code and checks whether the employee is late using the isLate() function. The current date and time are retrieved using the getSystemDateTime() function. Then, the employee's attendance record is written into the file "attendance.txt". If the employee logs in late, the system marks it and applies a 5% salary deduction later in the admin report. Additionally, the employee's information is saved inside the employees array, and the count of employees (empCount) is increased. If the user decides not to continue, the function gives an option to try again or exit. This section demonstrates function calls, conditional statements, file writing, and data storage using arrays.

Data Dictionary

The data dictionary describes all of the information used in the employee attendance and payment systems. It describes what each variable represents, the type of data it stores, and its size within the program. This section provides an in-depth understanding of how the system stores and handles data, especially when dealing with personnel records, time logs, and salary computations.

The table below is the list of all variables used in the program and is shown together with their size, data type, and description. Each variable serves a specific purpose in the system. For example, storing employee names, login details, or computing their weekly salary. This table makes the program easier to understand and makes it organized.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
1. Name	32	String	Employee's name
2. Position	24	string	Job type (Admin, Manager, Employee)
3. Employee Code	24	string	ID code
4. Hourly Worked	24	float[6]	Hours worked for 6 days (Mon–Sat)
5. Was Late	1	bool	True if the employee log in after 8:00 AM
6. Hourly Rates	4	float	Pay per hour
7. Total Hours	4	float	The total weekly hours for salary computation
8. Employees	—-	Employee [50]	List storing up employees
9. Employee Count	4	int	Number of employees in the system
10. Username	24	string	Admin login

11. Password	24	string	Admin login
12. Choice	4	int	Menu Option of the Admin
13. Action	4	int	Log in or Log out action
14. Confirm	24	string	Y/N confirmation from the user
15. Rate	4	float	Hourly rate
16. Salary	4	float	Weekly salary after calculation
17. Deduction	4	float	Salary deduction (if needed)
18. Date time	32	string	Date and time of logging in or out

Code

This section presents the code of the **Payroll and Time-Tracking System (PATTS)**. The program records employee attendance, tracks work hours, and calculates salaries automatically. It uses functions, loops, structures, and file handling to ensure smooth and efficient system operation.

```
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>
#include <cstdlib>
#include <sys/stat.h>
using namespace std;

struct Employee {
    string name;
    string position;
    string empCode;
    float hourlyRate;
    float hoursWorked[6];
    bool wasLate;
};
```

Figure 12. Libraries and Structure

The program starts with essential C++ libraries that handle input and output, file storage, string operations, time functions, and folder creation. These libraries allow the system to record attendance, display information, and manage files efficiently. After the libraries, the struct Employee is declared to store important employee details such

as name, position, code, hourly rate, and lateness status. This structure helps organize employee data clearly and allows the program to process records easily.

```
Employee employees[50];
int empCount = 0;

// Function Declarations
void adminDashboard();
void employeeLog(string role);
string getSystemDateTime();
bool isLate(string currentTime);
void ensureFolderExists();
```

Figure 13. Function Declaration

This part declares global variables and functions used throughout the program. The array employees[50] stores data for up to 50 employees. The integer empCount keeps track of how many employees have been logged. The declared functions—adminDashboard(), employeeLog(), getSystemDateTime(), isLate(), and ensureFolderExists()—are defined later in the code and handle specific operations, such as managing the admin menu, recording attendance, checking lateness, and managing folders.

```
// Main Program
int main() {
    string position;

    ensureFolderExists();

    while (true) {
        cout << "\nEnter position (Admin, Manager, Employee): ";
        cin >> position;

        if (position == "Admin" || position == "admin") {
            adminDashboard();

        } else if (position == "Manager" || position == "manager" ||
                   position == "Employee" || position == "employee") {
            employeeLog(position);
        } else {
            cout << "Invalid position entered. Try again.\n";
        }
    }

    return 0;
}
```

Figure 14. Main Program

The main() function serves as the starting point of the program. It first ensures that the “records” folder exists by calling ensureFolderExists(). Then, the program continuously prompts the user to identify their

role—whether as an **Admin**, **Manager**, or **Employee**. Based on the user's input, the program redirects to either the adminDashboard() or employeeLog() function. If the input does not match any valid role, the program notifies the user and asks again. This design ensures that only valid users can proceed with the system.

```
// Ensure the 'records' folder exists
void ensureFolderExists() {
    mkdir("records");
}
```

Figure 15 For File

This function ensures that a folder named "records" is created before storing any data. The mkdir() function (from <sys/stat.h>) creates a new directory if it doesn't already exist. This helps organize the system by saving all generated text files—such as attendance logs and salary reports—inside the same folder.

```
// Get System Date & Time
string getSystemDateTime() {
    time_t now = time(0);
    char *dt = ctime(&now);
    return string(dt);
}
```

Figure 16. Get System Date and Time

This function automatically retrieves the current date and time from the computer system. The variable now stores the current time using time(0), and the ctime() function converts it into a readable string format. This information is used to timestamp every employee log-in and log-out activity, making the attendance record accurate and reliable.

```
// Check if employee is late (after 8:00 AM)
bool isLate(string currentTime) {
    string timeStr = currentTime.substr(11, 5); // HH:MM
    int hour = atoi(timeStr.substr(0,2).c_str());
    int minute = atoi(timeStr.substr(3,2).c_str());

    if (hour > 8 || (hour == 8 && minute > 0))
        return true;
    return false;
}
```

Figure 17. Check if the employee is late

This function determines whether an employee is late by analyzing the system time. It extracts the hour and minute from the current time string and checks if the time is later than 8:00 AM. If the employee logs in after this time, the function returns `true` (late). Otherwise, it returns `false` (on time). This simple condition helps automate attendance monitoring and allows deductions in salary later.

```
// Admin Dashboard
void adminDashboard() {
    string username, password;
    int choice;

    while (true) {
        cout << "\nUsername: ";
        cin >> username;
        cout << "Password: ";
        cin >> password;

        if (username == "ryotheadmin" && password == "shin123") {
            cout << "\nWelcome to Admin Dashboard!\n";
            break;
        } else {
            cout << "Invalid credentials. Try again.\n";
            main();
        }
    }

    do {
        cout << "\n===== ADMIN MENU =====\n";
        cout << "[1] View Employee Records\n";
        cout << "[2] View Attendance Logs\n";
        cout << "[3] Weekly Salary Report\n";
        cout << "[4] Exit the Admin Dashboard\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                cout << "\n--- Employee Records ---\n";
                if (empCount == 0) {
                    cout << "No employee records yet.\n";
                } else {
                    for (int i = 0; i < empCount; i++) {
                        cout << "Name: " << employees[i].name << endl;
                        cout << "Position: " << employees[i].position << endl;
                        cout << "Employee Code: " << employees[i].empCode << endl;
                        cout << "Late Today: " << (employees[i].wasLate ? "Yes" : "No") << endl; //? is_true : is_false
                        cout << "--------------------------\n";
                    }
                }
                break;
            }

            case 2: {
                cout << "\n--- Attendance Logs ---\n";
                ifstream file("records/attendance.txt");
                if (file.is_open()) {
```

```
            string line;
            while (getline(file, line)) {
                cout << line << endl;
            }
            file.close();
        } else {
            cout << "No attendance logs found.\n";
        }
        break;
    }

    case 3: {
        cout << "\n--- Weekly Salary Report ---\n";
        if (empCount == 0) {
            cout << "No employees yet.\n";
        } else {
            for (int i = 0; i < empCount; i++) {
                float totalHours = 48; // 8 hrs/day * 6 days
                float rate = (employees[i].position == "Manager" ? 250 : 150);
                float salary = totalHours * rate;
                float deduction = 0.0;

                if (employees[i].wasLate) {
                    deduction = salary * 0.05;
                    salary -= deduction;
                }

                cout << "Name: " << employees[i].name
                     << " | Position: " << employees[i].position
                     << " | Weekly Salary: PHP" << salary
                     << " | Deduction (if late): PHP" << deduction << endl;
            }
        }
        break;
    }

    case 4:
        cout << "Exiting the dashboard...\n";
        return;

    default:
        cout << "Invalid option. Try again.\n";-


  }

} while (choice != 4);
}
```

Figure 18. Admin Dashboard

The adminDashboard() function provides administrative control of the system. It begins with a login prompt for the admin's username and password to ensure security. Once authenticated, the admin can view employee records, attendance logs, and weekly salary reports. Employee records display personal information and lateness status, attendance logs show all time-ins and time-outs, and salary reports calculate total pay with deductions if an employee was late. This dashboard acts as the control center of the entire system, enabling oversight and accountability.

```
// Employee Log Function
void employeeLog(string role) {
    string name, empCode, confirm;
    int action;
    string dateTime = getSystemDateTime();

    cout << "\nDo you want to:\n[1] LOG IN\n[2] LOG OUT\nEnter choice: ";
    cin >> action;

    cout << "Are you sure? (Yes/No): ";
    cin >> confirm;

    if (confirm == "Yes" || confirm == "yes" || confirm == "Y" || confirm == "y") {
        cin.ignore();
        cout << "Enter your name: ";
        getline(cin, name);
        cout << "Enter your employee code: ";
        getline(cin, empCode);

        bool late = isLate(dateTime);

        // Record in system
        employees[empCount].name = name;
        employees[empCount].position = role;
        employees[empCount].empCode = empCode;
        employees[empCount].wasLate = late;
        empCount++;

        // Log attendance in file
        ofstream file("records/attendance.txt", ios::app);
        if (file.is_open()) {
            file << "Time: [" << dateTime.substr(0, dateTime.size() - 1) << "] | "
                 << "Name: " << name << " | " << " Role: " << role  << " | " << "Code: " << empCode << " | "
                 << (action == 1 ? (late ? "LOGGED IN - LATE\n" : "LOGGED IN - On Time\n")
                                 : "LOGGED OUT\n") << "-----------------------------------------------\n";
            file.close();
        } else {
            cout << "Error writing attendance file.\n";
        }
        cout << "\nAttendance Recorded Successfully!" << endl;
        return;
    }
}
```

Figure 19. Employee Log

The employeeLog() function allows employees and managers to log their attendance. Users select whether they want to log in or out, confirm their action, and then input their name and employee code. The system automatically records the current date and time and checks for lateness. All information is stored in the attendance.txt file under the "records" folder. This process ensures that each attendance record is accurate, timestamped, and permanently stored for administrative review.

Results and Discussion

This section will introduce the results and discussion of the code, this includes the output of the code and discuss how it works.

Figure 20. Company Roles

The system will ask the users to enter their position either **Admin, Manager, Employee.** This input stage is designed to identify the user's role in the system.

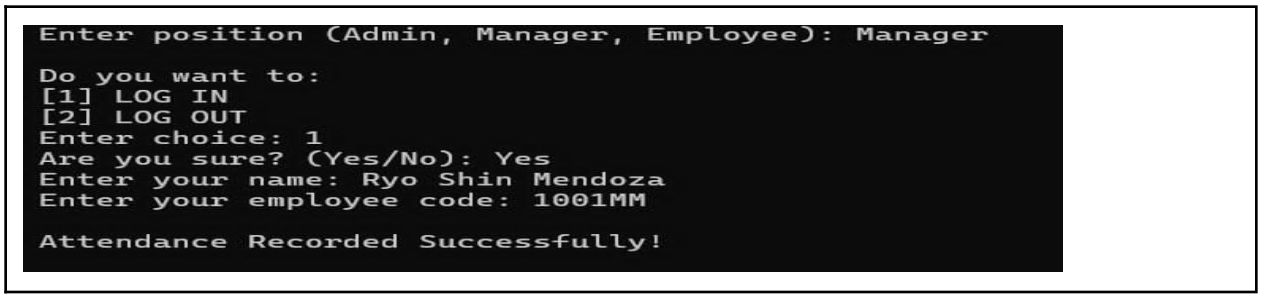

Figure 21. Manager/Employee Log

If the user chooses between Manager and Employee, the Log In/Out section will appear in the system, where the users will choose if they want to [1] LOG IN or [2] LOG OUT. After the users choose they need to input their Name and Employee Code

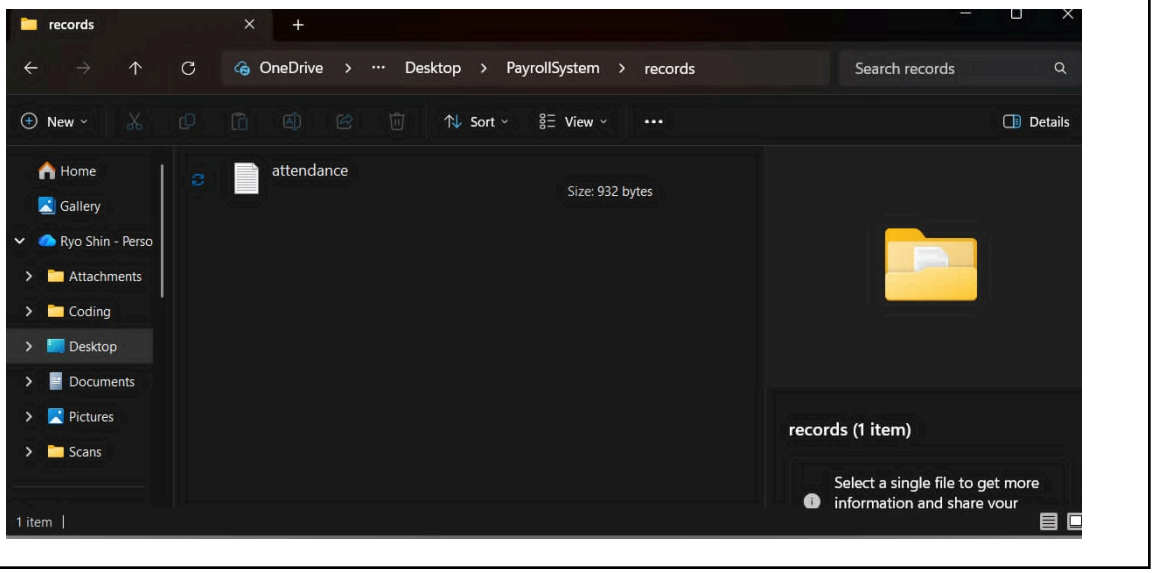

Figure 22. Attendance Log File

After doing the Attendance Log the program will record their response in a different file path (.txt) then it can view it in Notepad.

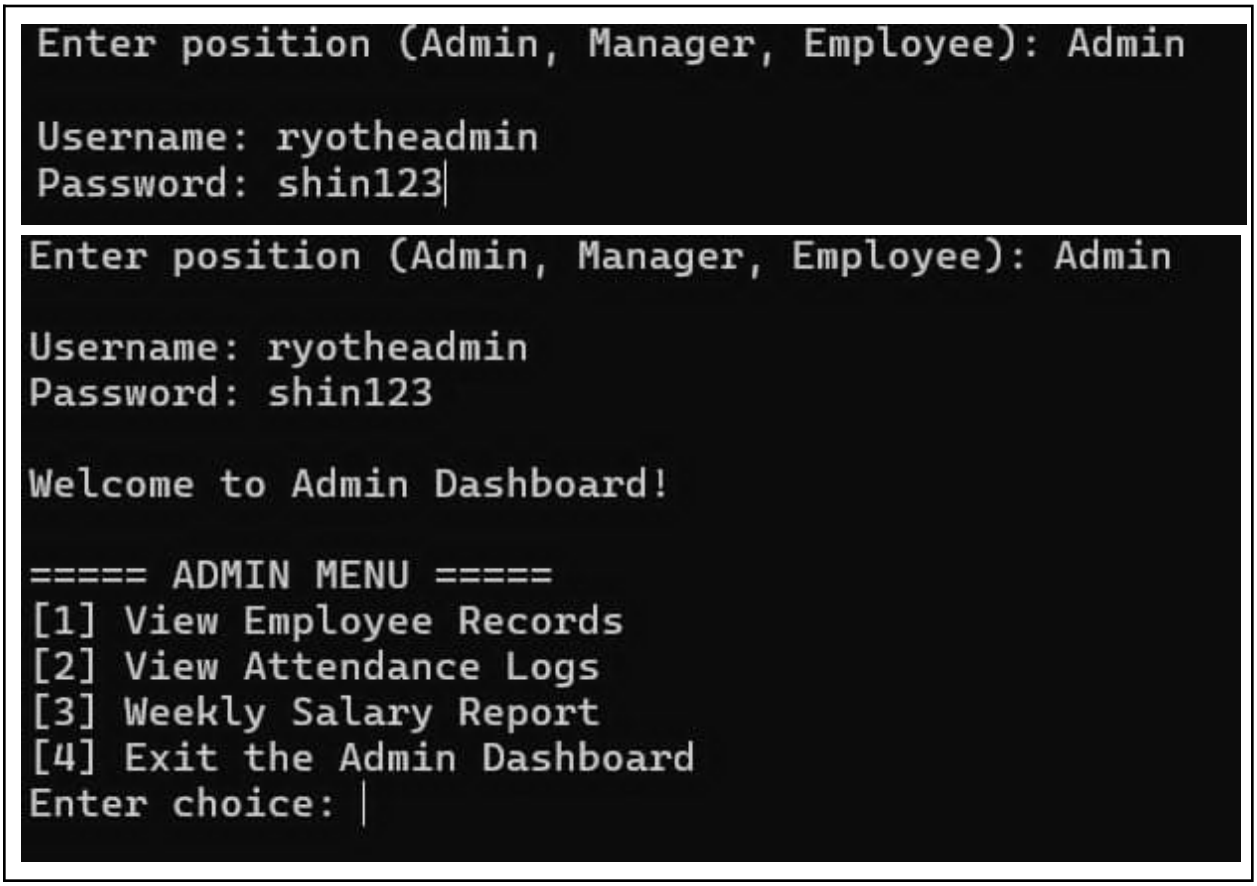

Figure 23. Admin Log

After that, If the users chose “Admin” the system will ask for the username, it identifies the specific account within the admin role while the password serves as a security to ensure that the users entering the username are authorized to access the account. This is used to authenticate the user attempting to log in as an Admin, once it’s done it will proceed to “Admin Menu” where the admin can view the Attendance Log, Weekly Salary, and Employee Records.

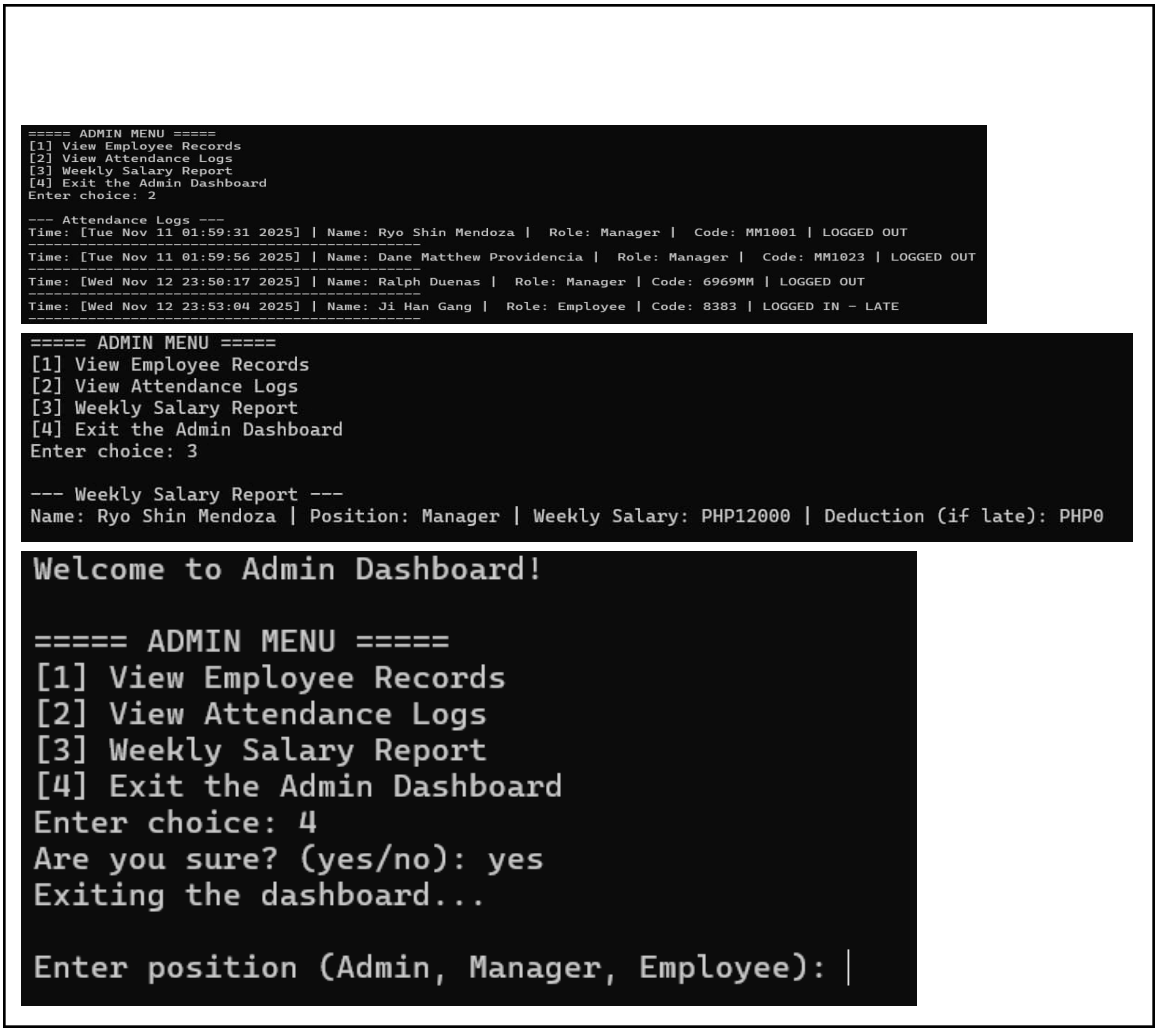

Figure 24. Admin Dashboard

If the Admin chooses the [1] the Employee Record will show the details of the Employees such as “Name, Position, Employee Code, and if they’re Late or Not”. If the Admin chooses the [2] it will proceed to the Attendance Log where the user can view the Date and Time Log. If the Admin chooses the [3] the admin can view their Weekly Salary Report, with info’s and the Deduction rate of 5% if Late. If the Admin chooses the [4] then it will Exit the Admin Dashboard and go back to the main program where the system will ask for the position.

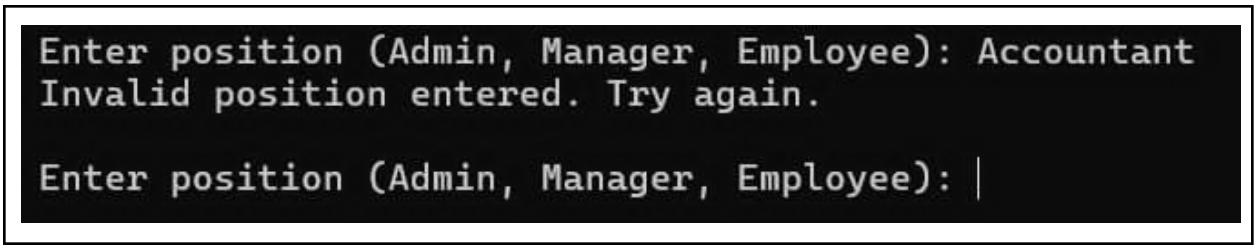

Figure 25. Invalid Position

If the user inputs a Wrong Spelling or Invalid Role, the system will ask again to enter a position.

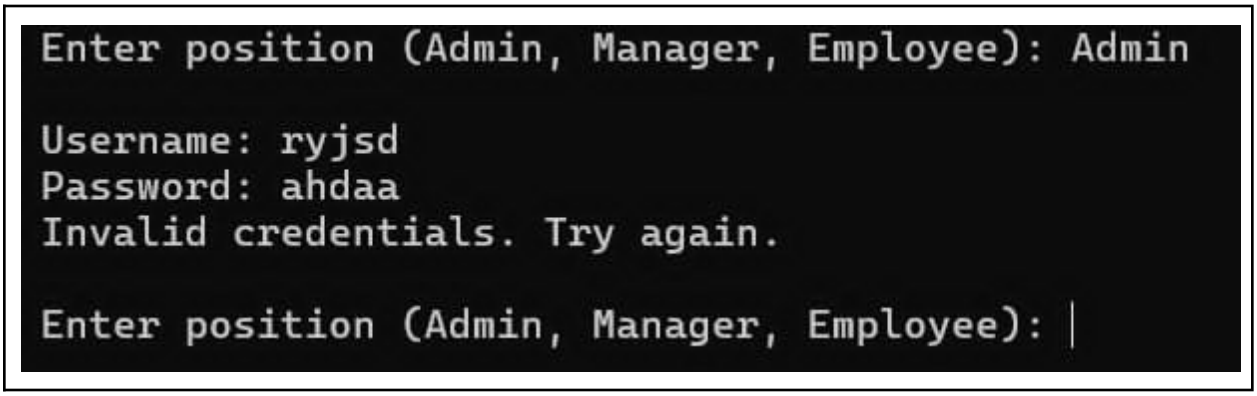

Figure 26. Invalid Credentials

If the user admin enter a wrong username and wrong password the program will be Invalid Credentials, and the system will go back to ask for the position

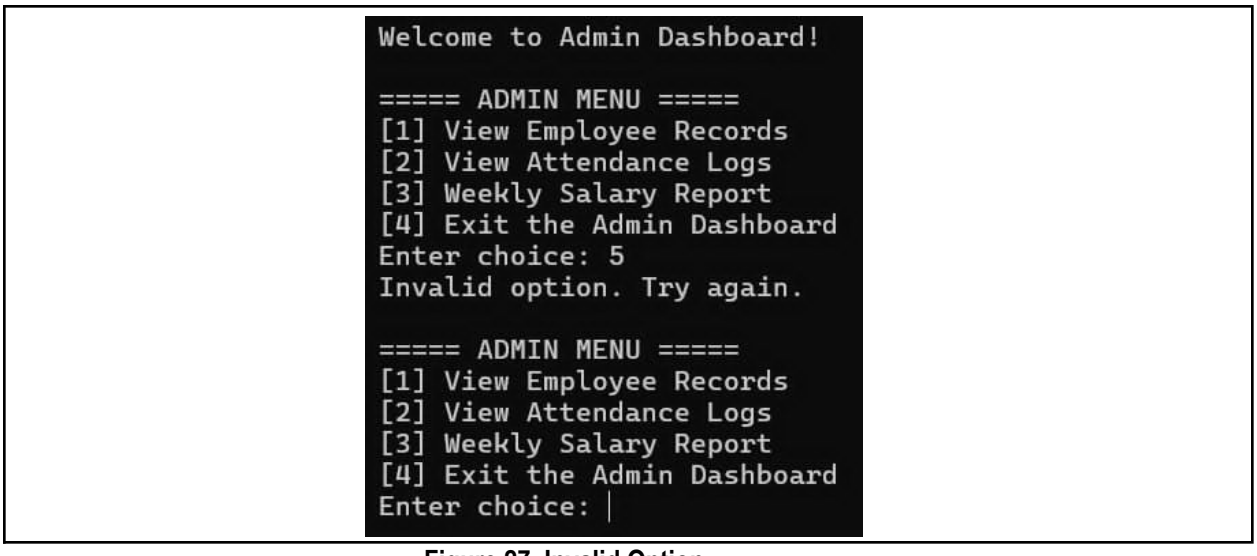

Figure 27. Invalid Option

In the Admin Dashboard if the admin chooses the Option that's not included or mistouch the keys, the system will go back to ask again what option they'll choose.

Conclusion

To conclude, this documentation paper aims to provide a company a convenient way of logging the employees time in and time out with less difficulties. By this system, it would give a service that can cater the companies need in automation that could lessen the manual calculations when it comes to their attendance and payroll. Hence, our team simplifies this system by using C++ as our programming language. This helps us also in computing the working hours and its salaries faster, efficiently and accurately not only for the employees but for the employers as well. Overall, if this system works, the company can maintain the accuracy and can easily track the mismatched schedule to the employee's attendance. With that it reduces stress for both the employees and employers.

Recommendation

- This system is a program in which it can handle computations in employees payroll and attendance. Moreover, this documentation paper contains how we managed to build a system with teams cooperation in regarding building a programmed system that aims to help a company to run faster and smoother. The teams recommendation to make this more innovative functional and effective are the following:

- New programmers should try to add a UI interface in the system that has a function to interact with the employees to the device system used. With this additional innovation the system will be more advanced in technology leading to a high-tech system.

- The next programmers should add an output monitor to display the attendance log after the data has been inputted. By adding monitor output to the system, employees can easily view their attendance and payroll sheet without having to do it manually.

- The next programmers in this system should consider putting a deduction percentage for the computation in payroll to automatically deduct the taxes. Putting an automatic deduction for taxes can help the company acquire the tax needed to comply in every payroll.

- Since this system focuses on giving the company a faster, more efficient and accurate task in computing the payroll and attendance they should think about adding a role for the workers. By implementing this the company will have an extra work by the employees without hiring a new ones.

References

Calamari. (2024). *When time tracking backfires: Challenges and pitfalls.* Retrieved from. https://www.calamari.io/blog/when-time-tracking-backfires-challenges-and-pitfalls

Connecteam. (2024). *Employee time tracking challenges.* Retrieved from https://connecteam.com/e-employee-time-tracking-challenges

Lift HCM. (2024). *Top issues with employee time tracking software.* Retrieved from https://lifthcm.com/article/top-issues-with-employee-time-tracking-software

W3schools (n.d) https://www.w3schools.com/

Workforce. (2024). *The pros and cons of the most common timekeeping systems for your business.* Retrieved from https://workforce.com/news/the-pros-and-cons-of-the-most-common-timekeeping-systems-for-your-business

+