PROJECT REPORT
ON
"PAYROLL MANAGEMENT SYSTEM"

Submitted in partial fulfillment of the

Requirement for the award of the degree of

Bachelor of Computer Application

Submitted To:
Dr. Dimple Chawla
Assistant Professor
VSIT

Submitted By:
Pranav Chamoli
10117702020

(Batch: 2020-2023)

Vivekananda Institute of Professional Studies – Technical Campus

(Affiliated to Guru Gobind Singh Indraprastha University)

CERTIFICATE

This is to certify that I, Pranav Chamoli of BCA 5th Semester from Vivekananda Institute of Professional Studies, Delhi has presented this project work entitled "PAYROLL MANAGEMENT SYSTEM" in partial fulfillment of the requirements for the award of the degree of Bachelor of Computer Applications under our supervision and guidance.

Dr. Dimple Chawla
Assistant Professor

ACKNOWLEDGEMENT

It is our proud privilege to express our profound gratitude to the entire management of Vivekananda Institute of Professional Studies and teachers of the institute for providing us with the opportunity to avail the excellent facilities and infrastructure. The knowledge and values inculcated have proved to be of immense help at the very start of my career. Special thanks to Hon'ble Founder, Vivekananda Institute of Professional Studies, Delhi for having provided us an excellent infrastructure at VSIT.

I am grateful to Prof. (Dr.) Supriya Madan (Dean, VSIT), and "project guide" for their astute guidance, constant encouragement and sincere support for this project work.

Sincere thanks to all my family members, seniors and friends for their support and assistance throughout the project.

Pranav Chamoli

(10117702020)

Table of Contents

		Page No.
1	**INTRODUCTION**	6
	1.1 Objective of the System	6
	1.2 Justification and need for the system	6
	1.3 Advantage of the system	7
	1.4 Previous work or related systems, how they are used.	7
2	**REQUIREMENT ANALYSIS**	9
	2.1 Analysis Study	9
	2.2 User Requirements	9
	2.3 Discussion with IT Experts	10
3	**DESIGN OF THE SYSTEM**	11
	3.1 Hardware, Software requirements	11
	3.2 Use Case Diagram	11
	3.3 ER Diagram of the system	12
	3.4 DFDs/Algorithms/Flow Charts	13
4	**IMPLEMENTATION & CODING**	16
	4.1 Operating System	16
	4.2 Languages	16
	4.3 S/W Tools	16
	4.4 Coding	17
5	**TESTING & TEST RESULTS**	34
	5.1 Software Testing and Objective of Testing	34
	5.2 Sample test data/ Output screen printouts etc.	38
6	**CONCLUSION**	42
	6.1 Conclusion	42
	6.2 Future Scope	42
	Bibliography	44

Table of Figures

Fig. 1 Actor and Use case	**11**
Fig. 2 Interaction between user and admin	**12**
Fig. 3 ER diagram	**13**
Fig. 4 DFD level 0	**14**
Fig. 5 DFD level 1	**14**
Fig. 6 Login Page	**29**
Fig. 7 Login Successful pop up	**29**
Fig. 8 Home page	**30**
Fig. 9 Adding Records	**31**
Fig. 10 Display Records	**31**
Fig. 11 Deleting a record	**32**
Fig. 12 Editing a record	**32**
Fig. 13 Printing all records	**33**
Fig. 14 Printing a single record	**33**
Fig. 15 Verify successful login	**38**
Fig. 16 Verify invalid login	**39**
Fig. 17 Verify invalid login with empty fields	**39**
Fig. 18 Adding new record test case	**40**
Fig.19 Deleting a record test case	**41**
Fig. 20 Updating a record test case	**42**

Chapter 1

INTRODUCTION

Payroll management is a crucial aspect of any business as it involves the calculation and distribution of employee paychecks, as well as tracking and reporting on taxes and other deductions. A payroll management system automates these processes, reducing the potential for errors and saving time for businesses.

The system developed in this project is designed to help businesses manage their payroll effectively and efficiently. It was developed using the Python programming language and the Tkinter package, which is a standard Python library for creating graphical user interfaces. The use of Python and Tkinter allows for a user-friendly interface and easy integration with other systems.

The goal of this project is to create a payroll management system that is easy to use and provides all the necessary features for businesses to manage their payroll effectively. This report will discuss the research methods used to gather information for the project, the process of developing the system, and the results and benefits of the final product.

1.1 OBJECTIVE OF THE SYSTEM

The objective of this project is to develop a payroll management system using the Python programming language and the Tkinter package. The system will automate the process of calculating and distributing employee paychecks, as well as tracking and reporting on taxes and other deductions. The main goals of the project are:

1. To create a user-friendly interface that allows businesses to easily manage their payroll.
2. To provide all the necessary features for businesses to manage their payroll effectively.
3. To automate the process of calculating and distributing employee paychecks, reducing the potential for errors and saving time for businesses.
4. To automate the process of tracking and reporting on taxes and other deductions, reducing the potential for errors and saving time for businesses.
5. To make the system easy to integrate with other systems, such as accounting or human resource management software.
6. To provide a solution that is easy to use and understand for the end-users.
7. To make the system customizable according to the requirements of the organization.

1.2 Justification and need for the system

The justification and need for a payroll management system are rooted in the fact that managing payroll can be a time-consuming and error-prone process for businesses. Manually calculating salaries, keeping track of employee information, and generating salary statements can be a tedious and complex task that requires a significant amount of time and resources.

A payroll management system automates these processes, making it easier and more efficient for businesses to manage employee payroll. By using a system, businesses can save time and reduce errors by automating tasks such as calculating salaries, tracking employee information, and generating salary statements.

Additionally, the system provides a secure way of storing employee data and can help to protect sensitive information. It also allows the business to easily access and manage employee information, making it easier to run reports, compliance checks and other actions.

1.3 Advantages of the system

The payroll management system developed using Python and Tkinter offers several advantages, including:

1. Automation of Payroll Processes: The system automates the process of calculating and distributing employee paychecks, as well as tracking and reporting on taxes and other deductions. This reduces the potential for errors and saves time for businesses.
2. User-Friendly Interface: The system has a user-friendly interface that is easy to use and understand for end-users. This makes it easy for businesses to manage their payroll effectively.
3. Customizable: The system can be customized to meet the specific requirements of an organization.
4. Cost-effective: The system reduces the need for manual labor, which can save businesses money on labor costs.

Overall, the system offers a comprehensive solution for managing payroll processes that can help businesses save time and money, while also improving their overall payroll accuracy and compliance.

1.4 Previous work or related systems, how they are used

Previous work or related systems to a payroll management system include various software solutions that automate the process of managing employee payroll. These systems can be broadly classified into two categories:

Standalone payroll software: These are software solutions that are designed specifically for payroll management. They typically include features such as the ability to calculate salaries, track employee information, and generate salary statements. Examples of standalone payroll software include QuickBooks, ADP, and Sage 50.

HR Management Systems: These are software solutions that are designed to manage various aspects of human resources, including payroll management. They typically include features such as the ability to track employee information, manage employee benefits, and generate reports.

Examples of HR management systems include Workday, SAP SuccessFactors, and Oracle HCM Cloud.

These systems are typically used by businesses of all sizes to automate the process of managing employee payroll. They can be used to calculate salaries, track employee information, and generate salary statements, as well as other tasks such as compliance checks and reports.

Additionally, these systems can also be integrated with other systems such as accounting systems, and can be accessed remotely using the internet. Some systems also provide mobile access which allows employees to access their information and perform certain actions using their mobile devices.

Chapter 2

REQUIREMENT ANALYSIS

Before we begin a new system, it is important to study the system that will be improved or replaced (if there is one). We need to analyze how this system uses hardware, software, network and the people resources. Thus, we should document how the information system activities of input, processing, output, storage and control are accomplished.

2.1 ANALYSIS STUDY

The analysis study for this payroll management system project was conducted to gather information on the current payroll processes of a business and identify areas for improvement. The following steps were taken to conduct the analysis study:

1. Research Questions: The research question for this analysis study was to evaluate the current payroll processes of the business and identify areas for improvement.
2. Data Collection: Data was collected from the current payroll system used by the business, as well as from employees and management. Information was gathered on the number of employees, pay frequencies, taxes, deductions, and any other relevant information. Surveys and interviews were conducted with employees and management to gather additional information.
3. Data Analysis: The collected data was analyzed to identify patterns, trends, and areas for improvement. The analysis revealed that the current payroll system was time-consuming and prone to errors. It also revealed that the system was not user-friendly and was difficult for employees to navigate.
4. Identification of key issues: The key issues with the current payroll system were identified as inefficiencies, inaccuracies, and compliance issues.
5. Recommendations for improvement: Based on the analysis, recommendations for improvement were made. These included automating the payroll process, implementing a user-friendly interface, and integrating the system with other systems such as accounting or human resource management software.

2.2 USER REQUIREMENTS

1. Cross-platform compatibility: The system should be compatible with multiple operating systems such as Windows, Mac and Linux.
2. Data storage: The system should have the capability of storing the data in an efficient manner, for example, using a database such as MySQL or SQLite.
3. Tkinter library compatibility: The system should be compatible with the Tkinter library, which is the standard Python library for creating graphical user interfaces.
4. Python script execution: The system should be able to execute Python scripts to automate the process of calculating and distributing employee paychecks, as well as tracking and reporting on taxes and other deductions.

5. Python version compatibility: The system should be compatible with the latest version of Python, in order to take advantage of the latest features and updates.

2.3 Discussion with IT experts

Creating an IT project for a beginner can at times become a challenging task. So, the discussion with the veterans in the field of IT becomes an important task which might lead to some great benefits for the developer. Some IT developers might consider it as a time-wasting process but, they would be missing out on a very important lesson ignoring this step. The people who are in the IT field for a long time knows the mindset of the user pretty well and might help in giving some important pointers which in turn would help in improvement of the project. Following were the outcome of the discussion held with our IT experts:

1. User friendly template was taken
2. More functionality was added

So, it was a very eventful and important step taken in the development of the project which leads to some interesting improvements in the project.

Chapter 3

DESIGN OF THE SYSTEM

3.1 Software requirements

Platform	Platform Independent
The Operating System	Windows, iOS, Linux
Framework	Python 3.1 or later versions
Backend	SQLite Database

3.2 Hardware Requirements

Processor	A minimum of a 1 GHz processor
RAM	Minimum 2GB RAM
Graphics	Integrated graphics card
Hard Disk	Minimum 256 GB

3.3 THE USE CASE APPROACH

A use case is a set of scenarios that describe an interaction between a user and a system.

A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

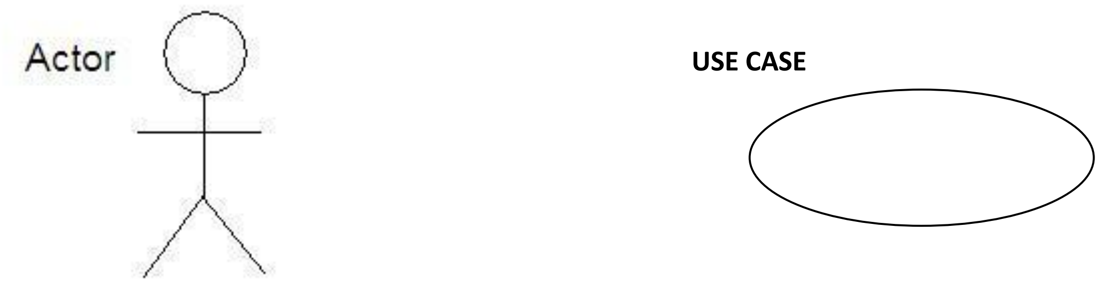


Fig. 1 Actor and Use case

An actor is representing a user or another system that will interact with the system you are modelling.

A use case is an external view of the system that represents some action the user might perform in order to complete a task.

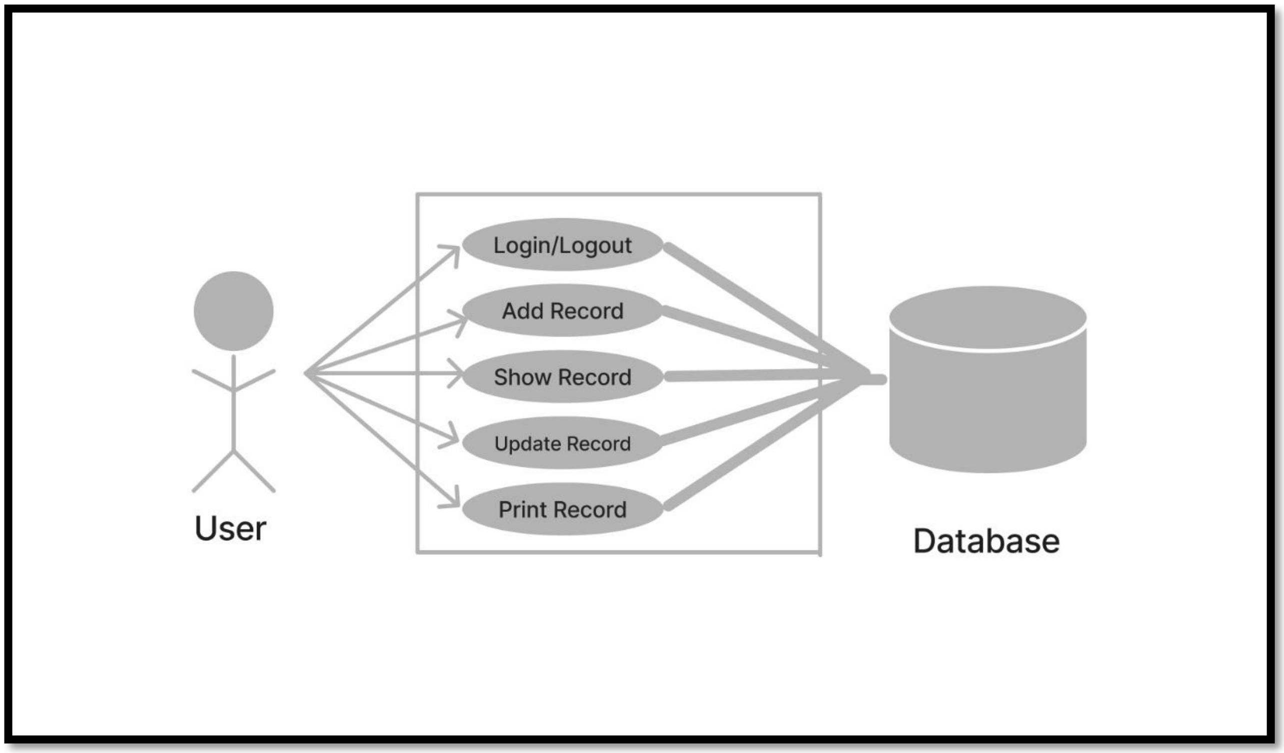


Fig. 2 Interaction between user and database

In The above diagram there is one actor or user who uses the system. This user is as follows:

1. Admin-The admin can login and logout of the system. Other than that, the admin of the system manages all the modules such as add an record, delete an record, update an record or print records.

3.4 ER DIAGRAM

An ER diagram (Entity Relationship diagram) is a graphical representation of entities and their relationships. It is often used to represent the data model of a system, including the entities, attributes, and relationships that make up the system.

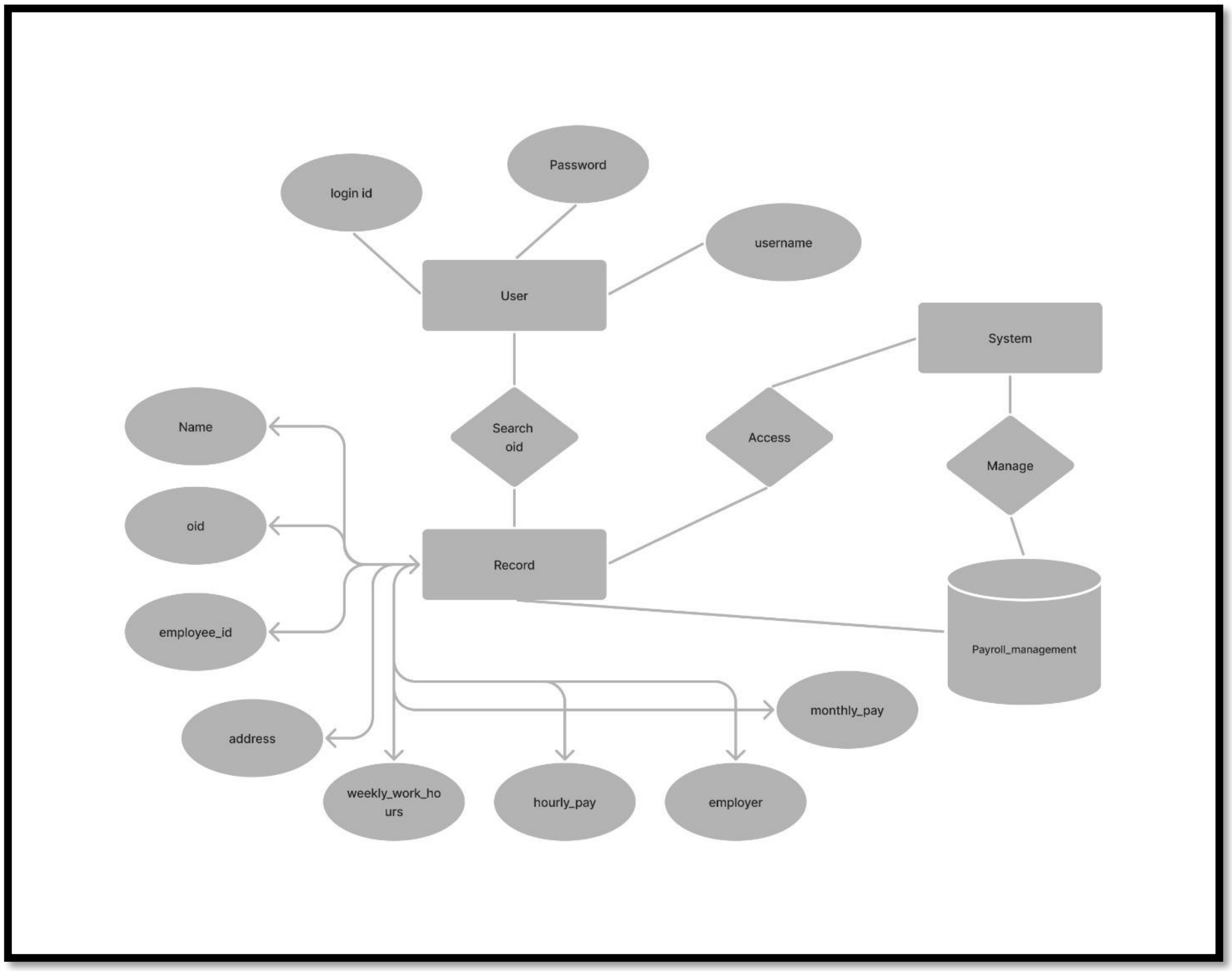


Fig. 3 ER diagram

3.5 DATA FLOW DIAGRAM (DFD)

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. Often, they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

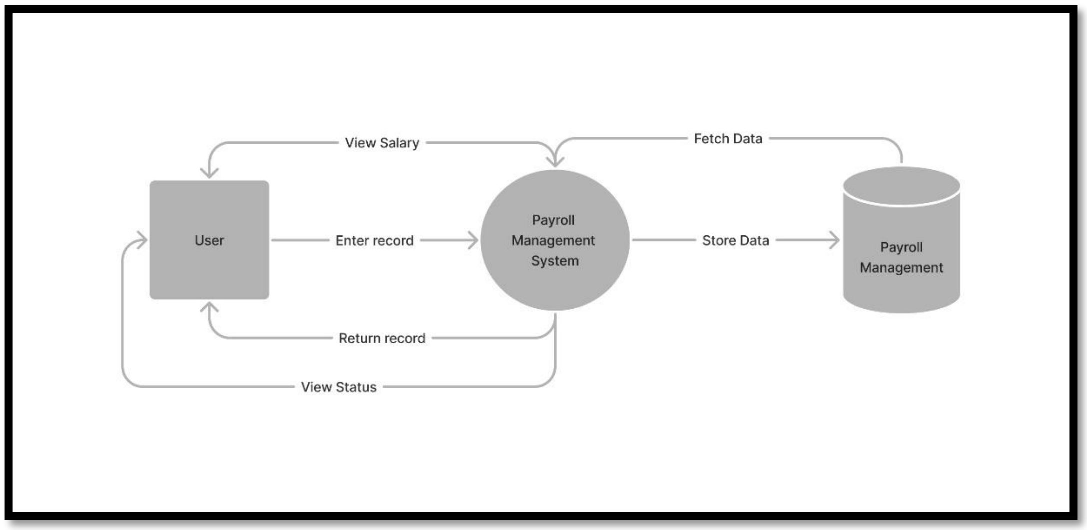


Fig. 4 DFD level 0

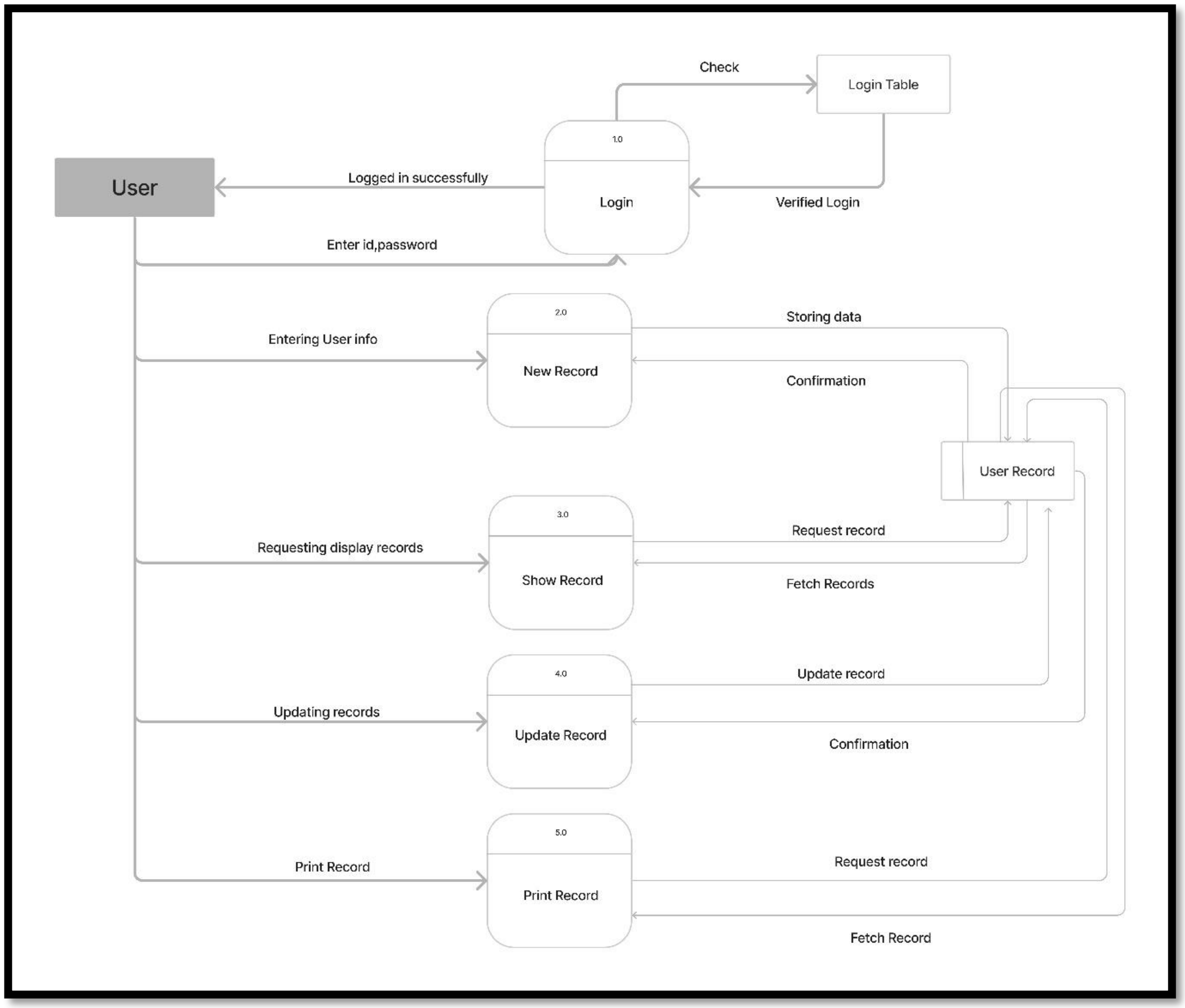


Fig.5 DFD level 1

A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

Chapter 4

IMPLEMENTATION AND CODING

4.1 OPERATING SYSTEM

Platform Independent: Since the project is done completely in Python, it also executes main properties of language. The application is platform independent. So, the client systems may have vista, Linux, Mac or any other operating system, but they can run the application easily without any dependencies of OS.

4.2 Languages used

Python is a high-level, interpreted, and general-purpose programming language that is widely used for web development, data analysis, artificial intelligence, and scientific computing. It was first released in 1991 by Guido van Rossum and has since become one of the most popular programming languages in the world. One of the main advantages of Python is its readability and ease of use. The language has a simple and straightforward syntax that makes it easy to learn and use. Python also has a large and active community of developers who have created a vast collection of libraries and frameworks that can be used to perform a wide range of tasks.

4.3 S/W Tools

VS Code

Visual Studio Code (VS Code) is a source code editor developed by Microsoft for Windows, Linux, and macOS. It is a free and open-source tool that is commonly used by developers to write and debug code. VS Code is built on top of the Electron framework, and it supports a wide range of programming languages including Python, C++, C#, Java, and JavaScript. It also has a built-in terminal, integrated Git control, and support for debugging and profiling. It also has a wide range of extensions that can be installed to add functionality such as linting, code completion, and debugging.

4.4 CODING

```
import tkinter
from tkinter import messagebox
from tkinter import *
import sqlite3
from tabulate import tabulate
from fpdf import FPDF

window = tkinter.Tk()
```

```
window.title("Login form")
window.geometry('340x440')
window.configure(bg='#333333')


def login():
    username = "admin"
    password = "admin"
    if username_entry.get() == username and password_entry.get() == password:
        messagebox.showinfo(title="Login Success",
                            message="You successfully logged in.")
        main()
    else:
        messagebox.showerror(title="Error", message="Invalid login.")


frame = tkinter.Frame(bg='#333333')

# Creating widgets
login_label = tkinter.Label(
    frame, text="Login", bg='#333333', fg="#FF3399", font=("Arial", 30))
username_label = tkinter.Label(
    frame, text="Username", bg='#333333', fg="#FFFFFF", font=("Arial", 16))
username_entry = tkinter.Entry(frame, font=("Arial", 16))
password_entry = tkinter.Entry(frame, show="*", font=("Arial", 16))
password_label = tkinter.Label(
    frame, text="Password", bg='#333333', fg="#FFFFFF", font=("Arial", 16))
login_button = tkinter.Button(
    frame, text="Login", bg="#FF3399", fg="#FFFFFF", font=("Arial", 16),
command=login)

# Placing widgets on the screen
login_label.grid(row=0, column=0, columnspan=2, sticky="news", pady=40)
username_label.grid(row=1, column=0)
username_entry.grid(row=1, column=1, pady=20)
password_label.grid(row=2, column=0)
password_entry.grid(row=2, column=1, pady=20)
login_button.grid(row=3, column=0, columnspan=2, pady=30)
```

```
def main():
    root = Tk()
    root.title('Payroll Management System')

    label = Label(root, text="Payroll Management System",
                  bg="White", fg="Black", font="Arial")
    label.grid(row="0", column=1)
    root.geometry('400x400')
    root.resizable(False, False)

    # root.configure(bg='#333333')
    # creating the database
    conn = sqlite3.connect('payroll_management.db')
    # creating cursor
    c = conn.cursor()

    # c.execute("""CREATE TABLE payroll (
    #     name text,
    #     employee_id integer,
    #     employer text,
    #     address text,
    #     weekly_work_hours integer,
    #     hourly_pay integer,
    #     monthly_pay integer
    #     )""")

    def get_monthly_pay(weekly_work_hours, hourly_pay):
        weekly_work_hours = int(weekly_work_hours)
        hourly_pay = int(hourly_pay)

        overtime_hours = (weekly_work_hours - 40)
        overtime_pay = overtime_hours * hourly_pay * 1.5
        gross_pay = hourly_pay * weekly_work_hours

        if overtime_hours > 0:
            gross_pay = hourly_pay * weekly_work_hours + \
                overtime_pay - overtime_hours * hourly_pay

        tax = gross_pay * 4 * 0.2
        return (gross_pay * 4) - tax

    # creating edit window
    def editing_window():
        editing = Tk()
        editing.title('Select ID')
        editing.geometry('400x300')
        select_box_label = Label(editing, text="Select ID")
        select_box_label.grid(row=0, column=0, pady=5)
        select_box = Entry(editing, width=30)
        select_box.grid(row=0, column=1, padx=20, pady=5)
        # creating update button
        editing_btn = Button(editing, text="Edit Record",
                             command=lambda: edit())
        editing_btn.grid(row=1, column=0)
        # creating edit function

        def edit():
            global editor
            editor = Tk()
            editor.title('Update A Record')
            editor.geometry('400x300')
            # creating the database
            conn = sqlite3.connect('payroll_management.db')
            # creating cursor
            c = conn.cursor()

            record_id = select_box.get()
            c.execute("SELECT * FROM payroll WHERE oid = " + record_id)
            records = c.fetchall()

            print(tabulate(records, headers=["Name", "Emp ID", "Employer",
"Address", "Weekly Work Hours",
                "Hourly Pay", "Net Payable Salary", "Primary Key"],
tablefmt="fancy_grid"))

            # creating global variables
            global name_editor
            global employee_id_editor
            global employer_editor
            global address_editor
            global weekly_work_hours_editor
            global hourly_pay_editor

            # creating text boxes
            name_editor = Entry(editor, width=30)
            name_editor.grid(row=0, column=1, padx=20, pady=(10, 0))
            employee_id_editor = Entry(editor, width=30)
            employee_id_editor.grid(row=1, column=1, padx=20)
            employer_editor = Entry(editor, width=30)
            employer_editor.grid(row=2, column=1, padx=20)
            address_editor = Entry(editor, width=30)
            address_editor.grid(row=3, column=1, padx=20)
            weekly_work_hours_editor = Entry(editor, width=30)
            weekly_work_hours_editor.grid(row=4, column=1, padx=20)
            hourly_pay_editor = Entry(editor, width=30)
            hourly_pay_editor.grid(row=5, column=1, padx=20)

            # creating text boxes labels
            name_label = Label(editor, text="Employee Name")
            name_label.grid(row=0, column=0, pady=(10, 0))
            employee_id_label = Label(editor, text="Employee ID")
            employee_id_label.grid(row=1, column=0)
            employer_label = Label(editor, text="Employer")
            employer_label.grid(row=2, column=0)
            address_label = Label(editor, text="Address")
            address_label.grid(row=3, column=0)
            weekly_work_hours_label = Label(editor, text="Weekly work hours")
            weekly_work_hours_label.grid(row=4, column=0)
            hourly_pay_label = Label(editor, text="Hourly pay")
            hourly_pay_label.grid(row=5, column=0)

            for record in records:
                name_editor.insert(0, record[0])
                employee_id_editor.insert(0, record[1])
                employer_editor.insert(0, record[2])
                address_editor.insert(0, record[3])
                weekly_work_hours_editor.insert(0, record[4])
                hourly_pay_editor.insert(0, record[5])
```

```
# creating save button
edit_btn = Button(editor, text="Save Record",
                 command=lambda: update())
edit_btn.grid(row=6, column=0, columnspan=2,
             pady=10, padx=10, ipadx=120)

# creating update function
def update():
    # creating the database
    conn = sqlite3.connect('payroll_management.db')
    # creating cursor
    c = conn.cursor()

    record_id = select_box.get()

    c.execute("""UPDATE payroll SET
        name = :name,
        employee_id = :employee_id,
        employer = :employer,
        address = :address,
        weekly_work_hours = :weekly_work_hours,
        hourly_pay = :hourly_pay
        WHERE oid = :oid""",
              {
                 'name': name_editor.get(),
                 'employee_id': employee_id_editor.get(),
                 'employer': employer_editor.get(),
                 'address': address_editor.get(),
                 'weekly_work_hours': weekly_work_hours_editor.get(),
                 'hourly_pay': hourly_pay_editor.get(),
                 'monthly_pay':
get_monthly_pay(weekly_work_hours_editor.get(), hourly_pay_editor.get()),
                 'oid': record_id
              })
    messagebox.showinfo(title="Update record",
                message="You updated record successfully.")
    # commit changes
    conn.commit()
    # closing connection
    conn.close()
```

```
        editor.destroy()

# creating delete_main function
def delete_called():
    delete_window = Tk()
    delete_window.title('delete')
    delete_window.geometry('400x300')
    # creating the database
    conn = sqlite3.connect('payroll_management.db')
    # creating cursor
    c = conn.cursor()
    delete_box_final = Entry(delete_window, width=30)
    delete_box_final.grid(row=0, column=1, padx=20, pady=5)
    delete_box_final_label = Label(delete_window, text="Select ID")
    delete_box_final_label.grid(row=0, column=0, pady=5)

    def delete():
        # creating the database
        conn = sqlite3.connect('payroll_management.db')
        # creating cursor
        c = conn.cursor()

        c.execute("DELETE FROM payroll WHERE oid= " +
                  delete_box_final.get())
        messagebox.showinfo(title="Delete record",
                            message="You deleted record successfully.")
        # commit changes
        conn.commit()
        # closing connection
        conn.close()
        delete_window.destroy()
    delete_btn = Button(
        delete_window, text="Delete Record", command=delete)
    delete_btn.grid(row=2, column=0)

    # commit changes
    conn.commit()
    # closing connection
    conn.close()
    # creating submit function
    def submit():
        # creating the database
        conn = sqlite3.connect('payroll_management.db')
        # creating cursor
        c = conn.cursor()

        # insert into table
        c.execute("INSERT INTO payroll VALUES(:name, :employee_id, :employer,
:address, :weekly_work_hours, :hourly_pay, :monthly_pay)",
                  {
                      'name': name.get(),
                      'employee_id': employee_id.get(),
                      'employer': employer.get(),
                      'address': address.get(),
                      'weekly_work_hours': weekly_work_hours.get(),
                      'hourly_pay': hourly_pay.get(),
                      'monthly_pay':                get_monthly_pay(weekly_work_hours.get(),
hourly_pay.get())
                  })
        messagebox.showinfo(title="Add record",
                            message="You added record successfully.")
        # commit changes
        conn.commit()
        # closing connection
        conn.close()

        # clearing textbox
        name.delete(0, END)
        employee_id.delete(0, END)
        employer.delete(0, END)
        address.delete(0, END)
        weekly_work_hours.delete(0, END)
        hourly_pay.delete(0, END)

    # creating query function
    def query():
        new_window = Tk()
        new_window.title('Record Table')
        new_window.geometry('800x400')
        # creating the database
        conn = sqlite3.connect('payroll_management.db')
        # creating cursor
        c = conn.cursor()
        # creating subheadings
        name_query_label = Label(new_window, text="Employee Name")
        name_query_label.grid(row=0, column=0, pady=(10, 0))
        employee_id_query_label = Label(new_window, text="Employee ID")
        employee_id_query_label.grid(row=0, column=1, pady=(10, 0))
        employer_query_label = Label(new_window, text="Employer")
        employer_query_label.grid(row=0, column=2, pady=(10, 0))
        address_query_label = Label(new_window, text="Address")
        address_query_label.grid(row=0, column=3, pady=(10, 0))
        weekly_work_hours_query_label = Label(
            new_window, text="Weekly work hours")
        weekly_work_hours_query_label.grid(row=0, column=4, pady=(10, 0))
        hourly_pay_query_label = Label(new_window, text="Hourly pay")
        hourly_pay_query_label.grid(row=0, column=5, pady=(10, 0))
        salary_query_label = Label(new_window, text="Net Salary")
        salary_query_label.grid(row=0, column=6, pady=(10, 0))
        oid_query_label = Label(new_window, text="OID")
        oid_query_label.grid(row=0, column=7, pady=(10, 0))
        pdf_btn = Button(new_window, text="Print All", command=pdf)
        pdf_btn.grid(row=0, column=10, padx=(10, 0), pady=(8, 0), ipadx=40)
        temp = c.execute("SELECT *,oid FROM payroll")
        # records = c.fetchall()
        i = 1
        for payroll in temp:
            for j in range(len(payroll)):
                e = Entry(new_window, width=10, fg='blue')
                e.grid(row=i, column=j, pady=(10, 0), padx=(10, 0))
                e.insert(END, payroll[j])
            i = i+1

        # commit changes
        conn.commit()
        # closing connection
        conn.close()

    def pdfsingle():
        print_window = Tk()
        print_window.title('Print Single Record')
        print_window.geometry('400x400')
        # creating the database
        conn = sqlite3.connect('payroll_management.db')
        # creating cursor
        c = conn.cursor()

        def printing():
            # creating the database
            conn = sqlite3.connect('payroll_management.db')
            # creating cursor
            c = conn.cursor()

            temp = c.execute(
                "SELECT *,oid FROM payroll WHERE oid =" +
    select_box_pdfsingle.get())
            records = c.fetchall()
            table = tabulate(records, headers=["Name", "Emp ID", "Employer",
    "Address", "Weekly Work Hours",
                        "Hourly", "Net Payable Salary", "Primary Key"],
    tablefmt="fancy_grid")
            print(table)

            pdf = FPDF(orientation='P', unit='mm', format='A4')
            pdf.add_page()
            headers = ["Name", "Emp ID", "Employer", "Address",
                    "Weekly Work Hours", "Hourly", "Net Payable Salary", "Primary
    Key"]

            for header in headers:
                # set font to bold and size 14 for headers
                pdf.set_font("Arial", 'B', size=7)
                pdf.cell(24, 10, header, 1, 0, "C")
            pdf.ln()
            for row in records:
                pdf.set_font("Arial", size=6)
                for item in row:
                    pdf.cell(24, 10, str(item), 1, 0, "C")
                pdf.ln()
            pdf.output("single.pdf")
            # commit changes
            conn.commit()
            # closing connection
            conn.close()
            print_window.destroy()

        select_box_pdfsingle_label = Label(print_window, text="Select ID")
        select_box_pdfsingle_label.grid(row=0, column=0, pady=5)
        select_box_pdfsingle = Entry(print_window, width=30)
        select_box_pdfsingle.grid(row=0, column=1, padx=20, pady=5)
        p_btn = Button(print_window, text="Print Record",
                       command=lambda: printing())
        p_btn.grid(row=1, column=0)

        # commit changes
        conn.commit()
        # closing connection
        conn.close()

    def pdf():
        # creating the database
        conn = sqlite3.connect('payroll_management.db')
        # creating cursor
        c = conn.cursor()

        temp = c.execute("SELECT *,oid FROM payroll")
        records = c.fetchall()
        table = tabulate(records, headers=["Name", "Emp ID", "Employer", "Address",
"Weekly Work Hours",
                        "Hourly", "Net Payable Salary", "Primary Key"],
tablefmt="fancy_grid")
        print(table)

        pdf = FPDF(orientation='P', unit='mm', format='A4')
        pdf.add_page()
        headers = ["Name", "Emp ID", "Employer", "Address",
                  "Weekly Work Hours", "Hourly", "Net Payable Salary", "Primary Key"]

        for header in headers:
            # set font to bold and size 14 for headers
            pdf.set_font("Arial", 'B', size=7)
            pdf.cell(24, 10, header, 1, 0, "C")
        pdf.ln()
        for row in records:
            pdf.set_font("Arial", size=6)
            for item in row:
                pdf.cell(24, 10, str(item), 1, 0, "C")
            pdf.ln()
        pdf.output("table.pdf")

        # commit changes
        conn.commit()
        # closing connection
        conn.close()

    # creating text boxes
    name = Entry(root, width=30)
    name.grid(row=1, column=1, padx=20, pady=(10, 0))
    employee_id = Entry(root, width=30)
    employee_id.grid(row=2, column=1, padx=20)
    employer = Entry(root, width=30)
    employer.grid(row=3, column=1, padx=20)
    address = Entry(root, width=30)
    address.grid(row=4, column=1, padx=20)
    weekly_work_hours = Entry(root, width=30)
    weekly_work_hours.grid(row=5, column=1, padx=20)
    hourly_pay = Entry(root, width=30)
    hourly_pay.grid(row=6, column=1, padx=20)

    # creating text boxes labels
    name_label = Label(root, text="Employee Name")
    name_label.grid(row=1, column=0, pady=(10, 0))
    employee_id_label = Label(root, text="Employee ID")
    employee_id_label.grid(row=2, column=0)
    employer_label = Label(root, text="Employer")
    employer_label.grid(row=3, column=0)
    address_label = Label(root, text="Address")
    address_label.grid(row=4, column=0)
    weekly_work_hours_label = Label(root, text="Weekly work hours")
    weekly_work_hours_label.grid(row=5, column=0)
    hourly_pay_label = Label(root, text="Hourly pay")
    hourly_pay_label.grid(row=6, column=0)

    # creating submit button
    submit_btn = Button(root, text="Add Record", command=submit)
    submit_btn.grid(row=7, column=0)

    # creating query button
    query_btn = Button(root, text="Show Records", command=query)
    query_btn.grid(row=7, column=1)

    # creating delete button
    delete_btn = Button(root, text="Delete Record", command=delete_called)
    delete_btn.grid(row=9, column=0)

    # creating update button
    edit_btn = Button(root, text="Edit Record", command=editing_window)
    edit_btn.grid(row=9, column=1)

    pdf_btn = Button(root, text="Print Single Record", command=pdfsingle)
    pdf_btn.grid(row=10, column=0)

    # commit changes
    conn.commit()

    # closing connection
    conn.close()


frame.pack()

window.mainloop()
```

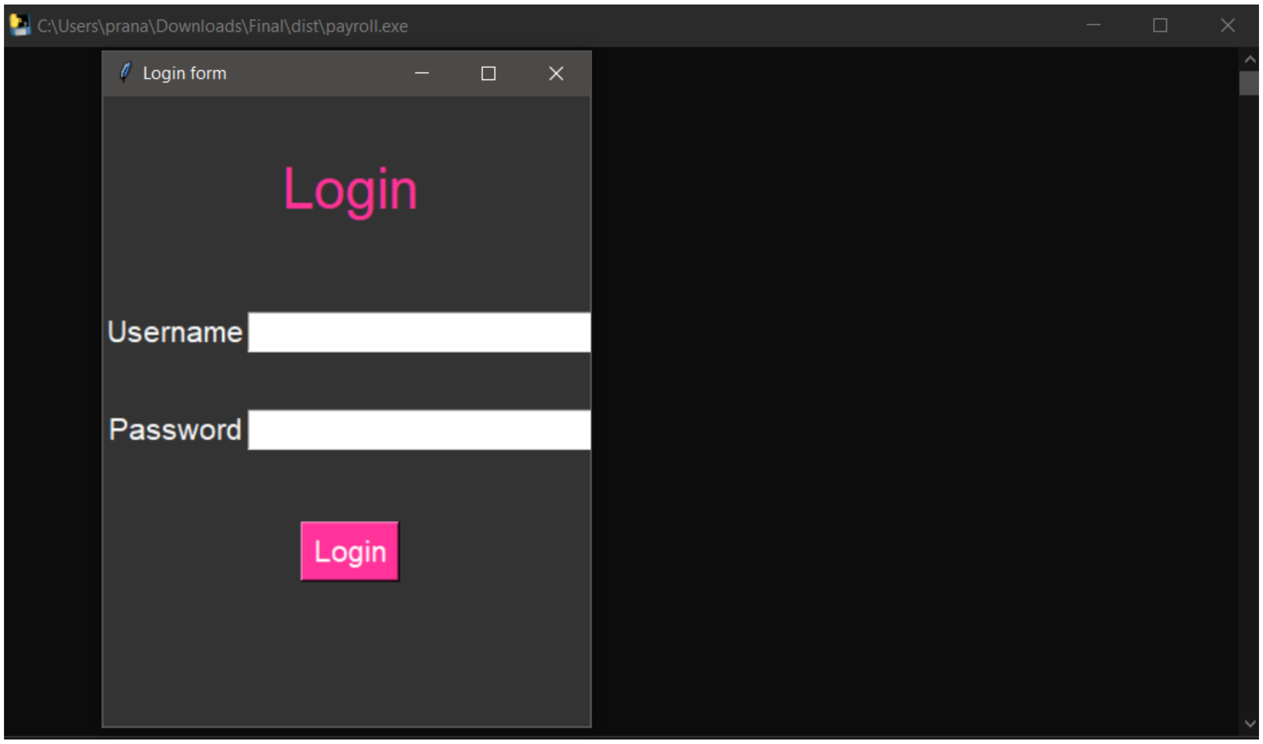
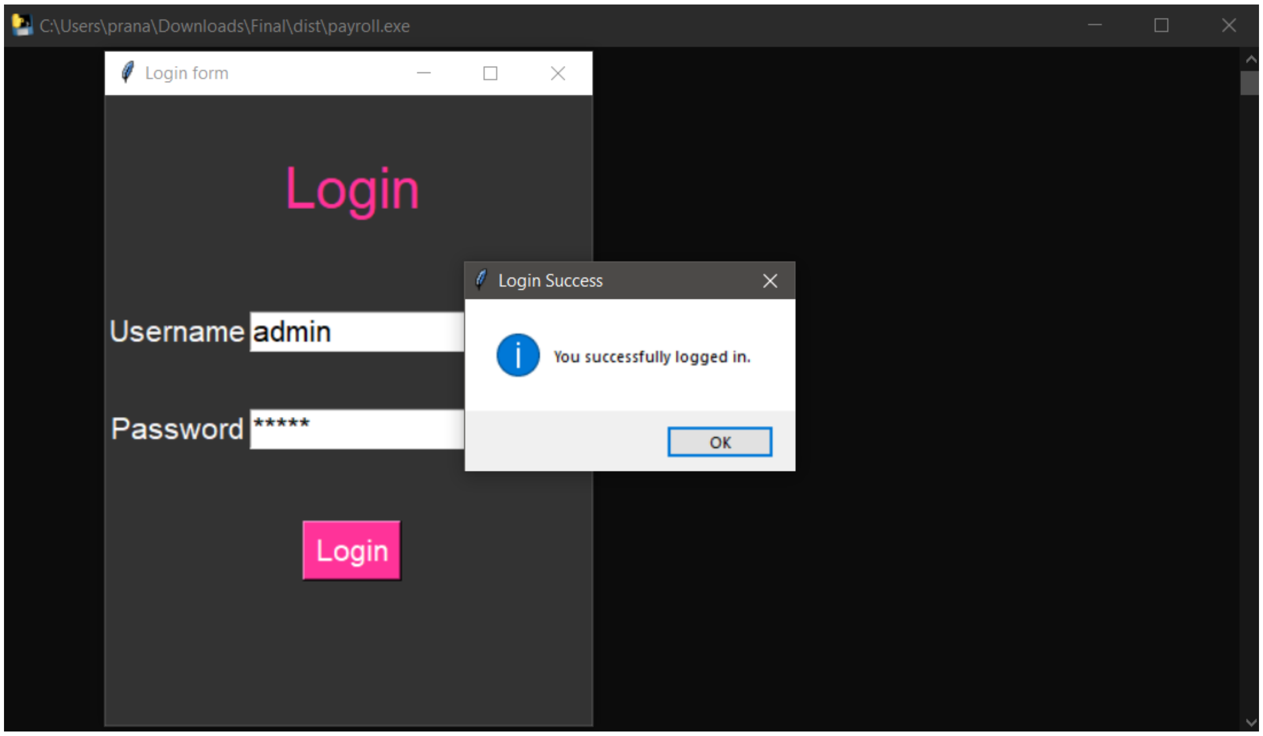


Fig. 6 Login Page



Username admin

Password *****

You successfully logged in.

Fig. 7 Login Successful Pop up

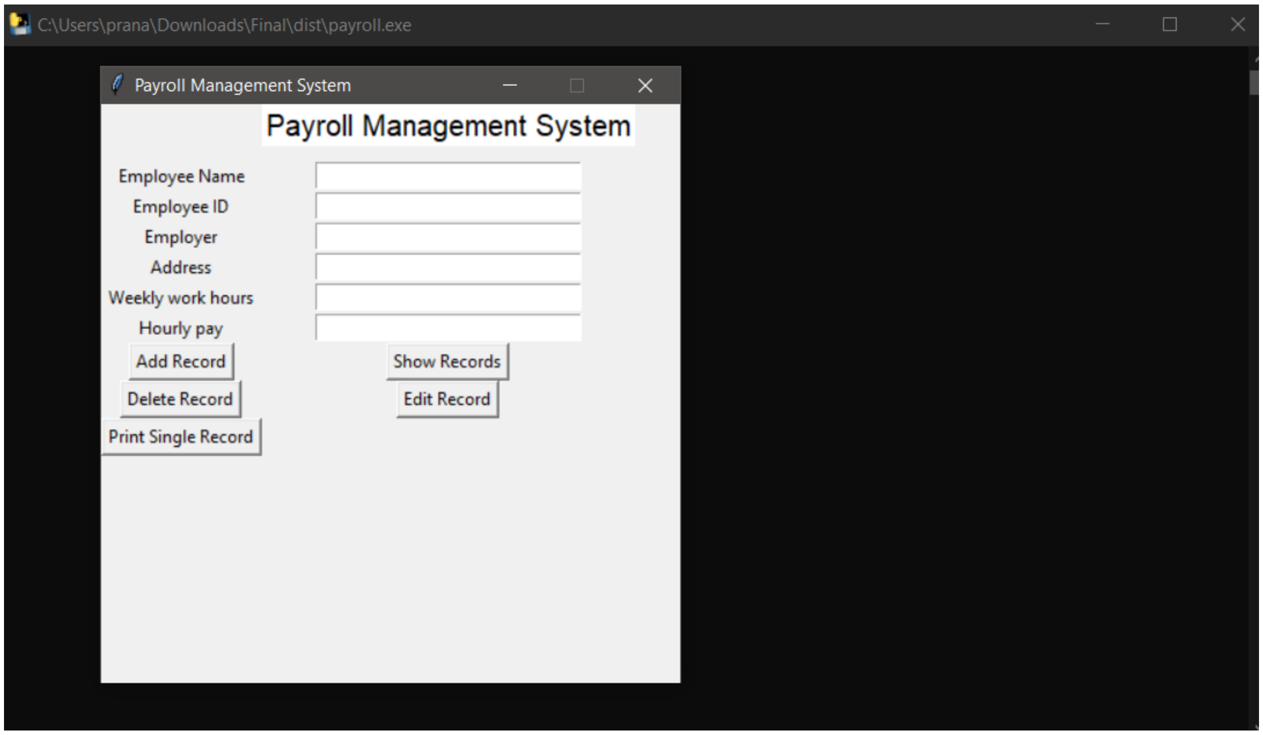
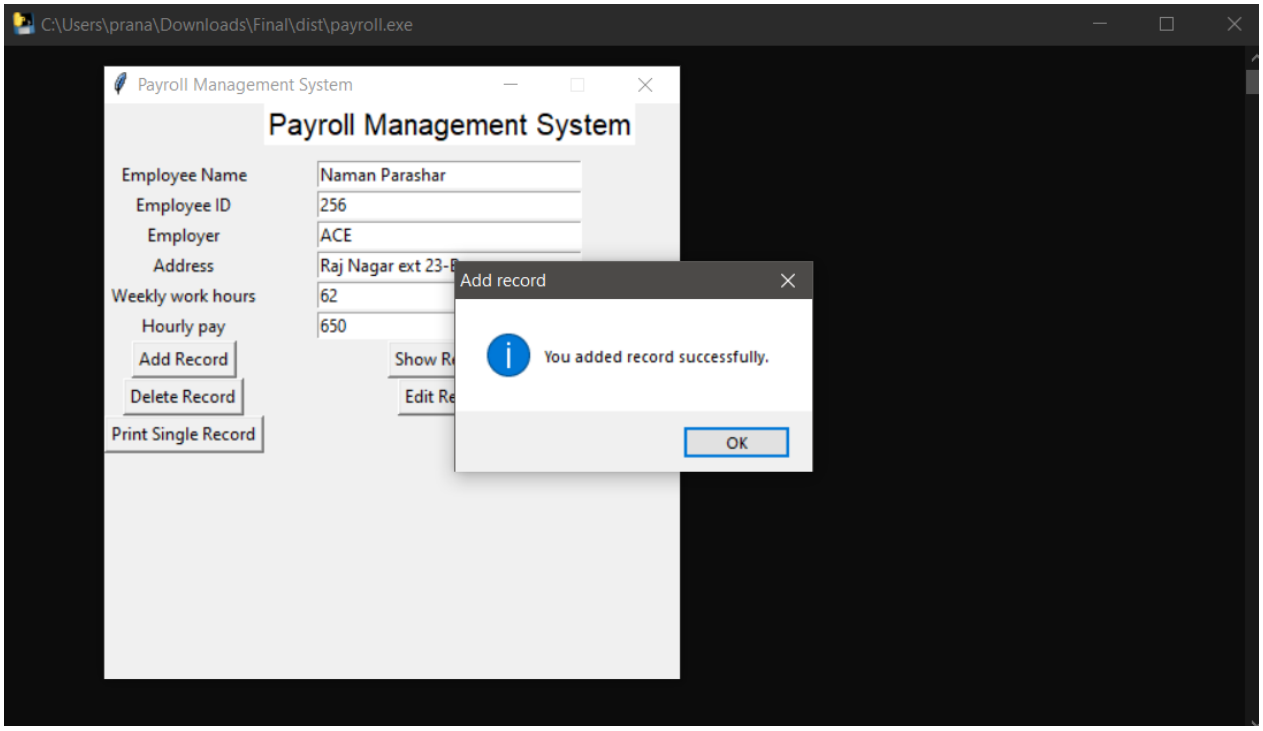


Fig. 8 Home Page


C:\Users\prana\Downloads\Final\dist\payroll.exe

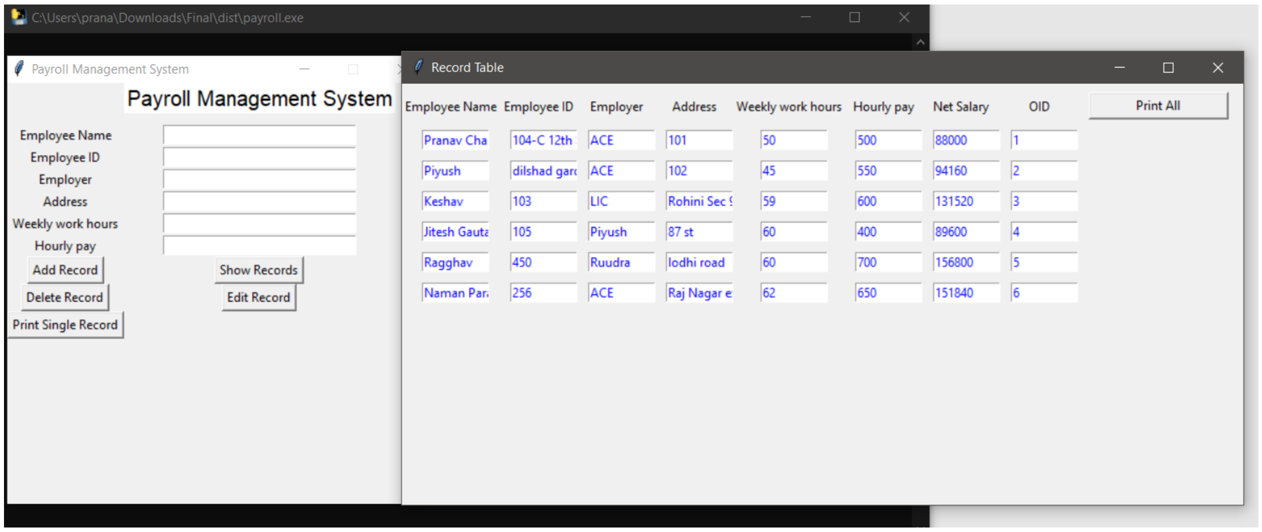
Payroll Management System

Employee Name	Naman Parashar
Employee ID	256
Employer	ACE
Address	Raj Nagar ext 23-E
Weekly work hours	62
Hourly pay	650

Add record

You added record successfully.

Fig. 9 Adding Records

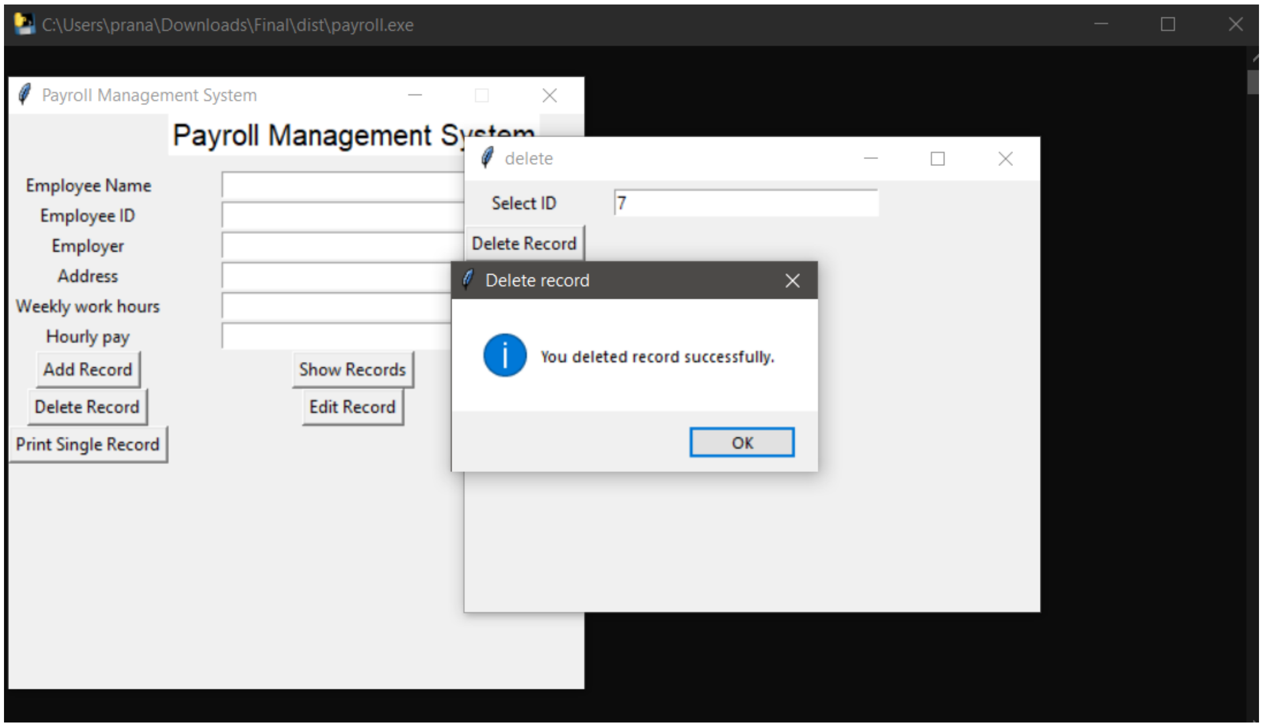

C:\Users\prana\Downloads\Final\dist\payroll.exe

Payroll Management System

Record Table

Employee Name	Employee ID	Employer	Address	Weekly work hours	Hourly pay	Net Salary	OID
Pranav Cha	104-C 12th	ACE	101	50	500	88000	1
Piyush	dilshad garc	ACE	102	45	550	94160	2
Keshav	103	LIC	Rohini Sec 9	59	600	131520	3
Jitesh Gauta	105	Piyush	87 st	60	400	89600	4
Ragghav	450	Ruudra	lodhi road	60	700	156800	5
Naman Par.	256	ACE	Raj Nagar e	62	650	151840	6

Fig. 10 Display Records



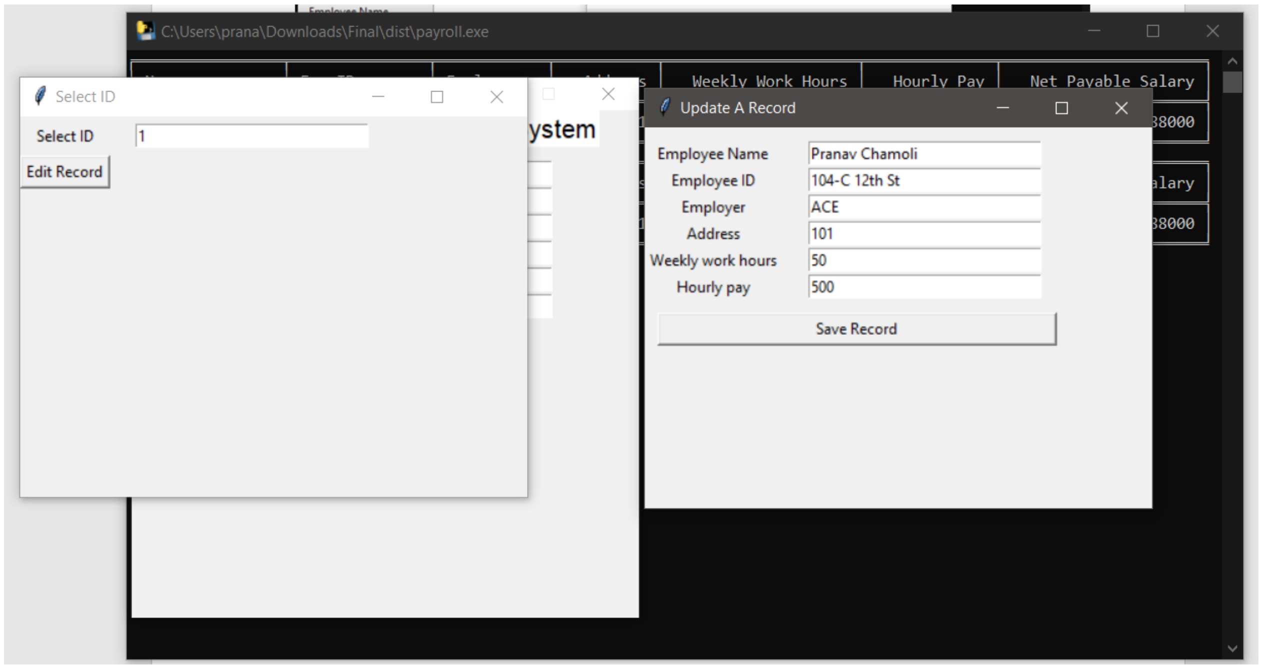
```
C:\Users\prana\Downloads\Final\dist\payroll.exe

Payroll Management System
Payroll Management System
Employee Name
Employee ID
Employer
Address
Weekly work hours
Hourly pay
Add Record        Show Records
Delete Record     Edit Record
Print Single Record

delete
Select ID   7
Delete Record

Delete record
You deleted record successfully.
OK
```

Fig. 11 Deleting a record



```
C:\Users\prana\Downloads\Final\dist\payroll.exe

Weekly Work Hours   Hourly Pay   Net Payable Salary
38000
38000

Select ID
Select ID   1
Edit Record

Update A Record
Employee Name       Pranav Chamoli
Employee ID         104-C 12th St
Employer            ACE
Address             101
Weekly work hours   50
Hourly pay          500
Save Record
```

Fig. 12 Editing a record

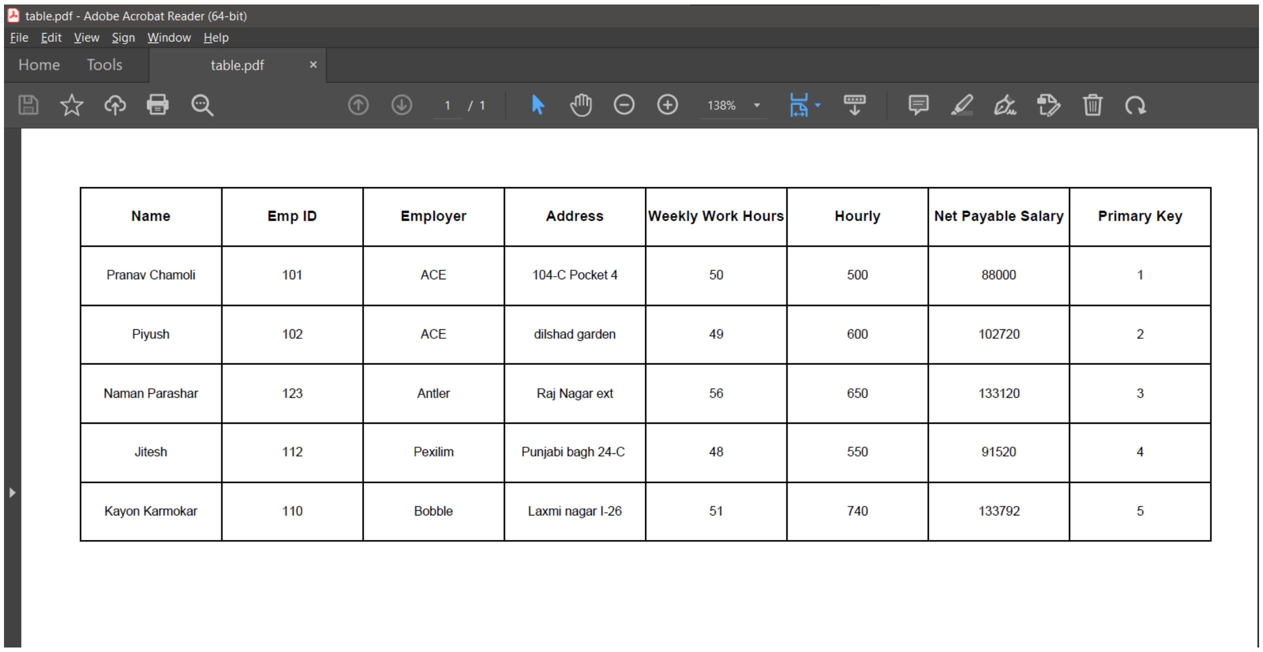
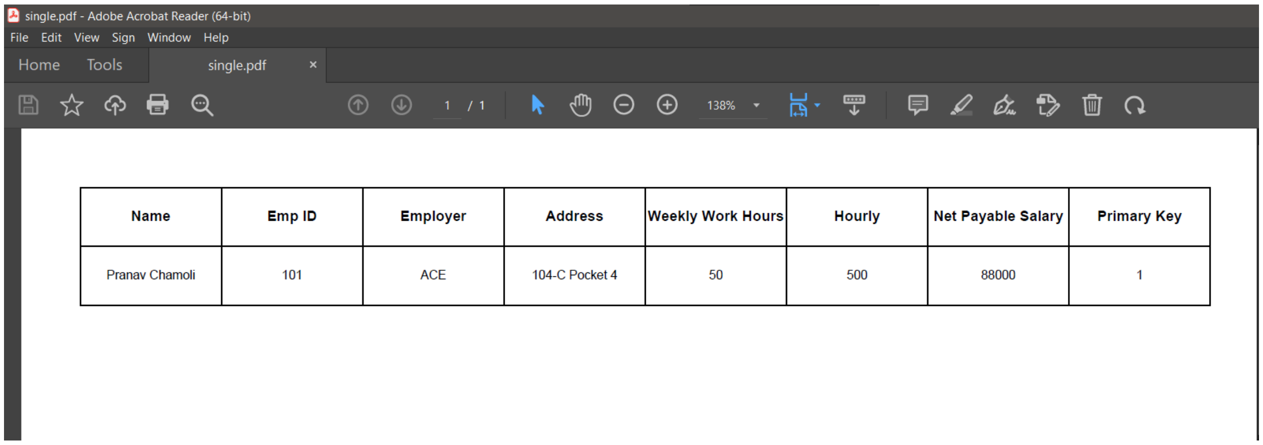


table.pdf - Adobe Acrobat Reader (64-bit)
File Edit View Sign Window Help
Home Tools table.pdf
1 / 1 138%

Name	Emp ID	Employer	Address	Weekly Work Hours	Hourly	Net Payable Salary	Primary Key
Pranav Chamoli	101	ACE	104-C Pocket 4	50	500	88000	1
Piyush	102	ACE	dilshad garden	49	600	102720	2
Naman Parashar	123	Antler	Raj Nagar ext	56	650	133120	3
Jitesh	112	Pexilim	Punjabi bagh 24-C	48	550	91520	4
Kayon Karmokar	110	Bobble	Laxmi nagar I-26	51	740	133792	5

Fig. 13 Printing all records


single.pdf - Adobe Acrobat Reader (64-bit)
File Edit View Sign Window Help
Home Tools single.pdf
1 / 1 138%

Name	Emp ID	Employer	Address	Weekly Work Hours	Hourly	Net Payable Salary	Primary Key
Pranav Chamoli	101	ACE	104-C Pocket 4	50	500	88000	1

Fig. 14 Printing a single record

Chapter 5

TESTING & TEST RESULTS

5.1 SOFTWARE TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing is an exposure of a system to trial input to see whether software meets correct output. Testing cannot be determined whether software meets user's needs, only whether it appears to confirm to requirements. Testing can show that a system is free of errors, only that it contains error. Testing finds errors, it does not correct errors. Software success is a quality product, on time and within cost. Through testing can reveal critical mistakes. Testing should therefore,

1. Validate Performance
2. Detects Errors
3. Identify Inconsistencies

5.2 Test Objective

- There is strong evidence that effective requirement management leads to overall project cost savings. The three primary reasons for this are,
- Requirement errors typically cost well over 10 times more to repair than other errors.
- Requirement errors typically comprise over 40% of all errors in a software project.
- Small reduction in the number of requirement errors pays big dividend in avoided rework costs and schedule delays.
- Systems are not designed as entire systems nor are they tested as single systems the analyst must perform both unit and system testing. For this different level testing are use:

5.2.1 Unit Testing

In unit testing Module is tested separately and the programmer simultaneously along with the coding of the module performs it.

In unit testing the analyst tests the programs making up a system. For this reason, unit testing is sometime called program testing. Unit testing gives stress on modules independently of one another, to find errors. This helps the tester in detecting errors in coding and logic that are contained within that module alone. The errors resulting from the interaction between modules are initially avoided.

Unit testing can be performed from the bottom up, starting with smallest and lowest-level modules and proceeding one at a time., for each module in Bottom-up testing a short program is used to execute the module and provides the needed data, so that the module is asked to perform the way it will when embed within the larger system.

5.2.2 System Testing

This is performed after the system is put together. The system is tested against the system requirement to check if all the requirements are met and if the system performs of specify by the requirements.

Testing is an important function to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully activated. Another reason for system testing is its utility as a user-oriented vehicle before implementation.

The function of testing is to detect the defects in the Software. The main goal testing is to uncover requirement, design and coding errors in the programs. The types of testing are discussed below:

5.2.3 MODULE TESTING

Module tests are typically dynamic white-box tests. This requires the execution of the software or parts of the software. The software can be executed in the target system, an emulator, simulator or any other suitable test environment.
The focus of the tests is:

- Set up of regression tests. This means the test environment once set up for a function can be re-used to check its performance e.g., after maintenance.
- Coverage of the relevant state of the art test methods like equivalence class building, boundary value analysis and condition coverage are used.

5.2.3 INTEGRATION TESTING

"If they all work individually, they should work when we put them together." The problem of course is "putting them together ". This can be done in two ways:

1. Top-down integration: Modules are integrated by moving downwards through the control hierarchy, beginning with main control module are incorporated into the structure in either a depth first or breadth first manner.
2. Bottom-up integration: It begins with construction and testing with atomic modules i.e. modules at the lowest level of the program structure. Because modules are integrated from the bottom up, processing required for the modules subordinate to a given level is always available and the need of stubs is eliminated.

5.2.4 BLACK-BOX TESTING

Black-box testing is a method of software testing that tests the functionality of an application as opposed to its internal structures or workings.

The system is tested just to assure whether it is meeting all the expectations or requirements from it, tester is not concerned with the internal logic of the module or system to be tested. Some inputs are given to system and it is observed whether the system is working as per the client's requirements or not or according to the requirements specified in SRS document. Specific knowledge of the application's code/internal structure and programming knowledge in general is not required.
Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid inputs and determines the correct output. There is no knowledge of the test object's internal structure. This method of test can be applied to all levels of software testing: unit, integration, functional, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well. Black box testing or functional testing is used to check that the outputs of a program, given certain inputs, conform to the functional specification of the program. The term black box indicates that the tester does not examine the internal implementation of the program being executed

5.2.5 WHITE-BOX TESTING

A software testing technique where by explicit knowledge of the internal workings of the item being tested are used to select the test data. Unlike black box testing, white box testing uses specific knowledge of programming code to examine outputs. The test is accurate only if the tester knows what the program is supposed to do. He or she can then see if the program diverges from its intended goal. White box testing does not account for errors caused by omission, and all visible code must also be readable.
Contrary to black-box testing, software is viewed as a white-box, or glass-box in white-box testing, as the structure and flow of the software under test are visible to the tester. Testing plans are made according to the details of the software implementation, such as programming language, logic, and styles. Test cases are derived from the program **structure**. White-box testing is also called glass-box testing, logic-driven testing or design-based testing. There are many techniques available in white-box testing, because the problem of intractability is eased by specific knowledge and attention on the structure of the software under test.

5.2.6 VALIDATION TESTING

TEST CASES

Login Module

Test Case ID	Test Case Description	Test Input	Expected Output	Actual Output	Pass/Fail
TC-01	Verify successful login	Valid user id and password	Login successful	Login successful	Pass

TC-02	Verify invalid login	Invalid user id or password	Error message indicating invalid login	Error message indicating invalid login	Pass
TC-03	Verify invalid login	Empty user id or password field	Error message indicating invalid login	Error message indicating invalid login	Pass

Add record Module

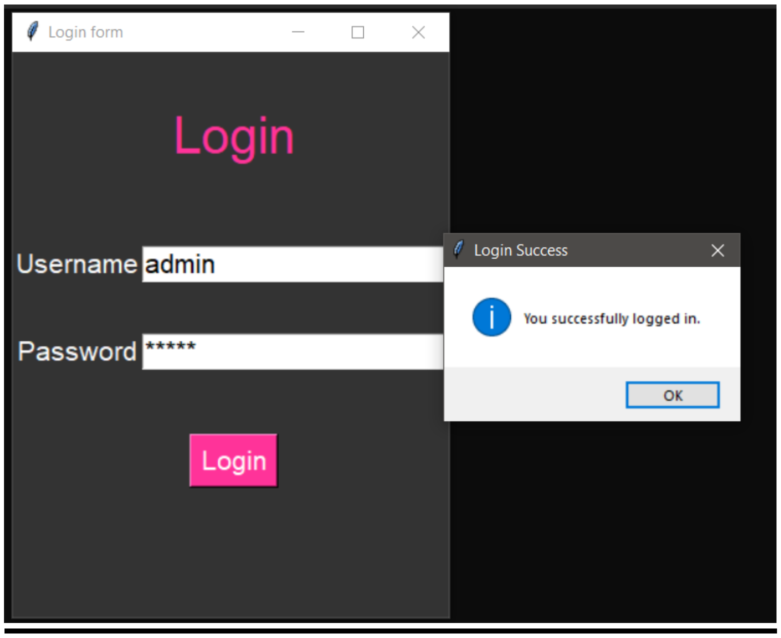
Test Case ID	Test Case Description	Test Input	Expected Output	Actual Output	Pass/Fail
TC-01	Adding new record	Valid user details	Message indicating that record was added	Message indicating that record was added	Pass
TC-02	Adding new record with empty fields	Empty fields	Error message indicating invalid input	Error message indicating invalid input	Pass

Delete record Module

Test Case ID	Test Case Description	Test Input	Expected Output	Actual Output	Pass/Fail
TC-01	Deleting a record	Valid user details	Message indicating that record was deleted	Message indicating that record was deleted	Pass
TC-02	Deleting a record with empty fields	Empty fields	Error message indicating invalid input	Error message indicating invalid input	Pass

Update record Module

Test Case ID	Test Case Description	Test Input	Expected Output	Actual Output	Pass/Fail
TC-01	Updating a record	Valid user details	Message indicating that record was updated	Message indicating that record was updated	Pass
TC-02	Updating a record with empty fields	Empty fields	Error message indicating invalid login	Error message indicating invalid input	Pass

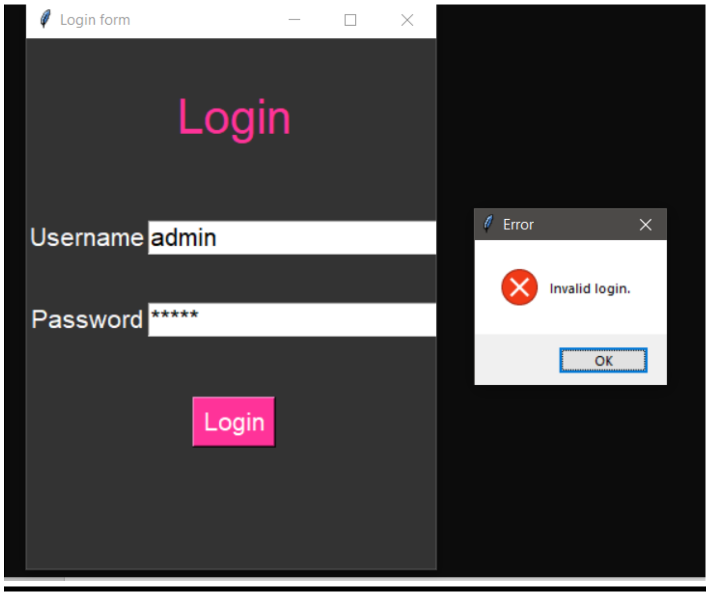


Username: admin

Password: *****

You successfully logged in.

Fig. 15 Verify Successful login



Login form

Login

Username admin

Password *****

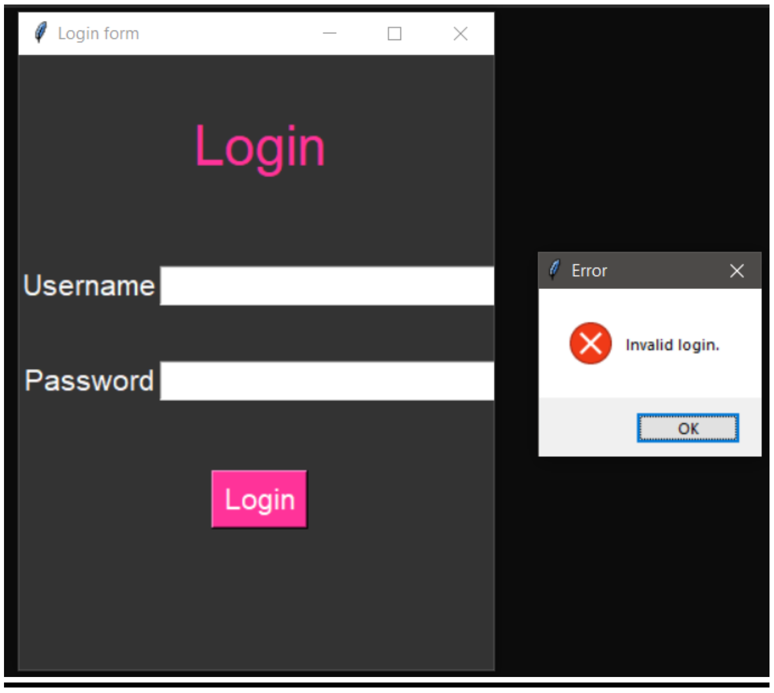
Login

Error

Invalid login.

OK

Fig. 16 Verify invalid login



Login form

Login

Username

Password

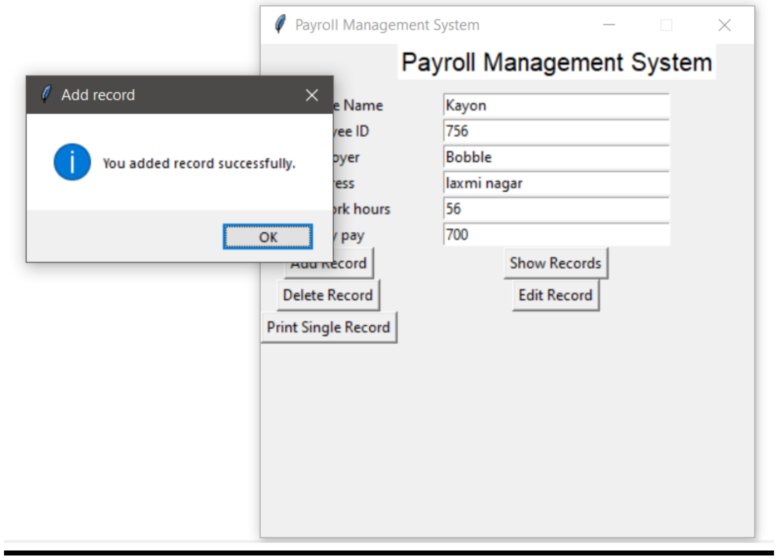
Login

Error

Invalid login.

OK

Fig. 17 Verify invalid login with empty fields

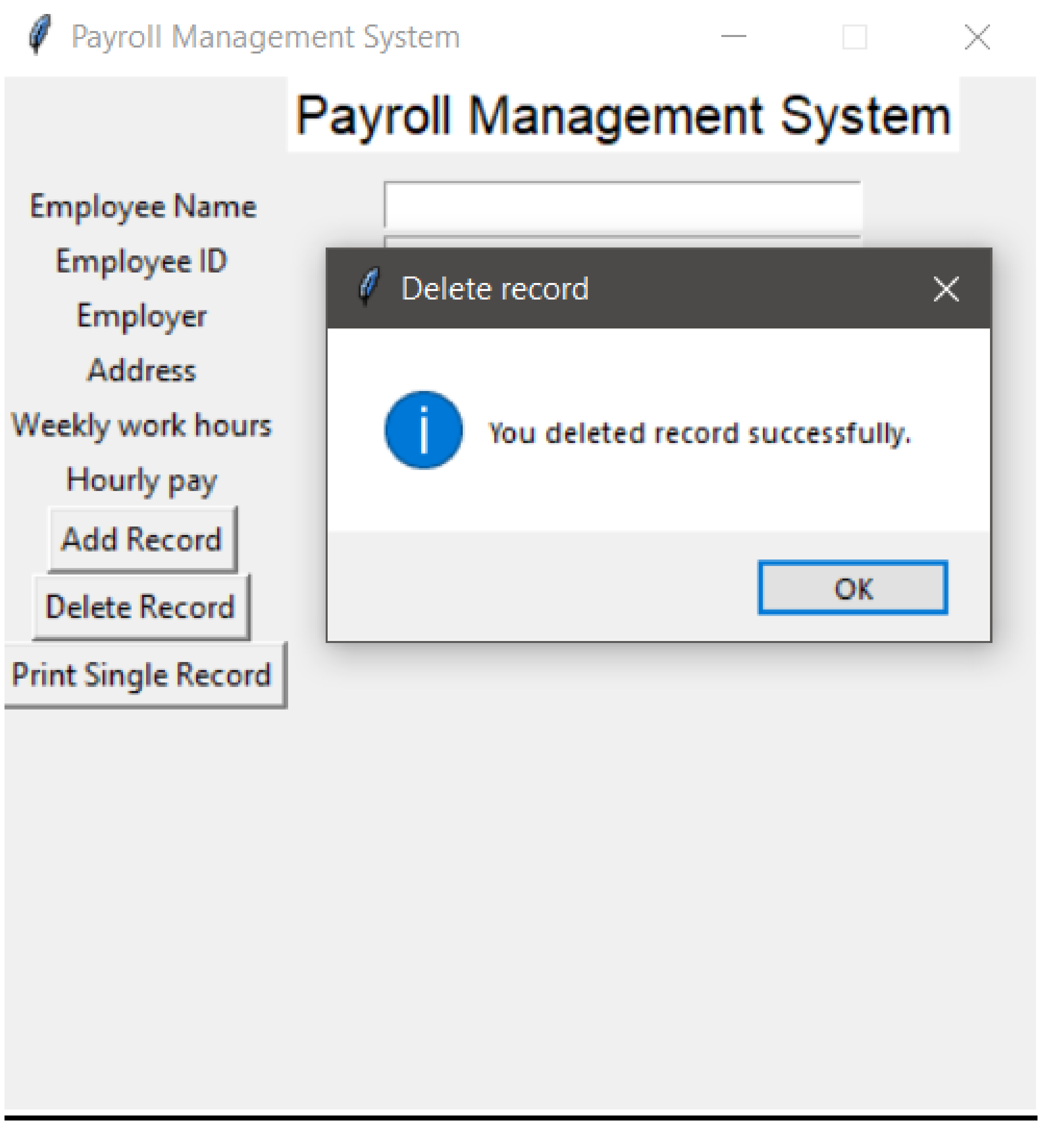


Payroll Management System

You added record successfully.

e Name Kayon
yee ID 756
oyer Bobble
ress laxmi nagar
ork hours 56
y pay 700

Fig. 18 Adding new record test case

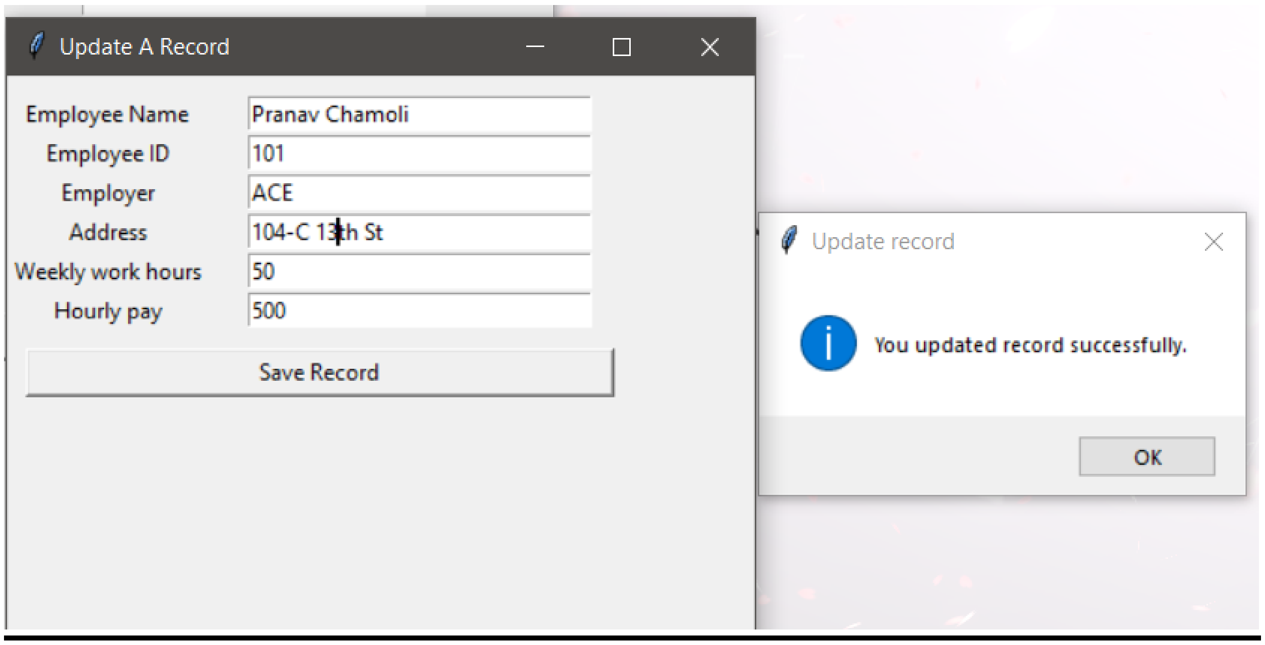


Payroll Management System

Delete record

You deleted record successfully.

Fig. 19 Deleting a record test case



Update A Record

Employee Name	Pranav Chamoli
Employee ID	101
Employer	ACE
Address	104-C 13th St
Weekly work hours	50
Hourly pay	500

Update record

You updated record successfully.

Fig. 20 Updating a record test case

Chapter 6

CONCLUSION

6.1 CONCLUSION

In conclusion, this project aimed to develop a payroll management system using Python language and Tkinter package. The system was designed to automate the process of managing employee payroll, including tasks such as calculating salaries, keeping track of employee information, and generating salary statements. The system was developed using an object-oriented approach, and it includes features such as the ability to add, delete, and update employee records, generate salary statements, and export data.

The system was tested and found to be fully functional and efficient. The user interface is interactive and easy to use. The system also provides a secure way of storing employee data. The system has been designed keeping in mind the scalability and security aspects.

6.2 FUTURE SCOPE

Every project whether large or small has some limitations no matter however diligently developed. In some cases, limitations are small while in other cases they may be broad also. The new system has got some limitations. Major areas where modifications can be done are as follows:

- Our system is not online so further it can be improved.
- The security is limited so some additional arrangement could be made to provide more security to the system.
- There is no provision of complain handling so further it can be added.

BIBLOGRAPHY

- https://openai.com/blog/chatgpt/
- https://www.geeksforgeeks.org/python-programming-language/?ref=shm
- https://docs.python.org/3/
- https://stackoverflow.com/
- https://www.wikipedia.org/
- https://code.visualstudio.com/docs
- https://www.sqlite.org/docs.html
- https://www.tutorialspoint.com/python/python_gui_programming.htm
- https://realpython.com/python-gui-tkinter/