Criterion C: Development

This project was developed using MySQL Workbench and Database, Python framework with the help of IntelliJ IDEA Integrated Development Environment.

1. Core Libraries Imported

Figure 1. Core Libraries Used to Employ Existing Functionalities (reduce development time)

```
import tkinter
from datetime import datetime
from tkcalendar import Calendar
import secrets
import string
from PIL import ImageTk, Image
import mysql.connector
```

2. Relational Database Manipulation:

The records are stored and manipulated in a MySQL database. Three tables are present: “people”, “leave_track”, “budget”, and “feedback”. The use of ‘mysql.connector’ external library allows for the connection of the database and interface, hence allowing manipulation through functions.

Figure 2. Connection Between Database and Interface

```
import mysql.connector
import dbconfig as cfg

mydb = mysql.connector.connect(host=cfg.mysql["host"],
                               user=cfg.mysql["user"],
                               password=cfg.mysql["password"],
                               db=cfg.mysql["db"])
```

```
mysql = {
    "host": "localhost",
    "user": "root",
    "password": "happy",
    "db": "hr_db",
}

use_anonymous = True
```

Queries are used to enter and extract data from the database. Data is entered through the ‘execute(Query)’ method which is a String. Data is extracted through the use of. fetchall() which retrieves data from the previously run query in the form of tuples. These are unpacked in Python.

SQL Commands Used:

Figure 3. SELECT keyword to extract employee records

```
my_cursor.execute("SELECT * FROM people WHERE employee_id=%s", (self.task_id, ))
details = my_cursor.fetchall()[0]
```

Figure 4. DELETE keyword to erase employee records

```
delete_statement3 = "DELETE FROM people WHERE employee_id=%s"
my_cursor.execute(delete_statement3, (val_delete, ))
```

Figure 5. UPDATE keyword to overwrite previous records

```
my_cursor.execute("UPDATE people SET password=%s WHERE employee_id=%s", (str(new_password), self.task_id))
```

Figure 6. INSERT keyword to add new employee records

```
insert_stmnt = "INSERT INTO people(employee_id, name, phone, email, dob, doj, sex, username, password) " \
               "VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s)"
my_cursor.execute(insert_stmnt, data)
```

The 'people' table has a **Primary Key** which auto-increments the employee_id field. The **Foreign Key** of the 'leave_track' and 'budget' tables reference the employee_id primary key to ensure that an employee that doesn't exist can't enter leaves or be given a salary. Since 'feedback' is anonymous, a foreign key is not used.

Figure 7. Relational Database Connection via Primary Key

```
CREATE TABLE people
(employee_id int PRIMARY KEY auto_increment NOT NULL,
```

```
CREATE TABLE budget
(employee_id INT NOT NULL,
FOREIGN KEY(employee_id) REFERENCES people(employee_id));
```

```
CREATE TABLE leave_track
(leaverId INT NOT NULL,
FOREIGN KEY(employee_id) REFERENCES people(employee_id));
```

3. Temporary Workstation: Cursor

The database.cursor() method was used to communicate queries with the database. The cursor is an object that behaves as a buffer, providing temporary storage to multiple rows of records. By default, the database server fetches one row at a time but the cursor allows a large number

of rows to be fetched at once. This increases system efficiency since a higher throughput is achieved.

Figure 8. Use of Cursor to Fetch All Budget Table Values

```
my_cursor = mydb.cursor()
```

```
# adds the extra cost value and type into the database
def add_extra_costs(employee_id, extra_cost: int, cost_type):
    my_cursor.execute("SELECT * FROM budget")
    budget_values = my_cursor.fetchall()
    added_budget_ids = []
```

4. User Defined Class

“IdFunctions” is a user-defined class with methods like “deleting_value” and “edit_button” which reduce code redundancy and improve the memory required to run the solution. Furthermore, the object-oriented approach defines a structure to each module. This would make it easier for the client to add additional features, reducing the development and debugging time in the future.

Figure 9. IdFunctions Class with some of its Methods

```
# A class for all functions that require an employee_id
class IdFunctions:

    def __init__(self, task_id):
        self.task_id = task_id

    # updates employee values in the database
    def edit_button(self, data):

        update = "UPDATE people SET name=%s, phone=%s, email=%s, dob=%s, doj=%s, sex=%s WHERE employee_id=%s"
        remade_data = []
        for value in data:
            remade_data.append(value)
        remade_data.append(self.task_id)
        my_cursor.execute(update, remade_data)
        mydb.commit()

    # deletes the employee from the database
    def deleting_value(self):
        val_delete = self.task_id
        delete_statement1 = "DELETE FROM leave_track WHERE leaverId=%s"
        delete_statement2 = "DELETE FROM budget WHERE employee_id=%s"
        delete_statement3 = "DELETE FROM people WHERE employee_id=%s"
        my_cursor.execute(delete_statement1, (val_delete, ))
        my_cursor.execute(delete_statement2, (val_delete, ))
        my_cursor.execute(delete_statement3, (val_delete, ))
        mydb.commit()
```

5. Authentication Model

The entered username and password are compared with the credentials in the database to authenticate users before letting them access their portal. The password is hidden behind asterisks to enhance security.

Figure 10. Read and Authenticate Username and Password

```
# reads login credentials and checks whether they are correct
def login_user():
    user = str(username.get())
    code = str(password.get())
    # calls function that compares with correct credentials
    request_id = functions.login_check(user, code)
    if request_id == 0:
        incorrect_login.grid(row=4, column=2)
        username.delete(0, tkinter.END)
        password.delete(0, tkinter.END)
```

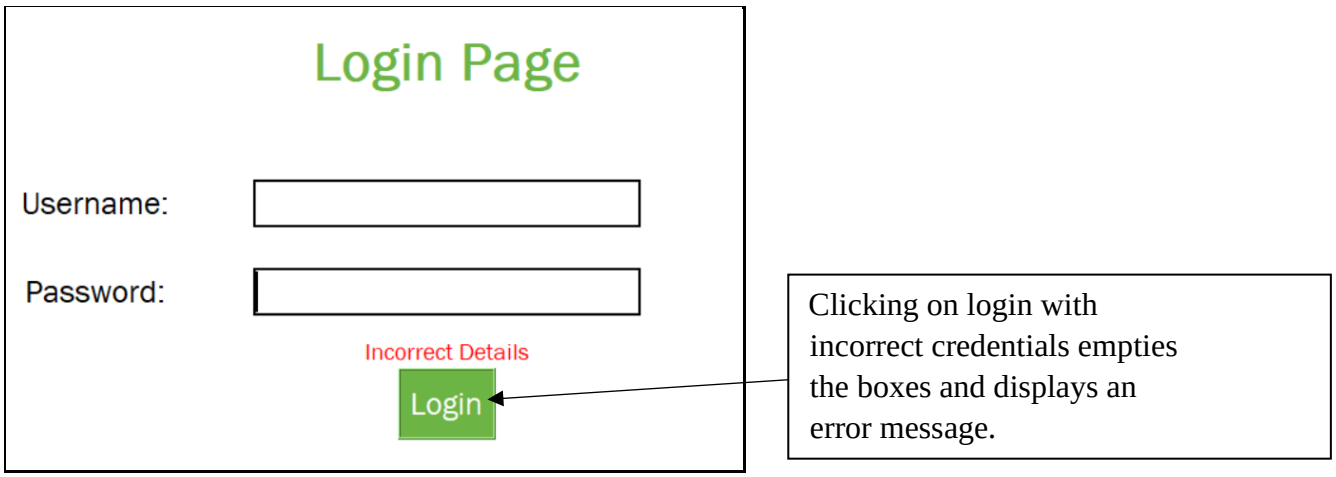
Checks whether the username and password match the correct credentials

Figure 11. Function that Compares with Correct Credentials (from Figure 10)

```
# Checks if login credentials exist
def login_check(username, password):
    my_cursor.execute("SELECT * FROM People")
    people = my_cursor.fetchall()
    for value in range(len(people)):
        if (people[value][7], people[value][8]) == (username, password):
            return people[value][0]
    return 0
```

Figure 12. Display of Incorrect Login Credentials Message (from Figure 10)

```
incorrect_login = tkinter.Label(login, text="Incorrect Credentials", font=("Franklin Gothic Book", 15),
                                fg="red", borderwidth=0, highlightthickness=0, bg="white")
incorrect_login.grid(row=4, column=2)
incorrect_login.grid_forget()
```



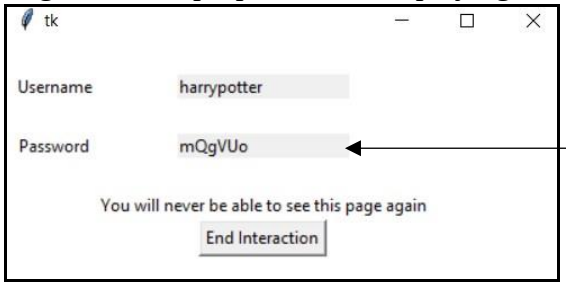
For an extra layer of security, the added employees receive auto-generated usernames (full name without spaces) and a randomly generated 6-character password. After the first login, an employee can change the password through the portal.

Figure 13. Initial Username & Password Creation through Randomization

```
# reads name and removes spaces to create username
text7 = entry1.get().replace(" ", "").casefold()
# creates a randomized password
alphabet = string.ascii_letters + string.digits
text8 = ''.join(secrets.choice(alphabet) for _ in range(6))
if not text1.isalpha() or not text2.isdigit() or len(text3) == 0:
    unfilled_add.grid(row=7, column=2)
```

Username is created by removing all spaces between the employee's name. Password is created by generating 6 random characters using a 'for loop'.

Figure 14. Pop-up Window displaying Username and Random Password Upon Employee Entry



The password can be copied and shared with employees.

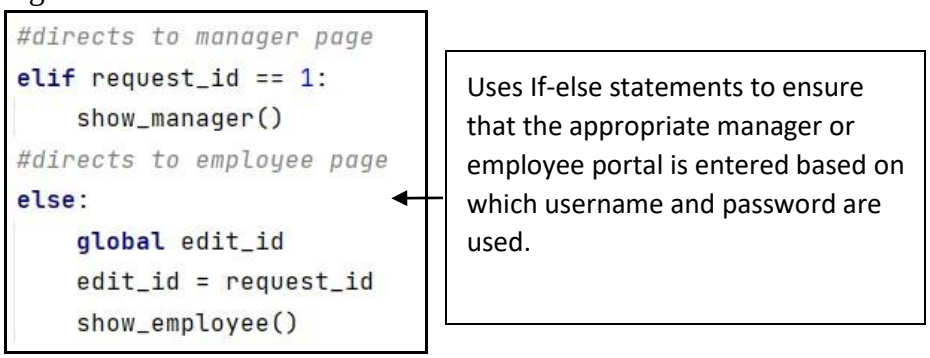
Figure 15. Change Password Functionality

```
# changing the password of an employee
def create_password():
    original = og_password.get()
    new = created_password.get()
    if len(original) != 0 or len(new) != 0:
        # creates an instance of IdFunctions class
        pass_changer = functions.IdFunctions(edit_id)
        returned = pass_changer.change_password(original=original, new_password=new)
        # Checks whether original password entered is correct
        if not returned:
            tkinter.Label(new_password, text="Incorrect Original Password", bg='white',
                          font=("Franklin Gothic Book", 20), fg='red').grid(row=5, column=2)
        else:
            for widget in new_password.winfo_children():
                if isinstance(widget, tkinter.Entry):
                    widget.delete(0, tkinter.END)
            show_employee()
```

6. Role-Based Access Control System

The solution provides certain privileges to authorized users, for example managers, that users in a different role, such as employees, don't have. The entered username and password determine the access provided.

Figure 16. Role Based Access



7. Linear Search Algorithm

Since the employee list is unsorted, a linear search is used to build the search function. The function uses a descending number of characters from the target to find the closest matches first and similar matches next. This allows the function to find employees even if their spelling is not accurate as per the client's requirements.

Figure 17. Search Function

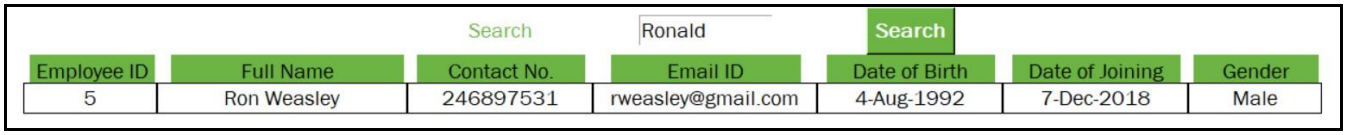
```
# using target from user input, searches for all matching employees
def searcher(target: str) -> list:
    my_cursor.execute("SELECT * FROM people")
    array = my_cursor.fetchall()
    del array[0]
    findings = []
    # searches for names in reverse order of characters
    length = len(target)
    for i in reversed(range(length)):
        # splits the characters of the String name
        to_find = target[:i+1]
        for item in array:
            if to_find.casefold() == item[1][:i+1].casefold():
                if item[0] not in findings:
                    findings.append(item[0])
    return findings
```

Using a reverse loop, the names are searched by assessing whether each progressive letter matches the search value. Similar matches are shown since even the 0th index match is printed.

```
# search employee page construction
def searcher():
    for entry in search.winfo_children():
        entry.destroy()
    if search_value.get() is None:
        pass
    else:
        search.overrideredirect(True)
        search.deiconify()
        tkinter.Label(search, text='', width=22, bg="white").grid(column=0)

        # calls the linear search algorithm
        returned = functions.searcher(search_value.get())
        # gets all employee details for the matching value as search
        required_people = functions.read_ids(returned)
        # creating the table of searched employees
        for a in range(len(required_people)):
            for b in range(7):
                if b == 0 or b == 6:
                    employee_entry = tkinter.Entry(search, font=("Franklin Gothic Book", 15),
                                                   borderwidth=1, relief="solid", width=10,
                                                   justify='center')
                    employee_entry.insert(0, required_people[a][b])
```

Figure 18. Sample search whereby similar matches can be found and all other employees are hidden from view



8. Lists and Tuples

Data structures such as lists and tuples are used for reading and unpacking data from MySQL. Lists are used to store the data of multiple employees. Their length is changeable, so a desired number of employees can be stored. Tuples are used to store the characteristics of each employee. As they are immutable (i.e., they have a constant set of values), they are faster for retrieving large amounts of data.

Figure 19. Tuples in a List to Read Budget Table

```
used = 0
salary = 0
extra = 0
all_values = my_cursor.fetchall()
for value in all_values:
    if value[2] is not None:
        salary += int(value[1])
        extra += int(value[2])
        used += int(value[1])
        used += int(value[2])
    else:
        salary += int(value[1])
        used += int(value[1])
```

Figure 20. Use of Tuples to Read Values from People Table

```
# Read values from the database and returns them wherever called
def collect_employee_details(self):
    my_cursor.execute("SELECT * FROM people WHERE employee_id=%s", (self.task_id, ))
    details = my_cursor.fetchall()[0]
    name = details[1]
    phone = details[2]
    email = details[3]
    dob_day = details[4].split("-")[0]
    dob_month = details[4].split("-")[1]
    dob_year = details[4].split("-")[2]
    doj_day = details[5].split("-")[0]
    doj_month = details[5].split("-")[1]
    doj_year = details[5].split("-")[2]
    sex = details[6]
    return name, phone, email, dob_day, dob_month, dob_year, doj_day, doj_month, doj_year, sex
```

Figure 21. Use of a List and Tuples to Unpack Data onto GUI

```
# creation of a table to display leaves to manager portal
all_leaves = functions.read_leaves()
for i in range(len(all_leaves)):
    for j in range(5):
        leave = tkinter.Entry(show_leaves, font=("Franklin Gothic Book", 15),
                              borderwidth=1, relief="solid", width=15)
        leave.grid(row=i + 3, column=j+1)
        leave.insert(0, all_leaves[i][j])
```

Employee ID	Full Name	Contact No.	Email ID	Date of Birth	Date of Joining	Gender
3	Hari Sharma	123456789	harish@ggmail.com	-12	-3	Male
4	Hermione Granger	135798642	hgranger@gmail.com	4-Sep-2000	5-Jan-2019	Female
5	Ron Weasley	246897531	rweasley@gmail.com	4-Aug-1992	7-Dec-2018	Male
8	Ninja Turtle	234134568	nturtle@gmail.com	1-Dec-1960	9-Mar-2022	Others
9	Harry Potter	12345678	hpotter@gmail.com	1-Jan-1960	1-Jan-2015	Male

9. Text File

A text file is used to store the total budget since only a single value needs to be written to for operations. Additionally, they take much less storage than tables, making the solution efficient.

Figure 22. Reading and Writing Using Text File

```
# writing or updating the total budget
def update_total_budget():
    write = open("total_budget.txt", "w")
    write.write(total_budget.get())
    write.close()
```

```
# reading the total budget from the text file
read = open("total_budget.txt", "r")
total_budget = int(read.read())
```

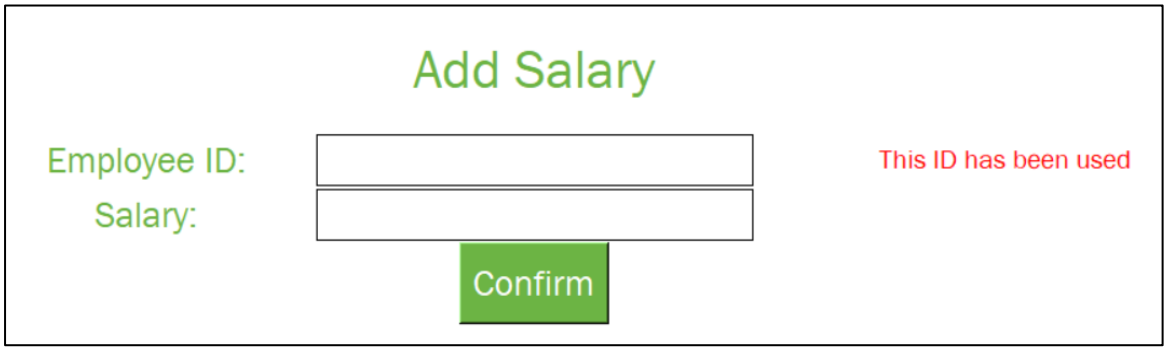
10. Error Handling Techniques

In addition to checking errors during the authentication of users, various other errors are prevented and handled throughout the process. Some are discussed below.

10.1. Employee Salary Double Entry Check

To ensure that the same employee's salary is not entered twice, the employee ID is compared first with the "budget" table to ensure the employee is not already present. If they are not, then the ID is compared with the "people" table to ensure that the employee exists. If either one of these criteria is not met, an error message is displayed.

Figure 23. Validating Employee Salary Entry



```python
# adds the employee salaries in the database
def add_budget(employee_id, salary):
    my_cursor.execute("SELECT * FROM budget")
    budget_values = my_cursor.fetchall()
    if len(budget_values) != 0:
        added_budget_ids = []
        for value in budget_values:
            added_budget_ids.append(value[0])

        for value in added_budget_ids:
            if int(employee_id) == int(value):
                return 0

    my_cursor.execute("SELECT * FROM people")
    people_values = my_cursor.fetchall()
    people_ids = []
    for value in people_values:
        people_ids.append(int(value[0]))

    if int(employee_id) not in people_ids:
        return 0
```

Ensures that the salary being entered is for an employee who does not already have a recorded salary value

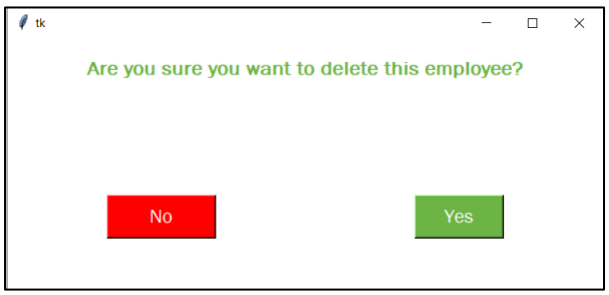
10.2. Incorrect Deletion of Employee

To increase usability of the code with familiar GUI components, a pop-up box to confirm the deletion of an employee is shown. Clicking on “Yes” redirects to the delete_confirm function while “No” continues to show_all employees.

Figure 24. Deletion Confirmation

```python
# creating delete confirmation pop-up

tkinter.Label(confirmation, text="Are you sure you want to delete this employee?")
tkinter.Button(confirmation, text="No", command=show_all)
tkinter.Button(confirmation, text="Yes", command=delete_confirm)
```



11. Graphical User Interface Creation

Various options of the Tkinter core library were used to create the 19 individual GUI pages.

11.1 Grid Method

The grid() method is an interface creation option whereby all widgets (mainly Labels, Buttons, and Entries) are arranged in a tabular format. Each widget has a corresponding row and column position on the grid. The grid adjusts the position of corresponding rows/columns based on widget size, making GUI creation and modification a less time-consuming process.

Figure 25. Usage of .grid() to Place Labels and Buttons on Specific Rows and Columns

```
# Creating Main Budget page GUI
reader = open("total_budget.txt", "r")
fill_budget = reader.read()
tkinter.Label(budget, text="Budget Management", font=("Franklin Gothic Book", 30, "bold"), bg="white",  fg="#6cb444").grid(row=1, column=1)
tkinter.Label(budget, text="", height=2, bg="white").grid(row=2)
total_budget_frame = tkinter.Frame(budget, bg="white")
total_budget_frame.grid(row=3, column=1, padx=20)
tkinter.Label(total_budget_frame, text="Total Budget:", font=("Franklin Gothic Book", 20), bg="white", fg="#6cb444").pack(side=tkinter.LEFT)
total_budget = tkinter.Entry(total_budget_frame, font=("Franklin Gothic Book", 20), borderwidth=1, relief="solid")
total_budget.pack(side=tkinter.LEFT, padx=20)
total_budget.insert(0, fill_budget)
tkinter.Button(total_budget_frame, text="Confirm", font=("Franklin Gothic Book", 20), command=update_total_budget, bg="#6cb444", fg="white")\
    .pack(side=tkinter.LEFT)
tkinter.Label(budget, text="", bg="white").grid(row=4)
tkinter.Button(budget, text="Add Salary", font=("Franklin Gothic Book", 20), fg="white", command=show_add_budget, bg="#6cb444", width=20)\
    .grid(row=5, column=0, sticky="e")
tkinter.Button(budget, text="Update Costs", font=("Franklin Gothic Book", 20), fg="white", command=show_update_budget, bg="#6cb444", width=20)\
    .grid(row=5, column=2, sticky="w")
tkinter.Label(budget, text="", bg="white").grid(row=6)
tkinter.Button(budget, text="Add Extra Costs", font=("Franklin Gothic Book", 20), fg="white", command=show_extra_costs, bg="#6cb444", width=20)\
    .grid(row=7, column=0, sticky="e")
tkinter.Button(budget, text="Budget Calculations", font=("Franklin Gothic Book", 20), fg="white", command=budget_show_all, bg="#6cb444", width=20)\
    .grid(row=7, column=2, sticky="w")
tkinter.Label(budget, text="", bg="white").grid(row=8)
tkinter.Button(budget, text="Back", command=show_manager, height=2, width=7, bg="dark gray").grid(column=5, row=0, sticky="n", padx=10, pady=4)
tkinter.Button(budget, text="X", width=7, height=2, bg="red", command=del_gui).grid(row=0, column=6, sticky="n", pady=4)
```

11.2 Image Insertion

The ABC logo is added on all pages. For this, an image was inserted using the Canvas widget. The Pillow (PIL) library was imported to avail the ImageTk and Image functionalities.

Figure 26. Addition of Image using Canvas, ImageTK, and Image


```
# adding an image on the top left of the page
extra_costs_canvas = tkinter.Canvas(extra_costs, width=350, height=150, highlightthickness=0, bg="white")
extra_costs_canvas.grid(row=0, column=0, sticky="w")
extra_costs_img = (Image.open("placeholder.jpg"))
extra_costs_new_image = ImageTk.PhotoImage(master=extra_costs_canvas, image=extra_costs_img)
extra_costs_canvas.create_image(40, 30, anchor="nw", image=extra_costs_new_image)
```

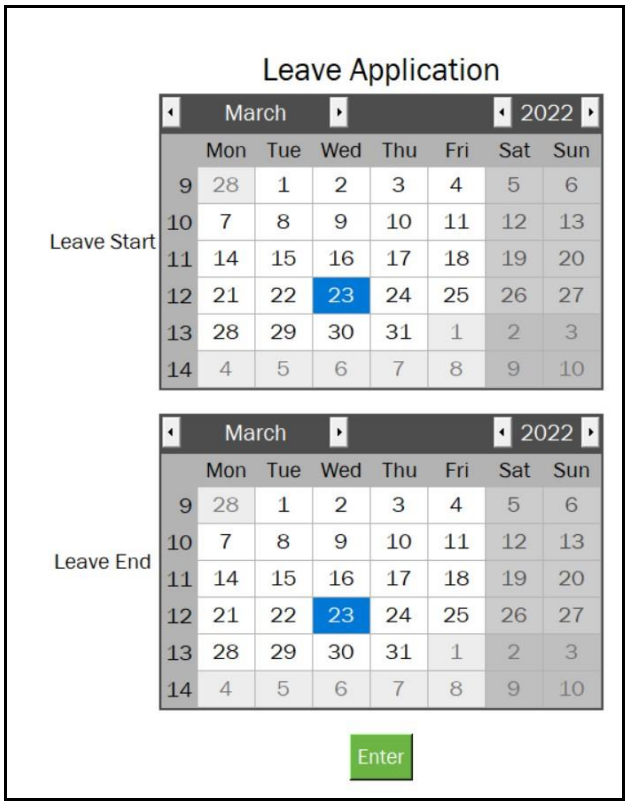
11.3 Calendar Creation

Calendars for leave entry are created using the TkCalendar library. This makes the interface more user-friendly. The calendars are interactive, and the dates selected are stored in the database.

Figure 27. Creation of Calendar for Leave Start Date

```
# reads the current date
day = datetime.now().day
month = datetime.now().month
year = datetime.now().year
# creation of calendar to enter start date of leaves
tkinter.Label(cal_tk, text="", width=65, height=2, bg="white").grid(row=0, column=0)
cal_tk_canvas = tkinter.Canvas(cal_tk, width=350, height=150, highlightthickness=0, bg="white")
cal_tk_canvas.grid(row=0, column=0, sticky="w")
cal_tk_img = (Image.open("placeholder.jpg"))
cal_tk_new_image = ImageTk.PhotoImage(master=cal_tk_canvas, image=cal_tk_img)
cal_tk_canvas.create_image(40, 30, anchor="nw", image=cal_tk_new_image)
tkinter.Label(cal_tk, text="Leave Application", font=("Franklin Gothic Book", 22), bg="white").grid(row=1, column=2)
tkinter.Label(cal_tk, text="Leave Start", font=("Franklin Gothic Book", 15), bg="white").grid(row=2, column=1)
cal1 = Calendar(cal_tk, selectmode='day', year=year, month=month, day=day, font=("Franklin Gothic Book", 15))
cal1.grid(row=2, column=2)
```

Figure 28. Calendar GUI

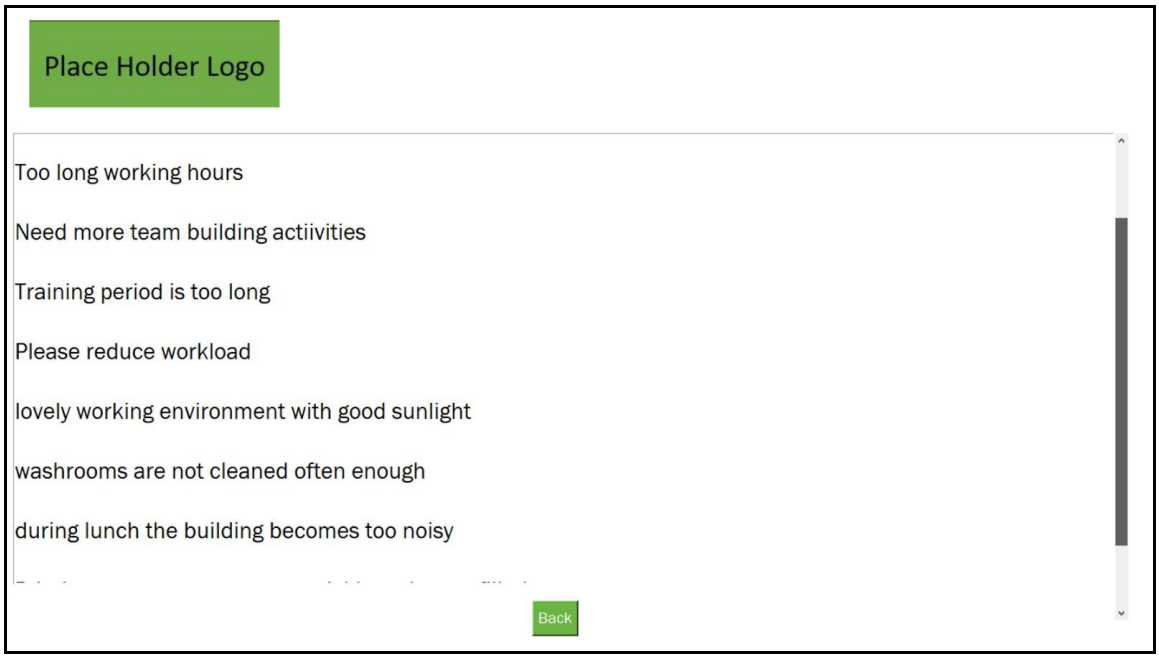


11.4 Scroll Bar Provision

Figure 29. Configuring Scrollbar onto the Textbox

```
# creating scroll bar and textbox for feedback
s = tkinter.Scrollbar(frame, troughcolor='red')
s.pack(side=tkinter.RIGHT, fill=tkinter.Y, padx=(0, 20), pady=20)
t = tkinter.Text(frame, height=15, wrap=tkinter.WORD,
                 yscrollcommand=s.set, font=("Franklin Gothic Book", 22))
t.pack(side=tkinter.TOP, fill=tkinter.X, pady=20, padx=(20, 0))
# configuring the scrollbar onto the textbox
s.config(command=t.yview)
# inserting feedback in the textbox
for o in read_feedback:
    t.insert(tkinter.END, str(o[1]) + "\n\n")
```

Figure 30. Scrollbar to View All Feedback Easily



11.5 Drop Down Menus

Drop Down Menus are created using the .OptionMenu method of Tkinter.

Figure 31. Using OptionMenu to create date of joining drop down menus

```
tkinter.OptionMenu(edit_doj_frame, edited_month_doj, *months).pack(side=tkinter.LEFT)
tkinter.OptionMenu(edit_doj_frame, edited_day_doj, *days).pack(side=tkinter.LEFT, padx=64)
tkinter.OptionMenu(edit_doj_frame, edited_year_doj, *doj_years).pack(side=tkinter.LEFT)
```

Figure 32. Drop Down Menu to enter the month of joining the company

12. First in First Out (FIFO) Queue structure for feedback

The feedback which is entered first is given the 0th index position of the list. While printing this list on the feedback page in the manager portal, a queue structure is used to ensure that the first entered feedback is visible at the top.

Figure 33. Inserting feedback from read_feedback list to the feedback textbox

```
# reads feedback from the database
def read_feedback() -> list:
    my_cursor.execute("SELECT * FROM feedback")
    return my_cursor.fetchall()
```

```
# inserting feedback in the textbox
for o in read_feedback:
    t.insert(tkinter.END, str(o[1]) + "\n\n")
```

Word Count: 956

Citations:

"Basics for Displaying Image in Tkinter Python." C# Corner, https://www.csharpcorner.com/blogs/basics-for-displaying-image-in-tkinter-python.

GregGreg 47322 gold badges66 silver badges2121 bronze badges, et al. "Python Username and Password Program." Code Review Stack Exchange, 1 Mar. 1965, https://codereview.stackexchange.com/questions/164359/python-username-and-passwordprogram.

"How to Connect Python with SQL Database?" GeeksforGeeks, 30 Sept. 2021, https://www.geeksforgeeks.org/how-to-connect-python-with-sql-database/.

"How to Create a Mysql Relational Database." HOW TO CREATE A MySQL RELATIONAL DATABASE (PART 1), http://www.richmediacs.com/user_manuals/Dreamweaver/RMCS_CMS/CreatingA_MySQL_DB.html.

Python - Tkinter Scrollbar, https://www.tutorialspoint.com/python/tk_scrollbar.htm.

Python Generate Random String and Password - Pynative.
https://pynative.com/pythongenerate-random-string/.

“Python Mysql Insert into Table.” Python MySQL Insert Into,
https://www.w3schools.com/python/python_mysql_insert.asp.

“Python Tuple.” Programiz, https://www.programiz.com/python-programming/tuple.

“Tkinter Grid Geometry Manager.” Python Tutorial - Master Python Programming For Beginners from Scratch, 14 Jan. 2021, https://www.pythontutorial.net/tkinter/tkinter-grid/.