Simple Employee Management System

Anshul Girish Dhavale

Vit Bhopal university

Contents:

Introduction

The Simple Employee Management system is a lightweight CLI-driven application designed to be scalable to small businesses to use in data management of employee records. Made with python and Pandas it ensures a error-free way of redundantly and efficiently saving employee data. Furthermore the code is made to be easily scalable with adding further rows and columns for various data types.

Problem Statement

Small businesses still use pen and paper, or convoluted spreadsheets to keep logs on employee records. This approach leads to several issues:

1. Data Redundancy: There can be duplicate entries for a single person, or multiple persons having the same entry.
2. ID Conflict: Assigning Ids manually is prone to human fault
3. Lack of Security: Unsecure/public spreadsheets or paper is prone to easy editing/vandalism without admin privelages.
4. Inefficiency: Manually searching for every persons on the list is time consuming and redundant.

This program addresses these issues by automating the CRUD (Create, Read, Update) process in a secured CLI environment.

Functional Requirements

This program pertains to the following functional requirements:

- *Add Employee:* The program allows managers to input an employee's name and automatically assigns a unique, random 4-digit ID.
- *Update Employee*: The program allows updates to an employee's name based on their ID, but only after successful administrator password authentication.
- *View All Employees*: The program displays a formatted list of all registered employees.
- *Data Persistence*: All data is saved to employees.csv immediately after modification.
- *Duplicate Prevention:* The ID generation algorithm checks existing records to ensure no ID is reused

Non-functional Requirements:

-*Usability*: The Command Line Interface (CLI) needs to be simple, with menu-driven navigation and unambiguous prompts.

-*Reliability*: To avoid crashes, the system must confirm the existence of the file before attempting read operations.

-*Performance*: For datasets with fewer than 10,000 records, operations like adding or searching for a user should appear instantly.

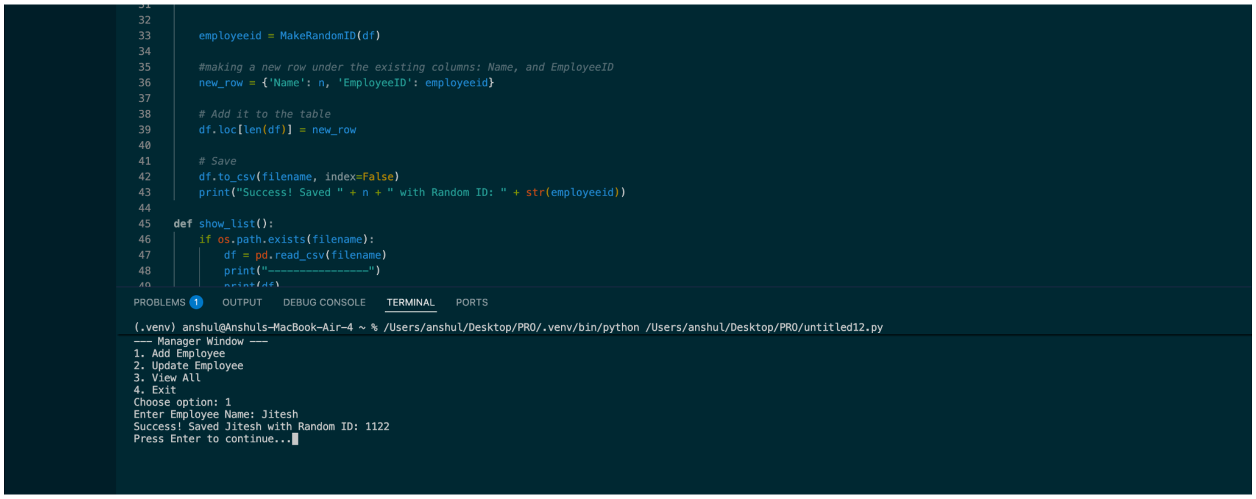
-*Portability*: Any system that has Pandas and Python installed should be able to run the script.

System Architecture:

- *Presentation Layer*: The user interface is provided by the Python functions "input()" and "print()."
- *Logic Layer*: Business logic and ID generation constraints are handled by Python functions (add_user, update_user, MakeRandomID).
- *Data Layer:* The Data logic is handled by the Pandas library.

Design Diagrams

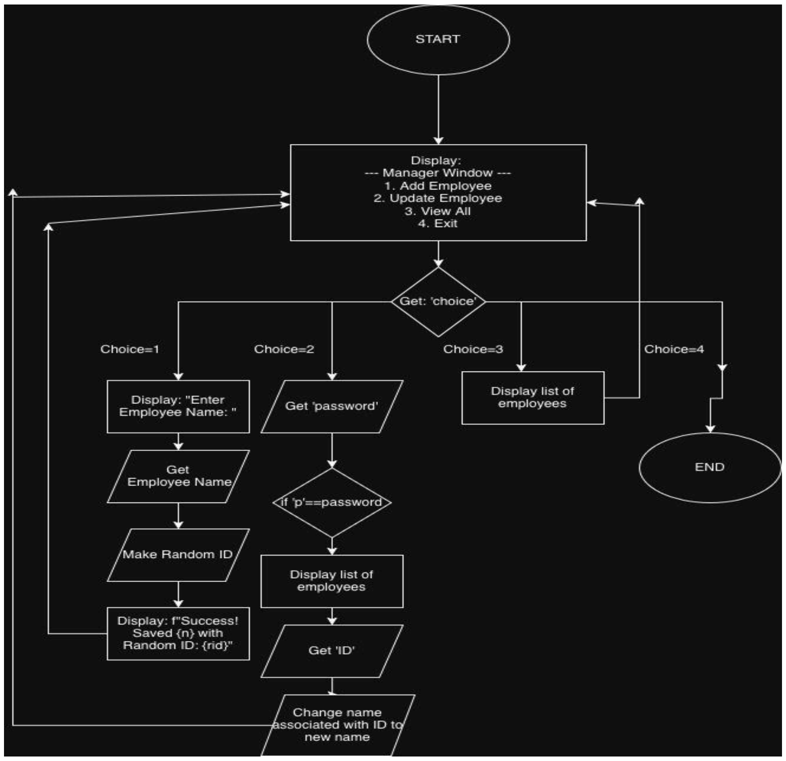
1. **Use Case Diagram**



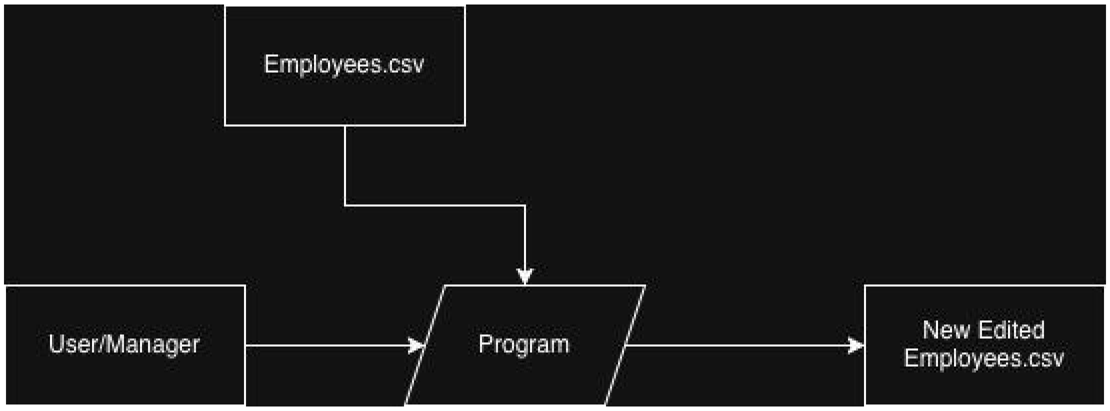
```
    employeeid = MakeRandomID(df)

    #making a new row under the existing columns: Name, and EmployeeID
    new_row = {'Name': n, 'EmployeeID': employeeid}

    # Add it to the table
    df.loc[len(df)] = new_row

    # Save
    df.to_csv(filename, index=False)
    print("Success! Saved " + n + " with Random ID: " + str(employeeid))

def show_list():
    if os.path.exists(filename):
        df = pd.read_csv(filename)
        print("---------------")
```

```
(.venv) anshul@Anshuls-MacBook-Air-4 ~ % /Users/anshul/Desktop/PRO/.venv/bin/python /Users/anshul/Desktop/PRO/untitled12.py
--- Manager Window ---
1. Add Employee
2. Update Employee
3. View All
4. Exit
Choose option: 1
Enter Employee Name: Jitesh
Success! Saved Jitesh with Random ID: 1122
Press Enter to continue...
```

2. Workflow Diagram:



3. Sequence Diagram:

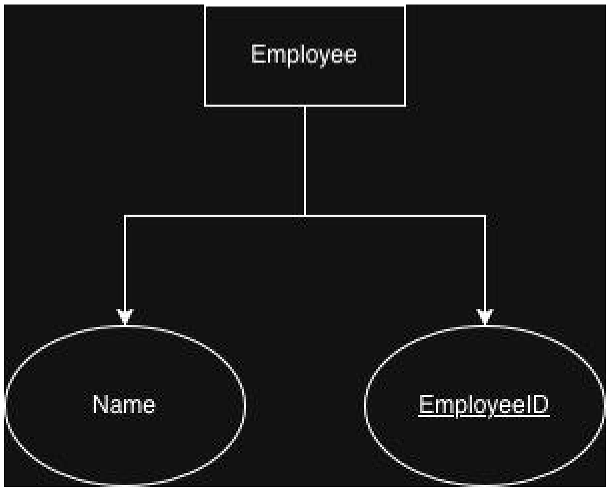


4. **Component Diagram:**

   Main program:

   - **Manager window (add/update/view/exit)**
   - **Password prompt.**
   - **Final push to csv file outside python code.**

5. **Entity Relationship diagram:**



Design Decisions & Rationale:

1. ***Python and Pandas:*** Python was selected due to its capacity for quick prototyping. Because Pandas manages data alignment, pretty-printing, and tabular read/write operations more robustly than the native csv module, it was chosen.
2. ***CSV Storage:*** Due to the small project scope and the fact that non-technical users can access the database in Excel if needed, a CSV file was selected over a SQL database.

3. ***Random IDs vs. Incremental IDs:*** Unlike straightforward auto-incrementing integers, random IDs were selected to offer a non-guessable identifier and to show algorithmic logic.

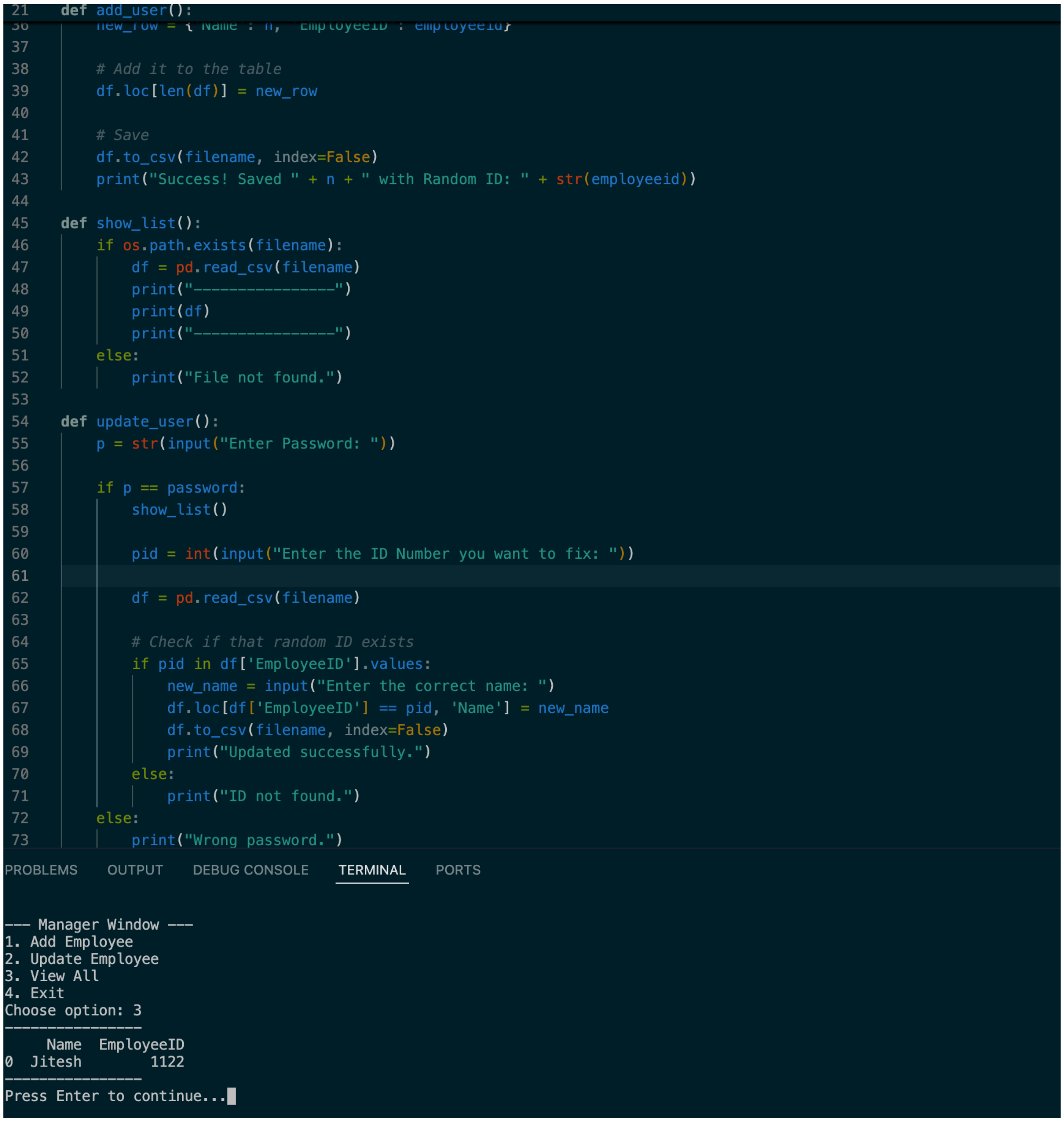
Specifics of Implementation:

The MakeRandomID function contains the essential logic. To guarantee uniqueness, it uses a while loop:

```python
def MakeRandomID(df):
    #making a random employee id
    rid = random.randint(1000, 9999)

    if len(df) > 0:
        while rid in df['EmployeeID'].values:
            rid = random.randint(1000, 9999)

    return rid
```

In order to improve user experience in between menus, the Main Menu uses os.system to clear the screen while operating inside a while True loop.

Screenshots:



```python
def add_user():
    new_row = {"Name": n, "EmployeeID": employeeid}

    # Add it to the table
    df.loc[len(df)] = new_row

    # Save
    df.to_csv(filename, index=False)
    print("Success! Saved " + n + " with Random ID: " + str(employeeid))

def show_list():
    if os.path.exists(filename):
        df = pd.read_csv(filename)
        print("-----------------")
        print(df)
        print("-----------------")
    else:
        print("File not found.")

def update_user():
    p = str(input("Enter Password: "))

    if p == password:
        show_list()

        pid = int(input("Enter the ID Number you want to fix: "))

        df = pd.read_csv(filename)

        # Check if that random ID exists
        if pid in df['EmployeeID'].values:
            new_name = input("Enter the correct name: ")
            df.loc[df['EmployeeID'] == pid, 'Name'] = new_name
            df.to_csv(filename, index=False)
            print("Updated successfully.")
        else:
            print("ID not found.")
    else:
        print("Wrong password.")
```

```
--- Manager Window ---
1. Add Employee
2. Update Employee
3. View All
4. Exit
Choose option: 3
-----------------
     Name  EmployeeID
0  Jitesh        1122
-----------------
Press Enter to continue...
```

Testing:

1. ***File Absence:*** Verified that the system handles the initial run (when employees.csv does not exist) without crashing.

2. ***Incorrect Password:*** Tested update_user with incorrect passwords to ensure access is denied.
3. ***Data storage:*** Verified that the Employee ID is stored as an integer and the Name as a string.

Learnings & Key Takeaways:

- **Pandas for Persistence:** Acquired knowledge of using df.to_csv() and pd.read_csv() as a basic database engine.
- **State Management:** Recognized the significance of verifying the current state (existing IDs) prior to creating new data.

Challenges faced:

Managing the file stream was one of the major challenges. The read operation would first fail if the file didn't exist. The addition of os.path.exists() checks fixed this. Making sure the clear screen command functions on various operating systems presented another difficulty because Windows requires cls while Unix/Linux uses clear.

Future Enhancements:

- ***Delete Functionality:*** Add option 5 to remove an employee from the CSV.
- ***GUI Implementation:*** Migrate the code to Tkinter or PyQt for a windowed application.