Assignment Cover Sheet

Qualification		**Module Number and Title**
HD in Computing/HD in Software Engineering		**Object Oriented Programming- CSE4006**
Student Name & No.		**Assessor**
W.H pramod ravindu gimhana soysa st20251283		Mrs.Upeka Wijesinghe
Hand out date		**Submission Date**
17.03.2023		26.04.2023
Assessment type WRIT1-Coursework	**Duration/Length of Assessment Type** 3000 words equivalent	**Weighting of Assessment** 100%

Learner declaration
I, W.H pramod ravindu gimhana soysa - st20251283, certify that the work submitted for this assignment is my own and research sources are fully acknowledged.

Marks Awarded			
First assessor			
IV marks			
Agreed grade			
Signature of the assessor		Date	

FEEDBACK FORM

INTERNATIONAL COLLEGE OF BUSINESS & TECHNOLOGY

Module:

Student:

Assessor:

Assignment:

Strong features of your work:

Areas for improvement:

Marks Awarded:

st20251283_CSE4006_Writi.pdf

ORIGINALITY REPORT

4%	0%	2%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Bipin Joshi. "Beginning Database Programming Using ASP.NET Core 3", Springer Science and Business Media LLC, 2019 Publication	2%
2	Submitted to University of Wales Institute, Cardiff Student Paper	1%
3	dsns.csie.nctu.edu.tw Internet Source	<1%
4	Submitted to NCC Education Student Paper	<1%
5	Submitted to Regis University Student Paper	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

Table of Contents

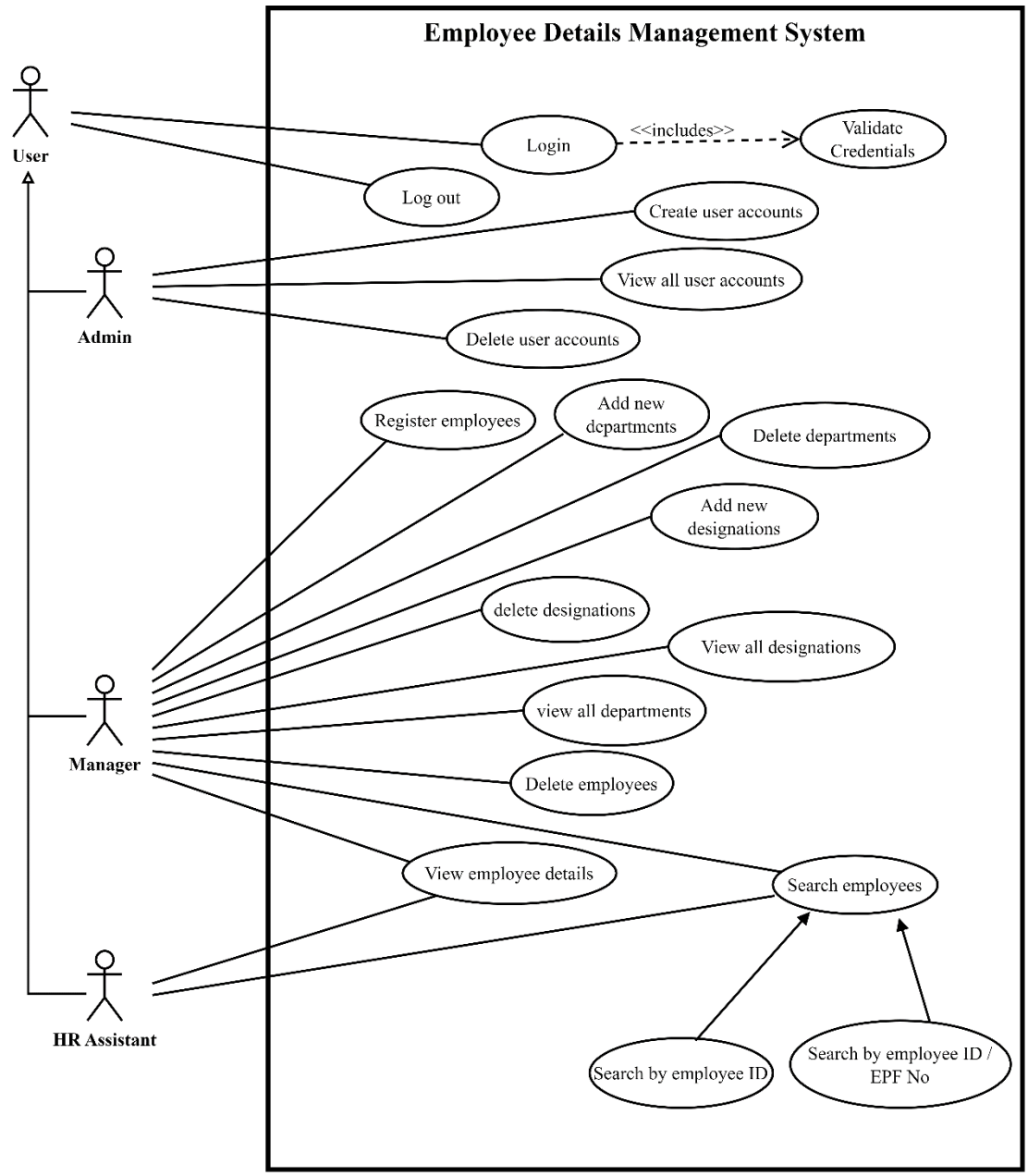
1. Acknowledgement

I would like to express my sincere gratitude to my lecturer, Mrs.Upeka Wijesinghe for providing the guidance throughout the module. As a result of this assignment, now I have a deeper understanding in Object oriented programming and object oriented concepts and how to develop a basic desktop software using object oriented concepts using Java programming language. Now I have updated knowledge and skills in object oriented programming that I can use in my future academic and professional life. I am glad that I had the chance to do this assignment. So, I would like to express my gratitude to my lecturer again.

2. Introduction

This report outlines the creation of a software system according to a given scenario. The software was created to provide end users with a user-friendly interface and to satisfy the requirements of the scenario. At first, the report includes diagrams such as use case, class and sequence diagrams depict the software's architecture and behavior. Then the documentation of the software describes how the object oriented programming concepts such as inheritance, encapsulation, polymorphism and abstraction used throughout the development process to develop a software that is scalable, adaptable and simple to maintain. Then the major functionalities of the software system are described in the report. Finally, a user manual for the software is included in the report to assist the end users in using the software.

3. Use case diagram



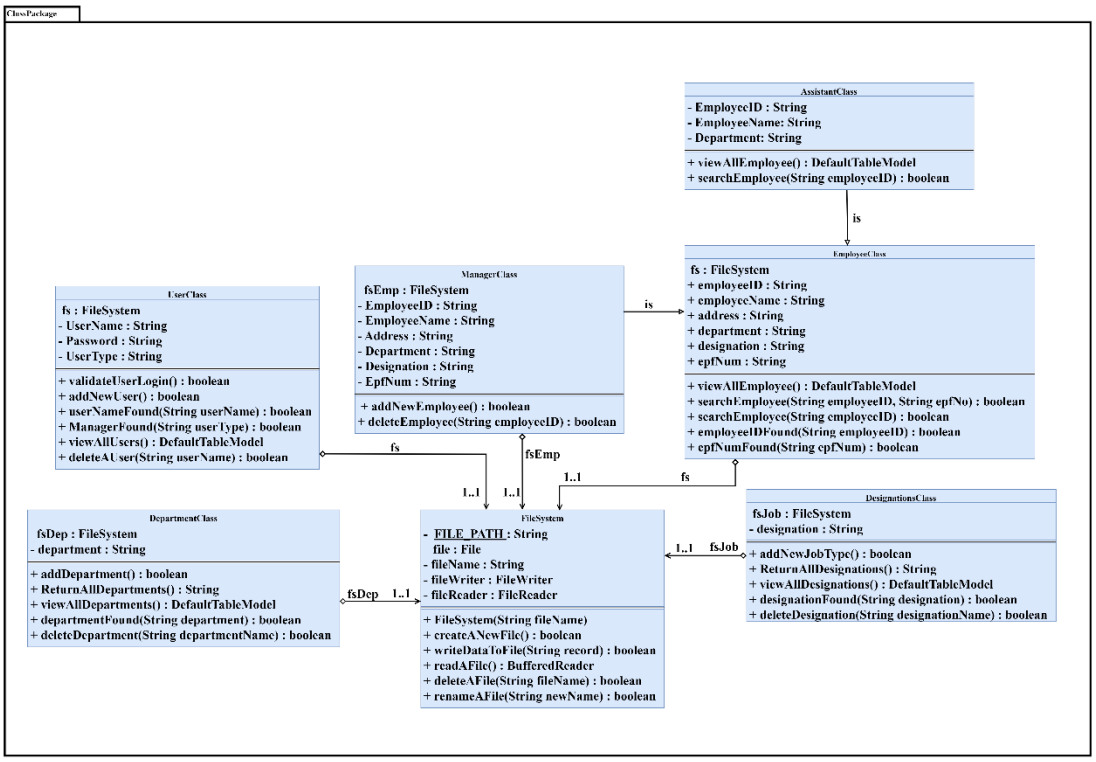
4. Assumptions

1. Only the administrator can manage user accounts.
2. Only the manager can register employees, add new departments and designations and also delete those details.
3. Manager can search employee details by providing both employee ID or EPF No, while HR assistant can search employee details by only employee ID.

5. Class Diagrams

Note: All the properties and behaviors such as public getter and setter methods are not included in the class diagrams. Class diagrams just describing the relationships between classes.

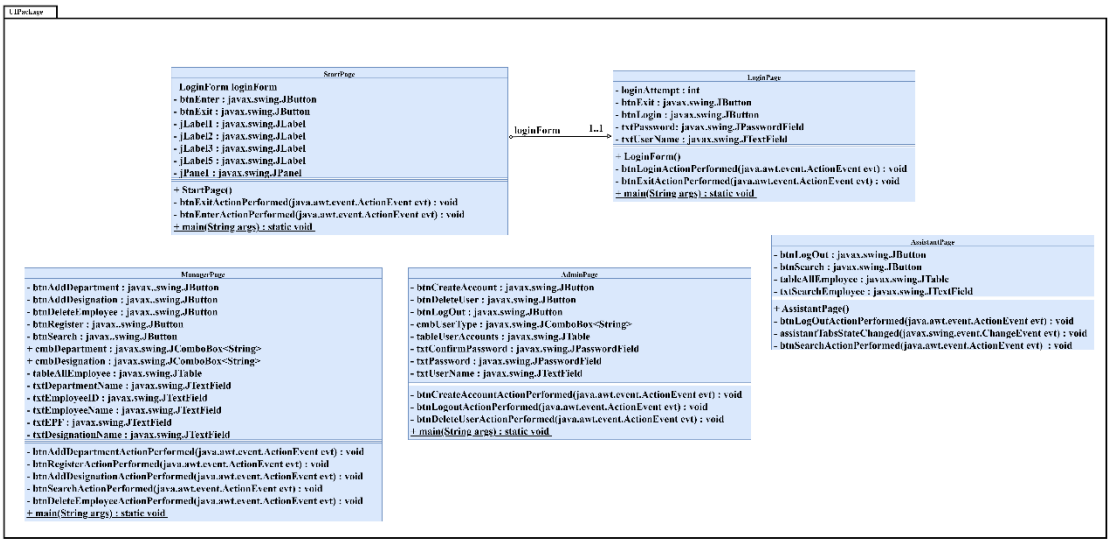
5.1. Class diagram of ClassPackage



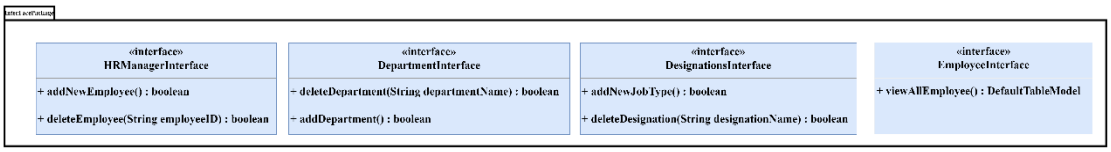
"ManagerClass" and "AssistantClass" inherit the members of "EmployeeClass".The "viewAllEmployee()" method in "EmployeeClass" is also implemented in the "AssistantClass" by method overriding. Inheritance and polymorphism concepts used in "AssistantClass".

"FileSystem" class is a part of all other classes except "AssistantClass". "FileSystem" class has an aggregation relationship with its relating classes.

5.2. Class diagram of UIPackage

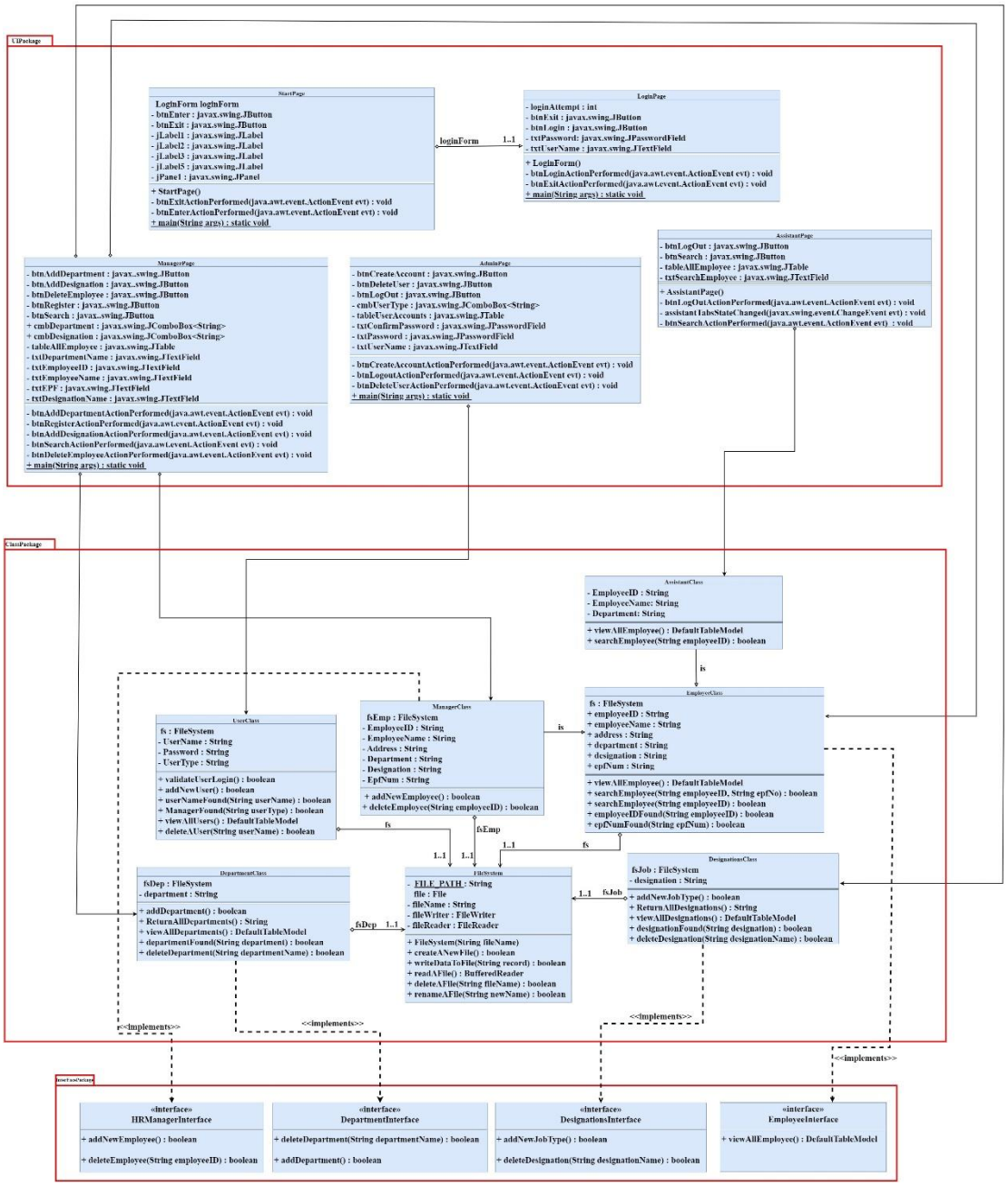


5.3. Class diagram of InterfacePackage

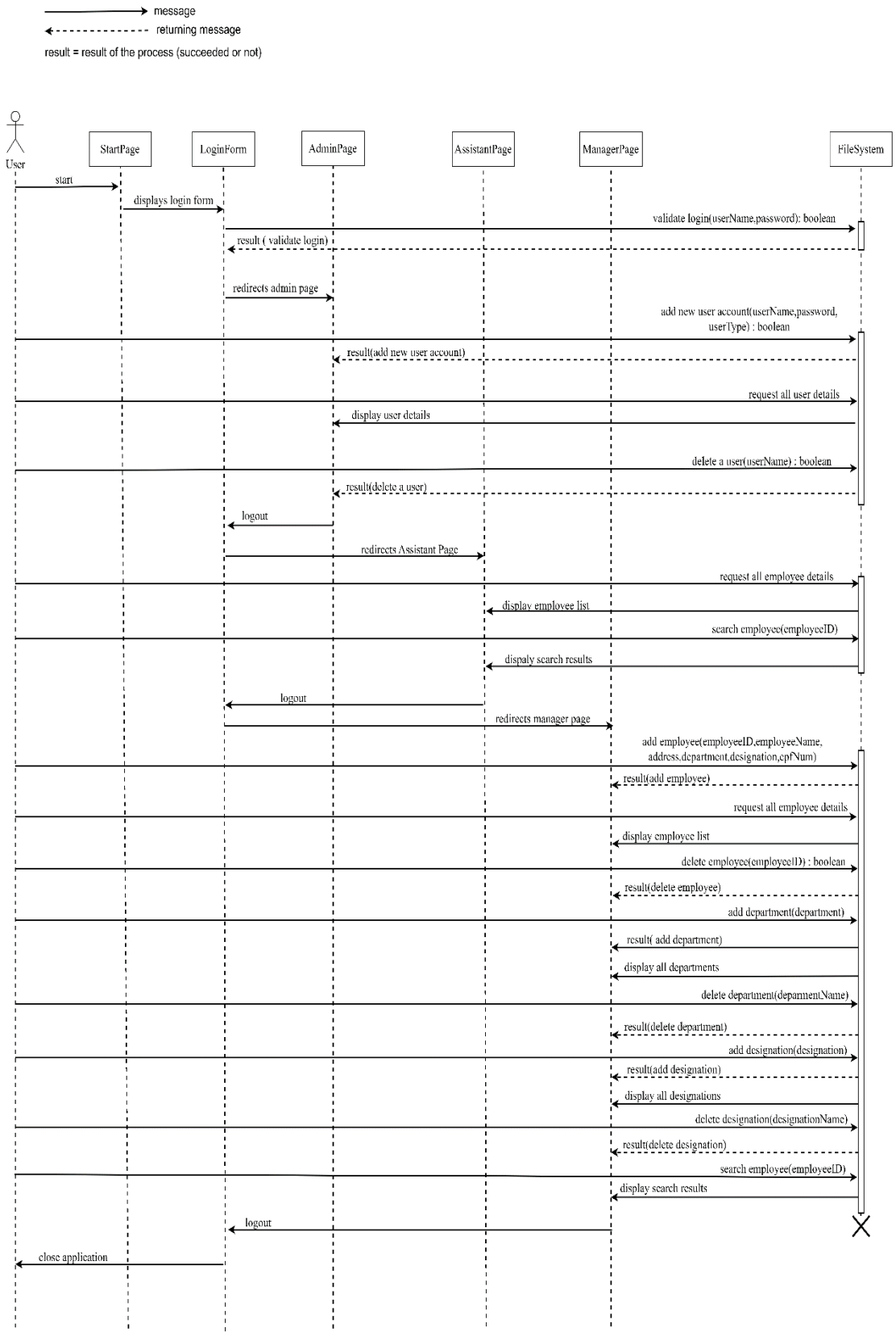


.

5.4. Complete class diagram



6. Sequence diagram



7. System documentation

The system is developed using four source packages. They are “ClassPackage, Interface Package, UIPackage and ImagesAndIcons”.

7.1. Object oriented concepts used in the system

Object

An object is an instance of a class. All the properties and behaviors in a class can be used through the object.

```
public class EmployeeClass implements EmployeeInterface {

    FileSystem fs = new FileSystem("Employee_Details.txt");

    public String employeeID;
    public String employeeName;
    public String address;
    public String department;
    public String designation;
    public String epfNum;
```

creating a new object using **new** keyword

Creating an object for the “EmployeeClass” in a different class.

```
FileSystem fsEmp = new FileSystem("Employee_Details.txt");
```

creating a new object for the FileSystem

```
EmployeeClass employeeClass = new EmployeeClass();
```

creating a new object for the EmployeeClass

```
private String EmployeeID     = employeeClass.employeeID;
private String EmployeeName   = employeeClass.employeeName;
private String Address        = employeeClass.address;
private String Department     = employeeClass.department;
private String Designation    = employeeClass.designation;
private String EpfNum         = employeeClass.epfNum;
```

using the variables in EmployeeClass through the object created for EmployeeClass

With the object of the “EmployeeClass”, it allows to access all the properties and behaviors of that object.

Likewise, there are many objects used in the system.

```
public class ManagerPage extends javax.swing.JFrame {

    EmployeeClass employeeClass = new EmployeeClass();
    ManagerClass managerClass = new ManagerClass();
    DepartmentClass departmentClass = new DepartmentClass();
    DesignationsClass designationsClass = new DesignationsClass();

    DefaultTableModel departmentTableModel = new DefaultTableModel();
    DefaultTableModel designationTableModel = new DefaultTableModel();
```

The above code explains how the methods of "EmployeeClass" are being used through its object.

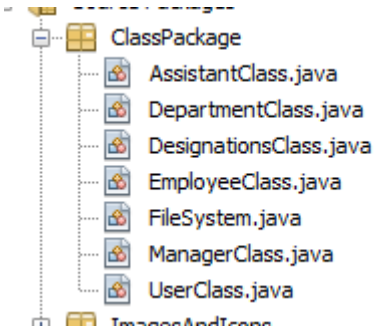
```
boolean employeeIDFound = employeeClass.employeeIDFound(employeeID);
boolean epfNumFound = employeeClass.epfNumFound(epfNum);
```

```
AdminPage adminPage=new AdminPage();
ManagerPage managerPage = new ManagerPage();
AssistantPage assistantPage = new AssistantPage();
```

This is how the objects are instantiated and the way of using the objects within the system.

Class

There are multiple classes used in the system because every class has its own properties and behaviors.



“ManagerClass” class and its properties and behaviors

```
public class ManagerClass extends EmployeeClass implements HRManagerInterface {

    public ManagerClass() {...3 lines }

    FileSystem fsEmp = new FileSystem("Employee_Details.txt");


    EmployeeClass employeeClass = new EmployeeClass();

    private String EmployeeID     =   employeeClass.employeeID;
    private String EmployeeName   =   employeeClass.employeeName;
    private String Address        =   employeeClass.address;
    private String Department     =   employeeClass.department;
    private String Designation    =   employeeClass.designation;
    private String EpfNum         =   employeeClass.epfNum;

    public ManagerClass(String employeeID, String employeeName, String address, String department,
            String designation, String epfNum) {...9 lines }

    public String getEmployeeID() {...3 lines }

    @Override
    public void setEmployeeID(String employeeID) {...3 lines }

    public String getEmployeeName() {...3 lines }

    @Override
```

“FileSystem” class and its properties and behaviors

```
public class FileSystem {

    private static String FILE_PATH = "..\\st20251283_CSE4006_Writil\\Files\\";
    File file;      //file constructor
    private String fileName;         //creating a variable to give a name to file when naming
    private FileWriter fileWriter;
    private FileReader fileReader;

    public FileSystem(String fileName) {...4 lines }

    public boolean createANewFile() {...15 lines }

    public boolean writeDataToFile(String record) {...17 lines }

    public BufferedReader readAFile() {...13 lines }

    public boolean deleteAFile(String fileName) {...14 lines }

    public boolean renameAFile(String newName) {...14 lines }

}
```

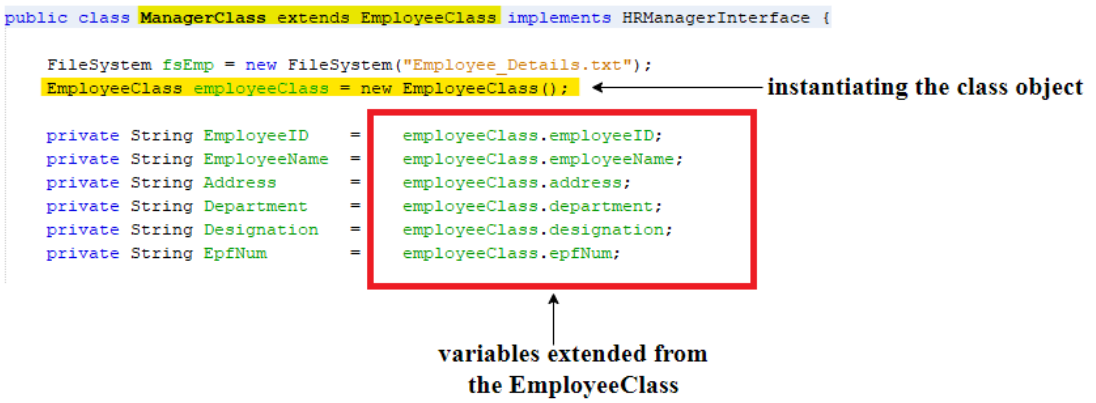
“EmployeeClass” class and its properties and behaviors

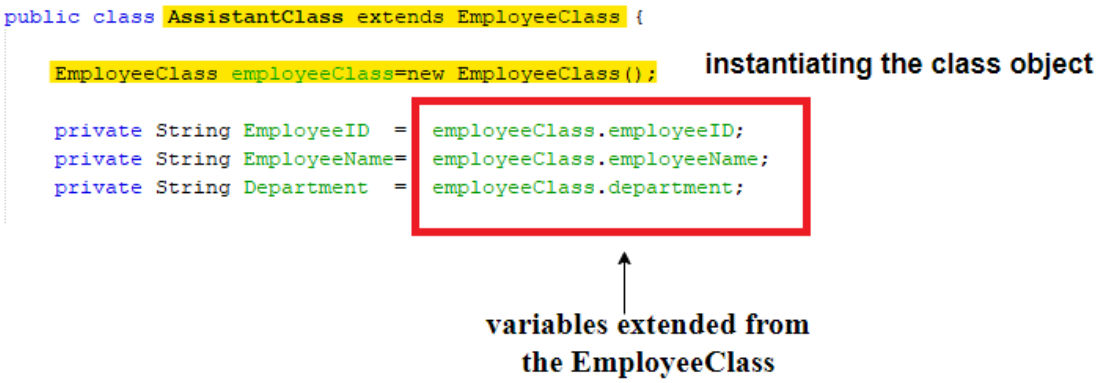
```
public class EmployeeClass implements EmployeeInterface {

    FileSystem fs = new FileSystem("Employee_Details.txt");

    public String employeeID;
    public String employeeName;
    public String address;
    public String department;
    public String designation;
    public String epfNum;

    public EmployeeClass() {...2 lines }
    public void setDepartment(String department) {...3 lines }
    public void setEmployeeName(String employeeName) {...3 lines }
    public void setAddress(String address) {...3 lines }
    public void setDesignation(String designation) {...3 lines }
    public void setEpfNum(String epfNum) {...3 lines }
    public void setEmployeeID(String employeeID) {...3 lines }

    @Override
    public DefaultTableModel viewAllEmployee() {...24 lines }

    public boolean searchEmployee(String employeeID, String epfNo) {...34 lines }

    public boolean searchEmployee(String employeeID) {...25 lines }

    public boolean employeeIDFound(String employeeID) {...26 lines }

    public boolean epfNumFound(String epfNum) {...29 lines }
}
```

Inheritance

The “AssistantClass” and “ManagerClass” extend the “EmployeeClass”, indicating that all the properties and behaviors of the “EmployeeClass” are allowable to the “AssistantClass” and “ManagerClass” .Classes in a different package also can use the properties and behaviors by importing the “EmployeeClass”.

```
public class EmployeeClass implements EmployeeInterface {

    FileSystem fs = new FileSystem("Employee_Details.txt");

    public String employeeID;
    public String employeeName;
    public String address;
    public String department;
    public String designation;
    public String epfNum;
```





The above codes show how the variables of "EmployeeClass" are accessed by two different classes.

```
public boolean searchEmployee(String employeeID, String epfNo) {
    boolean isFound = false;
    try {
        String[] emp = null;
        try ( BufferedReader bufferedReader = fs.readAFile()) {

            String line;

            while ((line = bufferedReader.readLine()) != null) {
```

The "searchEmployee" method is a method implemented in the "EmployeeClass".This method is also inherited and accessed by the "ManagerClass" through the object of the "ManagerClass".The code below code explains how it works.

```
if (managerClass.searchEmployee(employeeID, epfNo)) {

    searchResult.setText("\n \n" + "Employee ID : " + managerCla
            + "Employee Name : " + managerClass.getEmployeeName(
            + "Address : " + managerClass.getAddress() + "\n \n"
```

The above codes show how the system used the inheritance concept.

Encapsulation

Declaring variables as private makes those variables are inaccessible outside the class. This practice prevents the properties of an object from being modified or accessed from outside its class.

```
public class UserClass {

    FileSystem fs = new FileSystem("User_Info.txt");

    private String UserName;    //making private variables
    private String Password;
    private String UserType;
```

In the above code, the “UserName”, “Password” and “UserType” are private variables and cannot accessed from outside the class.

Public setter methods and getter methods are implemented to access and modify those private variables from outside the class. This is the only method to access private variables. So the modifications or access to those variables are can be done in a controlled way.

```
//setter and getter methods to access variables in other classes (ENCAPSULATION)
public String getUserName() {
    return UserName;
}

public void setUserName(String UserName) {
    this.UserName = UserName;
}

public String getPassword() {
    return Password;
}

public void setPassword(String Password) {
    this.Password = Password;
}

public String getUserType() {
    return UserType;
}

public void setUserType(String UserType) {
    this.UserType = UserType;
```

```
UserClass user = new UserClass(userName, password);

if (user.validateUserLogin() && loginAttempt < 3) {

    String userType = user.getUserType();   <-- public getter method
    JOptionPane.showMessageDialog(null, "Login succes

    if (userType.equals("Admin")) {

        adminPage.show();
        this.dispose();

    }
    if (userType.equals("Manager")) {
```

The “getUserType” method is a public getter method in the “UserClass” returning the value of the private variable “UserType”. The value of that private variable is accessed by a different class in a different package through the getter method.

Most of the classes in the system use the encapsulation concept for hiding the data from other classes. This is how the system achieved data encapsulation.

Polymorphism

Same object can be implemented in many times. But they need to differ from each other somehow. In other words same object can be implemented in many forms.

In the system, polymorphism is achieved in both the two ways of method overloading and method overriding.

Polymorphism by method overloading

```
public boolean searchEmployee(String employeeID) {
    boolean isFound = false;
    try {
        String[] emp = null;
        try (BufferedReader bufferedReader = fs.readAFile()) {
            String line;

            while ((line = bufferedReader.readLine()) != null) {
                emp = line.split(" ");

                if (emp[0].equalsIgnoreCase(employeeID)) {
                    isFound = true;
                    this.setEmployeeID(emp[0]);
                    this.setEmployeeName(emp[1]);
                    this.setDepartment(emp[3]);
                    break;
                }
            }
            bufferedReader.close();
        }
    } catch (IOException ex) {
        System.err.println("An error occurred while Searching Employee " + ex);
    }
    return isFound;
}
```

Above “searchEmployee” method only use one parameter.

```
public boolean searchEmployee(String employeeID, String epfNo) {
    boolean isFound = false;
    try {
        String[] emp = null;
        try (BufferedReader bufferedReader = fs.readAFile()) {

            String line;

            while ((line = bufferedReader.readLine()) != null) {
                emp = line.split(" ");

                if (emp[0].equalsIgnoreCase(employeeID) || emp[5].equalsIgnoreCase(

                    isFound = true;

                    this.setEmployeeID(emp[0]);
                    this.setEmployeeName(emp[1]);
                    this.setAddress(emp[2]);
                    this.setDepartment(emp[3]);
                    this.setDesignation(emp[4]);
                    this.setEpfNum(emp[5]);

                    break;

                }

            }
```

This “searchEmployee” method use two input parameters. Because of the number of parameters are different in those two methods, they act as two different methods even they have the same method name .This is how a method can be implemented in many times within the same class. This is a one way the system used the polymorphism concept.

Polymorphism by method overriding

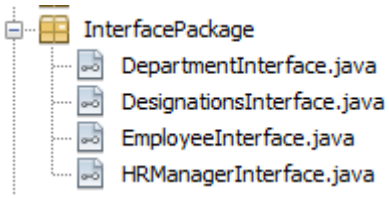
```
public DefaultTableModel viewAllEmployee() {

    DefaultTableModel defaultTableModel = new DefaultTableModel();

    String line;

    defaultTableModel.addColumn("Employee ID");
    defaultTableModel.addColumn("Employee Name");
    defaultTableModel.addColumn("Address");
    defaultTableModel.addColumn("Department");
    defaultTableModel.addColumn("Designation");
    defaultTableModel.addColumn("EPF No");

    try {
        BufferedReader bufferedReader = new BufferedReader(fs.readAFile());

        while ((line = bufferedReader.readLine()) != null) {
            String[] tableRow = line.split(" ");
            if (tableRow.length >= 6) {
                defaultTableModel.addRow(tableRow);
            }
        }
        bufferedReader.close();
    } catch (IOException ex) {

        System.err.println("An error occurred while reading data " + ex);

    }

    return defaultTableModel;
```

Same method can be implemented in a class and its subclass by overriding the same name to methods. But those methods can be differentiate by the functionality. This “viewAllEmployee” method implemented in the “EmployeeClass” is returning all the values reading from a text file into an array called “table Row”.

```java
@Override
public DefaultTableModel viewAllEmployee() {

    String line;

    DefaultTableModel defaultTableModel = new DefaultTableModel();

    defaultTableModel.addColumn("Employee ID");
    defaultTableModel.addColumn("Employee Name");
    defaultTableModel.addColumn("Department");

    try {
        try (BufferedReader bufferedReader = new BufferedReader(fs.readAFile())) {
            while ((line = bufferedReader.readLine()) != null) {
                String[] words = line.split(" ");
                if (words.length >= 4) {
                    String[] tableRow = {words[0], words[1], words[3]};
                    defaultTableModel.addRow(tableRow);
                }
            }
            bufferedReader.close();
        }
    } catch (IOException ex) {
        System.err.println("An error occurred while displaying data " + ex);
```

This “viewAllEmployee” method is implemented in the “AssistantClass”, which is a sub class of the “EmployeeClass”.Both methods have same name. But the functionality is different. The “viewAllEmployee” method in the super class returning all values from a text file into an array, while the “viewAllEmployee” method in the sub class only returning the 0^{th} , 1^{st} and 3^{rd} indexes of the array. Functionality wise those are two different methods. This is the way of using the method overriding concept.

Abstraction using interfaces



```java
package InterfacePackage;

public interface DesignationsInterface {
    public boolean addNewJobType();
    public boolean deleteDesignation( String designationName);
}
```

“DesignationsInterface” is an interface which defines two methods, “addNewJobType” and “deleteDesignation”. Those methods are implemented in the "DesignationClass”.

```java
public class DesignationsClass implements DesignationsInterface {

    FileSystem fsJob = new FileSystem("Designations.txt");
    private String designation;

    public DesignationsClass() {
    }

    public DesignationsClass(String designation) {
        this.designation = designation;
    }

    public String getDesignation() {
        return designation;
    }

    public void setDesignation(String designation) {
        this.designation = designation;
    }

    @Override
    public boolean addNewJobType() {
        String record = designation;
        if (!fsJob.createANewFile()) {

            return fsJob.writeDataToFile(record);
```

```java
@Override
public boolean deleteDesignation(String designationName) {

    boolean deleted = false;

    try {
        FileSystem fstemp = new FileSystem("temp_designation.txt");
        BufferedReader bufferedReader = fsJob.readAFile();
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            String[] words = line.split(" ");
            boolean found = false;
            for (String word : words) {
                if (word.equals(designationName)) {
                    found = true;
                    break;
```

The “DesignationClass” implements the “DesignationsInterface” and implemented all the methods defined in the interface. Classes that need to use the properties and behaviors of these abstracted methods can extend or inherit from the “DesignationClass” and able to reuse the implementation. Using the interface, the implementation can be modified without affecting the other parts of the code.

This is how the System use the abstraction concept.

7.2. Major functionalities of the system

1. Creating a new text-file to store data

The database of the system is text files. With the use of following methods, a new text file can be created. There is a relative file path to store the text files.

```
private static String FILE_PATH = "..\\st20251283_CSE4006_Writil\\Files\\";
File file;      //file constructor
private String fileName;         //creating a variable to give a name to file when naming
private FileWriter fileWriter;
private FileReader fileReader;

public FileSystem(String fileName) {
    this.fileName = fileName;
    createANewFile();
}

public boolean createANewFile() {
    try {
        file = new File(FILE_PATH + fileName);
        if (file.createNewFile()) {
            System.out.println("File Created: " + file.getName());
            return true;
        }
        System.out.println("File Already Exists " + file.getName());
        return false;
    } catch (IOException ex) {
        System.err.println("An error occured when creating the file " + ex);
        return false;
    }

}
```

2. Writing data into a text file

```
public boolean writeDataToFile(String record) {
    try {
        file.createNewFile();
        try (FileWriter fileWriter = new FileWriter(file, true);
                BufferedWriter bufferedWriter = new BufferedWriter(fileWriter)) {

            bufferedWriter.write(record);
            bufferedWriter.newLine();
            bufferedWriter.close();
            fileWriter.close();
        }
        return true;
    } catch (IOException ex) {
        System.err.println("An error occcured while writing data to the file " + ex);
        return false;
    }
}
```

"writeDataToFile()" takes an input as a string called "record" and returns a boolean value showing the data was successfully written or not to the text file. This method first create a new file if it is not exists. Then it writes the data into the file using the Filewriter and BufferedWriter .The append flag is set to true to append the data to the file instead of overwriting. Finally the method returns true if there is no any exceptions were thrown.

3. Reading data from a text file

```
public BufferedReader readAFile() {
    if (!createANewFile()) {
        try {
            FileReader fileReader = new FileReader(file);
            BufferedReader bufferedReader = new BufferedReader(fileReader);

            return bufferedReader;
        } catch (IOException ex) {
            System.err.println("An error occcured while reading data " + ex);
        }
    }
    return null;
}
```

This method returns a Bufferedreader which is used to read data. This method first checks the file is exist. If not, the method creates a new file .Then a FileReader and BufferedReader objects are created to read the data. This method returns null if there is any exception occurs.

4. Login

```
public boolean validateUserLogin() {
    try {
        String[] words = null;
        try (BufferedReader bufferReader = fs.readAFile()) {
            String user;

            while ((user = bufferReader.readLine()) != null) {
                words = user.split(" ");

                if (words[0].equals(UserName) && words[1].equals(Password)) {
                    this.setUserName(words[0]);
                    this.setPassword(words[1]);
                    this.setUserType(words[2]);
                    bufferReader.close();
                    return true;
                }

            }
            bufferReader.close();
        }
    } catch (IOException ex) {
        System.err.println("An error occured while validating login credentials" + ex);
    }

    return false;
}
```

This method is called when the user tried to login to the system .This method first reads a file using the readAFile() method. It then checks any userName and Password saved in the text file is matched with the user provided username and password. If a match is found it sets the variables with the matched values and returns true. If not, it returns false.

5. Add new employee

```
    @Override
    public boolean addNewEmployee() {
        if (!fsEmp.createANewFile()) {
            String record = employeeClass.employeeID + " " + EmployeeName + " "
                    + Address + " " + Department + " " + Designation + " " + EpfNum;
            return fsEmp.writeDataToFile(record);
        }
        return false;
    }
```

A new employee can be added into the system by calling this method. This is an override method from an interface and implemented in a class. This method first checks with a File object whether a file needs to be created or not. Then, the method declared a variable that contains the employee details which are provided by the user

and write those data into the relevant text file, if the file exists or after the creation of a new text file. Finally the method returns true if those data are written into the text file indicating a new employee is added. If not, it returns as false.

6. Displaying all employee details

```
@Override
public DefaultTableModel viewAllEmployee() {
    DefaultTableModel defaultTableModel = new DefaultTableModel();
    String line;
    defaultTableModel.addColumn("Employee ID");
    defaultTableModel.addColumn("Employee Name");
    defaultTableModel.addColumn("Address");
    defaultTableModel.addColumn("Department");
    defaultTableModel.addColumn("Designation");
    defaultTableModel.addColumn("EPF No");
    try {
        BufferedReader bufferedReader = new BufferedReader(fs.readAFile());
        while ((line = bufferedReader.readLine()) != null) {
            String[] tableRow = line.split(" ");
            if (tableRow.length == 6) {     //only executing the if conditio
                defaultTableModel.addRow(tableRow);
            }
        }
        bufferedReader.close();
    } catch (IOException ex) {
        System.err.println("An error occurred while reading data " + ex);
    }
    return defaultTableModel;

}
```

This is a method that is used to view employee details in a table form. This method use a java class called “DefaultTableModel”.Using that class the method is able to return values as a table form. First the method creates an object for the “DefaultTableModel” class and create a table from that with adding columns with names. Then it reads data from a text file as a String of lines. All lines are separated from a space (“ ”) delimiter. Then it adds every line into the “DefaultTableModel” object as table rows. Finally it returns the “DefaultTableModel” object which contains the data read from the text file.

7. Search for an employee

```
public boolean searchEmployee(String employeeID, String epfNo) {
    boolean isFound = false;
    try {
        String[] emp = null;
        try ( BufferedReader bufferedReader = fs.readAFile()) {
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                emp = line.split(" ");
                if (emp[0].equalsIgnoreCase(employeeID) || emp[5].equalsIgnoreCase(epfNo)) {
                    isFound = true;

                    this.setEmployeeID(emp[0]);
                    this.setEmployeeName(emp[1]);
                    this.setAddress(emp[2]);
                    this.setDepartment(emp[3]);
                    this.setDesignation(emp[4]);
                    this.setEpfNum(emp[5]);

                    break;

                }

            }
            bufferedReader.close();
        }
    } catch (IOException ex) {
        System.err.println("An error occurred while Searching Employee " + ex);
    }
    return isFound;
}
```

To find details of a specific employee, this method is called in the system. This method takes two parameters as strings. First, it declares a variable called “isFound” as false. Then it reads a data line by line from a text file until the last line of the text file while checking any value is matched with the input parameters. If a match found, it returns the “isFound” variable as true. Otherwise it returns false.

8. Delete an employee from the system

To delete an employee from the system the following method is called.

```
public boolean deleteEmployee(String employeeID) {

    boolean deleted = false;
    try {

        FileSystem fstemp = new FileSystem("temp_Employee.txt");
        BufferedReader bufferedReader = fsEmp.readAFile();
        String line;

        while ((line = bufferedReader.readLine()) != null) {
            String[] word = line.split(" ");
            if (word[0].equals(employeeID)) {
                continue;
            }
            fstemp.writeDataToFile(line);
        }

        bufferedReader.close();
        fsEmp.deleteAFile();
        fstemp.renameAFile("Employee_Details.txt");

        deleted = true;

    } catch (IOException ex) {
        System.err.println("An error occured while deleting data or renaming file " + ex);
    }
    return deleted;
}
```

This method takes an input parameter “employeeID” as a string. Then it creates an object for a temporary file to write the data temporarily. Then it reads the data stored in a text file line by line until the last line of the text file and stores those lines into an array .While reading the text file, it checks any line contains a string matches with the input parameter. If a match is found, it skips the line which contains the input parameter and write all other lines into the temporary file object. After that, it deletes the original file and renames the temporary file into the name of the original file. The deletion and the renaming of the files are done by calling two separate methods. Then it returns a boolean value indicating all the operations are getting true or not which means an employee is deleted or not from the system.

Those are the major functionalities of the system. Other than that, there are several functions such as creating user accounts, deleting user accounts, adding new departments and designations and deleting departments and designations and so on. Those methods also use the same structure as the above described methods using various techniques in the system such as writing and reading data from text files and also using the various object oriented concepts.

8. User Manual

Contents

1. Software overview

This is a desktop application which performs basic management of employee details of the “Colombo institute of studies”. Major functionalities of the application are employee management, user management, department and designation management.

2. User levels of the software

This application performs various operations based on 3 user levels. They are Administrator mode, Manager mode and Assistant mode. Based on the user level, the actions that can be performed by a particular user are different from each other.

2.1 Administrator mode

The administrator can manage user accounts of the users who are using the system. Administrator can create new user accounts for the HR manager and Assistants, view all user accounts and their details such as user name, password and user type. And also administrator can delete user account.

Note: System cannot create a new user account for another administrator. Administrator login credentials are already stored in the database and provided with the software.

2.2 Manager mode

The main action that a manager can perform within the system is employee details management. Manager can register new employee and allocating them into available departments and designations, view all of the employee details, delete an employee and their details from the system. And also manager can manage departments and designation details. Manager can add a new department and designation and also delete them. Other than that, the manager can search for an employee.

2.3 Assistant mode

Assistant is the least powerful user of the system. Assistant can perform only limited actions under certain restrictions. Assistant can only view employee details and search for an employee.

3. Starting and login

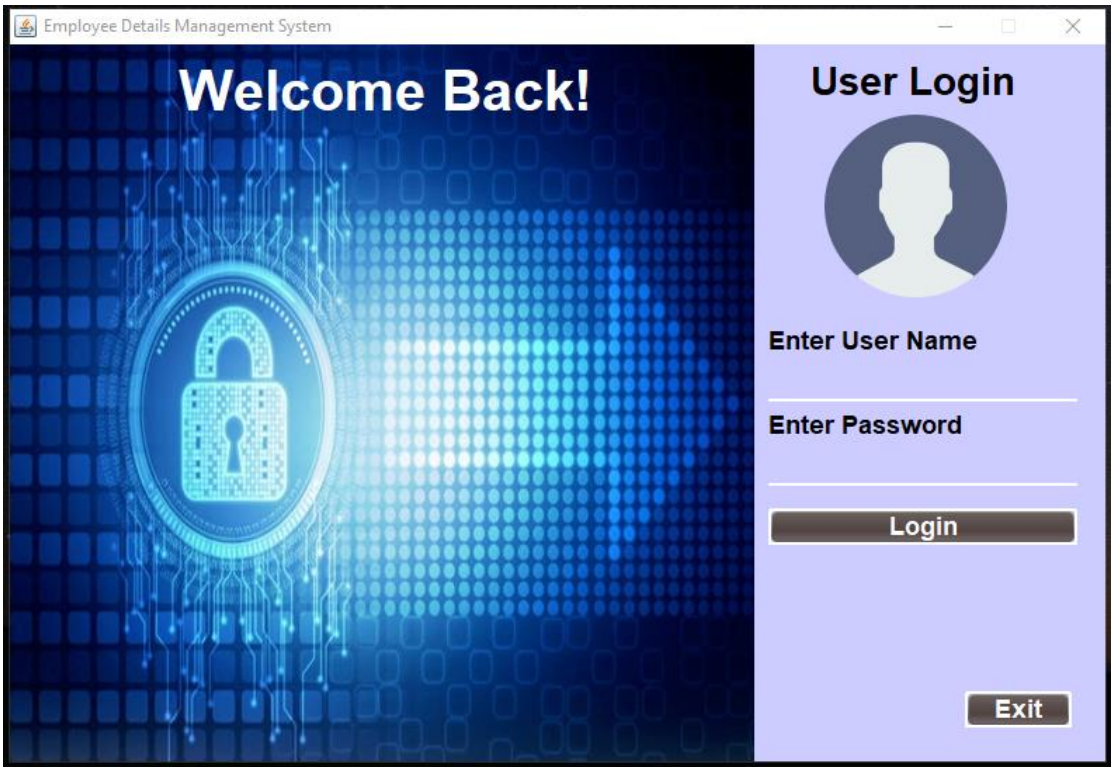
Step 1- Open the application

When the user opens the application, the start page will be displayed.

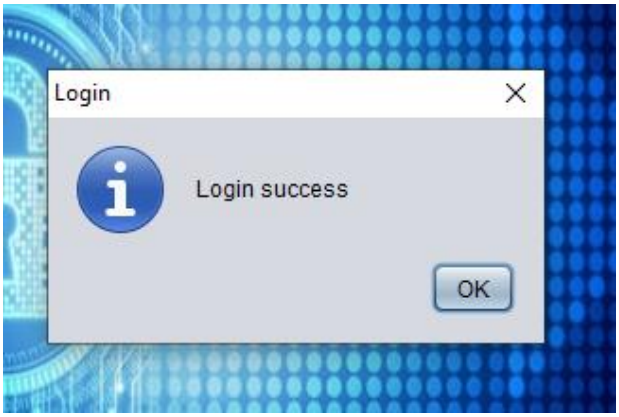
In this window, the user can either enter to the system or close the application by clicking the relevant buttons.

Step 2- Login

If the user choose to enter to the system, the login form will appear next. User needs to provide the correct user name and password to login.

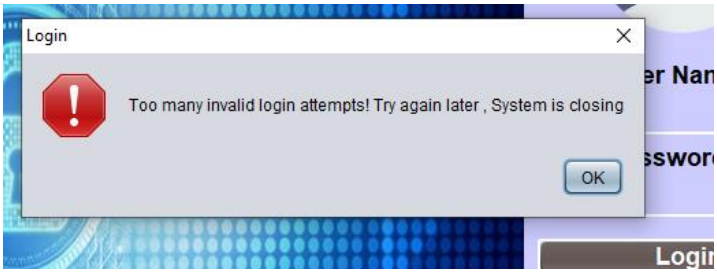


If the user provided valid user name and password, a message will display as “Login success” and, the user can login to the system according to the relevant user type.



If the user has provided invalid user name and password, a message will display as “Login failed”.

After 3 consecutive invalid login attempts, system is automatically closing to restrict the unauthorized login attempts furthermore.



4. Quick start guide for Administrator

4.1 Login with an Administrator account

If the user logged in with an administrator account, user can see the Admin page. Admin page includes 2 tabs.

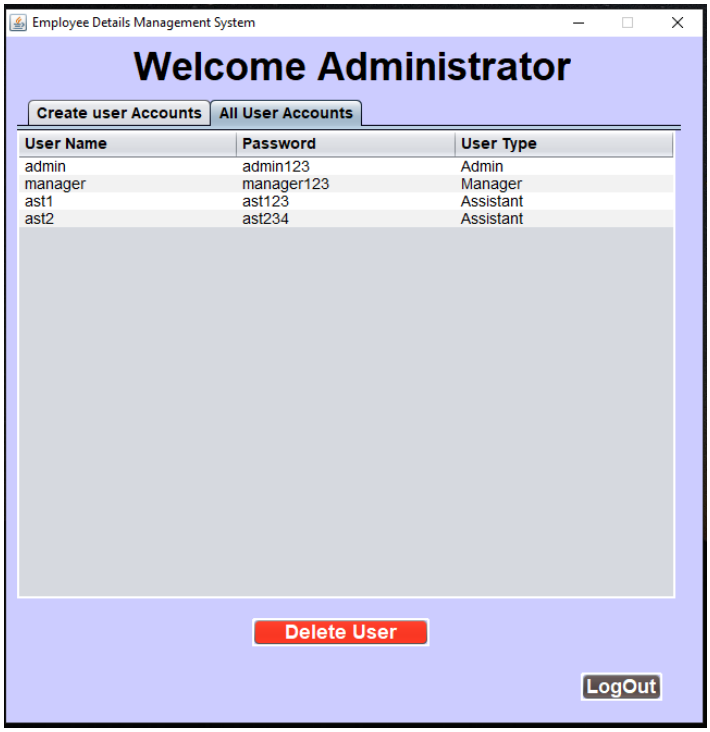
4.2 Create a new user account

In the first tab the admin can create new user accounts by entering the required data to the system.

If a new user is added to the system, a message will display as “User added successfully”.

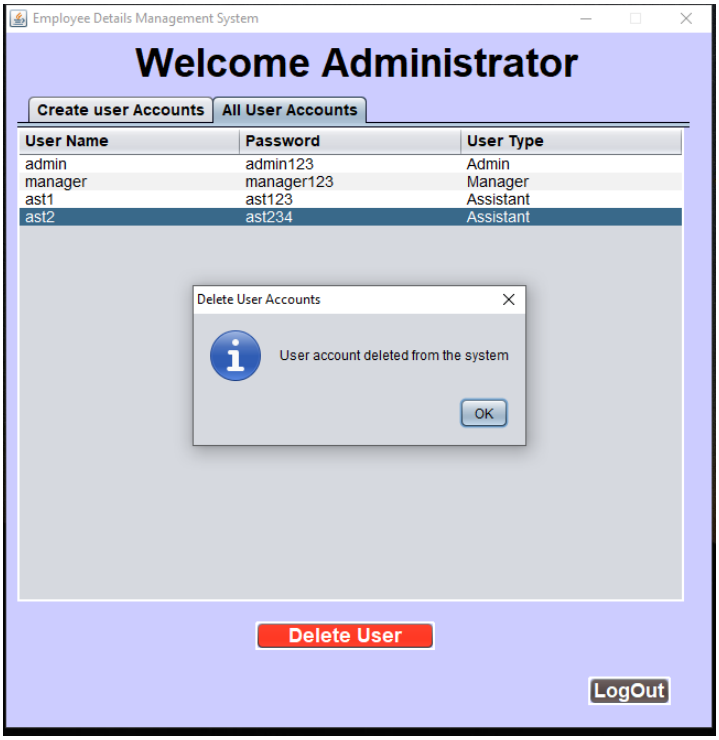
4.3 View all user accounts and their details

In the second tab, Admin can see all user accounts and their details such as user name, password and user type of a relevant user. And also the admin can delete a user account.

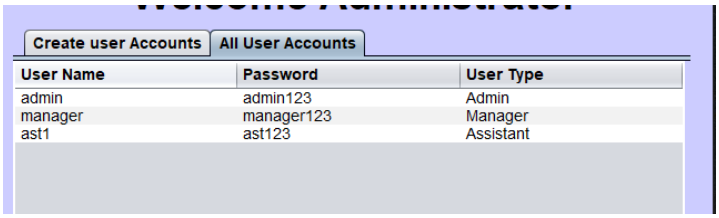


4.4 Delete a user account

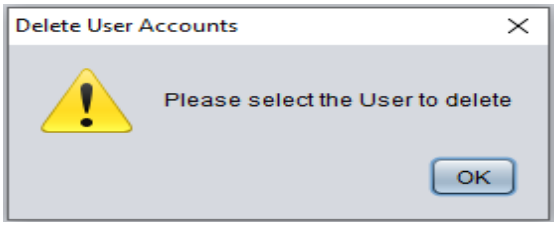
To delete a user account, the Admin first need to select the user account from the list and then click the red colored button named “Delete User”. A message will display if a user account has deleted successfully.



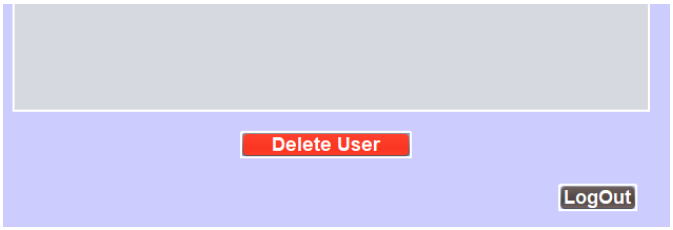
Then the admin can see the deleted user account is no longer available in the system. The user account of “ast2” is not in the system now.



If the admin tried to delete a user account without selecting, a warning message will display as follows. So make sure to select a user account from the list before clicking the delete button.



If the user does not have to perform tasks anymore with the system as the Admin, the user can logout by clicking the logout button which is available in the bottom right side of the application. Then the login page will be displayed again.



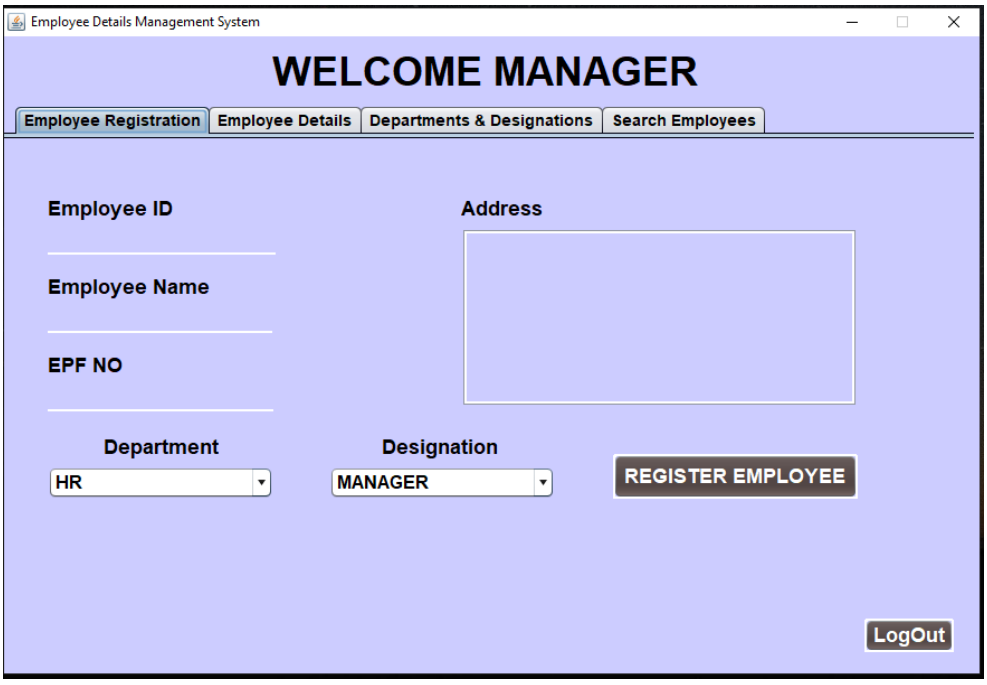
5. Quick start guide for Manager

5.1 Login with a manager account

If the user logged in as the Manager, user can see the Manager page. Manager page includes 4 tabs.

5.2 Register a new employee

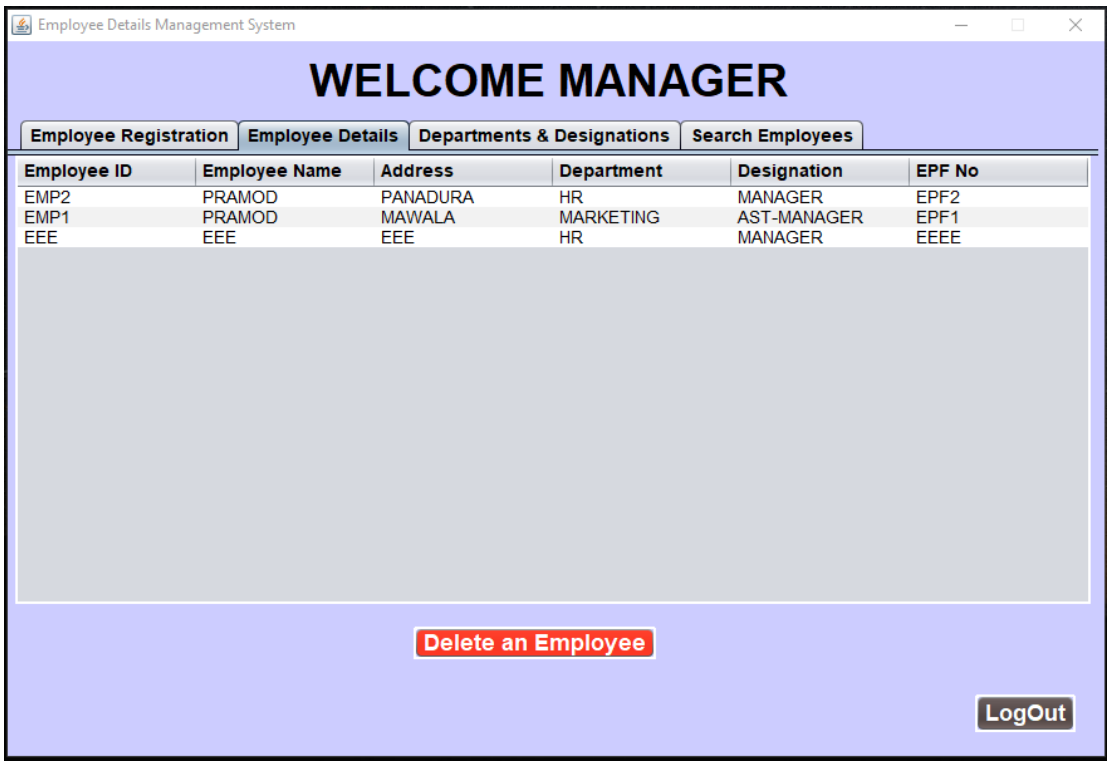
In the first tab, the manager can add new employees and their details to the system.



Manager can enter the relevant data and register a new employee from this tab.

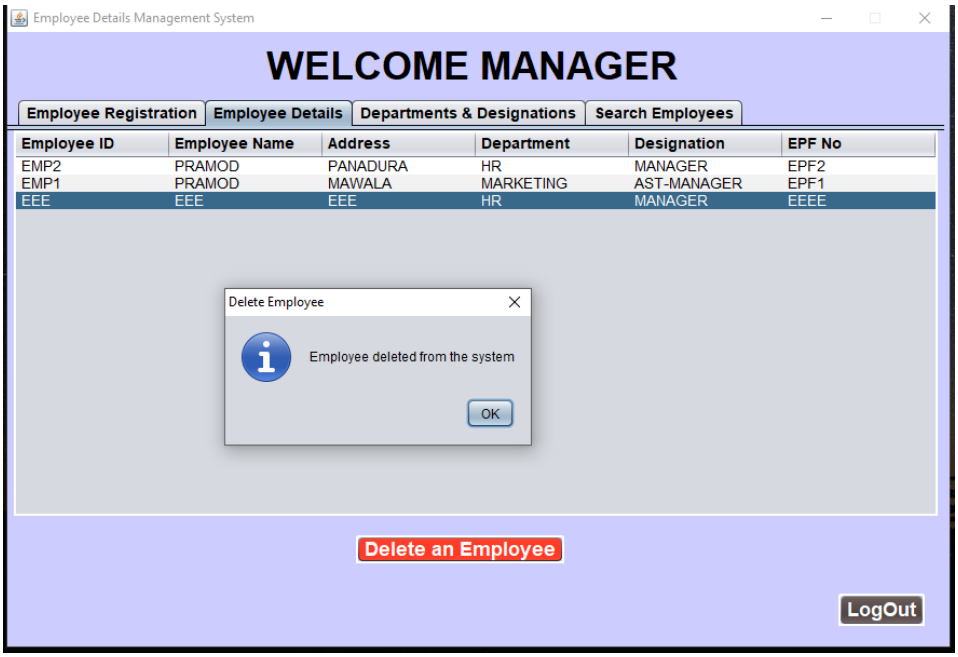
5.3 View all employee list and their details

In the second tab, the manager can see all employees and their details. Manager also can delete an employee from this tab.

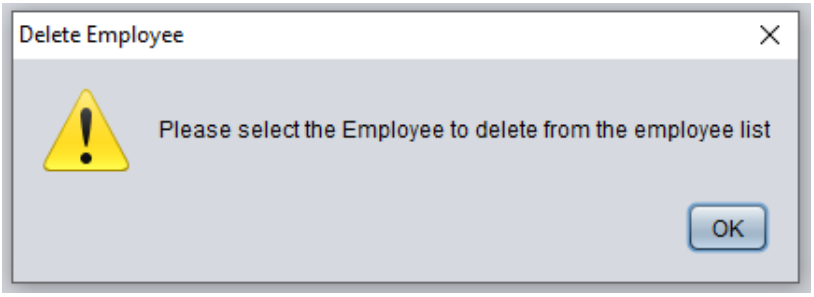


5.4 Delete an employee

Manager can delete an employee by selecting an employee and clicking the delete button. If an employee was deleted from the system, a message will be displayed as follows.

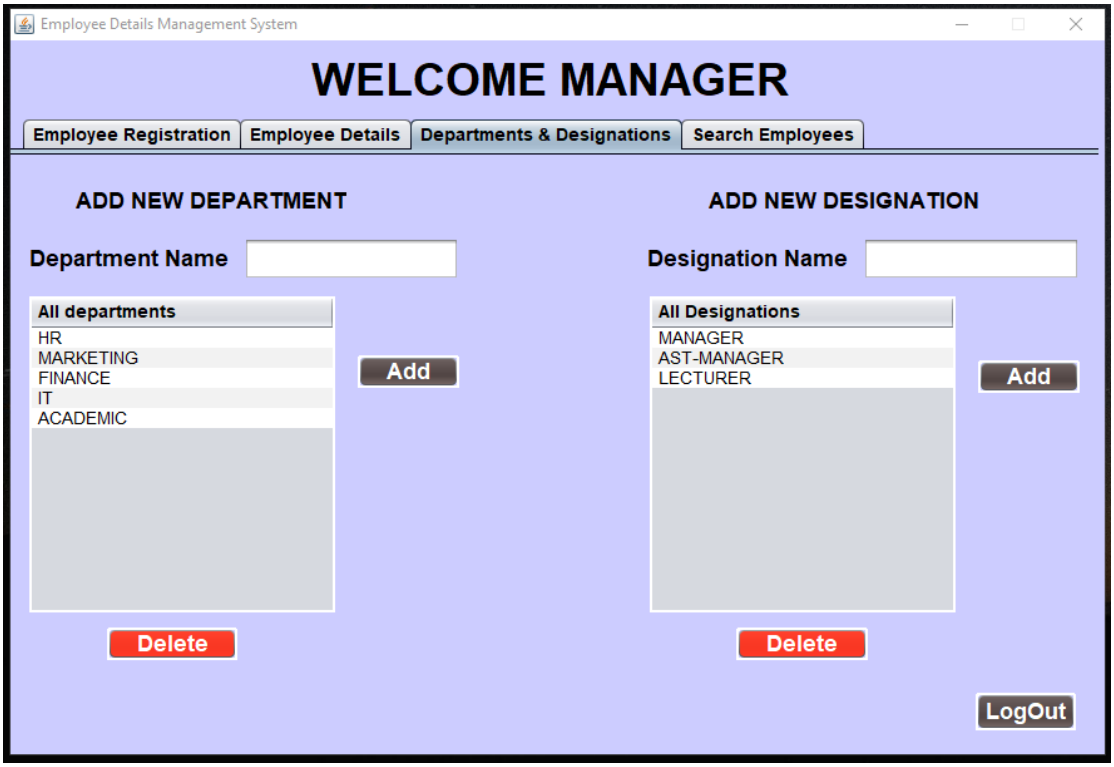


Manager have to select an employee before clicking the delete button. If the manager tried to delete an employee without selecting, a warning message will display as follows.

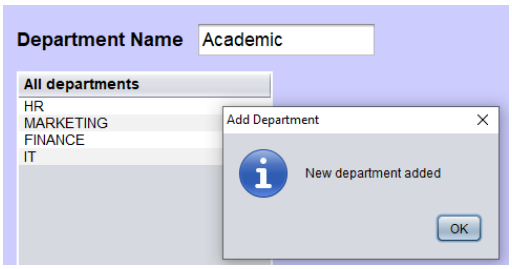


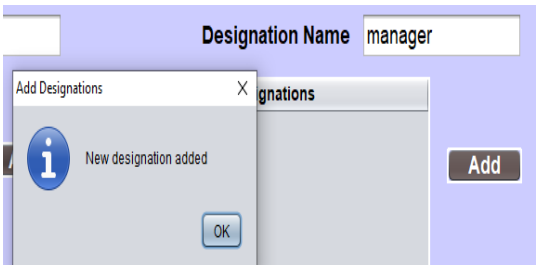
5.5 Add, view and delete departments and designations

From the third tab, the manager can perform certain tasks. Those are, adding new departments and designations, view available departments and designations and also delete a department and designation.

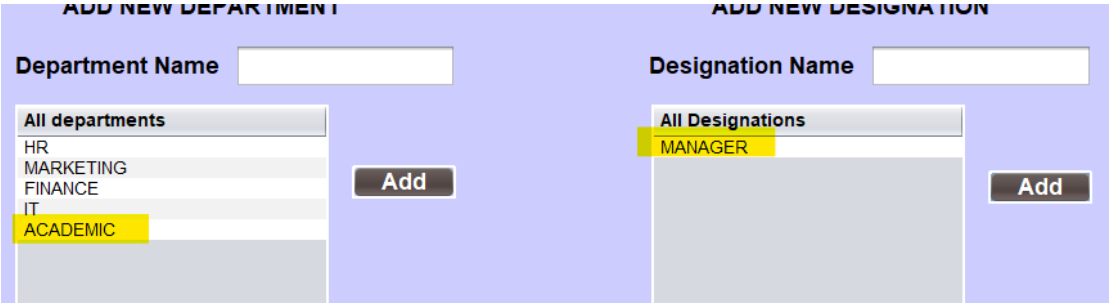


Following messages will be displayed if a new department and a designation is added to the system.



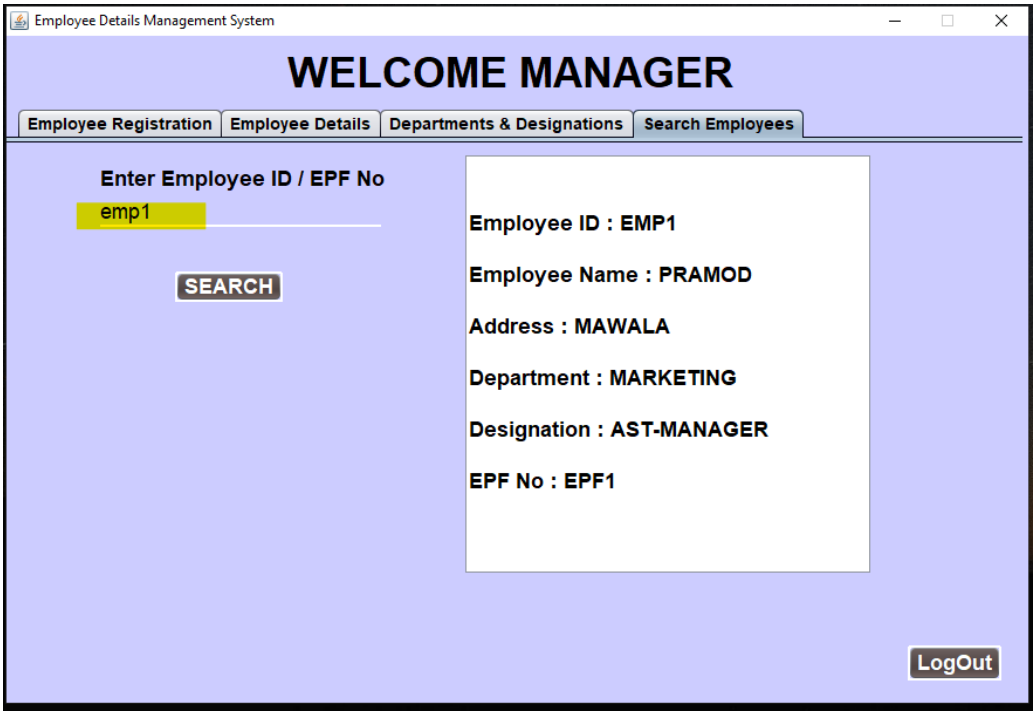


Now the manager can see the added departments are displayed in lists.

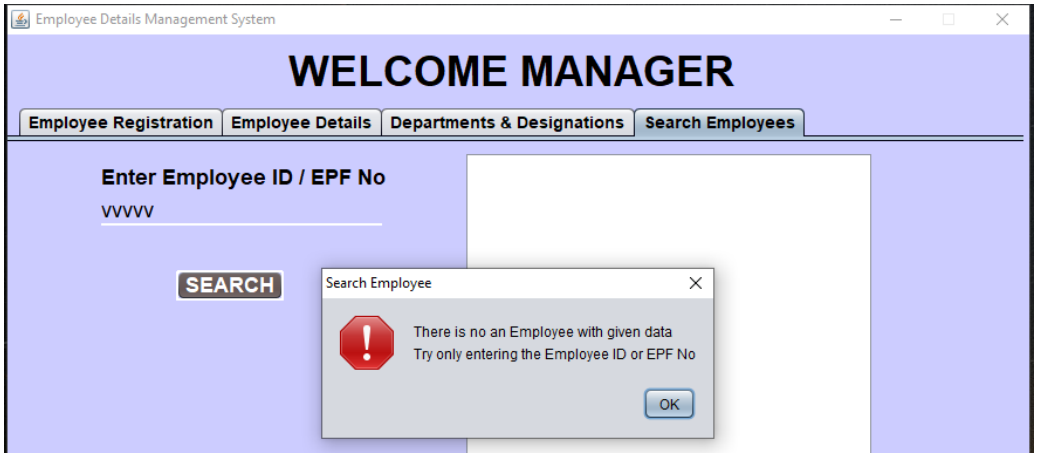


5.6 Search for an employee

In the last tab of the Manager page, the user can search for an employee. User can see all the details of a particular employee by providing the employee ID or the EPF No to the system. If an employee is registered in the system with given data, the details of that employee will be displayed.



If the user entered an employee ID or EPF No which is not allocated to an employee, the following error message will be displayed.



Now the user can logout as the manager from the system if the user done with those tasks belongs to the manager. User can simply logout by clicking the button displayed as logout. Then the login page will display again.

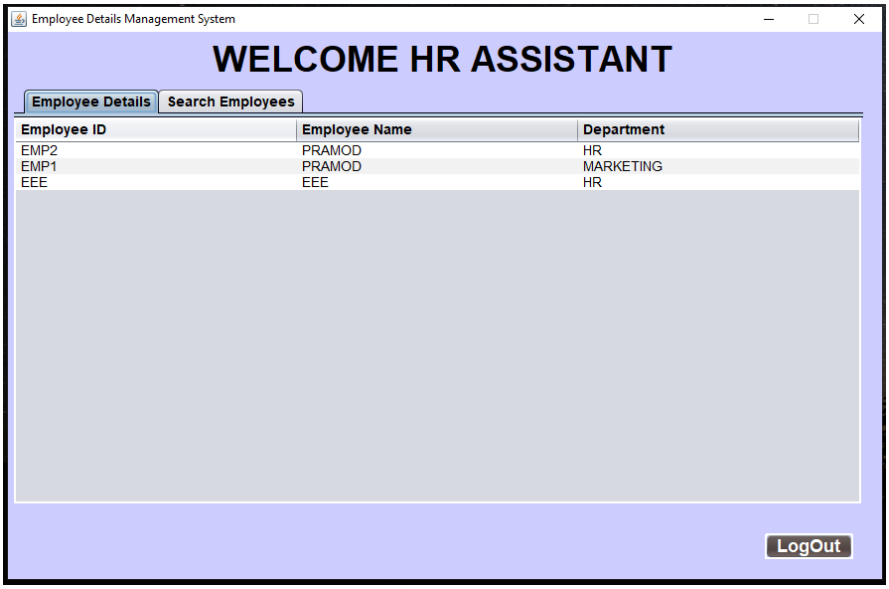
6. Quick start guide for assistants

6.1 Login with an assistant account

If the user logged in with an assistant user account, user can see the Assistant page. As per the limitations of the user level, assistant can only see limited details of employees. And also assistant can only provide the employee ID to search for an employee. Assistant page includes two tabs.

6.2 View all employees and their details

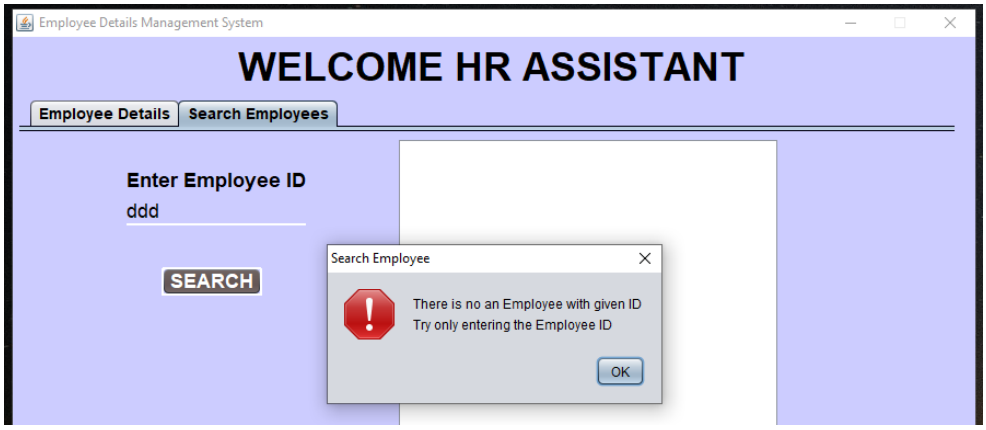
From the first tab, the user can see details of all employees.



6.3 Search for an employee

From the second tab, the Assistant can search for an employee by providing the employee ID to the system. If the user entered an employee ID of a registered employee to search, the details of the relevant employee will be displayed.

If the user entered an employee ID which is not registered into an Employee, an error message will displayed as follows.



If the user done with the tasks as the assistant, user can logout and the login form will be displayed again.

7. Exit from the application

The user can close the application by clicking the exit button in the login form.

Now the application will be closed.

9. Conclusion

In conclusion, this report described how a software system was implemented using object oriented concepts in order to develop a scalable, maintainable software. Use case diagram, class diagram and sequence diagram provides a thorough understanding of the design and the behavior of the system. And the user manual, which includes a detailed description of how to use the software, will help end users to use the software to its full potential.