# Applied Software Engineering


**Assignment 3 – Implementation and Report**
**Module Leader – Nasser Matoorian**

**Student ID – 21360293**

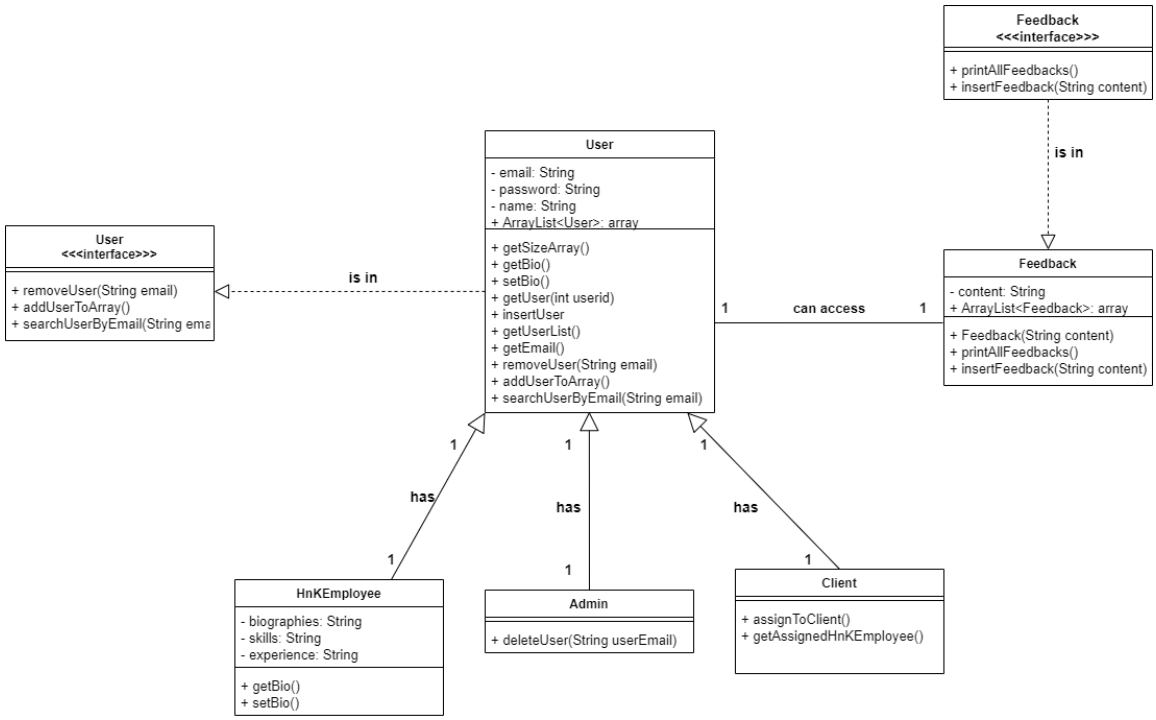
**Name - Pankaj Singh**

# Contents

# Implementation

In this stage, a lot of changes have been made such as new classes have been added such as User and Feedback and their respective interfaces. Additionally, new methods have been added in the client such as “assignToClient()” and “getAssignedHnk”

## Class Diagram




## Class description

### User

The User class is a generalisation of the HnKEmployee, Admin, Client subclasses. It contains the basic information of all types of users such as email, name, and password which are private. As well as the basic functionality of all types of users such as requesting and editing user info: getEmail(), getUser(), insertUser(), addUserByEmail(), etc. The ArrayList attribute is used to keep a track of the number of users.

### HnKEmployee

The HnKEmployee class inherits from the User class with the additional methods to view and create biographies and additional private attributes for the background of the employee such as skills and experience.

### Admin

The Admin class inherits from the User class. The Admin must be able to delete users using the additional deleteUser() method, add users and view their feedback.

### Client

The Client class inherits from the User class. It has the additional methods to be assigned an employee, assignToClient(), and view their assigned employee, getAssignedHnKEmployee().

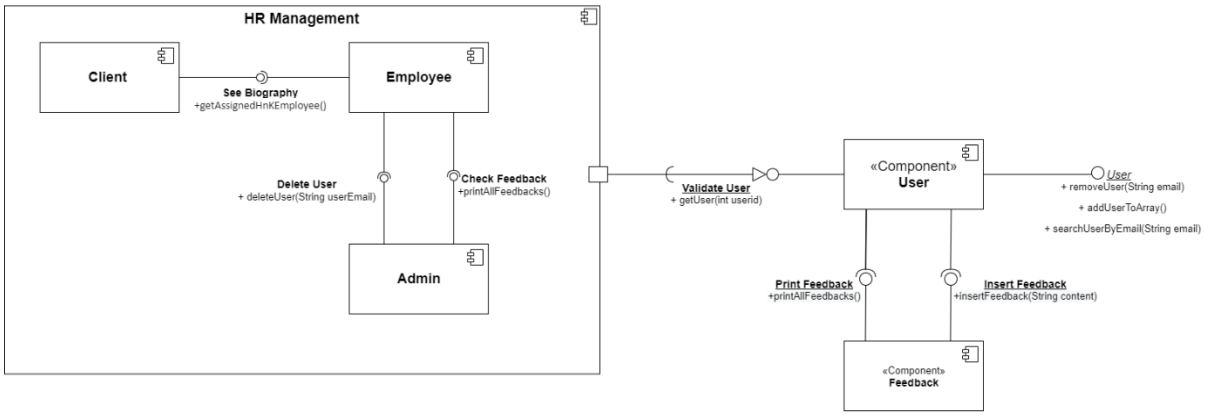
### Feedback

The Feedback class stores the feedback in an attribute named content and has methods to view and create the feedbacks. The ArrayList attribute is used to keep a track of the number of feedbacks and the Feedback() method is simply used to rename the content attribute in order to make programming simpler and more time efficient.

Interfaces

The interface classes for User and Feedback define and call methods that are implemented in the User and Feedback classes, respectively. This is good practice since often the methods will be defined in multiple classes but stored in one place.

# Component Model




The model illustrates the components designed from the class diagram. The component is blocky shape which is a higher level of abstraction than classes by grouping the whole system and the interface is wired to the components which allows them to interact. The interface is communicated through the lollipop (circle) and the semi-circle shape. The lollipop represents a provided interface, and the semi-circle represents a required interface.

The HR management component contains 3 users (Client, Employee and Admin). The client component can see the Hnk employee attributes by “getAssignedHnkEmployee()”.The admin component can remove the user using the “deleteUser” operation and check their feedback by “printAllFeedback()” method. The HR components has inheritance relationship with the user component in which it acquires the method and fields of the User component.

To access the data, the user needs to login first and the method “getUser” ensure that the user login is correct in order to access the system. Once the user login is validated then the user can use the feedback component to print and insert the feedback by “printAllFeedbacks()” and “insertFeedback()”.

# Post-Development testing

## Number of users

```
How many users would you like to insert:
3

User department: A - Admin, C - Client, H - HnK Employee
A
User name: JohnSmith
User email: JohnSmith@gmail.com
User password: pass1

User department: A - Admin, C - Client, H - HnK Employee
C
User name: JaneSmith
User email: JaneSmith@gmail.com
User password: pass2

User department: A - Admin, C - Client, H - HnK Employee
H
User name: JackSmith
User email: JackSmith@gmail.com
User password: pass3
Insert biography: Developer
Insert skills: Java
Insert experiences: Games.co
```

The program allows the user to add as many accounts as they want.

## Choosing the user's department

```
How many users would you like to insert:
1

User department: A - Admin, C - Client, H - HnK Employee
a
User name: JohnSmith
User email: JohnSmith@gmail.com
User password: pass1
Input error. User not inserted.

User department: A - Admin, C - Client, H - HnK Employee
A
User name: JohnSmith
User email: JohnSmith@gmail.com
User password: pass1
```

The program is case sensitive, meaning that the user must input their choice exactly as shown in the options. If an invalid input is given, an error message is shown and the user is prompted to again input an option.

## Admin and client details

```
User department: A - Admin, C - Client, H - HnK Employee
A
User name: JohnSmith
User email: JohnSmith@gmail.com
User password: pass1

User department: A - Admin, C - Client, H - HnK Employee
C
User name: JaneSmith
User email: JaneSmith@gmail.com
User password: pass2
```

New admins and clients are required to input their name, email, and password.

## HnK employee details

```
User department: A - Admin, C - Client, H - HnK Employee
H
User name: JackSmith
User email: JackSmith@gmail.com
User password: pass3
Insert biography: Developer
Insert skills: Java
Insert experiences: Games.co
```

New HnK employees are required to input their name, email, and password similar to the new admins and clients. Additionally, they are required to input their biography, skills, and experience.

## Login

```
Login with your credentials.

Enter email:
wrongEmail@gmail.com
Enter password:
pass1
No user found. Try again.
Enter email:
JohnSmith@gmail.com
Enter password:
pass1

Actions permitted:
A - Add user
D - Delete user
S - See feedbacks
T - Terminate program
```

If the user enters an email for an account that does not exist, the program displays an error message and prompts the user to login again. Once the user has given valid login details, the home menu is displayed.

## Admin's functionality

```
Actions permitted:
A - Add user
D - Delete user
S - See feedbacks
T - Terminate program
```

The admin is able to add or remove users, see all feedback, and end the application.

## Admin can add a user

```
Actions permitted:
A - Add user
D - Delete user
S - See feedbacks
T - Terminate program
A

User department: A - Admin, C - Client, H - HnK Employee
C
User name: Jane Doe
User email: JaneDoe@gmail.com
User password: pass2
```

The admin is prompted to enter the new user's details.

## Admin can delete a user

```
Actions permitted:
A - Add user
D - Delete user
S - See feedbacks
T - Terminate program
D

Enter email of the user to delete:
JaneDoe@gmail.com

Login with your credentials.

Enter email:
JaneDoe@gmail.com
Enter password:
pass2
No user found. Try again.
Enter email:
```

The admin enters the email of the user they want to delete. The chosen user is then wiped from the program.

## Admin can view all feedback

```
Actions permitted:
A - Add user
D - Delete user
S - See feedbacks
T - Terminate program
S
The employee I worked with was very friendly and easy to speak to.
10/10 service was great.
Very polite staff and efficient serivce.
```

All feedback is displayed.

## Client can insert feedback

```
Actions permitted:
I - Insert feedbacks
B - See biographies
A - Assign HnK Employee
T - Terminate program
I
Insert feedback:
The service was impeccable.
Done
```

The client is prompted to input their feedback.

```
Actions permitted:
I - Insert feedbacks
B - See biographies
A - Assign HnK Employee
T - Terminate program
B

Enter user email to be searched for the bio, skills and experiences:
Jackson@gmail.com
Jackson@gmail.com Lisa@gmail.com 3
Biography: Software developer
Skill(s): Java, C#
Experience(s): Games.co, Ubisoft
```

## Client can view a biography

The client is prompted to enter a HnK employee's email. The program then displays that employee's biographies: their assigned client, biography, skills, and experiences.

```
Enter user email to be searched for the bio, skills and experiences:
anon
anon Lisa@gmail.com 3

User not found try again.
```

If the user enters an email that does not exist in the system, an error message is displayed.

## Client can assign an employee

```
Actions permitted:
I - Insert feedbacks
B - See biographies
A - Assign HnK Employee
T - Terminate program
A

Enter user email to be assigned to you(Client):
Jackson@gmail.com
```

The client is prompted to enter the email of the employee they want to be assigned to them.

```
Enter user email to be assigned to you(Client):
anon

User not found try again.
```

If the email does not exist in the system, an error message is displayed.

## HnK employee can insert feedback

```
Actions permitted:
I - Insert feedbacks
B - See biographies
E - Edit biography
T - Terminate program
I
Insert feedback:
The project was well executed in a time efficient manner.
Done
```

The HnK employee is prompted to input their feedback.

## HnK employee can view a biography

```
Actions permitted:
I - Insert feedbacks
B - See biographies
E - Edit biography
T - Terminate program
B

Enter user email to be searched for the bio, skills and experiences:
Jackson@gmail.com
Jackson@gmail.com Lisa@gmail.com 3
Biography: Software developer
Skill(s): Java, C#
Experience(s): Games.co, Ubisoft
```

The employee is prompted to enter a HnK employee's email. The program then displays that employee's biographies: their assigned client, biography, skills, and experiences.

```
Enter user email to be searched for the bio, skills and experiences:
anon
anon Lisa@gmail.com 3

User not found try again.
```

If the user enters an email that does not exist in the system, an error message is displayed.

## HnK employee can edit their biography

```
Actions permitted:
I - Insert feedbacks
B - See biographies
E - Edit biography
T - Terminate program
E
Insert new bio:
Software engineer
```

The employee is prompted to enter their new biography.

```
Enter user email to be searched for the bio, skills and experiences:
Jackson@gmail.com
Jackson@gmail.com Lisa@gmail.com 3
Biography: Software engineer
Skill(s): Java, C#
Experience(s): Games.co, Ubisoft
```

The employee's biography is updated.

## All users can terminate the program

```
T - Terminate program
T
BUILD SUCCESSFUL (total time: 4 minutes 21 seconds)
```

All users have the option to terminate the application.

# OCL

## Feedback

context Feedback:: printAllFeedbacks ()

**pre:** self.content ->exist (f: feedback| f.content = content) //the pre-condition checks if the data exist then displays the content of the Feedback

```
//prints all feedbacks
@Override
public void printAllFeedbacks(){
    for(int i=0;i<this.feeds.size();i++)
        System.out.println(this.feeds.get(i).content);
}
```

context Feedback :: insertFeedback (String content)

**pre:** self.content -> exists (f: Feedback | f.content = content) //The operation insertFeedback will add the user feedback to the content

```
//adds a new feedback
@Override
public void insertFeedback(String content)
{
    Feedback a = new Feedback(content);
    this.feeds.add(a);
}
```

## User

context User :: removeUser (String email)

**pre:** self.email -> exists (u: User| u.email: email) //the pre-condition checks if the user email exist

**post:** self.email -> email@pre -> reject (u:user | u:email = email)

//the post-condition removes the user email

```
//remove user
@Override
public void removeUser(String email){
    int user = 0;
    for(User counter : this.users){
        if(email.equals(counter.getEmail()))
        {
            this.users.remove(user);
            break;
        }
        user++;
    }
}
```

Context User:: searchUserByEmail(String email)

**pre:** not self.email -> exists (u:User| u.email = email)

//the pre-condition checks if the user email exist

**post:** self.email -> exists (u:User | u.email = email)

//the post-condition validate that the user email exist

```
//returns a user searched by email
@Override
public User searchUserByEmail(String email){
    int foundID = 0;
    boolean found = false;
    for (User search : users){
        if(search.getEmail().equals(email)){//'C
            found = true;
            break;
        }
        foundID++;
    }
    if(found)
        return this.users.get(foundID);
    else
        return null;
```

## Main

context Main:: getUser (foundID)

**pre:** not users -> exists (m:Main| m.users = users)

//the pre-condition display an error message if the user does not exist

**post:** self.users -> exists (m:Main | m.user = users)

//the post-condition ensure that the user exist

```
//LOGIN
boolean cont = true;
while(cont = true){
    System.out.println("\nLogin with your credentials.\n");
    boolean found = false;
    do{
    System.out.println("Enter email: ");
    String userEmail = input.nextLine();
    System.out.println("Enter password: ");
    String userPassword = input.nextLine();
    int foundID = 0;
    for (User search : users.getUserList()){
        if(search.getEmail().equals(userEmail)){//'Critical evaluation': this could be ir
            activeUser = users.getUser(foundID);
            found = true;
            break;
        }
        foundID++;
    }
    if(foundID == users.getSizeArray())System.out.println("No user found. Try again.");
    }while(!found);
```

# Design Evaluation

Overall, the UML diagrams and the code have been developed from the case study. The main aspects of the design are updating changes made on the user accounts. This includes being able to add new users or remove old users. The case study suggests an admin function of being able to remove former employees. Clients are now able to view H&K employees and their background, including their skills and experiences as well as adding a function to give feedback. In addition, the employees can edit their data and check other employee backgrounds and give feedback on the system functionality.

The implementation of the design follows a component-based development method. The benefit of this method is re-usability whereby the same function can be stored in one place and execute in multiple stages of the code. This reduces computational effort and coding time.

In evaluation, using a command line interface is unappealing and the user is vulnerable to simple issues such as inputting wrong command which leads to a bad user experience. The user interface should be provided with GUI with interacting buttons and drop-down menus. This would lessen the simple issues caused by the incorrect user input. Hence, reducing the validation checks needed.