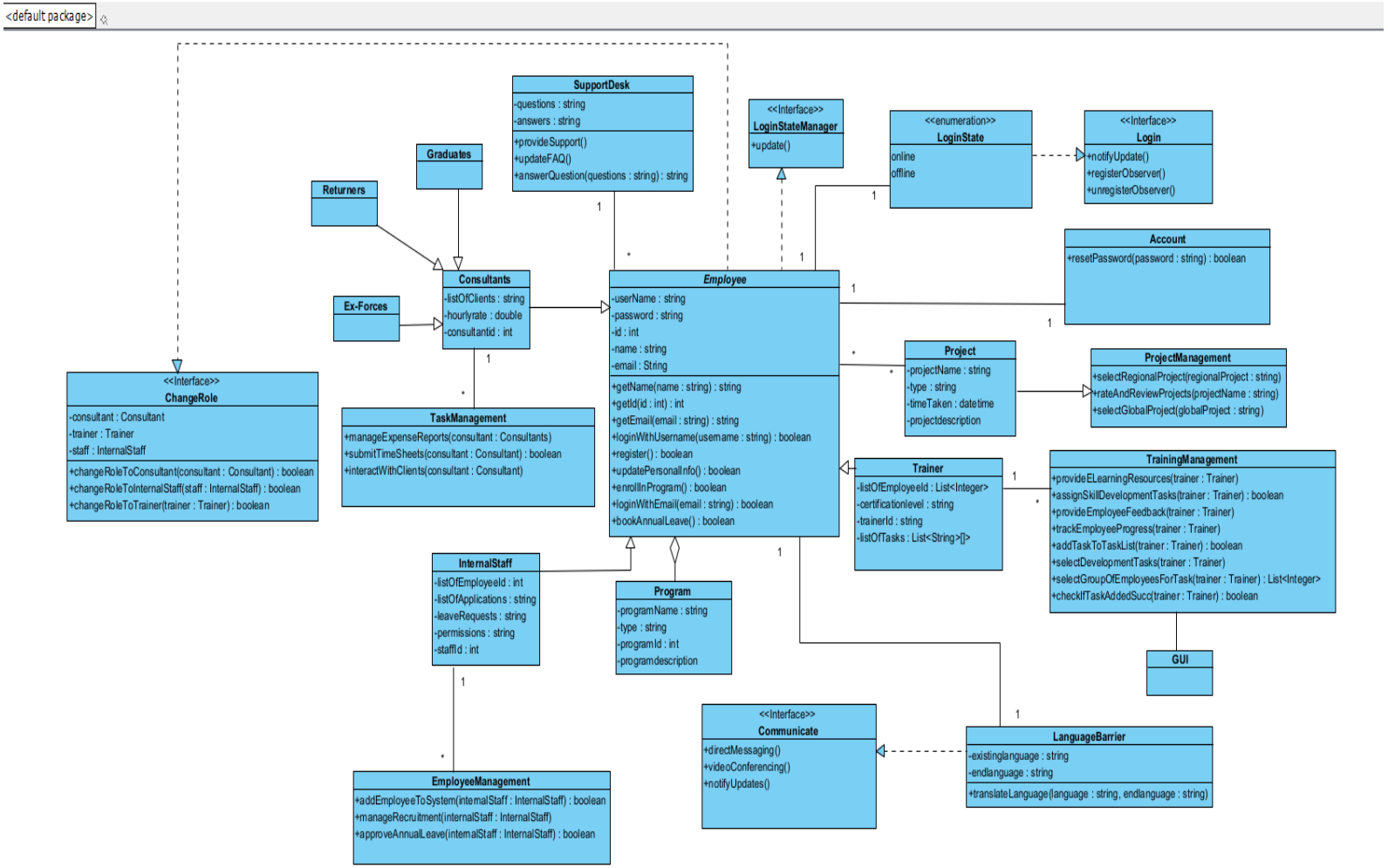
Group 32

FDM Employee Portal

ECS506U Software Engineering Group Project

Design Report

1. Class Diagram



2. Traceability matrix

Class	Requirement	Brief Description
Employee	RQ5	The Employee class holds the method updatePersonalInfo which allows all employees to update their personal information and therefore fulfil this data requirement.
TaskManagement	RQ21	The submitTimeSheets method is used in the TaskManagement class to enable consultants to fulfil this requirement (to submit time sheets).
TaskManagement	RQ22	The manageExpenseReports method can be utilised by consultants to manage the expense data and hence meet this data requirement.
Employee	RQ1	Each employee will have access to their own userName, password and email attributes which will be stored in the system and that is contained in the Employee class which they can then use to securely login to the employee portal. Further to this, employees will be allowed to either log in using the loginWithUsername method or loginWithEmail method. New employees will also be able to register once added to the portal using the register method.
EmployeeManagement	RQ26	The manageRecruitment method allows internal staff to manage and track candidates' applications. They will store the listOfEmployeeId and listOfApplications attributes to store this information.
TrainingManagement	RQ27, RQ38, RQ40	The trackEmployeeProgress method will allow trainers to store and track the data of employees/candidates.
EmployeeManagement	RQ39	Internal staff are able to receive annual leave requests by storing the leaveRequests attribute and then decide whether they want to approve or reject this using the approveAnnualLeave method.
SupportDesk	RQ18	The SupportDesk class will store questions and answers attributes on the system so that employees are able to get assistance from consultants. Frequently asked questions will also be stored and can be updated using the updateFAQ method.
ProjectManagement	RQ16	Employees can use the rateAndReviewProjects method to provide feedback on projects they've completed.

We originally only included RQ5 and RQ21 as a data requirement however, upon feedback and further consideration we have decided that we will also implement more of our functional requirements as data requirements.

3. Design Discussion

1. Explanation of design decisions such as why you did include the attributes or ignored others in classes that represent key entities.

In the context of the class diagram, various design decisions were made to represent the key entities effectively. These decisions are based on the requirements that the system needs to fulfil, as well as the principles of object-oriented design. Here's an explanation of some of these choices:

1. Inclusion of Attributes in Key Classes:

- Employee Class: This class is central to the diagram and has been given attributes like `userName`, `password`, `id`, `name`, `email`, and `String`. These attributes were chosen because they are essential for employee identification, authentication, and communication. Security is a significant concern in any system, which is why `userName` and `password` are necessary for secure logins. The `id` provides a unique identifier for each employee, which is critical for database management and internal processing. The `email` is a standard method for communication in business environments.
- Project and Program Classes: The `Project` class contains attributes like `projectName`, `type`, `timeTaken`, and `projectDescription`, which are vital for project tracking, categorisation, and management. Similarly, the `Program` class contains `programName`, `type`, `programId`, and `programDescription` to encapsulate the necessary details of training or development programs offered to employees.

2. Omission of Other Attributes:

- Attributes that are not directly linked to the system's primary functionality or that could complicate the system without adding significant value have been omitted. For instance, personal details like address or phone number are not included in the `Employee` class to avoid unnecessary complexity and to focus on attributes more pertinent to the system's operations.

3. Associations and Multiplicities:

- The associations between classes have multiplicities that indicate how many instances of a class can be associated with instances of another class. For example, the one-to-many relationship between `Employee` and `Project` signifies that one employee may be working on multiple projects simultaneously, which is a realistic scenario in most organisations.

4. Generalisation and Specialisation:

- The diagram uses inheritance to represent generalisation and specialisation relationships among different types of employees. For instance, `Consultant`, `Trainer`, and `InternalStaff` are all specialisations of the `Employee` class, indicating that they share common attributes and behaviours of an employee but also have additional, specialised characteristics.

5. Aggregation:

- Aggregation is used to show a whole-part relationship between classes, such as between `Program` and `Employee`. This implies that a program is composed of employees but does not strictly depend on the existence of any one employee, which is crucial for representing the autonomous nature of programs within an organisation.

In conclusion, the design decisions were driven by the need to create a coherent, efficient, and secure system that addresses the real-world complexities of managing employees, their roles, and the various tasks and programs they are involved with. The attributes and associations were

carefully selected to provide a comprehensive understanding of the system while maintaining a clear and manageable structure.

2. **Briefly explain associations (including aggregations) in your diagram and justify their correctness (especially using analysis from your earlier reports).**

Multiplicities:

- The one-to-many relationship between Employee and SupportDesk indicates that multiple employees can access the same support desk, which is essential in any large organisation where a single support entity serves an extensive employee base.
- Each Employee is associated with a single LoginState, illustrating that an employee can only have one login state at a time – either online or offline. This is critical for tracking the current status of an employee within the system.
- The one-to-one relationship between Employee and Account ensures that every employee has a unique set of credentials, a fundamental security requirement.
- A many-to-many association between Employee and Project under the ProjectManagement system allows for flexible assignment of employees to various projects and vice versa, which is typical in project-based work environments.
- Trainers and InternalStaff both have one-to-many relationships with their respective management classes (TrainingManagement and EmployeeManagement), reflecting the reality that one trainer or internal staff member may be responsible for multiple tasks or employees.

Generalisation:

- The class diagram uses inheritance to represent generalisation among different types of employees. Consultant, Trainer, and InternalStaff are subclasses of the Employee superclass, sharing common attributes like id, name, and email, while also having specialised functions.
- The subclasses such as Consultant, Trainer, and InternalStaff are specialisations that reflect the diverse roles within the organisation, each with their unique responsibilities and operations. For example, Consultants may focus on external client projects, while Trainers are specialised in developing and delivering training programs.

Aggregation:

- Aggregation is employed between the Program class and the Employee class to model the whole-part relationship. This indicates that a program is a grouping of employees who are involved in it, yet it can exist independently of any individual employee. This is important to show that while employees are essential to a program's operation, the program's existence is not contingent upon any single employee.

Associations:

- Associations are used to model the relationships between different classes. For example, the association between Employee and Program through EmployeeManagement indicates that employees are managed within the context of the programs they are involved in.
- The LanguageBarrier class is associated with Employee on a one-to-one basis, which signifies that each employee has the ability to use language translation services independently, a necessary feature in a multicultural and multilingual work environment.
- The Communicate interface is associated with Employee to signify that communication capabilities such as direct messaging, video conferencing, and notifications are integral to the employee's role within the organisation.

3. Explain any differences between design and your initial domain model and the explain the reason behind these changes.

Differences from Initial Domain Model:
In the initial domain model, the Employee class had a simple structure with basic attributes. However, in the design phase, enhancements were made to address specific requirements. For instance, the Employee class was enhanced with attributes like userName and password to enhance security and enable robust authentication. These additions reflect a more detailed representation of user credentials, aligning with the need for a secure login mechanism.
Additionally, new entities such as the LoginState interface and the ChangeRole interface were introduced. These were not initially present in the domain model. The LoginState interface was incorporated to manage the online/offline state of an employee, providing a real-time monitoring mechanism.
On the other hand, the ChangeRole interface was introduced to enable dynamic role changes, allowing for flexibility in role assignments for employees.
Furthermore, the SupportDesk class, representing the support system, was not explicitly modelled in the initial domain model. Its inclusion in the design addresses the requirement for a comprehensive support functionality, providing a more detailed view of the system.
The Program class, representing training programs, was another addition. This change aligns with the need to explicitly model training programs, contributing to a more complete and detailed system design.

Reasons for Changes:
The design decisions made during the transition from the domain model to the class diagram were driven by specific requirements and the need for a more comprehensive representation of the system.
The introduction of the userName and password attributes in the Employee class was essential to enhance security. This decision reflects a response to the requirement for a secure login and authentication process, addressing potential vulnerabilities.
The ChangeRole and LoginState interfaces were introduced to allow for dynamic role changes and real-time state management, respectively. These additions enhance the system's flexibility and responsiveness, aligning with the evolving needs of the application.
The inclusion of the SupportDesk class and the Program class reflects a commitment to providing a more detailed and feature-rich system. These changes respond directly to specific functional requirements related to support services and training programs.

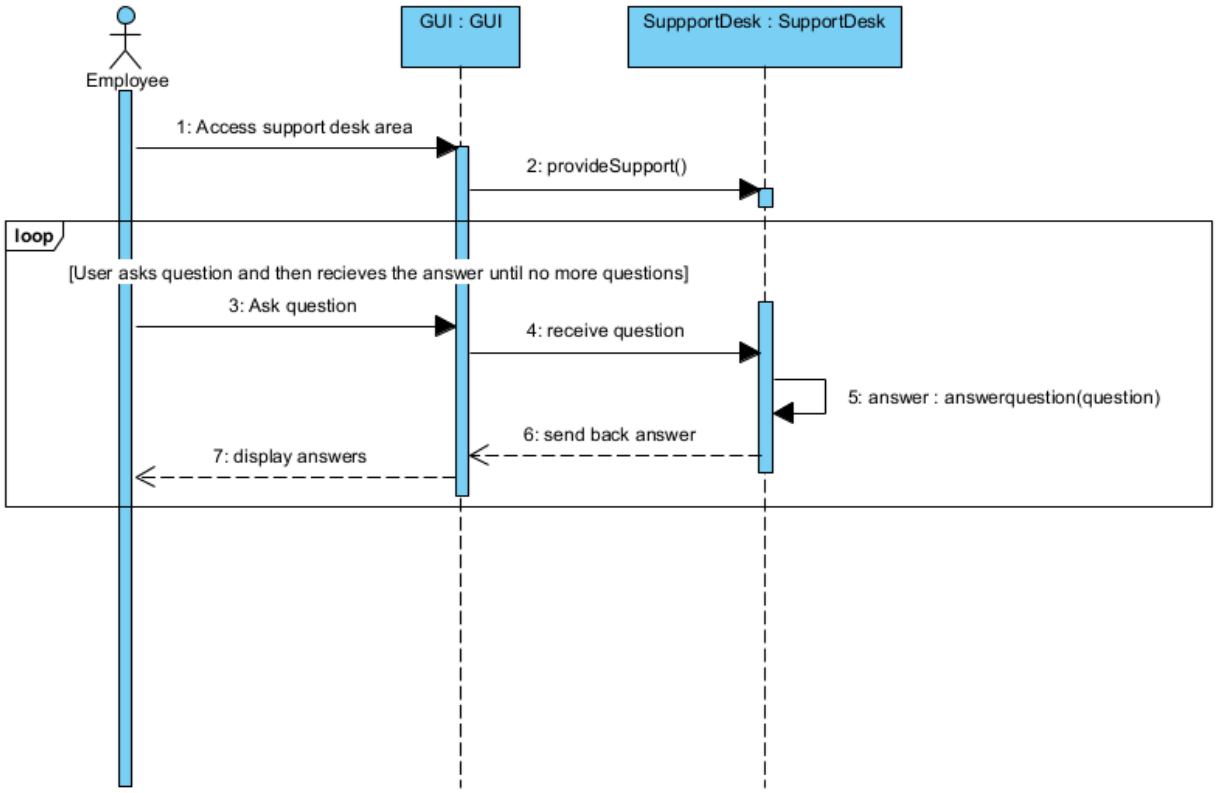
4. Discuss any design patterns you applied. Why did you apply that certain pattern and explain why you think that your design is a correct application of this pattern?

Design Patterns Application:
Three design patterns were applied in the class diagram: the Bridge pattern, the Observer pattern, and the Adapter pattern.
The Bridge pattern was employed to manage different roles (consultant, staff, internal trainer) through the ChangeRole interface. This decision allows for dynamic role changes without altering the client code, promoting a flexible and extensible design.

The Observer pattern was utilised in the Login interface to observe and notify changes in the online/offline state of an employee. This implementation ensures that real-time updates on the state of an employee are communicated throughout the system, enhancing system responsiveness.

The Adapter pattern was applied to connect with the LanguageBarrier class, responsible for language translation. This choice facilitates seamless integration with external language translation services, addressing the global communication needs of FDM.
In conclusion, the application of these design patterns contributes to the overall flexibility, responsiveness, and system integration of the Employee Portal.

4. Sequence Diagrams



Use Case 1: Chatbot

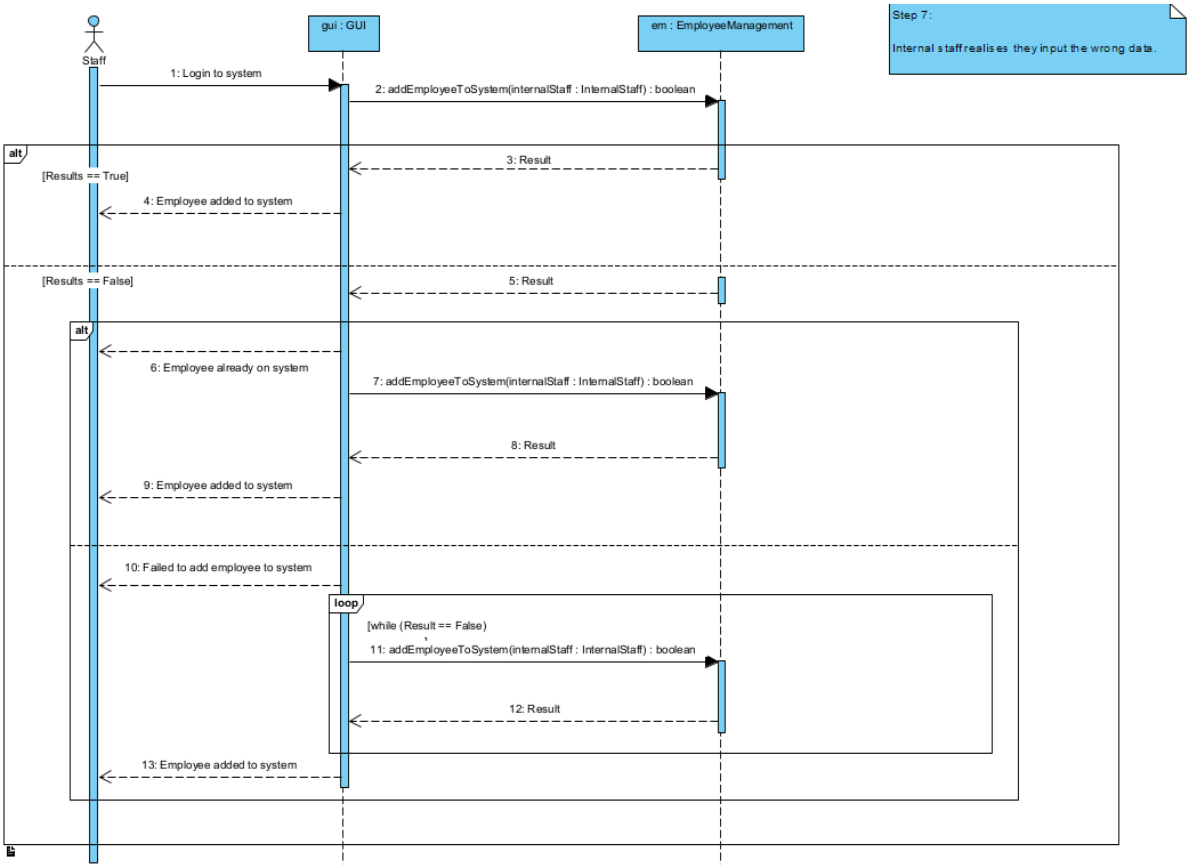
Basic Flow:

1. Employee navigates to support desk area.
2. System opens the chatbot.
3. User is prompted to ask a question.
4. Chatbot receives question.
5. Chatbot answers the question and answers are displayed.

6. Basic step 3 is repeated until user has run out of questions.
7. User exits the chatbot.

Prerequisites:

Employee must be logged in.



Use case 2: Adding employee to system.

Basic Flow:

8. Internal staff logs into the system.
9. Internal staff goes to the recruitment section.
10. Internal staff enters all the details of the employee to be added.
11. Internal staff allocates the new employee to a specific role.
12. Internal staff clicks the Submit button.
13. System adds new employee to the system.

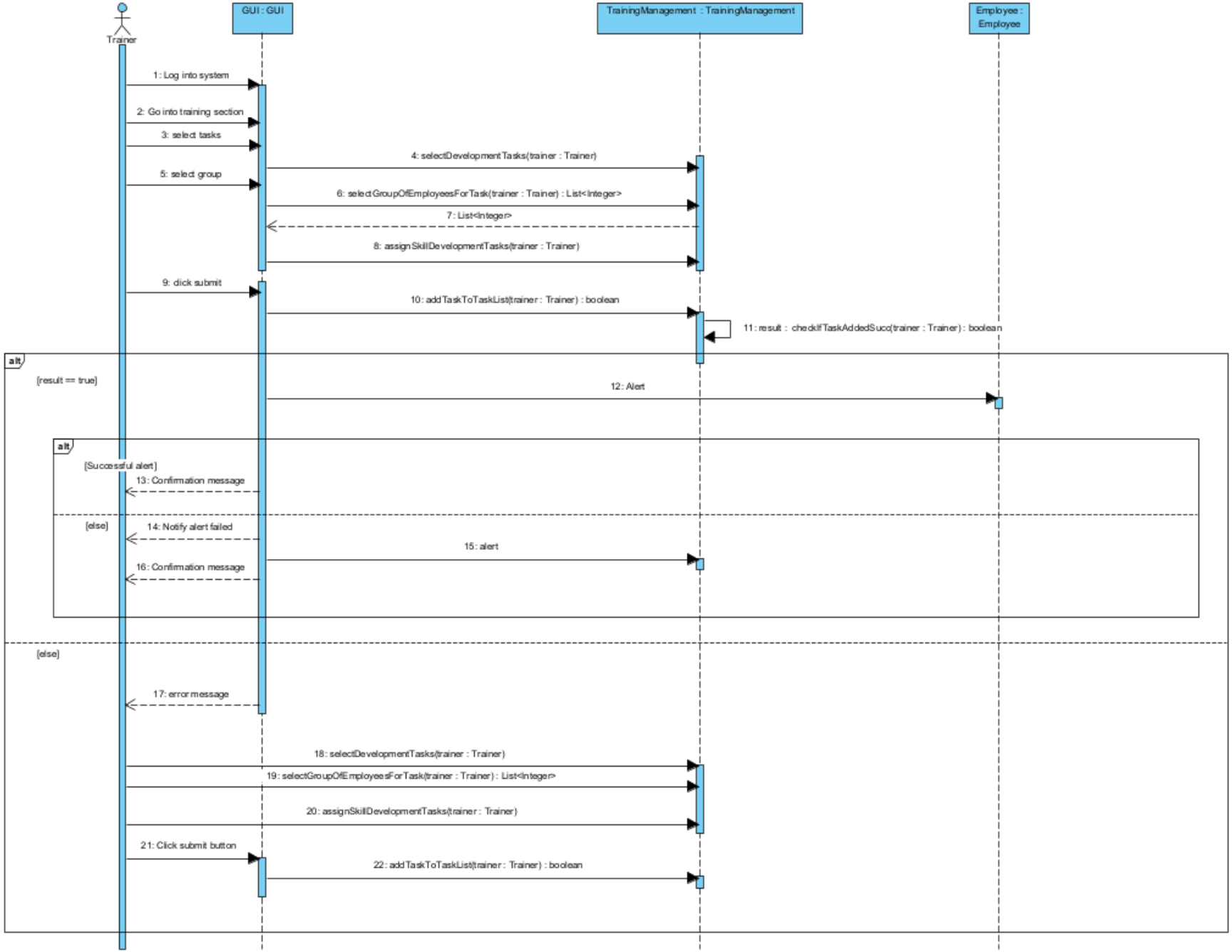
Alternative Flow:

1. Employee already on system

a. Internal staff realises that the employee has already been put into the system.
b. System carries on with Basic Flow step 3.

2. System fails to add new user to the system.

    a. User is asked to try again.
    b. System carries on with Basic Flow step 3 until the employee is added to the system


Trainer
GUI : GUI
TrainingManagement : TrainingManagement
Employee : Employee
1: Log into system
2: Go into training section
3: select tasks
4: selectDevelopmentTasks(trainer : Trainer)
5: select group
6: selectGroupOfEmployeesForTask(trainer : Trainer) : List<Integer>
7: List<Integer>
8: assignSkillDevelopmentTasks(trainer : Trainer)
9: click submit
10: addTaskToTaskList(trainer : Trainer) : boolean
11: result : checkIfTaskAddedSuco(trainer : Trainer) : boolean
alt
[result == true)
12: Alert
alt
[Successful alert]
13: Confirmation message
[else)
14: Notify alert failed
15: alert
16: Confirmation message
[else)
17: error message
18: selectDevelopmentTasks(trainer : Trainer)
19: selectGroupOfEmployeesForTask(trainer : Trainer) : List<Integer>
20: assignSkillDevelopmentTasks(trainer : Trainer)
21: Click submit button
22: addTaskToTaskList(trainer : Trainer) : boolean

Use case 3: Assign skill development task

Basic Flow

1. Trainer logs into the system.
2. Trainer goes to the training section.
3. Trainer selects which development tasks they will assign.
4. Trainer selects which group of employees they will allocate the chosen development tasks to.
5. Trainer clicks the Submit button.
6. System adds these tasks to the corresponding employee's task lists.
7. System alerts the relevant users that a new task has been set for them to complete.
8. System sends a confirmation message to the trainer that the tasks have been provided to the employees and that they have been notified

Alternate Flows

1. System fails to allocate the tasks into the employees' task lists.
    a. Trainer gets an error message from the system.
    b. Trainer selects the correct tasks for the correct group of employees.
    c. System carries on with Basic Flow step 5.

2. Employees are not notified that they have new tasks.
    a. Trainer informs the system that employees have not been notified.
    b. System carries on with Basic Flow step 7.

Prerequisites:

Users must be able to login