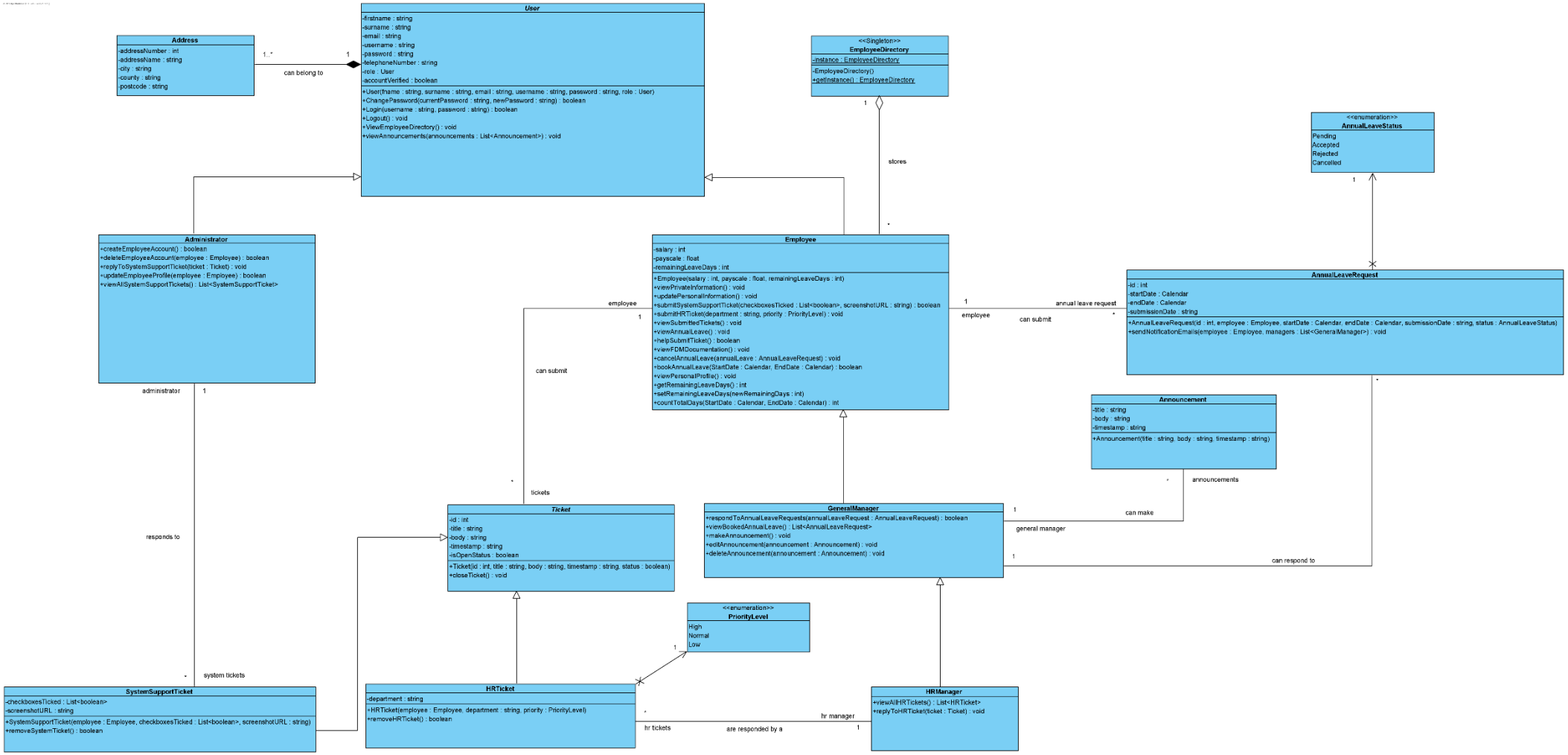
Group 16

FDM Employee Portal

ECS506U Software Engineering Group Project

Design Report

1. Class Diagram



2. Traceability matrix

Class	Requirement	Brief Explanation
Announcement	RQ6	The “body” attribute is a string which stores the body of the announcement. The “timestamp” attribute is a string which stores both the date and time of when the announcement was posted.
User	RQ7	We store the user’s full name using two string attributes: “firstname” and “lastname”. We store the user’s email address in the “email” string attribute of the User class. We store the user’s username in the “username” string attribute of the User class. We store the user’s password in the “password” string attribute of the User class.
Employee	RQ12	We store the public data of employees using the same attributes described as described above for RQ7. We store the private data of the employees using attributes such as “salary” and “remainingLeaveDays”. The “payscale” attribute stores each employees’ pay scale according to their role.
HRTicket	RQ26	We store the date and time of a HR Ticket in the “timestamp” string attribute. This attribute is inherited from its superclass (Ticket). We store the unique ticket number of a HR Ticket in the “id” attribute of type int. This attribute is inherited from its superclass (Ticket). We store the body of a HR Ticket in the “body” string attribute, which is inherited from the superclass (Ticket). Every HRTicket is associated with the Employee who submits the ticket. We use this Employee object to determine the sender’s name of a HRTicket.
SystemSupportTicket	RQ28	We store the unique ticket number of a System Support Ticket in the “id” attribute of type int. This attribute is inherited from its superclass (Ticket). Every System Support Ticket is associated with the Employee who submits the ticket. We use this Employee object to determine the sender’s name of a System Support Ticket.

		We store the selected checkboxes of a System Support Ticket using the “checkboxesTicked” attribute, which is of type List<boolean>. We store the body of a System Support Ticket in the “body” string attribute which the SystemSupportTicket class inherits from its superclass (Ticket). We store the attached image URL of a System Support Ticket in the “screenshotURL” attribute of type string. If no image is attached, we store an empty string. We store the date and time of a System Support Ticket in the “timestamp” string attribute, this is inherited from its superclass (Ticket).

3. Design Discussion

The class diagram we have designed is an extended version of the domain model previously created. However, there have been some small adaptations made in order to ensure that the system can meet all stated requirements.

One major addition is the User class. This is an abstract class in which the Employee, GeneralManager, HRManager and Administrator classes specialise from. The abstract class contains attributes that are common amongst all the previously mentioned classes, such as username and password, as well as common operations such as viewing announcements. The class was included because we made the decision that administrators should have access to some of the same operations that employees have access to. For example, changing the password to their account and viewing the employee directory. Without this abstract class, all of these attributes and operations would have been unnecessarily repeated in both the Administrator and the Employee class.

An Address class was also added, with a composition association to the User class. This is because the address of a user cannot exist independently without the user. The association also allows for a user to have more than one address. This is advantageous as it takes into consideration off-site FDM consultants who may be working and living in different locations for various projects.

Another addition is the EmployeeDirectory class, this is a class that conceptually represents the employee directory. It has been modelled using an aggregation association. This design decision was made because even though the directory will consist of employee instances, they are able to exist independently in the event that the EmployeeDirectory is ever deleted.

The AnnualLeave class from the domain model has been changed to AnnualLeaveRequest. This is because it better represents the system's intentions: an employee is able to make a request for annual leave, which is then responded to by a general manager. As a result, the AnnualLeaveRequest class has a uni-directional association with a newly implemented enumeration class, AnnualLeaveStatus. This class is used to set the status of the request (Pending, Accepted, Rejected, Cancelled). Through the class being uni-directional, AnnualLeaveStatus cannot access or see any details of AnnualLeaveRequest.

The association between Employee and Announcement (originally on the domain model) for viewing an announcement has been removed. This is because we decided that viewing information is not worthy of an association between two objects.

Despite some similarities, it was decided that the HRTicket and SystemSupportTicket classes will remain separate because they are conceptually different. Even though both classes do have common attributes and a single shared operation (hence the abstract Ticket class), both classes also have unique attributes and thus require unique constructors. Much like the domain model, the associations between Administrator and SystemSupportTicket as well as HRManager and HRTicket, will continue to be a one-to-many relationship. I.e., A single Administrator can respond to multiple System Support tickets and a HR manager can respond to many HR tickets.

For our system design, the singleton design pattern was applied on the EmployeeDirectory class. This is because we believe that there should only be one accessible instance of EmployeeDirectory, and so we needed to ensure that others cannot be made. If the singleton pattern was not used, user profiles may have potentially been stored in many EmployeeDirectory instances, and thus not achieving the objective of having all employee profiles accessible from a single location.

This design pattern was implemented by making the instance attribute and constructor method private, and then having a public method (getInstance) that can be called to access the instance of the class. This way, only one instance is ever allowed and it is the only one that can be accessed.

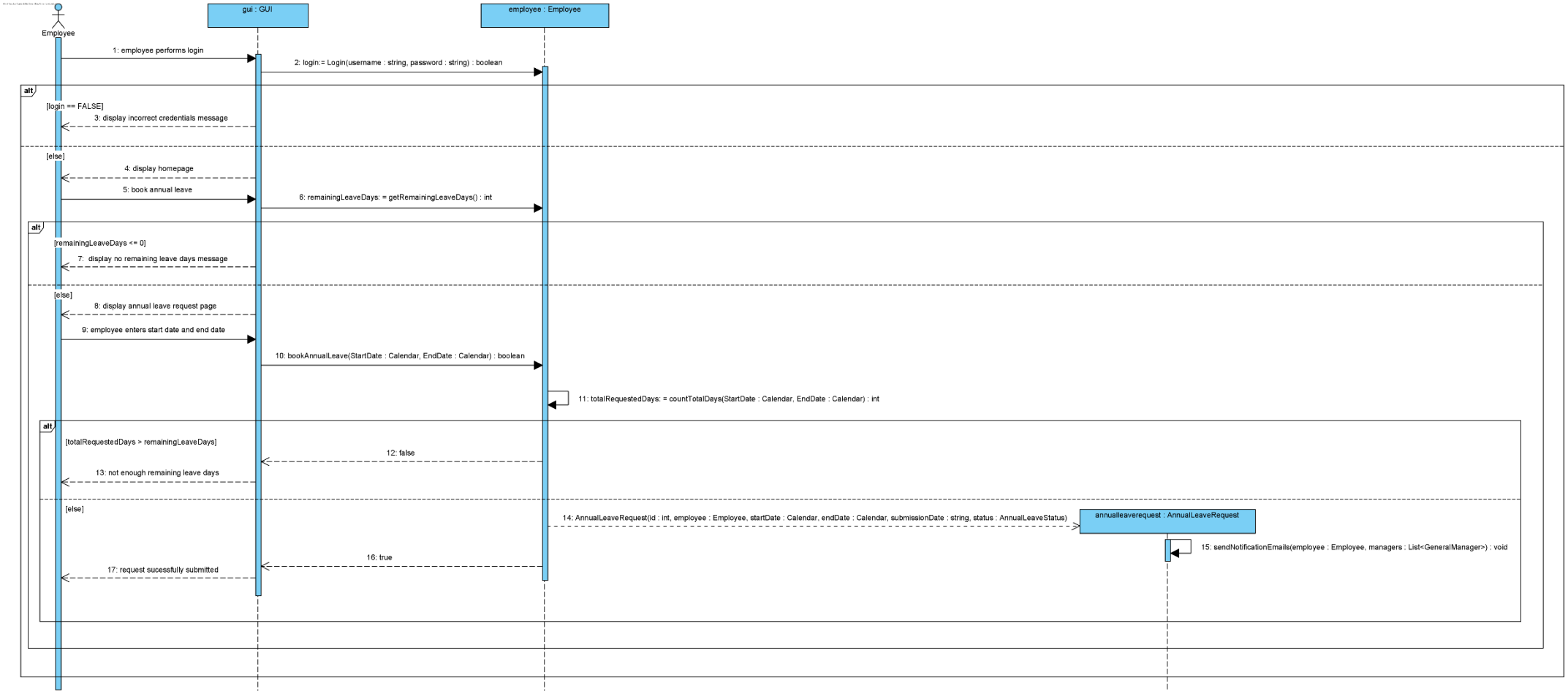
Overall, these decisions have been made to further improve the system's design so that it meets the specified requirements in an efficient manner. Despite these small changes, the class diagram is still reflective of the previously designed domain model and can be considered an extension of it.

4. Sequence Diagram 1: Requesting Annual Leave

Basic Employees, Managers and HR managers within FDM will be able to use the EPS to request annual leave between certain dates. Once the employee has entered their desired dates, the system will check that the total number of days selected does not exceed the amount that the particular employee is entitled to. Only if it passes this check, will the request be sent to the manager(s) for approval. The submitted request will be set to a status of 'Pending', and the employee will get a notification email confirming the request.

The pre-requisites for this task include:

- The user must have an internet connection in order to access the EPS.
- The user who performs the task must be the owner of the account.
- The user account must have already been verified via email in order to access the booking annual leave system functionality.

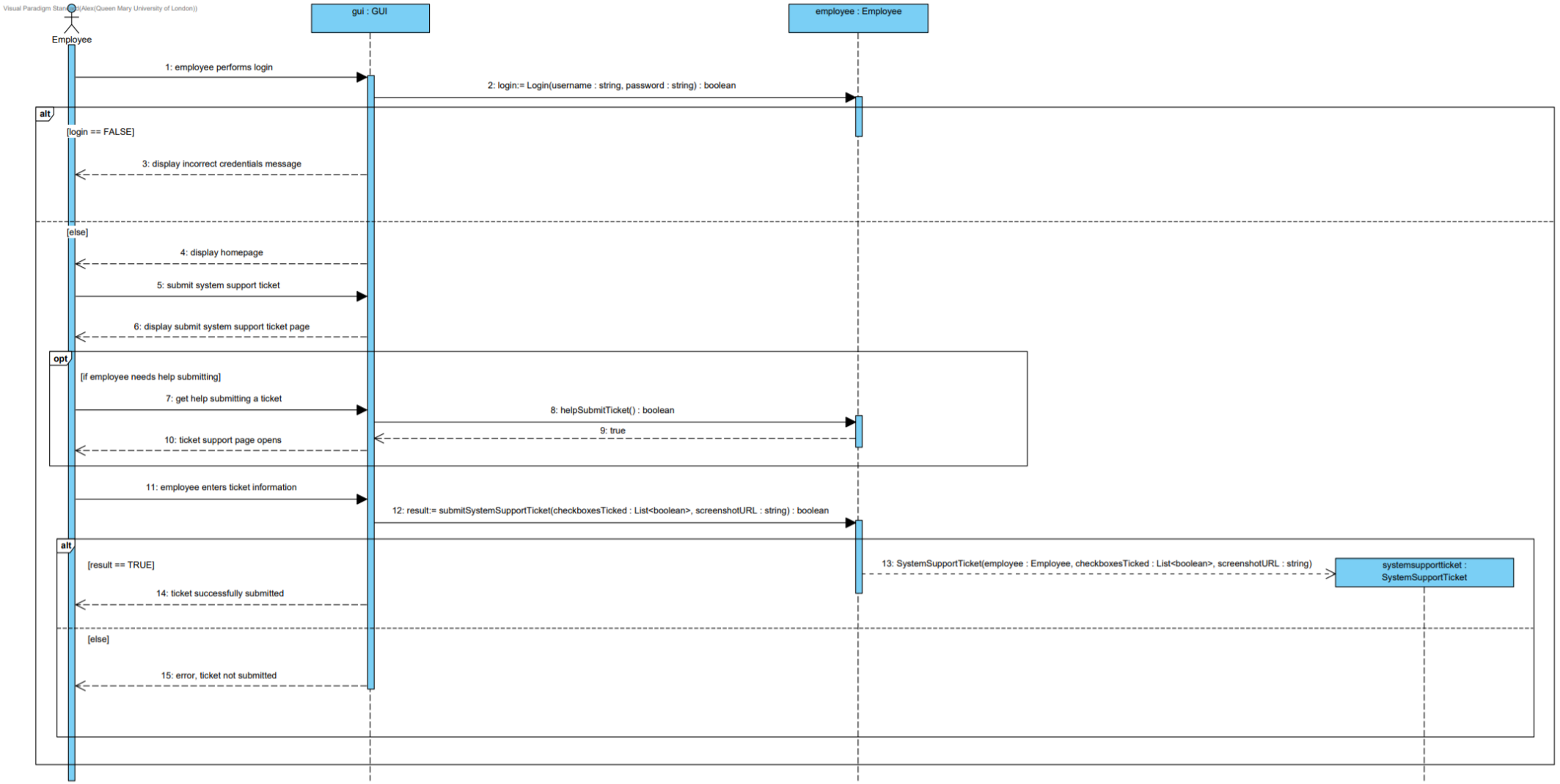

Employee
gui : GUI
employee : Employee
1: employee performs login
2: login:= Login(username : string, password : string) : boolean
alt
[login == FALSE]
3: display incorrect credentials message
[else]
4: display homepage
5: book annual leave
6: remainingLeaveDays: = getRemainingLeaveDays() : int
alt
[remainingLeaveDays <= 0]
7: display no remaining leave days message
[else]
8: display annual leave request page
9: employee enters start date and end date
10: bookAnnualLeave(StartDate : Calendar, EndDate : Calendar) : boolean
11: totalRequestedDays: = countTotalDays(StartDate : Calendar, EndDate : Calendar) : int
alt
[totalRequestedDays > remainingLeaveDays]
12: false
13: not enough remaining leave days
[else]
14: AnnualLeaveRequest(id : int, employee : Employee, startDate : Calendar, endDate : Calendar, submissionDate : string, status : AnnualLeaveStatus)
annualleaverequest : AnnualLeaveRequest
15: sendNotificationEmails(employee : Employee, managers : List<GeneralManager>) : void
16: true
17: request sucessfully submitted

5. Sequence Diagram 2: Submitting a System Support Ticket

The Employee Portal System provides the feature of submitting a system support ticket. Every employee can submit a ticket to the system Administrators in regards to any queries and/or problems they may have with the system. In the event that an employee requires help with the submission of a ticket, they are able to access a helper feature. Once the employee finishes entering the ticket contents (which may include a screenshot image), the ticket can be submitted. After submission, the ticket will be assigned an *open* status until it gets a response from the Administrator.

The pre-requisites for this task include:

- The user must have an internet connection in order to access the EPS.
- The user who performs the task must be the owner of the account.
- The user account must have already been verified via email in order to access the booking annual leave system functionality.


Visual Paradigm Standard(Alex(Queen Mary University of London))
Employee
gui : GUI
employee : Employee
1: employee performs login
2: login:= Login(username : string, password : string) : boolean
alt
[login == FALSE]
3: display incorrect credentials message
[else]
4: display homepage
5: submit system support ticket
6: display submit system support ticket page
opt
[if employee needs help submitting]
7: get help submitting a ticket
8: helpSubmitTicket() : boolean
9: true
10: ticket support page opens
11: employee enters ticket information
12: result:= submitSystemSupportTicket(checkboxesTicked : List<boolean>, screenshotURL : string) : boolean
alt
[result == TRUE]
13: SystemSupportTicket(employee : Employee, checkboxesTicked : List<boolean>, screenshotURL : string)
systemsupportticket : SystemSupportTicket
14: ticket successfully submitted
[else]
15: error, ticket not submitted