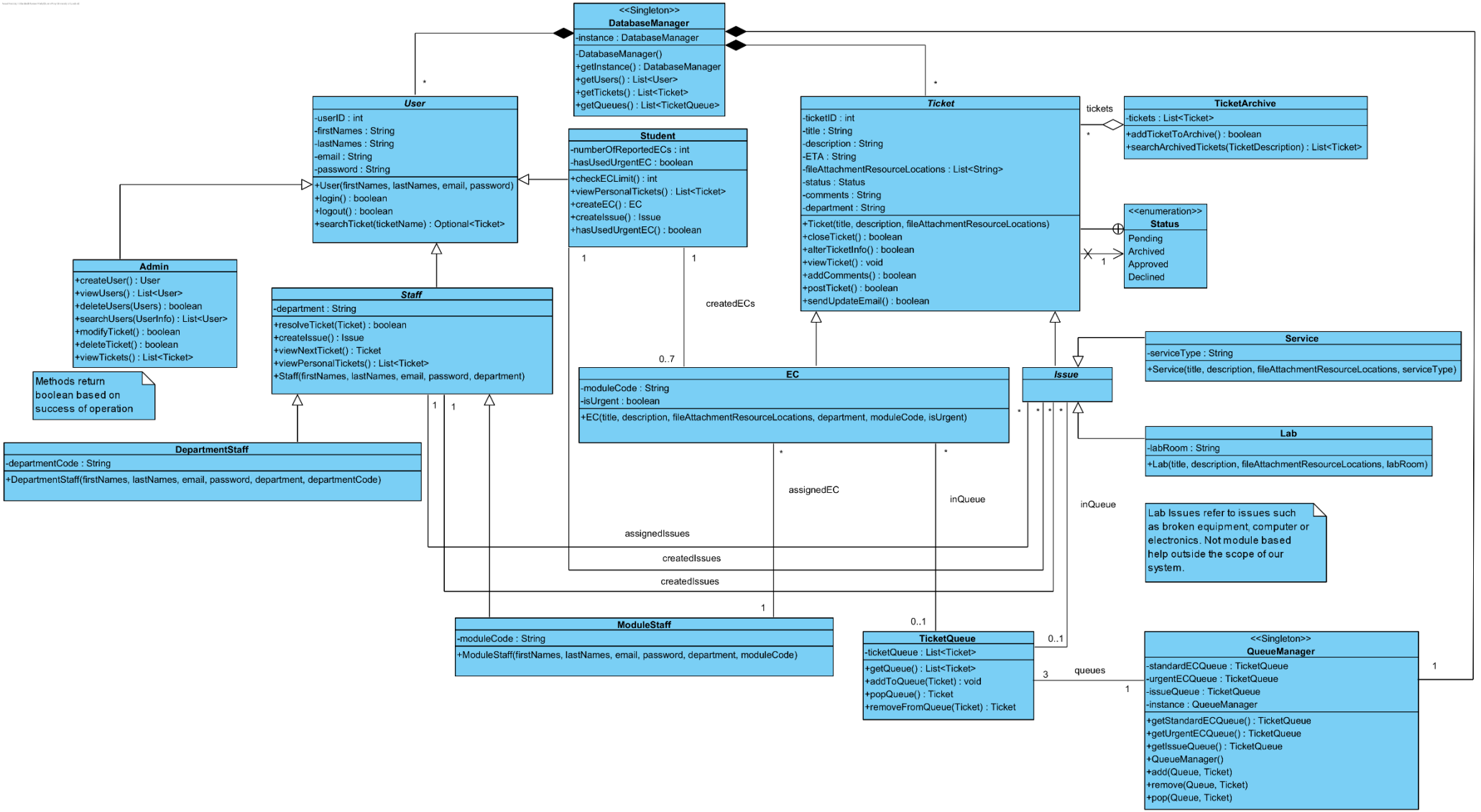
Group 4

QMUL Issues and Feedback System

ECS506U Software Engineering Group Project

Design Report

1. Class Diagram



2. Traceability matrix

Class	Requirement	Brief Explanation
Ticket	RQ 19	The ticket class has all the necessary attributes for creating a ticket.
Ticket	RQ 22	The ticket class has an attribute which allows file attachments to be included when creating a ticket.
User	RQ 29	Student and Staff creating their tickets will be able to view their tickets via the viewPersonalTickets operation.
QueueManager	RQ 30	In the QueueManager class, there are separate queues for urgent and standard EC tickets. When EC ticket is created, it will be placed into the respective queue based on whether it's an urgent EC or standard EC.
Ticket	RQ 31	Both EC and Issues class have a department attribute which will assign each ticket to a specific department to handle. Additionally, the EC class is given a module code so the correct staff with that module code can respond to the ticket.
TicketArchive	RQ 43	Once a ticket has been resolved by the staff and closed by the student, the ticket is moved into the archive.
User	RQ 44	All users including staff have IDs that are linked to their tickets.
Database Manager	RQ 46	The database manager has an operation called getTickets which can be used to view all issue-related tickets in the database. Based on the number of tickets received, we can determine the status of these services and notify users about them via the sendUpdateEmail operation.

The following requirements were later defined as data requirements.

Ticket	RQ 32	The ticket class has an attribute called ETA which will display how long it will take to resolve the ticket.
Admin	RQ 34	The admin class has an operation called modifyTicket which can be used to modify existing tickets.
Ticket	RQ 35	There is an attribute “comment” to store comments about a ticket.
Staff	RQ 38	When staff respond to a ticket, an email is sent to the student who created the ticket. There is an operation called sendUpdateEmail in the Ticket class which staff can use to send user an email.
Database Manager	RQ 45	The database manager has an operation called getTickets which can be used to view all issue-related tickets in the database. Based on the number of tickets received, we can determine the status of these services.
Database Manager	RQ 46	The database manager has an operation called getTickets which can be used to view all issue-related tickets in the database. Based on the number of tickets received, we can determine the status of these services and illustrate these results as a graph.
Admin	RQ 48	The admin class has operations such as viewUsers, viewTickets to manage tickets and handle user details.
Admin	RQ 51	The admin class has an operation called createUsers which can be used to create new users and send them their credentials.

3. Design Discussion

Justification for Design Decisions

Whilst most of our design decisions are clear, a few require further explanation. Firstly, our implemented Ticket Queue system. This went through multiple iterations in development, involving a player role pattern, an interface for implementation and multiple subclasses of the queue for each type. Eventually, we decided to simplify this to one class, TicketQueue, and a QueueManager singleton, holding three Queues with helper methods for interacting with them. This simplified the section, and allowed for the creation of more Queues in the future if needed. Another design decision we enforced was containing the Enum 'Status' inside the Ticket class. The reasoning for this decision is twofold. Firstly, we wanted to ensure that it was clear that this Status is coupled with the Ticket, and secondly, we wanted to hide this Enum from the rest of the System, as it is only used by the Ticket class. The final design decision requiring justification is the absence of certain attributes in constructors in our System. Namely, the Ticket constructor not requiring a status or ETA. This is because status is set automatically to pending, and ETA is set later by the Staff member assigned to it. Admin also does not have a constructor, as it has no unique attributes and simply inherits from User. The same can be said for Issue and Ticket, as an Issue takes most functionality from Ticket, as its child classes are the ones with specified attributes.

Justification for Associations and Relationships

Now that we have reached a development stage requiring less abstraction, it is important to highlight the relations between our classes. Firstly, it is important to highlight the 1 to 3 relation between QueueManager and TicketQueue. This is simply in place to accommodate the three Queues our System has: Standard EC, Urgent EC and Issue. We also have a 0 to 7 relation between Student and EC, as a student may only have 7 ECs in a given year. We also have created a DatabaseManager singleton, which is composed of Users, Tickets and the QueueManager. The reason for this decision was to ensure that all tickets are reliant on accessing the database, and are not stored on a local machine. For example, all users ECs should not be available to any Student.

Differences of Entities from Domain Model

The first difference between our Domain Model and Class Diagram is that we are now representing less abstract entities. Firstly, we have a DatabaseManager. This class interacts with our database and is populated with helper methods for accessing its data. Based on requirement analysis, we have also simplified the children of the Staff class to only two, Module and Department. This was because we realised that we did not need individual staff classes for each module, as we could implement this in a much simpler way with attributes. We have also added child classes to the issue class based on our Requirements Documentation, as we decided it would be better to allow different types of issues that require differing information. Based on client feedback, we realised we need a way to store tickets once closed. To implement this, we created a TicketArchive class, an aggregation of Tickets that have been marked as completed. This allows users to reopen tickets if they are unhappy with the results, as per our specified requirement for this functionality.

Implemented Design Patterns

We also implemented effective design patterns when designing our software. Firstly, we have decided to make our DatabaseManager class a singleton, as we want to ensure there is only one instance of this class, so as to not receive duplicate tickets from our database. By this logic, we have also made QueueManager (our ticket queues) and TicketArchive singletons. We want to ensure we have no duplicate or redundant queues, and we wish to ensure all archived tickets are in the same place. We have also implemented an abstraction occurrence pattern for Issues and ECs. Fundamentally, these are different entities, however they share similar properties such as being opened, closed and added to a queue. As such, we have created a generalised abstract ticket class from which they both inherit.

4. Sequence Diagrams

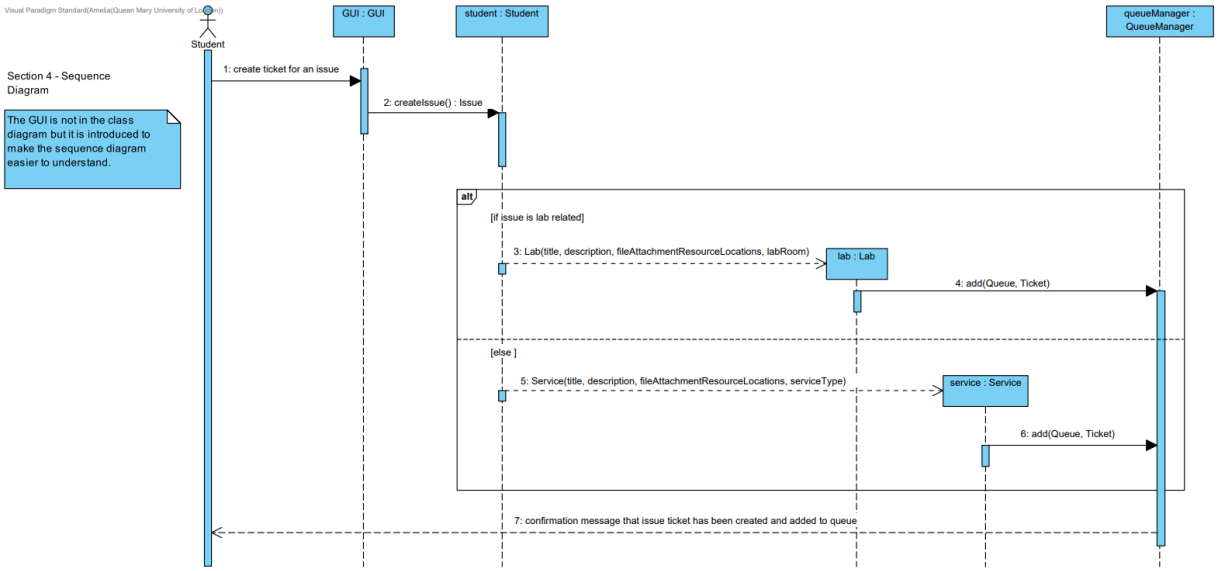
Sequence Diagram 1: Creating an issue ticket.

Scenario:

The scenario for this sequence diagram is for the process of creating an issue ticket which would be launched by a student who needs to report an issue with a service or lab. The system allows for this process through a GUI which the student will interact with to create a ticket and the system will recognise that the student has the functionality to raise an issue. Then, it will be determined by the student and system whether this issue is lab related or service related. Depending on which one the student wishes to create a ticket for, the ticket will be created through the relevant lab and service tickets. After this takes place, the queue manager will add the ticket to the queue and confirm a successful creation of the issue ticket to the student user.

Pre-requisites:

A pre-requisite is that the student actor will be logged into their account for the system, and they will have recognised an issue with a service or lab that they need to report motivating them to create an issue ticket. Another pre-requisite is that it is assumed the system has a ticket queue manager which allows for managing the queues of issues tickets and ensures the handling of any open tickets for issues. An additional pre-requisite is that the system can handle any errors or exceptions which could occur whilst creating an issue ticket.



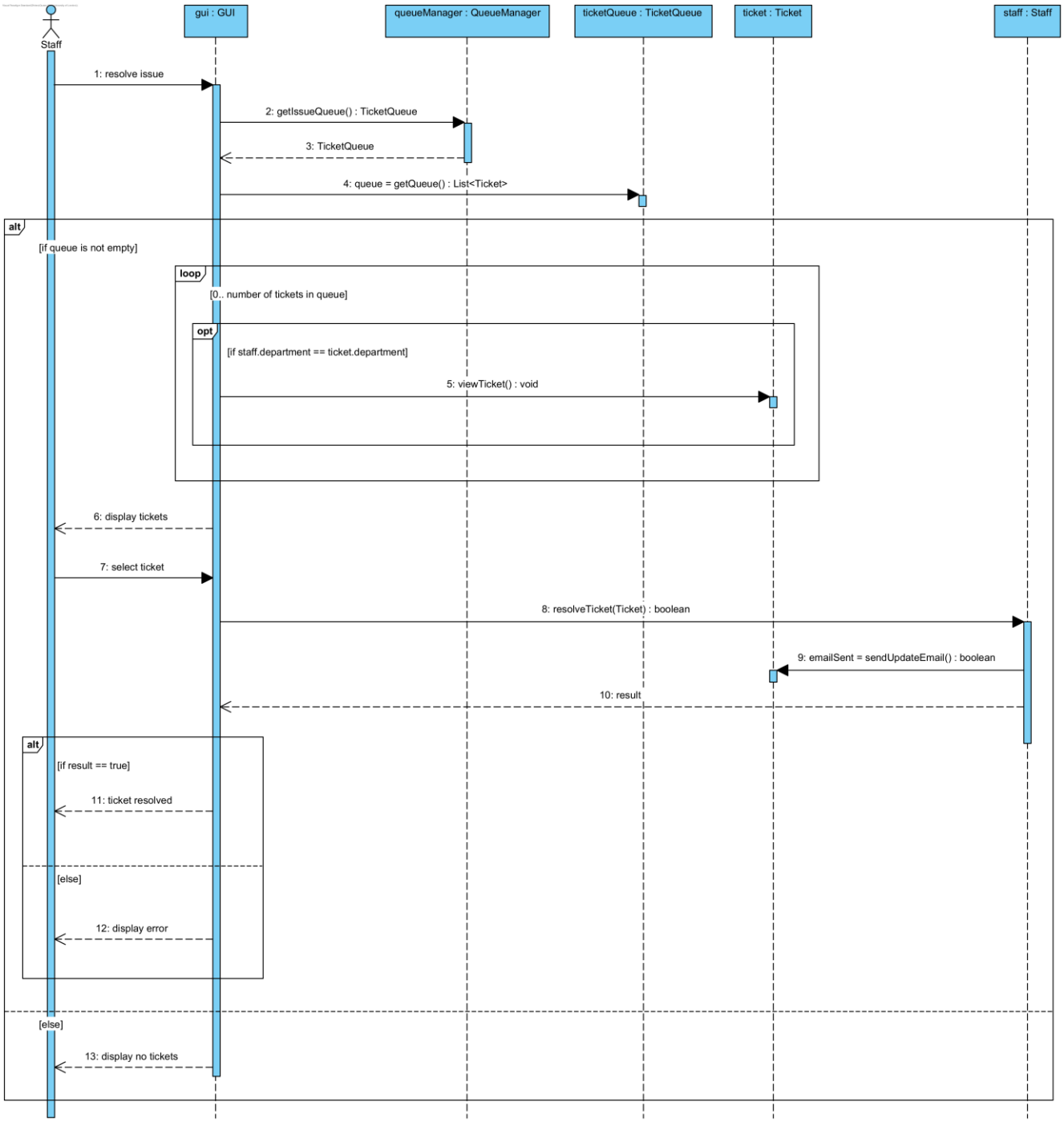
Sequence Diagram 2: Resolve Issue

Scenario:

Staff goes to resolve an issue, the issue queue is taken and filtered based on the staff's department, so they can only see the related issues to their department. These issues are then displayed to the staff, who then selects which issue to resolve. The issue is then resolved, and an update email is sent to notify users of a change in status. A message is then displayed based on whether the issue was resolved or failed to be resolved.

Pre-requisites:

The staff actor must have a stable internet connection throughout the process. The staff must also be logged in to the system.



Sequence Diagram 3: Creating an Extenuating Circumstance Ticket

Scenario:

The scenario for this sequence diagram is for the process of creating an Extenuating Circumstance (EC) ticket which would be launched by a student who needs to submit one. The system allows for this process through the GUI that the student will interact with to tell the system that they want to create an EC. Then, the student will fill out the details of their ticket. If the student has chosen to create an urgent EC, then the system will perform a check to see if the student has not already submitted an urgent EC, if so then the student will be denied from submitting their EC and an error message will be displayed. If the student has not submitted an urgent EC previously then they will be able to submit with no issues and their ticket will be added to the urgent queue. This is also true for creating a standard EC but there is no check for if the student has used an urgent EC before in this case and the ticket will be added to the standard queue.

Pre-requisites:

A pre-requisite is that the student actor will be logged into their account for the system (therefore they also must have internet access), and they will have submitted less than 7 Extenuating Circumstances. Another pre-requisite is that it is assumed the system has a ticket queue manager which allows for managing the queues of issues tickets and ensures the handling of any open tickets for EC's. An additional pre-requisite is that the system can handle any errors or exceptions which could occur whilst creating a ticket.