UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS301 - Software Engineering

Coviduous

Software Requirements and Design Specifications

June 19, 2021

Team Name: CAPSlock
Team Members: * - *indicates team leader*

Name	Surname	Student Number
Njabulo*	Skosana*	u18089102*
Rudolf	van Graan	u16040865
Clementime	Mashile	u18139508
Peter	Okumbe	u18052640
Chaoane	Malakoane	u18202374

Contents

1 Project Owner

The owners of this project are Matthew Gouws and Peter Rayner, who are both representatives from Amazon Web Services. They can be contacted at cos301@amazon.com.

2 Project Vision and Objectives

Coviduous is a COVID-19 management system that aims to solve the issue of manually managing employee health and workplace safety. This is achieved by handling the amount of people allowed in office spaces. The capacity of people is determined using the country alert level and using floor plans. Capacity management will ensure that a suitable distance can be maintained between employees who wish to work in the office.

The core functionality of the system is to allow the reading and changing of floor plans, the system should calculate the number of employees allowed on the premises. The system should come up with a way to position the employees to ensure a safe distance can be maintained. The system should manage the amount of employees within a office space. It should have end to end user encryption for security purposes. The above are the functional requirements that we will prioritize as a team in the development of the system.

In addition, the system should:

- Allow the employer creation, modification, and deletion of employee shifts.
- Contact tracing of employees infected by COVID-19.
- Employers should be alerted if capacity restrictions are being violated.
- Employers should be alerted of changes in COVID-19 regulations e.g level change.
- Employers should be able to check if COVID-19 equipment such as sanitizer are available/-manage inventory, cross check with number of employees.

3 Functional Requirements

R1 The system should allow registration and login, and differentiate between **employers** and **employees**.

R2 The system should allow employers to read and manipulate office floor plans.

R3 The system should assign desks and personal office spaces to employees.

R4 The system should optimize the number of workers allowed on the premises.

R5 The system should optimize the positioning of workers within an office space, while keeping team members close to each other.

R6 The system should provide a user interface through which a desk or office space can be booked.

R7 The system should provide a method to contact trace employees who have been infected by COVID-19.

R8 The system should be able to alert employees when capacity restrictions are being reached.

R9 The system should alert users of changes in South Africa's COVID-19 status.

R10 The system should provide a way for employers to check/manage statistics about their premises, including how sanitized rooms are, how much medical and cleaning equipment they have, and how many employees are present.

4 User Characteristics

The user should be to operate a computer and/or be able to use a smartphone to download and use the application. The user should have access to the internet and understand how to make bookings. There are two types of users who would make use of Coviduous: they are the **Employers (Admin)**, and the **Employees (User)**:

Admin (Employer)

- a person who manages the platform.
- a person who creates an account for their company.
- a person who inputs room dimensions and optimizes floor plans.
- a person who routinely manages users and assignment of shifts.
- a person who ensures smooth operation of the system.
- a technical person who understands the functionality of the system.

User (Coviduous Employee)

- a person who is employed under a specific company.

- a person who wants to work in the company's office space.
- a person who is assigned working shift hours.
- a person who wants to book an office space.
- a person who can operate and owns a computer or smartphone.

5 User Stories

User Subsystem

- As a User, I want to register an account so that I can see company updates.
- As a User, I want to update my account details so that I can make changes to my personal user account information.
- As an Admin, I want to delete a user account so that I manage access to company information.

Floorplan Subsystem

- As an Admin, I want to create an office floor plan so that I can receive the maximum amount of people allowed in each office room space.
- As an Admin, I want to update a floorplan so that I can view the updated amount of people allowed in the office space.
- As an Admin, I want to delete a floorplan so that I can remove a floor or room from being available in the office space.

Office Subsystem

- As a User, I want to book an office space so that I can have a reserved space to work in on arrival at the office.
- As a User, I want to view the office space so that I can see which areas of spaces in the office are available for booking.

Announcement Subsystem

- As an Admin, I want to make an announcement so that I can inform employees of changes in the office space or latest COVID-19 updates / regulations.
- As an Admin, I want to delete an announcement so that I can correct a mistake or clear the announcement board.
- As a User, I want to view an announcement so that I can stay informed about working conditions in the office space.

6 System Class Diagram

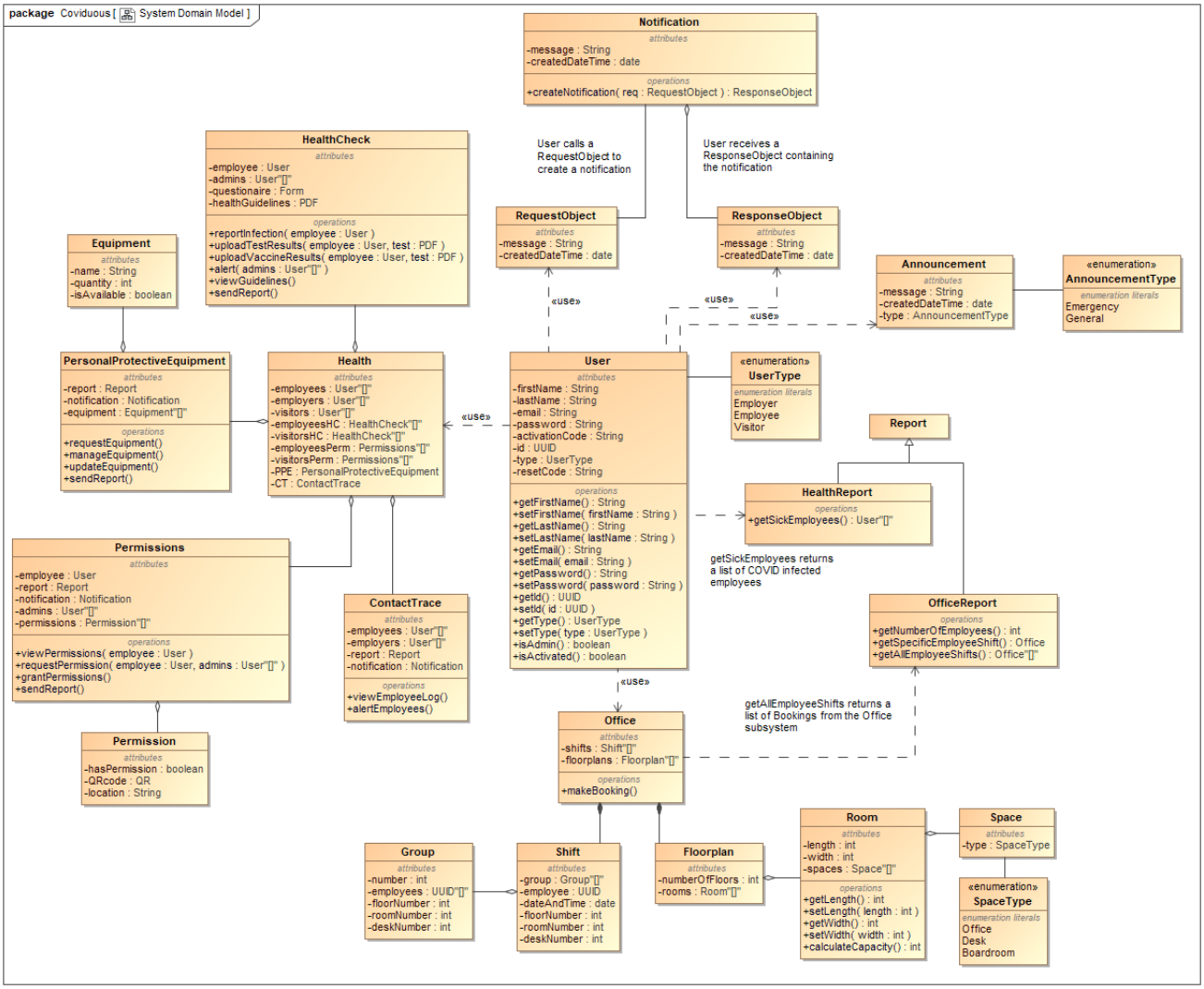

Figure 1: System class diagram of Coviduous

7 Subsystems

7.1 User

7.1.1 Use Cases

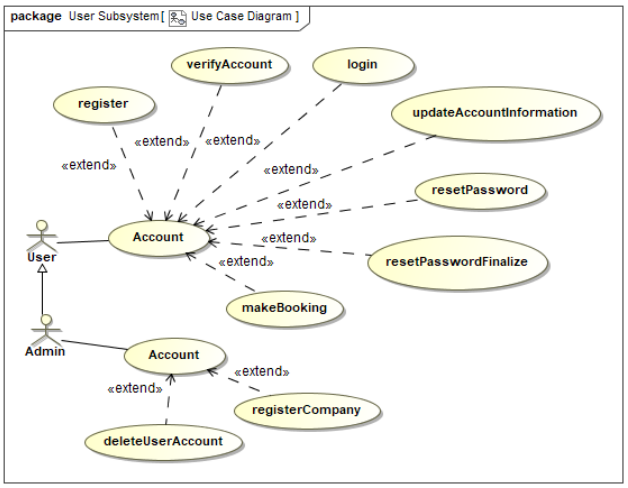

Figure 2: The scope of functionality required from the User Subsystem.

7.1.2 Domain Model

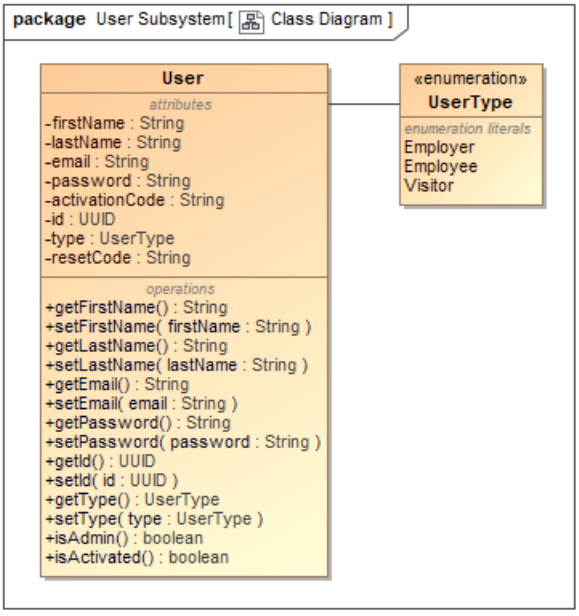

Figure 3: The domain model of the User Subsystem.

7.1.3 Service Contracts

U1 User Subsystem

U1.1 **Register User**: The register user use case should accept the required attributes namely:

- First name

- Last name
- Username
- Email
- Password

U1.2 **Verify Account**: The verification use case is used by a user to validate their account upon initial creation; subsequent email changes will not be validated. This use case requires the calling client to supply an activation code for the account which should be validated.

U1.3 **Login** : This use case accepts the user's credentials, namely a username and password. The system should validate the username and password presented by the user.

U1.4 **updateAccountInfo**: The updated account information use case is only concerned with updating user information not maintained by the system such as first name, last name and email. The system will not do any validation when the user updates their email address.

U1.5 **resetPassword**: Upon the user requesting a password reset, a reset code should be generated, and a reset expiry date should be calculated to be 4 hours from the time a password reset was requested.

U1.6 **resetPasswordFinalize**: When a password request is received, the validation code should be used to retrieve the user object. When the user resets their password it should be validated that the four-hour time frame has not elapsed, if so an exception ResetPasswordTimedOutException should be thrown. If the time frame has not elapsed, the user's password should be reset with the newly supplied password.

7.2 Floorplan

The **Floorplan** subsystem provides the functionality of maintaining the information related to an office floor plan. It defines how the floor plan is structured by calculating the arrangement of spaces per room in a floor, using the given floor space dimensions.

7.2.1 Use Cases

Three use cases have been identified for this subsystem so far, as shown in figure 6. Employers will be tasked at the creating, updating, and deleting of office floor plans. They will have the ability to create a floor plan giving the system a floor's room dimensions as input and receiving capacity management arrangements as output. They will also be able to assign shifts and work spaces to specific employees.

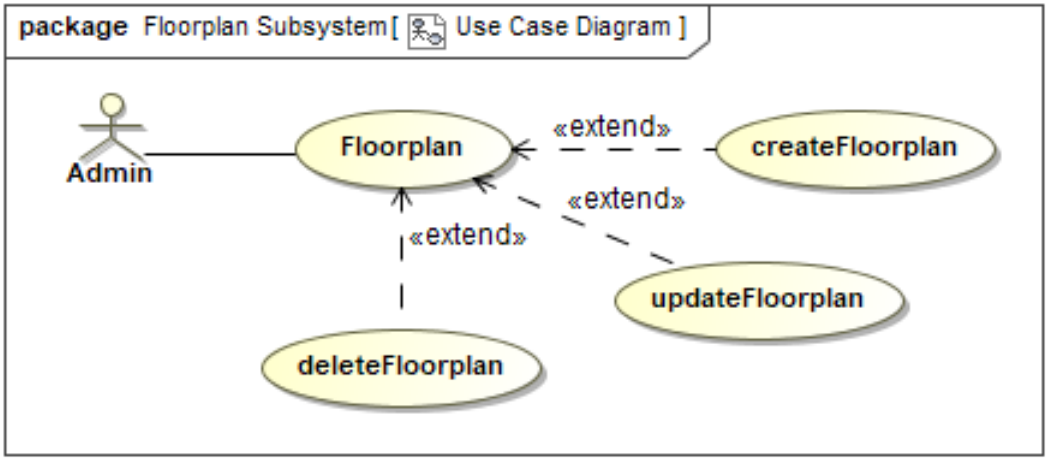

Figure 4: The scope of functionality required from the Floorplan Subsystem.

7.2.2 Domain Model

This subsystem consists of **three** classes. The main **Floorplan** class consists of all the information related to the construction of the specific office floor plan, such as the floor room dimensions, list of rooms, list of employees assigned to the floor, number of occupants in the floor, and maximum number of occupants allowed in the floor. The **Room** and **Space** classes contain information for their respective features as shown in figure 5 below.

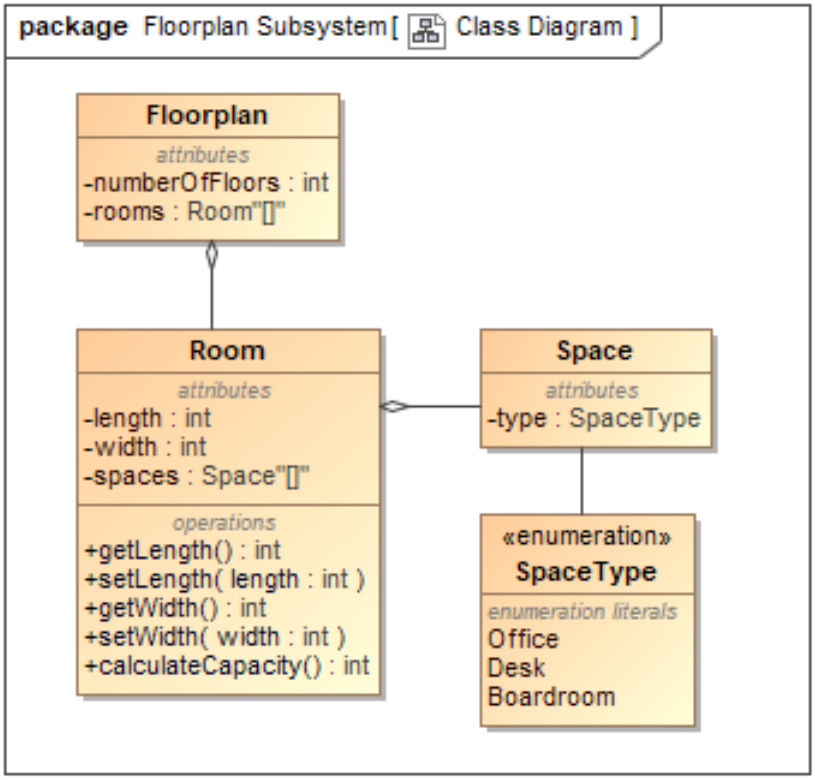

Figure 5: The domain model of the Floorplan Subsystem.

7.2.3 Service Contracts

U2 **Floorplan Subsystem**

U2.1 **createFloorplan:** This use case is initiated by an employer user. They request to create an office floor plan by uploading the dimensions for a specific floor's room as input, and a response of the capacity space arrangements is returned to the user.

U2.2 **updateFloorplan:** This use case is initiated by an employer user. They request to updaate an office floor plan by uploading the updated dimensions for a specific floor's room as input, and a response of the new capacity space arrangements is returned to the user. If the update was unsuccessful, an Exception is thrown.

U2.3 **deleteFloorplan:** This use case is initiated by an employer user. They request to delete an existing office floor plan, and a response of the deletion is returned to the user. If the deletion was unsuccessful due to an attempt to delete a non-existing floor plan, an Exception is thrown.

7.3 Office

The **Office** subsystem provides the functionality of maintaining the information related to an office space with regards to the number of floors, floor plans per floor, the number of occupants in the entire office space, and the max number of occupants allowed in the office space. This subsystem makes use of the **Floorplan** subsystem.

7.3.1 Use Cases

Four use cases have been identified for this subsystem so far as shown in figure 6. Employees will be able to view a visualization of the entire office space per floor, including the number of rooms and desks per room in the respective floor. Employees will have the ability to book/reserve a space and time for working in a particular room and/or desk, as well as view to which rooms or desks they have been assigned working spaces for.

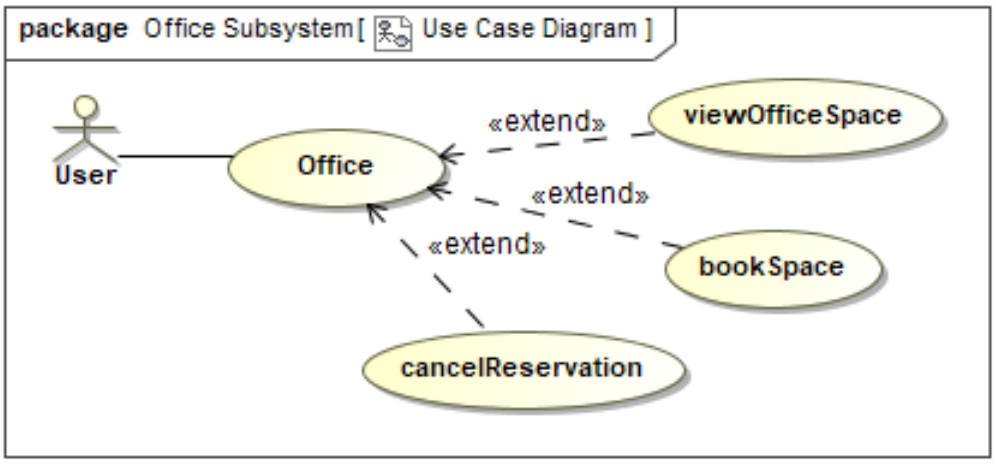

Figure 6: The scope of functionality required from the Office Subsystem.

7.3.2 Domain Model

This subsystem consists of one main class, which interacts with both the **Floorplan** and **Shift** subsystems. The Office class consists of a list of shifts, a list of floor plans for each floor, the number

of floors, the current number of occupants in the office space, as well as the maximum number of allowed occupants in the office space.

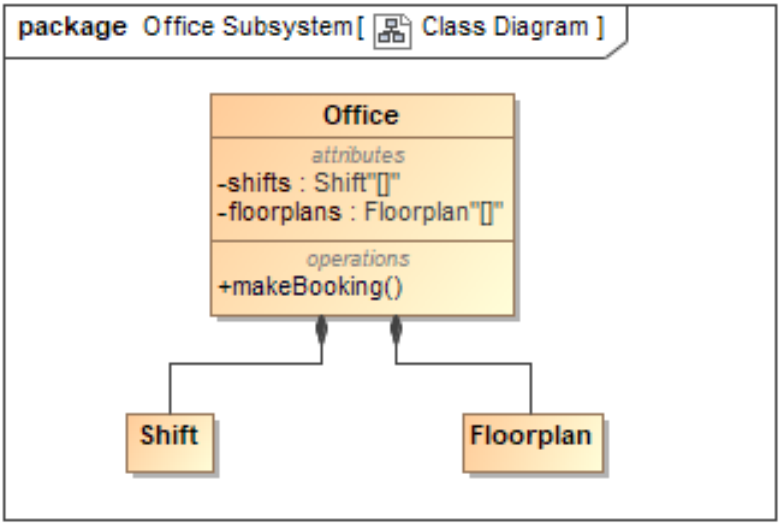

Figure 7: The domain model of the Office Subsystem.

7.3.3 Service Contracts

U3 **Office Subsystem**

U3.1 **viewOfficeSpace** This use case is initiated by an employee user. They request to view the office space for a specific floor, and a response 2D display of the floor's office space with its available rooms and desks is returned to the user.

U3.2 **bookRoom** This use case is initiated by an employee user. They request to book a room in a specific floor for a desired time, and a response message is returned to the user. An Exception is throw if the booking was not possible due to the maximum number of occupants reached in a room or the room not being available.

U3.3 **bookDesk** This use case is initiated by an employee user. They request to book a desk in a specific room for a desired time, and a response message is returned to the user. An Exception is throw if the booking was not possible due to the maximum number of occupants reached in a desk or the desk not being available.

U3.4 **cancelReservation** This use case is initiated by an employee user. They request to cancel a previously made booking for a room or desk space, and a response message is returned to the user. An Exception is throw if the cancellation was not successful due to a cancellation of a non-existing booking.

7.4 Shift

The **Shift** subsystem provides the functionality of maintaining the information related to the assignment of office working shift hours and office spaces to employees.

7.4.1 Use Cases

Three use cases have been identified for this subsystem so far, as shown in figure 8. Employers will be tasked at the creating, updating, and deleting of employee office shifts. They will have the ability to create a shift giving the system an employee's ID, their assigned team number, floor number, room number, and desk number as input. They will also be able to assign shifts to teams so that employees in specific teams can work and collaborate in close proximity with one other.

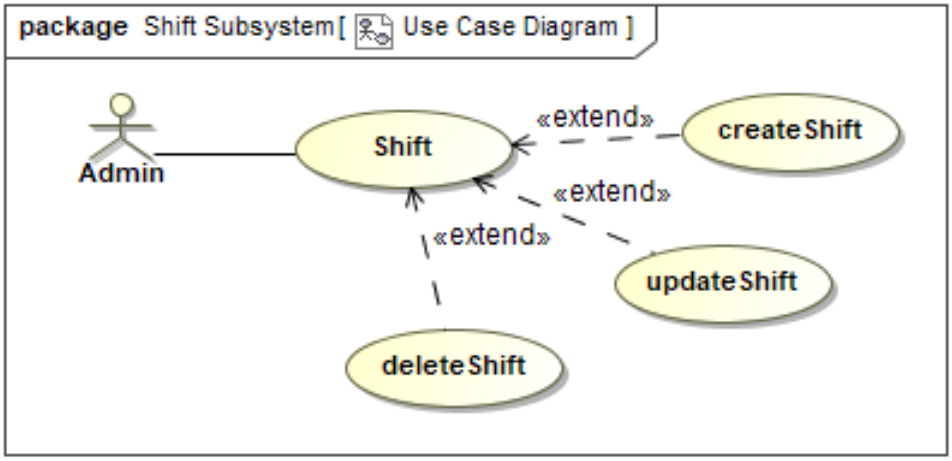

Figure 8: The scope of functionality required from the Shift Subsystem.

7.4.2 Domain Model

This subsystem consists of **two** classes. The main **Shift** class consists of all the information related to the construction of assigned shifts for respective employees, including their group number, assigned shift working time, floor number, room number and desk number. The **Group** class contains information relating to the assignment of shifts for different groups of employees, including those working together as a team.

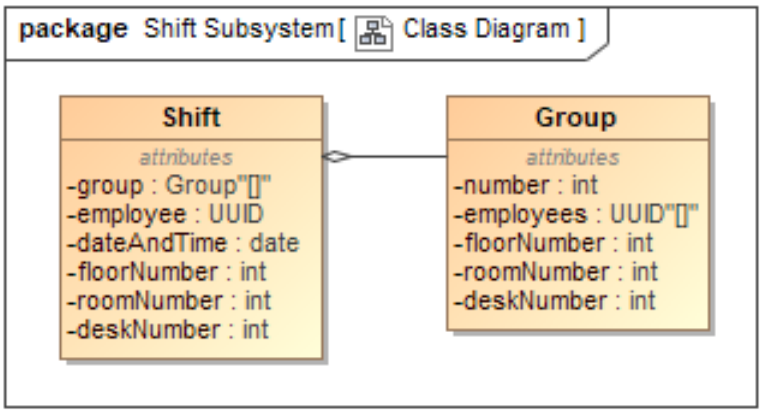

Figure 9: The domain model of the Shift Subsystem.

7.4.3 Service Contracts

U4 **Shift Subsystem**

U4.1 **createShift** This use case is initiated by an employer user. They request to create an office shift for a specific employee or group of employees (teams or those employees working at the office for the same scheduled time), and a response message confirmation of shift creation is returned to the user.

U4.2 **updateShift** This use case is initiated by an employer user. They request to update a scheduled office shift for a specific employee or group of employees (teams or those employees working at the office for the same scheduled time), and a response message confirmation of update for an existing shift, is returned to the user.

U4.3 **deleteShift** This use case is initiated by an employer user. They request to delete a scheduled office shift for a specific employee or group of employees (teams or those employees working at the office for the same scheduled time), and a response message confirmation of deletion for an existing shift, is returned to the user.

7.5 Health

The **Health** subsystem is responsible for managing the health of all employees and visitors in the company. It is responsible for giving employees and visitors a chance to complete a health assessment and based on the result grant them access permissions to the office space.It allows employees the ability to upload health documents.It allows the employer to manage permissions granted to employees, assess health documents, manage protective personal equipment and perform contact tracing.

7.5.1 Use Cases

In order to describe the use cases, users where broken down into three categories which are employer, employee and visitor:

Employee

- Complete Health Check
- Report Infection
- View health guidelines
- Upload COVID-19 health documents
- View permissions
- Request permissions

Visitor

- Complete Health Check
- View health guidelines
- View permissions

Employer

- View permissions
- Grant permissions
- Manage Personal Protective Equipment
- View employee history log
- Send employees notifications

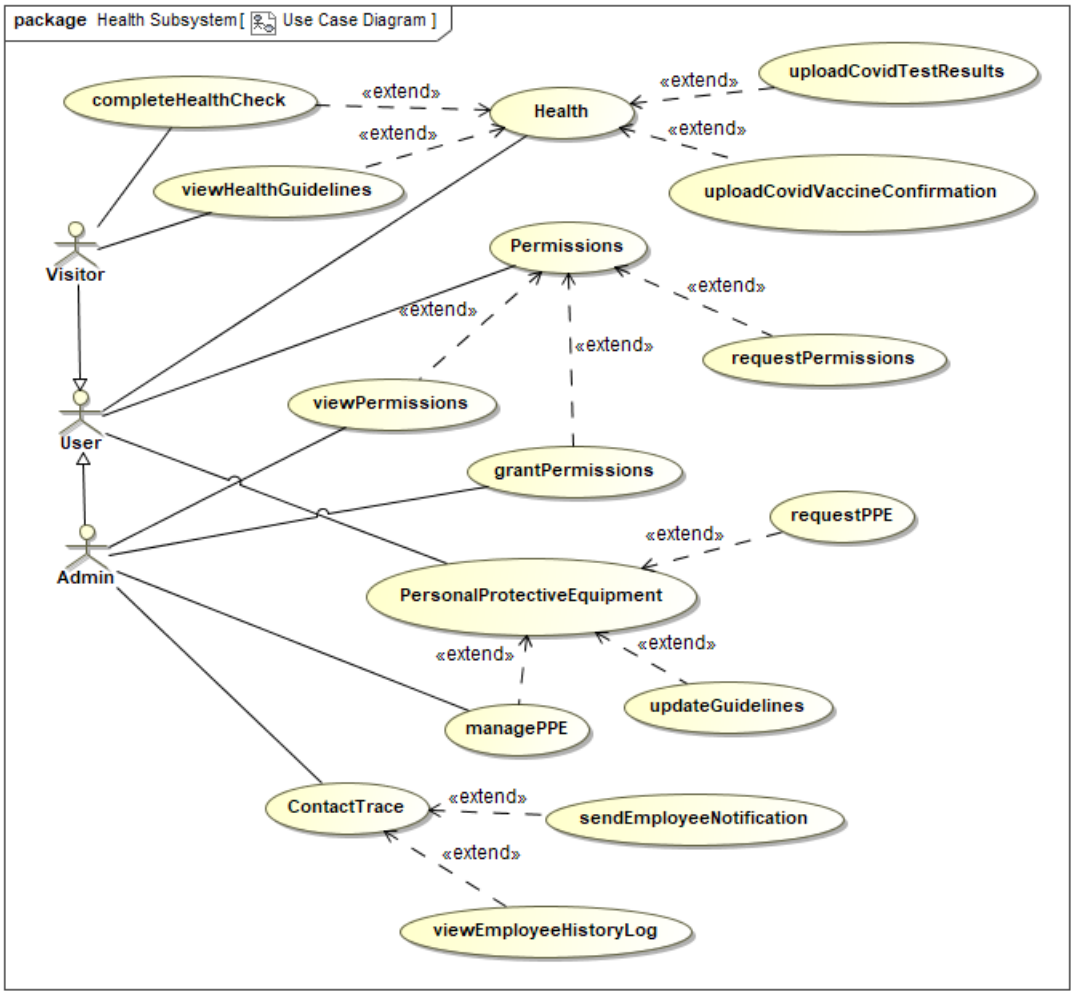

Figure 10: The scope of functionality required from the Health Subsystem.

7.5.2 Domain Model

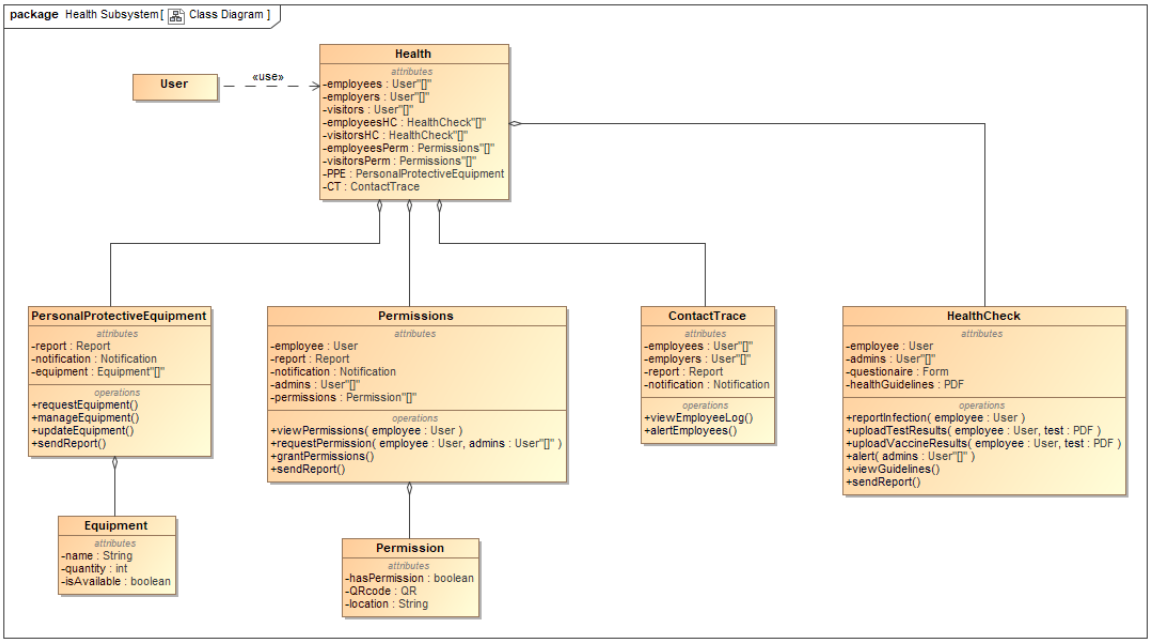

Figure 11: The domain model of the Health Subsystem.

7.5.3 Service Contracts

An employee can update their health status by completing a health check, reporting if they are infected with COVID-19, view company guidelines and upload pdf documents of medical records regarding COVID-19. A visitor can complete a health check and view company guidelines. A employee can view permissions granted to them by the system or the those granted by the employer, if an employee is denied access by the system they can send a request to the employer to review their permission status. A visitor can view their granted permissions which are determined by the system. Employees need enough personal protective equipment in their working area, if an employee requires any protective equipment they can request equipment from the employer. The employer can view permissions the system has granted to users, they can also revoke or give permissions to users and confirm if users medical documents are valid. The employer can manage the personal protective equipment in offices to ensure employees are safe and attend users requests regarding the personal protective equipment. The employer can update company health guidelines. The employer can perform contact tracing and notify employees who may have been exposed to the COVID-19 virus. The employer can view employees history log to track employees.

U5 **Health Subsystem**

U5.1 **completeHealthCheck**

U5.2 **reportInfections**

U5.3 **viewHealthGuidelines**

U5.4 **uploadCovidTestResults**

U5.5 **uploadCovidVaccineConforimation**

U5.6 **viewPremission**

U5.7 **requestPermission**

U5.8 **grandPermission**

U5.9 **requestPPE**

U5.10 **updateGuidelines**

U5.11 **managePPE**

U5.12 **sendEmployeeNotification**

U5.13 **viewEmployeeHistoryLog**

7.6 Notifications

The **Notifications** subsystem deals with creation and sending of notifications between users (usually an employee and an employer). The scope that this subsystem requires is access to the User subsystem. It is only used to send a message; it does not deal with any identifying information, other than the user the notification is for.

7.6.1 Use Cases

Two use cases have been identified for this subsystem so far, **notifyEmployerOfCompletedReport** and **notifyEmployeeOfCovidRisk**. Employers will be notified when an employee has completed a health report and/or made a successful booking, and employees will be notified if they are at risk of contracting COVID.

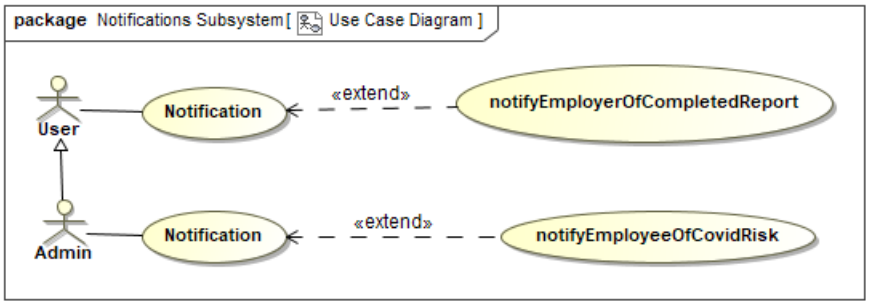

Figure 12: The scope of functionality required from the Notifications Subsystem.

7.6.2 Domain Model

This subsystem consists of three classes, **Notification**, **RequestObject** and **ResponseObject**. The Notification class deals with receiving requests to send notifications. A user calls a RequestObject to generate a new request, which the recipient receives in a ResponseObject. Each class contains a message and the date and time the notification was created.

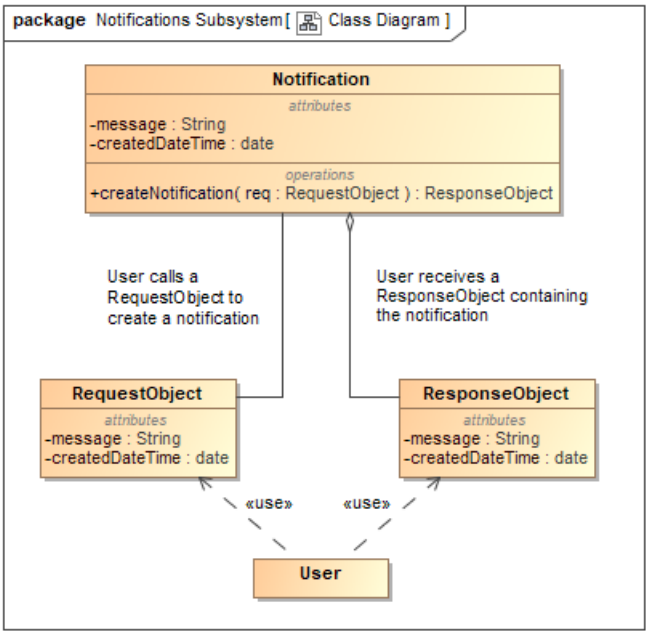

Figure 13: The domain model of the Notifications Subsystem.

7.6.3 Service Contracts

U6 **Notifications Subsystem**

U6.1 **notifyEmployerOfCompletedReport** This use case is initiated by an employee user. Whenever they complete a mandatory health report, a notification will be generated and sent to the employer. This notification can include a message such as "<employee name> has successfully completed a health report" or "<employee name> has failed the health report and is not allowed on the premises". No exceptions will need to be raised here.

U6.2 **notifyEmployeeOfCovidRisk** This use case is initiated by an employer user. Through the Reporting subsystem, the employer will receive information about which employees have been identified as being infected with COVID. They can then send out a notification to inform the employee's coworkers that they may also be infected, and it recommends that they go and get tested. The notification's message could include something along the lines of "An employee you have shared an office space with has contracted COVID-19. It is advised that you have a COVID test done to ensure that you are healthy".

7.7 Announcements

7.7.1 Use Cases

Announcement subsystem is responsible for alerting employee's about changes that can happen in the office any time of the day for example if it happens that the building has catches fire then an announcement of emergency has to be made to all employees.

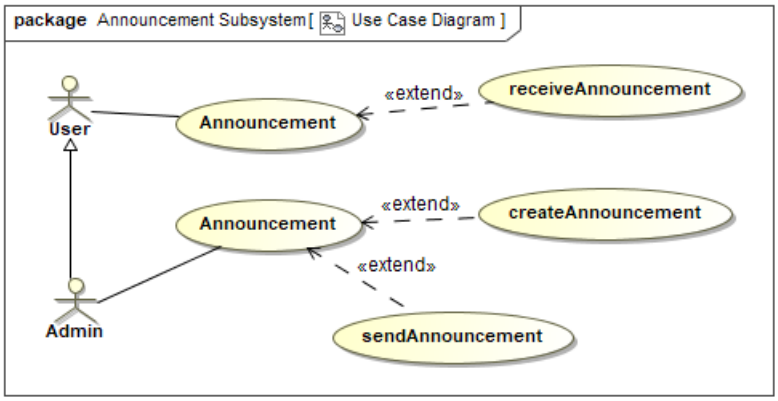

Figure 14: The scope of functionality required from the Announcements Subsystem.

7.7.2 Domain Model

The Announcements subsystem domain is responsible for making announcements for the system. The domain class responsible for the Announcements subsystem is Announcements and AnnouncementsType classes. The Announcements class is responsible to alert employees about the changes in the office and the type of announcement made.The Announcements subsystem domain is responsible for making announcements for the system. The domain class responsible for the Announcements subsystem is Announcements and AnnouncementsType classes. The Announcements class is responsible to alert employees about the changes in the office and the type of announcement made.

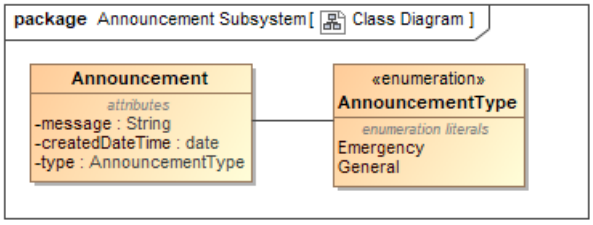

Figure 15: The domain model of the Announcements Subsystem.

7.7.3 Service Contracts

U7 **Announecements Subsystem**

U7.1 **createAnnouncement** This use case is used by the Admin (Employer) in the system, when the Admin wants to make an announcement about any changes or emergencies in the office space.

U7.2 **deleteAnnouncement** This use case is used by the Admin (Employer) in the system, when the Admin wants to delete an announcement to correct an error or clear out the announcement board.

U7.3 **viewAnnouncement** This use case is used by the User (Employee) in the system, when the user wants to see the announcements sent by the Admin (Employer).

7.8 Reporting

The **Reporting** subsystem is used to consolidate reports of company statistics, such as how many employees there are, what their shifts are, and how many employees are sick. The scope of this subsystem is the User subsystem and the Office subsystem. To ensure security of the subsystem, no information can be gathered from other subsystems.

7.8.1 Use Cases

Three use cases have been identified for this subsystem so far, **viewSickEmployees**, **viewEmployeeShifts** and **viewNumberOfEmployees**. Employer users will be able to view various statistics related to their business.

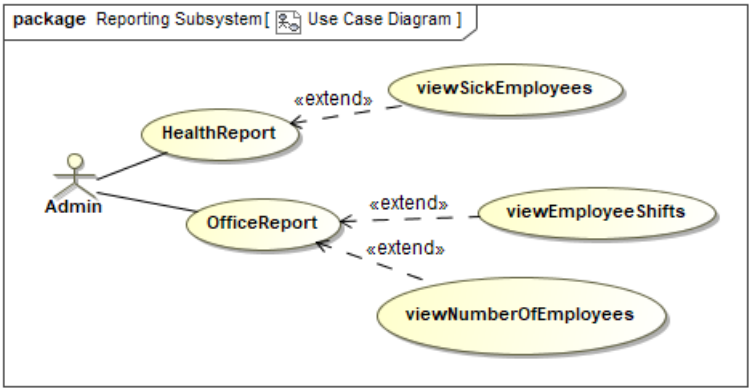

Figure 16: The scope of functionality required from the Reporting Subsystem.

7.8.2 Domain Model

This subsystem consists of two main classes, which inherit from the **Report** class. They are, namely, **HealthReport** and **OfficeReport**. HealthReport contains generated reports of how many employees have been infected with COVID, and OfficeReport contains generated reports of how many employees there are in the business, and what their shifts are.

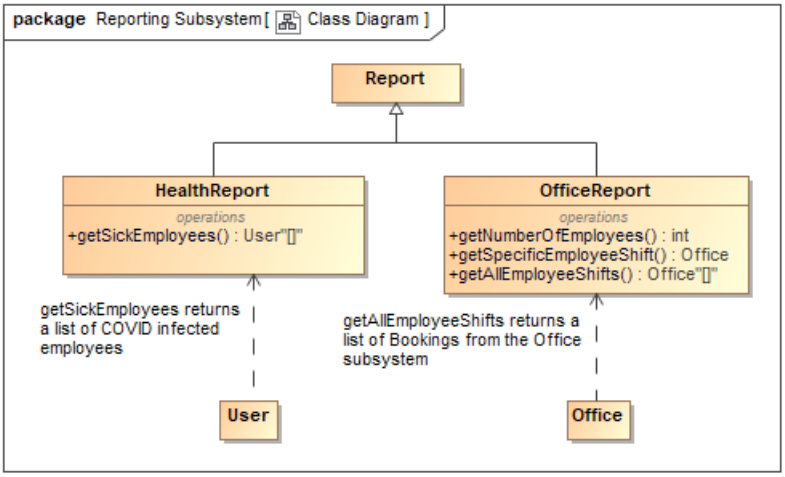

Figure 17: The domain model of the Reporting Subsystem.

7.8.3 Service Contracts

U8 **Reporting Subsystem**

U8.1 **viewSickEmployees** This use case is initiated by an employer user. They request to view how many of their employees are sick, and an integer number is the output.

U8.2 **viewEmployeeShifts** This use case is initiated by an employer user. They request to view a list of employee shifts, either for a single employee or for all of them. A timetable of Bookings is the output.

U8.3 **viewNumberOfEmployees** This use case is initiated by an employer user. They request to view the number of employees within their business, and an integer number is the output.

8 Traceability Matrix

The traceability matrix for requirements and use cases is presented in the following page. From this table we can see that all the requirements were satisfied and all the use cases have been mapped to the requirements.

		Requirements	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
		U1.1	X									
		U1.2	X									
		U1.3	X									
		U1.4	X									
		U1.5	X									
		U1.6	X									
		U2.1		X		X	X					
		U2.2		X								
		U2.3		X								
		U3.1						X				
		U3.2						X				
		U3.3						X				
		U3.4						X				
Use Cases		U4.1			X							
		U4.2			X							
		U4.3			X							
		U5.1										X
		U5.2										X
		U5.3										X
		U5.4										X
		U5.5										X
		U5.6										X
		U5.7							X			
		U5.8										X
		U5.9										X
		U5.10										X
		U5.11										X
		U5.12									X	
		U5.13								X		
		U6.1								X		
		U6.2								X		
		U6.3								X		
		U7.1									X	
		U7.2									X	
		U7.3									X	
		U8.1										X
		U8.2										X
		U8.3										X

Table 1: Requirement traceability matrix for Coviduous

9 Quality Requirements

- **Performance** - The Coviduous system needs to be accurate in response to the user's interaction with the application as tasks such as creating an office floor plan, viewing an office floor plan, and processing of user details in the registration process are all paramount in functioning accurately and correctly to avoid incorrect capacity management and assignments/bookings of office spaces being made. User experience metrics to consider: after a delay of one second the user is consciously aware of the delay but will still focus on the task at hand; after a delay of 3 seconds about 40% of users will close the app or website; when a delay of 10 seconds is reached a user's attention is all but lost and will abandon the website. The platform should thus aim to provide: a maximum delay of 3 to 5 seconds for logging in with the assistance of a loading animation; a maximum delay of 0.1 seconds for all text-based data; a maximum delay of 1 to 2 seconds for all non-text based data such as images. In addition, depending on a user's connection quality, lower resolution images may be sent to improve user experience.

- **Scalability** - Performance and scalability goes hand in hand. A system is all but useless if it can offer fantastic performance to only a small handful of users. Scalability allows the platform to meet and even exceed the targeted performance metrics, independent of the number of users currently using the system. Demand of a website or app fluctuates throughout the day and it can be especially prevalent at certain times of the year as with Coviduous when new employees are hired and current employees returning to work at the start of every year, the application will be in high demand. The system should be able to dynamically scale itself in response to demand throughout the day and the year. It should be able to scale up when demand is high and scale down again when demand has decreased so as not to waste unnecessary resources and the client's money.

- **Reliability** - We define the reliability of our system as how well it can consistently perform requested operations without failure. To be able to quantify the probability of a system failure we first need to specify reasons as to why the system might fail. The primary reason for system failure is poor development practices. The system should be thoroughly tested and put under pressure before it is made available to the public. The system should be reliable in ensuring the storage of data that it receives from user input after they have logged into their respective accounts. It should enable users to log back in with their created user profiles having all their user settings, assigned and booked office space information saved and available to review.

- **Security** – The main core quality requirement of the system. The Coviduous application should separate admin rights and user rights to prevent unauthorized users from making unnecessary and unintentional changes to the functionality of the system. Sensitive user data like passwords will make use of a hashing function. This will ensure that even if the hashed password is leaked it will remain useless to an attacker. Furthermore, a user will be prompted to create a strong password including special characters to guarantee a unique hash so that it can not be reversed by using a rainbow table.

- **Portability and Compatability** - Functionality and operation on web browsers and mobile

smartphones should be catered for. The CrowdBook application should support both iOS and Android mobile operating systems as well as operate on a variety of common web browsers. The website will be mobile responsive for users that do not have access to a laptop or desktop PC. With iOS there are far less legacy versions in the wild today as iOS users are eager to upgrade and most older iPhones are still supported. Supported iOS versions will thus be the latest major iOS release including the two previous releases.

- **Maintainability** - Weekly maintenance checks and updates of the application server and database should be made to ensure all systems continue to run well and efficiently as intended and to pick up any problems that might have risen. Creation, updating, and verification of data backups as well as server software updates should all be done during this process. Due to the component-based design Coviduous uses, the system's modules should have the ability to be modified, improved, and tested outside the system before being inserted or reintroduced again without disturbing the system itself or causing long down times. The maintainability of the Coviduous platform is thus expected to be very high. We should aim to have a maintainability of at least 80% in 24 hours. This is compounded by the fact that the application is able to operate with the current identified problem (if it is not too severe) while the residing problem's module is being fixed and tested outside the system all without having to take the Coviduous application offline.

- **Auditability/Monitorability** - Regular and continuous observations of the system should be enforced to ensure consistent operation and functionality, as well as to detect any defects and faults that may be harming the performance of the application. Techniques such as the use of error logs should be applied to keep track of performance and easily identify problems within the system. Error report sheets should be logged by the system per day and evaluated weekly to monitor system performance levels and average values of how many errors the system encountered over the past week should be evaluated and cross checked with user complaints to determine whether the system is performing at the intended level it is required to while providing user satisfaction at the same time.

- **Usability** - The system should cater for the best user experience possible such that even the ordinary user, not just employees working in an office alone, are all able to understand and use Coviduous. Placement of buttons and instructions that are simple and easy to follow should be integrated into the design of the application. Buttons, images, and text should be made big enough to increase clarity, improve user comfortability, decrease user frustration, and minimize room for selection errors. Run weekly usability tests a month prior to the release of the application and before release of any updates to receive user feedback, identify missed errors and test proper workings of the application.

- **Availability** - The system and all its modules should be accessible to any user making a request the vast majority of the time. The system cannot afford to be unavailable for long periods of time especially during peak demand times during general office working hours as it will cost the client money and trust. The system will require maintenance from time to time, but this maintenance can be scheduled when the system will be barely used like late at night. The

Coviduous system should aim to have an availability of 98% per month.