FIT 3077

Assignment 2 Design Rationale

Lab 4 Team 9

William Ho (32579756)
Rebecca Nga (30720591)

Task 2: Design Rationale

1. Login subsystem

The login subsystem acts as an intermediary between the web service and the actual user of the application. The login subsystem takes in a username and password in the form of a string. As this criteria is the first criteria of the assignment, we designed the program to have a LoginController class under the ViewController package. The LoginWeb class is created to act as a validation class to check whether the user matches the user endpoint provided, the LoginWeb class then also inherits the RESTApi class where it retrieves the information from the web service and this class would not be extensively called, only when necessary as we are trying to avoid unnecessary calls to the web service. The LoginWeb class has a login method that retrieves the jwt token and validates it with the get and post methods. Then the loadUser() method will provide the roles for the user.

There are 4 main roles for users, customers, administrators/receptionists, administers that conduct the test on patients and patients. Hence, an abstract User class is created under the User package. This abstract class is then further extended into Admin, Customer, HealthCareWorker concrete classes where these classes each represent a different role in the COVID Booking & Testing System process/procedure. A UserManager class is also created and has an association relationship with the User abstract class. This design rationale for making multiple classes to represent different users is to follow the Single Responsibility Principle (SRP) where each of the roles for users only has one responsibility and one “role”. This is also for future extensibility purposes where the roles of users may increase or decrease, this also follows the Open/Closed Principle (OCP) when making the User class an abstract class. In the future, assuming there may be an increase in roles to be added such as users with disabilities or special needs can be put into consideration, or the User can have multiple roles such as being an Admin as well as a Customer and the new classes added can inherit the abstract class and have its own class. This also avoids complications in the future and maintains a tidy order within the project/application.

2. Search for testing sites

For the searching on testing site features, the main users that would be using this application would be the customers/patients. The LoginController then calls the MainMenuController class in the same ViewController package and has an association relationship with the MainMenuController. This is because the MainMenuController acts as a window to allow the user to search for testing sites. Once the User chooses to search for testing sites to check whether there are any around them, the TestingSiteController class under the same package is called. The user would be able to find testing sites by suburbs with the findSuburb() method created. This provides a way for the user to search for testing sites in a quicker and more convenient method. The MainMenuController has an association relationship with the TestingSiteController. This is because the TestingSiteController class is in charge of retrieving the list of testing sites from the web services or the TestingSiteWeb class. This class is designed to simplify the searching process and make it overall more organised and easy to backtrack when needed. Under the WebController package which contains the TestingSiteWeb class is created to obtain the testing sites list from the web services. Within the TestingSiteWeb, there is also a findBySuburb() method that allows the users to find testing sites by suburb.

After obtaining the testing site details, the TestingSiteWeb class will then have an association relationship with the TestingSiteManager class which is located in the TestingSite package. The TestingSiteManager class is created to be a manager for the TestingSite class as the TestingSite class contains the attributes and details of the testing site, the manager class acts to sort out and organise the details of the testing site in the form of an ArrayList. This design of the TestingSiteManager/TestingSiteWeb also follows the Adapter Pattern structure to convert the interface of one object to allow the other object to easily understand it. Hence, the TestingSiteManager class has an association relationship with the TestingSite class. This design decision also obeys The Common Closure Principle (CCP) where the classes that are used or reused together are grouped together. To summarise, the design rationale for the functionality of searching for testing sites is that the TestingSiteController class uses the TestingSiteWeb class to get and list the testing sites, and the TestingSiteWeb class has the TestingSiteManager class which has a TestingSite class that helps to store all the testing sites retrieved from API into an ArrayList.

3. On-site Booking

For the On-site booking process, it would involve the main user using the application to be the Administrator/Receptionist. An OnsiteBookController class is created within the ViewController package to handle the on-site booking features. The MainMenuController class has a dependency relationship with the OnsiteBookController class. The reasoning behind this design is that we created classes to each have their own sub-function instead of doing everything inside just one controller. Within the OnsiteBookController class, there is a method with a sub-menu that allows the main users using the application to choose the actions as they wish and the actions conducted will be within the class itself. The example of actions available within this class is for users to place a booking or to check their remaining booking. This design maintains the Open/Closed Principle (OCP) and allows for any future extensions where there may be different actions that the site can have such as booking for more specific actions as times and economy may change, different users with different roles and backgrounds may also require different booking services. The class has a dependency relationship with the OnSiteBookWeb class that inherits the RESTApi class in the WebController package. This is designed so each controller class can have its own web service class to maintain order and organisation. The OnsiteBookWeb class is responsible for retrieving the booking details from the web service. The OnsiteBookController class also validates the authenticity of the booking made by the user by validating the PIN code generated from the application. This is important for the users as many possibilities of events may occur such as poor internet connection, double booking of appointments and more, so to avoid wasted trips to the testing sites, the booking is validated.

4. On-site Testing

For the on-site testing features, the main users these features would be the admins and healthcare workers, this is kept into consideration when designing the features for the on-site testing process as the healthcare workers need to conduct a short interview and suggest suitable tests for the patients/users. With this, an OnsiteTestingController is created and can be accessed from the same MainMenuController from the ViewController package, the MainMenuController would provide options to select the testing procedure. Once the selection is made, a sub-menu is designed within the class so that the admin can perform

more actions such as choosing the type of covid testing for the patient, respectively the PCR Test and the RAT Test options. Once the test is conducted, the result of the test would also be entered into the system through the TestingWeb class in the WebController package and further instructions would be given by then. The OnsiteTestingController has a dependency relationship with the User class, Booking class, TestingWeb class, CovidTest class, HealthCareWorker class and Admin class. This is because this specific functionality is an important part of the application and requires quite a few validations and to be done properly to avoid any circumstances where false results may be entered or false user details are matched with false results. The design of this class is designed in mind to follow the Interface Substitution Principle (ISP) where the methods are created purposefully and with usage to avoid unnecessary dependencies for interfaces and methods that won't be used. Each class that has been created has a meaningful task and follows the Single Responsibility Principle (SRP) where the class only has one responsibility. For future enhancements or when the pandemic stages slowly evolve to an endemic, such organised procedures may be modified to be used and implemented for other needs. The system can also use the Adapter pattern structure to further improve the system.

5. Home Booking Subsystem

For the home-booking system functionality, users are able to opt for home testing instead of onsite testing, however, the user has a precondition that he/she needs a RAT test kit at home, otherwise collected from the testing site. As this is additional functionality, an HomeBookingController is created and can be accessed from the same MainMenuController from the ViewController package, the MainMenuController would provide options to select the home booking procedure. With this additional functionality, the user is required to book the covid testing from the website with the same booking procedure, so a booking id would be required as usual. However, the controller would ask the user whether the user possesses a test kit or not and if the user has the test kit, further instruction would be provided. A HomeBookingWeb class is also created under the WebController package to update the QR code for the user’s booking. The separation of classes is to maintain the individuality of each class, and following the Liskov Substitution Principle (LSP) each class in the WebController package extends the RESTApi class and maintains its own functionality. An AdditionalInfo class is also created under the BookAndTest package that stores the required details of the

home testing procedure under that class, this maintains the Single Responsibility Principle (SRP).

Additional design rationales

1. **Application of Enums in the design**

We also utilised the importance of Enums, there's a package called Enum and within there are classes such as a Capable interface and a Capabilities class that inherits the functionalities of the interface, Enums such as COVIDTestTag and UserTag are also created where the COVIDTestTag is used to identify the choice between PCR and RAT tests and the UserTag is used to identify the users using the application and whether they are customers, health care workers or admins.

2. **Singleton Pattern in design**

Most classes created in our design use the singleton structure where this design pattern is used where a class created has only one instance and a global access point to this instance is created. Classes that use this design pattern are the LoginController, TestingSiteController, OnsiteBookController, OnsiteTestingController and the MainMenuController under the ViewController package.

3. **Facade structure in the design**

The Main class is the driver class of the whole application. It starts off with getting the user details for login. This class provides a simple interface such as a console to the whole application with different subsystem components.

References

Nazar, N. (2021). *Object Oriented Design Principles - I* [PowerPoint slides]. Moodle@MU. https://lms.monash.edu/mod/resource/view.php?id=10074847

Nazar, N. (2021). *Object Oriented Design Principles - II* [PowerPoint slides]. Moodle@MU. https://lms.monash.edu/mod/resource/view.php?id=9880724

Nazar, N. (2021). *Design Patterns- I* [PowerPoint slides]. Moodle@MU. https://lms.monash.edu/mod/resource/view.php?id=9880736

Nazar, N. (2021). *Design Patterns- II* [PowerPoint slides]. Moodle@MU. https://lms.monash.edu/mod/resource/view.php?id=9880751

Nazar, N. (2021). *Design Patterns- III* [PowerPoint slides]. Moodle@MU. https://lms.monash.edu/mod/resource/view.php?id=9880764