FIT 3077

Assignment 3 Design Rationale

Lab 4 Team 9

William Ho (32579756)
Rebecca Nga (30720591)

Design rationale on design principles, patterns, architectural patterns, refactoring, task 1 and task 2 done within Assignment 3.

Design principles

Liskov Substitution Principle (LSP)

Within the Model package, the Web-related classes such as the LoginWeb class, OnsiteBookWeb class, HomeBookingWeb class, TestingWeb class, TestingSiteWeb class, and the AdminPanelWeb class. These classes extend the RestApi class which contains the required web services information. The advantage of applying this principle in our software is that each of the web related classes in the Model package can implement its parent class and override the method based on what the required task is. This can also ensure that the exceptions thrown in the parent class are the same in the child classes. The reason for applying this design principle is that we can ensure that the preconditions applied in the base class are not modified or changed in the subclasses, and vice versa the subclasses shouldn't be weakening the postconditions in the base class.

Open-Closed Principle (OCP)

The AdminPanelWeb class in the Model package uses this principle as previously designed so that the various classes can inherit the RESTApi class within the Model package for future extension purposes. Why the OCP is implemented is due to the requirements of the assignment that we need to add new features into our system to further enhance the system. In this assignment, the OCP is confirmed to be an important design principle to maintain for future extensions of the system.

Single Responsibility Principle (SRP)

The SRP is used in the AdminPanelController within the Controller package as these classes hold one responsibility/task within its class, specifically for the AdminPanelController, this controller class is created to implement its own responsibilities and features instead of using any previous controllers to do the job. The reason for this is also to maintain the organisation of code, it helps when we are testing the code while looking for human errors and it avoids congested network traffic within the classes.

Stable Dependencies Principle (SDP) is used for the Model package as the package does not have a dependency relationship with other packages, in fact, the package is dependent on other packages such as the View and Controller package.

The Acrylic Dependency Principle (ADP) is avoided when the Model View Controller (MVC) architectural pattern is implemented as it avoids having cycles between packages. The reason why we try to avoid having cycles within our classes/packages is that it would be very difficult for us to track the errors within the system if there are any as if most of the packages are dependent on each other, if one class crashes, the system wouldn't be able to run its' basic functionality, such as the previous login system and place booking system.

Design patterns

Facade

The facade structure is used to provide convenient access to a particular part of the subsystem's functionality. As an example, it is used in the LoginController class within the Controller package and the LoginController class within the Controller package will call the MainMenuController which is another additional facade to access all the other controller classes. The client (user) will access the functionalities of the system with the facade structure. The reason why the facade structure is used as we let our system continue to work using the console in Java instead of using another user interface. The structure fits our idea of letting the user access different parts of the system without it being too complicated for the user to use.

Architectural patterns & Refactoring

Model View Controller (MVC)

The architectural pattern used in this assignment is the Model View Controller (MVC). The View, Model, and Controller packages are created to refactor the codes for this assignment. This is following the architectural patterns of the MVC to improve the design and architecture of the previous assignment 2 where the various related classes have been grouped together.

The View package would contain a View class which controls the front end inputs and outputs from the system, in other words, the View component manages all the display of information required. The entire system would be refactored to use this architectural pattern. Whereas the Controller package acts as a connector/bridge for the web classes and the front

end input and outputs where its classes handle the web related retrieval and updates, it would be interpreting the user inputs and passing the inputs into the Model classes to later on used in different functionalities of the system. The Model package then holds all the web related classes which is also managing all the behaviours and information of the sytem/application and later responds to the requests or provides the information depending on the current task. The main reason that we have chosen to use this architecture pattern is because of the nature that the View and the Controller depend on the Model, however, the Model will then depend on neither of these classes, which would help significantly for the future extension and modification of the system with the advantages that in the future extension of the system, it would be easier to make changes and updates considering the Model does not depend on the View class. Hence, when implementing new Views, it would not affect the Model. Besides this, by using the MVC pattern, it would be easier to perform testing for the user interface with the similar explanation that the Model is separated from the View classes.

Task 1: Booking modifications design rationale

For this task, while designing the classes and methods necessary to be used in the system at the design level, we have decided that we can further modify the existing OnsiteBookWeb class in the Model package as well as the OnsiteBookController class in the Controller package. As the task requirement is to allow the residents to log in and modify their bookings, such as to cancel the bookings or check the status of the booking. As it was a requirement to open the class for modification, we have decided to insert new functionality within the relevant classes as the previous methods are also an important part of the system. Even after considering the Open-Closed Principle(OCP) and the Interface Segregation Principle (ISP), we didn't want to have a class that has many unnecessary functions that arent used and redundant since an existing class has existed previously or create an interface for the subclasses to implement it, but we ultimately decided just to add additional functionality to modify the code into the OnsiteBookController and OnsiteBookWeb and maintain the Single Responsibility Principle (SRP).

Task 2: Admin booking interface design rationale

For this requirement, the AdminPanelController within the Controller package and the AdminPanelWeb within the Model package is implemented with consideration to the Open-Closed Principle (OCP) where the classes are implemented as an extension of the system's features instead of modifications within existing classes. The Admin User would

have access to view, delete and modify the booking records. As previously mentioned, the software architectural change used in this assignment is the Model View Controller (MVC), by using this architectural pattern, the proposed architectural pattern supports extensibility as MVC is beneficial when testing for the code especially when future functionality needs to be introduced, the classes added to the individual - model, view, controller packages will be beneficial when needing to test the code for bugs and errors as the MVC separated the concern of storing, displaying and updating data into three components that can be tested individually.

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

Nazar, N. (2022). *Software Engineering: Architecture and Design* [PowerPoint slides]. Moodle@MU. https://lms.monash.edu/mod/resource/view.php?id=9880793

Nazar, N. (2022). *Service Oriented Architecture* [PowerPoint slides]. Moodle@MU. https://lms.monash.edu/mod/resource/view.php?id=9880806

Nazar, N. (2022). *Refactoring* [PowerPoint slides]. Moodle@MU. https://lms.monash.edu/mod/resource/view.php?id=9880830