Design Report
MA_Lab03Team13

Members
Dilshan Jayasinghe (29344352)
Edmund Cheong (studentID)

Architectural Pattern Applied

MVC (Model View Controller)

We decided to refactor our previous code to implement MVC , with MVC our model can be built and tested independently which we couldn't do easily for our design in assignment 2 since the model knows nothing about the view but all logic is done by our controller . Our mvc is slightly different from what we have seen in the lecture slide as our view is updated through the data passed by the controller .Besides MVC also helps us to solve cyclic dependency as again the model doesn't need to call view to update itself but leave this task to the controller .

Class Design

Observer design pattern(Extended)

In assignment 2 we also have implemented observer design pattern for offShoreTestingSite We have a class called offShoreTestingSiteDataSource that acts as an observable that updates information about offShoreTestingSite and listens from from web Service. Then we have another class called offShoreTestingSite that acts as an observer. Whenever there is an update(ex: new booking is added, waiting time of the facility changed) from the web service for offShoreTestingSite, offShoreTestingSiteDataSource will notify all its observers so that the observer's data is updated.This design pattern help us to deal with the functionality for admin to view all the booking in run time , beside with observer design pattern we don't need to keep calling a get request every single second even though there are no updates from the web service but only to send a GET request whenever there is an update and it reduces the load to the server. However sometime observer design pattern might over complicated the design as it will introduced classes

Facade (Extended)

In assignment 2, we have applied the facade design pattern to the Testing Site Package to hide the complexity of the subsystem to the client.

We further extended it to other packages and systems, such as the Booking Package.

We extended our facility facade by extracting more

We decided that applying the Facade design pattern to the Login Package was not necessary as it was not a complex system in comparison to Booking and Testing Packages. Simply refactoring (mentioned under Refactoring section) the logic to be responsible under its respective classes, made our design smooth, readable and extensible while adhering to the Single Responsibility Principle and Open Closed Principles.

Advantages of the Facade design pattern

- Using facade, we managed to isolate our code from the complexity of the subsystem. For example, we implemented a facade class for the testing site package called “FacilityFacade” which hides the implementation of the subsystem “OffShoreTestingSiteCollection”, which implements an Observer design pattern.
- The facade pattern creates a balance between Common Closure Principle and Common Reuse Principle.

Disadvantages of the Facade design pattern

- A common issue with this design pattern is that it can become a god object, coupled to all classes of the system. However, in order to avoid this, each package that needs isolation from the complexity of its subsystem, has been implemented with the Facade design pattern (eg, Booking Package). This ensures that the Single Responsibility Principle is not broken.

Memento (New)

We applied a memento design pattern to keep track of the previous booking state . Memento design patterns allow us to save the previous state of a booking and restore the previous state of that booking. Applied memento design patterns will make us create more classes however it will allow us to restore easily where discard the unessential changes .Another disadvantage of memento is let’s say in the future we will allow user to revert back to all the previous booking they want , then we will have to save all the previous booking’s state and it will consume a lot of memory.

Advantages of the Memento design pattern

- You can produce snapshots of the object’s state without violating its encapsulation
- You can simplify the originator’s code by letting the caretaker maintain the history of the originator’s state.
- Keeping the saved state external from the key object helps to maintain cohesion,
- Provides easy-to-implement recovery capability.

Disadvantages of the Memento design pattern

- A lot of memory might be used if clients create mementos too often
- Saving and restoring state can be time-consuming

Package Design

Common Reuse Principle (CRP)

Classes that tend to be reused together have been placed in the same package together. This ensures that the classes are inseparable and interdependent. For instance, the

package Testing Site which implements the Observer design pattern have a number of classes that are interdependent inorder to apply the design pattern (Observable, Observer, OffShoreTestingSiteCollection etc). Similarly the Booking Package has been created with classes that only make up the booking features. All its classes are inseparable and interdependent thus adhering to the CRP. In both packages, the set of classes in them make up the component, as they are tightly coupled and heavily depend on each other.

Common-Closure Principle (CCP)

Common-Closure Principle states that the package should not have more than one reason to change. If changes were to happen in an application dependent on a number of packages, ideally one only wants changes to occur in one package rather than in a number of them. To follow this principle, we come back to our Booking and Testing Site Packages, where any modification or extension related to testing sites or bookings, the required changes will only need to be made in either of the related package's classes. This allows room for better maintainability of the system.

Maintaining a good balance between CRP and CCP is difficult as CCP tends to make packages large and CRP tends to make packages small. However, as we apply the Facade design pattern to both of these packages, an added advantage of the Facade pattern is that it automatically provides a good balance between CRP and CCP.

Refactoring

During most of our implementation, we followed the two hat strategy method where we implemented a functionality and then refactored it. However, there were certain code smells that we missed during our refactoring process, which we have implemented and recorded below.

Moved Login implementation to a new class

In our previous iteration we had implemented the login system logic inside the user interface class "LoginPage". This later on proved to be a bad programming practice resulting in difficulty for extensions and modifications in the future.

The code smells that were evident;

1. Long function
2. Feature Envy

We realised that the simple user interface method where you request for user input of username and password and return a correct output along with a display message was responsible for a lot of tasks besides what was mentioned above. (eg, http requests, jwt token verification, user creation). Moreover, feature envy was visible to us when the user

interface was calling methods of other classes such as login, jwtToken verification and user creation)

Thus to rectify the above smells, using the extract functions technique, we created a separate class for login (“LoginSystem”), a separate class for jwt token verification (“JwtToken”) and within LoginSystem class, added methods for login verification and user creation.

This tidied up our code and made it more readable to other users who would further design this system. Modifications and extensions were made easier as each class was responsible for one and only one feature (addhears to the Single Responsibility Principle and Open Closed Principle), thus any required changes would be localised to the class that implemented it.

Beside we also apply Extract method for our three model which is testingSiteDataSourceCollection , OffShoreTestingSiteCollection and HomeBooking Collection where we extract out the searchFacility and searchBooking so that we don't have repeated code and long function .With this replace temp with query is also applied in testingSiteDataSourceCollection setOffShoreTestingDataSource() method where we don’t need to write the logic searchFacility in this method and in will improve our code readability and also remove duplication.