# FIT3077- Software Engineering: Architecture and Design
## Assignment 3 - Extending and Refactoring the Design

## Design Rationale

In this assignment the focus was on extending and refactoring the design from the previous assignment while also keeping in mind some architectural and design patterns as well as design principles. Since we were building off the existing COVID booking and testing system from before, in order to fulfil the new requirements of this assignment, a few changes needed to be made.

Since the system involves users viewing and interacting with the system while there is data fetched and processed in the backend, the 2 architectural patterns that were first considered are: Model View controller (MVC) and Model-View-Viewmodel (MVVM). The difference between them is that for MVC there are 3 logical components, Model View Controller where Controller is the entry point. For MVVM, the 3 components are Model, View and ViewModel where View is the entry point and the ViewModel is responsible for converting and presenting the data objects from the Model. The MVVM architectural pattern for coding is also more event driven than MVC and this better suits our system in accordance with the requirements. The advantages of MVVM are the maintainability of the code, ease of extensibility in the future and more simple to test than MVC.

The advantages of MVC are that development of the system can be faster as each component can be worked on in parallel by individuals. Also there is the ability to have multiple views for a single model, i.e a one to many relationship.

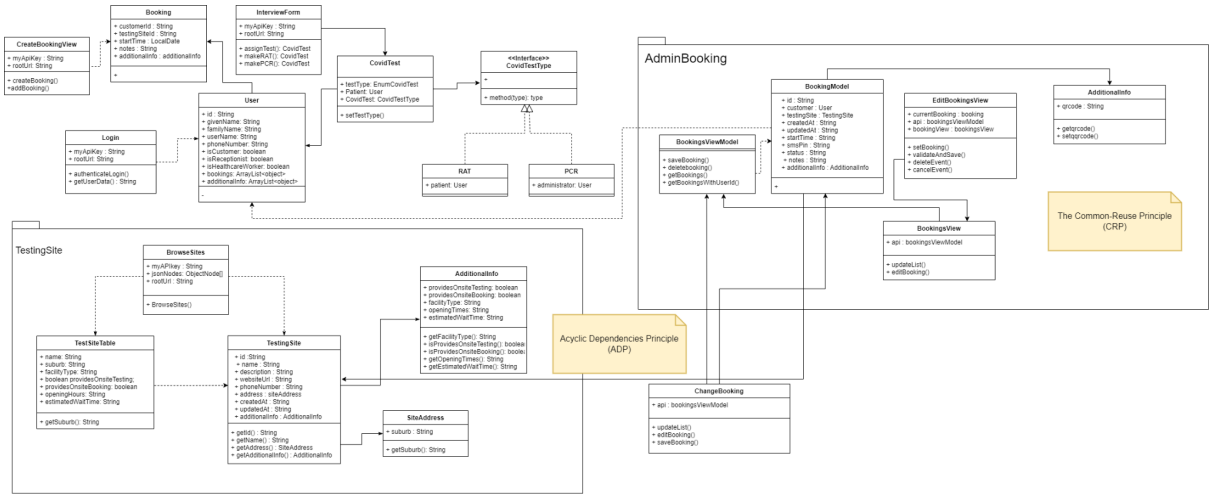
Overall, with respect to the previous system more refactoring would have been required to follow the MVC architectural pattern compared to the MVVM pattern, therefore we implemented the code using the latter pattern. This was the obvious choice considering the advantages were in favour of MVVM.

Refactoring was done to the system when new packages and classes were added to better increase the readability of the file names. Refactoring was also done when files had to be moved into the new packages to better fit the system.

In terms of package-level design principles, these principles below were kept in mind during the implementation process. Since a new package was also added to the system, package level principles were also looked at when developing the system.The Common-Reuse Principle (CRP) was adhered to in our system as the classes in the packages were reused with the other classes in the same package, and not outside of it. This increases the cohesion in the packages. Also since there is more than one package in the system, package coupling principles were also

considered so coupling can be reduced. The Acyclic Dependencies Principle (ADP) was also adhered to since there are no cyclic dependencies between the packages.

## UML diagram




# References


Learn:https://vaadin.com/docs
Software management: https://maven.apache.org/
https://spring.io/projects/spring-boot
https://start.spring.io/
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
Tutorials:
Vaadin login forms
Springboot Vaadin Tutorial