**COMP0004 report**

**Feature overview**

The app homepage allows the option to view the list of all patients, or view statistics about the medical practice and its patients. If you choose to visit the patient list, you can click a tile to view the full profile of a patient. The tiles are sorted into alphabetical order by surname to make it much easier to find a patient. The statistics allow you to view patients by city or age group, and view general medical practice stats such as male:female ratio and ethnicities represented.

There is also a navbar which allows you to quickly access the home page or search for a patient by name. Each page contains hyperlinking to ensure intuitive navigation without the need for using the back and forward arrows.

**Design**

In line with the specification, I built a DataFrame class that stored a list of Columns. In an instance of aggregation, the model accessed by the servlets initialises its own DataFrame with the patient details stored in the CSV file. This is performed with the DataLoader class, and the returned DataFrame is stored inside the Model class. The model itself is constructed and returned by the ModelFactory class. The ModelFactory class stores a path attribute which it passes to the Model to retrieve.

The model accessed by the servlets possesses methods that directly abstract away from the more specific DataFrame methods, such as Model’s getPatientIDs method which uses the DataFrame’s getColumnAsList method to simply return the patient ID column. These helper methods are utilised whenever possible to provide encapsulation.

The web interface utilises JSP files to display the correct content based on the CSV files. The content is provided by the servlet which calls the model using the previously mentioned abstracted methods. The servlets then post these attributes with the correct tag, to the web page. With CSS, I also built these pages with a UI that is friendlier to users. It is now cleaner and more natural to use.

Each class was designed intuitively to rely on the use of “getter” and “setter” methods. I ensured to initialise all instance variables as private so they could not be directly referenced outside of the class. I believe this is good practice as it strictly controls access to the internal state of each object, so data integrity will not be compromised. The “getter” methods also return the data in the required format, so no further manipulation is required by the caller.

**Evaluation**

I feel this program provides a great base interface for viewing patient data and general statistics. The app is easy to navigate and runs quickly, having been tested thoroughly and analysed to improve efficiency. In the future, the app could provide more statistics, and include the ability to view graphs such as the age distribution of the patient base. The ability to add or remove patients is also a good idea as it would make monitoring new users easier.

The design of the class system is excellent for a multi-model system, as the ModelFactory class provides a framework to build new models if required, maybe to cycle through different practices’ patients or view extra details such as appointments or illness history. The Model class itself may even be extended by a new subclass, with specialised methods based on additional CSV files’ contents. Then, another class could be created to aggregate all instances of the model classes and subclasses, with overarching methods to maintain the abstraction.