Design Patterns

In our project there are three main areas of patterns that we are using. In our Java backend you can see a few of these types of patterns. First, we are using a Strategy pattern to determine the type of sort we want to use on our sitter suggestions. There are six possible sorting algorithms that the user can chose from and based on their choice the program responds with that type of algorithm. This is all encapsulated inside of one sorting class. Another method that we are using is a façade pattern. The multiple service classes that we have act as a façade between the data repository and the user API. This is designed like this and we have several functions that act like this. Due to convenience and time not, all functions have been put behind this façade at this point. If we have time I would like to refactor our code to abstract many of the function like this, but the façade pattern has been used for much of the functionality. The main code that I am talking about is I would like the class DAO objects to contain the code to connect to the elastic search database in order to abstract this away from the endpoint controllers. Due to time and a need to ensure that the connection was working properly I we haven’t implemented this yet, but would be a good improvement to make if we have time to refactor. Another pattern we are using in many areas in the singleton pattern. We have singleton versions of each one of our service, DAO and endpoint classes. This is important to ensure that only one class of each is used so that we do not have multiple copies of data as well as confusion behind what instance of a class we are connected with. The main place this is important is we have a singleton for our RestClient connection class to the elastic search database. We need to ensure this is true so that we only open one connection to the database.

Another type of pattern we are using is the architecture pattern MVC for our react and spring connections. The front-end web pages controlled by react are the view of the system they reflect any changes that we make to the model. Spring controls the backend and is our controller in the MVC making changes to both the model and reflecting those changes in the view. The elastic search database represents our model of the data and the overall connections of the system. All information regarding users and their appointments are stored in this database. The backend retrieves this data when called and returns this data to the frontend.

Lastly, we are using several GRASP patterns in our system. We have many instances of a controller class. All our endpoints act as a controller to delegate and ensure that each command is fulfilled correctly. We are also using the information expert pattern. Our services are experts on their specific type of data. They know how to retrieve and return the data required about a specific type of object. Lastly, spring is acting as our creator. It is responsible for creating these objects (many of which are created as singletons) and then passing those objects to the classes that need them.