Doctor Finder
Project Part 4: Final Report
Kaiyue An, Sean Tranchetti, Shubham Mudgal, Tahani Almanie

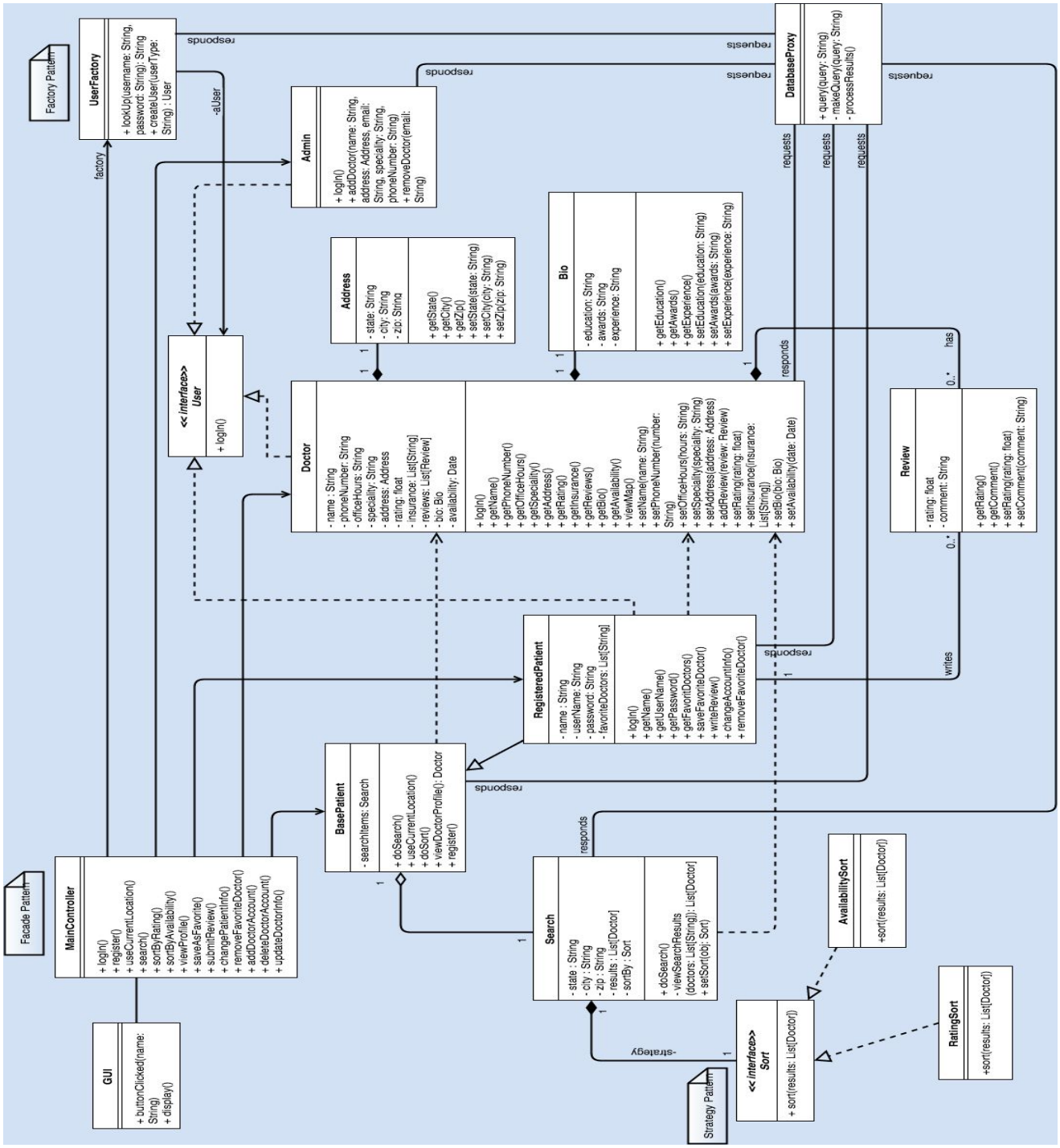
Our project is a website that allows users to search for doctors based on speciality, location, and insurance. The features we successfully implemented include searching for doctors, signing up for an account, logging in, sorting search results based on average doctor rating or nearest availability, viewing doctor profile information, writing a review for a specific doctor, adding doctors to a list of favorites, and allowing users to edit their profile and information (both patients and doctors).

We have successfully implemented all functionalities, mentioned in the Use Case document, in our system:

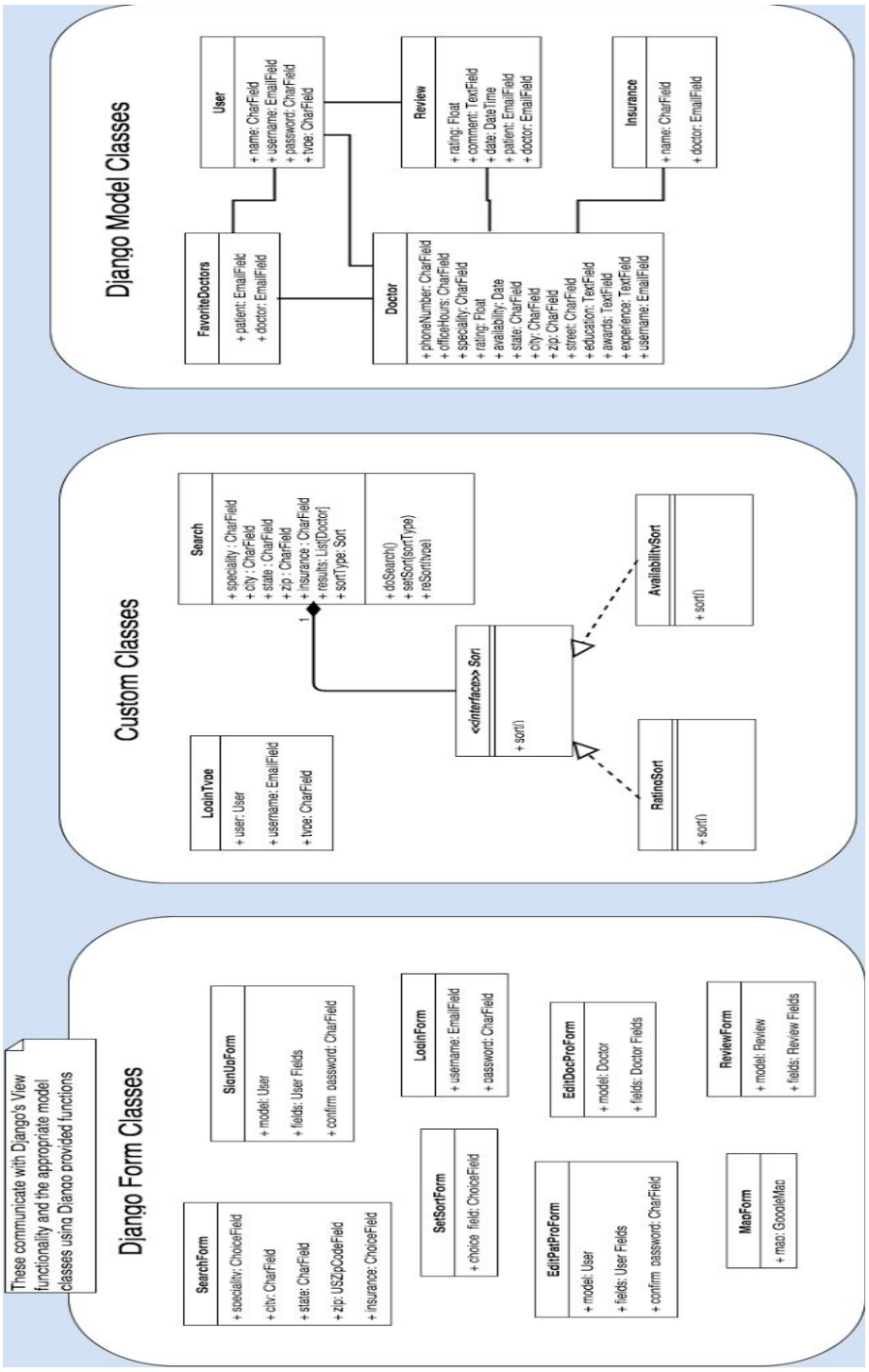
Use Case#	**Use Case Name**
UC-01	Login User
UC-02	Add Doctor's Account
UC-03	Delete Doctor's Account
UC-04	Update Doctor's Info
UC-05	Create Patient's Account
UC-06	Change Patient's Account Info
UC-07	Search for Doctor
UC-08	Sort Results
UC-09	Sort Results by Doctor's Rating
UC-10	Sort Results by Doctor's Availability
UC-11	View Doctor's Info
UC-12	Locate Doctor on Map
UC-13	Review and Rate Doctor
UC-14	Save Doctor as Favorite

The final class diagram for our project can be found below, as well as the original class diagram we created and submitted for part 2.

Original Class Diagram



Final Class Diagram

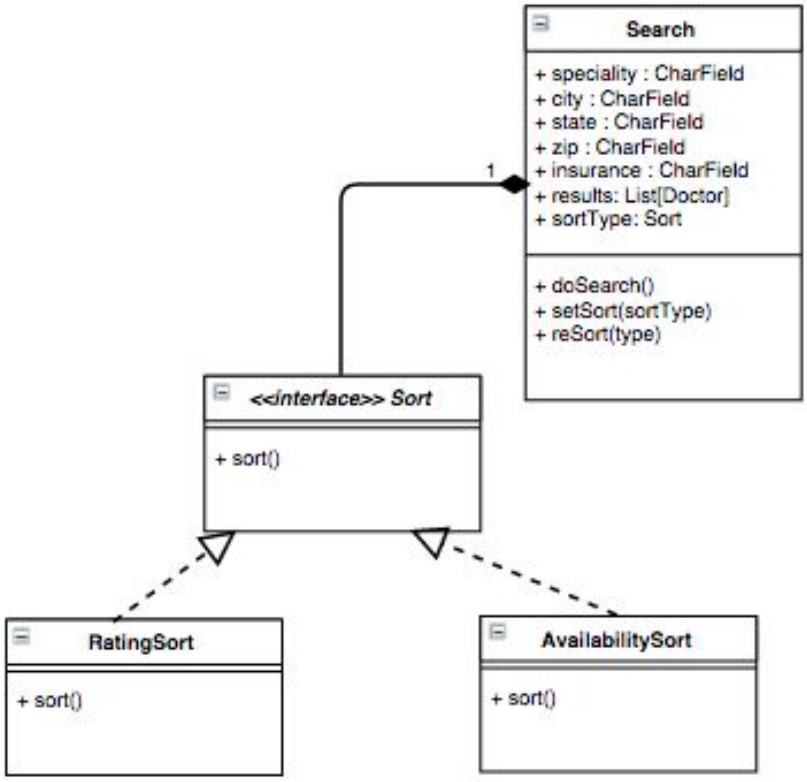


A quick glance at the two diagrams shows that there were obviously a lot of changes made to our original plan. The reason is largely because when we created the initial design, we wanted it to be as general as possible, not taking into account the various features that web frameworks provide, so we didn't lock ourselves into using one specific framework accidentally.

Once we decided that we wanted to use Django to design our final website, we reworked our classes to take advantage of that frameworks specific features. The first point to note is how our various Model classes (User, Doctor, Address, Bio, Review, Base/Registered Patient, and Admin) changed. The Admin class completely disappeared, since Django provides a nice admin interface by default which provided all the necessary functionality we had planned for our admin users to have. The rest slowly changed due to the features that Django provides that we thought would be better to use than coding them ourselves. The Address and Bio classes got absorbed into the Doctor class to make ORM easier. Since each doctor had one and only one entry in the Address and Bio tables, it didn't really make sense from a database standpoint to create separate tables for this information. As such, we placed them into the doctor class. The other classes became little more than data containers from an OO perspective, but that is largely how Django handles creating Models. When defining our models that we want Django to persist in a database, it is the fields that are important. We could specify various getters and setters, but Django provides built-in functionality to handle retrieving this information for us, as well as updating the database when things change. As such, we decided to forgo putting methods into our model classes, and just left them as various data fields for Django to map into tables for us.

Another large difference between the diagrams is the removal of the MainController god-class (which is just good design) and the GUI class. These two were meant to act as our View classes, but were removed because of how the Django Framework handles views. It sets up various functions in different files which get called in a chain to handle building the website page the user is looking at. Since this chain of command is so ingrained in the way Django works, inserting custom classes into is is not exactly an easy or viable task, so we simply adapted and structured our views to match what Django used. Part of that resulted in the creation of the various Form classes that can be seen in the final class diagram to handle building and validating web forms to get and handle user input.

The classes in the Custom Classes box in the final class diagram remained basically unchanged from their original design. The only real change was the addition of the LoginType class. This came about as a replacement for our UserFactory class which disappeared due to Django's way of handling Models. Instead of having to create the appropriate Patient, Doctor, or Admin object and using their implemented functionality, we could simply use Django to figure out what type of user logged in and then use the LoginType class to handle redirecting the user as appropriate.



Since many of our classes design changed, so did our use of design patterns. In our final design, the only one that we implemented according to our original plan was the use of Strategy to handle sorting our doctor results. Since we got rid of our MainController class due to how Django handles views, we no longer had a very good place to implement our Facade pattern. Similarly, we lost the use of the Factory Design pattern when we decided to change how we handled the redirection of users after login to better mesh with using Django's built-in manipulation of Model objects and not giving them methods in their definitions, and thus removed our UserFactory class. In the LoginType Class that took its place, we used a function to handle redirection of the user that has a similar style to a Factory creation method, but it does not actually handle the creation of objects, so it can't be called a true Factory design pattern. Instead, it is more of an adaptation of it to suit our needs.

The project was quite helpful in learning about the process of analysis and design. Since we did all of our diagrams and planning for the system design before we even opened a text editor to code, we were able to get nearly all of our design decisions and other things done beforehand, so the code itself was extremely straightforward and largely pretty simple since we'd figured out most of the hard parts of it already and had a blueprint to follow. The diagrams and associated work proved to be a very helpful and integral part of the whole process of design. They really helped break the whole project into parts that we could work on such as a use case, or a single class. It also made it extremely easy to tell when we had finished implementing a feature or finished the project in general, since we had some sort of hard copy plan written out that defined everything about the project. In short, we learned just how useful really analyzing and thinking about a system before you dive in and start coding can be.