Project : TravelNow

Srishti Rawal

Description: A website to help people find places, mark them as interested, group up with other travelers. It allows users to read description of a place. They will be able to request other users for grouping up.

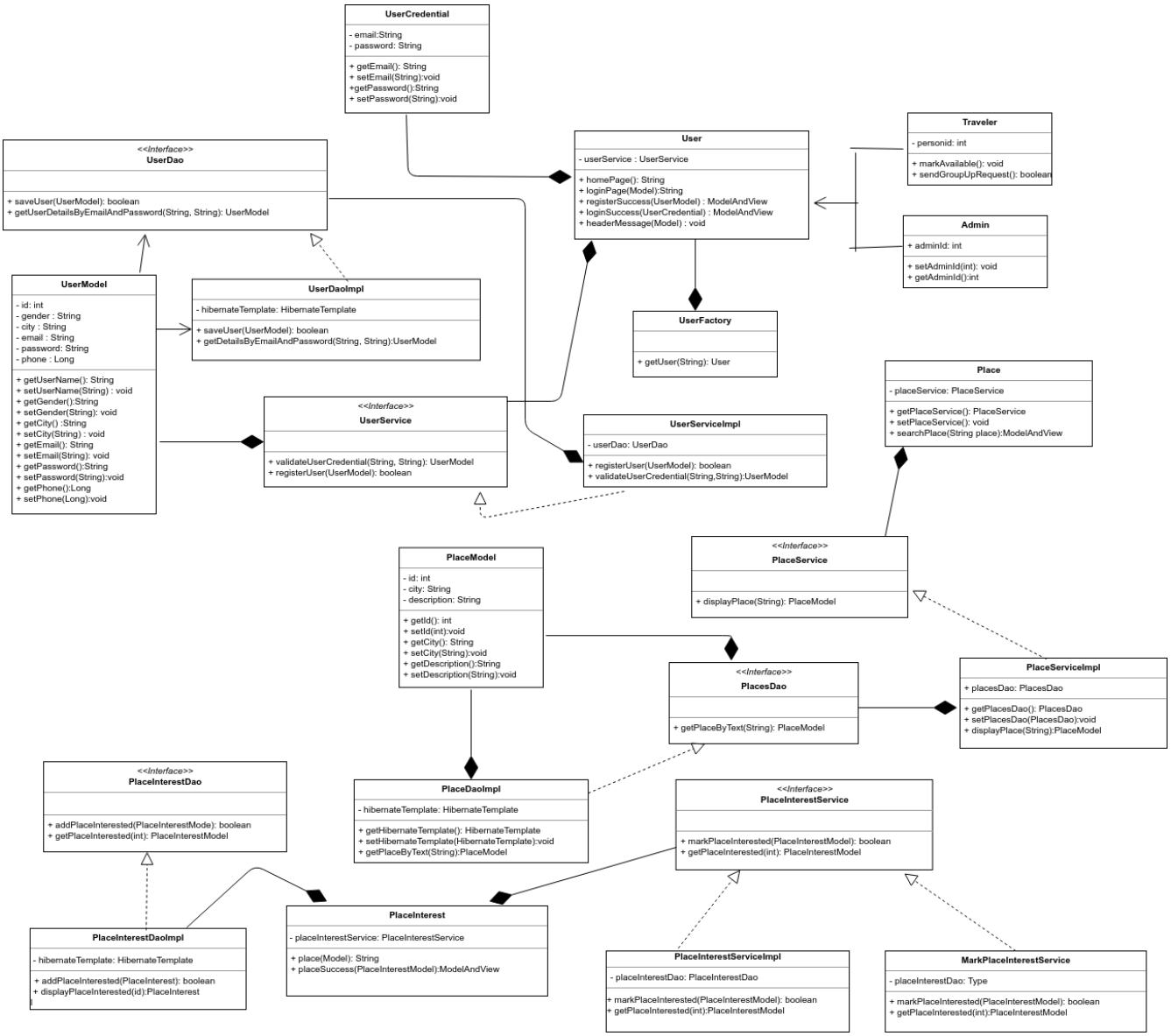
Features Implemented

ID	Feature
1	As a traveler, I should be able to register
2	As a traveler, I should be able to login
3	As an admin, I should be able to login
4	As a traveler, I should be able to search for places
5	As a traveler, I should be able to read about the place
6	As a traveler, I should be able to mark places as interesting
7	As a traveler, I should be able to view history
8	As an admin, I should be able to remove a traveler
9	As an admin, I should be able to search for place

Features not Implemented:

ID	Feature
1	As a traveler, I should be able to plan trip
2	As a traveler, I should be able to add review

Final Class Diagram



What changed in the class diagram?

In the initial design, I had not used MVC and there was no mention of classes like the model, controller , service layer etc as I did not know how to use them. Also, I have used DAO – Data Access Object pattern to communicate with the database. Due to the new design, the number of classes have significantly increased and they are less dependent on each other.

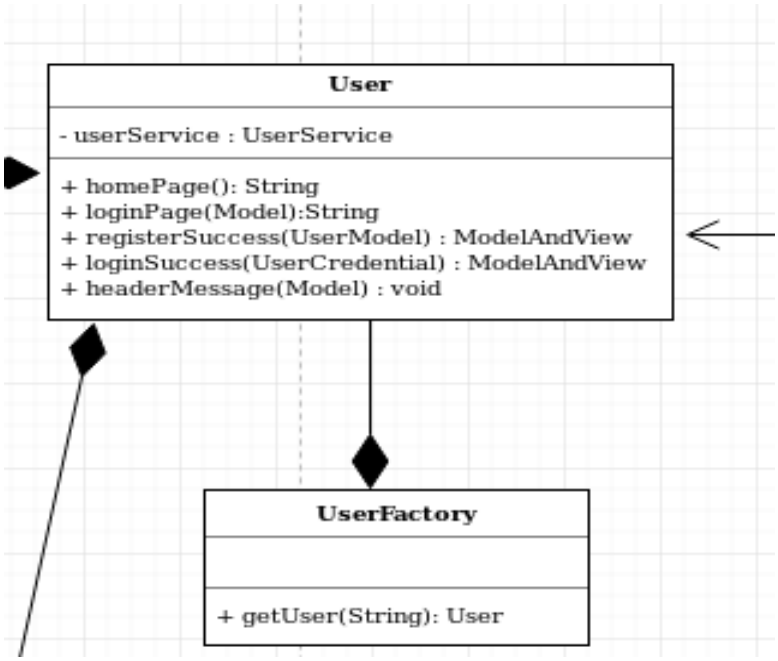
Why did the class diagram change?

The main reason why the class diagram changed would be lack of knowledge while making the first version. Also, I was earlier planning to use google maps API for getting details of places. In the final system, I created a table with required details of the place.

DESIGN PATTERNS

1) Factory Design Pattern

In my design, there are two types of users : Admin and Traveler. I introduced a factory class which controls instantiation of these users.

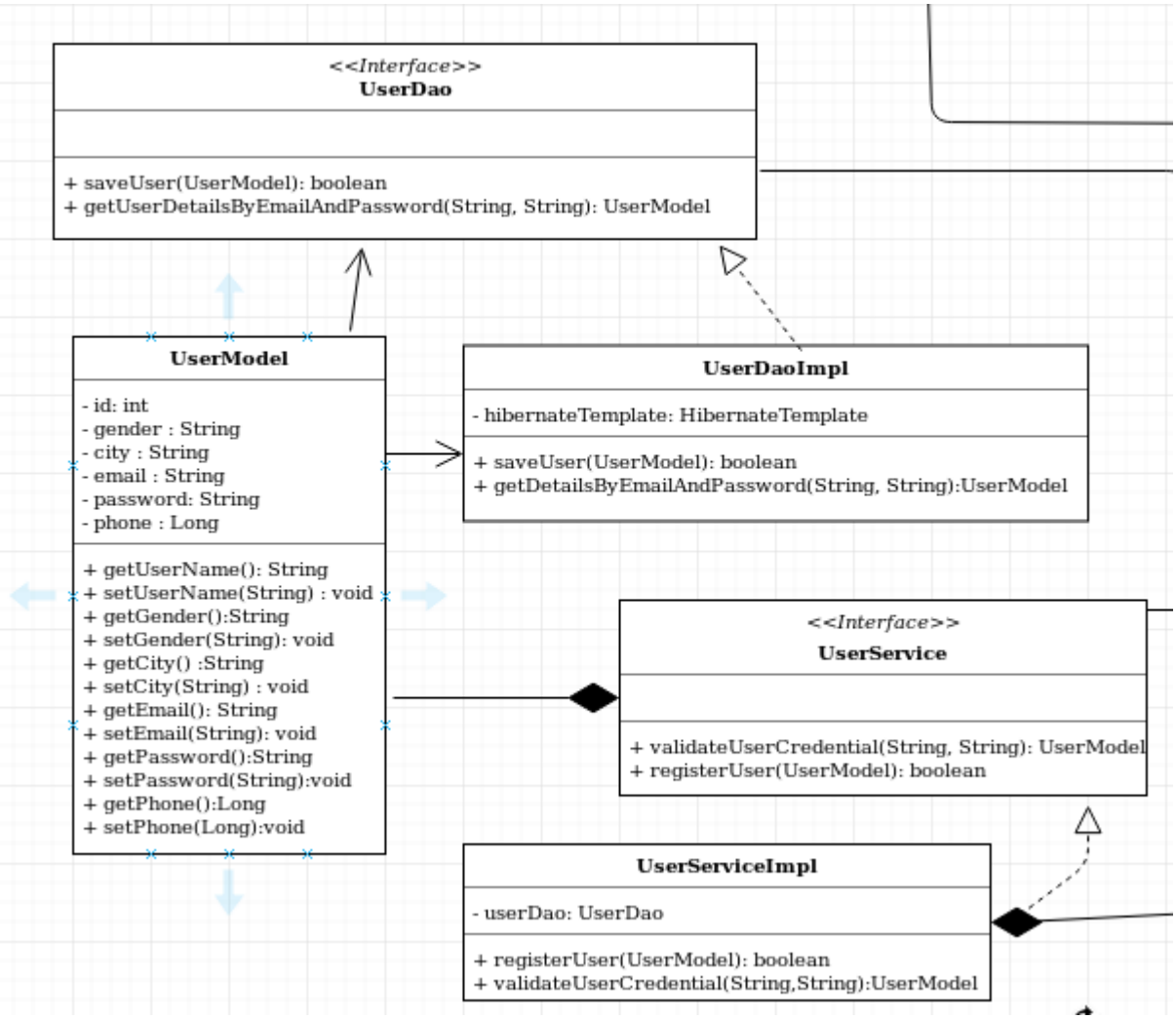


The reason for choosing this design pattern is that in future, I can add new type of users. For example: there can be a user who is just visiting the site and not using it. These

would be the people who have not registered. Code modification will be easier as object creation will be handled by a common interface.

2) **DAO Design pattern**

I have used DAO with 3 components : Model, DAO, DaoImpl. This ensures that out logic is separate and makes components loosely coupled.



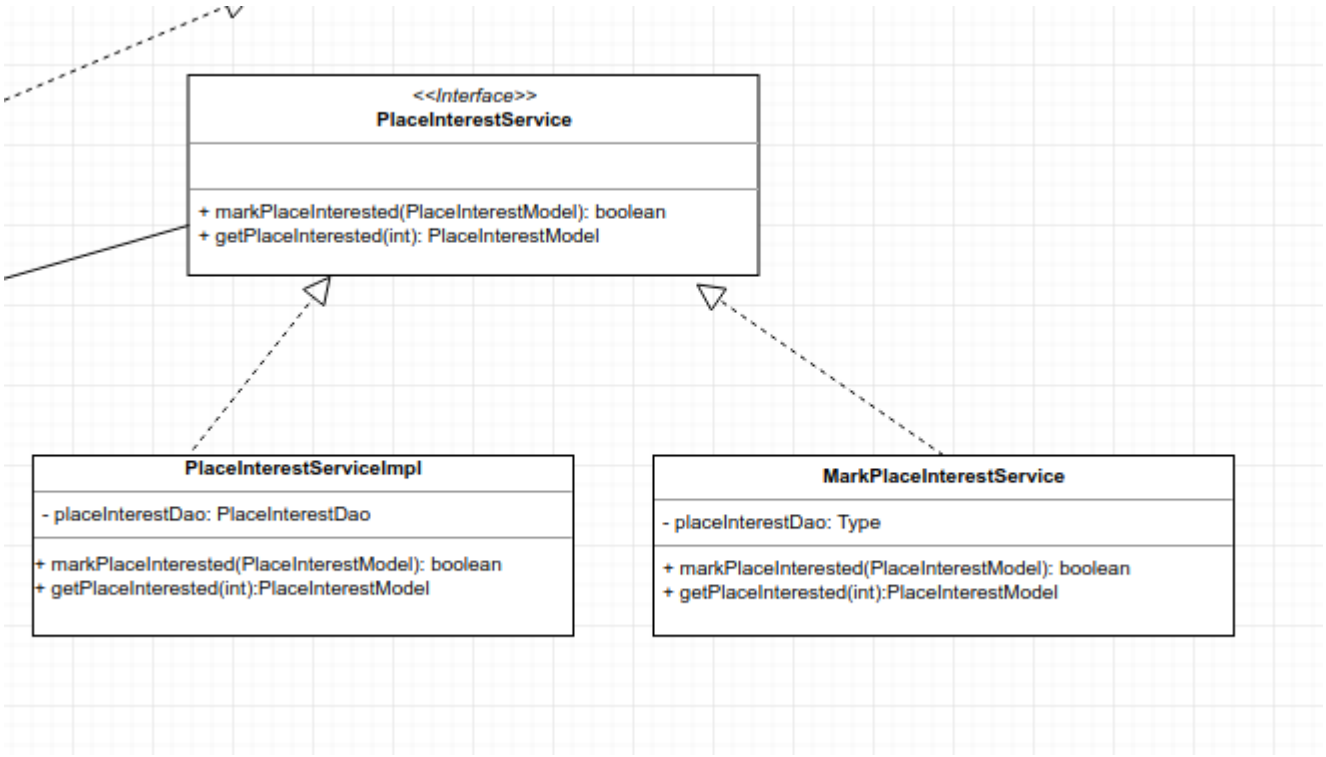
Advantages of this pattern is that the different layers do not need to know how the other is implement. If I wanted to change my database from mysql to nosql, only the DAO layer would need to changed.

As can be seen from the class diagram, I implemented the design pattern using different classes/ interfaces for each layer.

3) **Facade Design Pattern**

I used facade design pattern by creating and interface that would hide the complexity of code. Any class that might need the methods of this interface can implement it and

utilize the methods in it's own way as per requirements. The client can just be given the interface and not the actual details.



The reason for choosing the design pattern is hiding the complexity. If I would want to share my code with client, I would only need to share the interface and not the implementation.

My learning from the project

By completing this project, I learn several important factors that need to be considered while developing a software.
→ It is important to spend time designing the project. Earlier, I used to directly start implementing without thinking much about how the project should be designed. This actually used to take a lot more time.
→ The implementation part becomes much simpler if design is good.
→ I learnt how design pattern can improve the system. For example, by using the DAO pattern, I was able to reduce a lot of dependency between classes.
→ Design patterns make the code open to extension in future without much modification to the code.