Class diagram

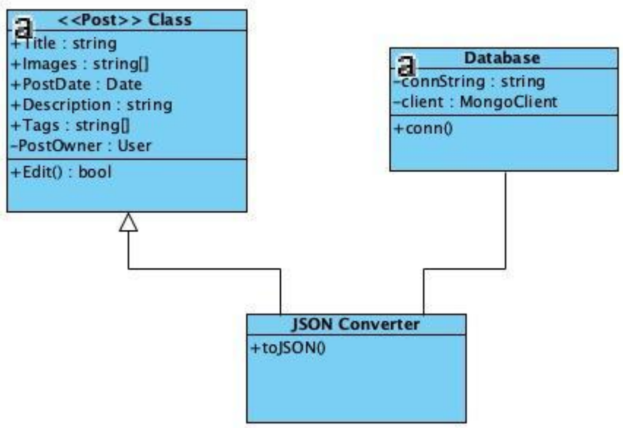
Design Patterns

Adapter pattern

In our website project, we encountered a scenario where we needed to integrate the Post class with a database. The Post class represents posts in the system, and the database is responsible for storing and retrieving posts. Thus, the JSON Converter adapter, database class, and posts class make up the components. In the diagram below, the adapter class (JSON Converter) is represented by a box connected to both the post-class and the database class. This illustrates the use of inheritance, where the adapter inherits from the post.

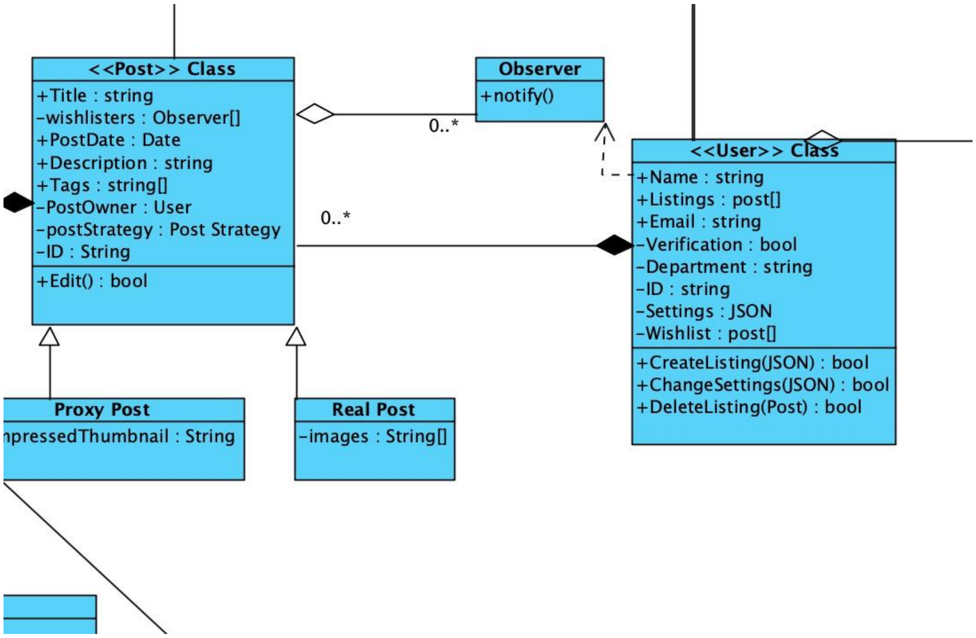
In this particular case, the adapter pattern makes use of delegation to communicate with the database and inheritance for the post-class.

By implementing this adapter pattern with inheritance and delegation, we achieve a cohesive interaction between the post-class, the database, and the JSON converter in our website project.



Observer pattern

In our website project, we have implemented the Observer Pattern to facilitate real-time updates when new posts are added or existing posts are modified. The components involved in this pattern include the Users-class acting as the subject (observeable), and various observers like UserProfile, NotificationSystem, etc.


<<Post>> Class
+Title : string
-wishlisters : Observer[]
+PostDate : Date
+Description : string
+Tags : string[]
-PostOwner : User
-postStrategy : Post Strategy
-ID : String
+Edit() : bool
Observer
+notify()
0..*
0..*
<<User>> Class
+Name : string
+Listings : post[]
+Email : string
-Verification : bool
-Department : string
-ID : string
-Settings : JSON
-Wishlist : post[]
+CreateListing(JSON) : bool
+ChangeSettings(JSON) : bool
+DeleteListing(Post) : bool
Proxy Post
npressedThumbnail : String
Real Post
-images : String[]