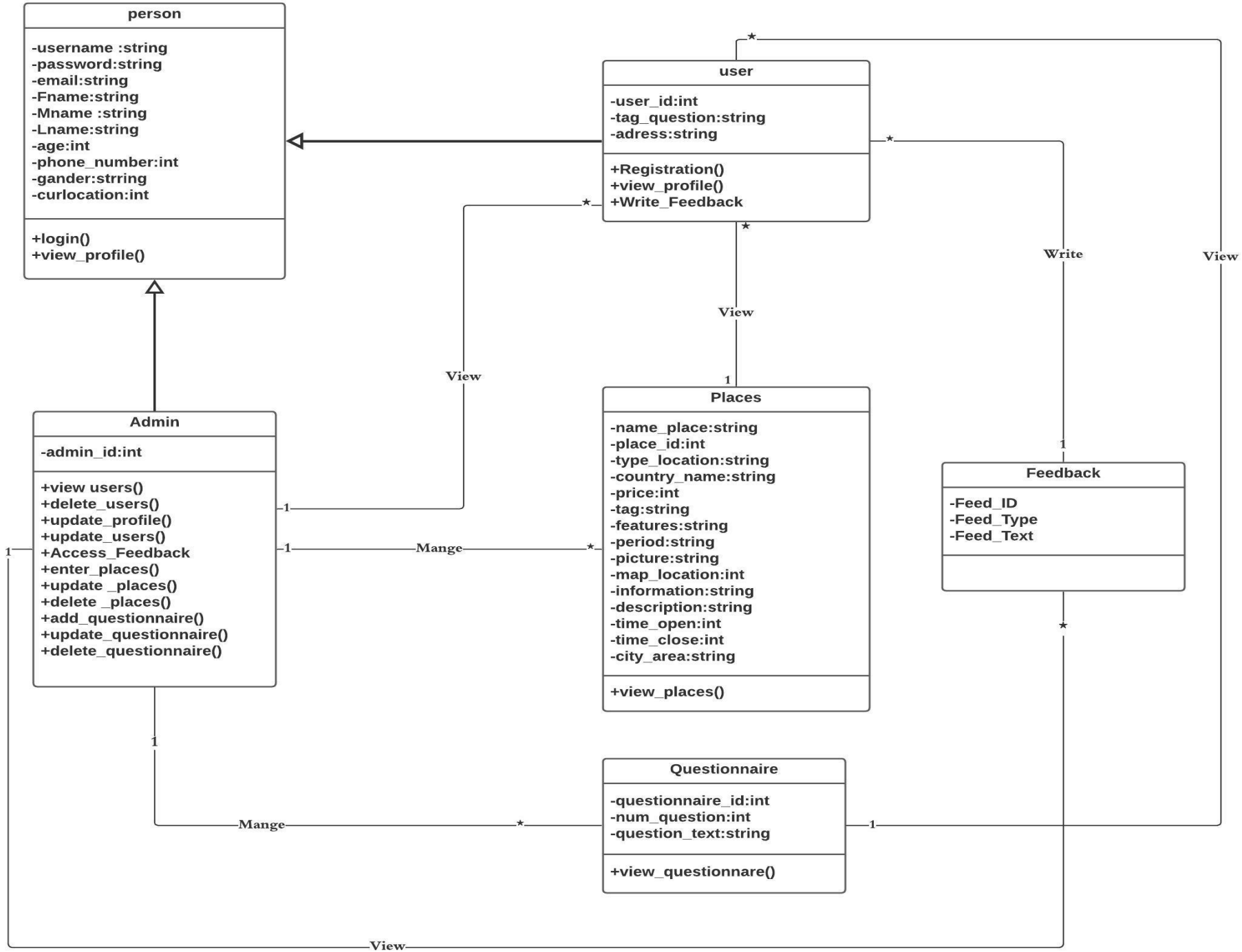
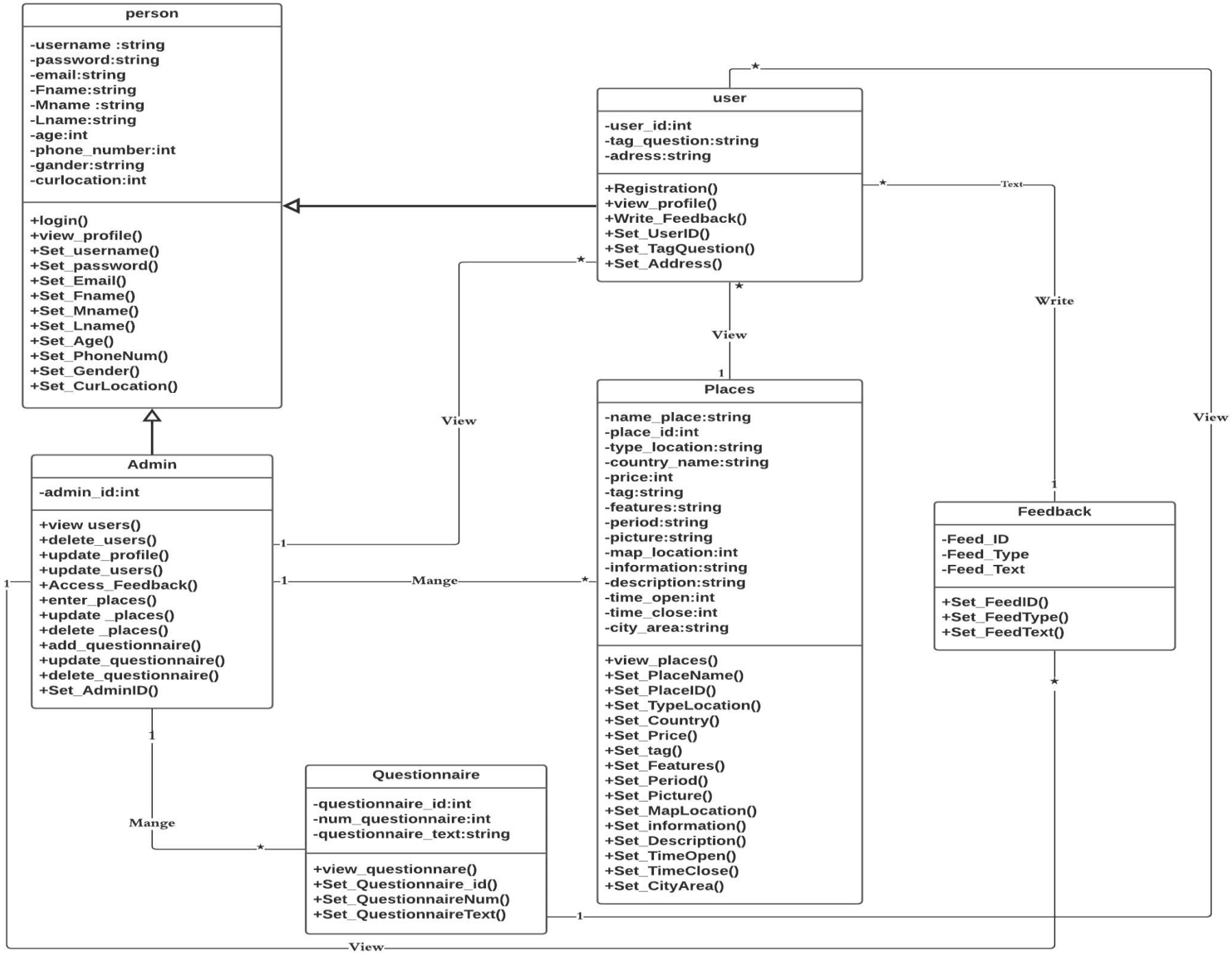
INTELLIGENT TOURIST SYSTEM


9 Class
Diagram

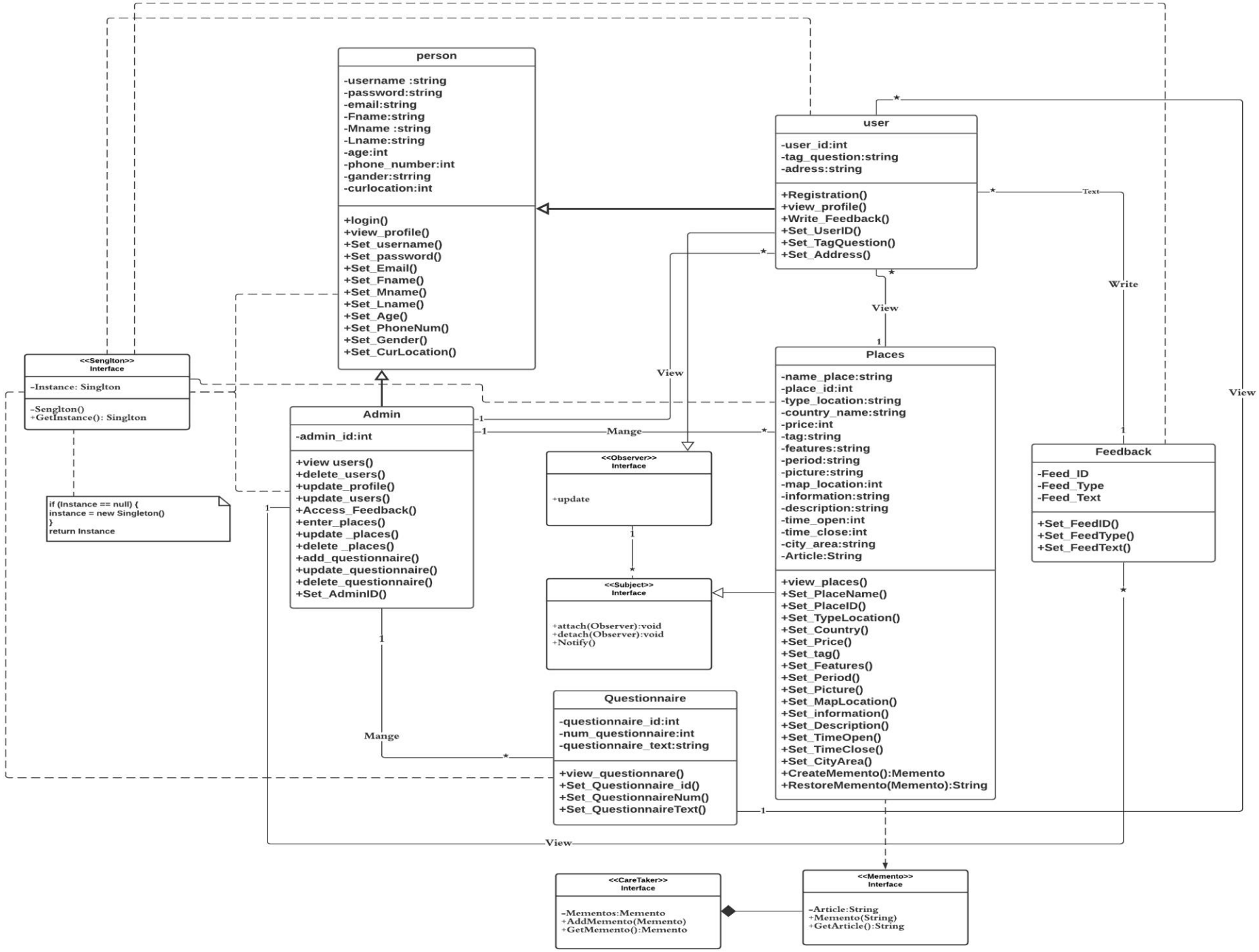
9.1 Class Diagram: Initial Version



9.2 Class Diagram: intermediate Version



9.3 Class Diagram: Final Version



11 Mandatory Design Pattern Applied

11.1 Singleton Pattern:

Type: creational pattern

Intent: singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

Problem:

-Ensure that a class has just a single instance

- Provide a global access point to that instance

Solution:

-Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.

-Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

Participants:

The singleton class must provide a global access point to get the instance of the class.

Singleton pattern is used for logging, drivers objects, caching and thread pool.

Singleton it use to limit the numbers of object instance to one

Consequences:

Controls access:

-The Singleton class can easily control who accesses its single instance and how.

Permits subclassing:

- The Singleton class can be subclassed. Subclasses can return an object that behaves differently than originally intended. This allows client applications to be configured at runtime by selecting a different subclass.

Enhances flexibility:

-An alternative to the Singleton class would be to create a class with only *static members* . Since static members have only one value per class, the immediate effect would be similar. However, pure static members do not lend themselves to subclassing. Furthermore, it is often easier to work with instance methods and fields when the same object must be used in different objects at the same time.

11.2 Immutable Pattern:

Type: Creational Pattern

Content: It's a pattern uses for make object with a constant state, so we can't change its state after creation.

Problem: How can we create a class that it's objects can't change its state after creation.

Solution: Create a constructor for the class that contain the constant values of the object, and create a method that can access these values without changing it.

How it used in the system: We used this design pattern to ignore the users and admin to change their emails after create their accounts

11.3 observer pattern (publish-subscribe):

Type: behavioral pattern

Intent: Observer is a behavioral design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.

Problem:

-A change to one object requires changing the other object and you do not know how many objects need to change.

-An object should be able to notify other objects without making assumptions about the identity of those objects.

- A one-to-many dependency between objects should be defined without making the objects tightly coupled.
- It should be ensured that when one object changes state, an open-ended number of dependent objects are updated automatically.
- It should be possible that one object can notify an open-ended number of other objects.

Solution:

-Define subject and observer objects
- when a subject changes state, all registered observers are notified and updated automatically (and probably asynchronously).

Participants:

***Subject**

-knows its observers. Any number of Observer objects may observe a subject.

-provides an interface for attaching and detaching Observer objects.

***Observer**

-defines an updating interface for objects that should be notified of changes in a subject.

***Concrete Subject**

-stores state of interest to Concrete Observer objects.

-sends a notification to its observers when its state changes.

***Concrete Observer**

-maintains a reference to a Concrete Subject object.

-stores state that should stay consistent with the subject's.

-implements the Observer updating interface to keep its state consistent with the subject's.

Consequences:

-Abstract coupling between subject and observer

-Support for broadcast communication

-Unexpected updates.

11.4 Memento pattern:

Type: Behavioral Pattern

Content: It's a pattern uses to take a capture or a copy of some objects to restore its initial data when its needed.

Problem: How can we make a copy of object's data and restore it to its initial values when it needed.

Solution: The initial state of the object which include some state variables will be stored. To use this pattern, we will create two interfaces, an interface to the caretaker that ignore the access to the stored initial state. The other interface is to the originator, and it allows originator to access the state variables to restore object's initial state.

How it used in the system : We used this design pattern with places function to restore values to the initial values after user used manual search ,so it is used to restore places (as history)