Design Pattern: Observer Pattern
Why did I choose the Observer Pattern?
The Observer Pattern is perfect for real-time communication, where multiple users should receive updates automatically without needing to refresh the page. In Tech Talk Forum (TTF), users need to receive new messages immediately when they are sent in a channel. The Observer Pattern ensures that every user ("Observer") gets a notification when something happens in the chat channels ("Subject").

Improvements:

- Users receive instant updates, which enhances the user experience.
- It's efficient, as the server doesn't need to send updates to all users all the time – only when something happens.
- The structure makes it easy to add more features (e.g., push notifications).

Use Case: Notification of new messages
Preconditions: The user is a member of a chat channel.
Postconditions: All members of the channel receive the new message in real time.
Main flow:

1. A user sends a message in a channel.
2. The message is sent to the server.
3. The server notifies all connected users in the channel.
4. The message is displayed in real time on their screens.

User Story: *"As a user, I want to automatically receive new messages in my channels in real time so I can quickly join discussions without having to refresh the page."*

Design Pattern: Singleton Pattern
Why did I choose the Singleton Pattern?
The Singleton Pattern helps ensure that there is only one instance of the central components of our chat application, such as the server and database connections. This pattern is ideal for a chat application where we want a global control point to handle all users' messages, connections, and potential database interactions.

Improvements:

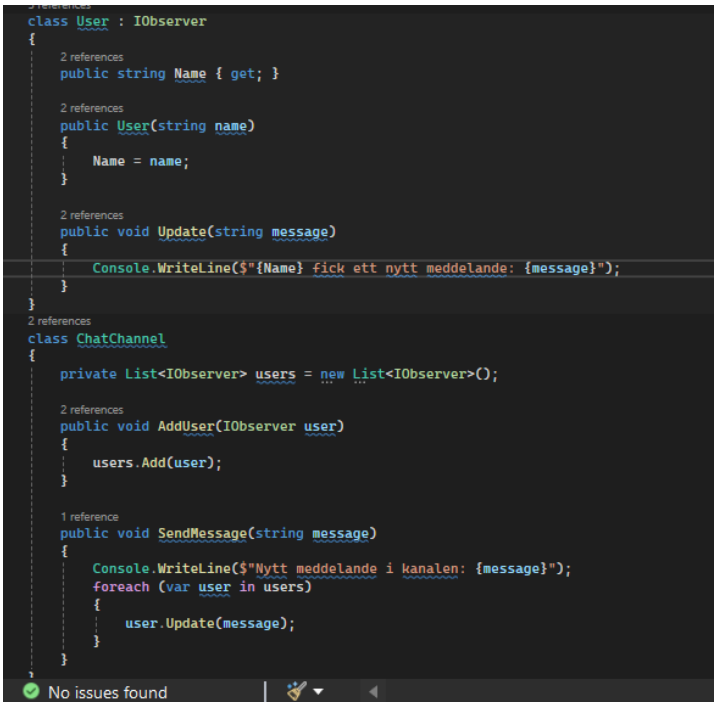
- Easy resource management: We avoid creating multiple instances of the same resource, which saves memory and processing power.
- Centralized management: All messages, user connections, and server functions are handled through a single instance, simplifying the codebase and making it more manageable.
- Performance benefits: Since we use a single instance for the server, we handle all connections and messages efficiently without creating unnecessary connections or instances.

Use Case: Send a message in a group chat
Preconditions: The server instance (which is a Singleton) is already running and managing server communication.
Postconditions: The user sends a message to a group that will be received by all members of the group.
Main flow:

1. The user opens the chat application and connects to a group.
2. When the user writes a message, the application sends it to the singleton server instance.
3. The server instance processes the message and forwards it to all group members.
4. All users in the group receive the message in real time.

User Story: *"As a user, I want to be able to send messages in a group chat so that all group members receive the message in real time."*

Code Example:



```
0 references
class Program
{
    0 references
    static void Main()
    {
        var server = Chatserver.GetInstance();
        ChatChannel channel = new ChatChannel();

        server.AddUser("User1");
        server.AddUser("User2");

        User Maria = new User("Maria");
        User Frank = new User("Frank");

        channel.AddUser(Maria);
        channel.AddUser(Frank);

        server.SendMessage("Hej alla!");
        channel.SendMessage("Välkommen till Tech Talk Forum");

    }
}
```