Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни <<Технології розробки програмного забезпечення>>

Тема <<Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»>>

Виконав:
студент ІА-23
Содолиснький Вадим

Перевірив:
Мягкий М. Ю.

Київ 2024

Тема: Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Хід роботи

Варіант №13

Office communicator

Мережевий комунікатор для офісу повинен нагадувати функціонал програми Skype з можливостями голосового / відео / конференц-зв'язку, відправки текстових повідомлень і файлів (можливо, оффлайн), веденням організованого списку груп / контактів.

Короткий опис патернів

Патерн Abstract Factory

Abstract Factory — це шаблон проектування, який надає інтерфейс для створення сімейств пов'язаних або залежних об'єктів без зазначення їх конкретних класів. Він дозволяє клієнтському коду працювати з об'єктами різних груп, забезпечуючи їхню взаємозамінність і сумісність.

Патерн Factory Method

Factory Method — це патерн, що дозволяє делегувати створення об'єктів підкласам. Замість створення об'єктів безпосередньо, базовий клас визначає метод, який викликається для створення об'єкта, а підкласи реалізують його з потрібною логікою. Factory Method зручний у випадках, коли потрібно створювати об'єкти, тип яких визначається динамічно, наприклад, при роботі з різними форматами файлів або драйверами.

Патерн Memento

Memento — це патерн, який дозволяє зберігати та відновлювати попередній стан об'єкта без порушення його інкапсуляції. Цей шаблон корисний для реалізації функцій скасування або створення контрольних точок у програмах. Об'єкт зберігає свій стан, який може бути переданий іншому об'єкту, наприклад, для зберігання історії змін.

Патерн Observer

Observer — це патерн, який визначає залежність "один-до-багатьох" між об'єктами. Коли стан одного об'єкта змінюється, всі залежні від нього об'єкти сповіщаються та оновлюються автоматично. Цей шаблон часто використовується в системах з підпискою, наприклад, у реалізації нотифікацій, оновлень інтерфейсу або змін у моделі даних.

Патерн Decorator

Decorator — це патерн, що дозволяє динамічно додавати нову поведінку до об'єктів без зміни їхнього коду. Він використовує композицію: оригінальний об'єкт «обгортається» в інший об'єкт, який додає нову функціональність. Цей підхід корисний для створення гнучких систем, де об'єкти потребують різної поведінки залежно від контексту. Наприклад, у графічних інтерфейсах Decorator може додавати до кнопки додатковий ефект, не змінюючи базовий клас кнопки.

Вибір патерну Observer замість Abstract Factory

Вибір патерну Observer замість Abstract Factory пояснюється характером функціональності, яку має забезпечувати система. Office Communicator – це додаток для обміну повідомленнями та спільної роботи, де ключовою задачею є відслідковування змін у станах об’єктів користувачів, оновлення контактів і негайне сповіщення учасників про ці зміни. У такій ситуації головним пріоритетом є забезпечення реактивної поведінки системи, щоб кожен підписаний об’єкт миттєво отримував актуальну інформацію.

Патерн Observer дозволяє створити механізм, при якому об’єкти можуть спостерігати за змінами в інших об’єктах. Цей патерн дозволяє реалізувати асинхронну та автоматичну синхронізацію, що є критично важливим для системи, яка працює в режимі реального часу.

У випадку Abstract Factory, то вона використовується для створення об’єктів із певною категорією залежно від контексту. Це більше підходить для випадків, коли потрібно організувати фабрики для динамічного створення певних типів об’єктів, залежно від конфігурацій або умов. Однак, у випадку Office Communicator основна задача не стосується організації створення об’єктів, а фокусується на обробці динамічних змін та їхньому сповіщенні.

Таким чином, вибір Observer краще відповідає вимогам Office Communicator, оскільки цей патерн забезпечує ефективний механізм відслідковування змін та оновлення стану компонентів, що є ключовою частиною функціональності комунікаційної платформи.

Патерн Наглядач (Observer)

Переваги

На зміну об’єкта реагують всі елементи, які його використовують, що значно зменшує обсяг коду

Observer забезпечує слабке зв’язування між об’єктами, що полегшує розширення системи.

Проблема

Для офісного комунікатора, який працює в реальному часі потрібно реалізувати функціонал, який дозволяє відстежувати зміни об'єктів таких як: чат, учасники групи і тд. Якщо створювати для кожного елемента окремий об’єкт, який часто змінює свій стан, то від цього сильно постраждає продуктивність застосунка.

Рішення

Даний функціонал реалізовано за допомогою патерну Observer та технології SignalR, які відслідковують зміни стану об’єкта.

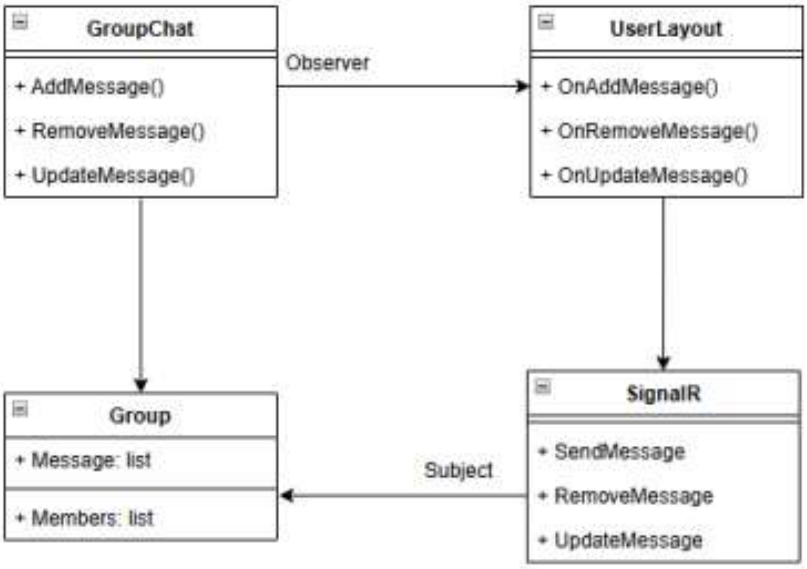


Рис. 1 – Реалізація прослуховування об’єкту Group

Діаграма показує використання патерну Observer у системі групових чатів для роботи з повідомленнями. Основна ідея полягає в тому, що об'єкти, які спостерігають за змінами, автоматично оновлюються, коли джерело змін повідомляє їх про це.

Клас GroupChat є спостерігачем, який відповідає за обробку змін у груповому чаті. Він реагує на додавання, видалення або оновлення повідомлень за допомогою відповідних методів, таких як AddMessage(), RemoveMessage() та UpdateMessage(). Інший спостерігач, UserLayout, також отримує сповіщення про зміни, але його основне завдання — оновлювати інтерфейс користувача. Метод OnAddMessage використовується для відображення нового повідомлення, OnRemoveMessage видаляє повідомлення з інтерфейсу, а OnUpdateMessage оновлює відображувану інформацію.

Клас Group і SignalR є джерелами змін. Group містить списки повідомлень і учасників групи, а також повідомляє спостерігачів про зміни в чаті. SignalR відповідає за передачу змін у реальному часі між клієнтом і сервером. Наприклад, коли нове повідомлення надсилається через метод SendMessage, воно додається до списку повідомлень у Group, і всі спостерігачі отримують повідомлення про це.

```
private async Task DeleteMessage(int messageId)
{
    Message? message = Group.Chat.Messages.FirstOrDefault(m => m.Id == messageId);
    if (message == null) return;
    Group.Chat.Messages.Remove(message);
    await OnDeleteMessage.InvokeAsync(new DocumentMessageChatId(0, messageId, Group.ChatId, message.UniqueIdentifier));

}
```

Рис. 2 – Виклик методи видалення повідомлення

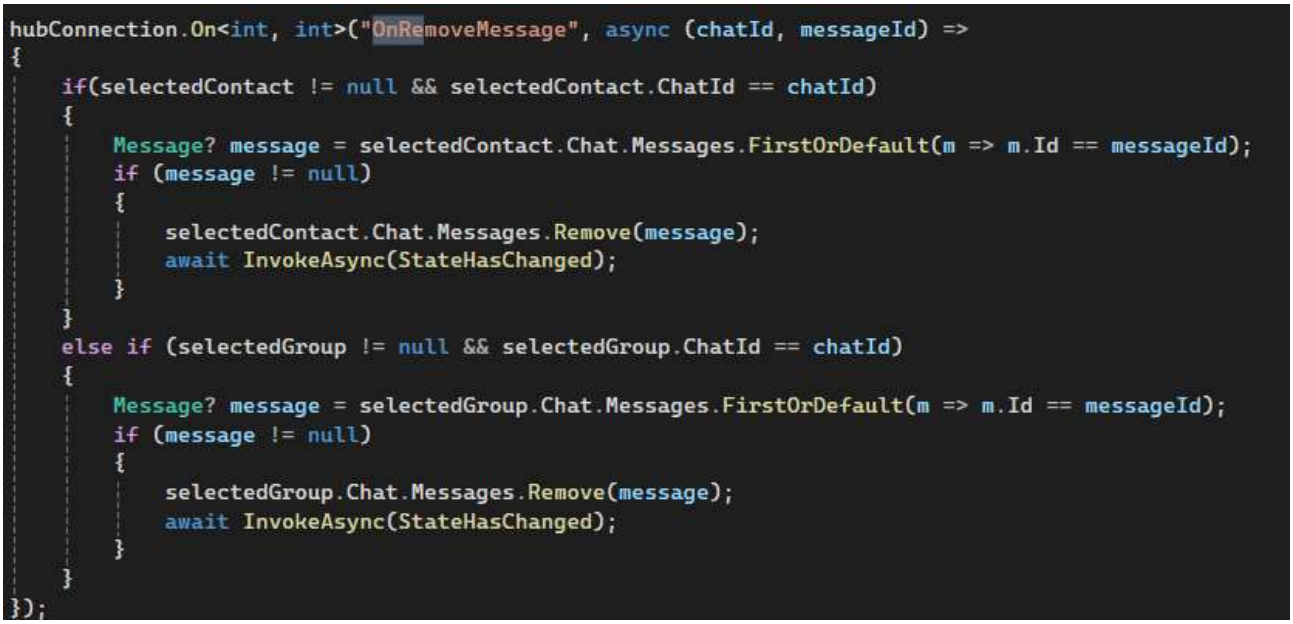
Метод DeleteMessage() відповідає за видалення повідомлення з локального списку повідомлень у межах групового чату. За допомогою LINQ-запиту повідомлення знаходиться у списку повідомлень (Group.Chat.Messages) за його messageId.

Потім викликається подія OnDeleteMessage, яка інформує інших клієнтів через SignalR про видалення повідомлення, передаючи його унікальний ідентифікатор та інформацію про чат.

```
private async Task RemoveMessage(DocumentMessageChatId documentMessageChatId)
{
    if (!string.IsNullOrEmpty(documentMessageChatId.UniqueIdentifier)) await MessageRepository.RemoveMessage(documentMessageChatId.UniqueIdentifier);
    else
    {
        await MessageRepository.RemoveMessage(documentMessageChatId.MessageId);
        var response = await ChatApiService.DeleteMessageAsync(documentMessageChatId.MessageId, token);
        if (!response.IsSuccess)
        {
            errorMessage = "Message wasn't removed";
            isError = true;
            return;
        }
        if (response.IsSuccess) await hubConnection.SendAsync("RemoveMessage", documentMessageChatId.ChatId, documentMessageChatId.MessageId);
    }
}
```

Рис. 3 – Видалення повідомлення

Метод RemoveMessage демонструє взаємодію з сервером для видалення повідомлення. Якщо видалення завершилося успішно, сервер через SignalR розсилає повідомлення про видалення всім підключеним клієнтам за допомогою методу hubConnection.SendAsync().



```
hubConnection.On<int, int>("OnRemoveMessage", async (chatId, messageId) =>
{
    if(selectedContact != null && selectedContact.ChatId == chatId)
    {
        Message? message = selectedContact.Chat.Messages.FirstOrDefault(m => m.Id == messageId);
        if (message != null)
        {
            selectedContact.Chat.Messages.Remove(message);
            await InvokeAsync(StateHasChanged);
        }
    }
    else if (selectedGroup != null && selectedGroup.ChatId == chatId)
    {
        Message? message = selectedGroup.Chat.Messages.FirstOrDefault(m => m.Id == messageId);
        if (message != null)
        {
            selectedGroup.Chat.Messages.Remove(message);
            await InvokeAsync(StateHasChanged);
        }
    }
});
```

Рис. 4 – Прослуховування на видалення повідомлення

Метод OnRemoveMessage є обробником події, який викликається, коли сервер через SignalR повідомляє про видалення повідомлення. Після видалення викликається метод InvokeAsync(StateHasChanged), який оновлює інтерфейс користувача, щоб відобразити актуальний стан чату.

Висновок: я реалізував синхронізоване видалення повідомлень між локальним клієнтом, сервером і іншими користувачами в реальному часі за допомогою SignalR та патерну Observer. Повідомлення видаляється з локального списку, сервер підтверджує видалення, а зміни розсилаються всім підключеним клієнтам. Метод InvokeAsync() гарантує відображення змін у всіх об’єктах.