МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра інформаційних систем

Звіт про проходження виробничої практики

Студента 6 курсу, групи ПМіМ-22с
Кізло Т. М.
Керівник від кафедри
доц. Бернакевич І. Є.

Львів – 2021

Зміст

1. Вступ

На сьогодні важко уявити наше життя без інтернету. Глобальна мережа набула широкої популярності серед мільйонів людей. Зараз багато хто з нас не уявляє життя без цього винаходу.

Якщо проаналізувати розвиток послуг Глобальної мережі, то можна побачити яким чином збільшувались вимоги користувачів. Ми перейшли від сторінок наповнених статичним контентом, до вебсайтів, наповнених зображень, малюнків, аудіо- та відеофайлів тощо.

Для реалізації мережевої взаємодії використовують різноманіття протоколів: HTTP, WebSocket, gRPC тощо. Хоч HTTP протокол являється найпростішим рішенням для реалізації взаємодії між клієнтом та сервером, однак все більшої популярності набуває протокол двонапрямного повнодуплексного зв'язку WebSocket. Він призначений для обміну інформацією між браузером та вебсервером в режимі реального часу.

2. Постановка задачі

Об’єктом дослідження є робота WebSocket протоколу та його використання бібліотекою SignalR.

Метою дослідження є побудова та аналіз взаємодії між сервером та клієнтом.

Завданням практики є реалізації взаємодії між сервером та клієнтом за допомогою SignalR бібліотеки на основі WebSocket протоколу.

Серед основних пунктів реалізації можна виділити наступні:

- реалізація надсилання повідомлень від сервера клієнту;
- реалізація ідемпотентності повідомлень;
- реалізація at-least-once семантики;
- реалізація логіки повторного підключення при розриві з’єднання;
- тестування та виправлення недоліків.

3. Теоретичні відомості

SignalR — це бібліотека, яка спрощує процес обміну даними в реальному часі між клієнтом та сервером.

SignalR надає простий API для створення викликів віддалених процедур (RPC) «сервер-клієнт», які викликають функції JavaScript в браузері з коду .Net на сервері, *рисунок 3.1*:

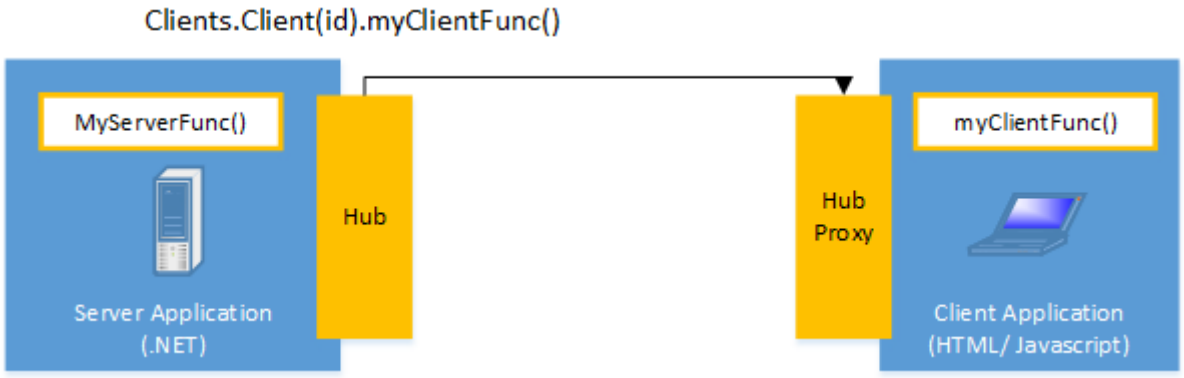

Рисунок 3.1. *Виклик процедури із сервера на клієнті*

Окрім цього, взаємодія «клієнт-сервер» також підтримується.

Варто зазначити, що для транспорту SignalR підтримує 4 типи:

- WebSocket — протокол двоняарямного повнодуплексного зв'язку;
- ServerSentEvents — технологія взаємодії «сервер-клієнт» на основі HTTP протоколу;
- ForeverFrame — підхід, який полягає в створені прихованого IFrame на клієнті для отримання та виконання JavaScript коду;
- LongPolling — підхід, який полягає в опитуванні сервера з певним інтервалом на наявність нових подій за допомогою HTTP запитів.

4. Опис проведеної роботи

Реалізуємо взаємодію між клієнтом та сервером. Для цього необхідно на сервері визначити кінцеву точку взаємодії, *рисунок 4.1:*

```
public class ChatHub : Hub
{
    public async Task SendServer(string message)
    {
        await this.Clients.All.SendAsync("SendClient", message);
    }
}
```

Рисунок 4.1. *Кінцева точка взаємодії на сервері*

Як бачимо на рисунку вище, визначено процедуру *SendServer*, що може бути викликана клієнтом. Своєю чергою сервер викликає процедуру *SendClient* із параметром *message* для усіх підключених клієнтів.

Описати точку взаємодії не достатньо. Необхідно також вказати за яким HTTP шляхом відбувається підключення до неї, *рисунок 4.2:*

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapHub<ChatHub>("/chat");
});
```

Рисунок 4.2. *HTTP шлях підключення до кінцевої точки*

Для того, щоб опрацьовувати та викликати процедури на сервері необхідно встановити з’єднання із кінцевою точкою, та описати метод-обробник, що спрацьовуватиме при надсиланні подій з клієнта на сервер. Важливо, що прототипи методів на клієнті та сервері повинні збігатися, *рисунок 4.3:*

```
const hubConnection = new signalR.HubConnectionBuilder()
    .withUrl("/chat")
    .build();

hubConnection.on("SendClient", function (data) {
    console.log("Виклик процедури на клієнті");
});

document.getElementById("btn").addEventListener("click", function (e) {
    let message = document.getElementById("message").value;

    hubConnection.invoke("SendServer", message);
});

hubConnection.start();
```

Рисунок 4.3. *Встановлення з'єднання та обробка запитів на клієнтів*

При цьому, клієнт отримує повідомлення, лише в момент активного підключення. Якщо з'єднання було не активне, або на стороні клієнта виникли неполадки, що спричинили розрив з'єднання усі повідомлення надіслані сервером клієнту будуть втрачені. Для цього, щоб розв'язувати цю проблему, додамо на сервері додатковий журнал подій, в який будемо записувати підключенні з'єднання, ідентифікатор події, яка відбулась, та які клієнти обробили дане повідомлення, *рисунок 4.4*:

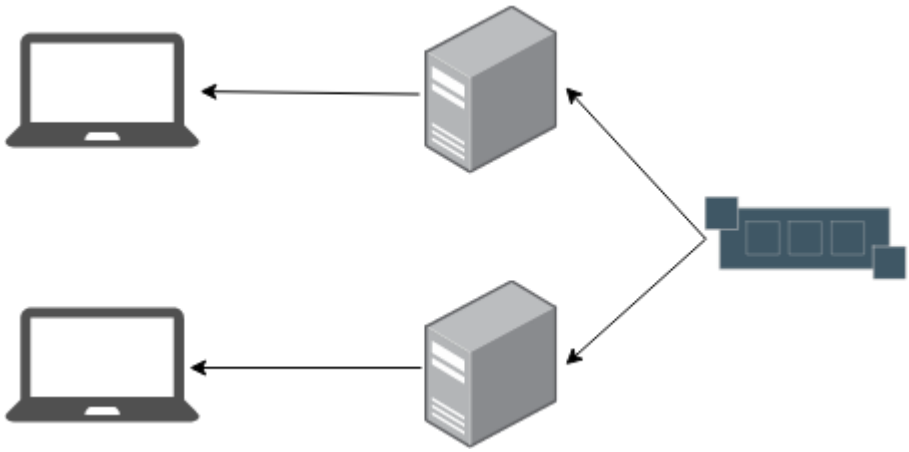

Рисунок 4.4. *Архітектура взаємодії із наявністю журналу подій*

Даний підхід також допоможе розв'язувати проблему, горизонтального масштабування. Коли клієнти підключені до різних серверів, та необхідно сповістити клієнтів підключених до першого сервера про зміни, що відбулись на другому сервері.

Окрім цього можливі випадки коли клієнт отримав повідомлення та під час його обробки відбувся розрив з’єднання. При повторному підключені, клієнт отримує те саме повідомлення повторно, що може призвести до неочікуваних результатів. Щоб цього уникнути, необхідно, або обробка повідомлень володіла властивістю ідемпотентності. Ідемпотентність — це властивість, при якій повторна дія над об’єктом не змінює його стану.

Для того, щоб реалізувати цю властивість створимо структуру даних в яку будемо записувати ідентифікатори оброблених подій. При цьому, необхідно враховувати, що потік подій безперервний, що може спричинити виділення великої кількості ресурсів. Щоб цього уникнути, важливо, аби задана структура із певним проміжком часу очищала ідентифікатори застарілих подій.

Важливо зазначити, що сам алгоритм відновлення з’єднання при його розриві не являється тривіальним, та залежить від багатьох факторів:

- що стало причиною розриву;
- скільки активних з’єднань;
- навантаження сервера.

Наївна реалізація цього алгоритму намагається встановити з’єднання із певним інтервалом. Як наслідок, такий підхід лише більше навантажує сервер DDoS атаками та не дозволяє йому відновити своєї роботи.

Куди кращим рішенням буде додати реалізацію випадкового відхилення періодичності з якою відновлюється з’єднання. Таким чином клієнти будуть намагатись звертатись до сервера у різні проміжки часу, та загальне навантаження буде нижчим

. Висновки

Отже, під час проходження виробничої практики, було досліджено та реалізовану модель взаємодії «сервер-клієнт».

Основними переваги використання даного підходу являється:

- інтерактивна взаємодія. Досвід взаємодії з аплікацією стає приємнішим та природнішим. Пропадає необхідність оновлювати сторінку через те, що клієнт отримує інформацію про зміни на сервері в реальному режимі;
- синхронізація даних між клієнтами. Усі користувачі даної системи мають завжди актуальні дані.

Окрім цього даний підхід володіє і рядом недоліків:

- складність розробки. Реалізація взаємодії «сервер-клієнт» вимагає написання більше коду ніж взаємодія «клієнт-сервер». При цьому варто враховувати багато факторів, таких як масштабування, відновлення з’єднання, тощо;
- складність підтримки. Наразі доступна незначна кількість інструментів для налагодження коду, а серед доступних може не виявитись необхідного.

6. Список використаної літератури

1. WebSocket. — Available from: https://uk.wikipedia.org/wiki/WebSocket
2. Introduction to SignalR. — Available from:
3. https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr
4. How to build a real time notification system using SignalR Core. — Available from: https://medium.com/@NanoBrasca/how-to-build-a-real-timenotificationsystem-using-signalr-core-1fd4160454fa
5. How to get SignalR user connection id out side the hub class. — Available from: https://stackoverflow.com/questions/19447974/how-to-get-signalruserconnection-id-out-side-the-hub-class
6. Scaleout-in-signalr. — Available from: https://docs.microsoft.com/en-us/aspnet/signalr/overview/performance/scaleout-in-signalr
7. WebSockets vs Polling? — Available from: https://stackoverflow.com/questions/44731313/at-what-point-are-websocketsless-efficient-than-polling
8. WebSockets vs. Server-Sent events — Available from: https://stackoverflow.com/questions/5195452/websockets-vs-server-sent-eventseventsource
9. WebSockets vs Long Polling — Available from: https://ably.com/blog/websockets-vs-long-polling