Киселевич Володимир,
здобувач першого (бакалаврського) рівня вищої освіти
фізико-математичного факультету
Яценко Олександр,
асистент кафедри комп'ютерних наук та інформаційних технологій,
Житомирський державний університет імені Івана Франка,
м. Житомир, Україна

ТЕХНОЛОГІЇ З'ЄДНАННЯ В РЕАЛЬНОМУ ЧАСІ З ВИКОРИСТАННЯМ ASP.NET CORE

Інтернет є невід'ємною частиною життя людей в інформаційну епоху, що розвивається з неймовірною швидкістю. Вимоги користувачів змінювалися від потреби доступу до інформації в епоху Web 1.0, взаємодії з інформацією в епоху Web 2.0 і до сучасної взаємодії в реальному часі, яка з'являється у все більшій кількості веб-додатків.

Мета статті – дослідити існуючі підходи реалізації взаємодії клієнта та сервера в реальному часу на платформі ASP.NET Core.

Сучасні веб-додатки потребують сучасних рішень. Для забезпечення функціювання взаємодії в реальному часі необхідна реалізація прямого з'єднання клієнта та сервера. Повсякденний HTTP (Hyper Text Transfer Protocol) (рис. 1), який працює у форматі запит-відповідь, нездатний повноцінно задовільнити дану потребу, оскільки передача інформації між клієнтом і сервером займає певний час. Сервер, отримавши запит клієнта, обробляє його, надсилає відповідь і закриває з'єднання. У цій моделі взаємодії сервери не можуть надсилати дані на клієнт в режимі реального часу.

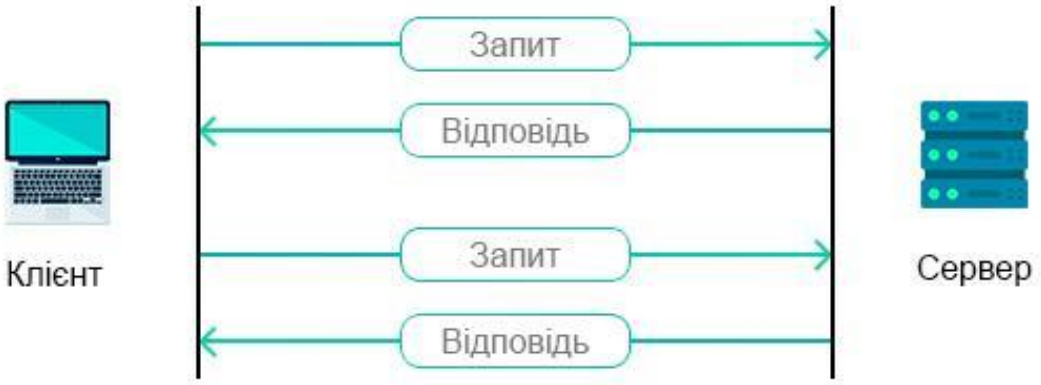

Рис. 1. Схема роботи HTTP між клієнтом та сервером

Polling, Long Polling та HTTP streaming були основними рішеннями, які в минулому веб-розробники використовували для реалізації передачі даних в режимі реального часу між браузером і сервером.

Розглянемо детальніше кожен з цих підходів [1].

Polling. Представляє собою найстаріше рішення для з'єднання в реальному часі. Програмне забезпечення, яке за користувача автоматично оновлює сторінку в браузері. Єдиною перевагою такого підходу є легка реалізація та відсутність додаткових вимог до клієнта і сервера. Однак у цьому рішенні є очевидні й за сьогоднішніми мірками вагомі недоліки:

– дуже важко підібрати частоту оновлення, тому клієнт не може вчасно отримати актуальні дані;

– якщо дані не змінюються, то клієнт все одно буде відправляти запити на сервер, що в свою чергу генерує непотрібний мережевий трафік і спричиняє зайве навантаження на сервер;

– постійне оновлення сторінки є досить незручним для користувача.

Long Polling. Для комунікації сервера та браузера в будь-який момент, було розроблено механізм, в якому клієнт відправляє запит на сервер, що в свою чергу зберігає запит в пам'яті на певний період. Як тільки дані оновлювались, сервер відправляв відповідь з актуальними даними на клієнт. Після отримання даних, клієнт знову робить новий запит і очікує відповіді від сервера. Такий підхід вже більш схожий на з'єднання в реальному часі, але все ще має недолік – велике навантаження на сервер та потреба у великому об'ємі пам'яті, де сервер буде зберігати активні запити.

HTTP Streaming. Для забезпечення HTTP Streaming сервер налаштований на утримання певного запиту від клієнта та зберігання відповіді у відкритому стані, щоб він міг передавати через нього дані. Коли дані на стороні сервера оновились, сервер надсилає відповідь через канал запит-відповідь і закриває з'єднання лише тоді, коли йому явно вказано це зробити. Таким чином, клієнт може прослуховувати оновлення від сервера та миттєво отримувати їх без значних ресурсних витрат, пов'язаних з відкриттям/закриттям з'єднань. Це також усуває потребу в polling server.

З появою HTML5 з'явилася підтримка WebSocket протоколу (рис. 2), який реалізує передачу даних у режимі реального часу та розглядається як найкраще

рішення для вирішення даної проблеми. WebSocket – це передова технологія, яка дає змогу відкривати двосторонній інтерактивний сеанс з'єднання між браузером користувача та сервером. За допомогою цієї технології можна надсилати повідомлення на сервер і отримувати відповіді, керовані подіями, без ініціалізації окремого запиту з клієнта та відповіді сервера [2].

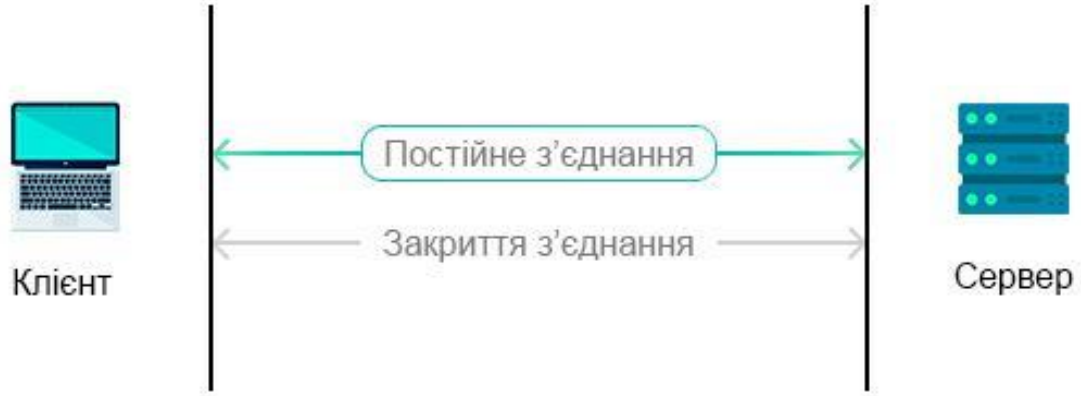

Рис. 2. З'єднання за допомогою WebSocket протоколу

Компанія Microsoft розробила для своєї платформи ASP.NET Core власну бібліотеку SignalR. Дана бібліотека є надбудовою над існуючими підходам реалізації з'єднання в реальному часі, котра автоматично обирає найкращий серед методів транспортування інформації, таких як: WebSockets, Long Polling, HTML streaming тощо [3].

Основні функції бібліотеки:

– автоматично керує підключенням;

– надсилає повідомлення всім підключеним клієнтам одночасно (наприклад, чат);

– надсилає повідомлення певним клієнтам або групам клієнтів;

– масштабується для обробки зростаючого трафіку.

ASP.NET Core SignalR використовує в роботі хаби, що забезпечує двосторонній зв'язок між сервером і клієнтом. Хаби викликають код на стороні клієнта, надсилаючи повідомлення, які містять назву та параметри методу. Клієнт намагається зіставити назву з методом у клієнтському коді. Коли збіг знайдено, він викликає метод і передає йому десеріалізовані дані параметра.

Поглянемо на приклад використання SignalR на стороні сервера та клієнта. Для початку необхідно створити власний хаб клас, в якому описати методи-події (рис. 3). Використовуючи властивість «Clients» класу Hub, ми маємо можливість всім клієнтам або певній групі відправити подію, які в свою чергу її отримають та опрацюють.

```
// Створюємо власний Hub
public class ExampleHub : Hub
{
    // Метод, який буде реагувати на подію "Send"
    public async Task Send(string data)
    {
        // Генеруємо подію Send для всіх клієнтів
        await Clients.All.SendAsync("Send", data);
    }
}
```

Рис. 3. Приклад створення Hub на стороні сервера

На стороні клієнта необхідно встановити пакет «@microsoft/signalr», який знадобиться для встановлення з'єднання. За допомогою HubConnectionBuilder створюємо з'єднання й відкриваємо його (рис. 4). Для підписки на події використовується метод «on», відповідно для генерації події метод «invoke».

```
// Створюємо з'єднання
let connection = new signalR.HubConnectionBuilder()
    .withUrl("/example")
    .build();

// Підписка на подію "Send"
connection.on("Send", data => {
    // Виведення отриманих даних в консоль
    console.log(data);
});

// Розпочинаємо з'єднання.
// Після успішного запуску, генеруємо подію "Send"
connection.start().then(() => connection.invoke("Send", "Hello"));
```

Рис. 4 Приклад роботи SignalR на стороні клієнта

Таким чином, аналіз підходів до реалізації взаємодії клієнта та сервера в режимі реального часу показав, що в певний період розвитку веб-розробки деякі з методів являли собою революційне та актуальне рішення, але на сьогоднішній день абсолютно не прийнятні у серйозних проектах.

Підводячи підсумки дослідження, можна зробити висновок, що існує досить багато як хороших, так і поганих рішень. При роботі з платформою ASP.NET Core варто звернути увагу на бібліотеку SignalR, яка являє собою універсальне та оптимальне рішення для взаємодії в реальному часі, до того ж, робота з якою не потребує значних зусиль зі сторони розробника.

Список використаних джерел та літератури

1. D. G. Synodinos, «HTML 5 Web Sockets vs. Comet and Ajax» 2008. URL: http://www.infoq.com/news/2008/12/websockets-vs-comet-ajax (дата звернення 02.10.2022)

2. Документація WebSockets. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення 02.10.2022)

3. Документація SignalR. URL: https://learn.microsoft.com/en-us/aspnet/signalr/ (дата звернення 01.10.2022)