Пловдивски университет „Паисий Хилендарски”



Факултет по математика и информатика

Курсова работа

По дисциплина „Фреймуърк системи за уеб програмиране“

На тема: „Bull Market“

|  |  |
| --- | --- |
| Изготвили:  Александър Йончев,  Генчо Драганов,  Светозар Митров  Специалност: Софтуерни технологии и дизайн | Проверил:  / гл. ас. д-р В. Кюркчиев / |

Съдържание

[1. Какво е SignalR? 3](#_Toc65759728)

[2. Конфигуриране на SignalR 3](#_Toc65759729)

[1. Конфигурация на сървъра 3](#_Toc65759730)

[2. Конфигурация на клиента 5](#_Toc65759731)

[3. Технологии и подходи, които SignalR използва 6](#_Toc65759732)

[1. WebSocket 6](#_Toc65759733)

[2. Server-Sent Events 7](#_Toc65759734)

[3. Long Polling 9](#_Toc65759735)

[4. Компоненти предоставени от SignalR 10](#_Toc65759736)

[1. Деклариране на събитията на сървъра, за които клиента може да се абонира 10](#_Toc65759737)

[2. Деклариране на входните точки на сървъра, където клиентът може да изпраща данни 11](#_Toc65759738)

[3. Логическо разграничаване на клиентите 12](#_Toc65759739)

[4. Изграждане на връзка от клиентска страна 12](#_Toc65759740)

[5. Абониране на клиента за събития настъпили на сървъра 13](#_Toc65759741)

[6. Изпращане на данни от клиентска страна към сървъра 14](#_Toc65759742)

[5. Функционалност предоставена от Bull Market 14](#_Toc65759743)

[6. Библиография 15](#_Toc65759744)

# Какво е SignalR?

SignalR е open-source библиотека, която опростява имплементирането на комуникация в реално време между сървър и клиент в едно приложение. Библиотеката използва Remote procedure calls (RPC), които се изпращат от сървъра и предизвикват изпълнението на функции, които се намират на клиента.

SignalR има две основни версии – SignalR и SignalR Core, като съответно първата използва .NET Framework и работи единствено върху Windows операционната система, а втората - .NET Core, таргетира и други високопопулярни операционни системи, като macOS и Linux. Технологията позволява на програмистите работещи с езици, които използват .NET платформата да създават приложения използващи двупосочна комуникация на живо.

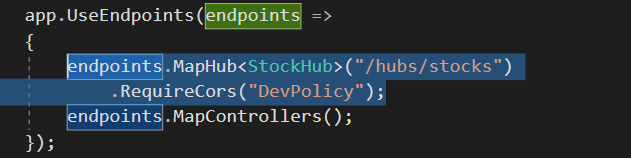
Фреймуъркът предоставя на програмистите една абстракция, която улеснява работата с установяване на връзката между клиент и сървър. SignalR използва различни похвати и технологии за постигане на двупосочна комуникация. Примери за това са WebSocket, Server-Sent Events и Long Polling. Именно в този ред е и приоритета за избор на технология осигуряваща връзката, тоест първо винаги се прави опит за осъществяване на комуникация чрез WebSockets, а като краен вариант се използва Long Polling. Библиотеката позволява на програмистите да се фокусират повече върху изграждането на желаната функционалност, като се грижи за избирането на най-ефективния метод за пренос на данни.

# Конфигуриране на SignalR

## Конфигурация на сървъра

SignalR разполага с две библиотеки – една за сървъра и една за клиента. Библиотеката за сървъра е добавена автоматично при създаването на нов ASP .NET Core 3.1 проект. За конфигурирането на SignalR в един проект за уеб приложения е нужно да добавим няколко команди. За успешното инжектиране в различните класове, където ще ни е нужна комуникация в реално време е нужно да добавим съответния програмен код в ConfigureServices метода на Startup класа, показан на Фигура 1.

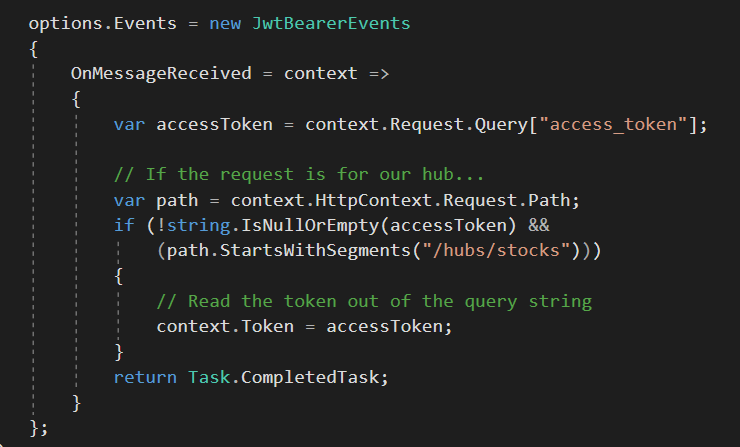
Фигура 1 Конфигурацията нужна за инжектиране на библиотеката в различни класове от приложението

Конфигурирането на входните точки за комуникация със сървъра се случва в Configure метода на същия клас и е представена на Фигура 2.

Фигура 2 Конфигуриране входната точка на комуникацията

Това, което прави съответният код, е да укаже, че **StockHub** класът, трябва да осъществи комуникацията и може да бъде достъпен с url от типа *domain:port/hubs/stocks*. Основно понятие при работата със **SignalR** е така наречения **Hub**, който се явява базов клас, позволяващ осъществяването на този тип комуникация. Горепосочения **StockHub** е клас наследник на **Hub** класа. Също така конфигурацията позволява използването на CORS политики.

При нужда от оторизирана комуникация със сървъра, при която се използва JWT е нужно да добавим и съответния отрязък от код, който инструктира сървъра да вземе съответния тоукън от query параметрите на заявката.



Фигура 3 Конфигурация за оторизиране чрез JWT

## Конфигурация на клиента

От Майкрософт са разработили клиентски библиотеки както за .NET платформата, така и за други програмни езици, като например **JavaScript.** По този начин разработчиците използващи различни библиотеки базирани на **JavaScript** могат много лесно да свържат потребителския интерфейс със сървъра. Клиентската библиотека е нужно да бъде добавена ръчно към клиентското приложение. За създаване на клиентско приложение разработено, чрез **C#** е нужно да добавим пакетът **Microsoft.AspNetCore.SignalR.Client,** чрез **NuGet** пакетния мениджър. От друга страна, за създаване на клиентско приложение с помощта на JavaScript предоставената библиотека от Майкрософт е ***@microsoft/signalr***. Тя може да бъде добавена с **npm**, използвайки:

***npm install @microsoft/signalr@latest***.

Или с **LibMan** използвайки:

***libman install @microsoft/signalr@latest -p unpkg -d wwwroot/js/signalr --files dist/browser/signalr.js --files dist/browser/signalr.min.js***

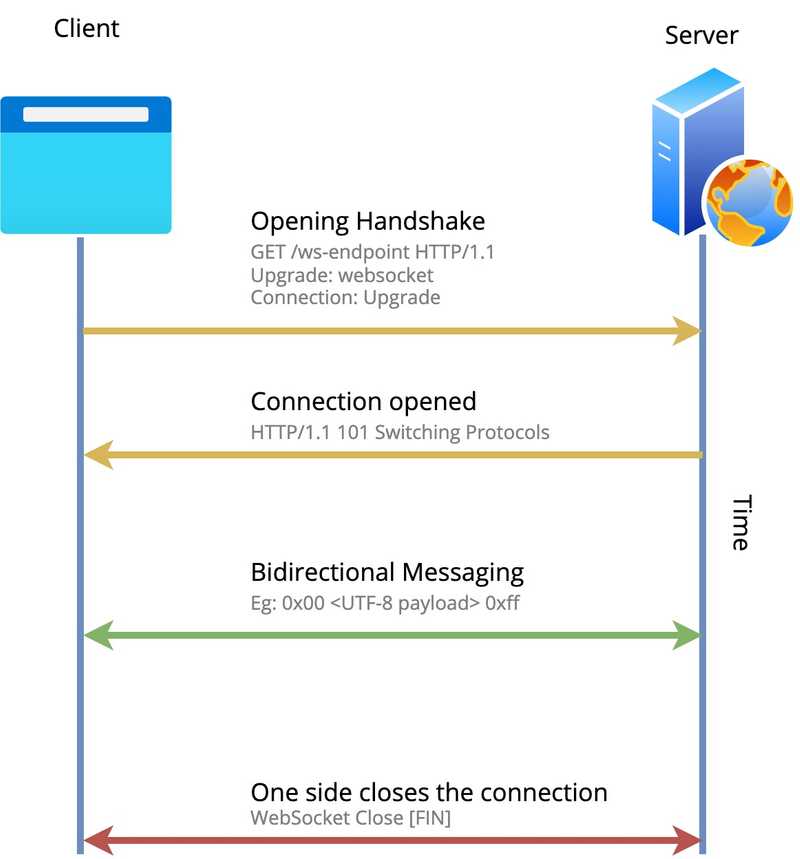
# Технологии и подходи, които SignalR използва

## WebSocket

WebSocket протоколът използва HTTP 1.1 протокола за комуникацията и TCP за преноса на данни. Използва се свойството на HTTP 1.1 за създаване на продължителна връзка (persistent connection) между клиента и сървъра, при което е възможно да с преизползва една връзка за множество заявки.

За установяване на WebSocket връзка, клиентът изпраща заявка за отварящо ръкостискане (opening handshake request), което е HTTP заявка и сървърът отговаря с HTTP 101 Switching Protocols response. Всеки статус код различен от 101 означава, че комуникацията не може да бъде осъществена. След това комуникацията преминава в двупосочен бинарен протокол, което позволява информация да бъде изпращана както от клиента към сървъра така и обратно без установяването на допълнителни връзки. Коя да е от страните може във всеки един момент да затвори връзката чрез изпращане на заявка за затварящо ръкостискане (closing handshake request). Целият процес е изобразен на Фигура 4.

WebSocket протоколът е поддържан от всички модерни уеб браузъри и именно затова е предпочитания подход за осъществяване на подобен тип комуникация.



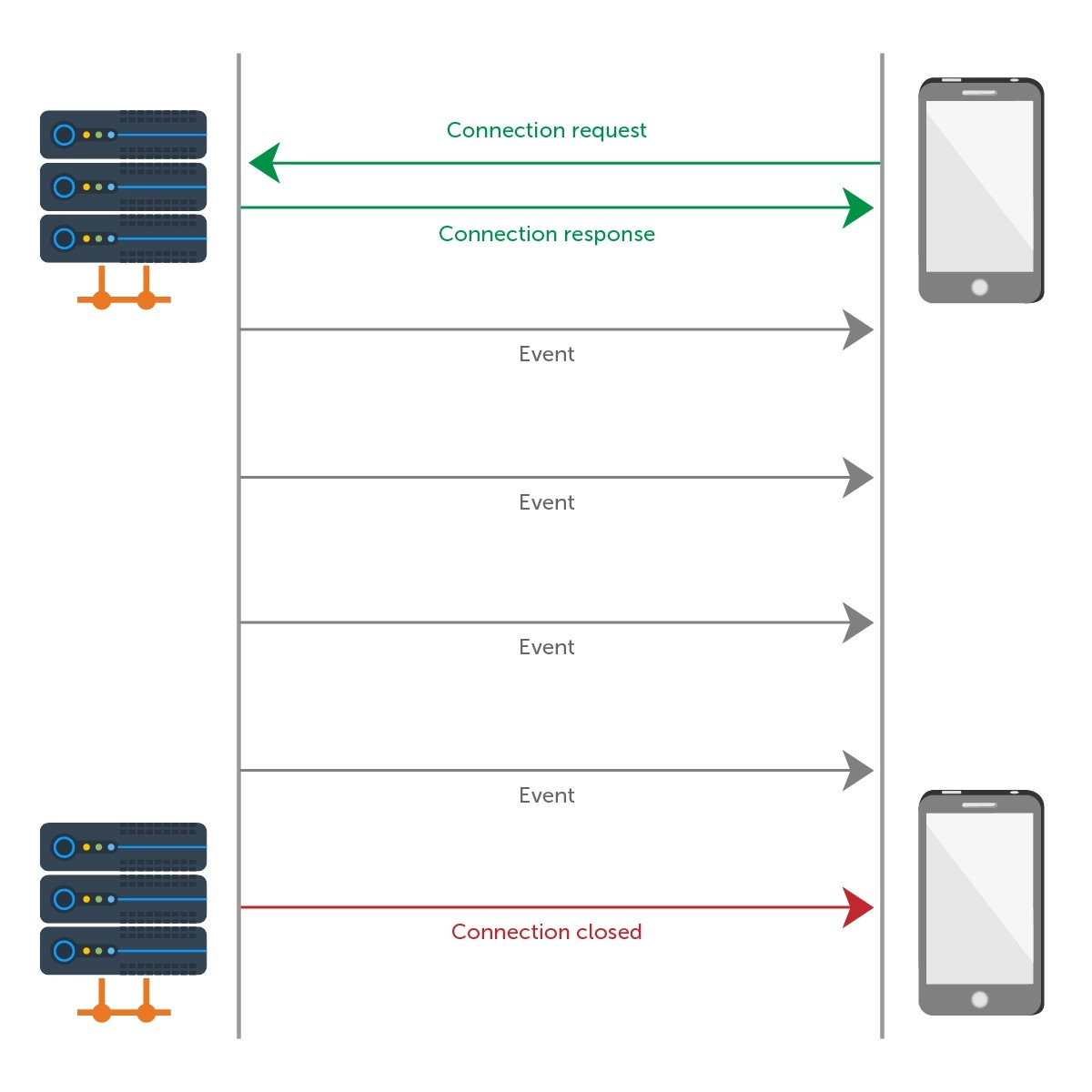
Фигура 4 Отваряне на връзка чрез WebSocket

## Server-Sent Events

Когато не е възможна комуникация чрез WebSocket, следващия подход, с който се прави опит за осъществяването и е чрез Server-Sent Events. При този метод на комуникация, клиента изпраща HTTP заявка към сървъра, а той отговаря чрез отваряне на връзка между клиента и сървъра, наречена Event Source. По време на съществуването на връзката сървърът периодично изпраща информация за настъпили събития. Тази връзка е еднопосочна и служи единствено за изпращане на информация от сървъра към клиента както е показано на Фигура 5.

Когато клиента трябва да изпрати информация към сървъра, той трябва да използва традиционна HTTP заявка.

Един недостатък на този подход е, че при свързване чрез HTTP 1.1 максималния брой отворени връзки към домейн за един браузър е 6, което може да създаде проблем ако например отворим няколко отделни таба към определено приложение в браузъра. От друга страна при HTTP/2 този брой е увеличен на 100.

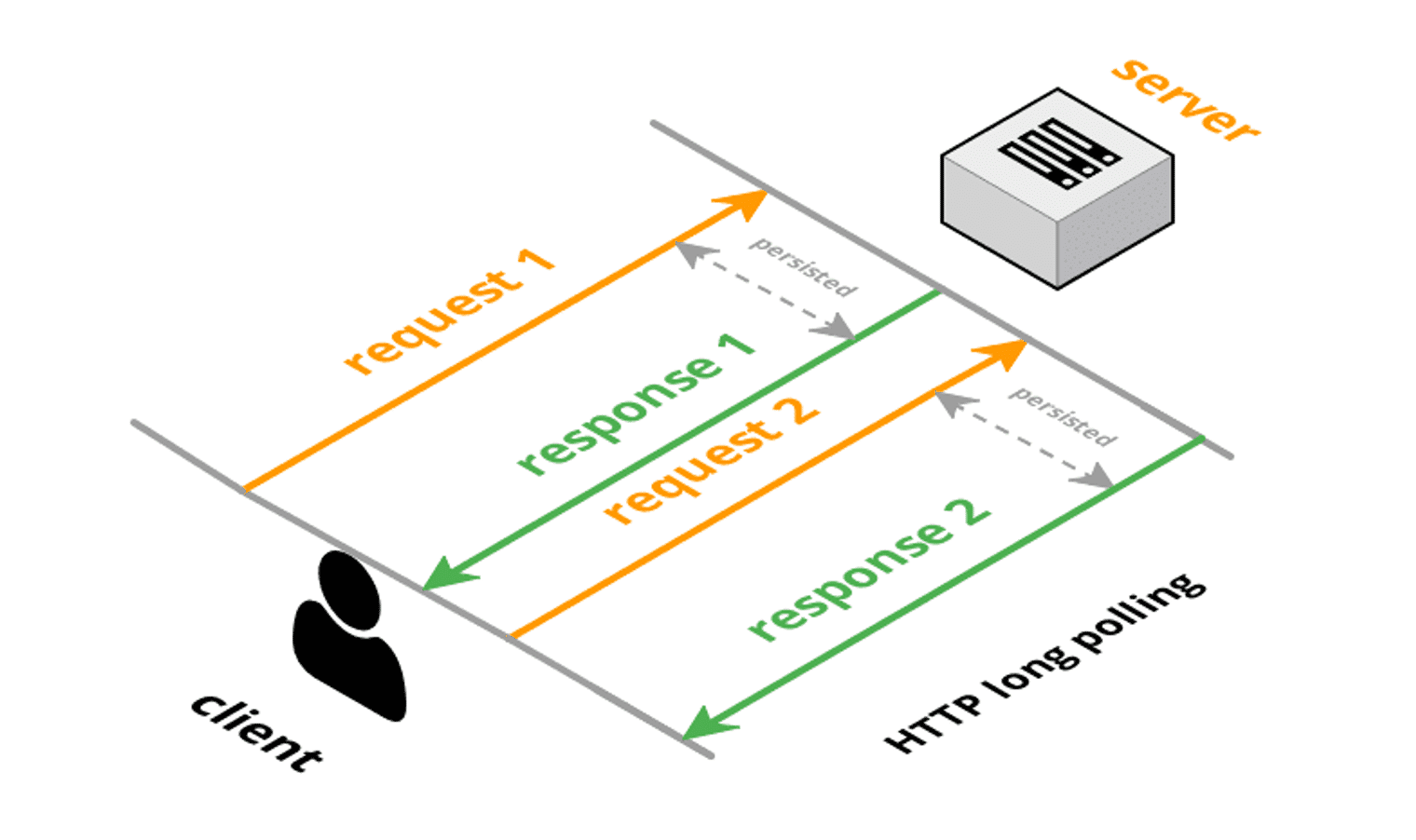


Фигура 5 Комуникация осъществена чрез Server-Sent Events

## Long Polling

Последния вариант, когато не е възможно да се използва нито WebSocket връзка, нито Server-Sent Events, остава Long Polling. При този метод на комуникация клиентът изпраща HTTP заявка към сървъра. Сървърът не изпраща отговор на заявката, а изчаква да настъпи дадено събитие, за да може да уведоми клиента и тогава отговаря. При получаването на отговор или при автоматичното прекратяване на връзката (стандартното време за изчакване на отговор е 2 минути) се изпраща нова заявка, която отваря нова връзка към сървъра и отново изчаква за отговор. По този начин се създава един непрекъснат цикъл на изпращане на заявка и получаване на отговор, където за да съществува комуникация на живо е нужно винаги да има изпратена HTTP заявка на сървъра, на която той трябва да отговори.

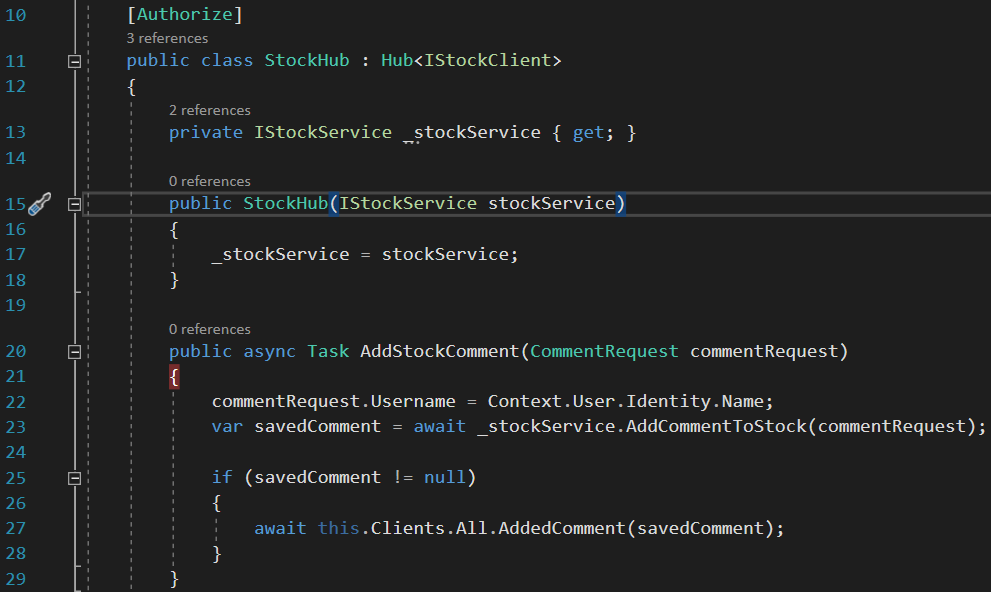
Това е най-стария подход от трите изброени досега. Той е най-неефективен и затова рядко се използва. Плюсът при използването му е, че се поддържа от много стари версии на браузърите, следователно е идеален за остарели системи, нуждаещи се от този тип функционалност.

****

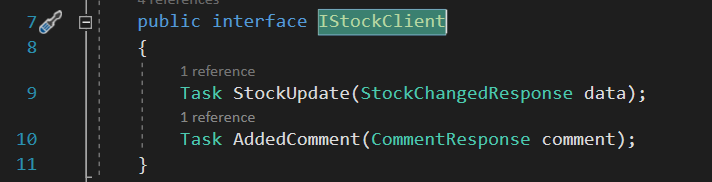
Фигура 6 Обмяна на данни чрез Long Polling подхода

# Компоненти предоставени от SignalR

SignalR използва т.нар. Hub , като входна точка на клиентите към приложението. Hub представлява един канал , който е на по-високо ниво на абстракция и позволява на сървъра да извиква методи на клиента и обратно. На Фигура 7 е изобразен Hub класът, който използваме в приложението Bull Market. Този клас е наречен StockHub и е основното средство, което използваме в приложението ни, за комуникация на живо.



Фигура 7 Представяне на основните характеристики на Hub класа



Фигура 8 Описание на събитията, за които клиента може да се абонира

## Деклариране на събитията на сървъра, за които клиента може да се абонира

Ясно се вижда, че класът наследява базовия клас Hub, на който е подаден генерален (generic) параметър с интерфейса IStockClient. В този интерфейс се описани всички събития, за които клиентът, може да се абонира, както е показано на Фигура 8. Описанието на събитията по споменатия начин характеризира нашият Hub, като силно типизиран (strongly typed). Това е и предпочитания начин на работа, понеже така биваме подпомогнати от компилатора и е са ясно описани типовете на данните, които ще бъдат изпратени. В случая е описано, че за клиента могат да настъпят две събития – StockUpdate и AddedComment. В стандартния подход, изобразен на Фигура 9, се използва метода SendAsync и като параметри се подават името на събитието, както и самите данни. Работата с магически низове е възможно да създаде много проблеми при разработка и затова не се препоръчва.



Фигура 9 Изпращане на уведомление при използване на слаба типизация

Понеже в нашия случай е използвана силна типизация, уведомлението на клиентите може да се види на Фигура 10 или на ред 27 от Фигура 7. Това става посредством свойство Clients, идващо от базовия клас Hub. Това свойство ни позволява да изпращаме информация на клиентите, като в случая е зададено да се изпрати информация към всички клиенти със свойството All. Последната част от командата е специфицирането на събитието, което настъпва, а именно AddedComment и подаване на данните за събитието.



Фигура 10 Уведомление чрез използване на силна типизация

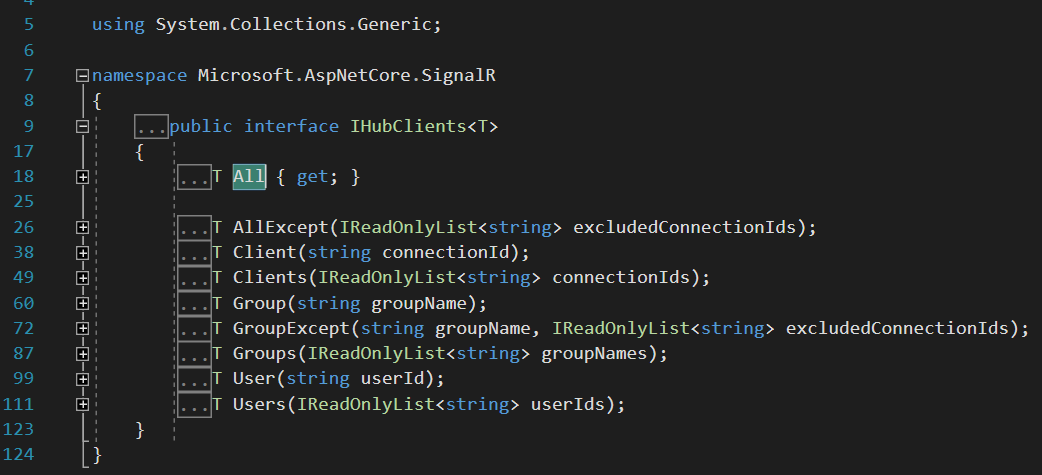
**Важно е да се отбележи, че двата споменати подхода не могат да бъдат използвани заедно в рамките на един наследник на Hub класа.**

## Деклариране на входните точки на сървъра, където клиентът може да изпраща данни

Методът AddStockComment, изобразен на Фигура 7, предоставя възможността на клиента да изпраща данни към сървъра, стига те да отговарят на посочения формат, който е представен в CommentRequest класа. На ред 27 на същата фигура е изобразен механизмът за обратно изпращане на данни към клиента. Това което се случва в метода е, че се взима информацията за даден коментар, записва се в базата данни и след това всички клиенти на приложението, биват известени за настъпилото събитие.

## Логическо разграничаване на клиентите

В SignalR има множество различни начини за логическо разграничаване на отделни групи от нашите клиент, които са изобразени на Фигура 9. Най-широко използваните варианти са All, Group, User, Users. Вторият подход ни дава възможност да вкараме множество наши клиентски връзки в смислова група, която да бъде нотифицирана при различни обстоятелства. При предпоследния и последния варианти се вижда, че можем също така да изпращаме данни към конкретен потребител или множество от потребители.



Фигура 11 Различни начини за уведомяване на клиентите

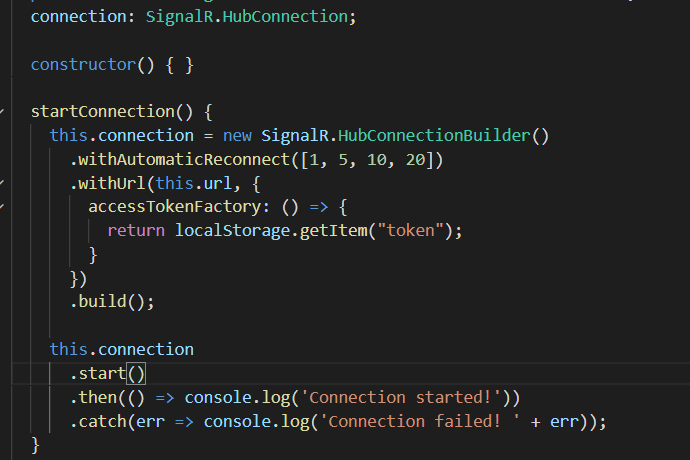
## Изграждане на връзка от клиентска страна

За успешно изграждане на връзка от страна на клиента е нужно да минем през няколко стъпки. Първо е нужно да вложим клиентската библиотека във файла откъдето ще установим връзката, както е показано на Фигура 12.



Фигура 12 Влагане на клиентската библиотека

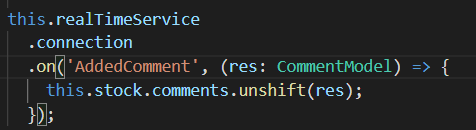
След това е нужно да изградим самата връзка с помощта на HubConnectionBuilder, където подаваме url на сървъра, както и да стартираме връзката. Всичко това е показано на Фигура 13. Важно е да се отбележи, че самата връзка се пази в свойството connection на класа StockRealTimeService в клиентското ни приложение.



Фигура 13 Изграждане и стартиране на връзката

## Абониране на клиента за събития настъпили на сървъра

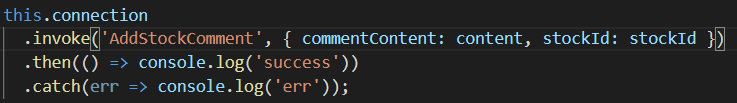
Клиента може да бъде уведомен за настъпилите събития, само след като се абонира. Това става посредством извикване на метода on на стартираната връзка. На този метод се подават два параметъра – името на събитието, както и функция обработваща получените данни.



Фигура 14 Абониране на клиента за събитие

## Изпращане на данни от клиентска страна към сървъра

Изпращането на данни към сървъра е изключително лесно. Използва се метода invoke на връзката на клиента и се подават името на сървърния метод, както и данните.



Фигура 15 Изпращане на данни към сървъра

SignalR предоставя два протокола за съобщенията: текстов, който предоставя информацията в JSON формат и бинарен, който е базиран на MessagePack протокола. Разликата между двата е, че MessagePack създава съобщения с по-малък размер.

# Функционалност предоставена от Bull Market

Целта на проекта, който сме разработили е да покаже част от възможностите на SignalR.

Той съдържа клиент, който е Angular приложение и представлява User Interface, чрез който потребителите могат да влязат или регистрират и да получават информация за определени акции в реално време, да оставят коментари за всяка акция и да разглеждат по-подробна информация за нея.

Angular клиента комуникира с ASP .NET Core 3.1 Web приложение, което се състои както от REST API за предоставяне на детайлната информация за всяка акция, така и от SignalR Hubs, които организират WebSocket връзки, чрез които те предоставят на клиентите информация в реално време за цените на акциите. Примери за двата вида комуникация са: информацията в реално време за цените на акциите и коментарите към всяка акция (WebSocket) , детайлната информация за всяка акция (HTTP).

.NET Core приложението взема информация за текущите цени на акциите посредством Alpaca Streaming API , като отваря WebSocket връзка с Alpaca API при стартирането на приложението и я поддържа отворена докато не бъде спряно.

Информацията за акциите, коментарите и потребителите са съхраняват в SQL база данни, като за комуникация с нея .NET Core приложението използва EF Core 3.1.

Кодът на приложението е публикуван на [GitHub Repository](https://github.com/DarkLaugh/BullMarket).

# Библиография

Madeshvaran, S. (н.д.). *Xamarin.Forms MVVM: How to Work with SQLite DB(C# — Xaml)*. Извлечено от https://medium.com: https://medium.com/swlh/xamarin-forms-mvvm-how-to-work-with-sqlite-db-c-xaml-26fcae303edd

Microsoft. (н.д.). *Data Binding Basics*. Извлечено от https://docs.microsoft.com: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/data-binding-basics

Microsoft. (н.д.). *Frame*. Извлечено от https://docs.microsoft.com: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/layouts/frame

Microsoft. (н.д.). *From Data Bindings to MVVM*. Извлечено от https://docs.microsoft.com: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/data-bindings-to-mvvm

Microsoft. (н.д.). *Images in Xamarin.Forms*. Извлечено от https://docs.microsoft.com: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/images?tabs=windows